v3.4.3及以上版本

jnpf-web

代码结构说明

```
jnpf-web
     ├─── build - 构建相关
     ├─── plop-templates - 基本模板
     ├─── public - 静态资源
     ├     ├─── cdn - 静态资源
     ├     ├─── favicon.ico - favicon 图标
     ├     └─── index.html - html 模板
     ├─── src - 源代码
     ├     ├──  api - 请求接口
     ├     ├──  assets - 主题、字体等静态资源
     ├     ├──  components - 全局公用组件
     ├     ├──  directive - 全局指令
     ├     ├──  filters - 全局 filter
     ├     ├──  lang - 国际化 language
     ├     ├──  layout - 全局 layout
     ├     ├──  router - 路由
     ├     ├──  store - 全局 store 管理
     ├     ├──  styles - 全局样式
     ├     ├──  utils - 全局公用方法
     ├     ├──  views - views 所有页面
     ├     ├──  App.vue - 入口页面
     ├     ├──  main.js - 入口文件 加载组件 初始化等
     ├     ├──  permission.js - 权限管理
     ├     └─── settings.js - 基础配置
     ├─── tests - 测试
     ├─── .env.xxx - 环境变量配置
     ├─── .eslintrc.js -  eslint 配置
     ├─── .babelrc - eslint 配置
     ├─── vue.config.js - vue-cli 配置
     ├─── postcss.config.js - postcss 配置
     └─── package.json- 版本信息、执行脚本、依赖信息
```

依赖说明

```
"dependencies": {
  "@fullcalendar/bootstrap": "^4.4.0", // 日程管理
  "@fullcalendar/core": "^4.4.0", // 日程管理
  "@fullcalendar/daygrid": "^4.4.0", // 日程管理
  "@fullcalendar/interaction": "^4.4.0", // 日程管理
  "@fullcalendar/timegrid": "^4.4.0", // 日程管理
  "@fullcalendar/vue": "^4.4.0", // 日程管理
  "@tinymce/tinymce-vue": "^3.2.0", // tinymce 富文本编辑器
  "axios": "0.18.1", // axios 依赖
  "clipboard": "2.0.4", // 复制剪贴板
  "codemirror": "^5.58.2", // 在线代码编辑器
  "dayjs": "^1.8.29", // dayjs 库
  "driver.js": "0.9.5", // 新用户引导
  "dropzone": "5.5.1", // 文件拖拽类库
  "echarts": "4.2.1", // echarts 可视化图表
  "echarts-stat": "^1.1.1", // echarts 可视化图表
  "element-resize-detector": "^1.2.1", // 元素监听
  "element-ui": "2.15.5", // element-ui 库
  "file-saver": "2.0.1", // 导出文件
  "fuse.js": "3.4.4", // 模糊搜索
  "highcharts": "7.1.3", // highcharts 可视化图表
  "highcharts-vue": "^1.3.5", // highcharts 可视化图表
  "js-base64": "^2.6.2", // base64
  "js-md5": "^0.7.3", // md5 加密
  "jsbarcode": "^3.11.0", // 一维码
  "jsencrypt": "^3.0.0-rc.1", // jsencrypt 加密解密
  "jsonlint": "1.6.3", // json 规范
  "jszip": "3.2.1", // js 压缩
  "monaco-editor": "^0.20.0", // 代码编辑器
  "normalize.css": "7.0.0", // CSS 库
  "nprogress": "0.2.0", // 进度条
  "path-to-regexp": "2.4.0", // 字符串转化为正则表达式
  "qrcodejs2": "0.0.2", // 二维码
  "reconnecting-websocket": "^4.4.0", // websocket
  "screenfull": "4.2.0", // 全屏
  "script-loader": "0.7.2", // 脚本加载器
  "showdown": "^1.9.1", // js 库(markdown 转 html)
  "sortablejs": "1.8.4", // 拖拽
  "spark-md5": "^3.0.2", // md5
  "throttle-debounce": "^2.1.0", // 防抖
  "vcrontab": "^0.3.5", // cron 表达式
  "vue": "2.6.10", // vue
  "vue-count-to": "1.0.13", // 计数
  "vue-drag-resize": "^1.4.2", // 拖拽缩放
  "vue-esign": "^1.0.5", // 签名
```

```
    "vue-grid-layout": "2.3.12-legacy", // 拖拽式布局
    "vue-i18n": "7.3.2", // 多语言
    "vue-organization-chart": "^1.1.6", // 组织结构图
    "vue-quill-editor": "^3.0.6", // quill 富文本编辑器
    "vue-router": "3.1.1", // 路由
    "vue-simple-uploader": "^0.7.6", // 分片上传
    "vue-splitpane": "1.0.4", // 窗口分割
    "vuedraggable": "2.20.0", // 拖拽
    "vuex": "3.1.0" // vuex
  },
  "devDependencies": {
    "@babel/core": "7.0.0", // babel 编译
    "@babel/register": "7.0.0", // babel 编译
    "@vue/cli-plugin-babel": "3.5.3", // vue-cli
    "@vue/cli-plugin-eslint": "^3.9.1", // vue-cli
    "@vue/cli-plugin-unit-jest": "3.5.3", // vue-cli
    "@vue/cli-service": "3.5.3", // vue-cli
    "@vue/test-utils": "1.0.0-beta.29", // 测试
    "autoprefixer": "^9.5.1", // 自动补全前缀
    "babel-core": "7.0.0-bridge.0", // babel 编译
    "babel-eslint": "10.0.1", // babel 编译
    "babel-jest": "23.6.0", // babel 编译
    "babel-plugin-dynamic-import-node": "^2.3.3", // babel 编译
    "chalk": "2.4.2", // 修改终端输出字符样式
    "chokidar": "2.1.5", // 监听变化依赖包
    "connect": "3.6.6", // 处理中间件
    "cross-env": "^7.0.2", // 环境变量
    "eslint": "5.15.3", // eslint
    "eslint-plugin-vue": "5.2.2", // eslint
    "html-webpack-plugin": "3.2.0", // 创建 html 的入口文件
    "husky": "1.3.1", // git hooks
    "lint-staged": "8.1.5", // git 暂存文件上运行 linters 的工具
    "monaco-editor-webpack-plugin": "^1.9.0", // monaco-editor 编辑器
    "plop": "2.3.0", // 模板
    "runjs": "^4.3.2", // JavaScript 解释器
    "sass": "^1.26.10", // sass 依赖
    "sass-loader": "^7.1.0", // sass 依赖
    "script-ext-html-webpack-plugin": "2.1.3", // webpack 插件
    "serve-static": "^1.13.2", // 服务静态文件
    "svg-sprite-loader": "4.1.3", // svg-sprite 依赖
    "svgo": "1.2.0", // svg 压缩工具
    "vue-template-compiler": "2.6.10" // vue 模板编译
  },
```

快速开始

开发环境准备

Windows环境

> 本文档环境基于 `Windows 10 x64`

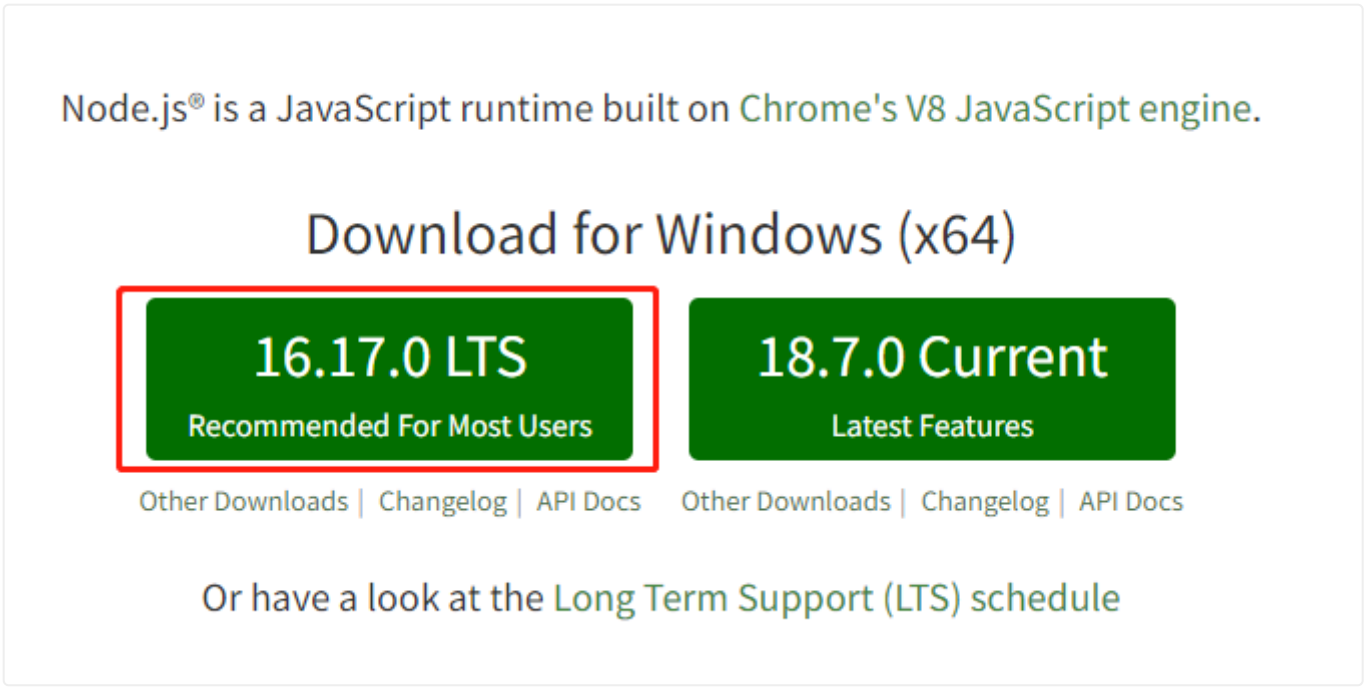
Git

> **注意：** 前端部分依赖需要 `git` 环境

打开 `https://git-scm.com/` ,下载Git安装包，按提示安装即可

Node.js

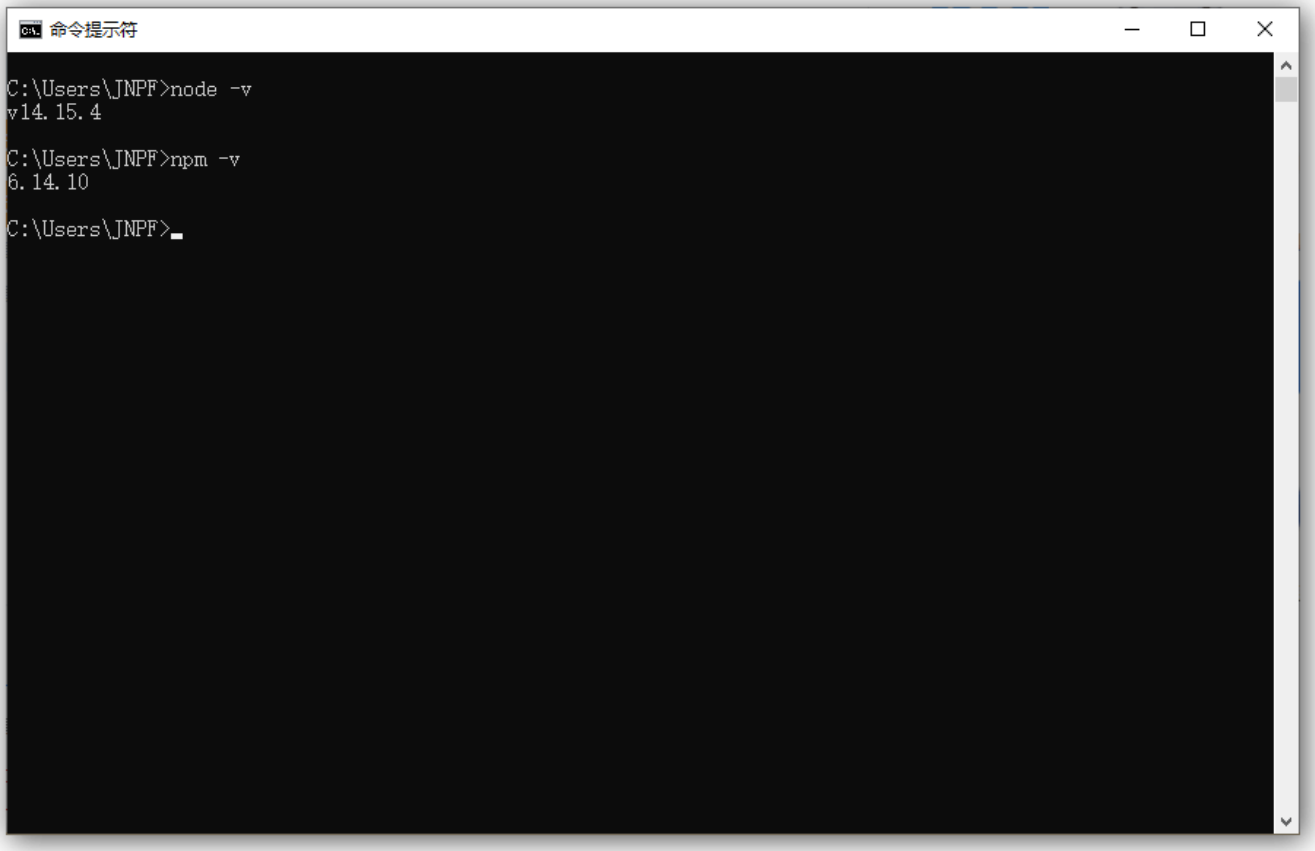
打开 `https://nodejs.org/en/` ，下载16.17.0的LTS版本



**

按提示完成安装即可。之后打开命令依次输入如下命令查看安装信息

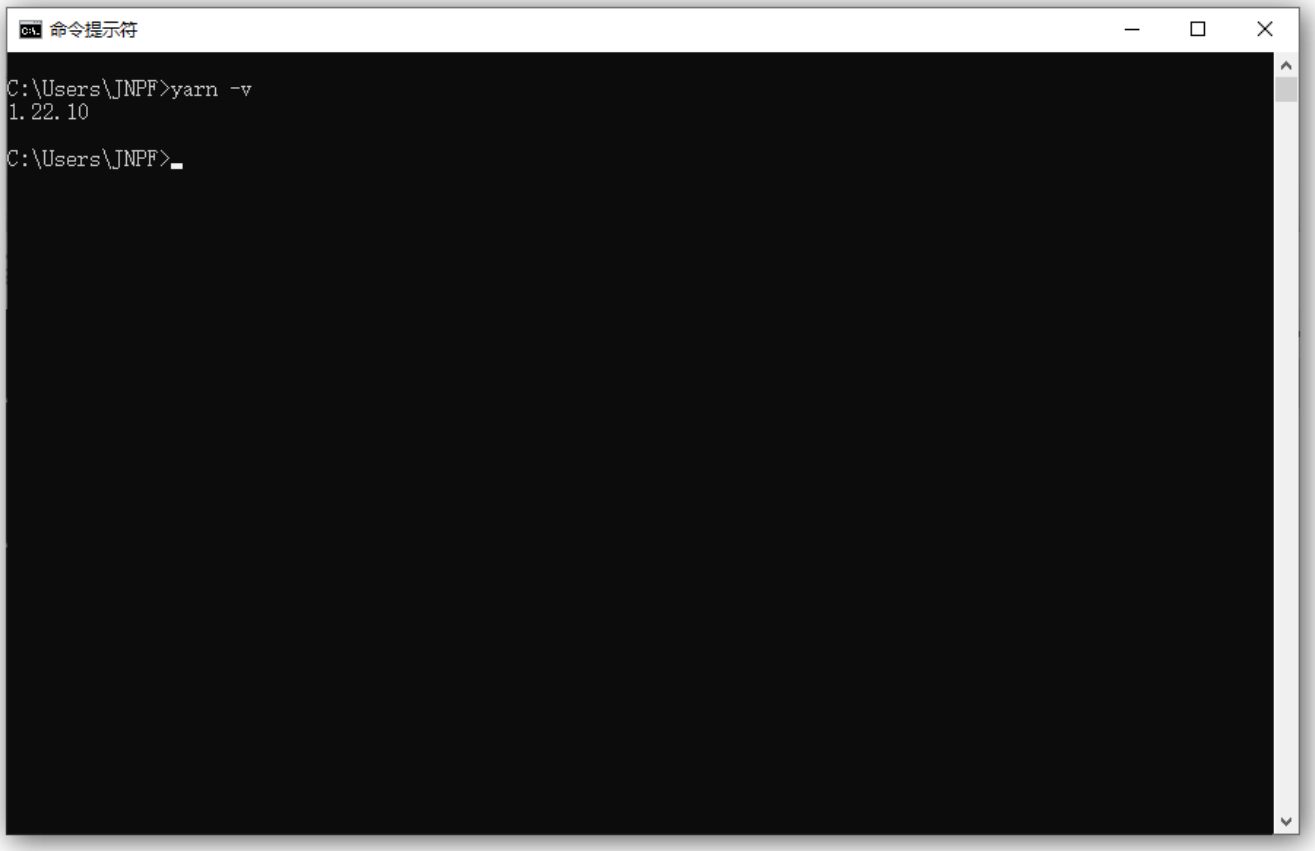
```
node -v
npm -v
```



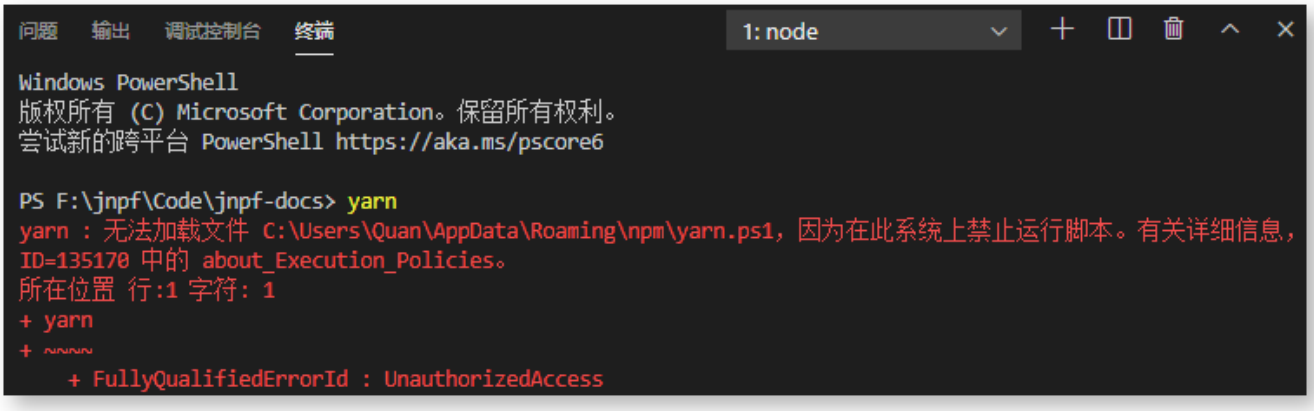
Yarn 默认为最新版本

首先安装 `Node.js` ,然后在命令行输入 `npm install --global yarn` ，回车即可安装。

- 查看安装信息



- 常见问题



解决方法：打开 `PowerShell` ，输入 `set-executionpolicy remotesigned`

VSCode

打开 `https://code.visualstudio.com` 下载安装即可。

插件推荐

- Vetur
- Auto Close Tag
- Auto Rename Tag
- JavaScript(ES6) code snippets
- Beautify

MacOS环境

> 本文档环境基于 `macOS Big Sur 11.2`

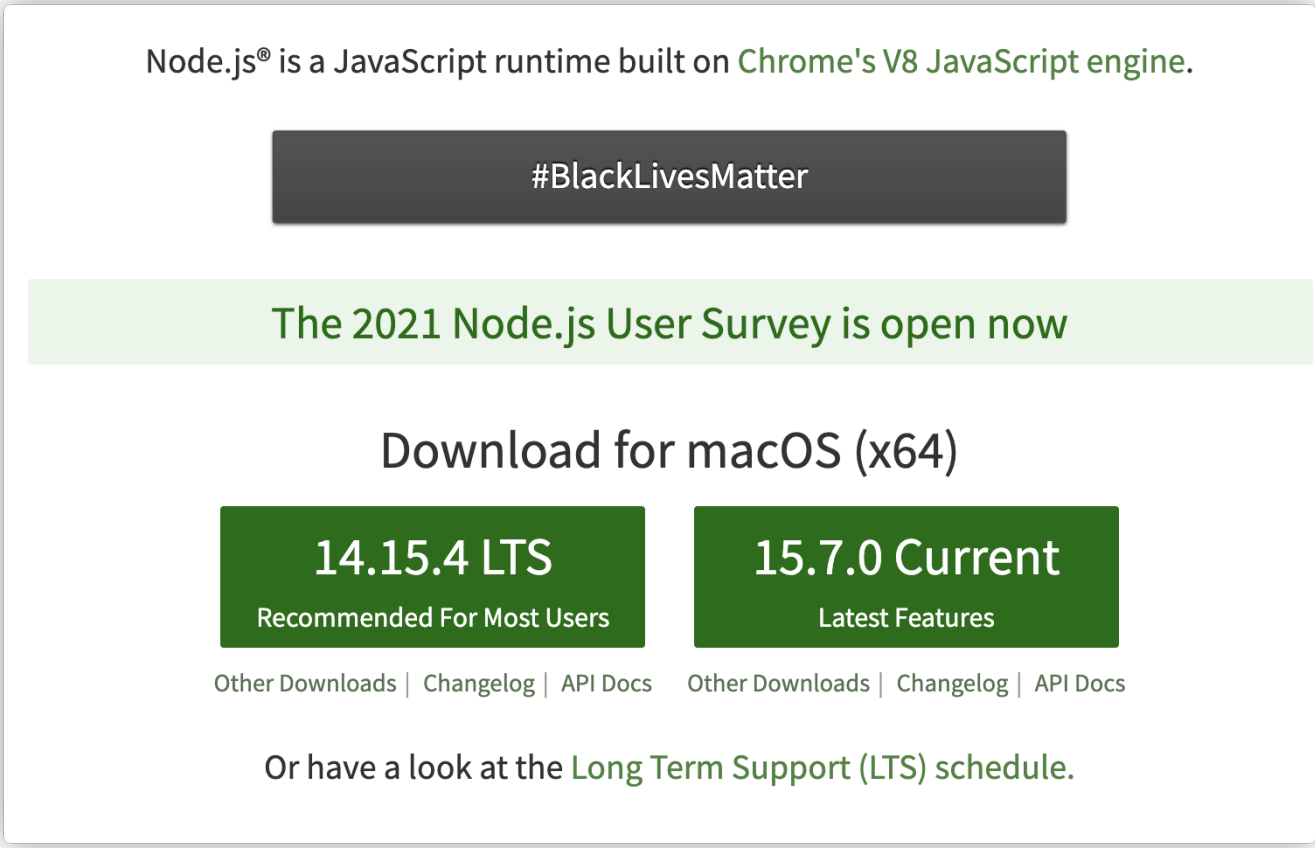
Git

> **注意：**前端部分依赖需要 `git` 环境

`macOS` 中自带 `git`

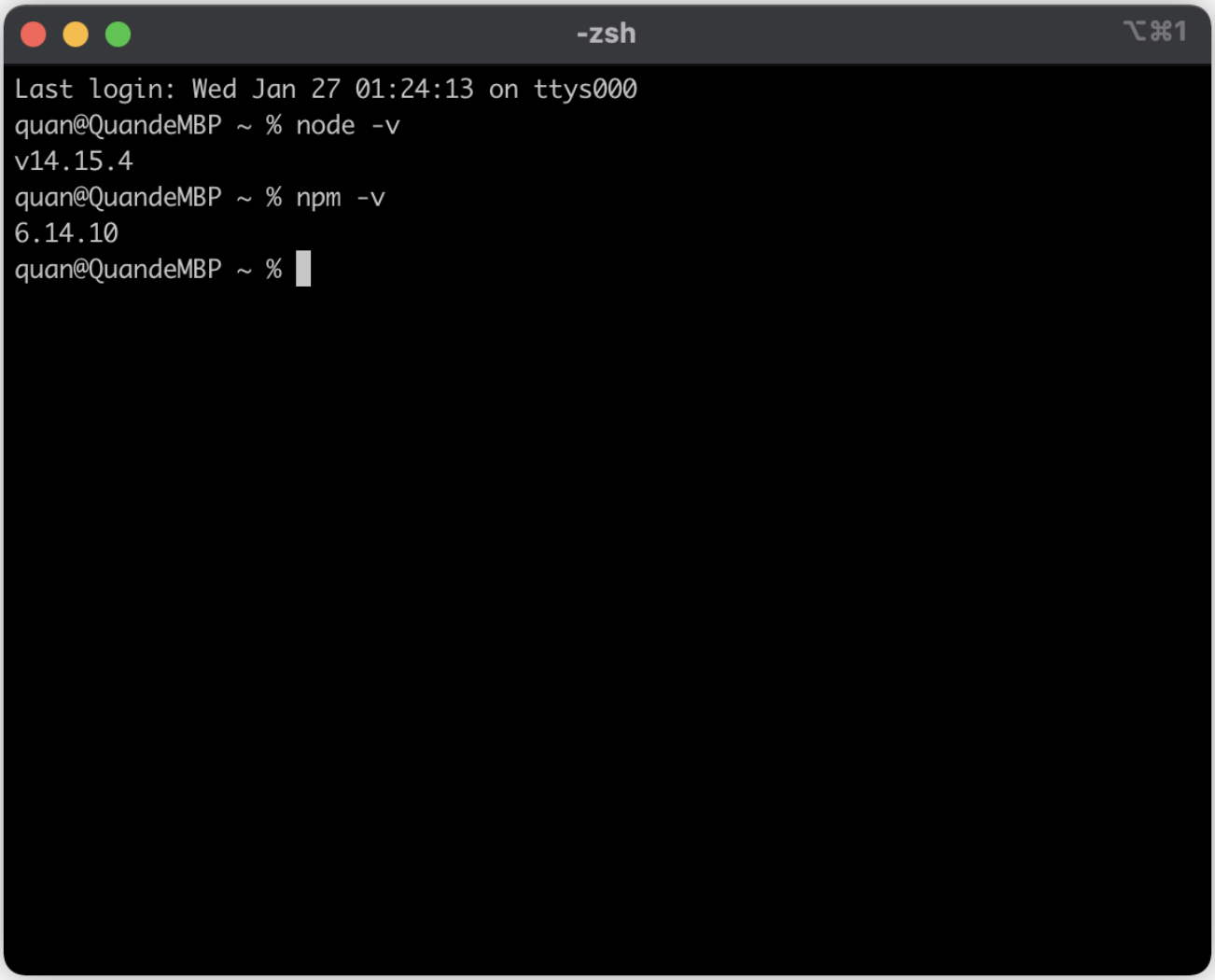
Node.js 最新LTS版本

打开 `https://nodejs.org/en/` ，下载16.17.0的LTS版本，按提示完成安装即可



打开命令依次输入如下命令查看安装信息

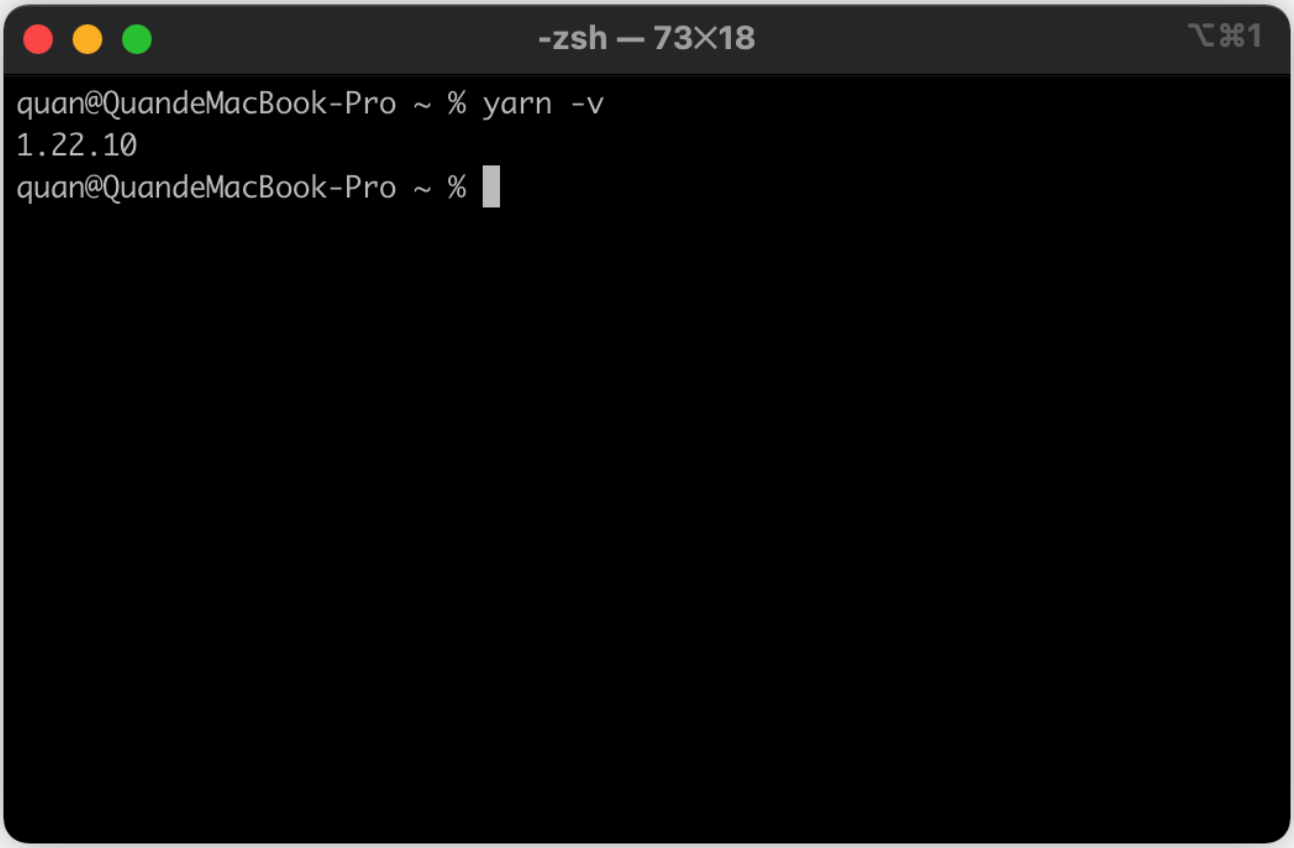
```
node -v
npm -v
```



Yarn

首先安装 `Node.js` ,然后在命令行输入 `npm install --global yarn` ，回车即可安装。

查看安装信息



VSCode

打开 `https://code.visualstudio.com` 下载安装即可。

插件推荐

- Vetur
- Auto Close Tag
- Auto Rename Tag
- JavaScript(ES6) code snippets
- Beautify

前端代码部署

项目说明

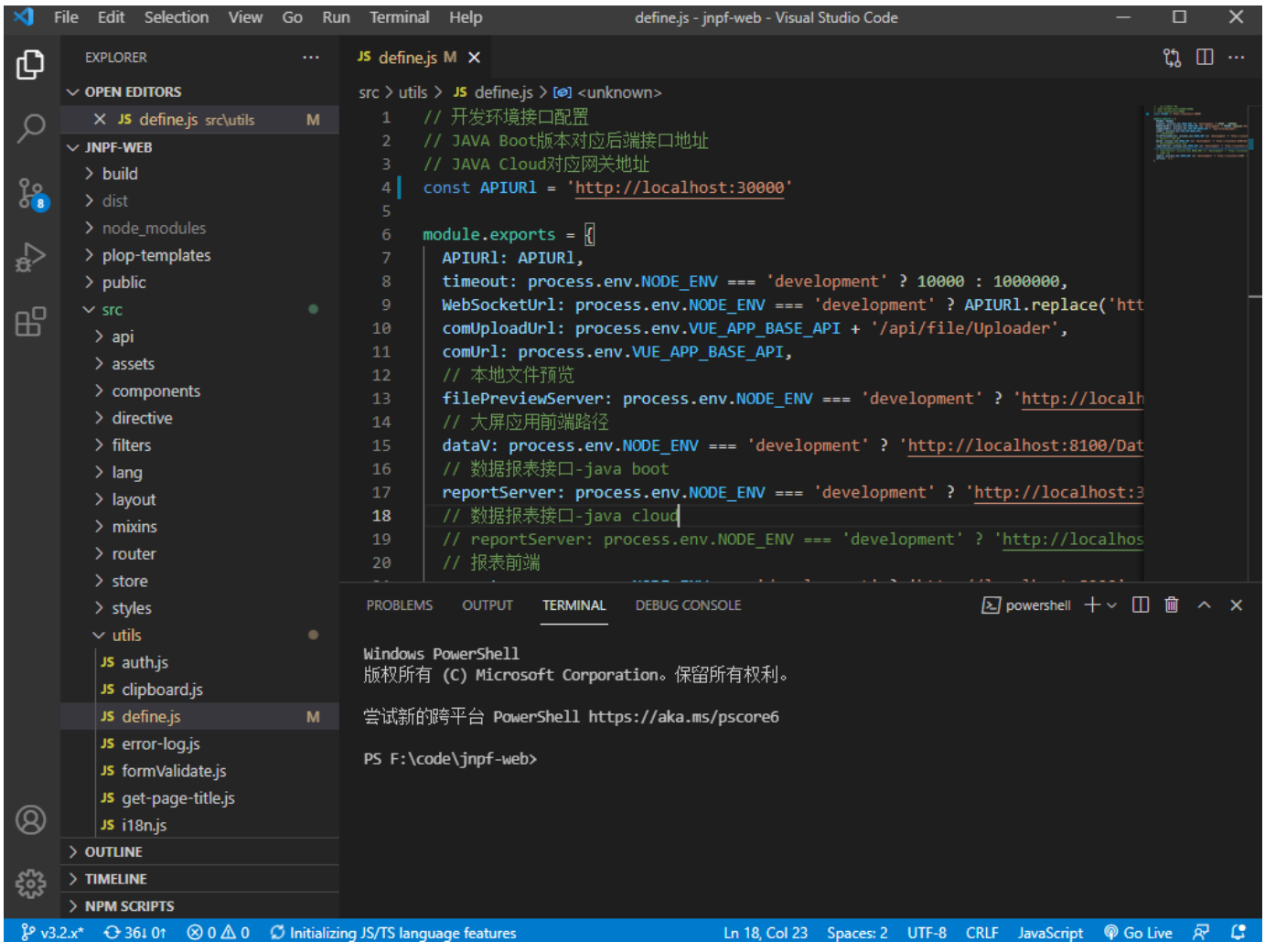
项目名	说明
`jnpf-web`	主项目前端

jnpf-web-datascreen	大屏前端
jnpf-web-datareport	报表前端
jnpf-web-tenant	多租户前端

项目导入

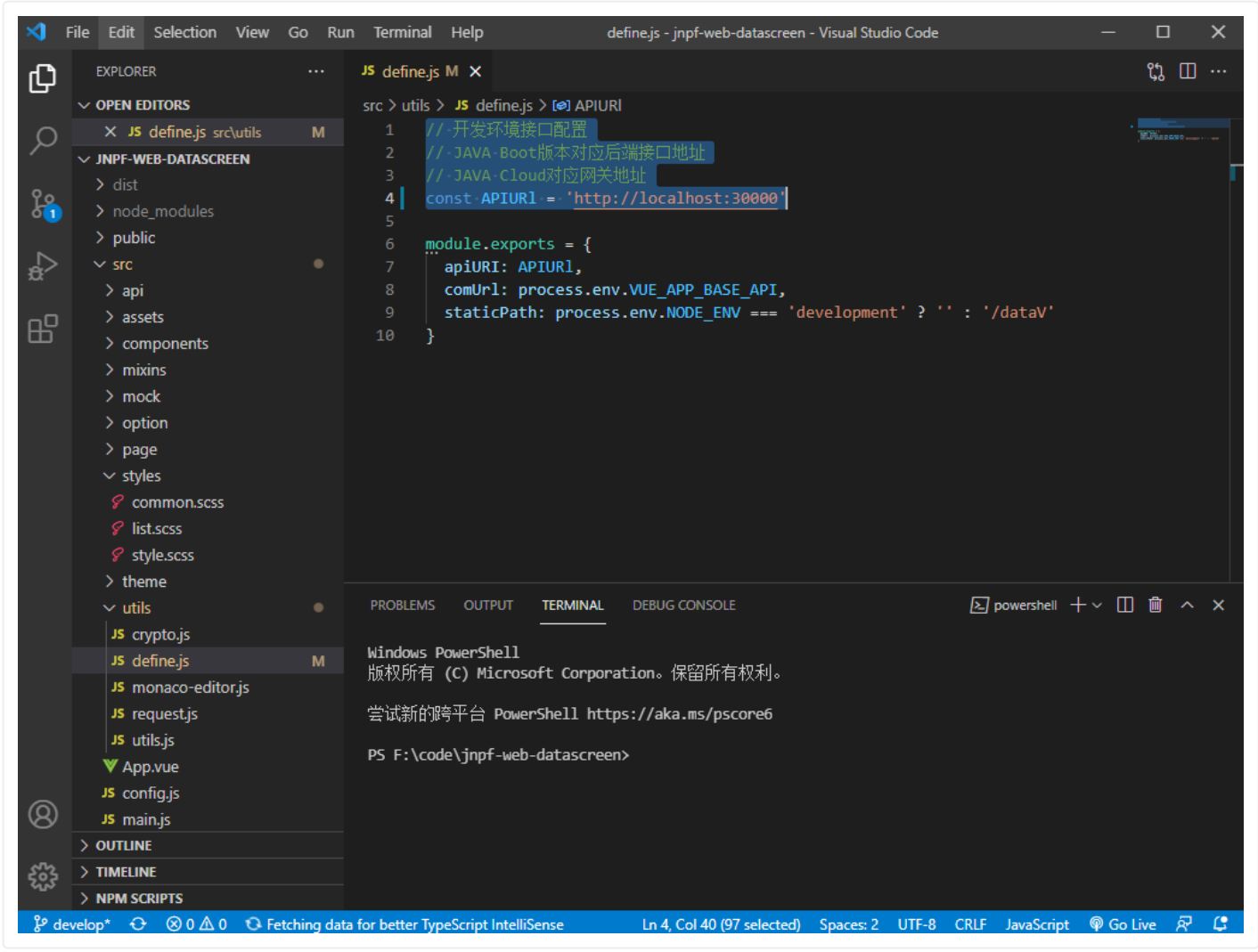
主项目前端

- 方式1：打开 Visual Studio Code 编辑器，把 jnpf-web 项目拖到编辑器主界面即可
- 方式2：依次选择菜单 文件 -> 打开文件夹，选中 jnpf-web 前端工程目录



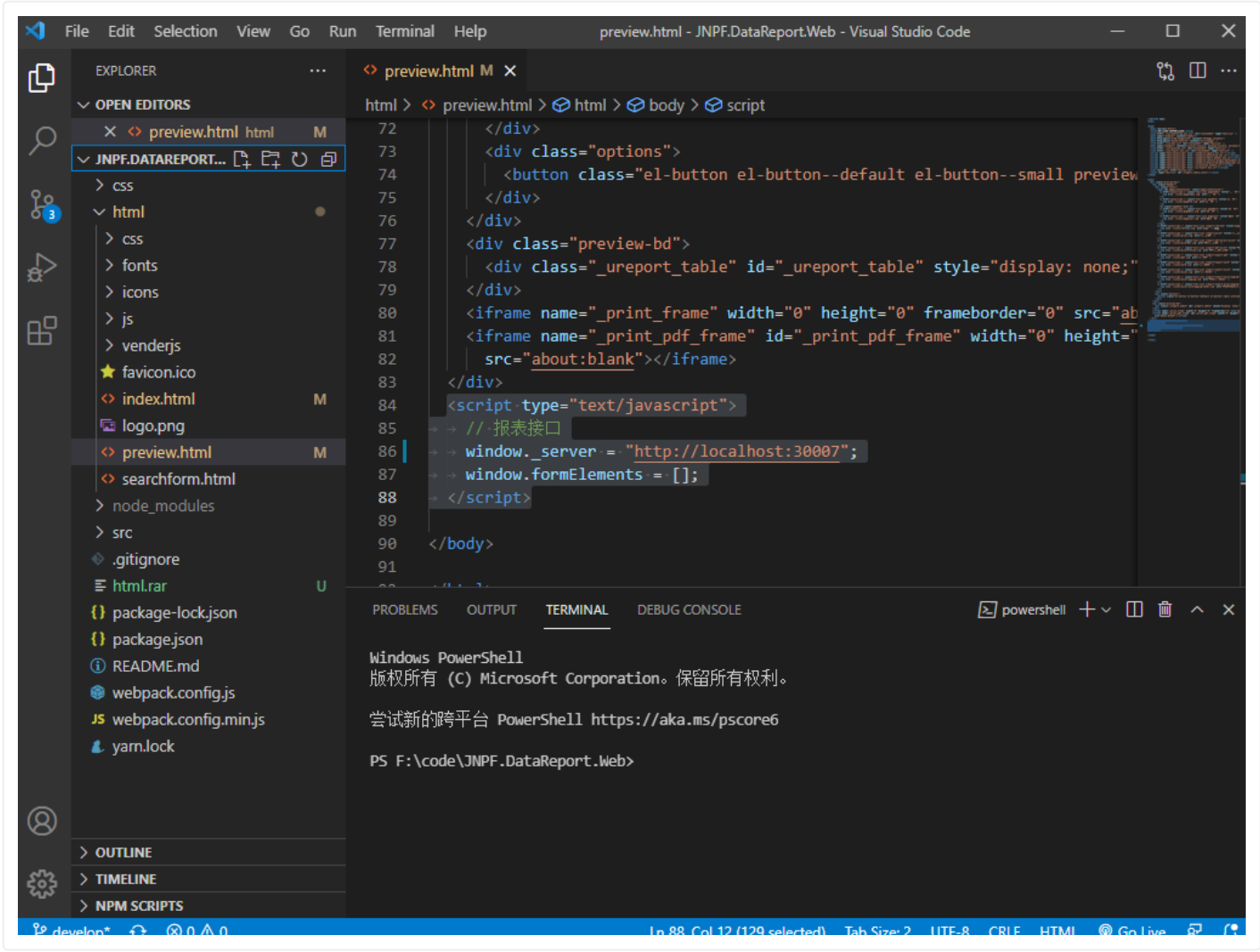
大屏前端

- 方式1：打开 Visual Studio Code 编辑器，把 jnpf-web-datascreen 项目拖到编辑器主界面即可
- 方式2：依次选择菜单 文件 -> 打开文件夹，选中 jnpf-web-datascreen 前端工程目录



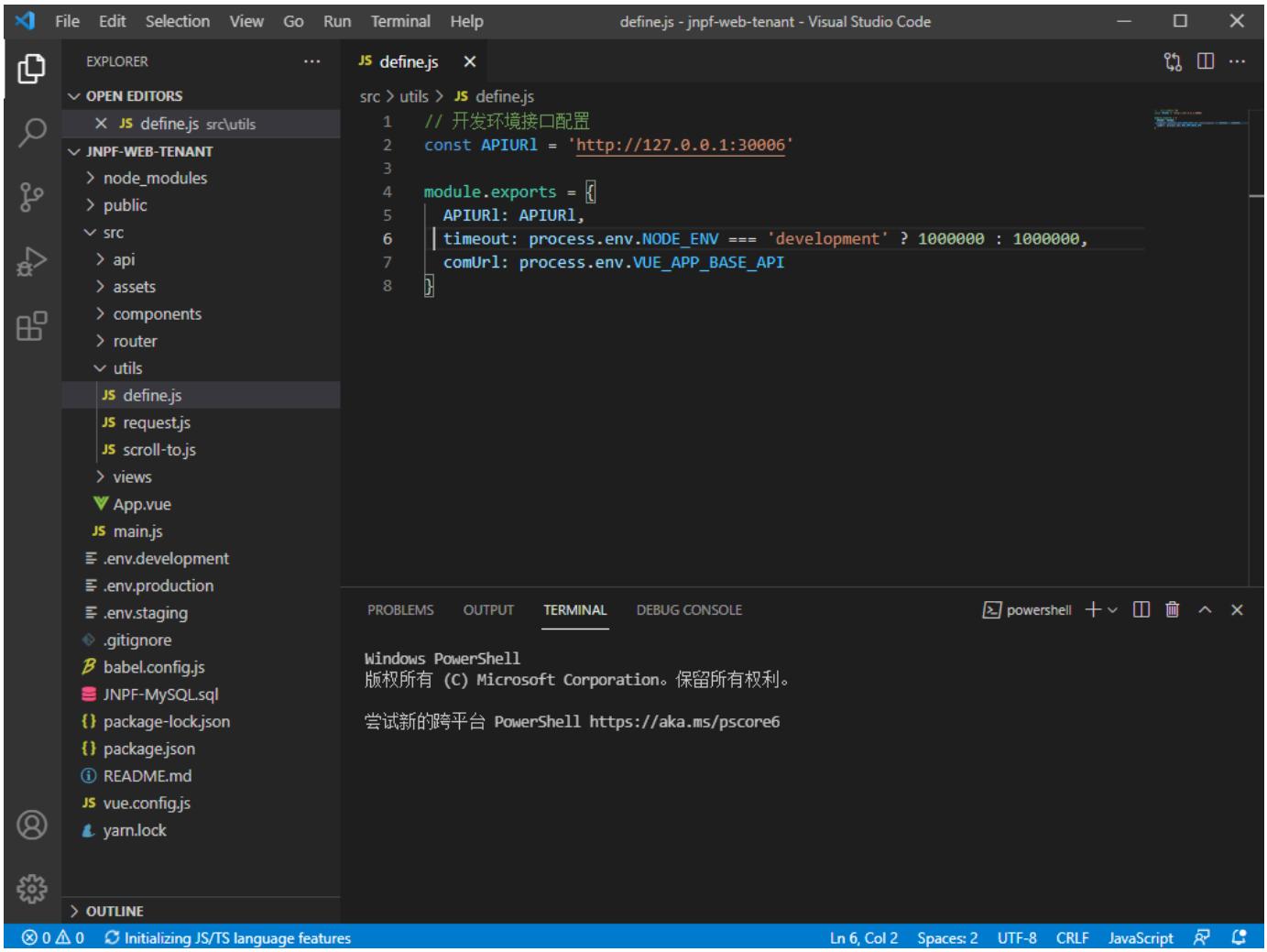
报表前端

- 方式1：打开 `Visual Studio Code` 编辑器，把 `jnpf-web-datareport` 项目拖到编辑器主界面即可
- 方式2：依次选择菜单 `文件` -> `打开文件夹`，选中 `jnpf-web-datareport` 前端工程目录



多租户前端

- 方式1：打开 `Visual Studio Code` 编辑器，把 `jnpf-web-tenant` 项目拖到编辑器主界面即可
- 方式2：依次选择菜单 `文件` -> `打开文件夹`，选中 `jnpf-web-tenant` 前端工程目录



项目运行

主项目前端

1）打开 `src/utils/define.js` ,修改开发环境接口

```
// 开发环境接口配置
const APIUrl = 'http://192.168.0.25:30000'

module.exports = {
  APIUrl: APIUrl,
  timeout: process.env.NODE_ENV === 'development' ? 10000 : 1000000,
  WebSocketUrl: process.env.NODE_ENV === 'development' ? APIUrl.replace('http', 'ws'
) + '/websocket' : process.env.VUE_APP_BASE_WSS,
  comUploadUrl: process.env.VUE_APP_BASE_API + '/api/file/Uploader',
  comUrl: process.env.VUE_APP_BASE_API,
  // 大屏应用前端路径
  dataV: process.env.NODE_ENV === 'development' ? 'http://localhost:8100/DataV' : pr
ocess.env.VUE_APP_BASE_API + '/DataV',
```

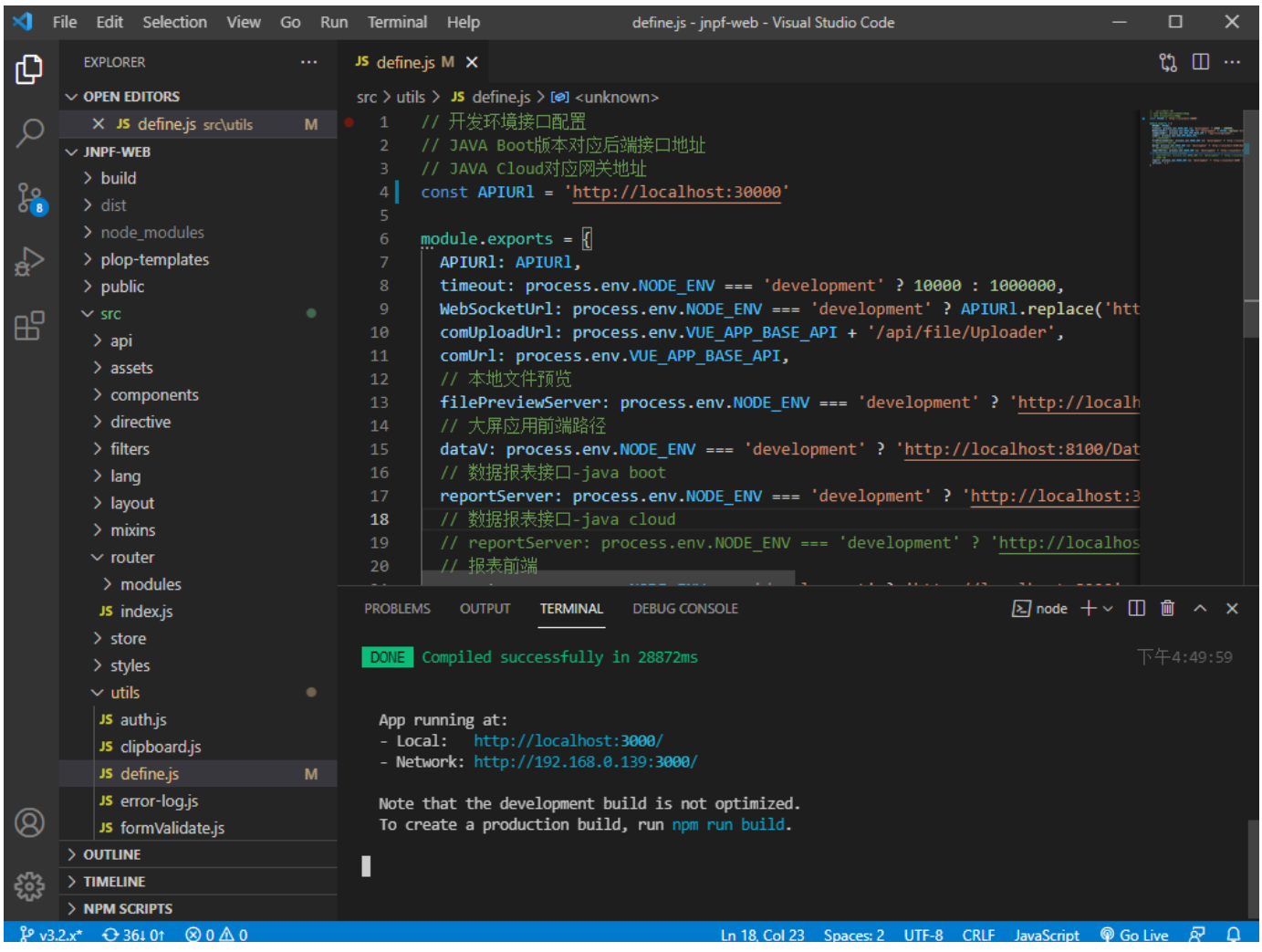
```
  // 数据报表
  reportServer: process.env.NODE_ENV === 'development' ? 'http://localhost:9000' : p
rocess.env.VUE_APP_BASE_API + '/ReportServer',
  report: process.env.NODE_ENV === 'development' ? 'http://localhost:8200' : proces
s.env.VUE_APP_BASE_API + '/Report'
}
```

2）依次选择菜单 `终端` -> `新终端`，执行以下命令，安装项目所需依赖

```
git config --global url."https://".insteadOf git://

npm install --registry http://registry.npmmirror.com
```

3）安装完依赖后，执行以下命令来运行您的项目，

```
npm run dev
```



3）运行成功后，会自动打开项目界面，如下图所示

大屏前端

1）打开 `src/utils/define.js` ,修改开发环境接口

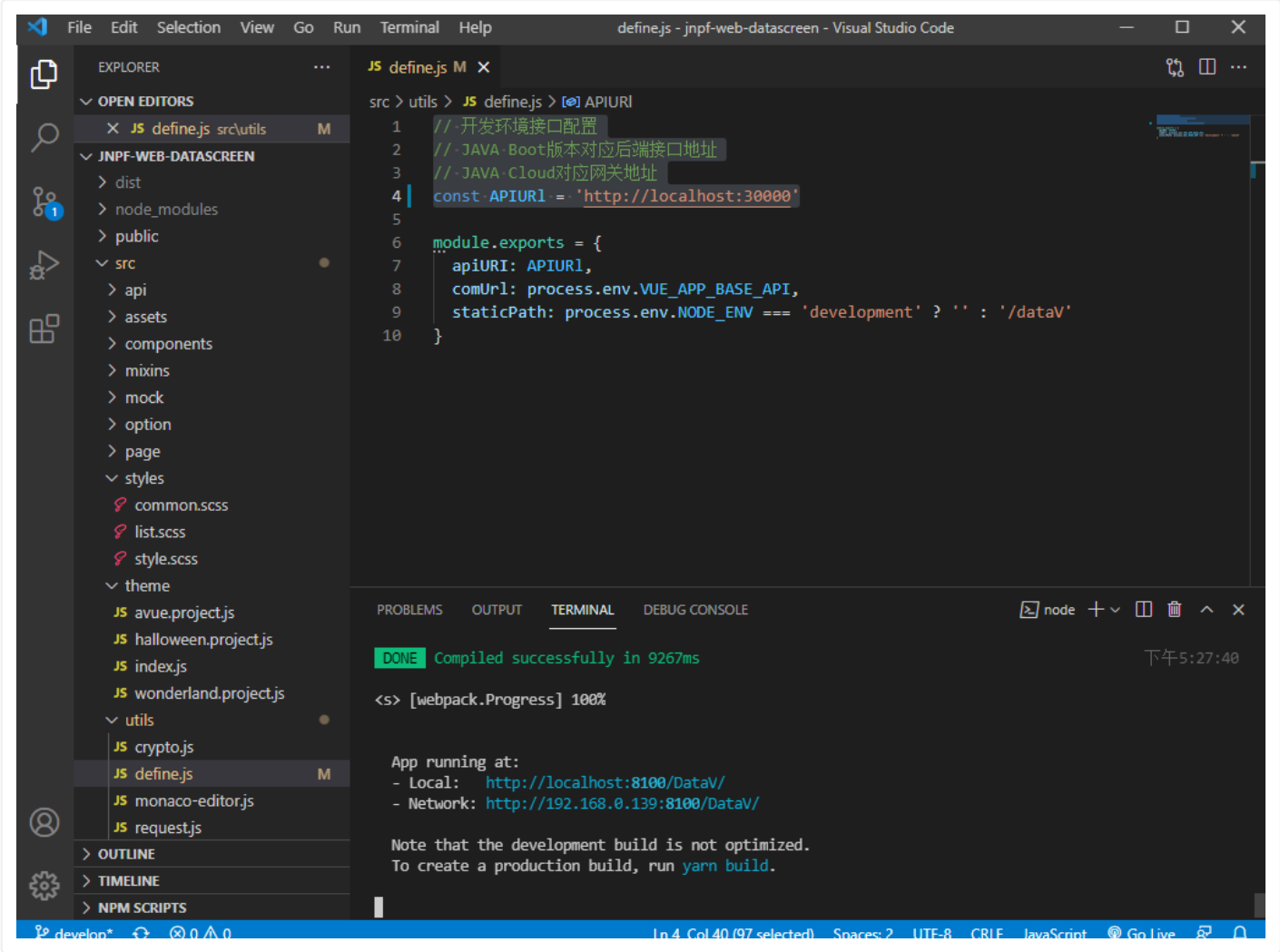
```
// 开发环境接口配置
const APIUrl = 'http://192.168.0.25:30000'

module.exports = {
  apiURI: APIUrl,
  comUploadUrl: process.env.VUE_APP_BASE_API + '/api/visualdev/DataScreen/Images/',
  comUrl: process.env.VUE_APP_BASE_API,
  staticPath: process.env.NODE_ENV === 'development' ? '' : '/dataV'
}
```

2）依次选择菜单 `终端` -> `新终端` ，执行以下命令，安装项目所需依赖

```
yarn
```

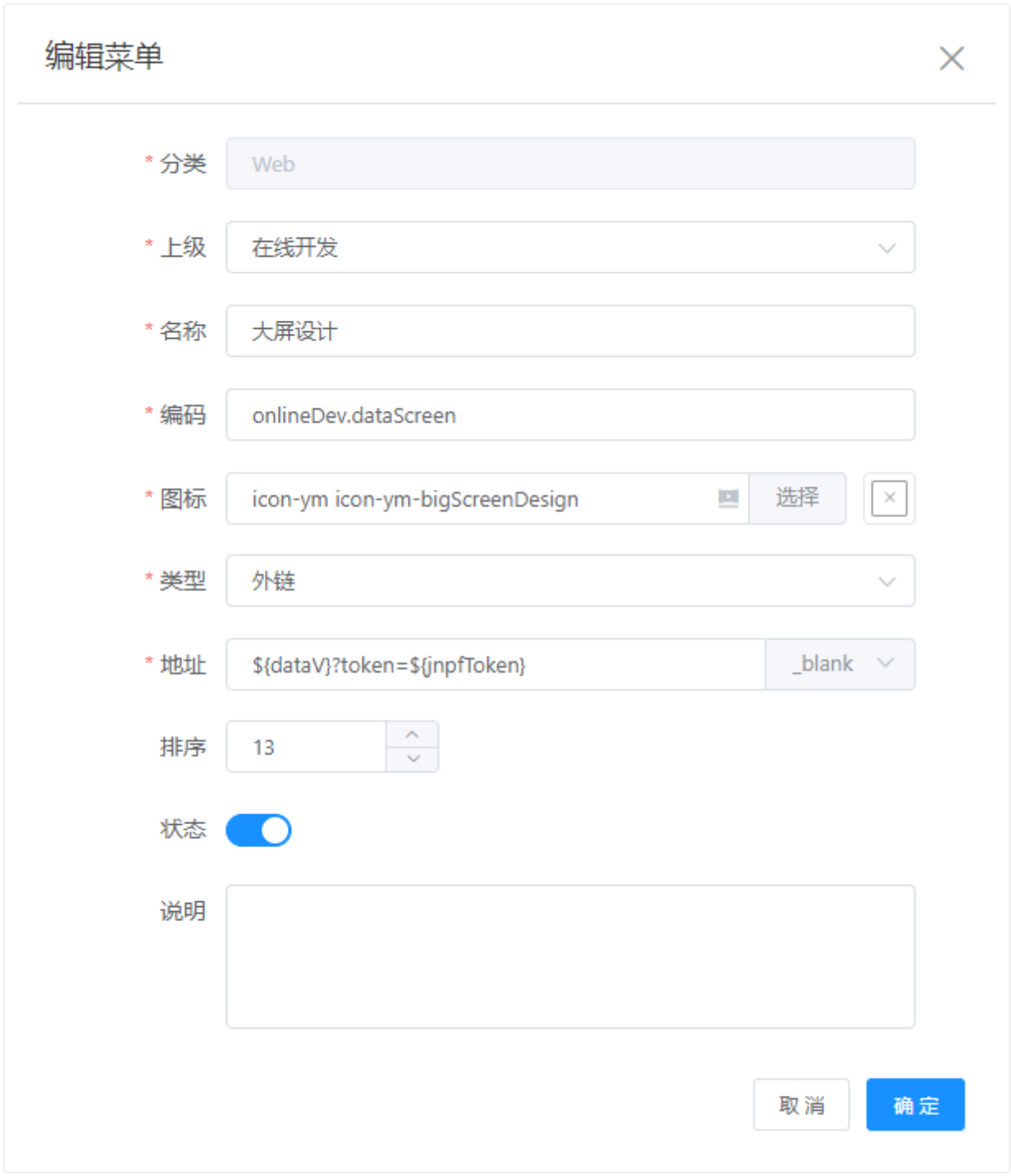
3）安装完依赖后，执行以下命令来运行您的项目

```
yarn dev
```



3）运行成功后，通过主项目前端界面打开，【在线开发】-【大屏设计】-【新建大屏】中查看（建议先运行后端）

注：旧版大屏设计更新至新版需修改菜单配置



报表前端

1）安装 `http-server`

```
npm install --global http-server
```

2）进入 `html` 目录，并修改 `index.html` 、 `preview.html` 和 `searchform.html` 的配置

index.html文件

```
<script type="text/javascript">
    // 报表接口
    window._server = "http://localhost:30007";
```

```
    // 报表前端
    window._contextPath = "http://localhost:8200";
    // 主项目接口地址
    window._mainServer = "http://localhost:30000";
  </script>
```

preview.html

```
<script type="text/javascript">
  // 报表接口
  window._server = "http://localhost:30007";
  window.formElements = [];
</script>
```

searchform.html

```
<script type="text/javascript">
    window._server = "http://localhost:30007";
    // 报表前端
    window._contextPath = "http://localhost:8200";
</script>
```

3）依次选择菜单`终端` -> `新终端`，进入`html`目录，在`html`目录下执行以下命令

```
http-server -a localhost -p 8200
```

4）运行成功后，通过主项目前端界面打开，【在线开发】-【报表设计】-【新建报表】中查看（建议先运行后端）

多租户前端

1）打开`src/utils/define.js`,修改开发环境接口

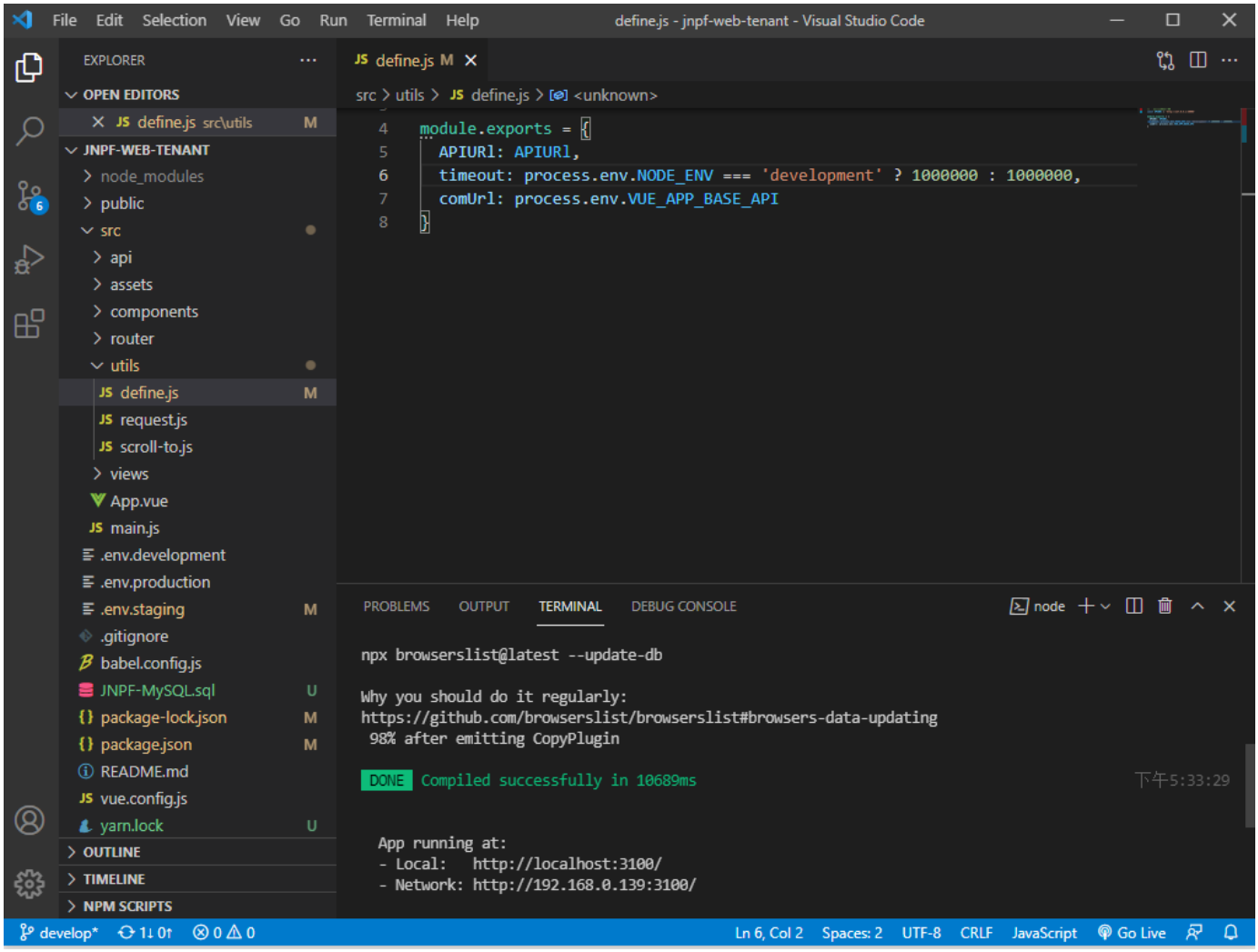
```
// 开发环境接口配置
module.exports = {
  APIURl: 'http://192.168.0.100:7100'
}
```

2）依次选择菜单`终端` -> `新终端`，执行以下命令，安装项目所需依赖

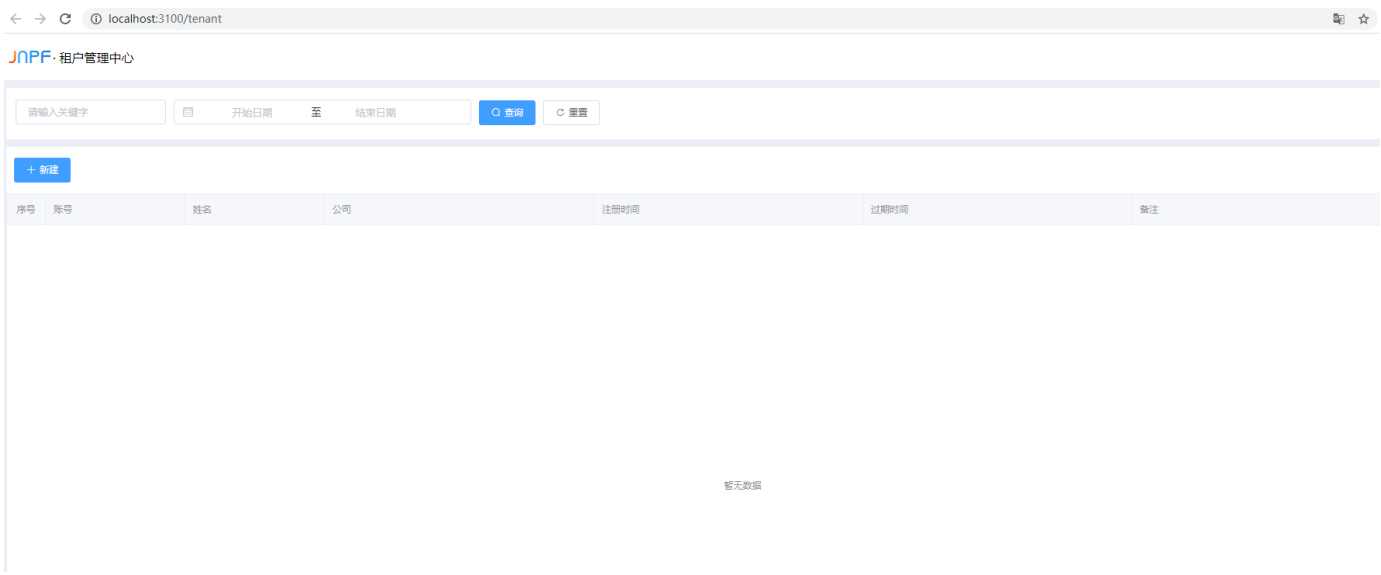
```
yarn
```

3）安装完依赖后，执行以下命令来运行您的项目

```
yarn dev
```



3）运行成功后，会自动打开项目界面，如下图所示



开发指南

package.json说明

> 本文档仅说明package.json中使用的包，请勿拷贝到项目中使用

- 前端主项目(JNPF.Vue/jnpf-web)

> 前端主项目是基于 vue-element-admin 后台前端解决方案定制开发的，基础文档可参考 https://panjiachen.github.io/vue-element-admin-site/zh/

```
{
  "name": "jnpf-admin",
  "version": "3.0.0",
  "description": "JNPF快速开发平台",
  "author": "JNPF快速开发平台",
  "license": "MIT",
   // 打包指令
  "scripts": {
    // 开发环境指令，命令行中执行：npm run dev，对应.env.development
    "dev": "vue-cli-service serve --open",
    // 生产环境指令，命令行中执行：npm run build，对应.env.product配置
    "build": "cross-env NODE_ENV=production vue-cli-service build",
    // 测试环境指令，命令行中执行：npm run build:staging，对应.env.staging配置
    "build:staging": "cross-env NODE_ENV=production vue-cli-service build --mode sta
ging",
    // 分析构建文件体积，命令行中执行：npm run build:preview
    "preview": "node build/index.js --preview",
    "lint": "eslint --ext .js,.vue src",
    "test:unit": "jest --clearCache && vue-cli-service test:unit",
    "test:ci": "npm run lint && npm run test:unit",
    "new": "plop"
  },
  "husky": {
    "hooks": {
      "pre-commit": "lint-staged"
    }
  },
  "lint-staged": {
    "src/**/*.{js,vue}": [
      "eslint --fix",
      "git add"
    ]
  },
  "keywords": [
```

```
    "vue",
    "admin",
    "JNPF",
    "element-ui",
    "JNPF-admin",
    "management-system"
  ],
  "dependencies": {
     // 日历插件，插件效果见扩展应用-日程安排
    "@fullcalendar/bootstrap": "^4.4.0",
    "@fullcalendar/core": "^4.4.0",
    "@fullcalendar/daygrid": "^4.4.0",
    "@fullcalendar/interaction": "^4.4.0",
    "@fullcalendar/timegrid": "^4.4.0",
    "@fullcalendar/vue": "^4.4.0",
    // 请求处理
    "axios": "0.18.1",
     // 实现复制文本到剪贴板
    "clipboard": "2.0.4",
     // 线代码高亮显示
    "codemirror": "^5.58.2",
     // 日期格式化处理
    "dayjs": "^1.8.29",
    // 页面分步引导
    "driver.js": "0.9.5",
    // 拖拽
    "dropzone": "5.5.1",
    // 百度图表
    "echarts": "4.2.1",
    // ECharts 的统计和数据挖掘工具
    "echarts-stat": "^1.1.1",
     // 监听dom元素
    "element-resize-detector": "^1.2.1",
     // 基于 Vue 2.0 的桌面端组件库
    "element-ui": "2.15.0",
     // 文件导出
    "file-saver": "2.0.1",
     // 模糊搜索库
    "fuse.js": "3.4.4",
    // 图表库
    "highcharts": "7.1.3",
    "highcharts-vue": "^1.3.5",
    // md5加密
    "js-md5": "^0.7.3",
     // 用于生成二维码、条形码等
```

```
    "jsbarcode": "^3.11.0",
     // RSA加密工具
    "jsencrypt": "^3.0.0-rc.1",
    // 格式化和校验JSON
    "jsonlint": "1.6.3",
    // 用于创建、读取和编辑.zip文件
    "jszip": "3.2.1",
    // 基于 VS Code 的代码编辑器
    "monaco-editor": "^0.20.0",
    // 基础CSS,
    "normalize.css": "7.0.0",
    // 进度条插件
    "nprogress": "0.2.0",
    // 正则处理
    "path-to-regexp": "2.4.0",
     // 生成二维码
    "qrcodejs2": "0.0.2",
     // 解决websocket掉线重连
    "reconnecting-websocket": "^4.4.0",
     // 全屏插件
    "screenfull": "4.2.0",
    // 代码区域高亮
    "showdown": "^1.9.1",
    // 脚本加载器
    "script-loader": "0.7.2",
    // 拖放排序列表
    "sortablejs": "1.8.4",
    // 限制函数的执行频率
    "throttle-debounce": "^2.1.0",
    // 国产markdown编辑器
    "tui-editor": "1.3.3",
    // vue.js核心包
    "vue": "2.6.10",
    // 数字滚动计数
    "vue-count-to": "1.0.13",
    // 拖拽缩放组件
    "vue-drag-resize": "^1.4.2",
     // 手写电子签名
    "vue-esign": "^1.0.5",
     // 拖拽布局
    "vue-grid-layout": "2.3.12-legacy",
    // 前端国际组件
    "vue-i18n": "7.3.2",
    // 富文本编辑器
    "vue-quill-editor": "^3.0.6",
```

```
    // Vue路由
    "vue-router": "3.1.1",
    // 分割面板
    "vue-splitpane": "1.0.4",
    // 基于Sortable.js实现的vue拖拽插件
    "vuedraggable": "2.20.0",
    // Vue状态管理
    "vuex": "3.1.0",
    // 表情转换
    "wechat-emoji-parser": "^1.0.4"
  },
  "devDependencies": {
    // js转换器
    "@babel/core": "7.0.0",
    "@babel/register": "7.0.0",
    // vue 手脚架
    "@vue/cli-plugin-babel": "3.5.3",
    "@vue/cli-plugin-eslint": "^3.9.1",
    "@vue/cli-plugin-unit-jest": "3.5.3",
    "@vue/cli-service": "3.5.3",
    "@vue/test-utils": "1.0.0-beta.29",
    // 解析CSS文件并且添加浏览器前缀到CSS规则里
    "autoprefixer": "^9.5.1",
    // js转换器
    "babel-core": "7.0.0-bridge.0",
    "babel-eslint": "10.0.1",
    "babel-jest": "23.6.0",
    // 按需加载插件
    "babel-plugin-dynamic-import-node": "^2.3.3",
    // 终端字符串样式
    "chalk": "2.4.2",
    // 监听文件变化
    "chokidar": "2.1.5",
     // 中间件处理
    "connect": "3.6.6",
    // 跨平台设置和使用环境变量的脚本
    "cross-env": "^7.0.2",
     //
    "eslint": "5.15.3",
    "eslint-plugin-vue": "5.2.2",
    // 编译 Webpack 项目中的 html 类型的文件
    "html-webpack-plugin": "3.2.0",
    // Git hooks 工具，可以防止使用 Git hooks 的一些不好的 commit 或者 push
    "husky": "1.3.1",
     // 代码规范
```

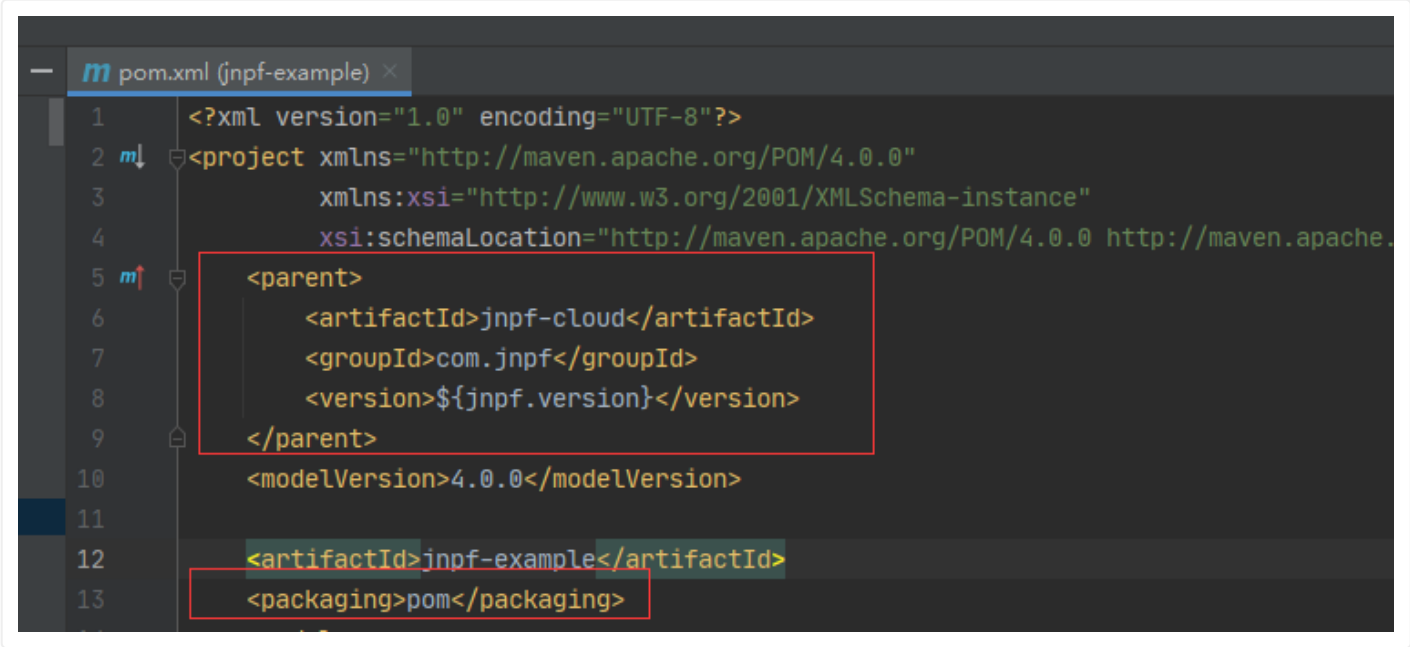
```
    "lint-staged": "8.1.5",
     // monaco编辑器webpack扩展
    "monaco-editor-webpack-plugin": "^1.9.0",
    // 根据模板文件自动化创建组件
    "plop": "2.3.0",
     // JavaScript解析器
    "runjs": "^4.3.2",
    // CSS扩展语言
    "sass": "^1.26.10",
    "sass-loader": "^7.1.0",
     // html-webpack-plugin扩展插件
    "script-ext-html-webpack-plugin": "2.1.3",
    // 静态服务器
    "serve-static": "^1.13.2",
    // svg格式处理
    "svg-sprite-loader": "4.1.3",
    "svgo": "1.2.0",
    // 模块解析
    "vue-template-compiler": "2.6.10"
  },
  "engines": {
    "node": ">=8.9",
    "npm": ">= 3.0.0"
  },
  "browserslist": [
    "> 1%",
    "last 2 versions"
  ]
}
```

新页面开发

1、新增页面

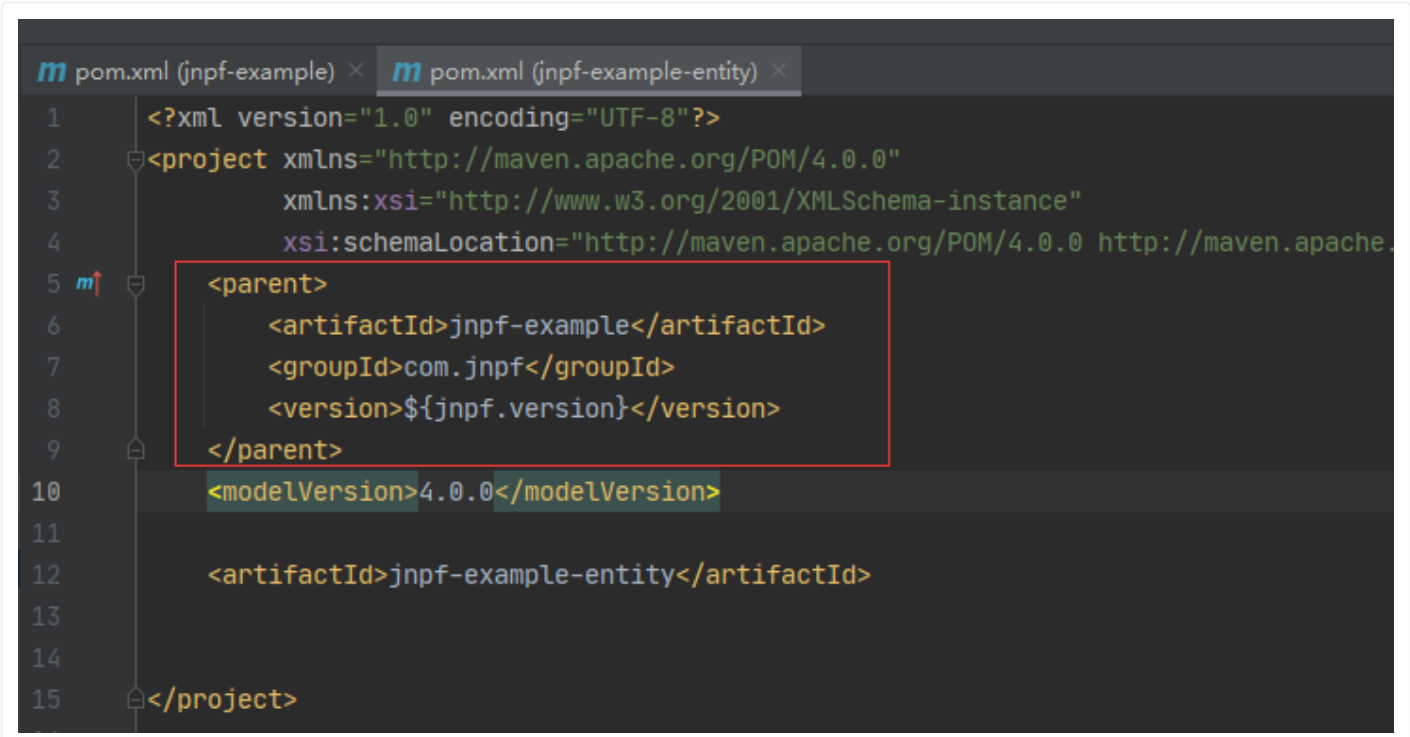
1.创建文件

在`src/views`任一目录下新建*xxx.vue*文件 例：`extend/barCode/index.vue`

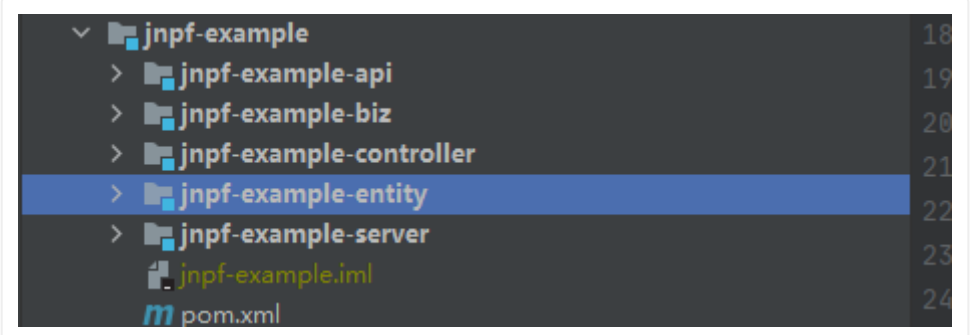


2、动态路由配置

打开菜单管理->选择Web菜单->点击新建按钮

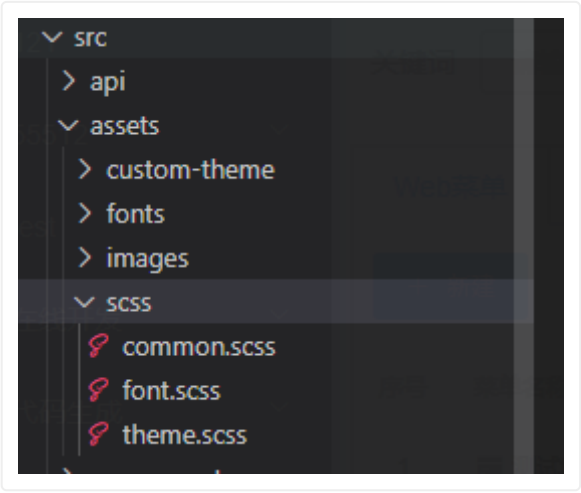


新增菜单后，通过管理员分配权限后，即可在相对应位置看到菜单，点击菜单打开页面，说明新建页面成功



2、新增样式

页面的样式，全局公用的样式放在 `src/asssets/scss/common.scss` 内



每一个页面的样式就写在当前页面下面，请记住加上 `scoped` 或者命名空间，避免造成全局的样式污染。

```
<style lang="scss" scoped>
.BarCode-container {
  padding: 0;
  #qrcode {
    width: 265px;
    height: 265px;
    border: 1px solid #eee;
  }
  #barcode {
    width: 265px;
    height: 80px;
    border: 1px solid #eee;
  }
}
</style>
```

和服务端进行交互

前端请求流程

在主项目前端中，一个完整的前端UI交互到服务端处理流程是这样的：

- UI 组件交互操作；
- 调用统一管理的 api service 请求函数；

- 使用封装的 request.js 发送请求；
- 获取服务端返回；
- 更新 data；

从上面的流程可以看出，为了方便管理维护，统一的请求处理都放在 `src/api` 文件夹中，并且一般按照 model 纬度进行拆分文件，如：

```
api/
  extend/
        bigData.js
        document.js
 onlineDev/
        map.js
        protal.js
 ...
```

request.js

其中 `src/utils/request.js` 是基于 axios 的封装，便于统一处理 POST，GET 等请求参数，请求头，以及错误提示信息等。具体可以参看 request.js。 它封装了全局 `request拦截器`、 `response拦截器`、 `统一的错误处理`、 `统一做了超时处理`、 `baseURL设置` 等。

请求示例

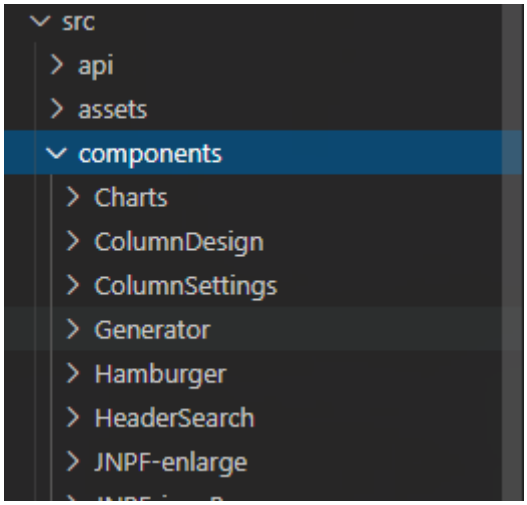
```
// api/extend/bigData.js
import request from '@/utils/request'
// 获取数据列表（分页）
export function BigDataList(data) {
  return request({
    url: '/api/Extend/BigData',
    method: 'get',
    data
  })
}


// views/extend/bigData/index
import { BigDataList } from '@/api/extend/bigData'
export default {
  data() {
   keyword: '',
```

```
      list: [],
      total: 0,
      listLoading: true,
      listQuery: {
        currentPage: 1,
        pageSize: 20,
        sort: 'desc'
      }
  },
  methods: {
    initData() {
      this.listLoading = true
      let query = {
        ...this.listQuery,
        keyword: this.keyword
      }
      BigDataList(query).then(res => {
        this.list = res.data.list
        this.total = res.data.pagination.total
        this.listLoading = false
      })
    }
  }
}
```

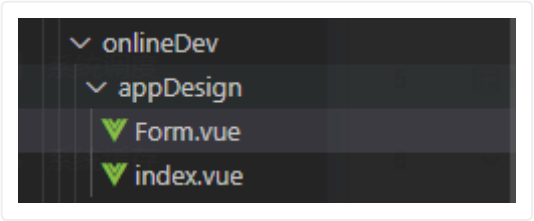
组件开发及使用

1、自定义组件开发及使用

全局的组件放在 `src/components` 目录下，如树形下拉框，搜索等组件



每个页面或者模块特定的业务组件则会写在当前 views 下面，如 `Form.vue`



自定义组件使用

全局注册

在 `src/components/index.js` 中引入

```
import JNPFTreeSelect from '@/components/JNPF-treeSelect'

export default {
  install(Vue, options) {
    Vue.component('JNPFTreeSelect', JNPFTreeSelect)
  }
}
```

局部注册

在要使用组件的页面内引入

```
<script>
import JNPFTreeSelect from '@/components/JNPF-treeSelect'
export default {
  components: { JNPFTreeSelect },
}
</script>
```

2、引入外部组件

除了 `element-ui` 组件以及脚手架内置的业务组件，有时我们还需要引入其他外部组件，这里以引入 `vue-esign` 为例进行介绍。

引入依赖

```
npm install vue-esign --save
```

> 加上 `--save` 参数会自动添加依赖到 package.json 中去

外部组件使用

全局注册

在`src/main.js`中引入

```
import vueEsign from 'vue-esign'
Vue.component('vueEsign', vueEsign)
```

局部注册

在要使用组件的页面内引入

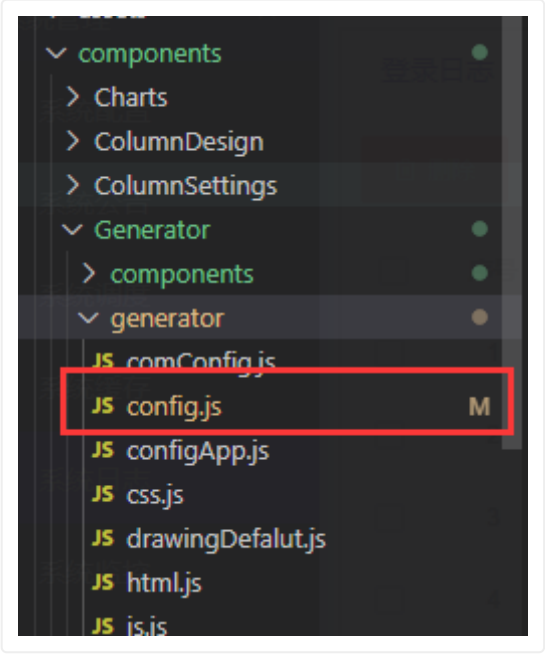
```
<script>
import vueEsign from 'vue-esign'
export default {
  components: { vueEsign },
}
</script>
```

表单设计器增加控件示例

> 我们以 `input` 文本框为例

1、添加左侧控件按钮

在`src\components\Generator\generator\config.js`中新增控件



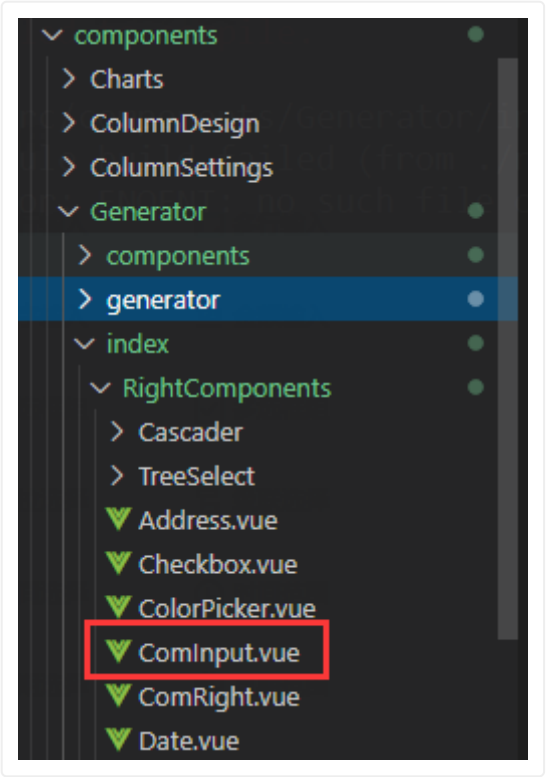

```
{
        // 组件的自定义配置
    __config__: {
      jnpfKey: 'comInput',
      label: '单行输入',
      labelWidth: null,
      showLabel: true,
      tag: 'el-input',
      tagIcon: 'icon-ym icon-ym-generator-input',
      defaultValue: undefined,
      required: false,
      layout: 'colFormItem',
      span: 24,
      dragDisabled: false,
      regList: [],
      trigger: 'blur'
    },
        // 组件的插槽属性
    __slot__: {
      prepend: '',
      append: ''
    },
        // 其余的为可直接写在组件标签上的属性
    placeholder: '请输入',
    style: { width: '100%' },
    clearable: true,
    'prefix-icon': '',
    'suffix-icon': '',
    maxlength: null,
    'show-word-limit': false,
```

```
    readonly: false,
    disabled: false
  }
```

在页面中就可以看到左侧增加了控件按钮

2、添加右侧属性配置页面

在 `src/components/Generator/index/RightComponents` 目录下新增属性配置页面ComInput.vue



并在 `src/components/Generator/index/RightPanel.vue` 引入即可

```
// RightPanel.vue
<template>
<JNPFComInput v-if="activeData.__config__.jnpfKey==='comInput'"
 :active-data="activeData" />
</template>

<script>
import JNPFComInput from './RightComponents/ComInput'

export default {
  components: { JNPFComInput },
}
</script>
```

js日期控件如何获取当天日期


企业基础
岗位
员工类型
办公地点
请输入
备注
入职时间
请选择
员工状态
试用
离职时间
请选择
控件栅格
标题宽度
0
默认值
选择默认值
时间类型
日(date)
时间格式
yyyy-MM-dd
能否清空
是否只读
是否禁用
是否必填
组件事件
日期默认当天怎么设置?


组件属性
表单属性
表单尺寸
中等
较小
迷你
标签对齐
左对齐
右对齐
顶部对齐
标题宽度
100
栅格间隔
15
弹窗类型
全屏弹窗
表单样式
默认风格
表单宽度
100%
表单按钮
确定
确 定
取消
取 消

在线js方式：

```
({ formData, setFormData, setShowOrHide, setRequired, setDisabled, request, getFieldOptions, setFieldOptions }) => {
    // 在此编写代码
    var a = (new Date()).toLocaleDateString()//获取当前日期
    var nowdate = ((new Date(a)) / 1000) * 1000;//把当前日期变成时间戳
    console.log(nowdate);
    setFormData("f_orderdate", nowdate);//设置某组件日期
}
```

默认主页修改说明

如何修改默认主页：

- **1. 注释默认的”home”页**

Src/router/modules下**base**.js

README.md | zh.js | base.js | index.vue | main.js | index.js

```
1  // 基础路由
2  const baseRouter = [
3  //   {
4  // path: '/home',
5  // component: (resolve) => require(['@/views/dashboard'], resolve),
6  // name: 'home',
7  //  meta: {
8  //  title: 'home',
9  //  icon: 'icon-ym icon-ym-nav-home',
10 //  affix: true,
11 //  zhTitle: '首页'
12 //  }
13 // },
14 {
15   path: '/dashboard',
16   component: (resolve) => require(['@/views/basic/dashboard'], resol
17   name: 'dashboard',
18   meta: {
19     title: 'dashboard',
```

- **2.修改登录后的页面：**
  Src/views/login 下index.vue，第124行的home 改成 dashboard

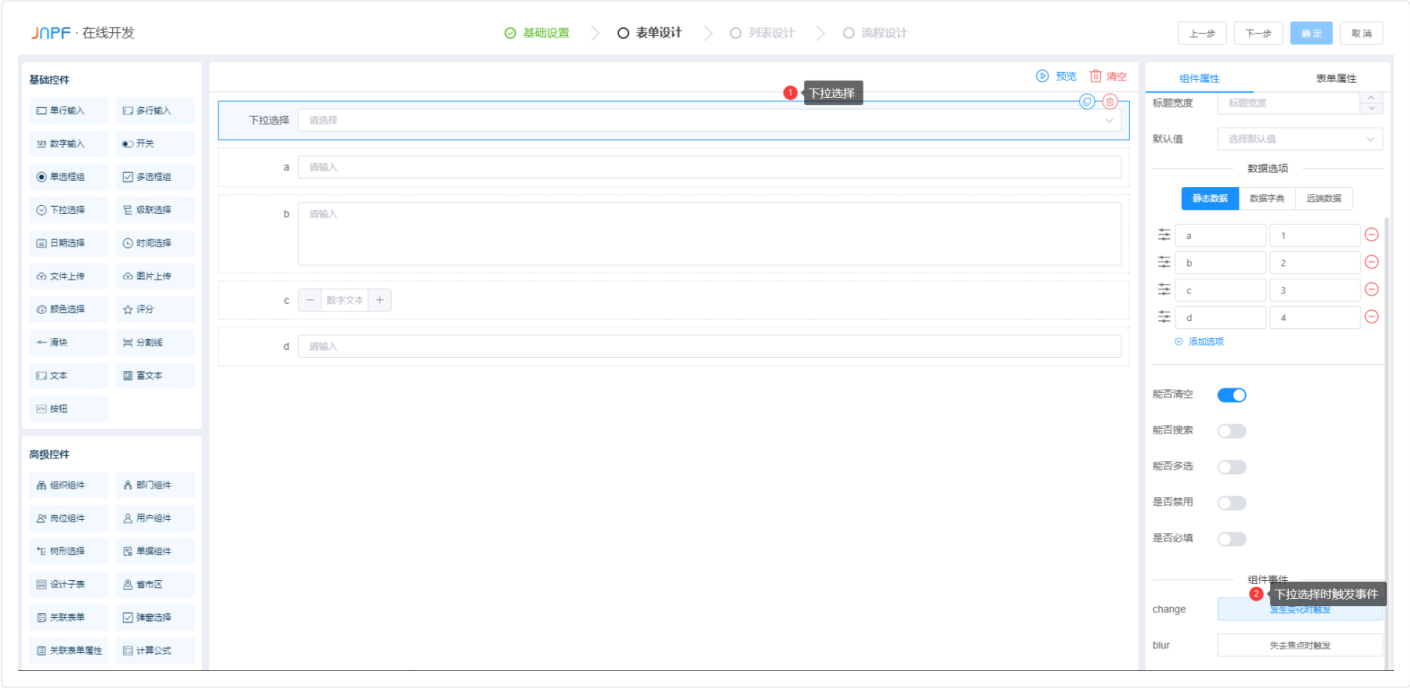
README.md | zh.js | base.js | index.vue | main.js | index.js

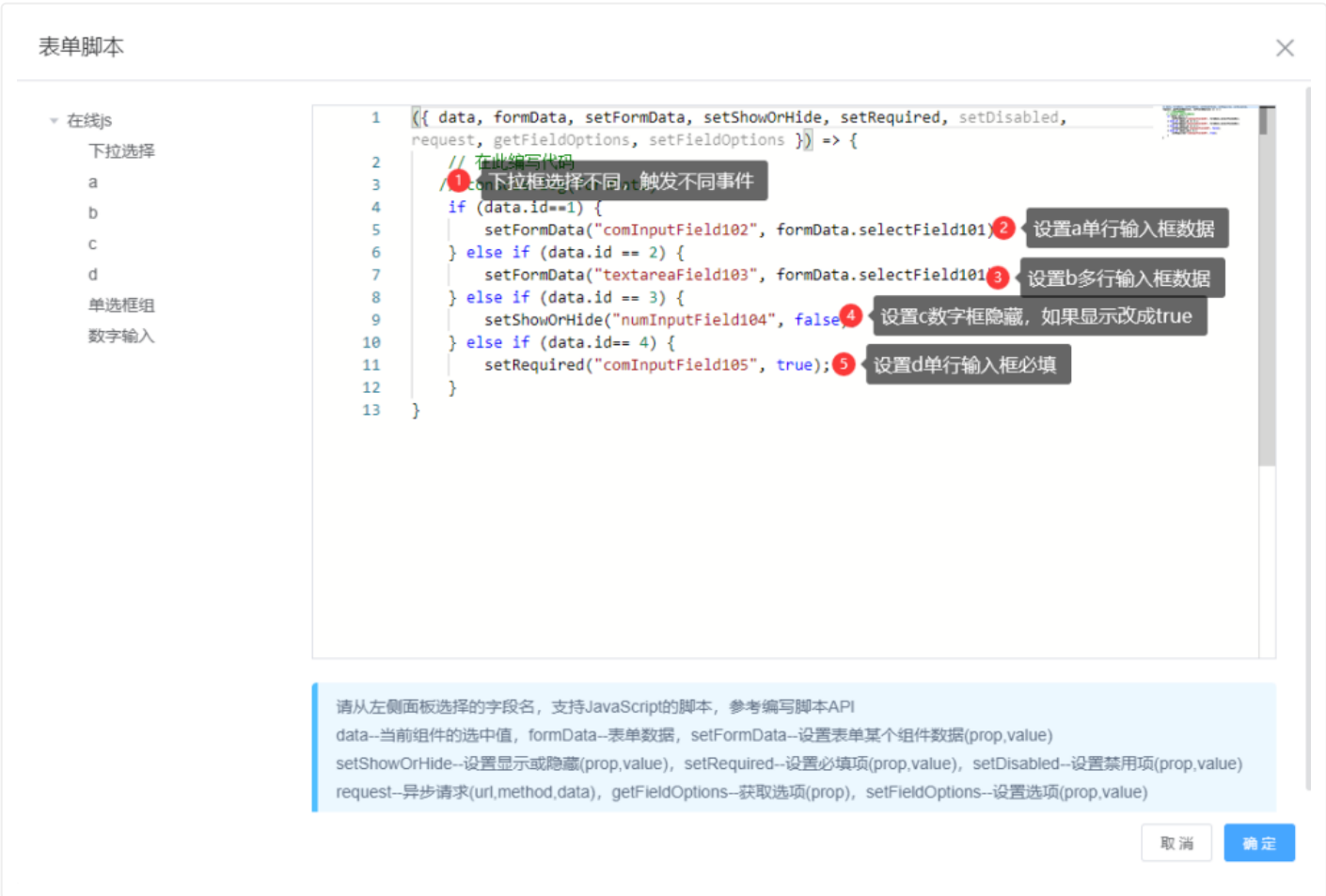
```
112       this.capsTooltip = key && key.length === 1 && key >= 'A' && key <= 'Z'
113     },
114     handleLogin() {
115       if (this.loading) return
116       this.$refs.loginForm.validate(valid => {
117         if (valid) {
118           this.loading = true
119           this.$store.commit( type: 'user/SET_LOGIN_LOADING',  payload: true)
120           this.$store
121             .dispatch( type: 'user/login', this.loginForm)
122             .then(() => {
123               this.$router.push( location: {
124                 path: this.redirect || '/home',
125                 query: this.otherQuery
126               })
127             })
128             .catch(() => {
129               this.$store.commit( type: 'user/SET_LOGIN_LOADING',  payload: false)
130             })
131         } else {
132           return false
133         }
134       })
```

- **注意：门户目前无“默认”门户的设计，建议设计好一个门户后，给所有角色分配一个门户，不然“无门户”的用户登录后看不到门户。**

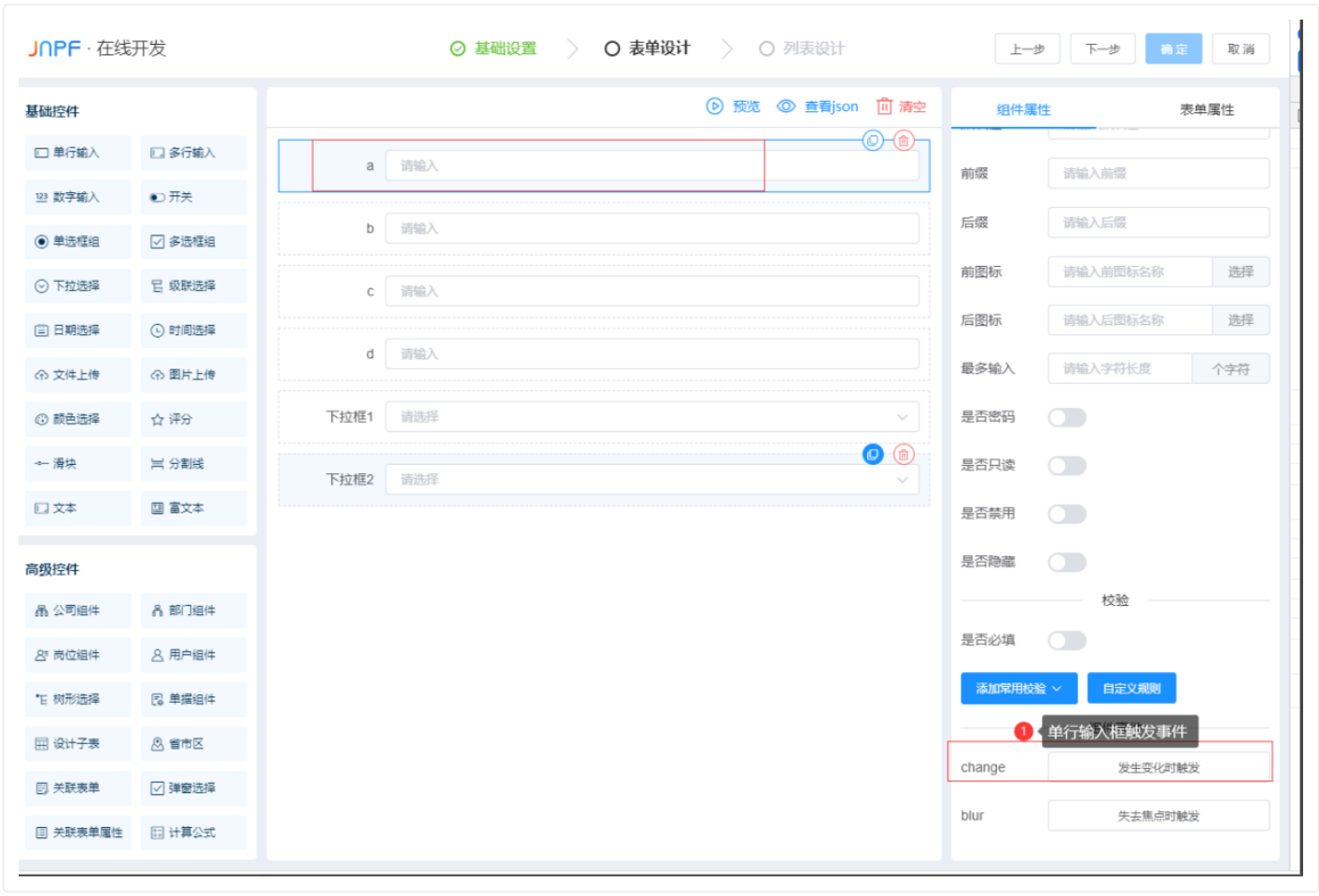
在线js示例

1.下拉选择触发示例





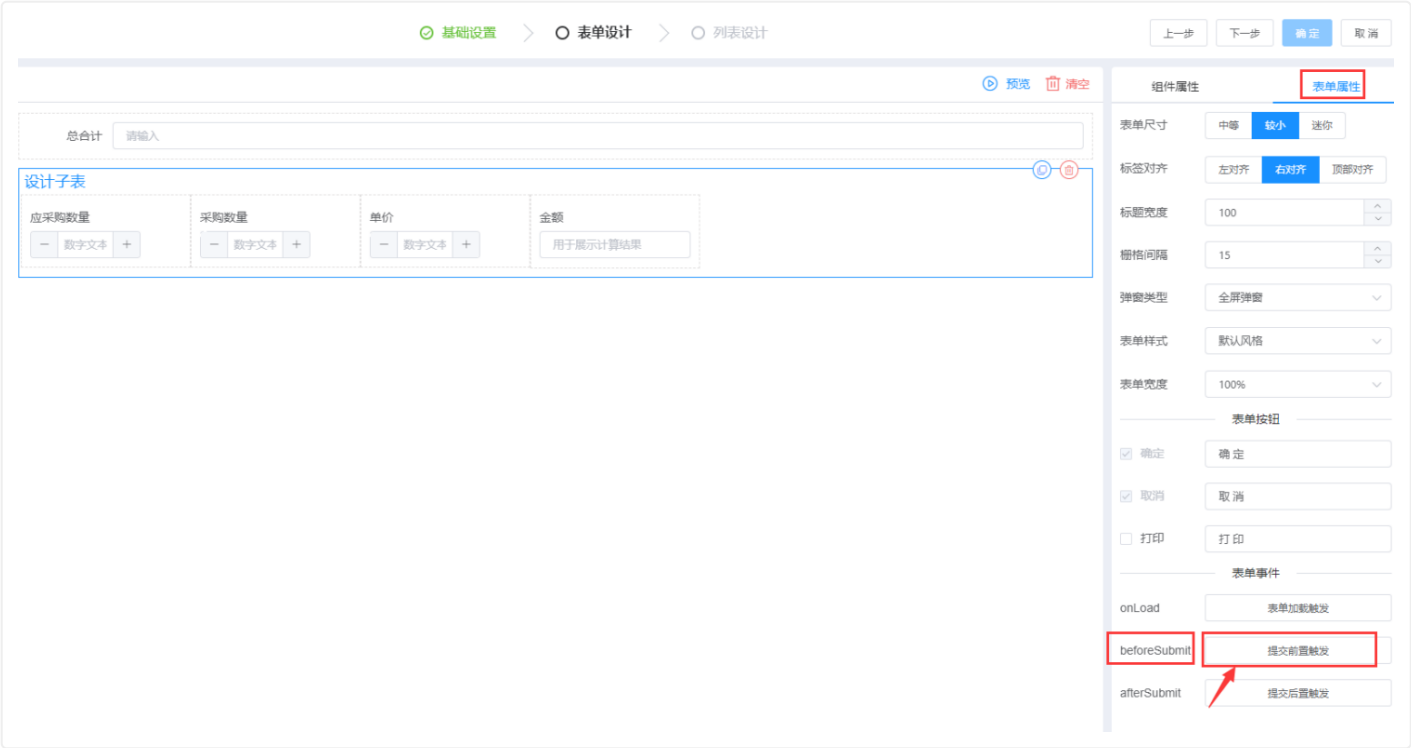
2.单行输入框触发示例


JNPF · 在线开发
基础设置
表单设计
列表设计
上一步
下一步
确定
取消
基础控件
单行输入
多行输入
数字输入
开关
单选框组
多选框组
下拉选择
级联选择
日期选择
时间选择
文件上传
图片上传
颜色选择
评分
滑块
分割线
文本
富文本
高级控件
公司组件
部门组件
岗位组件
用户组件
树形选择
单据组件
设计子表
省市区
关联表单
弹窗选择
关联表单属性
计算公式
预览
查看json
清空
a
b
c
d
下拉框1
下拉框2
请输入
请选择
组件属性
表单属性
前缀
后缀
前图标
后图标
最多输入
是否密码
是否只读
是否禁用
是否隐藏
校验
是否必填
添加常用校验
自定义规则
1 单行输入框触发事件
change
发生变化时触发
blur
失去焦点时触发


表单脚本
在线js列子2
a
b
c
d
下拉框1
下拉框2
({ data, formData, setFormData, setShowOrHide, setRequired, setDisabled,
request, getFieldOptions, setFieldOptions }) => {
// 在此编写代码
console.log(data)
// 在此编写代码
setFormData("comInputField106", data) 1 设置a单行输入数值
if (data.id == 1) 2 当前组件的数据
setShowOrHide("comInputField105", true);
setRequired("comInputField107", false);
setDisabled("comInputField101", false);
} else {
setShowOrHide("comInputField105", false);
setRequired("comInputField101", true);
setDisabled("comInputField107", true);
}
console.log(getFieldOptions("selectField103")) 3 打印获取下拉框1选项
let list = [
{
"fullName": "选项三",
"id": "3"
}
]
setFieldOptions('selectField104', list) 4 设置下拉框2的选项
request('/api/extend/BigData', 'post').then(res => {
5 设置异步请求
请从左侧面板选择的字段名，支持JavaScript的脚本，参考编写脚本API
data--当前组件的选中值，formData--表单数据，setFormData--设置表单某个组件数据(prop,value)
setShowOrHide--设置显示或隐藏(prop,value)，setRequired--设置必填项(prop,value)，setDisabled--设置禁用项(prop,value)
request--异步请求(url,method,data)，getFieldOptions--获取选项(prop)，setFieldOptions--设置选项(prop,value)
取消
确定

```
({ data, formData, setFormData, setShowOrHide, setRequired, setDisabled, request, ge
tFieldOptions, setFieldOptions }) => {
    // 在此编写代码
    console.log(data)
    // 在此编写代码
    setFormData("comInputField106", data);
    if (data.id == 1) {
        setShowOrHide("comInputField105", true);
        setRequired("comInputField107", false);
        setDisabled("comInputField101", false);
    } else  {
        setShowOrHide("comInputField105", false);
        setRequired("comInputField101", true);
        setDisabled("comInputField107", true);
    }
    console.log(getFieldOptions("selectField103"))
    let list = [
        {
            "fullName": "选项三",
            "id": "3"
        }
    ]
    setFieldOptions('selectField104', list)
    request('/api/extend/BigData', 'post').then(res => {
        console.log(res)
    })
}
```

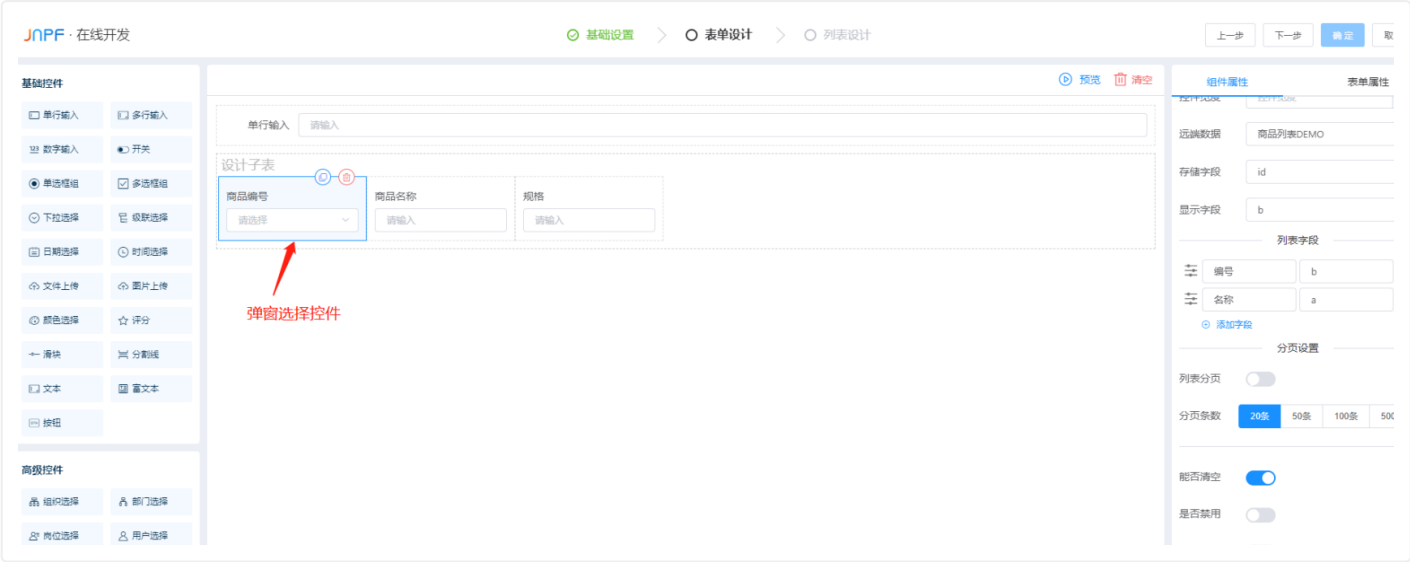
3.计算公式：计算子表金额，赋值给总合计显示

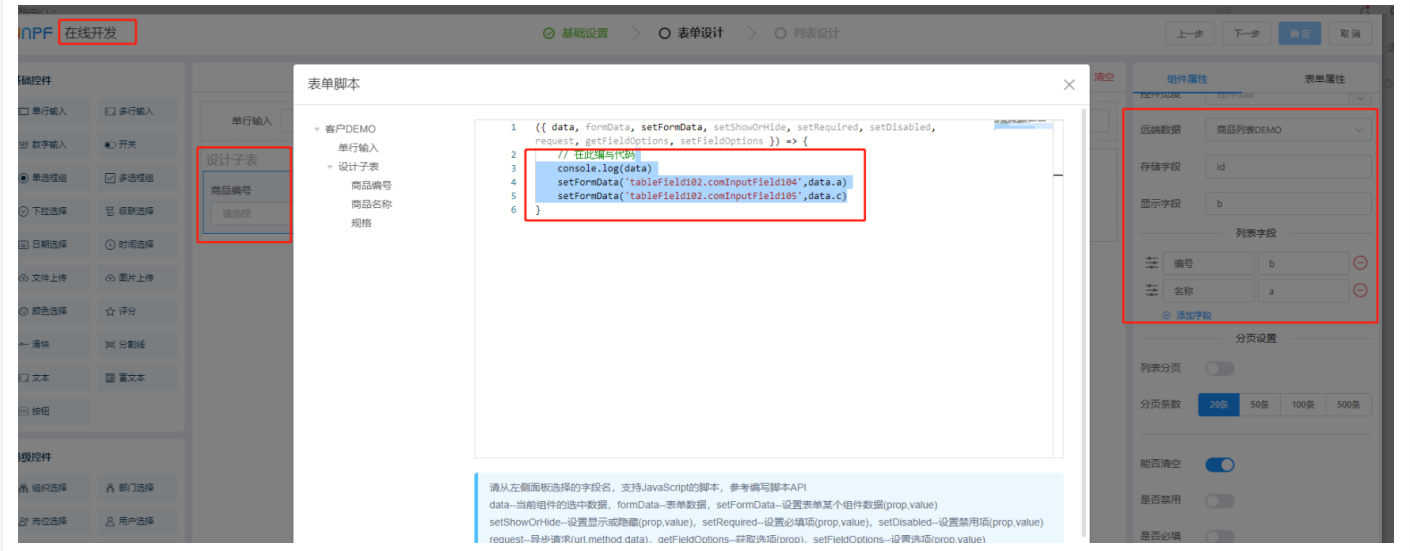



```
({ formData, setFormData, setShowOrHide, setRequired, setDisabled, request, getFieldOptions, setFieldOptions }) => {
    // 在此编写代码
    var a = 0;
    for (let i in formData.tableField102) {
        a += formData.tableField102[i].calculateField106;
```

```
    }
    setFormData('comInputField101', a);

    // 继续执行返回true
    return true
}
```

4.弹窗如何把产品属性赋值给其他字段



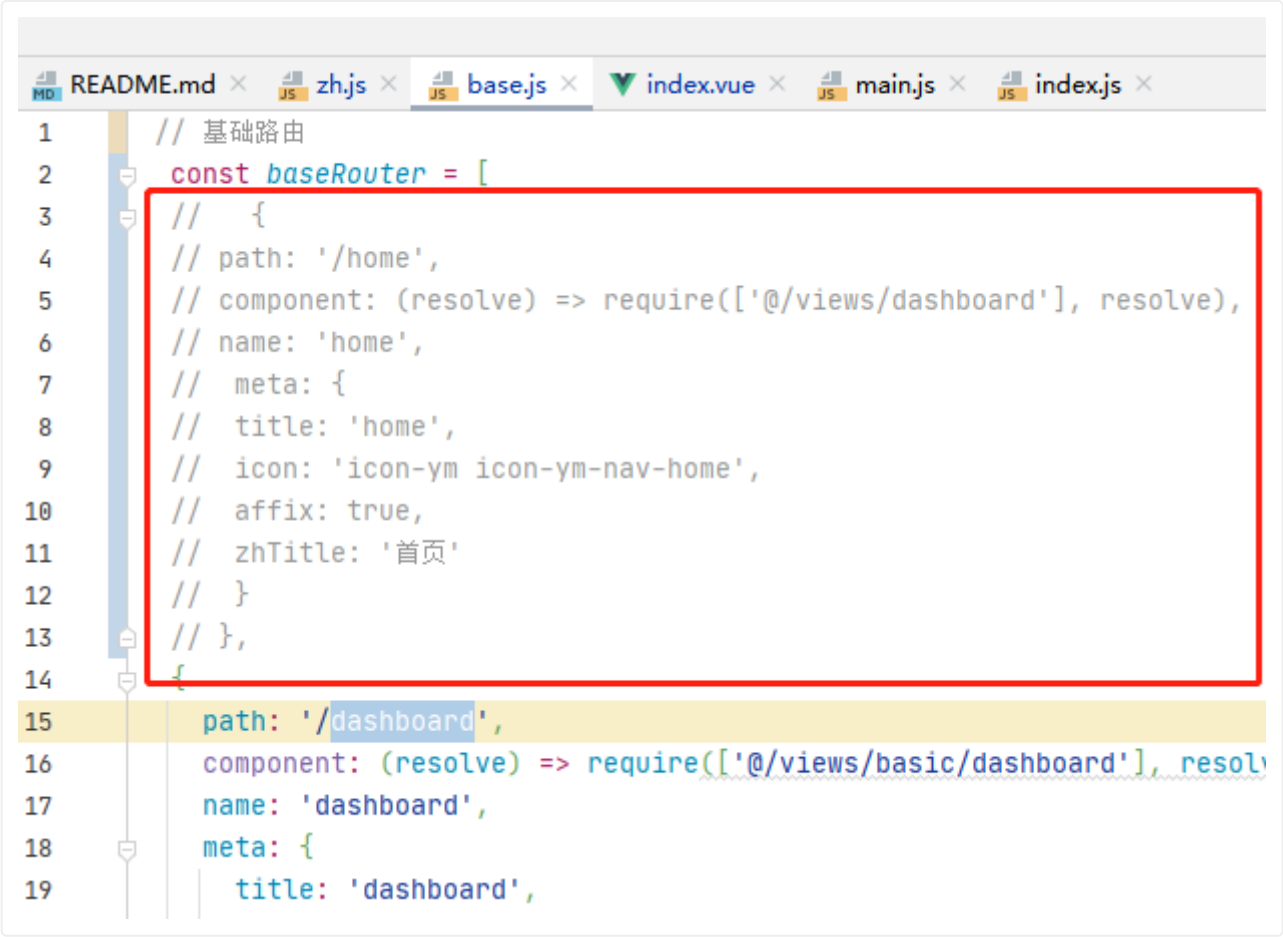



```
({ data, formData, setFormData, setShowOrHide, setRequired, setDisabled, request, ge
tFieldOptions, setFieldOptions }) => {
    // 在此编写代码
    console.log(data)
    setFormData('tableField102.comInputField104',data.a)
    setFormData('tableField102.comInputField105',data.c)
}
```

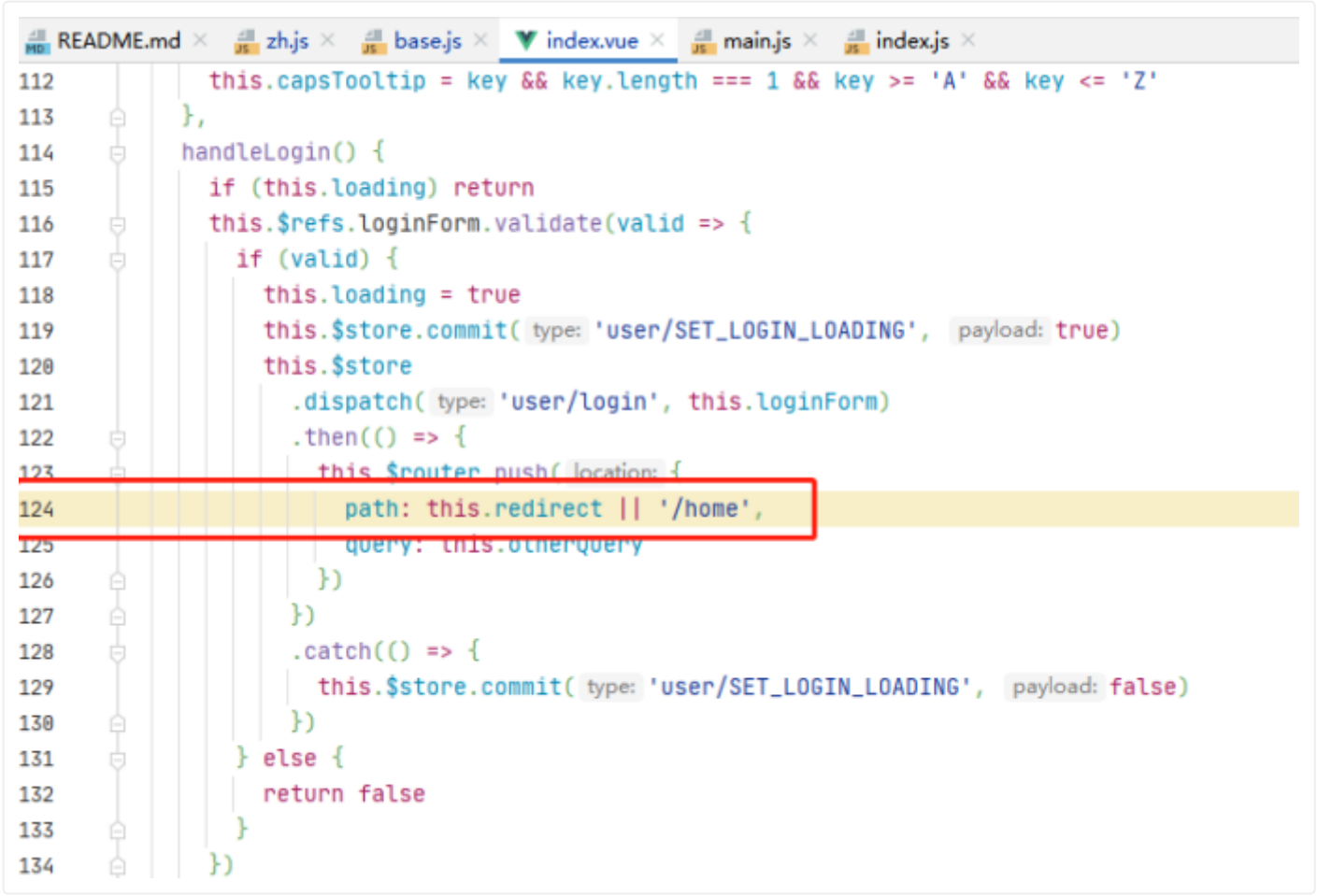
修改默认主页

一、注释默认的 `Home` 页：

打开 `jnpf-web` 项目中的 `src/router/modules/base.js`



二、修改登录后的页面：`src/views/login/index.vue`，第124行的 `home` 改成 `dashboard`



注意：门户目前无“默认”门户的设计，建议设计好一个门户后，给所有角色分配一个门户，不然“无门户”的用户登录后看不到门户。

jnpf-app

代码结构说明

```
jnpf-app
    ├── api - 请求接口
    ├── assets - 公共样式
    ├    ├── iconfont - 字体图标
    ├    └── scss - 公共样式
    ├── components - 组件
    ├── filters - 过滤方法
    ├── hybrid - App端存放本地html文件的目录
    ├── libs - 公共方法
    ├── pages - 业务页面文件存放的目录
    ├    ├── apply - 功能页面
    ├    ├── componentLibrary - 组件示例
    ├    ├── index- 底部导航页
    ├    ├── launch - 启动页
    ├    ├── login - 登录页
```

```
├    ├──── message - 消息页
├    ├──── my - 个人中心页
├    └──── workFlow - 工作流页
├──── static - 静态资源
├──── store - 状态管理仓库
├──── uni_modules - 存放[uni_module](/uni_modules)
├──── utils - 公共方法、环境配置
├──── unpackage - 一般存放运行或发行的编译结果
├    ├──── dist - 功能页面
├    └──── res - 静态资源
├    ├    ├──── icons - 图标
├    ├    └──── startup - 启动图
├──── App.vue - 入口页面
├──── main.js - Vue初始化入口文件
├──── manifest.json - 配置应用名称、appid、logo、版本等打包信息
├──── pages.json - 配置页面路由、导航条、选项卡等页面类信息
├──── template.h5.html - h5模板
└──── uni.scss - uni-app内置的常用样式变量
```

快速开始

本地环境准备

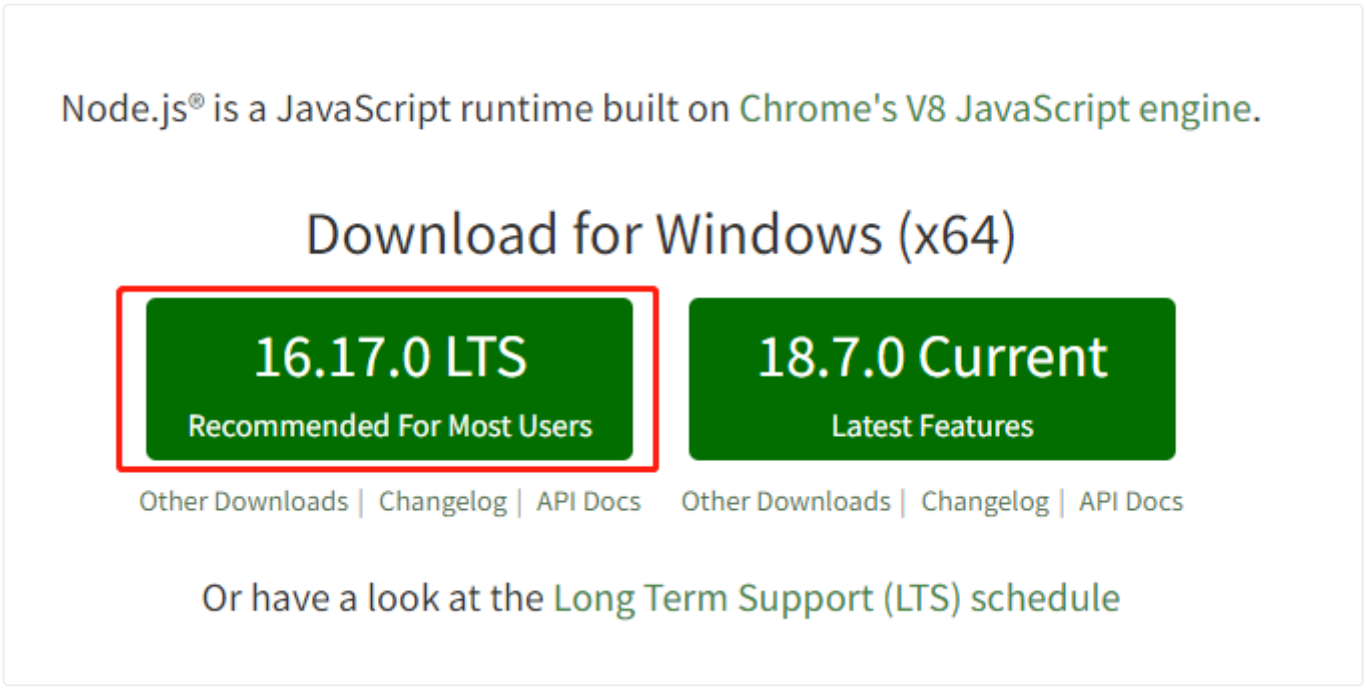
开发环境准备

Windows环境

> 本档环境基于 `Windows 10 x64`

Node.js

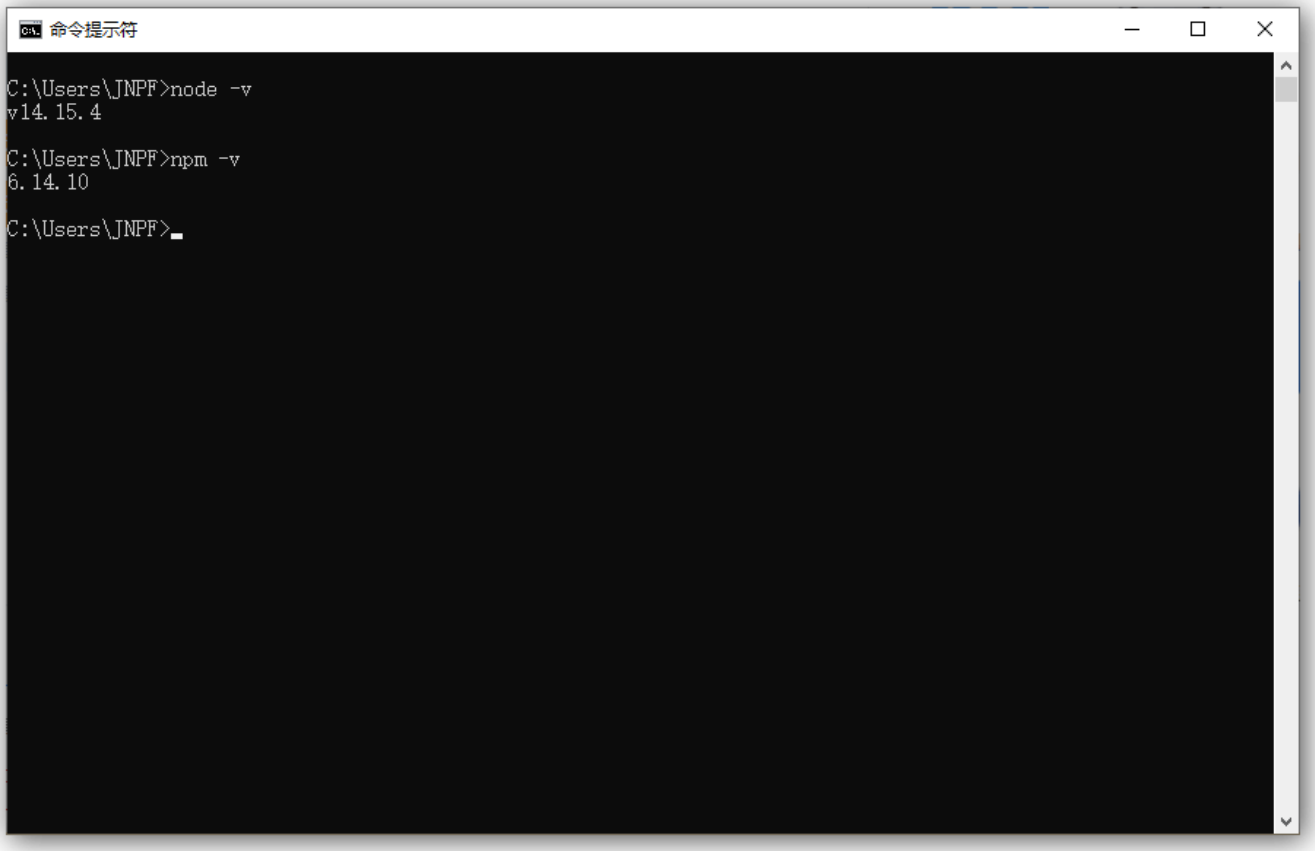
打开 `https://nodejs.org/en/` ，下载16.17.0的LTS版本



**

按提示完成安装即可。之后打开命令依次输入如下命令查看安装信息

```
node -v
npm -v
```



安装小程序开发者工具

打开 `https://developers.weixin.qq.com/miniprogram/dev/devtools/download.html` ,下载安装 `微信小程序开发者工具`

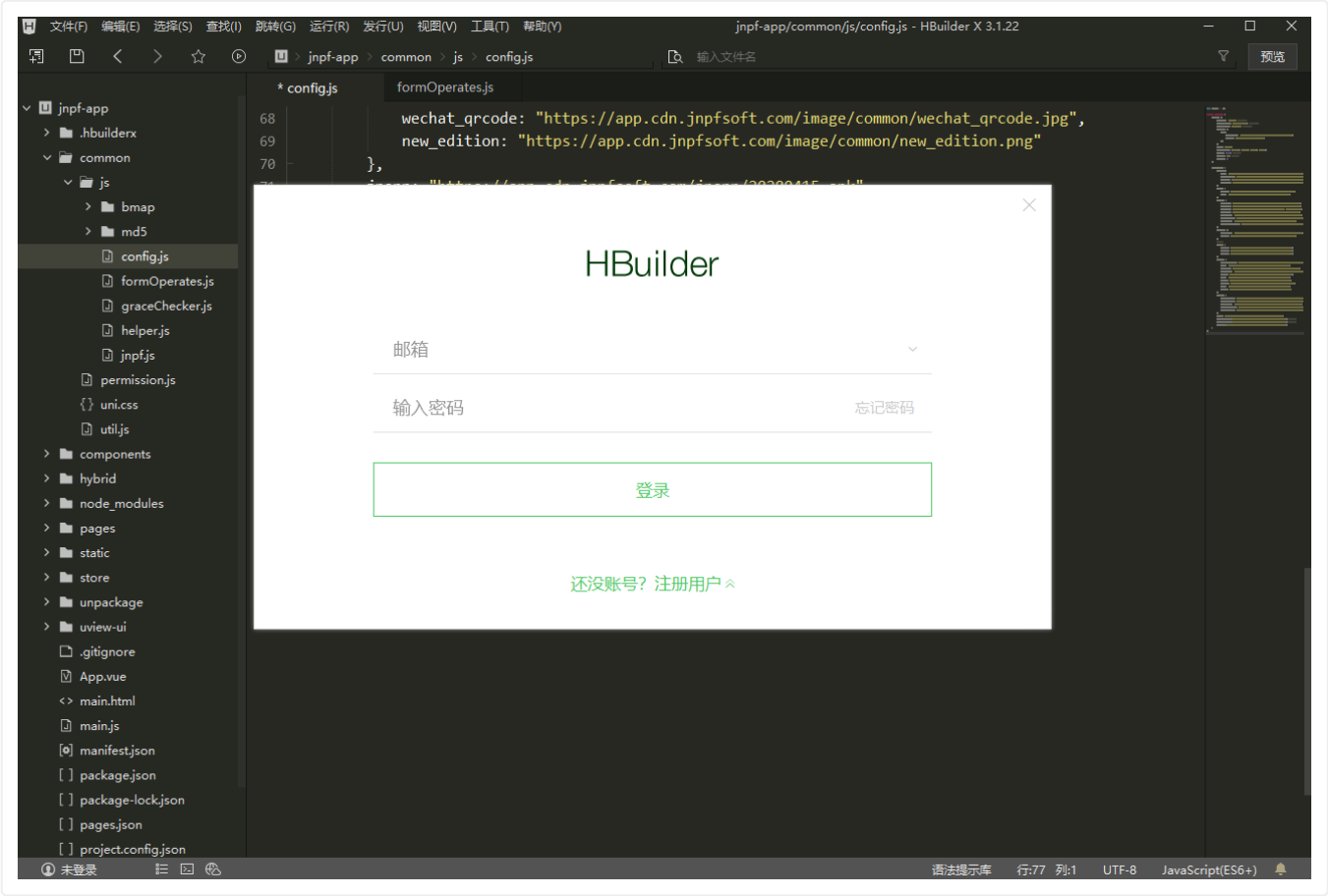
安装uniapp开发工具

打开 `https://www.dcloud.io/hbuilderx.html` ,下载安装 `HBuilder X` （ `Windows版-App开发版` ）

注册账号

- 1、注册dcloud开发者账号，地址：https://dev.dcloud.net.cn

- 2、在HBuilderX编辑器中登录

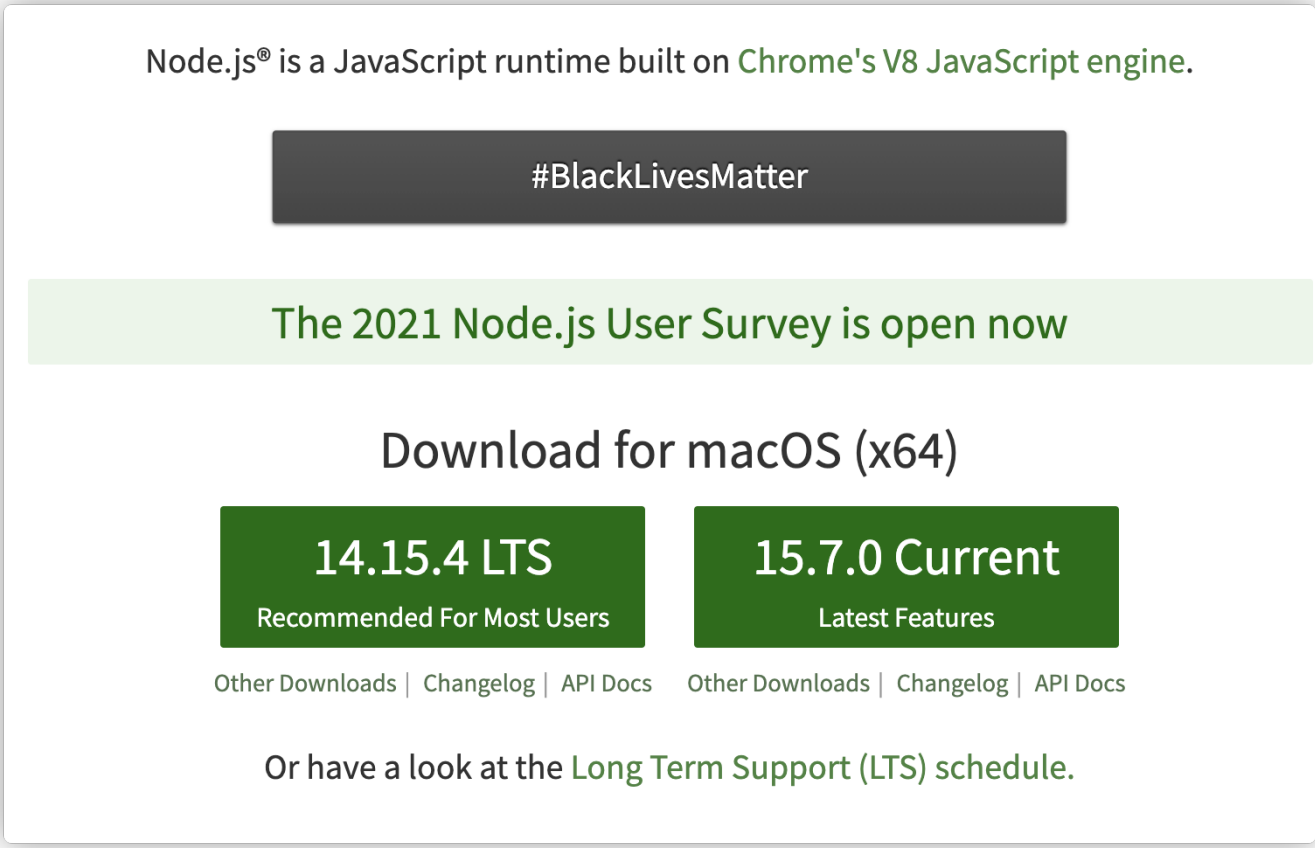


MacOS环境

> 本文档环境基于 `macOS Big Sur 11.2`

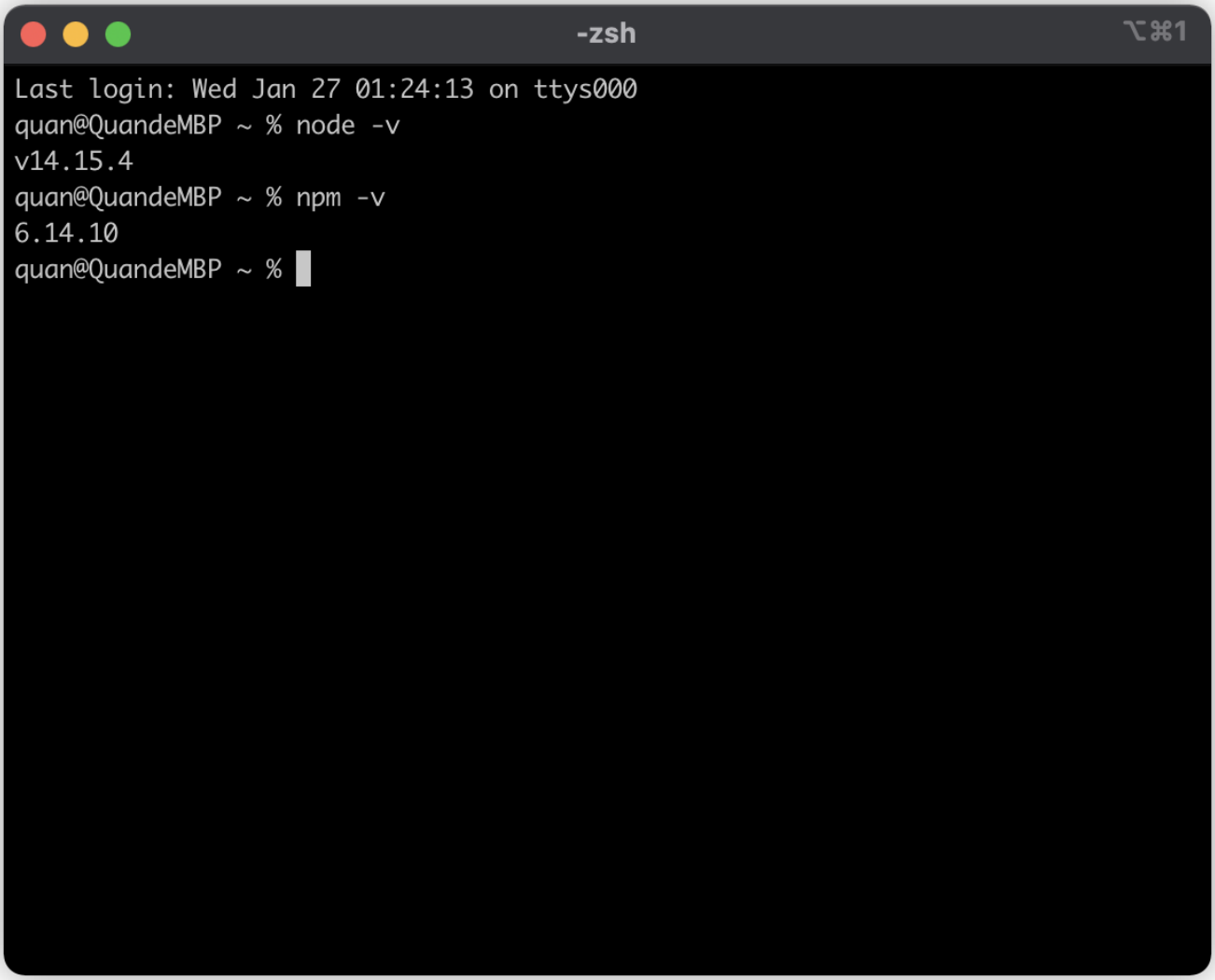
Node.js 最新LTS版本

打开 `https://nodejs.org/en/` ，下载16.17.0的LTS版本，按提示完成安装即可



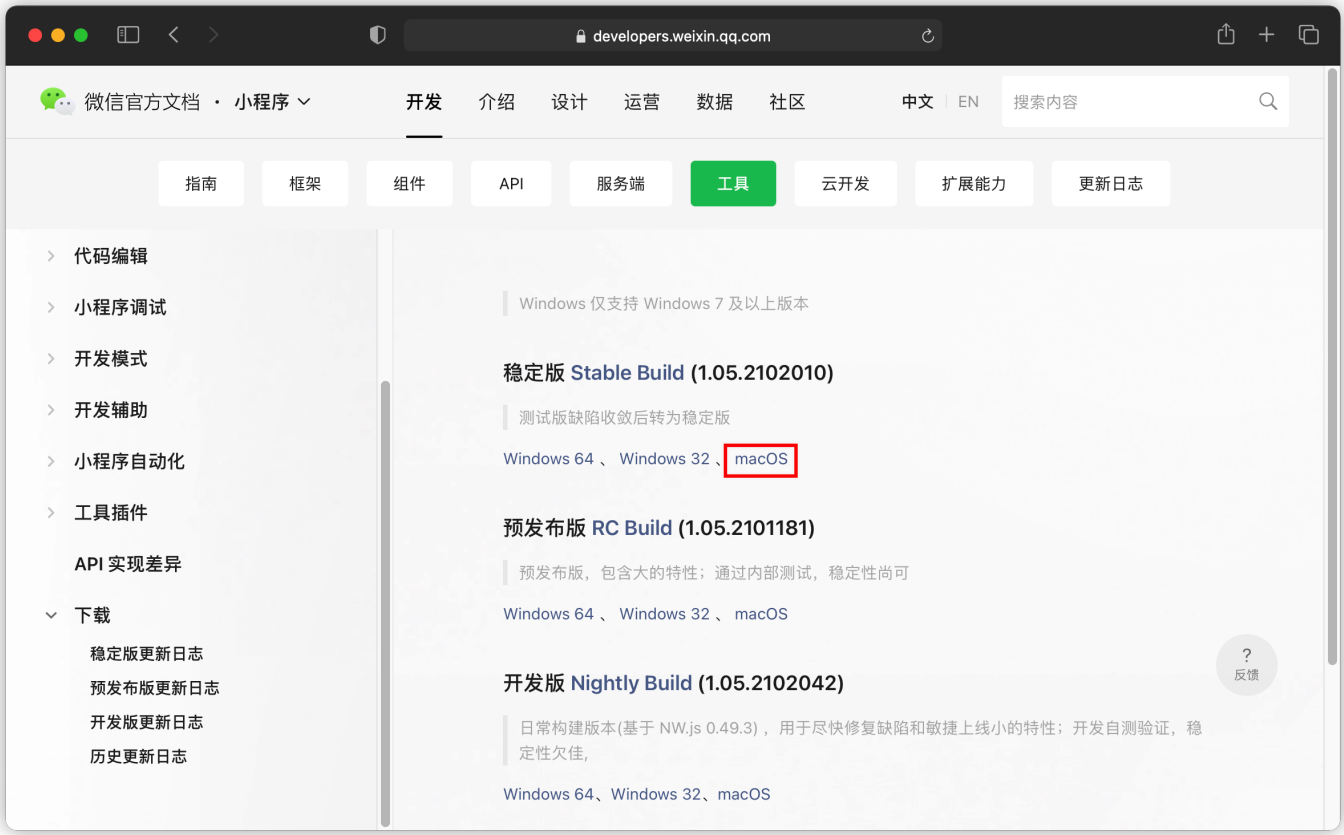
打开命令依次输入如下命令查看安装信息

```
node -v
npm -v
```



安装小程序开发者工具

打开 `https://developers.weixin.qq.com/miniprogram/dev/devtools/download.html` ,下载安装 `微信小程序开发者工具`



安装移动端开发工具

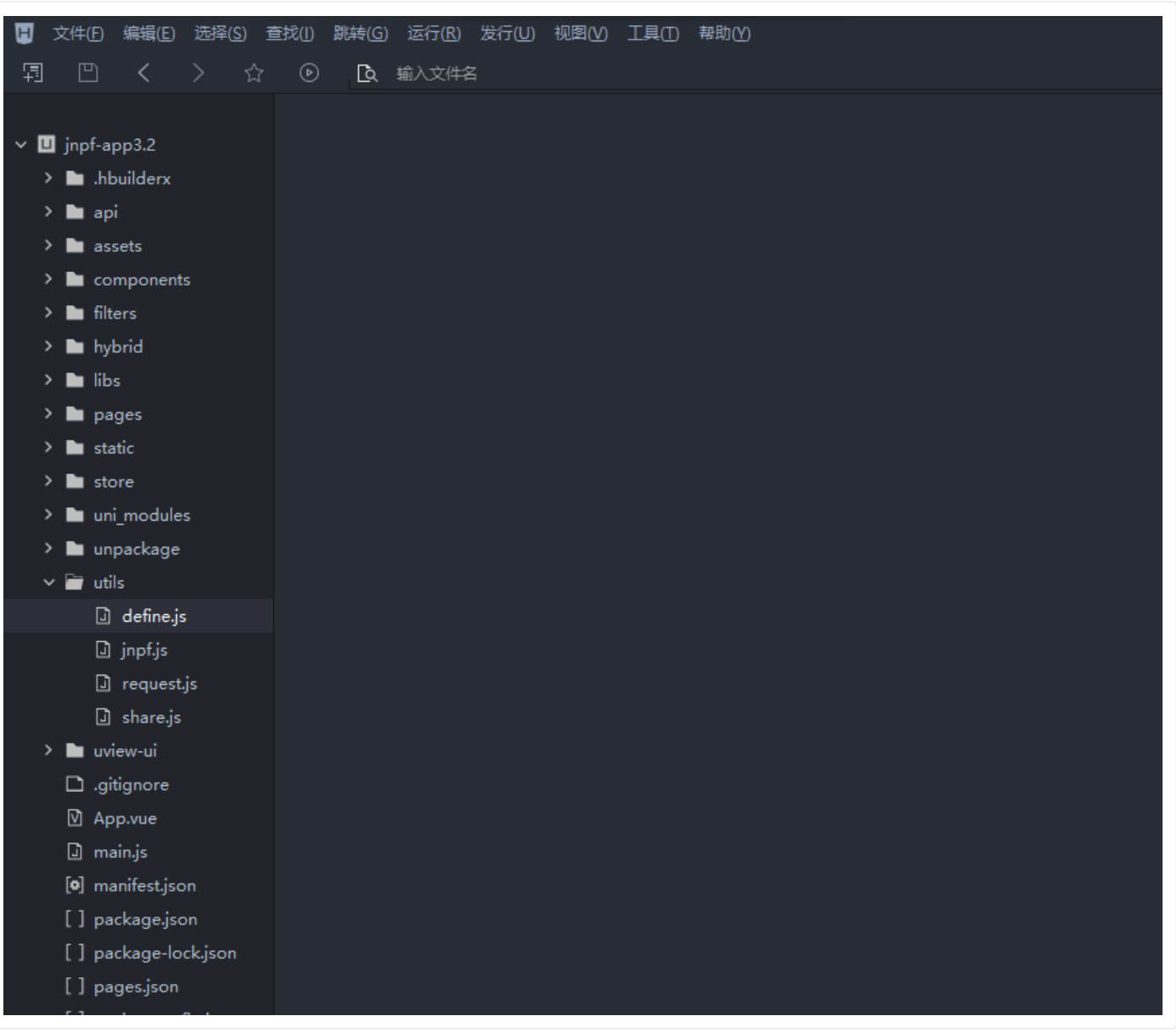
打开 `https://www.dcloud.io/hbuilderx.html` ,下载安装 `HBuilder X` （ `MacOS版-App开发版` ）

注册开发者账号

移动端代码运行

项目导入

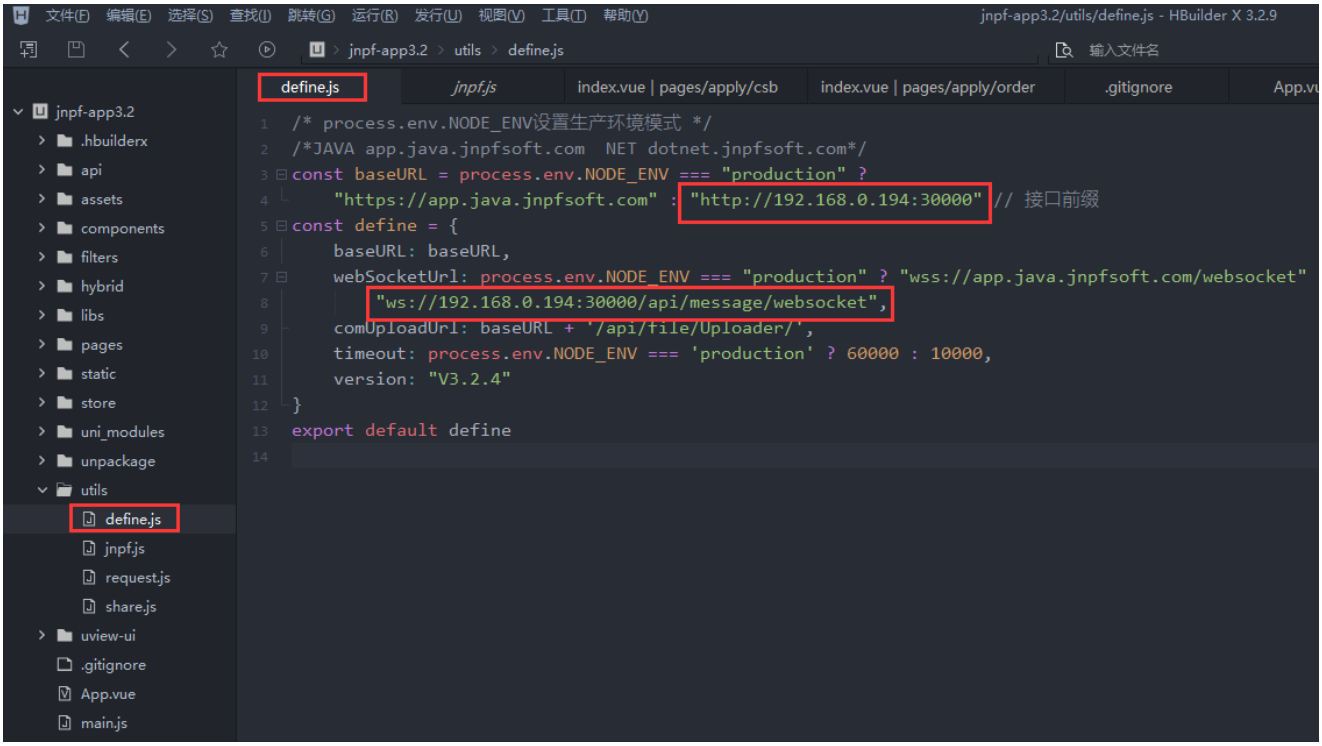
直接将 `jnpf-app` 项目拖到 `HBuilderX` 即可、



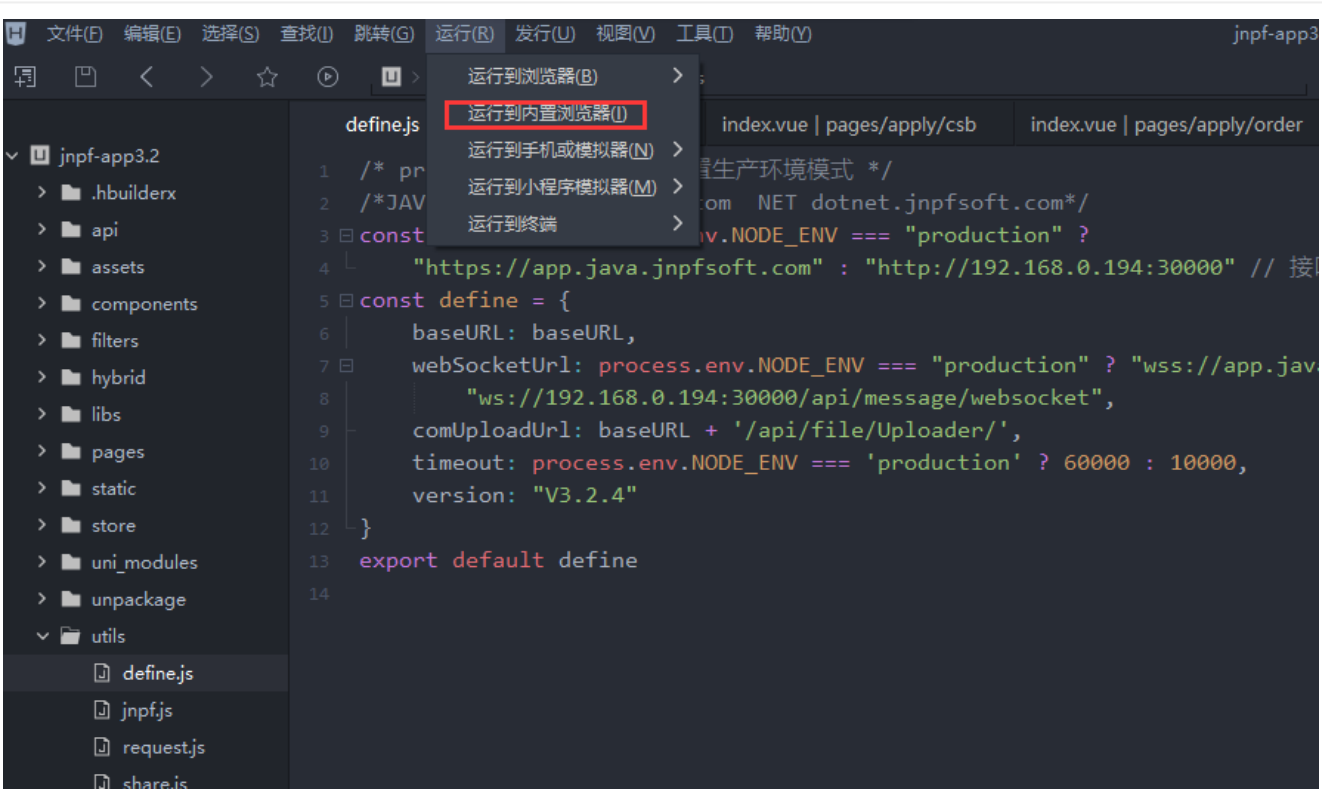
项目运行

1. 登录uini插件市场(https://ext.dcloud.net.cn)下载sass插件

2. 打开jnpf-app/utils/define.js修改接口配置



3. 运行



开发指南

一、必要知识点

1、uniapp 框架熟悉 (框架配置、跟硬件交互等以及官方的一些常见问题)
https://uniapp.dcloud.io/collocation/pages
2、vue 相关知识(组件、状态管理等)
3、uview-ui 移动端UI框架
https://www.uviewui.com

二、开发准备

1、开发工具：
a. HBuilder必须登录 【DCloud开发者账号】
b. 微信开发者工具
c.Android Studio（跨平台）
d.Xcode（MacOS）itunes(Win)
2、DCloud开发者账号（注册后需求完成相关实名认证）
3、微信小程序开发者账号（费用：300元/年，申请周期3-5个工作日）
4、Android证书（免费，直接本地生成）
https://ask.dcloud.net.cn/article/35777
5、苹果开发者账号及相关证书(必须先申请D-U-N-S码【免费】,苹果开发者一年99$,申请周期最快一个月)

三、分包加载配置

分包加载配置，此配置为小程序的分包加载机制。
subPackages 节点接收一个数组，数组每一项都是应用的子包，其属性值如下：

属性	类型	是否必填	描述
root	String	是	子包的根目录
pages	Array	是	子包由哪些页面组成，参数同 pages

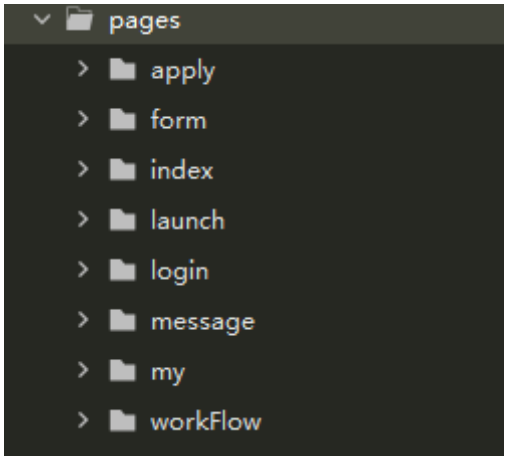
注意：
subPackages 里的pages的路径是 root 下的相对路径，不是全路径。
使用方法：
假设支持分包的 uni-app 目录结构如下：

```
┌─pages
│  ├─index
│  │  └─index.vue
│  └─login
│     └─login.vue
├─pagesA
│  ├─static
│  └─list
│     └─list.vue
├─pagesB
│  ├─static
│  └─detail
│     └─detail.vue
├─static
├─main.js
├─App.vue
├─manifest.json
└─pages.json
```

则需要在 pages.json 中填写

```
"subPackages": [{
        "root": "pages/message",          子包根目录
        "pages": [{
                "path": "userDetail/index",          root 下的相对路径
                "style": {
                    "navigationBarTitleText": "",
                    "navigationBarTextStyle": "white",
                    "navigationBarBackgroundColor": "#D8D8D8"
                }
            },
            {
                "path": "message/index",
                "style": {
                    "navigationBarTitleText": ""
                }
            },
            {
                "path": "messageDetail/index",
                "style": {
                    "navigationBarTitleText": ""
                }
            },
            {
                "path": "im/index",
                "style": {
                    "navigationBarTitleText": ""
                }
            }
        ]
    },
```

新增的文件请放pages根目录下



分包预下载

开发者可以通过配置，在进入某个页面时，由框架自动预下载可能需要的分包，提升进入后续分包页面时的启动速度

配置方法

```
"preloadRule": {
    "pages/index/index": {
        "network": "all",
        "packages": ["pages/message"]
    },
    "pages/index/workFlow": {
        "network": "all",
        "packages": ["pages/workFlow"]
    },
    "pages/index/apply": {
        "network": "all",
        "packages": ["pages/apply"]
    },
    "pages/index/my": {
        "network": "all",
        "packages": ["pages/my"]
    }
},
```

preloadRule 中，key 是页面路径，value 是进入此页面的预下载配置，每个配置有以下几项：

字段	类型	必填	默认值	说明
packages	StringArray	是	无	进入页面后预下载分包的 root 或 name。__APP__表示主包。
network	String	否-	wifi	在指定网络下预下载，可选值为：all: 不限网络，wifi: 仅wifi下预下载

链接

小程序官网：https://developers.weixin.qq.com/miniprogram/dev/framework/subpackages.html

uni-app官网：https://uniapp.dcloud.io/collocation/pages?id=subpackages

生产部署

Nginx配置

```
# JNPF-START
    #设置上传文件的大小
    client_max_body_size 100m;

    #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
    #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    client_header_buffer_size 128k;

    #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
    large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';

    # 预检命令的缓存，如果不缓存每次会发送两次请求
    add_header Access-Control-Max-Age 3600;

    #带cookie请求需要加上这个字段，并设置为true
    add_header Access-Control-Allow-Credentials true;

    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #    表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;

    location / {
      try_files $uri $uri/ /index.html;
    }

    # api
    location /api/ {
      proxy_pass http://localhost:30000;
    }

    # websocket
    location /websocket {
      proxy_pass http://localhost:30000/api/message/websocket;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
      proxy_read_timeout 600s;
    }
# JNPF-END
```

apk打包指南

（1）Android证书（免费，直接本地生成）
https://ask.dcloud.net.cn/article/35777
步骤：

- 建议将JRE安装路径添加到系统环境变量，已配置可跳过此章节
  打开命令行（cmd），输入以下命令：

```
d:
set PATH=%PATH%;"C:\Program Files\Java\jre1.8.0_171\bin"
```

第一行：切换工作目录到D:路径
第二行：将jre命令添加到临时环境变量中

- 生成签名证书
  使用keytool -genkey命令生成证书：

```
keytool -genkey -alias testalias -keyalg RSA -keysize 2048 -validity 36500 -keystore
test.keystore
```

testalias是证书别名，可修改为自己想设置的字符，建议使用英文字母和数字
test.keystore是证书文件名称，可修改为自己想设置的文件名称，也可以指定完整文件路径
36500是证书的有效期，表示100年有效期，单位天，建议时间设置长一点，避免证书过期
回车后会提示：

```
Enter keystore password:  //输入证书文件密码，输入完成回车
Re-enter new password:  //再次输入证书文件密码，输入完成回车
What is your first and last name?
  [Unknown]:  //输入名字和姓氏，输入完成回车
What is the name of your organizational unit?
  [Unknown]:  //输入组织单位名称，输入完成回车
What is the name of your organization?
  [Unknown]:  //输入组织名称，输入完成回车
What is the name of your City or Locality?
  [Unknown]:  //输入城市或区域名称，输入完成回车
What is the name of your State or Province?
  [Unknown]:  //输入省/市/自治区名称，输入完成回车
What is the two-letter country code for this unit?
  [Unknown]:  //输入国家/地区代号（两个字母），中国为CN，输入完成回车
Is CN=XX, OU=XX, O=XX, L=XX, ST=XX, C=XX correct?
  [no]:  //确认上面输入的内容是否正确，输入y，回车

Enter key password for <testalias>
        (RETURN if same as keystore password):  //确认证书密码与证书文件密码一样（HBuilder
|HBuilderX要求这两个密码一致），直接回车就可以
```

以上命令运行完成后就会生成证书，路径为“D:\test.keystore”。

- 查看证书信息
  可以使用以下命令查看：

```
keytool -list -v -keystore test.keystore
Enter keystore password: //输入密码，回车
```

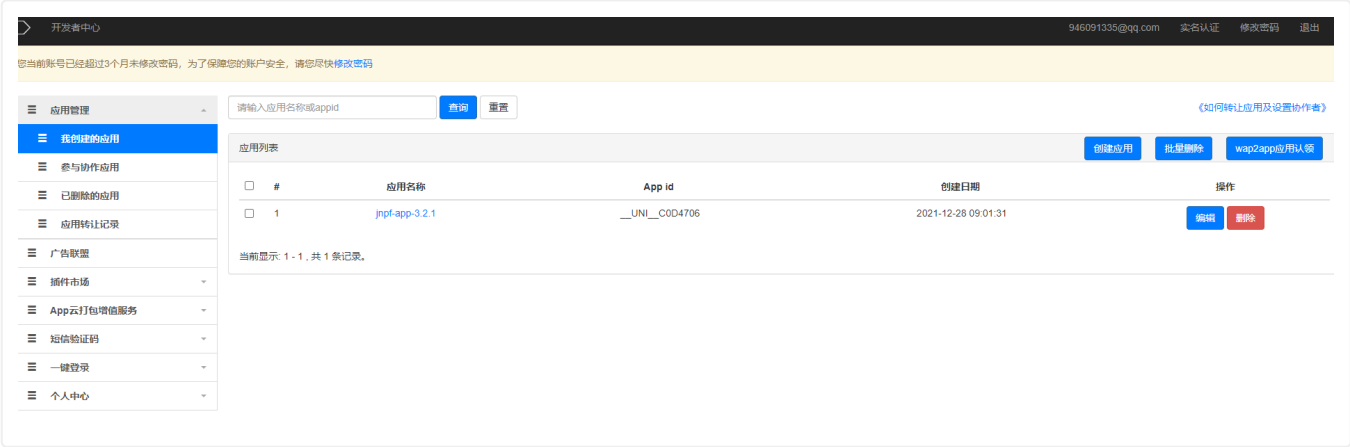
会输出以下格式信息：

```
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: test
Creation date: 2019-10-28
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Tester, OU=Test, O=Test, L=HD, ST=BJ, C=CN
Issuer: CN=Tester, OU=Test, O=Test, L=HD, ST=BJ, C=CN
Serial number: 7dd12840
Valid from: Fri Jul 26 20:52:56 CST 2019 until: Sun Jul 02 20:52:56 CST 2119
Certificate fingerprints:
         MD5:  F9:F6:C8:1F:DB:AB:50:14:7D:6F:2C:4F:CE:E6:0A:A5
         SHA1: BB:AC:E2:2F:97:3B:18:02:E7:D6:69:A3:7A:28:EF:D2:3F:A3:68:E7
         SHA256: 24:11:7D:E7:36:12:BC:FE:AF:2A:6A:24:BD:04:4F:2E:33:E5:2D:41:96:5F:5
0:4D:74:17:7F:4F:E2:55:EB:26
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
```

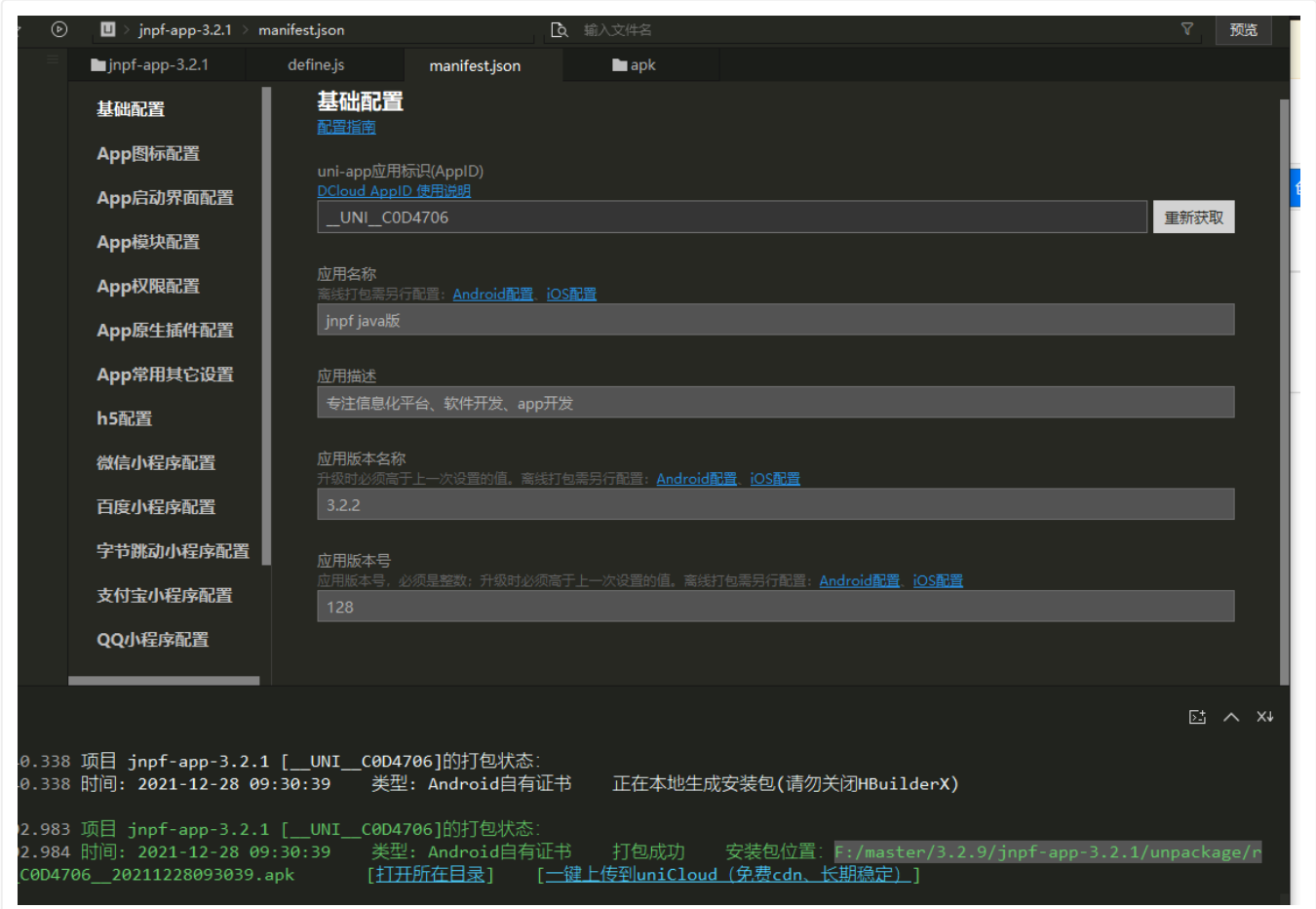
```
输入密钥库口令:
密钥库类型: JKS
密钥库提供方: SUN

您的密钥库包含 1 个条目

别名: testalias
创建日期: 2021-12-28
条目类型: PrivateKeyEntry
证书链长度: 1
证书[1]:
所有者: CN=chen, OU=jnpf, O=jnpf, L=putian, ST=fujian, C=cn
发布者: CN=chen, OU=jnpf, O=jnpf, L=putian, ST=fujian, C=cn
序列号: 20f212fc
有效期为 Tue Dec 28 09:22:31 CST 2021 至 Thu Dec 04 09:22:31 CST 2121
证书指纹:
         MD5:  DC:7B:EE:34:7E:01:84:6D:B3:60:02:0E:CC:08:FC:C1
         SHA1: 3F:01:3A:E3:28:57:A7:60:18:61:16:2D:9E:71:F1:93:4E:C4:D5:79
         SHA256: 3B:C3:91:90:FC:8E:1B:15:2B:6D:C4:D5:7F:D5:7D:3C:C7:BC:04:E0:F5:91:60:29:6B:E8:14:42:98:A2:05:FD
签名算法名称: SHA256withRSA
主体公共密钥算法: 2048 位 RSA 密钥
版本: 3

扩展:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 24 54 7B 89 92 E3 EF 11   92 AD 01 CD 25 8C 29 F9  $T.........%.).
```

- HBuilder打包

  登录官方网站生成App id



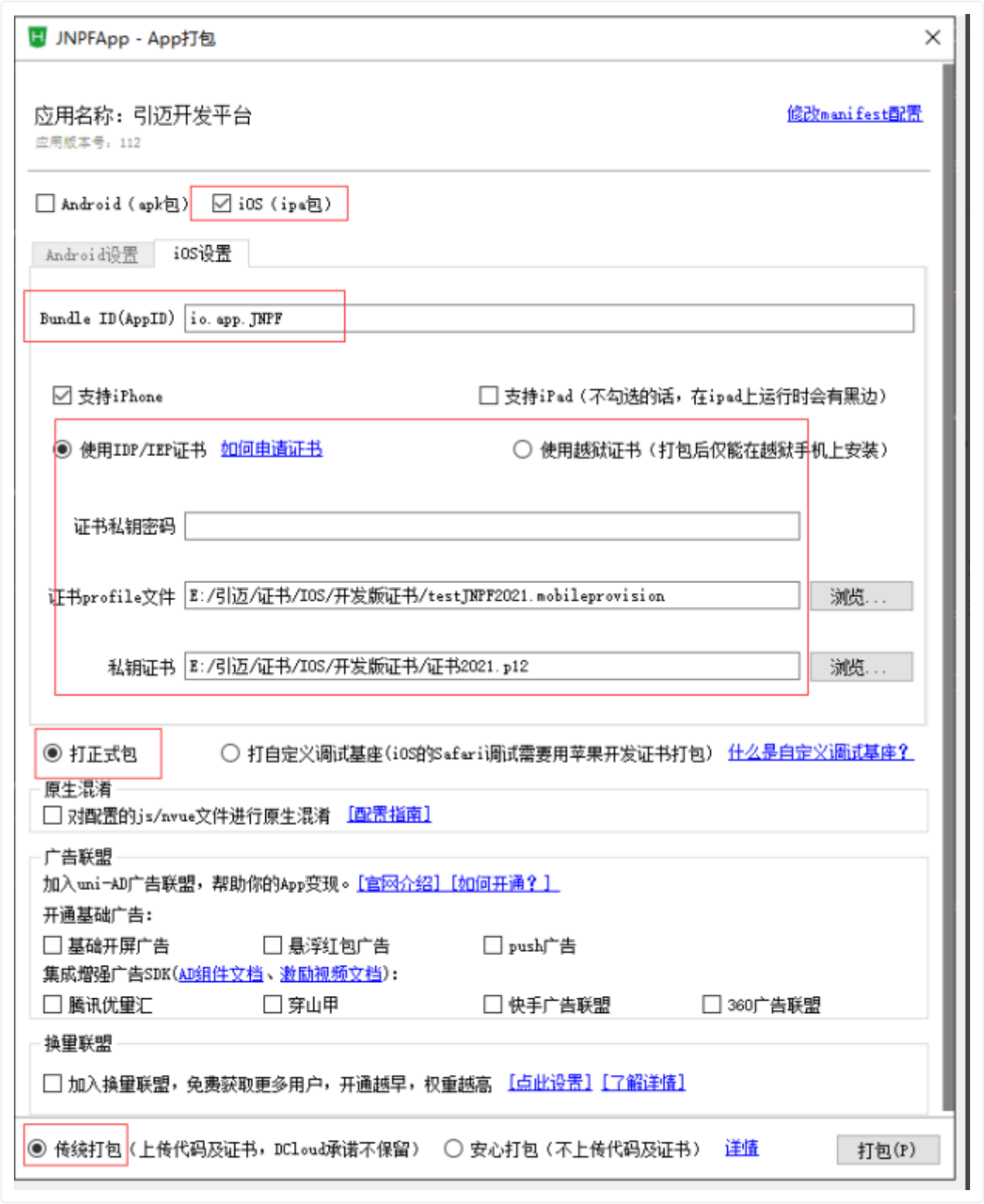
  修改manifest配置



点击发行-原生app（云打包）

填写基础信息打包即可

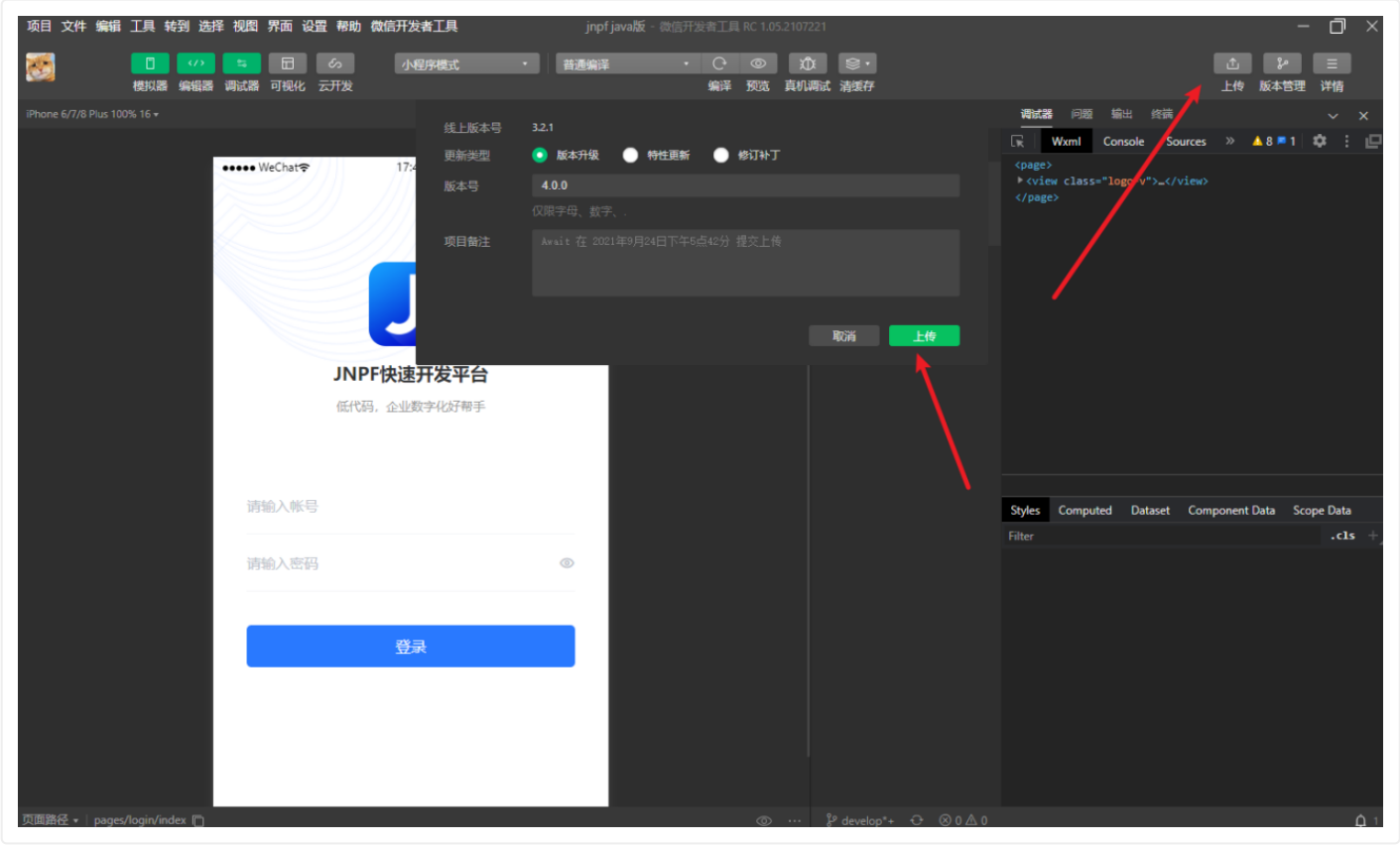
IOS打包主要配置如下：


JNPFApp - App打包
应用名称：引迈开发平台
修改manifest配置
应用版本号：112
Android（apk包）
iOS（ipa包）
Android设置
iOS设置
Bundle ID(AppID)
io.app.JNPF
支持iPhone
支持iPad（不勾选的话，在ipad上运行时会有黑边）
使用IDP/IEP证书
如何申请证书
使用越狱证书（打包后仅能在越狱手机上安装）
证书私钥密码
证书profile文件
E:/引迈/证书/IOS/开发版证书/testJNPF2021.mobileprovision
浏览...
私钥证书
E:/引迈/证书/IOS/开发版证书/证书2021.p12
浏览...
打正式包
打自定义调试基座(iOS的Safari调试需要用苹果开发证书打包)
什么是自定义调试基座？
原生混淆
对配置的js/nvue文件进行原生混淆
[配需指南]
广告联盟
加入uni-AD广告联盟，帮助你的App变现。[官网介绍][如何开通？]
开通基础广告：
基础开屏广告
悬浮红包广告
push广告
集成增强广告SDK(AD组件文档、激励视频文档)：
腾讯优量汇
穿山甲
快手广告联盟
360广告联盟
换量联盟
加入换量联盟，免费获取更多用户，开通越早，权重越高
[点此设置][了解详情]
传统打包（上传代码及证书，DCloud承诺不保留）
安心打包（不上传代码及证书）
详情
打包(P)

H5打包

进入配置打包地址页面，点击发行

小程序发布



java-boot

快速开始

代码结构

> 本文档适用 `jnpf-java-boot` 3.4.3版本

```
jnpf-java-boot
     ├─── docker - Docker构建脚本
     ├─── jnpf-admin -- 启动类及应用配置
     ├     ├─── java - 启动类
     ├     └─── resources - 应用配置
     ├     ├─────     ├─── mapper - mapper文件存放目录
     ├     ├─────     ├─── ... - 其他配置
     ├     ├─────     ├─── application - 应用配置
     ├     ├─────     ├─── application-dev - 开发环境配置
     ├     ├─────     ├─── application-preview - 预发环境配置
     ├     ├─────     ├─── application-pro - 生产环境配置
     ├     ├─────     ├─── application-test - 测试环境配置
```

```
├    ├──      └── logback-spring.xml - 日志配置
├── jnpf-app -- 移动端模块
├    ├── controller - 控制层
├    ├── entity - 实体类
├    ├── mapper - mapper层
├    ├── model - 模型
├    └── service - 业务逻辑层
├── jnpf-common -- 公共模块
├    ├── annotation - 注解包
├    ├── base - 基础类
├    ├── config - 配置
├    ├── constant - 常用字符串
├    ├── database - 数据库配置与多数据库兼容
├    ├── exception - 自定义异常
├    ├── listerner - 监控器
├    ├── model -  基础模型
├    └── util - 工具类
├── jnpf-example -- 模块示例模板
├    ├── controller - 控制层
├    ├── entity - 实体类
├    ├── mapper - mapper层
├    ├── model - 模型
├    ├── service - 业务逻辑层
├    └── util - 工具类
├── jnpf-exception -- 异常处理
├── jnpf-extend -- 扩展模块
├    ├── controller - 控制层
├    ├── entity - 实体类
├    ├── mapper - mapper层
├    ├── model - 模型
├    └── service - 业务逻辑层
├── jnpf-file -- 文件模块
├    ├── controller - 控制层
├    ├── entity - 实体类
├    ├── mapper - mapper层
├    ├── model - 模型
├    ├── service - 业务逻辑层
├    └── utils - 工具类
├── jnpf-oauth -- 登录模块
├── jnpf-system -- 系统模块
├    ├── base - 系统管理模块
├    ├── message - 消息模块
├    ├── permission - 权限模块
├    ├── print - 打印模块
├    └── scheduletask - 系统调度模块
```

```
    ├─── jnpf-visualdata -- 大屏模块
    ├    ├─── controller - 控制层
    ├    ├─── entity - 实体类
    ├    ├─── enums - 枚举类
    ├    ├─── mapper - mapper层
    ├    ├─── model - 模型
    ├    └─── service - 业务逻辑层
    ├─── jnpf-visualdev-- 可视化开发模块
    ├    ├─── base -- 基础模块
    ├    ├─── generater -- 代码生成模块
    ├    ├─── onlinedev -- 在线开发模块
    ├    └─── portal -- 门户设计模块
    └─── jnpf-workflow -- 工作流程模块
    ├    ├─── engine - 流程引擎
    ├    └─── form - 流程表单，用于存放代码生成流程表单的源代码
```

依赖说明

> 本文档适用 `jnpf-java-boot` 3.4.3版本

artifactId	版本号	说明
spring-boot-maven-plugin	2.7.0	SpringBoot打包插件
logback-classic	1.2.11	日志记录
easypoi-base	4.1.2	excel导出导出
hutool-all	5.8.0	工具类整合包
druid	1.2.9	数据库连接池
fastjson	1.2.83	Json工具类
dysmsapi20170525	2.0.8	阿里云发送短信api
tea-openapi	2.0.8	阿里云发送短信api
tencentcloud-sdk-java	3.1.278	腾讯短信支持
commons-compress	1.21	压缩解压工具包
yitter-idgenerator	1.0.6	生成雪花id工具类
lombok	1.18.24	生成构造方法、Get、Set等

jackson-annotations	2.13.3	Json工具类
spring-boot-starter-data-redis	2.7.0	Redis整合springboot
spring-boot-configuration-processor	2.7.0	配置文件提示
spring-boot-starter-websocket	2.7.0	Websocket、springboot整合包
commons-pool2	2.11.1	对象池工具
springfox-boot-starter	3.0.0	Swagger整合包
tomcat-embed-core	9.0.63	Tomcat容器
mybatis-plus-boot-starter	3.5.1	Mybatis、springboot整合包
dynamic-datasource-spring-boot-starter	3.5.1	动态切换数据源
hibernate-validator	6.2.3.Final	字段验证
commons-codec	1.15	编码包
quartz	2.3.2	任务调用
poi	4.1.2	导入导出
commons-lang3	3.12.0	工具包
httpclient	4.5.13	HTTP工具
pinyin4j	2.5.1	语言包
Core(com.google.zxing)	3.5.0	二维码生成工具
nimbus-jose-jwt	9.22	Jwt工具类
mysql-connector-java	8.0.29	MySQL驱动
sqljdbc4	4.0	SqlServer驱动
ojdbc8	21.5.0.0	Oracle驱动
orai18n	21.5.0.0	Oracle驱动
DmJdbcDriver18	1.8.0	达梦驱动
kingbase8-jdbc	2.0	人大金仓驱动
postgresql	42.3.5	Postgre驱动

minio	8.4.0	Minio文件存储
commons-fileupload	1.4	文件上传工具
antisamy	1.6.4	处理XSS注入
pagehelper	5.3.0	Mybatis分页插件
okhttp	4.9.3	http工具类
dom4j	2.1.3	DOM,SAX和JAXP解析工具
spring-boot-starter-web	2.7.0	Springboot、Tomcat整合包
itextpdf	5.5.13.3	处理PDF文件
itext-asian	5.2.0	处理PDF文件
spring-boot-starter-mail	2.7.0	Email、Springboot整合包
aspectjweaver	1.9.7	AspectJ语言
signclient	3.0.1	永中签名生成
spring-boot-starter-security	2.7.0	权限控制
spring-security-oauth2-autoconfigure	2.6.8	权限控制
guava	31.1-jre	工具类
commons-text	1.9	工具类
spring-boot-starter-aop	2.7.0	Aop支持
mybatis-plus-generator	3.4.1	Mybatis代码生成
Thumbnailator	0.4.17	生成图像缩略图
oshi-core	5.4.0	采集系统信息工具
jna	5.11.0	运行期动态访问系统本地库
jna-platform	5.11.0	运行期动态访问系统本地库
commons-io	2.11.0	io流工具
velocity-engine-core	2.3	velocity模板
taobao-sdk-java	1.0	钉钉支持
taobao-sdk-java-source	1.0	钉钉支持

依赖项目

> 本文档适用 `jnpf-java-boot` 3.4.3版本

项目	说明
jnpf-database	数据库脚本
jnpf-scheduletask	系统调度服务端(XXL-JOB引擎)
jnpf-file-core-starter	文件服务核心工具包
jnpf-resourecs	静态资源

jnpf-database

> 本文档适用 `jnpf-java-boot` 3.4.3版本

> 源码项目：jnpf-database

> 本项目为数据脚本文件，包含【平台数据库】初始数据库脚本及文件

目录说明

- `MySQL`：MySQL `平台数据库` 脚本，单体、微服务共用
- `SQLServer`：SQLServer `平台数据库` 脚本，单体、微服务共用
- `PostgreSQL`：PostgreSQL `平台数据库` 脚本，单体、微服务共用
- `Oracle`：Oracle `平台数据库` 脚本，单体、微服务共用
- `DM8`：达梦DM8 `平台数据库` 文件，单体、微服务共用
- `KingbaseES`：人大金仓 `平台数据库` 文件，单体、微服务共用

jnpf-scheduletask

> 本文档适用 `jnpf-java-boot` 3.4.3版本

> 源码项目：jnpf-scheduletask

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

> 本项目为任务调度的基础依赖，可上传到私服或使用本地导入的方式引用该项目

一 结构说明

```
jnpf-scheduletask
     ├─── jnpf-scheduletask-client - xxl-job客户端需要加入此模块
     ├─── jnpf-scheduletask-model - 基础模型
     ├─── xxl-job-core - xxl-job核心包
     ├─── xxl-job-admin - xxl-job服务端及启动类
```

二 使用方式

2.1 基础依赖

> `jnpf-java-boot` 和 `jnpf-java-cloud` 项目xxl-job客户端依赖

2.1.1 私服发布

> 需要Maven私服仓库

2.1.1.1 若相同版本重新发布，需要先登录私服把 `maven-releases` 库中的 `com.jnpf` 目录下的 `jnpf-scheduletask` 删除

2.1.1.2 修改Maven配置

修改Maven文件下conf文件夹中的 `setttings.xml` ，在 `<servers></servers>` 标签中增加 `<server><server>` ，示例:

```
<server>
    <id>maven-releases</id>
    <username>jnpf-user（账号，结合私服配置设置）</username>
```

```
    <password>123456（密码，结合私服配置设置）</password>
</server>
<server>
    <id>maven-snapshots</id>
    <username>jnpf-user（账号，结合私服配置设置）</username>
    <password>123456（密码，结合私服配置设置）</password>
</server>
```

修改 `jnpf-scheduletask` 项目根目录下的 `pom.xml` 文件，增加maven配置

```
  <repository>
    <id>maven-releases(与server-id一致)</id>
    <name>maven-releases</name>
    <url>http://ip:port/repository/maven-releases/(仓库地址)</url>
  </repository>
```

2.1.1.3 发布到私服

双击 `jnpf-scheduletask` 中的 `deploy` 插件，刷新Maven配置即可

2.1.2 本地安装

打开 `jnpf-scheduletask` 项目, 运行install插件， 将 `jnpf-scheduletask` 中的包安装至本地

2.1.3 本地导入

点击 `File` > `Project Structure..` > `module` ，点击右边的+号，点击 `import module` ，找到 `jnpf-scheduletask` 项目后点击ok,点击apply，ok,刷新Maven即可

2.2 服务端

> `jnpf-java-boot` 和 `jnpf-java-cloud` 项目xxl-job服务端

2.2.1 导入数据库脚本

> 暂只支持 `MySQL`

将 `/jnpf-database/MySQL/3.4.x/3.4.3/xxl_job.sql` 导入数据库

2.2.1 数据源配置

> 暂只支持 MySQL

打开 `xxl-job-admin/src/main/resources/application-dev.yml` ,修改数据源配置

```
spring:
  # 数据源
  datasource:
    url: jdbc:mysql://192.168.0.10:3306/xxl_job_dev?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&serverTimezone=GMT%2B8&useSSL=false
    username: xxl_job_dev
    password: nPLGwNBt32GmyWsL
    driver-class-name: com.mysql.cj.jdbc.Driver
```

2.2.3 开发环境

启动 `/xxl-job-admin/src/main/java/com.xxl.job.admin/XxlJobAdminApplication`

2.2.4 部署发布

`package` 之后将 `/xxl-job-admin/target/xxl-job-admin-1.0-RELEASE.jar` 上传至服务器

jnpf-file-core-starter

> 本文档适用 `jnpf-java-boot` 3.4.3版本

> 源码项目：jnpf-file-core-starter

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

> 本项目为文件服务的基础依赖，可上传到私服或使用本地导入的方式引用该项目

使用方式

私服发布

> 需要Maven私服仓库

若相同版本重新发布，需要先登录私服把 `maven-releases` 库中的 `com.jnpf` 目录下的 `jnpf-file-core-starter` 删除

修改Maven配置

修改Maven文件下conf文件夹中的 `setttings.xml` ，在 `<servers></servers>` 标签中增加 `<server><server>` ，示例:

```
<server>
    <id>maven-releases</id>
    <username>jnpf-user（账号，结合私服配置设置）</username>
    <password>123456（密码，结合私服配置设置）</password>
</server>
<server>
    <id>maven-snapshots</id>
    <username>jnpf-user（账号，结合私服配置设置）</username>
    <password>123456（密码，结合私服配置设置）</password>
</server>
```

修改 `jnpf-file-core-starter` 项目根目录下的 `pom.xml` 文件，增加maven配置

```
  <repository>
    <id>maven-releases(与server-id一致)</id>
    <name>maven-releases</name>
    <url>http://ip:port/repository/maven-releases/(仓库地址)</url>
  </repository>
```

发布到私服

双击 `jnpf-file-core-starter` 中的 `deploy` 插件，刷新Maven配置即可

本地安装

打开 `jnpf-file-core-starter` 项目, 运行install插件， 将 `jnpf-file-core-starter` 中的包安装至本地

本地导入

点击 `File` > `Project Structure..` > `module` ，点击右边的+号，点击 `import module` ，找到 `jnpf-file-core-starter` 项目后点击ok,点击apply，ok,刷新Maven即可

jnpf-resourecs

> 本文档适用 `jnpf-java-boot` 3.4.3版本

> 源码项目：jnpf-resourecs

> 本项目为静态资源初始结构及文件

目录结构

```
├─── BiVisualPath                # 大屏设计
├─── CodeTemp                    # 代码生成器临时目录
├─── DocumentFile                # 文档
├─── DocumentPreview             # 文档预览
├─── EmailFile                   # 邮件附件
├─── IMContentFile               # IM聊天附件
├─── SystemFile                  # 系统附件
├─── TemplateCode                # 代码生成器模板
│    ├─── TemplateCode1          # 流程表单模板
│    ├─── TemplateCode2          # 功能表单模板
│    ├─── TemplateCode3          # 功能流程模板
│    └─── TemplateCode4          # 纯表单模板
├─── TemplateFile                # 其他模板文档
├─── TemporaryFile               # 临时存放目录
├─── UserAvatar                  # 用户头像
└─── WebAnnexFile                # 其他
```

配置说明

打开 `jnpf-admin/src/main/resources/application-*.yml` ,找到 `文件存储配置`

> 其中 `*` 对应 `application.yml` 中的 `spring.profiles.active` 的值，默认为 `dev`

> 特别注意：配置中的格式(yaml格式)及目录最后的结束符号

本地存储

- 修改默认存储平台(`default-platform`)为 `local-plus-1`
- 修改基础路径 `base-path` 的值
  - 本地开发为项目所在路径，如： `F:/work/jnpf-resources/`
  - 部署环境，如 `/www/wwwroot/jnpf-resources/`

```
  file-storage: #文件存储配置，不使用的情况下可以不写
    default-platform: local-plus-1 #默认使用的存储平台
    local-plus: # 本地存储升级版
      - platform: local-plus-1 # 存储平台标识
        enable-storage: true  #启用存储
        enable-access: true #启用访问（线上请使用 Nginx 配置，效率更高）
        domain: "" # 访问域名，例如：“http://127.0.0.1:8030/”，注意后面要和 path-patterns 保持一致，“/”结尾，本地存储建议使用相对路径，方便后期更换域名
        base-path: F:/work/jnpf-resources/ # 基础路径
        path-patterns: /** # 访问路径
        storage-path:  # 存储路径
```

对象存储(含minio)

> 以MinIO为例

- 修改默认存储平台(`default-platform`)为 `minio-1`
- 配置 `MiniIO` ，涉及如下参数
  - `access-key` ：access-key
  - `secret-key` ：secret-key
  - `end-point` ：minio服务地址
  - `bucket-name` ：桶名

```
  file-storage: #文件存储配置，不使用的情况下可以不写
    default-platform: local-plus-1 #默认使用的存储平台
    thumbnail-suffix: ".min.jpg" #缩略图后缀，例如【.min.jpg】【.png】
    local-plus: # 本地存储升级版
      - platform: local-plus-1 # 存储平台标识
        enable-storage: true  #启用存储
        enable-access: true #启用访问（线上请使用 Nginx 配置，效率更高）
        domain: "" # 访问域名，例如：“http://127.0.0.1:8030/”，注意后面要和 path-patterns 保持一致，“/”结尾，本地存储建议使用相对路径，方便后期更换域名
        base-path: F:/work/jnpf-resources/ # 基础路径
        path-patterns: /** # 访问路径
        storage-path:  # 存储路径
    minio: # MinIO，由于 MinIO SDK 支持 AWS S3，其它兼容 AWS S3 协议的存储平台也都可配置在这里
      - platform: minio-1 # 存储平台标识
        enable-storage: true  # 启用存储
        access-key: Q9jJs2b6Tv
        secret-key: Thj2WkpLu9DhmJyJ
        end-point: http://192.168.0.207:9000/
        bucket-name: jnpfsoftoss
```

```
        domain:  # 访问域名，注意“/”结尾，例如：http://minio.abc.com/abc/
        base-path:  # 基础路径
```

本地运行

> 本文档适用 `jnpf-java-boot` 3.4.3版本

> 特别说明：源码、JDK、MySQL、Redis等存放路径禁止包含中文、空格、特殊字符等

一 技术栈

- 主框架：`Spring Boot` + `Spring Framework`
- 持久层框架：`MyBatis-Plus`
- JSON序列化: `Jackson` & `Fastjson`
- 缓存：`Redis`
- 数据库：`MySQL` (默认)、`SQLServer`、`Oracle`、`PostgreSQL`、`达梦数据库`、`人大金仓数据库`
- 项目构建：`Maven`
- 安全框架：`spring-cloud-security-oauth2` + `jwt`
- 模板引擎：`Velocity`
- 任务调度：`XXL-JOB`
- 即时通讯：`spring-boot-starter-websocket`
- AOP：`spring-boot-starter-aop`

二 环境要求

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，可使用 `OpenJDK 8/11`、`Alibaba Dragonwell 8/11`、`BiShengJDK 8/11`

> `Alibaba Dragonwell 8/11` 下载地址：https://dragonwell-jdk.io

> `BiShengJDK 8/11` 下载地址：https://www.hikunpeng.com/developer/devkit/compiler/jdk

> 在使用JDK11时需要把项目根目录下的 `pom.xml` 中的 改成`11`

项目	推荐版本	说明
IDEA	IDEA2020及以上版本	

JDK	1.8.0_281及以上版本	JAVA环境依赖(需配置环境变量)
Maven	3.6.3及以上版本	项目构建(需配置环境变量)
Redis	3.2.100(Windows)/4.0.x+(Linux,Mac)	缓存
MySQL	5.7.x+	数据库任选一(默认)
SQLServer	2012+	数据库任选一
Oracle	11g+	数据库任选一
PostgreSQL	12+	数据库任选一
达梦数据库	DM8	数据库任选一
人大金库	KingbaseES V8 R6	数据库任选一

三 IDEA插件

- `Lombok` (必须)
- `Alibaba Java Coding Guidelines`
- `MybatisX`

四 Maven私服配置

> 因部分依赖在阿里云Maven私服、Maven官方无法下载

依赖包差异

- com.sqlserver:sqljdbc4:4.0
- com.oracle:ojdbc6:11.2.0
- com.dm:DmJdbcDriver18:1.8.0
- com.kingbase8:kingbase8-jdbc:2.0
- dingtalk-sdk-java:taobao-sdk-java-source:1.0
- dingtalk-sdk-java:taobao-sdk-java:1.0
- yozo:signclient:3.0.1

打开 `maven` 下 `conf/settings.xml` x文件，

```
# 在<servers></servers>中添加如下内容
<server>
  <id>maven-releases</id>
  <username>jnpf-user</username>
```

```
  <password>HLrQ0MA%S1nE</password>
</server>
<server>
  <id>maven-snapshots</id>
  <username>jnpf-user</username>
  <password>HLrQ0MA%S1nE</password>
</server>

# 在<mirrors></mirrors>中添加
<mirror>
  <id>maven-snapshots</id>
  <mirrorOf>*</mirrorOf>
  <name>maven-snapshots</name>
  <url>https://repository.jnpfsoft.com/repository/maven-public/</url>
</mirror>
```

五 开发环境

5.1 环境准备

> 开发环境以Windows 10为例

5.1.1 基础环境

项目	说明
IDEA	使用IDEA 2020及以上版本，也可以用eclipse等其他JAVA开发工具
JDK	1.8.0_281及以上版本
Maven	3.6.3及以上版本
Redis	3.2.100
Navicat(可选)	数据库管理工具
RDM(可选)	Redis管理工具

5.1.2 关联项目

项目	说明
jnpf-database	数据库脚本文件

jnpf-scheduletask	任务调度服务端(XXL-JOB引擎)
jnpf-file-core-starter	文件服务核心工具包
jnpf-resourecs	静态资源

5.2 初始化数据库

> 以 MySQL 数据库为例

5.2.1 创建数据库

创建平台数据库

5.2.2 导入数据库脚本

将 `/jnpf-database/MySQL/<版本>/jnpf_init.sql` 导入至上述创建的数据库中

5.3 准备依赖项目

5.3.1 导入系统调度服务端

详见 `jnpf-scheduletask` 项目中的 `README.md` 文档说明

5.3.2 导入文件服务核心工具包

详见 `jnpf-file-core-starter` 项目中的 `README.md` 文档说明

5.4 环境配置

5.4.1 主配置文件

打开编辑 `jnpf-admin/src/main/resources/application.yml`

指定环境配置

```
# application.yml第5行,可选值: dev(默认)|test|pro|preview
active: dev
```

其他配置参考 `application.yml` 文件中的说明

5.4.2 环境配置文件

项目中环境配置文件说明

- `application-dev.yml` 开发环境(默认)
- `application-test.yml` 测试环境
- `application-preview.yml` 预发布环境
- `application-pro.yml` 生产环境

> 以开发环境以例

打开编辑 `application-dev.yml` ,需配置以下相关信息

- 应用端口(`port`)
- 数据源配置
- Redis配置
- 系统调度服务端配置
- 静态资源配置
- 第三方登录配置(可选)
- 任务调度配置

5.5 启动项目

找到 `jnpf-admin/src/main/java/JnpfAdminApplication.java` ，右击运行即可。

六 项目发布

- 在 `IDEA` 右侧 `Maven` - `jnpf-java-boot(root)` - `Lifecycle` 中双击 `clean` 清理下项目
- 双击 `package` 打包项目
- 打开项目目录，依次打开 `jnpf-java-boot\jnpf-admin\target` ，将 `jnpf-admin-{version}-RELEASE.jar` 上传至服务器

七 swagger接口文档

- `http://localhost:30000/swagger-ui/`

部署环境

nginx配置参考文档

`Nginx` 配置参考文档:

```
server
{
    listen 80;
    server_name 192.168.20.199;
    index index.php index.html index.htm default.php default.htm default.html;
    root /www/wwwroot/jnpf-web/;

    #SSL-START SSL相关配置，请勿删除或修改下一行带注释的404规则
    #error_page 404/404.html;
    #SSL-END

    #ERROR-PAGE-START  错误页配置，可以注释、删除或修改
    #error_page 404 /404.html;
    #error_page 502 /502.html;
    #ERROR-PAGE-END

    #PHP-INFO-START  PHP引用配置，可以注释或修改
    include enable-php-00.conf;
    #PHP-INFO-END

    #REWRITE-START URL重写规则引用,修改后将导致面板设置的伪静态规则失效
    include /www/server/panel/vhost/rewrite/139.217.118.160.conf;
    #REWRITE-END

    #禁止访问的文件或目录
    location ~ ^/(\.user.ini|\.htaccess|\.git|\.svn|\.project|LICENSE|README.md)
    {
        return 404;
    }

    #一键申请SSL证书验证目录相关设置
    location ~ \.well-known{
        allow all;
    }

    # location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
    # {
    #     expires      30d;
    #     error_log /dev/null;
    #     access_log /dev/null;
    # }

      #JNPF-Start

  # 前端伪静态配置
```

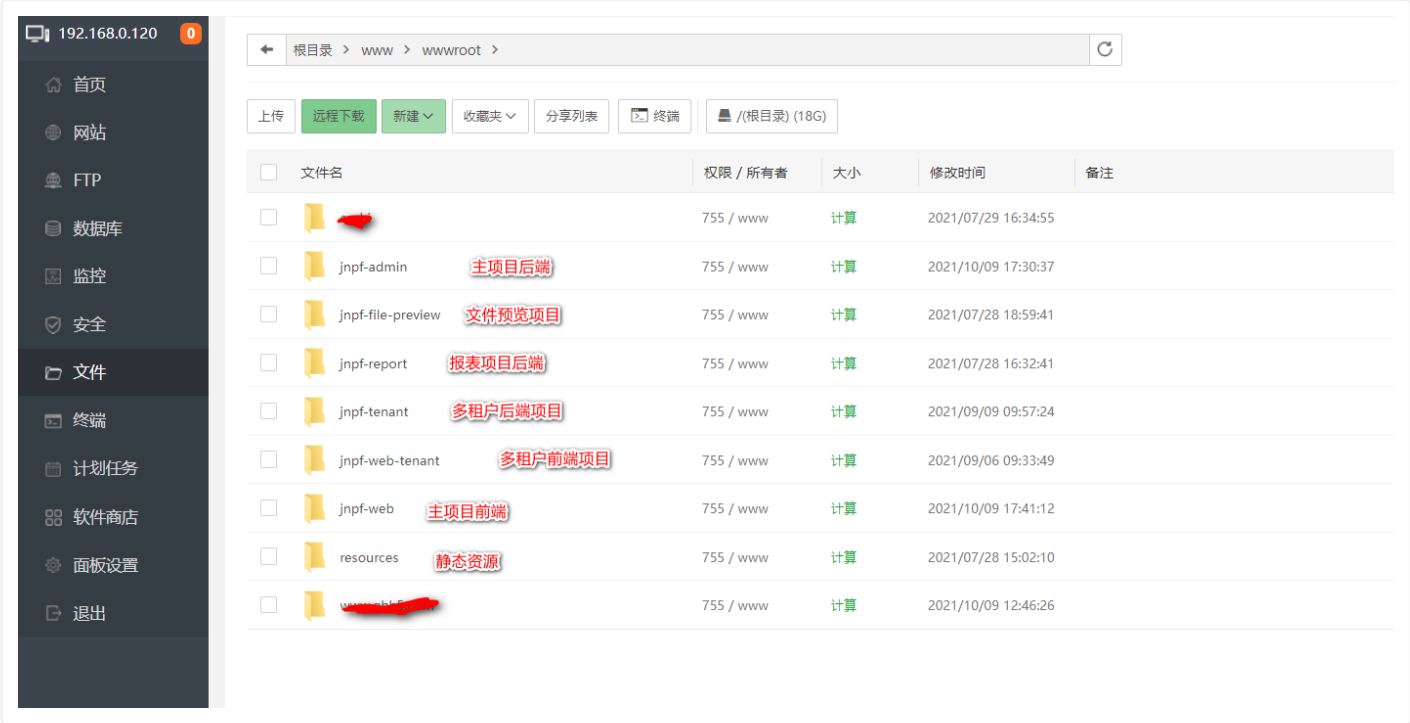
```
# 主项目前端
location / {
  try_files $uri $uri/ /index.html;
}

# 大屏前端
location /DataV {
  try_files $uri $uri/ /DataV/index.html;
}

#设置上传文件的大小
client_max_body_size 100m;

#添加头部信息
proxy_set_header Cookie $http_cookie;
proxy_set_header X-Forwarded-Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

#请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
client_header_buffer_size 128k;

#指令参数4为个数，128k为大小，默认是8k。申请4个128k。
large_client_header_buffers 4 128k;

#指定允许跨域的方法，*代表所有
add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';

# 预检命令的缓存，如果不缓存每次会发送两次请求
add_header Access-Control-Max-Age 3600;

#带cookie请求需要加上这个字段，并设置为true
add_header Access-Control-Allow-Credentials true;

#  表示允许这个域跨域调用（客户端发送请求的域名和端口）
#  $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
add_header Access-Control-Allow-Origin $http_origin;
#  表示请求头的字段 动态获取
add_header Access-Control-Allow-Headers $http_access_control_request_headers;


# 后端接口(按实际情况修改端口)
location /api/ {
  proxy_pass http://localhost:30000/api/;
```

```
    proxy_read_timeout 180s;
  }

  # webscket接口(按实际情况修改端口)
  location /websocket {
    proxy_pass http://localhost:30000/api/message/websocket;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_read_timeout 600s;
  }

  # 报表设计接口配置(按实际情况修改端口)
  location /ReportServer/ {
    proxy_pass http://localhost:30007/;
  }


  # 文件预览服务
  location /FileServer {
    proxy_pass http://localhost:30090;
  }

  # 解决文件预览服务无法加载js,css问题
  location ~ /FileServer/*.*\.(js|css)?$ {
                proxy_pass http://localhost:30090;
  }

#JNPF-End

    location ~ .*\.(js|css)?$
    {
        expires      12h;
        error_log /dev/null;
        access_log /dev/null;
    }
    access_log  /www/wwwlogs/192.168.20.199.log;
    error_log  /www/wwwlogs/192.168.20.199.error.log;




}
```

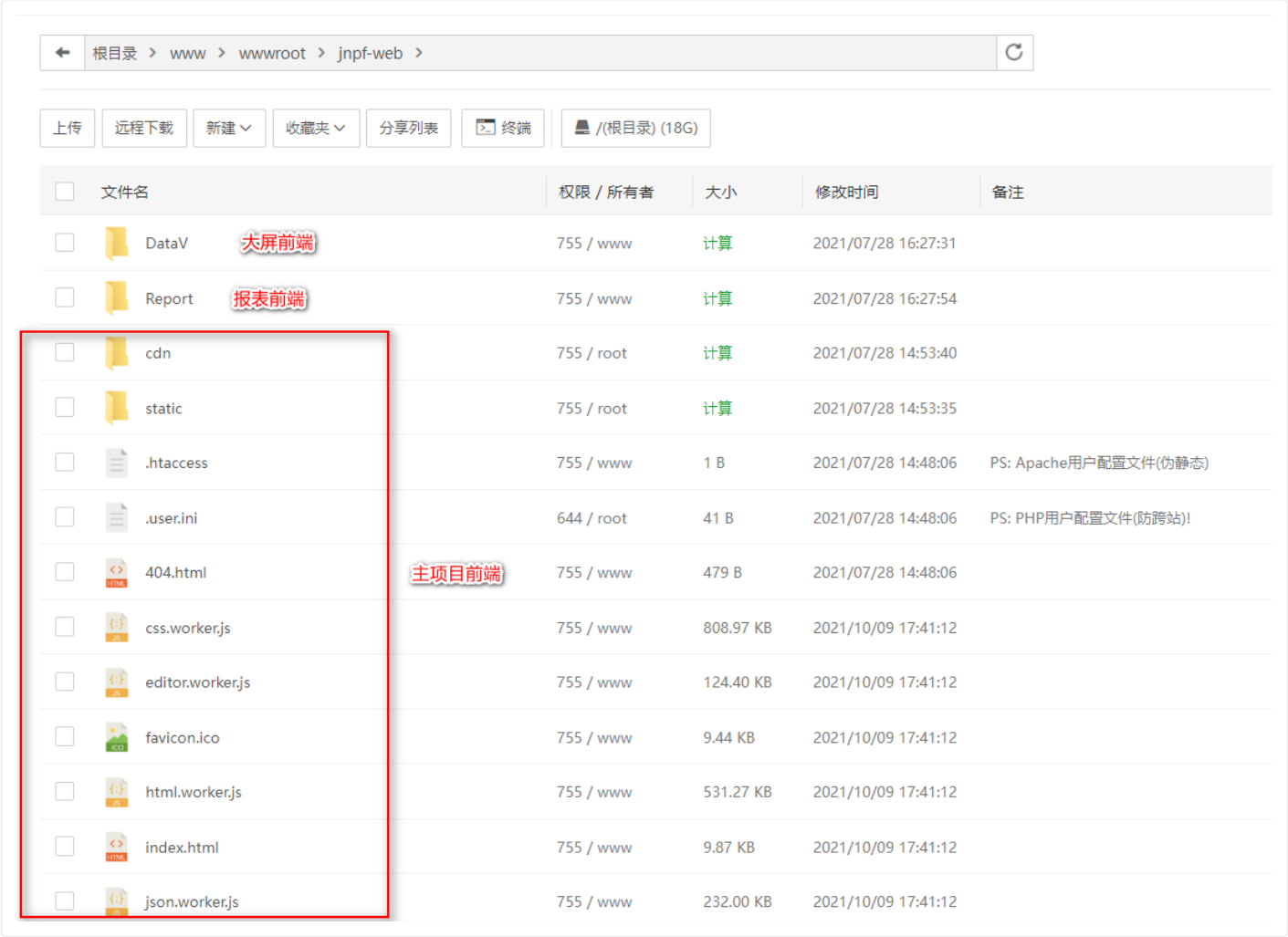
注：ip地址和接口根据实际数据修改

部署服务器目录结构

1.部署文件主要目录



2.前端项目目录结构



主项目前端

配置

在项目根目录打开 `.env.staging` (测试环境配置)，部署生产环境请打开 `.env.product` 文件

```
# 测试默认配置
ENV = 'staging'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.10'
# websocket接口
VUE_APP_BASE_WSS = 'ws://192.168.0.10/websocket'
```

```
# 生成环境默认配置
ENV = 'production'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.10'
```

```
# websocket接口
VUE_APP_BASE_WSS = 'ws://192.168.0.10/websocket'
```

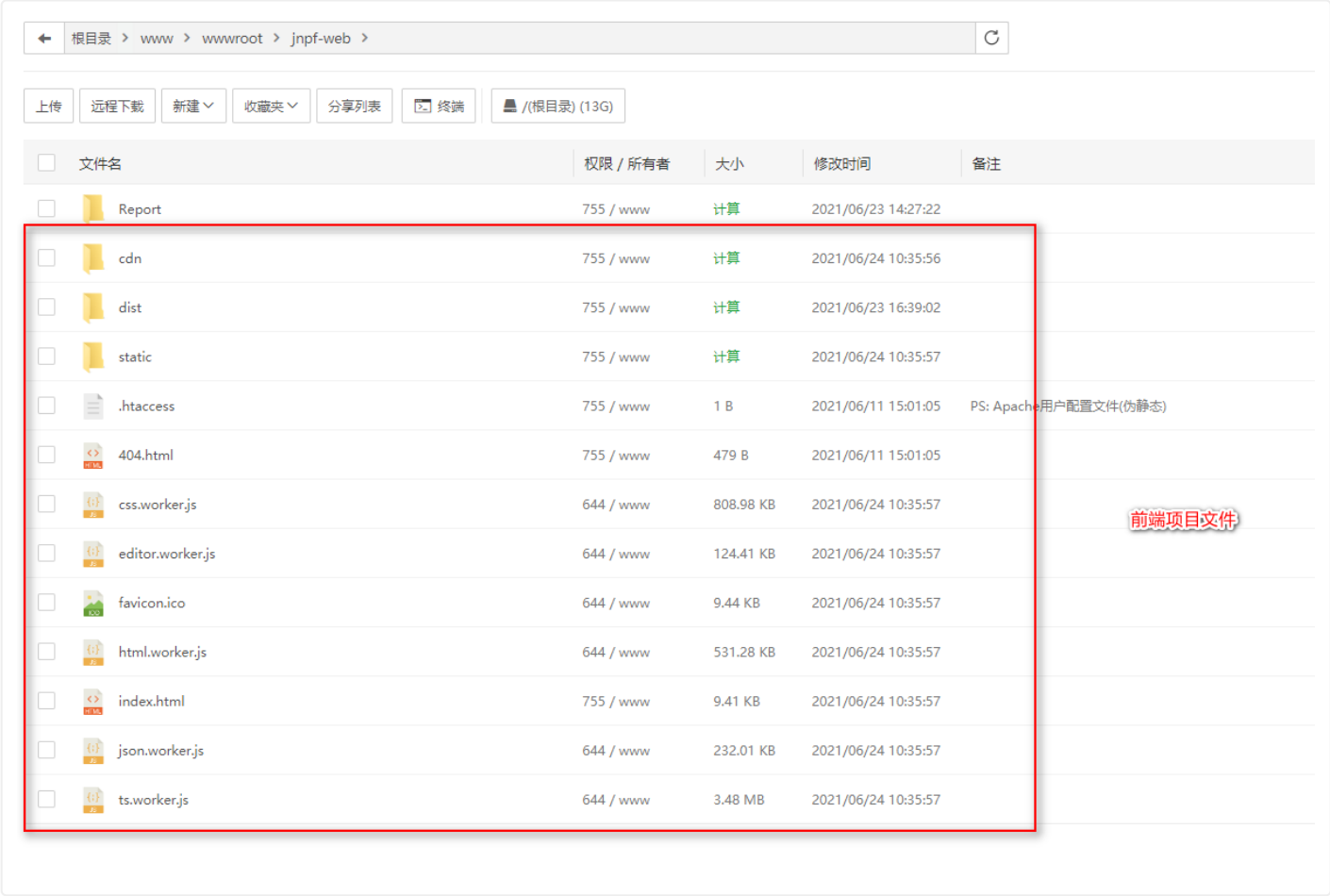
构建

```
# 构建测试环境
npm run build:staging

# 构建生产环境
npm run build
```

发布到服务器

- 方式1：压缩dist目录，上传到服务器/www/wwwroot/jnpf-web并解压
- 方式2：调整构建指令如下

```
npm run build:staging
scp -P 1802 -r  ./dist/* ssh root@192.168.0.10:/www/wwwroot/jnpf-web
```



常见问题

- 访问网站或者域名访问时，头像等无法显示问题
  注释nginx默认配置

```
# location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
    # {
    #     expires      30d;
    #     error_log /dev/null;
    #     access_log /dev/null;
    # }
```

主项目后端部署

配置

参考 `Java-boot - 快速开始 - 主项目运行`

数据源配置：

打开 `application-dev` 修改：（根据实际配置文件命名修改）

```
  datasource:
    db-type: MySQL #数据库类型(可选值 MySQL、SQLServer、Oracle、DM8、KingbaseES、PostgreSQL，请严格按可选值填写)
    host: 192.168.0.10
    port: 3306
    username: java_boot_test
    password: pBx5HaW6WMGSTdDf
    db-name: java_boot_test
    db-schema: #金仓达梦选填
    prepare-url: #自定义url
```

Redis配置：

```
  redis:
    database: 1 #缓存库编号
    host: 127.0.0.1
    port: 6379
    password:
    timeout: 3000 #超时时间(单位：秒)
    lettuce: #Lettuce为Redis的Java驱动包
      pool:
```

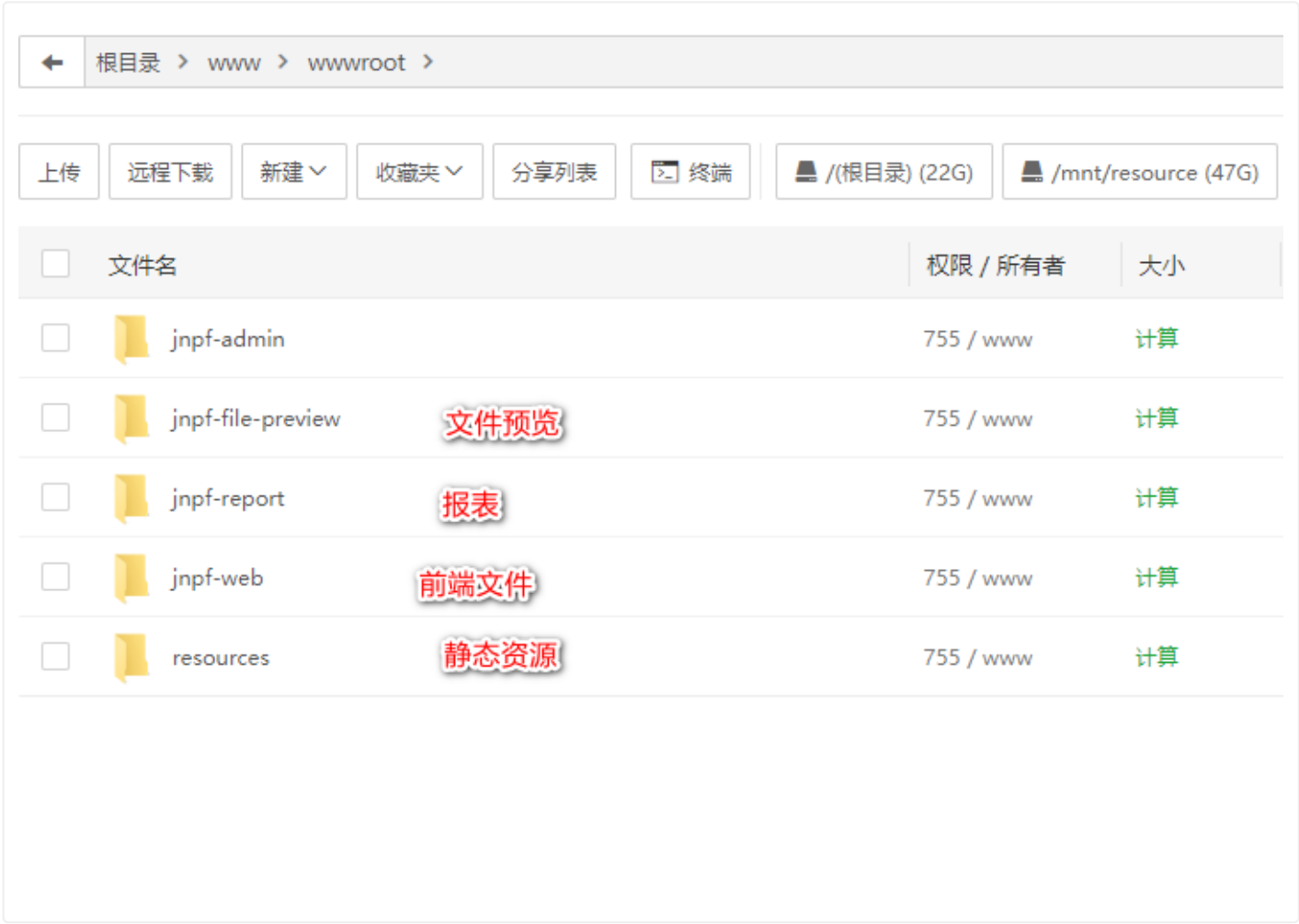
```
        max-active: 8 # 连接池最大连接数
        max-wait: -1ms  # 连接池最大阻塞等待时间（使用负值表示没有限制）
        min-idle: 0 # 连接池中的最小空闲连接
        max-idle: 8 # 连接池中的最大空闲连接
```

静态资源配置

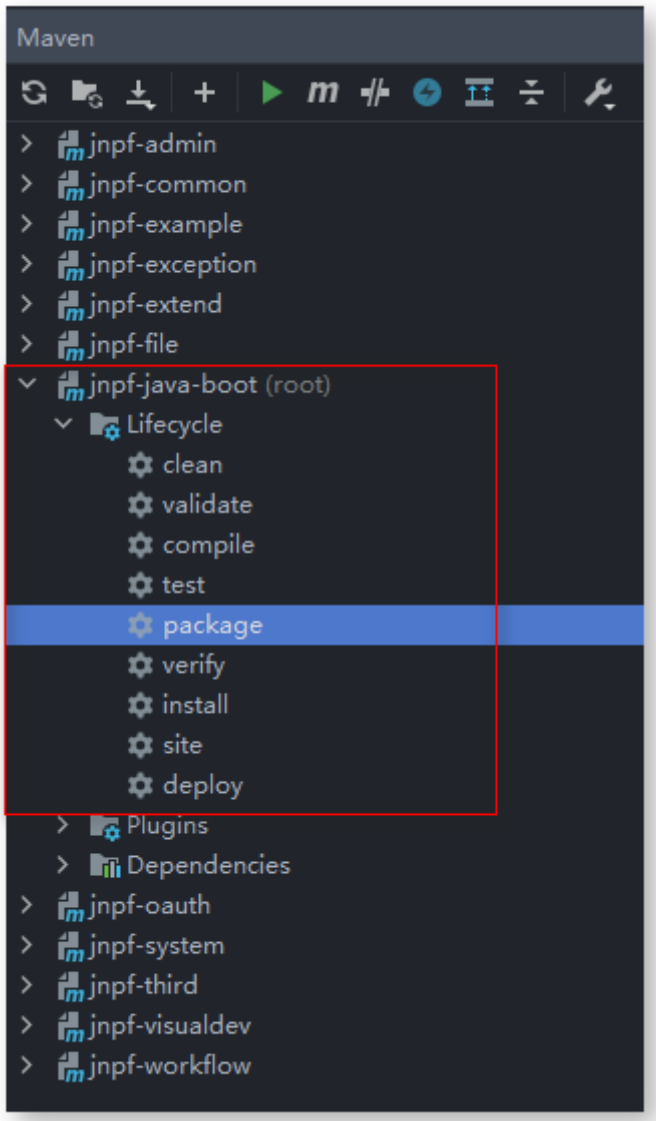
打开 `application-dev` 修改：

```
#====================== 文件模板配置 ======================
  fileType: local #文件存储类型(local-本地存储，minio-网络存储)
  Path: F:\project\project\resource\jnpf-resources-v3.1\ #Windows配置（静态资源根目录和代码生成器临时目录）
  #Path: /www/wwwroot/resources/ #Linux配置（静态资源根目录和代码生成器临时目录）
```

上传静态文件至服务器：



打包



上传到服务器

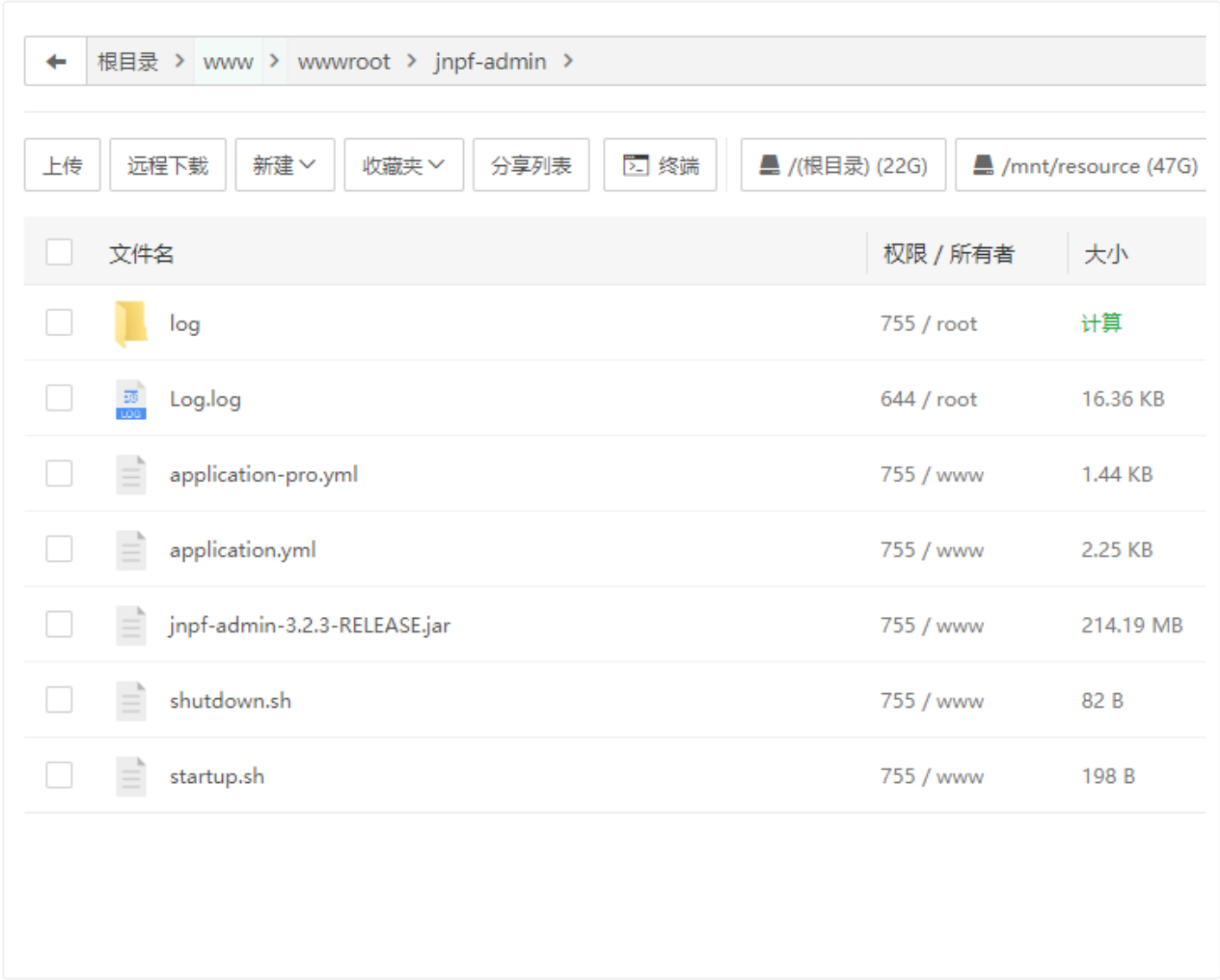
- `jnpf-java-boot\jnpf-admin\target` 的 `.jar` 文件上传到服务器 `/www/wwwroot/jnpf-admin` 目录下
- 运行该文件即可。
- 运行命令为：

```
nohup java -jar -Xmx2000m -Xms2000m -Xmn2g -Xss256k jnpf-admin-3.2.4-RELEASE.jar >Log.log 2>&1 &
```

- 停止运行命令：

```
kill -9 $(netstat -nlp | grep 30000 | awk '{print $7}' | awk -F"/" '{ print $1 }')
```

注：使用.sh脚本运行



常见问题

- Linux下运行 `.sh` 启动提示 `$'\r': 未找到命令的解决` 等问题

在服务器上安装

```
yum install dos2unix
```

然后通过执行 `dos2unix **.sh` 进行转换

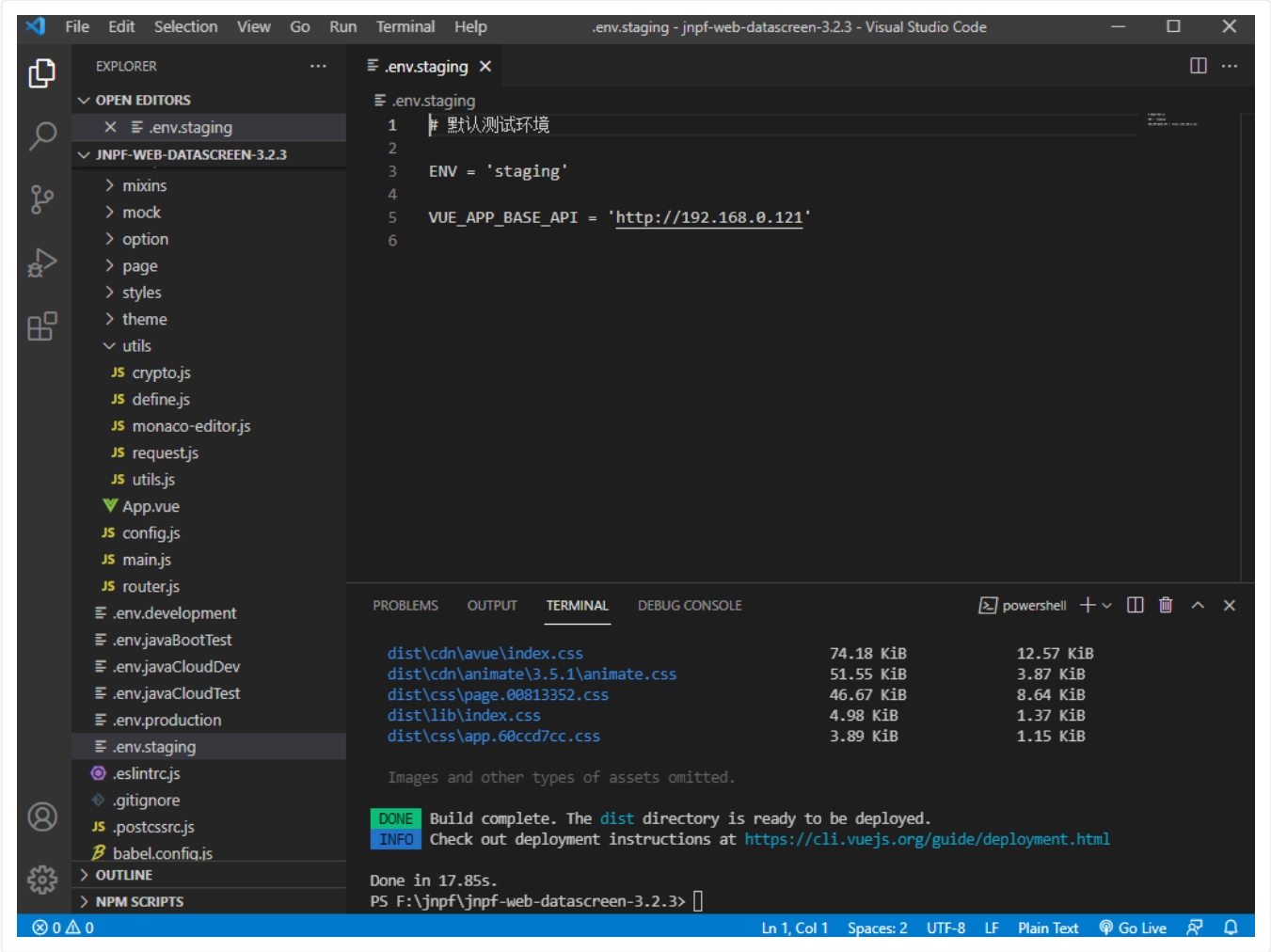
- 访问网站或者域名访问时，头像等无法显示问题
  注释nginx默认配置

```
# location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
   # {
   #     expires      30d;
   #     error_log /dev/null;
   #     access_log /dev/null;
   # }
```

大屏项目部署

项目打包

1. 打开大屏前端项目 `jnpf-web-datascreen`，修改 `.env.staging` 文件



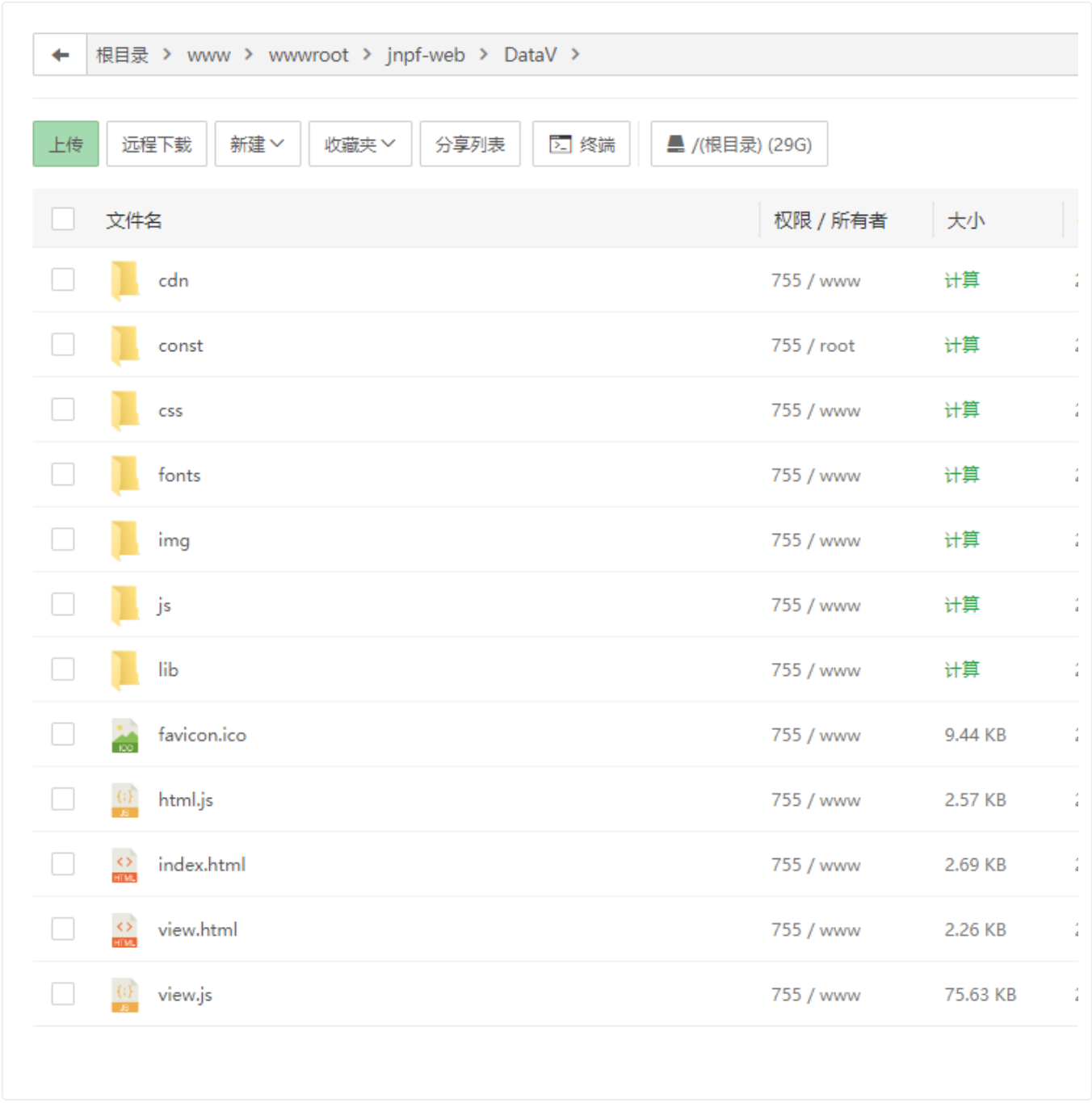
2. 安装依赖

```
yarn
```

3. 打包项目

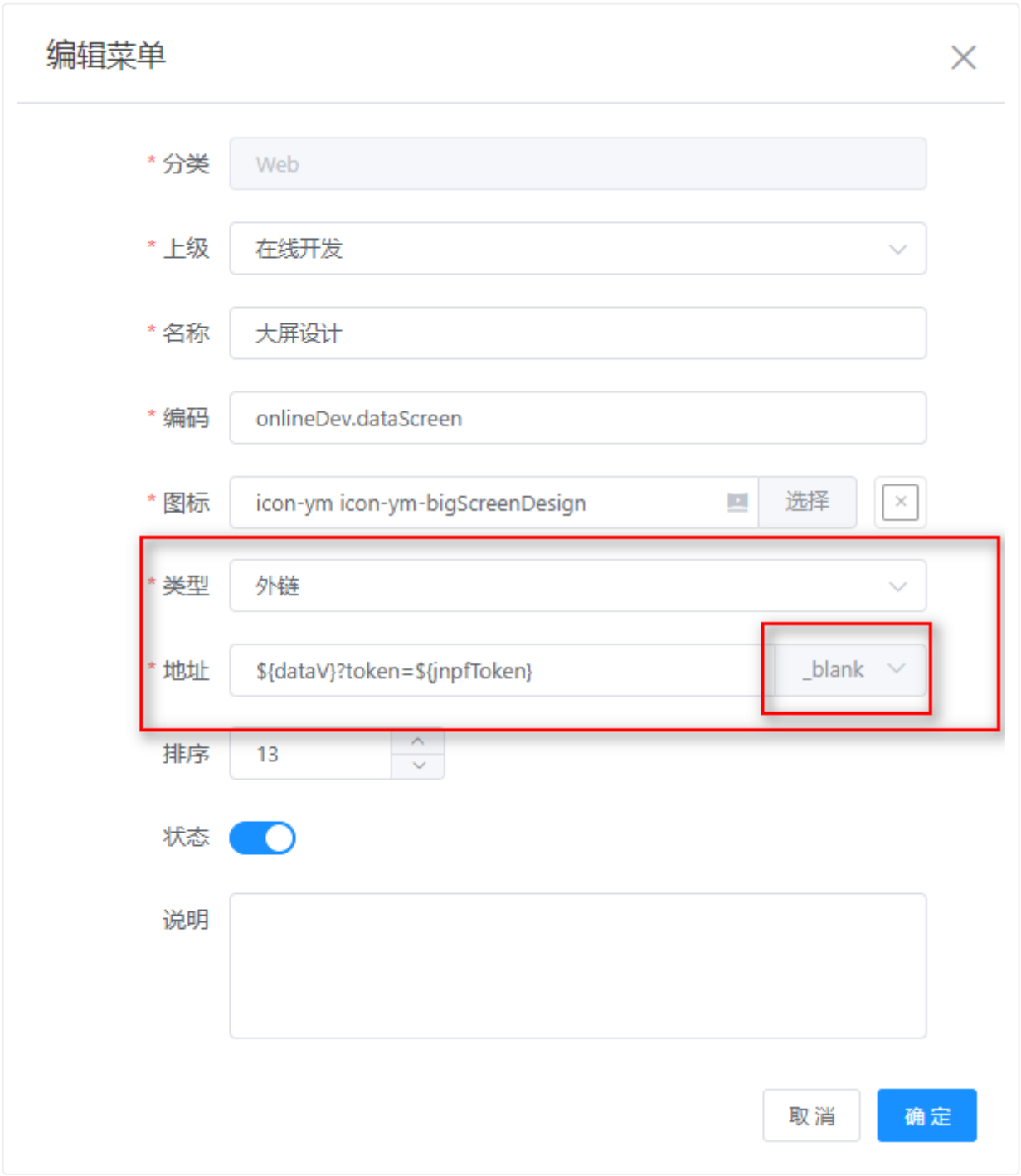
```
yarn build:staging
```

4. 上传dist文件夹全部内容到服务器DataV目录下



5. 登录后修改菜单配置

```
${dataV}?token=${jnpfToken}
```



6.配置nginx

```
# 大屏前端
 location /DataV {
   try_files $uri $uri/ /DataV/index.html;
 }
```

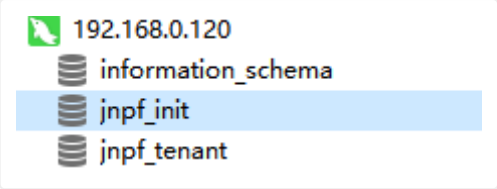
多租户部署

一、脚本准备

- `jnpf_init.sql`(初始库脚本)：用于生成租户库后同步数据，文件位于`jnpf-databae/MySQL/`；
- `JNPF-MySQL.sql`(多租户创库脚本)：用于生成租户库表结构，文件位于`jnpf-databae/MySQL/多租户/3.2.x/`；
- `jnpf_tenant.sql`(多租户数据库)：用于记录租户信息，文件位于`jnpf-databae/MySQL/多租户/`；

二、导入数据库脚本

1、新建`jnpf_init`数据库，并将`jnpf_init.sql`导入（以`新建查询`导入）；
2、新建`jnpf_tenant`数据库，并将`jnpf_tenant.sql`导入



三、修改`application.yml`配置

- 修改`MultiTenancy`配置，开启多租户，

```
MultiTenancy: true
MultiTenancyUrl: http://127.0.0.1:30006/api/tenant/DbName/
```

```
#多租户是否开启
MultiTenancy: true
MultiTenancyUrl: http://127.0.0.1:30006/api/tenant/DbName/
IgexinEnabled: true
IgexinAppid: HLFY9T2d1z7MySY8hwGwh4
IgexinAppkey: 6Uiduugq648YDChhCjAt59
IgexinMastersecret: pEyQm156SJ9iS7PbyjLCZ6
SoftVersion: V3.2.4
```

- 修改`tenant.yaml`配置

```
spring:
# 多租户创库脚本目录（JNPF-MySQL.sql所在路径）
file: /www/wwwroot/jnpf-tenant/
redis:
  database: 1
  host: 127.0.0.1
  port: 6379
  password:
# 数据源
dataType: mysql
```

```
datasource:
  # 初始库
  dbinit: jnpf_init
  # 租户库
  dbname: jnpf_tenant
  url: jdbc:mysql://192.168.0.10:3306/{dbName}?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8&nullCatalogMeansCurrent=true
  username: root
  password: 123456
  driver-class-name: com.mysql.cj.jdbc.Driver
```

四、多租户管理前端页面配置

> 打开 jnpf-web-tenant 项目并安装项目依赖

- 开发环境

打开 `src/utils/define.js` ,修改 开发环境接口配置

```
// 开发环境接口配置
const APIURl = 'http://192.168.0.26:30000'
```

- 测试环境

编辑根目录下 `.env.staging` 文件

```
# 测试默认配置
ENV = 'staging'
VUE_APP_BASE_API = 'http://192.168.0.25'
```

修改后在命令行运行 `yarn build:staging` 打包发布项目代码

- 生产环境

编辑根目录下 `.env.production` 文件

```
#  生产默认配置
ENV = 'production'
VUE_APP_BASE_API = 'https://tenant.java.jnpfsoft.com'
```

修改后在命令行运行 `yarn build` 打包发布项目代码

五、多租户管理前端页面部署

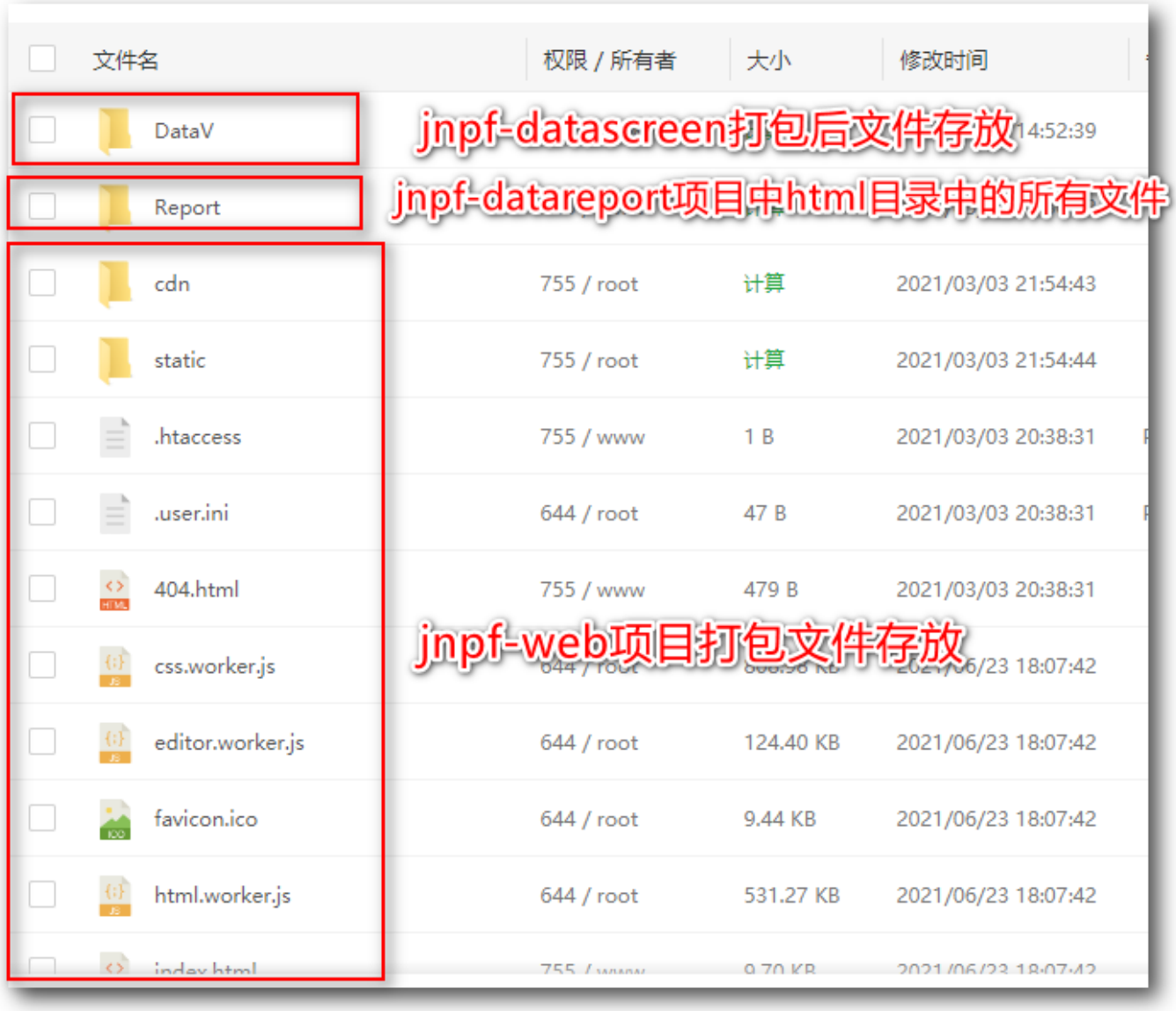
多租户前端需要单独部署，nginx核心配置如下

```
#JNPF-Start
#设置上传文件的大小
client_max_body_size 100m;

#添加头部信息
proxy_set_header Cookie $http_cookie;
proxy_set_header X-Forwarded-Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

#请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
client_header_buffer_size 128k;

#指令参数4为个数，128k为大小，默认是8k。申请4个128k。
large_client_header_buffers 4 128k;

#指定允许跨域的方法，*代表所有
add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';

# 预检命令的缓存，如果不缓存每次会发送两次请求
add_header Access-Control-Max-Age 3600;

#带cookie请求需要加上这个字段，并设置为true
add_header Access-Control-Allow-Credentials true;

#    表示允许这个域跨域调用（客户端发送请求的域名和端口）
#    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
add_header Access-Control-Allow-Origin $http_origin;
#    表示请求头的字段 动态获取
add_header Access-Control-Allow-Headers $http_access_control_request_headers;

# 前端伪静态配置
location / {
  try_files $uri $uri/ /index.html;
}

# 多租户接口
 location /api/ {
  proxy_pass http://192.168.0.18:30006/api/;
}

#JNPF-End
```

报表部署

测试生产环境

前端部署结构说明




```
├── jnpf-web                    # 假设这个目录是存放测试或生产环境的前端
│   ├── DataV                   # 大屏(`jnpf-datascreen`)打包后文件存放目录
│   ├── Report                  # 报表(`jnpf-datareport`)html下的文件
│   └── 主项目前端打包后的文件    # 主项目(`jnpf-web`)打包后存放在根目录
```

前端

- 将 `jnpf-web-datareport` 下的 `html` 文件夹中的所有拷贝到 `jnpf-web` 中的 `Report` 目录下，如 `Report` 目录不存在请手动建立
- 接口配置
  - 打开 `/Report/index.html` ,做如下修改

```
# 在index.html文件中第24行开始
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.20:90/ReportServer";
// 报表前端
window._contextPath = "http://192.168.0.20:90/Report";
// 主项目接口地址
window._mainServer = "http://192.168.0.20:90";
</script>
```

  - 打开 `/Report/preview.html` ,做如下修改,

```
# 在preview.html文件中第86行
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.20:90/ReportServer";
</script>
```

  - 在searchform.html文件中修改接口参数：

```
 window._server = "http://192.168.0.20:90/ReportServer"
// 报表前端
window._contextPath = "http://192.168.0.20:90/Report";
```

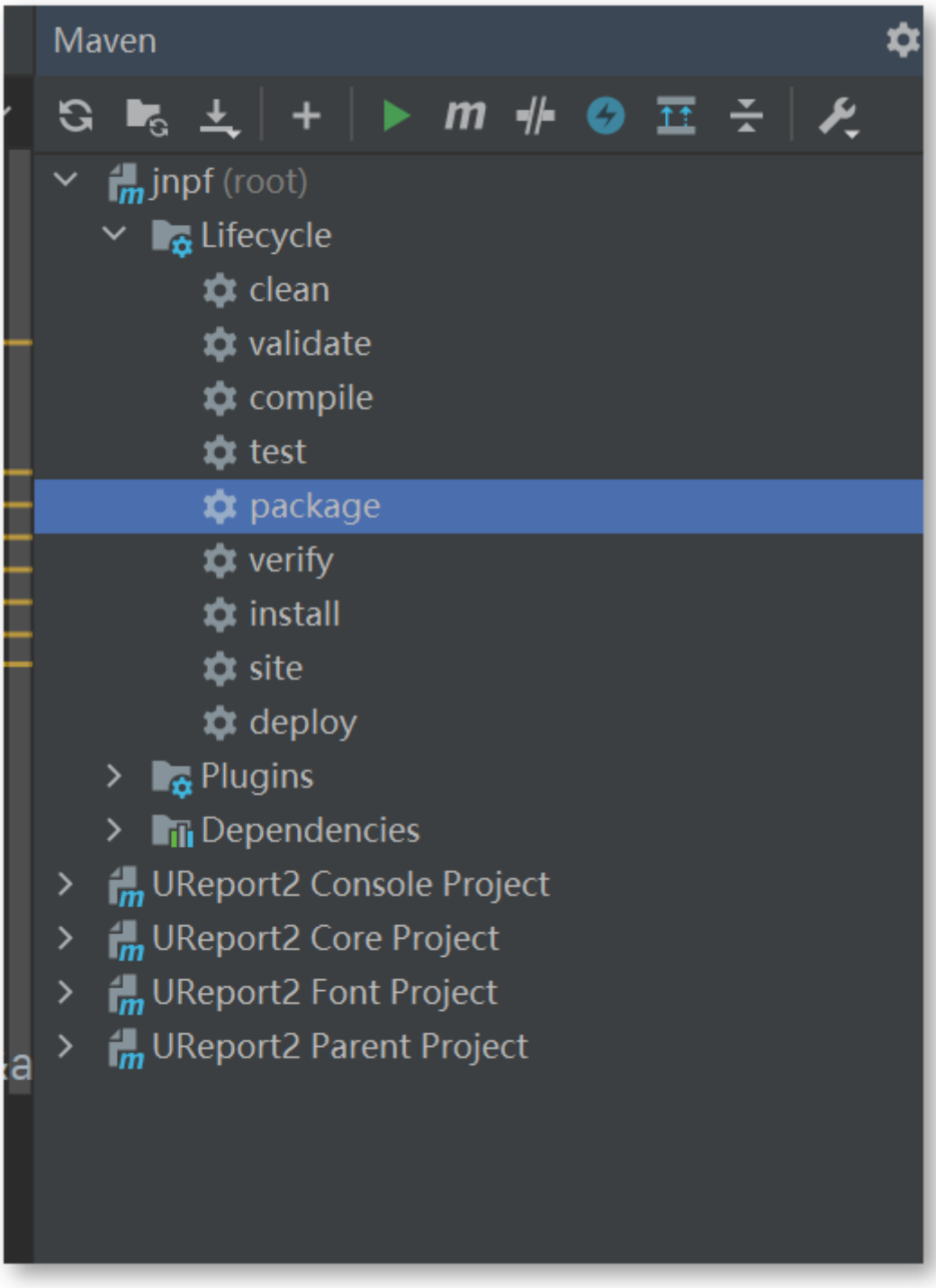
- 配置说明：
  - 本示例中， `http://192.168.0.20:90` 为项目在测试环境中访问入口，在部署中根据实际情况调整;
  - 报表接口中， `ReportServer` 为虚拟目录，需要在Nginx增加相关配置，具体配置如下：

```
# 数据报表接口配置
  location /ReportServer/ {
     proxy_pass http://localhost:30007/;
  }
```

后端

- `JNPF.DataReport` 项目修改application.yml中数据库配置

- 在 `IDEA` 右侧 `Maven` - `jnpf(root)` - `Lifecycle` 双击 `package` 打包项目



- 打包后上传至服务器，启动应用即可.

注：可将application.yml文件单独放置服务器文件夹下，方便修改配置

![图片](/api/project/8709815/files/24506572/imagePrevi

文档预览部署

部署指南

环境要求

- JDK1.8+
- OpenOffice或LiberOffice(Windows下已内置，CentOS或Ubuntu下会自动下载安装，MacOS下需要自行安装)

部署运行

1、开发环境

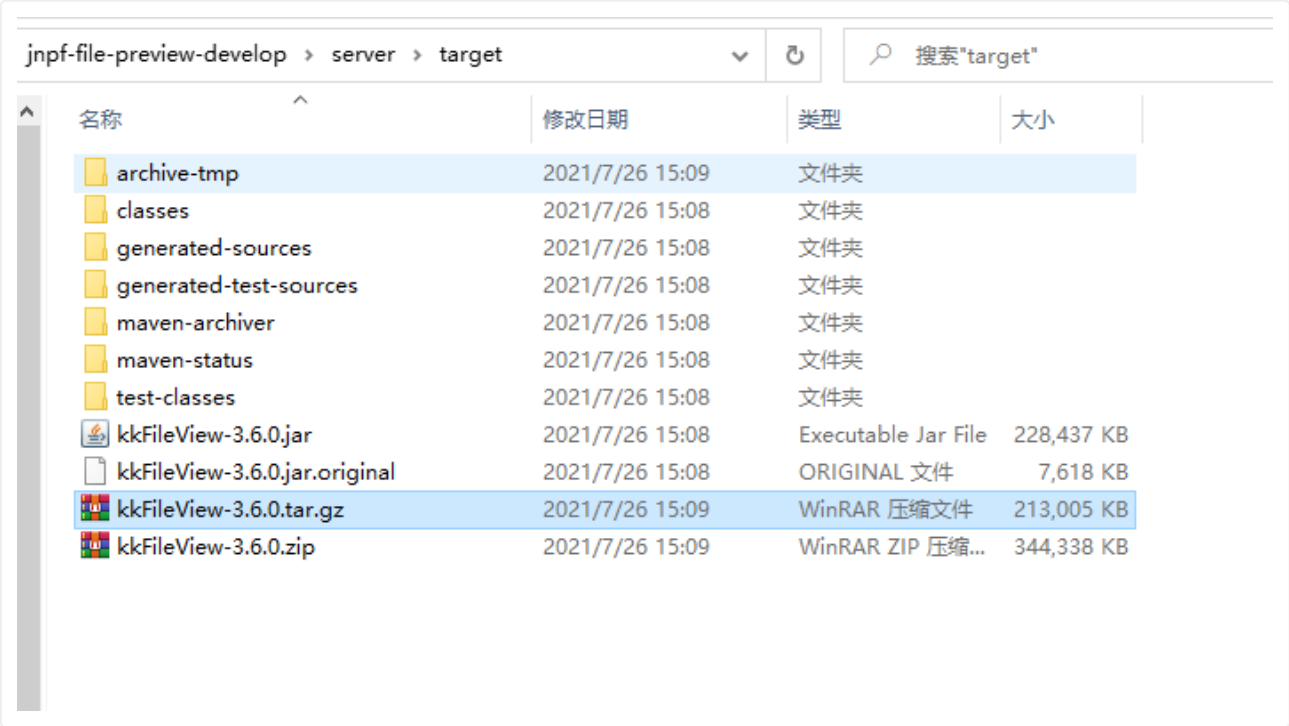
- a.IDEA导入项目
- b.调整配置，打开 `server/src/main/config/application.properties`
- c.启动项目， `server/src/main/java/cn/keking/ServerMain`
- d.打开 `http://localhost:30090` 测试页面

2、测试生产环境

- a.打包

  - 打开 `\server\target` 主要有以下几个文件
  - kkFileView-xxx.jar(一般用于更新)
  - kkFileView-xxx.zip(windows环境下首次部署)

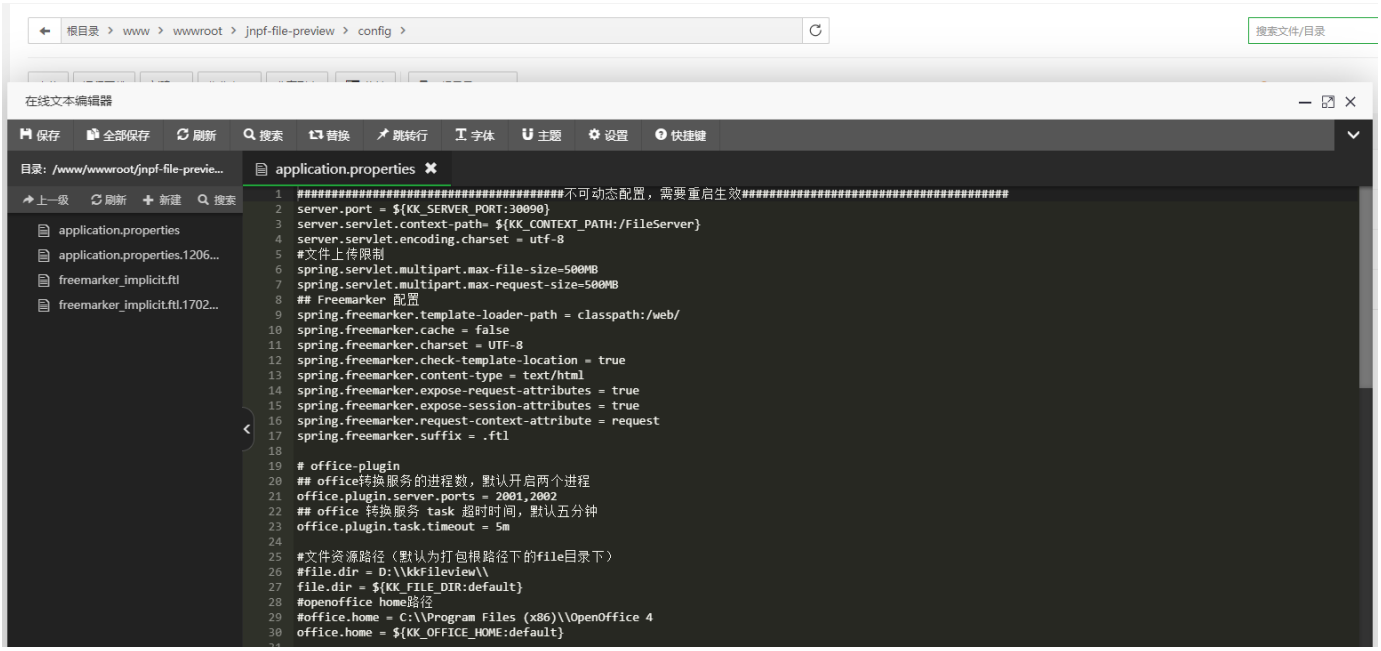
- kkFileView-xxx.tar.gz(Linux环境下首次部署)



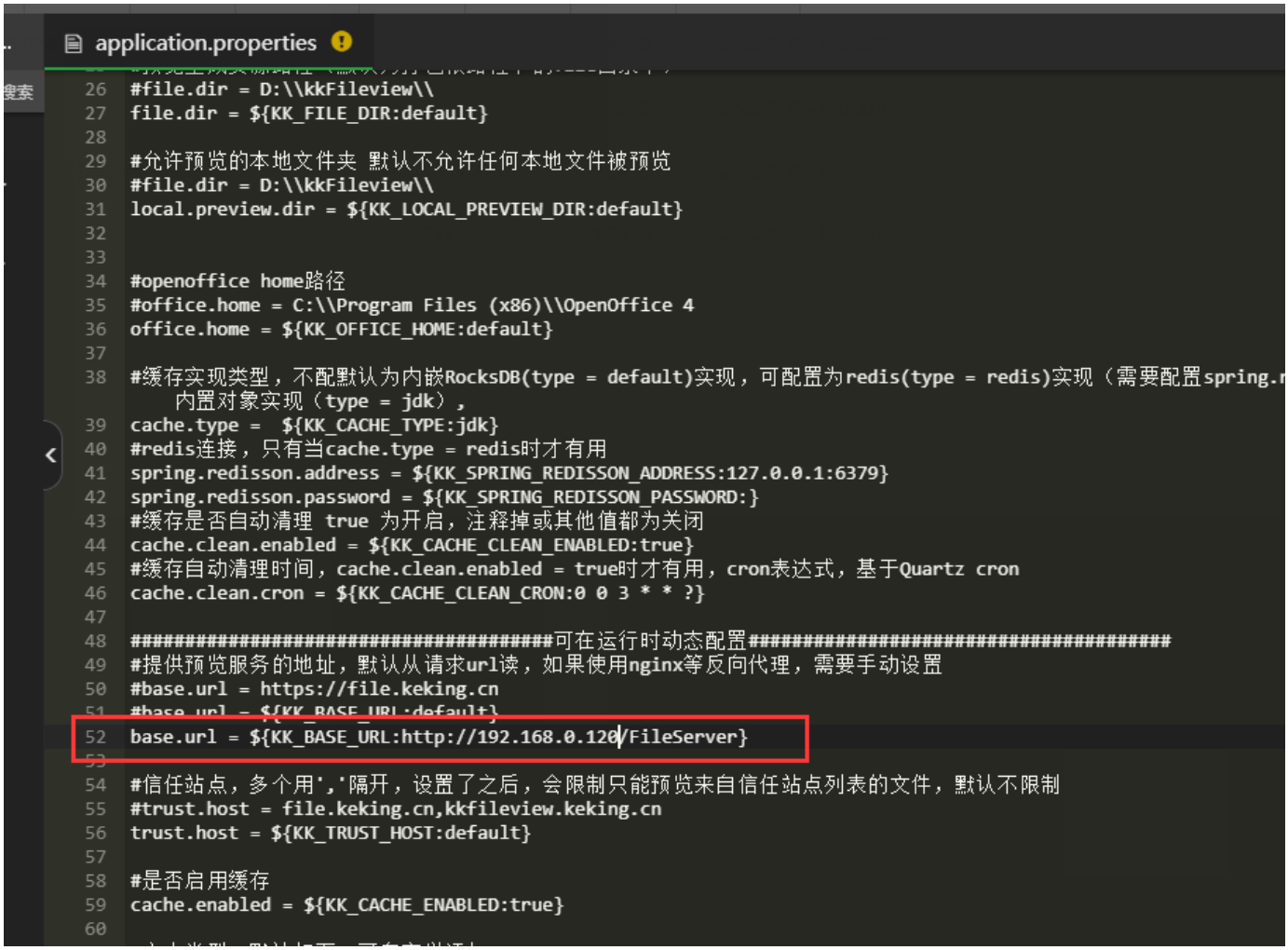
- b.上传至服务器
  - 打开解压后文件夹的bin目录，运行startup脚本(首次部署，之后更新及维护都在bin目录下)

- 修改文件预览配置文件如下两项,打开服务器上的`kkFileView-xxx/config/application.properties`（第3行和第45行，注：新版只需修改第52行：）

```
修改成：
server.servlet.context-path= ${KK_CONTEXT_PATH:/FileServer}
base.url = ${KK_BASE_URL:http://192.168.0.120/FileServer}
```



新版只需修改第52行：



- c.Nginx配置说明
  例如nginx的访问地址为 `https://java.jnpfsoft.com` ,文件预览部署在内网192.168.0.18服务器上，需要在nginx中添加反向代理如下

```
# 文件预览服务
location /FileServer {
  proxy_pass 192.168.0.18:30090;
}

# 解决文件预览服务无法加载js,css问题
location ~ /FileServer/*.*\.(js|css)?$ {
  proxy_pass 192.168.0.18:30090;
}
```

使用指南

1、单文件预览

```
// 要预览文件的访问地址
let previewUrl = 'http://127.0.0.1:8080/file/test.txt'

// 使用
```

```
http://127.0.0.1:30090/onlinePreview?url='+encodeURIComponent(Base64.encode(previewUrl))
```

2、http/https下载流url预览

```
// 要预览文件的访问地址
let previewUrl = 'http://127.0.0.1:8080/filedownload?fileId=1'

// 使用
http://127.0.0.1:30090/onlinePreview?url='+encodeURIComponent(Base64.encode(previewUrl))
```

3、更多参考官方文档（`https://kkfileview.keking.cn/zh-cn/docs/usage.html`）

常见问题

1、预览乱码(字体问题)

大部分Linux系统上并没有预装中文字体或字体不全，需要把常用字体拷贝到Linux服务器上，具体操作如下： 下载如下字体包`http://kkfileview.keking.cn/fonts.zip`文件解压完整拷贝到Linux下的`/usr/share/fonts`目录。然后依次执行`mkfontscale`、`mkfontdir`、`fc-cache`使字体生效

> 特别说明：安装字体前确保Linux服务器已安装mkfontscale、fontconfig，如未安装运行以下命令

CentOS运行如下命令

```
# 使mkfontscale和mkfontdir命令正常运行
yum install mkfontscale

# 使fc-cache命令正常运行
yum install fontconfig
```

Ubuntu运行如下命令

```
# 使mkfontscale和mkfontdir命令正常运行
sudo apt-get install ttf-mscorefonts-installer

# 使fc-cache命令正常运行
sudo apt-get install fontconfig
```

2、更多问题请参考官方文档说明(`https://kkfileview.keking.cn/zh-cn/docs/faq.html`)

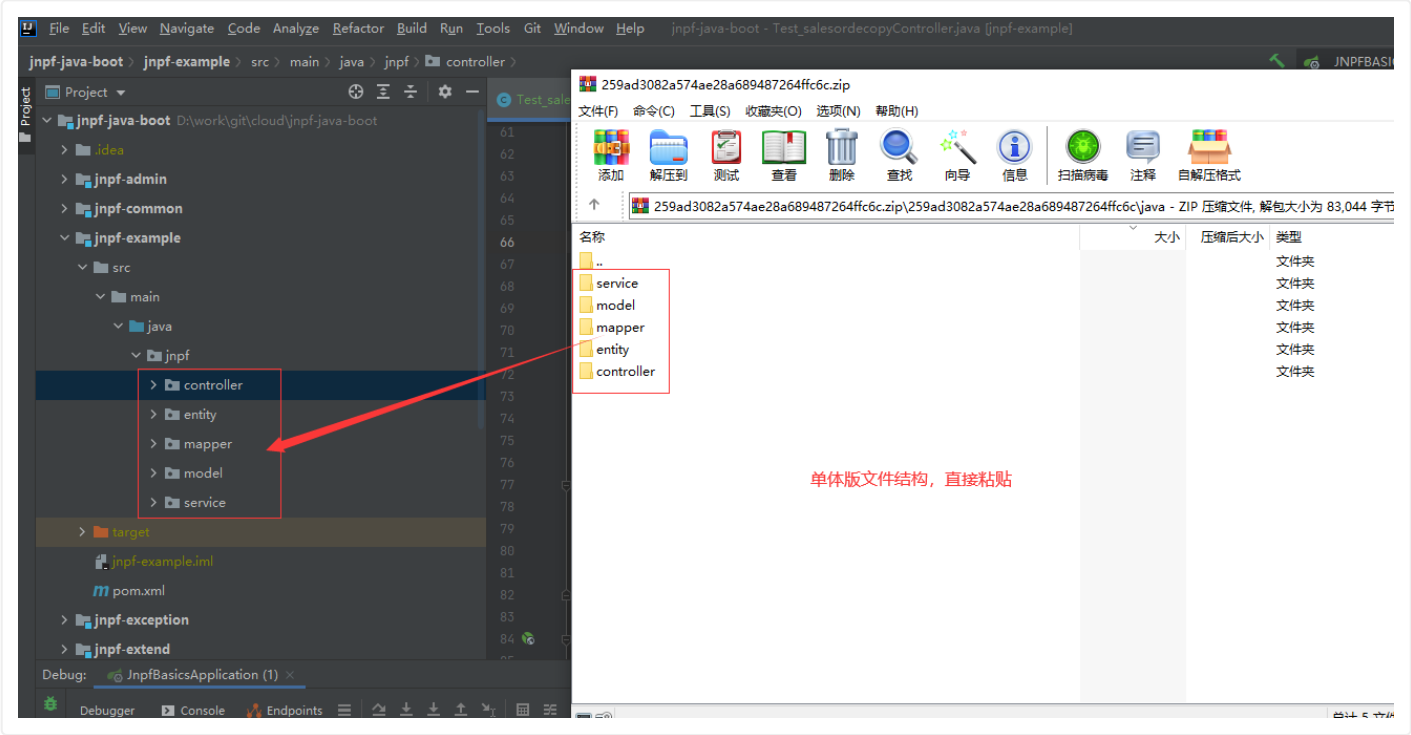
开发指南

代码生成器的使用

生成的代码结构如下

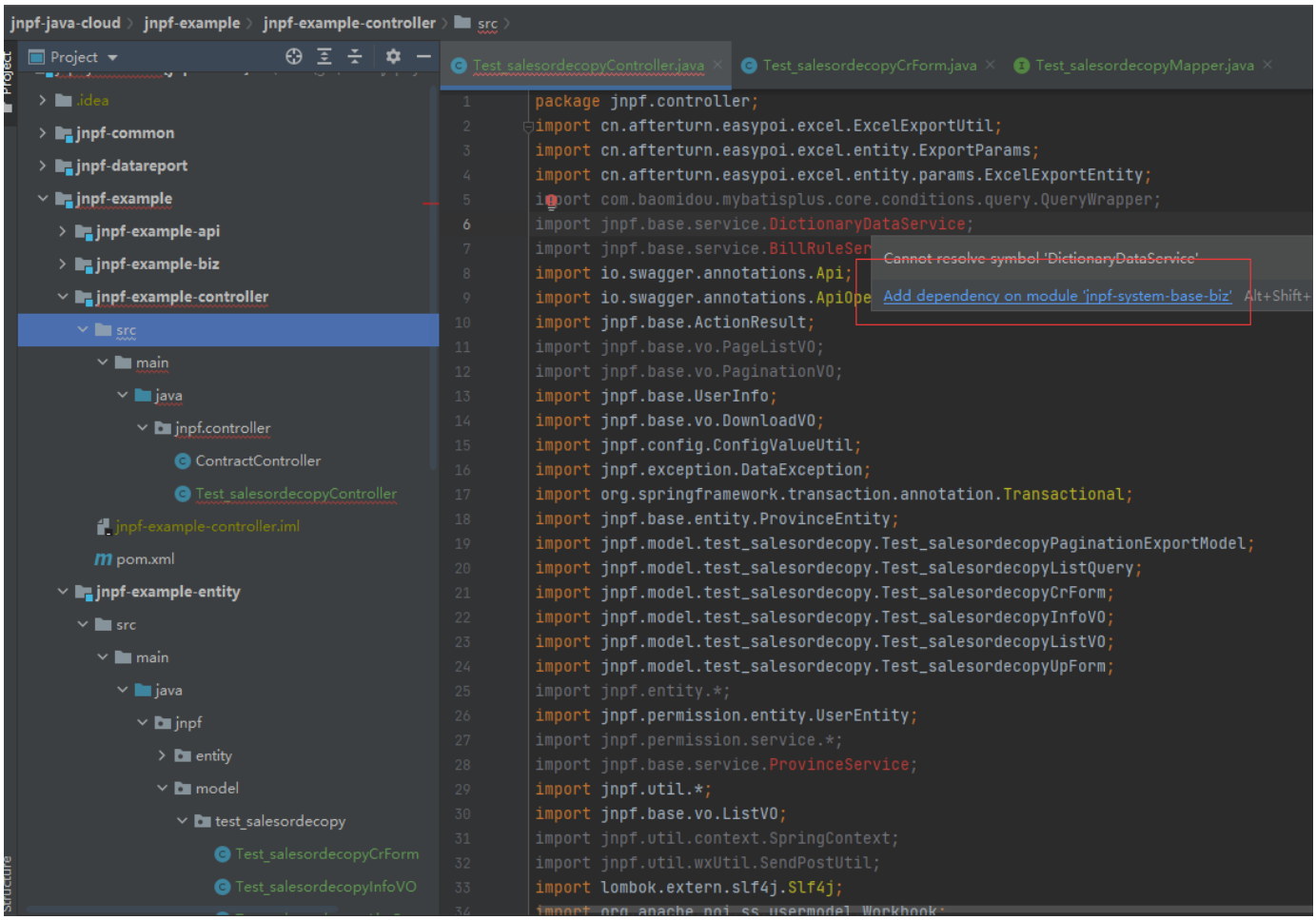
example模块为例

- 打开java文件夹中的文件



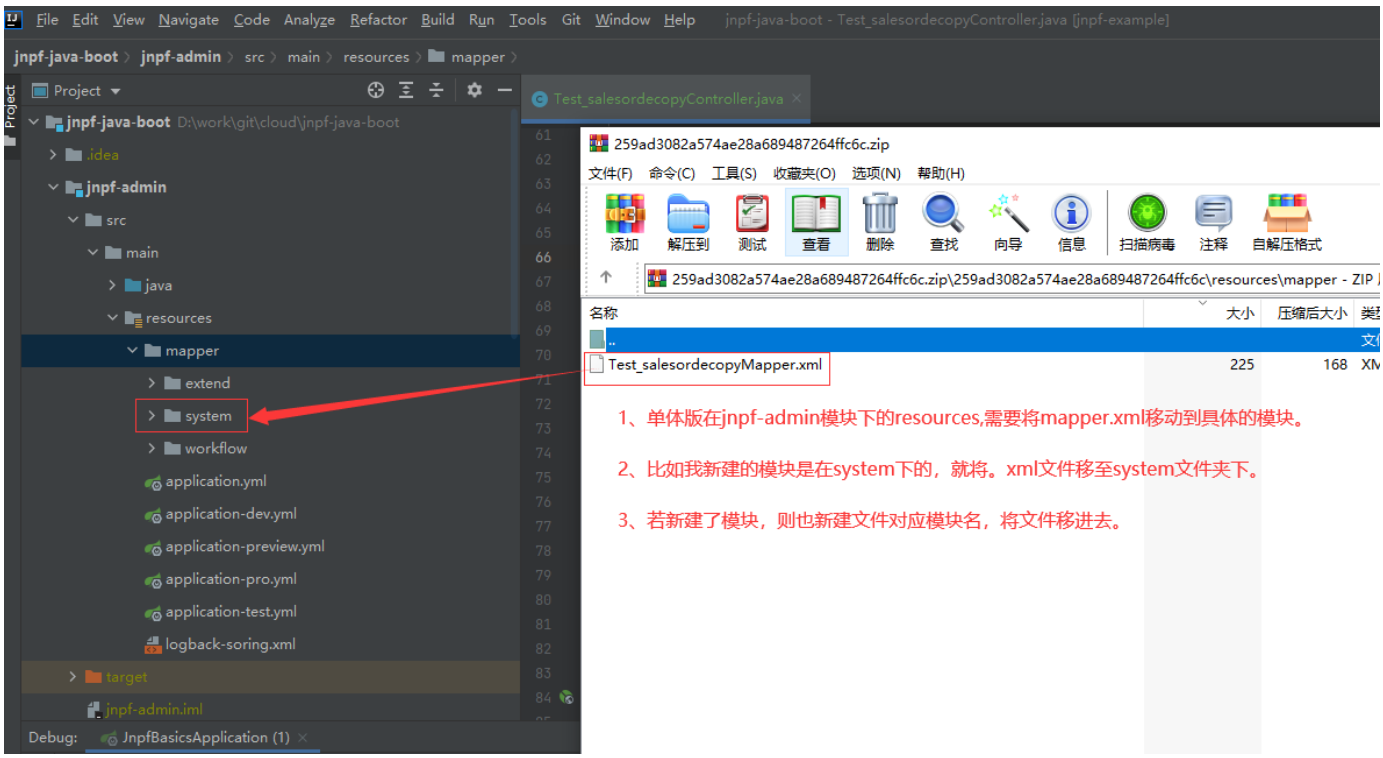
复制右边的全部文件，粘贴在jnpf上即可。

报红问题：有些需要引入别的模块代码会报红，此时将鼠标移动到报红的地方进行导包，如图



- 移动resources文件夹

需要进入resources->mapper->mapper.xml



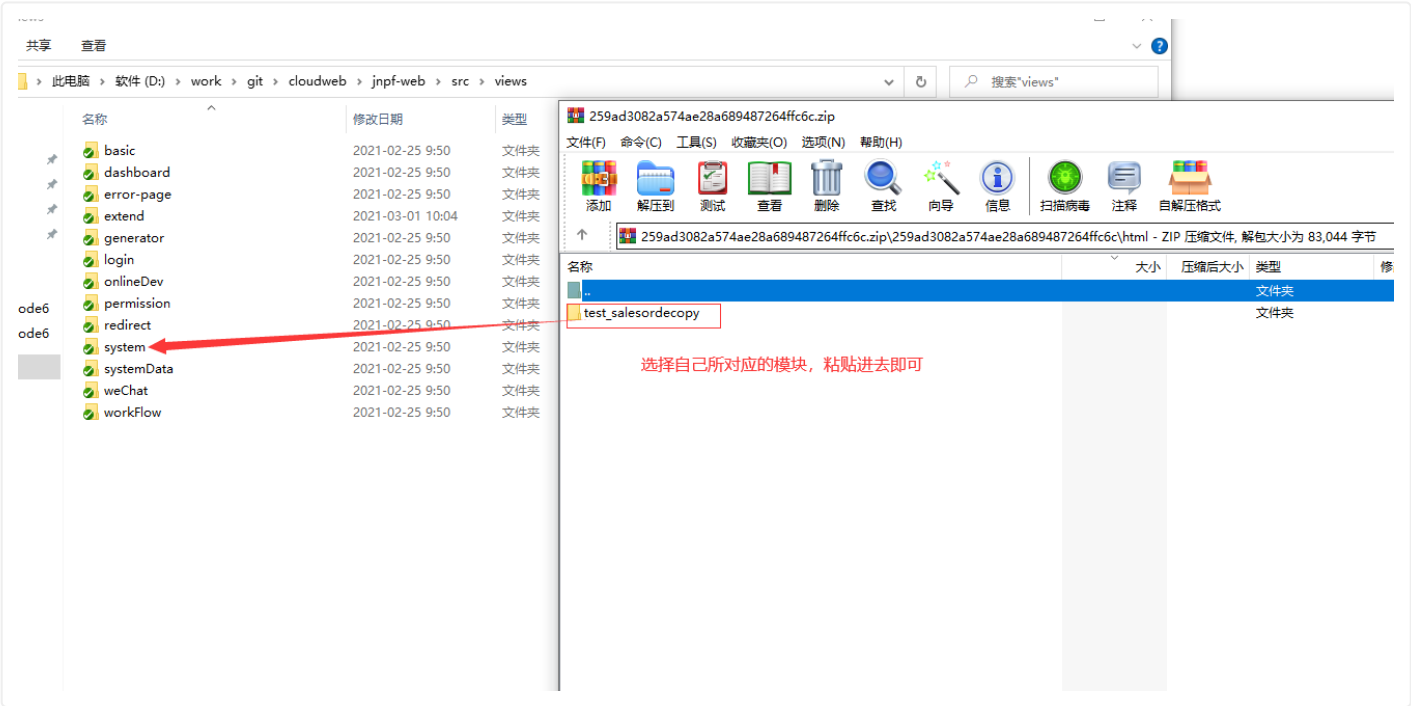
搬迁前端代码

来自 CODING

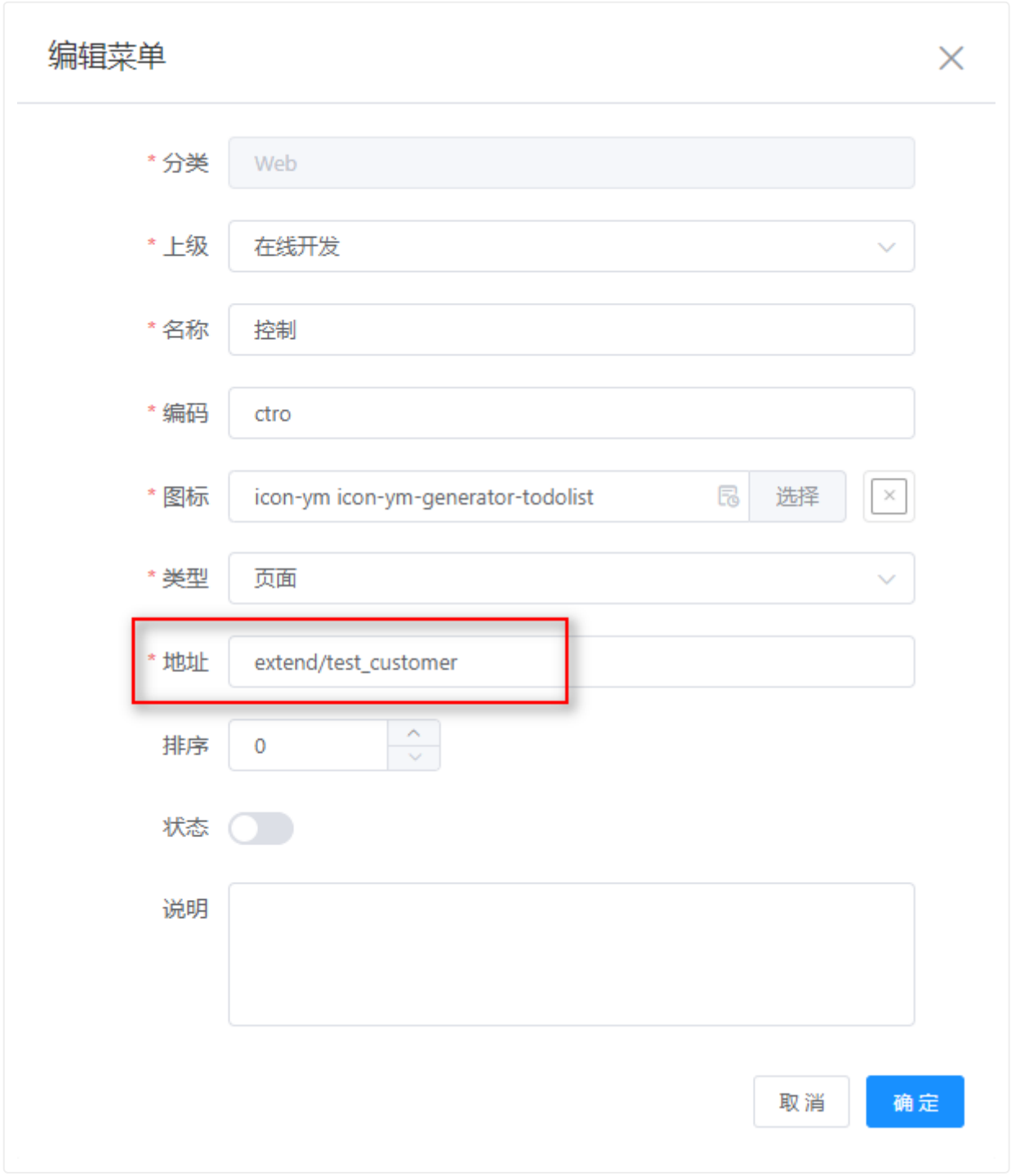
/

- 打开 `html` 文件夹

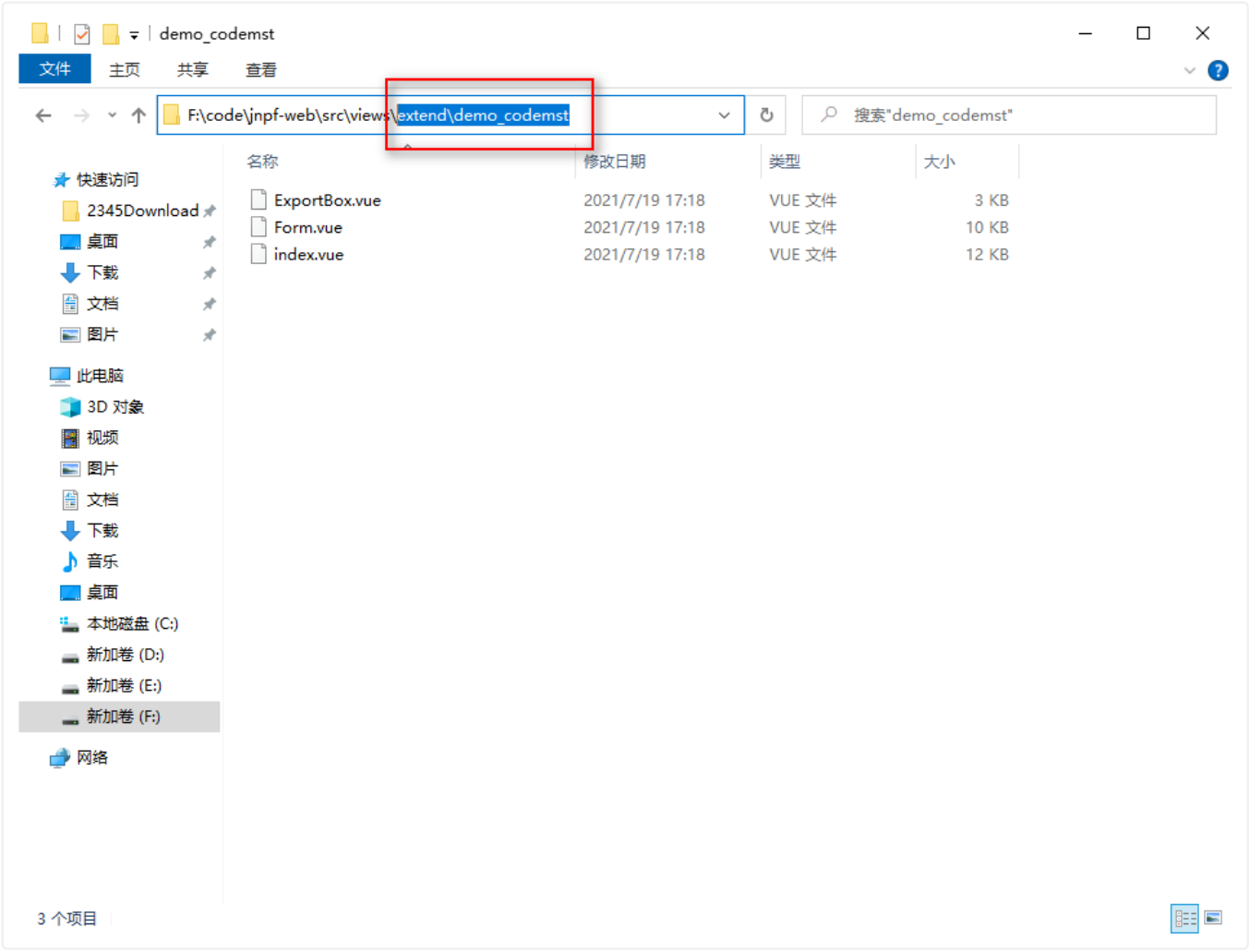
- 进入前端文件夹\src\views



- 登录后台后选择系统管理--系统菜单配置页面



注：地址栏填写前端代码放置页面，注意“/”隔开



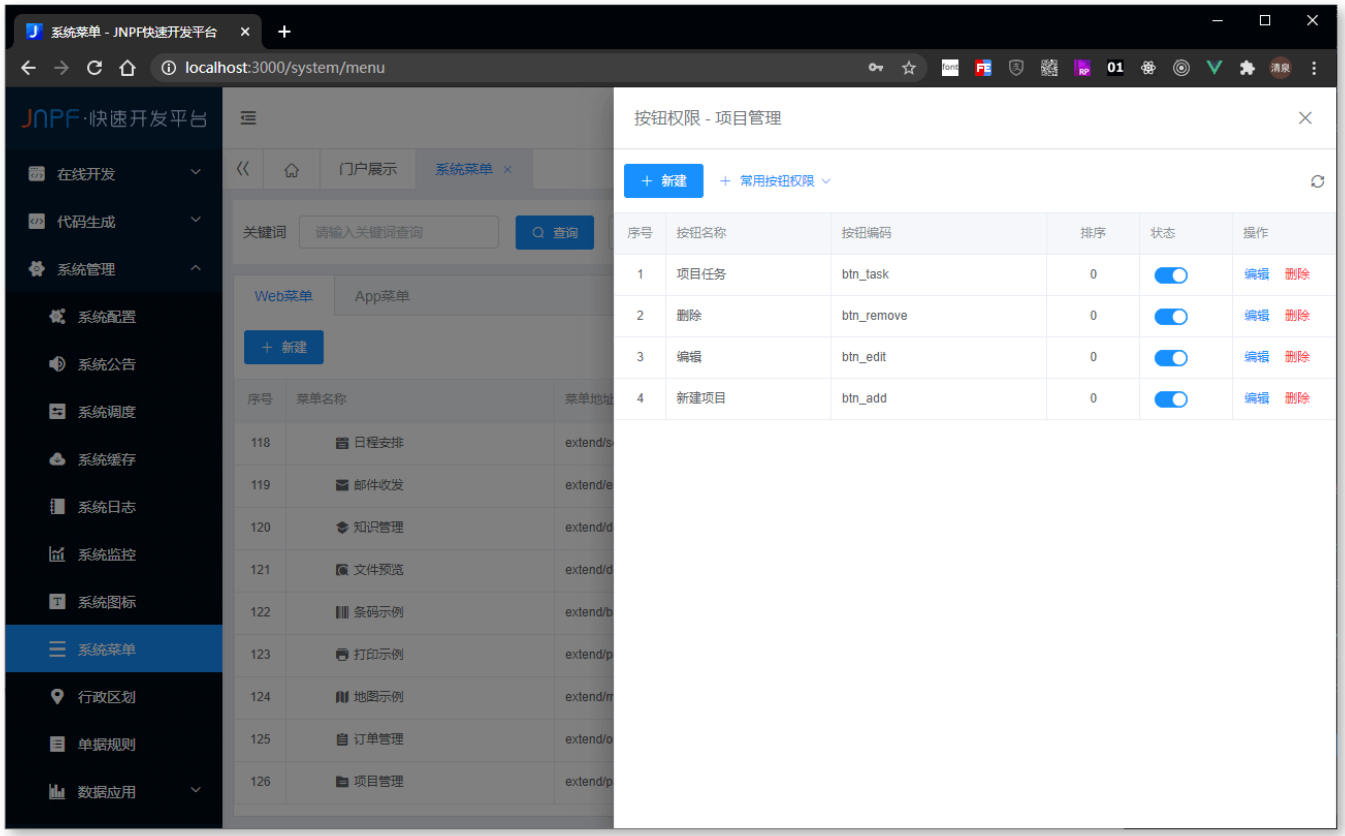
如何控制权限

- 菜单权限控制
- 按钮权限控制
- 列表权限控制
- 数据权限控制

按钮权限控制

示例：

1）选择菜单管理->扩展->项目管理->更多->按钮权限



2）前端代码添加

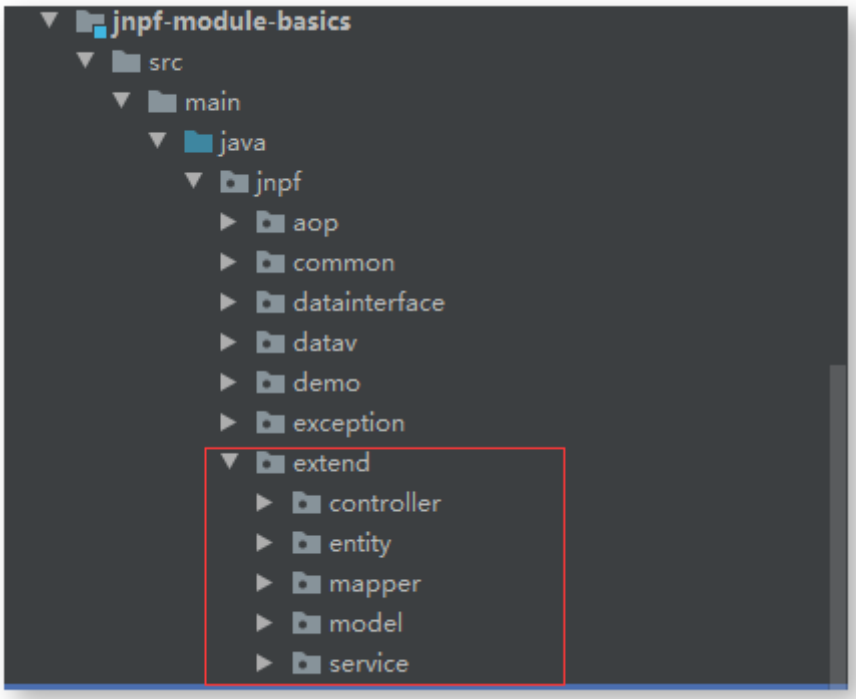
```
<el-button v-has="'btn_add'">新建项目</el-button>
```

在button标签里面添加 `v-has="字段名称"` 即可实现按钮权限控制

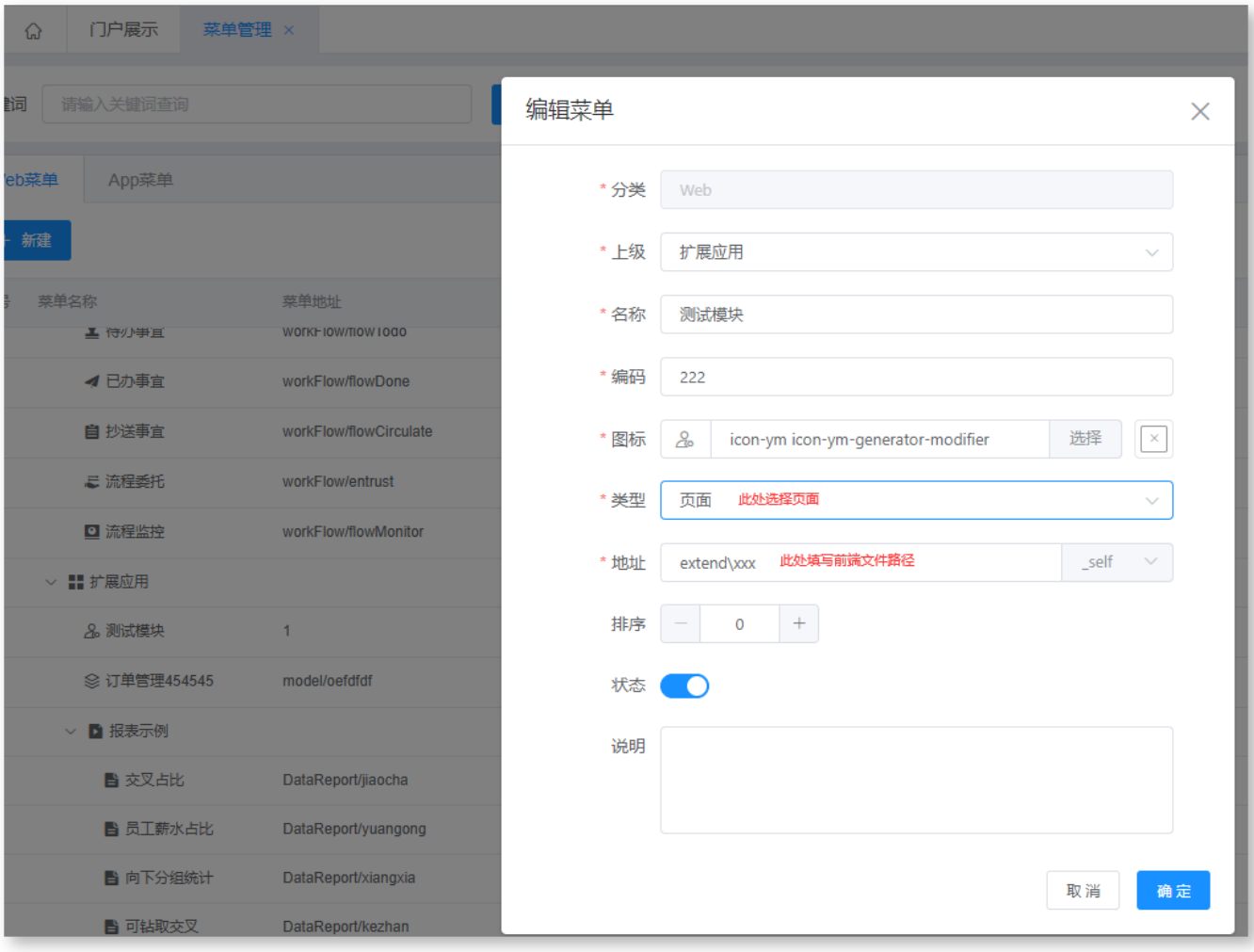
管理员默认不受权限控制，添加和启用按钮权限之后若想控制其他人员的按钮权限可以使用管理员账户在权限管理或者角色管理里面控制角色的按钮权限

菜单权限控制

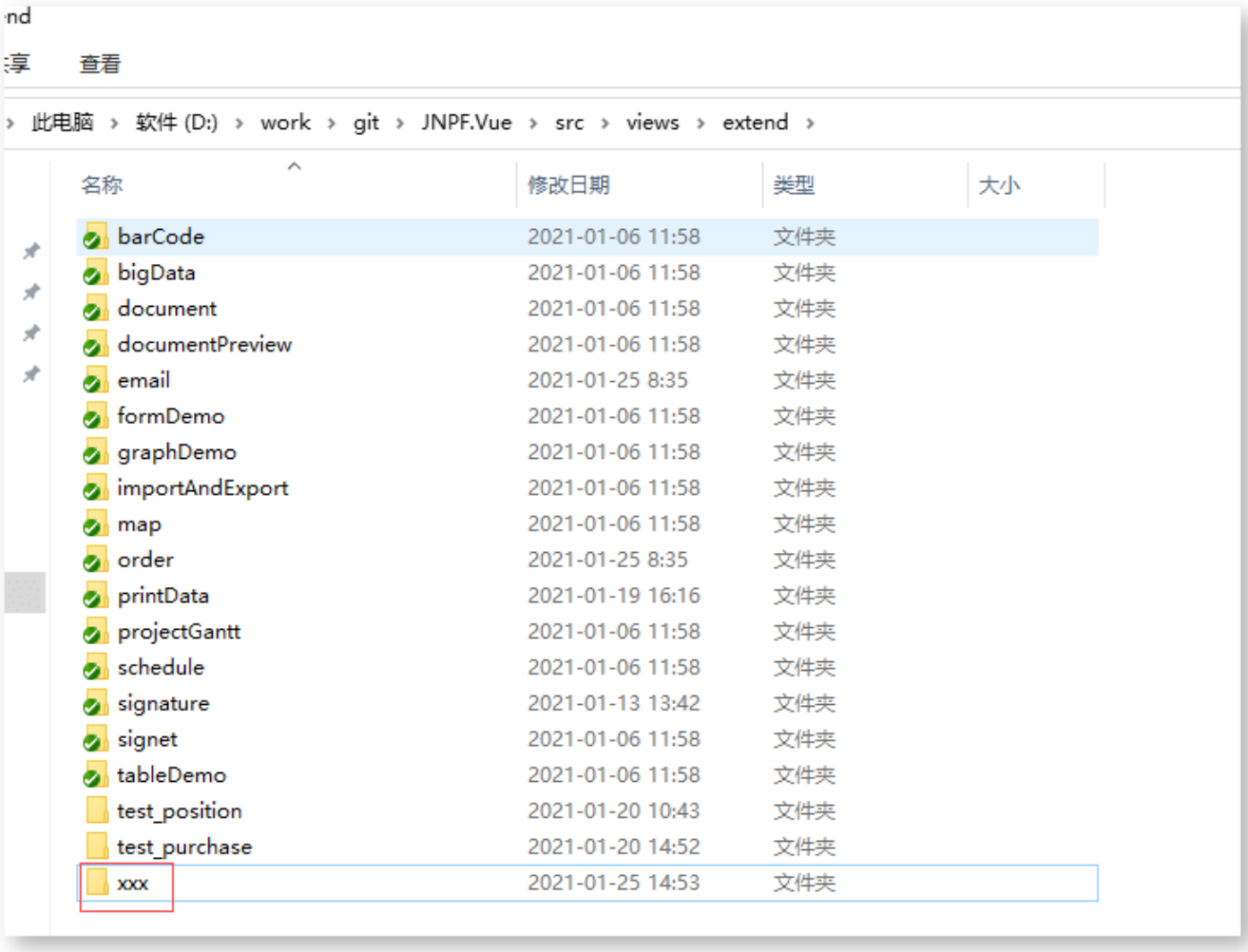
以 `扩展应用` 为例子



添加一个新功能时，可以选择在现有模块上添加，或者新建一个模块，结构如扩展应用模块所示



由于功能在扩展模块则将功能放在 `\src\views\extend` 模块下

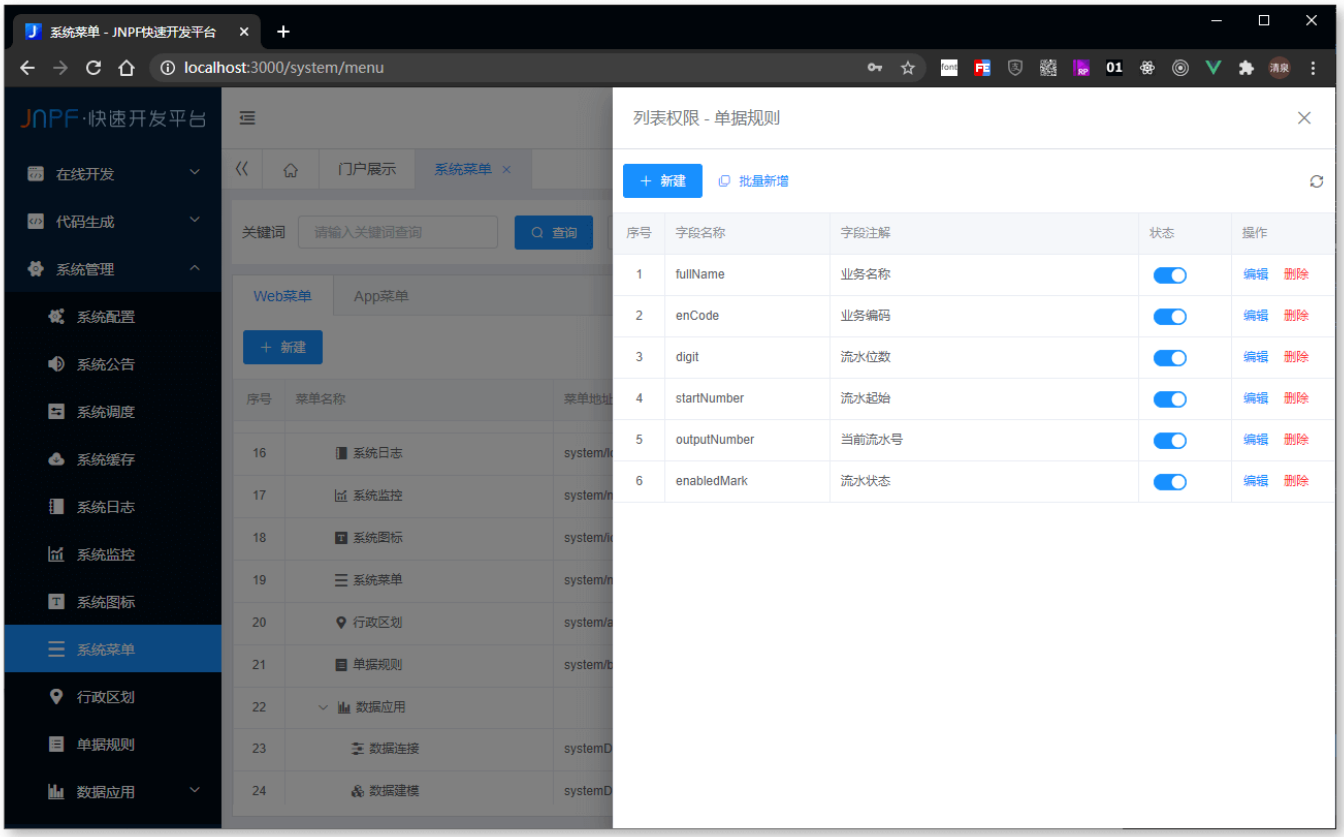


添加完之后若代码正常则可以正常显示，管理员默认不受权限控制，添加和启用菜单之后若想控制其他人员的菜单权限可以使用管理员账户在权限管理或者角色管理里面控制角色的菜单权限

列表权限控制

示例：

- 选择菜单管理->单据规则->更多->列表权限



- 前端代码添加

```
<el-table-column prop="fullName" label="业务名称" width="200" v-if="jnpf.hasP('fullName')" />
```

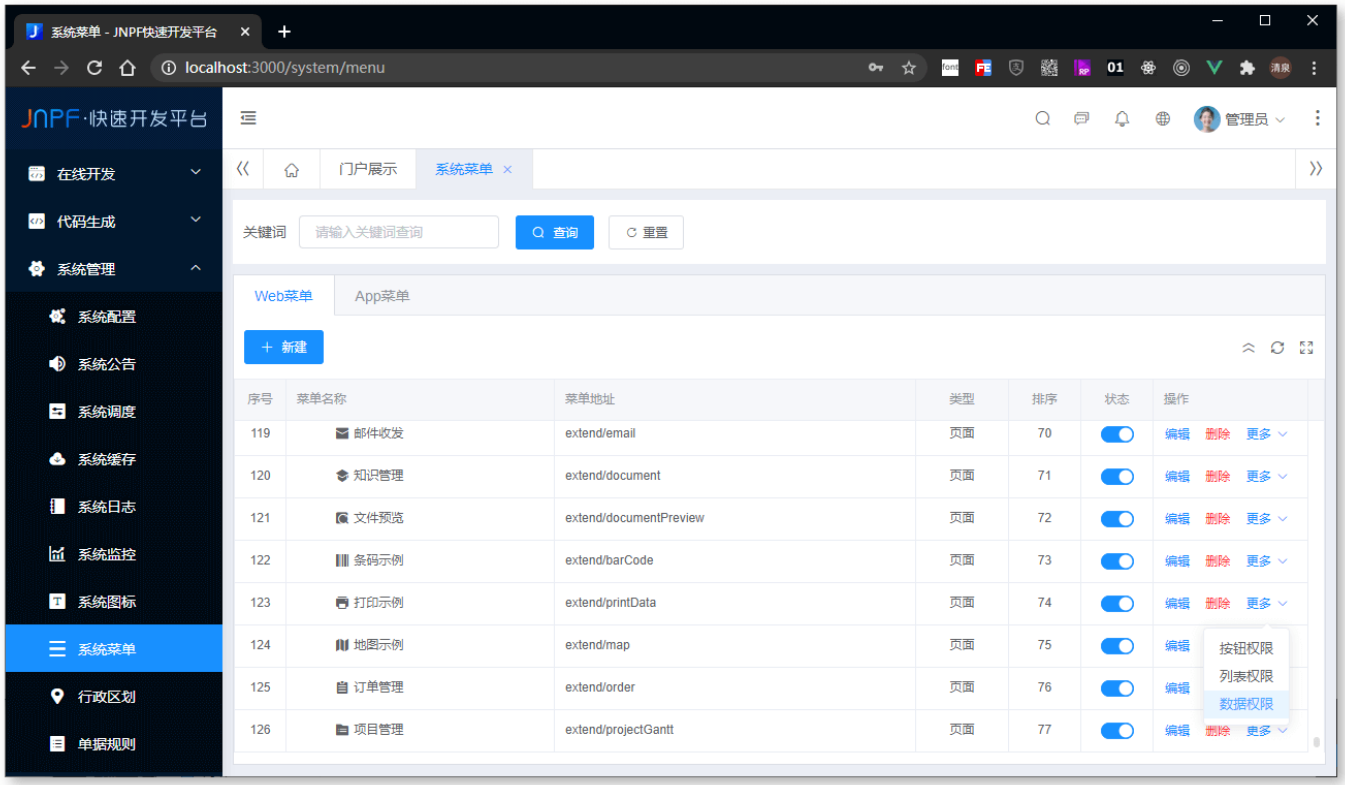
在前端列表字段上添加 `v-if="jnpf.hasP('fullName')"` ，与菜单管理中添加的字段相同，即可添加列表权限

管理员默认不受权限控制，添加和启用列表权限之后若想控制其他人员的列表权限可以使用管理员账户在权限管理或者角色管理里面控制角色的列表权限限控制，添加和启用按钮权限之后若想控制其他人员的按钮权限可以使用管理员账户在权限管理或者角色管理里面控制角色的按钮权限

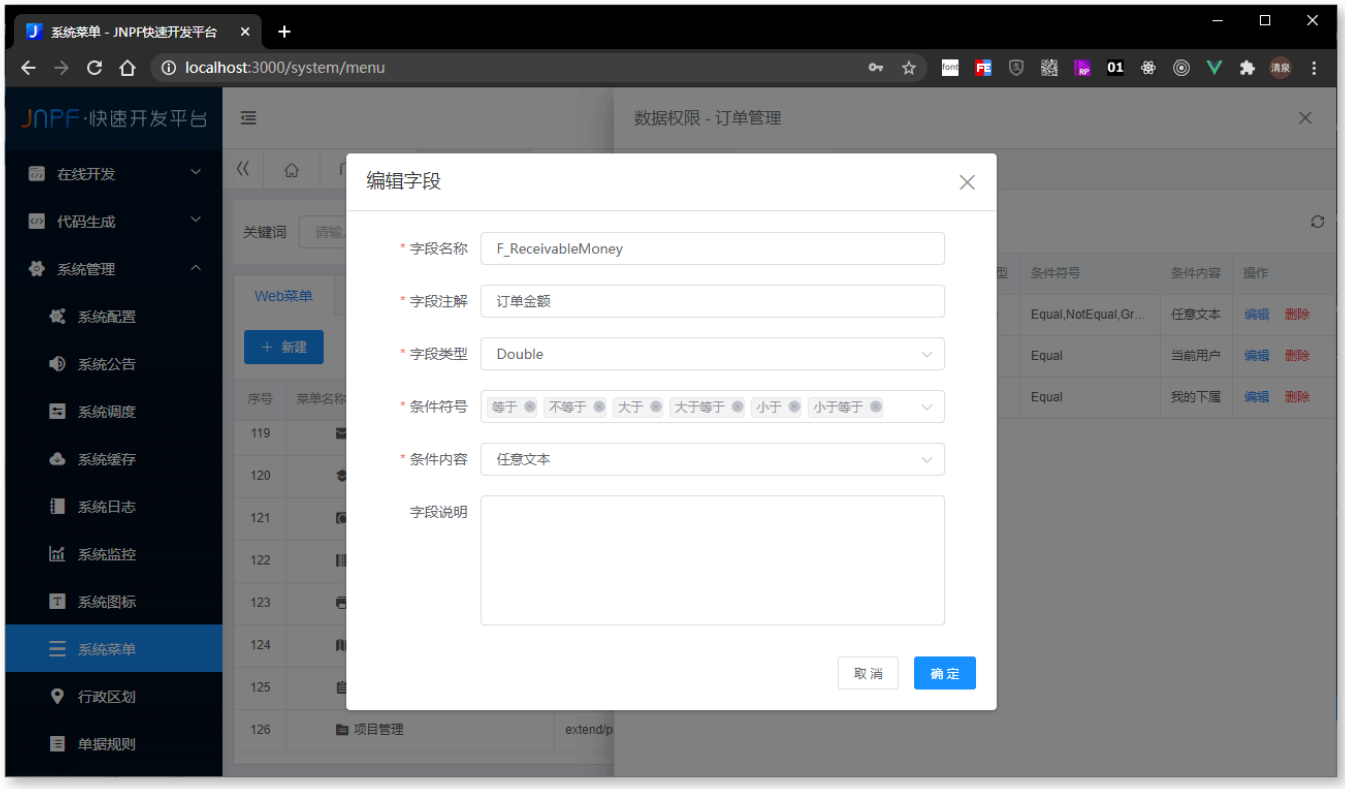
数据权限控制

> 在设置数据权限时，为避免单纯在前端设置导致数据紊乱或者出现数据bug，需要同时在数据库和后台代码中添加数据权限过滤才能生效，以下是设置数据权限的步骤。

以订单管理为例，先在页面上添加数据权限,选择菜单管理->订单管理->更多->数据权限

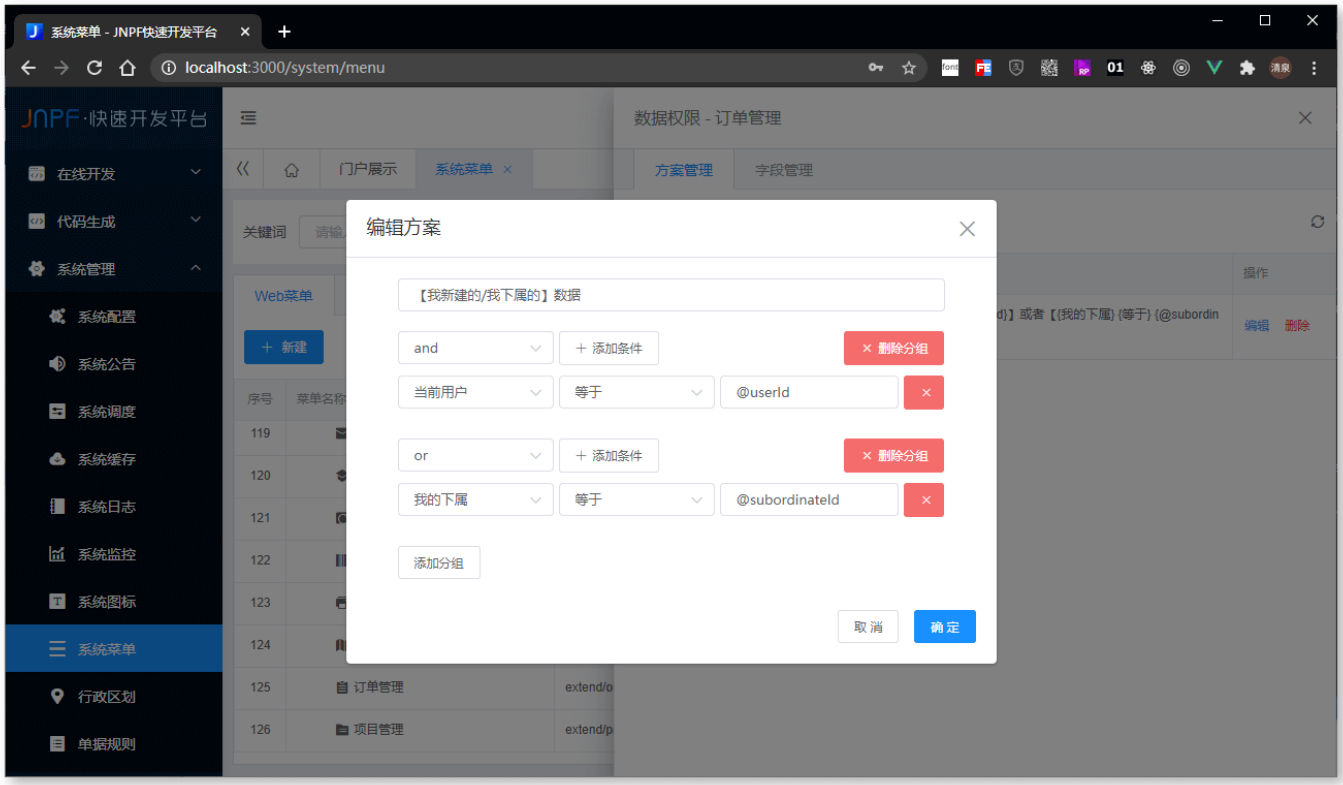


1）在 字段管理 中添加字段和查询内容

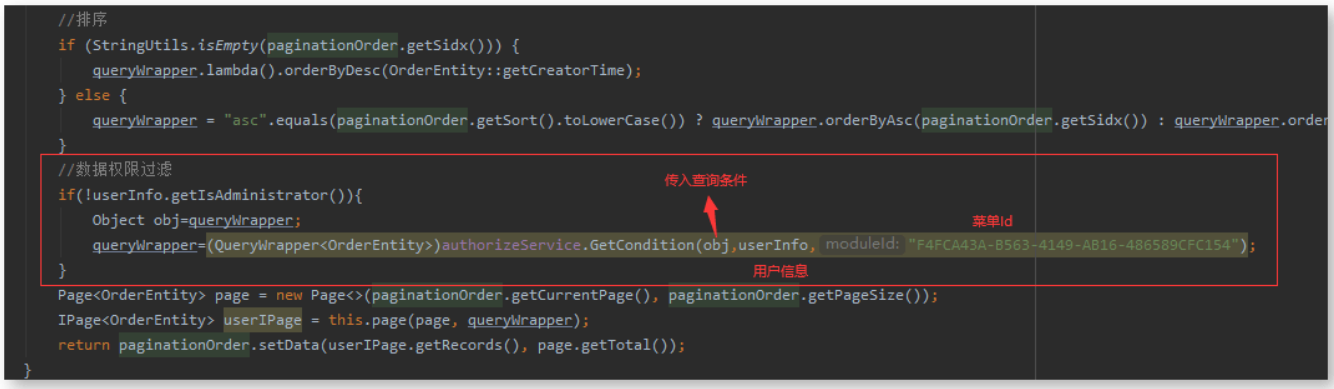


- 字段名称与数据库字段要一致

2）在 方案管理 中搭配字段组合



3）后台代码中添加以下代码



一般是在排序之前或之后添加权限过滤代码

通过框架语法查询的数据库数据添加以下代码

```
UserInfo userInfo = userProvider.get();
//数据权限过滤
if(!userInfo.getIsAdministrator()){
    Object obj=queryWrapper;
    queryWrapper=(QueryWrapper<OrderEntity>)authorizeService.getCondition(obj,userIn
fo,"xxxxxxxxx");
}
```

若是通过sql拼接查询数据库数据则添加以下代码

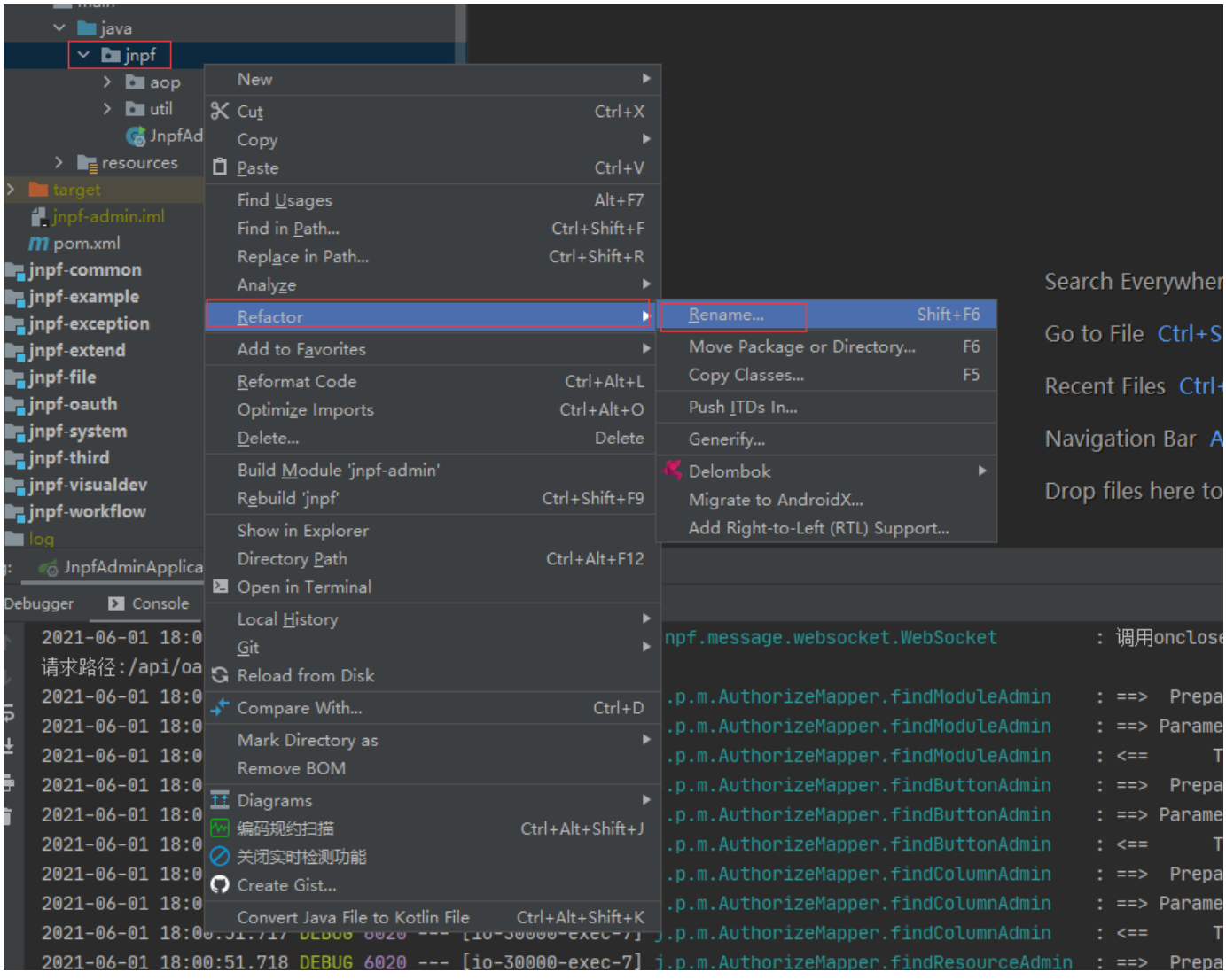
```
UserInfo userInfo = userProvider.get();
//数据权限过滤
if (!userInfo.getIsAdministrator()) {
    sql.append(authorizeService.GetConditionSql(userInfo, "xxxxxxxxxxx"));
}
```

管理员默认不受权限控制，添加和启用数据权限之后若想控制其他人员的数据权限可以使用管理员账户在权限管理或者角色管理里面控制角色的数据权限

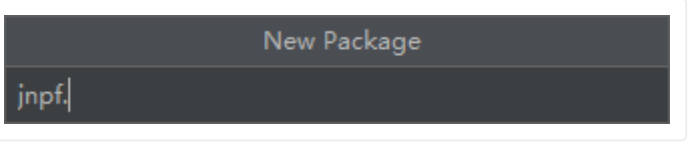
如何修改包名

修改包名：

1.找到要改的包，右键后点击如图所示的选项

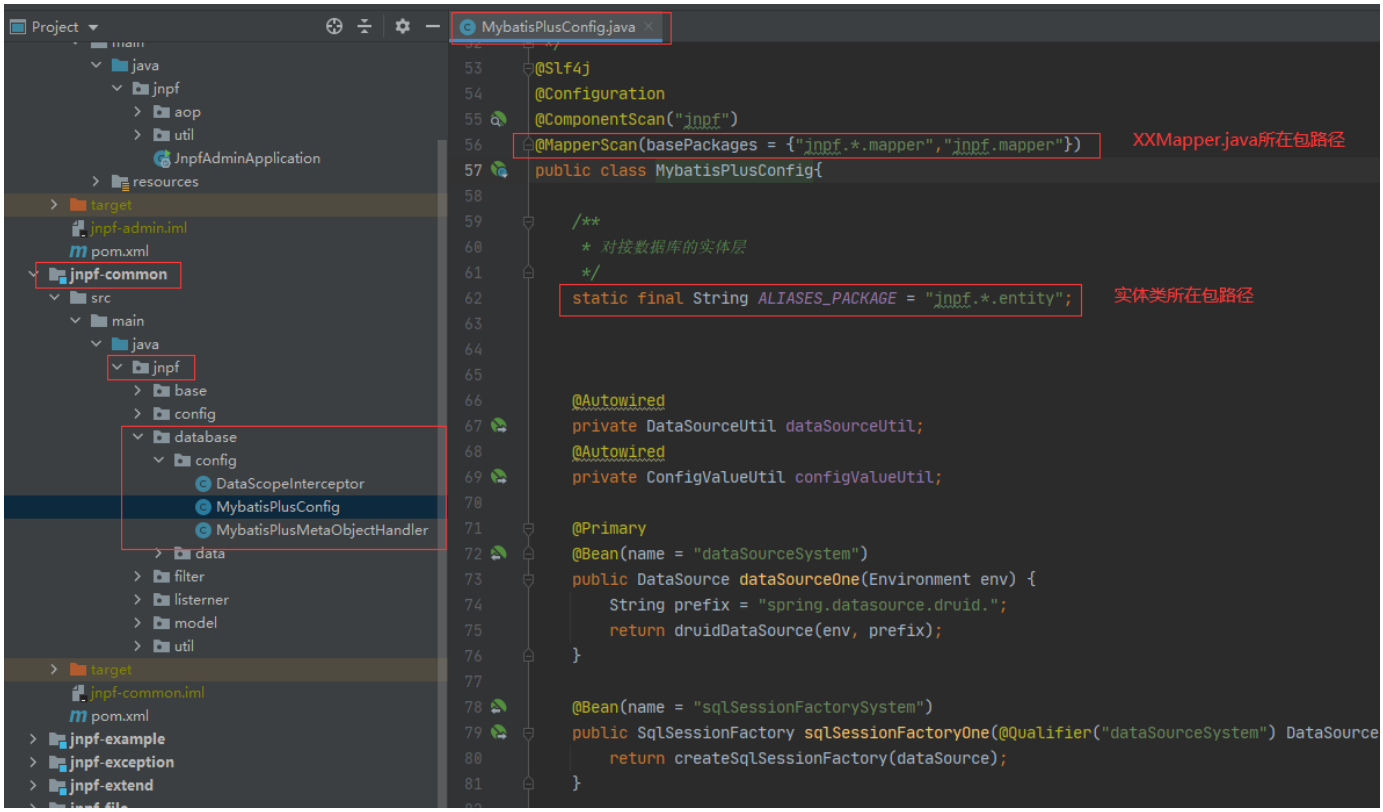


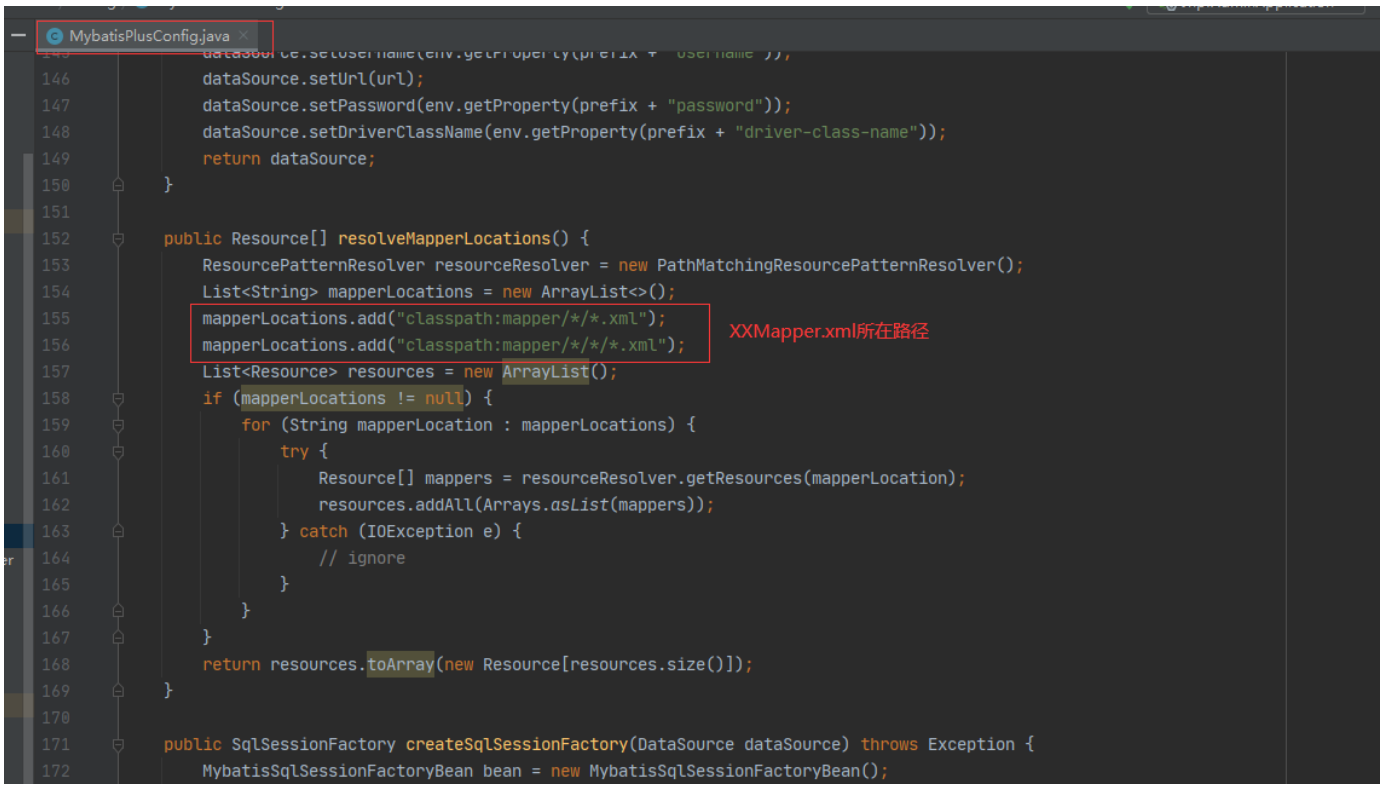
弹出new package的窗口后进行修改

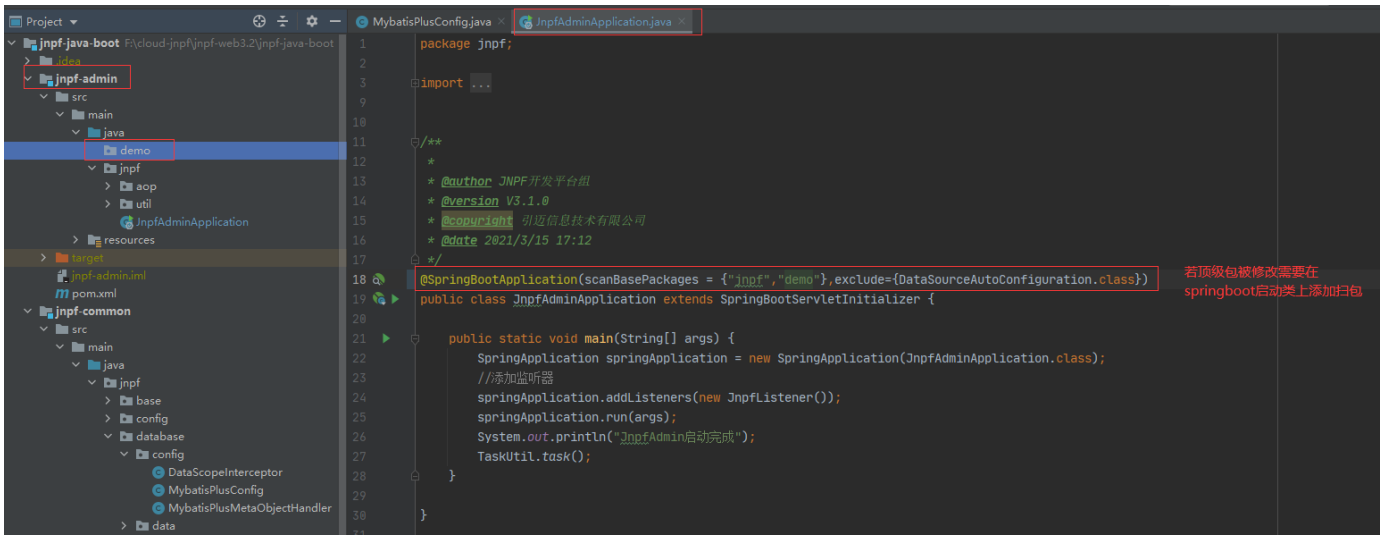


2.若实体类、XXMapper.java或XXMapper.xml文件的路径发生变化，需要修改MybatisPlusConfig.java中的实体类

路径和mapper路径,如图所示：

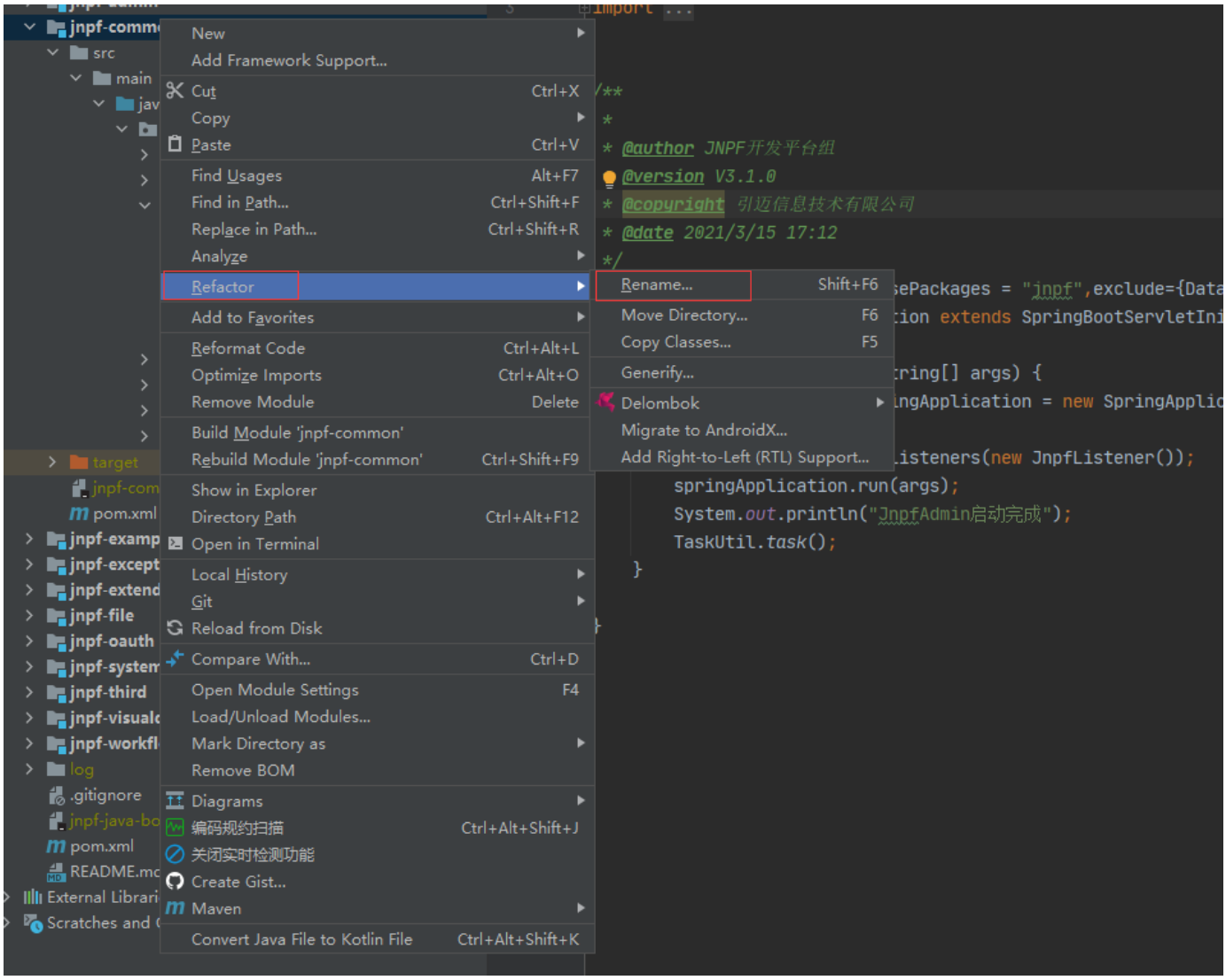




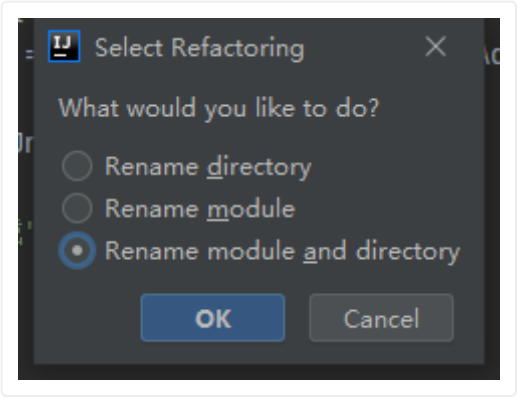


修改Maven名：

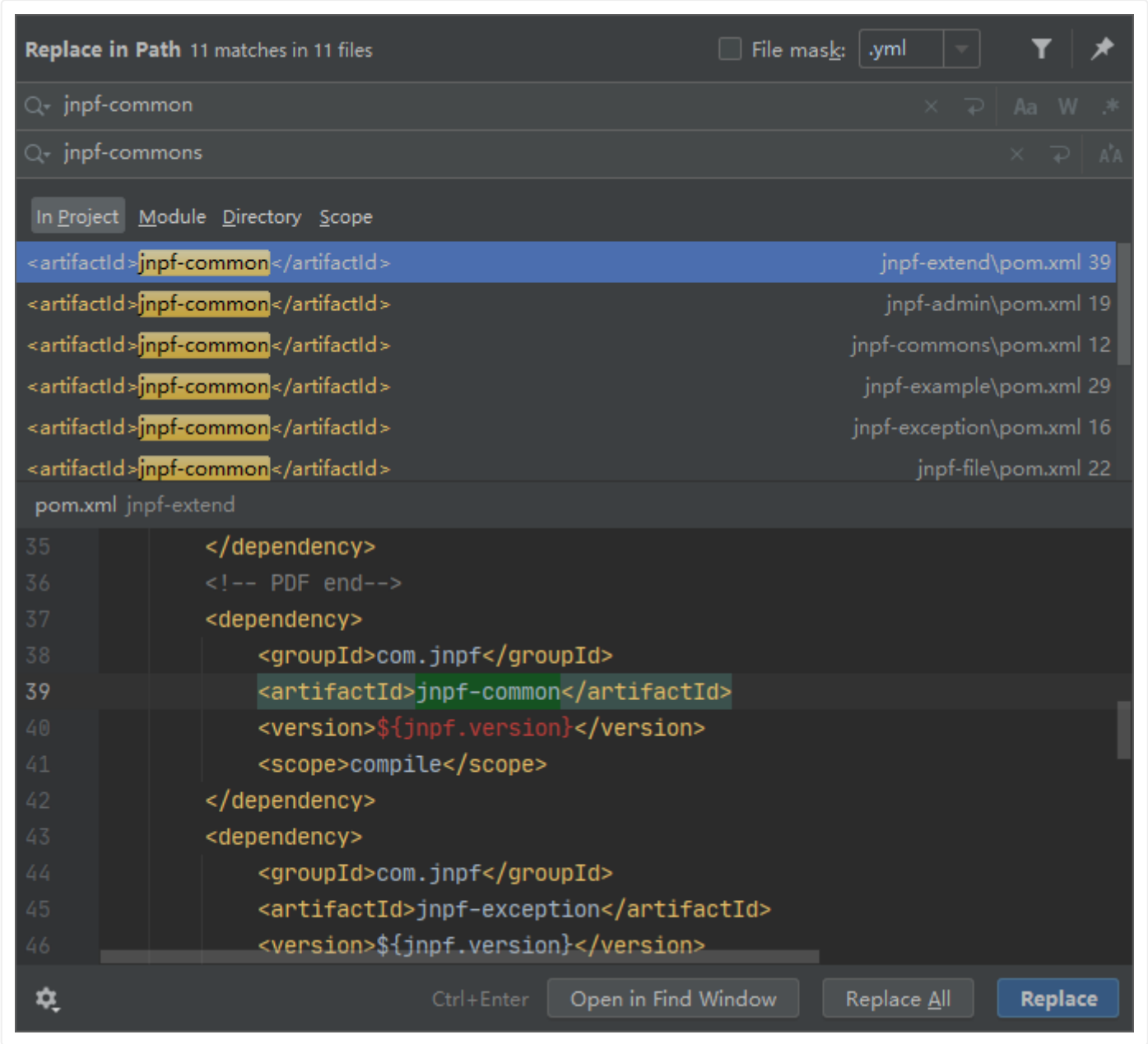
1.找到要修改的Maven项目，右键点击如图所示的选项



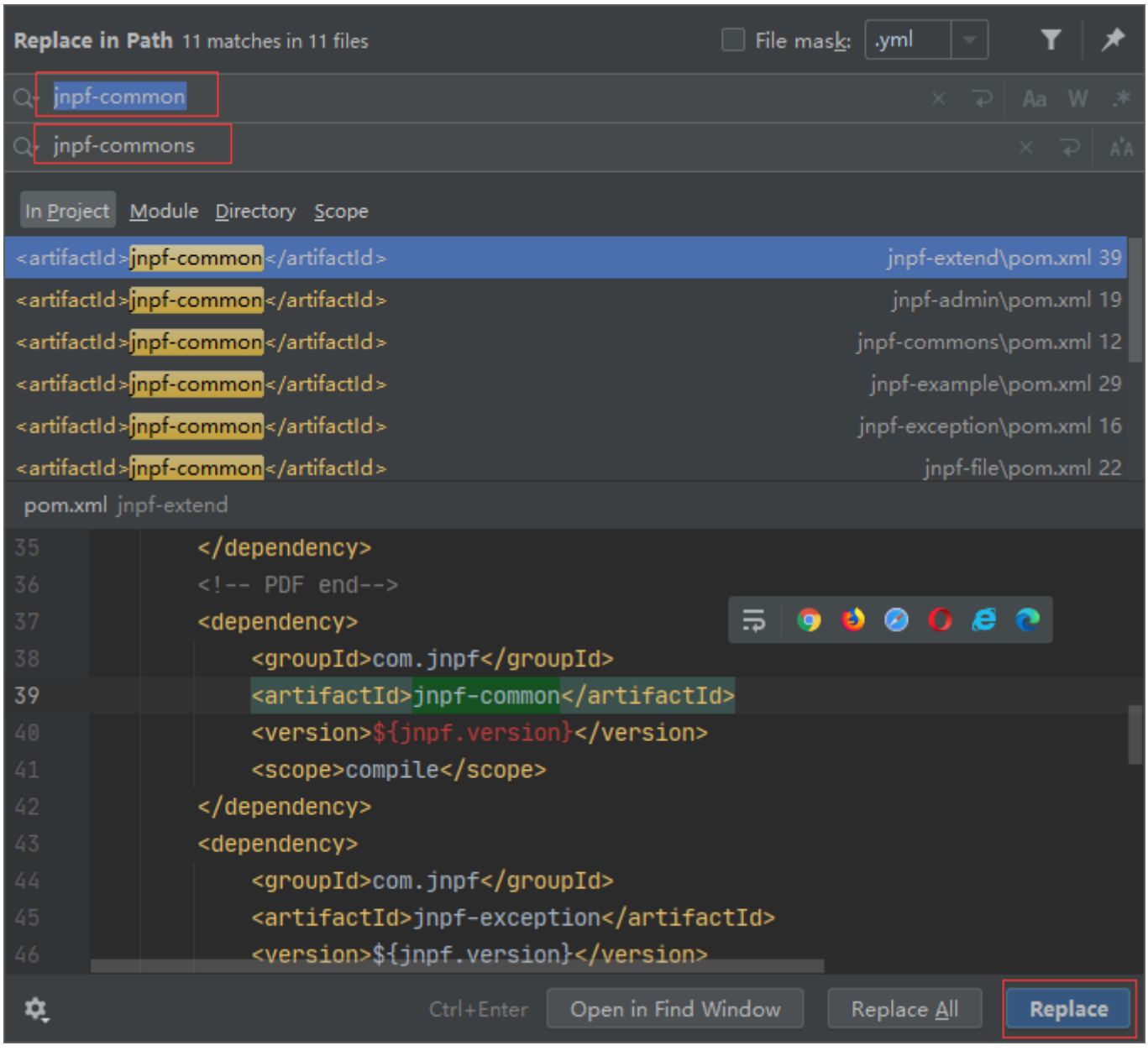
弹出会话框后选择第三个，点击Ok，如图所示：



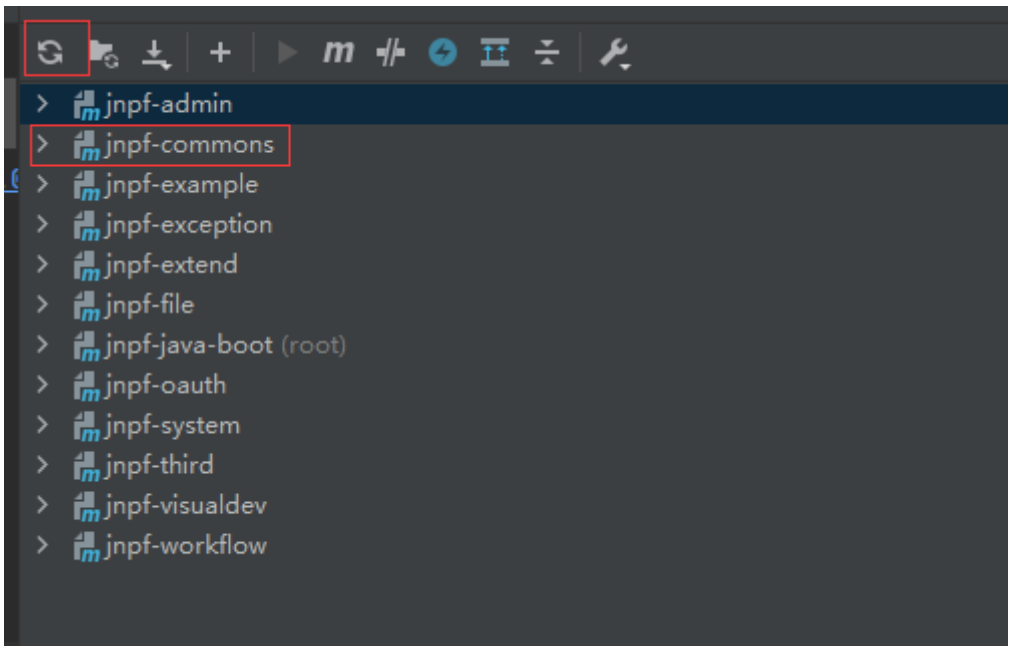
2.刷新maven，全局搜索被修改前的项目名，全局替换成新的maven名，如图所示：



建议点击replace依次替换，以免将不需要修改的改到！



刷新Maven，到此，Maven名称修改完成



东方通tong-web部署war包教程

xml排除Tomact

> `jnpf-java-boot` 项目中由于内嵌了Tomcat容器，所以我们需要将所有的Tomcat排除掉，大概有这些包需要排除：

1. `spring-boot-starter-websocket` 不是必须排除 `spring-boot-starter-web` ，而是必须排除其中的Tomcat包，一个spring-boot项目一般来说只有一个spring-boot-starter-web包，因此这里我直接排除了 `spring-boot-starter-web`

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-websocket</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-web</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
```

2. `tomcat-embed-core` 替换，版本可自行选择，只要兼容即可

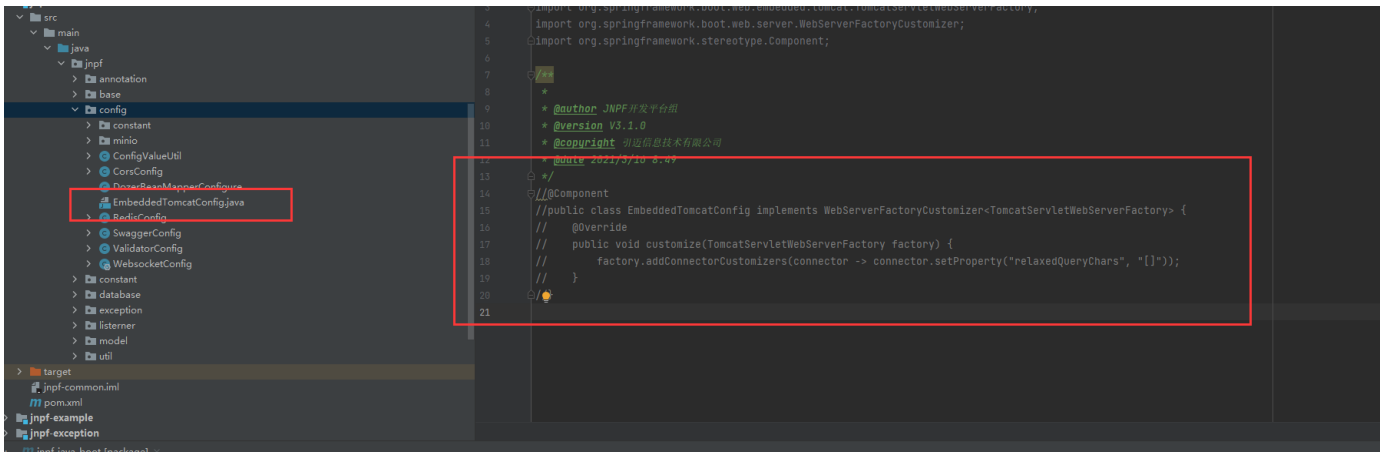
```
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.0-b05</version>
        </dependency>
        <dependency>
            <groupId>javax.websocket</groupId>
            <artifactId>javax.websocket-api</artifactId>
            <version>1.0</version>
        </dependency>
```

```
        <groupId>io.springfox</groupId>
        <artifactId>springfox-boot-starter</artifactId>
    </dependency>
<!--    <dependency>-->
<!--        <groupId>org.apache.tomcat.embed</groupId>-->
<!--        <artifactId>tomcat-embed-core</artifactId>-->
<!--    </dependency>-->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.0-b05</version>
    </dependency>
    <dependency>
        <groupId>javax.websocket</groupId>
        <artifactId>javax.websocket-api</artifactId>
        <version>1.0</version>
    </dependency>
    <dependency>
```

3.最重要的就是 `spring-boot-starter-web` 了，排除掉 `spring-boot-starter-tomcat`

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
```

注释掉 EmbeddedTomcatConfig 类：



4.排除所有携带Tomcat的包（上面仅仅为示例，可能还要需要自己排除）后，验证是否排除掉了所有的Tomcat依赖（注：不要导入任何容器），启动JnpfAdminApplication，报错如下：

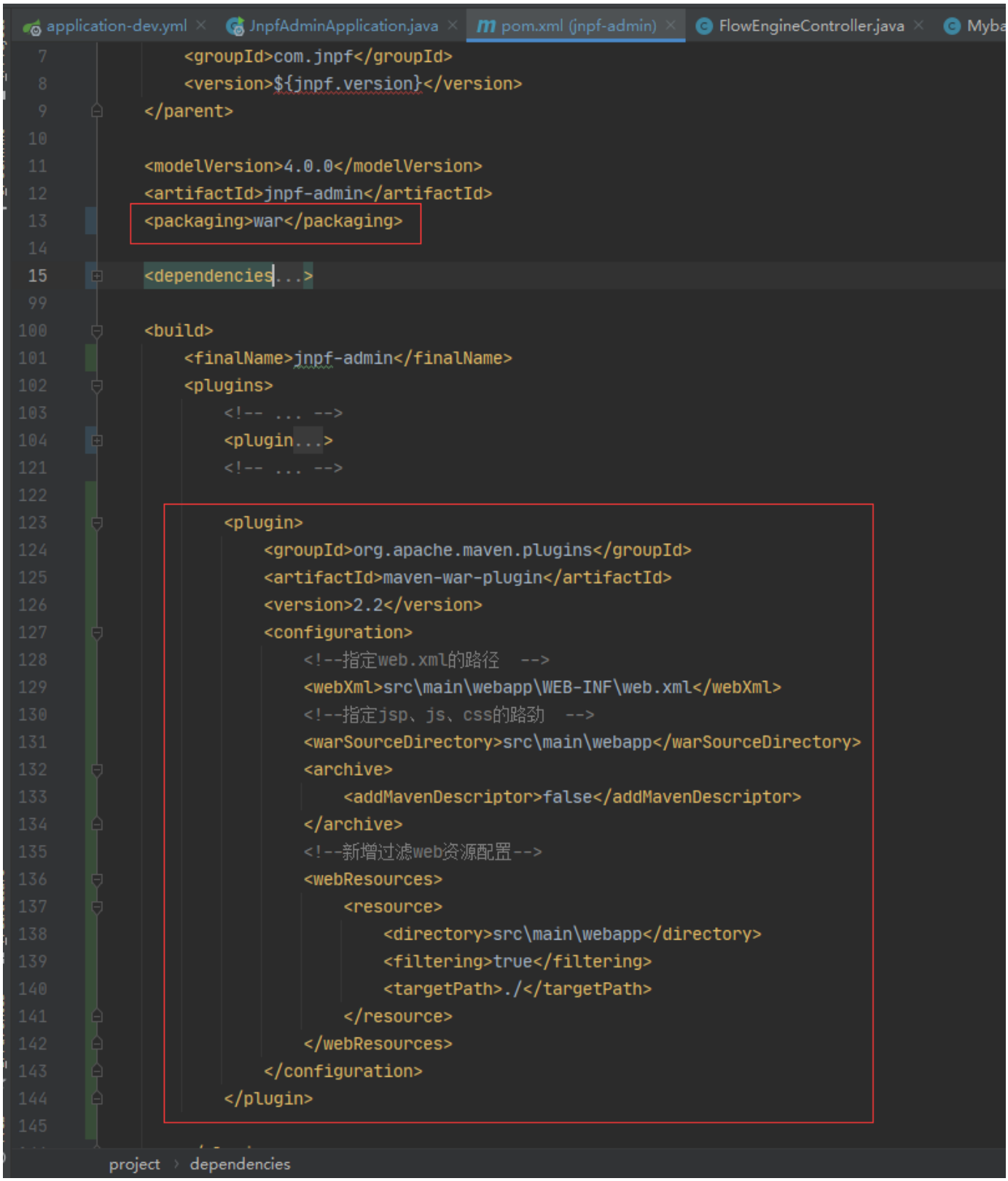
```
Error starting ApplicationContext. To display the conditions report re-run your appl
ication with 'debug' enabled.
2021-07-26 15:22:45.638 ERROR 3428 --- [           main] o.s.boot.SpringApplication
```

```
: Application run failed

org.springframework.context.ApplicationContextException: Unable to start web server;
nested exception is org.springframework.context.ApplicationContextException: Unable
 to start ServletWebServerApplicationContext due to missing ServletWebServerFactory
 bean.
        at org.springframework.boot.web.servlet.context.ServletWebServerApplicationC
ontext.onRefresh(ServletWebServerApplicationContext.java:161)
        at org.springframework.context.support.AbstractApplicationContext.refresh(Ab
stractApplicationContext.java:545)
        at org.springframework.boot.web.servlet.context.ServletWebServerApplicationC
ontext.refresh(ServletWebServerApplicationContext.java:143)
        at org.springframework.boot.SpringApplication.refresh(SpringApplication.jav
a:755)
        at org.springframework.boot.SpringApplication.refresh(SpringApplication.jav
a:747)
        at org.springframework.boot.SpringApplication.refreshContext(SpringApplicati
on.java:402)
        at org.springframework.boot.SpringApplication.run(SpringApplication.java:312
)
        at jnpf.JnpfAdminApplication.main(JnpfAdminApplication.java:26)
Caused by: org.springframework.context.ApplicationContextException: Unable to start
 ServletWebServerApplicationContext due to missing ServletWebServerFactory bean.
        at org.springframework.boot.web.servlet.context.ServletWebServerApplicationC
ontext.getWebServerFactory(ServletWebServerApplicationContext.java:205)
        at org.springframework.boot.web.servlet.context.ServletWebServerApplicationC
ontext.createWebServer(ServletWebServerApplicationContext.java:177)
        at org.springframework.boot.web.servlet.context.ServletWebServerApplicationC
ontext.onRefresh(ServletWebServerApplicationContext.java:158)
        ... 7 common frames omitted
```

这个错误的出现就说明了Tomcat容器被排除干净了

打包war

1.pom.xml文件调整（注：jnpf-admin中的pom），调整如下：

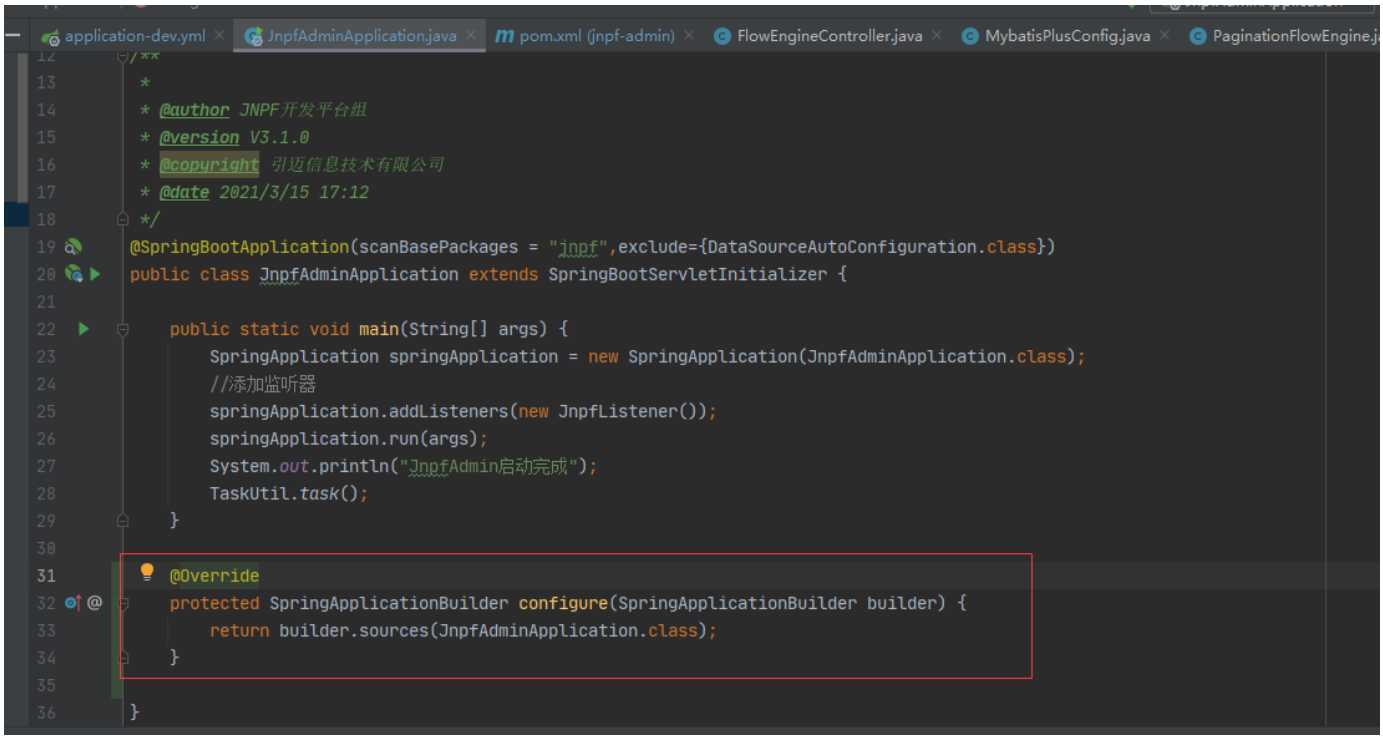


插件代码，版本可自行选择：

```
        <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.2</version>
                <configuration>
                    <!--指定web.xml的路径   -->
```

```
                <webXml>src\main\webapp\WEB-INF\web.xml</webXml>
                <!--指定jsp、js、css的路劲  -->
                <warSourceDirectory>src\main\webapp</warSourceDirectory>
                <archive>
                    <addMavenDescriptor>false</addMavenDescriptor>
                </archive>
                <!--新增过滤web资源配置-->
                <webResources>
                    <resource>
                        <directory>src\main\webapp</directory>
                        <filtering>true</filtering>
                        <targetPath>./</targetPath>
                    </resource>
                </webResources>
            </configuration>
        </plugin>
```

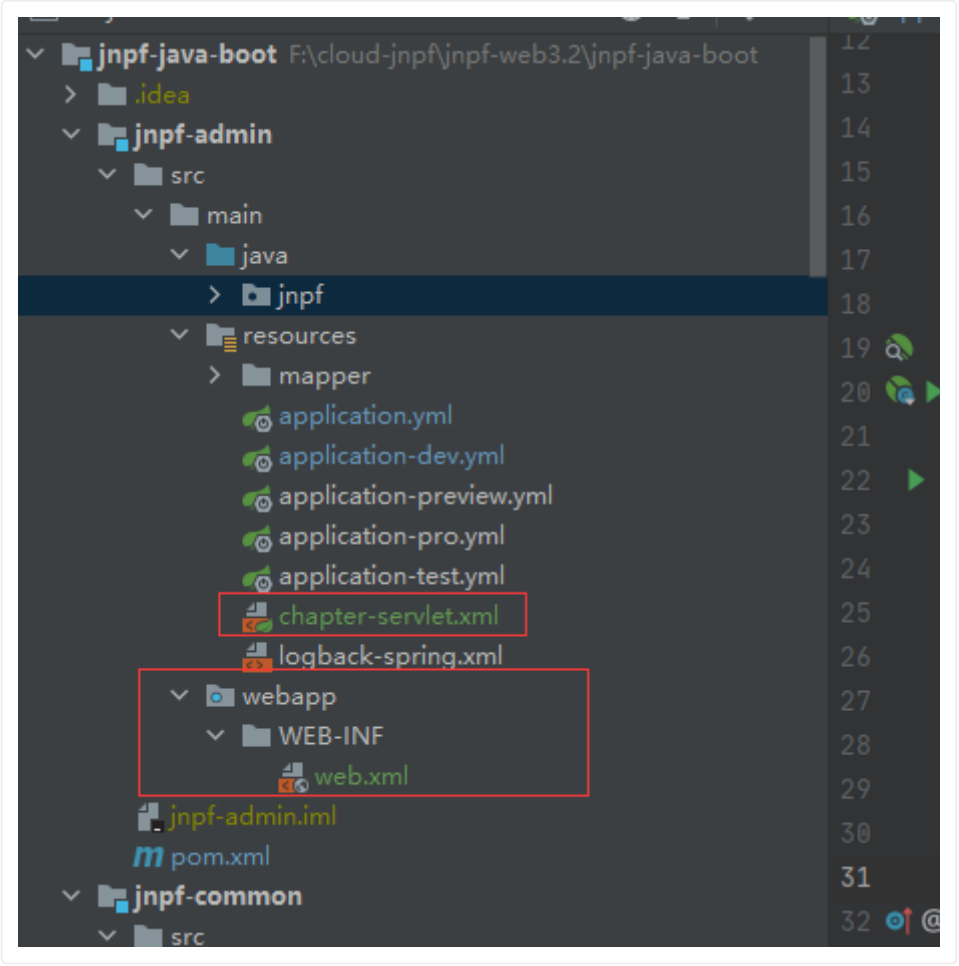
2.修改启动类，调整如下：



启动类添加代码：

```
@Override
protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
    return builder.sources(JnpfAdminApplication.class);
}
```

3.添加webapp，即web.xml，结构如下：



详细代码：

- web.xml

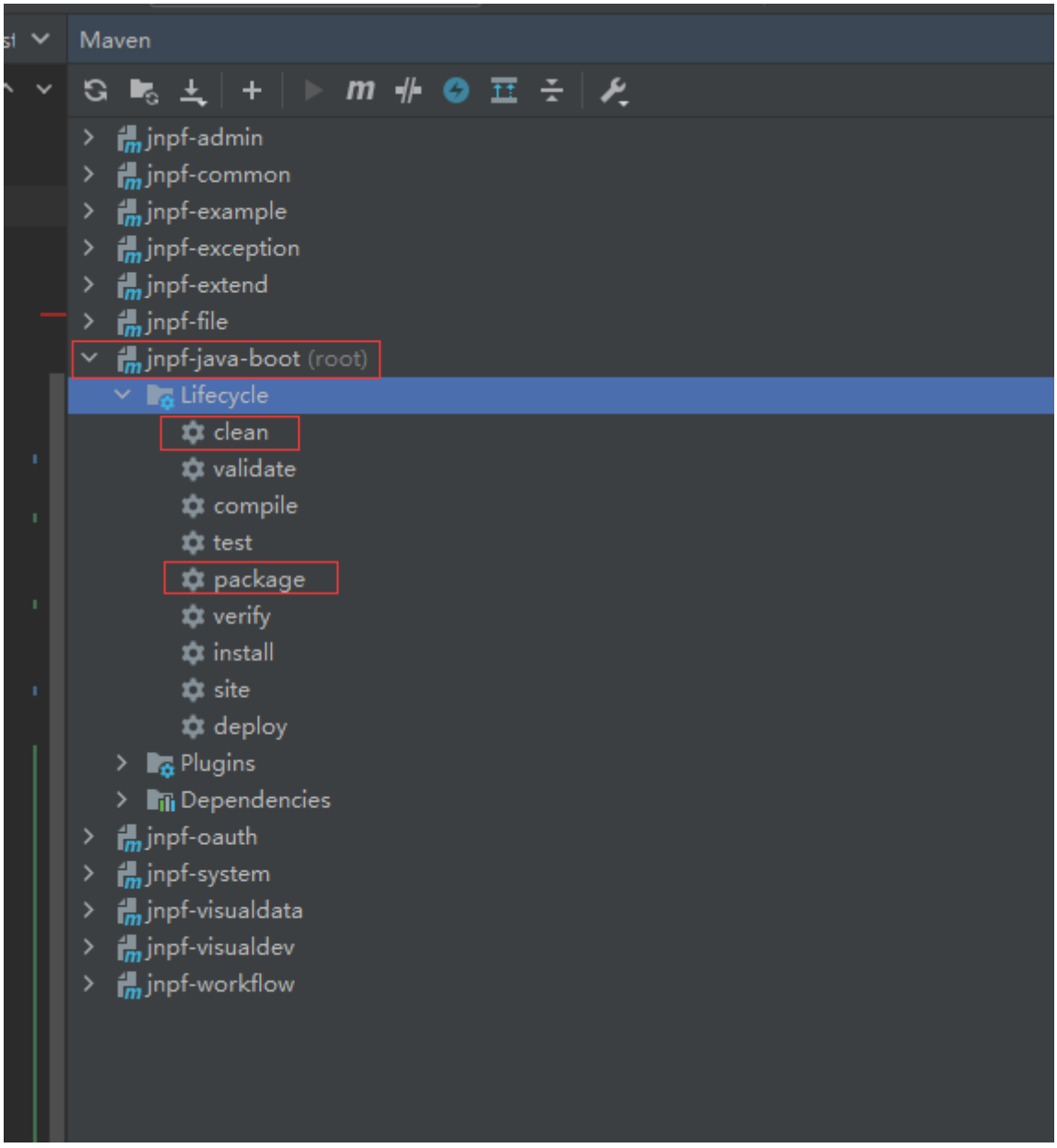
```
     <?xml version="1.0" encoding="UTF-8"?>
    <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jc
p.org/xml/ns/javaee/web-app_4_0.xsd"
             version="4.0">
        <servlet>
            <servlet-name>chapter</servlet-name>
            <servlet-class>org.springframework.web.servlet.DispatcherServlet</servle
t-class>
            <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>classpath*:chapter-servlet.xml</param-value>
            </init-param>
            <load-on-startup>1</load-on-startup>
        </servlet>

        <servlet-mapping>
            <servlet-name>chapter</servlet-name>
```

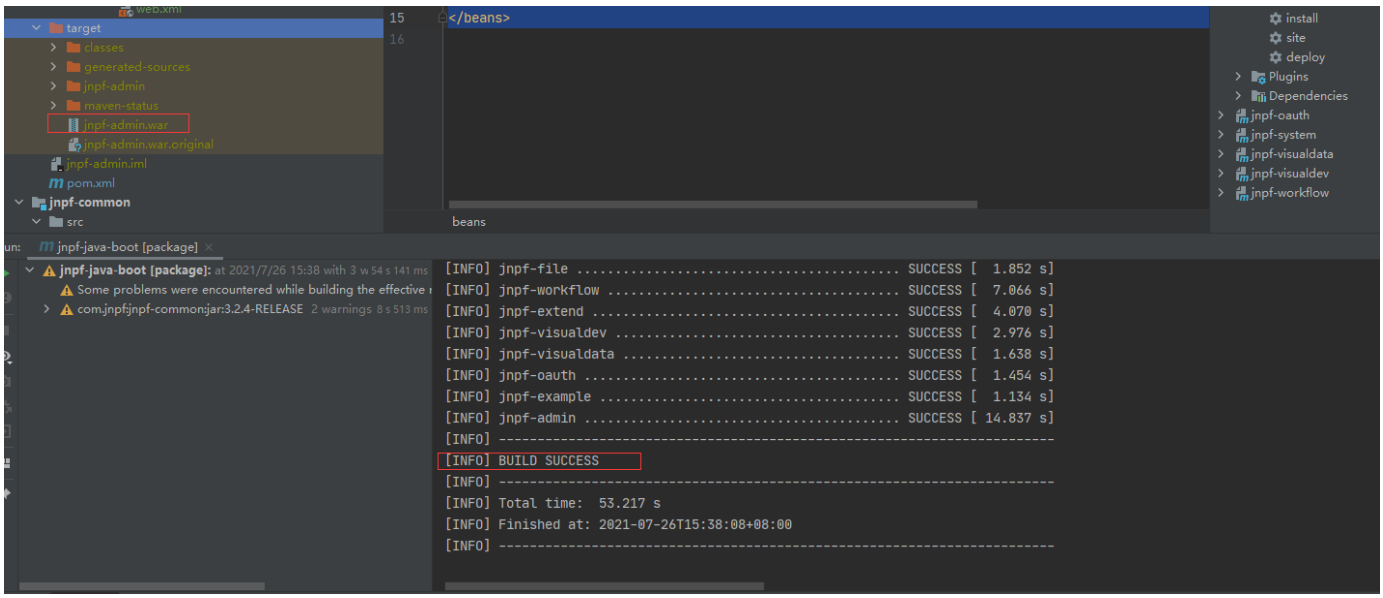
```
            <url-pattern>/</url-pattern>
        </servlet-mapping>

        <context-param>
            <param-name>log4jExposeWebAppRoot</param-name>
            <param-value>false</param-value>
        </context-param>
    </web-app>
```

- chapter-servlet.xml：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.sp
ringframework.org/schema/beans/spring-beans.xsd">

    <bean id="propertyPlaceholderConfigurer"
          class="org.springframework.beans.factory.config.PropertyPlaceholderConfigu
rer">
        <property name="locations">
            <list>
                <value>classpath:application.yml</value>
            </list>
        </property>
    </bean>

</beans>
```

4.使用Maven打包，点击clean、package



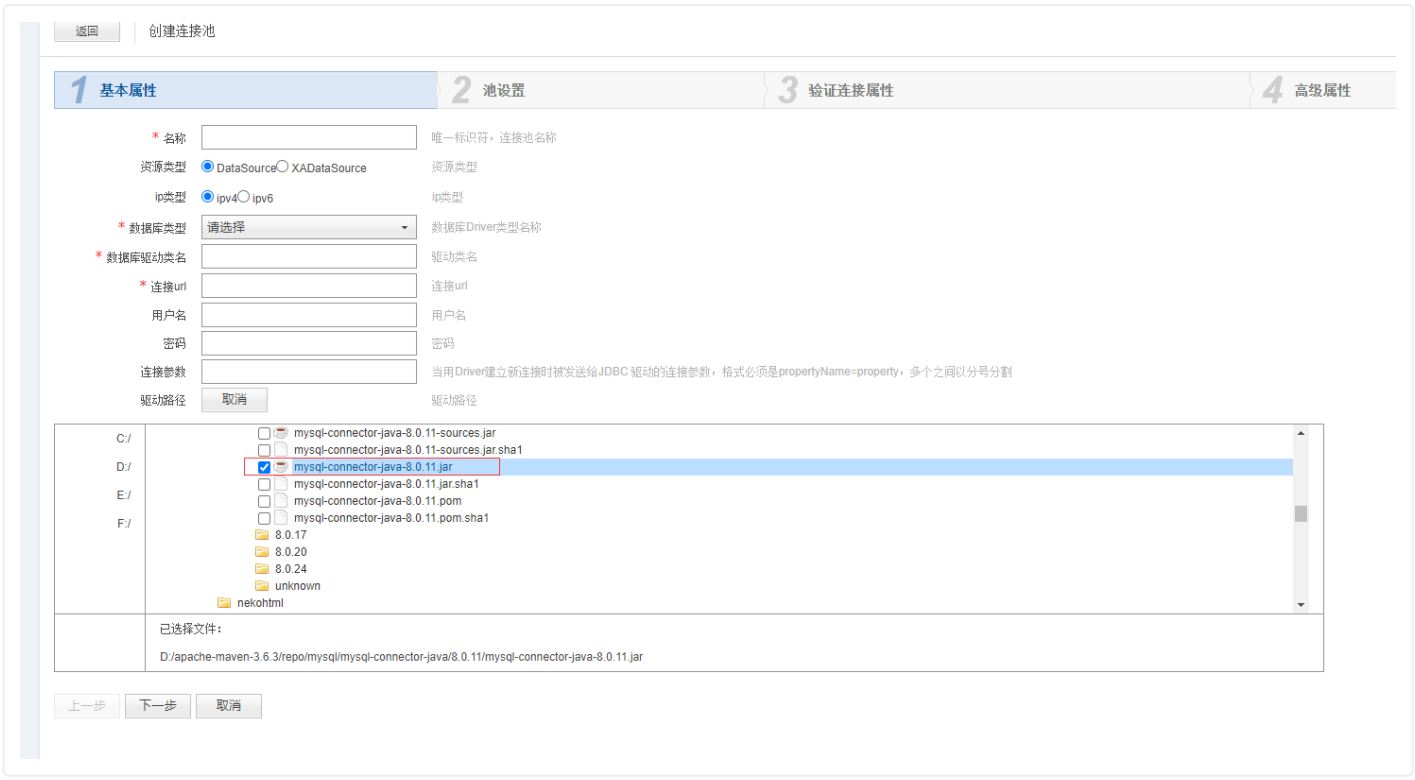


使用TongWeb部署jar

1.安装TongWeb并启动

2.登陆TongWeb

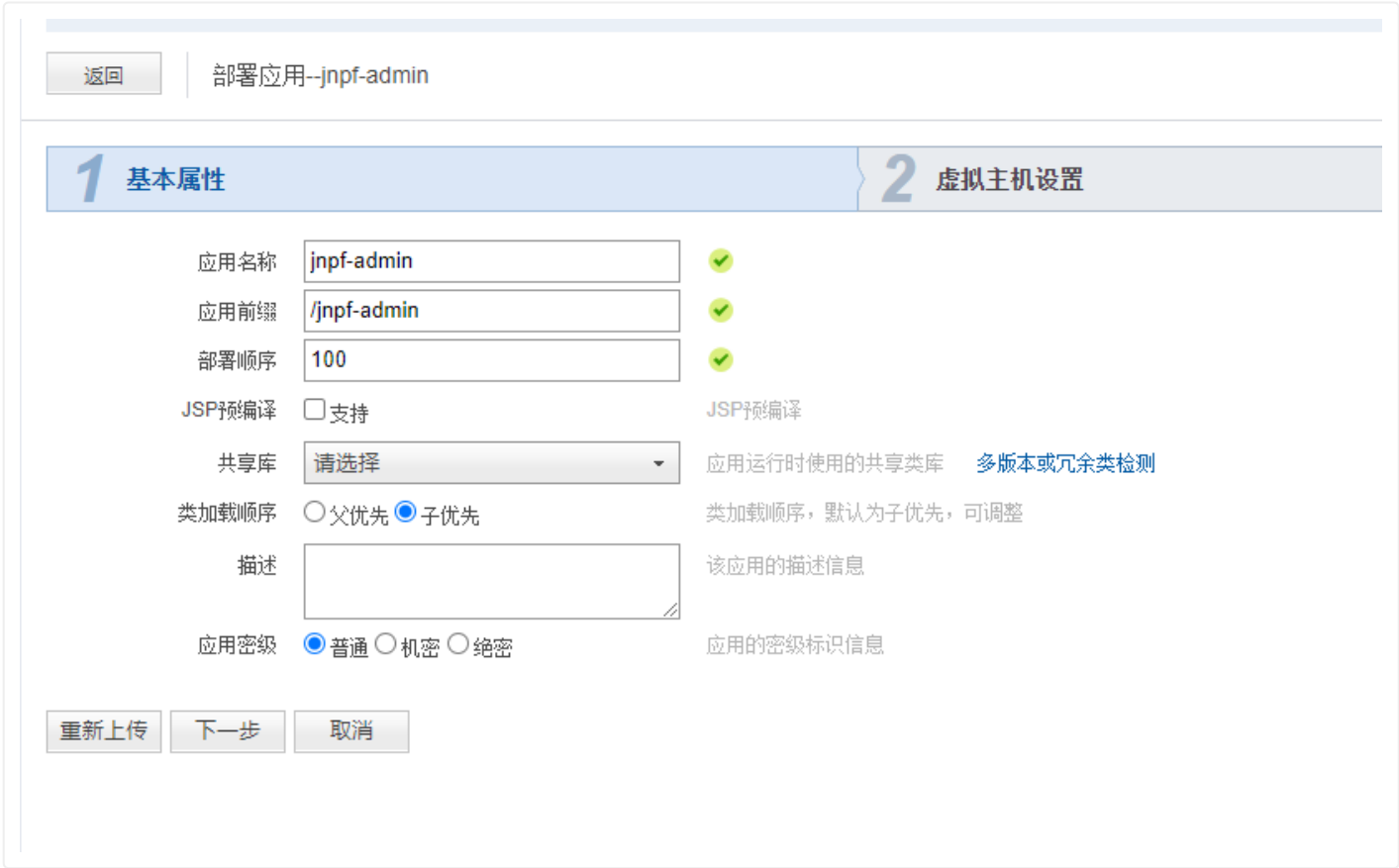
3.配置jdbc并测试连接

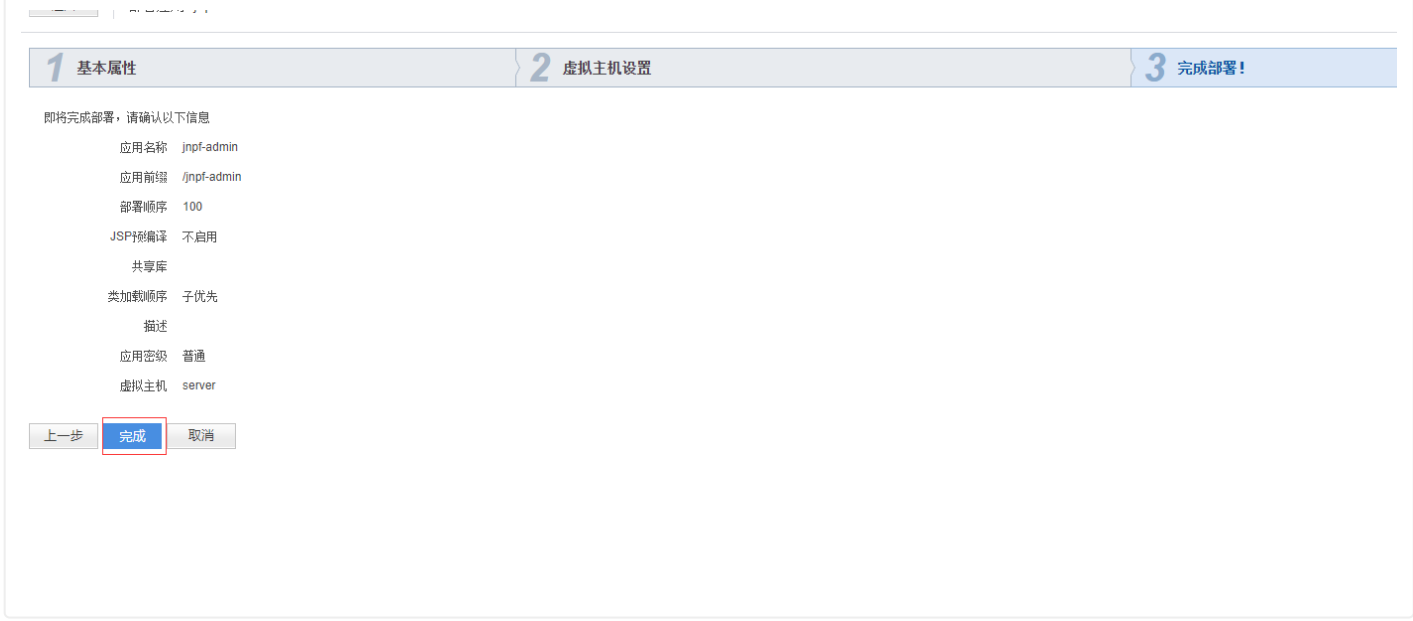


4.开始部署


首页
监视概览
启动参数配置
WEB容器配置
JDBC配置
工作管理器
JMS服务
EJB
应用管理
JCA
服务
安全服务
JNDI
JavaMail配置
应用管理
此页显示了已经部署到TongWeb的Java EE 应用程序和独立应用程序模块列表。可以对所安装的应用程序和模块进行启动、停止、更新（重新部署）、删除(解部署)操作或者部署新的应用或模块。
部署应用
启动
停止
解部署
名称
前缀
应用类型
部署源类型
部署方式
虚拟主机
状态
操作
jnpf-admin
/jnpf-admin
war
文件部署
控制台部署
server
已启动
重部署 http访问 https访问
genericra
rar
目录部署
控制台部署
server
已启动
重部署


返回
部署应用
文件位置
本机
服务器
选择上传文件位置
部署文件
选择文件
jnpf-admin.war
开始部署
取消


返回
部署应用--jnpf-admin
1 基本属性
2 虚拟主机设置
应用名称
jnpf-admin
应用前缀
/jnpf-admin
部署顺序
100
JSP预编译
支持
JSP预编译
共享库
请选择
应用运行时使用的共享类库
多版本或冗余类检测
类加载顺序
父优先
子优先
类加载顺序，默认为子优先，可调整
描述
该应用的描述信息
应用密级
普通
机密
绝密
应用的密级标识信息
重新上传
下一步
取消



访问接口成功则完成部署（我这里是开启了测试环境）

minIO部署

> 本文档在 CentOS 环境下部署minIO

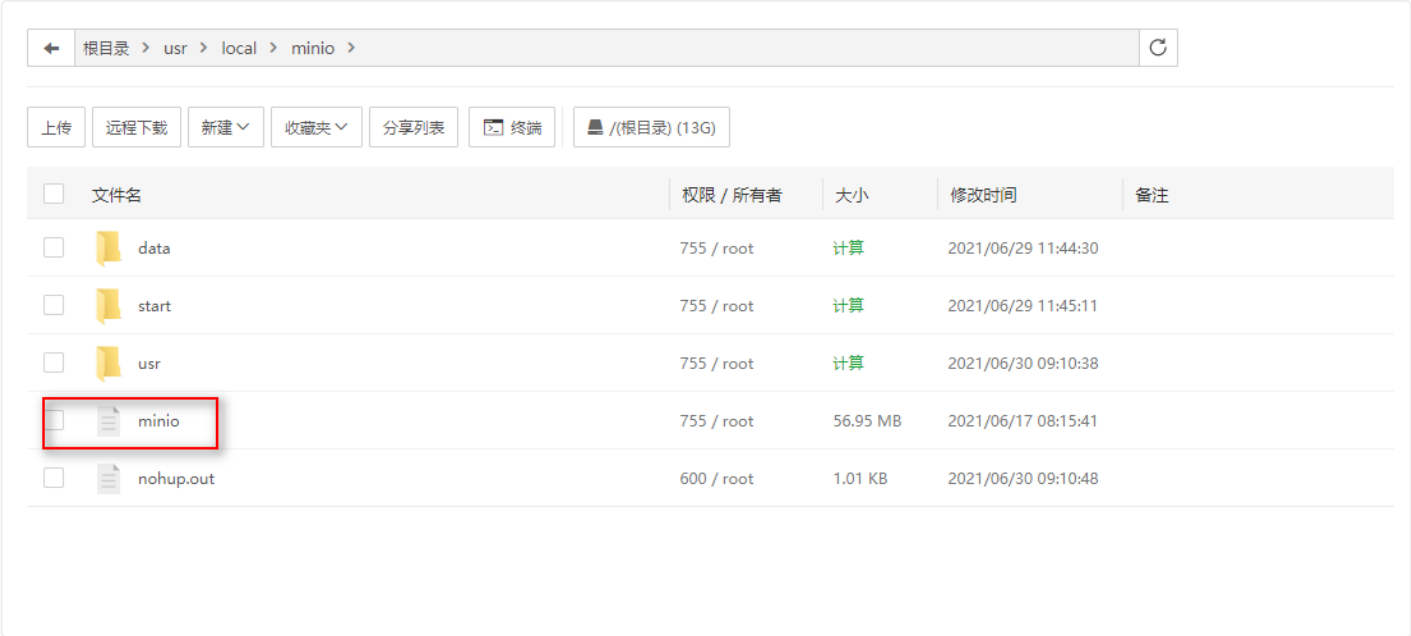
下载安装包

- 方式一：在 `Coding/项目/JNPF开发文档/文件网盘/minIO/` 目录下载安装包并上传至 `/usr/local/minio`
- 方式二：进入目录 `/usr/local/minio`，使用命令下载

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
```

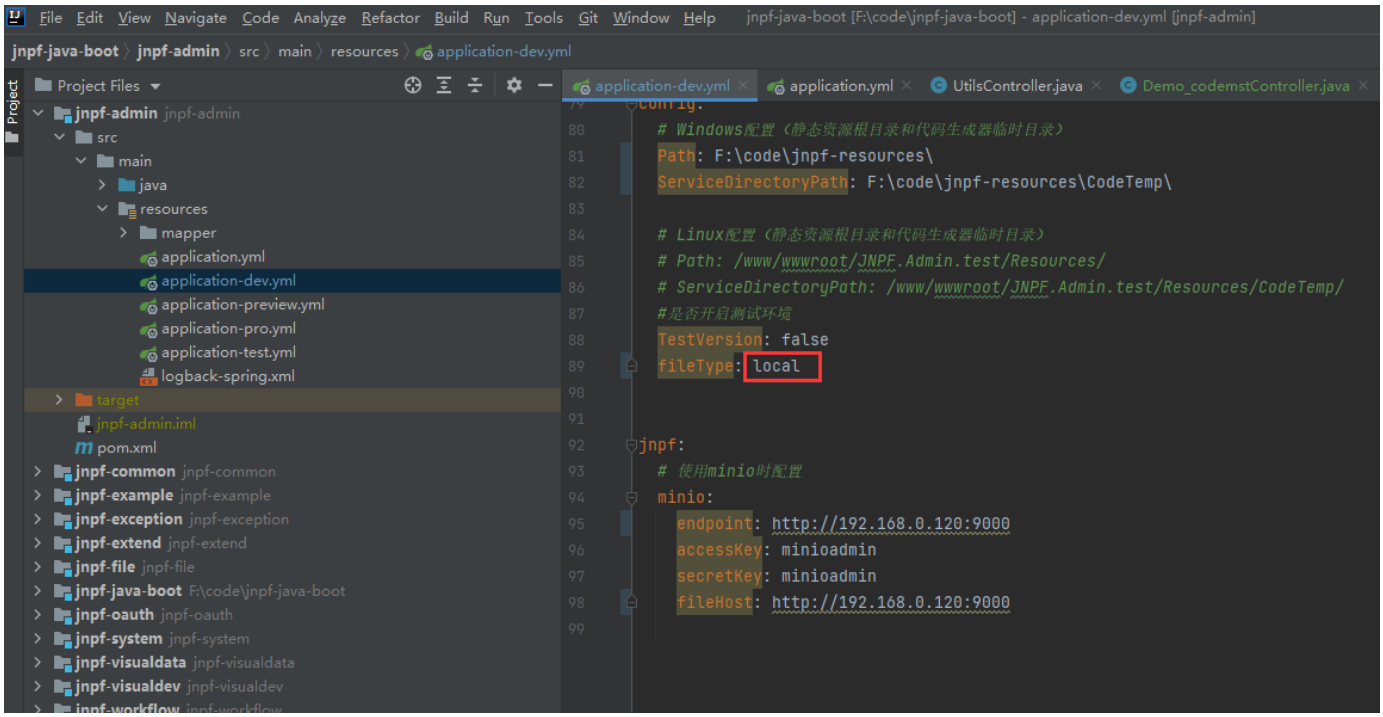
部署

```
chmod +x minio
mkdir -p /home/minio/data

# 开放minio端口:9000
firewall-cmd --zone=public --add-port=30000/tcp --permanent
# 重载防火墙
firewall-cmd --reload

# 方式一：启动minio服务
/usr/local/minio/minio server /home/minio/data/

# 方式二：后台运行minio
nohup /usr/local/minio/minio server /home/minio/data > /home/minio/data/minio.log 2>&1 &
```



官方文档地址： `https://docs.min.io/`

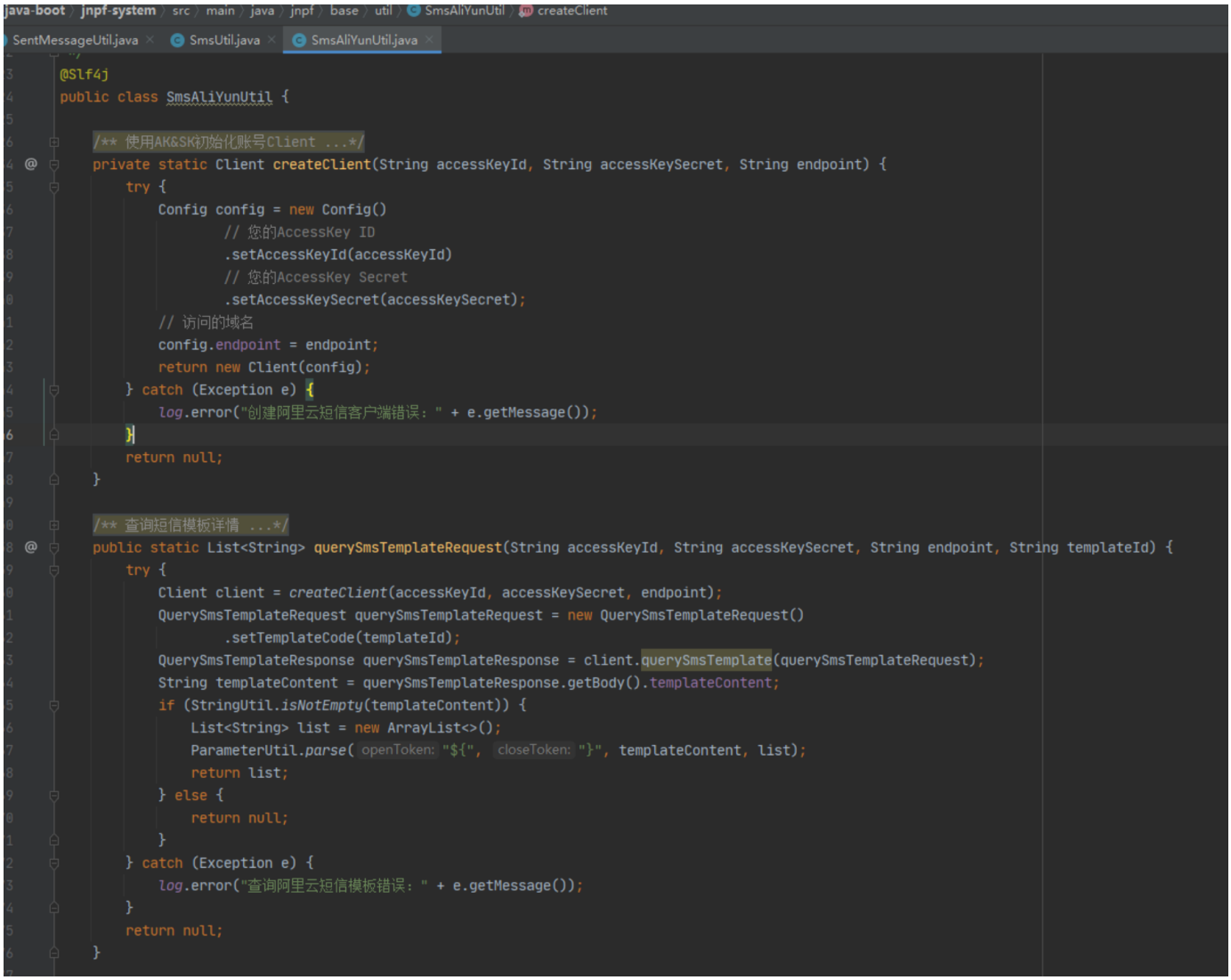
注：本地开发时不使用minio功能可修改 `jnpf-admin/src/main/resources/application-dev.yml` 的 `fileType: local`



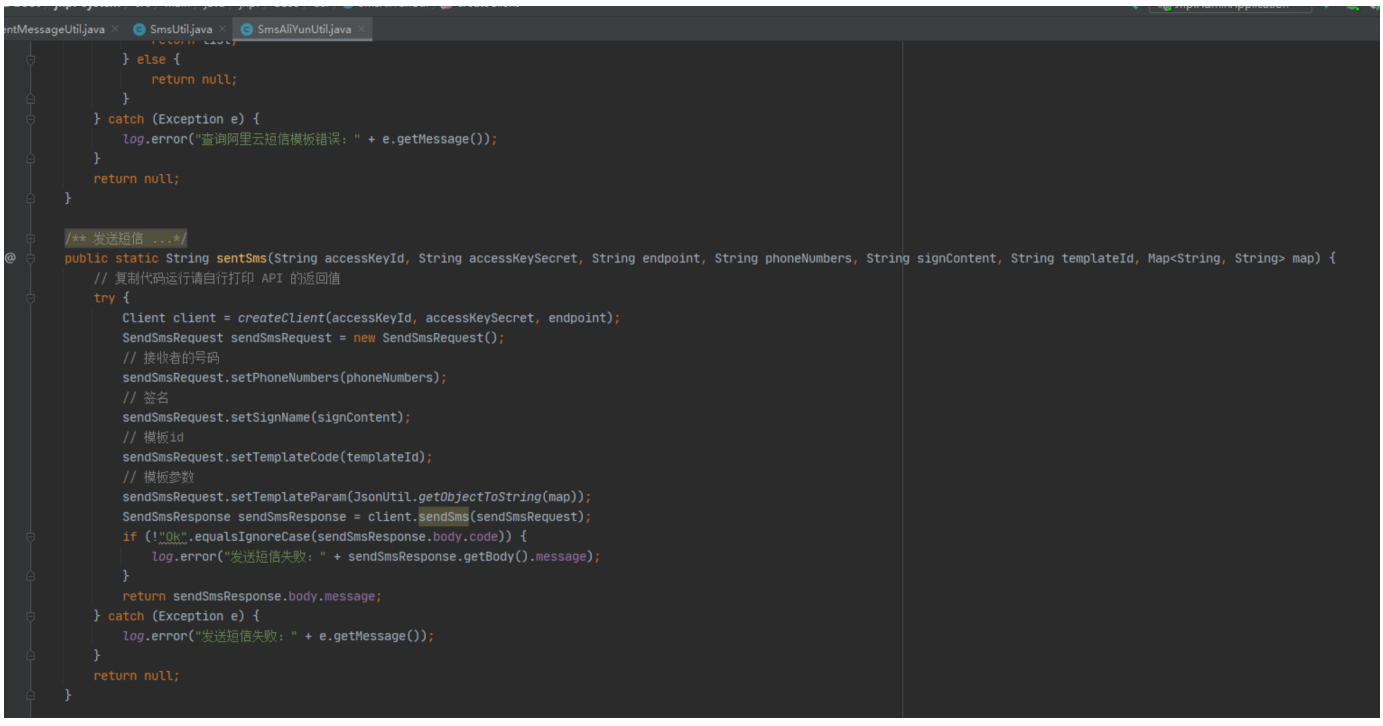
消息模块扩展教程

> 本文档以阿里云文档为例

添加相应的依赖后，编写对应的工具类，如SmsAliYunUtil，代码举例截图：



```
java-boot › jnpf-system › src › main › java › jnpf › base › util › SmsAliYunUtil › createClient
SentMessageUtil.java × SmsUtil.java × SmsAliYunUtil.java ×

@Slf4j
public class SmsAliYunUtil {

    /** 使用AK&SK初始化账号Client ...*/
    private static Client createClient(String accessKeyId, String accessKeySecret, String endpoint) {
        try {
            Config config = new Config()
                    // 您的AccessKey ID
                    .setAccessKeyId(accessKeyId)
                    // 您的AccessKey Secret
                    .setAccessKeySecret(accessKeySecret);
            // 访问的域名
            config.endpoint = endpoint;
            return new Client(config);
        } catch (Exception e) {
            log.error("创建阿里云短信客户端错误：" + e.getMessage());
        }
        return null;
    }

    /** 查询短信模板详情 ...*/
    public static List<String> querySmsTemplateRequest(String accessKeyId, String accessKeySecret, String endpoint, String templateId) {
        try {
            Client client = createClient(accessKeyId, accessKeySecret, endpoint);
            QuerySmsTemplateRequest querySmsTemplateRequest = new QuerySmsTemplateRequest()
                    .setTemplateCode(templateId);
            QuerySmsTemplateResponse querySmsTemplateResponse = client.querySmsTemplate(querySmsTemplateRequest);
            String templateContent = querySmsTemplateResponse.getBody().templateContent;
            if (StringUtil.isNotEmpty(templateContent)) {
                List<String> list = new ArrayList<>();
                ParameterUtil.parse( openToken: "${", closeToken: "}", templateContent, list);
                return list;
            } else {
                return null;
            }
        } catch (Exception e) {
            log.error("查询阿里云短信模板错误：" + e.getMessage());
        }
        return null;
    }
```



```
ntMessageUtil.java × SmsUtil.java × SmsAliYunUtil.java ×

            } else {
                return null;
            }
        } catch (Exception e) {
            log.error("查询阿里云短信模板错误：" + e.getMessage());
        }
        return null;
    }

    /** 发送短信 ...*/
    public static String sentSms(String accessKeyId, String accessKeySecret, String endpoint, String phoneNumbers, String signContent, String templateId, Map<String, String> map) {
        // 复制代码运行请自行打印 API 的返回值
        try {
            Client client = createClient(accessKeyId, accessKeySecret, endpoint);
            SendSmsRequest sendSmsRequest = new SendSmsRequest();
            // 接收者的号码
            sendSmsRequest.setPhoneNumbers(phoneNumbers);
            // 签名
            sendSmsRequest.setSignName(signContent);
            // 模板id
            sendSmsRequest.setTemplateCode(templateId);
            // 模板参数
            sendSmsRequest.setTemplateParam(JsonUtil.getObjectToString(map));
            SendSmsResponse sendSmsResponse = client.sendSms(sendSmsRequest);
            if (!"Ok".equalsIgnoreCase(sendSmsResponse.body.code)) {
                log.error("发送短信失败：" + sendSmsResponse.getBody().message);
            }
            return sendSmsResponse.body.message;
        } catch (Exception e) {
            log.error("发送短信失败：" + e.getMessage());
        }
        return null;
    }
```

如何调用，可根据调用的范围将工具类放到对应的模块下，然后使用类名.方法(参数…)直接调用即可，如：

```java
/**
 *
 * @param type
 * @param accessKeyId
 * @param accessKeySecret
 * @param templateId
 * @return
 */
public static List<String> querySmsTemplateRequest(Integer type, String accessKeyId, String accessKeySecret, String endpoint, String region, String templateId) {
    if (type == 1) {
        return SmsAliYunUtil.querySmsTemplateRequest(accessKeyId, accessKeySecret, endpoint, templateId);
    }
    return SmsTenCentCloudUtil.querySmsTemplateRequest(accessKeyId, accessKeySecret, endpoint , region, templateId);
}

/**
 * 发送消息
 *
 * @param type
 * @param accessKeyId
 * @param accessKeySecret
 * @param phoneNumbers
 * @param appId           腾讯消息使用
 * @param signContent
 * @param templateId
 * @param map
 * @return
 */
public static String sentSms(Integer type, String accessKeyId, String accessKeySecret, String endpoint, String region, String phoneNumbers, String appId, String signContent, String templateId, Map<String, String> map) {
    if (type == 1) {
        return SmsAliYunUtil.sentSms(accessKeyId, accessKeySecret, endpoint, phoneNumbers, signContent, templateId, map);
    }
    return SmsTenCentCloudUtil.sentSms(accessKeyId, accessKeySecret, endpoint, region, phoneNumbers, appId, signContent, templateId, map);
}

}
```

由于也兼容了腾讯云短信，因此将具体的发送短信在提取到了SmsUtil.class，使用也是类名.方法(参数…)直接调用，如：

```java
public ActionResult<?> getTemplateById(@PathVariable("id") String id) {
    // 定义返回对象
    List<String> list = new ArrayList<>();
    SmsTemplateEntity entity = smsTemplateService.getInfo(id);
    if (entity != null && entity.getCompany() != null) {
        // 得到系统配置
        SmsModel smsModel = smsTemplateService.getSmsConfig();
        list = SmsUtil.querySmsTemplateRequest(entity.getCompany(), smsModel.getSmsKeyId(), smsModel.getSmsKeySecret(), smsModel.getDomain()
    }
    if (list == null) {
        return ActionResult.success(new ArrayList<>());
    }
    return ActionResult.success(list);
}

@ApiOperation("发送测试短信")
@PostMapping("/testSent")
public ActionResult testSentSms(@RequestBody SmsTemplateCrForm smsTemplateCrForm) {
    if (smsTemplateCrForm.getCompany() != null) {
        // 得到短信模型
        SmsModel smsModel = smsTemplateService.getSmsConfig();
        // 如果是阿里的话
        String sentCode = SmsUtil.sentSms(smsTemplateCrForm.getCompany(), smsModel.getSmsKeyId(),
                smsModel.getSmsKeySecret(), smsModel.getDomain(), smsModel.getRegion(), smsTemplateCrForm.getPhoneNumbers(),
                smsTemplateCrForm.getAppId(), smsTemplateCrForm.getSignContent(),
                smsTemplateCrForm.getTemplateId(), smsTemplateCrForm.getParameters());
        if ("OK".equalsIgnoreCase(sentCode)) {
            return ActionResult.success("验证通过");
        }
    }
    return ActionResult.fail("验证失败");
}
```

java-cloud

快速开始

代码结构

> 本文档适用 `jnpf-java-cloud` 3.4.3版本

基础依赖

> 源码项目：jnpf-common

```
jnpf-common
     ├─── jnpf-common-auth - 认证模块
     ├─── jnpf-common-core - 基础类及常用工具
     ├─── jnpf-common-database - 数据库配置及多数据库兼容
     ├─── jnpf-common-dubbo - dubbo拦截器, 自动封装认证信息
     ├─── jnpf-common-feign - 远程调用Feign组件配置
     ├─── jnpf-common-redis - 缓存工具Redis组件配置
     ├─── jnpf-common-security - 接口鉴权配置
     ├─── jnpf-common-swagger - API组件Swagger配置
     └─── jnpf-common-utils - 工具包
     ├          ├─── jnpf-common-connector - 单点数据推送模块
     ├          ├─── jnpf-common-file - 文件工具类模块
     ├          ├─── jnpf-common-sms - 短信模块
     ├          ├─── jnpf-common-office - office操作模块
     ├          ├─── jnpf-common-scheduletask - 调度工具模块
     ├          └─── jnpf-common-seata - seata依赖
```

任务调度客户端依赖及服务端

> 源码项目：jnpf-scheduletask

```
jnpf-scheduletask
     ├─── jnpf-scheduletask-client - xxl-job客户端需要加入此模块
     ├─── jnpf-scheduletask-model - 基础模型
     ├─── xxl-job-core - xxl-job核心包
     ├─── xxl-job-admin - xxl-job服务端及启动类
```

微服务

> 源码项目：jnpf-java-cloud

```
jnpf-java-cloud
     ├─── dockerfile -- Dockerfile文件
     ├─── jnpf-app -- 移动端服务 [30012]
     ├     ├─── jnpf-app-api - 对外(服务)暴露接口
     ├     ├─── jnpf-app-biz - 业务逻辑层
     ├     ├─── jnpf-app-controller - 控制层
     ├     ├─── jnpf-app-entity - 模型、实体类
     ├     └─── jnpf-app-server - 当前服务启动类及服务配置
     ├─── jnpf-common -- 公共模块
     ├     ├─── jnpf-common-springaop-- aop通用封装
     ├─── jnpf-datareport -- 报表服务 [30007]
     ├─── jnpf-example -- 服务示例模板(非功能)  [30100]
     ├─── jnpf-extend -- 扩展应用服务  [30019]
     ├     ├─── jnpf-extend-api - 对外(服务)暴露接口
     ├     ├─── jnpf-extend-biz - 业务逻辑层
     ├     ├─── jnpf-extend-controller - 控制层
     ├     ├─── jnpf-extend-entity - 模型、实体类
     ├     └─── jnpf-extend-server - 当前服务启动类及服务配置
     ├─── jnpf-file -- 文件服务  [30005]
     ├     ├─── jnpf-file-api - 对外(服务)暴露接口
     ├     ├─── jnpf-file-core - 文件服务核心代码
     ├     ├─── jnpf-file-dubboservice - 文件调用dubbo实现
     ├     ├─── jnpf-file-model - 模型、实体类
     ├     ├─── jnpf-file-server - 当前服务启动类及服务配置
     ├     ├─── jnpf-file-storage - 文件服务存储方案
     ├     └─── jnpf-file-yozo - 整合永中文档预览
     ├     └─── jnpf-file-upload - 文件上传
     ├─── jnpf-gateway-- 网关服务  [30000]
     ├─── jnpf-message -- 消息中心服务  [30008]
     ├     ├─── jnpf-message-api - 对外(服务)暴露接口
     ├     ├─── jnpf-message-biz - 业务逻辑层
     ├     ├─── jnpf-message-controller - 控制层
     ├     ├─── jnpf-message-entity - 模型、实体类
     ├     └─── jnpf-message-server - 当前服务启动类及服务配置
     ├─── jnpf-oauth-- 授权服务  [30001]
     ├     ├─── jnpf-oauth-api - 对外(服务)暴露接口
     ├     ├─── jnpf-oauth-biz - 业务逻辑层
     ├     ├─── jnpf-oauth-controller - 控制层
     ├     ├─── jnpf-oauth-entity - 模型、实体类
     ├     └─── jnpf-oauth-server - 当前服务启动类及服务配置
     ├─── jnpf-permission-- 通用用户权限管理 [30010]
     ├     ├─── jnpf-permission-api - 对外(服务)暴露接口
     ├     ├─── jnpf-permission-biz - 业务逻辑层
     ├     ├─── jnpf-permission-controller - 控制层
     ├     ├─── jnpf-permission-entity - 模型、实体类
```

```
├    └──── jnpf-permission-server - 当前服务启动类及服务配置
├──── jnpf-provider -- Dubbo提供者
├    ├──── jnpf-provider-example  - example服务调用示例
├    ├──── jnpf-provider-file - 文件服务调用
├    ├──── jnpf-provider-permission - 权限服务调用示例
├    ├──── jnpf-provider-system - 系统服务调用示例
├    └──── jnpf-provider-workflow - 工作流程服务调用示例
├──── jnpf-scheduletask -- 任务调度服务(客户端)  [30009]
├    ├──── jnpf-scheduletask-controller - 控制层
├    ├──── jnpf-scheduletask-server - 当前服务启动类及服务配置
├    └──── jnpf-scheduletask-service - 任务调度客户端实现
├──── jnpf-system -- 系统基础服务  [30002]
├    ├──── jnpf-system-api - 对外(服务)暴露接口
├    ├──── jnpf-system-biz - 业务逻辑层
├    ├──── jnpf-system-controller - 控制层
├    ├──── jnpf-system-dubboservice - 系统调用dubbo实现
├    ├──── jnpf-system-entity - 模型、实体类
├    └──── jnpf-system-server - 当前服务启动类及服务配置
├──── jnpf-tenant -- 多租户服务  [30006]
├    ├──── jnpf-tenant-biz - 业务逻辑层
├    ├──── jnpf-tenant-controller - 控制层
├    ├──── jnpf-tenant-entity - 模型、实体类
├    └──── jnpf-tenant-server - 当前服务启动类及服务配置
├──── jnpf-visualdata -- 大屏服务  [30011]
├    ├──── jnpf-visualdata-api - 对外(服务)暴露接口
├    ├──── jnpf-visualdata-biz - 业务逻辑层
├    ├──── jnpf-visualdata-controller - 控制层
├    ├──── jnpf-visualdata-entity - 模型、实体类
├    └──── jnpf-visualdata-server - 当前服务启动类及服务配置
├──── jnpf-visualdev-- 可视化开发服务  [30003]
├    ├──── jnpf-visualdev-base -- 基础模块
├    ├──────    ├──── jnpf-visualdev-base-api - 对外(服务)暴露接口
├    ├──────    ├──── jnpf-visualdev-base-biz - 业务逻辑层
├    ├──────    ├──── jnpf-visualdev-base-controller - 控制层
├    ├──────    └──── jnpf-visualdev-base-entity - 模型、实体类
├    ├──── jnpf-visualdev-generater -- 代码生成模块
├    ├──────    ├──── jnpf-visualdev-generater-api - 对外(服务)暴露接口
├    ├──────    ├──── jnpf-visualdev-generater-biz - 业务逻辑层
├    ├──────    ├──── jnpf-visualdev-generater-controller - 控制层
├    ├──────    └──── jnpf-visualdev-generater-entity - 模型、实体类
├    ├──── jnpf-visualdev-onlinedev -- 在线开发模块
├    ├──────    ├──── jnpf-visualdev-onlinedev-api - 对外(服务)暴露接口
├    ├──────    ├──── jnpf-visualdev-onlinedev-biz - 业务逻辑层
├    ├──────    ├──── jnpf-visualdev-onlinedev-controller - 控制层
├    ├──────    └──── jnpf-visualdev-onlinedev-entity - 模型、实体类
```

```
├    ├──── jnpf-visualdev-portal -- 门户设计模块
├    ├────     ├──── jnpf-visualdev-portal-api - 对外(服务)暴露接口
├    ├────     ├──── jnpf-visualdev-portal-biz - 业务逻辑层
├    ├────     ├──── jnpf-visualdev-portal-controller - 控制层
├    ├────     └──── jnpf-visualdev-portal-entity - 模型、实体类
├    └──── jnpf-visualdev-server -- 当前服务启动类及服务配置
└──── jnpf-workflow -- 工作流程服务  [30004]
├    ├──── jnpf-workflow-engine - 流程引擎
├    ├────     ├──── jnpf-workflow-engine-api - 对外(服务)暴露接口
├    ├────     ├──── jnpf-workflow-engine-biz - 业务逻辑层
├    ├────     ├──── jnpf-workflow-engine-controller - 控制层
├    ├────     └──── jnpf-workflow-engine-entity - 模型、实体类
├    ├──── jnpf-workflow-form - 流程表单，用于存放代码生成流程表单的源代码
├    ├────     ├──── jnpf-workflow-form-biz - 业务逻辑层
├    ├────     ├──── jnpf-workflow-form-controller - 控制层
├    ├────     └──── jnpf-workflow-form-entity - 模型、实体类
├    └──── jnpf-workflow-server - 当前服务启动类及服务配置
```

依赖说明

> 本文档适用 `jnpf-java-cloud` 3.4.3版本

artifactId	版本号	说明
maven-deploy-plugin	2.8.2	私服发布插件
javax.servlet-api	4.0.1	Java Servlet工具
commons-collections4	4.4	java反序列化工具
spring-cloud-starter-openfeign	3.1.1	不同服务之间的调用
lock4j-redis-template-spring-boot-starter	2.2.2	以Redis为基础实现分布式锁
spring-cloud-starter-oauth2	2.2.5.RELEASE	权限及Oauth认证
alibaba-dingtalk-service-sdk	1.0.1	阿里云短信支持
dysmsapi20170525	2.0.8	
tencentcloud-sdk-java	3.1.278	腾讯短信支持
okhttp	4.8.1	网络通信库
commons-compress	1.21	压缩解压工具包

spring-cloud-starter-alibaba-seata	2.1.0.RELEASE	分布式事务
aliyun-sdk-oss	3.14.1	阿里云存储
qiniu-java-sdk	7.10.2	七牛云存储
cos_api	5.6.79	腾讯云存储
spring-cloud-starter-alibaba-nacos-config	2021.0.1.0	分布式配置中心
spring-cloud-starter-alibaba-nacos-discovery	2021.0.1.0	分布式注册中心
spring-cloud-starter-alibaba-sentinel	2021.0.1.0	限流、熔断等控制
sentinel-datasource-nacos	1.8.3	Sentinel整合Nacos
spring-cloud-starter-gateway	3.1.1	微服务网关（转发）
spring-cloud-alibaba-sentinel-gateway	2021.0.1.0	Sentinel整合网关
swagger-bootstrap-ui	1.9.6	Swagger页面
javax.mail	1.6.2	Java有线API
spring-cloud-starter-stream-rocketmq	2021.0.1.0	SpringCloudStream、RocketMq整合包
dubbo-spring-boot-starter	2.7.15	Dubbo RPC调用
sentinel-apache-dubbo-adapter	1.8.4	Dubbo整合Sentinel
spring-boot-maven-plugin	2.7.0	SpringBoot打包插件
logback-classic	1.2.11	日志记录
easypoi-base	4.1.2	excel导出导出
hutool-all	5.8.0	工具类整合包
druid	1.2.9	数据库连接池
fastjson	1.2.83	Json工具类
yitter-idgenerator	1.0.6	生成雪花id工具类
lombok	1.18.24	生成构造方法、Get、Set等
jackson-annotations	2.13.3	Json工具类

spring-boot-starter-data-redis	2.7.0	Redis整合springboot
spring-boot-configuration-processor	2.7.0	配置文件提示
spring-boot-starter-websocket	2.7.0	Websocket、springboot整合包
commons-pool2	2.11.1	对象池工具
springfox-boot-starter	3.0.0	Swagger整合包
tomcat-embed-core	9.0.63	Tomcat容器
mybatis-plus-boot-starter	3.5.1	Mybatis、springboot整合包
dynamic-datasource-spring-boot-starter	3.5.1	动态切换数据源
hibernate-validator	6.2.3.Final	字段验证
commons-codec	1.15	编码包
quartz	2.3.2	任务调用
poi	4.1.2	导入导出
commons-lang3	3.12.0	工具包
httpclient	4.5.13	HTTP工具
pinyin4j	2.5.1	语言包
Core(com.google.zxing)	3.5.0	二维码生成工具
nimbus-jose-jwt	9.22	Jwt工具类
mysql-connector-java	8.0.29	MySQL驱动
sqljdbc4	4.0	SqlServer驱动
ojdbc8	21.5.0.0	Oracle驱动
orai18n	21.5.0.0	Oracle驱动
DmJdbcDriver18	1.8.0	达梦驱动
kingbase8-jdbc	2.0	人大金仓驱动
postgresql	42.3.5	Postgre驱动
minio	8.4.0	Minio文件存储
commons-fileupload	1.4	文件上传工具

antisamy	1.6.4	处理XSS注入
pagehelper	5.3.0	Mybatis分页插件
okhttp	4.9.3	http工具类
dom4j	2.1.3	DOM,SAX和JAXP解析工具
spring-boot-starter-web	2.7.0	Springboot、Tomcat整合包
itextpdf	5.5.13.3	处理PDF文件
itext-asian	5.2.0	
spring-boot-starter-mail	2.7.0	Email、Springboot整合包
aspectjweaver	1.9.7	AspectJ语言
signclient	3.0.1	永中签名生成
spring-boot-starter-security	2.7.0	权限控制
spring-security-oauth2-autoconfigure	2.6.8	
guava	31.1-jre	工具类
commons-text	1.9	工具类
spring-boot-starter-aop	2.7.0	Aop支持
mybatis-plus-generator	3.4.1	Mybatis代码生成
Thumbnailator	0.4.17	生成图像缩略图
oshi-core	5.4.0	采集系统信息工具
jna	5.11.0	运行期动态访问系统本地库
jna-platform	5.11.0	
commons-io	2.11.0	io流工具
velocity-engine-core	2.3	velocity模板
taobao-sdk-java	1.0	钉钉支持
taobao-sdk-java-source	1.0	

依赖项目

> 本文档适用 `jnpf-java-cloud` 3.4.3版本

项目	说明
jnpf-database	数据库脚本，包含【配置库】和【平台数据库】初始数据库
jnpf-common	微服务基础依赖
jnpf-registry	微服务中间件
jnpf-scheduletask	系统调度客户端依赖及服务端(XXL-JOB引擎)
jnpf-file-core-starter	文件服务核心工具包
jnpf-resourecs	静态资源

jnpf-registry

> 本文档适用 `jnpf-java-cloud` 3.4.3版本

> 源码项目：jnpf-registry

> 本项目为微服务的中间件

目录说明

项目	版本	官方文档地址	端口	说明
nacos-server	v2.1.1	Alibaba Nacos 文档	30099	服务注册&发现和配置中心
seata-server	v1.5.2	Alibaba Seata 文档	30095	分布式事务
sentinel-server	v1.8.5	Alibaba Sentinel 文档	30098	服务熔断
spring-boot-admin	2.7.0	Spring Boot Admin 文档	30097	服务监控
apache-skywalking-	v9.2.0	Apache SkyWalking 文	30096	链路追踪

apm		档		
skywalking-agent	v8.12.0	Apache SkyWalking 文档		探针
rocketmq	4.9.4	Apache RocketMQ 文档	30094	消息队列
rocketmq-console		文档	30093	RocketMQ控制台

nacos-server

> 所在路径：jnpf-registry/nacos-server

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

> Nacos只支持 `MySQL` 数据库

一 部署前准备

1.1 导入数据库脚本

> 导入数据库脚本前需要创建数据库

1.1.1 方式一

> Nacos官方提供的数据库脚本

将 `/conf/nacos-mysql.sql` 导入数据库中

1.1.2 方式二

> JNPF官方提供的数据库脚本，包括 `Seata` 相关表、 `Seata` 在 `Nacos` 中的配置、环境相关配置信息

将 `jnpf-database` 项目中的 `/java微服务/3.4.x/3.4.3/nacos-mysql.sql` 导入数据库中

1.2 修改配置文件

打开 `/conf/application.properties` 配置文件，找到第41-43行，修改数据库相关配置,如下所示

```
db.url.0=jdbc:mysql://127.0.0.1:3306/jnpf_nacos?characterEncoding=utf8&connectTimeout=1000&socketTimeout=3000&autoReconnect=true&useUnicode=true&useSSL=false&serverTimezone=GMT%2B8
db.user=root
db.password=123456
```

二 单机模式部署

2.1 修改启动配置

> JNPF官方提供的默认为单机模式

打开 `/bin/startup.cmd` (Windows环境)或 `/bin/startup.sh` (Linux)文件，找到第55行将 `MODE` 改为 `standalone`

```
export MODE="standalone"
```

三 集群模式部署

3.1 修改配置

在 `1.2 修改配置文件` 基础上，找到第27行，调整如下：

```
# 调整前
#nacos.inetutils.prefer-hostname-over-ip=true

# 调整后
nacos.inetutils.prefer-hostname-over-ip=true
```

3.2 修改集群配置

打开 `conf` 目录，将 `cluster.conf.example` 重命名为 `cluster.conf` 并打开编辑，在配置文件中增加节点地址，如下所示

```
10.233.105.11:30099
10.233.105.12:30099
10.233.105.13:30099
```

四、启动、停止操作

4.1 Windows环境

- 启动：双击`startup.cmd`
- 停止：双击`shutdown.cmd`

4.2 Linux环境

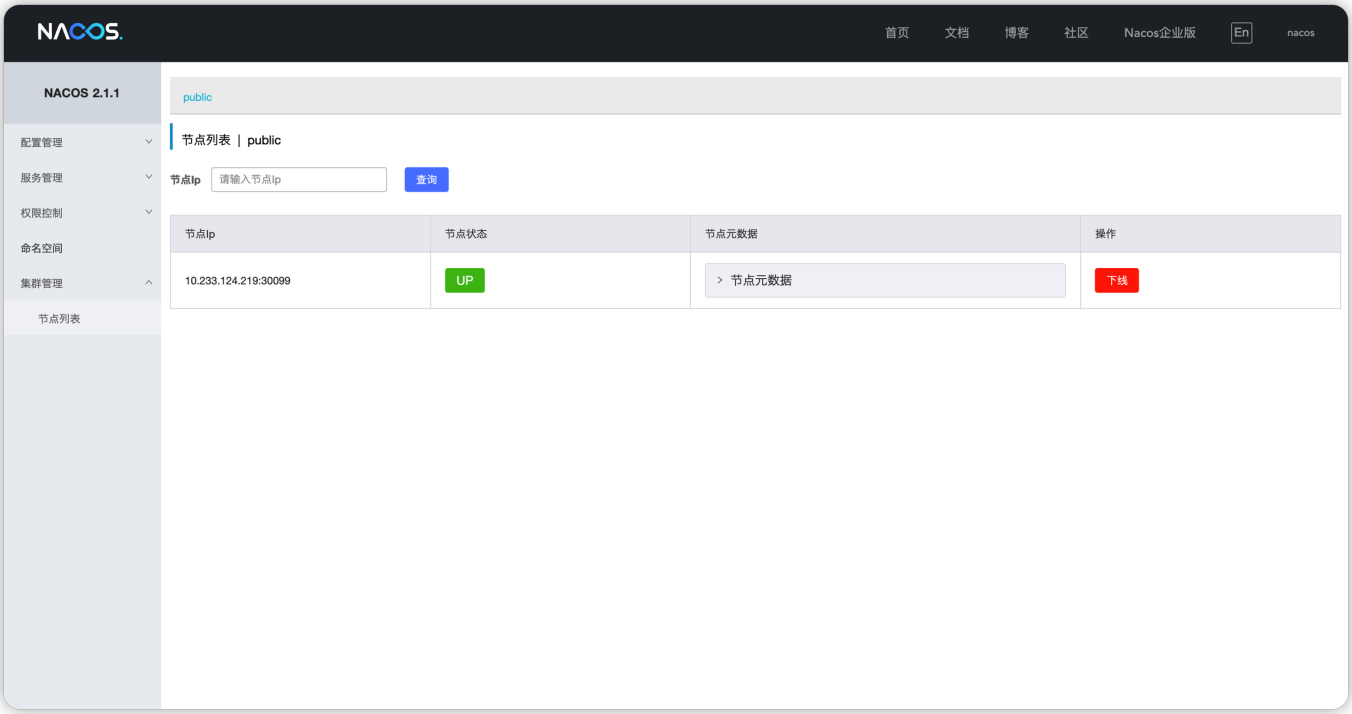
```
# 启动
sh startup.sh

# 停止
sh shutdown.sh
```

五 访问控制台

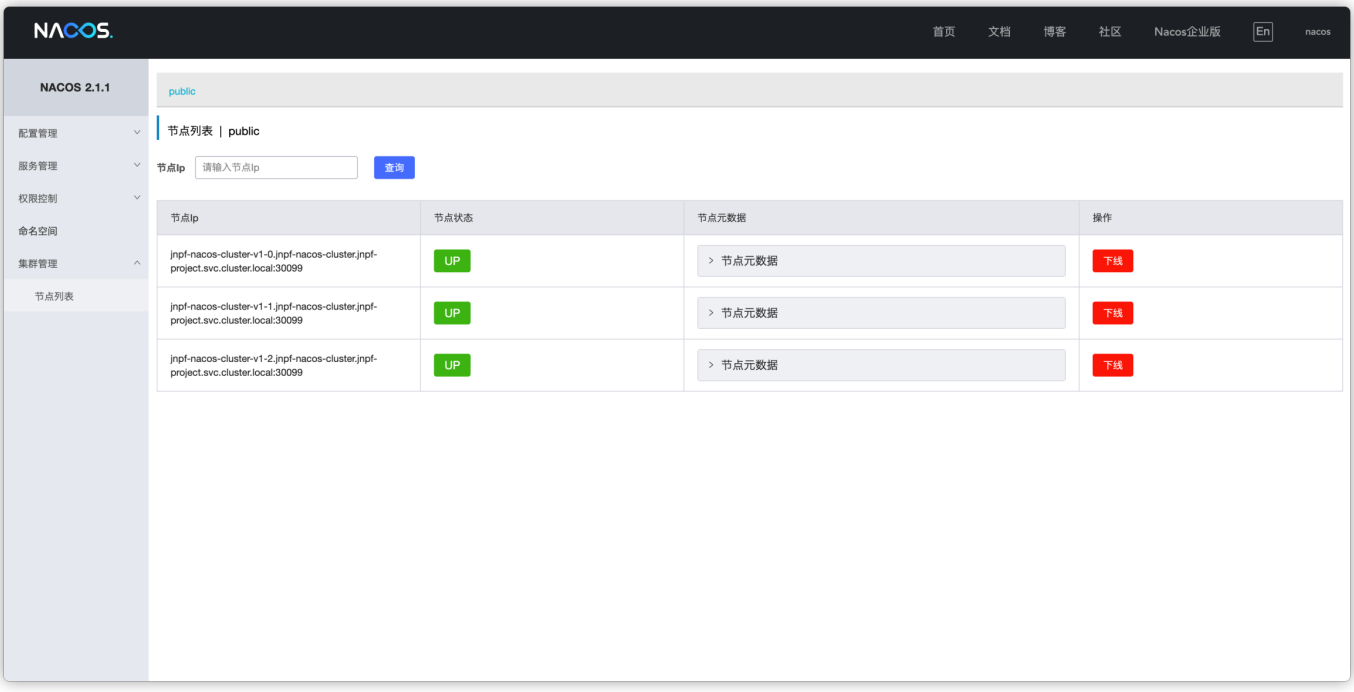
- 访问地址：http://127.0.0.1:30099/nacos
- 用户名：nacos
- 密码：nacos

六 页面效果

6.1 单机模式



6.2 集群模式



seata-server

> 所在路径：jnpf-registry/seata-server

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

一 部署前准备

1.1 导入数据库脚本

> 导入数据库脚本前需要创建数据库

1.1.1 方式一

> Seata官方提供的数据库脚本

将 `/script/server/db/mysql.sql` 导入数据库中

1.1.2 方式二

> JNPF官方提供的数据库脚本，包括 `Seata` 相关表、 `Seata` 在 `Nacos` 中的配置、环境相关配置信息

将 `jnpf-database` 项目中的 `/java微服务/3.4.x/3.4.3/nacos-mysql.sql` 导入数据库中

1.2 修改配置文件

注册至Nacos

打开 `/conf/application.yml` 配置文件，找到第11行，修改 `Nacos` 相关配置,如下所示

```yaml
spring:
  application:
    name: seata-server
  cloud:
    nacos:
      discovery:
        # 服务注册地址
        server-addr: 127.0.0.1:30099
      config:
        server-addr: ${spring.cloud.nacos.discovery.server-addr}
```

读取nacos上的配置

打开 `/conf/application.yml` 配置文件，找到第46行,如下所示，

`seata-server.properties` 为Nacos的配置

```
seata:
  config:
    # support: nacos 、 consul 、 apollo 、 zk  、 etcd3
    type: nacos
    nacos:
      # nacos ip地址
      server-addr: ${spring.cloud.nacos.discovery.server-addr}
      group: DEFAULT_GROUP
      namespace:
      username: "nacos"
      password: "nacos"
      ##if use MSE Nacos with auth, mutex with username/password attribute
      #access-key: ""
      #secret-key: ""
      # 读取nacos上的配置文件
      data-id: seata-server.properties
```

1.3 增加Nacos配置

打开 `Nacos` 控制台并登录，依次点击【配置管理】-【配置列表】-【public】，点击右侧【+】，

- Data ID： `seata-server` （注意：和上述1.2中读取 `nacos上的配置` 中的 `data-id` 名称保持一致）
- Group：默认即可
- 配置格式： `Properties` （注意：和上述1.2中读取 `nacos上的配置` 中的 `data-id` 格式保持一致）
- 配置内容：复制 `/script/config-center/config.txt` 所有内容，并做如下调整

```
# 第26行
#######事务分组名#######
service.vgroupMapping.default_tx_group=default

# 第28行
#######Seata服务地址#######
service.default.grouplist=127.0.0.1:30095

# 第69-71
######mode改成db模式(根据实际场景修改)######
store.mode=db
store.lock.mode=db
store.session.mode=db

# 第86-89行
```

```
######MySQL配置内容######
store.db.datasource=druid
store.db.dbType=mysql
# MySQL5.7.x 驱动为com.mysql.jdbc.Driver,MySQL8.0.x驱动为com.mysql.cj.jdbc.Driver
store.db.driverClassName=com.mysql.jdbc.Driver
store.db.url=jdbc:mysql://127.0.0.1:3306/jnpf_nacos?characterEncoding=utf8&rewriteBa
tchedStatements=true&connectTimeout=1000&socketTimeout=3000&autoReconnect=true&useUn
icode=true&useSSL=false&serverTimezone=GMT%2B8
store.db.user=username
store.db.password=password
```

二、启动、停止操作

2.1 Windows环境

- 启动：双击 `seata-server.bat`
- 停止：双击 `shutdown.bat`

2.2 Linux环境

```
# 启动
sh startup.sh

# 停止
sh shutdown.sh
```

三 访问控制台

- 访问地址： http://127.0.0.1:30095
- 用户名：seata
- 密码：seata

四 常见问题

如果直接使用官方下载的安装包并在 `JDK11` 环境下运行，需要将 `bin/seata-server.sh` 文件中 `第132行` 替换成如下内容

```
JAVA_MAJOR_VERSION=$($JAVACMD -version 2>&1 | head -1 | cut -d'"' -f2 | sed 's/^1
\.//' | cut -d'.' -f1)
```

sentinel-sever

> 所在路径：jnpf-registry/sentinel-sever

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

一 开发环境

1.1 转为Maven项目

在`sentinel-sever`根目录找到`pom.xml`，右击转为maven项目

1.2 修改配置

打开`sentinel-sever/src/main/resources/application.yml`， 修改`Nacos`服务注册地址

```
spring:
  profiles:
    active: dev
  application:
    name: sentinel-server
  cloud:
    nacos:
      discovery:
        server-addr: 127.0.0.1:30099
```

1.3 运行

找到`sentinel-server/src/main/java/com/alibaba/csp/sentinel/dashboard/DashboardApplication.javan`，右击运行即可

二 部署环境

2.1 打包

方式一

用IDEA直接打包

方式二

在 `sentinel-sever` 根目录执行 `mvn clean package -Dmaven.test.skip=true`

2.2 修改配置

打开 `sentinel-sever/conf/application.yml` ， 修改 `Nacos` 服务注册地址和命名空间

```
spring:
  profiles:
    active: dev
  application:
    name: sentinel-server
  cloud:
    nacos:
      discovery:
        server-addr: 127.0.0.1:30099
```

2.3 相关资源上传至服务器

> 确保以下资源存放在同一目录下

- sentinel-sever/target/sentinel-dashboard.jar
- sentinel-sever/bin/server.sh(Linux环境)
- sentinel-sever/bin/startup.bat(Windows环境)
- sentinel-sever/conf/application.yml(配置文件)
- sentinel-sever/conf/logback-spring.yml(可选，日志配置)

在Windows服务器环境中，直接双击 `startup.bat` 即可运行

在Linux服务器环境中执行以下命令

```
# 启动应用
sh server.sh start

# 重启应用
sh server.sh restart

# 停止应用
sh server.sh stop
```

三 访问控制台

- 访问地址： http://127.0.0.1:30098
- 用户名：sentinel
- 密码：sentinel

spring-boot-admin

> 所在路径：jnpf-registry/spring-boot-admin

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

一 开发环境

1.1 转为Maven项目

在`spring-boot-admin`根目录找到`pom.xml`，右击转为maven项目

1.2 修改配置

打开`spring-boot-admin/src/main/resources/application-dev.yml`， 修改`Nacos`服务注册地址和命名空间，

```yaml
spring:
  application:
    # 应用名称
    name: srping-boot-admin
  security:
    user:
      name: admin
      password: admin
  cloud:
    nacos:
      discovery:
        # 服务注册地址
        server-addr: 127.0.0.1:30099
        namespace: 69c4eecb-05bd-4041-81fe-1473f95f578c
```

1.3 运行

找到 `spring-boot-admin/src/main/java/jnpf/JnpfAdminApplication`，右击运行即可

二 部署环境

2.1 打包

方式一

用IDEA直接打包

方式二

在 `spring-boot-admin` 根目录执行 `mvn clean package -Dmaven.test.skip=true`

2.2 修改配置

> 需要和微服务在同一个命名空间

打开 `spring-boot-admin/conf/application-dev.yml`， 修改 `Nacos` 服务注册地址和命名空间

```
spring:
  application:
    # 应用名称
    name: srping-boot-admin
  security:
    user:
      name: admin
      password: admin
  cloud:
    nacos:
      discovery:
        # 服务注册地址
        server-addr: 127.0.0.1:30099
        namespace: 69c4eecb-05bd-4041-81fe-1473f95f578c
```

2.3 相关资源上传至服务器

> 确保以下资源存放在同一目录下

- spring-boot-admin/target/srping-boot-admin.jar
- spring-boot-admin/bin/server.sh(Linux环境)
- spring-boot-admin/bin/startup.bat(Windows环境)
- spring-boot-admin/conf/application-dev.yml(配置文件)
- spring-boot-admin/conf/logback-spring.yml(可选，日志配置)

在Windows服务器环境中，直接双击 `startup.bat` 即可运行

在Linux服务器环境中执行以下命令

```
# 启动应用
sh server.sh start
# 重启应用
sh server.sh restart
# 停止应用
sh server.sh stop
```

三 访问控制台

- 访问地址： http://127.0.0.1:30094
- 用户名：admin
- 密码：admin

apache-skywalking-apm

> 所在路径：jnpf-registry/apache-skywalking-apm

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

使用说明

> 以MySQL为例

创建数据库

> 启动应用后会自动创建相关表字段

创建数据库，库名如 `jnpf-skywalking`

修改配置

打开 `apache-skywalking-apm/config/application.xml` ,修改以下内容

```
# 第133行，存储改成 mysql
selector: ${SW_STORAGE:mysql}

# 第180-182行，修改mysql连接配置
jdbcUrl: ${SW_JDBC_URL:"jdbc:mysql://127.0.0.1:3306/jnpf_skywalking?characterEncoding=utf8&rewriteBatchedStatements=true&connectTimeout=1000&socketTimeout=3000&autoReconnect=true&useUnicode=true&useSSL=false&serverTimezone=GMT%2B8"}
dataSource.user: ${SW_DATA_SOURCE_USER:root}
dataSource.password: ${SW_DATA_SOURCE_PASSWORD:123456}
```

启动

Windows环境

双击 `apache-skywalking-apm/bin/startup.bat` 即可

Linux环境

```
sh startup.sh
```

访问控制台

- 访问地址： http://127.0.0.1:30096

skywalking-agent

> 所在路径：jnpf-registry/skywalking-agent

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

开发环境

在 IDEA中加上以下内容即可

```
-javaagent:E:\Code\jnpf-registry\skywalking-agent\skywalking-agent.jar
-Dskywalking.agent.service_name=jnpf-gateway
-Dskywalking.collector.backend_service=localhost:11800
```

部署环境

将 `skywalking-agent/skywalking-agent.jar` 上传至指定位置(如 `/www/wwwroot/skywalking-agent/`)

在服务启动命令的 `-jar` 加上以下内容

```
-javaagent:/www/wwwroot/skywalking-agent/skywalking-agent.jar -Dskywalking.agent.ser
vice_name=jnpf-gateway -Dskywalking.collector.backend_service=localhost:11800
```

以 `jnpf-gateway` 服务以例，完整的启动命令如下：

```
java -javaagent:/www/wwwroot/skywalking-agent/skywalking-agent.jar -Dskywalking.agen
t.service_name=jnpf-gateway -Dskywalking.collector.backend_service=localhost:11800 -
jar jnpf-gateway-3.4.2-RELEASE.jar >Log.log 2>&1 &
```

rocketmq

> 所在路径：jnpf-registry/rocketmq

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

修改配置

修改broker.conf

打开 `rocketmq/conf/broker.conf` ，修改 `namesrvAddr` 和 `brokerIP1` 的IP地址为服务器IP地址

```
# nameServer地址
namesrvAddr = 192.168.0.230:30094

# 网卡InetAddress，当前broker监听的IP
brokerIP1 = 192.168.0.230
```

修改默认端口

打开 `rocketmq/conf/namesrv.properties`，修改 `listenPort`

```
listenPort=30094
```

启动

```
# 启动 NameServer
nohup sh /data/rocketmq/bin/mqnamesrv -c /data/rocketmq/conf/namesrv.properties >> /data/logs/namesrv_stdout.log 2>&1 &

# 启动 Broker
nohup sh /data/rocketmq/bin/mqbroker -c /data/rocketmq/conf/broker.conf >> /data/logs/broker_stdout.log 2>&1 &
```

在微服务中使用

打开 `jnpf-java-cloud` 项目，修改 `/jnpf-message/jnpf-message-server/src/main/resources/bootstrap.yml` 中的 `name-server`

```
    # stream配置
    stream:
      rocketmq:
        binder:
          name-server: 192.168.0.230:30094
        bindings:
          output:
            producer:
              sync: true
              group: jnpf-group1
```

rocketmq-console

> 所在路径：jnpf-registry/rocketmq-console

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

一 开发环境

1.1 转为Maven项目

在`rocketmq-console`根目录找到`pom.xml`，右击转为maven项目

1.2 修改配置

打开`rocketmq-console/src/main/resources/application.propertis`，修改`RocketMQ`服务地址

```
rocketmq.config.namesrvAddr=192.168.0.230:30094
```

1.3 运行

找到`rocketmq-console/src/main/java/org/apache/rocketmq/console/App`，右击运行即可

二 部署环境

2.1 打包

方式一

用IDEA直接打包

方式二

在`rocketmq-console`根目录执行`mvn clean package -Dmaven.test.skip=true`

2.2 修改配置

打开`rocketmq-console/conf/application.yml`，修改`RocketMQ`服务地址

```
rocketmq.config.namesrvAddr=192.168.0.230:30094
```

2.3 相关资源上传至服务器

> 确保以下资源存放在同一目录下

- rocketmq-console/target/rocketmq-console-ng-2.0.0.jar
- rocketmq-console/bin/server.sh(Linux环境)
- rocketmq-console/bin/startup.bat(Windows环境)
- rocketmq-console/src/main/resources/application.propertis (配置文件)
- rocketmq-console/src/main/resources/users.propertis (可选，控制台账号密码)

在Windows服务器环境中，直接双击 `startup.bat` 即可运行

在Linux服务器环境中执行以下命令

```
# 启动应用
sh server.sh start

# 重启应用
sh server.sh restart

# 停止应用
sh server.sh stop
```

三 访问控制台

- 访问地址： http://127.0.0.1:30093
- 用户名：admin
- 密码：admin

jnpf-database

> 本文档适用 `jnpf-java-cloud` 3.4.3版本

> 源码项目：jnpf-database

> 本项目为数据脚本文件，包含【配置库】和【平台数据库】初始数据库脚本及文件

目录说明

- `java微服务` ：java微服务 `配置数据库` 脚本
- `MySQL` ：MySQL `平台数据库` 脚本，单体、微服务共用
- `SQLServer` ：SQLServer `平台数据库` 脚本，单体、微服务共用
- `PostgreSQL` ：PostgreSQL `平台数据库` 脚本，单体、微服务共用
- `Oracle` ：Oracle `平台数据库` 脚本，单体、微服务共用
- `DM8` ：达梦DM8 `平台数据库` 文件，单体、微服务共用
- `KingbaseES` ：人大金仓 `平台数据库` 文件，单体、微服务共用

jnpf-common

> 源码项目：jnpf-common

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

> 本项目为 `jnpf-java-cloud` 的基础依赖，可上传到私服或使用本地导入的方式引用该项目

一 结构说明

```
jnpf-common
     ├─── jnpf-common-auth - 认证模块
     ├─── jnpf-common-core - 基础类及常用工具
     ├─── jnpf-common-database - 数据库配置及多数据库兼容
     ├─── jnpf-common-dubbo - dubbo拦截器, 自动封装认证信息
     ├─── jnpf-common-feign - 远程调用Feign组件配置
     ├─── jnpf-common-redis - 缓存工具Redis组件配置
     ├─── jnpf-common-security - 接口鉴权配置
     ├─── jnpf-common-swagger - API组件Swagger配置
     └─── jnpf-common-utils - 工具包
     ├          ├─── jnpf-common-connector - 单点数据推送模块
     ├          ├─── jnpf-common-file - 文件工具类模块
     ├          ├─── jnpf-common-message - 短信模块
     ├          ├─── jnpf-common-office - office操作模块
     ├          ├─── jnpf-common-scheduletask - 调度工具模块
     ├          └─── jnpf-common-seata - seata依赖
```

二 使用方式

2.1 私服发布

> 需要Maven私服仓库，若相同版本重新发布，需要先登录私服把 `maven-releases` 库中的 `com.jnpf` 目录删除

2.1.1 修改Maven配置

修改Maven文件下conf文件夹中的 `setttings.xml` ，在 `<servers></servers>` 标签中增加 `<server><server>` ，示例:

```
<server>
    <id>maven-releases</id>
    <username>jnpf-user（账号，结合私服配置设置）</username>
    <password>123456（密码，结合私服配置设置）</password>
</server>
<server>
    <id>maven-snapshots</id>
    <username>jnpf-user（账号，结合私服配置设置）</username>
    <password>123456（密码，结合私服配置设置）</password>
</server>
```

修改 `jnpf-common` 项目根目录下的 `pom.xml` 文件，增加maven配置

```
<repository>
    <id>maven-releases(与server-id一致)</id>
    <name>maven-releases</name>
    <url>http://ip:port/repository/maven-releases/(仓库地址)</url>
</repository>
```

2.1.2 发布到私服

打开 `jnpf-common` 项目,运行 `deploy` 插件，刷新Maven配置即可

2.2 本地安装

打开 `jnpf-common` 项目,运行 `install` 插件，将 `jnpf-common` 中的包安装至本地

2.3 本地导入

点击 `File` > `Project Structure..` > `module` ，点击右边的+号，点击 `import module` ，找到 `jnpf-common` 项目后点击ok,点击apply，ok,刷新Maven即可

jnpf-scheduletask

> 本文档适用 `jnpf-java-cloud` 3.4.3版本

> 源码项目：jnpf-scheduletask

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

> 本项目为任务调度的基础依赖和服务端，作为基础依赖时需要上传到私服或使用本地导入的方式引用该项目，作为服务端时需求单独部署

一 结构说明

```
jnpf-scheduletask
     ├─── jnpf-scheduletask-client - xxl-job客户端需要加入此模块
     ├─── jnpf-scheduletask-model - 基础模型
     ├─── xxl-job-core - xxl-job核心包
     ├─── xxl-job-admin - xxl-job服务端及启动类
```

二 使用方式

2.1 作为客户端依赖

2.1.1 Maven私服配置

> 需要Maven私服仓库,若相同版本重新发布，需要先登录私服把 `maven-releases` 库中的 `com.jnpf` 目录下的 `jnpf-scheduletask` 删除

修改Maven文件下conf文件夹中的 `setttings.xml` ，在 `<servers></servers>` 标签中增加 `<server></server>` ，示例:

```
<server>
    <id>maven-releases</id>
    <username>jnpf-user（账号，结合私服配置设置）</username>
    <password>123456（密码，结合私服配置设置）</password>
</server>
<server>
    <id>maven-snapshots</id>
    <username>jnpf-user（账号，结合私服配置设置）</username>
    <password>123456（密码，结合私服配置设置）</password>
</server>
```

修改 `jnpf-scheduletask` 项目根目录下的 `pom.xml` 文件，增加maven配置

```
  <repository>
    <id>maven-releases(与server-id一致)</id>
    <name>maven-releases</name>
```

```
    <url>http://ip:port/repository/maven-releases/(仓库地址)</url>
  </repository>
```

2.1.2 发布到私服

双击 `jnpf-scheduletask` 中的 `deploy` 插件，刷新Maven配置即可

2.1.3 本地安装

打开 `jnpf-scheduletask` 项目,运行install插件， 将 `jnpf-scheduletask` 中的包安装至本地

2.1.4 本地导入

点击 `File` > `Project Structure..` > `module` ，点击右边的+号，点击 `import module` ，找到 `jnpf-scheduletask` 项目后点击ok,点击apply，ok,刷新Maven即可

2.2 作为服务端

2.2.1 数据源配置

> 暂只支持 `MySQL`

打开 `xxl-job-admin/src/main/resources/application-dev.yml` 修改数据源及Redis配置

2.2.2 打包部署

Maven打包后，将 `/xxl-job-admin/target/xxl-job-admin-1.0-RELEASE.jar` 上传至服务并运行

2.2.3 修改Nacos相关配置

打开 `Nacos` 控制台，依次点击 `配置管理-配置列表-develop` 中的，编辑 `datasource-scheduletask.yaml` 相关配置

jnpf-file-core-starter

> 本文档适用 `jnpf-java-cloud` 3.4.3版本

> 源码项目：jnpf-file-core-starter

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/11` 、 `BiShengJDK 8/11`

> 本项目为文件服务的基础依赖，可上传到私服或使用本地导入的方式引用该项目

使用方式

私服发布

> 需要Maven私服仓库，若相同版本重新发布，需要先登录私服把 `maven-releases` 库中的 `com.jnpf` 目录下的 `jnpf-file-core-starter` 删除

修改Maven配置

修改Maven文件下conf文件夹中的 `setttings.xml` ，在 `<servers></servers>` 标签中增加 `<server></server>` ，示例:

```
<server>
    <id>maven-releases</id>
    <username>jnpf-user（账号，结合私服配置设置）</username>
    <password>123456（密码，结合私服配置设置）</password>
</server>
<server>
    <id>maven-snapshots</id>
    <username>jnpf-user（账号，结合私服配置设置）</username>
    <password>123456（密码，结合私服配置设置）</password>
</server>
```

修改 `jnpf-file-core-starter` 项目根目录下的 `pom.xml` 文件，增加maven配置

```
  <repository>
    <id>maven-releases(与server-id一致)</id>
    <name>maven-releases</name>
    <url>http://ip:port/repository/maven-releases/(仓库地址)</url>
  </repository>
```

发布到私服

打开 `jnpf-file-core-starter` 项目,双击 `deploy` 插件，刷新Maven配置即可

本地安装

打开 `jnpf-file-core-starter` 项目,运行 `install` 插件， 将 `jnpf-file-core-starter` 中的包安装至本地

本地导入

点击 `File` > `Project Structure..` > `module` ，点击右边的+号，点击 `import module` ，找到 `jnpf-file-core-starter` 项目后点击ok,点击apply，ok,刷新Maven即可

jnpf-resourecs

> 本文档适用 `jnpf-java-cloud` 3.4.3版本

> 源码项目： `jnpf-resourecs`

> 本项目为静态资源初始结构及文件

目录结构

```
├─── BiVisualPath              # 大屏设计
├─── CodeTemp                  # 代码生成器临时目录
├─── DocumentFile              # 文档
├─── DocumentPreview           # 文档预览
├─── EmailFile                 # 邮件附件
├─── IMContentFile             # IM聊天附件
├─── SystemFile                # 系统附件
├─── TemplateCode              # 代码生成器模板
│    ├─── TemplateCode1        # 流程表单模板
│    ├─── TemplateCode2        # 功能表单模板
│    ├─── TemplateCode3        # 功能流程模板
│    └─── TemplateCode4        # 纯表单模板
├─── TemplateFile              # 其他模板文档
├─── TemporaryFile             # 临时存放目录
├─── UserAvatar                # 用户头像
└─── WebAnnexFile              # 其他
```

配置说明

打开 `Nacos` 控制台，依次点击 `配置管理-配置列表-develop` ，编辑 `resources.yaml` 调整相关配置

本地存储

> 设置为本地存储时，静态资源文件需要和文件服务在同一台服务器

- 修改默认存储平台(`default-platform`)为 `local-plus-1`
- 修改基础路径 `base-path` 的值
  - 本地开发为项目所在路径，如： `F:/work/jnpf-resources/`
  - 部署环境，如 `/www/wwwroot/jnpf-resources/`

```
  file-storage: #文件存储配置，不使用的情况下可以不写
    default-platform: local-plus-1 #默认使用的存储平台
    local-plus: # 本地存储升级版
      - platform: local-plus-1 # 存储平台标识
        enable-storage: true  #启用存储
        enable-access: true #启用访问（线上请使用 Nginx 配置，效率更高）
        domain: "" # 访问域名，例如：“http://127.0.0.1:8030/”，注意后面要和 path-patterns
保持一致，“/”结尾，本地存储建议使用相对路径，方便后期更换域名
        base-path: F:/work/jnpf-resources/ # 基础路径
        path-patterns: /** # 访问路径
        storage-path:  # 存储路径
```

对象存储(含minio)

> 以MinIO为例

- 修改默认存储平台(`default-platform`)为 `minio-1`
- 配置 `MiniIO` ，涉及如下参数
  - `access-key` ：access-key
  - `secret-key` ：secret-key
  - `end-point` ：minio服务地址
  - `bucket-name` ：桶名

```
  file-storage: #文件存储配置，不使用的情况下可以不写
    default-platform: local-plus-1 #默认使用的存储平台
    thumbnail-suffix: ".min.jpg" #缩略图后缀，例如【.min.jpg】【.png】
    local-plus: # 本地存储升级版
      - platform: local-plus-1 # 存储平台标识
        enable-storage: true  #启用存储
        enable-access: true #启用访问（线上请使用 Nginx 配置，效率更高）
        domain: "" # 访问域名，例如：“http://127.0.0.1:8030/”，注意后面要和 path-patterns
保持一致，“/”结尾，本地存储建议使用相对路径，方便后期更换域名
        base-path: F:/work/jnpf-resources/ # 基础路径
        path-patterns: /** # 访问路径
        storage-path:  # 存储路径
    minio: # MinIO，由于 MinIO SDK 支持 AWS S3，其它兼容 AWS S3 协议的存储平台也都可配置在这
里
      - platform: minio-1 # 存储平台标识
        enable-storage: true  # 启用存储
```

```
        access-key: Q9jJs2b6Tv
        secret-key: Thj2WkpLu9DhmJyJ
        end-point: http://192.168.0.207:9000/
        bucket-name: jnpfsoftoss
        domain:  # 访问域名，注意“/”结尾，例如：http://minio.abc.com/abc/
        base-path:  # 基础路径
```

本地运行

> 本文档适用 `jnpf-java-cloud` 3.4.3版本

> 特别说明：源码、JDK、MySQL、Redis、Nacos、Seata、Sentinel等存放路径禁止包含中文、空格、特殊字符等

一 技术栈

- 主框架：`Spring Cloud Alibaba` + `Spring Boot` + `Spring Framework`
- 持久层框架：`MyBatis-Plus`
- JSON序列化: `Jackson` & `Fastjson`
- 缓存：`Redis`
- 数据库：`MySQL` (默认)、`SQLServer` 、`Oracle` 、`PostgreSQL` 、`达梦数据库` 、`人大金仓数据库`
- API网关：`SpringCloud Gateway`
- 服务注册&发现和配置中心: `Alibaba Nacos`
- 服务监控：`Spring Boot Admin`
- 服务消费：`OpenFeign` / `Dubbo RPC`
- 负载均衡：`Ribbon`
- 服务熔断：`Alibaba Sentinel`
- 项目构建：`Maven`
- 分布式事务：`Alibaba Seata`
- 消息队列：`Apache RocketMQ` (默认), `RabittMQ` , `Apache Kafka`
- 安全框架：`spring-cloud-security-oauth2` + `jwt`
- 链路追踪：`Apache SkyWalking`
- 模板引擎：`Velocity`
- 任务调度：`XXL-JOB`
- 即时通讯：`spring-boot-starter-websocket`
- AOP：`spring-boot-starter-aop`

二 配置推荐

- 开发环境：I5 CPU+16G内存
- 测试生产环境： 8核32G(最低要求)

三 环境要求

> 官方建议：JDK版本不低于 `1.8.0_281` 版本，兼容JDK 8、JDK11,可使用 `OpenJDK 8/11` 、 `Alibaba Dragonwell 8/` 、 `BiShengJDK 8/11`

> 【配置库】仅支持MySQL,【平台数据库】支持MySQL、SQLServer、Oracle、PostgreSQL、达梦数据库、人大金库数据库

> `Alibaba Dragonwell 8/11` 下载地址：https://dragonwell-jdk.io

> `BiShengJDK 8/11` 下载地址：https://www.hikunpeng.com/developer/devkit/compiler/jdk

> 在使用JDK11时需要把项目根目录下的 `pom.xml` 中的 改成`11`

项目	推荐版本	说明
IDEA	IDEA2020及以上版本	
JDK	1.8.0_281及以上版本	JAVA环境依赖(需配置环境变量)
Maven	3.6.3及以上版本	项目构建(需配置环境变量)
Redis	3.2.100(Windows)/4.0.x+(Linux,Mac)	缓存
MySQL	5.7.x+	数据库任选一(默认)
SQLServer	2012+	数据库任选一
Oracle	11g+	数据库任选一
PostgreSQL	12+	数据库任选一
达梦数据库	DM8	数据库任选一
人大金库	KingbaseES V8 R6	数据库任选一

Nacos	v2.1.1	jnpf-registry项目中的nacos-server
Seata	v1.5.2	jnpf-registry项目中的seata-server
Sentinel(可选)	v1.8.5	jnpf-registry项目中的sentinel-server
Spring-Boot-Admin(可选)	2.7.0	jnpf-registry项目中的spring-boot-admin
SkyWalking APM(可选)	v9.2.0	jnpf-registry项目中的apache-skywalking-apm
SkyWalking Agent (可选)	v8.12.0	jnpf-registry项目中的skywalking-agent
Apache RocketMQ(必需)	4.9.4	jnpf-registry项目中的rocketmq

四 IDEA插件

- `Lombok` (必需)
- `Alibaba Java Coding Guidelines`
- `MybatisX`

五 Maven私服配置

> 因部分依赖在阿里云Maven私服、Maven官方无法下载

依赖包差异

- com.sqlserver:sqljdbc4:4.0
- com.oracle:ojdbc6:11.2.0
- com.dm:DmJdbcDriver18:1.8.0
- com.kingbase8:kingbase8-jdbc:2.0
- dingtalk-sdk-java:taobao-sdk-java-source:1.0
- dingtalk-sdk-java:taobao-sdk-java:1.0
- yozo:signclient:3.0.1

打开`maven`下`conf/settings.xml`x文件，

```
# 在<servers></servers>中添加如下内容
<server>
    <id>maven-releases</id>
```

```
    <username>jnpf-user</username>
    <password>HLrQ0MA%S1nE</password>
</server>
<server>
    <id>maven-snapshots</id>
    <username>jnpf-user</username>
    <password>HLrQ0MA%S1nE</password>
</server>

# 在<mirrors></mirrors>中添加
<mirror>
  <id>maven-snapshots</id>
  <mirrorOf>*</mirrorOf>
  <name>maven-snapshots</name>
  <url>https://repository.jnpfsoft.com/repository/maven-public/</url>
</mirror>
```

六 服务端口说明

服务名称	默认端口	描述
jnpf-gateway	30000	服务网关
jnpf-oauth	30001	认证服务
jnpf-system	30002	系统基础服务
jnpf-visualdev	30003	可视化开发(在线开发、代码生成、门户设计)
jnpf-workflow	30004	工作流
jnpf-file	30005	文件服务
jnpf-tenant	30006	多租户服务
jnpf-datareport	30007	报表服务
jnpf-message	30008	消息中心
jnpf-scheduletask	30009	任务调度(客户端)
jnpf-permission	30010	权限服务
jnpf-visualdata	30011	数据大屏服务
jnpf-app	30012	移动端服务

jnpf-extend	30019	扩展(系统内置示例)
jnpf-example	30100	服务模板(参考示例)

七 开发环境

7.1 环境准备

> 开发环境以Windows 10为例

7.1.1 基础环境

项目	说明
IDEA	使用IDEA 2020及以上版本，也可以用eclipse等其他JAVA开发工具
JDK	1.8.0_281及以上版本
Maven	3.6.3及以上版本
Redis	3.2.100
Navicat(可选)	数据库管理工具
RDM(可选)	Redis管理工具

7.1.2 关联项目

项目	说明
jnpf-database	数据库脚本，包含【配置库】和【平台数据库】初始数据库
jnpf-common	微服务基础依赖
jnpf-registry	微服务中间件
jnpf-scheduletask	系统调度服务端(XXL-JOB引擎)
jnpf-file-core-starter	文件服务核心工具包
jnpf-resourecs	静态资源

7.2 初始化数据库

7.2.1 创建数据库

分别创建【配置库】(如 `jnpf-nacos`)和【平台数据库】(如 `jnpf-cloud`)

7.2.2 导入数据库脚本

将 `/jnpf-database/java微服务/<版本>/java_nacos.sql` 导入至上述创建的【配置库】中

将 `/jnpf-database/MySQL/<版本>/java_init.sql` 导入至上述创建的【平台数据库】中

7.3 中间件部署

详见 `jnpf-registry` 项目下相关中间件的 `README.md` 文档说明

7.4 准备依赖项目

7.4.1 导入基础依赖

详见 `jnpf-common` 项目中的 `README.md` 文档说明

7.4.2 系统调度服务端

详见 `jnpf-scheduletask` 项目中的 `README.md` 文档说明

7.4.3 文件服务核心工具包

详见 `jnpf-file-core-starter` 项目中的 `README.md` 文档说明

7.5 Nacos配置

登录 `Nacos` 控制台，依次点击 `配置管理` - `配置列表` ，选择 `develop` 环境并调整以下配置

- `datasource.yaml` ： 数据源及Redis配置
- `resources.yaml` ： 静态资源配置
- `tenant.yaml` ： 多租户配置(默认不需要配置)
- `system-config.yaml` ： 系统基础配置
- `datasource-scheduletask.yaml` ： 任务调度客户端数据源配置
- `logger.yaml` ： 日志配置
- `frame-config.yaml` ： 构架配置
- `oauth.yaml` ： 单点登录(SSO)配置

7.6 项目启动

7.6.1 移动端服务

端口配置：/jnpf-app/jnpf-app-server/src/main/resources/application.yml

服务配置：/jnpf-app/jnpf-app-server/src/main/resources/bootstrap.yml

启动类：/jnpf-app/jnpf-app-server/src/main/java/jnpf/JnpfApplication

7.6.2 报表服务

端口配置：/jnpf-datareport/report-console/src/main/resources/application.yml

服务配置：/jnpf-datareport/report-console/src/main/resources/bootstrap.yml

启动类：/jnpf-datareport/report-console/src/main/java/com/bstek/ureport/console/JnpfDataReportApplication

7.6.3 扩展应用服务

端口配置：/jnpf-extend/jnpf-extend-server/src/main/resources/application.yml

服务配置：/jnpf-extend/jnpf-extend-server/src/main/resources/bootstrap.yml

启动类：/jnpf-extend/jnpf-extend-server/src/main/java/jnpf/JnpfExtendApplication

7.6.4 文件服务

> 需要先发布 `jnpf-file-core` 项目，并在 `application.yml` 修改相关配置

端口配置：/jnpf-file/jnpf-file-server/src/main/resources/application.yml

服务配置：/jnpf-file/jnpf-file-server/src/main/resources/bootstrap.yml

启动类：/jnpf-file/jnpf-file-server/src/main/java/jnpf/JnpfFileAplication

7.6.4 网关服务

端口配置：/jnpf-gateway/src/main/resources/application.yml

服务配置：/jnpf-gateway/src/main/resources/bootstrap.yml(环境配置，默认启用开发环境bootstrap-dev.yml)

启动类：/jnpf-gateway/src/main/java/jnpf/system/base/JnpfGatewayApplication

7.6.5 消息服务

> 需要配置MQ，默认使用RocketMQ

端口配置：/jnpf-message/jnpf-message-server/src/main/resources/application.yml

服务配置：/jnpf-message/jnpf-message-server/src/main/resources/bootstrap.yml

启动类：/jnpf-message/jnpf-message-server/src/main/java/jnpf/JnpfMessageApplication

7.6.6 认证服务

端口配置：/jnpf-oauth/jnpf-oauth-server/src/main/resources/application.yml

服务配置：/jnpf-oauth/jnpf-oauth-server/src/main/resources/bootstrap.yml

启动类：/jnpf-oauth/jnpf-oauth-server/src/main/java/jnpf/JnpfOauthApplication

7.6.7 权限服务

端口配置：/jnpf-permission/jnpf-permission-server/src/main/resources/application.yml

服务配置：/jnpf-permission/jnpf-permission-server/src/main/resources/bootstrap.yml

启动类：/jnpf-permission/jnpf-permission-server/src/main/java/jnpf/JnpfPermissionApplication

7.6.8 任务调度服务

> 依赖 `jnpf-scheduletask` 项目，并在 `application.yml` 修改任务调度服务端配置

端口配置：/jnpf-scheduletask/jnpf-scheduletask-server/src/main/resources/application.yml

服务配置：/jnpf-scheduletask/jnpf-scheduletask-server/src/main/resources/bootstrap.yml

启动类：/jnpf-scheduletask/jnpf-scheduletask-server/src/main/java/jnpf/JnpfScheduleTaskApplication

7.6.9 系统服务

端口配置：/jnpf-system/jnpf-system-server/src/main/resources/application.yml

服务配置：/jnpf-system/jnpf-system-server/src/main/resources/bootstrap.yml

启动类：/jnpf-system/jnpf-system-server/src/main/java/jnpf/JnpfSystemApplication

7.6.10 文件服务

> 依赖 `jnpf-file-core` 项目

端口配置：/jnpf-file/jnpf-file-server/src/main/resources/application.yml

服务配置：/jnpf-file/jnpf-file-server/src/main/resources/bootstrap.yml

启动类：/jnpf-file/jnpf-file-server/src/main/java/jnpf/JnpfFileAplication

7.6.11 租户服务

> 默认不启用

端口配置：/jnpf-tenant/jnpf-tenant-server/src/main/resources/application.yml

服务配置：/jnpf-tenant/jnpf-tenant-server/src/main/resources/bootstrap.yml

启动类：/jnpf-tenant/jnpf-tenant-server/src/main/java/jnpf/JnpfTenantApplication

7.6.12 大屏服务

端口配置：/jnpf-visualdata/jnpf-visualdata-server/src/main/resources/application.yml

服务配置：/jnpf-visualdata/jnpf-visualdata-server/src/main/resources/bootstrap.yml

启动类：/jnpf-visualdata/jnpf-visualdata-server/src/main/java/jnpf/JnpfVisualdataApplication

7.6.13 可视化开发服务

端口配置：/jnpf-visualdev/jnpf-visualdev-server/src/main/resources/application.yml

服务配置：/jnpf-visualdev/jnpf-visualdev-server/src/main/resources/bootstrap.yml

启动类：/jnpf-visualdev/jnpf-visualdev-server/src/main/java/jnpf/JnpfVisualdevApplication

7.6.14 工作流服务

端口配置：/jnpf-workflow/jnpf-workflow-server/src/main/resources/application.yml

服务配置：/jnpf-workflow/jnpf-workflow-server/src/main/resources/bootstrap.yml

启动类：/jnpf-workflow/jnpf-workflow-server/src/main/java/jnpf/JnpfWorkflowApplication

八 Swagger接口

8.1 全局(网关)

访问地址：http://localhost:30000/swagger-ui/

8.2 移动端服务

访问地址：http://localhost:30012/swagger-ui/

8.3 扩展应用服务

访问地址：http://localhost:30019/swagger-ui/

8.4 文件服务

访问地址：http://localhost:30005/swagger-ui/

8.5 消息服务

访问地址：http://localhost:30008/swagger-ui/

8.6 认证服务

访问地址：http://localhost:30001/swagger-ui/

8.7 权限服务

访问地址：http://localhost:30010/swagger-ui/

8.8 任务调度服务

访问地址：http://localhost:30009/swagger-ui/

8.9 系统服务

访问地址：http://localhost:30002/swagger-ui/

8.10 文件服务

访问地址：http://localhost:30005/swagger-ui/

8.11 租户服务

访问地址：http://localhost:30006/swagger-ui/

8.12 大屏服务

访问地址：http://localhost:30011/swagger-ui/

8.13 可视化开发服务

访问地址：http://localhost:30003/swagger-ui/

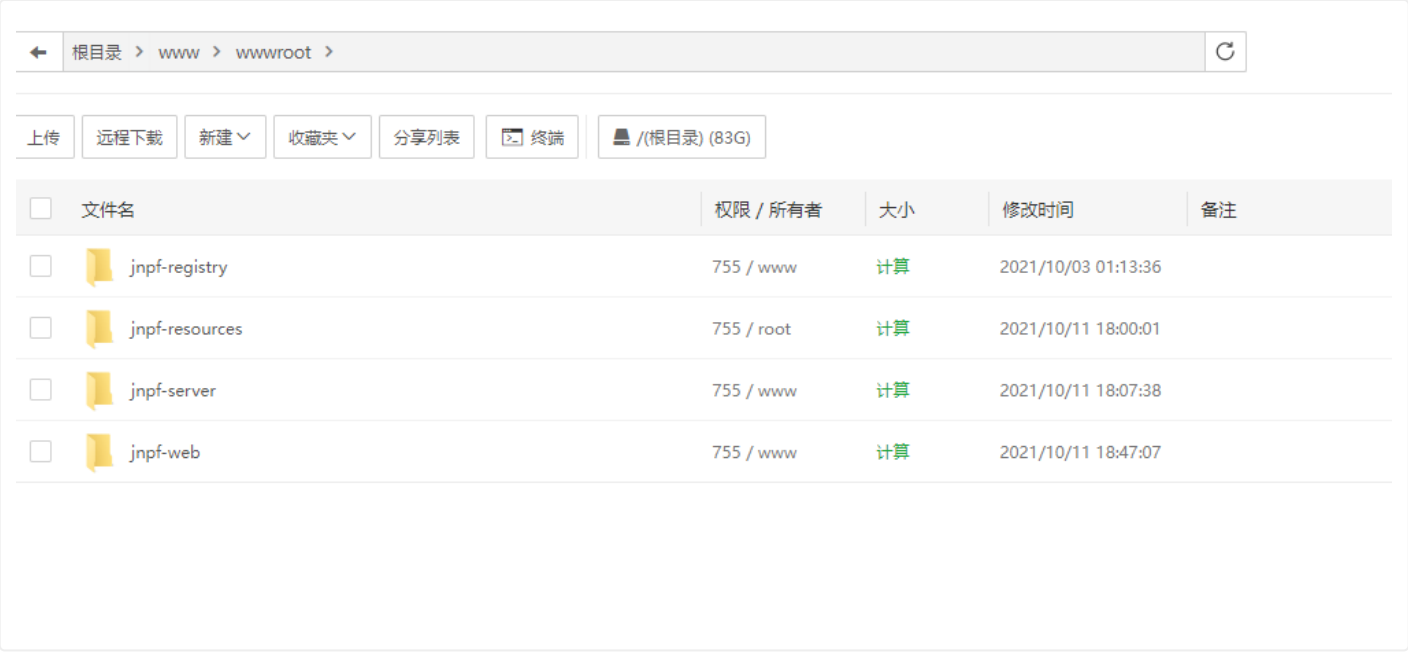
8.14 工作流服务

访问地址：http://localhost:30004/swagger-ui/

部署环境

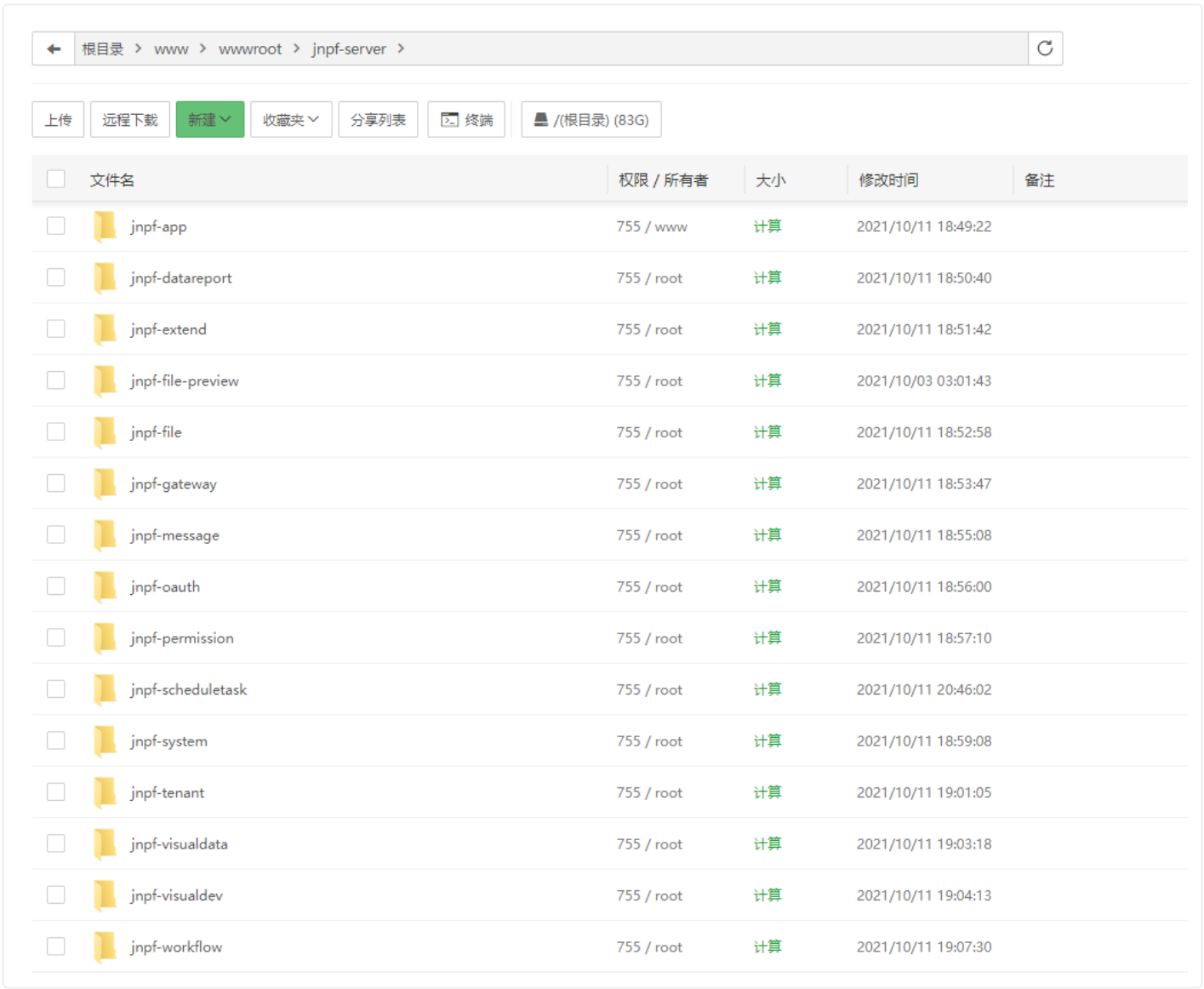
Linux部署目录结构说明

1.部署服务器目录结构

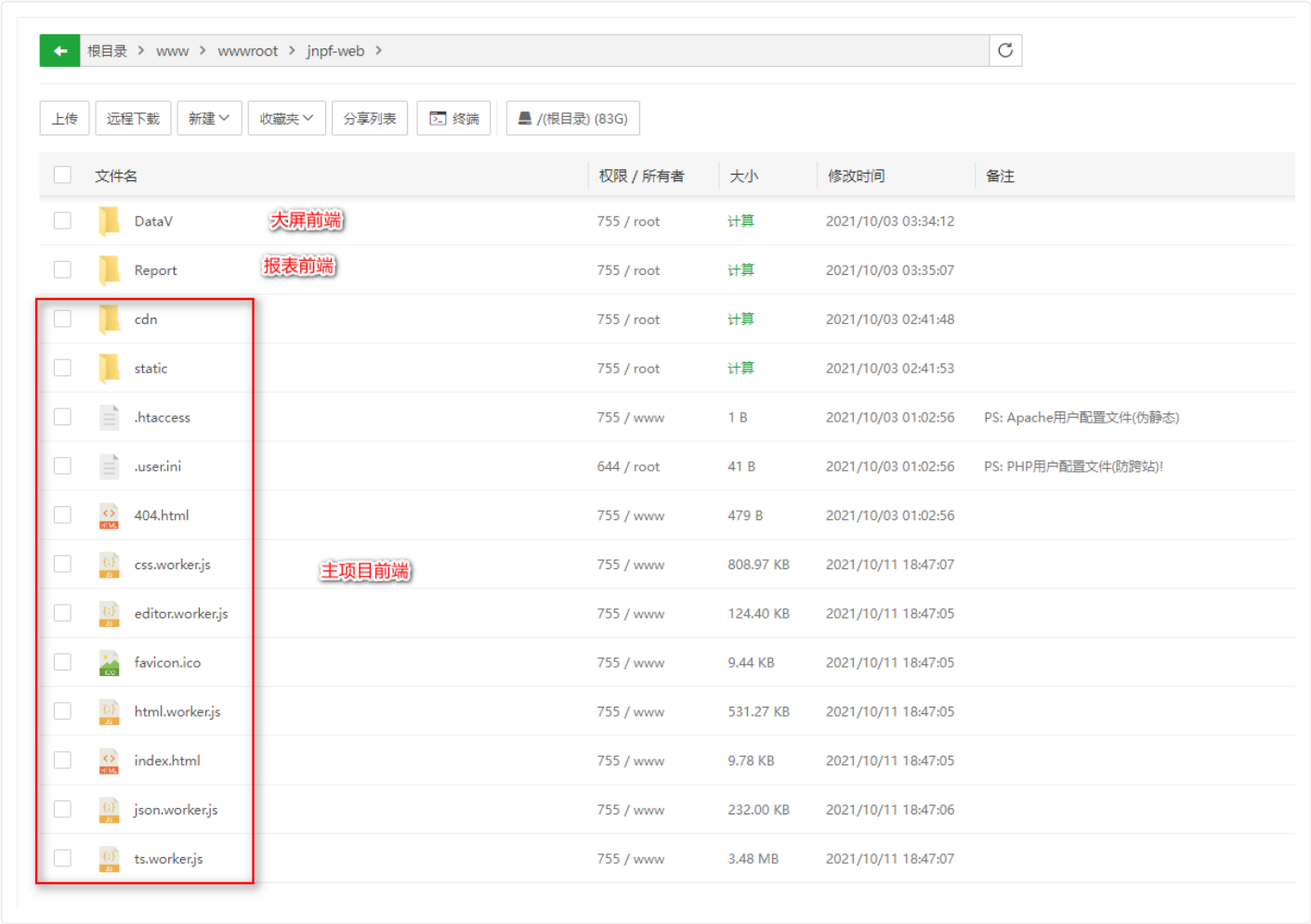


2.中间件目录结构

3.其他服务目录结构



4.前端目录结构



Nginx配置说明

在宝塔中配置

> 以下配置均使用项目默认端口，如有修改按实际情况调整

```
# JNPF-START
#设置上传文件的大小
client_max_body_size 100m;

#添加头部信息
proxy_set_header Cookie $http_cookie;
proxy_set_header X-Forwarded-Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

#请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
```

```
    client_header_buffer_size 128k;

    #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
    large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';

    # 预检命令的缓存，如果不缓存每次会发送两次请求
    add_header Access-Control-Max-Age 3600;

    #带cookie请求需要加上这个字段，并设置为true
    add_header Access-Control-Allow-Credentials true;

    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #    表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;

    # 前端主项目(`jnpf-web`)伪静态
    location / {
      try_files $uri $uri/ /index.html;
    }

    # 前端大屏(`jnpf-web-datascreen`)伪静态
    location /DataV {
      try_files $uri $uri/ /DataV/index.html;
    }

    # 接口
    location /api/ {
      proxy_pass http://localhost:30000;
      proxy_read_timeout 180s;
    }

    # websocket
    location /websocket {
      proxy_pass http://localhost:30000/api/message/websocket;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
      proxy_read_timeout 600s;
    }
```

```
    # 数据报表接口配置
    location /ReportServer/ {
      proxy_pass http://localhost:30000/;
    }

    # 文件预览服务
    location /FileServer {
      proxy_pass http://localhost:30090;
    }

    # 解决文件预览服务无法加载js,css问题
    location ~ /FileServer/*.*\.(js|css)?$ {
       proxy_pass http://localhost:30090;
    }
    # JNPF-END
```

> 特别说明：如果采用宝塔工具部署，需要把以下配置注释

```
    # location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
    # {
    #     expires      30d;
    #     error_log /dev/null;
    #     access_log /dev/null;
    # }
```

单独配置

```
#user  nobody;
worker_processes  1;
worker_rlimit_nofile 65535;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;


events {
    worker_connections  1024;
```

```
}


http {
    include       mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    #access_log  logs/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    #keepalive_timeout  0;
    keepalive_timeout  65;

    gzip on;
        gzip_min_length  1k;
        gzip_buffers     4 16k;
        gzip_http_version 1.1;
        gzip_comp_level 2;
        gzip_types     text/plain application/javascript application/x-javascript te
xt/javascript text/css application/xml;
        gzip_vary on;
        gzip_proxied   expired no-cache no-store private auth;
        gzip_disable   "MSIE [1-6]\.";
        map $http_upgrade $connection_upgrade {
                default upgrade;
                ''      close;
        }

        server {
            listen       80;
           server_name  java.jnpfsoft.com;
                index index.html;
                root C:\wwwroot\java.jnpfsoft.com;


                # JNPF-START

                #设置上传文件的大小
                client_max_body_size 100m;
```

```
        #添加头部信息
        proxy_set_header Cookie $http_cookie;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
        client_header_buffer_size 128k;

        #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
        large_client_header_buffers 4 128k;

        # 前端主项目伪静态
        location / {
          try_files $uri $uri/ /index.html;
        }
        # 前端大屏项目伪静态
        location /DataV {
          try_files $uri $uri/ /DataV/index.html;
        }

        # 请求接口配置
        location /api/ {
                proxy_pass http://localhost:30000;
        }

        # websocket
        location ^~ /websocket {
          proxy_pass http://localhost:30000/api/message/websocket;
          proxy_http_version 1.1;
          proxy_set_header Upgrade $http_upgrade;
          proxy_set_header Connection "upgrade";
          proxy_read_timeout 600s;
        }

        # JNPF-END

        access_log  logs/java.jnpfsoft.com.log;
        error_log  logs/java.jnpfsoft.com.error.log;
    }

}
```

minIO部署文档

> 本文档在 CentOS 7.9 环境下部署minIO

下载安装包

- 方式一：在 `Coding/项目/JNPF开发文档/文件网盘/minIO/` 目录下载安装包并上传至 `/usr/local/minio`
- 方式二：进入目录 `/usr/local/minio`，使用命令下载

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
```

部署

```
chmod +x minio
mkdir -p /home/minio/data

# 开放minio端口:9000
firewall-cmd --zone=public --add-port=30000/tcp --permanent

firewall-cmd --reload

# 方式一：启动minio服务
/usr/local/minio server /home/minio/data/

# 方式二：后台运行minio
nohup /usr/local/minio server /home/minio/data > /home/minio/data/minio.log 2>&1 &
```

官方文档地址：`https://docs.min.io/`

多租户部署文档

一、脚本准备

- `jnpf_init.sql` (初始库脚本)：用于生成租户库后同步数据，文件位于 `jnpf-databae/MySQL/`；
- `JNPF-MySQL.sql` (多租户创库脚本)：用于生成租户库表结构，文件位于 `jnpf-databae/MySQL/多租户/3.2.2/`；
- `jnpf_tenant.sql` (多租户数据库)：用于记录租户信息，文件位于 `jnpf-databae/MySQL/多租户/`；

二、导入数据库脚本

1、新建 `jnpf_init` 数据库，并将 `jnpf_init.sql` 导入（以 `新建查询` 导入）；
2、新建 `jnpf_tenant` 数据库，并将 `jnpf_tenant.sql` 导入
3、新版本增加保护库概念，需要新建JNPF_PROTECT命名的库，需要导入主库中的表“oauth_client_details"，目的是登录前的前端验证登录的合法性

三、修改 `Nacos` 配置

> 配置管理 - 配置列表 - dev

- 修改 `system-config.yaml` 配置，开启多租户，

```
MultiTenancy: true
MultiTenancyUrl: http://127.0.0.1:30000/api/tenant/DbName/
```

- 修改 `tenant.yaml` 配置

```
spring:
# 多租户创库脚本目录（JNPF-MySQL.sql所在路径）
file: /www/wwwroot/jnpf-server/jnpf-tenant
redis:
  database: 1
  host: 127.0.0.1
  port: 6379
  password:
# 数据源
dataType: mysql
datasource:
  # 初始库
  dbinit: jnpf_init
  # 租户库
  dbname: jnpf_tenant
  url: jdbc:mysql://192.168.0.10:3306/{dbName}?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8&nullCatalogMeansCurrent=true
  username: root
  password: 123456
  driver-class-name: com.mysql.cj.jdbc.Driver
```

四、多租户管理前端页面配置

> 打开 jnpf-web-tenant 项目并安装项目依赖

- 开发环境

打开 `src/utils/define.js` ,修改 `开发环境接口配置`

```
// 开发环境接口配置
const APIURl = 'http://192.168.0.26:30000'
```

- 测试环境

编辑根目录下 `.env.staging` 文件

```
# 测试默认配置
ENV = 'staging'
VUE_APP_BASE_API = 'http://192.168.0.25'
```

修改后在命令行运行 `yarn build:staging` 打包发布项目代码

- 生产环境

编辑根目录下 `.env.production` 文件

```
#  生产默认配置
ENV = 'production'
VUE_APP_BASE_API = 'https://tenant.java.jnpfsoft.com'
```

修改后在命令行运行 `yarn build` 打包发布项目代码

五、多租户管理前端页面部署

多租户前端需要单独部署，nginx核心配置如下

```
    #JNPF-Start
    #设置上传文件的大小
    client_max_body_size 100m;

    #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    client_header_buffer_size 128k;

    #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
```

```
    large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';

    # 预检命令的缓存，如果不缓存每次会发送两次请求
    add_header Access-Control-Max-Age 3600;

    #带cookie请求需要加上这个字段，并设置为true
    add_header Access-Control-Allow-Credentials true;

    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #    表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;

    # 前端伪静态配置
    location / {
      try_files $uri $uri/ /index.html;
    }

    # 多租户接口
     location /api/ {
      proxy_pass http://192.168.0.18:30000/api/;
    }

    #JNPF-End
```

主项目前端部署

配置

在项目根目录打开 `.env.staging` (测试环境配置)，部署生产环境请打开 `.env.product` 文件

```
# 测试默认配置
ENV = 'staging'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.10'
# websocket接口
VUE_APP_BASE_WSS = 'ws://192.168.0.10/websocket'
```

```
# 生成环境默认配置
ENV = 'production'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.10'
# websocket接口
VUE_APP_BASE_WSS = 'ws://192.168.0.10/websocket'
```

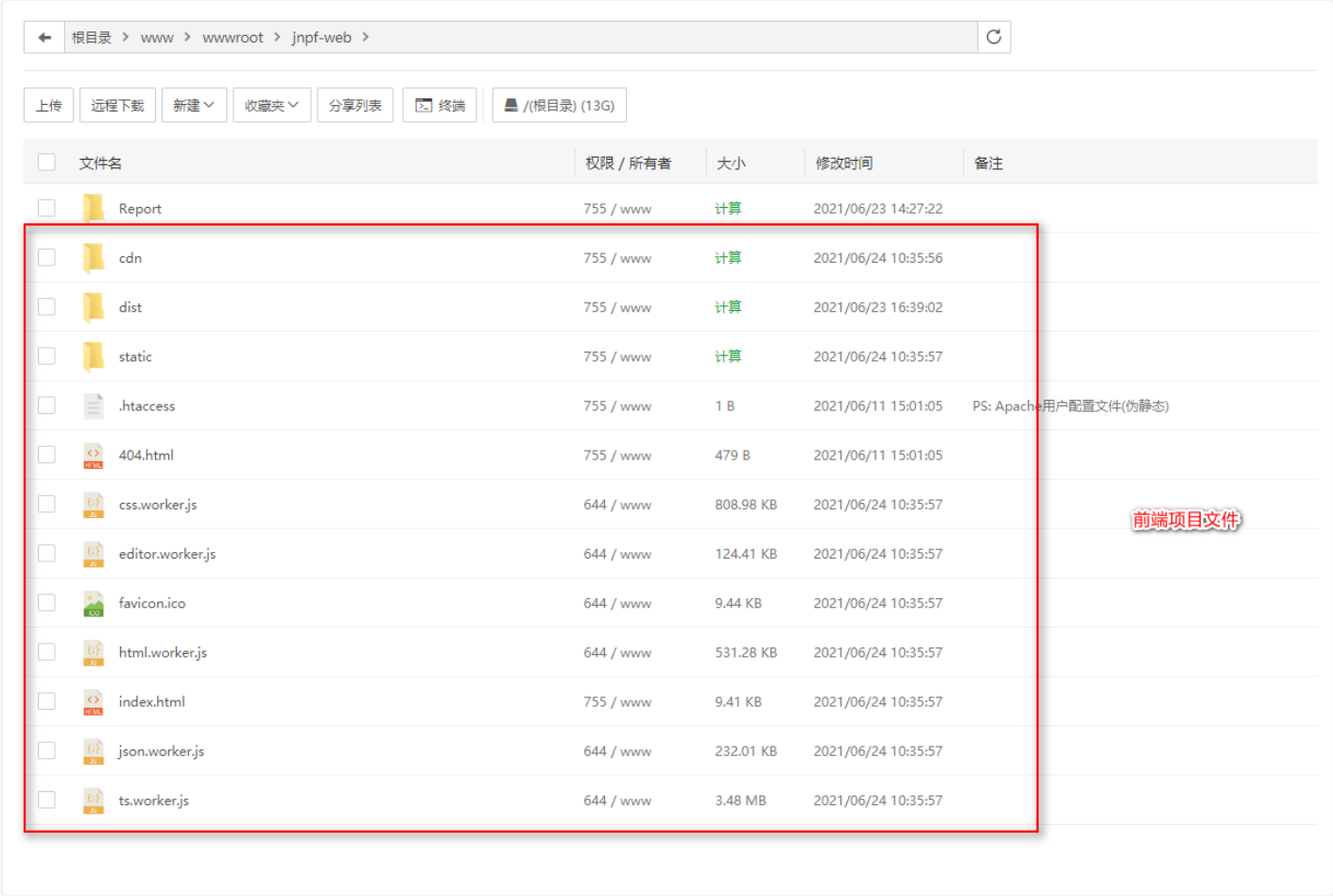
构建

```
# 构建测试环境
npm run build:staging

# 构建生产环境
npm run build
```

发布到服务器

- 方式1：压缩dist目录，上传到服务器/www/wwwroot/jnpf-web并解压
- 方式2：调整构建指令如下

```
npm run build:staging
scp -P 1802 -r  ./dist/* ssh root@192.168.0.10:/www/wwwroot/jnpf-web
```



常见问题

- 访问网站或者域名访问时，头像等无法显示问题
  注释nginx默认配置

```
# location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
    # {
    #     expires       30d;
    #     error_log /dev/null;
    #     access_log /dev/null;
    # }
```

服务部署说明

> 本文基于 Jenkins 自动打包(jar)部署

在服务器上 `/www/wwwroot/gcl-server/` 目录下分别创建各服务目录并在各目录下创建 `startup.sh` (启用脚本)和 `shutdown.sh` (停止脚本)文件，具体配置如下：

> 每个服务的jvm可根据实际情况调整

1. `jnpf-datareport`
   - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx300m -Xms300m -Xmn100m -Xss1024k jnpf-datareport-3.2.4-R
ELEASE.jar >Log.log 2>&1 &
```

   - 停止命令脚本

```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30007 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

2. `jnpf-extend`
   - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx200m -Xms200m -Xmn80m -Xss1024k jnpf-extend-3.2.4-RELEAS
E.jar >Log.log 2>&1 &
```

   - 停止命令脚本

```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30019 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

3. `jnpf-file`
   - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx300m -Xms300m -Xmn120m -Xss1024k jnpf-file-3.2.4-RELEAS
E.jar >Log.log 2>&1 &
```

   - 停止命令脚本

```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30005 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

4. `jnpf-gateway`
   - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx400m -Xms400m -Xmn150m -Xss1024k jnpf-gateway-3.2.4-RELE
```

```
ASE.jar >Log.log 2>&1 &
```

- 停止命令脚本

```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30000 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

5. `jnpf-message`
   - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx200m -Xms200m -Xmn100m -Xss1024k jnpf-message-3.2.4-RELE
ASE.jar >Log.log 2>&1 &
```

   - 停止命令脚本

```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30008 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

6. `jnpf-oauth`
   - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx200m -Xms200m -Xmn80m -Xss1024k jnpf-oauth-3.2.4-RELEAS
E.jar >Log.log 2>&1 &
```

   - 停止命令脚本

```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30001 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

7. `jnpf-permission`
   - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx200m -Xms200m -Xmn100m -Xss1024k jnpf-permission-3.2.4-R
ELEASE.jar >Log.log 2>&1 &
```

   - 停止命令脚本

```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30010 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

8. `jnpf-scheduletask
   - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx1500m -Xms1500m -Xmn500m -Xss1024k jnpf-scheduletask-3.
2.4-RELEASE.jar >Log.log 2>&1 &
```

   - 新版本采用xxl-job服务，打包地址：jnpf-scheduletask\xxl-job-admin\target
   - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx1500m -Xms1500m -Xmn500m -Xss1024k xxl-job-admin-3.4.1-R
ELEASE.jar >Log.log 2>&1 &
```

   - 停止命令脚本

```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30009 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

9. `jnpf-system`
   - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx300m -Xms300m -Xmn150m -Xss1024k jnpf-system-3.2.4-RELEA
SE.jar >Log.log 2>&1 &
```

   - 停止命令脚本

```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30002 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

10. `jnpf-visualdata`
    - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx200m -Xms200m -Xmn80m -Xss1024k jnpf-visualdata-3.2.4-RE
LEASE.jar >Log.log 2>&1 &
```

- 停止命令脚本

```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30011 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

11. `jnpf-visualdev`
    - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx600m -Xms600m -Xmn200m -Xss1024k jnpf-visualdev-3.2.4-RE
LEASE.jar >Log.log 2>&1 &
```

    - 停止命令脚本

```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30003 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

12. `jnpf-workflow`
    - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx300m -Xms300m -Xmn100m -Xss1024k jnpf-workflow-3.2.4-REL
EASE.jar >Log.log 2>&1 &
```

    - 停止命令脚本

```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30004 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

13. `jnpf-tenant`
    - 启动命令脚本

```
#! /bin/bash
nohup java -jar -Xmx300m -Xms300m -Xmn100m -Xss1024k jnpf-tenant-3.2.4-RELEA
SE.jar >Log.log 2>&1 &
```

    - 停止命令脚本

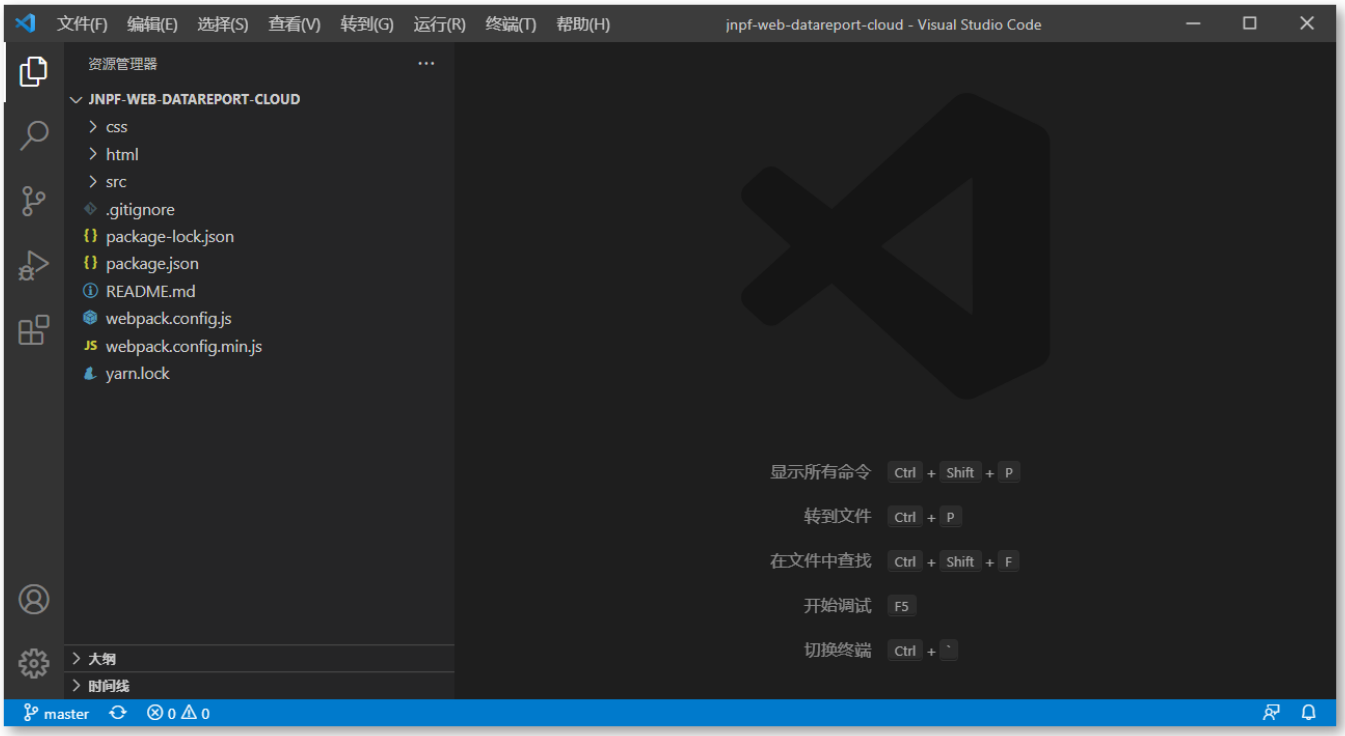
```
#! /bin/bash
kill -9 $(netstat -nlp | grep 30006 | awk '{print $7}' | awk -F"/" '{ print
 $1 }')
```

报表部署

开发环境

前端

1. 导入项目：将 `jnpf-web-datareport-cloud` 拖入 `Visual Studio Code` 编辑器



2.安装依赖及静态服务器，在 `Visual Studio Code` 的终端执行以下命令

```
# 安装依赖
yarn
# 安装Node静态服务器
npm install http-server -g
```

3.接口配置

- 打开 `/html/index.html` ,做如下修改

```
# 在index.html文件中第24行开始
<script type="text/javascript">
  // 报表接口
  window._server = "http://localhost:30000";
  // 报表前端
```

```
  window._contextPath = "http://localhost:8200";
  // 主项目接口地址
  window._mainServer = "http://localhost:30000";
</script>
```

- 打开 `/html/preview.html` ,做如下修改,

```
# 在preview.html文件中第86行
<script type="text/javascript">
  // 报表接口
  window._server = "http://localhost:30000";
</script>
```

  - 在searchform.html文件中修改接口参数:

```
 window._server = "http://localhost:30000"
// 报表前端
window._contextPath = "http://localhost:8200";
```

- 配置说明:
  - `window._server` : 报表接口，对应微服务的网关，本地开发环境默认为 `http://localhost:30000` ;
  - `window._contextPath` : 报表前端，本地开发环境默认为 `http://localhost:8200` ;
  - `window._mainServer` : 主项目接口地址，对应微服务的网关，本地开发环境默认为 `http://localhost:30000`

4.运行报表前端，注：在html目录下执行

```
# 在html目录下执行以下命令
http-server -a localhost -p 8200
```

5.报表前端的开发

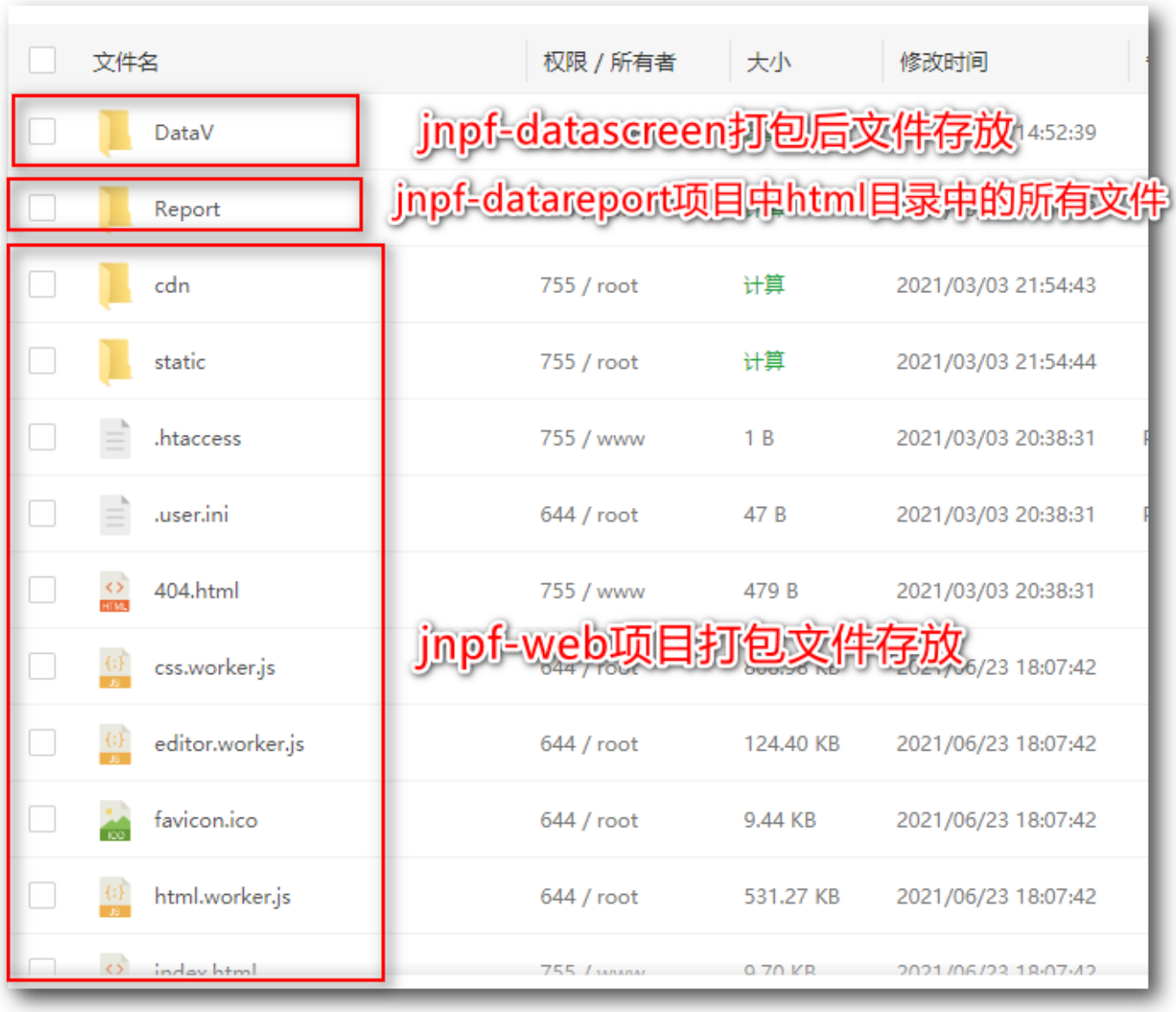
修改 `src` 目录下相关文件后，需要执行打包命令 `yarn build` ,打包的文件是存放在 `html` 文件夹下的 `js` 文件夹中

后端

- 打开 `jnpf-java-cloud` 项目;
- 找到 `jnpf-datareport/ureport2-console/src/main/java/com.bstek.ureport.console/JnpfDataReportApplication` 启动项目即可。

测试生产环境

前端部署结构说明




```
├── jnpf-web                    # 假设这个目录是存放测试或生产环境的前端
│   ├── DataV                   # 大屏(`jnpf-datascreen`)打包后文件存放目录
│   ├── Report                  # 报表(`jnpf-datareport`)html下的文件
│   └── 主项目前端打包后的文件    # 主项目(`jnpf-web`)打包后存放在根目录
```

前端

- 将 `jnpf-web-datareport` 下的 `html` 文件夹中的所有拷贝到 `jnpf-web` 中的 `Report` 目录下，如 `Report` 目录不存在请手动建立
- 接口配置
  - 打开 `/Report/index.html` ,做如下修改

```
# 在index.html文件中第24行开始
<script type="text/javascript">
```

```
// 报表接口
window._server = "http://192.168.0.26/ReportServer";
// 报表前端
window._contextPath = "http://192.168.0.26/Report";
// 主项目接口地址
window._mainServer = "http://192.168.0.26";
</script>
```

- 打开 `/Report/preview.html` ,做如下修改,

```
# 在preview.html文件中第86行
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.26/ReportServer";
</script>
```

- 在searchform.html文件中修改接口参数:

```
 window._server = "http://192.168.0.26/ReportServer"
// 报表前端
window._contextPath = "http://192.168.0.26/Report";
```

- 配置说明:
  - 本示例中，`http://192.168.0.26为项目在测试环境中访问入口，在部署中根据实际情况调整;
  - 报表接口中， `ReportServer` 为虚拟目录，需要在Nginx增加相关配置，具体配置如下:

```
# 数据报表接口配置
  location /ReportServer/ {
     proxy_pass http://localhost:30000/;
  }
```

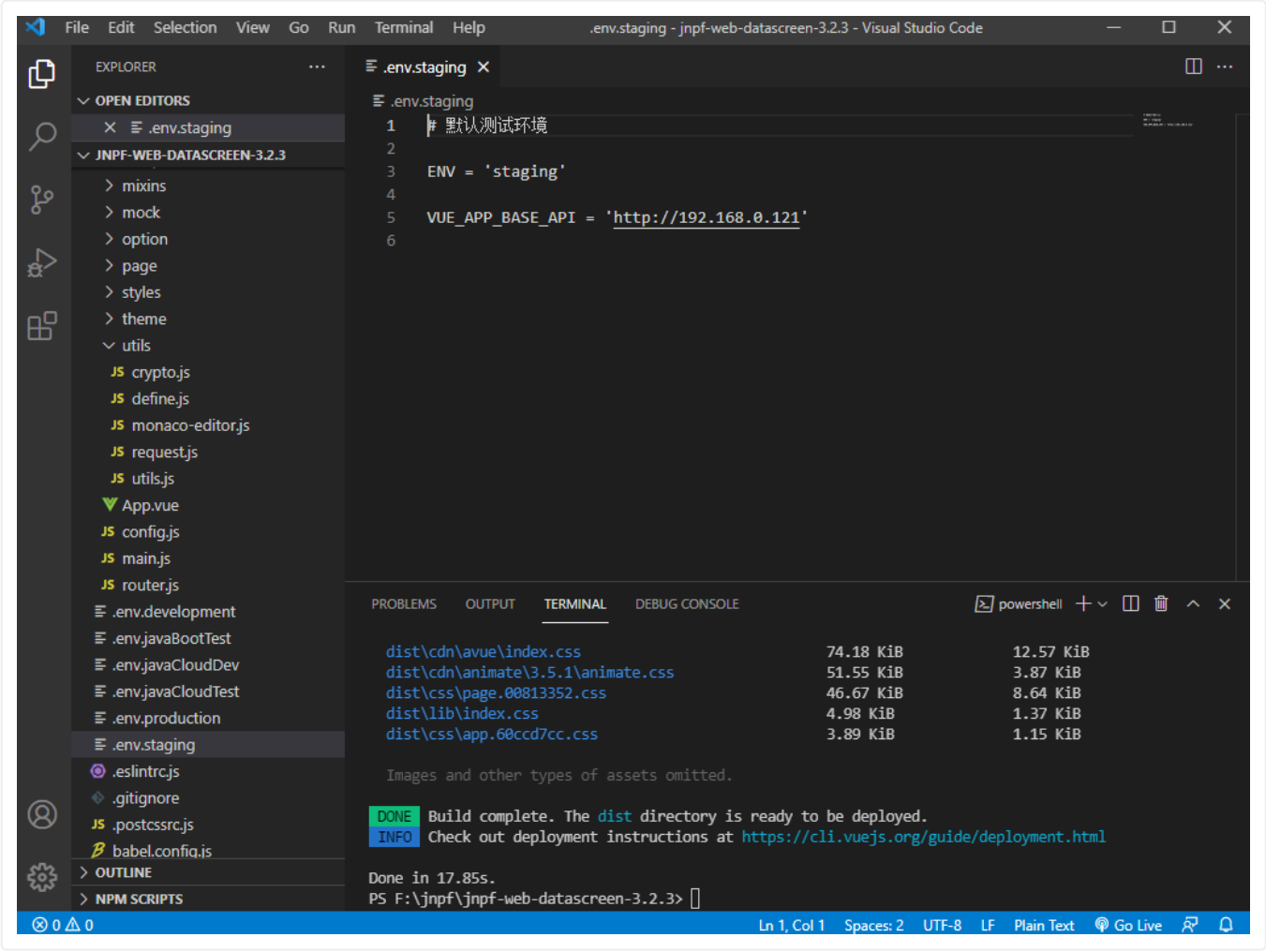
后端

- 部署微服务打包后 `jnpf-datareport-3.x.x.RELEASE.jar` 服务即可。

大屏项目部署

项目打包

1. 打开大屏前端项目 `jnpf-web-datascreen`，修改 `.env.staging` 文件



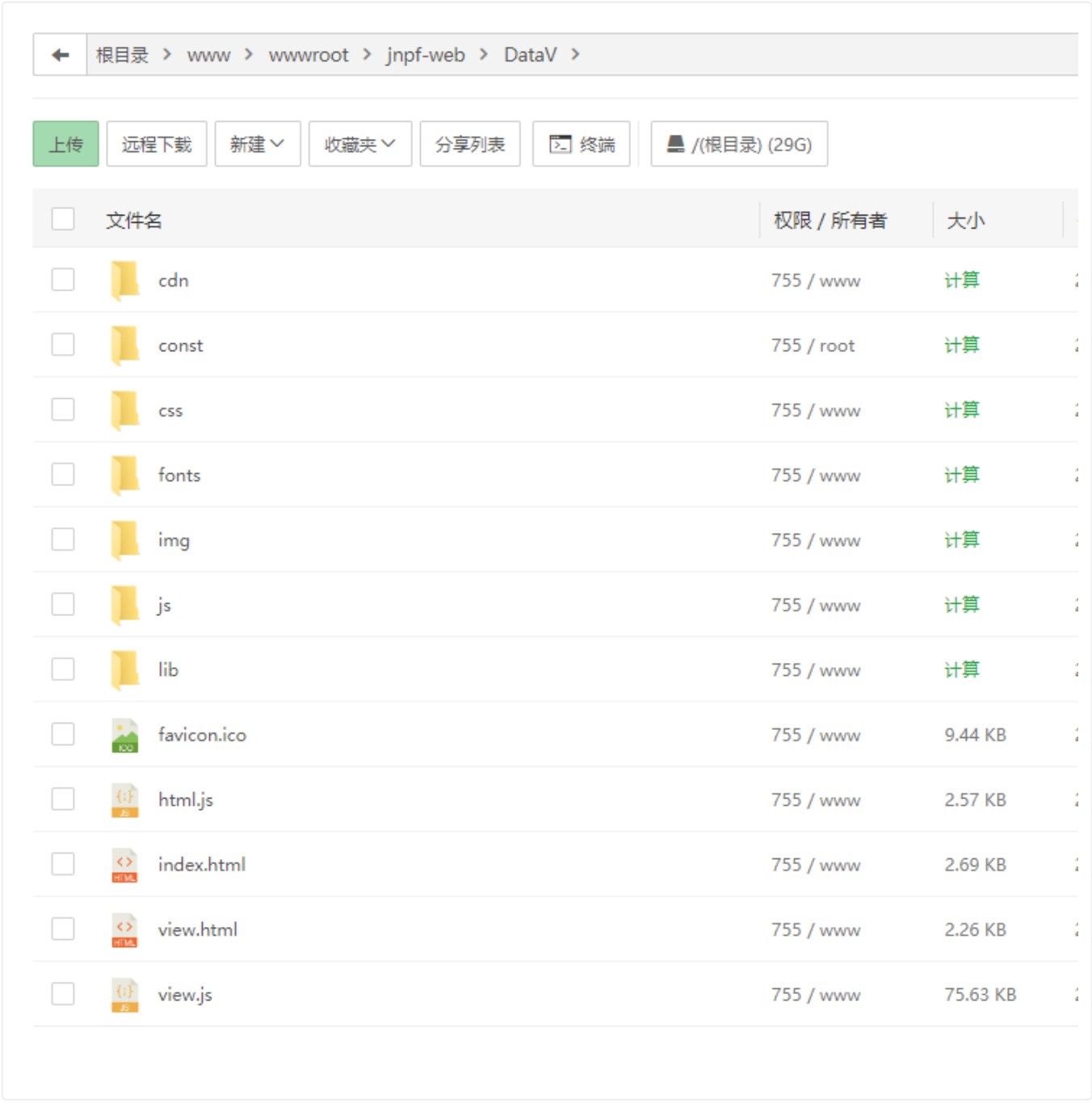
2. 安装依赖

```
yarn
```

3. 打包项目

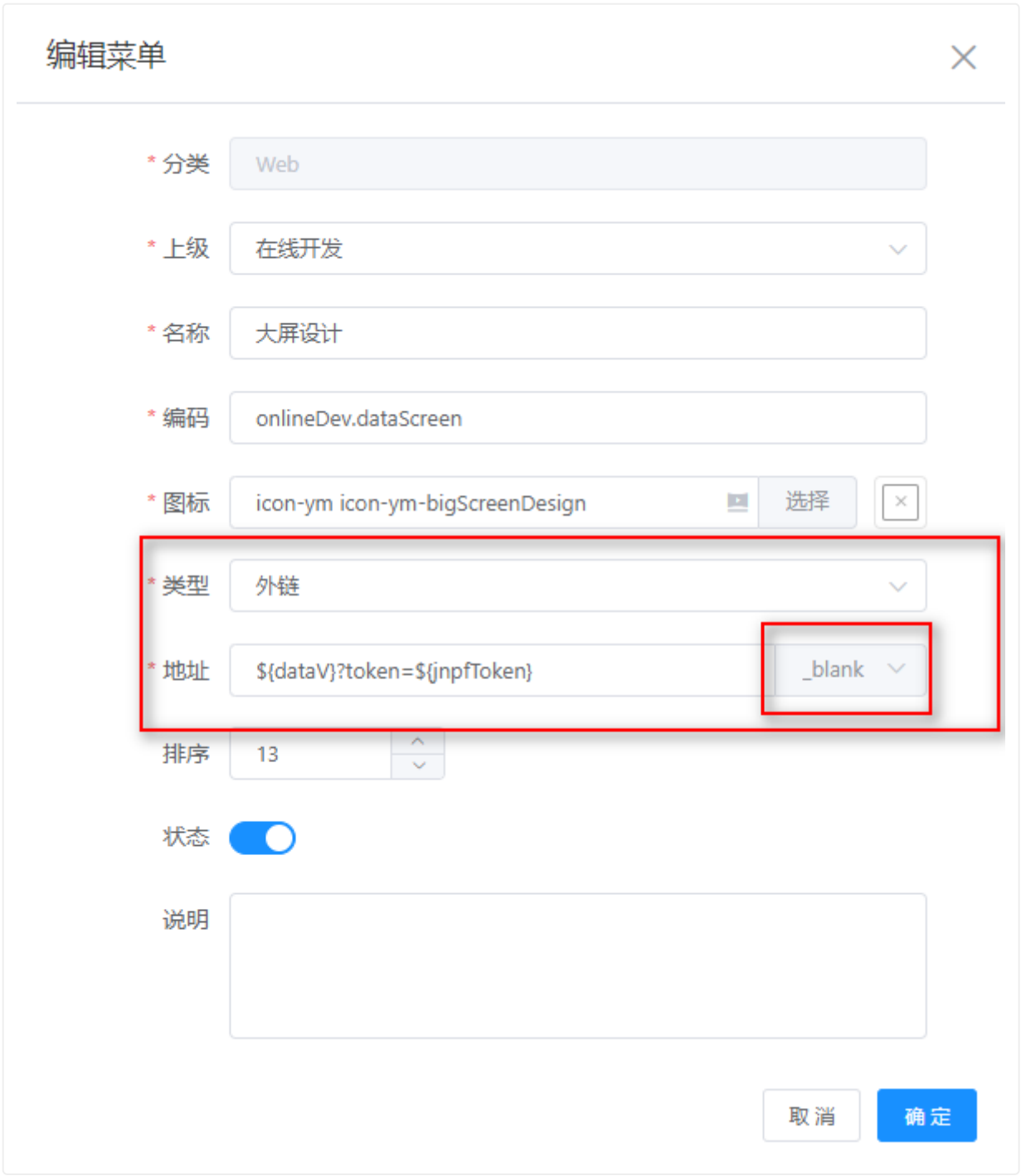
```
yarn build:staging
```

4. 上传dist文件夹全部内容到服务器DataV目录下



5. 登录后修改菜单配置

```
${dataV}?token=${jnpfToken}
```



6.配置nginx

```
# 大屏前端
 location /DataV {
   try_files $uri $uri/ /DataV/index.html;
 }
```

文档预览部署

部署指南

环境要求

- JDK1.8+
- OpenOffice或LiberOffice(Windows下已内置，CentOS或Ubuntu下会自动下载安装，MacOS下需要自行安装)

部署运行

1、开发环境

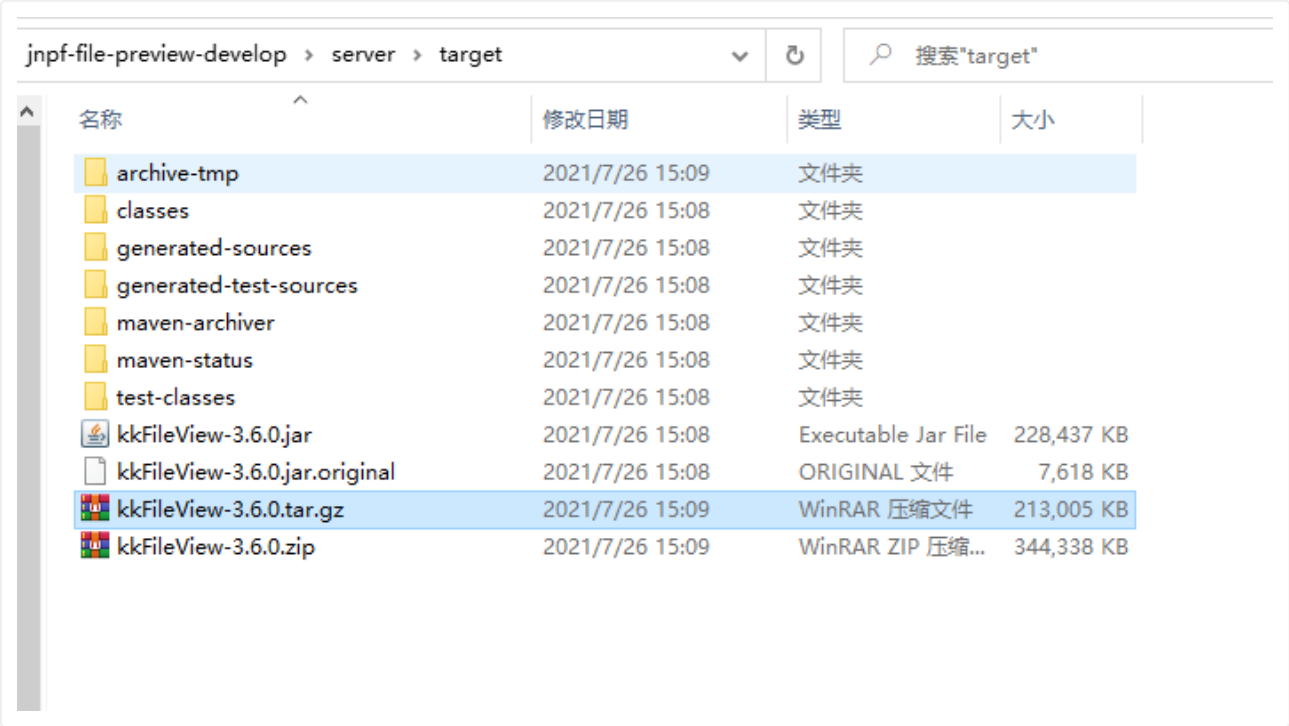
- a.IDEA导入项目
- b.调整配置，打开 `server/src/main/config/application.properties`
- c.启动项目， `server/src/main/java/cn/keking/ServerMain`
- d.打开 `http://localhost:30090` 测试页面

2、测试生产环境

- a.打包

  - 打开 `\server\target` 主要有以下几个文件
  - kkFileView-xxx.jar(一般用于更新)
  - kkFileView-xxx.zip(windows环境下首次部署)

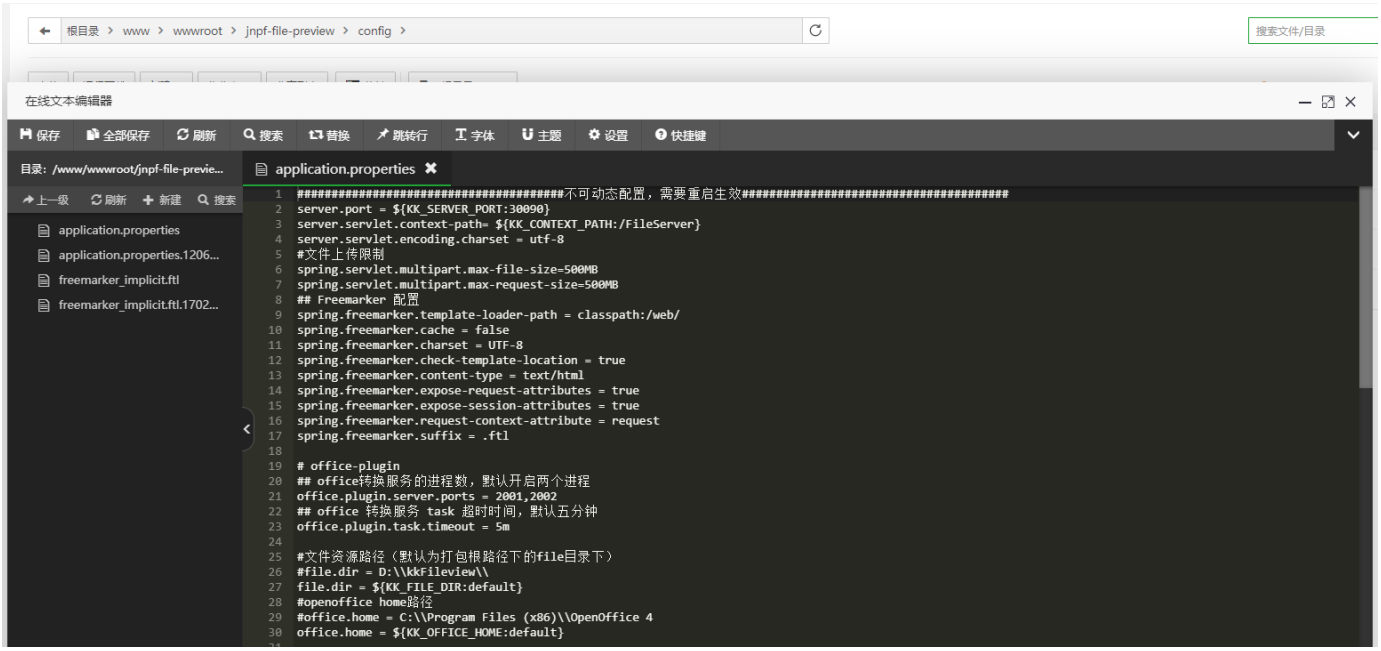
- kkFileView-xxx.tar.gz(Linux环境下首次部署)



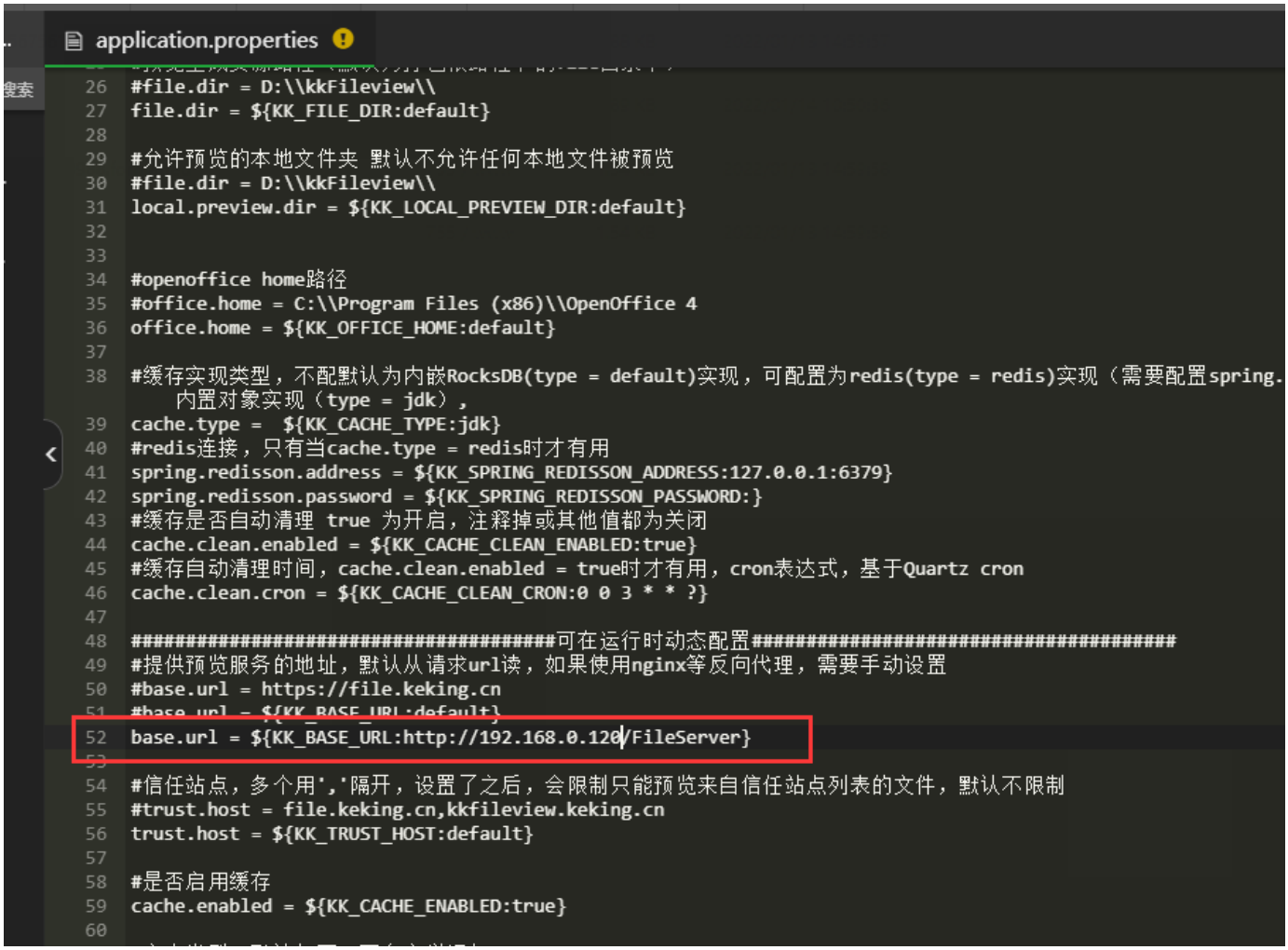
- b.上传至服务器
  - 打开解压后文件夹的bin目录，运行startup脚本(首次部署，之后更新及维护都在bin目录下)

- 修改文件预览配置文件如下两项,打开服务器上的`kkFileView-xxx/config/application.properties`（第3行和第45行，注，新版只需修改第52行：）

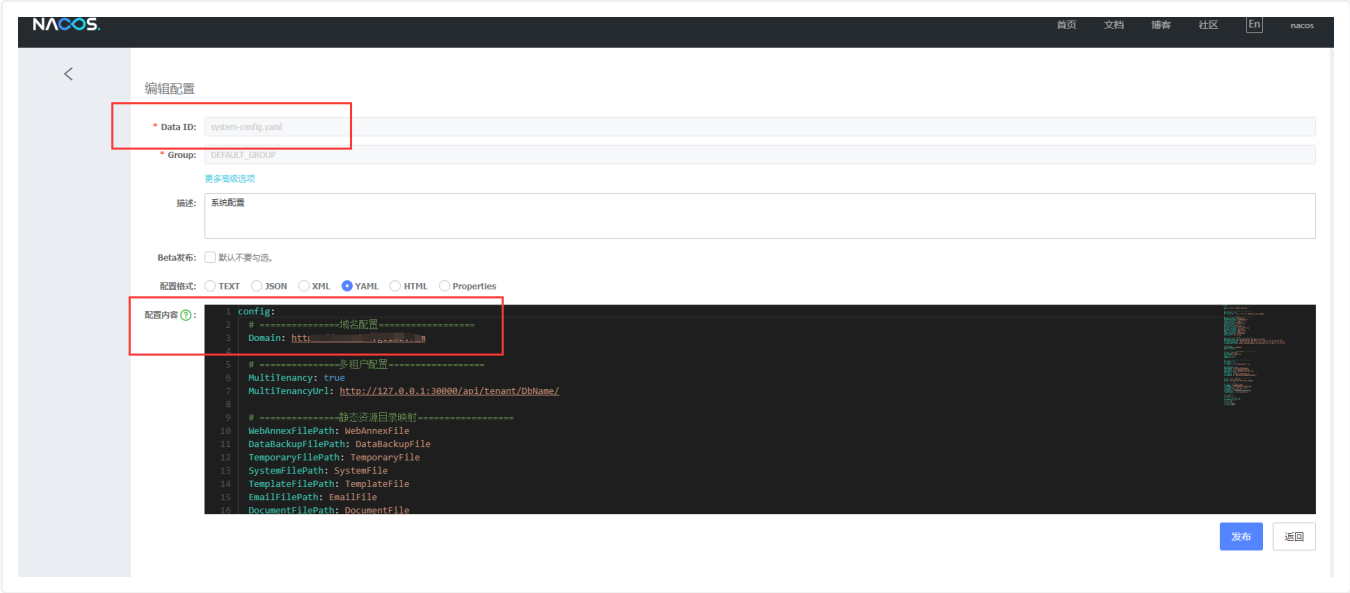
```
修改成:
server.servlet.context-path= ${KK_CONTEXT_PATH:/FileServer}
base.url = ${KK_BASE_URL:http://192.168.0.120/FileServer}
```


根目录 > www > wwwroot > jnpf-file-preview > config >
搜索文件/目录
在线文本编辑器
保存
全部保存
刷新
搜索
替换
跳转行
字体
主题
设置
快捷键
目录：/www/wwwroot/jnpf-file-previe...
application.properties
上一级
刷新
新建
搜索
application.properties
application.properties.1206...
freemarker_implicit.ftl
freemarker_implicit.ftl.1702...
1 ####################################不可动态配置，需要重启生效####################################
2 server.port = ${KK_SERVER_PORT:30090}
3 server.servlet.context-path= ${KK_CONTEXT_PATH:/FileServer}
4 server.servlet.encoding.charset = utf-8
5 #文件上传限制
6 spring.servlet.multipart.max-file-size=500MB
7 spring.servlet.multipart.max-request-size=500MB
8 ## Freemarker 配置
9 spring.freemarker.template-loader-path = classpath:/web/
10 spring.freemarker.cache = false
11 spring.freemarker.charset = UTF-8
12 spring.freemarker.check-template-location = true
13 spring.freemarker.content-type = text/html
14 spring.freemarker.expose-request-attributes = true
15 spring.freemarker.expose-session-attributes = true
16 spring.freemarker.request-context-attribute = request
17 spring.freemarker.suffix = .ftl
18
19 # office-plugin
20 ## office转换服务的进程数，默认开启两个进程
21 office.plugin.server.ports = 2001,2002
22 ## office 转换服务 task 超时时间，默认五分钟
23 office.plugin.task.timeout = 5m
24
25 #文件资源路径（默认为打包根路径下的file目录下）
26 #file.dir = D:\\kkFileview\\
27 file.dir = ${KK_FILE_DIR:default}
28 #openoffice home路径
29 #office.home = C:\\Program Files (x86)\\OpenOffice 4
30 office.home = ${KK_OFFICE_HOME:default}

新版修改位置（第52行）：


application.properties
26 #file.dir = D:\\kkFileview\\
27 file.dir = ${KK_FILE_DIR:default}
28
29 #允许预览的本地文件夹 默认不允许任何本地文件被预览
30 #file.dir = D:\\kkFileview\\
31 local.preview.dir = ${KK_LOCAL_PREVIEW_DIR:default}
32
33
34 #openoffice home路径
35 #office.home = C:\\Program Files (x86)\\OpenOffice 4
36 office.home = ${KK_OFFICE_HOME:default}
37
38 #缓存实现类型，不配默认为内嵌RocksDB(type = default)实现，可配置为redis(type = redis)实现（需要配置spring.
内置对象实现（type = jdk），
39 cache.type = ${KK_CACHE_TYPE:jdk}
40 #redis连接，只有当cache.type = redis时才有用
41 spring.redisson.address = ${KK_SPRING_REDISSON_ADDRESS:127.0.0.1:6379}
42 spring.redisson.password = ${KK_SPRING_REDISSON_PASSWORD:}
43 #缓存是否自动清理 true 为开启，注释掉或其他值都为关闭
44 cache.clean.enabled = ${KK_CACHE_CLEAN_ENABLED:true}
45 #缓存自动清理时间，cache.clean.enabled = true时才有用，cron表达式，基于Quartz cron
46 cache.clean.cron = ${KK_CACHE_CLEAN_CRON:0 0 3 * * ?}
47
48 ##可在运行时动态配置##
49 #提供预览服务的地址，默认从请求url读，如果使用nginx等反向代理，需要手动设置
50 #base.url = https://file.keking.cn
51 #base.url = ${KK_BASE_URL:default}
52 base.url = ${KK_BASE_URL:http://192.168.0.120/FileServer}
53
54 #信任站点，多个用','隔开，设置了之后，会限制只能预览来自信任站点列表的文件，默认不限制
55 #trust.host = file.keking.cn,kkfileview.keking.cn
56 trust.host = ${KK_TRUST_HOST:default}
57
58 #是否启用缓存
59 cache.enabled = ${KK_CACHE_ENABLED:true}
60

- nacos需修改system-config.yaml配置文件为访问地址：



- c.Nginx配置说明
  例如nginx的访问地址为`https://java.jnpfsoft.com`,文件预览部署在内网192.168.0.18服务器上，需要在nginx中添加反向代理如下

```
# 文件预览服务
location /FileServer {
  proxy_pass 192.168.0.18:30090;
}

# 解决文件预览服务无法加载js,css问题
location ~ /FileServer/*.*\.(js|css)?$ {
  proxy_pass 192.168.0.18:30090;
}
```

使用指南

1、单文件预览

```
// 要预览文件的访问地址
let previewUrl = 'http://127.0.0.1:8080/file/test.txt'

// 使用
http://127.0.0.1:30090/onlinePreview?url='+encodeURIComponent(Base64.encode(previewUrl))
```

2、http/https下载流url预览

```
// 要预览文件的访问地址
let previewUrl = 'http://127.0.0.1:8080/filedownload?fileId=1'

// 使用
http://127.0.0.1:30090/onlinePreview?url='+encodeURIComponent(Base64.encode(previewUrl))
```

3、更多参考官方文档（`https://kkfileview.keking.cn/zh-cn/docs/usage.html`）

常见问题

1、预览乱码(字体问题)

大部分Linux系统上并没有预装中文字体或字体不全，需要把常用字体拷贝到Linux服务器上，具体操作如下： 下载如下字体包 `http://kkfileview.keking.cn/fonts.zip` 文件解压完整拷贝到Linux下的 `/usr/share/fonts` 目录。然后依次执行 `mkfontscale`、`mkfontdir`、`fc-cache` 使字体生效

> 特别说明：安装字体前确保Linux服务器已安装mkfontscale、fontconfig，如未安装运行以下命令

CentOS运行如下命令

```
# 使mkfontscale和mkfontdir命令正常运行
yum install mkfontscale

# 使fc-cache命令正常运行
yum install fontconfig
```

Ubuntu运行如下命令

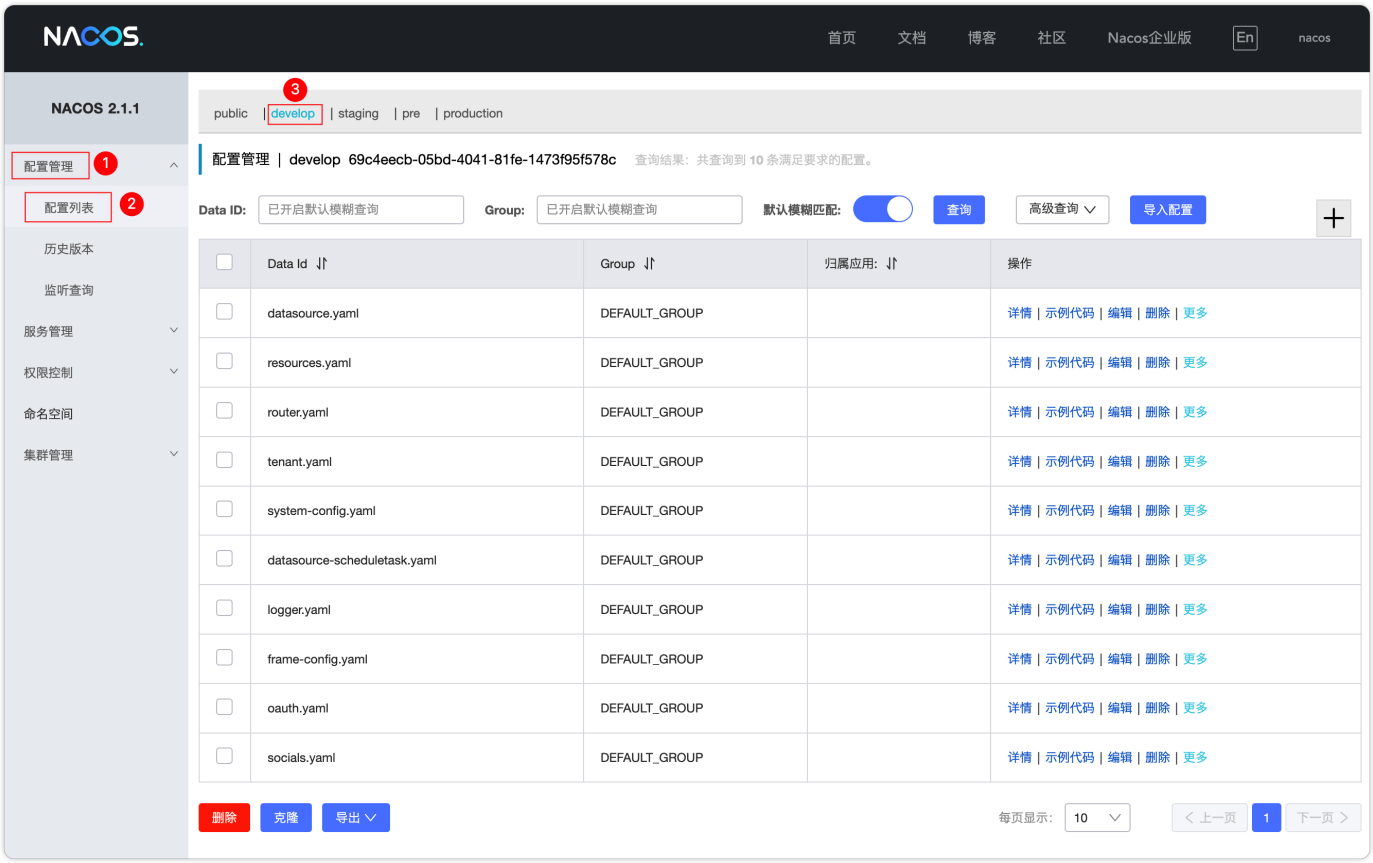
```
# 使mkfontscale和mkfontdir命令正常运行
sudo apt-get install ttf-mscorefonts-installer

# 使fc-cache命令正常运行
sudo apt-get install fontconfig
```

2、更多问题请参考官方文档说明(`https://kkfileview.keking.cn/zh-cn/docs/faq.html`)

开发指南

Nacos配置说明

登录 `Nacos` 控制台， `配置管理` - `配置列表` - `develop` ,可以看到如下配置列表



Data Id	说明
datasource.yaml	数据源及Redis配置
resources.yaml	静态资源配置
router.yaml	网关路由配置
tenant.yaml	多租户配置
system-config.yaml	系统基础配置
datasource-scheduletask.yaml	任务调度数据源配置
logger.yaml	日志配置
frame-config.yaml	框架配置
oauth.yaml	单点登录(SSO)配置
socials.yaml	第三方登录配置

datasource.yaml

> 数据源及Redis配置

```
spring:
  # redis单机模式
  redis:
    #缓存库编号
    database: 1
    host: 127.0.0.1
    port: 6379
    password: 123456 # 密码为空时，请将本行注释
    timeout: 3000 #超时时间(单位：秒)
    lettuce: #Lettuce为Redis的Java驱动包
      pool:
        max-active: 8 # 连接池最大连接数
        max-wait: -1ms  # 连接池最大阻塞等待时间（使用负值表示没有限制）
        min-idle: 0 # 连接池中的最小空闲连接
        max-idle: 8 # 连接池中的最大空闲连接

  # redis集群模式
  # redis:
  #   cluster:
  #     nodes:
  #       - 192.168.0.225:6380
  #       - 192.168.0.225:6381
  #       - 192.168.0.225:6382
  #       - 192.168.0.225:6383
  #       - 192.168.0.225:6384
  #       - 192.168.0.225:6385
  #   password: 123456 # 密码为空时，请将本行注释
  #   timeout: 3000 # 超时时间(单位：秒)
  #   lettuce: #Lettuce为Redis的Java驱动包
  #     pool:
  #       max-active: 8 # 连接池最大连接数
  #       max-wait: -1ms  # 连接池最大阻塞等待时间（使用负值表示没有限制）
  #       min-idle: 0 # 连接池中的最小空闲连接
  #       max-idle: 8 # 连接池中的最大空闲连接

  # 数据源
  datasource:
    # 数据库类型(可选值 MySQL、SQLServer、Oracle、DM、KingbaseES、PostgreSQL，请严格按可选值
填写)
    db-type: MySQL
    db-name: jnpf_test_333
    host: 192.168.0.10
```

```
    port: 3306
    username: jnpf_test_333
    password: fmKnaWWnJDPxtsyB
    db-schema: #金仓达梦选填
    prepare-url: #自定义连接串

    # 多数据源配置
    dynamic:
      seata: false
      # 设置默认的数据源或者数据源组,默认值即为master
      primary: master
      # 严格匹配数据源,默认false. true未匹配到指定数据源时抛异常,false使用默认数据源
      strict: true
      # datasource:
      #    # 额外的数据源 Service类中使用@DS动态切换
      #   SQLSERVER11:
      #     url: jdbc:mysql://192.168.0.30:3306/jnpf_project?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8
      #     username: jnpf_project
      #     password: DsNnnYHscifFDfEZ
      #     driver-class-name: com.mysql.cj.jdbc.Driver

# swagger开关
swagger:
  enable: true
```

配置说明

一 Redis配置

Redis在平台上的使用场景：平台基础配置、用户登录信息、工作流延迟任务、超时任务等

Redis配置提供了两种配置：**单机配置**和**集群配置**，来满足不同的业务场景需求，具体配置说明如下

1.1 Redis单机模式

> 默认配置采用单机模式

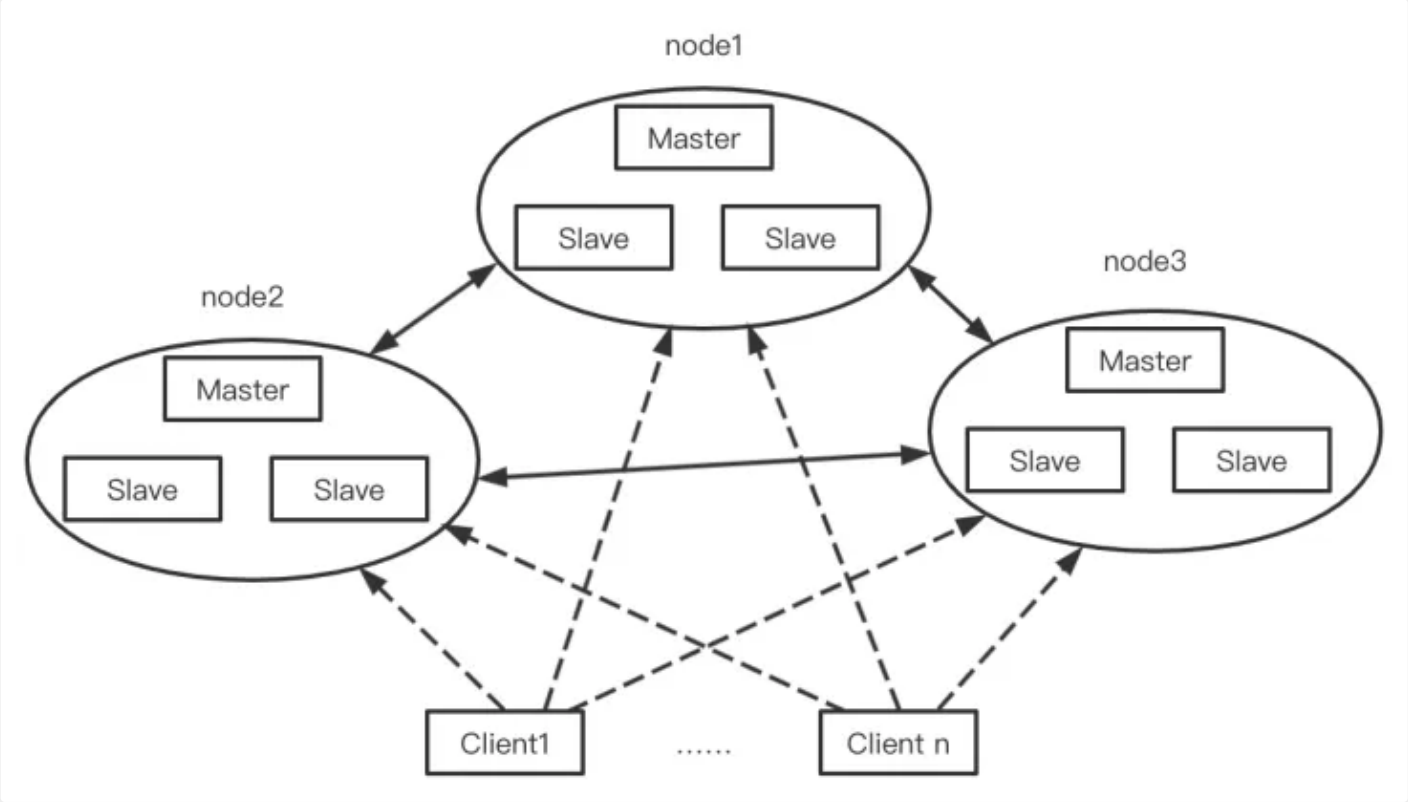
配置如下：

```
redis:
    #缓存库编号
```

```
    database: 1
    host: 127.0.0.1
    port: 6379
    password: 123456 # 密码为空时，请将本行注释
    timeout: 3000 #超时时间(单位：秒)
    lettuce: #Lettuce为Redis的Java驱动包
      pool:
        max-active: 8 # 连接池最大连接数
        max-wait: -1ms  # 连接池最大阻塞等待时间（使用负值表示没有限制）
        min-idle: 0 # 连接池中的最小空闲连接
        max-idle: 8 # 连接池中的最大空闲连接
```

1.2 Redis集群模式

配置中是采用三主三从的集群模式



二 数据源配置

数据源配置设计之初始是为了简化配置，在传统的配置方式上进行了封装，可以打开 `jnpf-common` 项目中的 `jnpf-common/jnpf-common-database/src/main/java/jnpf.database/source/impl/` 下查看相关数据库默认的数据库连接串

MySQL

> DbMySQL.java

```
protected void init(){
        setInstance(
                DbBase.MYSQL,
                DbType.MYSQL,
                "3306",
                "mysql",
                "com.mysql.cj.jdbc.Driver",
                "jdbc:mysql://{host}:{port}/{dbname}?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&serverTimezone=GMT%2B8&useSSL=false"
        );
    }
```

SQLServer

> DbSQLServer.java

```
protected void init() {
        setInstance(
                DbBase.SQL_SERVER,
                DbType.SQL_SERVER,
                "1433",
                "sqlserver",
                "com.microsoft.sqlserver.jdbc.SQLServerDriver",
                "jdbc:sqlserver://{host}:{port};databaseName={dbname}");
 }
```

Oracle

> DbOracle.java

```
protected void init() {
        setInstance(
                DbBase.ORACLE,
                DbType.ORACLE,
                "1521",
                "oracle",
                "oracle.jdbc.OracleDriver",
                "");
    }
```

PostgreSQL

DbPostgre.java

```
protected void init() {
        setInstance(
                DbBase.POSTGRE_SQL,
                DbType.POSTGRE_SQL,
                "5432",
                "postgresql",
                "org.postgresql.Driver",
                "jdbc:postgresql://{host}:{port}/{dbname}");
    }
```

DM

DbDM.java

```
protected void init() {
        setInstance(
                DbBase.DM,
                DbType.DM,
                "5236",
                "dm",
                "dm.jdbc.driver.DmDriver",
                "jdbc:dm://{host}:{port}/{schema}");
    }
```

KingbaseES

DbKingbase.java

```
protected void init() {
        setInstance(
                DbBase.KINGBASE_ES,
                DbType.KINGBASE_ES,
                "54321",
                "kingbase8",
                "com.kingbase8.Driver",
                "jdbc:kingbase8://{host}:{port}/{dbname}?currentSchema={schema}");
    }
```

在配置上需要注意的是：

- `db-type`：可选MySQL(默认)、SQLServer、Oracle、DM、KingbaseES、PostgreSQL
- `db-schema`：模式，数据库如果是人大金仓数据或达梦数据库时需要填写
- `prepare-url`：数据库连接串，如 jdbc:mysql://192.168.0.10:3306/xxl_job_dev?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&serverTimezone=GMT%2B8&useSSL=false

三 多数据源配置

多数据源的配置和使用可以参考：https://baomidou.com/pages/a61e1b/

swagger开关

开启时，可通过访问 `http://{ip}:30000/swagger-ui/` 查看API接口

resources.yaml

> 静态资源配置

```
config:
  TestVersion: false
  # 文件存储配置
  file-storage: #文件存储配置，不使用的情况下可以不写
    default-platform: local-plus-1 #默认使用的存储平台
    thumbnail-suffix: ".min.jpg" #缩略图后缀，例如【.min.jpg】【.png】
    local-plus: # 本地存储升级版
      - platform: local-plus-1 # 存储平台标识
        enable-storage: true  #启用存储
        enable-access: true #启用访问（线上请使用 Nginx 配置，效率更高）
        domain: "" # 访问域名，例如：“http://127.0.0.1:8030/”，注意后面要和 path-patterns 保持一致，“/”结尾，本地存储建议使用相对路径，方便后期更换域名
        base-path: /www/wwwroot/jnpf-resources/ # 基础路径
        path-patterns: /** # 访问路径
        storage-path:  # 存储路径
    aliyun-oss: # 阿里云 OSS ，不使用的情况下可以不写
      - platform: aliyun-oss-1 # 存储平台标识
        enable-storage: true  # 启用存储
        access-key: ??
        secret-key: ??
        end-point: ??
        bucket-name: ??
```

```
        domain: ?? # 访问域名，注意“/”结尾，例如：https://abc.oss-cn-shanghai.aliyuncs.com/
        base-path:  # 基础路径
    qiniu-kodo: # 七牛云 kodo ，不使用的情况下可以不写
      - platform: qiniu-kodo-1 # 存储平台标识
        enable-storage: true  # 启用存储
        access-key: ??
        secret-key: ??
        bucket-name: ??
        domain: ?? # 访问域名，注意“/”结尾，例如：http://abc.hn-bkt.clouddn.com/
        base-path:  # 基础路径
    tencent-cos: # 腾讯云 COS
      - platform: tencent-cos-1 # 存储平台标识
        enable-storage: true  # 启用存储
        secret-id: ??
        secret-key: ??
        region: ?? #存仓库所在地域
        bucket-name: ??
        domain: ?? # 访问域名，注意“/”结尾，例如：https://abc.cos.ap-nanjing.myqcloud.com/
        base-path: hy/ # 基础路径
    minio: # MinIO，由于 MinIO SDK 支持 AWS S3，其它兼容 AWS S3 协议的存储平台也都可配置在这里
      - platform: minio-1 # 存储平台标识
        enable-storage: true  # 启用存储
        access-key: minioadmin
        secret-key: wNN8V4unEzVsIc5w
        end-point: http://192.168.0.207:9000/ #服务端地址
        bucket-name: jnpfsoftoss
        domain: # 访问域名，注意“/”结尾，例如：http://minio.abc.com/abc/
        base-path:  # 基础路径
```

配置说明

文件存储配置

v3.4.3开始重构了文件存储，兼容更多平台且配置更灵活

存储平台

> 兼容国内大多数对象存储平台

router.yaml

> 网关路由配置

```
spring:
  cloud:
    gateway:
      routes:
        # 认证中心
        - id: jnpf-oauth
          uri: lb://jnpf-oauth
          predicates:
            - Path=/api/oauth/**
          filters:
            - StripPrefix=2
        # 系统基础服务
        - id: jnpf-system
          uri: lb://jnpf-system
          predicates:
            - Path=/api/system/**
          filters:
            - StripPrefix=2
        # 示例
        - id: jnpf-example
          uri: lb://jnpf-example
          predicates:
            - Path=/api/example/**
          filters:
            - StripPrefix=2
        # 扩展服务
        - id: jnpf-extend
          uri: lb://jnpf-extend
          predicates:
            - Path=/api/extend/**
          filters:
            - StripPrefix=2
        # 报表服务
        - id: jnpf-datareport
          uri: lb://jnpf-datareport
          predicates:
            - Path=/api/datareport/**
          filters:
            - StripPrefix=2
```

```
        # 可视化开发服务
        - id: jnpf-visualdev
          uri: lb://jnpf-visualdev
          predicates:
            - Path=/api/visualdev/**
          filters:
            - StripPrefix=2
        # 工作流服务
        - id: jnpf-workflow
          uri: lb://jnpf-workflow
          predicates:
            - Path=/api/workflow/**
          filters:
            - StripPrefix=2
        # 文件服务
        - id: jnpf-file
          uri: lb://jnpf-file
          predicates:
            - Path=/api/file/**
          filters:
            - StripPrefix=2
        # 多租户
        - id: jnpf-tenant
          uri: lb://jnpf-tenant
          predicates:
            - Path=/api/tenant/**
          filters:
            - StripPrefix=2
        # 消息
        - id: jnpf-message
          uri: lb://jnpf-message
          predicates:
            - Path=/api/message/**
          filters:
            - StripPrefix=2
        # 任务调度
        - id: jnpf-scheduletask
          uri: lb://jnpf-scheduletask
          predicates:
            - Path=/api/scheduletask/**
          filters:
            - StripPrefix=2
        # 权限
        - id: jnpf-permission
          uri: lb://jnpf-permission
```

```
            predicates:
              - Path=/api/permission/**
            filters:
              - StripPrefix=2
          # 大屏
          - id: jnpf-visualdata
            uri: lb://jnpf-visualdata
            predicates:
              - Path=/api/blade-visual/**
            filters:
              - StripPrefix=2
          # app
          - id: jnpf-app
            uri: lb://jnpf-app
            predicates:
              - Path=/api/app/**
            filters:
              - StripPrefix=2
#接口放行地址 与GatewayWhite中的默认URL合并
gateway:
  #禁止访问
  block-url:
  #- /路径1
  #- /路径2
  #不验证Token, 记录访问
  white-url:
  #放行不记录
  exclude-url:
```

tenant.yaml

> 多租户配置

```
spring:
  # 多租户创库脚本目录
  file: D:\cloud\jnpf\tenant\jnpf-java-tenant\
  redis:
    database: 8
    host: 127.0.0.1
    port: 6379
  # 数据源
  datasource:
```

```
    dbtype: MySQL
    dbname: jnpf_tenant_341
    dbinit: java_init_342
    host: 192.168.0.10
    port: 3306
    username: root
    password: 88b91610c7c9b987
    # 表空间(当数据库为Oracle、达梦DM8、金仓KingbaseES时表空间必须指定，其他数据库为空即可)
    tablespace:
  #maxkey租户同步api接口相关配置
  sso:
    enabled: true
    url: http://localhost:8526/sso-mgt-api
    appId: 745057899234983936
    secret: r12FMTQwNzIwMjIyMDM1MTEzMTUzoh
```

system-config.yaml

> 系统基础配置

```
config:
  # ===============域名配置=================
  #后端域名带端口
  Domain: http://javacloud.release.jnpf.work
  #前端域名带端口(消息跳转使用)
  frontDomain: http://javacloud.release.jnpf.work
  #H5域名带端口(消息跳转使用)
  appDomain: http://javacloud.release.jnpf.work


  # ===============多租户配置=================
  MultiTenancy: false
  MultiTenancyUrl: http://127.0.0.1:30000/api/tenant/DbName/
  #COLUMN、SCHEMA模式
  MultiTenantType: SCHEMA

  # ===============静态资源目录映射=================
  WebAnnexFilePath: WebAnnexFile
  DataBackupFilePath: DataBackupFile
  TemporaryFilePath: TemporaryFile
  SystemFilePath: SystemFile
  TemplateFilePath: TemplateFile
```

```
  EmailFilePath: EmailFile
  DocumentFilePath: DocumentFile
  DocumentPreviewPath: DocumentPreview
  UserAvatarFilePath: UserAvatar
  IMContentFilePath: IMContentFile
  MPMaterialFilePath: MPMaterial
  TemplateCodePath: TemplateCode
  BiVisualPath: BiVisualPath

  # ===============功能格式限制==================
  MPUploadFileType: bmp,png,jpeg,jpg,gif,mp3,wma,wav,amr,mp4
  WeChatUploadFileType: jpg,png,doc,docx,ppt,pptx,xls,xlsx,pdf,txt,rar,zip,csv,amr,mp4
  AllowUploadImageType: jpg,gif,png,bmp,jpeg,tiff,psd,swf,svg,pcx,dxf,wmf,emf,lic,eps,tga
  AllowUploadFileType: jpg,gif,png,bmp,jpeg,doc,docx,ppt,pptx,xls,xlsx,pdf,txt,rar,zip,csv

  AllowPreviewFileType: doc,docx,xls,xlsx,ppt,pptx,pdf #允许预览文件类型
  PreviewType: kkfile #文件预览方式 （1.yozo 2.kkfile）默认使用kk
  kkFileUrl: http://javacloud.release.jnpf.work/FileServer/ #kk文件预览服务地址

  # 代码生成器命名空间
  CodeAreasName: example
  WebDirectoryPath:

  # ===============系统错误邮件报告反馈相关==================
  SoftName: JNPF.JAVA
  SoftFullName: JNPF软件开发平台
  AppVersion: V3.3.3
  AppUpdateContent: ;

  # ====================================
  RecordLog: true
  ErrorReport: false
  ErrorReportTo: surrpot@yinmaisoft.com

  IgexinEnabled: true
  IgexinAppid: HLFY9T2d1z7MySY8hwGwh4
  IgexinAppkey: 6Uiduugq648YDChhCjAt59
  IgexinMastersecret: pEyQm156SJ9iS7PbyjLCZ6
  SoftVersion: V3.3.3
  AccessKeyId: LTAI4Fu4iJnKmDu5QG78ibfa
  AccessKeySecret: pMSYlgOz3Tm6sYqWA8sKiyxgNY97J7
```

```
  # ===============跨域配置=================
  Origins: http://127.0.0.1
  Methods: GET,HEAD,POST,PUT,DELETE,OPTIONS

  # =========永中配置文件=========
  YozoAppId: yozoAgR41jgC0062
  YozoAppKey: fc3134a9ba8bc6f4c69d635f9adf
  YozoCloudDomain: //dmc.yozocloud.cn
  YozoDomain: //dcsapi.com/
  YozoDomainKey: 5746225028446289934 4489
  YozoEditDomain: //eic.yozocloud.cn

  #====================== 接口鉴权 ======================
  # 开启接口鉴权
  EnablePreAuth: true
  # 请求来源验证
  EnableInnerAuth: true
```

datasource-scheduletask.yaml

> 任务调度客户端数据源配置

```
config:
  # 忽略多租户配置
  MutiTenancy: false

# ====================== 任务调度配置 ======================
xxl:
  job:
    accessToken: ''
    i18n: zh_CN
    logretentiondays: 30
    triggerpool:
      fast:
        max: 200
      slow:
        max: 100
    # xxl-job服务端地址
    admin:
      addresses: http://127.0.0.1:30020/xxl-job-admin/
    executor:
      address: ''
```

```
      appname: xxl-job-executor-sample1
      ip: ''
      logpath: /data/applogs/xxl-job/jobhandler
      logretentiondays: 30
      port: 9999
  # rest调用xxl-job接口地址
  admin:
    register:
      handle-query-address: ${xxl.job.admin.addresses}api/handler/queryList
      job-info-address: ${xxl.job.admin.addresses}api/jobinfo
      log-query-address: ${xxl.job.admin.addresses}api/log
      task-list-address: ${xxl.job.admin.addresses}api/ScheduleTask/List
      task-info-address: ${xxl.job.admin.addresses}api/ScheduleTask/getInfo
      task-save-address: ${xxl.job.admin.addresses}api/ScheduleTask
      task-update-address: ${xxl.job.admin.addresses}api/ScheduleTask
      task-remove-address: ${xxl.job.admin.addresses}api/ScheduleTask/remove
      task-start-or-remove-address: ${xxl.job.admin.addresses}api/ScheduleTask/updat
eTask

spring:
  # Redis配置
  redis:
    database: 69
    host: 192.168.0.210
    port: 6379
    password: 123456
  # 数据源
  datasource:
    # 暂只支持MySQL
    dbtype: MySQL
    host: 192.168.0.210
    port: 3306
    username: xxl_job_test
    password: CPDXGse8pYLCsaDb
    db-name: xxl_job_test
    db-schema: #金仓达梦选填
    prepare-url: #自定义url
# swagger开关
swagger:
  enable: true
```

logger.yaml

日志配置

```
logging:
  config: classpath:logback-spring.xml
  level: #自定义第三方包名日志等级
    com.alibaba.nacos: ERROR
    com.alibaba.cloud.nacos: ERROR
    io.seata: ERROR
    org.apache.dubbo: ERROR
    # 解除注释后Druid连接池打印SQL语句
    druid.sql.Statement: debug
    #druid.sql.ResultSet: debug
log:
  level:
    root: ERROR
    #自定义服务的日志等级, 名称是服务中的spring.application.name: 等级 TRACE,DEBUG,INFO,WAR
N,ERROR
    jnpf-oauth: ERROR
    jnpf-system: ERROR
    jnpf-example: ERROR
    jnpf-datareport: ERROR
    jnpf-visualdev: ERROR
    jnpf-workflow: ERROR
    jnpf-file: ERROR
    jnpf-tenant: ERROR
    jnpf-message: ERROR
    jnpf-scheduletask: ERROR
    jnpf-permission: ERROR
    jnpf-visualdata: ERROR
    jnpf-app: ERROR
  path: log/${spring.application.name}
```

frame-config.yaml

框架配置

```
spring:
  cloud:
    sentinel:
      # 取消控制台懒加载
      eager: true
```

```
      transport:
        dashboard: 127.0.0.1:30098
      log:
        dir: log/${spring.application.name}/sentinel

feign:
  sentinel:
    enabled: true
  client:
    config:
      default:
        connectTimeout: 5000
        readTimeout: 300000
  httpclient:
    connection-timeout: 5000

seata:
  enabled: true
  application-id: ${spring.application.name}
  #此处配置自定义的seata事务分组名称
  tx-service-group: default_tx_group
  #  enable-auto-data-source-proxy: true    #开启数据库代理
  config:
    type: nacos
    nacos:
      namespace:
      serverAddr: ${spring.cloud.nacos.discovery.server-addr}
      group: "DEFAULT_GROUP"
      userName: "nacos"
      password: "nacos"
      data-id: seata-server.properties
  registry:
    type: ${seata.config.type}
    nacos:
      application: seata-server
      serverAddr: ${spring.cloud.nacos.discovery.server-addr}
      group: ${seata.config.nacos.group}
      namespace:
      cluster: default
      userName: ${seata.config.nacos.userName}
      password: ${seata.config.nacos.password}

# dubbo 配置
dubbo:
  # 指定Dubbo服务实现类的包
```

```
  scan:
    base-packages: jnpf.provider.impl
  protocols:
    dubbo:
      # 协议名称
      name: dubbo
      # 端口 -1为20880自增长
      port: -1
  registry:
    # 挂载到nacos注册中心
    address: nacos://${spring.cloud.nacos.discovery.server-addr}
  # 启动时不检查 consumer
  consumer:
    check: false
  application:
    name: dubbo-${spring.application.name}
    logger: slf4j
```

oauth.yaml

> 单点登录(SSO)配置

```
oauth:
  #启用单点登录, 普通登录不可用
  ssoEnabled: false
  #轮询票据有效期
  ticketTimeout: 120
  #默认单点登录协议
  defaultSSO: cas
  #后端登录接口地址
  loginPath: http://127.0.0.1:30000/api/oauth/Login
  #轮询登录模式是否输出结果
  ticketOutMessage: false
  login:
    #JWT生成秘钥 不填写为默认值
    #jwtSecretKey: WviMjFNC72VKwGqm5LPoheQo5XN9iN4d
  sso:
    #登录成功后跳转到前端的页面
    sucessFrontUrl: http://127.0.0.1:3000/sso
    #logoutFrontUrl: http://sso.maxkey.top:8527/maxkey
    auth2:
      enabled: true
```

```
      clientId: 747887288041603072
      clientSecret: MYgMMjIwNzIwMjIxNTU4MTAxNzQlKQ
      baseUrl: http://127.0.0.1:8527
      authorizeUrl: ${oauth.sso.auth2.baseUrl}/sign/authz/oauth/v20/authorize
      accessTokenUrl: ${oauth.sso.auth2.baseUrl}/sign/authz/oauth/v20/token
      userInfoUrl: ${oauth.sso.auth2.baseUrl}/sign/api/oauth/v20/me
    cas:
      enabled: true
      baseUrl: ${oauth.sso.auth2.baseUrl}
      serverLoginUrl: ${oauth.sso.cas.baseUrl}/sign/authz/cas/login
      serverValidateUrl: ${oauth.sso.cas.baseUrl}/sign/authz/cas
```

socials.yaml

> 第三方登录配置

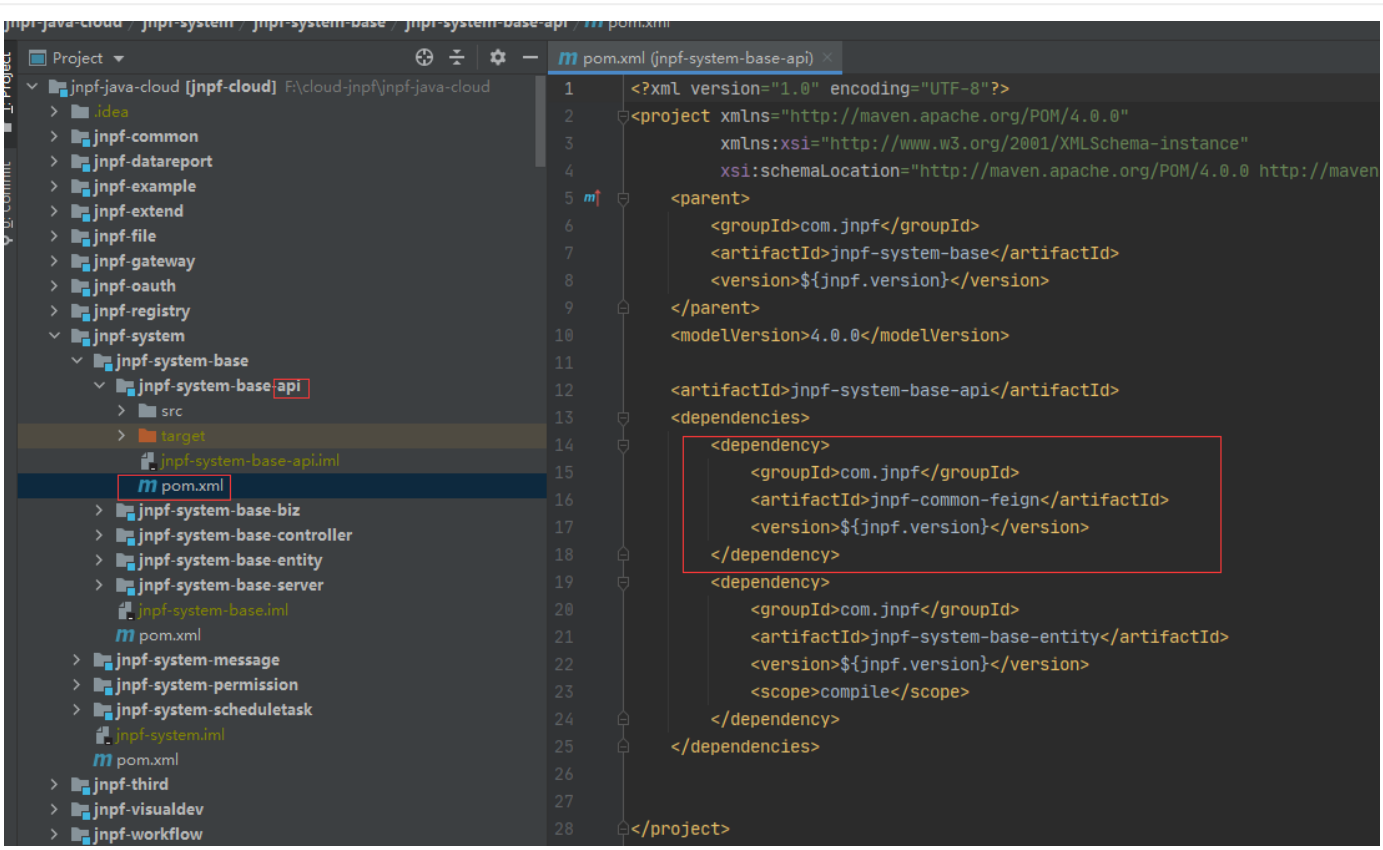
```
socials:
  # 第三方登录功能开关(false-关闭, true-开启)
  socials-enabled: false
  config:
    - # 微信
      provider: wechat_open
      client-id: your-client-id
      client-secret: your-client-secret
    - # qq
      provider: qq
      client-id: your-client-id
      client-secret: your-client-secret
    - # 企业微信
      provider: wechat_enterprise
      client-id: your-client-id
      client-secret: your-client-secret
      agentId: your-agentId
    - # 钉钉
      provider: dingtalk
      client-id: your-client-id
      client-secret: your-client-secret
      agentId: your-agentId
    - # 飞书
      provider: feishu
      client-id: your-client-id
      client-secret: your-client-secret
```

```
    - # 小程序
      provider: wechat_applets
      client-id: your-client-id
      client-secret: your-client-secret
```

Feign的使用

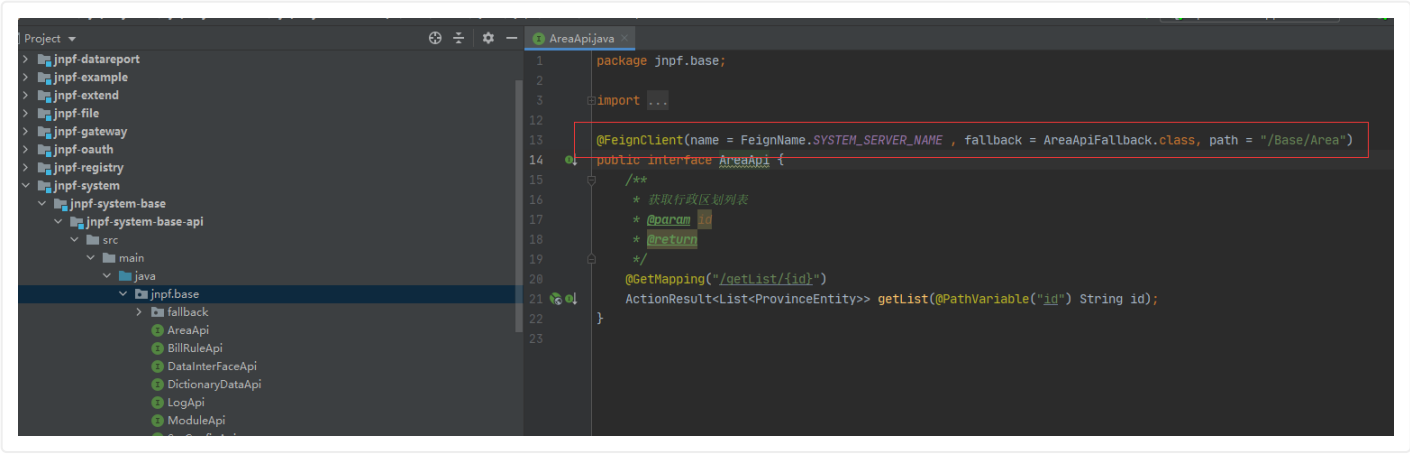
添加依赖

在 `xxx-api` 的maven中添加 `jnpf-common-feign` 依赖

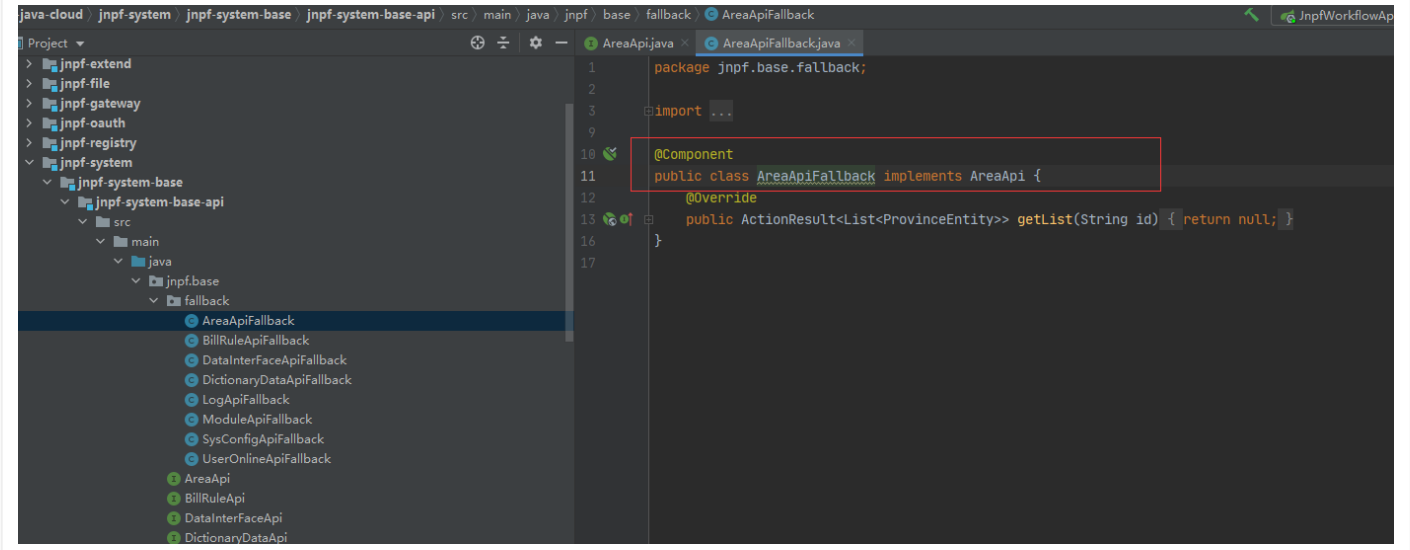


定义Feign调用对象

创建一个接口，添加 `@FeignClient` 注解，指定服务，降级类

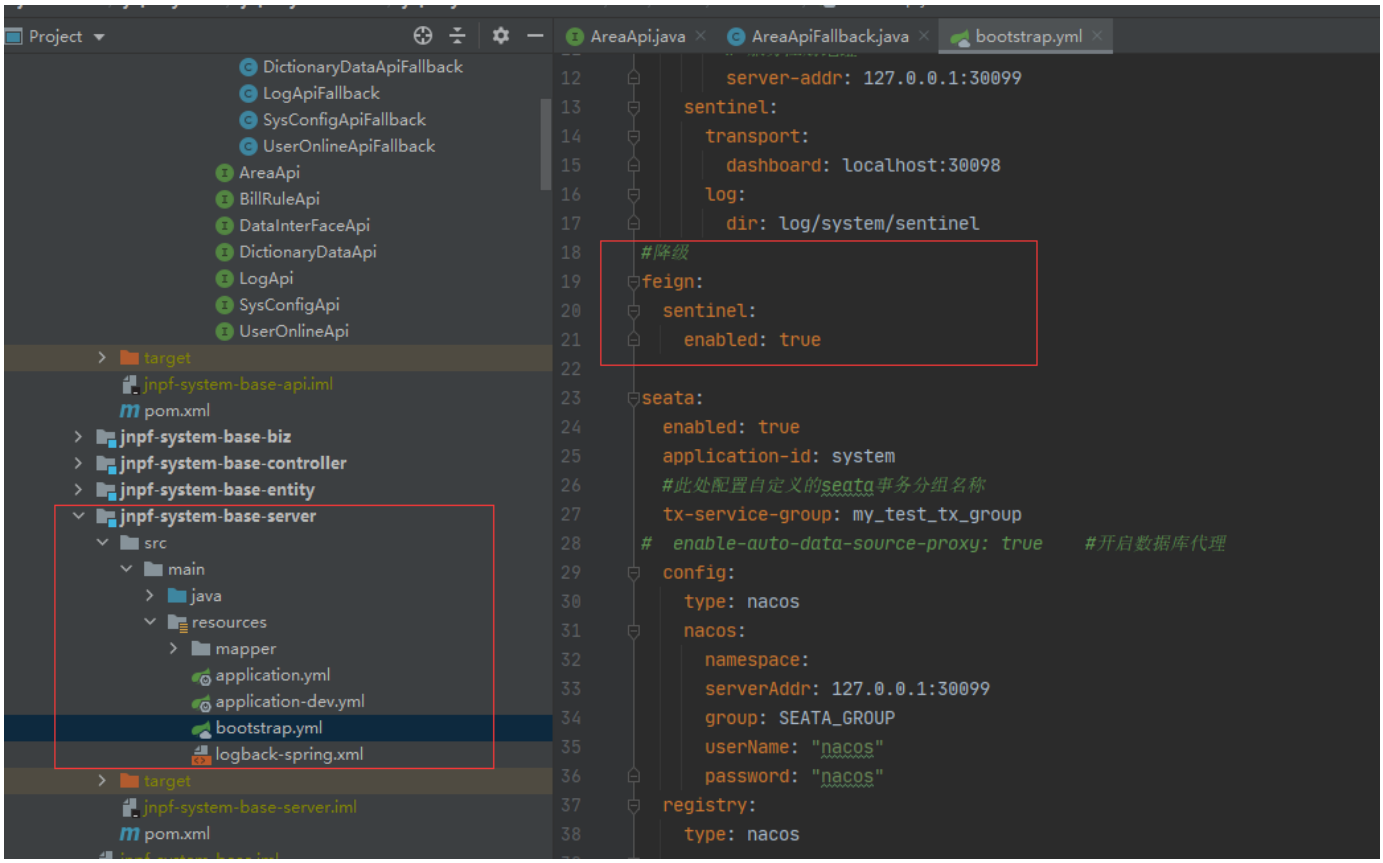


创建一个类实现接口并重写里面的方法，在类上添加@Component



bootstrap.yml开启降级

在xxx-server项目下的resource\bootstrap.yml中开启降级



使用Feign访问接口

使用@Autowired注解得到实例，通过.xxx()调用里面的方法即可得到数据

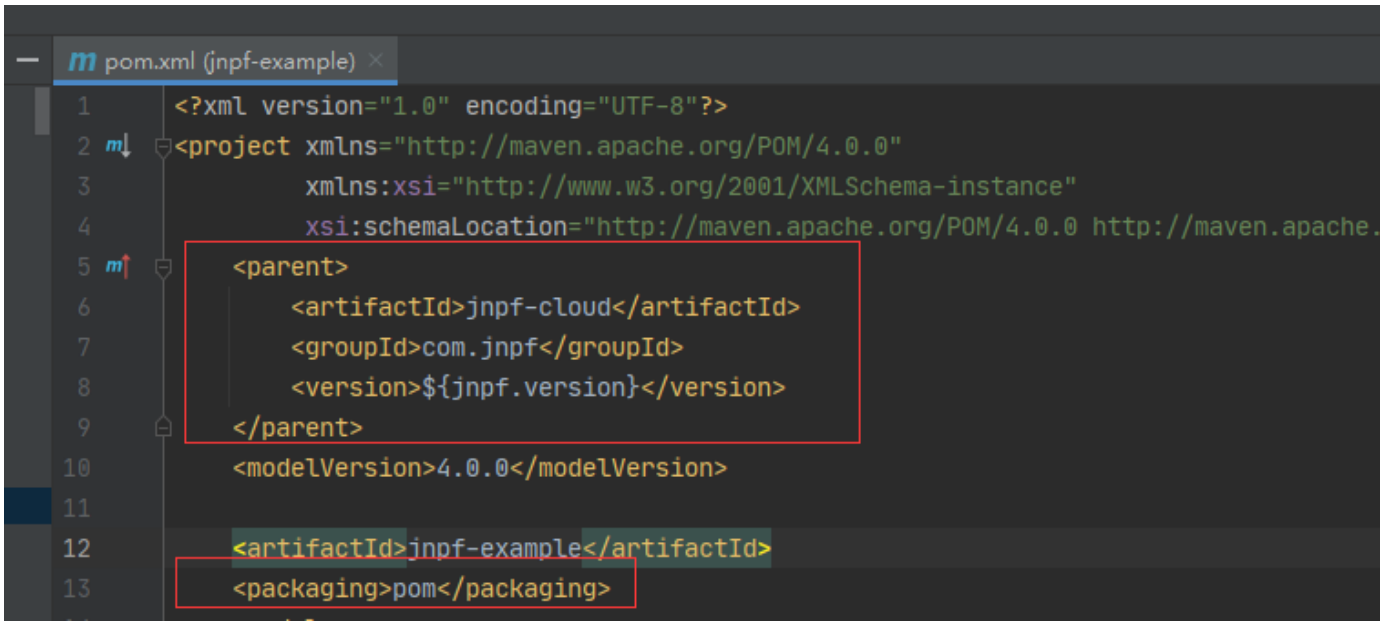
```
@Autowired
private AreaApi areaApi;
```

```
List<ProvinceEntity> data = areaApi.getList("-1")
```

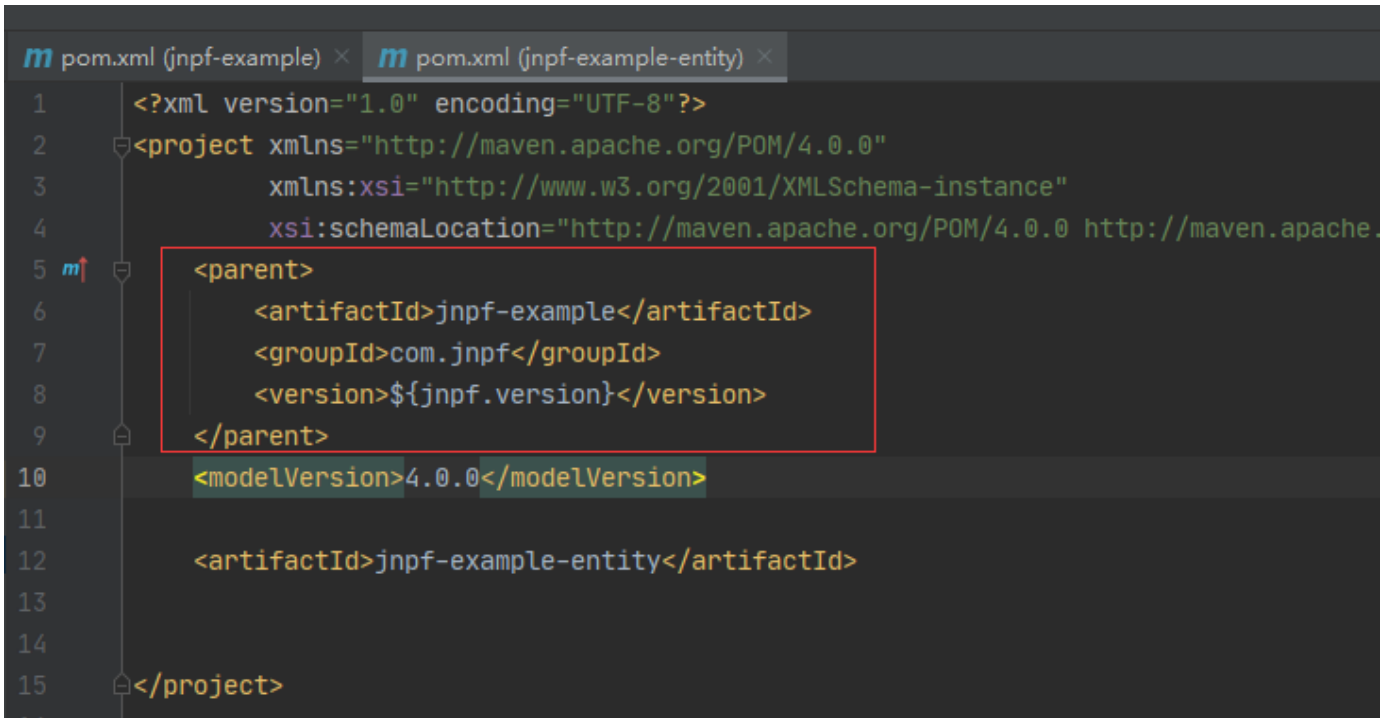
如何添加子系统（子服务）

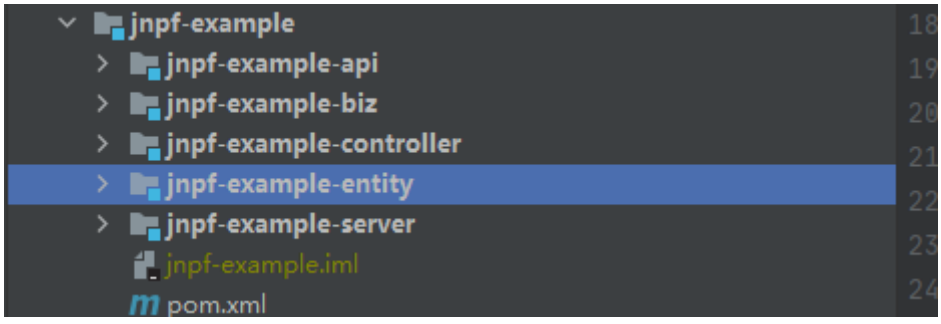
一、自己创建子模块

创建一个Maven项目，继承`jnpf-cloud`，packaging为pom



再创建该模块的entity、controller、biz、api等Maven项目继承刚刚创建好的Maven项目，此次无需指定为pom，例：

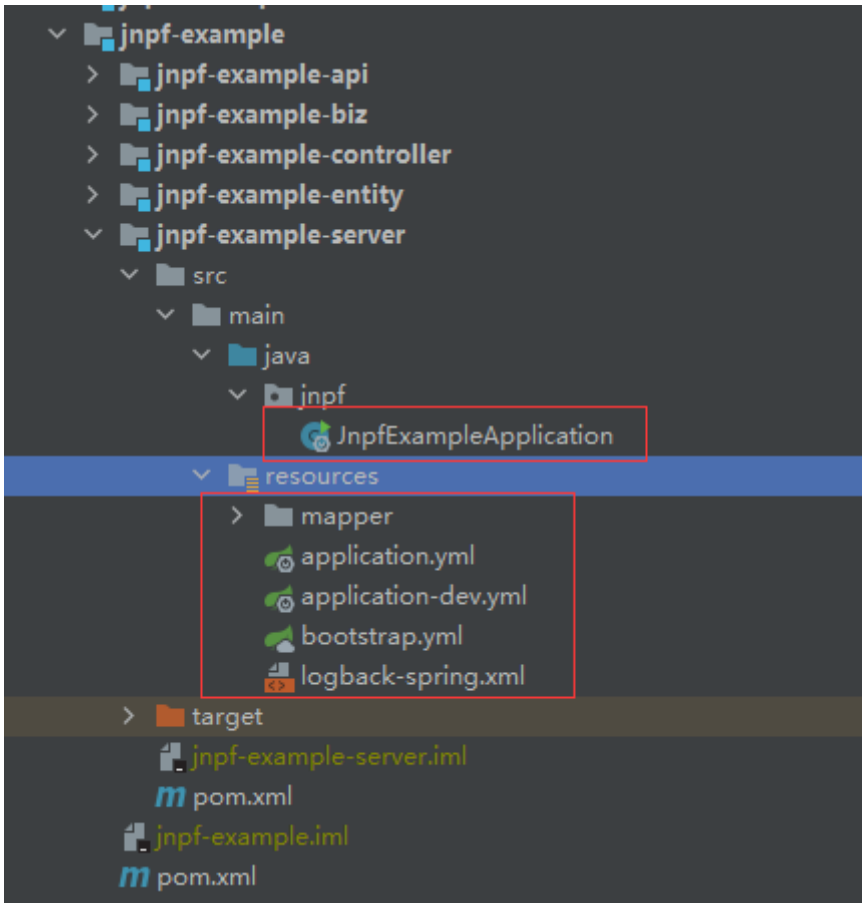




若有需要使用Swagger、Database、Aop、Redis、OpenFeign则在需要使用的Maven中导入jnpf-common-xxx，例：

```xml
<dependencies>

    <dependency>
        <groupId>com.jnpf</groupId>
        <artifactId>jnpf-common-database</artifactId>
        <version>${jnpf.version}</version>
    </dependency>

</dependencies>
```

jnpf-xxx-server下创建启动类和配置文件、mapper,xml等常用配置，将jnpf-xxx-controller依赖进pom.xml中，例：

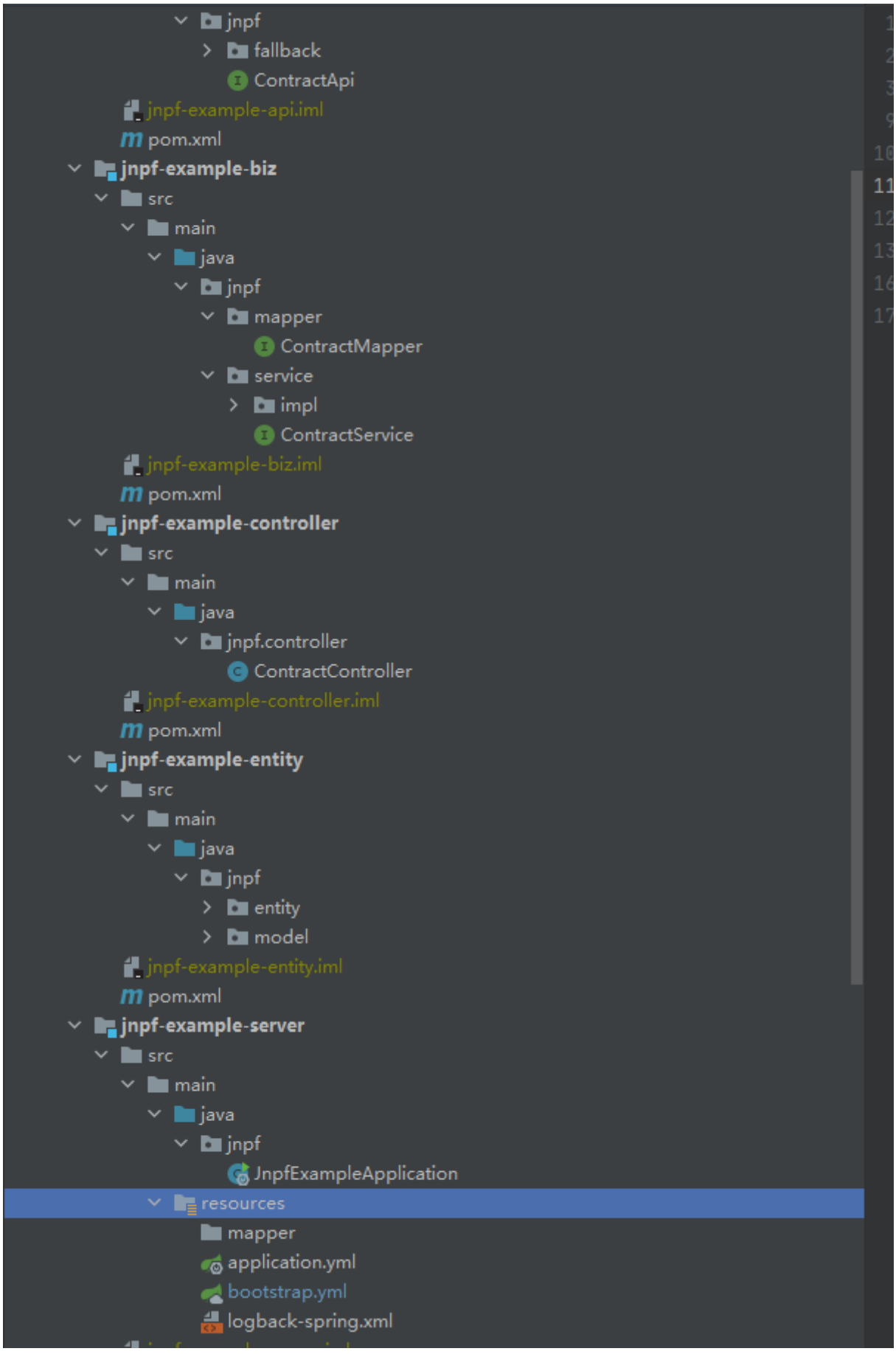



```xml
<dependencies>
    <dependency>
        <groupId>com.jnpf</groupId>
        <artifactId>jnpf-example-controller</artifactId>
        <version>${jnpf.version}</version>
    </dependency>
    <dependency>
        <groupId>com.jnpf</groupId>
        <artifactId>jnpf-common-feign</artifactId>
        <version>${jnpf.version}</version>
    </dependency>
</dependencies>

<build>
    <finalName>jnpf-example-${jnpf.version}</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <!-- 指定该Main Class为全局的唯一入口 -->
                <mainClass>jnpf.JnpfExampleApplication</mainClass>
                <layout>ZIP</layout>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>repackage</goal><!--可以把依赖的包都打包到生成的Jar包中-->
                    </goals>
                </execution>
```

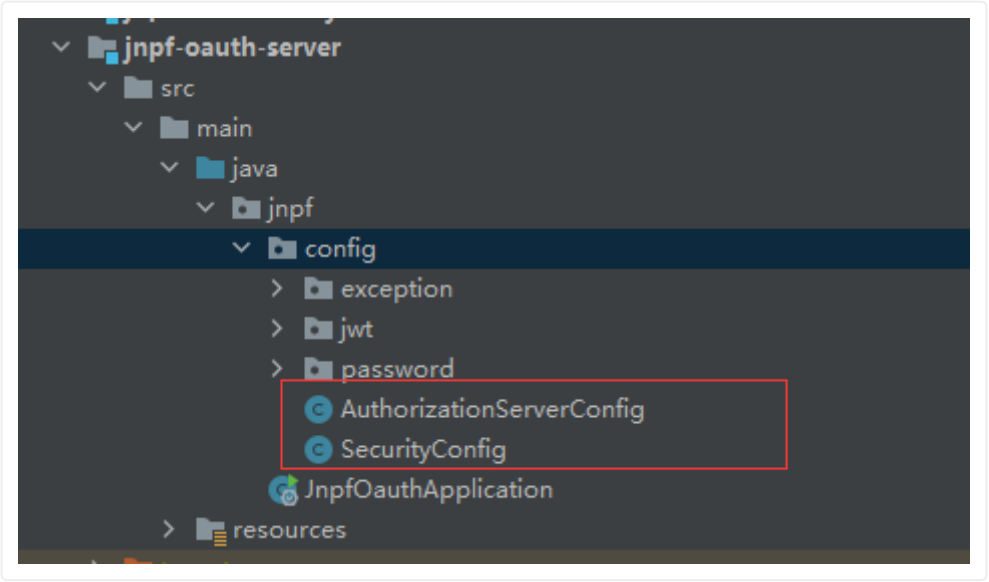
打包指定启动类

```
            </executions>
        </plugin>
    </plugins>
</build>
```

根据自己的需求创建entity，mapper、service、controller、model和util，例：



若有configuration文件则放置到server下，例：



注意：若entity、mapper、扫描不到，则在启动类上添加相应的扫包注解，若是需要通过Feign调用则需要在server的pom.xml中添加jnpf-common-fein，启动类上添加@EnableFeignClients，例：

正常情况下：

```
@SpringBootApplication
@EnableFeignClients
```

扫描不到时可通过注解自己选择扫包路径：

```
@SpringBootApplication(scanBasePackages = {"jnpf"})
@EnableFeignClients(basePackages = {"jnpf","xxxx"})
@MapperScan(basePackages = {"jnpf","xxxx"})
```

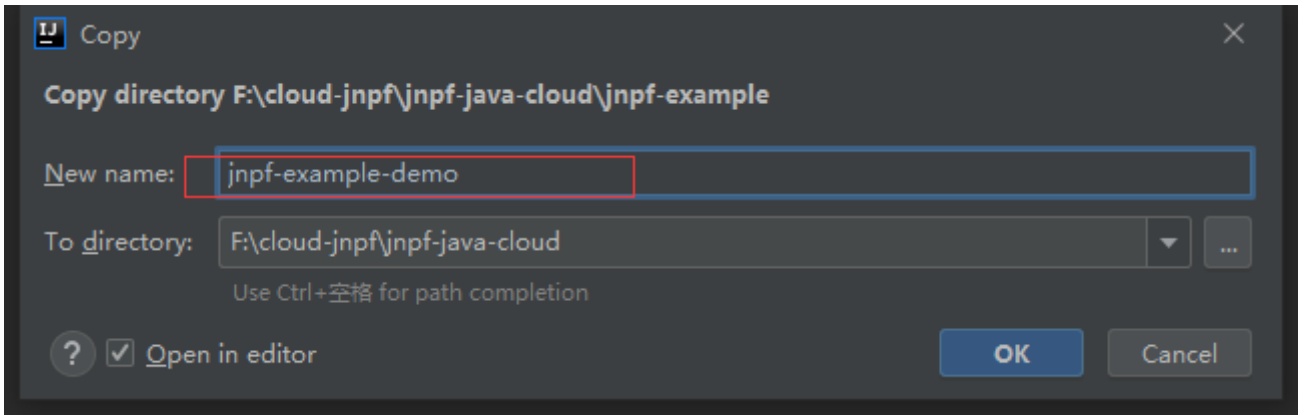
项目扫包路径，具体可自己调整

feign扫包

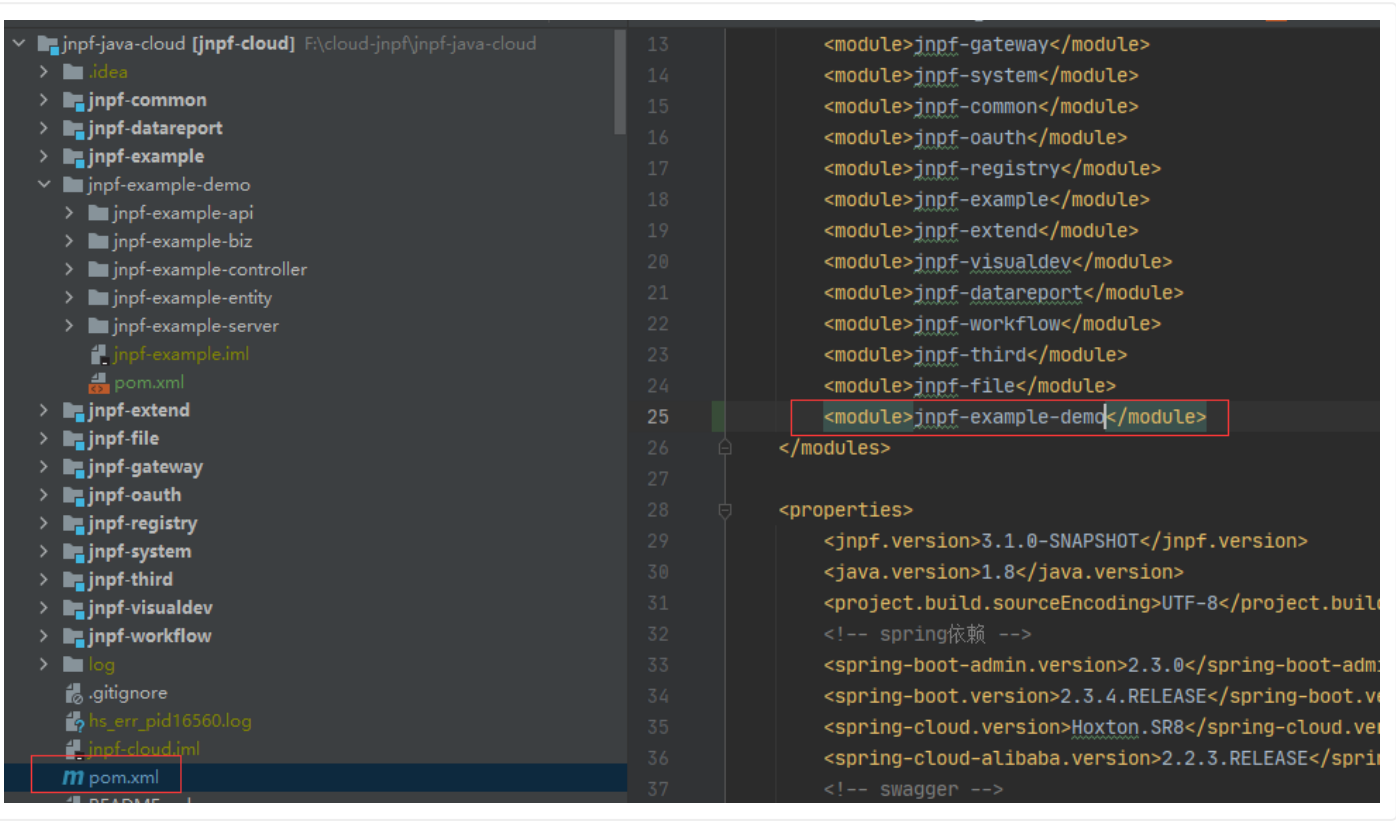
mybatis扫包

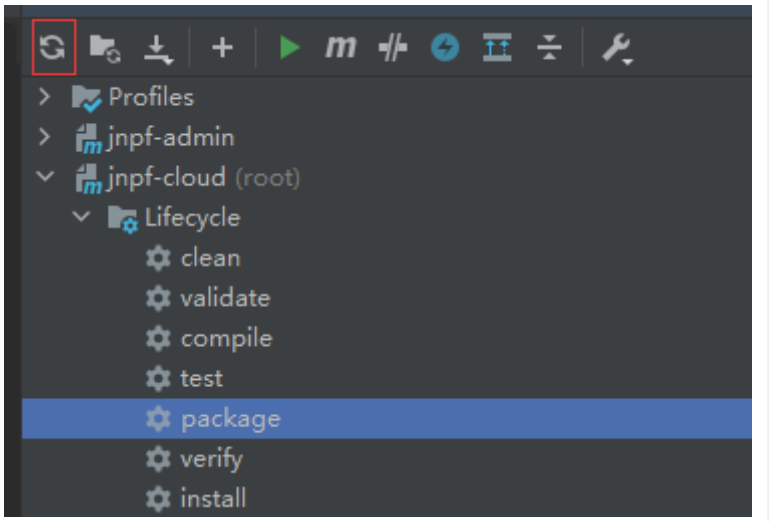
二、使用Example项目快速创建

复制Example项目粘贴到jnpf-cloud下，修改项目名某点击OK

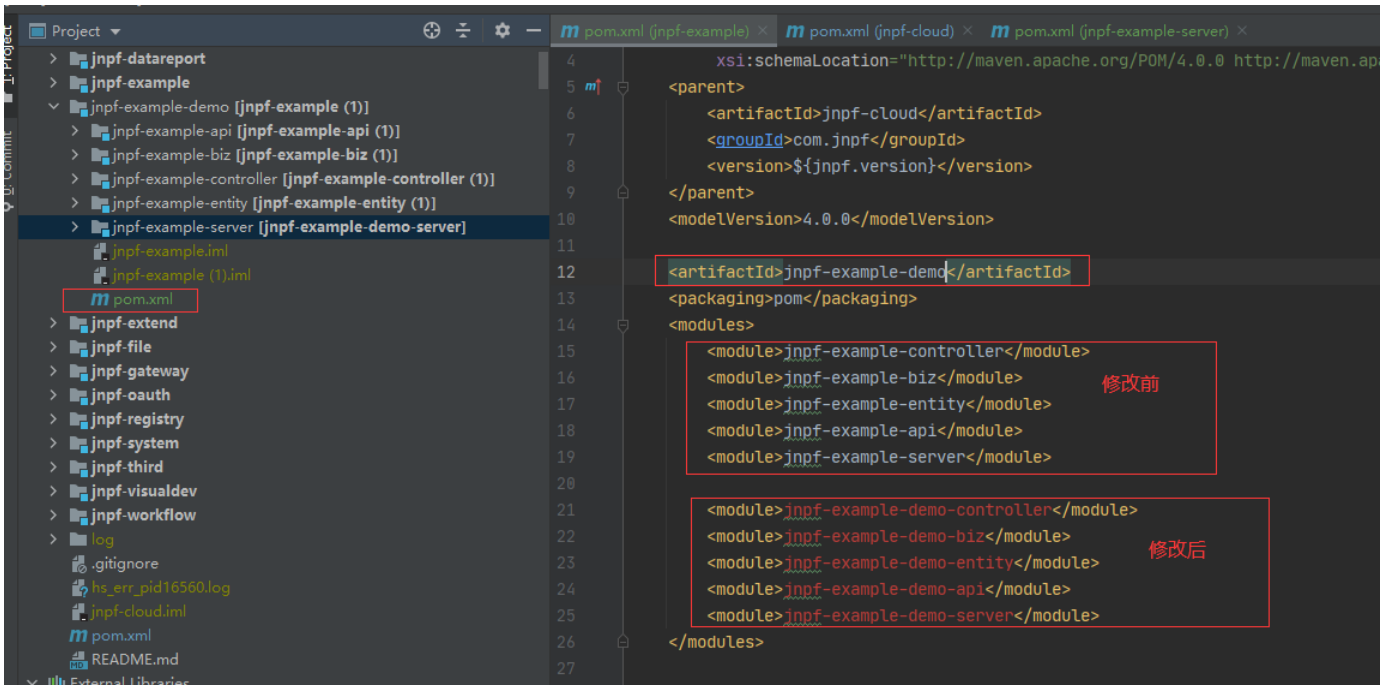


粘贴之后在最外层的Maven中添加如下代码，点击刷新Maven

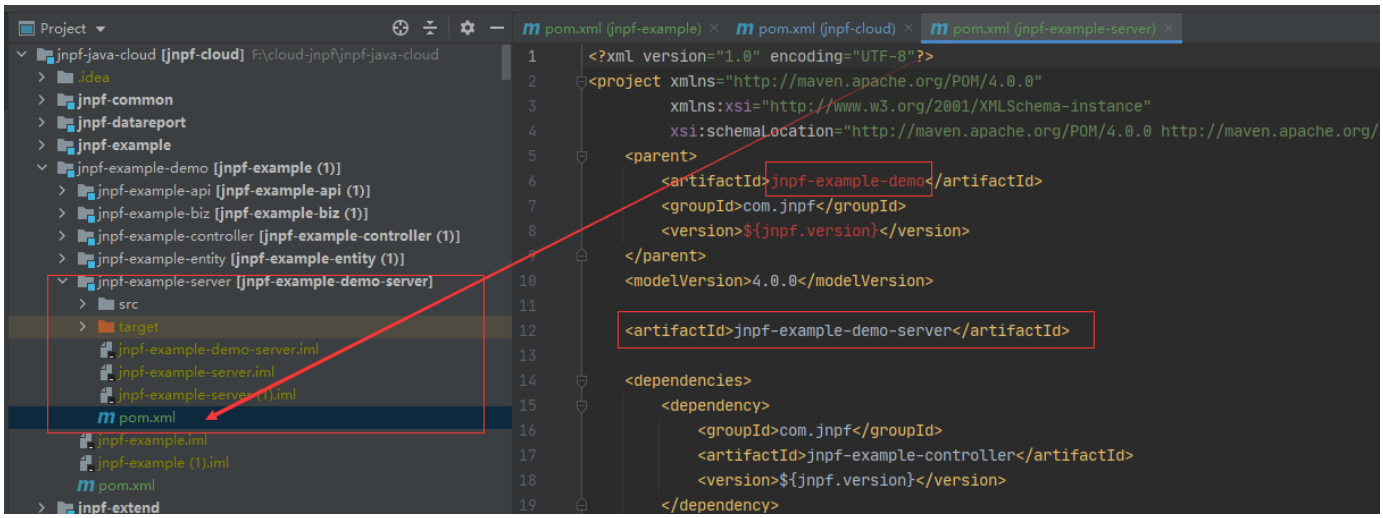




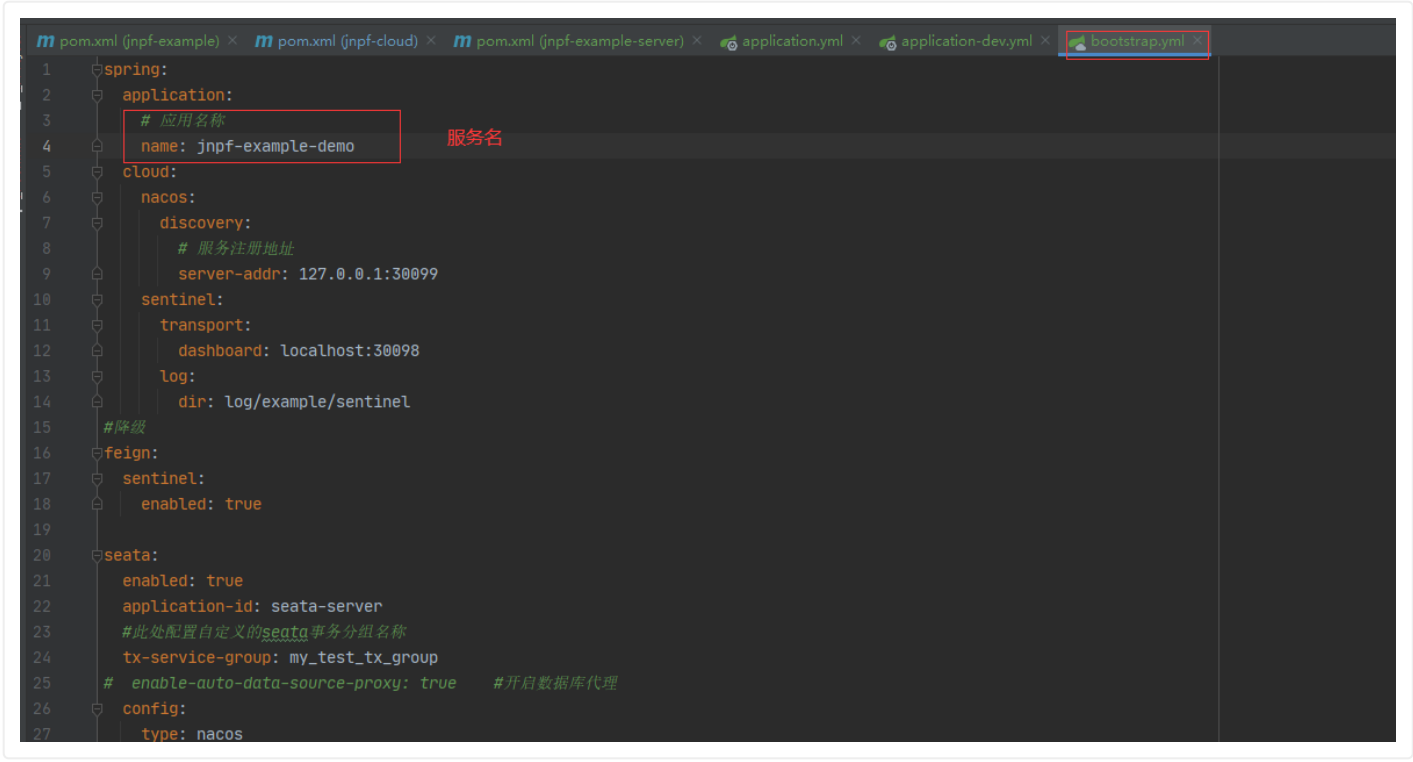
修改刚刚创建好的项目的pom.xml，例：



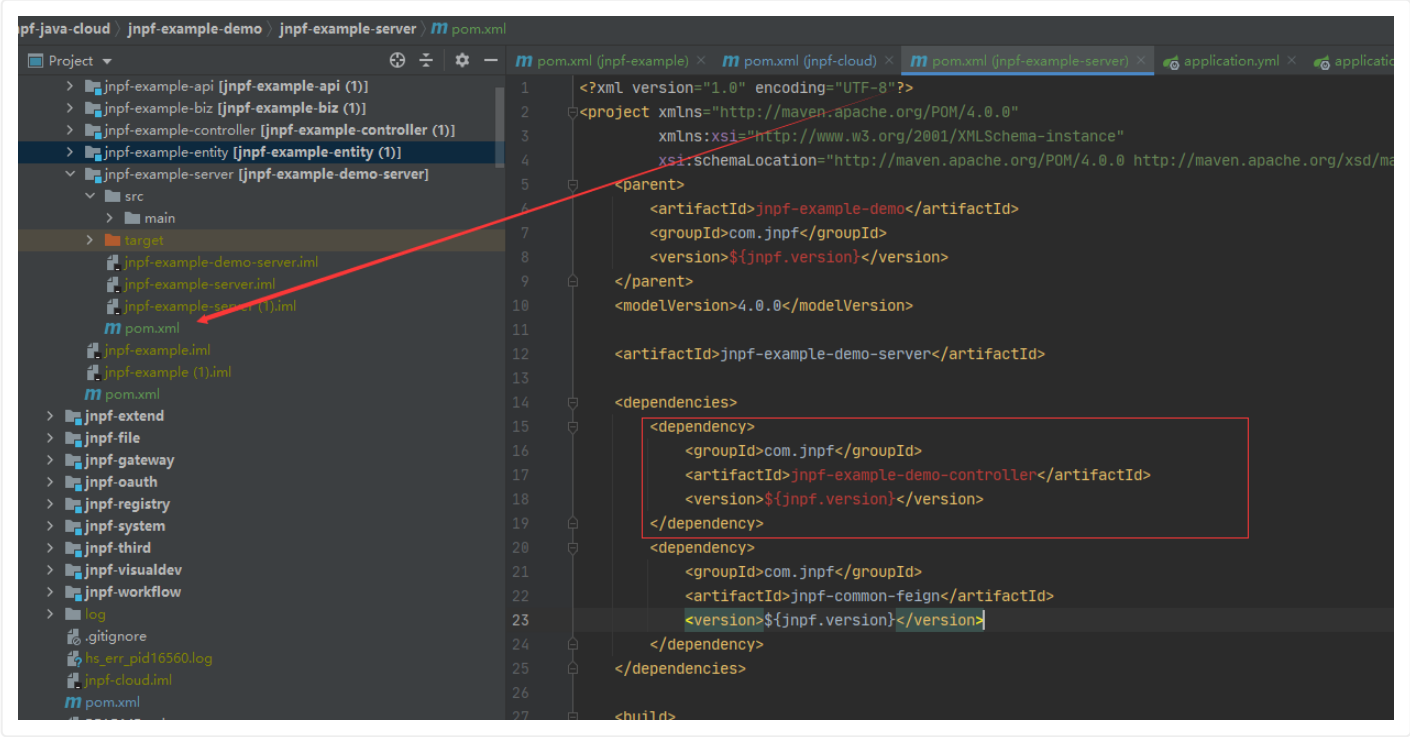
对应的子模块pom.xml，例如server：

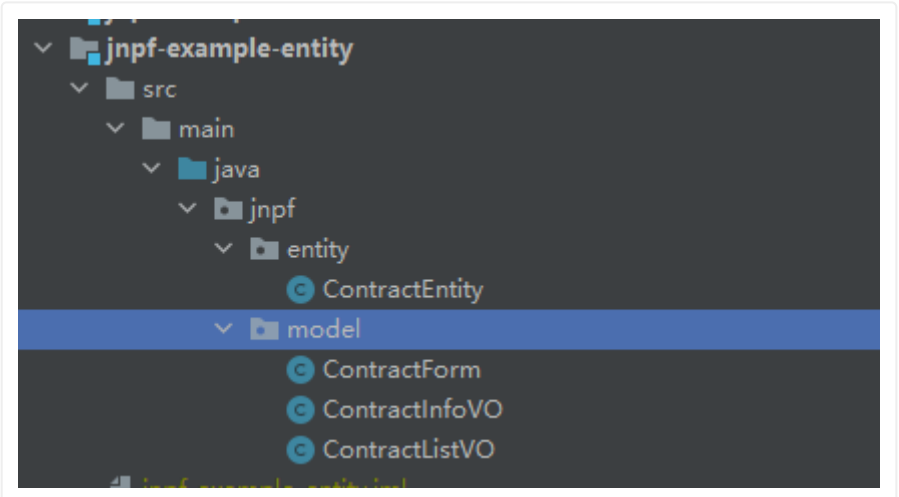


服务名不能重复，需要自己修改：



然后将包下的类删除，类似依赖也要改成现在的新名称，然后创建自己的类即可：

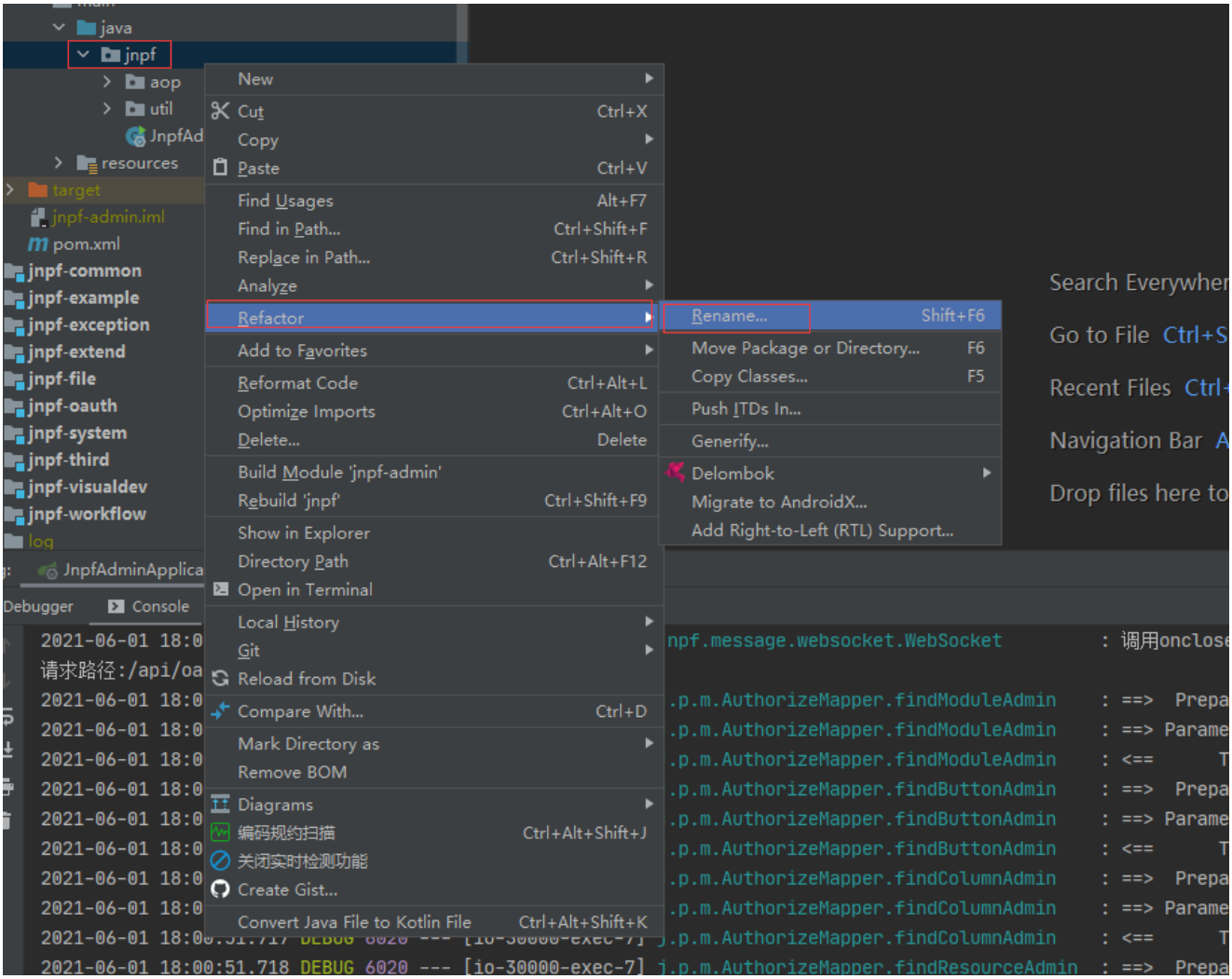

jpf-java-cloud
jnpf-example-demo
jnpf-example-server
pom.xml
Project
jnpf-example-api [jnpf-example-api (1)]
jnpf-example-biz [jnpf-example-biz (1)]
jnpf-example-controller [jnpf-example-controller (1)]
jnpf-example-entity [jnpf-example-entity (1)]
jnpf-example-server [jnpf-example-demo-server]
src
main
target
jnpf-example-demo-server.iml
jnpf-example-server.iml
pom.xml
jnpf-example.iml
jnpf-example (1).iml
pom.xml
jnpf-extend
jnpf-file
jnpf-gateway
jnpf-oauth
jnpf-registry
jnpf-system
jnpf-third
jnpf-visualdev
jnpf-workflow
log
.gitignore
hs_err_pid16560.log
jnpf-cloud.iml
pom.xml
pom.xml (jnpf-example)
pom.xml (jnpf-cloud)
pom.xml (jnpf-example-server)
application.yml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/ma
<parent>
<artifactId>jnpf-example-demo</artifactId>
<groupId>com.jnpf</groupId>
<version>${jnpf.version}</version>
</parent>
<modelVersion>4.0.0</modelVersion>
<artifactId>jnpf-example-demo-server</artifactId>
<dependencies>
<dependency>
<groupId>com.jnpf</groupId>
<artifactId>jnpf-example-demo-controller</artifactId>
<version>${jnpf.version}</version>
</dependency>
<dependency>
<groupId>com.jnpf</groupId>
<artifactId>jnpf-common-feign</artifactId>
<version>${jnpf.version}</version>
</dependency>
</dependencies>
<build>


jnpf-example-entity
src
main
java
jnpf
entity
ContractEntity
model
ContractForm
ContractInfoVO
ContractListVO

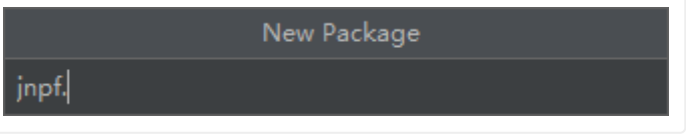
如何修改包名

修改包名

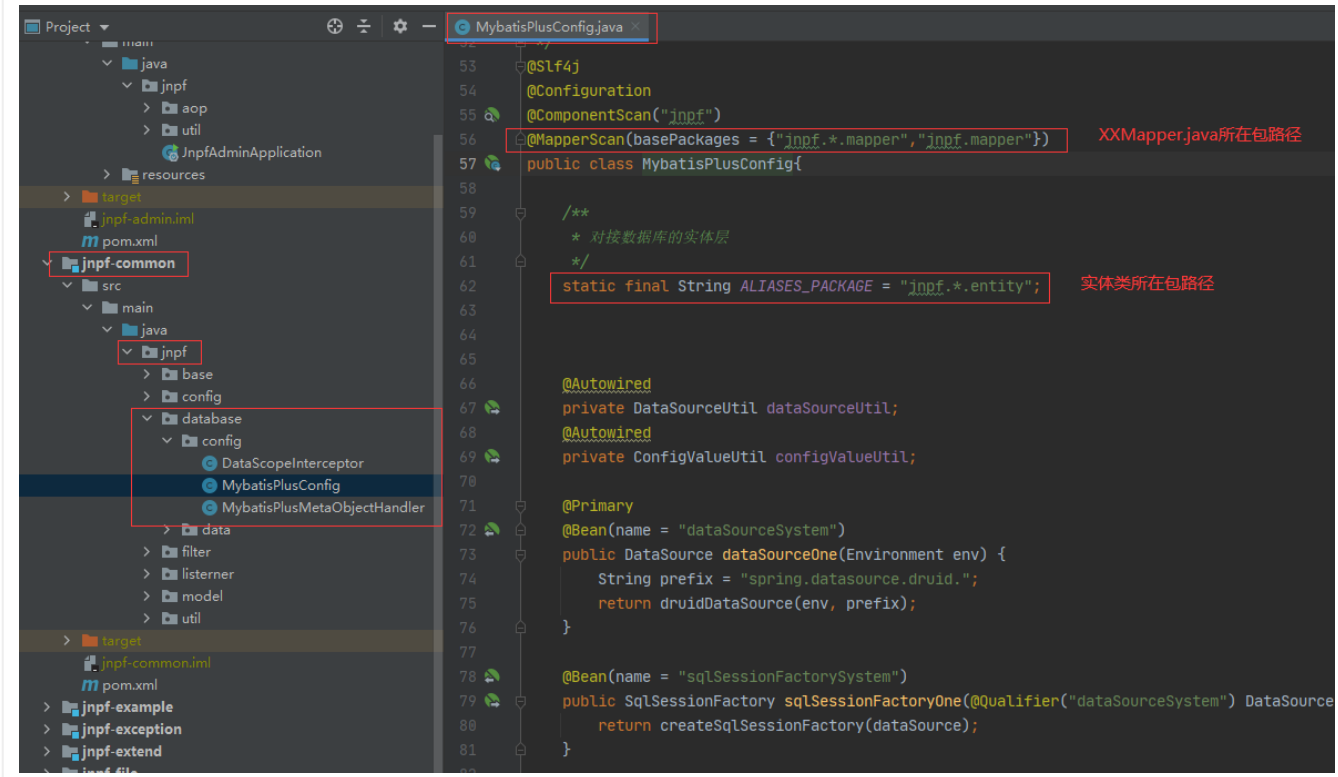
1. 找到要改的包，右键后点击如图所示的选项

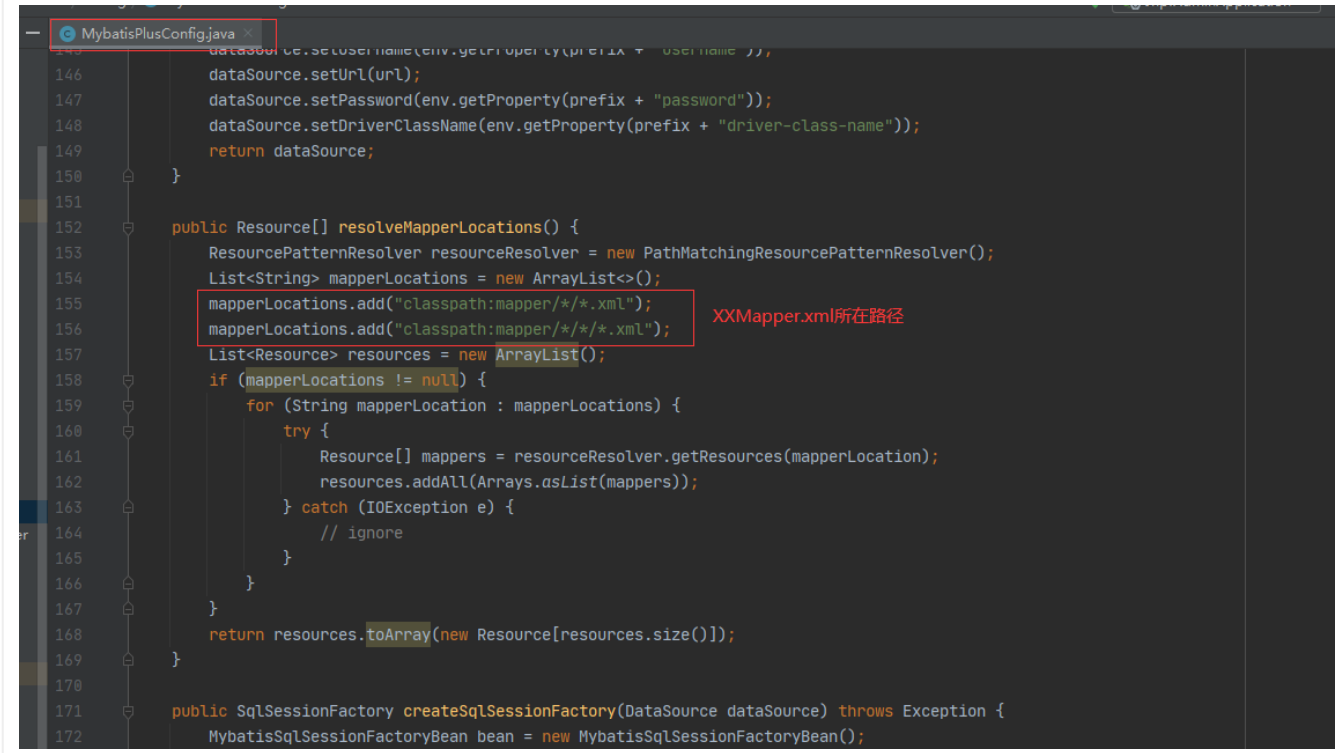


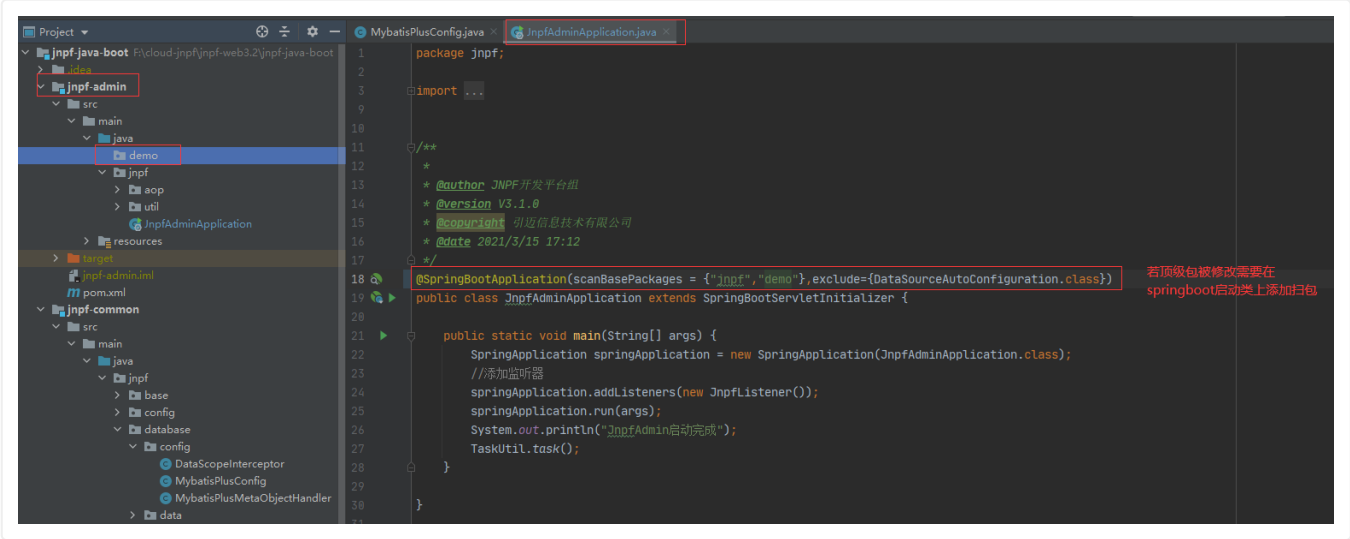
弹出new package的窗口后进行修改



2. 若实体类、XXMapper.java或XXMapper.xml文件的路径发生变化，需要修改MybatisPlusConfig.java中的实体类路径和mapper路径,如图所示：

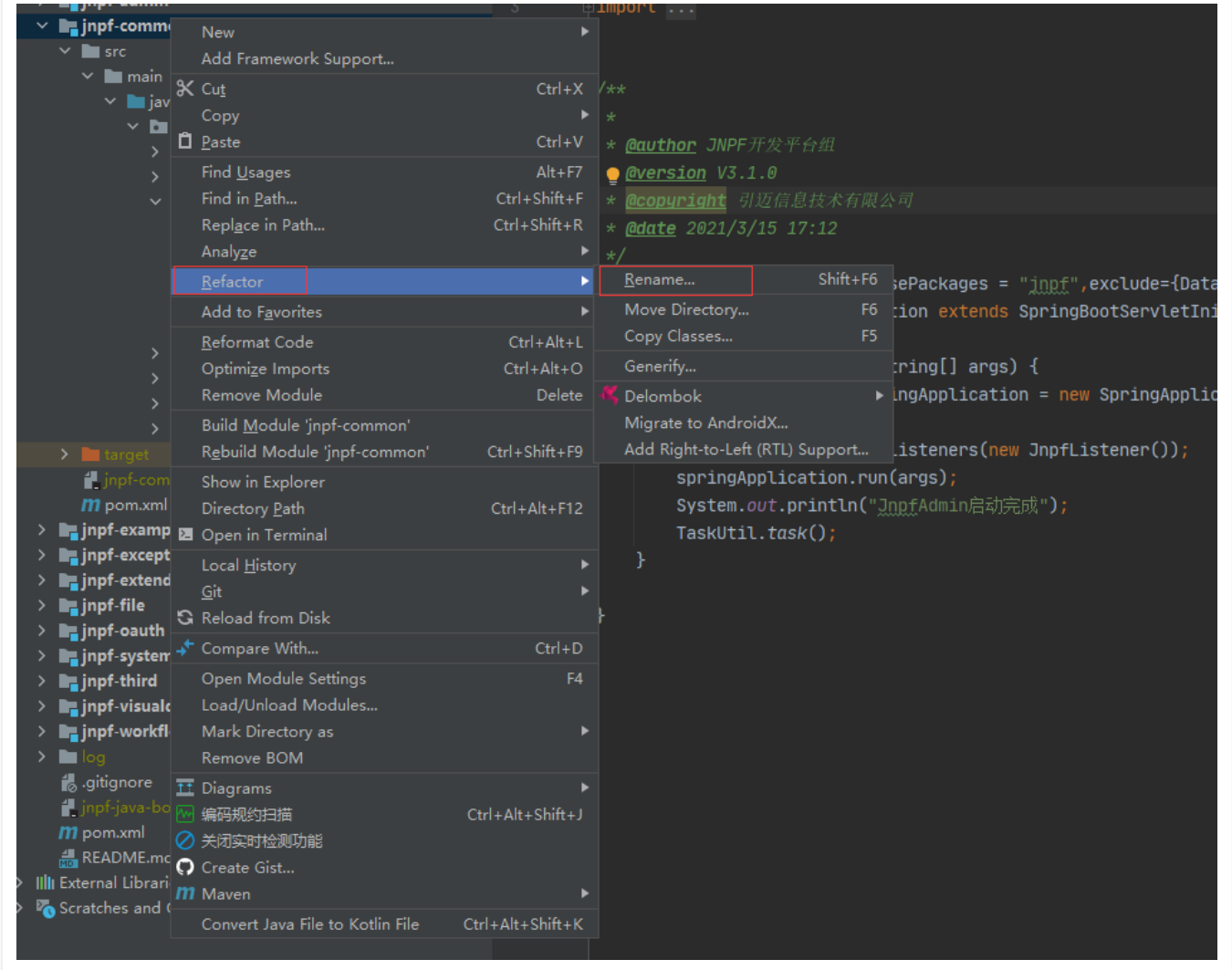

Project
java
jnpf
aop
util
JnpfAdminApplication
resources
target
jnpf-admin.iml
pom.xml
jnpf-common
src
main
java
jnpf
base
config
database
config
DataScopeInterceptor
MybatisPlusConfig
MybatisPlusMetaObjectHandler
data
filter
listerner
model
util
target
jnpf-common.iml
pom.xml
jnpf-example
jnpf-exception
jnpf-extend
MybatisPlusConfig.java
@Slf4j
@Configuration
@ComponentScan("jnpf")
@MapperScan(basePackages = {"jnpf.*.mapper","jnpf.mapper"})
XXMapper.java所在包路径
public class MybatisPlusConfig{
/**
* 对接数据库的实体层
*/
static final String ALIASES_PACKAGE = "jnpf.*.entity";
实体类所在包路径
@Autowired
private DataSourceUtil dataSourceUtil;
@Autowired
private ConfigValueUtil configValueUtil;
@Primary
@Bean(name = "dataSourceSystem")
public DataSource dataSourceOne(Environment env) {
String prefix = "spring.datasource.druid.";
return druidDataSource(env, prefix);
}
@Bean(name = "sqlSessionFactorySystem")
public SqlSessionFactory sqlSessionFactoryOne(@Qualifier("dataSourceSystem") DataSource
return createSqlSessionFactory(dataSource);
}


MybatisPlusConfig.java
dataSource.setUrl(url);
dataSource.setPassword(env.getProperty(prefix + "password"));
dataSource.setDriverClassName(env.getProperty(prefix + "driver-class-name"));
return dataSource;
}
public Resource[] resolveMapperLocations() {
ResourcePatternResolver resourceResolver = new PathMatchingResourcePatternResolver();
List<String> mapperLocations = new ArrayList<>();
mapperLocations.add("classpath:mapper/*/*.xml");
mapperLocations.add("classpath:mapper/*/*/*.xml");
XXMapper.xml所在路径
List<Resource> resources = new ArrayList();
if (mapperLocations != null) {
for (String mapperLocation : mapperLocations) {
try {
Resource[] mappers = resourceResolver.getResources(mapperLocation);
resources.addAll(Arrays.asList(mappers));
} catch (IOException e) {
// ignore
}
}
}
return resources.toArray(new Resource[resources.size()]);
}
public SqlSessionFactory createSqlSessionFactory(DataSource dataSource) throws Exception {
MybatisSqlSessionFactoryBean bean = new MybatisSqlSessionFactoryBean();

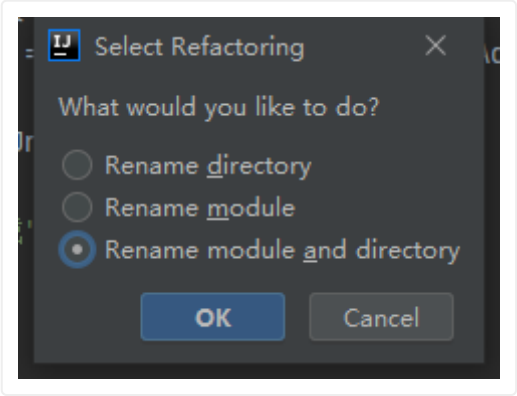


修改Maven名

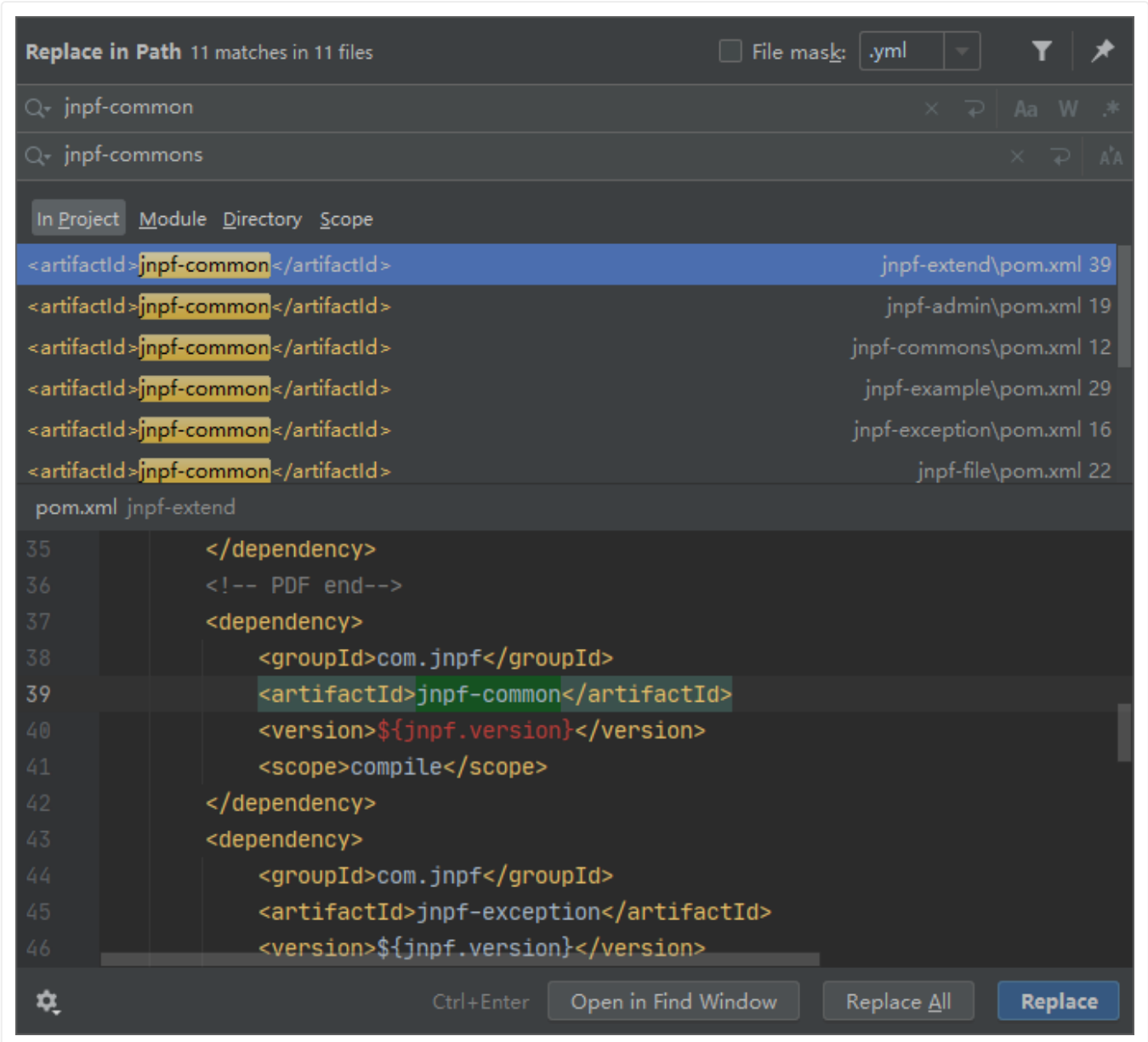
3. 找到要修改的Maven项目，右键点击如图所示的选项



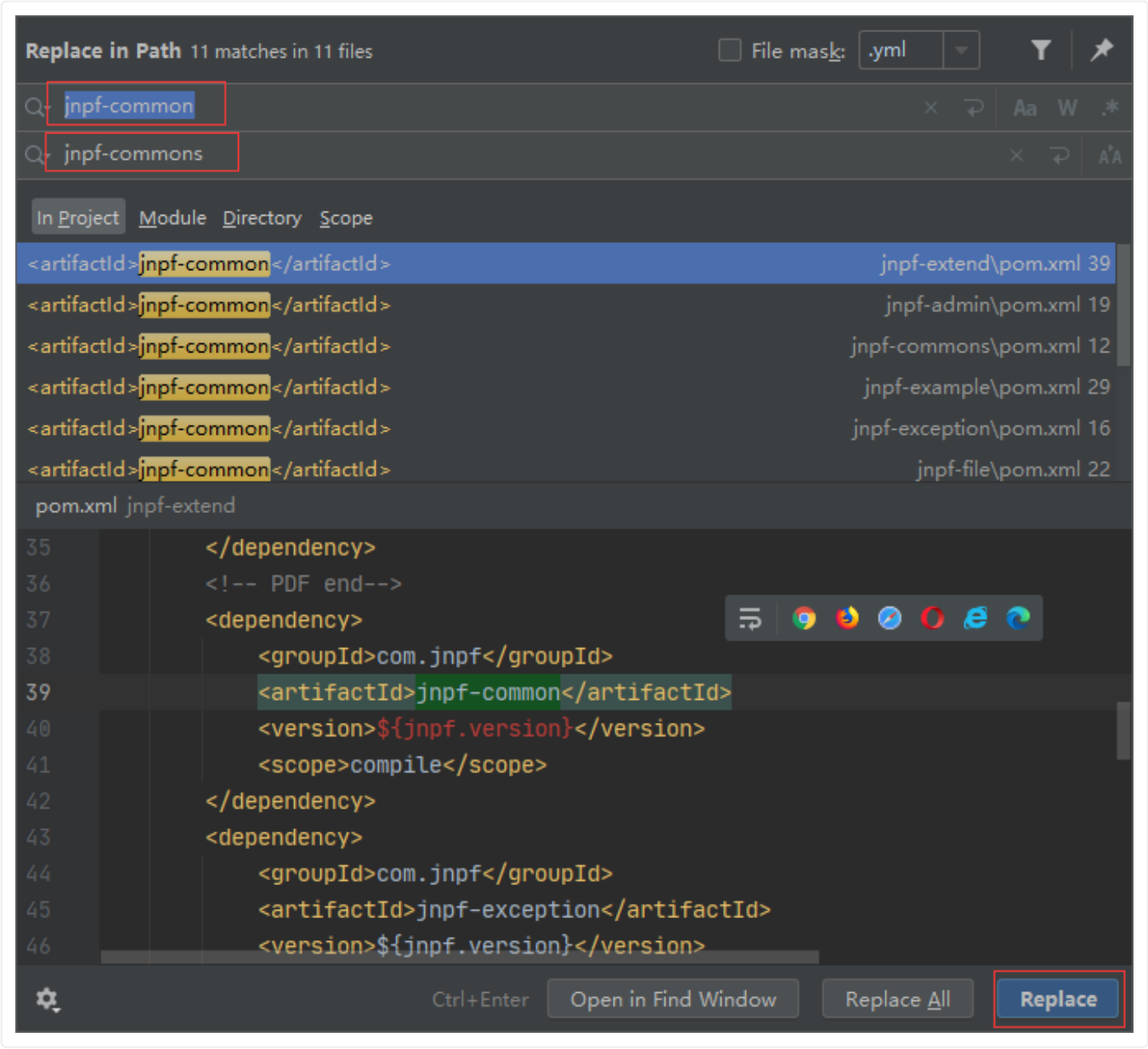
弹出会话框后选择第三个，点击Ok，如图所示：



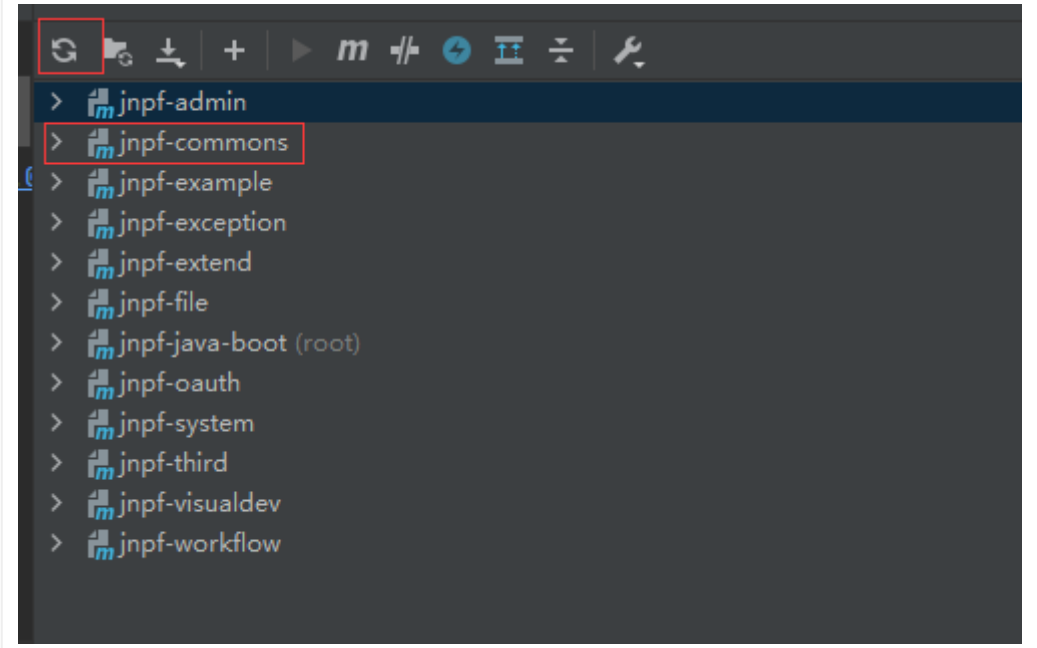
4. 刷新maven，全局搜索被修改前的项目名，全局替换成新的maven名，如图所示：



建议点击replace依次替换，以免将不需要修改的改到！



刷新Maven，到此，Maven名称修改完成

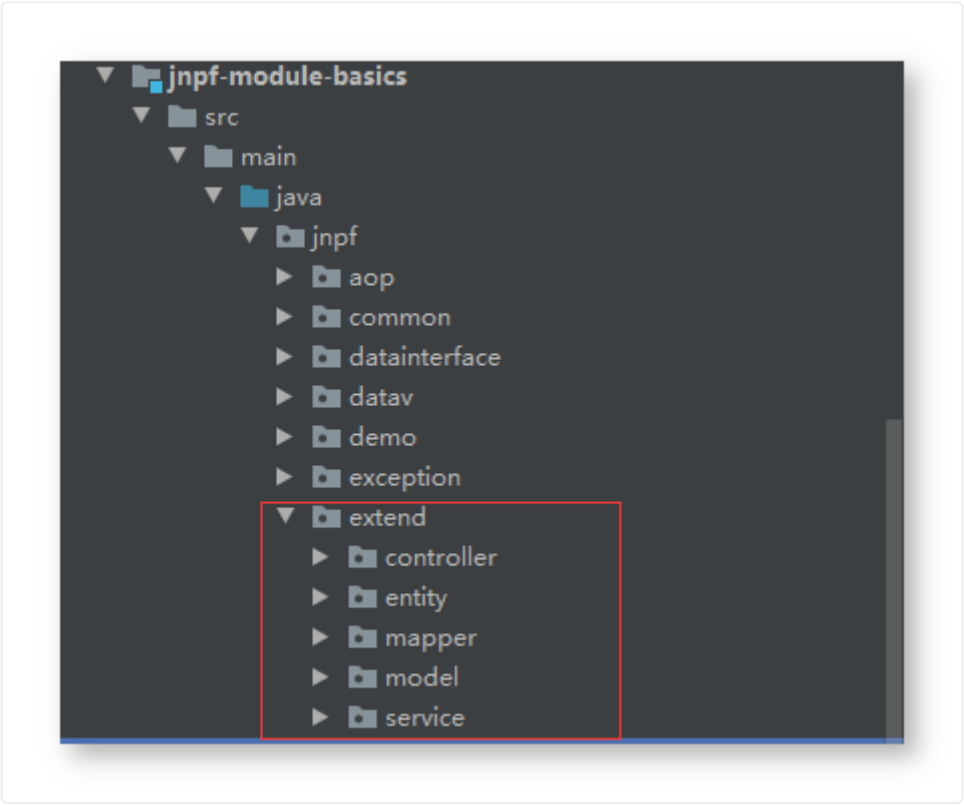


如何控制权限

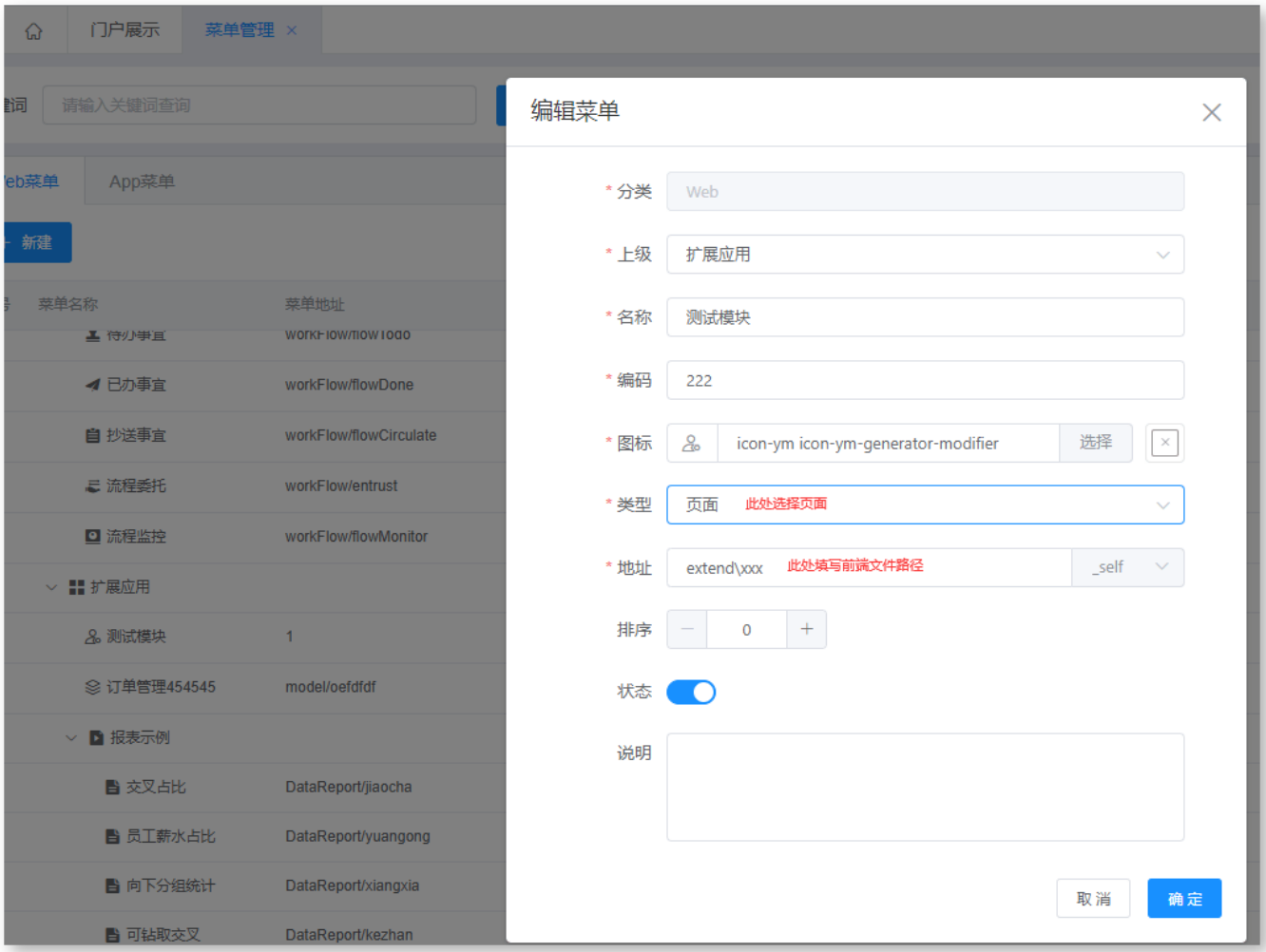
- 菜单权限控制
- 按钮权限控制
- 列表权限控制
- 数据权限控制

菜单权限控制

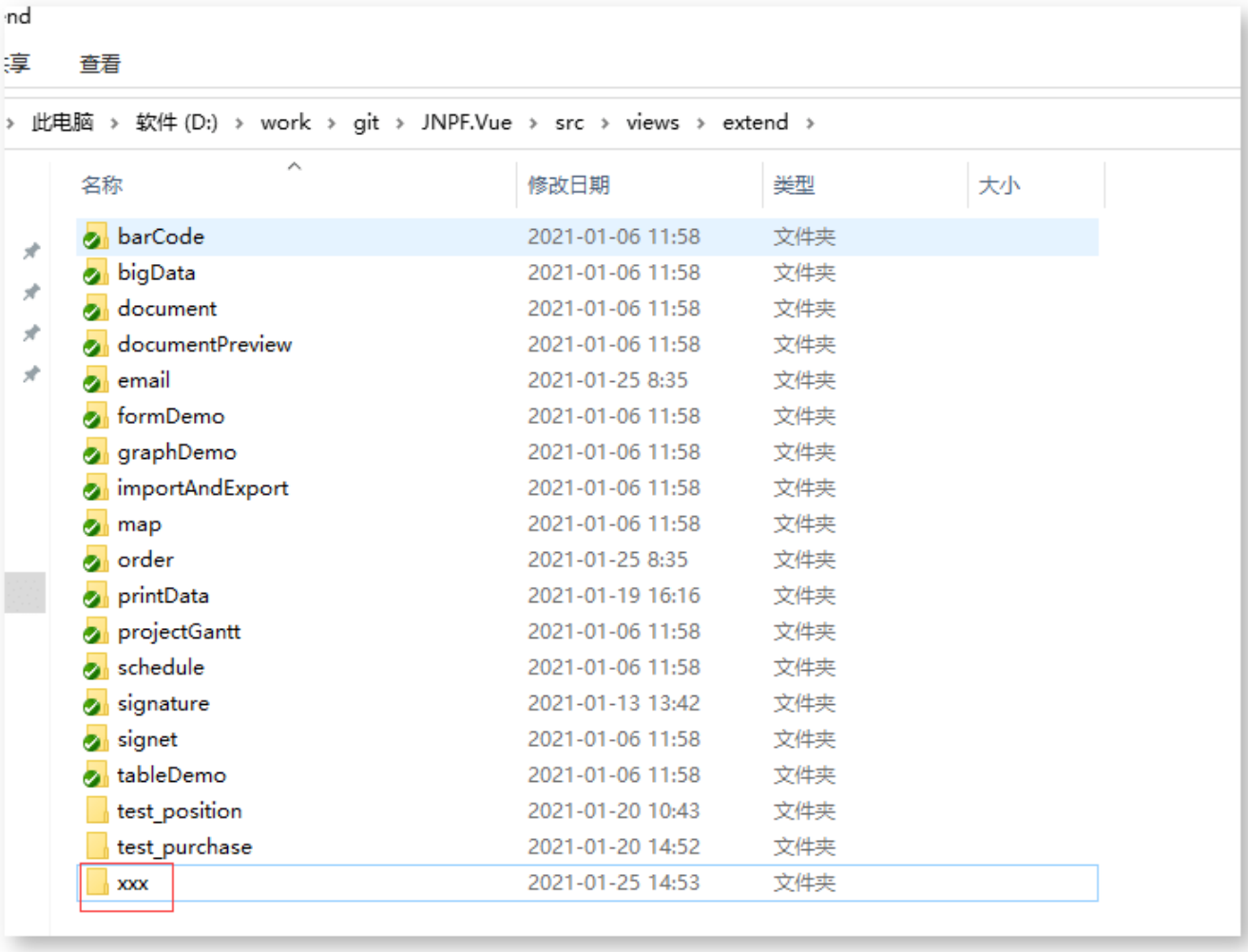
以 `扩展应用` 为例子



添加一个新功能时，可以选择在现有模块上添加，或者新建一个模块，结构如扩展应用模块所示



由于功能在扩展模块则将功能放在 `\src\views\extend` 模块下

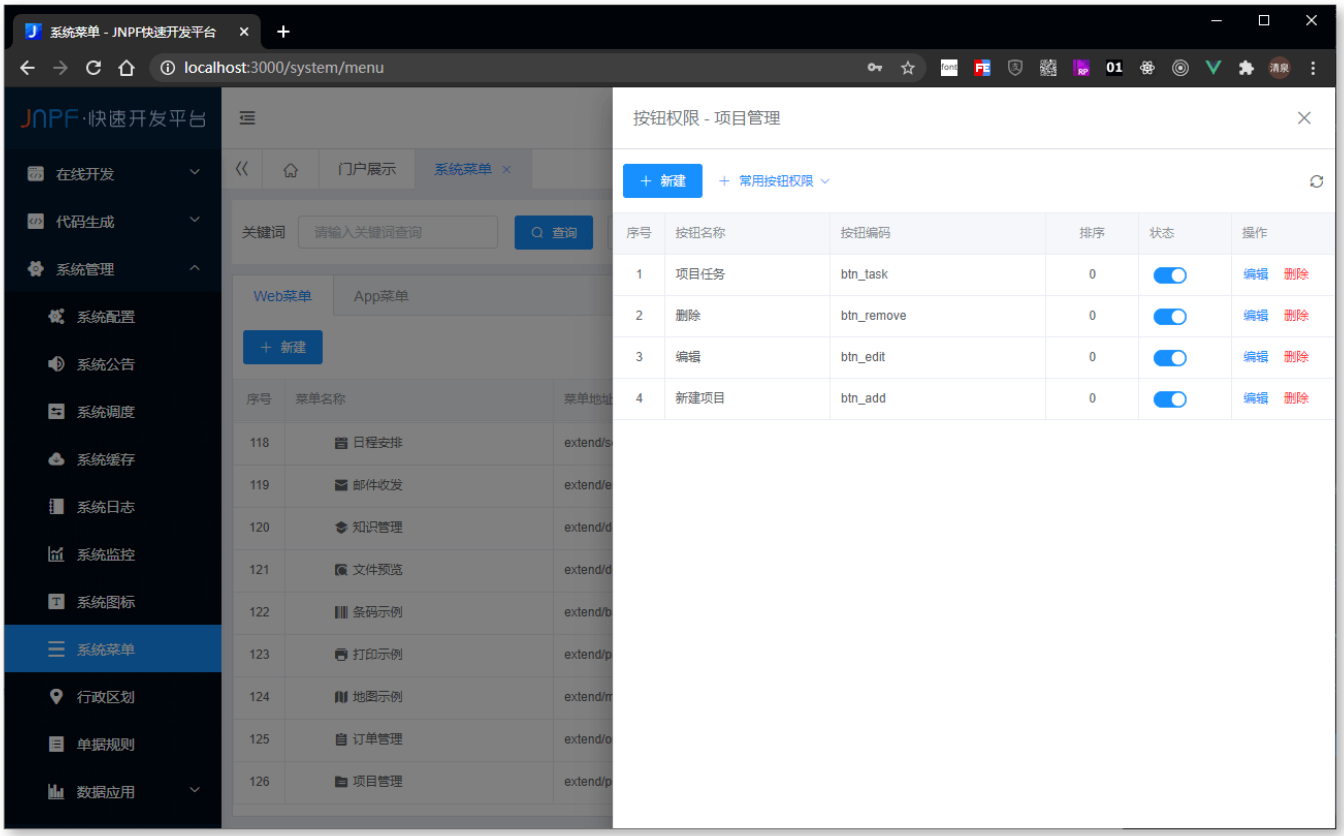


添加完之后若代码正常则可以正常显示，管理员默认不受权限控制，添加和启用菜单之后若想控制其他人员的菜单权限可以使用管理员账户在权限管理或者角色管理里面控制角色的菜单权限

按钮权限控制

示例：

1）选择菜单管理->扩展->项目管理->更多->按钮权限



2）前端代码添加

```
<el-button v-has="'btn_add'">新建项目</el-button>
```

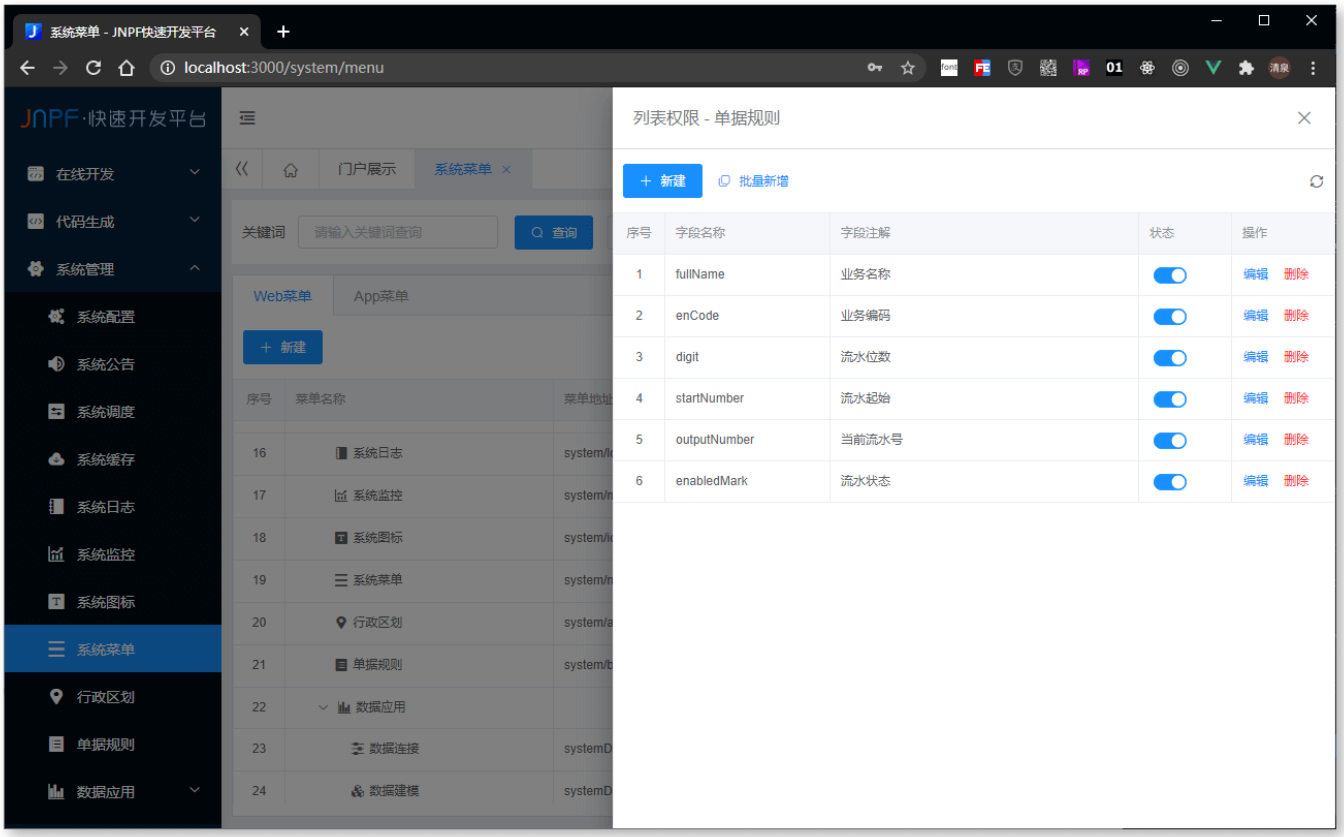
在button标签里面添加 `v-has="字段名称"` 即可实现按钮权限控制

管理员默认不受权限控制，添加和启用按钮权限之后若想控制其他人员的按钮权限可以使用管理员账户在权限管理或者角色管理里面控制角色的按钮权限

列表权限控制

示例：

- 选择菜单管理->单据规则->更多->列表权限



- 前端代码添加

```
<el-table-column prop="fullName" label="业务名称" width="200" v-if="jnpf.hasP('fullName')" />
```

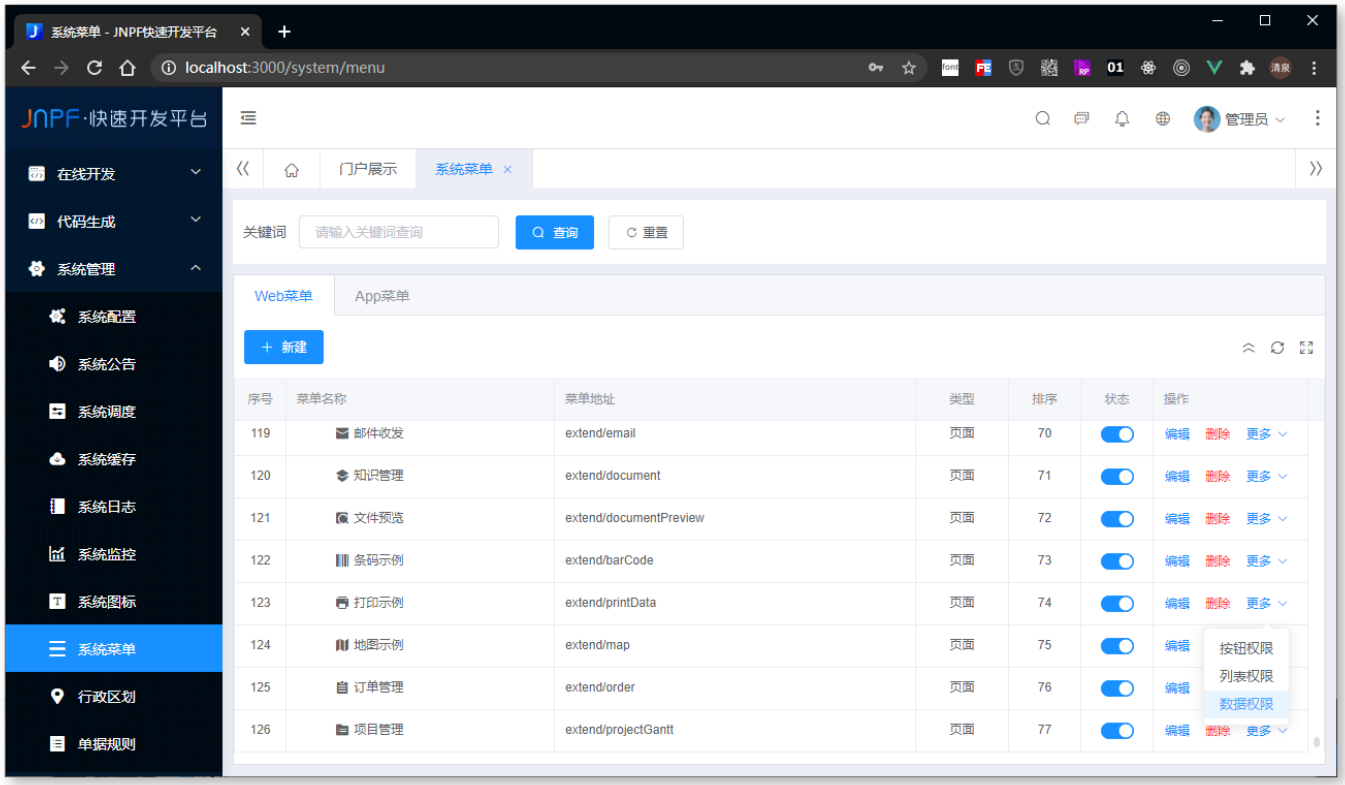
在前端列表字段上添加 `v-if="jnpf.hasP('fullName')"` ，与菜单管理中添加的字段相同，即可添加列表权限

管理员默认不受权限控制，添加和启用列表权限之后若想控制其他人员的列表权限可以使用管理员账户在权限管理或者角色管理里面控制角色的列表权限限控制，添加和启用按钮权限之后若想控制其他人员的按钮权限可以使用管理员账户在权限管理或者角色管理里面控制角色的按钮权限

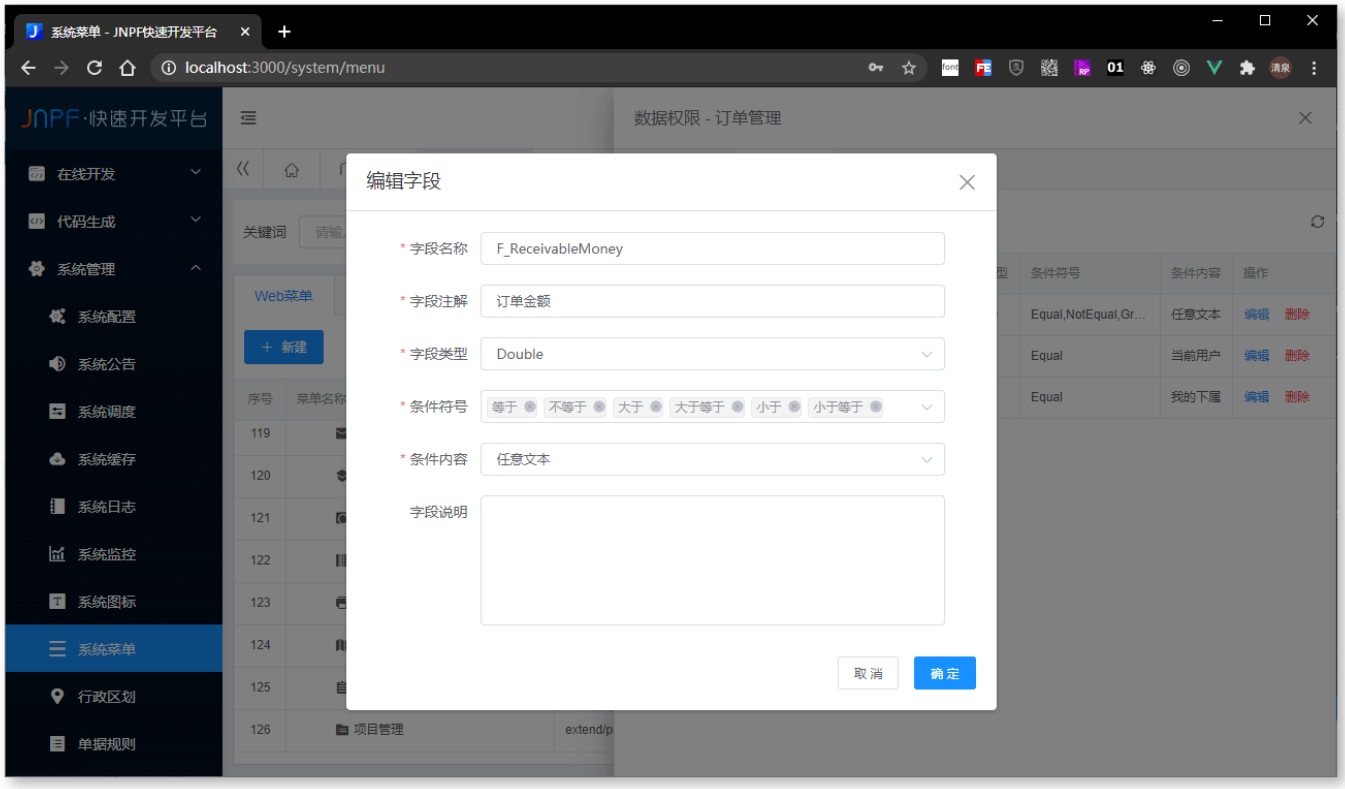
数据权限控制

> 在设置数据权限时，为避免单纯在前端设置导致数据紊乱或者出现数据bug，需要同时在数据库和后台代码中添加数据权限过滤才能生效，以下是设置数据权限的步骤。

以订单管理为例，先在页面上添加数据权限,选择菜单管理->订单管理->更多->数据权限

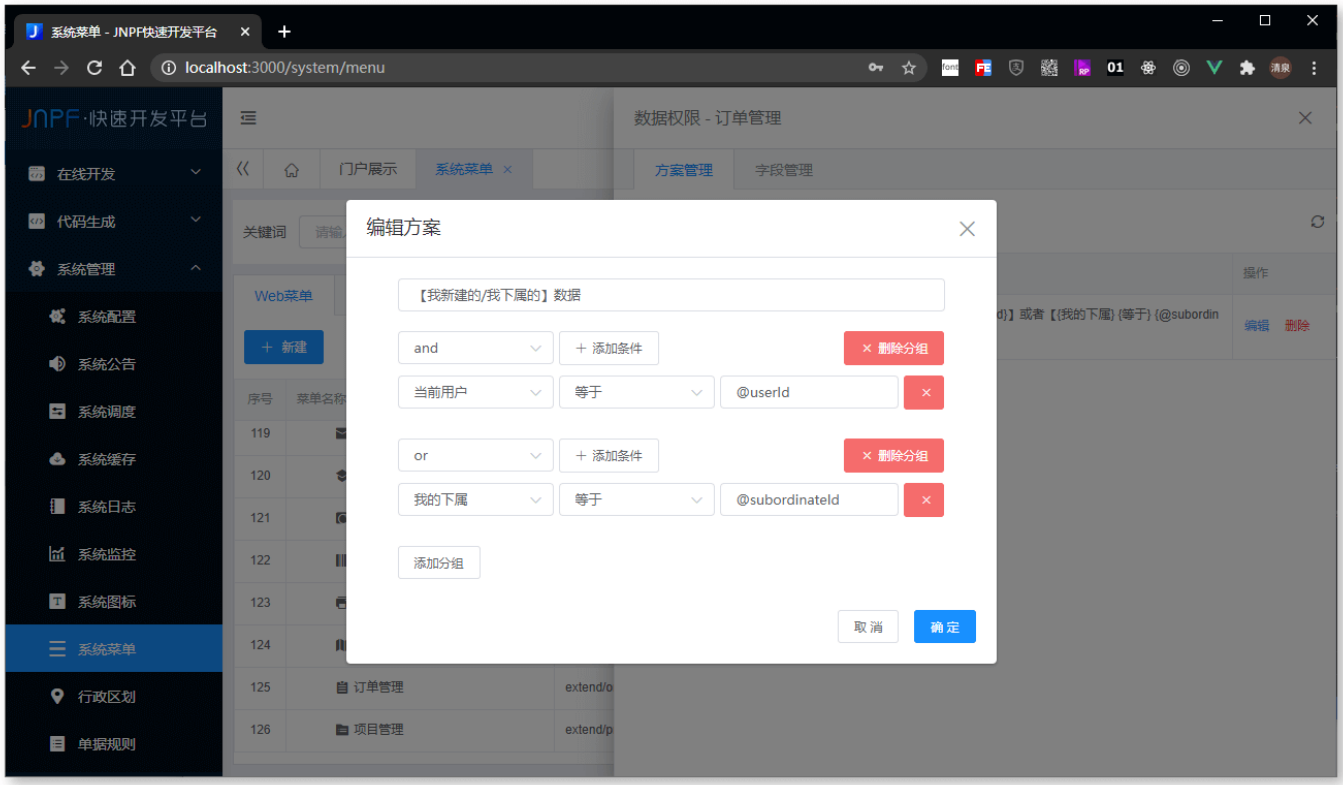


1）在 字段管理 中添加字段和查询内容

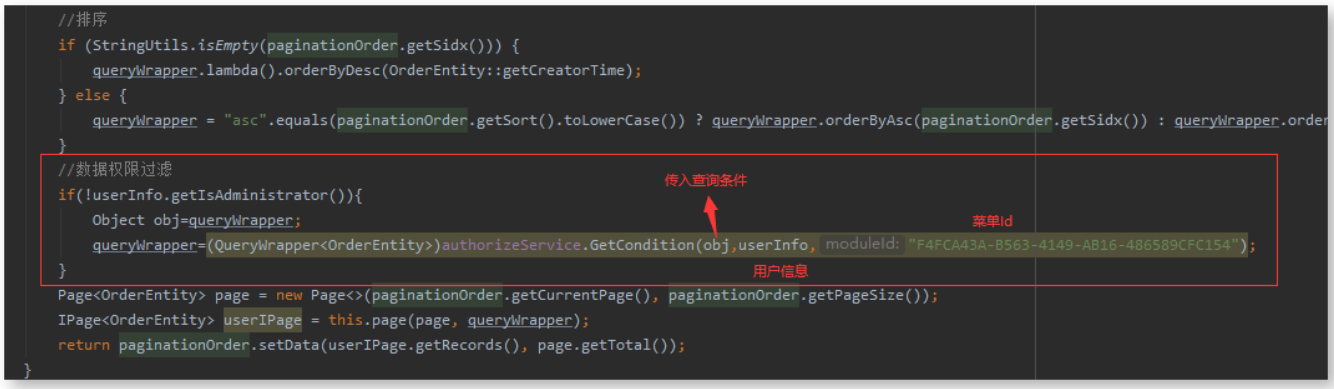


- 字段名称与数据库字段要一致

2）在 方案管理 中搭配字段组合



3）后台代码中添加以下代码



一般是在排序之前或之后添加权限过滤代码

通过框架语法查询的数据库数据添加以下代码

```
UserInfo userInfo = userProvider.get();
//数据权限过滤
if(!userInfo.getIsAdministrator()){
    Object obj=queryWrapper;
    queryWrapper=(QueryWrapper<OrderEntity>)authorizeService.GetCondition(obj,userInfo,"xxxxxxxxx");
}
```

若是通过sql拼接查询数据库数据则添加以下代码

```
UserInfo userInfo = userProvider.get();
//数据权限过滤
if (!userInfo.getIsAdministrator()) {
    sql.append(authorizeService.GetConditionSql(userInfo, "xxxxxxxxxxx"));
}
```

管理员默认不受权限控制，添加和启用数据权限之后若想控制其他人员的数据权限可以使用管理员账户在权限管理或者角色管理里面控制角色的数据权限

多租户部署文档

一、脚本准备

- `jnpf_init.sql` (初始库脚本)：用于生成租户库后同步数据，文件位于 `jnpf-databae/MySQL/`；
- `JNPF-MySQL.sql` (多租户创库脚本)：用于生成租户库表结构，文件位于 `jnpf-databae/MySQL/多租户/3.2.x/`；
- `jnpf_tenant.sql` (多租户数据库)：用于记录租户信息，文件位于 `jnpf-databae/MySQL/多租户/`；

二、导入数据库脚本

1、新建 `jnpf_init` 数据库，并将 `jnpf_init.sql` 导入（以 `新建查询` 导入）；
2、新建 `jnpf_tenant` 数据库，并将 `jnpf_tenant.sql` 导入

三、修改 `Nacos` 配置

> 配置管理 - 配置列表 - dev

- 修改 `system-config.yaml` 配置，开启多租户，

```
MultiTenancy: false
MultiTenancyUrl: http://127.0.0.1:30000/api/tenant/DbName/
```

- 修改 `tenant.yaml` 配置

```
spring:
# 多租户创库脚本目录（JNPF-MySQL.sql所在路径）
file: /www/wwwroot/jnpf-server/jnpf-tenant
redis:
  database: 1
  host: 127.0.0.1
  port: 6379
```

```
  password:
# 数据源
dataType: mysql
datasource:
  # 初始库
  dbinit: jnpf_init
  # 租户库
  dbname: jnpf_tenant
  url: jdbc:mysql://192.168.0.10:3306/{dbName}?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8&nullCatalogMeansCurrent=true
  username: root
  password: 123456
  driver-class-name: com.mysql.cj.jdbc.Driver
```

四、多租户管理前端页面配置

> 打开 jnpf-web-tenant 项目并安装项目依赖

- 开发环境

打开 `src/utils/define.js` ,修改 开发环境接口配置

```
// 开发环境接口配置
const APIURl = 'http://192.168.0.26:30000'
```

- 测试环境

编辑根目录下 `.env.staging` 文件

```
# 测试默认配置
ENV = 'staging'
VUE_APP_BASE_API = 'http://192.168.0.25'
```

修改后在命令行运行 `yarn build:staging` 打包发布项目代码

- 生产环境

编辑根目录下 `.env.production` 文件

```
#  生产默认配置
ENV = 'production'
VUE_APP_BASE_API = 'https://tenant.java.jnpfsoft.com'
```

修改后在命令行运行 `yarn build` 打包发布项目代码

五、多租户管理前端页面部署

多租户前端需要单独部署，nginx核心配置如下

```
    #JNPF-Start
    #设置上传文件的大小
    client_max_body_size 100m;

    #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    client_header_buffer_size 128k;

    #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
    large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';

    # 预检命令的缓存，如果不缓存每次会发送两次请求
    add_header Access-Control-Max-Age 3600;

    #带cookie请求需要加上这个字段，并设置为true
    add_header Access-Control-Allow-Credentials true;

    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #    表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;

    # 前端伪静态配置
    location / {
      try_files $uri $uri/ /index.html;
    }

    # 多租户接口
     location /api/ {
      proxy_pass http://192.168.0.18:30000/api/;
```

```
}

#JNPF-End
```

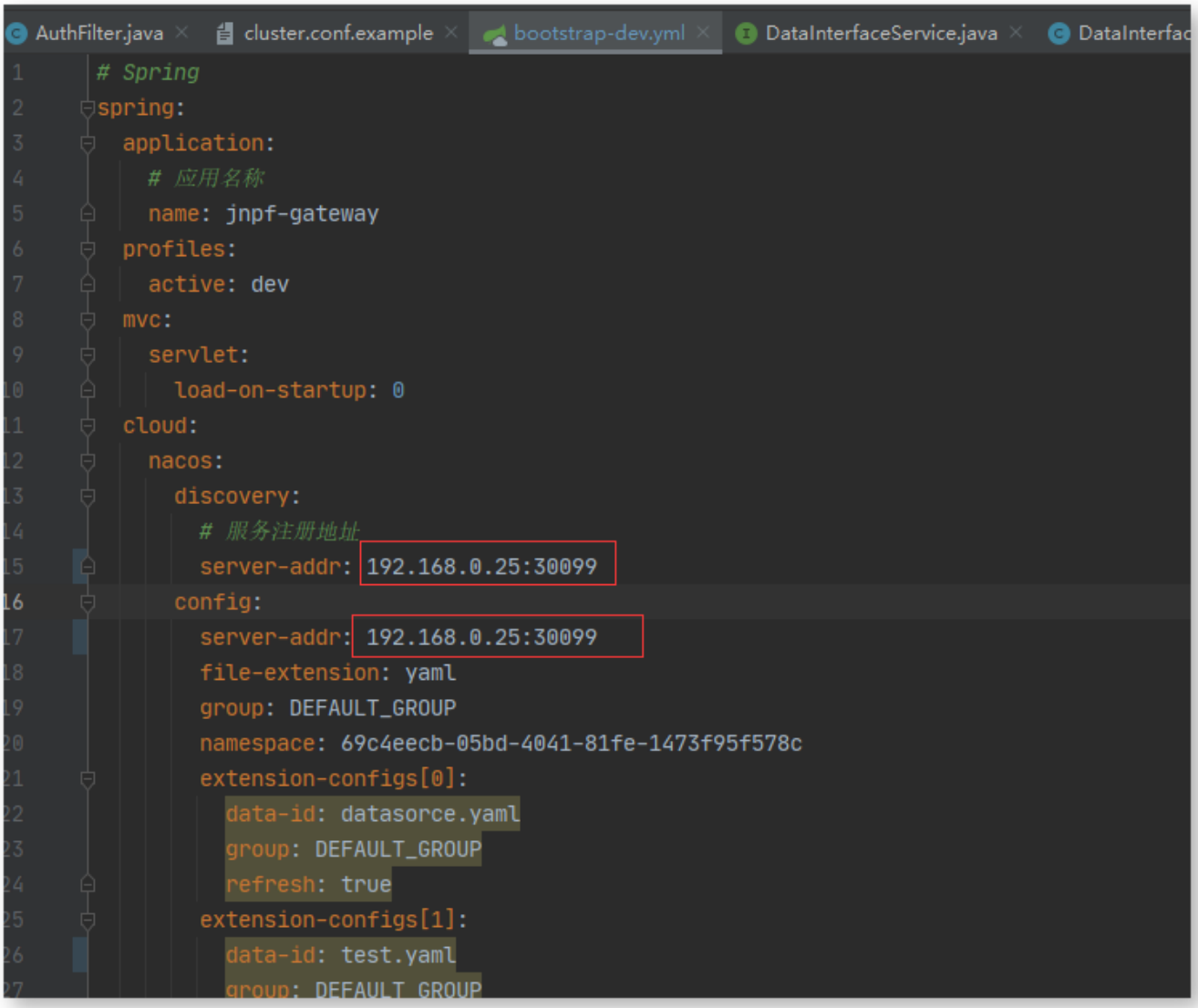
多人协同开发(网关)

以扩展模块为例，只需启动前端界面、网关和本地的扩展服务，配置调整情况：

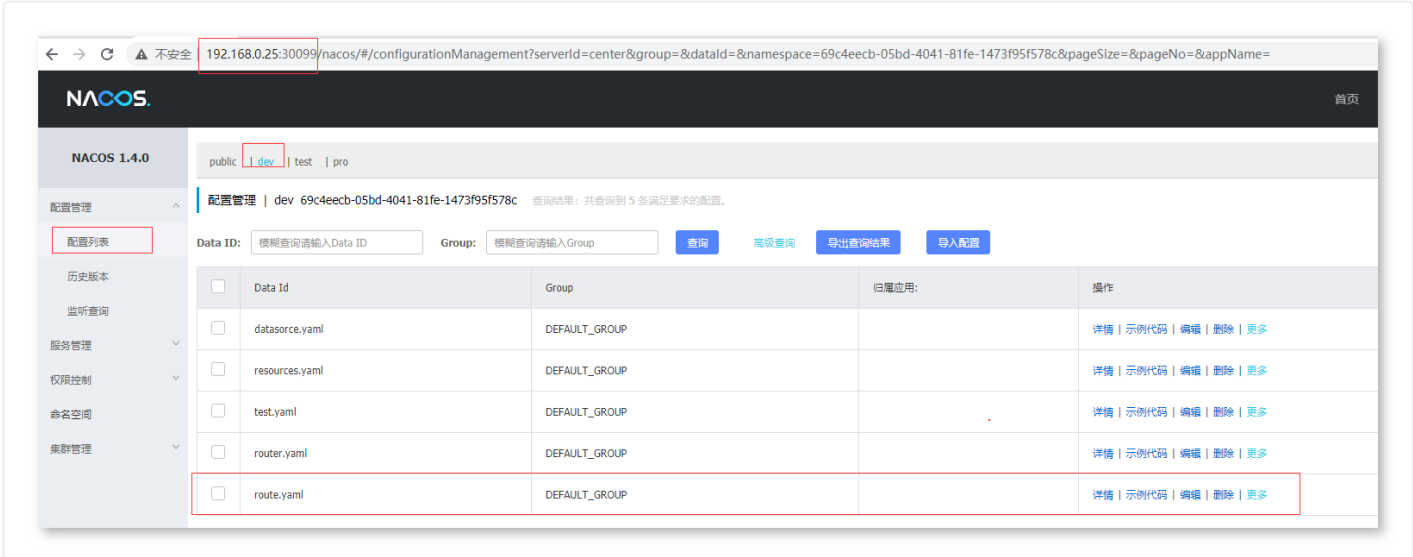
网关配置文件有两种配置方式

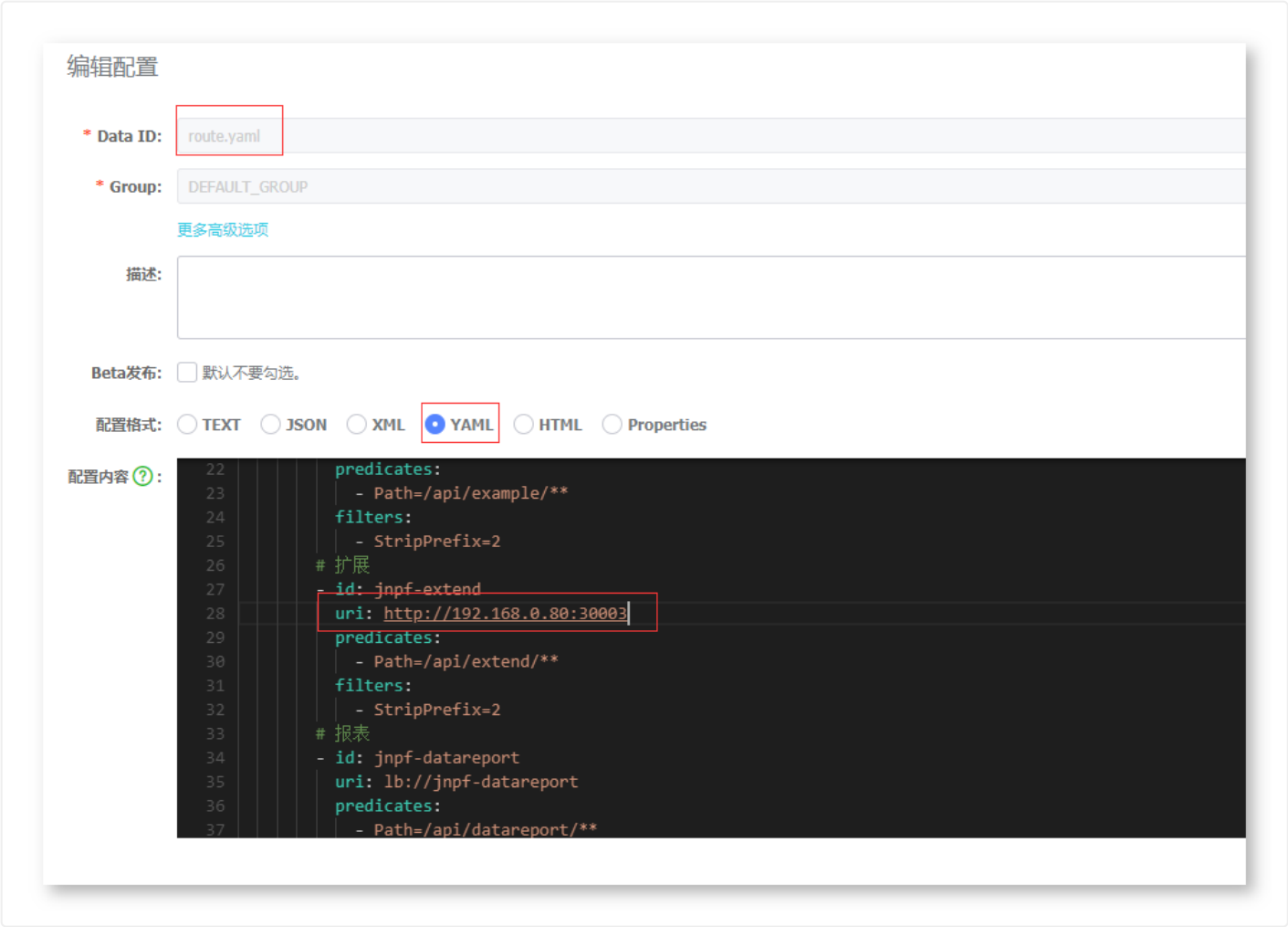
第一种方式

1.修改 `nacos` 地址，将网关中 `bootstrap-dev.yml` 中原本使用自己的 `nacos` 注册地址修改为内网 `nacos` 地址（所有的都要修改，包括 `sentinel` ），如：



2.在 `nacos` 上创建一个自己使用的路由规则，路由规则先复制原有的，然后稍作修改，如下：





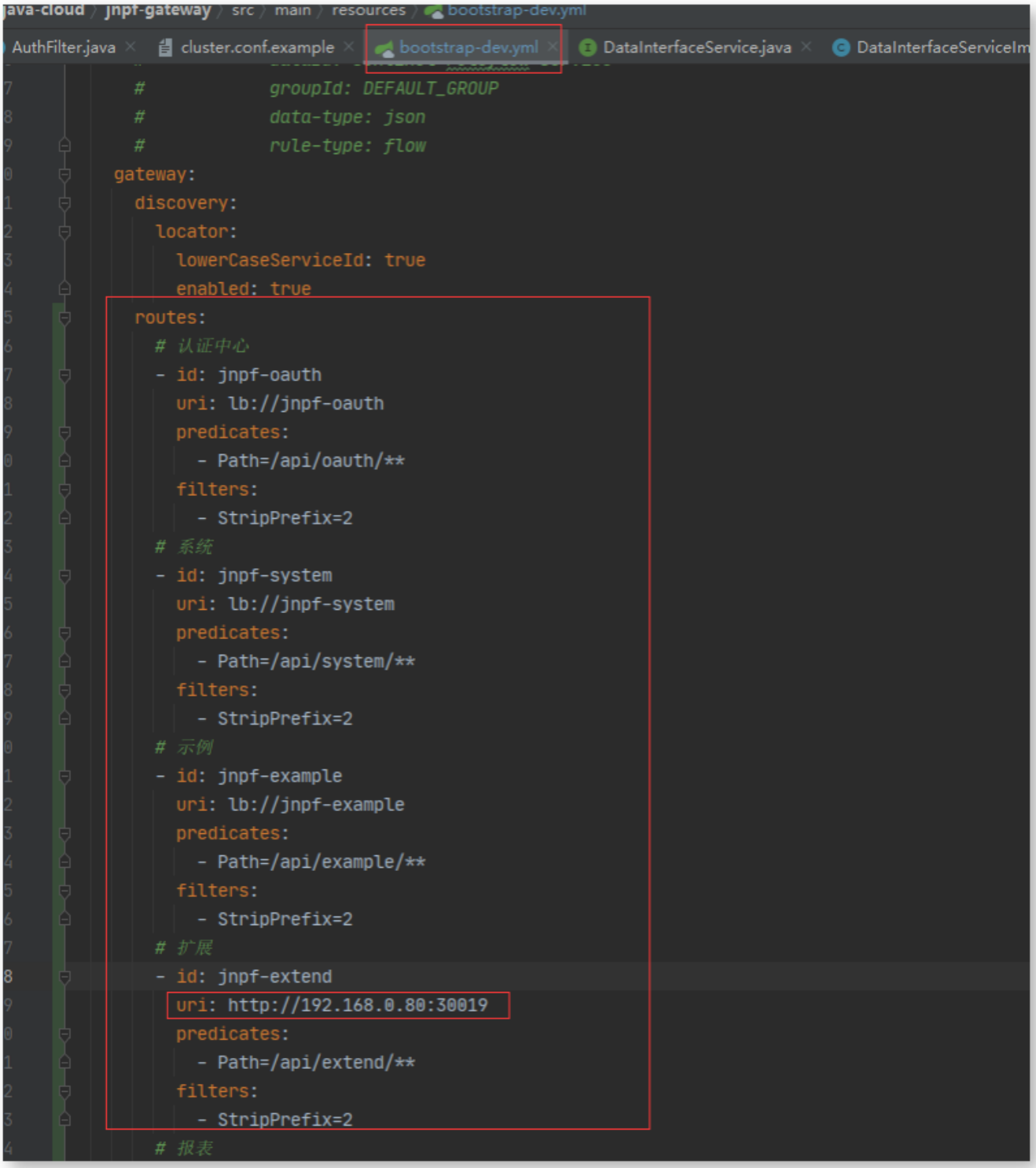
3.修改读取的路由配置，配置如下：

```yaml
    server-addr: 192.168.0.25:30099
  config:
    server-addr: 192.168.0.25:30099
    file-extension: yaml
    group: DEFAULT_GROUP
    namespace: 69c4eecb-05bd-4041-81fe-1473f95f578c
    extension-configs[0]:
      data-id: datasorce.yaml
      group: DEFAULT_GROUP
      refresh: true
    extension-configs[1]:
      data-id: test.yaml
      group: DEFAULT_GROUP
      refresh: true
    # 添加新的网关新路由
    extension-configs[2]:
      data-id: route.yaml
      group: DEFAULT_GROUP
      refresh: true
sentinel:
```

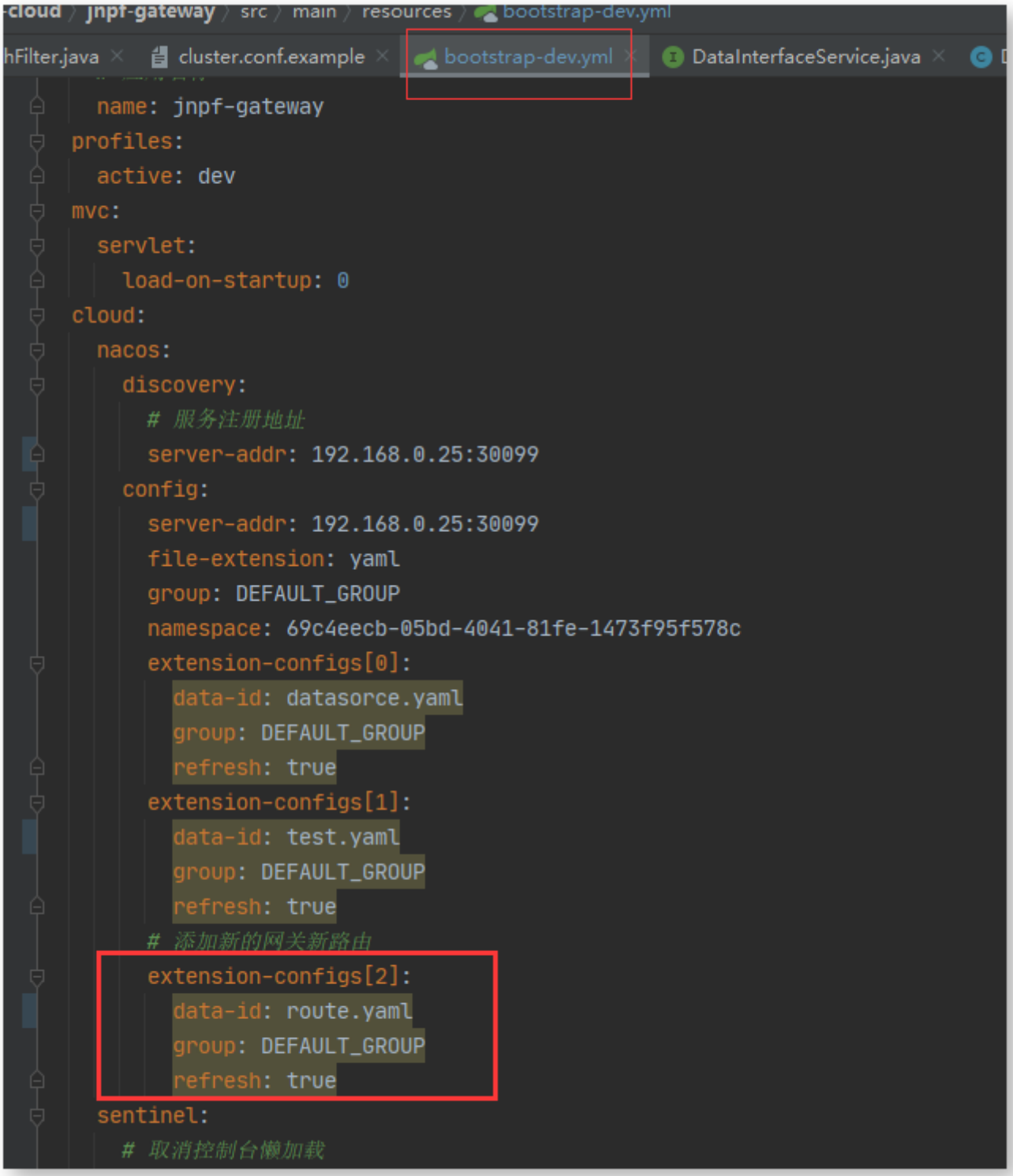
第二种配置方式（防止nacos上配置）：

1.使用第一种方式的第一点，修改 `nacos` 地址和 `sentinel` 地址

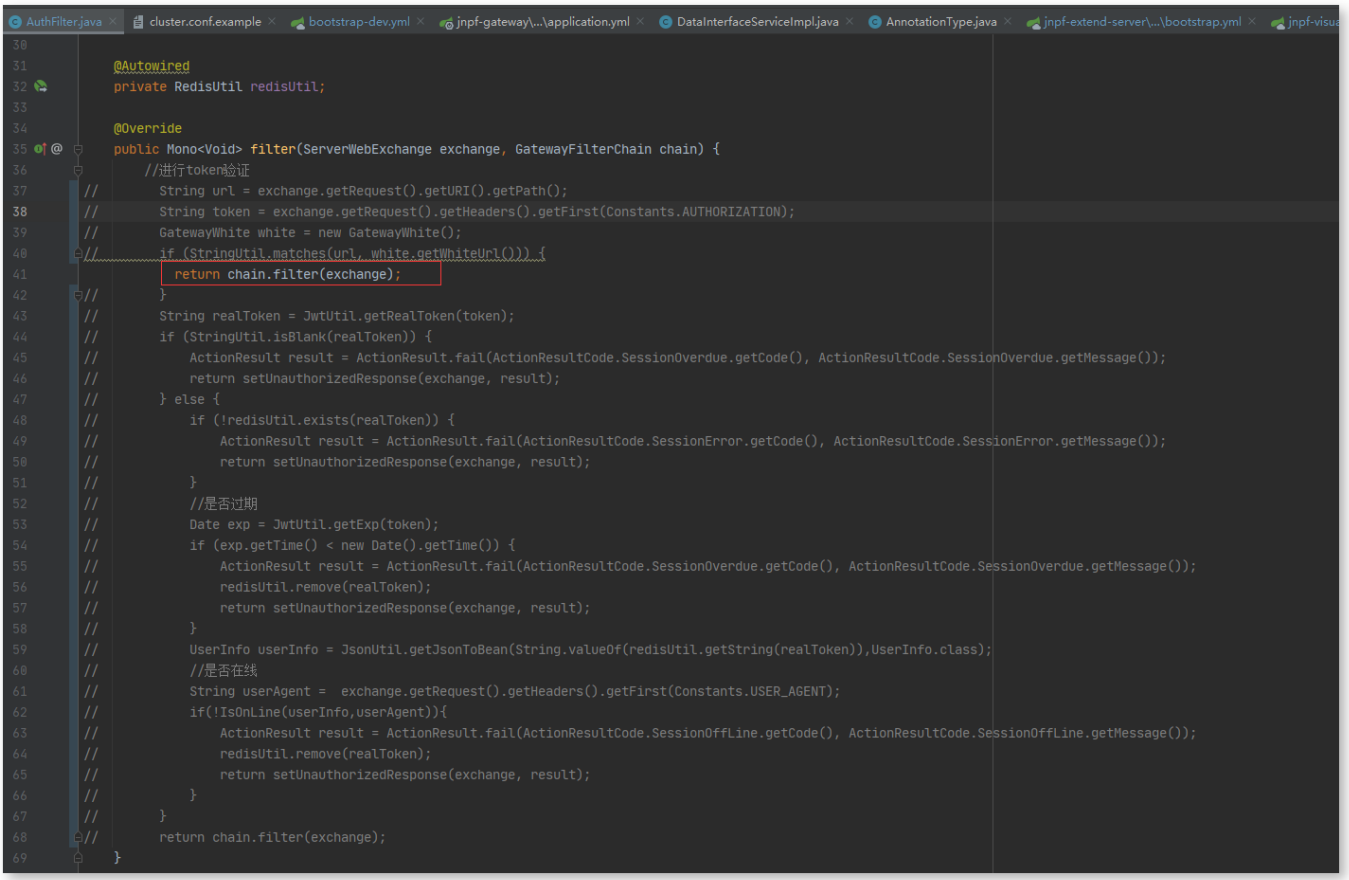
2.将 `nacos` 中的 `router.yaml` 的值复制出来，稍作处理后粘贴在 `spring: cloud: gateway:` 下的节点，如：



3.删除从 `nacos` 上读取路由的配置：



配置完成后关闭本地网关token验证，例如：

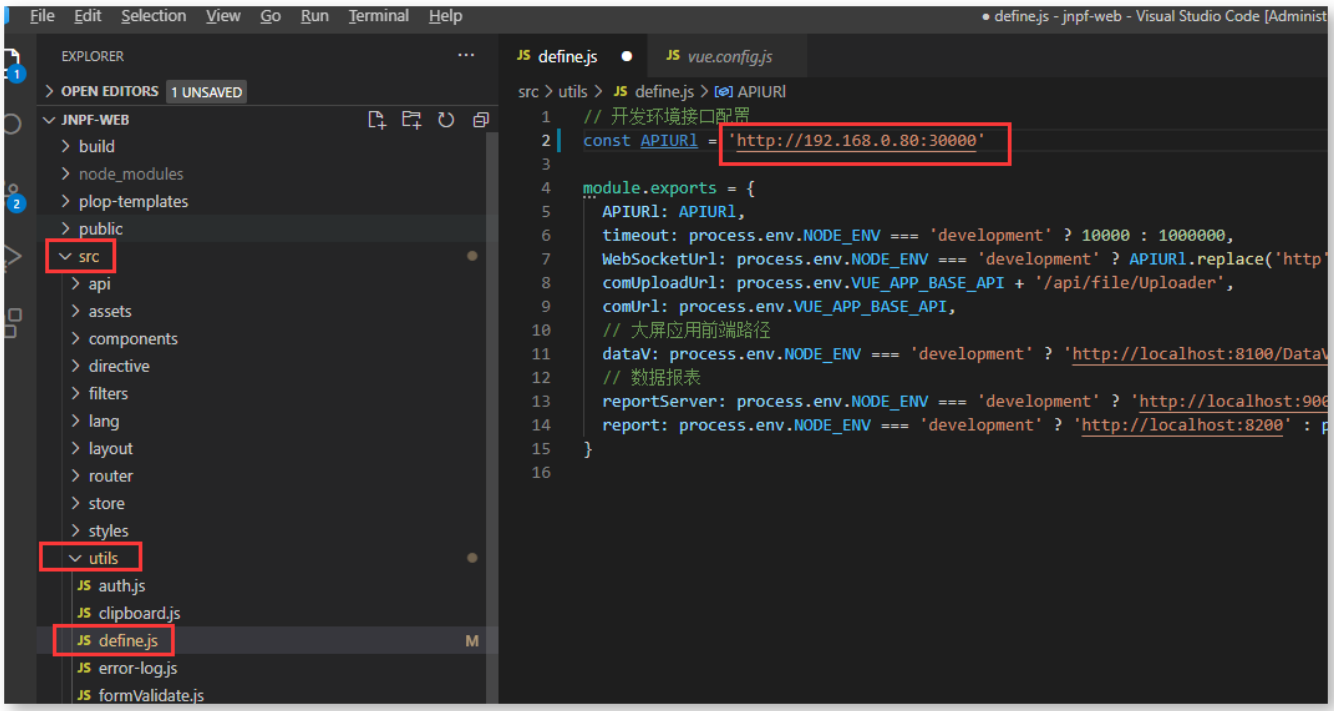


不想关闭token验证和想记录请求日志的请使用本地的系统和登录认证模块，或者将本地网关使用的 `redis` 修改为内网的 `reidis` 配置！

扩展模块配置：

将 `bootstrap.yml` 中的服务注册地址， `sentinel` 地址， `seata` 地址改为内网 `nacos` 地址

前端修改请求路径为本地的网关地址：



配置完成后即可启动前端、扩展和网关去调试本地的服务了

多人协助开发(jar)

Maven私服配置，推荐使用修改 `conf\settings.xml` 文件的方式：

- 找到 `servers` 节点，添加 `server` ,如：

```
        <server>
            <id>maven-releases</id>
            <username>admin</username>
            <password>[redacted]</password>
        </server>
        <server>
            <id>maven-snapshots</id>
            <username>admin</username>
            <password>[redacted]</password>
        </server>
        <!-- Another sample, using keys to authenticate.
        <server>
          <id>siteServer</id>
          <privateKey>/path/to/private/key</privateKey>
          <passphrase>optional; leave empty if not used.</passphrase>
        </server>
        -->
    </servers>
```

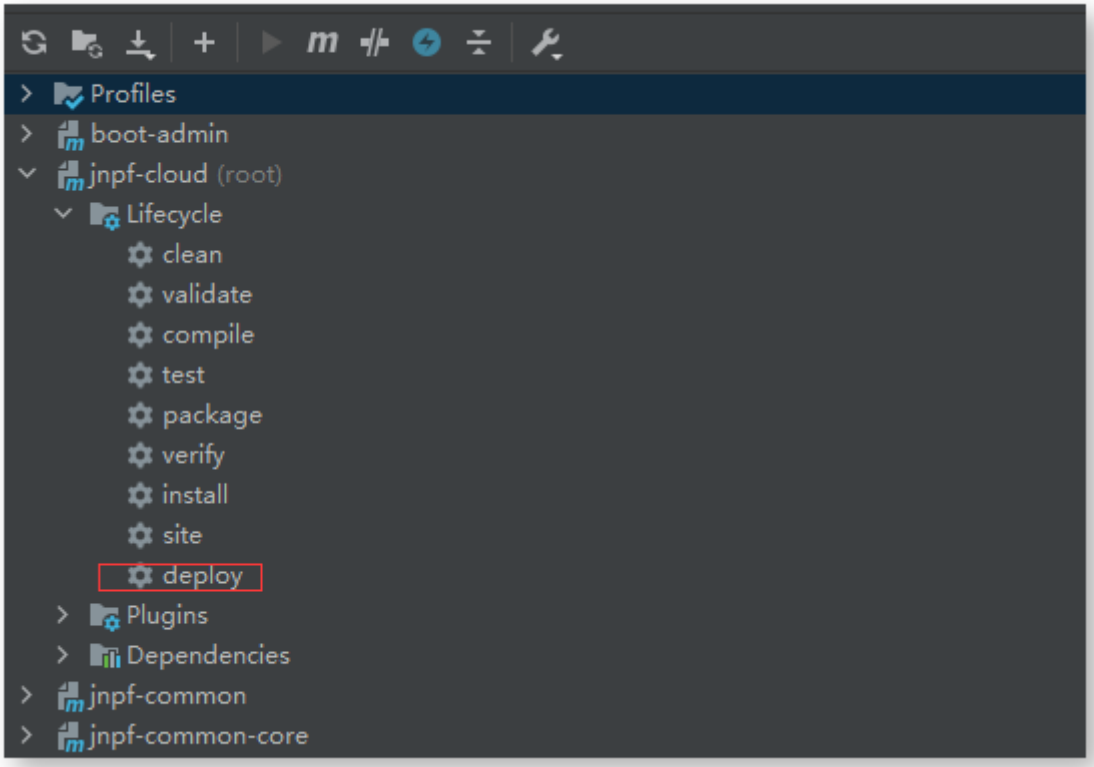
- 找到mirrors，添加私服仓库地址：

```
<!--        </mirror>-->
        <mirror>
            <id>maven-snapshots</id>
            <mirrorOf>*</mirrorOf>
            <name>maven-snapshots</name>
            <url>http://139.224.81.92:5000/repository/maven-public/</url>
        </mirror>

    </mirrors>
```

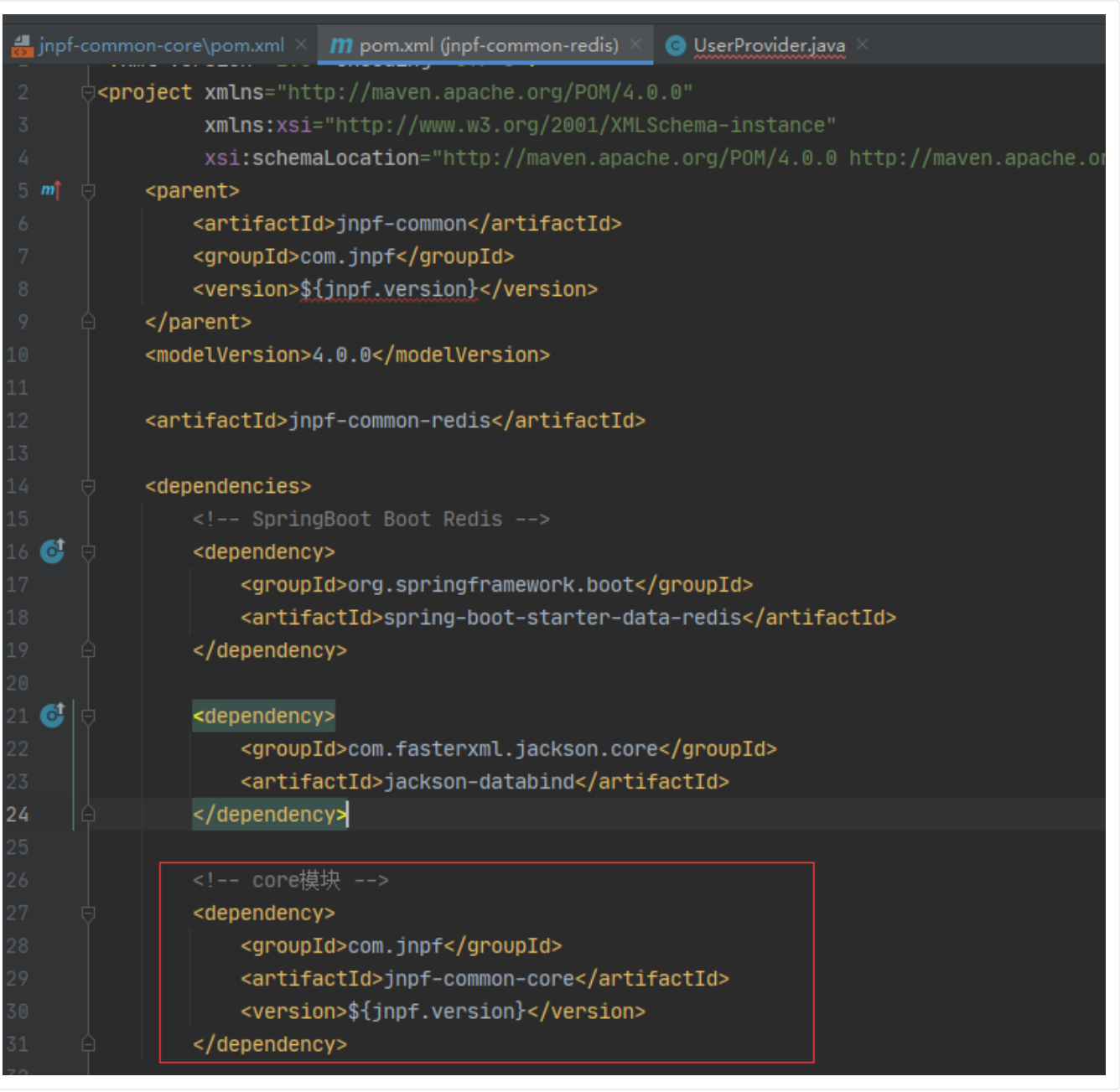
- 补全jar的版本，如图所示

```xml
<dependencies>

    <dependency>
        <groupId>com.nimbusds</groupId>
        <artifactId>nimbus-jose-jwt</artifactId>
        <version>8.16</version>
    </dependency>

    <!-- Spring Web -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>5.2.9.RELEASE</version>
    </dependency>

    <!-- Alibaba Fastjson -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>fastjson</artifactId>
        <version>1.2.74</version>
    </dependency>

    <!-- lang3 -->
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.7</version>
    </dependency>

    <!-- Java Servlet -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
    </dependency>
```

- 通过IDEA后侧的Lifecycle下的 `deploy` 将项目打包上传到私服（建议先点击 `clean` ）


Profiles
boot-admin
jnpf-cloud (root)
Lifecycle
clean
validate
compile
test
package
verify
install
site
deploy
Plugins
Dependencies
jnpf-common
jnpf-common-core

- 项目中需要使用某个项目则需要手动导入jar包，刷新Maven即可调用



minIO部署文档

> 本文档在 CentOS 环境下部署minIO

下载安装包

- 方式一：在 `Coding/项目/JNPF开发文档/文件网盘/minIO/` 目录下载安装包并上传至 `/usr/local/minio`
- 方式二：进入目录 `/usr/local/minio`，使用命令下载

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
```

部署

```
chmod +x minio
mkdir -p /home/minio/data

# 开放minio端口:9000
firewall-cmd --zone=public --add-port=30000/tcp --permanent

firewall-cmd --reload

# 方式一：启动minio服务
/usr/local/minio server /home/minio/data/

# 方式二：后台运行minio
nohup /usr/local/minio server /home/minio/data > /home/minio/data/minio.log 2>&1 &
```

官方文档地址：`https://docs.min.io/`

Skywalking配置说明

基本配置

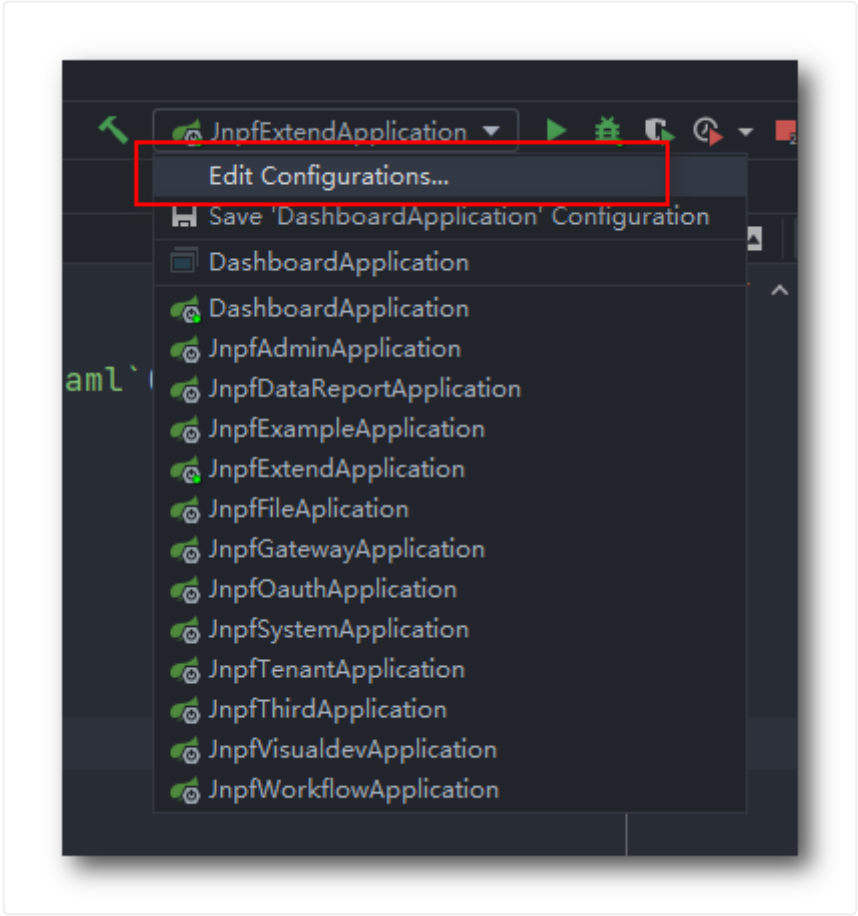
- 打开 `skywalking/config/application.yml` ,修改下数据源配置(`第164-166行`)
- 创建 `skywalking` 初始表
  - 运行 `skywalking/bin/oapServiceInit.bat` (windows环境)
  - 运行 `skywalking/bin/oapServiceInit.sh` (Linux、Mac环境)

集成到项目中

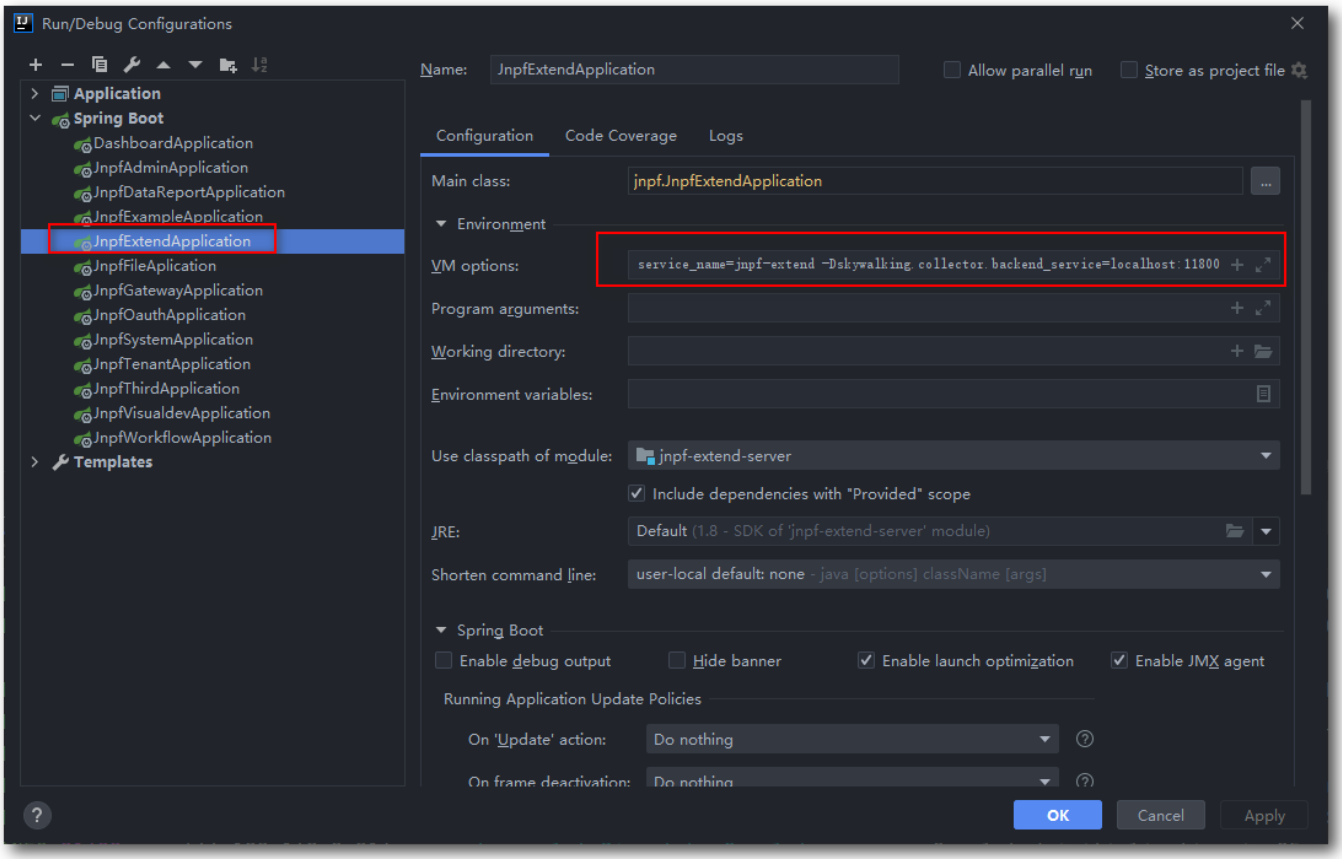
开发环境

> 本文档以 `jnpf-extend` 以例

在 `IDEA` 中打开 `运行配置`

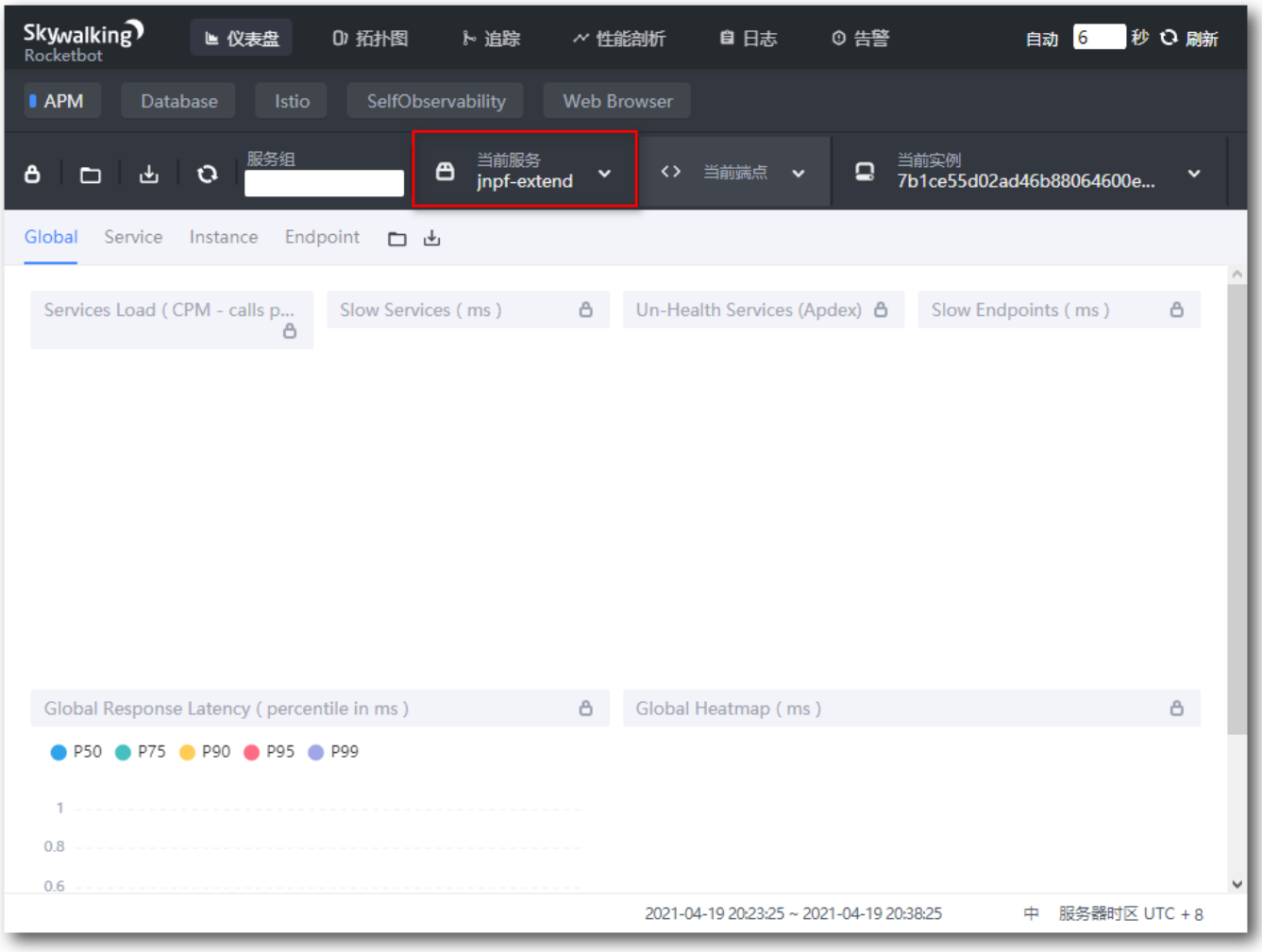


在右侧 `Configuration` - `Environment` 下的 `VM options` 添加如下配置



```
-Xmx1500m
-Xms1500m
-Xmn1180m
-Xss1024k
-javaagent:D:\Code\jnpf-java-cloud\jnpf-registry\skywalking\agent\skywalking-agent.jar
-Dskywalking.agent.service_name=jnpf-extend
-Dskywalking.collector.backend_service=localhost:11800
```

配置保存后启动或重启 `jnpf-extend` 扩展服务，然后打开 `Skywalking` 控制查看配置结果



> 具体配置说明

1、JVM内存配置（可根据实际情况调整)）

```
-Xmx1500m
-Xms1500m
-Xmn1180m
-Xss1024k
```

2、`skywalking-agent`路径(绝对路径)

```
-javaagent:D:\Code\jnpf-java-cloud\jnpf-registry\skywalking\agent\skywalking-agent.jar
```

3、服务名

```
-Dskywalking.agent.service_name=jnpf-extend
```

4、`gRPCPort`端口，默认端口`11800`，

```
-Dskywalking.collector.backend_service=localhost:11800
```

其他服务类似操作，具体配置参考如下：

> 配置完成后重启相应的服务

- `jnpf-gateway`网关

```
  -Xmx400m
  -Xms400m
  -Xmn150m
  -Xss1024k
  -javaagent:D:\Code\jnpf-java-cloud\jnpf-registry\skywalking\agent\skywalking-agent.jar
  -Dskywalking.agent.service_name=jnpf-gateway
  -Dskywalking.collector.backend_service=localhost:11800
```

- `jnpf-oauth`认证服务

```
  -Xmx500m
  -Xms500m
  -Xmn150m
  -Xss1024k
  -javaagent:D:\Code\jnpf-java-cloud\jnpf-registry\skywalking\agent\skywalking-agent.jar
  -Dskywalking.agent.service_name=jnpf-oauth
  -Dskywalking.collector.backend_service=localhost:11800
```

- `jnpf-system`系统基础服务

```
  -Xmx500m
  -Xms500m
  -Xmn180m
```

```
  -Xss1024k
  -javaagent:D:\Code\jnpf-java-cloud\jnpf-registry\skywalking\agent\skywalking-a
gent.jar
  -Dskywalking.agent.service_name=jnpf-system
  -Dskywalking.collector.backend_service=localhost:11800
```

- `jnpf-visualdev` 可视化开发

```
  -Xmx800m
  -Xms800m
  -Xmn300m
  -Xss1024k
  -javaagent:D:\Code\jnpf-java-cloud\jnpf-registry\skywalking\agent\skywalking-a
gent.jar
  -Dskywalking.agent.service_name=jnpf-visualdev
  -Dskywalking.collector.backend_service=localhost:11800
```

- `jnpf-workflow` 工作流

```
  -Xmx800m
  -Xms800m
  -Xmn300m
  -Xss1024k
  -javaagent:D:\Code\jnpf-java-cloud\jnpf-registry\skywalking\agent\skywalking-a
gent.jar
  -Dskywalking.agent.service_name=jnpf-jnpf-workflow
  -Dskywalking.collector.backend_service=localhost:11800
```

- `jnpf-file` 文件服务

```
  -Xmx200m
  -Xms200m
  -Xmn100m
  -Xss1024k
  -javaagent:D:\Code\jnpf-java-cloud\jnpf-registry\skywalking\agent\skywalking-a
gent.jar
  -Dskywalking.agent.service_name=jnpf-file
  -Dskywalking.collector.backend_service=localhost:11800
```

- `jnpf-tenant` 租户服务

```
  -Xmx200m
  -Xms200m
  -Xmn100m
```

```
  -Xss1024k
  -javaagent:D:\Code\jnpf-java-cloud\jnpf-registry\skywalking\agent\skywalking-a
gent.jar
  -Dskywalking.agent.service_name=jnpf-tenant
  -Dskywalking.collector.backend_service=localhost:11800
```

- `jnpf-datareport` 报表服务

```
  -Xmx800m
  -Xms800m
  -Xmn300m
  -Xss1024k
  -javaagent:D:\Code\jnpf-java-cloud\jnpf-registry\skywalking\agent\skywalking-a
gent.jar
  -Dskywalking.agent.service_name=jnpf-datareport
  -Dskywalking.collector.backend_service=localhost:11800
```

- `jnpf-extend` 扩展

```
  -Xmx1500m
  -Xms1500m
  -Xmn1180m
  -Xss1024k
  -javaagent:D:\Code\jnpf-java-cloud\jnpf-registry\skywalking\agent\skywalking-a
gent.jar
  -Dskywalking.agent.service_name=jnpf-extend
  -Dskywalking.collector.backend_service=localhost:11800
```

- `jnpf-third` 第三方应用

```
  -Xmx200m
  -Xms200m
  -Xmn80m
  -Xss1024k
  -javaagent:D:\Code\jnpf-java-cloud\jnpf-registry\skywalking\agent\skywalking-a
gent.jar
  -Dskywalking.agent.service_name=jnpf-third
  -Dskywalking.collector.backend_service=localhost:11800
```

测试生产环境

测试生产环境中，在服务启动命令中添加

- `jnpf-gateway` 网关

```
  nohup java -Xmx400m -Xms400m -Xmn150m -Xss1024k -javaagent:/www/wwwroot/jnpf-registry/skywalking-server/agent/skywalking-agent.jar -Dskywalking.agent.service_name=jnpf-gateway -Dskywalking.collector.backend_service=localhost:11800 -jar jnpf-gateway-3.1.0-SNAPSHOT.jar >Log.log 2>&1 &
```

- `jnpf-oauth` 认证服务

```
  nohup java -Xmx200m -Xms200m -Xmn80m -Xss1024k -javaagent:/www/wwwroot/jnpf-registry/skywalking-server/agent/skywalking-agent.jar -Dskywalking.agent.service_name=jnpf-oauth -Dskywalking.collector.backend_service=localhost:11800 -jar jnpf-oauth-3.1.0-SNAPSHOT.jar >Log.log 2>&1 &
```

- `jnpf-system` 系统基础服务

```
  nohup java -Xmx500m -Xms500m -Xmn180m -Xss1024k -javaagent:/www/wwwroot/jnpf-registry/skywalking-server/agent/skywalking-agent.jar -Dskywalking.agent.service_name=jnpf-system -Dskywalking.collector.backend_service=localhost:11800 -jar jnpf-system-3.1.0-SNAPSHOT.jar >Log.log 2>&1 &
```

- `jnpf-visualdev` 可视化开发

```
  nohup java -Xmx800m -Xms800m -Xmn300m -Xss1024k -javaagent:/www/wwwroot/jnpf-registry/skywalking-server/agent/skywalking-agent.jar -Dskywalking.agent.service_name=jnpf-visualdev -Dskywalking.collector.backend_service=localhost:11800 -jar jnpf-visualdev-3.1.0-SNAPSHOT.jar >Log.log 2>&1 &
```

- `jnpf-workflow` 工作流

```
  nohup java -Xmx800m -Xms800m -Xmn300m -Xss1024k -javaagent:/www/wwwroot/jnpf-registry/skywalking-server/agent/skywalking-agent.jar -Dskywalking.agent.service_name=jnpf-workflow -Dskywalking.collector.backend_service=localhost:11800 -jar jnpf-workflow-3.1.0-SNAPSHOT.jar >Log.log 2>&1 &
```

- `jnpf-file` 文件服务

```
  nohup java -Xmx200m -Xms200m -Xmn100m -Xss1024k -javaagent:/www/wwwroot/jnpf-registry/skywalking-server/agent/skywalking-agent.jar -Dskywalking.agent.service_name=jnpf-file -Dskywalking.collector.backend_service=localhost:11800 -jar jnpf-file-3.1.0-SNAPSHOT.jar >Log.log 2>&1 &
```

- `jnpf-tenant` 租户服务

```
  nohup java -Xmx200m -Xms200m -Xmn80m -Xss1024k -javaagent:/www/wwwroot/jnpf-registry/skywalking-server/agent/skywalking-agent.jar -Dskywalking.agent.service_name=jnpf-tenant -Dskywalking.collector.backend_service=localhost:11800 -jar jnpf-tenant-3.1.0-SNAPSHOT.jar >Log.log 2>&1 &
```

- `jnpf-datareport` 报表服务

```
  nohup java -Xmx800m -Xms800m -Xmn300m -Xss1024k -javaagent:/www/wwwroot/jnpf-registry/skywalking-server/agent/skywalking-agent.jar -Dskywalking.agent.service_name=jnpf-datareport -Dskywalking.collector.backend_service=localhost:11800 -jar jnpf-datareport-3.1.0-SNAPSHOT.jar >Log.log 2>&1 &
```

- `jnpf-extend` 扩展

```
  nohup java -Xmx200m -Xms200m -Xmn80m -Xss1024k -javaagent:/www/wwwroot/jnpf-registry/skywalking-server/agent/skywalking-agent.jar -Dskywalking.agent.service_name=jnpf-extend -Dskywalking.collector.backend_service=localhost:11800 -jar jnpf-extend-3.1.0-SNAPSHOT.jar >Log.log 2>&1 &
```

- `jnpf-third` 第三方应用

```
  nohup java -Xmx200m -Xms200m -Xmn80m -Xss1024k -javaagent:/www/wwwroot/jnpf-registry/skywalking-server/agent/skywalking-agent.jar -Dskywalking.agent.service_name=jnpf-third -Dskywalking.collector.backend_service=localhost:11800 -jar jnpf-third-3.1.0-SNAPSHOT.jar >Log.log 2>&1 &
```

jnpf-java-license配置

环境要求

- jdk-8u151 +
- Maven 3.6.3+

项目模块说明

- `jnpf-license-core`： 核心配置及工具类.
- `jnpf-license-provider`： 获取系统硬件信息及license证书的生成
- `jnpf-valida-spring-boot-starter`： 证书校验

使用JDK自带的 `keytool` 命令生成公钥和私钥

1. keytool参数说明

参数	说明
-genkeypair	生成密钥对
-keysize	密钥位大小
-validity	有效天数
-alias	别名
-keystore	密钥库名称
-storepass	密钥库口令
-keypass	密钥口令
-dname	指定证书拥有者信息
-exportcert	导出证书
-file	参数指定导出到文件的文件名
-import	将已签名数字证书导入密钥库

1. 生成私有keystore

```
keytool -genkeypair -keysize 2048 -validity 3650 -alias "privateKey" -keystore "privateKeys.keystore" -storepass "public_password1234" -keypass "private_password1234" -dname "CN=localhost, OU=localhost, O=localhost, L=SH, ST=SH, C=CN"
```

1. 导出证书

```
keytool -exportcert -alias "privateKey" -keystore "privateKeys.keystore" -storepass "public_password1234" -file "certfile.cer"
```

1. 将导出的证书导入至私有keystore生成对应的公有keystore

```
keytool -import -alias "publicCert" -file "certfile.cer" -keystore "publicCerts.keystore" -storepass "public_password1234"
```

在项目使用

1. 生成证书

- 将生成的 `privateKeys.keystore` 复制到 `jnpf-license-provider` 的 `resources` 下
- 打开 `jnpf-license-provider\src\main\resources\application.yml` ，调整证书生成路径，默认是在 `D:/license/` 下（需求在D盘下建立 `license` 文件夹）
- 启动 `jnpf-license-provider` ,将得到的硬件信息组装成对应的 `model` ,可查看 `jnpf-java-license\jnpf-license-core\src\main\java\jnpf\license\model\LicenseModelCreate.java` 中的属性和注释进行操作,可以看到属性说明

```
{
  "subject":"jnpf",
  "privateAlias":"privateKey",
  "keyPass":"private_password1234",
  "storePass":"public_password1234",
  "privateKeysStorePath":"/privateKeys.keystore",
  "issuedTime":"2020-05-0108:30:00",
  "expiryTime":"2021-08-0108:30:00",
  "description":"系统软件许可证书",
  "licensePath":"D:/license/license.lic",
  "licenseModel":{
    "ipCheck":true,
    "idDeviceCheck":true,
    "isBoardCheck":true,
    "isCpuCheck":true,
    "macCheck":true,
    "ipAddress":[
      "192.168.0.151"
    ],
    "deviceAddress":[
      "2C-F0-5D-ED-2E-71"
    ],
    "cpuSerial":"BFEBFBFF000A0653",
    "mainBoardSerial":"L3G0015695"
  }
}
```

- 访问 `http://localhost:8080/generateLicense` 生成证书即可.

1. 校验证书

- 把 `license-valida-spring-boot-starter` 打成jar包后上传到本地仓库或上传到私服，导入到需要校验证书的项目 `pom.xml` 中

1. 将生成后的 `D:/license/license.lic` 提供给使用者，然后在对应的项目中的 `application.yml` 文件中添加如下配置

```
jnpf:
  license:
    subject: jnpf
    publicAlias: publicCert
    storePass: public_password1234
    licensePath: D:/license.lic
    publicKeysStorePath: publicCerts.keystore
```

消息中心说明

消息调用api接口，传入对应的实体对象

```
调用系统消息Api
Author: JNPF开发平台组
Version: V3.1.0
@FeignClient(name = FeignName.MESSAGE_SERVER_NAME, fallback = SentMessageApiFallback.class,path = "/MessageAll")
public interface SentMessageApi {

    发送消息
    Params: sentMessageForm –
    Returns:
    @PostMapping
    void sendMessage(@RequestBody SentMessageForm sentMessageForm);

}
```

实体对象参数说明：

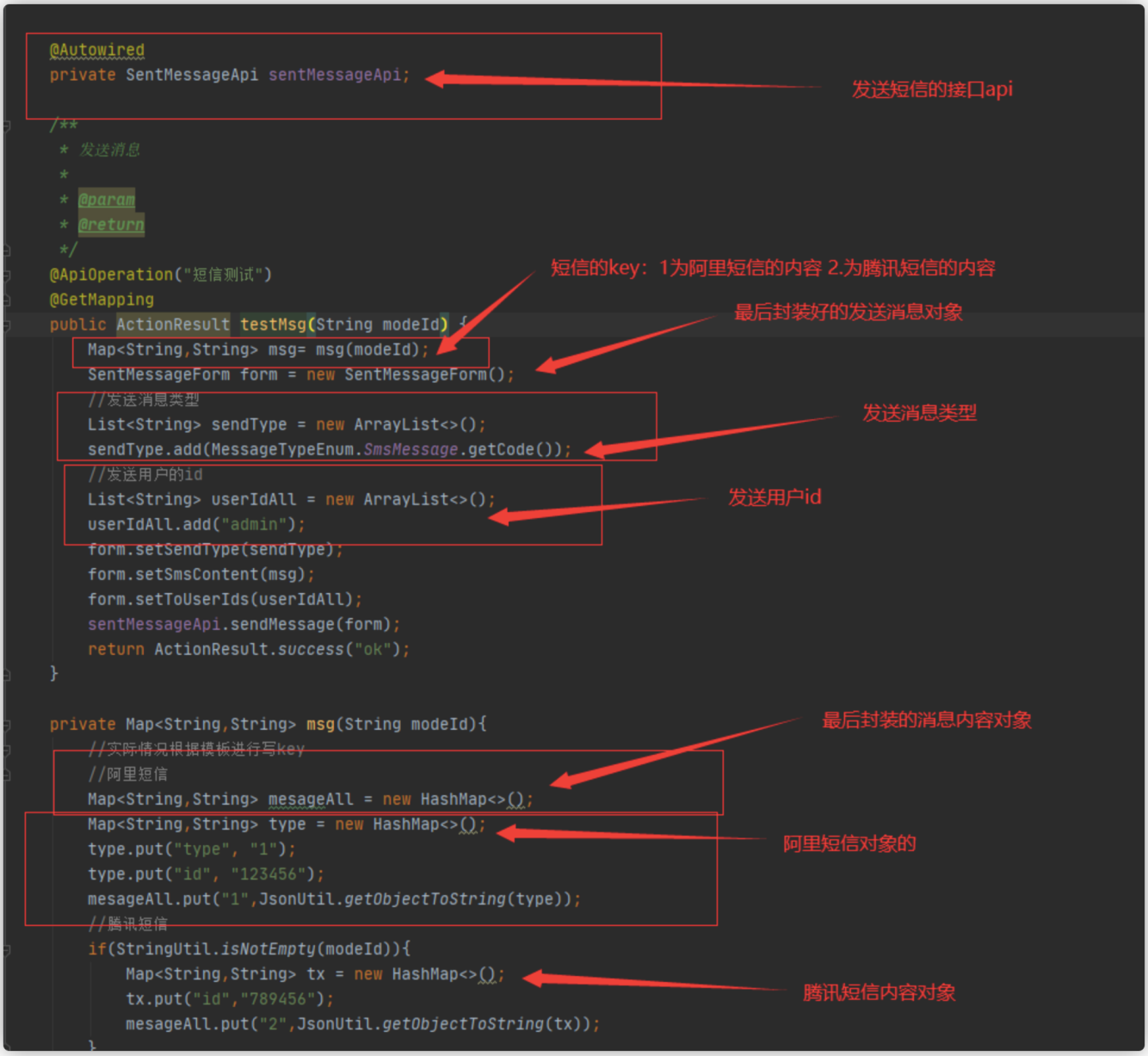
- sendType：发送消息类型,代码有对应的标识
- toUserIds：接收的用户id组
- titile：标题名称
- content: 内容
- smsContent：短信内容(如果key为1:是阿里短信json key为2：是腾讯短信内容)

```
消息模型
@Data
public class SentMessageForm {
    @ApiModelProperty(value = "发送消息类型组合")
    private List<String> sendType;

    @ApiModelProperty(value = "接收人员用户ID组")
    private List<String> toUserIds;

    @ApiModelProperty(value = "标题")
    private String title;

    @ApiModelProperty(value = "内容")
    private String content;

    @ApiModelProperty(value = "短信内容")
    private Map<String,String> smsContent;
}
```

模板的枚举类

```java
/**
 * 消息类型枚举
 */
public enum MessageTypeEnum {
    /**
     * 站内消息
     */
    SysMessage( code: "1",  message: "站内消息"),
    /**
     * 发送邮件
     */
    MailMessage( code: "2",  message: "发送邮件"),
    /**
     * 发送短信
     */
    SmsMessage( code: "3",  message: "发送短信"),
    /**
     * 钉钉消息
     */
    DingMessage( code: "4",  message: "发送钉钉消息"),
    /**
     * 企业微信
     */
    QyMessage( code: "5",  message: "发送企业微信消息");

    private String code;
    private String message;

    MessageTypeEnum(String code, String message) {
        this.code = code;
        this.message = message;
    }

    public String getCode() { return code; }

    public void setCode(String code) { this.code = code; }

    public String getMessage() { return message; }

    public void setMessage(String message) { this.message = message; }

    /**
     * 根据状态code获取枚举值
     * Returns:
```

发送短信的测试类：



消息中心扩展说明

> 本文档以阿里云短信为例

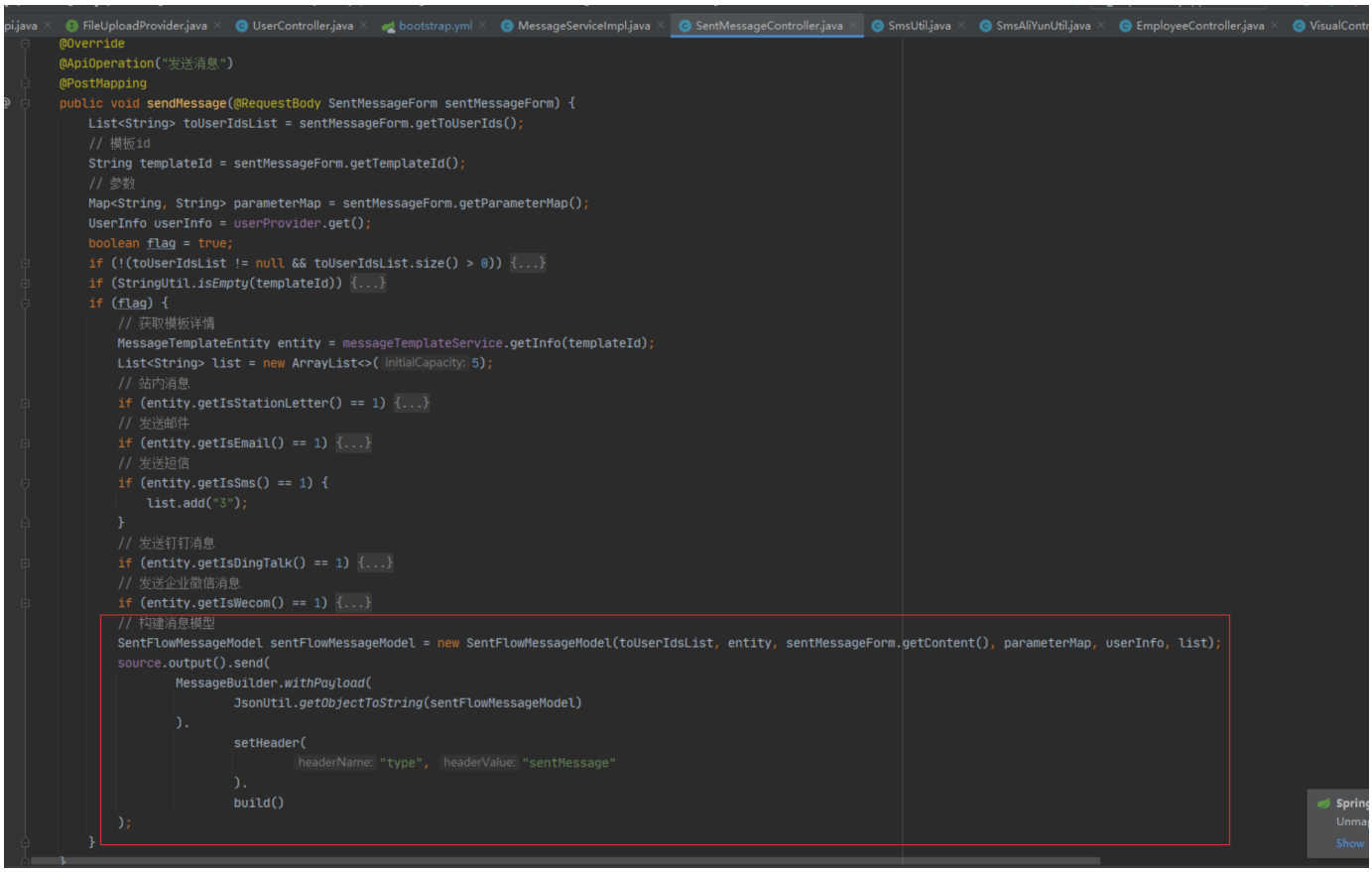
使用rocketMQ去实现具体的发送，调用发送的接口只需要处理数据后将消息发送到消息队列即可，由响应的监听器去实现发送消息。具体代码截图：

1.创建对应的工具类，详情见SmsAliYunUtil.class，代码截图：

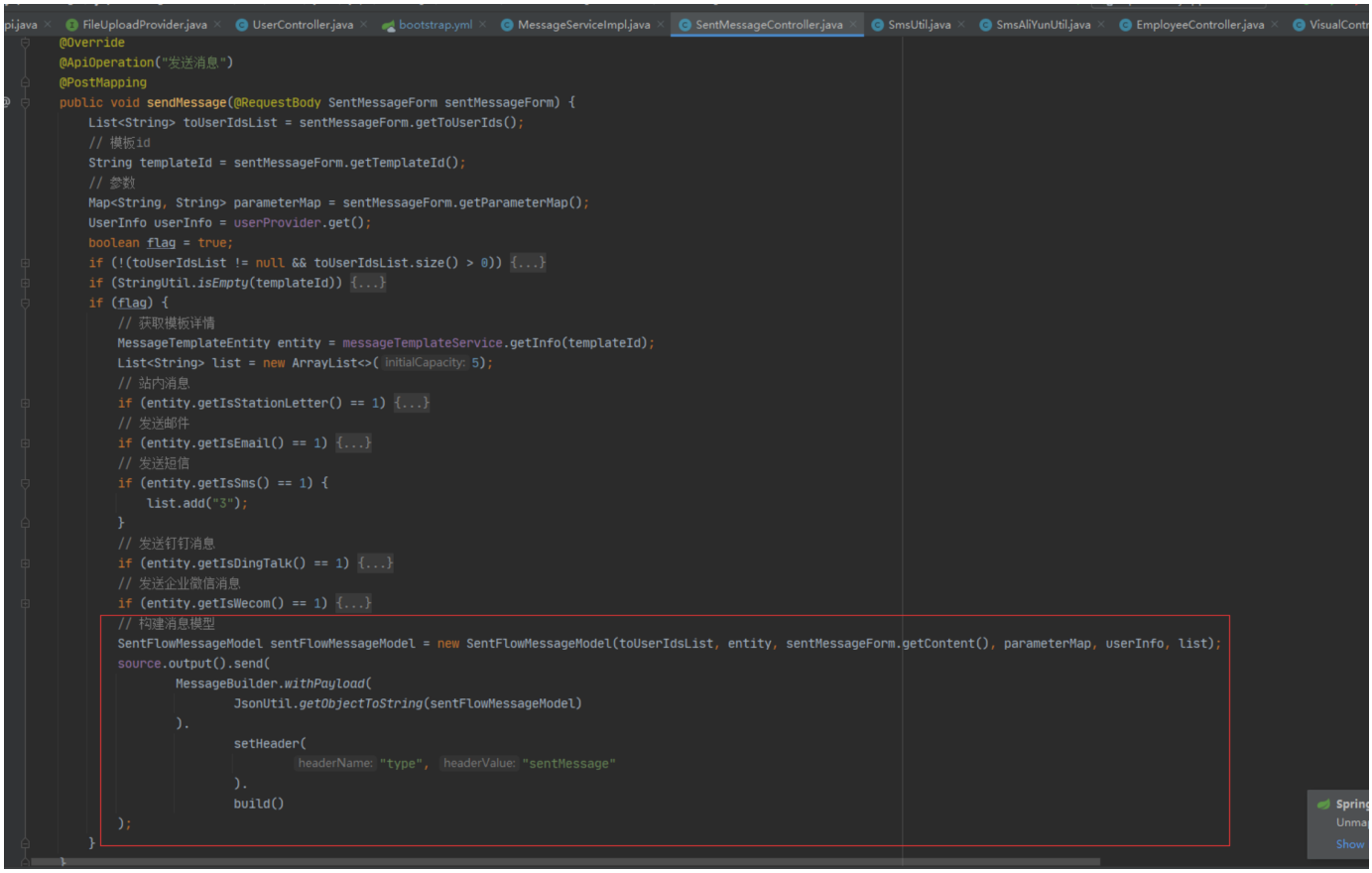
```java
*/
@Slf4j
public class SmsAliYunUtil {

    /**
     * 使用AK&SK初始化账号Client
     *
     * @param accessKeyId
     * @param accessKeySecret
     * @param endpoint
     * @return Client
     */
    private static Client createClient(String accessKeyId, String accessKeySecret, String endpoint) {
        try {
            Config config = new Config()
                    // 您的AccessKey ID
                    .setAccessKeyId(accessKeyId)
                    // 您的AccessKey Secret
                    .setAccessKeySecret(accessKeySecret);
            // 访问的域名
            config.endpoint = endpoint;
            return new Client(config);
        } catch (Exception e) {
            log.error("创建阿里云短信客户端错误：" + e.getMessage());
        }
        return null;
    }
```

```java
    /** 查询短信模板详情 ...*/
    public static List<String> querySmsTemplateRequest(String accessKeyId, String accessKeySecret, String endpoint, String templateId) {
        try {
            Client client = createClient(accessKeyId, accessKeySecret, endpoint);
            QuerySmsTemplateRequest querySmsTemplateRequest = new QuerySmsTemplateRequest()
                    .setTemplateCode(templateId);
            QuerySmsTemplateResponse querySmsTemplateResponse = client.querySmsTemplate(querySmsTemplateRequest);
            String templateContent = querySmsTemplateResponse.getBody().templateContent;
            if (StringUtil.isNotEmpty(templateContent)) {
                List<String> list = new ArrayList<>();
                ParameterUtil.parse( openToken: "${", closeToken: "}", templateContent, list);
                return list;
            } else {
                return null;
            }
        } catch (Exception e) {
            log.error("查询阿里云短信模板错误：" + e.getMessage());
        }
        return null;
    }

    /** 发送短信 ...*/
    public static String sentSms(String accessKeyId, String accessKeySecret, String endpoint, String phoneNumbers, String signContent, String templateId, Map<String, String> map) {
        // 复制代码运行请自行打印 API 的返回值
        try {
            Client client = createClient(accessKeyId, accessKeySecret, endpoint);
            SendSmsRequest sendSmsRequest = new SendSmsRequest();
            // 接收者的号码
            sendSmsRequest.setPhoneNumbers(phoneNumbers);
            // 签名
            sendSmsRequest.setSignName(signContent);
            // 模板id
            sendSmsRequest.setTemplateCode(templateId);
            // 模板参数
            sendSmsRequest.setTemplateParam(JsonUtil.getObjectToString(map));
            SendSmsResponse sendSmsResponse = client.sendSms(sendSmsRequest);
            if (!"OK".equalsIgnoreCase(sendSmsResponse.body.code)) {
                log.error("发送短信失败：" + sendSmsResponse.getBody().message);
            }
```

2.具体使用及接口调用，调用示例SentMessageController.class，发送到mq消息队列中，代码截图：



```java
@Override
@ApiOperation("发送消息")
@PostMapping
public void sendMessage(@RequestBody SentMessageForm sentMessageForm) {
    List<String> toUserIdsList = sentMessageForm.getToUserIds();
    // 模板id
    String templateId = sentMessageForm.getTemplateId();
    // 参数
    Map<String, String> parameterMap = sentMessageForm.getParameterMap();
    UserInfo userInfo = userProvider.get();
    boolean flag = true;
    if (!(toUserIdsList != null && toUserIdsList.size() > 0)) {...}
    if (StringUtil.isEmpty(templateId)) {...}
    if (flag) {
        // 获取模板详情
        MessageTemplateEntity entity = messageTemplateService.getInfo(templateId);
        List<String> list = new ArrayList<>( initialCapacity: 5);
        // 站内消息
        if (entity.getIsStationLetter() == 1) {...}
        // 发送邮件
        if (entity.getIsEmail() == 1) {...}
        // 发送短信
        if (entity.getIsSms() == 1) {
            list.add("3");
        }
        // 发送钉钉消息
        if (entity.getIsDingTalk() == 1) {...}
        // 发送企业微信消息
        if (entity.getIsWecom() == 1) {...}
        // 构建消息模型
        SentFlowMessageModel sentFlowMessageModel = new SentFlowMessageModel(toUserIdsList, entity, sentMessageForm.getContent(), parameterMap, userInfo, list);
        source.output().send(
                MessageBuilder.withPayload(
                        JsonUtil.getObjectToString(sentFlowMessageModel)
                ).
                        setHeader(
                                headerName: "type", headerValue: "sentMessage"
                        ).
                        build()
        );
    }
}
```



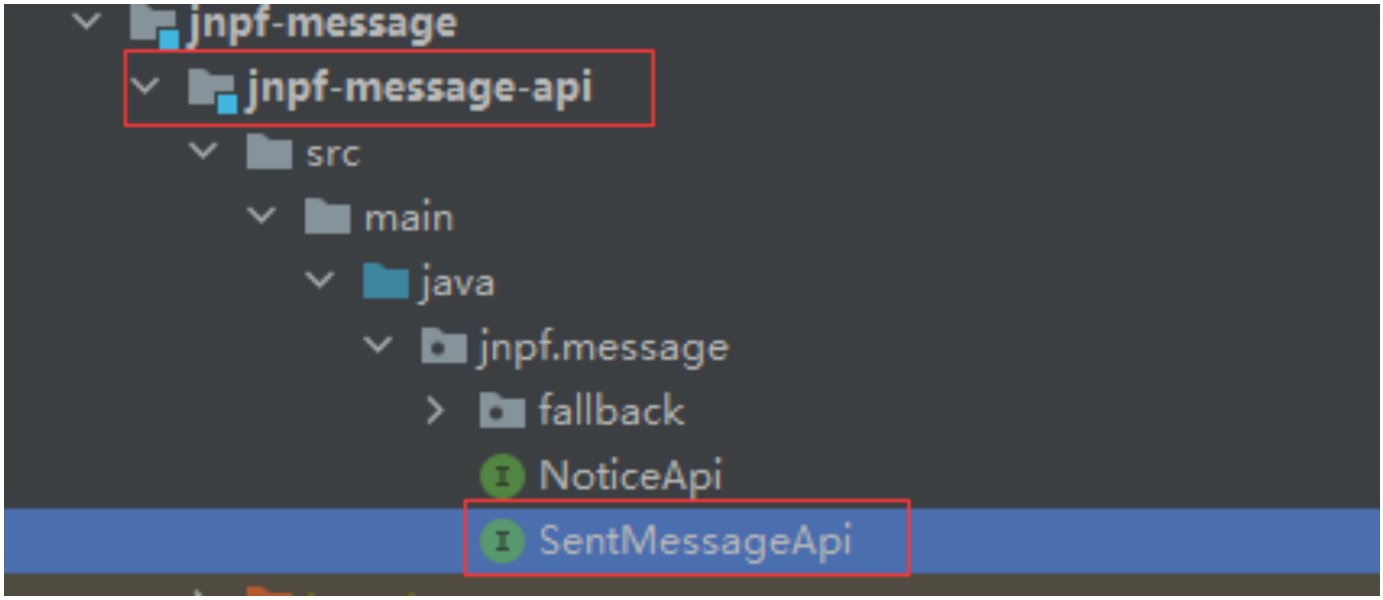
```java
@Override
@ApiOperation("发送消息")
@PostMapping
public void sendMessage(@RequestBody SentMessageForm sentMessageForm) {
    List<String> toUserIdsList = sentMessageForm.getToUserIds();
    // 模板id
    String templateId = sentMessageForm.getTemplateId();
    // 参数
    Map<String, String> parameterMap = sentMessageForm.getParameterMap();
    UserInfo userInfo = userProvider.get();
    boolean flag = true;
    if (!(toUserIdsList != null && toUserIdsList.size() > 0)) {...}
    if (StringUtil.isEmpty(templateId)) {...}
    if (flag) {
        // 获取模板详情
        MessageTemplateEntity entity = messageTemplateService.getInfo(templateId);
        List<String> list = new ArrayList<>( initialCapacity: 5);
        // 站内消息
        if (entity.getIsStationLetter() == 1) {...}
        // 发送邮件
        if (entity.getIsEmail() == 1) {...}
        // 发送短信
        if (entity.getIsSms() == 1) {
            list.add("3");
        }
        // 发送钉钉消息
        if (entity.getIsDingTalk() == 1) {...}
        // 发送企业微信消息
        if (entity.getIsWecom() == 1) {...}
        // 构建消息模型
        SentFlowMessageModel sentFlowMessageModel = new SentFlowMessageModel(toUserIdsList, entity, sentMessageForm.getContent(), parameterMap, userInfo, list);
        source.output().send(
                MessageBuilder.withPayload(
                        JsonUtil.getObjectToString(sentFlowMessageModel)
                ).
                        setHeader(
                                headerName: "type", headerValue: "sentMessage"
                        ).
                        build()
        );
    }
}
```

监听方法sentMessage(String flowMessageModel)

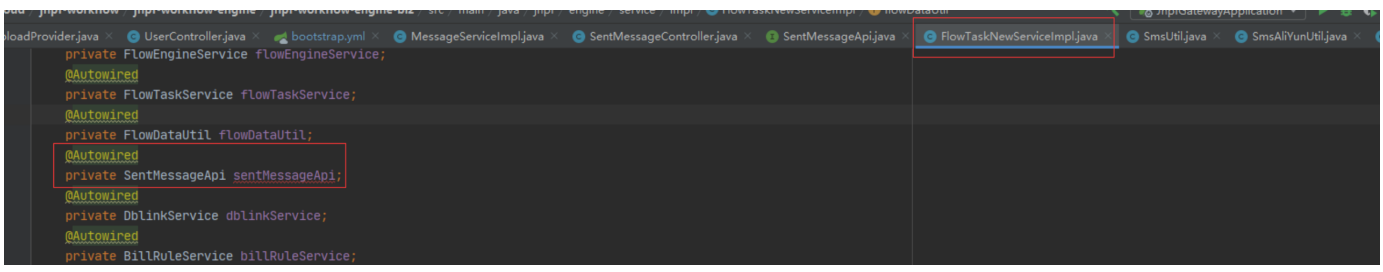
```java
        );
    }
}

/**
 * Mq监听器
 *
 * @param flowMessageModel 发送消息模型
 */
@StreamListener(value = Sink.INPUT, condition = "headers['type']='sentMessage'")
public void sentMessage(String flowMessageModel) {
    try {
        SentFlowMessageModel sentFlowMessageModel = JsonUtil.getJsonToBean(flowMessageModel, SentFlowMessageModel.class);
        // 消费消息
        for (String sendType : sentFlowMessageModel.getSendType()) {
            switch (sendType) {
                case "发送短信":
                    // 发送短信
                    sendSms(sentFlowMessageModel.getToUserIdsList(), sentFlowMessageModel.getEntity(), sentFlowMessageModel.getParameterMap());
                    break;
                default:
                    break;
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

其他服务如需调用发送消息，使用openFeign的方式调用

接口代码路径：



导入jnpf-message-api后，获取bean后调用方法即可，如：



```java
    }
}

/**
 * 发送消息
 *
 * @param flowMsgModel
 */
private void message(FlowMsgModel flowMsgModel) {
    Map<String, Object> data = flowMsgModel.getData();
    List<SentMessageForm> messageList = new ArrayList<>();
    FlowTaskEntity taskEntity = flowMsgModel.getTaskEntity();
    FlowTaskNodeEntity taskNodeEntity = flowMsgModel.getTaskNodeEntity();
    List<FlowTaskNodeEntity> nodeList = flowMsgModel.getNodeList();
    List<FlowTaskOperatorEntity> operatorList = flowMsgModel.getOperatorList();
    List<FlowTaskCirculateEntity> circulateList = flowMsgModel.getCirculateList();
    //等待
    if (flowMsgModel.isWait()) {...}

    //通过
    if (flowMsgModel.isApprove()) {...}

    //驳回
    if (flowMsgModel.isReject()) {...}

    //催办
    if (flowMsgModel.isPress()) {...}

    //子流程
    if (flowMsgModel.isLaunch()) {...}

    for (SentMessageForm messageForm : messageList) {
        sentMessageApi.sendMessage(messageForm);
    }
}

/**
```

工作流-节点事件示例

一、准备接口

```java
import ...

@Slf4j
@RestController
@Api(tags = "pd_order", value = "pd_order")
@RequestMapping("/api/order")
public class PdeController {

    test
    Returns:
    @GetMapping("/test")
    public ActionResult list(String taskId, String taskNodeId) {
        System.out.println("taskId:" + taskId + "  taskNodeId:" + taskNodeId);
        System.out.println("进入网关事件");
        return ActionResult.success();
    }

}
```

二、配置流程设计

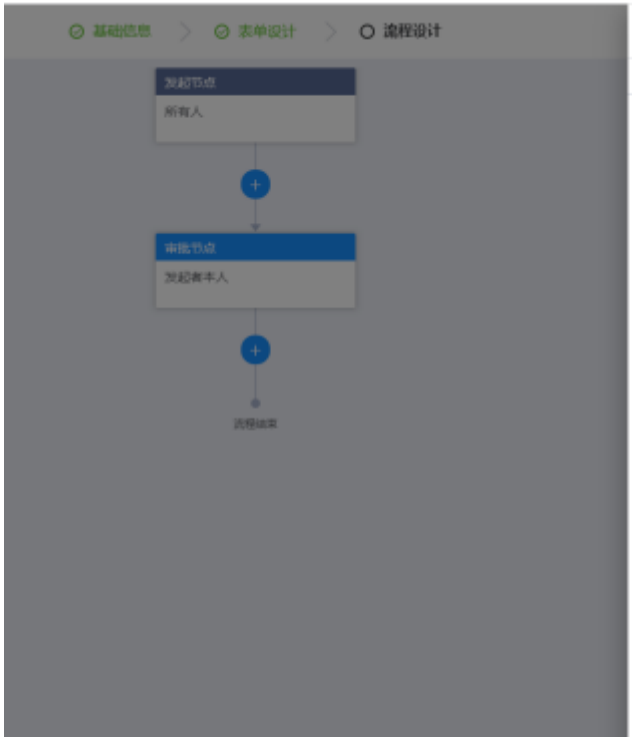
1.先设置流程

2.每个节点都可以设置节点事件

3.开始事件和结束事件在发起节点设置

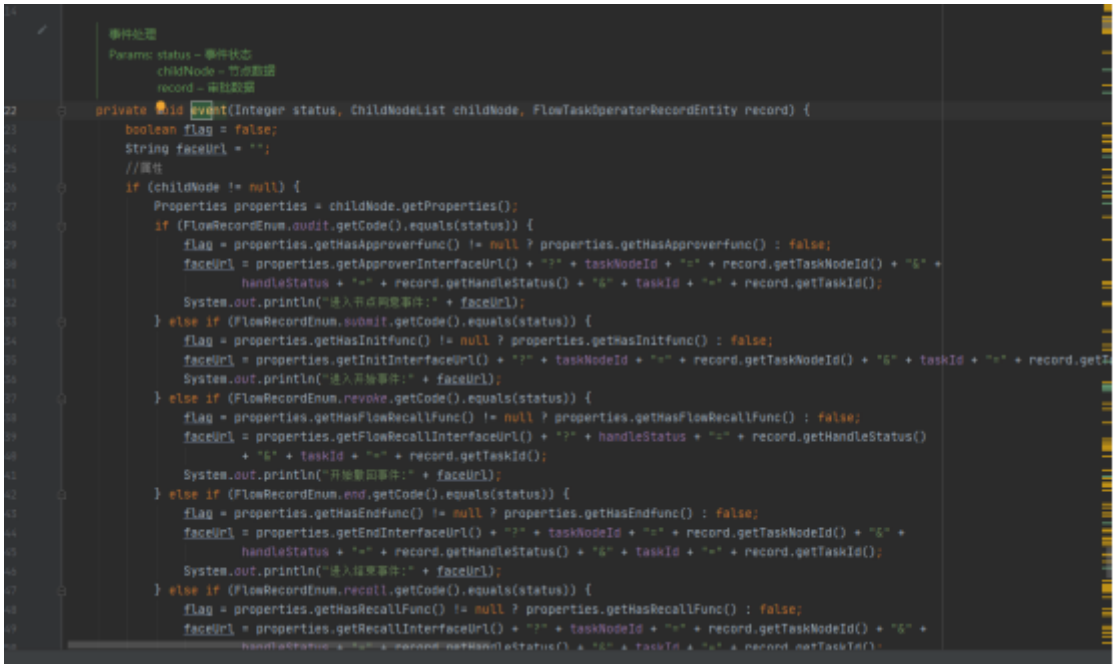
4.写好全路径的请求

5.taskId和taskNodeId后端会拼接上去



三、事件处理的代码

可以自己修改事件的参数



```
事件处理
Params: status – 事件状态
        childNode – 节点数据
        record – 审批数据
private void event(Integer status, ChildNodeList childNode, FlowTaskOperatorRecordEntity record) {
    boolean flag = false;
    String faceUrl = "";
    //属性
    if (childNode != null) {
        Properties properties = childNode.getProperties();
        if (FlowRecordEnum.audit.getCode().equals(status)) {
            flag = properties.getHasApproverfunc() != null ? properties.getHasApproverfunc() : false;
            faceUrl = properties.getApproverInterfaceUrl() + "?" + taskNodeId + "=" + record.getTaskNodeId() + "&" +
                    handleStatus + "=" + record.getHandleStatus() + "&" + taskId + "=" + record.getTaskId();
            System.out.println("进入节点同意事件:" + faceUrl);
        } else if (FlowRecordEnum.submit.getCode().equals(status)) {
            flag = properties.getHasInitfunc() != null ? properties.getHasInitfunc() : false;
            faceUrl = properties.getInitInterfaceUrl() + "?" + taskNodeId + "=" + record.getTaskNodeId() + "&" + taskId + "=" + record.getT
            System.out.println("进入开始事件:" + faceUrl);
        } else if (FlowRecordEnum.revoke.getCode().equals(status)) {
            flag = properties.getHasFlowRecallFunc() != null ? properties.getHasFlowRecallFunc() : false;
            faceUrl = properties.getFlowRecallInterfaceUrl() + "?" + handleStatus + "=" + record.getHandleStatus()
                    + "&" + taskId + "=" + record.getTaskId();
            System.out.println("开始撤回事件:" + faceUrl);
        } else if (FlowRecordEnum.end.getCode().equals(status)) {
            flag = properties.getHasEndfunc() != null ? properties.getHasEndfunc() : false;
            faceUrl = properties.getEndInterfaceUrl() + "?" + taskNodeId + "=" + record.getTaskNodeId() + "&" +
                    handleStatus + "=" + record.getHandleStatus() + "&" + taskId + "=" + record.getTaskId();
            System.out.println("进入结束事件:" + faceUrl);
        } else if (FlowRecordEnum.recall.getCode().equals(status)) {
            flag = properties.getHasRecallFunc() != null ? properties.getHasRecallFunc() : false;
            faceUrl = properties.getRecallInterfaceUrl() + "?" + taskNodeId + "=" + record.getTaskNodeId() + "&" +
```

四、其他事件

参考审批事件

工作流-审批-服务示例

一、准备接口

```
@RestController
@RequestMapping("/api/example/Contract")
```

```
public class ContractController {

    /**
     *
     * @param taskId 任务ID
     * @param taskNodeId 任务节点ID
     * @return
     */
    @GetMapping("/getUsers")
    public ActionResult getUsers(String taskId, String taskNodeId){
        Map<String, String> map = new HashMap<>();
        //base_user的f_id字段， 多个用户用英文逗号隔开
        map.put("handleId", "id1,id2,id3,id4");
        return ActionResult.success(map);
    }
}
```

二、配置流程设计

1.点击基础设置，配置服务的审批设置

2.写好全路径的请求

3.taskId和taskNodeId后端会拼接上去， 服务接口填写可以访问上面Controller的路径

`http://127.0.0.1:30000/api/exampl/Contract/getUsers`

三、获取服务数据

1.服务接收的数据格式：{"code":"200","data":{"handleId":"admin"}}

2.多人可以在handleId的参数用逗号形式传过来

3.handleId的值只认base_user的f_id值

```
if (FlowTaskOperatorEnum.Serve.getCode().equals(type)) {
            String url = properties.getGetUserUrl() + "?" + taskNodeId + "=" + child
Node.getTaskNodeId() + "&" + taskId + "=" + childNode.getTaskId();
            String token = UserProvider.getToken();
            JSONObject object = HttpUtil.httpRequest(url, "GET", null, token);
            if (object != null) {
                if (object.get("data") != null) {
                    JSONObject data = object.getJSONObject("data");
                    List<String> handleId = StringUtil.isNotEmpty(data.getString("ha
ndleId")) ? Arrays.asList(data.getString("handleId").split(",")) : new ArrayList<>
();
                    userIdAll.addAll(handleId);
                }
            }
        }
```

SEATA事务回滚

> 1、A服务Seata全局事务中通过Feign调用B服务， B服务接口保存数据之后报错AB服务都不会回滚数据
> 全局回滚需要A服务报错才可以回滚其他服务的数据

A:
ResultException中排除Seata调用时报错的统一封装返回结果
Feign接口 去除@FeignClient中的fallback或者自定义处理之后抛出异常

B:
调用Feign之后判断返回结果执行报错操作

动态数据源使用手册

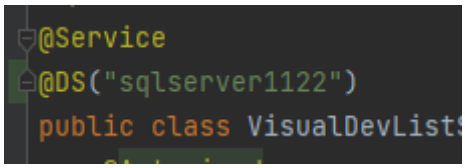
方法1： 增加数据源配置文件

1、spring.datasource.dynamic.datasource 节点下增加数据源

```
spring:
  datasource:
    # 数据库类型(可选值 MySQL、SQLServer、Oracle、DM8、KingbaseES、PostgreSQL，请严格按可选值填写)
    dbtype: MySQL
    dbname: java_boot_init
    host: 192.168.0.10
    port: 3306
    username: root
    password: root
    # 表空间(当数据库为Oracle、达梦DM8、金仓KingbaseES时表空间必须指定，其他数据库为空即可)
    tablespace:
    # 多数据源配置
    dynamic:
      enabled: true
      seata: false
      # 设置默认的数据源或者数据源组,默认值即为master
      primary: master
      # 严格匹配数据源,默认false. true未匹配到指定数据源时抛异常()可用于确认切库失败,false使用默认数据源
      strict: true
      datasource: #若只有一个数据源下方可为空， 本节点保留
```

```
        # 额外的数据源 Service类中使用DS动态切换
        master_2: #master_1, master_2 数据源前缀相同可组成数据源池负载均衡
          url: jdbc:mysql://192.168.0.10:3306/JNPF_PROTECT?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8
          username: JNPF_PROTECT
          password: zG4rxnNrkFeCsri8
          driver-class-name: com.mysql.cj.jdbc.Driver
        sqlserver1122: #master_1, master_2 数据源前缀相同可组成数据源池负载均衡
          url: jdbc:mysql://192.168.0.10:3306/JNPF_PROTECT?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8
          username: JNPF_PROTECT
          password: zG4rxnNrkFeCsri8
          driver-class-name: com.mysql.cj.jdbc.Driver
```

2、Service类添加DS注解切换数据源， DS内的数据源名称为配置文件中增加的名字， 本服务内的数据库操作都自动切换指定数据源, @DS("sqlserver1122")



方法2：代码动态生成数据源

1、继承接口DynamicSourceGeneratorInterface 重写getDataSource方法生成数据源配置

(1)直接返回配置信息

```
        @Override
        public DataSourceUtil getDataSource() {
                DataSourceUtil data = new DataSourceUtil();
                data.setHost("127.0.0.1");
                data.setPort(3306);
                data.setPassword("123456");data.setUserName ("root");
                data.setDbType(DbOracle.ORACLE);
                data.setDbName("inpf-cloud");
                return data;
        }
```

(2)从 `数据连接` 管理功能获取数据配置， 本例使用数据管理内的连接名称获取连接配置


```
        @Autowired
        private DblinkService dblinkService;

        @Override
        public DataSourceUtil getDataSource(){
                QueryWrapper<DbLinkEntity> queryWrapper = new QueryWrapper<>();
                queryWrapper.lambda().eq(DbLinkEntity::getFullName, this.getClass().
getAnnotation(DS.class).value());
                //queryWrapper.lambda().eq(DbLinkEntity::getFullName, "数据源连接名
称");
                return dblinkService.getOne(queryWrapper);
        }
```

(3)可重写方法cachedConnection决定是否每次获取新配置

注意事项

1：使用 `@DSTranslation` 替代 `@Translation` 开启事务， `@Translation` 只支持单数据源会锁定数据连接导致无法切库

（也可使用Seata的 `@GlobalTranslation` ， 将配置中 `spring.datasource.dynamic.seata` 设为 `true` ，注意Seata限制过多， 且只支持Mysql,Oracle,PostgreSql三个数据库， 当前业务无分布式事务需求不建议使用, 多数据源需要设置 `seata.enable-auto-data-source-proxy: false` ）

2：Service类中同一个方法内同时调用主数据库和第三方数据库操作使用

```
DynamicDataSourceContextHolder.poll();//清除当前动态库， 后面的数据库操作将在主库进行
DynamicDataSourceContextHolder.push(“其他数据源名称”);//动态切换当前数据库， 后面代码的数据库操作将在第三方数据库进行
```

多网卡设置项目注册IP

Nacos注册

```
spring:
  cloud:
    inetutils:
      preferred-networks: 192.168.0 #填写网段或IP
```

Dubbo注册

```
dubbo:
  protocol: #根据配置方式选择填写这里或者下面的配置
    host: ${spring.cloud.client.ip-address} #或者直接填写IP

  protocols:
    dubbo:
      host: ${spring.cloud.client.ip-address} #或者直接填写IP
```

XSS过滤

工具类：jnpf.util.XSSEscape

相关文件目录

单体版：jnpf-admin\src\main\resources
微服务：jnpf-common\jnpf-common-core\src\main\resources

相关文件

AntiSamy_zh_CN.properties 提示语言包
antisamy-ebay.xml Ebay正在使用的规则文件
antisamy-empty.xml 空验证规则文件

使用方法

过滤用户输入内容：XSSEscape.escape(String)
过滤文件路径：XSSEscape.escapePath(Path)

Nginx配置

> 以下配置均使用项目默认端口，如有修改按实际情况调整

```
    # JNPF-START
    #设置上传文件的大小
    client_max_body_size 100m;

    #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    client_header_buffer_size 128k;

    #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
    large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';

    # 预检命令的缓存，如果不缓存每次会发送两次请求
    add_header Access-Control-Max-Age 3600;

    #带cookie请求需要加上这个字段，并设置为true
    add_header Access-Control-Allow-Credentials true;

    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #    表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;
```

```
# 前端主项目(`jnpf-web`)伪静态
location / {
  try_files $uri $uri/ /index.html;
}

# 前端大屏(`jnpf-web-datascreen`)伪静态
location /DataV {
  try_files $uri $uri/ /DataV/index.html;
}

# 接口
location /api/ {
  proxy_pass http://localhost:30000;
}

# websocket
location /websocket {
  proxy_pass http://localhost:30000/api/message/websocket;
  proxy_http_version 1.1;
  proxy_set_header Upgrade $http_upgrade;
  proxy_set_header Connection "upgrade";
  proxy_read_timeout 600s;
}

# 数据报表接口配置
location /ReportServer/ {
  proxy_pass http://localhost:30000/;
}

# 文件预览服务
location /FileServer {
  proxy_pass http://localhost:30090;
}

# 解决文件预览服务无法加载js,css问题
location ~ /FileServer/*.*\.(js|css)?$ {
   proxy_pass http://localhost:30090;
}
# JNPF-END
```

> 特别说明：如果采用宝塔工具部署，需要把以下配置注释

```
    # location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
    # {
    #     expires      30d;
    #     error_log /dev/null;
    #     access_log /dev/null;
    # }
```

java-boot迁移至java-cloud操作说明

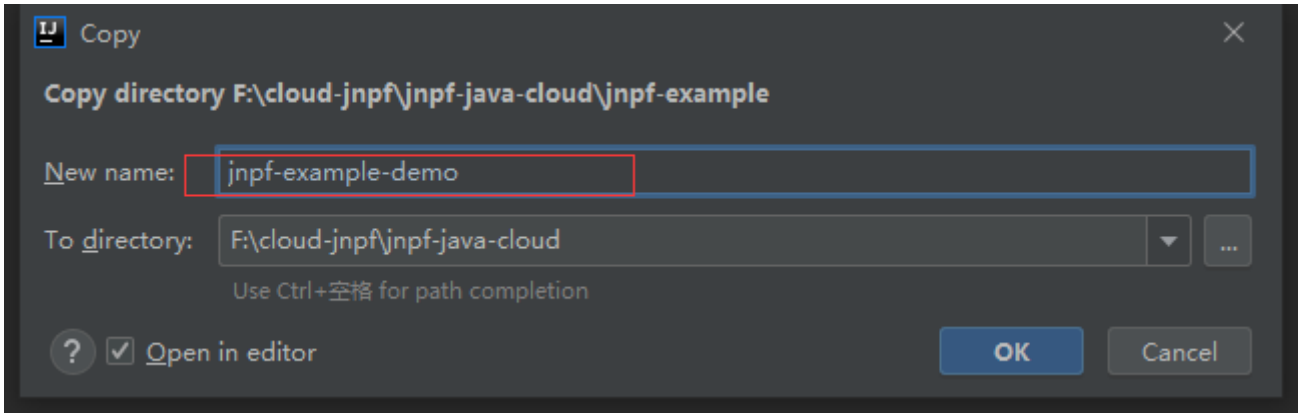
> JNPF官方的java-boot版本和java-cloud版本除了代码结构上的差异，功能是一模一样的

新建服务

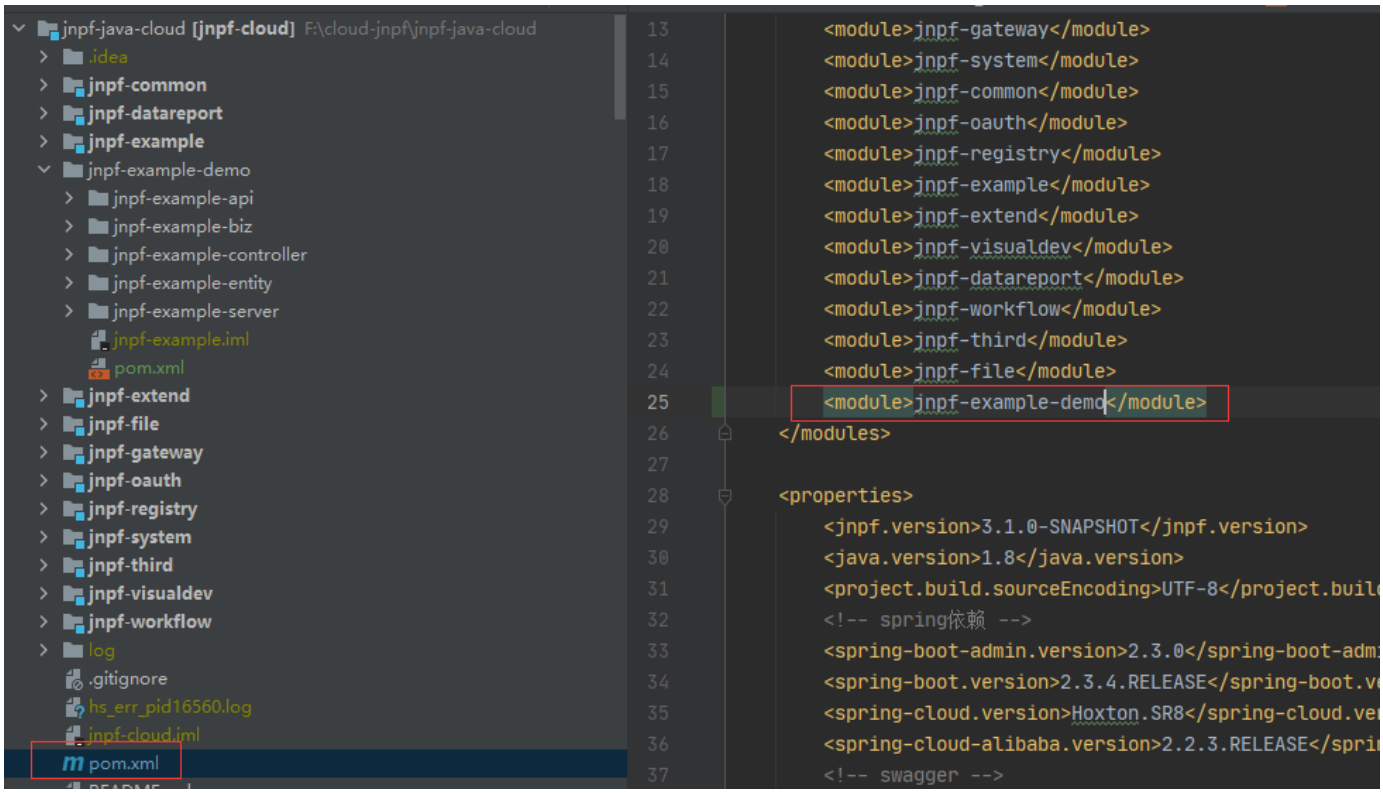
java-cloud版本中没有对应的业务服务

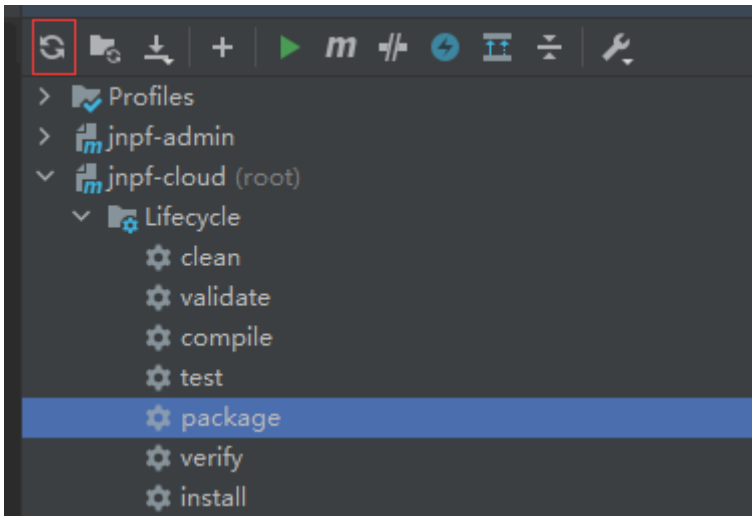
使用 `jnpf-example` 快速创建子系统/子服务

复制Example项目粘贴到jnpf-cloud下，修改项目名点击OK

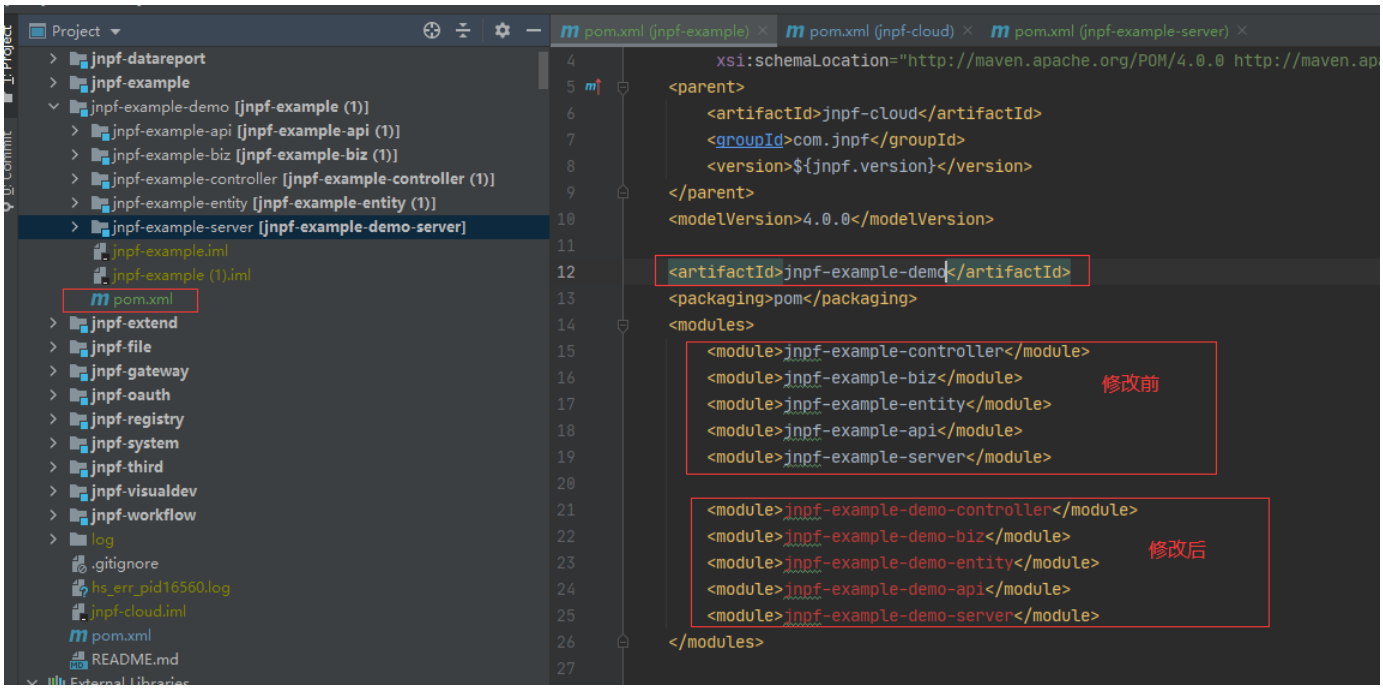


粘贴之后在最外层的Maven中添加如下代码，点击刷新Maven

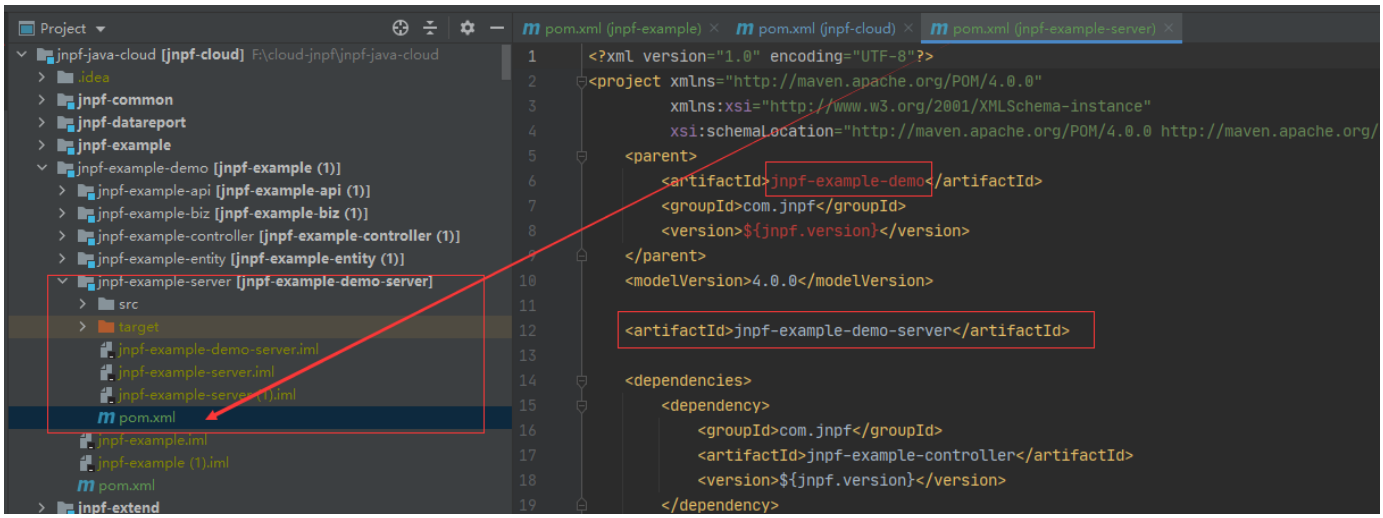




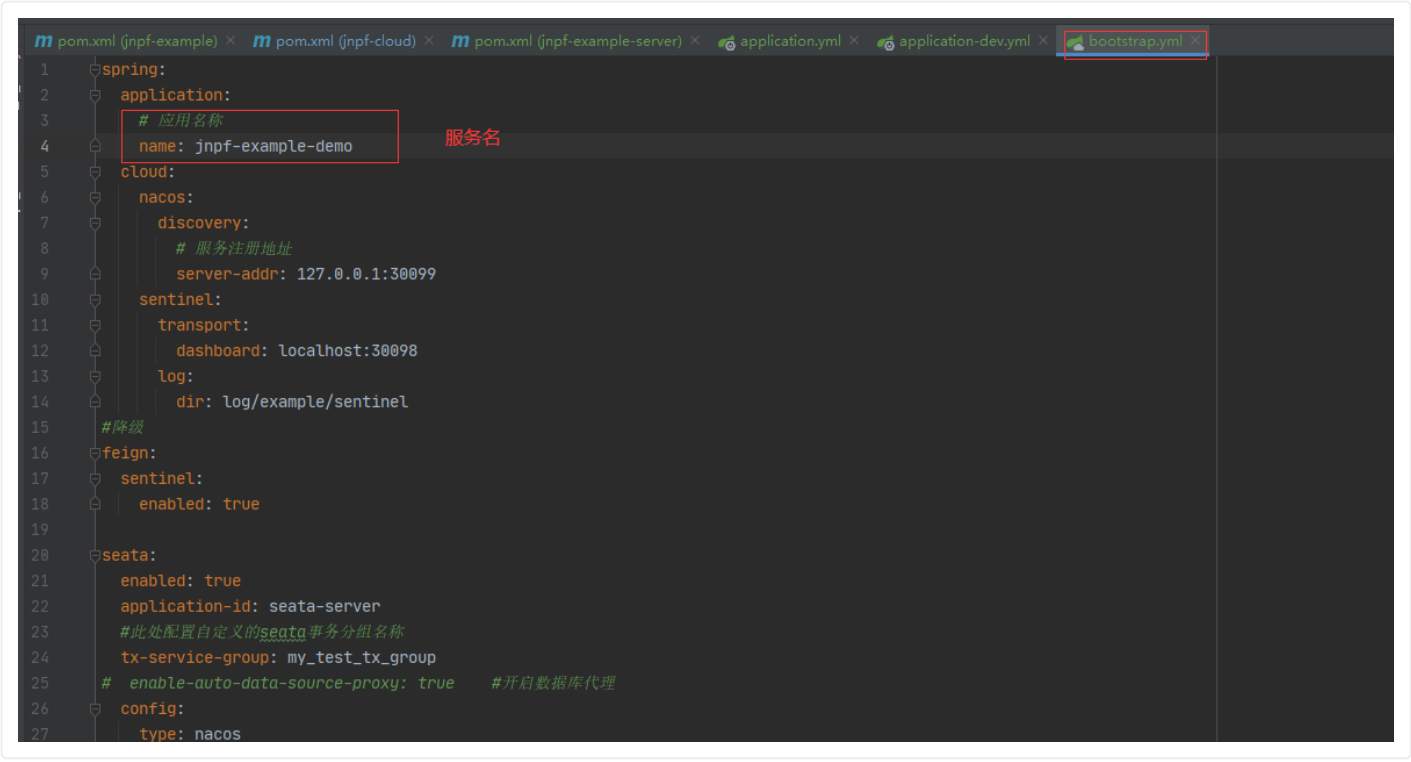
修改刚刚创建好的项目的pom.xml，例：



对应的子模块pom.xml，例如server：

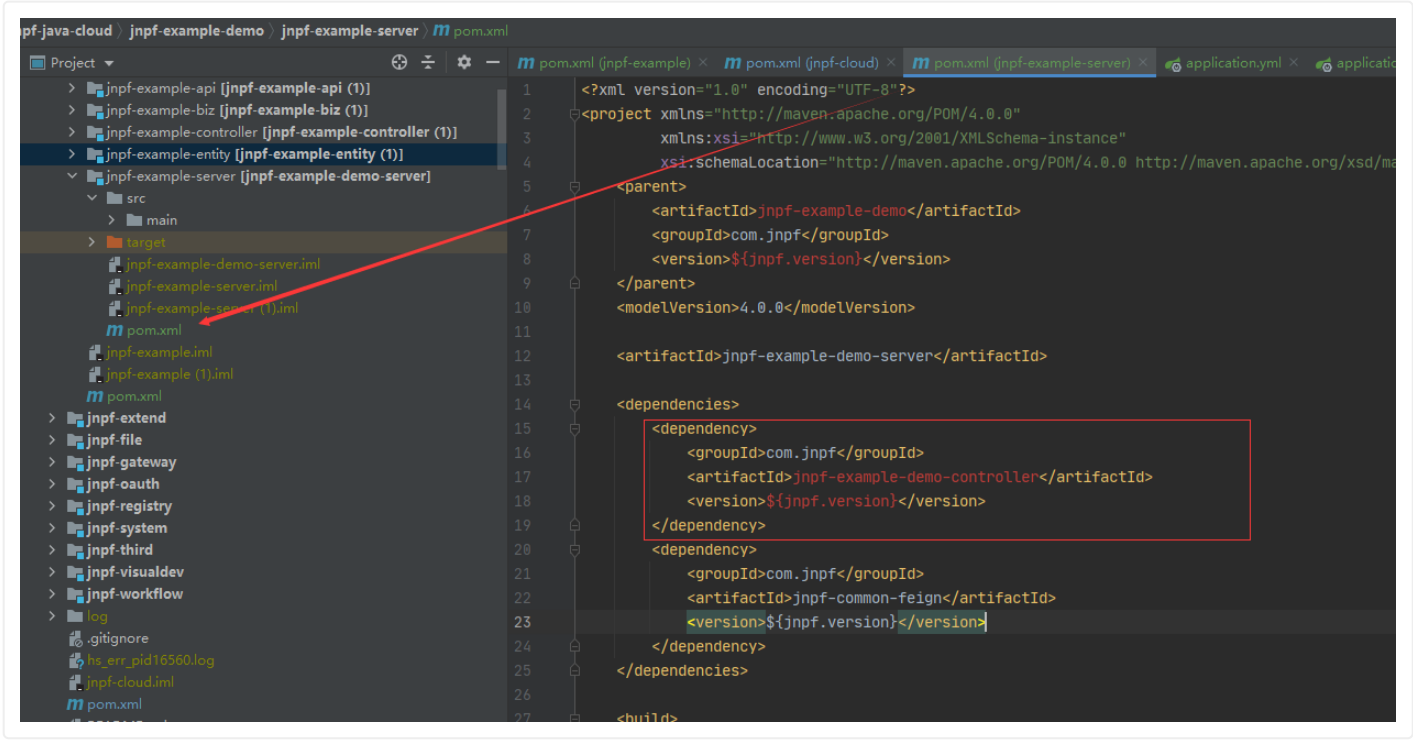


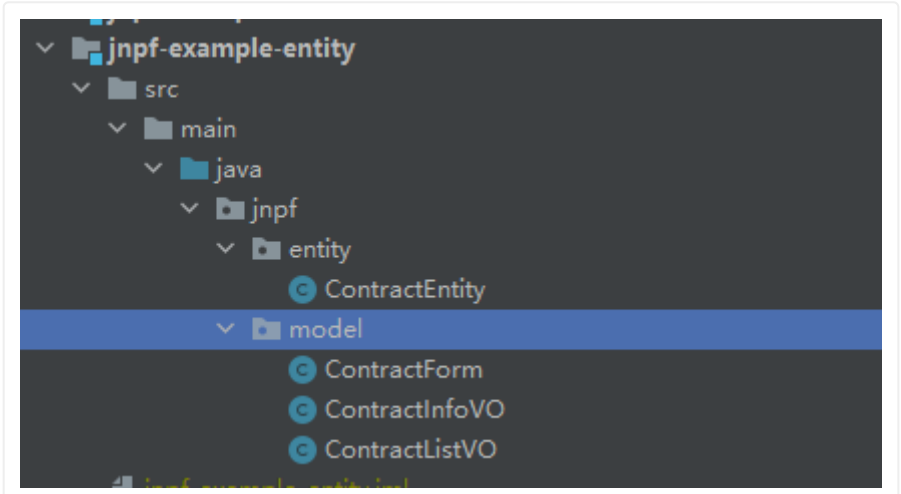
服务名不能重复，需要自己修改：




```yaml
spring:
  application:
    # 应用名称
    name: jnpf-example-demo
  cloud:
    nacos:
      discovery:
        # 服务注册地址
        server-addr: 127.0.0.1:30099
    sentinel:
      transport:
        dashboard: localhost:30098
      log:
        dir: log/example/sentinel
#降级
feign:
  sentinel:
    enabled: true

seata:
  enabled: true
  application-id: seata-server
  #此处配置自定义的seata事务分组名称
  tx-service-group: my_test_tx_group
#  enable-auto-data-source-proxy: true    #开启数据库代理
  config:
    type: nacos
```

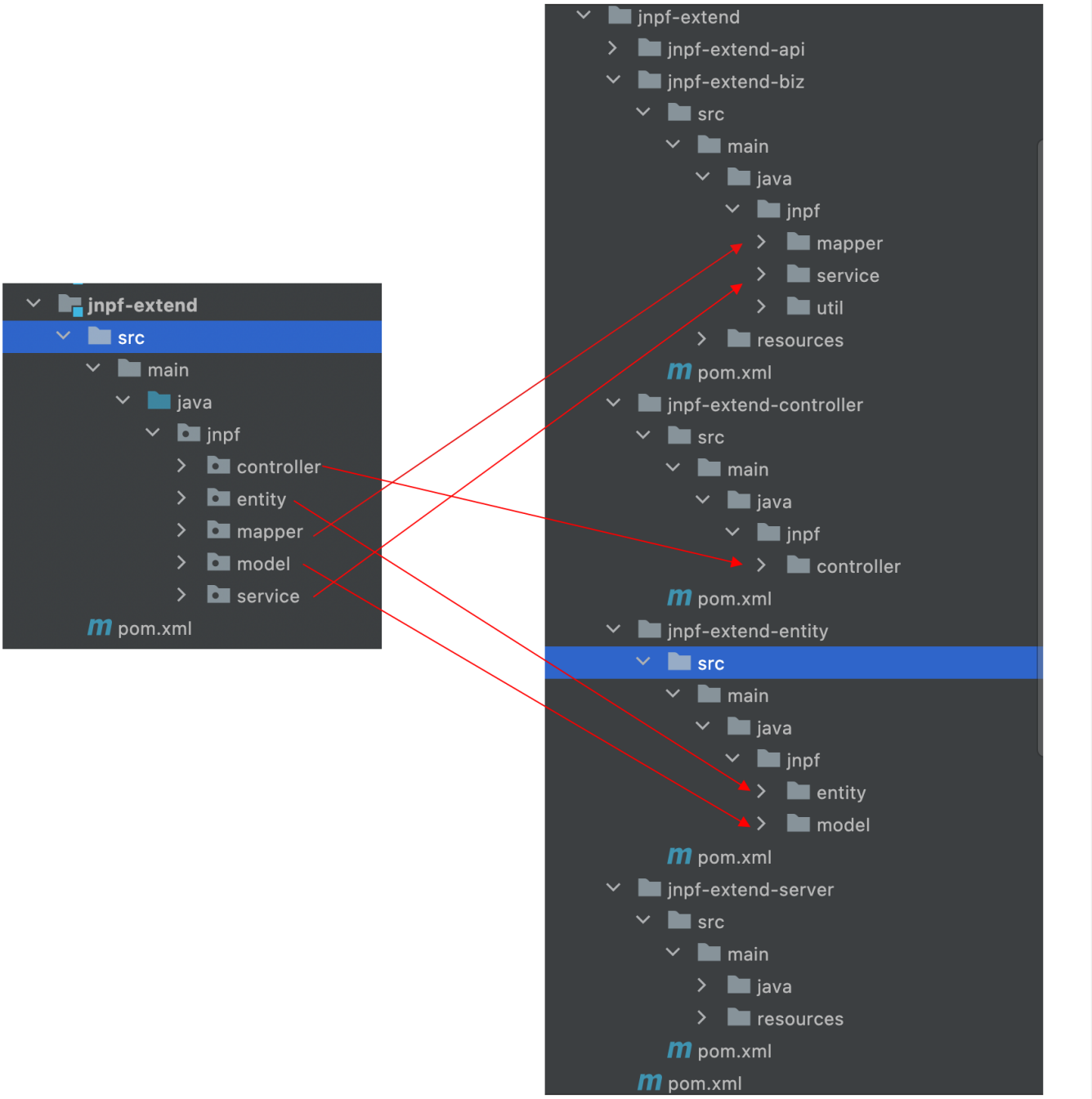
然后将包下的类删除，类似依赖也要改成现在的新名称，然后创建自己的类即可：





java-cloud版本中有对应的业务服务

这里以 `jnpf-extend` 服务为例子，把java-boot中按以下目录对应迁移过去即可。



Nacos注册与配置获取 开启账号认证

Nacos开启获取配置认证

开启后获取配置与注册服务均需要用户认证

配置路径 `nacos\conf\application.properties`

```
nacos.core.auth.enabled=true
```

Seata服务端配置账号密码

配置路径 `seata-server\conf\registry.conf`

7行和63行 修改用户名密码

```
nacos { # 不完整配置
    username = "nacos"
    password = "123456"
  }
```

Sentinel服务端配置账号密码

配置路径 `sentinel-server\src\main\resources\application.yml`

nacos节点下添加用户名密码配置

```
spring:
  cloud:
    nacos:
      username: nacos
      password: 123456
```

SpringBoot Admin配置账号密码

配置路径 `srping-boot-admin\src\main\resources\application-dev.yml`

nacos节点下添加用户名密码配置

```
spring:
  cloud:
    nacos:
      username: nacos
      password: 123456
```

Nacos客户端配置账号密码

配置路径 项目下的 `bootstrap.yml` 文件 nacos节点下添加用户名密码配置

```
spring:
  cloud:
    nacos:
      username: nacos
      password: 123456
```

Seata客户端配置账号密码

配置路径 项目下的 `bootstrap.yml` 文件 修改seata下的两个nacos节点用户名密码

```
seata: # 不完整配置
  config:
    type: nacos
    nacos:
      userName: ${spring.cloud.nacos.username}
      password: ${spring.cloud.nacos.password}
  registry:
    type: ${seata.config.type}
    nacos:
      userName: ${spring.cloud.nacos.username}
      password: ${spring.cloud.nacos.password}
```

Dubbo配置账号密码

配置路径 项目下的 `bootstrap.yml` 文件 在address后方添加用户名密码配置

```
dubbo: # 不完整配置
  registry:
    # 挂载到nacos注册中心
    address: nacos://${spring.cloud.nacos.discovery.server-addr}/serviceManagement?namespace=${spring.cloud.nacos.discovery.namespace}&username=${spring.cloud.nacos.username}&password=${spring.cloud.nacos.password}
```

SaToken接口权限控制（3.4.3起）

使用

> 3.4.3版本起移除SpringSecurity 改用SaToken认证

1. 引用模块

> 在需要做权限控制的Controller模块中引用

```
        <!-- 单体、微服务相同 -->
        <dependency>
            <groupId>com.jnpf</groupId>
            <artifactId>jnpf-common-security</artifactId>
            <version>${jnpf.version}</version>
        </dependency>
```

2. 注解鉴权

1、添加权限控制

> 在接口上添加相关注解

```
@SaCheckPermission("onlineDev.webDesign::btn_add'")
@GetMapping("/hello")
public ActionResult<String> hello() {
    return ActionResult.success("hello");
}
```

```
@SaCheckLogin: 登录校验 —— 只有登录之后才能进入该方法。
@SaCheckRole("角色编码"): 角色校验 —— 必须具有指定角色标识才能进入该方法。
@SaCheckPermission("权限标识符"): 权限校验 —— 必须具有指定权限才能进入该方法。
@SaIgnore: 忽略校验 —— 表示被修饰的方法或类无需进行注解鉴权和路由拦截器鉴权。

# 包含指定权限
@SaCheckPermission('权限标识符')
# 包含任意权限
@SaCheckPermission(value = {"权限标识符1", "权限标识符2"}, mode = SaMode.OR)
# 包含指定角色
@SaCheckRole("角色编码")
# 包含任意角色
@SaCheckRole(value = {"角色编码1", "角色编码2"}, mode = SaMode.OR)
```

2、打开鉴权开关配置

> 默认关闭状态， 开启后会在登录后添加相关权限数据到缓存中， 如不需要此功能保持关闭状态可提高性能

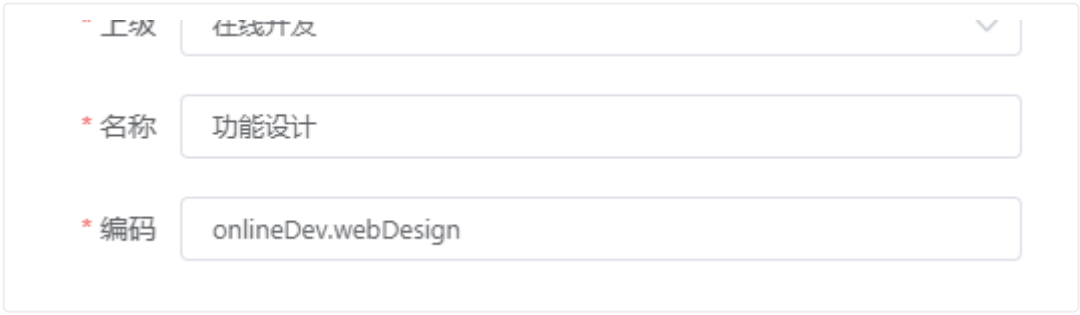
单体版在`application.yml`中

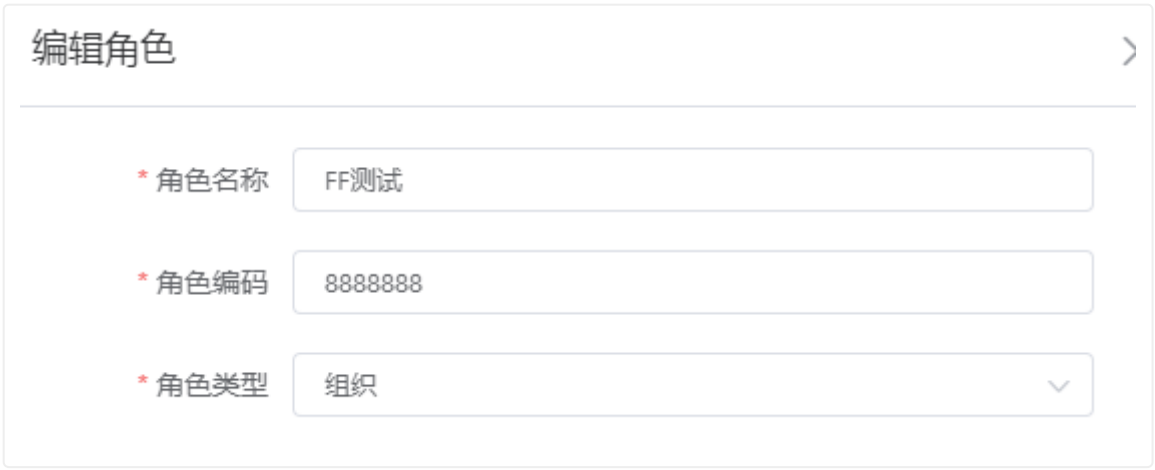
微服务版在Nacos上`system-config.yaml`配置中

```
config:
  EnablePreAuth: true
```

3、权限标识符说明

```
常用于查询
@SaCheckPermission('菜单编码')
常用于功能按钮
@SaCheckPermission('菜单编码::按钮编码')
@SaCheckRole('角色编码')
```





3. 代码验证权限

权限校验

```
// 获取：当前账号所拥有的权限集合
StpUtil.getPermissionList();

// 判断：当前账号是否含有指定权限，返回 true 或 false
StpUtil.hasPermission("权限标识符");

// 校验：当前账号是否含有指定权限，如果验证未通过，则抛出异常：NotPermissionException
StpUtil.checkPermission("权限标识符");

// 校验：当前账号是否含有指定权限 [指定多个，必须全部验证通过]
StpUtil.checkPermissionAnd("权限标识符1", "权限标识符2", "权限标识符3");

// 校验：当前账号是否含有指定权限 [指定多个，只要其一验证通过即可]
StpUtil.checkPermissionOr("权限标识符1", "权限标识符2", "权限标识符3");
```

角色校验

```
// 获取：当前账号所拥有的角色集合
StpUtil.getRoleList();

// 判断：当前账号是否拥有指定角色，返回 true 或 false
StpUtil.hasRole("角色编码");

// 校验：当前账号是否含有指定角色标识，如果验证未通过，则抛出异常：NotRoleException
StpUtil.checkRole("角色编码");

// 校验：当前账号是否含有指定角色标识 [指定多个，必须全部验证通过]
StpUtil.checkRoleAnd("角色编码1", "角色编码2");
```

```
// 校验：当前账号是否含有指定角色标识 [指定多个，只要其一验证通过即可]
StpUtil.checkRoleOr("角色编码1", "角色编码2");
```

相关代码

PermissionInterfaceImpl，动态提供验证权限和角色列表
LoginServiceImpl ，**initSecurityAuthorities**方法在登陆时初始化缓存用户权限角色列表
jnpf.config.SecurityConfiguration 是否开启注解鉴权配置

RocketMQ配置说明

一、增加 spring-cloud-stream-mq 依赖

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-rocketmq</artifactId>
</dependency>
```

二、在 bootstrap.yml 中增加 springcloudstream 配置

> 注意以下为yml格式

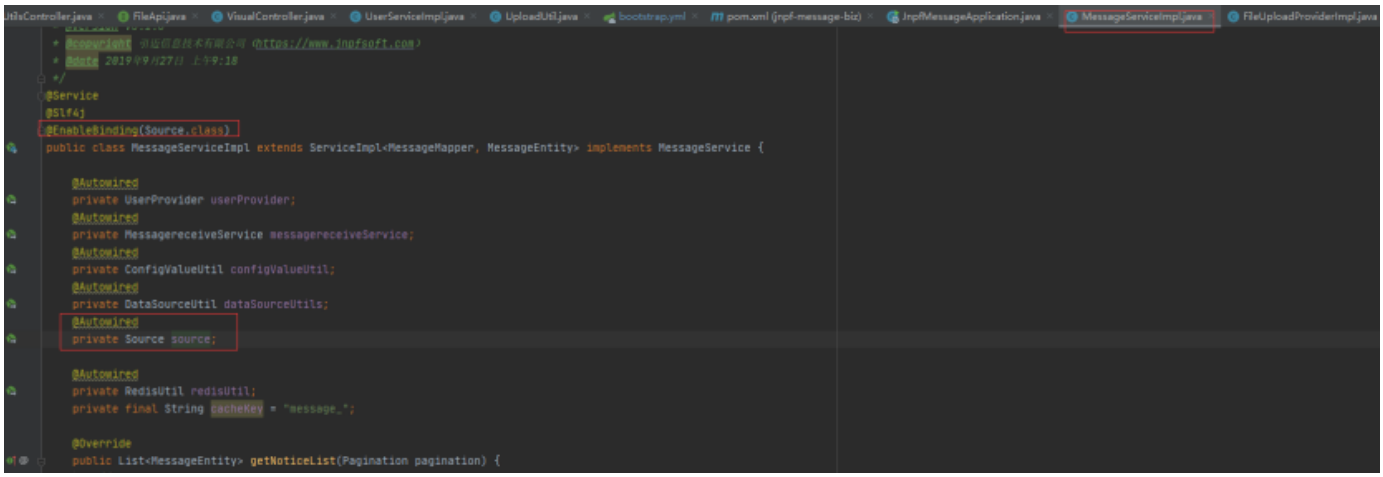
```
spring:
  cloud:
    # stream配置
    stream:
# rocketMq配置
      rocketmq:
# rocket地址
        binder:
          name-server: 192.168.0.34:30095
# 设置使用同步的消费方式
        bindings:
          output:
            producer:
              sync: true
              group: jnpf-group
      bindings:
# 输出使用自带的output（发送消息）
```

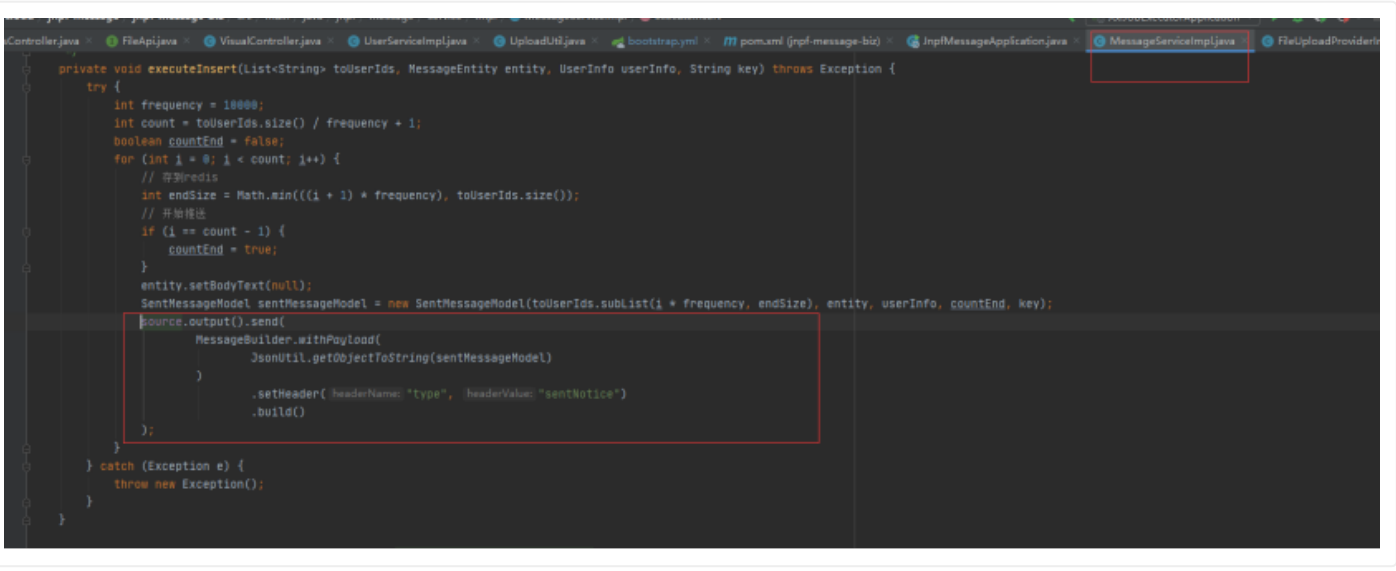
```
        output:
          content-type: text/json
          destination: jnpf-topic
          group: jnpf-group
# 接收消息
        input:
          content-type: text/json
          destination: jnpf-topic
          group: jnpf-group
```

三、绑定消费者

```
@EnableBinding(Source.class)
使用消费者
@Autowired
private Source source;
# 此处增加了消息头
source.output().send(
        MessageBuilder.withPayload(
                JsonUtil.getObjectToString(sentMessageModel)
        )
                .setHeader("type", "sentNotice")
                .build()
);
```

相关代码截图：



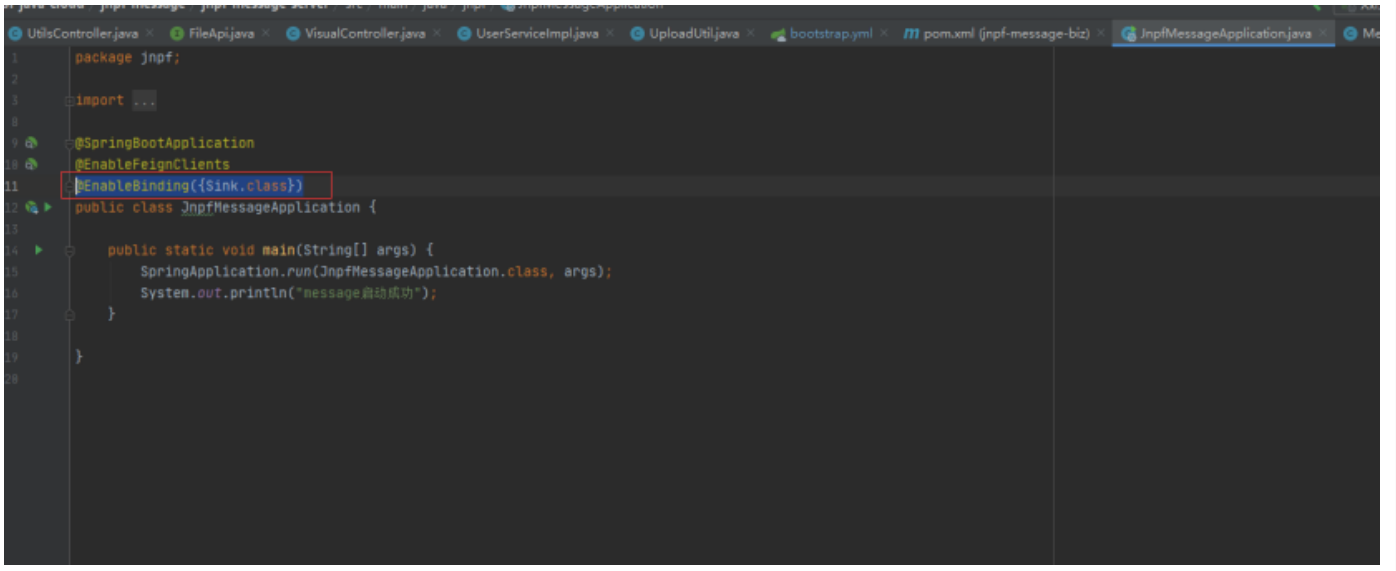


```
private void executeInsert(List<String> toUserIds, MessageEntity entity, UserInfo userInfo, String key) throws Exception {
    try {
        int frequency = 10000;
        int count = toUserIds.size() / frequency + 1;
        boolean countEnd = false;
        for (int i = 0; i < count; i++) {
            // 存到redis
            int endSize = Math.min(((i + 1) * frequency), toUserIds.size());
            // 开始推送
            if (i == count - 1) {
                countEnd = true;
            }
            entity.setBodyText(null);
            SentMessageModel sentMessageModel = new SentMessageModel(toUserIds.subList(i * frequency, endSize), entity, userInfo, countEnd, key);
            source.output().send(
                    MessageBuilder.withPayload(
                            JsonUtil.getObjectToString(sentMessageModel)
                    )
                            .setHeader( headerName: "type",  headerValue: "sentNotice")
                            .build()
            );
        }
    } catch (Exception e) {
        throw new Exception();
    }
}
```

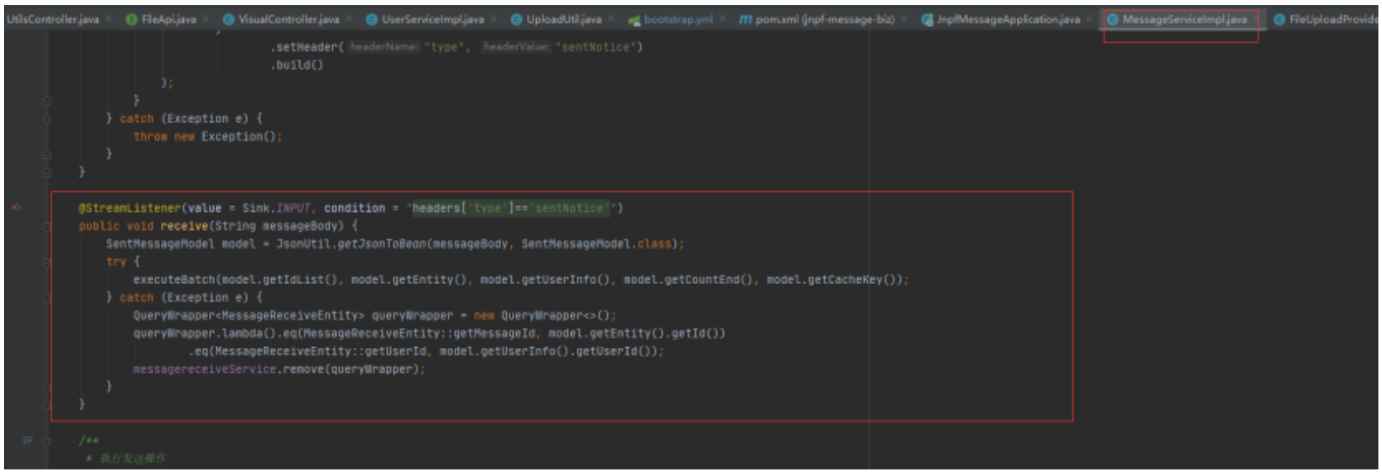
四、绑定接收者

```
@EnableBinding({Sink.class})
# 此处增加了头部验证
@StreamListener(value = Sink.INPUT, condition = "headers['type']=='sentNotice'")
```

相关代码截图：



```
package jnpf;

import ...

@SpringBootApplication
@EnableFeignClients
@EnableBinding({Sink.class})
public class JnpfMessageApplication {

    public static void main(String[] args) {
        SpringApplication.run(JnpfMessageApplication.class, args);
        System.out.println("message启动成功");
    }

}
```



五、broker参考配置如下

> 请根据实际环境配置调整

```
#其他配置见BrokerConfig,MessageStoreConfig 类

#接受客户端连接的监听端口
listenPort=11601

#nameServer 地址
namesrvAddr=192.168.20.133:11201

#网卡的 InetAddress       当前 broker 监听的 IP
brokerIP1=192.168.20.133

#跟 brokerIP1 一样        存在主从 broker 时，如果在 broker 主节点上配置了 brokerIP2 属性，br
oker 从节点会连接主节点配置的 brokerIP2 进行同步
#brokerIP2

#broker 的名称
brokerName=broker-jnpf

#DefaultCluster 本 broker 所属的 Cluser 名称
brokerClusterName=DefaultCluster

#broker id, 0 表示 master, 其他的正整数表示 slave
brokerId=0

#$HOME/store/   存储根路径
storePathRootDir=D:/Work/Storage/RocketMQ/JNPF/MAIN

#$HOME/store/commitlog/ 存储 commit log 的路径
```

```
storePathCommitLog=D:/Work/Storage/RocketMQ/LOGS/JNPF/MAIN

#1024 * 1024 * 1024(1G) commit log 的映射文件大小
#mappedFileSizeCommitLog

#在每天的什么时间删除已经超过文件保留时间的 commit log
deleteWhen=04

#以小时计算的文件保留时间
fileReservedTime=72

#SYNC_MASTER/ASYNC_MASTER/SLAVE
brokerRole=ASYNC_MASTER

#SYNC_FLUSH/ASYNC_FLUSH SYNC_FLUSH 模式下的 broker 保证在收到确认生产者之前将消息刷盘。ASYNC_FLUSH 模式下的 broker 则利用刷盘一组消息的模式，可以取得更好的性能。
flushDiskType=ASYNC_FLUSH

#是否自动创建消费组
autoCreateSubscriptionGroup=true
#是否自动创建主题
autoCreateTopicEnable=true
```

RocketMQ 集群配置

RocketMQ配置

参考RocketMQ 官方配置 `conf\2m-2s-async\` 目录下的配置文件

- **主从服务A**
  - **NameSrv: `namesrv-a.properties`**

```
#接受客户端连接的监听端口
listenPort=9876

#是否支持顺序消息，默认为false
orderMessageEnable=true
```

  - **主服务Broker: `broker-a.properties`**

```
#其他配置见BrokerConfig,MessageStoreConfig 类

#接受客户端连接的监听端口
listenPort=11601

#nameServer 地址 服务1;服务2
namesrvAddr=192.168.20.133:9876;192.168.20.133:9877

#网卡的 InetAddress     当前 broker 监听的 IP
brokerIP1=192.168.20.133

#跟 brokerIP1 一样      存在主从 broker 时，如果在 broker 主节点上配置了 brokerIP2 属性，broker 从节点会连接主节点配置的 brokerIP2 进行同步
#brokerIP2

#broker 的名称 存在主从broker时 brokerName必须一样
brokerName=broker-jnpf-a

#DefaultCluster        本 broker 所属的 Cluser 名称
brokerClusterName=DefaultCluster

#broker id, 0 表示 master, 其他的正整数表示 slave
brokerId=0

#$HOME/store/ 存储根路径
storePathRootDir=D:/Work/Storage/RocketMQ/JNPF/A-M

#$HOME/store/commitlog/        存储 commit log 的路径
storePathCommitLog=D:/Work/Storage/RocketMQ/LOGS/JNPF/A-M

#1024 * 1024 * 1024(1G)        commit log 的映射文件大小
#mappedFileSizeCommitLog

#在每天的什么时间删除已经超过文件保留时间的 commit log
deleteWhen=04

#以小时计算的文件保留时间
fileReservedTime=72

#SYNC_MASTER/ASYNC_MASTER  同步复制/异步复制      从broker固定SLAVE
brokerRole=ASYNC_MASTER

#SYNC_FLUSH/ASYNC_FLUSH SYNC_FLUSH 模式下的 broker 保证在收到确认生产者之前将消息刷盘。ASYNC_FLUSH 模式下的 broker 则利用刷盘一组消息的模式，可以取得更好的性能。
flushDiskType=ASYNC_FLUSH
```

```
#是否自动创建消费组
autoCreateSubscriptionGroup=true
#是否自动创建主题
autoCreateTopicEnable=true
```

- **从服务Broker：** `broker-a-s.properties`

```
#配置同上 修改以下配置

#接受客户端连接的监听端口
listenPort=11651

#broker id, 0 表示 master, 其他的正整数表示 slave
brokerId=1

#$HOME/store/ 存储根路径
storePathRootDir=D:/Work/Storage/RocketMQ/JNPF/A-S

#$HOME/store/commitlog/        存储 commit log 的路径
storePathCommitLog=D:/Work/Storage/RocketMQ/LOGS/A-S

 #SYNC_MASTER/ASYNC_MASTER  同步复制/异步复制      从broker固定SLAVE
brokerRole=SLAVE
```

- **主从服务B**

- **NameSrv:** `namesrv-b.properties`

```
#接受客户端连接的监听端口
listenPort=9877

#是否支持顺序消息，默认为false
orderMessageEnable=true
```

- **主服务Broker：** `broker-b.properties`

```
#其他配置见BrokerConfig,MessageStoreConfig 类

#接受客户端连接的监听端口
listenPort=11701

#nameServer 地址 服务1;服务2
namesrvAddr=192.168.20.133:9876;192.168.20.133:9877
```

```
#网卡的 InetAddress     当前 broker 监听的 IP
brokerIP1=192.168.20.133

#跟 brokerIP1 一样      存在主从 broker 时，如果在 broker 主节点上配置了 brokerIP2 属性，broker 从节点会连接主节点配置的 brokerIP2 进行同步
#brokerIP2

#broker 的名称 存在主从broker时 brokerName必须一样
brokerName=broker-jnpf-b

#DefaultCluster        本 broker 所属的 Cluser 名称
brokerClusterName=DefaultCluster

#broker id, 0 表示 master, 其他的正整数表示 slave
brokerId=0

#$HOME/store/ 存储根路径
storePathRootDir=D:/Work/Storage/RocketMQ/JNPF/B-M

#$HOME/store/commitlog/        存储 commit log 的路径
storePathCommitLog=D:/Work/Storage/RocketMQ/LOGS/JNPF/B-M

#1024 * 1024 * 1024(1G)        commit log 的映射文件大小
#mappedFileSizeCommitLog

#在每天的什么时间删除已经超过文件保留时间的 commit log
deleteWhen=04

#以小时计算的文件保留时间
fileReservedTime=72

 #SYNC_MASTER/ASYNC_MASTER  同步复制/异步复制      从broker固定SLAVE
brokerRole=ASYNC_MASTER

#SYNC_FLUSH/ASYNC_FLUSH SYNC_FLUSH 模式下的 broker 保证在收到确认生产者之前将消息刷盘。ASYNC_FLUSH 模式下的 broker 则利用刷盘一组消息的模式，可以取得更好的性能。
flushDiskType=ASYNC_FLUSH

#是否自动创建消费组
autoCreateSubscriptionGroup=true
#是否自动创建主题
autoCreateTopicEnable=true
```

- **从服务Broker：** `broker-b-s.properties`

```
#配置同上 修改以下配置

#接受客户端连接的监听端口
listenPort=11702

#broker id, 0 表示 master, 其他的正整数表示 slave
brokerId=1

#$HOME/store/ 存储根路径
storePathRootDir=D:/Work/Storage/RocketMQ/JNPF/B-S

#$HOME/store/commitlog/       存储 commit log 的路径
storePathCommitLog=D:/Work/Storage/RocketMQ/LOGS/JNPF/B-S

 #SYNC_MASTER/ASYNC_MASTER  同步复制/异步复制      从broker固定SLAVE
brokerRole=SLAVE
```

设置环境变量

添加 `ROCKETMQ_HOME` 设置为RocketMQ安装目录

启动命令

- 启动NameSrv

```
mqnamesrv.cmd -c ../conf/2m-2s-sync/namesrv-a.properties
mqnamesrv.cmd -c ../conf/2m-2s-sync/namesrv-b.properties
```

- 启动Broker

```
mqbroker.cmd -c ../conf/2m-2s-sync/broker-a.properties
mqbroker.cmd -c ../conf/2m-2s-sync/broker-a-s.properties
mqbroker.cmd -c ../conf/2m-2s-sync/broker-b.properties
mqbroker.cmd -c ../conf/2m-2s-sync/broker-b-s.properties
```

修改项目配置 `jnpf-message\jnpf-message-server\src\main\resources\bootstrap.yml`

```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          name-server: 192.168.20.133:9877;192.168.20.133:9876
```

消息队列更换， RabbitMQ，Kafka

RabbitMQ

- 修改jnpf-message/jnpf-message-biz/pom.xml的pom.xml文件

```
配置文件修改前：
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-rocketmq</artifactId>
</dependency>

修改后：
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
</dependency>
```

- 修改消息服务配置文件 jnpf-message\jnpf-message-server\src\main\resources\bootstrap.yml

```
旧配置：
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          name-server: 192.168.0.34:30095
        bindings:
          output:
```

```
            producer:
              sync: true
              group: jnpf-group
      bindings:
        output:
          content-type: text/json
          destination: jnpf-topic
          group: jnpf-group
        input:
          content-type: text/json
          destination: jnpf-topic
          group: jnpf-group
```

新加入的配置：

```
spring:
  cloud:
    # stream配置
    stream:
      binders: # 在此处配置要绑定的rabbitmq的服务信息
        defaultRabbit: # 表示定义的名称，用于binding整合
          type: rabbit # 消息组件类型
          environment: # 设置rabbitmq的相关环境配置
            spring:
              rabbitmq:
                host: localhost
                port: 5672
                username: guest
                password: guest
      bindings:
        output:
          destination: jnpf-topic # 表示要使用的Exchange名称定义
          content-type: application/json # 设置消息类型
          binder: defaultRabbit # 设置要绑定的消息服务的具体设置
          group: jnpf-group
        input: # 这个名字是一个通道的名称
          destination: jnpf-topic # 表示要使用的Exchange名称定义
          content-type: application/json # 设置消息类型
          binder: defaultRabbit # 设置要绑定的消息服务的具体设置
          group: jnpf-group
```

Kafka

- 修改jnpf-message/jnpf-message-biz/pom.xml的pom.xml文件

配置文件修改前：

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-rocketmq</artifactId>
</dependency>
```

修改后：

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-kafka</artifactId>
</dependency>
```

- 修改消息服务配置文件 jnpf-message\jnpf-message-server\src\main\resources\bootstrap.yml

旧配置：

```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          name-server: 192.168.0.34:30095
        bindings:
          output:
            producer:
              sync: true
              group: jnpf-group
      bindings:
        output:
          content-type: text/json
          destination: jnpf-topic
          group: jnpf-group
        input:
          content-type: text/json
          destination: jnpf-topic
          group: jnpf-group
```

新加入的配置：

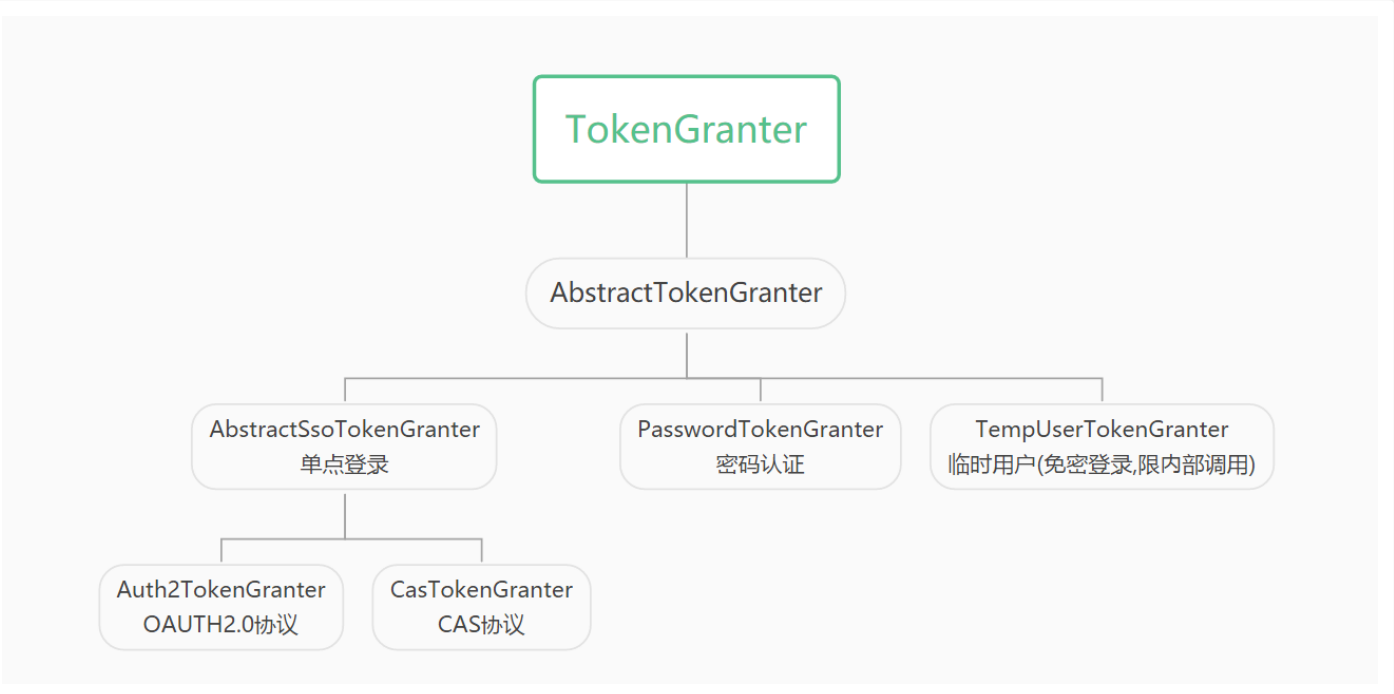
```
spring:
  cloud:
    stream:
      kafka:
        binder:
          name-server: localhost:9092
      bindings:
        output:
```

```
          content-type: text/json
          destination: jnpf-topic
          group: jnpf-group
        input:
          content-type: text/json
          destination: jnpf-topic
          group: jnpf-group
```

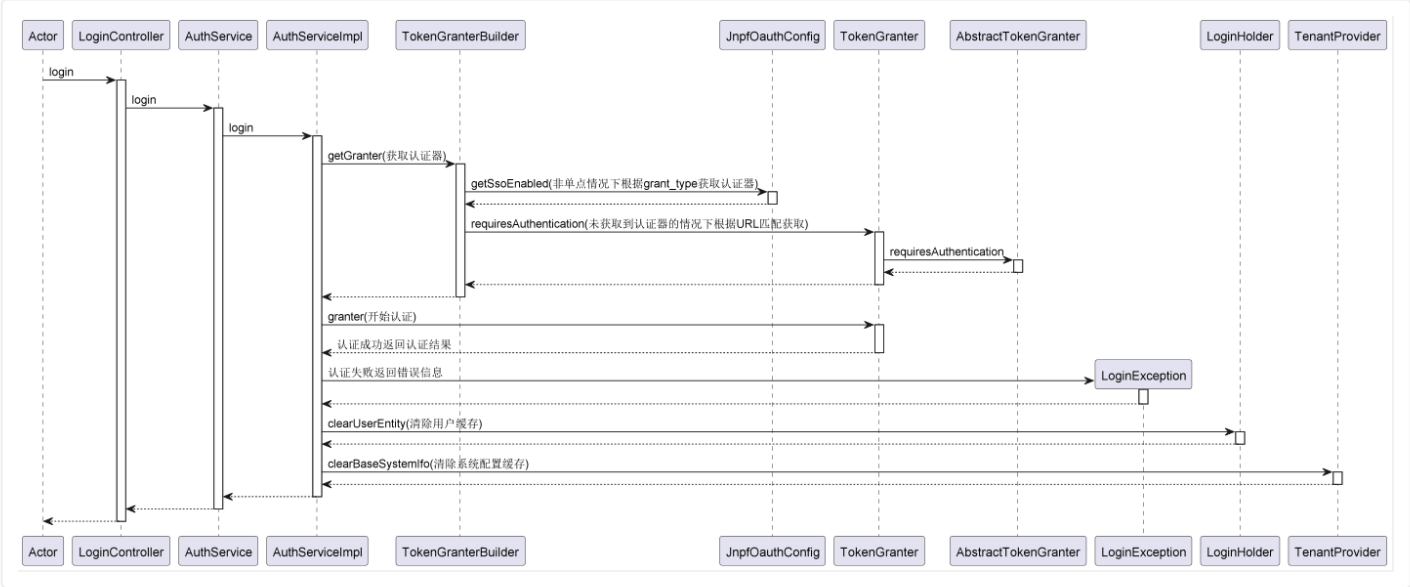
登录与认证信息管理(3.4.3)

> 本文档适合JNPF 3.4.3起的版本

1. 登录授权模块关系



2. 登录认证流程



3. 认证信息缓存与使用

认证信息缓存

Redis缓存

- `Authorization:login:token:用户TOKEN`, 每个Token的缓存, 存储LoginID
- `Authorization:login:token-session:用户TOKEN`, 每个Token的用户缓存， 存储UserInfo
- `Authorization:login:session:租户ID:用户ID`, 每个用户的统一缓存, 存储用户接口权限、当前用户的Token列表
- `Authorization:var:id-token`, 临时ID缓存, 用于鉴定请求来源

本地缓存

> UserProvider._USER_CACHE _线程缓存

临时切换用户

- 在非Web环境无法获取Token的情况切换用户

```
//引用jnpf-oauth-api模块

//登录临时用户 此用户已经登录将返回现有用户Token 未登录将直接使用用户ID进行免密登录返回Token
String token = AuthUtil.loginTempUser(userId, tenantId);
//获取Redis中的UserInfo
UserInfo userInfo = UserProvider.getUser(token);
```

```
//将UserInfo 存入本地缓存
UserProvider.setLocalLoginUser(userInfo);
//业务操作， 后续UserProvider.getUser()操作将会自动获取上方的UserInfo
```

- 在业务中临时切换用户

```
//备份旧的用户
UserInfo oldUser = UserProvider.getUser();
//登录临时用户 此用户已经登录将返回现有用户Token 未登录将直接使用用户ID进行免密登录返回Token
String token = AuthUtil.loginTempUser(userId, tenantId);
//获取Redis中的UserInfo
UserInfo userInfo = UserProvider.getUser(token);
//将UserInfo 存入本地缓存
UserProvider.setLocalLoginUser(userInfo);
//业务操作， 后续UserProvider.getUser()操作将会自动获取上方的UserInfo

//业务完成后， 将UserInfo还原
UserProvider.setLocalLoginUser(oldUser);
```

UserProvider.getUser()

- 优先获取本地线程缓存中的UserInfo
- 若不存在调用UserProvider.*getUser*(token), 根据当前Token获取对应的UserInfo

4. 跨服务传递用户信息

4.1. Dubbo

> jnpf-common-dubbo模块中 `MyDubboTokenFilter` 过滤器

1. Web环境中且有Token时调用Dubbo自动设置认证信息
2. 若在非Web环境中或无Token调用Dubbo， 参考上方临时切换用户, 先设置用户信息

认证信息自动设置说明

- Consumer端获取当前UserInfo, 设置到Dubbo属性中
- Provider端获取Dubbo属性的中的UserInfo
- 将UserInfo设置到本地线程缓存

4.2. OpenFeign

> FeignHolder, Feign调用帮助类
> FeignConfig, 认证信息配置类

非Web环境调用

1. 有Token的情况，直接将Token传入调用
2. 无Token的情况, 参考上方临时切换用户， 使用Token传入或设置本地缓存(结合下方FeignConfig顺序说明)

```
Map<String, String> headers = ImmutableMap.of(Constants.AUTHORIZATION.toLowerCas
e(), token);
//FeignHolder.asyncFeign 异步调用
UserEntity entity = FeignHolder.sendFeign(headers, ()-> {
 //其他业务逻辑
 return userApi.getInfoById(toUserId);
});
```

FeignConfig配置认证信息顺序

1. FeignHolder.*get*(), 获取FeignHolder设置的本地缓存中的Token
2. UserProvider.*getLocalLoginUser*(), 获取本地用户信息中的Token
3. 获取Request Header中的Token

5. 服务中的请求拦截器

> SecurityConfiguration

- 续期Token， Redis获取UserInfo存至本地缓存后续无需访问Redis重新获取
- 根据配置决定是否校验请求来源
- 请求结束后清空当前请求的UserInfo本地缓存
- 接口鉴权初始化

jnpf-netcore

快速开始

开发环境运行

本地环境准备

项目说明

> 以下为 `JNPF 3.4.x` .NET项目命名规则

项目名	说明
JNPF-Dotnet	.NET主项目
JNPF-Dotnet-Tentant	.NET多租户项目
JNPF.DataReport	Java报表项目
JNPF-file-preview	Java文档预览项目

环境要求

基础要求

> 环境具体安装教程

环境	版本	描述
Visual Studio 2022	16.8.x	.NetCore开发工具及开发环境(Windows)
Visual Studio for Mac	默认最新版本	.NetCore开发工具及开发环境(MacOS)
IntelliJ IDEA	2020.1 +	Java开发工具及开发环境(MacOS)
JDK	1.8.x	JAVA环境依赖(需配置环境变量)
SDK	5.0版	.NET环境依赖(后端代码版本是3.3.x及以下版本)
SDK	6.0版	.NET环境依赖(后端代码版本是3.4.x及以上版本)
Maven	3.6.3	项目构建(需配置环境变量)
Redis	3.2.100(Windows)/6.0.x(Linux,Mac)	

MySQL	5.7.x+	数据库任选一
SQLServer	2012+	数据库任选一
Orale	11g+	数据库任选一

推荐IDE

> 特别说明：IntelliJ IDEA 2019.1 和 Maven 3.6.3 存在不兼容问题

- Visual Studio 2022 16.8.4 + (Windows)
- Visual Studio for Mac (MacOS)
- IntelliJ IDEA 2020.1 +

推荐插件

- `Lombok`
- `MyBatisPlus`
- `Alibaba Java Coding Guidelines`

JDK 1.8.x

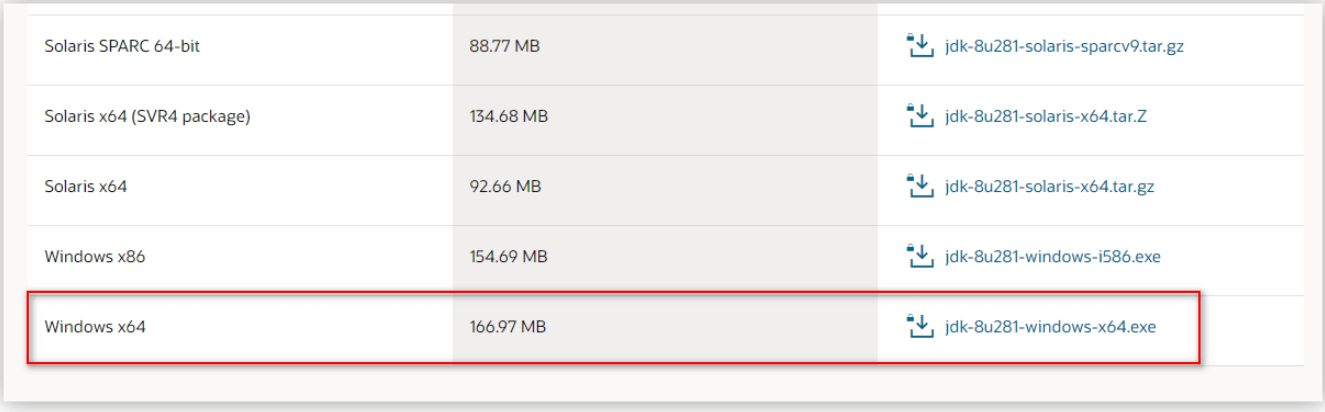
> 报表开发环境

特别说明

日前发现，`jdk1.8.0_25`版本无法正常运行项目。

下载并安装

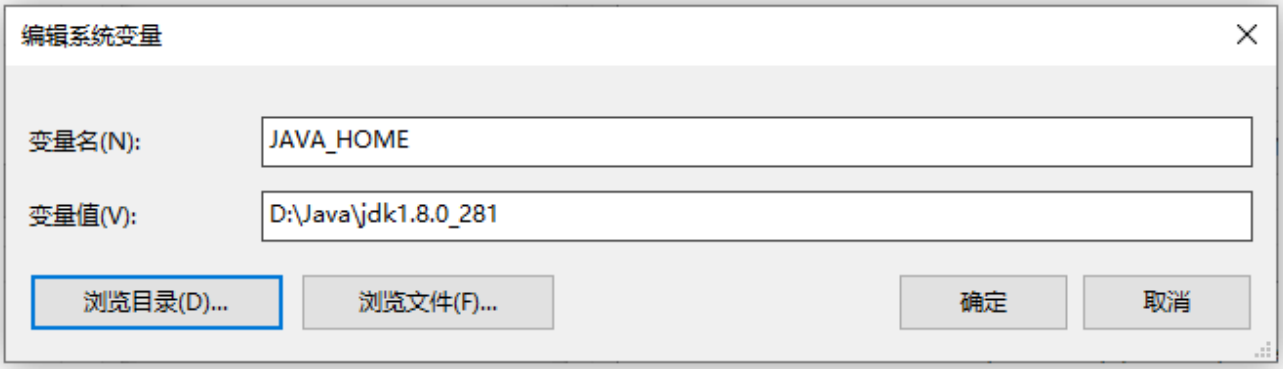
打开`https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html`,选择`Windows x64`版本下载并安装



环境变量配置

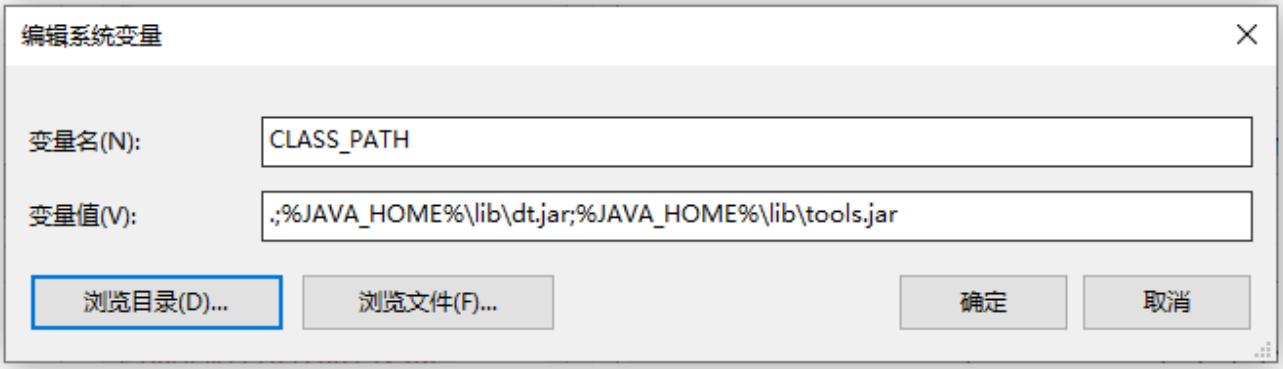
1）新建系统变量`JAVA_HOME`

- 变量名：`JAVA_HOME`
- 变量值：`D:\Java\jdk1.8.0_281`

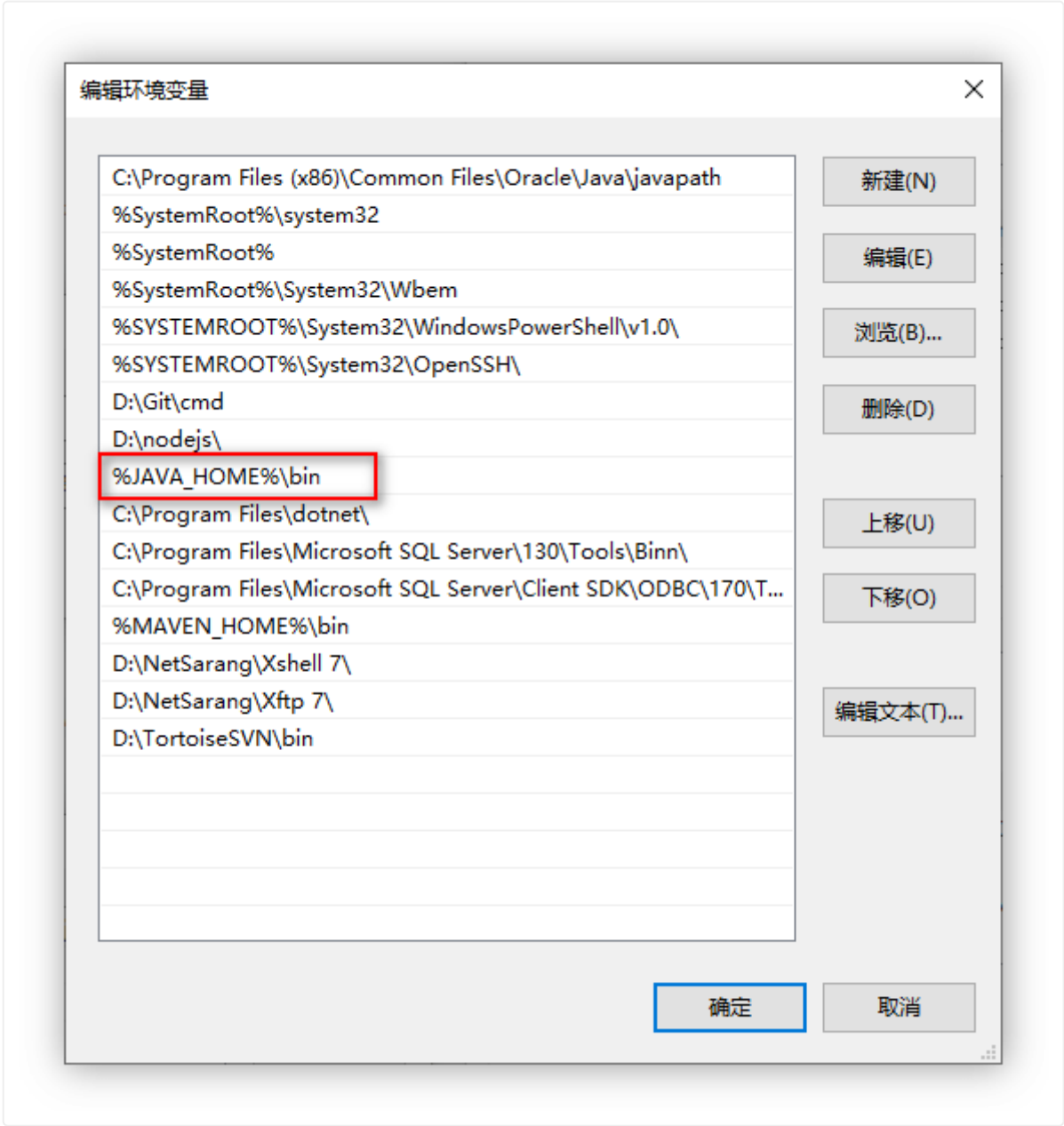


2）新建系统变量`CLASS_PATH`

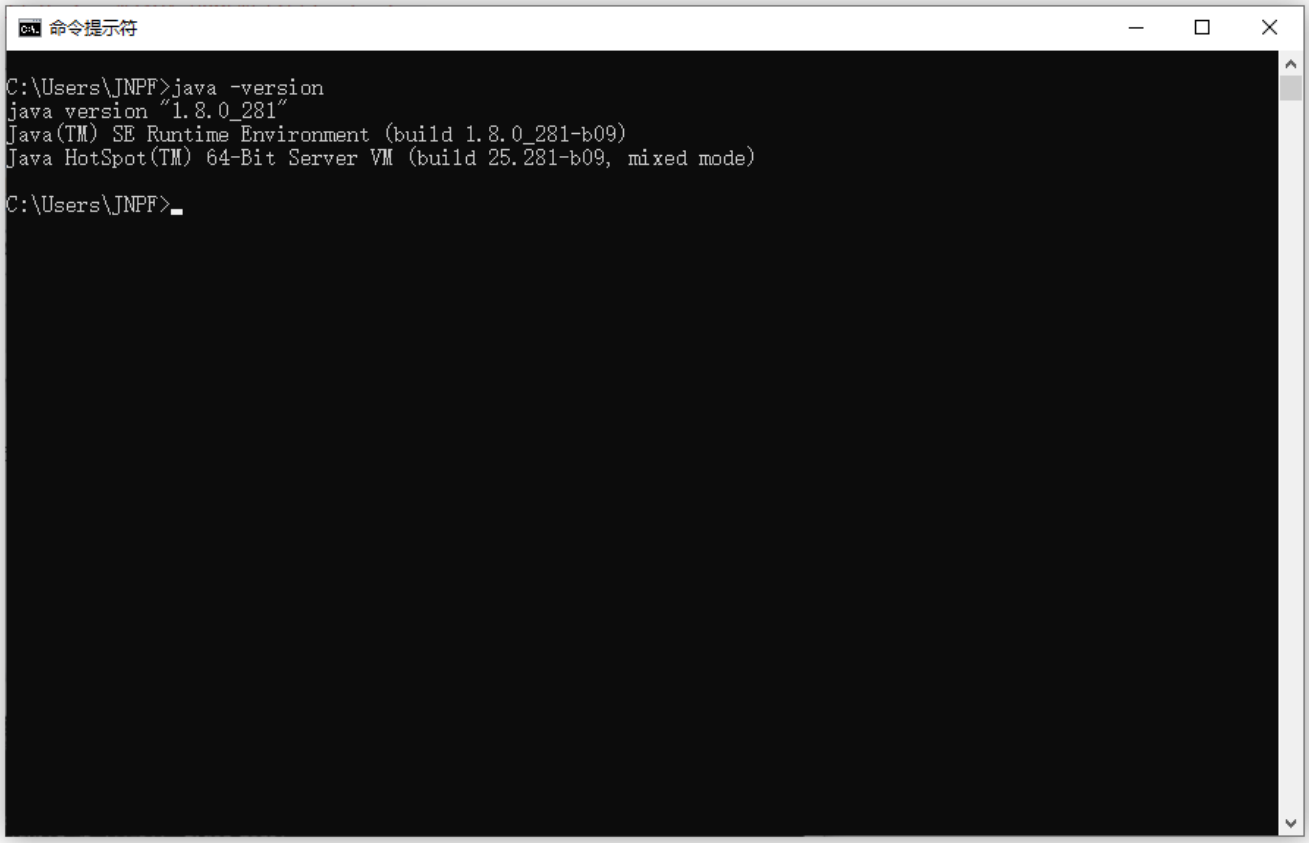
- 变量名：`CLASS_PATH`
- 变量值：`.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar`



3）双击`系统变量`中的`path`，点击新建，输入`%JAVA_HOME%\bin`



4）命令行输入 `java -version` 检查JDK环境是否配置成功

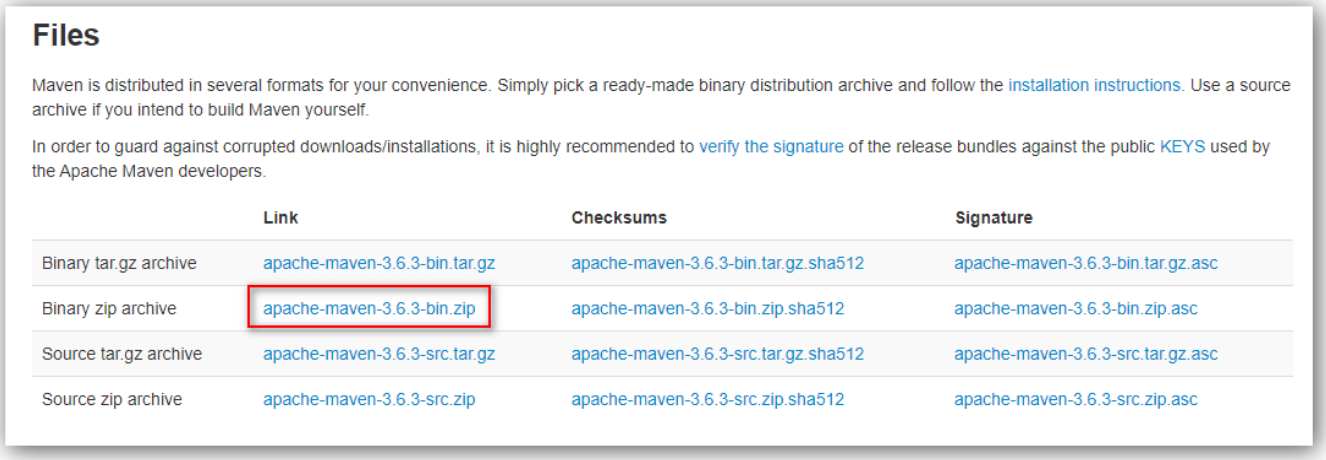


Maven 3.6.3

> JAVA版本开发环境

下载并解压

打开 `http://maven.apache.org/download.cgi` ，下载 `apache-maven-3.6.3-bin.zip` 并解压



Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the installation instructions. Use a source archive if you intend to build Maven yourself.

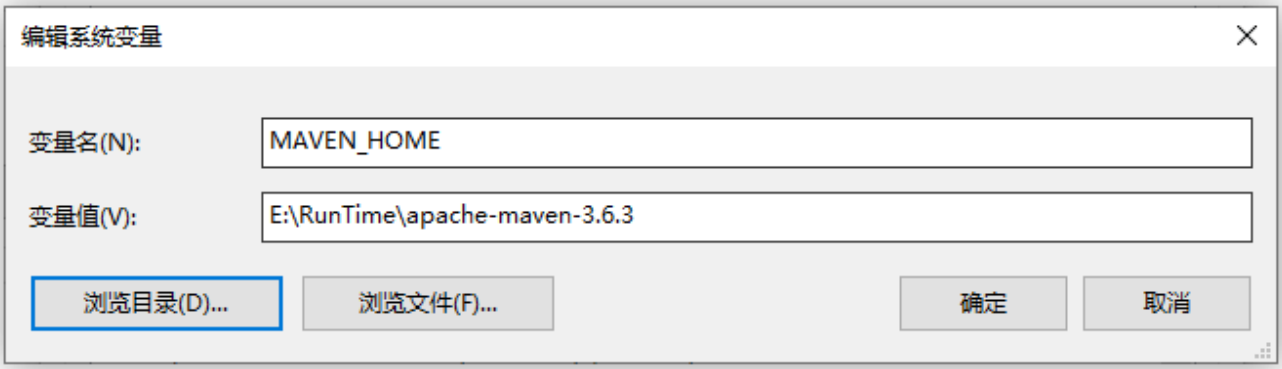
In order to guard against corrupted downloads/installations, it is highly recommended to verify the signature of the release bundles against the public KEYS used by the Apache Maven developers.

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.6.3-bin.tar.gz	apache-maven-3.6.3-bin.tar.gz.sha512	apache-maven-3.6.3-bin.tar.gz.asc
Binary zip archive	apache-maven-3.6.3-bin.zip	apache-maven-3.6.3-bin.zip.sha512	apache-maven-3.6.3-bin.zip.asc
Source tar.gz archive	apache-maven-3.6.3-src.tar.gz	apache-maven-3.6.3-src.tar.gz.sha512	apache-maven-3.6.3-src.tar.gz.asc
Source zip archive	apache-maven-3.6.3-src.zip	apache-maven-3.6.3-src.zip.sha512	apache-maven-3.6.3-src.zip.asc

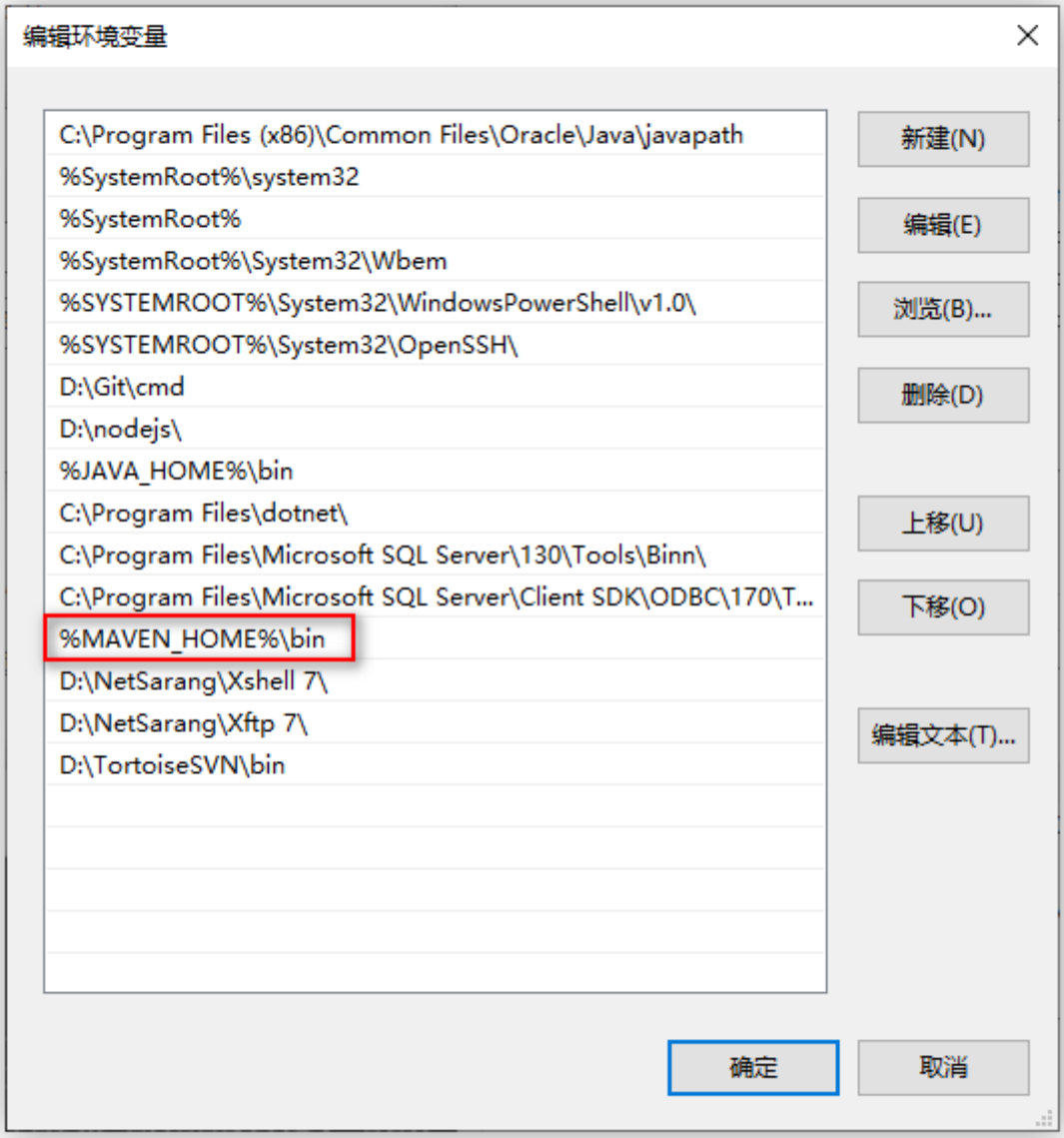
环境变量配置

1）新建系统变量`MAVEN_HOME`

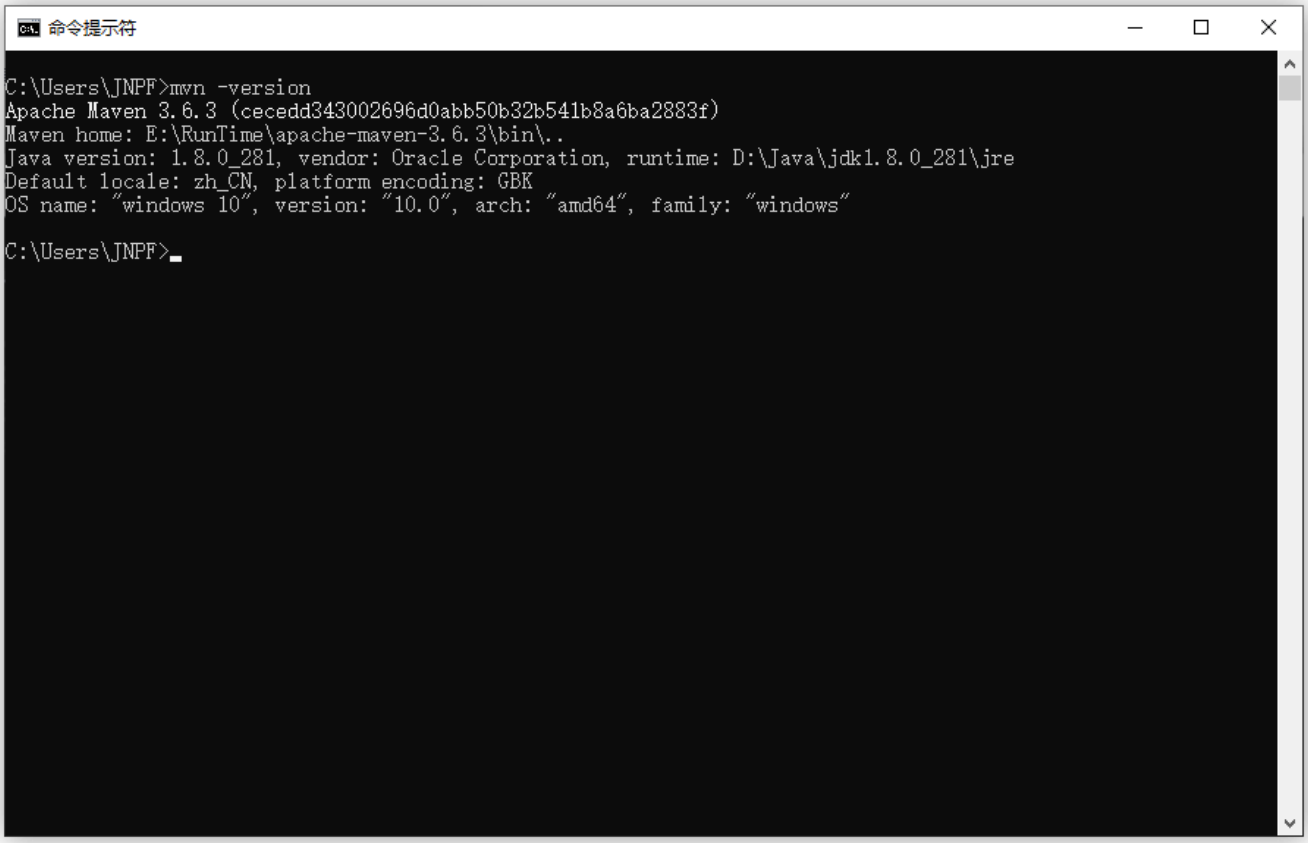
- 变量名：`MAVEN_HOME`
- 变量值：`D:\apache-maven-3.6.3`（`maven`路径）



2）双击`系统变量`中的`path`，点击新建，输入`%MAVEN_HOME%\bin`



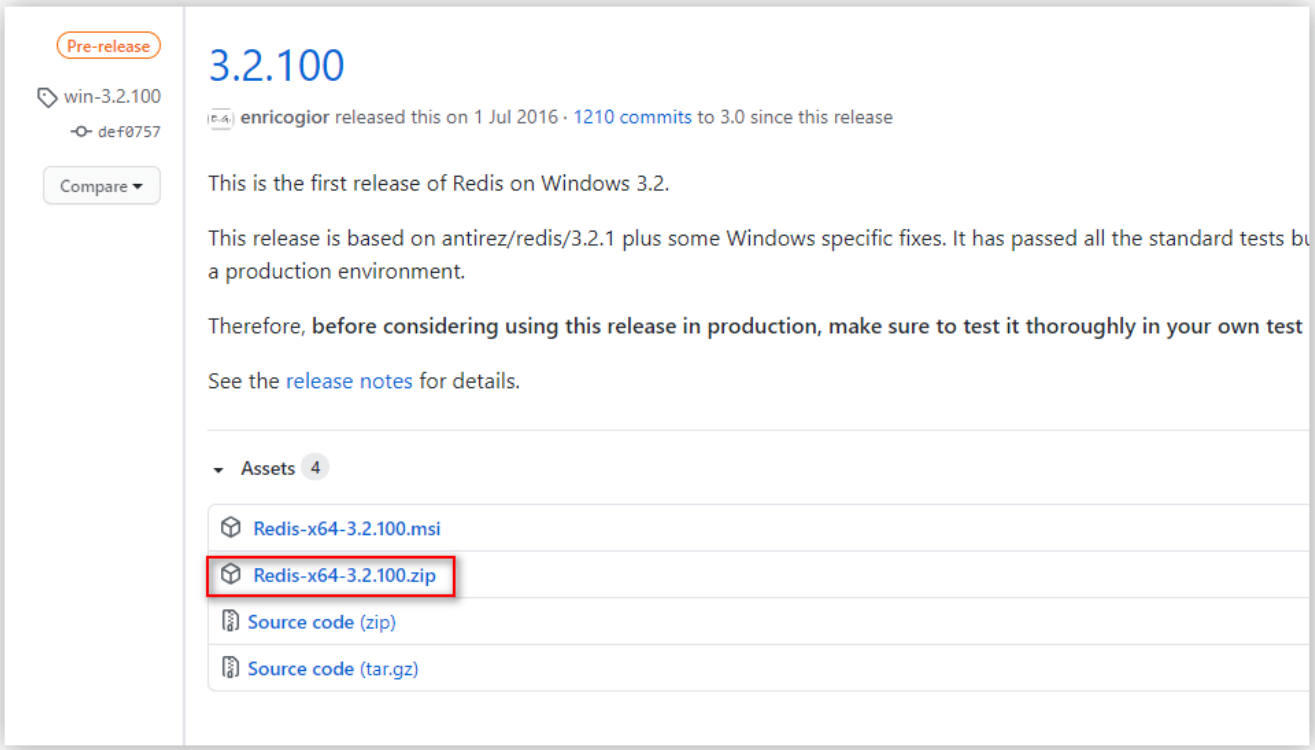
3）命令行输入 `mvn -version` 检查Maven环境是否配置成功



Redis 3.2.100

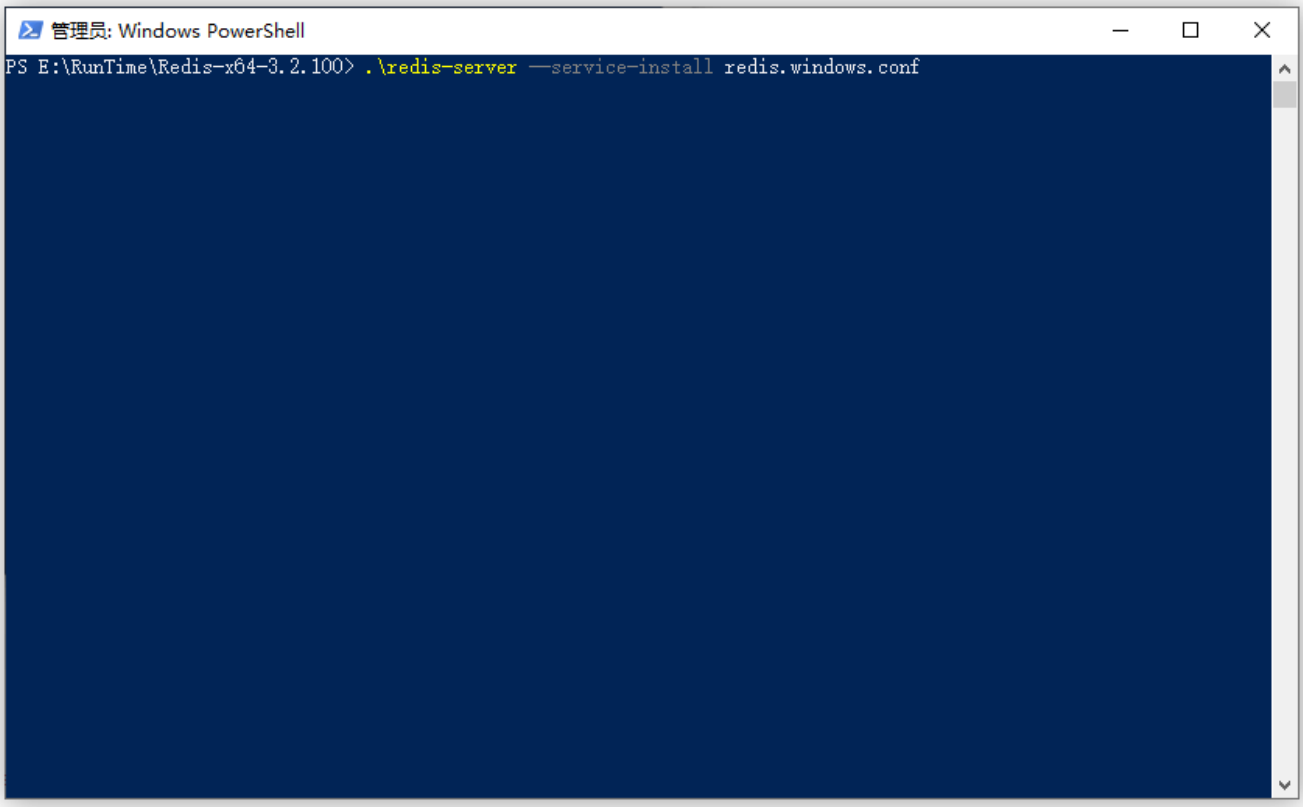
下载并解压

打开 `https://github.com/microsoftarchive/redis/releases` ，下载 `3.2.100` 版本

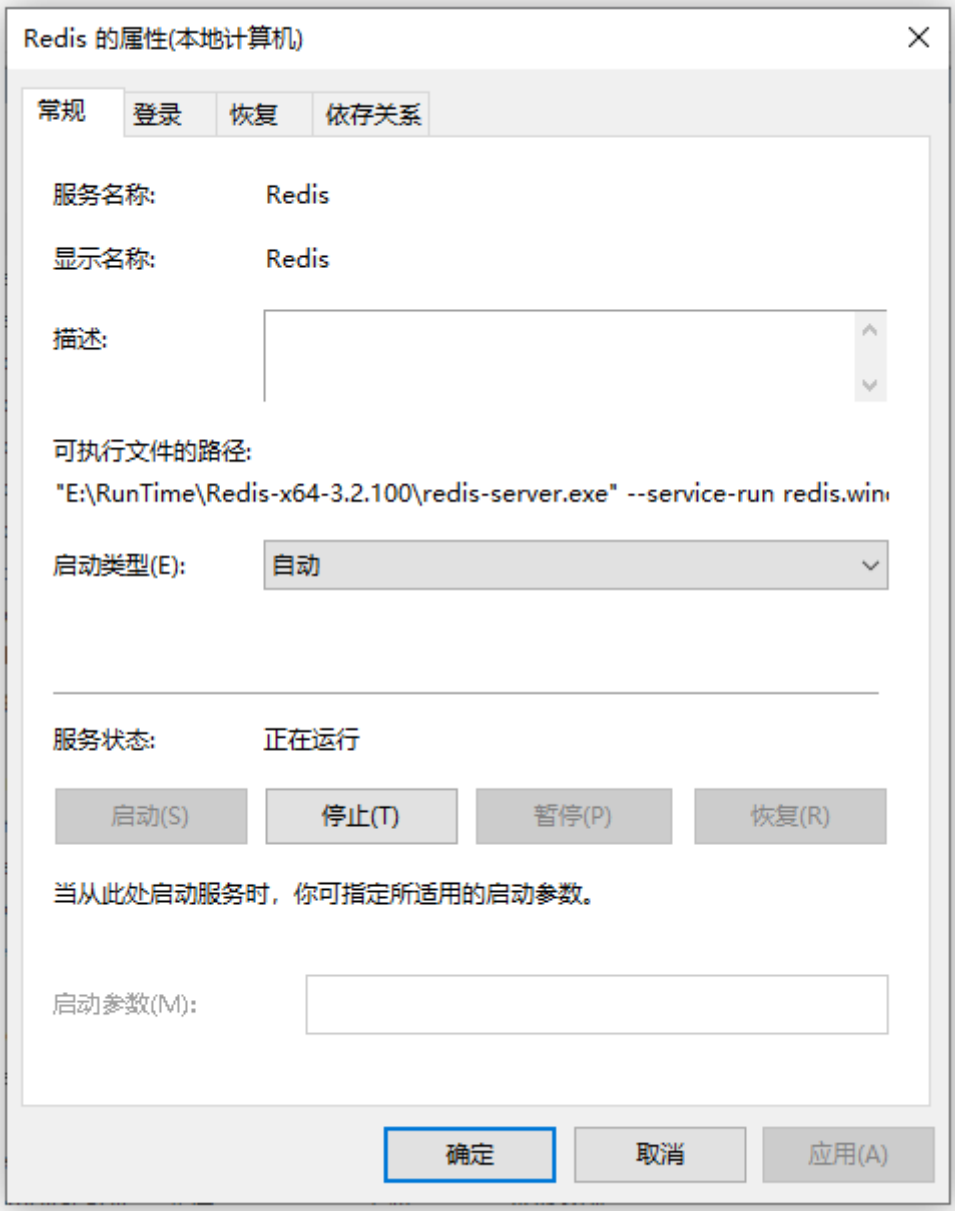


配置并设置为服务

1） 打开命令行(`Windows PowerShell`)，输入 `.\redis-server --service-install redis.windows.conf`



2） 右击 `电脑` - `管理` - `服务和应用程序` - `服务` 启动服务

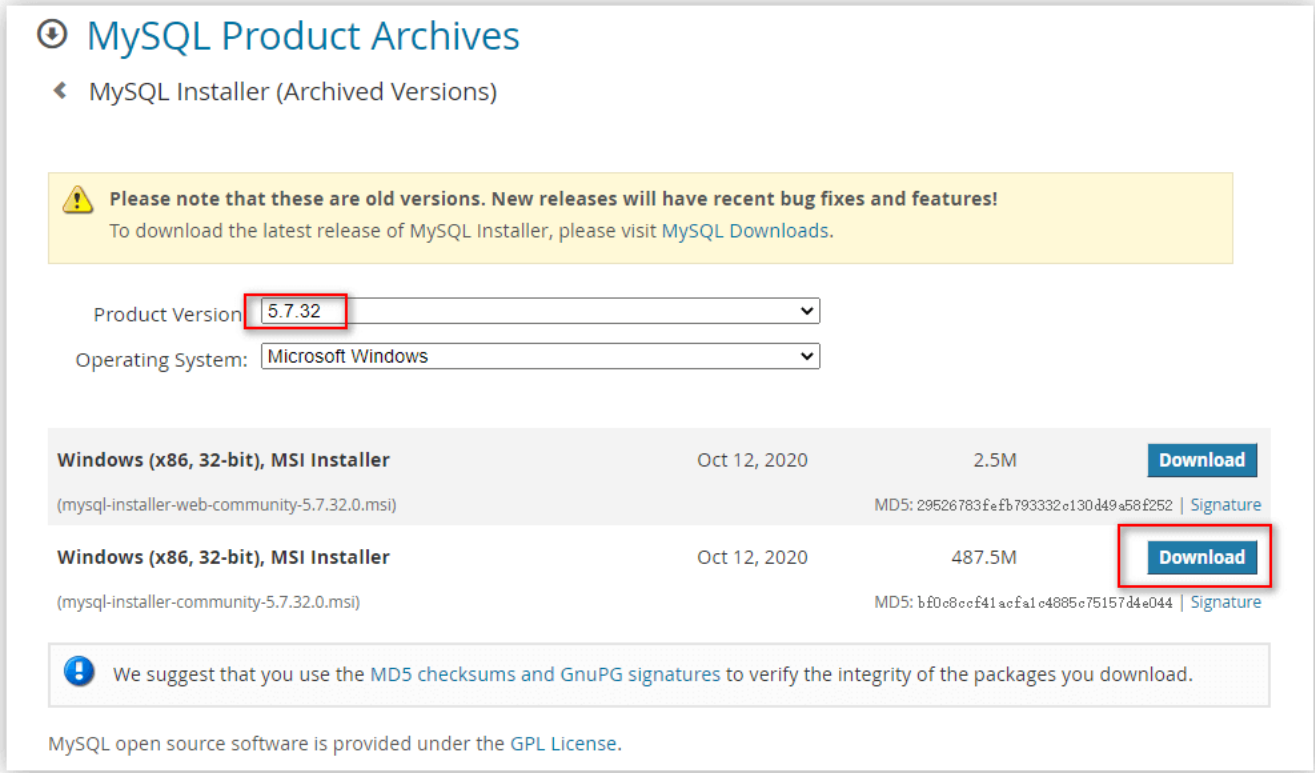


3） `Redis` 常用的指令

- 卸载服务：`redis-server --service-uninstall`
- 开启服务：`redis-server --service-start`
- 停止服务：`redis-server --service-stop`

MySQL 5.7.x

打开 `https://downloads.mysql.com/archives/installer/` ,下载 `MySQL5.7.x` ,按提示安装即可



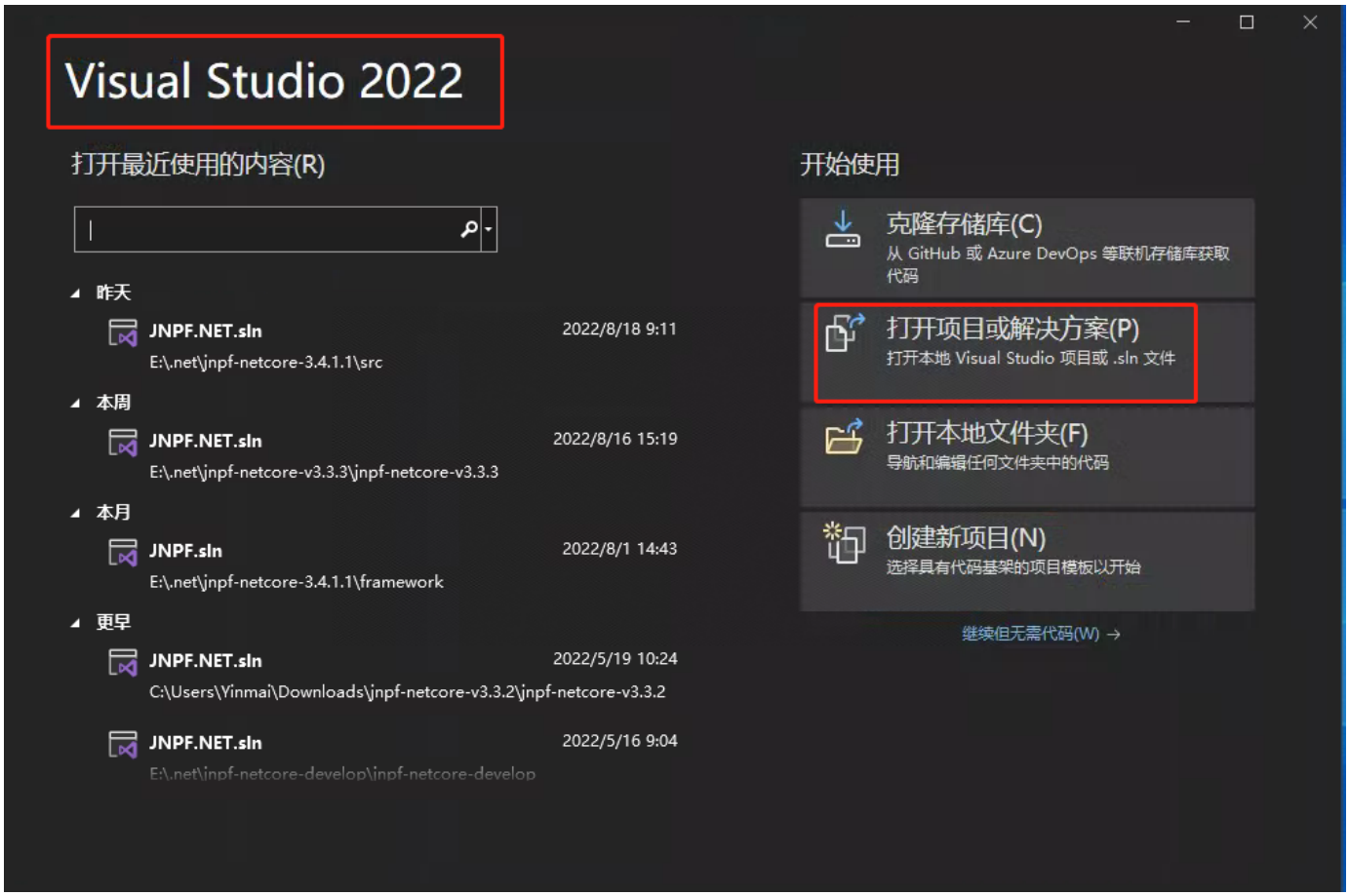
SQLServer 2012

下载 `SQL Server 2012 Developer` （全功能免费版本，许可在非生产环境下用作开发和测试数据库），按提示安装即

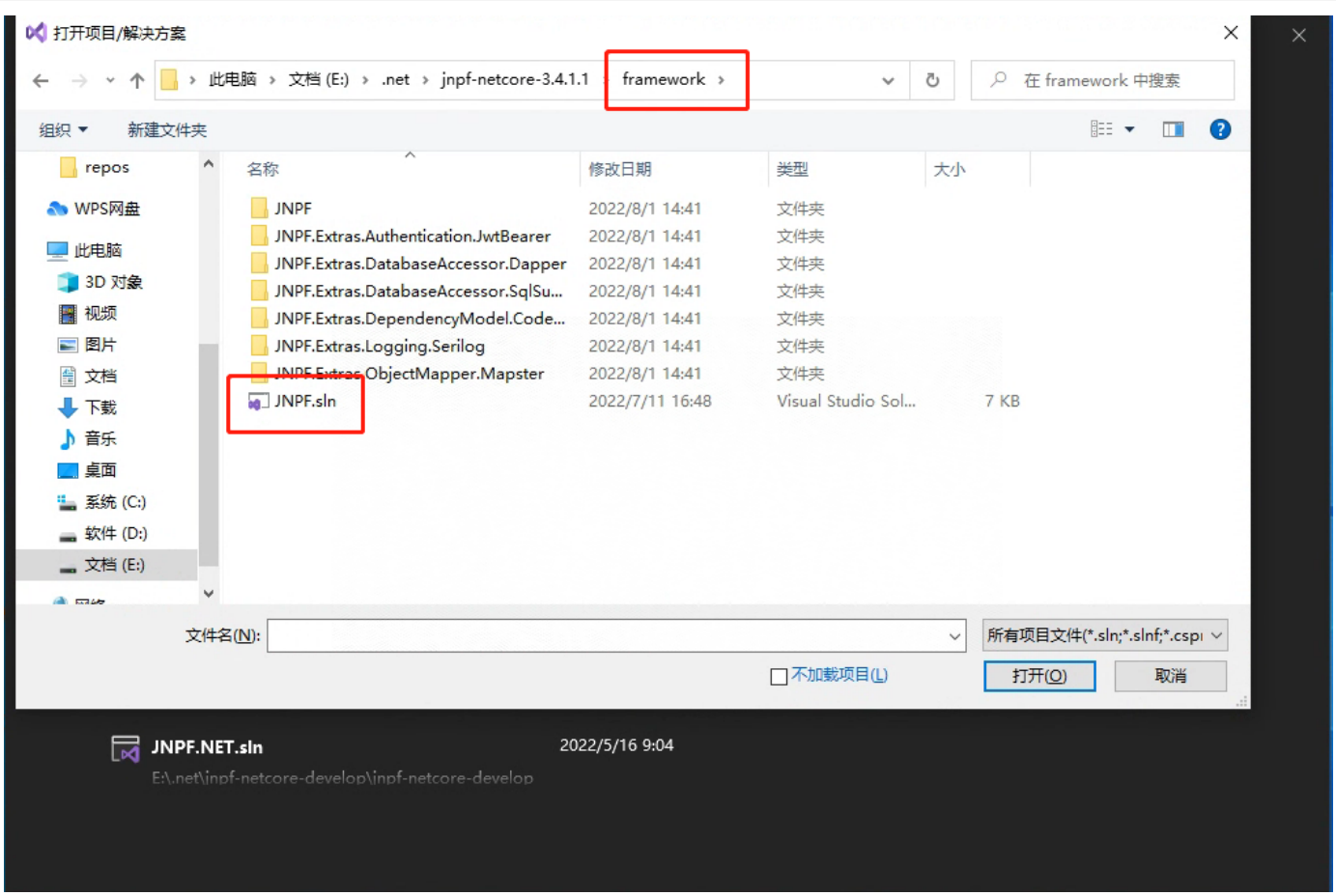
本地项目运行（3.4.x及以上版本）

环境准备

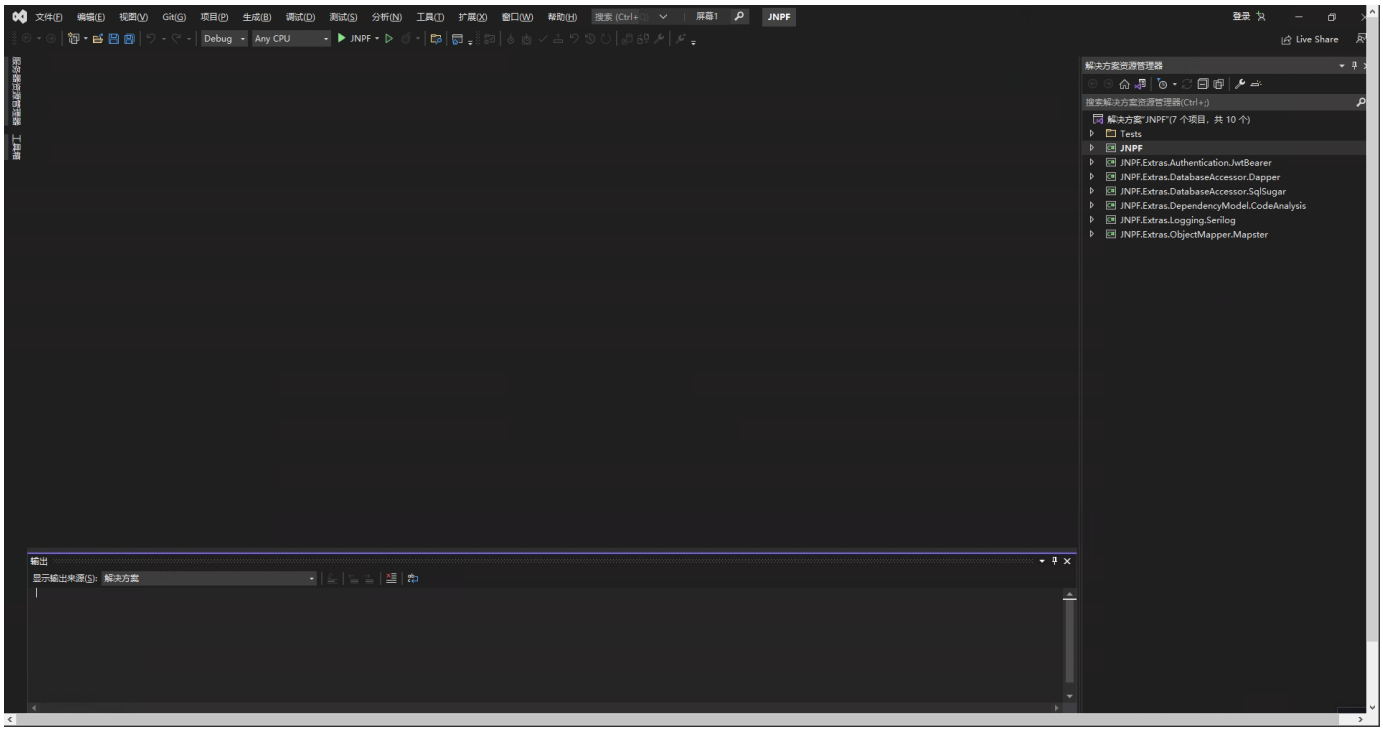
> 参考 `.NETCORE` -环境准备
>
> ## 项目导入
>
> - 打开 `Visual Studio 2022` ,打开项目或解决方案，如下图



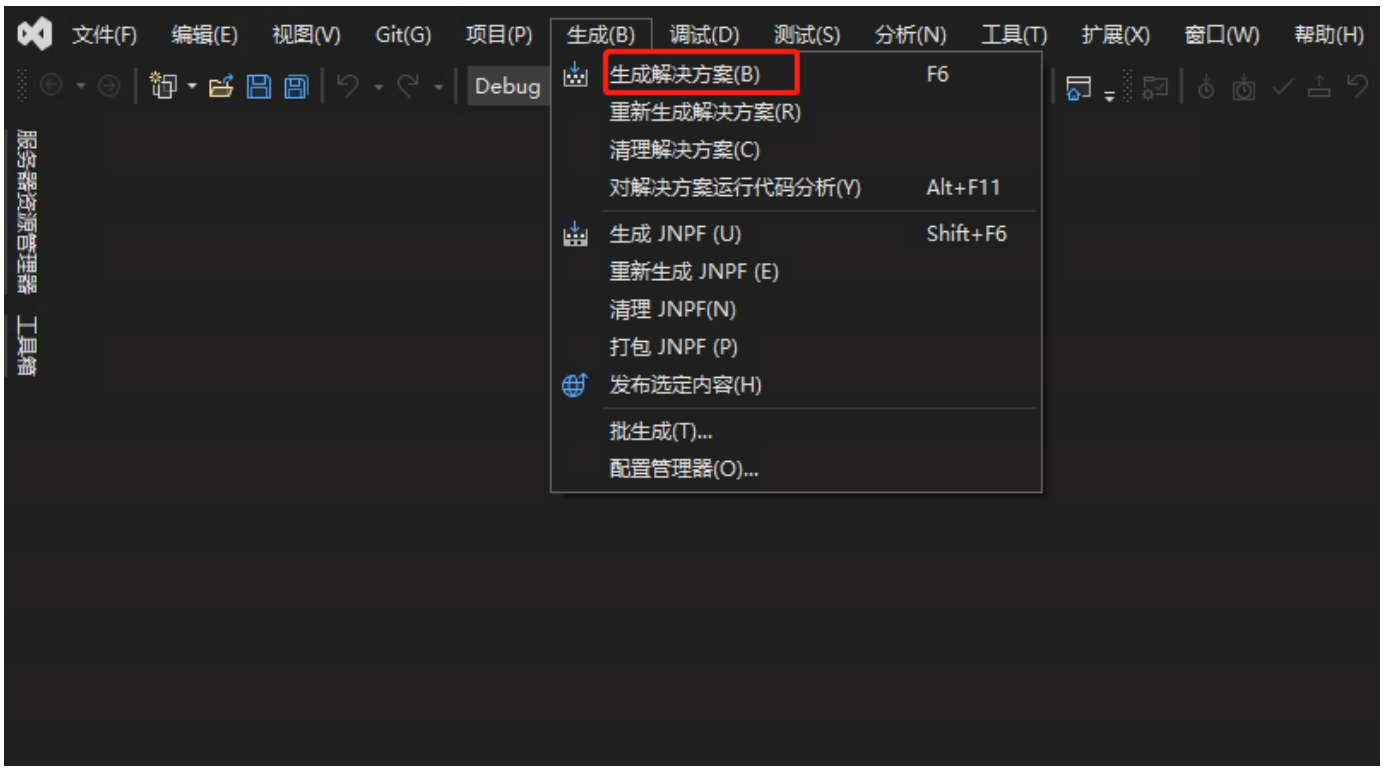
- 选择项目根目录下的framework文件夹里面的 `JNPF.sln` 文件



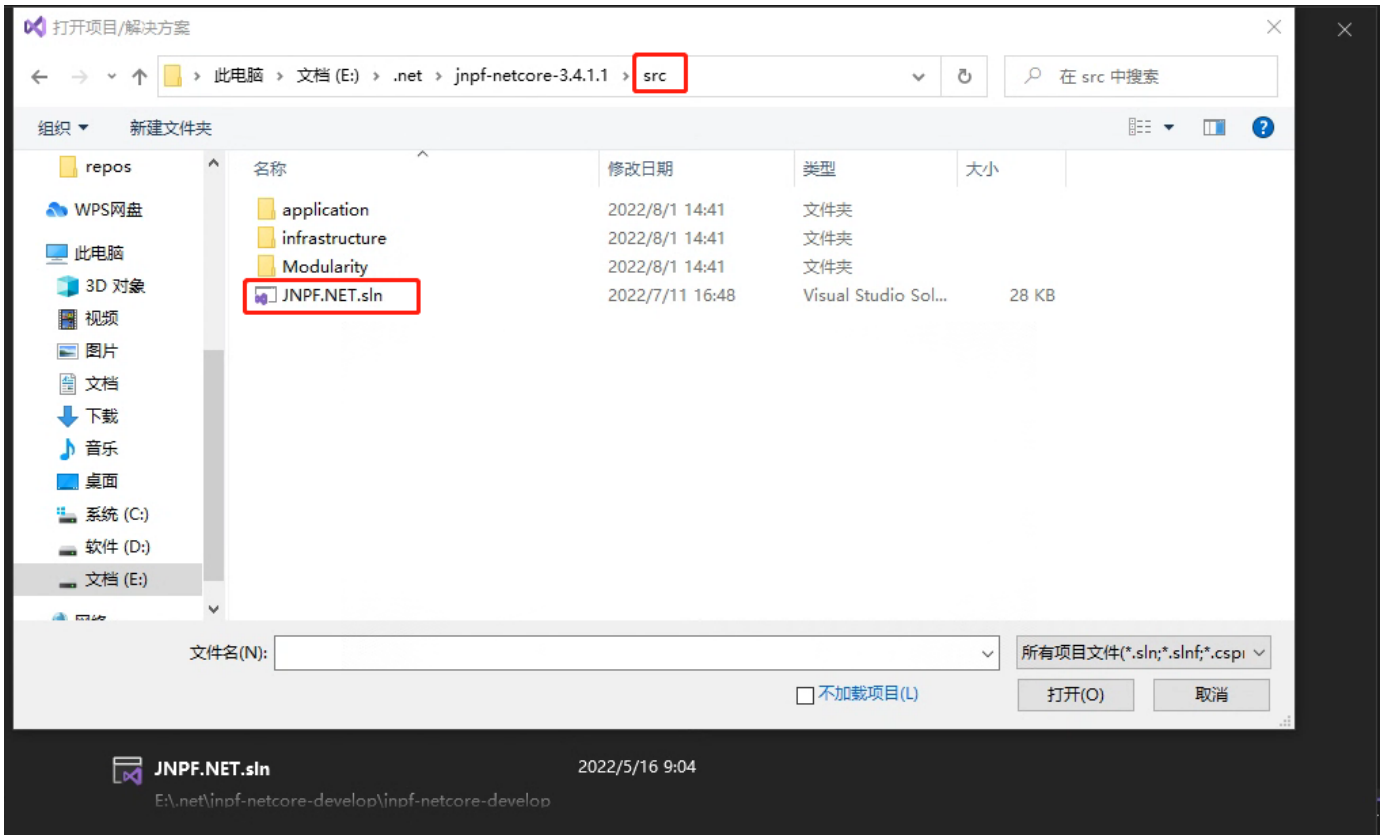
- 打开后，如下图界面



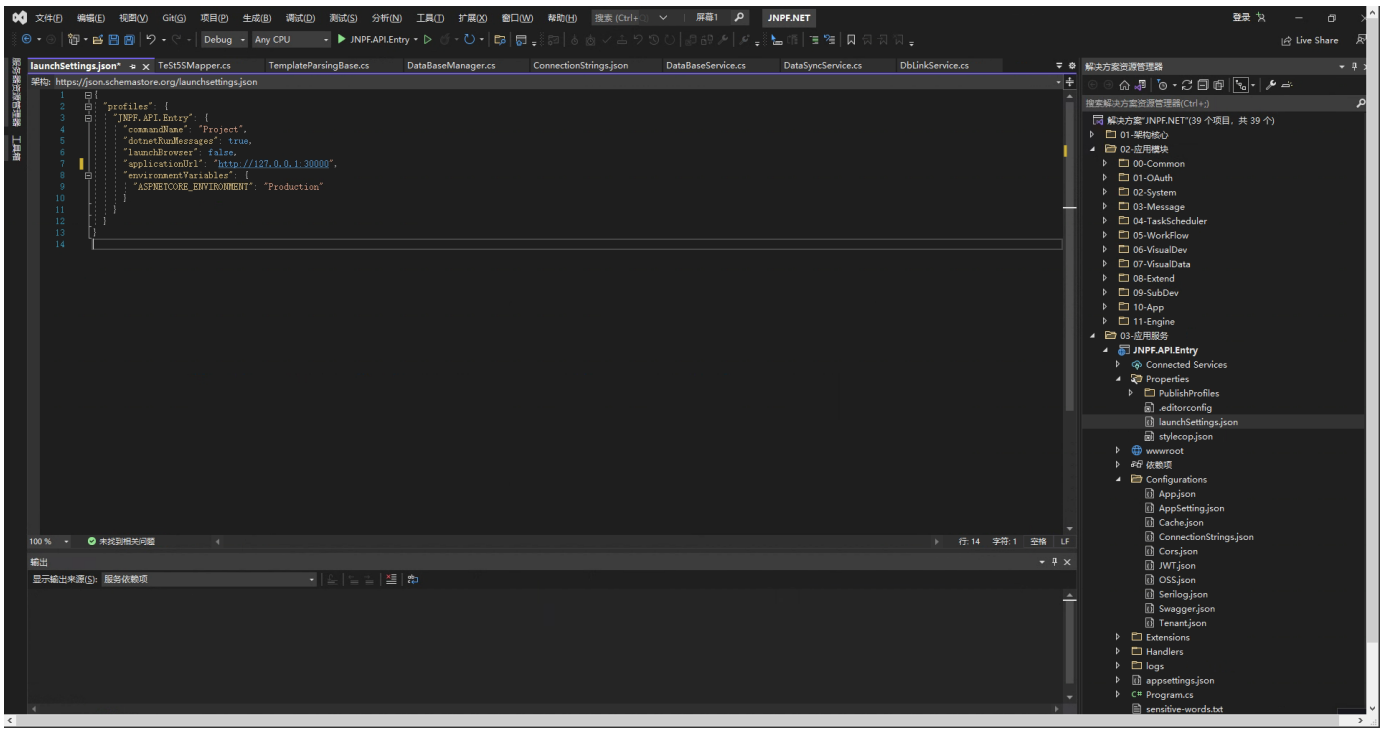
- 打开后，生成解决方案，然后关掉



- 再选择项目根目录下的src文件夹里面的 `JNPF.NET.sln` 文件

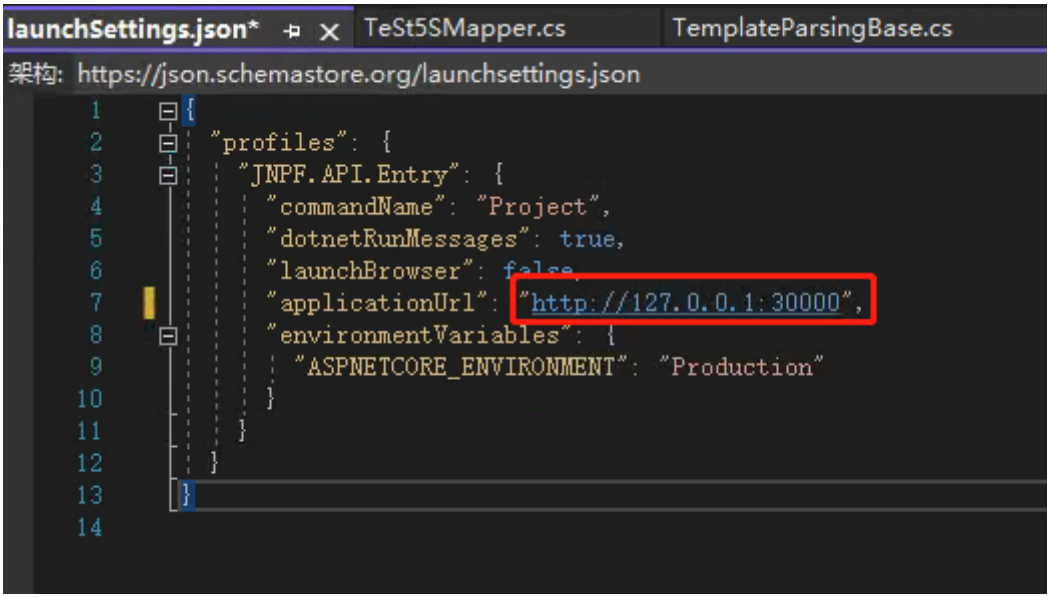


- 打开后，如下图界面

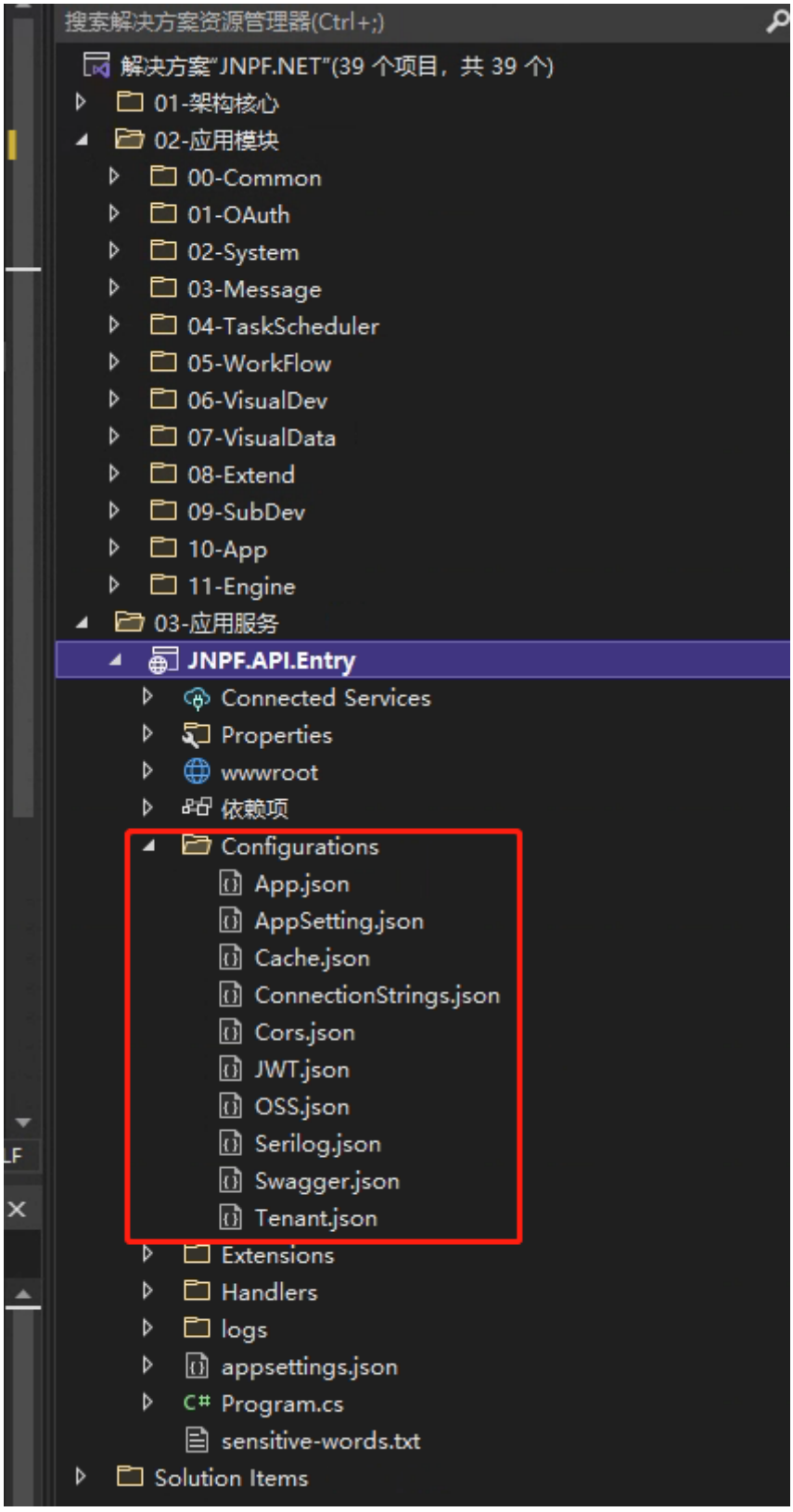


项目运行

1. 打开 `launchSettings.json` 修改applicationUrl参数

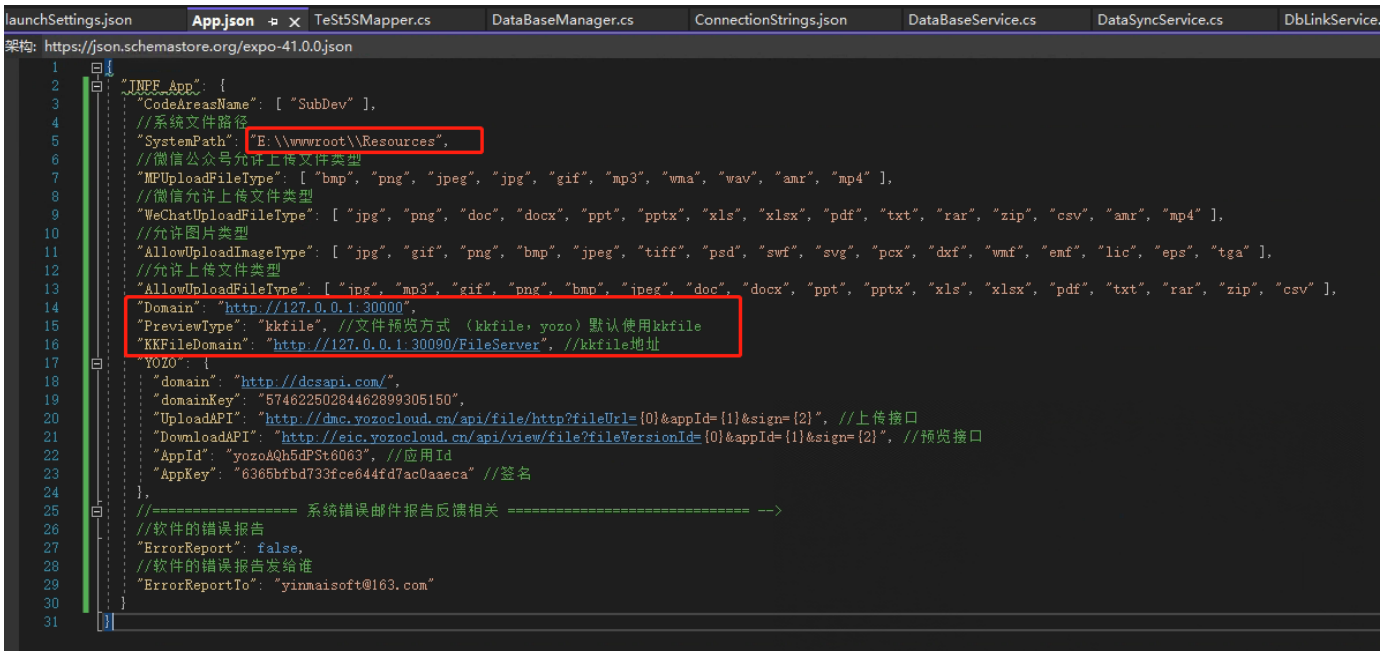


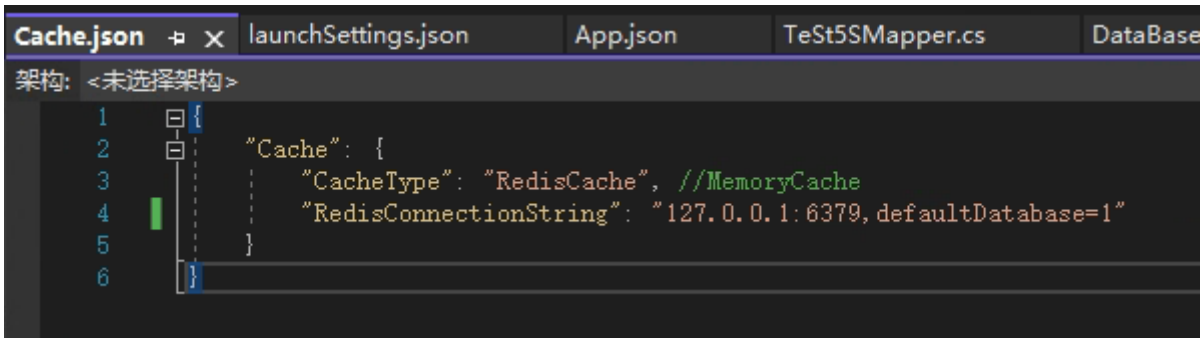
2.打开Configurations文件夹，修改底下的json文件相关配置



- 配置静态文件目录

> 注意Windows是以 \\ ,Linux或MacOS是以 / 来分隔路径的，结尾不要以 \\ 或者 / 结束



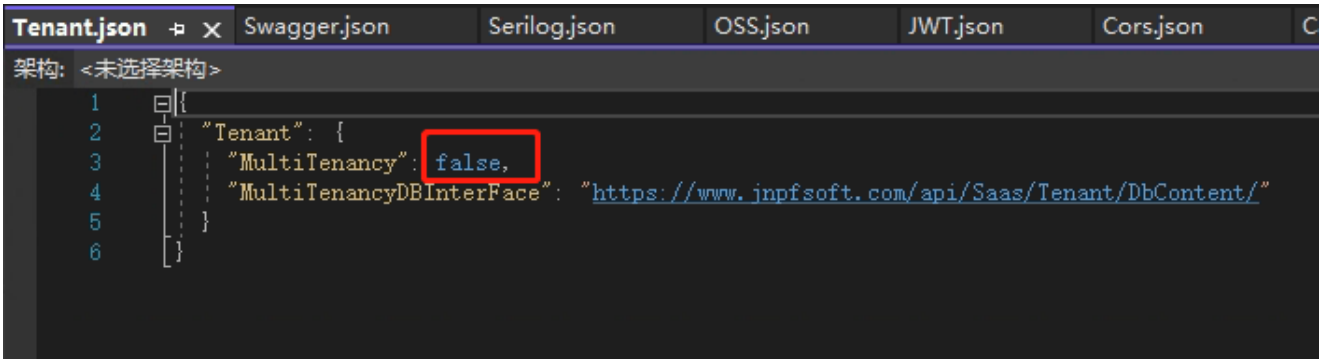


- 配置数据库

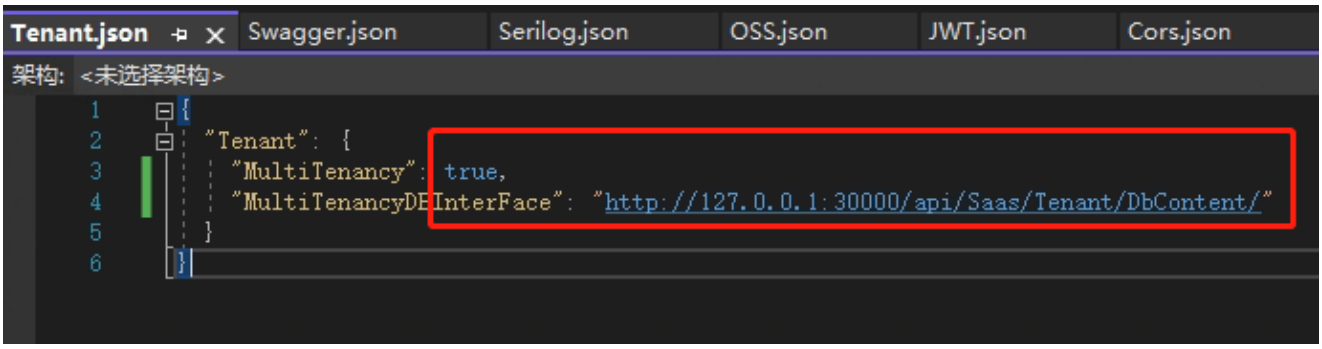
```
{
"ConnectionStrings": {
  "ConfigId": "default",
  //"DBName": "jnpf_init",
  "DBName": "netcore_test",
  "DBType": "SqlServer", //MySql;SqlServer;Oracle;PostgreSQL;Dm;Kdbndp;Sqlite;
  "Host": "192.168.0.11",
  "Port": "1433",
  "UserName": "netcore_test",
  "Password": "EtMhNTLJsGTeZEaD",
  //SqlServer
  "DefaultConnection": "Data Source=192.168.0.11;Initial Catalog={0};User ID=net
core_test;Password=EtMhNTLJsGTeZEaD;MultipleActiveResultSets=true"
  //Kdbndp
  //"DefaultConnection": "Server=192.168.0.103;Port=54321;UID=YANYU;PWD=123456;d
atabase=YANSOURCE"
  //Dm
  //"DefaultConnection": "Server=192.168.0.50; User Id=JNPFTEST; PWD=I97eH!bRfy5
```

```
5qGzF;DATABASE=JNPFTEST"
  //Oracle
  //"DefaultConnection": "Data Source=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=
192.168.0.19)(PORT=1521))(CONNECT_DATA=(SERVER = DEDICATED)(SERVICE_NAME=JNPFCLO
UD)));User Id=JNPFCLOUD;Password=JNPFCLOUD"
  //PostgreSQL
  //"DefaultConnection": "PORT=5432;DATABASE=java_boot_dev_postgresql;HOST=192.1
68.0.103;PASSWORD=123456;USER ID=postgres"
  //MySql
  //"DefaultConnection": "server=192.168.0.10;Database=netcore_test;Uid=netcore_
test;Pwd=jhpGB3A88CF57fBC;AllowLoadLocalInfile=true"
}
}
```

如果是非多租户就改成false

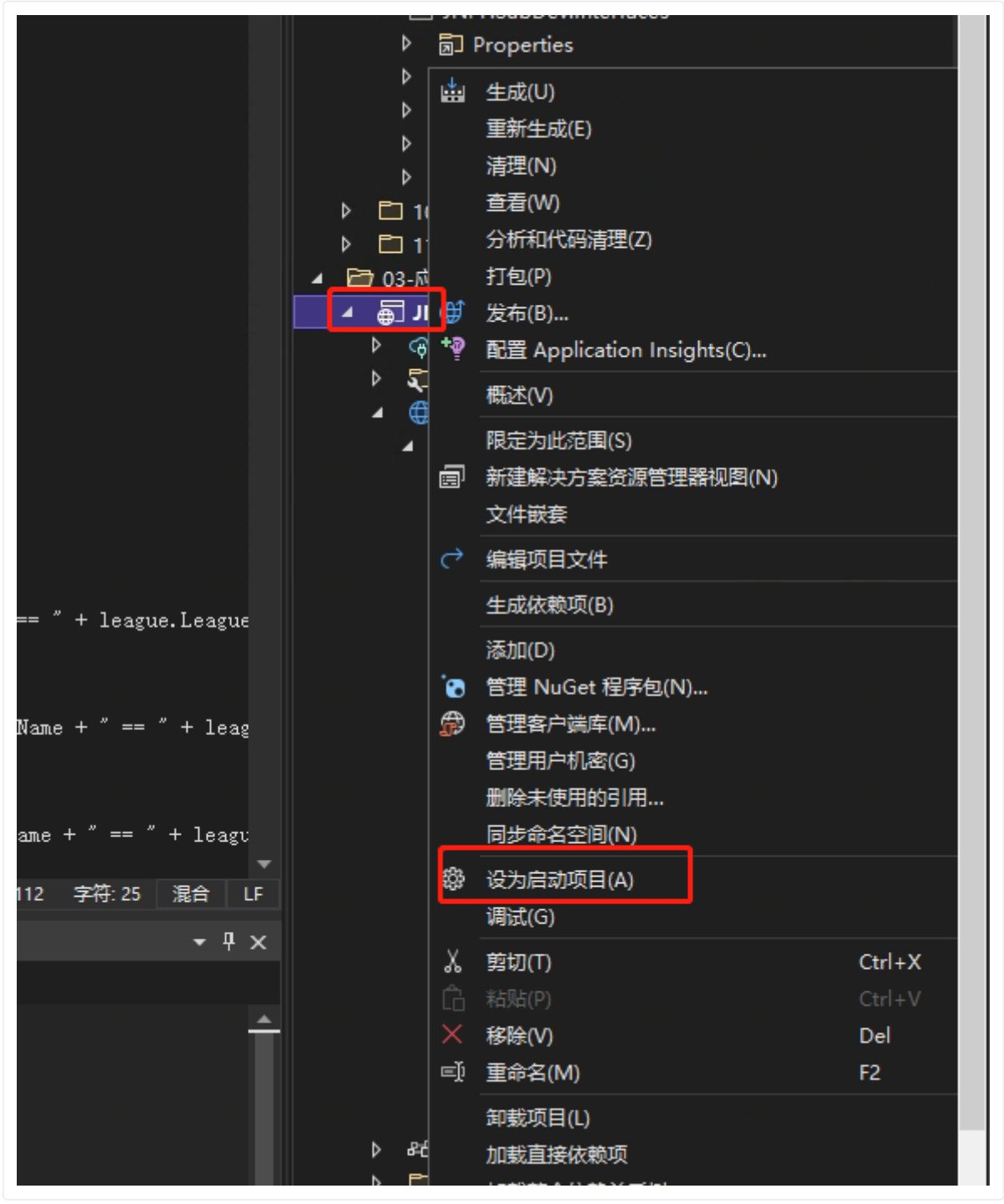


如果是多租户就改成true

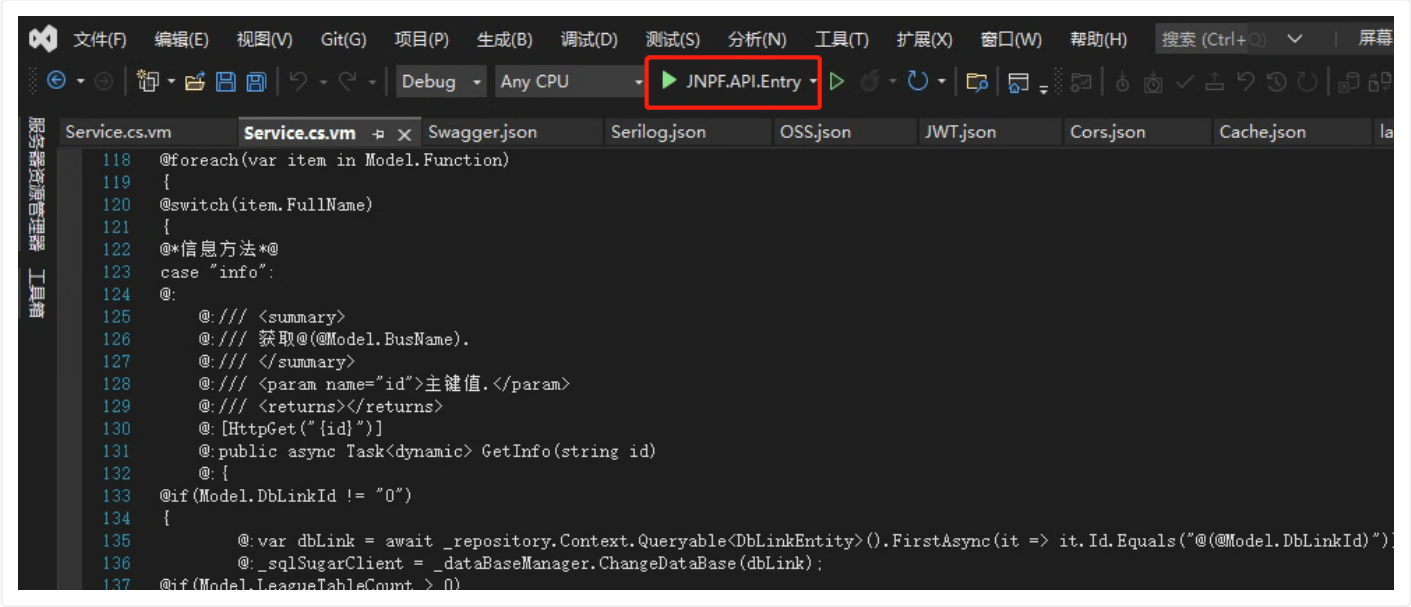


1. 设置默认启动项目

- 右击应用服务中的JNPF.Api.Entry,选择设为启动项目



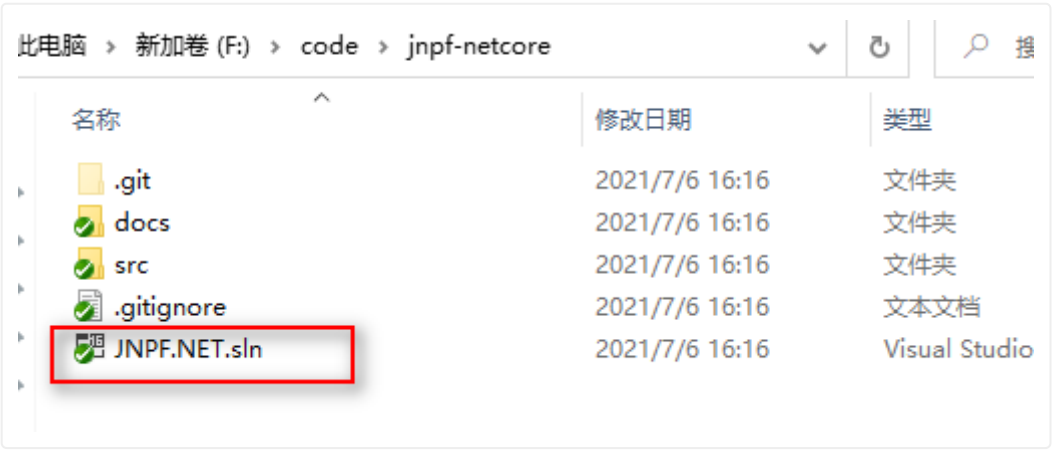
- 此时，在编辑器菜单栏下如下显示



本地项目运行（3.3.x及以下版本）

环境准备

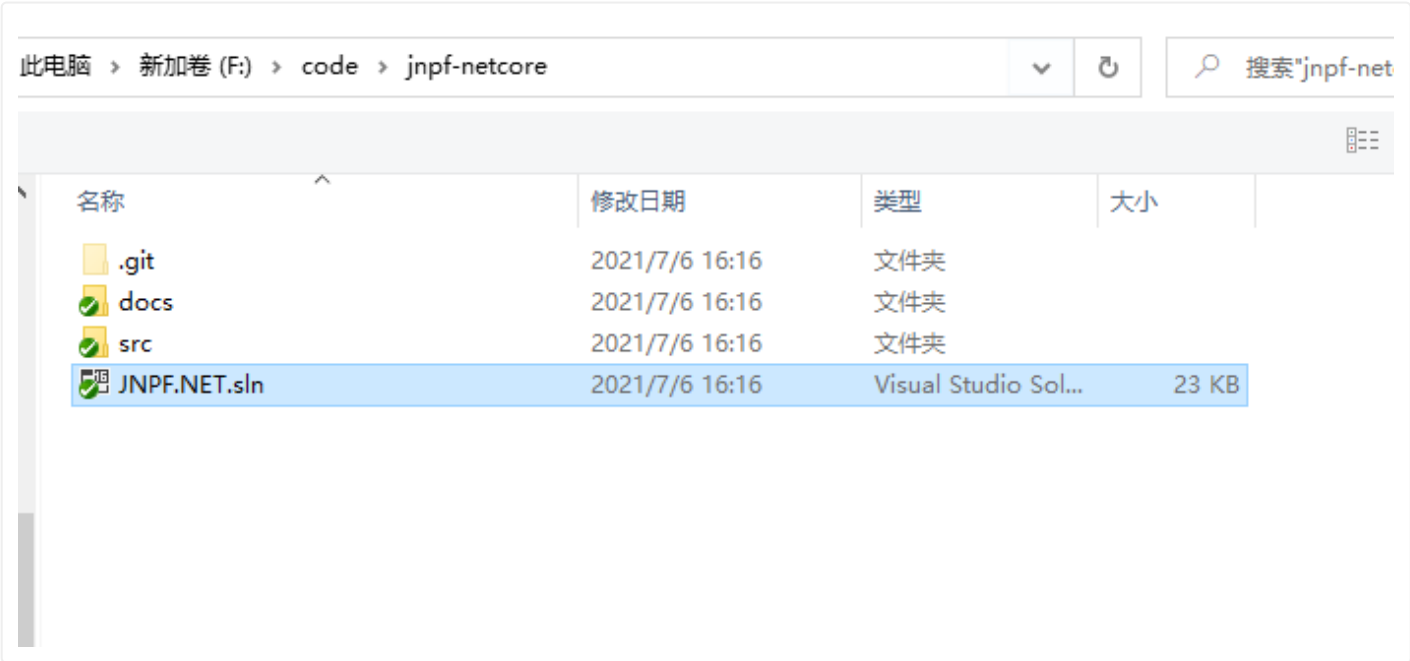
> 参考 .NETCORE -环境准备
>
> ## 项目导入
>
> 方式一：直接双击项目根目录下的 JNPF.NET.sln 文件



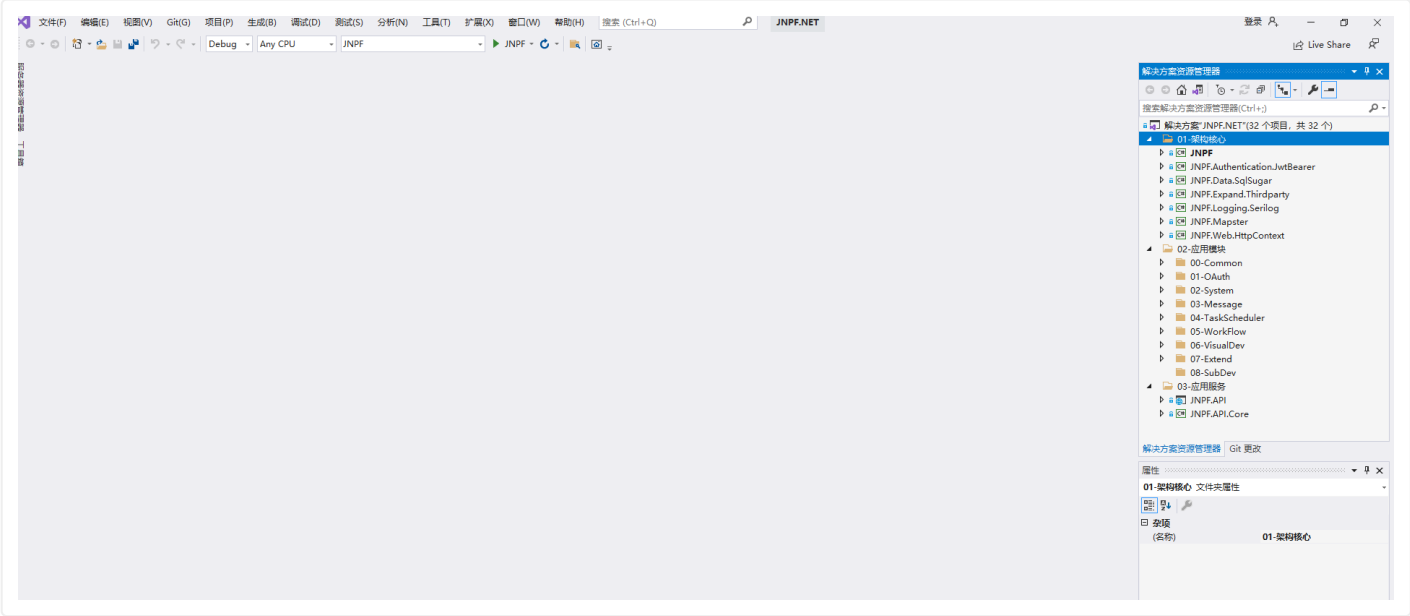
方式二：

- 打开 Visual Studio 2019 ,打开项目或解决方案，如下图

- 选择项目根目录下的 `JNPF.NET.sln` 文件



- 打开后，如下图界面



项目运行

1. 打开 `appsettings.json` 修改相关配置

```
"Cache": {
    "CacheType": "RedisCache", //MemoryCache
    "RedisConnectionString": "127.0.0.1:6379,defaultDatabase=1"
  },
```

- 配置系统文件根目录

> 注意结尾 \\(Windows)，/(Linux或MacOS)

```
"JNPF_App": {
    "SystemPath": "J:\\wwwroot\\Resources\\",
```

```
  ],
  "JNPF_App": {
    "SystemPath": "J:\\wwwroot\\Resources\\",
    //微信公众号允许上传文件类型
    "MPUploadFileType": "bmp,png,jpeg,jpg,gif,mp3,wma,wav,amr,mp4",
    //微信允许上传文件类型
    "WeChatUploadFileType": "jpg,png,doc,docx,ppt,pptx,xls,xlsx,pdf,txt,rar,zip,csv,amr,mp4",
    //允许图片类型
    "AllowUploadImageType": "jpg,gif,png,bmp,jpeg,tiff,psd,swf,svg,pcx,dxf,wmf,emf,lic,eps,tga",
    //允许上传文件类型
    "AllowUploadFileType": "jpg,gif,png,bmp,jpeg,doc,docx,ppt,pptx,xls,xlsx,pdf,txt,rar,zip,csv"
  }
}
```

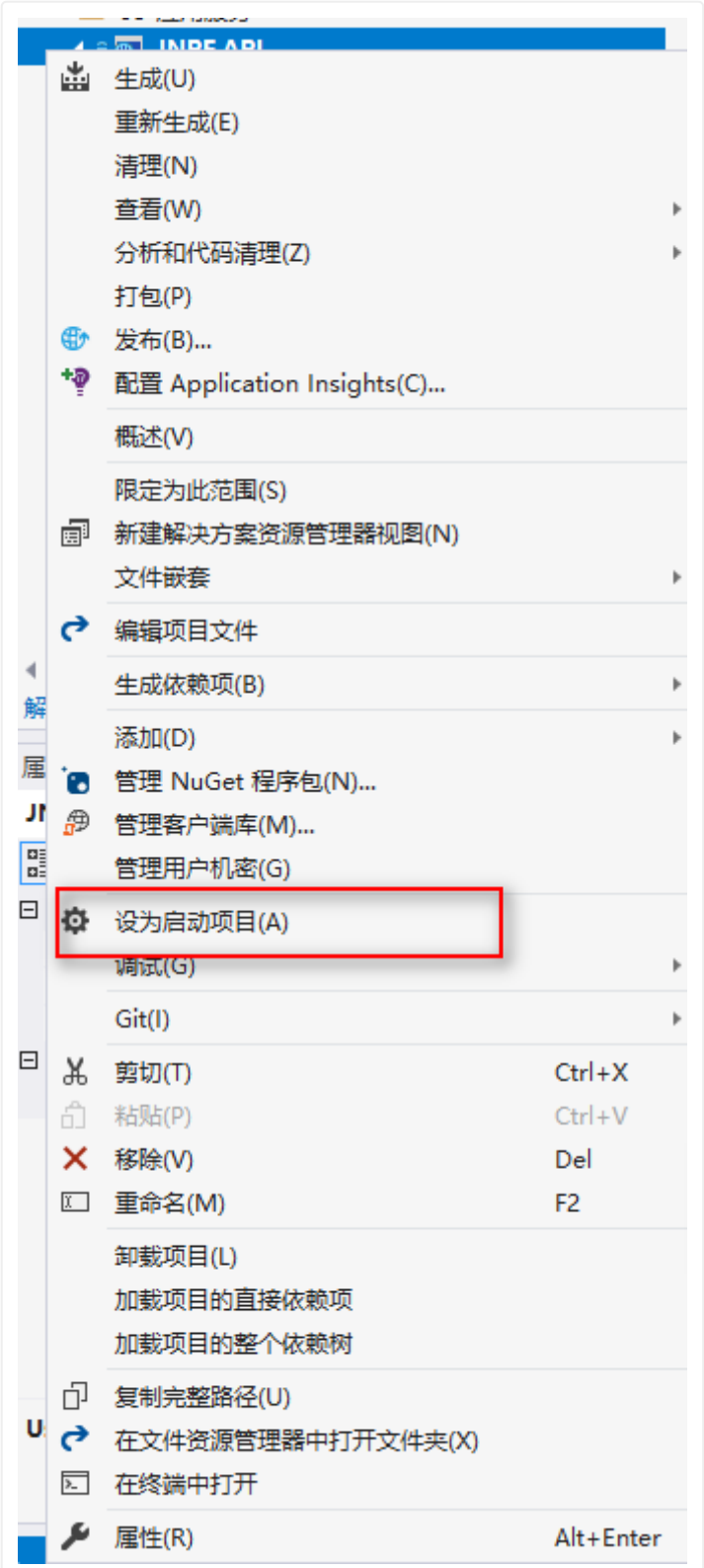
- 配置数据库

```
"AllowedHosts": "*",
  "ConnectionStrings": {
    "DBName": "netcore_test",
    "DefaultConnection": "Data Source=127.0.0.1;Initial Catalog={0};User ID=netcore_test;Password=EtMhNTLJsGTeZEaD"
    //"DefaultConnection": "Data Source=127.0.0.1;Initial Catalog={0};User ID=netcore_dev;Password=tNZL2pSzE3hpDWZX"
  },
```

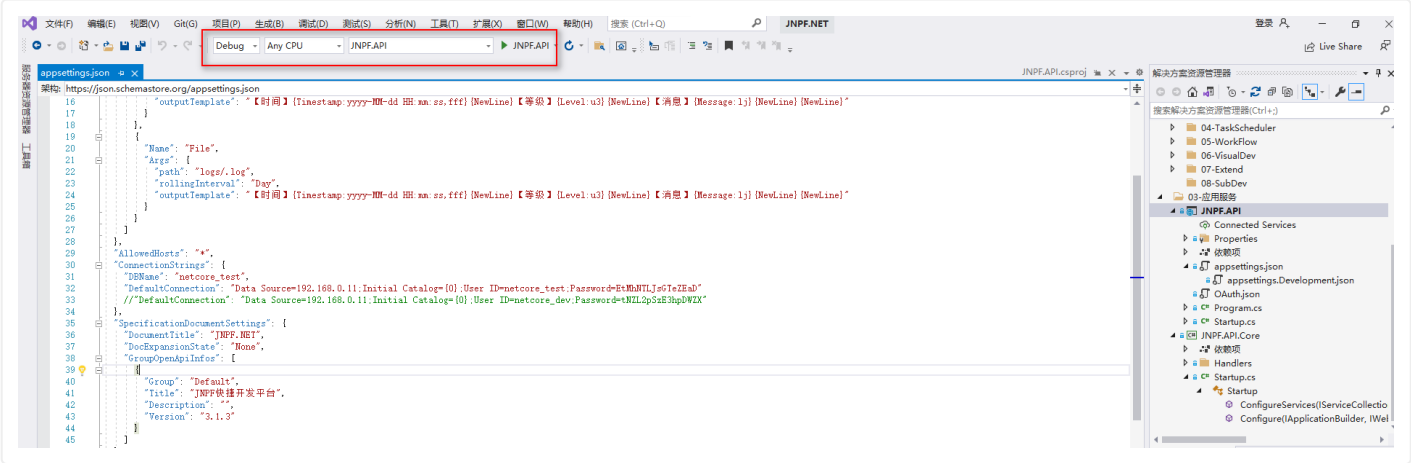
```
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DBName": "netcore_test",
    "DefaultConnection": "Data Source=192.168.0.11;Initial Catalog={0};User ID=netcore_test;Password=EtMhNTLJsGTeZEaD"
    //"DefaultConnection": "Data Source=192.168.0.11;Initial Catalog={0};User ID=netcore_dev;Password=tNZL2pSzE3hpDWZX"
  },
  "SpecificationDocumentSettings": {
    "DocumentTitle": "JNPF.NET",
    "DocExpansionState": "None",
    "GroupOpenApiInfos": [
      {
        "Group": "Default",
        "Title": "JNPF快捷开发平台",
        "Description": "",
        "Version": "3.1.3"
      }
    ]
```

1. 设置默认启动项目

- 右击应用服务中的JNPF.Api,选择设为启动项目



- 此时，在编辑器菜单栏下如下显示



服务器环境部署

Linux环境

本文档环境基于 CentOS-8.2.2004

硬件配置参考

本文档中的硬件参考配置为最低配置要求

项目	CPU	内存	硬盘	描述
单体服务	2C	4G	30G	
微服务	2C	32G	50G	一般一个服务内存占用1G-3G

1. 安装宝塔

- 官网：https://www.bt.cn/
- 安装方法：

```
yum install -y wget && wget -O install.sh http://download.bt.cn/install/install_6.0.sh && sh install.sh
```

按提示安装成功后，会显示管理地址、用户名及密码（请务必妥善保管）

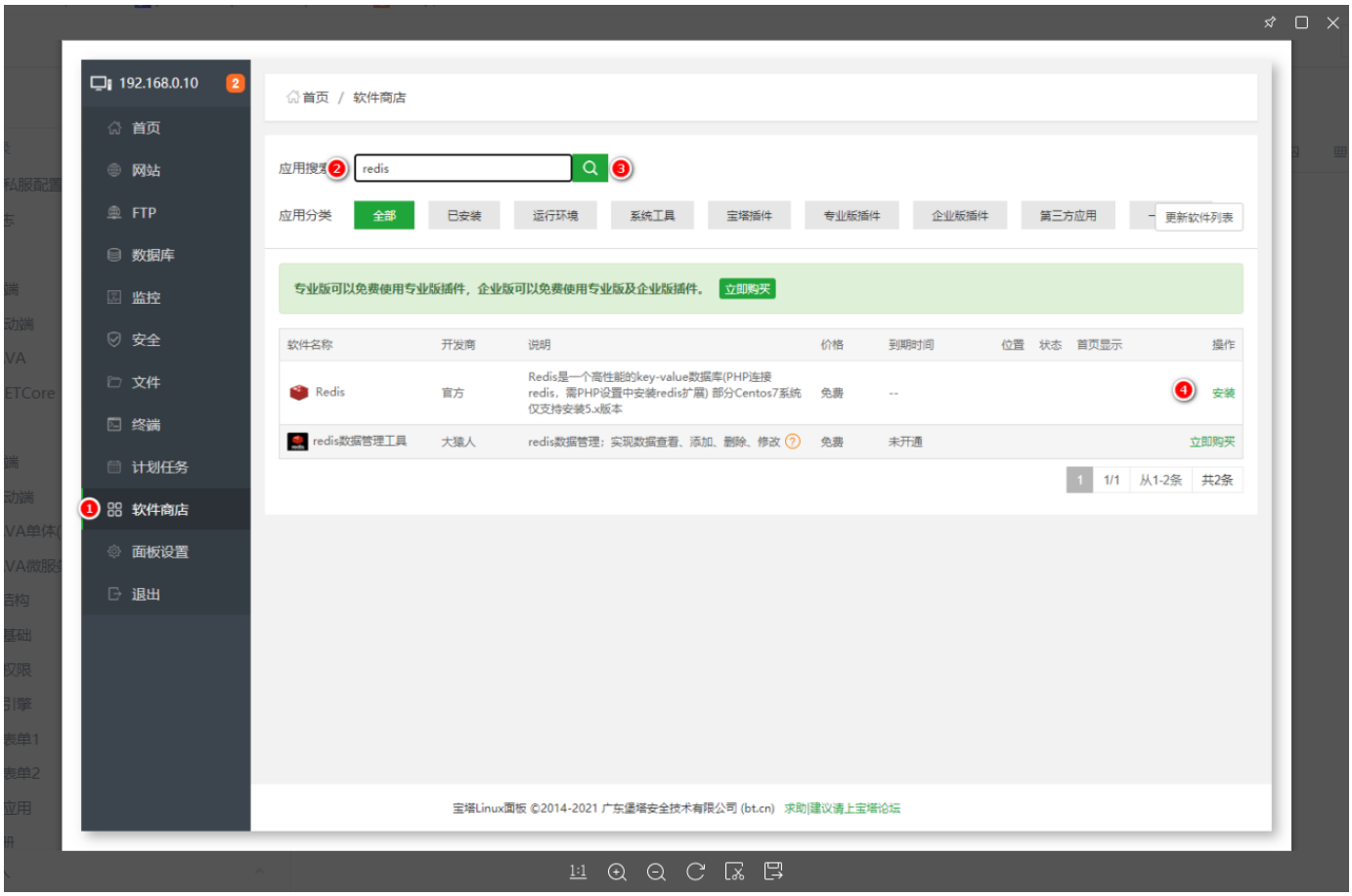
```
  Verifying  : firewalld-filesystem-0.6.3-8.el7.noarch

Updated:
  firewalld.noarch 0:0.6.3-13.el7_9

Dependency Updated:
  firewalld-filesystem.noarch 0:0.6.3-13.el7_9        python-firewall.noarch 0:0.6.3-13.el7_9

Complete!
success
==================================================================
Congratulations! Installed successfully!
==================================================================
■ ■ ■ ■ ■ ■ : http://125.78.229.116:8888/ae6d2f50
■ ■ ■ ■ ■ ■ : http://192.168.0.226:8888/ae6d2f50
username: sumuhhxv
password: 3ede7211
If you cannot access the panel,
release the following panel port [8888] in the security group
■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ /■ ■ ■ ■ ■ ■ ■ ■ ■ [8888]■ ■
==================================================================
Time consumed: 1 Minute!
[root@JNPFsoft ~]# _
```

(1)进入宝塔Linux面板

① 选择安装Nginx和MySQL(JAVA环境),如下图：

② 安装Redis

在软件商店搜索Redis并安装



③ 因宝塔软件商店没提供JDK，需手动安装JDK，具体操作如下

首先上传JDK安装包至服务器，本文档中是存放在/usr/local/jnpf/，然后执行以下操作（可以coding云盘下载）

```
# 1.进入安装包所在目录
cd /usr/local/jnpf

# 2.解压
tar -zxvf jdk-8u261-linux-x64.tar.gz

# 3.打开配置文件
vi /etc/profile

# 4.JAVA环境变量配置，在最后面输入以下内容
JAVA_HOME=/usr/local/jnpf/jdk1.8.0_261
JRE_HOME=/usr/local/jnpf/jdk1.8.0_261/jre
CLASSPATH=.:$JAVA_HOME/lib:/dt.jar:$JAVA_HOME/lib/tools.jar
PATH=$PATH:$JAVA_HOME/bin

# 5.使配置生效
source /etc/profile

# 6.查看安装结果
java -version
```

④ 如果在命令行中显示以下信息，说明已安装成功

```
java version "1.8.0_261"
Java(TM) SE Runtime Environment (build 1.8.0_261-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.261-b12, mixed mode)
```

2. 安装.NET SDK

（1）官网链接：https://docs.microsoft.com/zh-cn/dotnet/core/install/linux-centos
（2）在根目录下命令安装：

```
sudo rpm -Uvh https://packages.microsoft.com/config/centos/7/packages-microsoft-prod.rpm

sudo yum install dotnet-sdk-6.0

sudo yum install aspnetcore-runtime-6.0
```

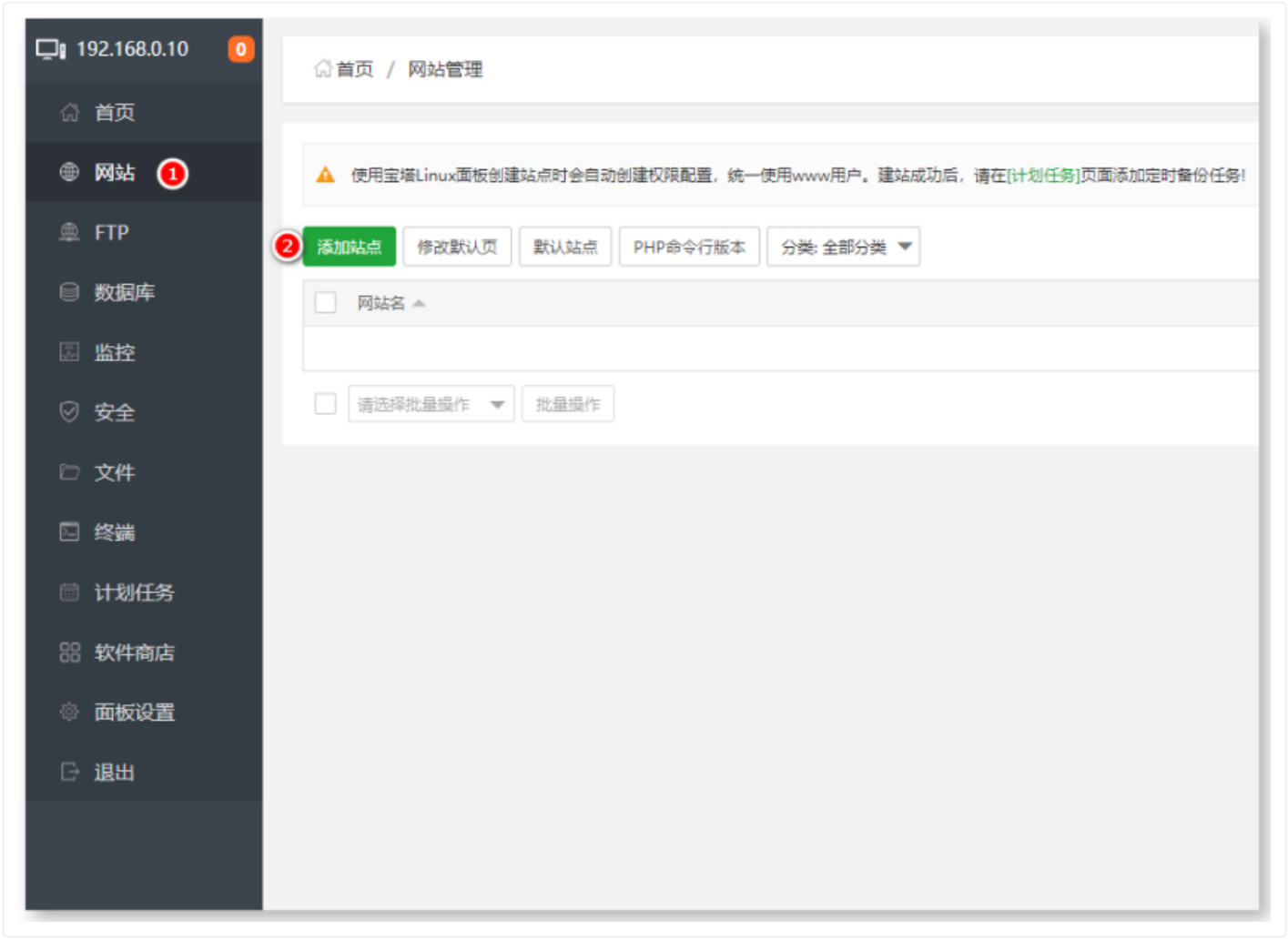
安装完成后，输入：

```
dotnet -version
```

查看版本号成功即安装成功。

3. 配置站点

(1)进入网站，添加站点



(2)站点配置如下

(3)添加成功后，我们先配置前端的配置文件

具体配置如下：

1.http网址的转发配置，如下：

```
server
{
  listen 80;
  server_name 192.168.0.10;
  index index.php index.html index.htm default.php default.htm default.html;
  root /www/wwwroot/jnpf-web;

  #JNPF-Start
  # 前端伪静态配置
  # 主项目前端
  location / {
    try_files $uri $uri/ /index.html;
  }
```

```
    # 大屏前端
    location /DataV {
      try_files $uri $uri/ /DataV/index.html;
    }
    #设置上传文件的大小
    client_max_body_size 100m;
    #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    client_header_buffer_size 128k;
    #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
    large_client_header_buffers 4 128k;
    #指定允许跨域的方法，*代表所有
    add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';
    # 预检命令的缓存，如果不缓存每次会发送两次请求
    add_header Access-Control-Max-Age 3600;
    #带cookie请求需要加上这个字段，并设置为true
    add_header Access-Control-Allow-Credentials true;
    #  表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #  $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #  表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;

    # 后端接口(按实际情况修改端口)
    location /api/ {
      proxy_pass http://localhost:30000/api/;
    }
    # webscket接口(按实际情况修改端口)
    location /websocket {
      proxy_pass http://localhost:30000/api/message/websocket;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
      proxy_read_timeout 600s;
    }
    # 报表设计接口配置(按实际情况修改端口)
    location /ReportServer/ {
      proxy_pass http://localhost:30007/;
    }
    # 文件预览服务
    location /FileServer {
```

```
    proxy_pass http://localhost:30090;
  }
  # 解决文件预览服务无法加载js,css问题
  location ~ /FileServer/*.*\.(js|css)?$ {
                proxy_pass http://localhost:30090;
  }
#JNPF-End
}
```

2.https网址的转发配置，如下：

```
server
{
    listen 80;
    listen 443 ssl http2;
    server_name 192.168.0.10;
    index index.php index.html index.htm default.php default.htm default.html;
    root /www/wwwroot/jnpf-web;

    #SSL-START SSL相关配置，请勿删除或修改下一行带注释的404规则
    #error_page 404/404.html;
    #HTTP_TO_HTTPS_START
    if ($server_port !~ 443){
        rewrite ^(/.*)$ https://$host$1 permanent;
    }
    #HTTP_TO_HTTPS_END
    ssl_certificate    /www/server/panel/vhost/cert/192.168.0.10/fullchain.pem;
    ssl_certificate_key    /www/server/panel/vhost/cert/192.168.0.10/privkey.pem;
    ssl_protocols TLSv1.1 TLSv1.2 TLSv1.3;
    ssl_ciphers EECDH+CHACHA20:EECDH+CHACHA20-draft:EECDH+AES128:RSA+AES128:EECDH+AE
S256:RSA+AES256:EECDH+3DES:RSA+3DES:!MD5;
    ssl_prefer_server_ciphers on;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;
    add_header Strict-Transport-Security "max-age=31536000";
    error_page 497  https://$host$request_uri;


    #SSL-END

    #ERROR-PAGE-START  错误页配置，可以注释、删除或修改
    #error_page 404 /404.html;
    #error_page 502 /502.html;
    #ERROR-PAGE-END
```

```
    #PHP-INFO-START  PHP引用配置，可以注释或修改
    include enable-php-00.conf;
    #PHP-INFO-END

    #REWRITE-START URL重写规则引用,修改后将导致面板设置的伪静态规则失效
    include /www/server/panel/vhost/rewrite/192.168.0.10;
    #REWRITE-END

    #禁止访问的文件或目录
    location ~ ^/(\.user.ini|\.htaccess|\.git|\.svn|\.project|LICENSE|README.md)
    {
        return 404;
    }

    #一键申请SSL证书验证目录相关设置
    location ~ \.well-known{
        allow all;
    }

    # location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
    # {
    #     expires      30d;
    #     error_log /dev/null;
    #     access_log /dev/null;
    # }

    # JNPF-START
    #设置上传文件的大小
    # client_max_body_size 100m;

    # #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    # #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    # client_header_buffer_size 128k;

    # #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
    # large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    # 预检命令的缓存，如果不缓存每次会发送两次请求
    #带cookie请求需要加上这个字段，并设置为true
```

```
    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号     #
表示请求头的字段 动态获取
    # 前端主项目(jnpf-web)伪静态
    location / {
      try_files $uri $uri/ /index.html;
    }

    # 前端大屏(jnpf-web-datascreen)伪静态
    location /DataV {
      try_files $uri $uri/ /DataV/index.html;
    }

    # 主项目
    location /api/ {
      proxy_pass http://localhost:30000;
    }

    location /websocket {
      proxy_pass http://localhost:30000/api/message/websocket;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
      proxy_read_timeout 600s;
    }

    # 报表（jnpf-datareport）接口
    location /ReportServer/ {
     proxy_pass http://localhost:30007/;
    }

    # 文件预览（jnpf-file-preview）
    location /FileServer {
      proxy_pass http://localhost:30090;
    }

    location ~ /FileServer/*.*\.(js|css)?$ {
                        proxy_pass http://localhost:30090;
    }
    # JNPF-End

    location ~ .*\.(js|css)?$
    {
        expires      12h;
        error_log /dev/null;
```

```
        access_log /dev/null;
    }
    access_log  /www/wwwlogs/netcore.log;
    error_log  /www/wwwlogs/netcore.error.log;
}
```

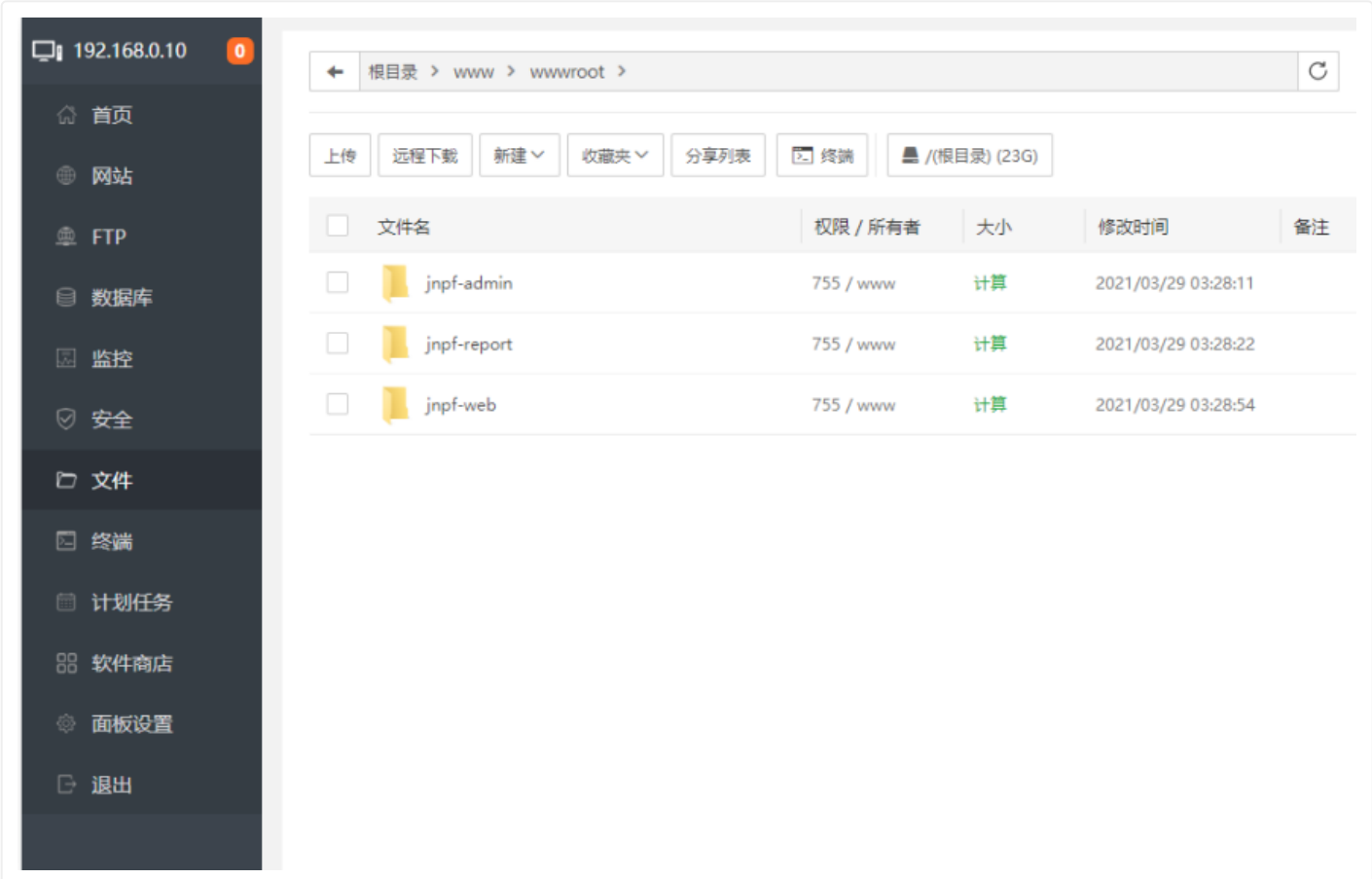
3.APP的接口转发配置，如下：

```
    #JNPF-Start

    #设置上传文件的大小
    client_max_body_size 100m;

    #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    client_header_buffer_size 128k;

    #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
    large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';

    # 预检命令的缓存，如果不缓存每次会发送两次请求
    add_header Access-Control-Max-Age 3600;

    #带cookie请求需要加上这个字段，并设置为true
    add_header Access-Control-Allow-Credentials true;

    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #    表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;


    # 伪静态
    location / {
      try_files $uri $uri/ /index.html;
    }
```

```
# 后端接口
location /api/ {
  proxy_pass http://localhost:30000;
}

# websocket
location /websocket {
  proxy_pass http://localhost:30000/api/message/websocket;
  proxy_http_version 1.1;
  proxy_set_header Upgrade $http_upgrade;
  proxy_set_header Connection "upgrade";
  proxy_read_timeout 600s;
}
#JNPF-End
```

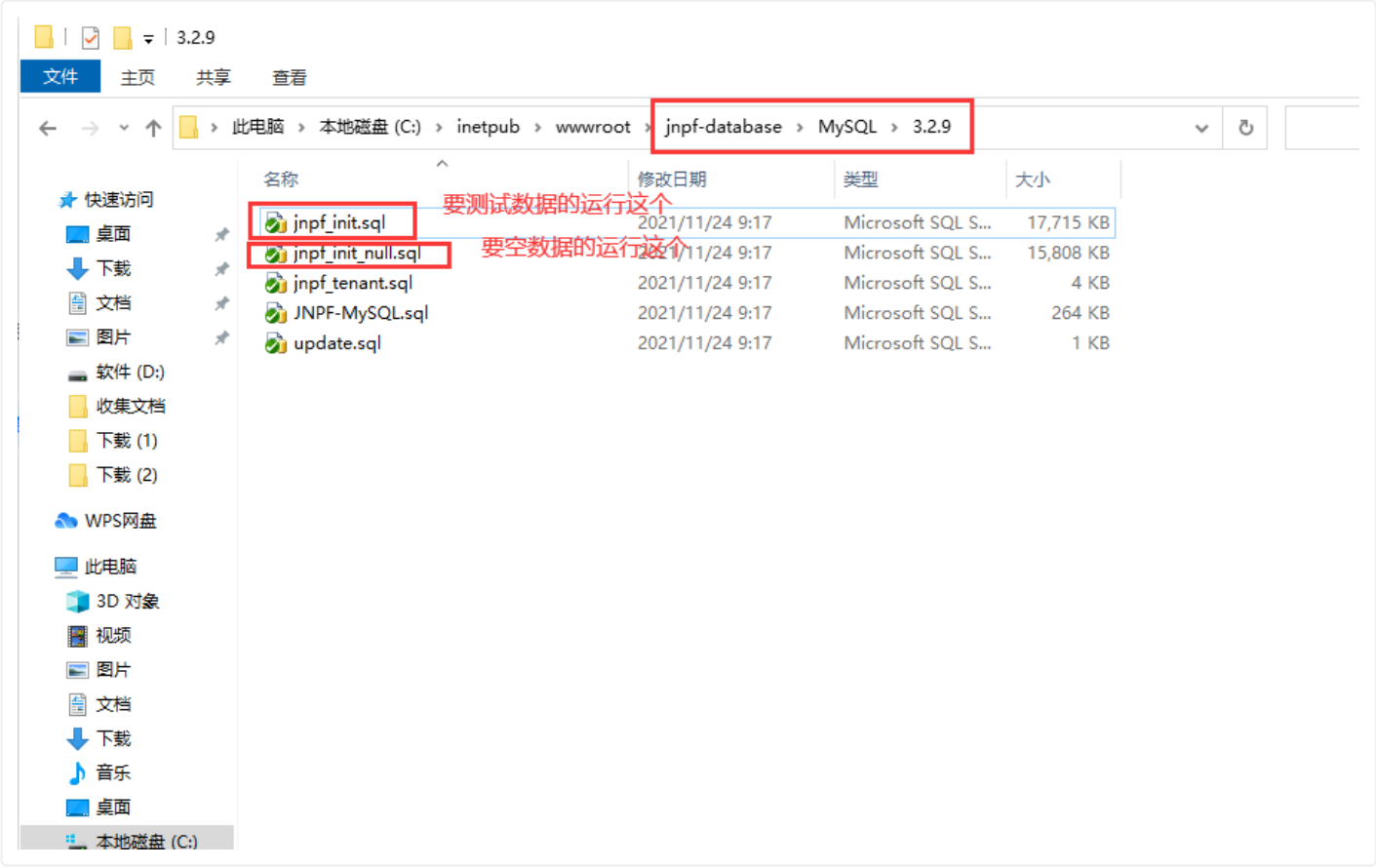
(4)目录建立

进入文件管理，按上文提供的目录约定创建对应目录



4.MySql创建数据库，创建用户，密码要记住

导入如下SQL文件



5.前端配置

主项目前端

配置
在项目根目录打开.env.staging(测试环境配置)，部署生产环境请打开.env.product文件

```
# 测试默认配置
ENV = 'staging'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.10'
# websocket接口
VUE_APP_BASE_WSS = 'ws://192.168.0.10/websocket'
```

```
# 生产环境默认配置
ENV = 'production'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.10'
# websocket接口
VUE_APP_BASE_WSS = 'ws://192.168.0.10/websocket'
```

构建

```
# 构建测试环境
npm run build:staging

# 构建生产环境
npm run build
```

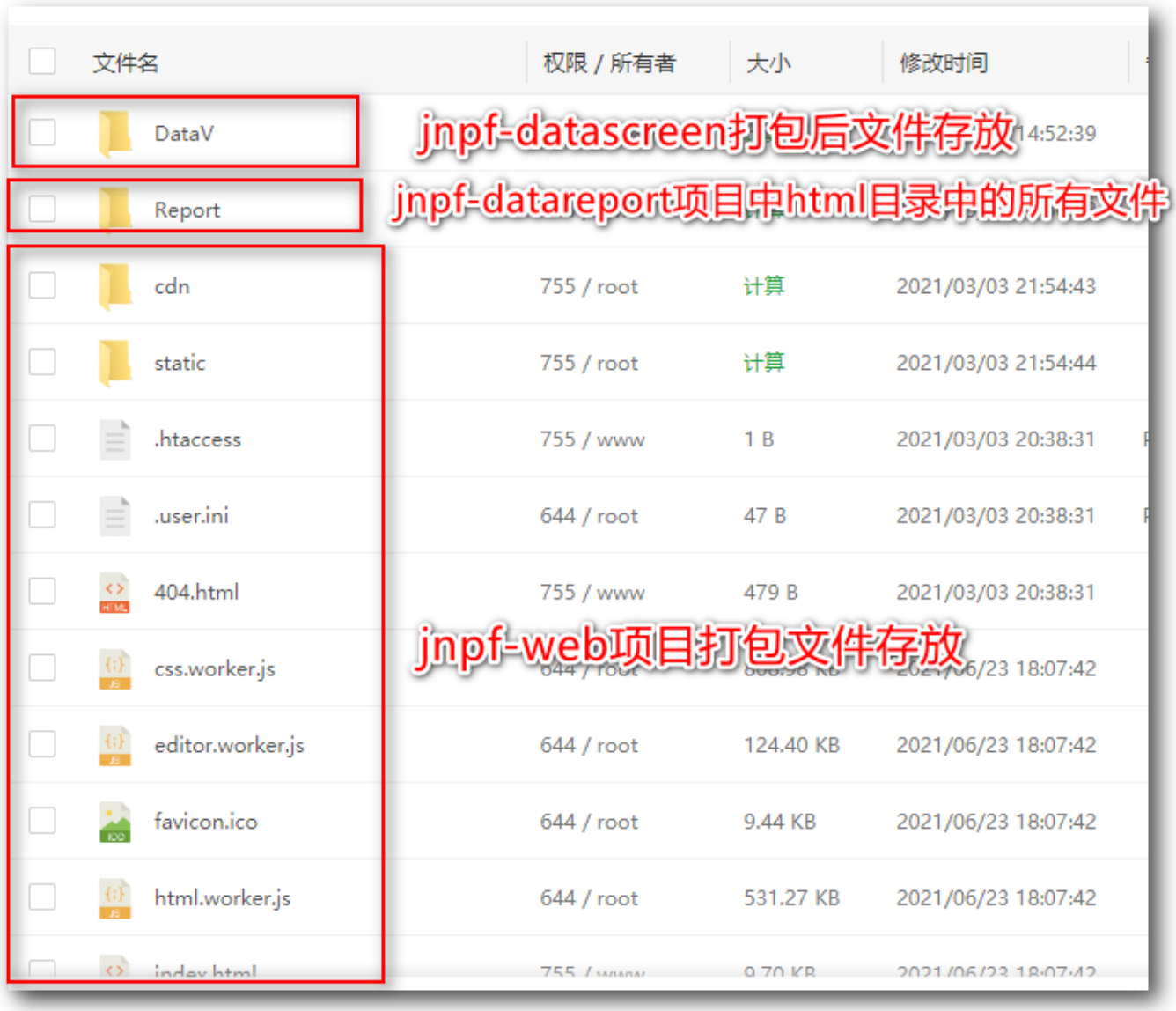
发布到服务器

- 方式1：压缩dist目录，上传到服务器/www/wwwroot/jnpf-web并解压
- 方式2：调整构建指令如下

```
npm run build:staging
scp -P 1802 -r  ./dist/* ssh root@192.168.0.10:/www/wwwroot/jnpf-web
```

6.报表前端部署

前端部署结构说明




```
├── jnpf-web                    # 假设这个目录是存放测试或生产环境的前端
│   ├── DataV                   # 大屏(`jnpf-datascreen`)打包后文件存放目录
│   ├── Report                  # 报表(`jnpf-datareport`)html下的文件
│   └── 主项目前端打包后的文件    # 主项目(`jnpf-web`)打包后存放在根目录
```

前端

- 将 `jnpf-web-datareport` 下的 `html` 文件夹中的所有拷贝到 `jnpf-web` 中的 `Report` 目录下, 如 `Report` 目录不存在请手动建立
- 接口配置
  - 打开 `/Report/index.html` ,做如下修改

```
# 在index.html文件中第24行开始
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.10/ReportServer";
// 报表前端
```

```
window._contextPath = "http://192.168.0.10/Report";
// 主项目接口地址
window._mainServer = "http://192.168.0.10";
</script>
```

- 打开 `/Report/preview.html` ,做如下修改,

```
# 在preview.html文件中第86行
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.10/ReportServer";
</script>
```

- 打开 `/Report/searchform.html` ,做如下修改,

```
# 在searchform.html文件中
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.10/ReportServer";
// 报表前端
window._contextPath = "http://192.168.0.10/Report";
</script>
```

- 配置说明:
  - 本示例中， `http://192.168.0.10` 为项目在测试环境中访问入口，在部署中根据实际情况调整;
  - 报表接口中， `ReportServer` 为虚拟目录，需要在Nginx增加相关配置，具体配置如下:

```
# 数据报表接口配置
  location /ReportServer/ {
     proxy_pass http://localhost:30007/;
  }
```

7.大屏前端部署

配置

在项目根目录打开 `.env.staging` (测试环境配置)，部署生产环境请打开 `.env.product` 文件

```
# 测试默认配置
ENV = 'staging'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.10'
```

```
# 生产环境默认配置
ENV = 'production'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.10'
```

构建

```
# 构建测试环境
yarn build:staging

# 构建生产环境
yarn build
```

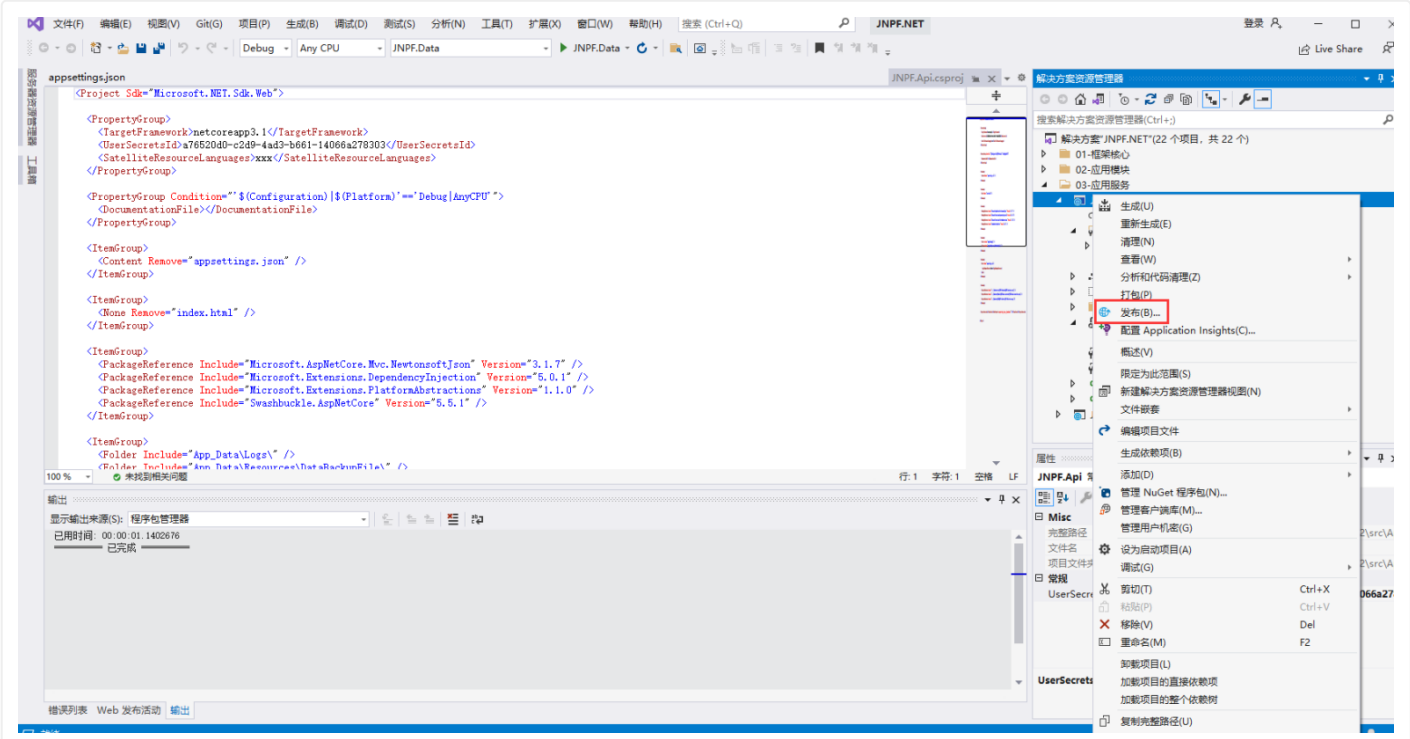
发布到服务器

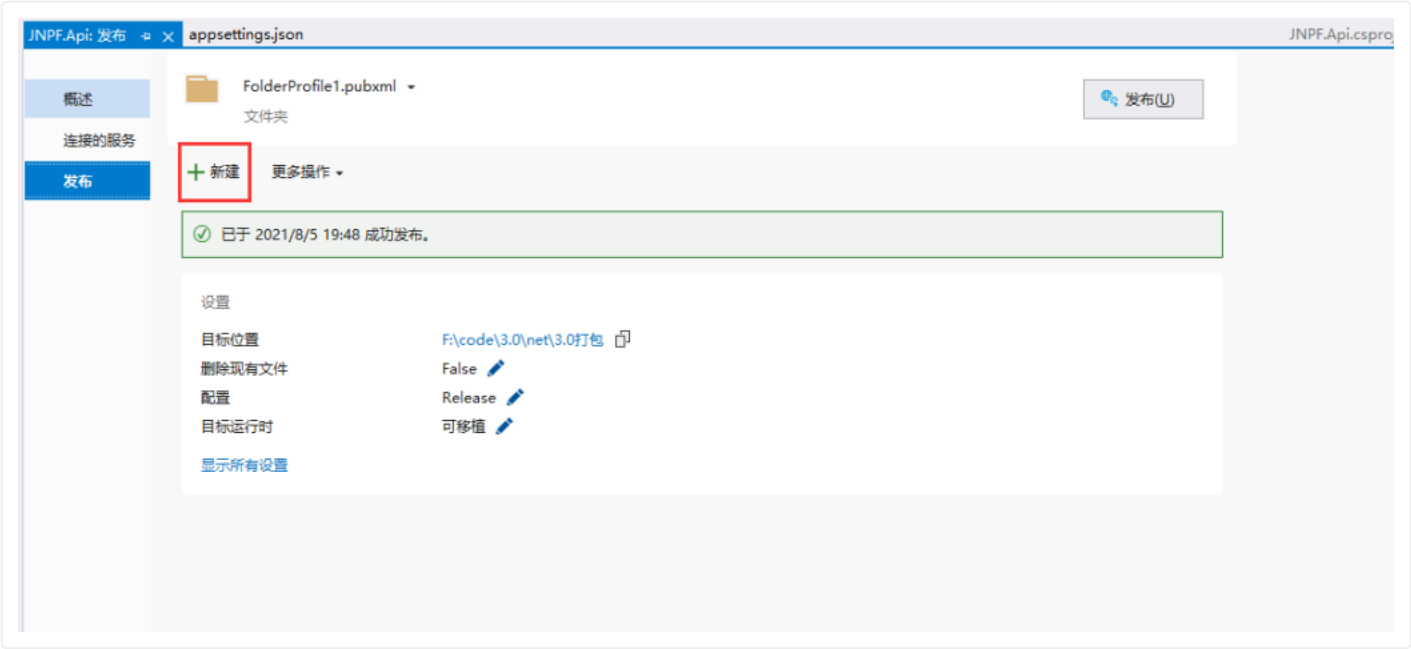
- 方式1：压缩dist目录，上传到服务器/www/wwwroot/jnpf-web/DataV并解压
- 方式2：调整构建指令如下

```
yarn build:staging
scp -P 1802 -r  ./dist/* ssh root@192.168.0.10:/www/wwwroot/jnpf-web/DataV
```

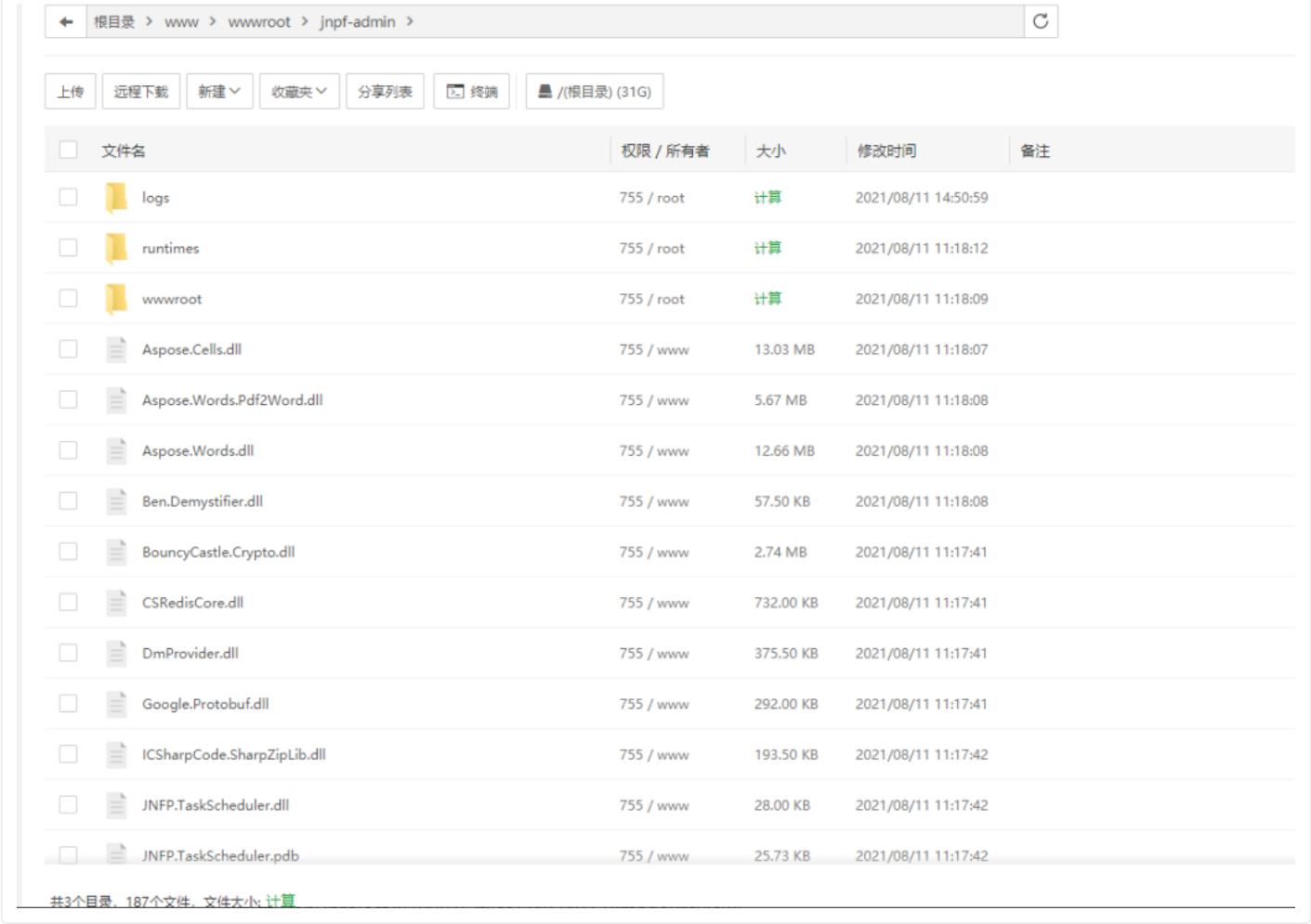
8.后端配置

Visual studio 2019打开后端项目，在应用服务-JNPF-API 右键选择“发布”





发布文件拷贝指定目录（新建文件夹），修改appsetting.josn 配置：



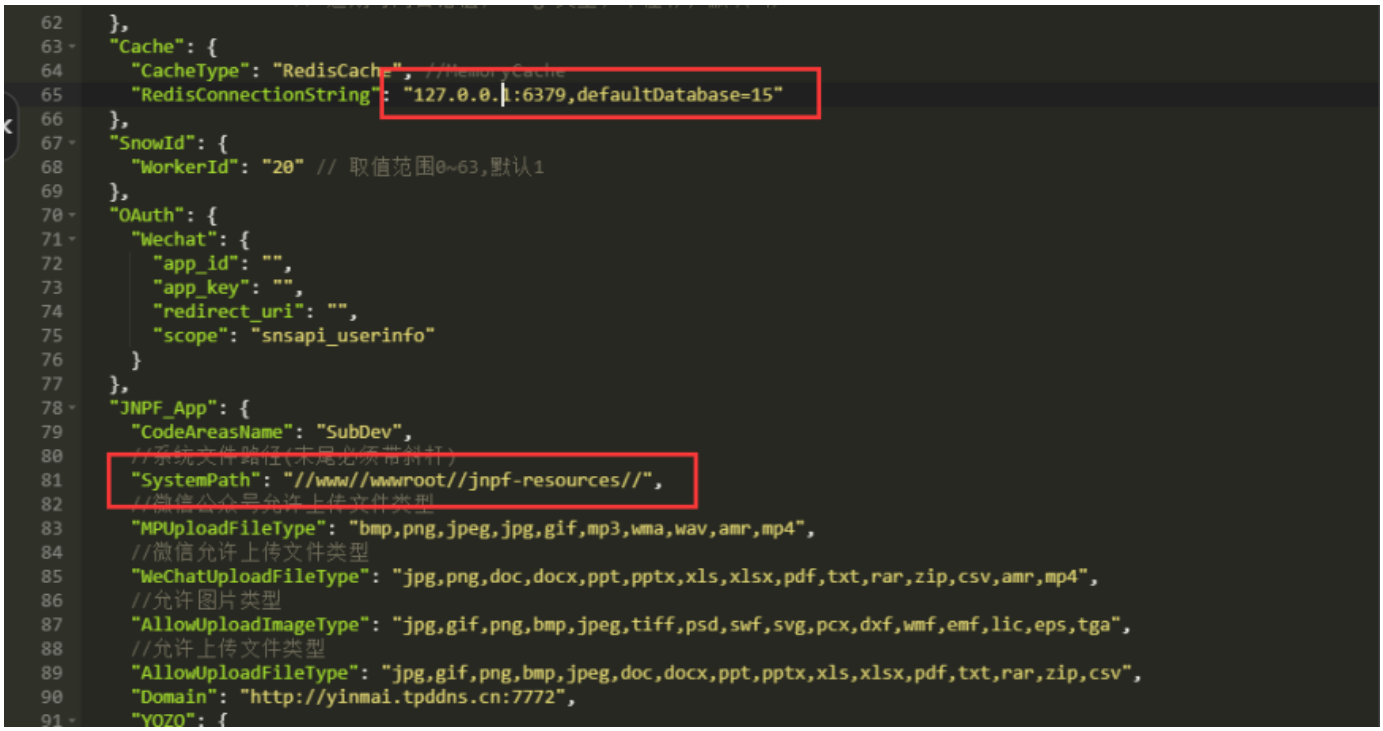
连接SQLServer：

```
  "ConnectionStrings": {
    "ConfigId": "default",
    "DBName": "jnpf_init",
    "DBType": "SqlServer", //MySql;SqlServer;Oracle;PostgreSQL;Dm;Kdbndp;Sqlite;
    "Host": "192.168.0.10",
    "Port": "1433",
    "UserName": "sa",
    "Password": "K3%*S4D!XDqE3k@",
    //SqlServer
    "DefaultConnection": "Data Source=192.168.0.10;Initial Catalog={0};User ID=sa;Password=K3%*S4D!XDqE3k@;MultipleActiveResultSets=true"
  },
```

```
  "ConnectionStrings": {
    "ConfigId": "default",
    "DBName": "jnpf_init",
    "DBType": "SqlServer", //MySql;SqlServer;Oracle;PostgreSQL;Dm;Kdbndp;Sqlite;
    "Host": "192.168.0.10",
    "Port": "1433",
    "UserName": "sa",
    "Password": "K3%*S4D!XDqE3k@",
    //SqlServer
    "DefaultConnection": "Data Source=192.168.0.10;Initial Catalog={0};User ID=sa;Password=K3%*S4D!XDqE3k@;MultipleActiveResultSets=true"
  },
```

连接MYSQL：

```
  "ConnectionStrings": {
    "ConfigId": "default",
    "DBName": "jnpf_init",
    "DBType": "MySql", //MySql;SqlServer;Oracle;PostgreSQL;Dm;Kdbndp;Sqlite;
    "Host": "192.168.0.10",
    "Port": "3306",
    "UserName": "jnpf_init",
    "Password": "K3%*S4D!XDqE3k@",
    //MySql
    "DefaultConnection": "server=192.168.0.10;Database={0};Uid=jnpf_init;Pwd=pBx5HaW6WMGSTdDf;AllowLoadLocalInfile=true"
  },
```



前台命令运行：

```
dotnet JNPF.API.dll --urls=http://*:30000
```

#端口根据nginx配置调整

后台运行：

```
nohup dotnet JNPF.API.dll --urls="http://*:30000" --environment=Delopment > /dev/null 2>&1 &
```

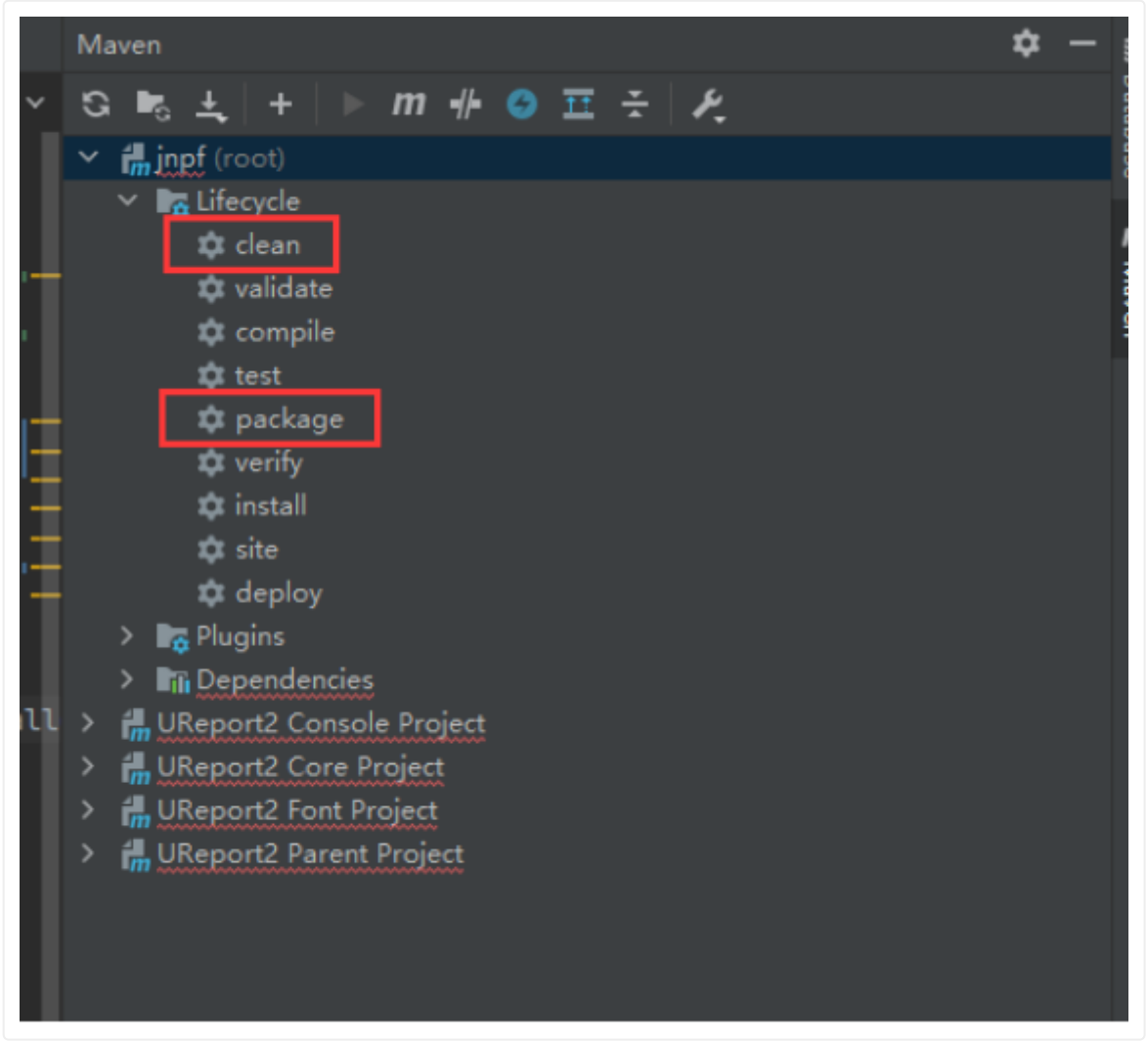
#端口根据nginx配置调整

停止运行：

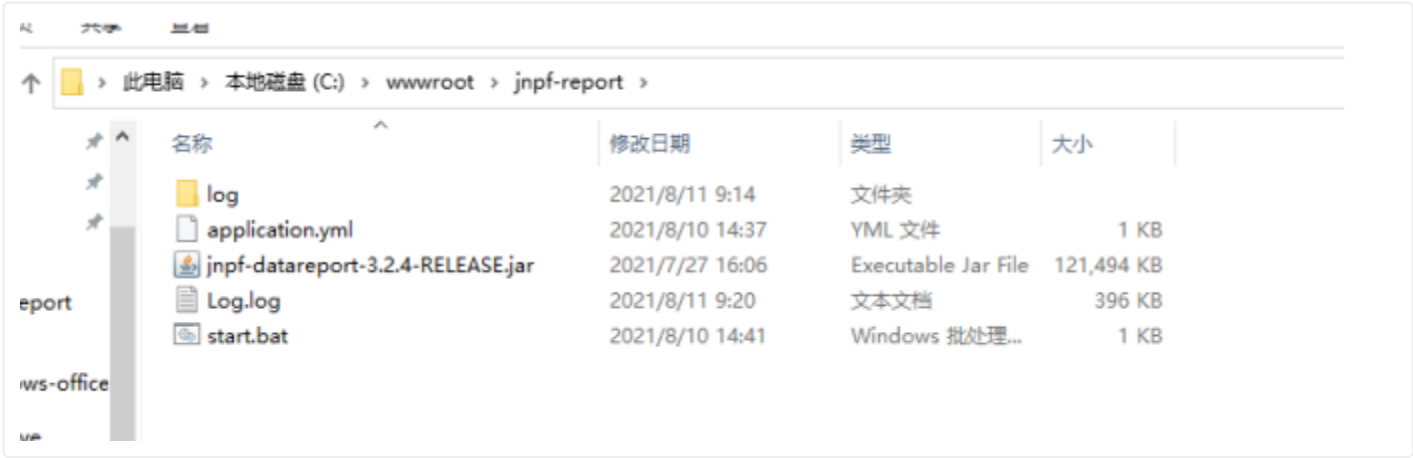
```
kill -9 $(netstat -nlp | grep 30000 | awk '{print $7}' | awk -F"/" '{ print $1 }')
```

9.报表后端项目打包

Intellij IDEA打开报表后端项目jnpf-datareport,打包



上传指服务器指定目录下（可以新建）



打开application.Yml文件，修改数据库配置并保存：

```yaml
# 配置端口
server:
  port: 30007
  max-http-header-size: 102400
  compression:
    enabled: true
    min-response-size: 102400
spring:
  # 表空间(Oracle)
  tableSpace: JNPFCLOUD
  datasource:
    # MySQL配置
    druid:
      dbinit: jnpf_init
      dbname: jnpf_init
      dbnull: jnpf_init
      url: jdbc:mysql://localhost:3306/{dbName}?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8
      username: root
      password: 123456
      driver-class-name: com.mysql.cj.jdbc.Driver

  #Redis配置
  redis:
    database: 1
    host: localhost
    port: 6379
    password:
    timeout: 3000
    lettuce:
      pool:
        max-active: 8 # 连接池最大连接数
        max-wait: -1ms  # 连接池最大阻塞等待时间（使用负值表示没有限制）
        min-idle: 0 # 连接池中的最小空闲连接
        max-idle: 8 # 连接池中的最大空闲连接
logging:
  level:
    root: info
    com.bstek.ureport.console: debug
  file:
    path: log
config:
```

MySQL配置：

```yaml
    druid:
      dbinit: jnpf_init
      dbname: jnpf_init
      dbnull: jnpf_init
      url: jdbc:mysql://localhost:3306/{dbName}?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8
      username: root
      password: 123456
      driver-class-name: com.mysql.cj.jdbc.Driver
```

SQLServer配置:

```yaml
    druid:
      dbinit: jnpf_init
      dbname: jnpf_init
      dbnull: jnpf_init
      url: jdbc:sqlserver://localhost:1433;DatabaseName={dbName}
      username: sa
      password: JNPF@2020
      driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
```

Oracle配置:

```
# 表空间(Oracle)
tableSpace: JNPFCLOUD
```

```
druid:
  dbinit: JNPFCLOUD
  dbname: JNPFCLOUD
  dbnull: JNPFCLOUD
  url: jdbc:oracle:thin:@192.168.0.10:1521:{dbName}
  username: JNPFCLOUD
  password: JNPFCLOUD
  driver-class-name: oracle.jdbc.driver.OracleDriver
```

运行.jar文件:

```
nohup java -jar -Xmx2000m -Xms2000m -Xmn1000m -Xss256k -XX:+HeapDumpOnOutOfMemoryErr
or jnpf-datareport-3.2.11-RELEASE.jar >Log.log 2>&1 &
```

注：新建文本，输入命令后，修改后缀为.sh，用sh命令运行。
查看日志运行成功即可使用报表模块。

10.文档预览部署

环境要求

- JDK1.8+
- OpenOffice或LiberOffice(Windows下已内置，CentOS或Ubuntu下会自动下载安装，MacOS下需要自行安装)

部署运行

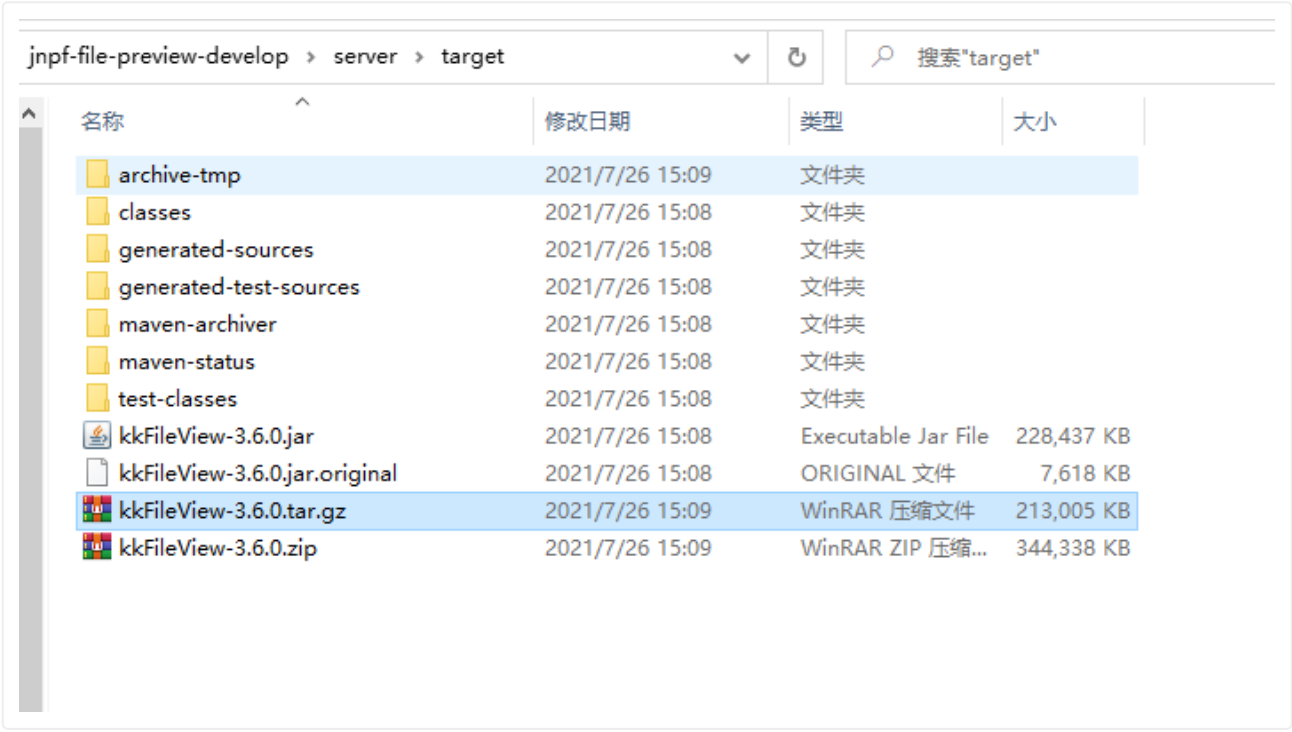
1、开发环境

- a.IDEA导入项目
- b.调整配置，打开 `server/src/main/config/application.properties`
- c.启动项目， `server/src/main/java/cn/keking/ServerMain`
- d.打开 `http://localhost:30090` 测试页面

2、测试生产环境

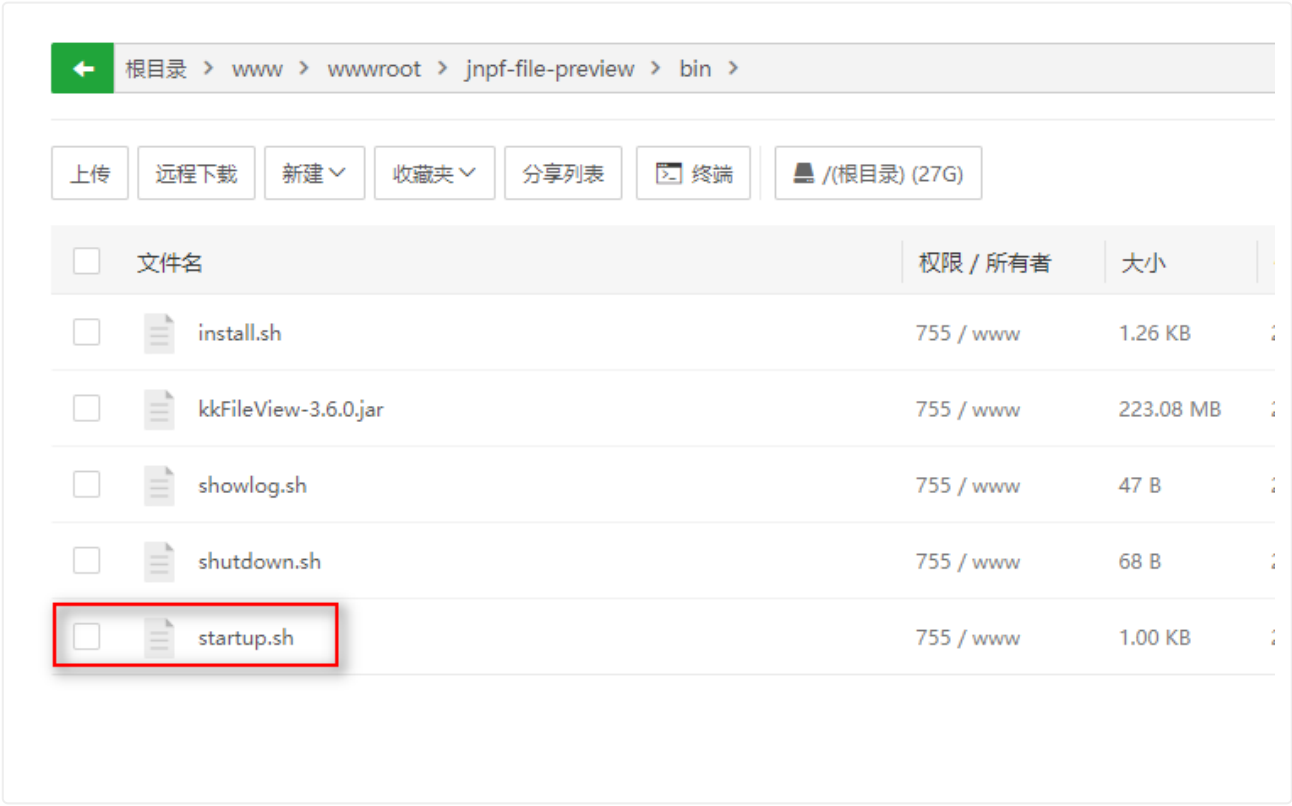
- a.打包

  - 打开 `\server\target` 主要有以下几个文件
  - kkFileView-xxx.jar(一般用于更新)
  - kkFileView-xxx.zip(windows环境下首次部署)
  - kkFileView-xxx.tar.gz(Linux环境下首次部署)



- b.上传至服务器

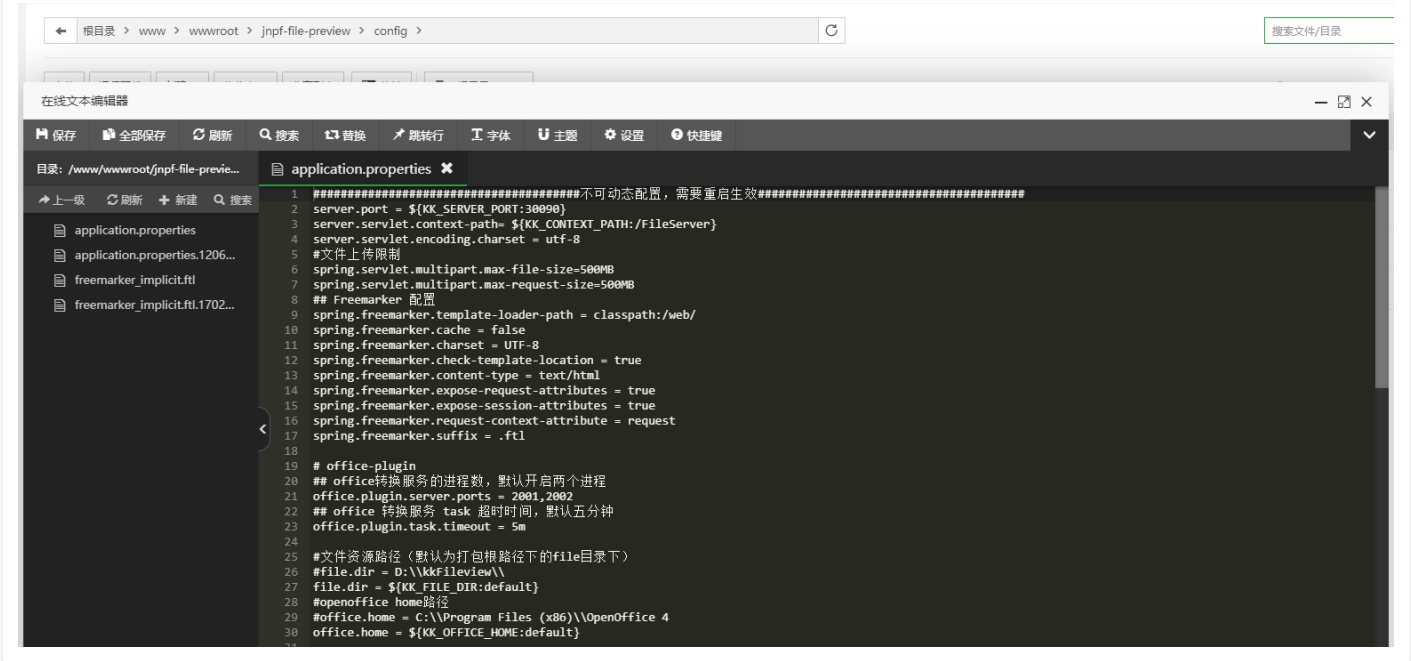
- 打开解压后文件夹的bin目录，运行startup脚本(首次部署，之后更新及维护都在bin目录下)



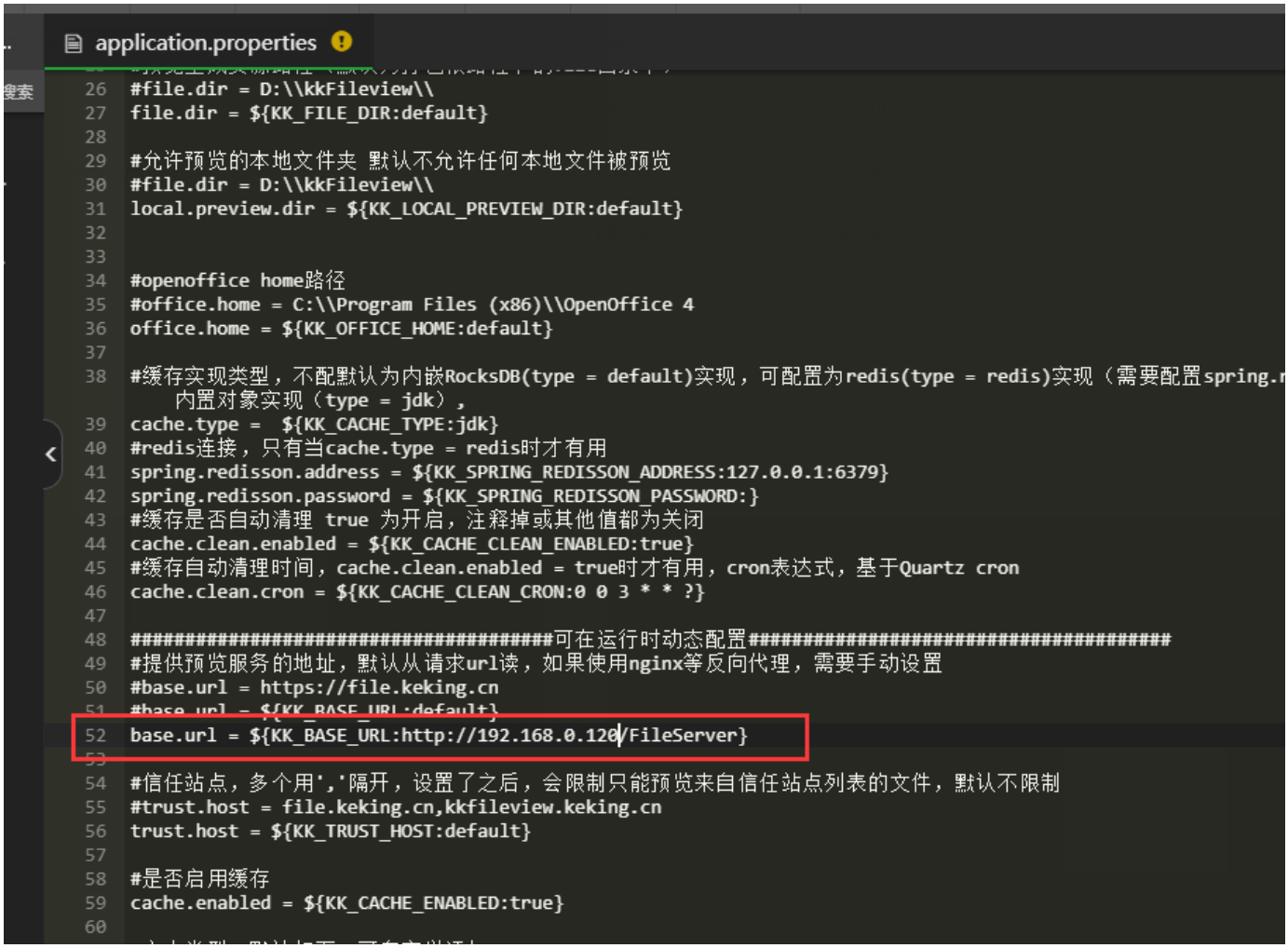
- 修改文件预览配置文件如下两项,打开服务器上的 `kkFileView-xxx/config/application.properties`（第3行和第45行）

修改成:

```
server.servlet.context-path= ${KK_CONTEXT_PATH:/FileServer}
base.url = ${KK_BASE_URL:http://192.168.0.10/FileServer}
```



新版只需修改第52行：



- c.Nginx配置说明
  例如nginx的访问地址为 `https://netcore.jnpfsoft.com` ,文件预览部署在内网192.168.0.10服务器上，需要在nginx中添加反向代理如下

```
# 文件预览服务
location /FileServer {
  proxy_pass 192.168.0.10:30090;
}

# 解决文件预览服务无法加载js,css问题
location ~ /FileServer/*.*\.(js|css)?$ {
  proxy_pass 192.168.0.10:30090;
}
```

使用指南

1、单文件预览

```
// 要预览文件的访问地址
let previewUrl = 'http://127.0.0.1:8080/file/test.txt'

// 使用
http://127.0.0.1:30090/onlinePreview?url='+encodeURIComponent(Base64.encode(previewUrl))
```

2、http/https下载流url预览

```
// 要预览文件的访问地址
let previewUrl = 'http://127.0.0.1:8080/filedownload?fileId=1'

// 使用
http://127.0.0.1:30090/onlinePreview?url='+encodeURIComponent(Base64.encode(previewUrl))
```

3、更多参考官方文档（`https://kkfileview.keking.cn/zh-cn/docs/usage.html`）

常见问题

1、预览乱码(字体问题)

大部分Linux系统上并没有预装中文字体或字体不全，需要把常用字体拷贝到Linux服务器上，具体操作如下： 下载如下字体包 `http://kkfileview.keking.cn/fonts.zip` 文件解压完整拷贝到Linux下的 `/usr/share/fonts` 目录。然后依次执行 `mkfontscale` 、 `mkfontdir` 、 `fc-cache` 使字体生效

> 特别说明：安装字体前确保Linux服务器已安装mkfontscale、fontconfig，如未安装运行以下命令

CentOS运行如下命令

```
# 使mkfontscale和mkfontdir命令正常运行
yum install mkfontscale

# 使fc-cache命令正常运行
yum install fontconfig
```

Ubuntu运行如下命令

```
# 使mkfontscale和mkfontdir命令正常运行
sudo apt-get install ttf-mscorefonts-installer

# 使fc-cache命令正常运行
sudo apt-get install fontconfig
```

2、更多问题请参考官方文档说明(`https://kkfileview.keking.cn/zh-cn/docs/faq.html`)

11.常见问题

（1）mysql数据库执行查询命令：（解决只有三个菜单问题）

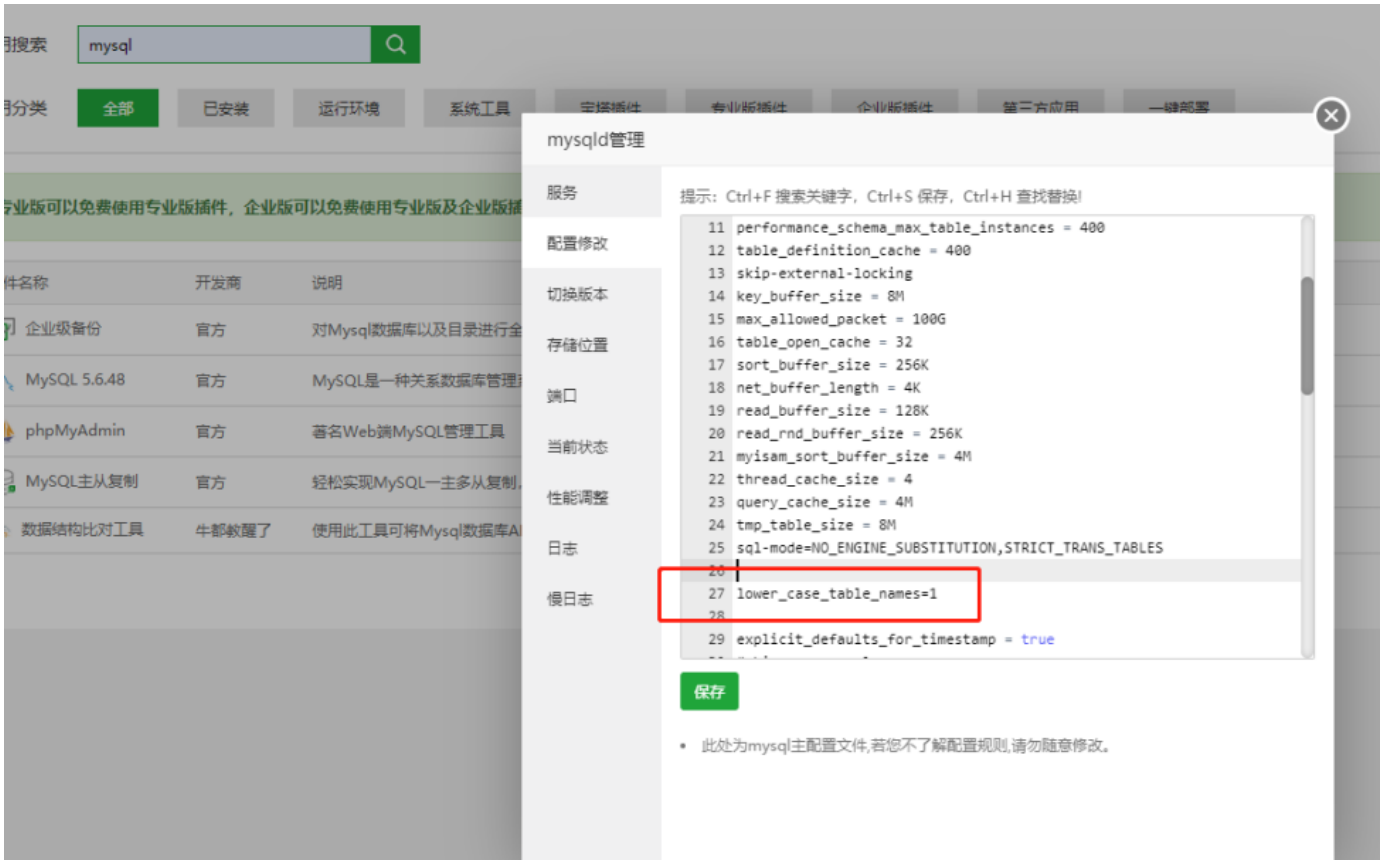
```
update base_module set F_DeleteMark=null
```

（2）数据库添加my.cnf：

```
lower_case_table_names = 1
```

解决mysql大小写转换问题。

```
【时间】2021-08-11 13:41:49,661
【等级】ERR
【消息】SqlSugar.SqlSugarException: English Message : Connection open error . Unable to connect to any of the specified MySQL hosts.
Chinese Message :  连接数据库过程中发生错误，检查服务器是否正常连接字符串是否正确，实在找不到原因请先Google错误信息：Unable to connect to any of the
    specified MySQL hosts..
   at SqlSugar.AdoProvider.GetDataReaderAsync(String sql, SugarParameter[] parameters)
   at SqlSugar.QueryableProvider`1.GetDataAsync[TResult](KeyValuePair`2 sqlObj)
   at SqlSugar.QueryableProvider`1._ToListAsync[TResult]()
   at SqlSugar.QueryableProvider`1.FirstAsync()
   at SqlSugar.QueryableProvider`1.FirstAsync(Expression`1 expression)
   at SqlSugar.SqlSugarRepository`1.FirstOrDefaultAsync(Expression`1 whereExpression) in F:\code\3.2net\jnpf-netcore-master\src\Infrastructure\JNPF.Data
       .SqlSugar\Repositories\SqlSugarRepository.cs:line 226
   at JNPF.System.Service.Permission.UsersService.GetInfoByAccount(String account) in F:\code\3.2net\jnpf-netcore-master\src\Modularity\System\JNPF
       .System\Service\Permission\UsersService.cs:line 544
   at JNPF.OAuth.Service.OAuthService.Login(LoginInput input) in F:\code\3.2net\jnpf-netcore-master\src\Modularity\OAuth\JNPF.OAuth\Service\OAuthService
       .cs:line 123
```



解决出现死循环问题：base_user：F_ManagerId清空。

```
09-aa7f-c0a6890e4546','f8c18307-065c-4dd3-a6c6-a2b95ba17ed6','f96d5799-c894-4f00-81dc-4caa97e294e4',
'fa0719d2-4a8b-4bac-ba2b-b65fbc286c1b','3bbad457-663e-4314-9b10-33c67c35cbcb','048657c1-ecb1-4e5d-ac
ed-6126ceae779e','0a85de84-4de0-4317-9767-25f95fdd9b1e','0c1d9463-221f-49ba-8e38-28c1f1df0c4b','146f
6c3a-9eaa-4295-817f-8bf4e19b1a20','1adea3ff-8a25-420b-a628-c5cb78fd51c0','1d91dd85-989e-4252-8d7f-e4
2e04bc6b3d','22736bf4-f2d5-4081-9fb4-ca43d6235f56','25e10fa4-7c68-4bc4-b751-92fa54f24c5b','26fe6815-
20a2-4be7-93b4-25e195d22833','27c57c5a-4c50-4572-954b-f9dafbbbca89','2ab1ce27-0903-4e09-8a40-a1b0f1e
8066c','317549d0-c4e5-400d-9ee5-72066869b1b3','02ed7f16-1337-454d-ac47-3250551a1588','3d855aed-83af-
4fe3-92fd-3d105a3ebd02','474f8b7a-17ce-466e-8b14-4ce53d610d40','4c33d1a2-759d-4b6a-8a37-b5e7249fce24
','52d997ed-9a1e-45de-ad89-6d4f542a321a','639528cd-4299-48ae-854a-2b7a33384f96','65ca58f2-ee3a-478f-
9343-da4a932ee143','69e88d0c-2bf0-45de-bfb9-7e023a78bb21','75b1d3f0-737e-46f6-afb1-c096c230de49','75
cd6e33-afa0-42c1-aec4-c8a0bf4c0d71','7d4ecaef-6be5-4ee5-bd9b-696d3f7dec23','7f04a44f-533d-46a8-b3ab-
f23f488a57dc','8c410beb-f10d-4dd6-941f-1f9c63962c28')    AND ( `F_DeleteMark` IS NULL )ORDER BY `F_S
ORTCODE` ASC
SELECT `F_Id` FROM `BASE_USER`  WHERE `F_MANAGERID` IN ('bd8c8775-5a47-4d06-952e-7c4cfe86d13e','964b
44e3-b813-432a-8d05-cf6c36056b0d','969b6be5-c9c2-432b-b952-7953c93577b7','99e7703f-f0e0-48ab-8d6d-82
1a4da6e8bc','a0923414-c5d3-4bee-8ab1-b03d879a227c','a4a0a204-ebbe-4428-accf-0c123abdd983','admin','a
fe0a868-3d70-4c6b-8200-58898c0e4e72','b0afc597-c533-4b21-950a-7f33081b3413','b10cfe40-6855-4934-8ea2
-47171e9ec9ad','b77b8f5b-1a57-4c4a-8233-7dad385cc611','bae583b0-a120-4a4d-9892-287a0ef0210b','bbb00a
3b-abbd-4e30-8b83-0ae0b06d5623','91e531c1-eba1-45e3-8243-2fd1cc84dde9','c76ecfed-30c1-4e3d-8517-18de
ad5e59ee','cf22426e-227d-40a3-8f9f-b45106934e04','e0d9ff45-4b86-4139-a8a4-548e01d34996','e10c139d-3e
e3-4b19-9b5b-93459ae31628','e3bccece-4c8f-49c3-86fb-feb9ba143726','e4582489-a568-4527-b31d-7b011c089
340','eb3ea438-2f7e-47d9-81a9-30e73ff3b909','f3322d81-a007-4e28-9c43-55489ef1d3db','f77c6a01-e3e5-4d
09-aa7f-c0a6890e4546','f8c18307-065c-4dd3-a6c6-a2b95ba17ed6','f96d5799-c894-4f00-81dc-4caa97e294e4',
'fa0719d2-4a8b-4bac-ba2b-b65fbc286c1b','3bbad457-663e-4314-9b10-33c67c35cbcb','048657c1-ecb1-4e5d-ac
ed-6126ceae779e','0a85de84-4de0-4317-9767-25f95fdd9b1e','0c1d9463-221f-49ba-8e38-28c1f1df0c4b','146f
6c3a-9eaa-4295-817f-8bf4e19b1a20','1adea3ff-8a25-420b-a628-c5cb78fd51c0','1d91dd85-989e-4252-8d7f-e4
2e04bc6b3d','22736bf4-f2d5-4081-9fb4-ca43d6235f56','25e10fa4-7c68-4bc4-b751-92fa54f24c5b','26fe6815-
20a2-4be7-93b4-25e195d22833','27c57c5a-4c50-4572-954b-f9dafbbbca89','2ab1ce27-0903-4e09-8a40-a1b0f1e
8066c','317549d0-c4e5-400d-9ee5-72066869b1b3','02ed7f16-1337-454d-ac47-3250551a1588','3d855aed-83af-
4fe3-92fd-3d105a3ebd02','474f8b7a-17ce-466e-8b14-4ce53d610d40','4c33d1a2-759d-4b6a-8a37-b5e7249fce24
','52d997ed-9a1e-45de-ad89-6d4f542a321a','639528cd-4299-48ae-854a-2b7a33384f96','65ca58f2-ee3a-478f-
9343-da4a932ee143','69e88d0c-2bf0-45de-bfb9-7e023a78bb21','75b1d3f0-737e-46f6-afb1-c096c230de49','75
cd6e33-afa0-42c1-aec4-c8a0bf4c0d71','7d4ecaef-6be5-4ee5-bd9b-696d3f7dec23','7f04a44f-533d-46a8-b3ab-
f23f488a57dc','8c410beb-f10d-4dd6-9
```

解决Dotnet无法预览问题：

appsettings.json

```
      "app_id": "",
      "app_key": "",
      "redirect_uri": "",
      "scope": "snsapi_userinfo"
    }
  },
  "JNPF_App": {
    "CodeAreasName": "SubDev",
    //系统文件路径(末尾必须带斜杆)
    "SystemPath": "//www//wwwroot//resources//",
    //微信公众号允许上传文件类型
    "MPUploadFileType": "bmp,png,jpeg,jpg,gif,mp3,wma,wav,amr,mp4",
    //微信允许上传文件类型
    "WeChatUploadFileType": "jpg,png,doc,docx,ppt,pptx,xls,xlsx,pdf,txt,rar,zip,csv,amr,mp4",
    //允许图片类型
    "AllowUploadImageType": "jpg,gif,png,bmp,jpeg,tiff,psd,swf,svg,pcx,dxf,wmf,emf,lic,eps,tga",
    //允许上传文件类型
    "AllowUploadFileType": "jpg,gif,png,bmp,jpeg,doc,docx,ppt,pptx,xls,xlsx,pdf,txt,rar,zip,csv",
    "Domain": "http://192.168.0.186",
    "YOZO": {
      "domain": "http://dcsapi.com/",
      "domainKey": "57462250284462899305150"
    },
    //Minio
    "BucketName": "jnpfsoftoss",
    //文件存储类型(本地:local,MinIo:minio,阿里云:aliyun-oss,腾讯云:tencent-cos)
    "FileStoreType": "local",
    //================== 系统错误邮件报告反馈相关 ============================== -->
    //软件的错误报告
    "ErrorReport": "false",
    //软件的错误报告发给谁
    "ErrorReportTo": "yinmaisoft@163.com"
  }
```

附录：

Windows环境

项目说明

> 以下为 `JNPF 3.2.x` .NET项目命名规则

项目名	说明
`JNPF.Net`	.NET主项目

环境要求

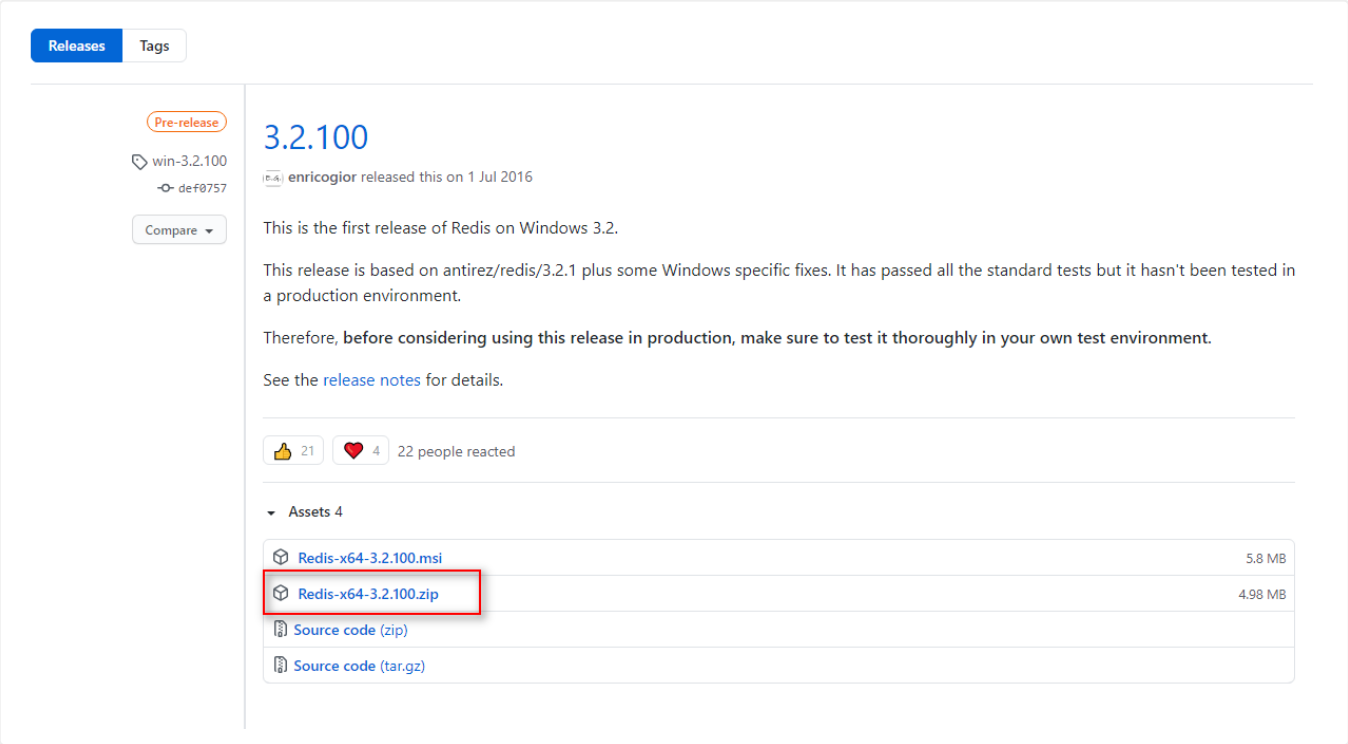
基础要求

> 环境具体安装教程

环境	推荐版本	说明
Nginx		
JDK	1.8.x	JAVA环境依赖(需配置环境变量)
SDK	最新版	.NET环境依赖
Redis	3.2.100(Windows)/6.0.x(Linux,Mac)	
MySQL	5.7.x+	数据库任选一
SQLServer	2012+	数据库任选一
Orale	11g+	数据库任选一

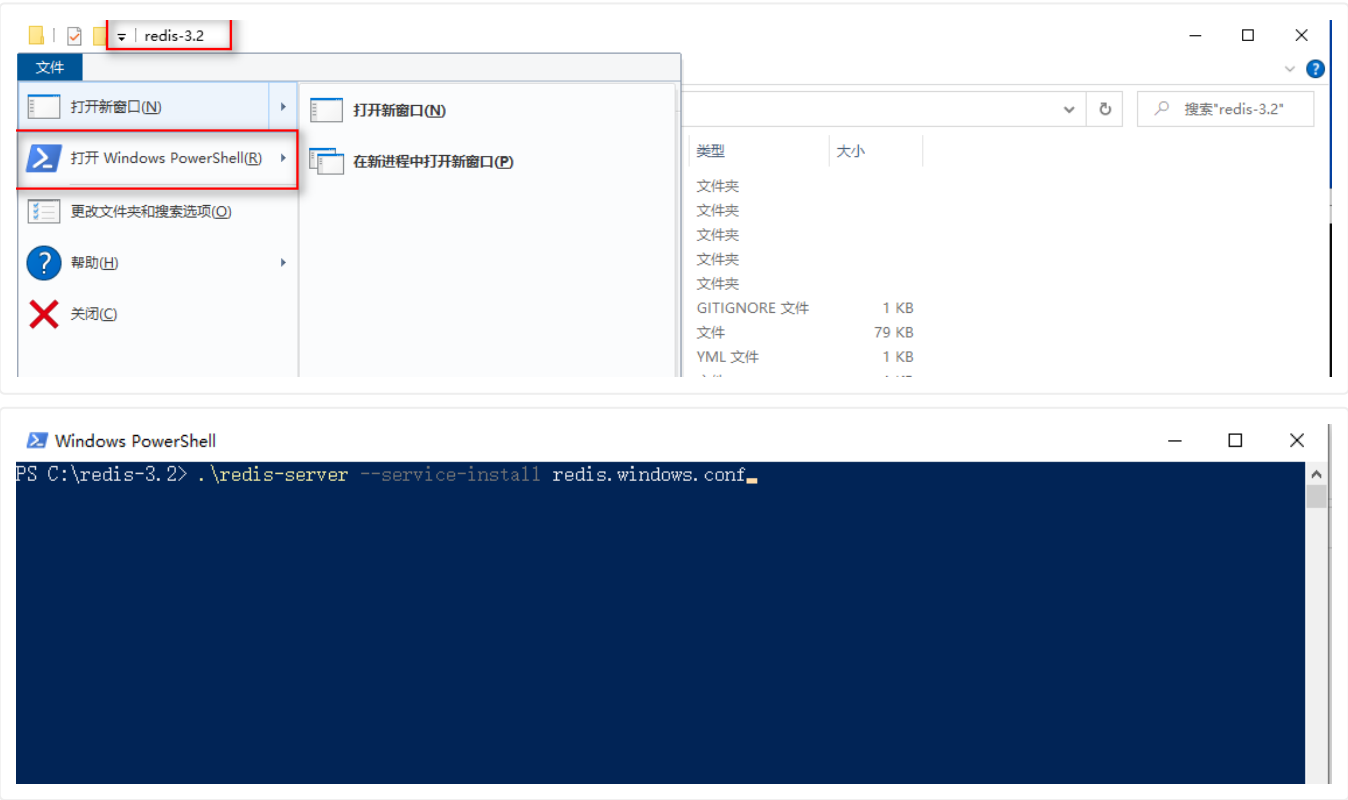
1. Redis安装

- Redis下载：https://github.com/microsoftarchive/redis/releases



配置并设置为服务

1） 打开命令行(`Windows PowerShell`)，输入.\redis-server --service-install redis.windows.conf



2） 右击 `电脑` – `管理` – `服务和应用程序` – `服务` 启动服务

3） `Redis` 常用的指令

- 卸载服务：`redis-server --service-uninstall`
- 开启服务：`redis-server --service-start`
- 停止服务：`redis-server --service-stop`

2. nginx安装

Nginx下载（Windows版）：http://nginx.org/en/download.html
启动命令

```
#启动命令:
start nginx.exe
#停止命令:
nginx.exe -s stop
#修改nginx.conf后不断服务重启:
nginx.exe -s reload
```

进入 `nginx-conf` 文件夹，修改 `nginx.conf` 配置文件：

1.http网址的转发配置，如下：

```
server
{
  listen 80;
  server_name 192.168.0.247;
  index index.php index.html index.htm default.php default.htm default.html;
  root c:\wwwroot\jnpf-web;

  #JNPF-Start
  # 前端伪静态配置
  # 主项目前端
  location / {
    try_files $uri $uri/ /index.html;
  }
  # 大屏前端
  location /DataV {
    try_files $uri $uri/ /DataV/index.html;
  }
  #设置上传文件的大小
  client_max_body_size 100m;
  #添加头部信息
  proxy_set_header Cookie $http_cookie;
  proxy_set_header X-Forwarded-Host $host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
  client_header_buffer_size 128k;
  #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
  large_client_header_buffers 4 128k;
  #指定允许跨域的方法，*代表所有
  add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';
  # 预检命令的缓存，如果不缓存每次会发送两次请求
  add_header Access-Control-Max-Age 3600;
  #带cookie请求需要加上这个字段，并设置为true
  add_header Access-Control-Allow-Credentials true;
```

```
    #  表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #  $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #  表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;

    # 后端接口(按实际情况修改端口)
    location /api/ {
      proxy_pass http://localhost:30000/api/;
    }
    # webscket接口(按实际情况修改端口)
    location /websocket {
      proxy_pass http://localhost:30000/api/message/websocket;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
      proxy_read_timeout 600s;
    }
    # 报表设计接口配置(按实际情况修改端口)
    location /ReportServer/ {
      proxy_pass http://localhost:30007/;
    }
    # 文件预览服务
    location /FileServer {
      proxy_pass http://localhost:30090;
    }
    # 解决文件预览服务无法加载js,css问题
    location ~ /FileServer/*.*\.(js|css)?$ {
                  proxy_pass http://localhost:30090;
    }
  #JNPF-End
  }
```

2.https网址的转发配置，如下：

```
server
{
    listen 80;
    listen 443 ssl http2;
    server_name 192.168.0.247;
    index index.php index.html index.htm default.php default.htm default.html;
    root c:\wwwroot\jnpf-web;

    #SSL-START SSL相关配置，请勿删除或修改下一行带注释的404规则
    #error_page 404/404.html;
```

```
    #HTTP_TO_HTTPS_START
    if ($server_port !~ 443){
        rewrite ^(/.*)$ https://$host$1 permanent;
    }
    #HTTP_TO_HTTPS_END
    ssl_certificate    c:\www\192.168.0.247\fullchain.pem;
    ssl_certificate_key    c:\www\192.168.0.247\privkey.pem;
    ssl_protocols TLSv1.1 TLSv1.2 TLSv1.3;
    ssl_ciphers EECDH+CHACHA20:EECDH+CHACHA20-draft:EECDH+AES128:RSA+AES128:EECDH+AE
S256:RSA+AES256:EECDH+3DES:RSA+3DES:!MD5;
    ssl_prefer_server_ciphers on;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;
    add_header Strict-Transport-Security "max-age=31536000";
    error_page 497  https://$host$request_uri;


    #SSL-END

    #ERROR-PAGE-START  错误页配置，可以注释、删除或修改
    #error_page 404 /404.html;
    #error_page 502 /502.html;
    #ERROR-PAGE-END

    #PHP-INFO-START  PHP引用配置，可以注释或修改
    include enable-php-00.conf;
    #PHP-INFO-END

    #REWRITE-START URL重写规则引用,修改后将导致面板设置的伪静态规则失效
    include c:\www\192.168.0.247;
    #REWRITE-END

    #禁止访问的文件或目录
    location ~ ^/(\.user.ini|\.htaccess|\.git|\.svn|\.project|LICENSE|README.md)
    {
        return 404;
    }

    #一键申请SSL证书验证目录相关设置
    location ~ \.well-known{
        allow all;
    }

    # location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
    # {
```

```
    #     expires      30d;
    #     error_log /dev/null;
    #     access_log /dev/null;
    # }

    # JNPF-START
    #设置上传文件的大小
    # client_max_body_size 100m;

    # #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    # #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    # client_header_buffer_size 128k;

    # #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
    # large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    # 预检命令的缓存，如果不缓存每次会发送两次请求
    #带cookie请求需要加上这个字段，并设置为true
    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号     #
表示请求头的字段 动态获取
    # 前端主项目(jnpf-web)伪静态
    location / {
      try_files $uri $uri/ /index.html;
    }

    # 前端大屏(jnpf-web-datascreen)伪静态
    location /DataV {
      try_files $uri $uri/ /DataV/index.html;
    }

    # 主项目
    location /api/ {
      proxy_pass http://localhost:30000;
    }

    location /websocket {
      proxy_pass http://localhost:30000/api/message/websocket;
      proxy_http_version 1.1;
```

```
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_read_timeout 600s;
      }

      # 报表（jnpf-datareport）接口
      location /ReportServer/ {
       proxy_pass http://localhost:30007/;
      }

      # 文件预览（jnpf-file-preview）
      location /FileServer {
        proxy_pass http://localhost:30090;
      }

      location ~ /FileServer/*.*\.(js|css)?$ {
                          proxy_pass http://localhost:30090;
      }
      # JNPF-End

      location ~ .*\.(js|css)?$
      {
          expires      12h;
          error_log /dev/null;
          access_log /dev/null;
      }
  }
```

3.APP的接口转发配置，如下：

```
    #JNPF-Start

    #设置上传文件的大小
    client_max_body_size 100m;

    #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    client_header_buffer_size 128k;

    #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
```

```
    large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';

    # 预检命令的缓存，如果不缓存每次会发送两次请求
    add_header Access-Control-Max-Age 3600;

    #带cookie请求需要加上这个字段，并设置为true
    add_header Access-Control-Allow-Credentials true;

    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #    表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;


    # 伪静态
    location / {
      try_files $uri $uri/ /index.html;
    }

    # 后端接口
    location /api/ {
      proxy_pass http://localhost:30000;
    }

    # websocket
    location /websocket {
      proxy_pass http://localhost:30000/api/message/websocket;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
      proxy_read_timeout 600s;
    }
    #JNPF-End
```

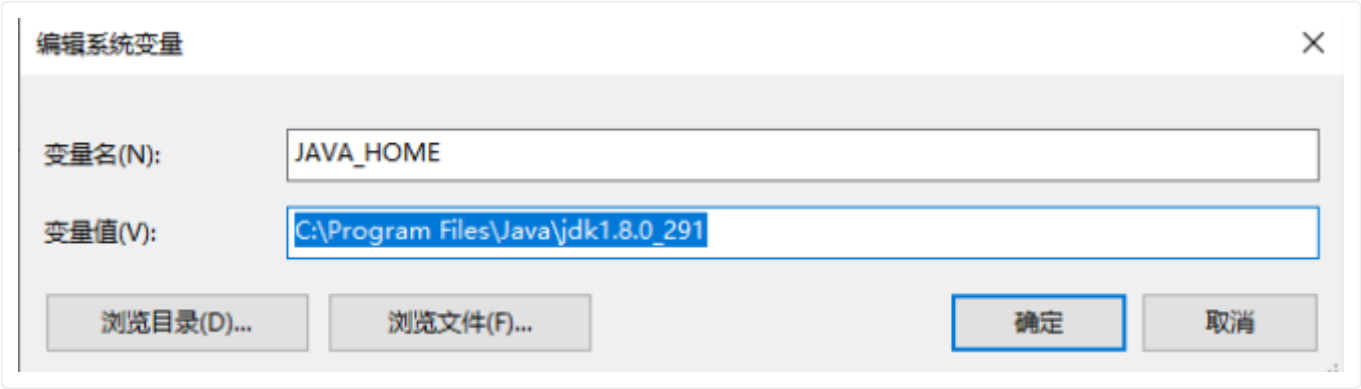
重启ngin.exe。

3. Java安装

1).JDK下载http://www.oracle.com/technetwork/java/javase/downloads/index.html
或者coding网盘下载(下载1.8.291版本)

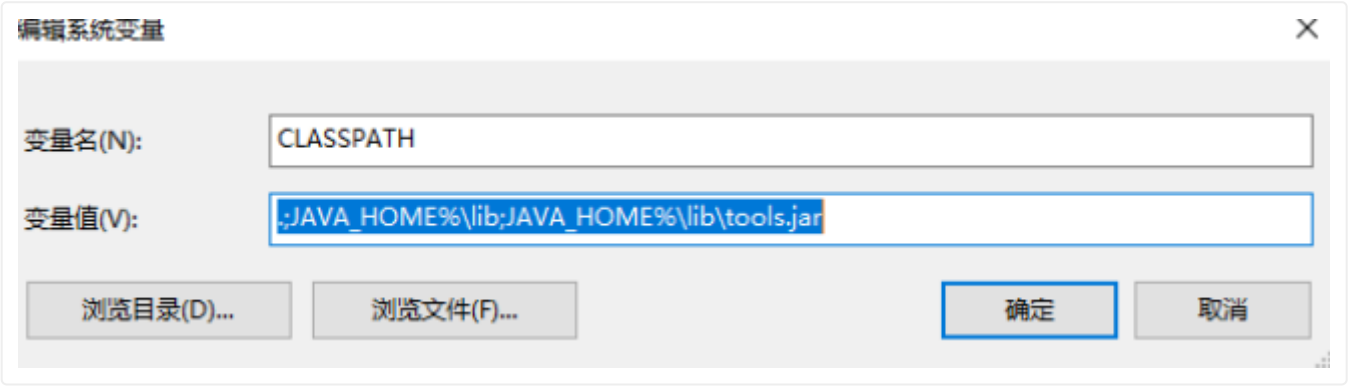
2).默认安装直至最后一步

3).点击系统变量下面的新建按钮，变量名JAVA_HOME（代表你的JDK安装路径），值对应的是你的JDK的安装路径。



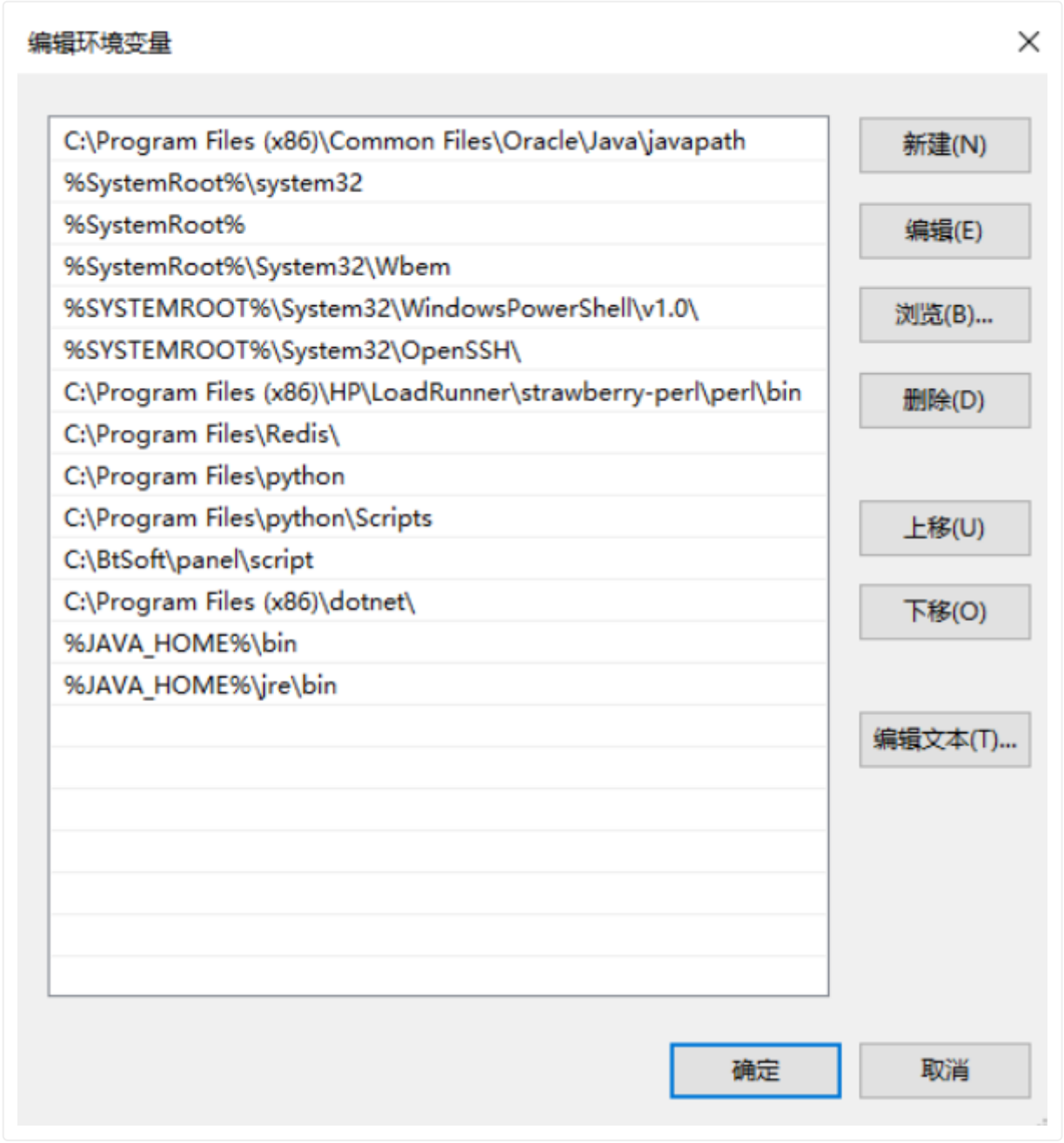
4).继续在系统变量里面新建一个CLASSPATH变量，其变量值如下图所示：

```
.;JAVA_HOME%\lib;JAVA_HOME%\lib\tools.jar
```



5).在你的系统变量里面找一个变量名是PATH的变量，需要在它的值域里面追加一段如下的代码：

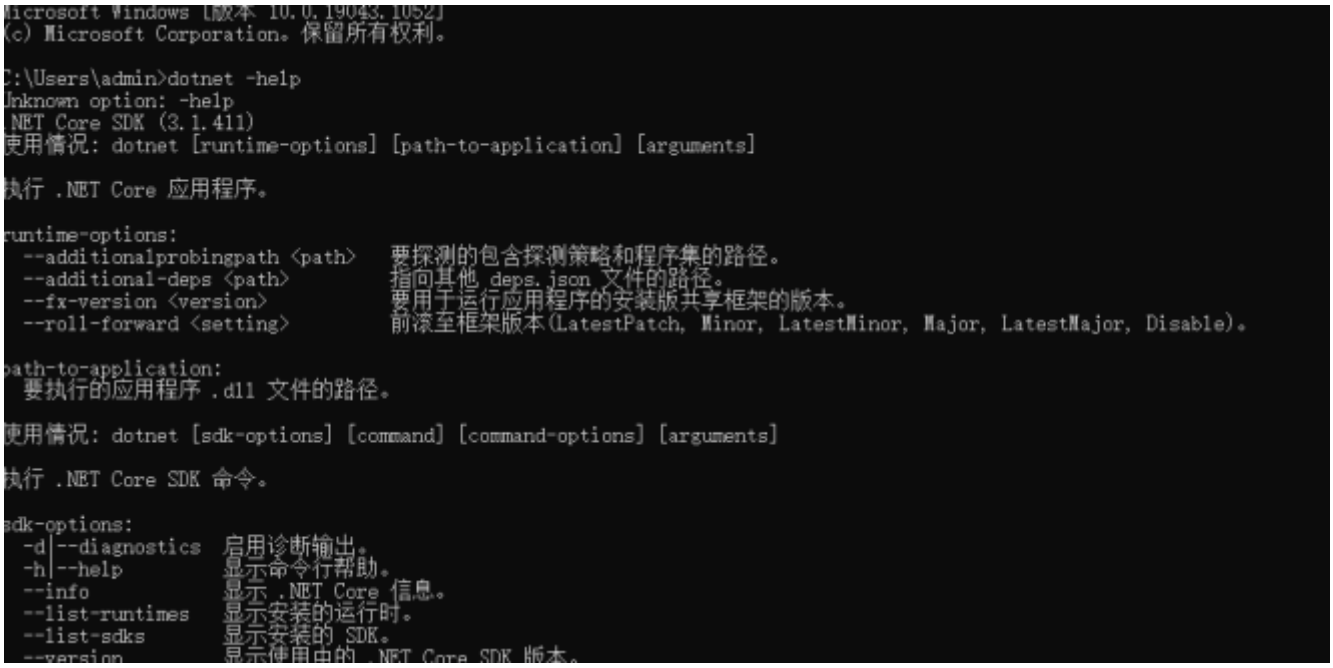
```
%JAVA_HOME%\bin
%JAVA_HOME%\jre\bin
```



6).测试自己所配置的环境变量是否正确，WINDOWS+R键，输入cmd，进入命令行界面，如下所示：

4.ASP.NET Core SDK安装

下载：https://dotnet.microsoft.com/download/dotnet/6.0
下载后默认安装直至最后一步。
WINDOWS+R键，输入cmd，命令窗口输入dotnet -help，如图显示安装成功。



注：liunx安装.net SDK工具：https://docs.microsoft.com/zh-cn/dotnet/core/install/linux-centos

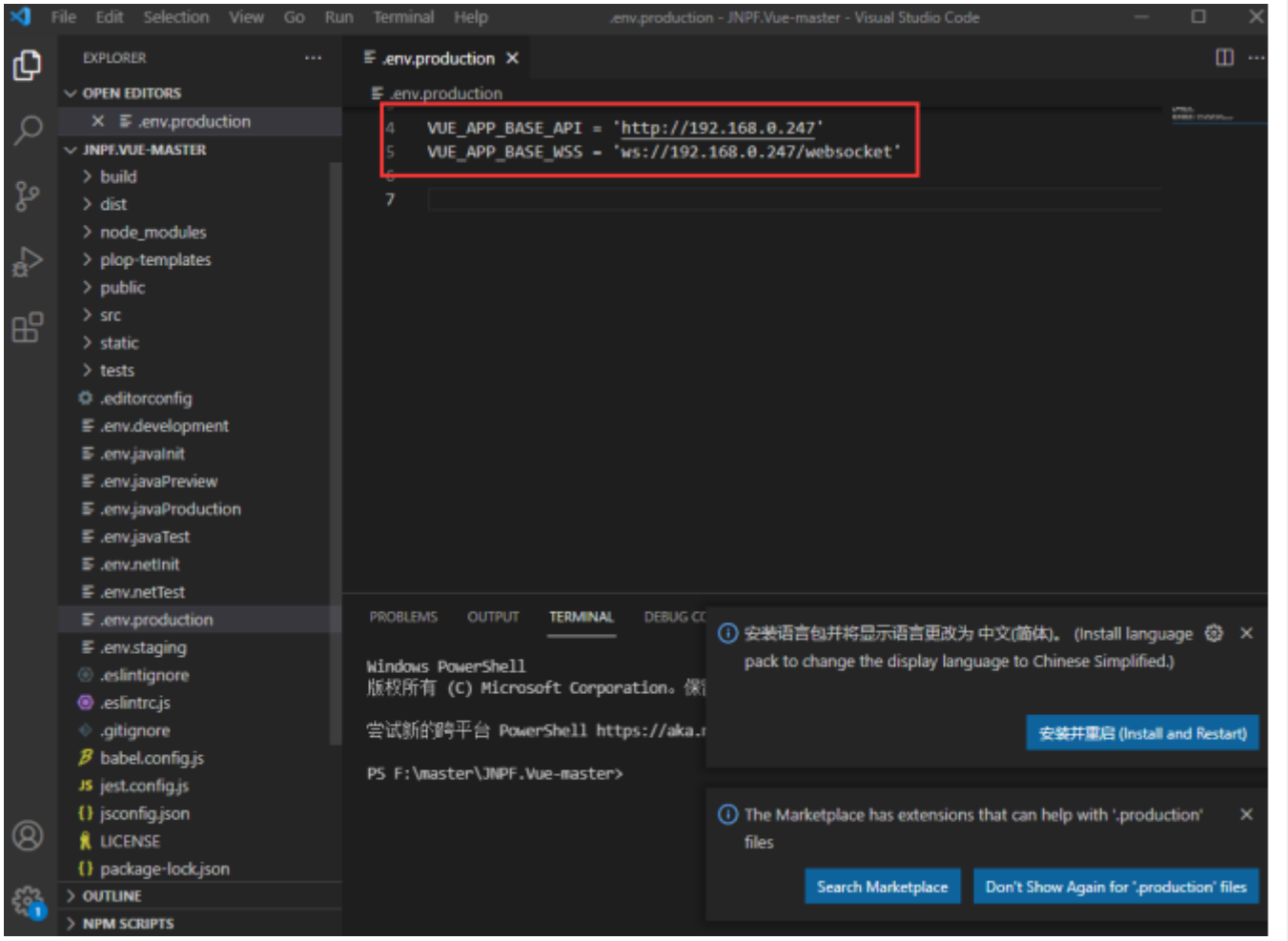
5.主项目前端项目部署

`Visual studio cood` 打开前端项目：

安装依赖

```
npm i
```

修改.env.production数据：



打包

```
npm run build
```

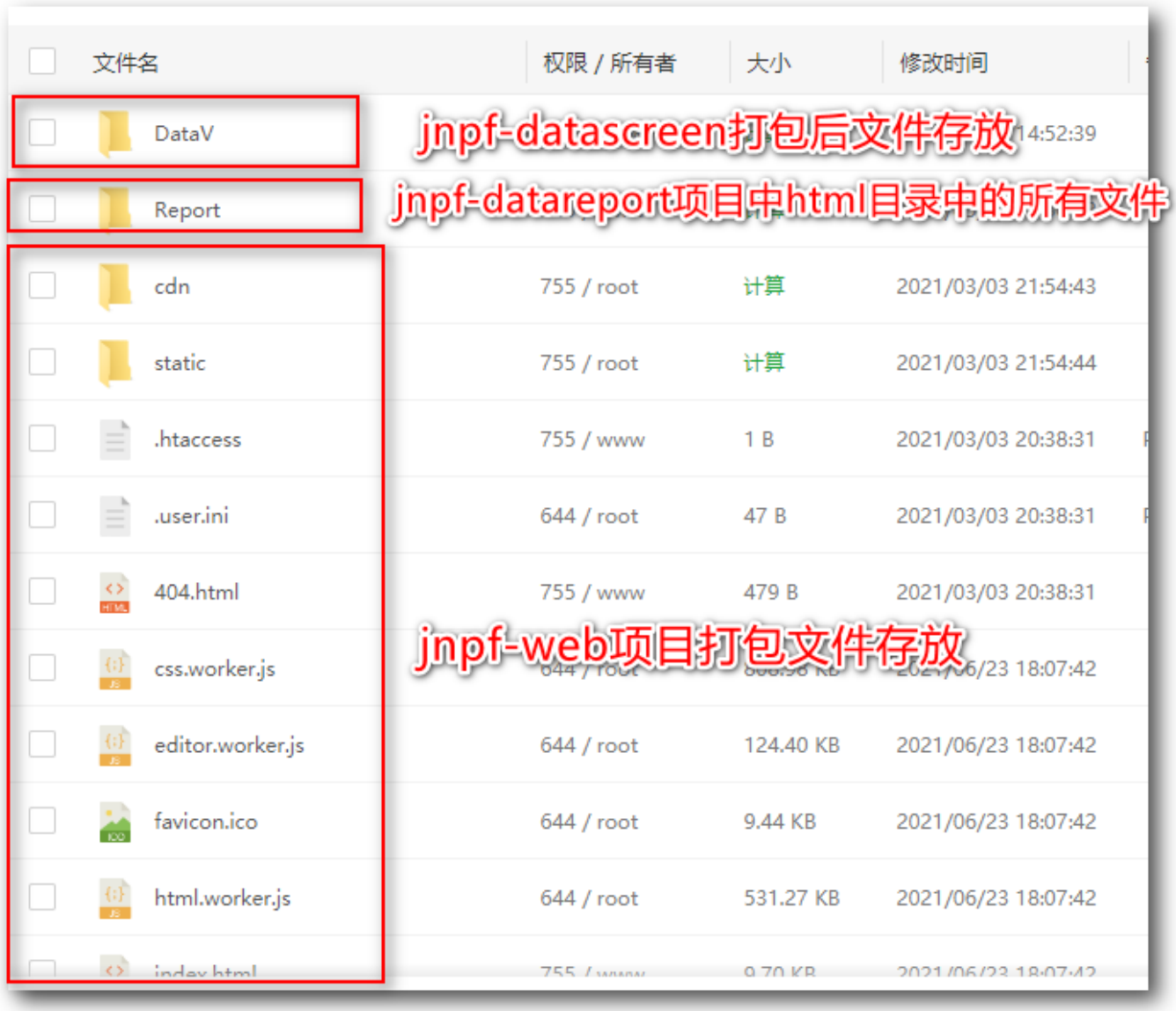
上传目录dist文件夹下全部文件到指定目录（或者配置nginx.cof）

```
server
{
    listen 80;
    listen 443 ssl http2;
    server_name 192.168.0.247;
    index index.php index.html index.htm default.p
    root c:\wwwroot\jnpf-web;
```

重启nginx访问即可。

6.报表前端部署

前端部署结构说明




```
├── jnpf-web                    # 假设这个目录是存放测试或生产环境的前端
│   ├── DataV                   # 大屏(`jnpf-datascreen`)打包后文件存放目录
│   ├── Report                  # 报表(`jnpf-datareport`)html下的文件
│   └── 主项目前端打包后的文件    # 主项目(`jnpf-web`)打包后存放在根目录
```

前端

- 将 `jnpf-web-datareport` 下的 `html` 文件夹中的所有拷贝到 `jnpf-web` 中的 `Report` 目录下, 如 `Report` 目录不存在请手动建立
- 接口配置
  - 打开 `/Report/index.html` ,做如下修改

```
# 在index.html文件中第24行开始
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.247/ReportServer";
// 报表前端
```

```
window._contextPath = "http://192.168.0.247/Report";
// 主项目接口地址
window._mainServer = "http://192.168.0.247";
</script>
```

- 打开 `/Report/preview.html` ,做如下修改,

```
# 在preview.html文件中第86行
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.247/ReportServer";
</script>
```

- 打开 `/Report/searchform.html` ,做如下修改,

```
# 在searchform.html文件中
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.247/ReportServer";
// 报表前端
window._contextPath = "http://192.168.0.247/Report";
</script>
```

- 配置说明:
  - 本示例中， `http://192.168.0.0:247` 为项目在测试环境中访问入口，在部署中根据实际情况调整;
  - 报表接口中， `ReportServer` 为虚拟目录，需要在Nginx增加相关配置，具体配置如下:

```
# 数据报表接口配置
  location /ReportServer/ {
     proxy_pass http://localhost:30007/;
  }
```

7.大屏前端部署

配置

在项目根目录打开 `.env.staging` (测试环境配置)，部署生产环境请打开 `.env.product` 文件

```
# 测试默认配置
ENV = 'staging'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.247'
```

```
# 生产环境默认配置
ENV = 'production'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.247'
```

构建

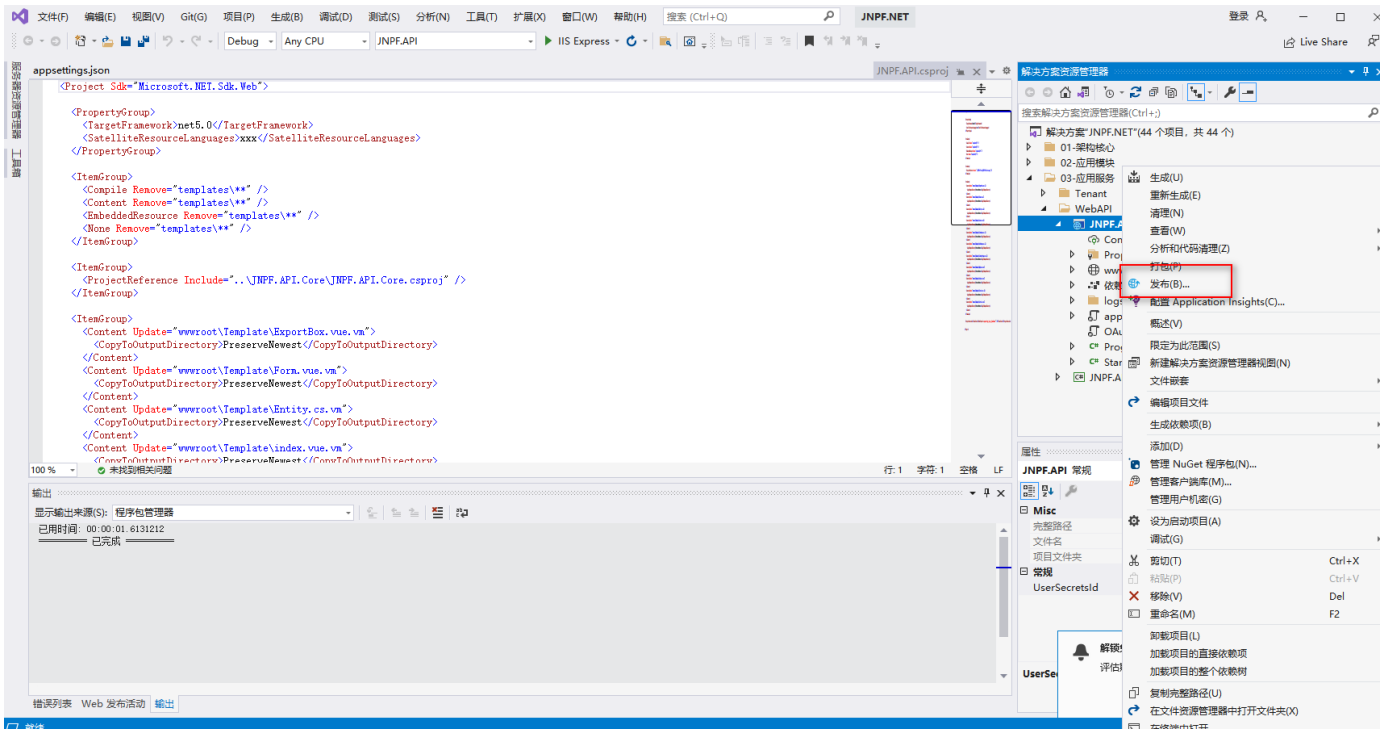
```
# 构建测试环境
yarn build:staging

# 构建生产环境
yarn build
```

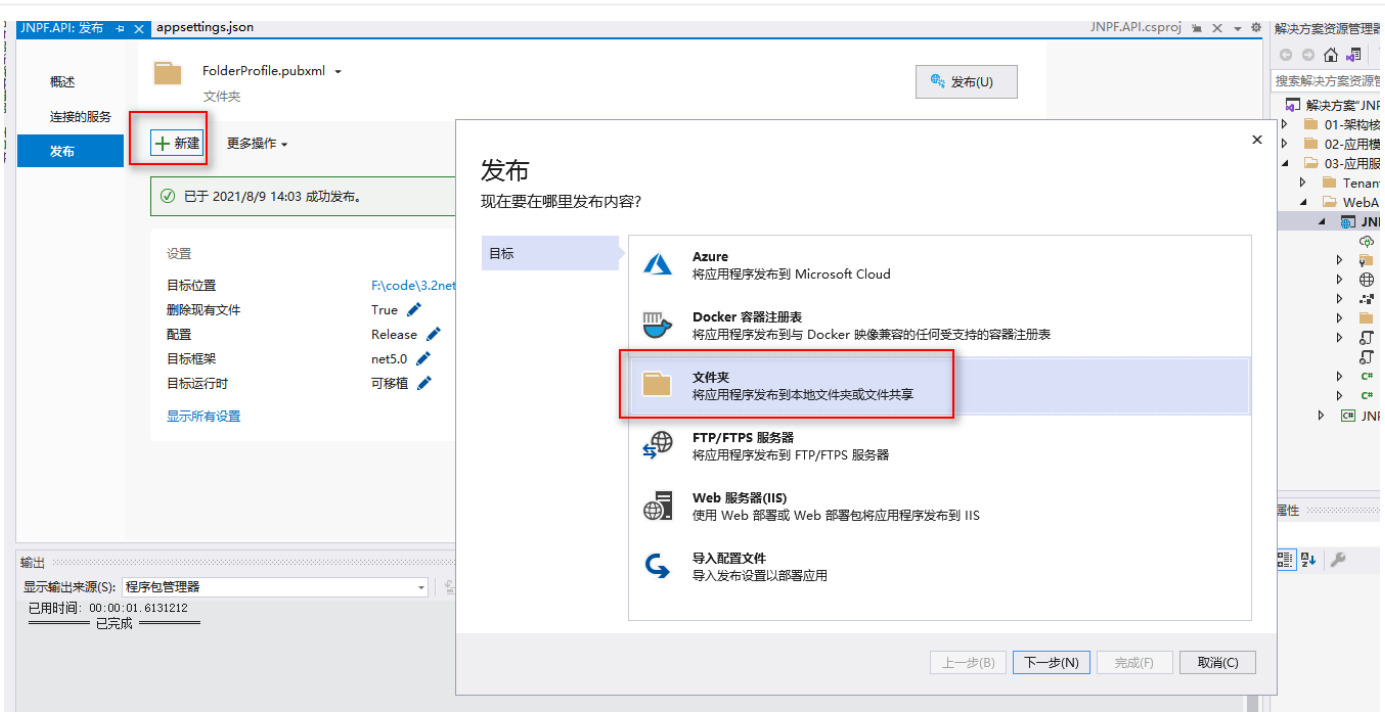
发布到服务器

- 压缩dist目录，上传到服务器C:\wwwroot\jnpf-web\DataV并解压

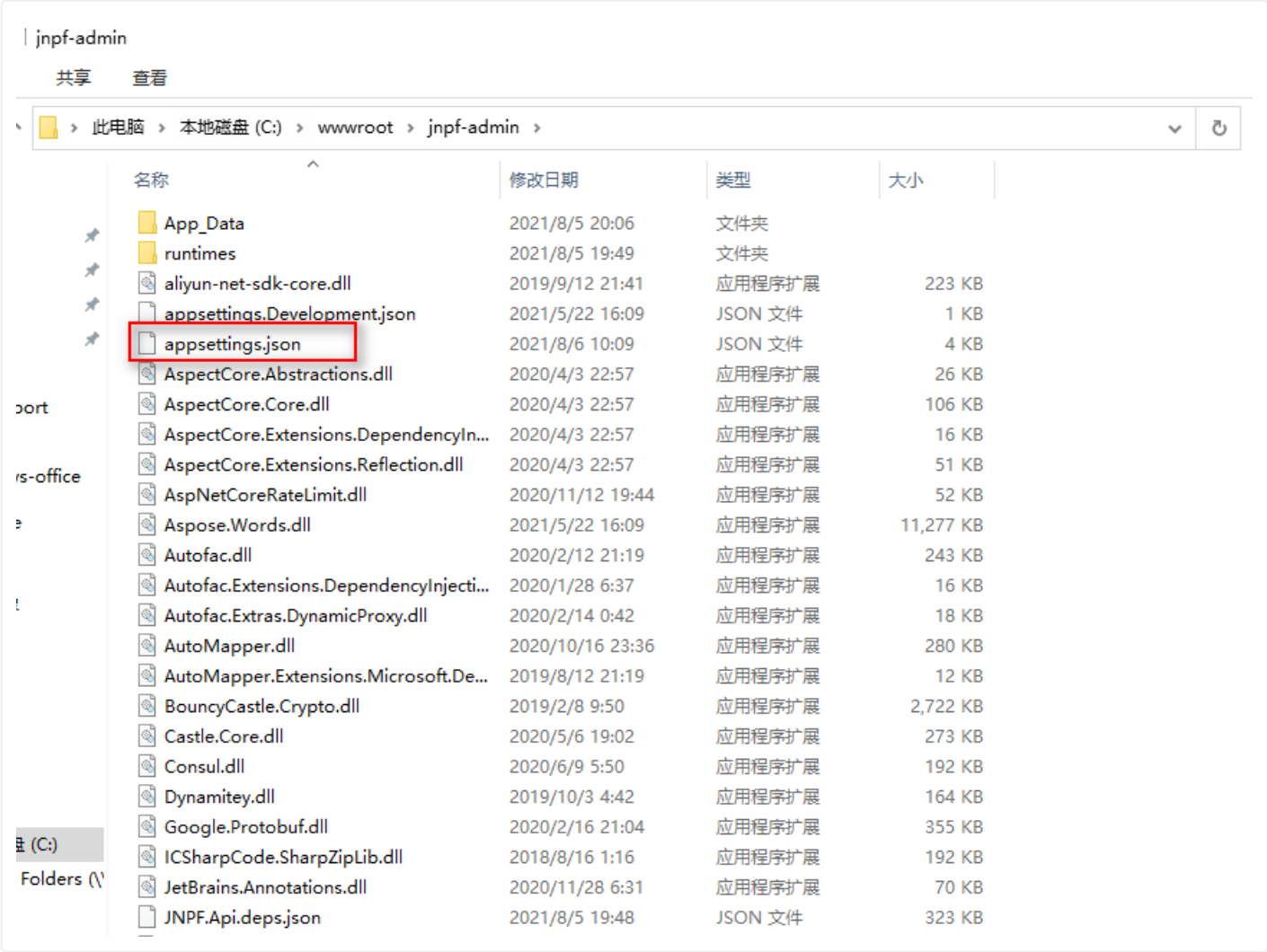
8.后端项目打包

Visual studio 2019打开后端项目，在应用服务-JNPF-API 右键选择“发布”


appsettings.json
JNPF.API.csproj
解决方案资源管理器
发布(B)...
输出
显示输出来源(S): 程序包管理器
已用时间: 00:00:01.6131212
已完成


JNPF.API: 发布
appsettings.json
JNPF.API.csproj
概述
连接的服务
发布
FolderProfile.pubxml
文件夹
发布(U)
新建
更多操作
已于 2021/8/9 14:03 成功发布。
设置
目标位置
删除现有文件
配置
目标框架
目标运行时
显示所有设置
发布
现在要在哪里发布内容?
目标
Azure
将应用程序发布到 Microsoft Cloud
Docker 容器注册表
将应用程序发布到与 Docker 映像兼容的任何受支持的容器注册表
文件夹
将应用程序发布到本地文件夹或文件共享
FTP/FTPS 服务器
将应用程序发布到 FTP/FTPS 服务器
Web 服务器(IIS)
使用 Web 部署或 Web 部署包将应用程序发布到 IIS
导入配置文件
导入发布设置以部署应用
上一步(B)
下一步(N)
完成(F)
取消(C)
输出
显示输出来源(S): 程序包管理器
已用时间: 00:00:01.6131212
已完成

发布文件拷贝指定目录（新建文件夹），修改appsetting.josn 配置。



1).修改Redis：

2).修改静态文件resouces文件位置：

3).修改数据库地址：

连接SQLServer：

```
  "ConnectionStrings": {
    "ConfigId": "default",
    "DBName": "jnpf_init",
    "DBType": "SqlServer", //MySql;SqlServer;Oracle;PostgreSQL;Dm;Kdbndp;Sqlite;
    "Host": "192.168.0.10",
    "Port": "1433",
    "UserName": "sa",
    "Password": "K3%*S4D!XDqE3k@",
    //SqlServer
    "DefaultConnection": "Data Source=192.168.0.10;Initial Catalog={0};User ID=sa;Password=K3%*S4D!XDqE3k@;MultipleActiveResultSets=true"
  },
```

```
  "ConnectionStrings": {
    "ConfigId": "default",
    "DBName": "jnpf_init",
    "DBType": "SqlServer", //MySql;SqlServer;Oracle;PostgreSQL;Dm;Kdbndp;Sqlite;
    "Host": "192.168.0.247",
    "Port": "1433",
    "UserName": "sa",
    "Password": "K3%*S4D!XDqE3k@",
    //SqlServer
    "DefaultConnection": "Data Source=192.168.0.247;Initial Catalog={0};User ID=sa;Password=K3%*S4D!XDqE3k@;MultipleActiveResultSets=true"
  },
```

连接MYSQL：

```
  "ConnectionStrings": {
    "ConfigId": "default",
    "DBName": "jnpf_init",
    "DBType": "MySql", //MySql;SqlServer;Oracle;PostgreSQL;Dm;Kdbndp;Sqlite;
    "Host": "192.168.0.10",
    "Port": "3306",
    "UserName": "jnpf_init",
    "Password": "K3%*S4D!XDqE3k@",
    //MySql
    "DefaultConnection": "server=192.168.0.10;Database={0};Uid=jnpf_init;Pwd=pBx5HaW6WMGSTdDf;AllowLoadLocalInfile=true"
  },
```

```
  "ConnectionStrings": {
    "ConfigId": "default",
    "DBName": "jnpf_init",
    "DBType": "MySql", //MySql;SqlServer;Oracle;PostgreSQL;Dm;Kdbndp;Sqlite;
    "Host": "192.168.0.247",
    "Port": "3306",
    "UserName": "jnpf_init",
    "Password": "K3%*S4D!XDqE3k@",
    //MySql
    "DefaultConnection": "server=192.168.0.247;Database={0};Uid=jnpf_init;Pwd=pBx5HaW6WMGSTdDf;AllowLoadLocalInfile=true"
  },
```

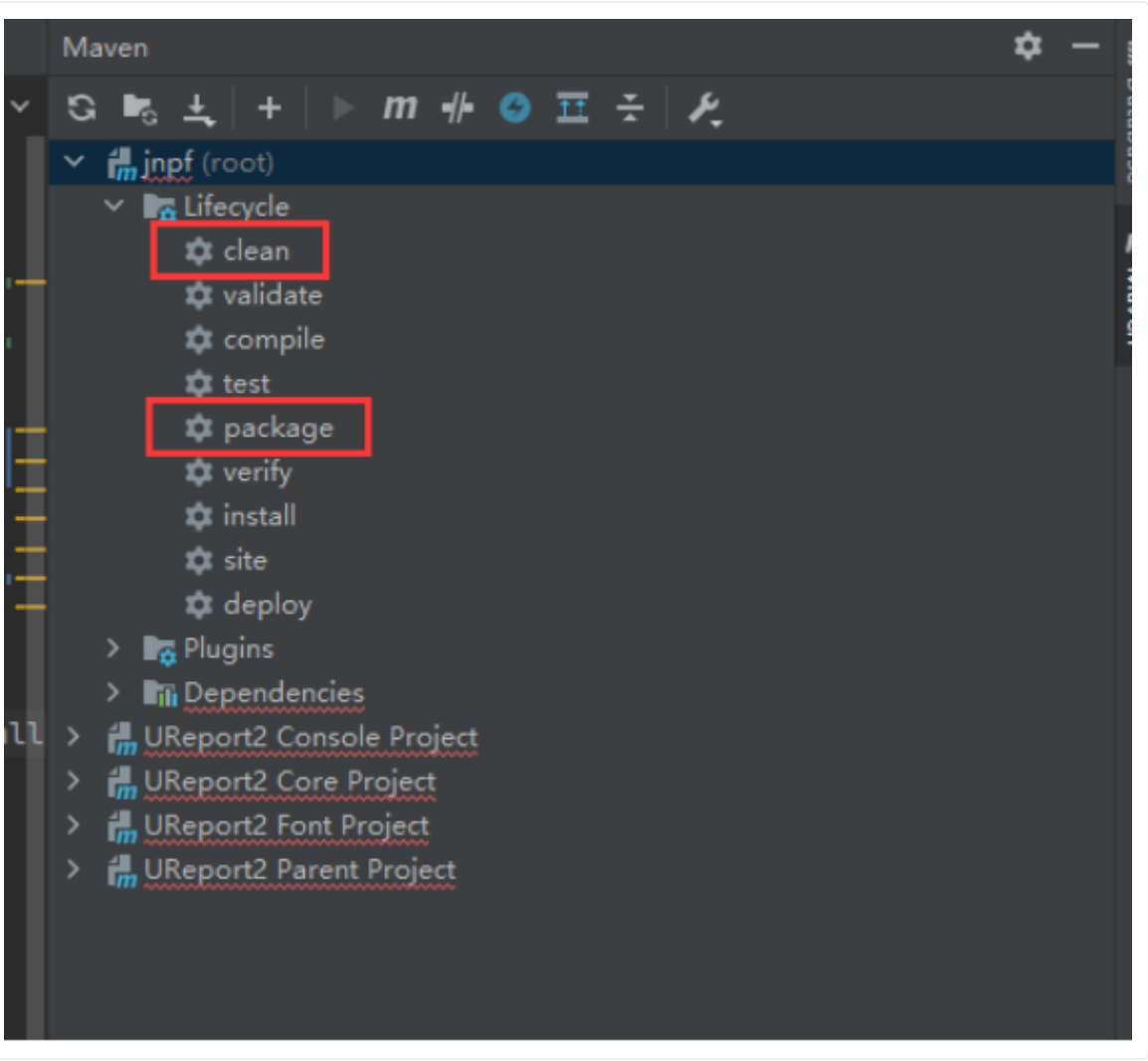
进入 c:\wwwroot\jnpf-admin\ 目录，在命令行下执行

```
dotnet JNPF.API.dll --urls="http://*:30000" --environment=Delopment > /dev/null 2>lo
g &

#根据nginx配置修改实际端口启动
```

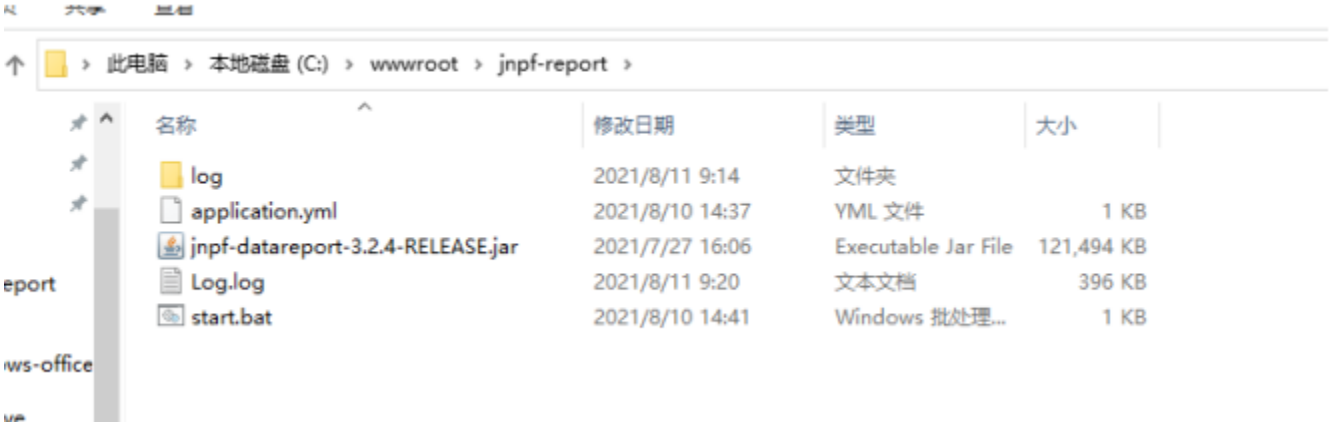
登录成功即可（报错时进入c:\wwwroot\jnpf-admin\logs可查看日志）

9.报表后端项目打包

Intellij IDEA打开报表后端项目jnpf-datareport,打包



上传指服务器指定目录下（可以新建）



打开application.Yml文件，修改数据库配置并保存：

```yaml
# 配置端口
server:
  port: 30007
  max-http-header-size: 102400
  compression:
    enabled: true
    min-response-size: 102400
spring:
  # 表空间(Oracle)
  tableSpace: JNPFCLOUD
  datasource:
    # MySQL配置
    druid:
      dbinit: jnpf_init
      dbname: jnpf_init
      dbnull: jnpf_init
      url: jdbc:mysql://localhost:3306/{dbName}?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8
      username: root
      password: 123456
      driver-class-name: com.mysql.cj.jdbc.Driver

  #Redis配置
  redis:
    database: 1
    host: localhost
    port: 6379
    password:
    timeout: 3000
    lettuce:
      pool:
        max-active: 8 # 连接池最大连接数
        max-wait: -1ms  # 连接池最大阻塞等待时间（使用负值表示没有限制）
        min-idle: 0 # 连接池中的最小空闲连接
        max-idle: 8 # 连接池中的最大空闲连接
logging:
  level:
    root: info
    com.bstek.ureport.console: debug
  file:
    path: log
config:
```

MySQL配置：

```yaml
    druid:
      dbinit: jnpf_init
      dbname: jnpf_init
      dbnull: jnpf_init
      url: jdbc:mysql://localhost:3306/{dbName}?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8
      username: root
      password: 123456
      driver-class-name: com.mysql.cj.jdbc.Driver
```

SQLServer配置:

```yaml
    druid:
      dbinit: jnpf_init
      dbname: jnpf_init
      dbnull: jnpf_init
      url: jdbc:sqlserver://localhost:1433;DatabaseName={dbName}
      username: sa
```

```
      password: JNPF@2020
      driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
```

Oracle配置:

```
  # 表空间(Oracle)
  tableSpace: JNPFCLOUD
```

```
    druid:
      dbinit: JNPFCLOUD
      dbname: JNPFCLOUD
      dbnull: JNPFCLOUD
      url: jdbc:oracle:thin:@192.168.0.247:1521:{dbName}
      username: JNPFCLOUD
      password: JNPFCLOUD
      driver-class-name: oracle.jdbc.driver.OracleDriver
```

运行.jar文件：

```
java -jar -Dfile.encoding=utf-8 -Xmx500m -Xms500m jnpf-datareport-3.2.9-RELEASE.jar
 -> Log.log
```

注：新建文本，输入命令后，修改后缀为.bat，双击直接运行。
查看日志运行成功即可使用报表模块。

10.文档预览部署

环境要求

- JDK1.8+
- OpenOffice或LiberOffice(Windows下已内置)

部署运行

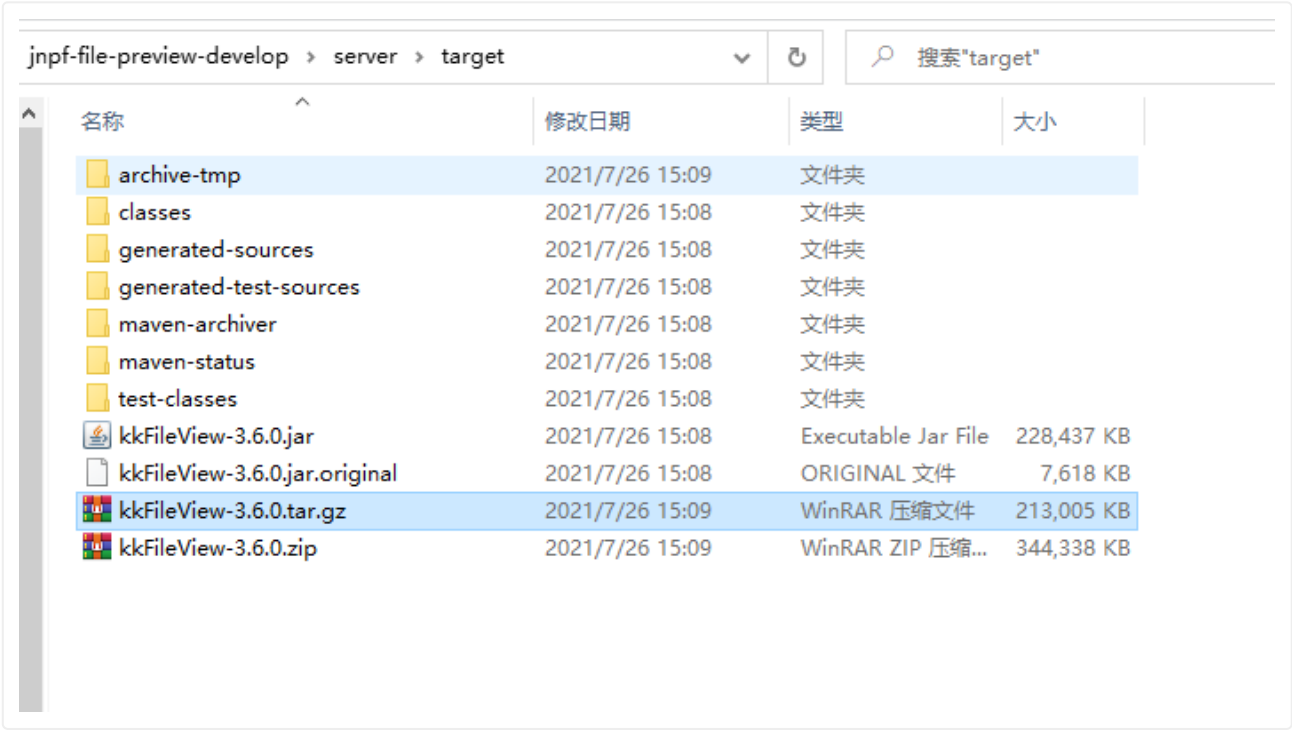
1、开发环境

- a.IDEA导入项目
- b.调整配置，打开 `server/src/main/config/application.properties`
- c.启动项目， `server/src/main/java/cn/keking/ServerMain`
- d.打开 `http://localhost:30090` 测试页面

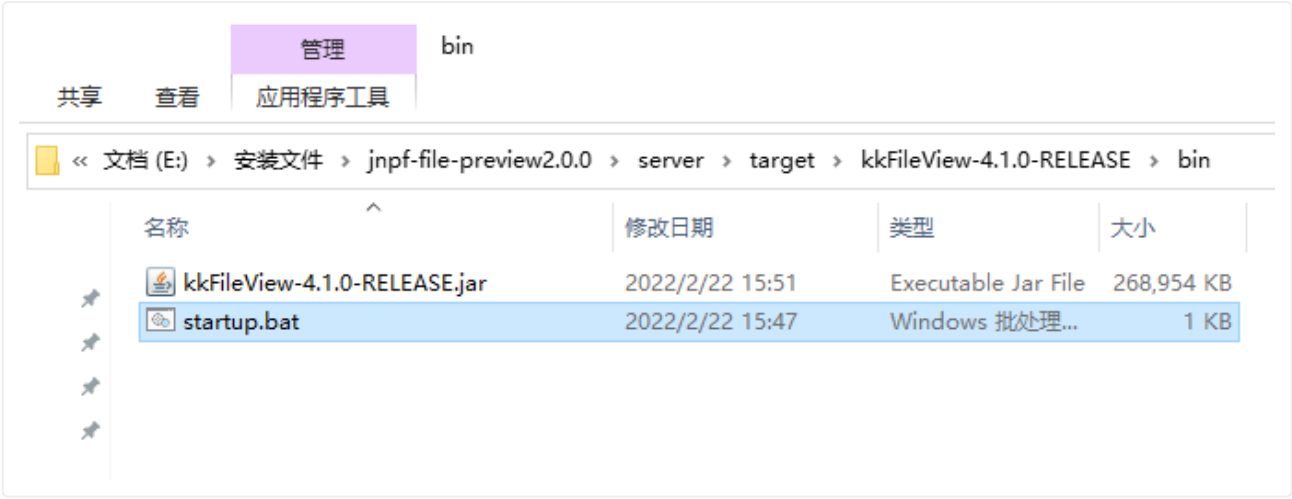
2、测试生产环境

- a.打包

  - 打开 `\server\target` 主要有以下几个文件
  - kkFileView-xxx.jar(一般用于更新)
  - kkFileView-xxx.zip(windows环境下首次部署)
  - kkFileView-xxx.tar.gz(Linux环境下首次部署)



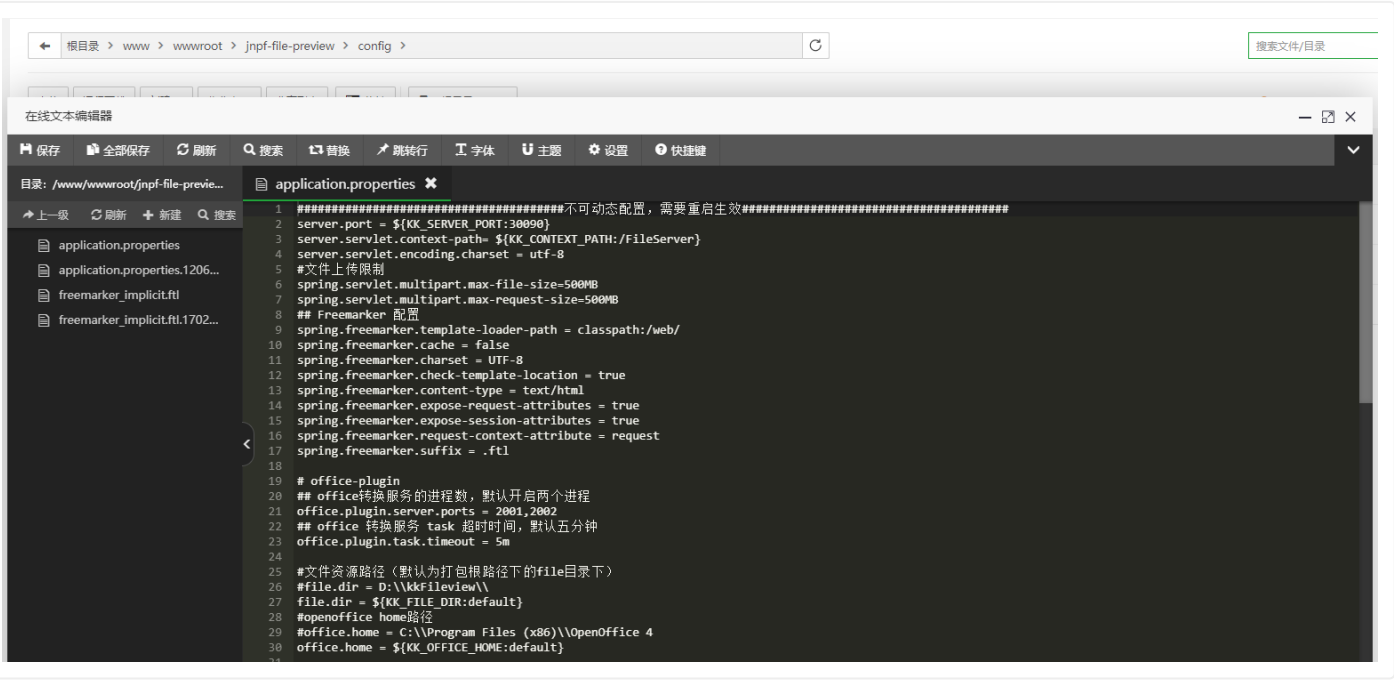
- b.上传至服务器
  - 打开解压后文件夹的bin目录，运行startup脚本(首次部署，之后更新及维护都在bin目录下)



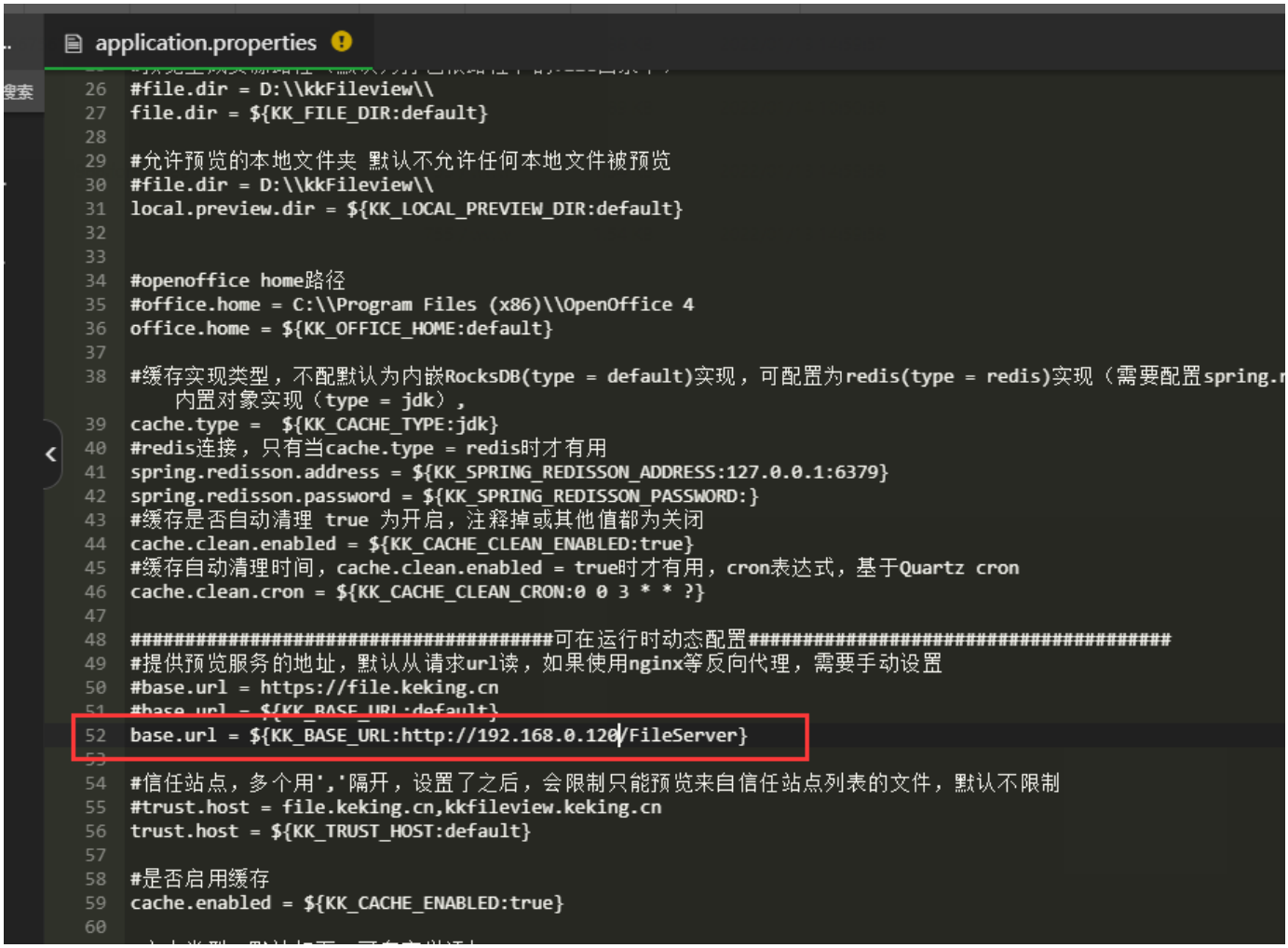
- 修改文件预览配置文件如下两项,打开服务器上的`kkFileView-xxx/config/application.properties`（第3行和第45行）

修改成：

```
server.servlet.context-path= ${KK_CONTEXT_PATH:/FileServer}
base.url = ${KK_BASE_URL:http://192.168.0.247/FileServer}
```



新版只需修改第52行：



- c.Nginx配置说明
  例如nginx的访问地址为 `https://netcore.jnpfsoft.com` ,文件预览部署在内网192.168.0.247服务器上，需要在nginx中添加反向代理如下

```
# 文件预览服务
location /FileServer {
  proxy_pass 192.168.0.247:30090;
}

# 解决文件预览服务无法加载js,css问题
location ~ /FileServer/*.*\.(js|css)?$ {
  proxy_pass 192.168.0.247:30090;
}
```

使用指南

1、单文件预览

```
// 要预览文件的访问地址
let previewUrl = 'http://127.0.0.1:8080/file/test.txt'

// 使用
http://127.0.0.1:30090/onlinePreview?url='+encodeURIComponent(Base64.encode(previewUrl))
```

2、http/https下载流url预览

```
// 要预览文件的访问地址
let previewUrl = 'http://127.0.0.1:8080/filedownload?fileId=1'

// 使用
http://127.0.0.1:30090/onlinePreview?url='+encodeURIComponent(Base64.encode(previewUrl))
```

3、更多参考官方文档（ `https://kkfileview.keking.cn/zh-cn/docs/usage.html` ）

11.常见问题

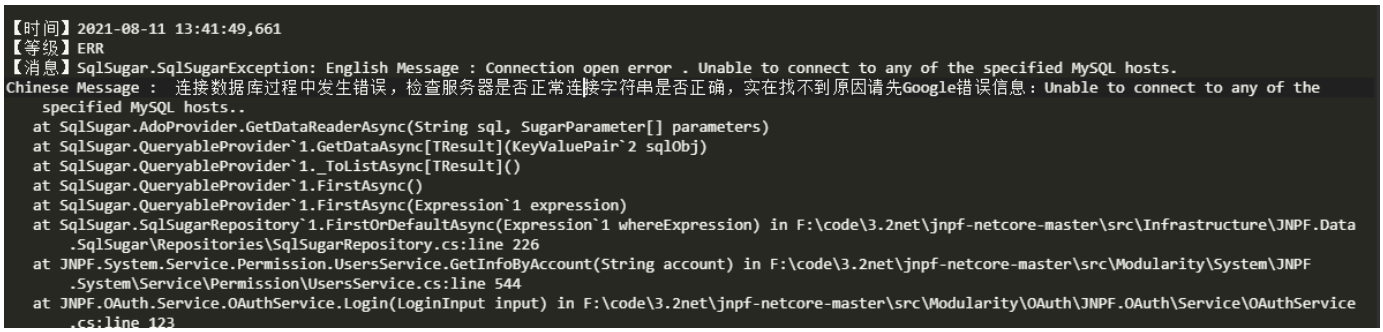
（1）mysql数据库执行查询命令：（解决只有三个菜单问题）

```
update base_module set F_DeleteMark=null
```

（2）数据库添加my.cnf：

```
lower_case_table_names = 1
```

解决mysql大小写转换问题。



解决出现死循环问题：base_user：F_ManagerId清空。

解决Dotnet无法预览问题：

appsettings.json

```
      "app_id": "",
      "app_key": "",
      "redirect_uri": "",
      "scope": "snsapi_userinfo"
    }
  },
  "JNPF_App": {
    "CodeAreasName": "SubDev",
    //系统文件路径(末尾必须带斜杆)
    "SystemPath": "//www//wwwroot//resources//",
    //微信公众号允许上传文件类型
    "MPUploadFileType": "bmp,png,jpeg,jpg,gif,mp3,wma,wav,amr,mp4",
    //微信允许上传文件类型
    "WeChatUploadFileType": "jpg,png,doc,docx,ppt,pptx,xls,xlsx,pdf,txt,rar,zip,csv,amr,mp4",
    //允许图片类型
    "AllowUploadImageType": "jpg,gif,png,bmp,jpeg,tiff,psd,swf,svg,pcx,dxf,wmf,emf,lic,eps,tga",
    //允许上传文件类型
    "AllowUploadFileType": "jpg,gif,png,bmp,jpeg,doc,docx,ppt,pptx,xls,xlsx,pdf,txt,rar,zip,csv",
    "Domain": "http://192.168.0.186",
    "YOZO": {
      "domain": "http://dcsapi.com/",
      "domainKey": "57462250284462899305150"
    },
    //Minio
    "BucketName": "jnpfsoftoss",
    //文件存储类型(本地:local,MinIo:minio,阿里云:aliyun-oss,腾讯云:tencent-cos)
    "FileStoreType": "local",
    //================== 系统错误邮件报告反馈相关 ============================== -->
    //软件的错误报告
    "ErrorReport": "false",
    //软件的错误报告发给谁
    "ErrorReportTo": "yinmaisoft@163.com"
  }
```

附录

在线预览字体乱码

启动脚本添加

```
-Dfile.encoding=UTF-8
```

注：命令窗口不可关闭

开发指南

流程节点事件接口配置

1.准备接口

```
[HttpGet("")]
[AllowAnonymous]
0 个引用
public async Task<dynamic> test([FromQuery] TestQuery input)
{
    try
    {
        Console.Write("已进入事件方法中");
        var taskId = input.taskId;
        var taskNodeId = input.taskId;
        var userId = input.userId;
        //执行你要完成逻辑


        return null;
    }
    catch (global::System.Exception ex)
    {
        throw;
    }
}
```

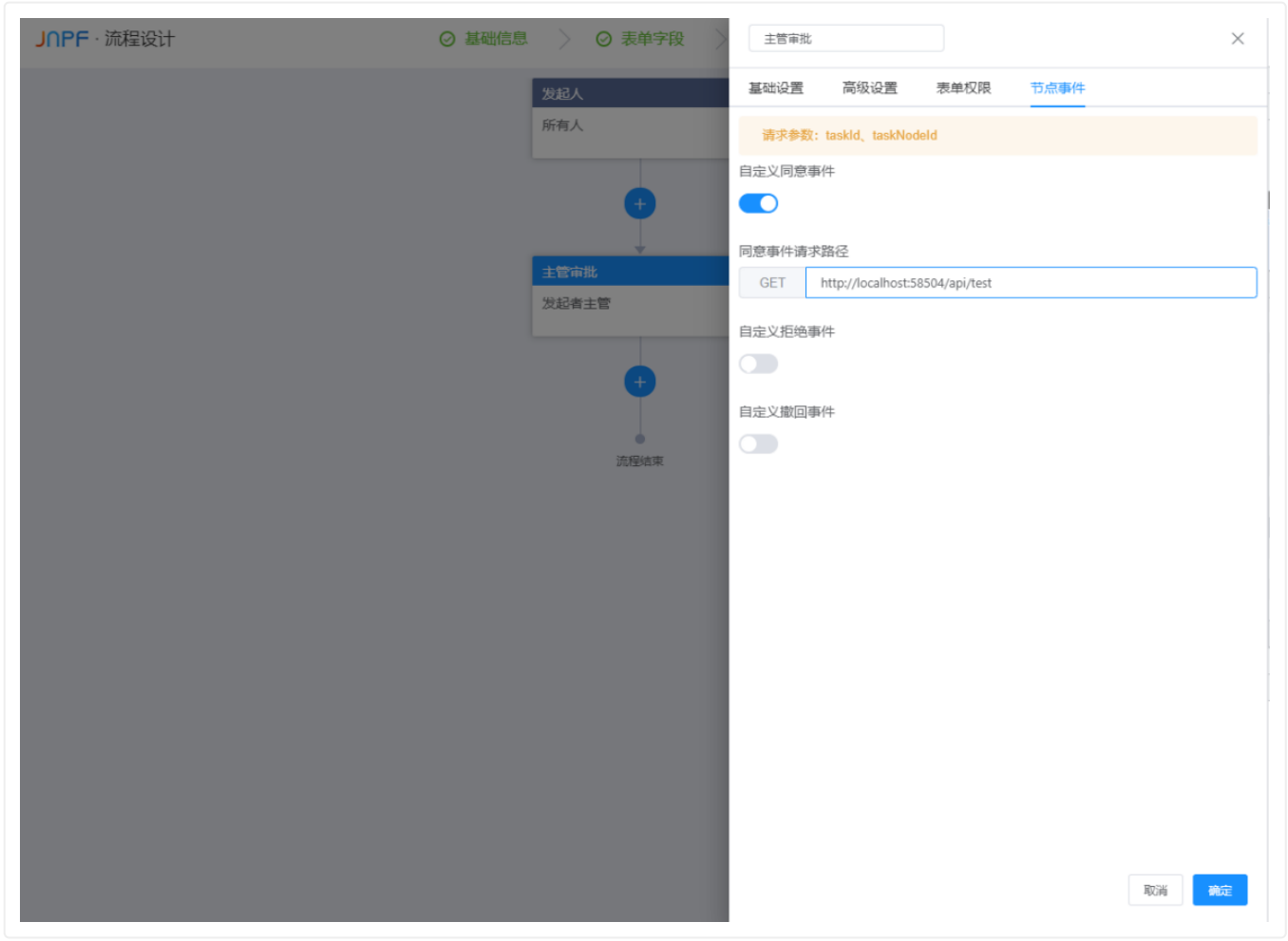
- 接口入参配置

```
1 个引用
public class TestQuery
{
    2 个引用
    public string taskId { get; set; }

    0 个引用
    public string taskNodeId { get; set; }
    //后续添加的参数可在这里继续补充
    //如添加一个用户Id参数
    0 个引用
    public string  userId { get; set; }
}
```

2.配置流程设计

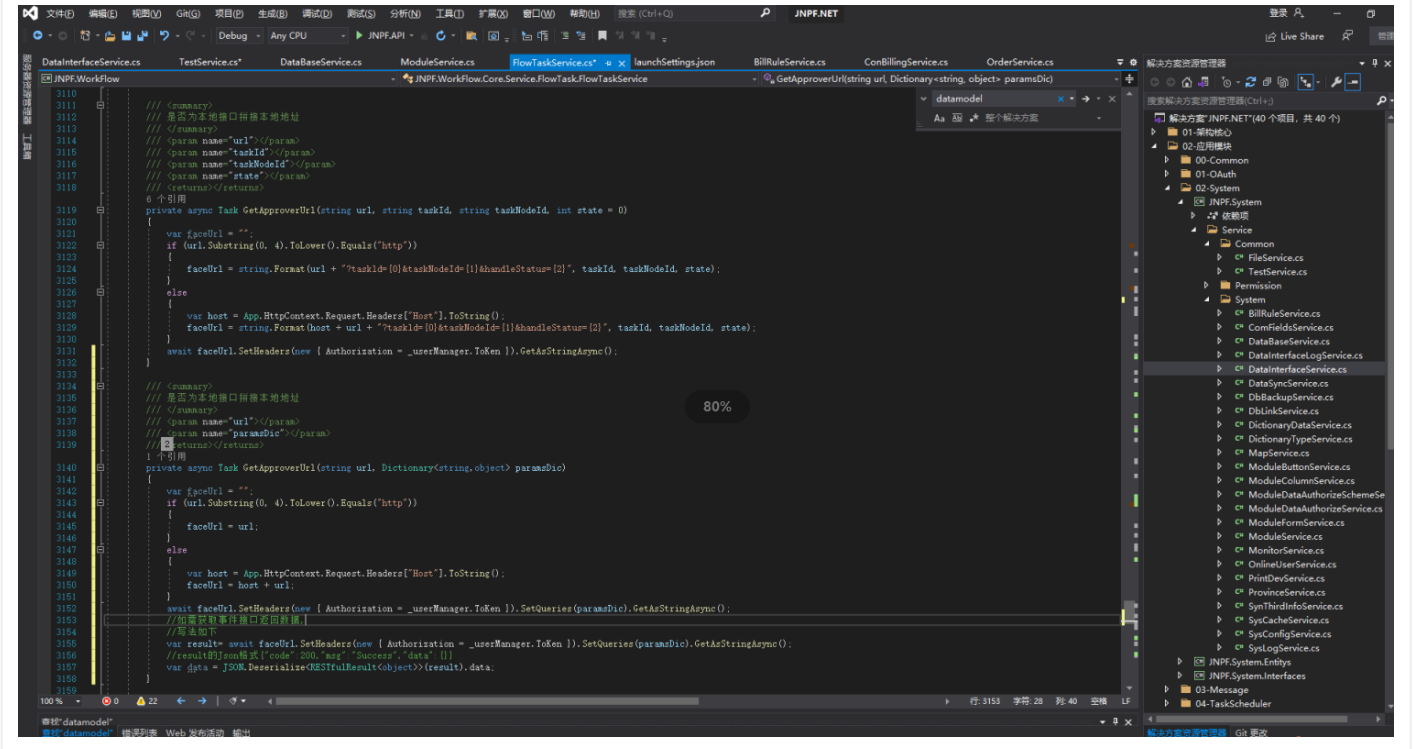
- 先设置流程
- 每个节点都可以设置节点事件
- 开始事件和结束事件在发起节点设置
- 写好全路径的请求

- taskId和taskNodeId后端会拼接上去



3.事件处理的代码

- 可以自己修改事件的参数



- 第二个方法为优化后的方法，可根据具体情况使用

4.其他事件

参考审批事件

流程服务接口配置

1.准备接口

```
[HttpGet("")]
[AllowAnonymous]
0 个引用
public async Task<dynamic> test([FromQuery] TestQuery input)
{
    try
    {
        Console.Write("已进入服务方法中");
        var taskId = input.taskId;
        var taskNodeId = input.taskId;
        var userId = input.userId;
        //执行你要完成逻辑
        var handleIdList = new List<string>();//审批人员的用户id
        handleIdList.Add("admin");
        var handleIds = string.Join(",", handleIdList);
        return new { handleId= handleIds };
    }
    catch (global::System.Exception ex)
    {
        throw;
    }
}
```

- 接口入参配置

```
1 个引用
public class TestQuery
{
    2 个引用
    public string taskId { get; set; }

    0 个引用
    public string taskNodeId { get; set; }
    //后续添加的参数可在这里继续补充
    //如添加一个用户Id参数
    0 个引用
    public string  userId { get; set; }
}
```

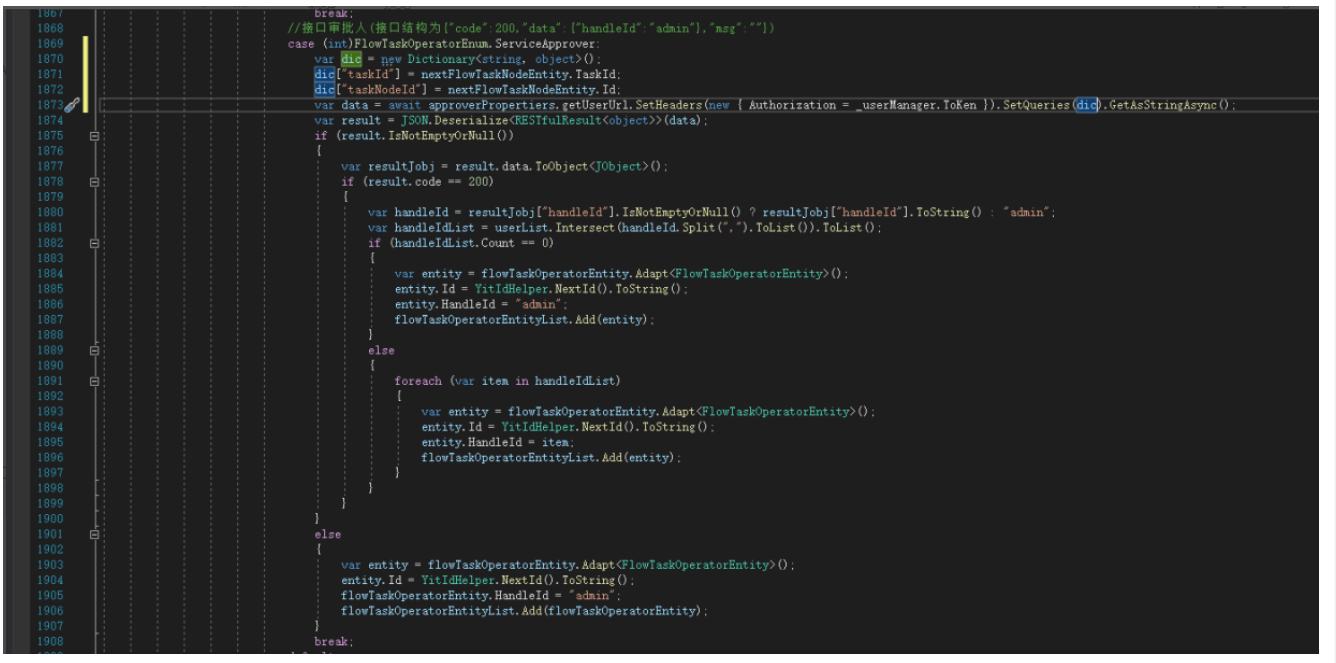
2.配置流程设计

- （1）点击基础设置，配置服务的审批设置

- （2）写好全路径的请求

3.获取服务数据

- 1.服务接收的数据格式：{"code":"200","data":{"handleId":"admin"}}
- 2.多人可以在handleId的参数用逗号形式传过来
- 3.handleId的值只认base_user的f_id值



```
                    break;
                //接口审批人(接口结构为{"code":200,"data":{"handleId":"admin"},"msg":""})
                case (int)FlowTaskOperatorEnum.ServiceApprover:
                    var dic = new Dictionary<string, object>();
                    dic["taskId"] = nextFlowTaskNodeEntity.TaskId;
                    dic["taskNodeId"] = nextFlowTaskNodeEntity.Id;
                    var data = await approverProperties.getUserUrl.SetHeaders(new { Authorization = _userManager.Token }).SetQueries(dic).GetAsStringAsync();
                    var result = JSON.Deserialize<RESTfulResult<object>>(data);
                    if (result.IsNotEmptyOrNull())
                    {
                        var resultJobj = result.data.ToObject<JObject>();
                        if (result.code == 200)
                        {
                            var handleId = resultJobj["handleId"].IsNotEmptyOrNull() ? resultJobj["handleId"].ToString() : "admin";
                            var handleIdList = userList.Intersect(handleId.Split(",").ToList()).ToList();
                            if (handleIdList.Count == 0)
                            {
                                var entity = flowTaskOperatorEntity.Adapt<FlowTaskOperatorEntity>();
                                entity.Id = YitIdHelper.NextId().ToString();
                                entity.HandleId = "admin";
                                flowTaskOperatorEntityList.Add(entity);
                            }
                            else
                            {
                                foreach (var item in handleIdList)
                                {
                                    var entity = flowTaskOperatorEntity.Adapt<FlowTaskOperatorEntity>();
                                    entity.Id = YitIdHelper.NextId().ToString();
                                    entity.HandleId = item;
                                    flowTaskOperatorEntityList.Add(entity);
                                }
                            }
                        }
                    }
                    else
                    {
                        var entity = flowTaskOperatorEntity.Adapt<FlowTaskOperatorEntity>();
                        entity.Id = YitIdHelper.NextId().ToString();
                        flowTaskOperatorEntity.HandleId = "admin";
                        flowTaskOperatorEntityList.Add(flowTaskOperatorEntity);
                    }
                    break;
```

jnpf-netcore-cloud

快速开始

开发环境运行

本地环境准备

项目说明

> 以下为 `JNPF 3.3.x` .NET项目命名规则

项目名	说明
jnpf-netcore-cloud	.NET微服务项目
jnpf-datareport	报表项目(java)
jnpf-file-preview	本地文档预览项目(java)

环境要求

基础要求

> 环境具体安装教程

环境	版本	描述
Visual Studio 2019	16.8.x	.NetCore开发工具及开发环境(Windows)
Visual Studio for Mac	默认最新版本	.NetCore开发工具及开发环境(MacOS)
IntelliJ IDEA	2020.1 +	Java开发工具及开发环境(MacOS)
JDK	1.8.x	JAVA环境依赖(需配置环境变量)

SDK	最新版	.NET环境依赖
Maven	3.6.3	项目构建(需配置环境变量)
Redis	3.2.100(Windows)/6.0.x(Linux,Mac)	
MySQL	5.7.x+	数据库任选一
SQLServer	2012+	数据库任选一
Orale	11g+	数据库任选一

推荐IDE

> 特别说明：IntelliJ IDEA 2019.1 和 Maven 3.6.3 存在不兼容问题

- Visual Studio 2019 16.8.4 + (Windows)
- Visual Studio for Mac (MacOS)
- IntelliJ IDEA 2020.1 +

推荐插件

- `Lombok`
- `MyBatisPlus`
- `Alibaba Java Coding Guidelines`

JDK 1.8.x

> 报表开发环境

特别说明

日前发现，`jdk1.8.0_25`版本无法正常运行项目。

下载并安装

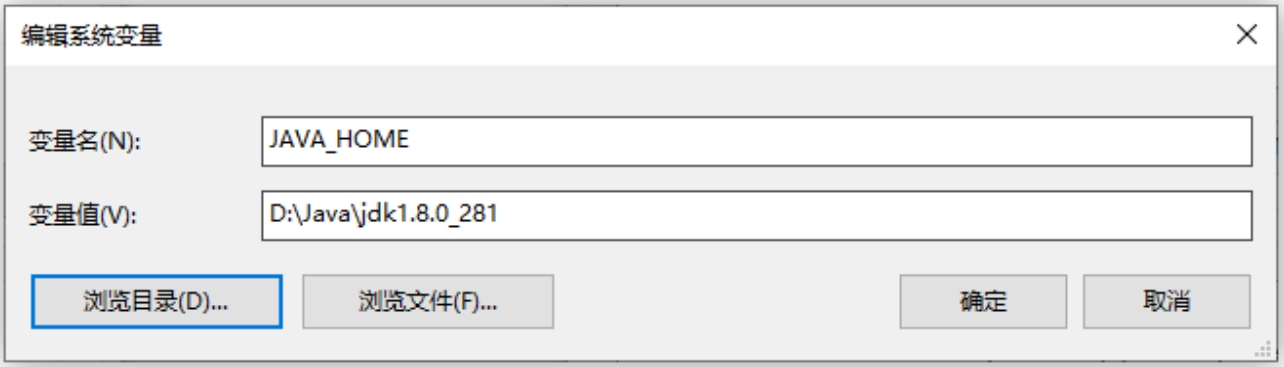
打开`https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html`,选择`Windows x64`版本下载并安装

Solaris SPARC 64-bit	88.77 MB	jdk-8u281-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	134.68 MB	jdk-8u281-solaris-x64.tar.Z
Solaris x64	92.66 MB	jdk-8u281-solaris-x64.tar.gz
Windows x86	154.69 MB	jdk-8u281-windows-i586.exe
Windows x64	166.97 MB	jdk-8u281-windows-x64.exe

环境变量配置

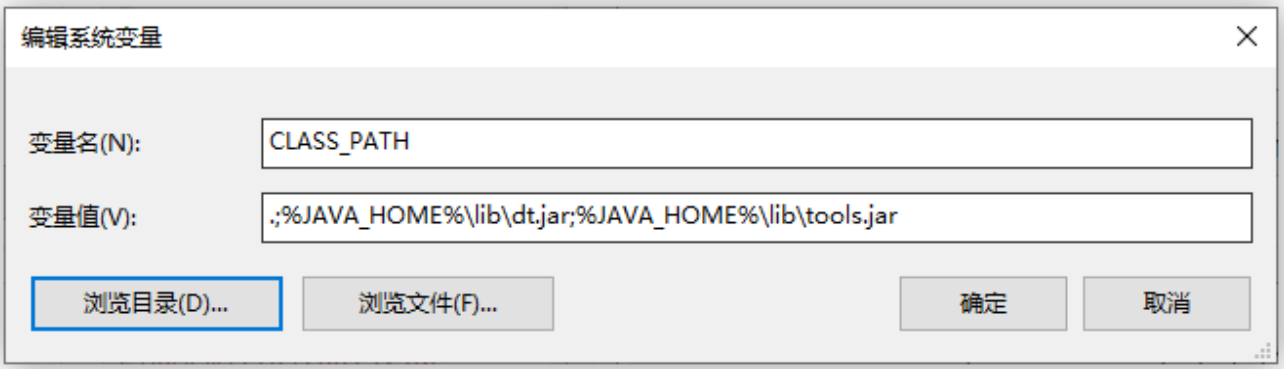
1）新建系统变量`JAVA_HOME`

- 变量名：`JAVA_HOME`
- 变量值：`D:\Java\jdk1.8.0_281`

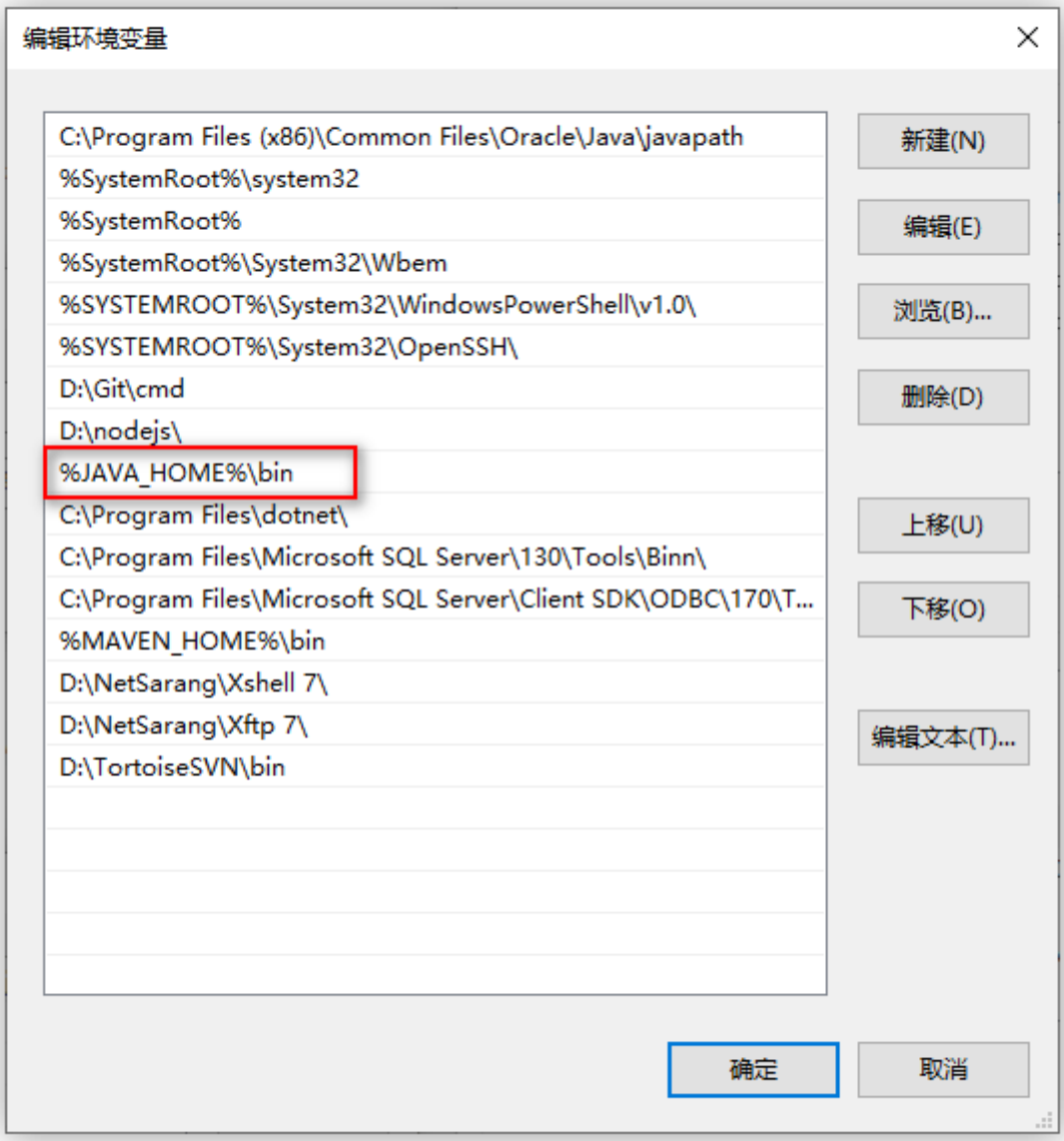


2）新建系统变量`CLASS_PATH`

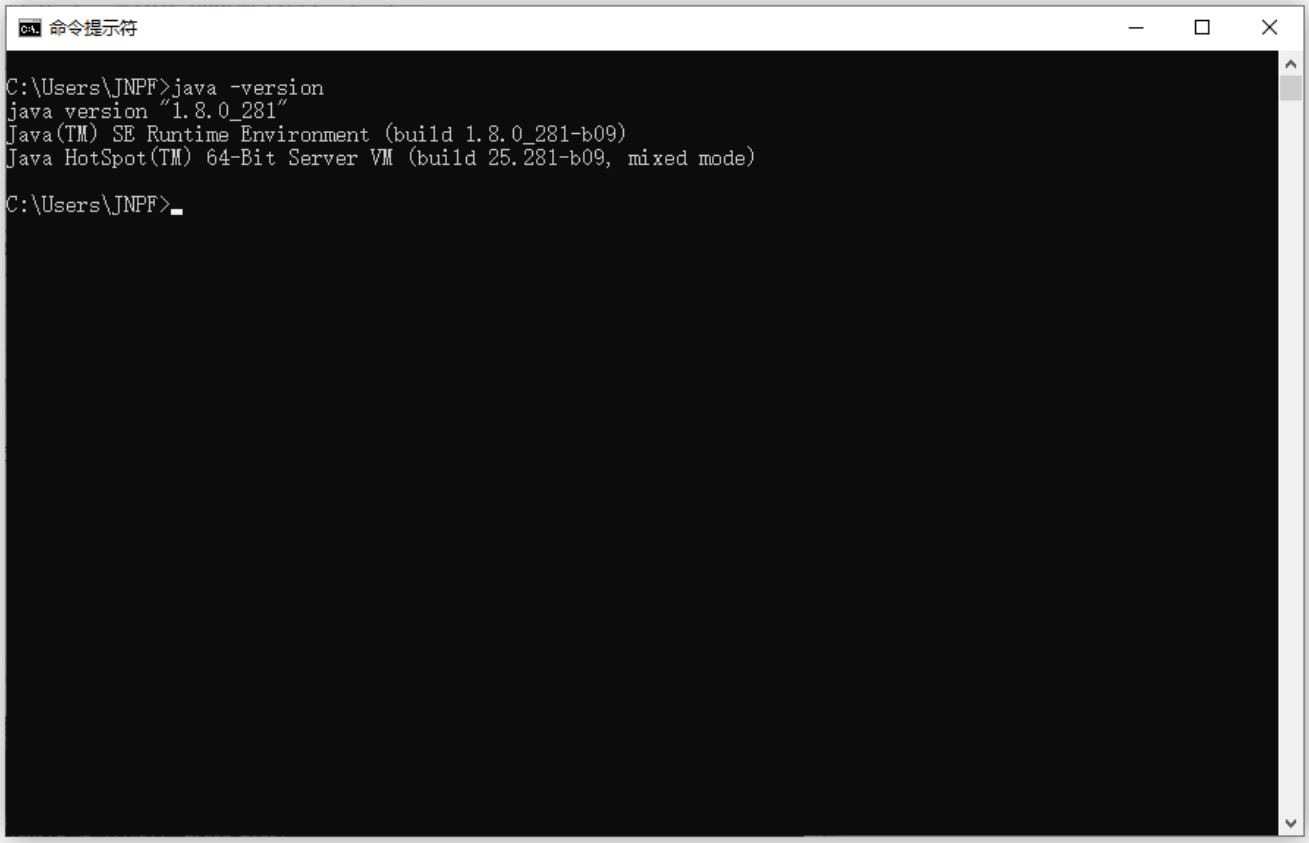
- 变量名：`CLASS_PATH`
- 变量值：`.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar`



3）双击 `系统变量` 中的 `path` ，点击新建，输入 `%JAVA_HOME%\bin`



4）命令行输入 `java -version` 检查JDK环境是否配置成功

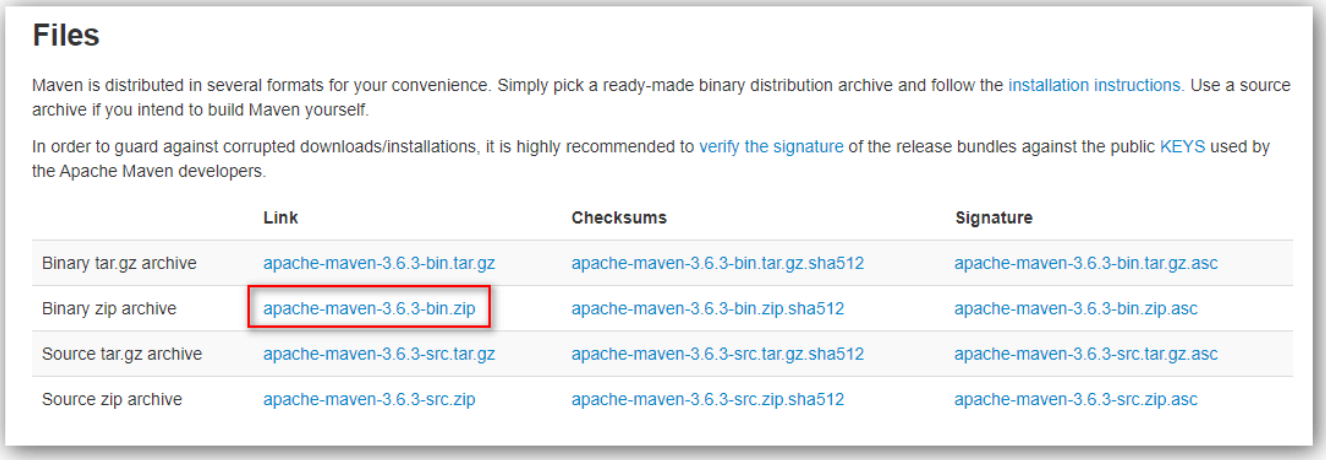


Maven 3.6.3

> JAVA版本开发环境

下载并解压

打开 `http://maven.apache.org/download.cgi` ，下载 `apache-maven-3.6.3-bin.zip` 并解压



Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the installation instructions. Use a source archive if you intend to build Maven yourself.

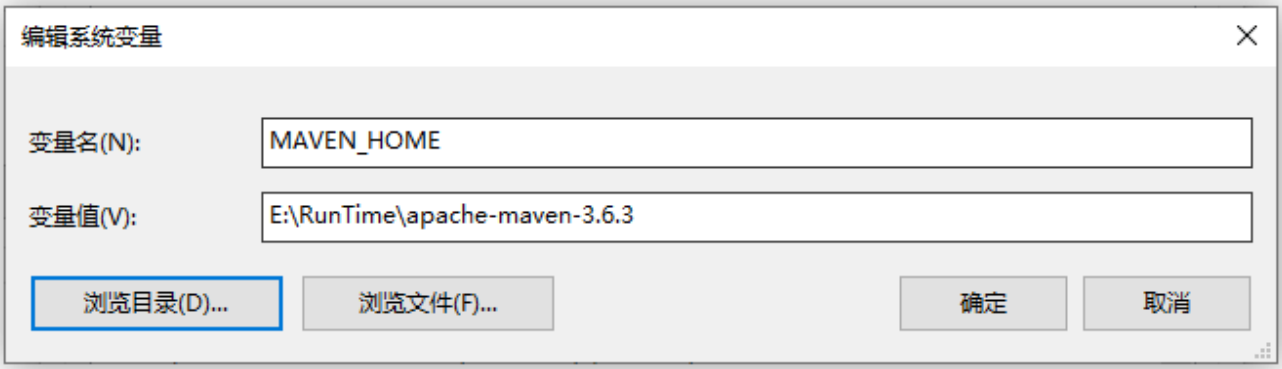
In order to guard against corrupted downloads/installations, it is highly recommended to verify the signature of the release bundles against the public KEYS used by the Apache Maven developers.

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.6.3-bin.tar.gz	apache-maven-3.6.3-bin.tar.gz.sha512	apache-maven-3.6.3-bin.tar.gz.asc
Binary zip archive	apache-maven-3.6.3-bin.zip	apache-maven-3.6.3-bin.zip.sha512	apache-maven-3.6.3-bin.zip.asc
Source tar.gz archive	apache-maven-3.6.3-src.tar.gz	apache-maven-3.6.3-src.tar.gz.sha512	apache-maven-3.6.3-src.tar.gz.asc
Source zip archive	apache-maven-3.6.3-src.zip	apache-maven-3.6.3-src.zip.sha512	apache-maven-3.6.3-src.zip.asc

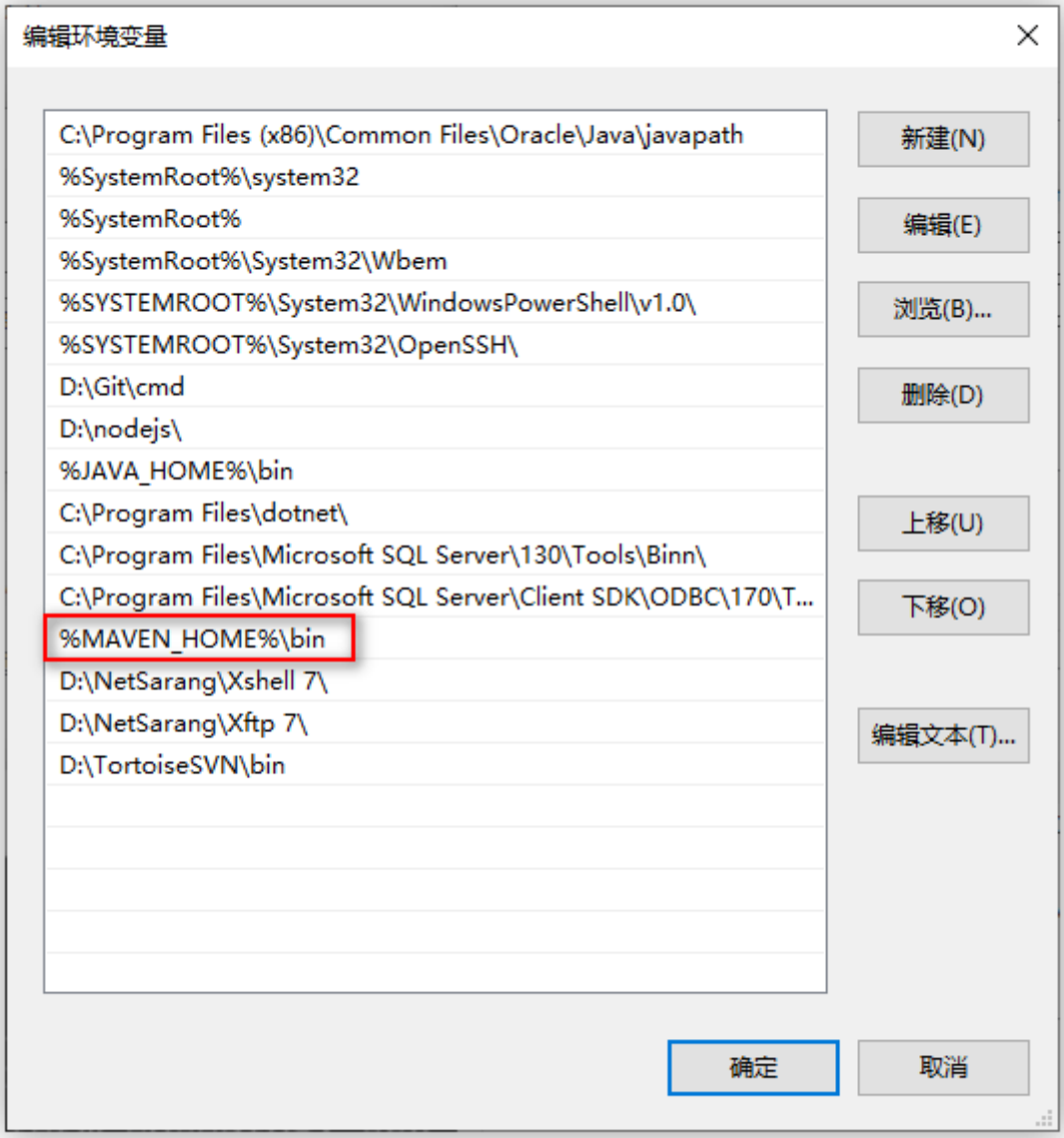
环境变量配置

1）新建系统变量`MAVEN_HOME`

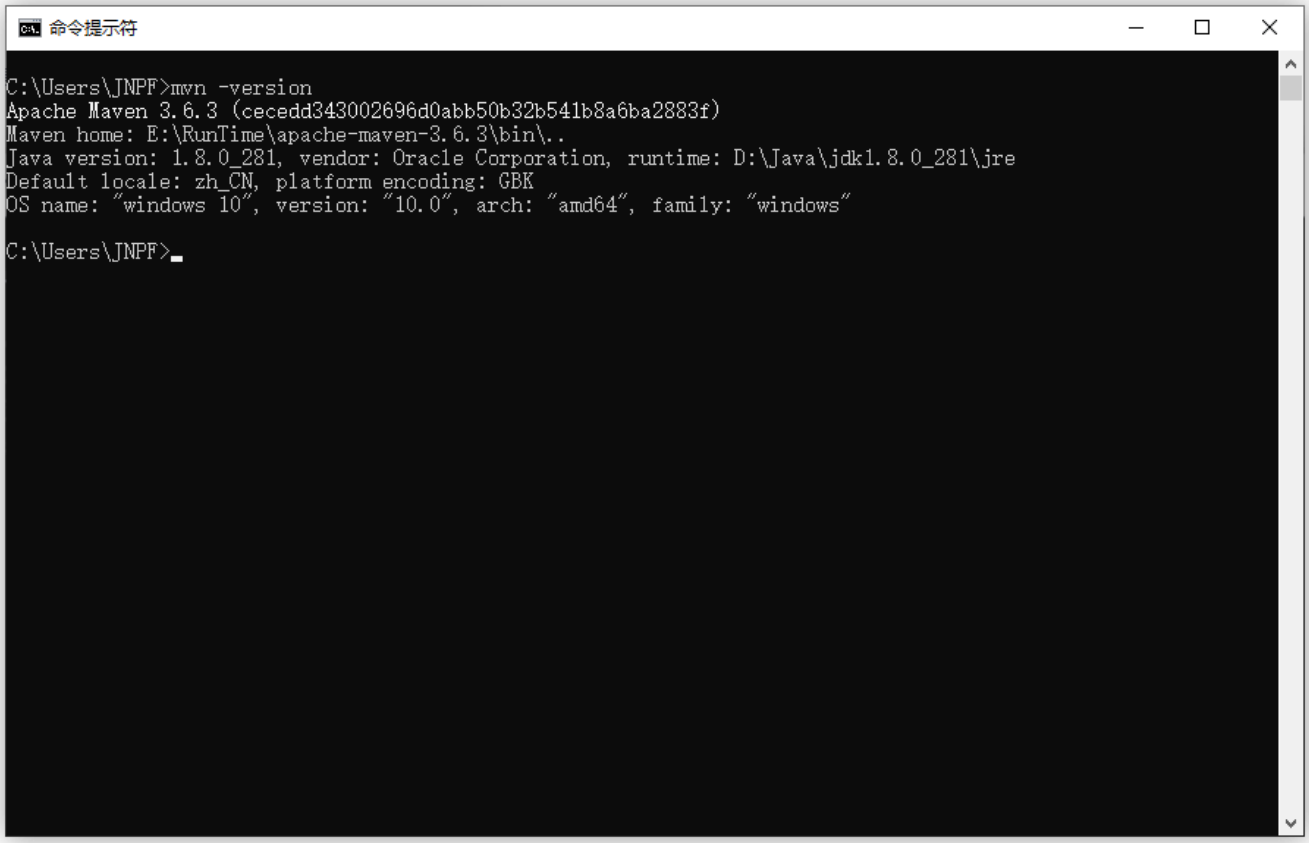
- 变量名：`MAVEN_HOME`
- 变量值：`D:\apache-maven-3.6.3`（`maven`路径）



2）双击`系统变量`中的`path`，点击新建，输入`%MAVEN_HOME%\bin`



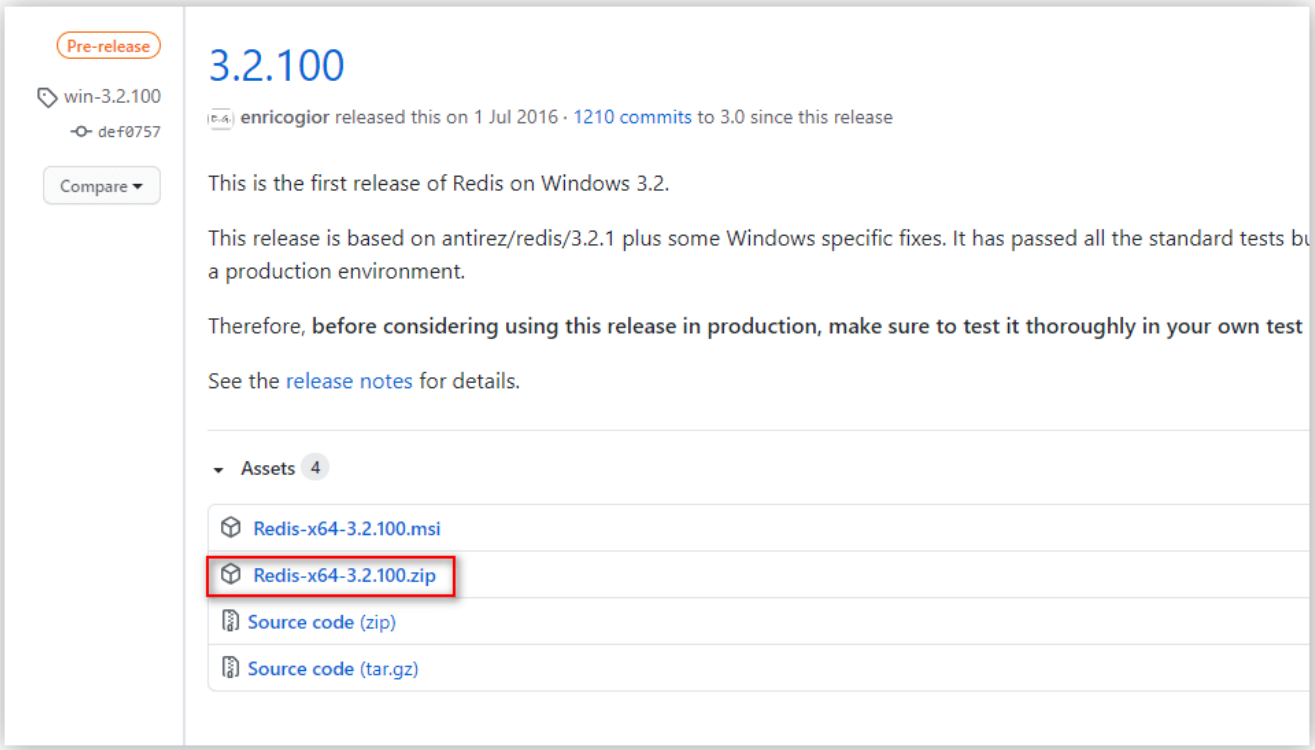
3）命令行输入`mvn -version`检查Maven环境是否配置成功



Redis 3.2.100

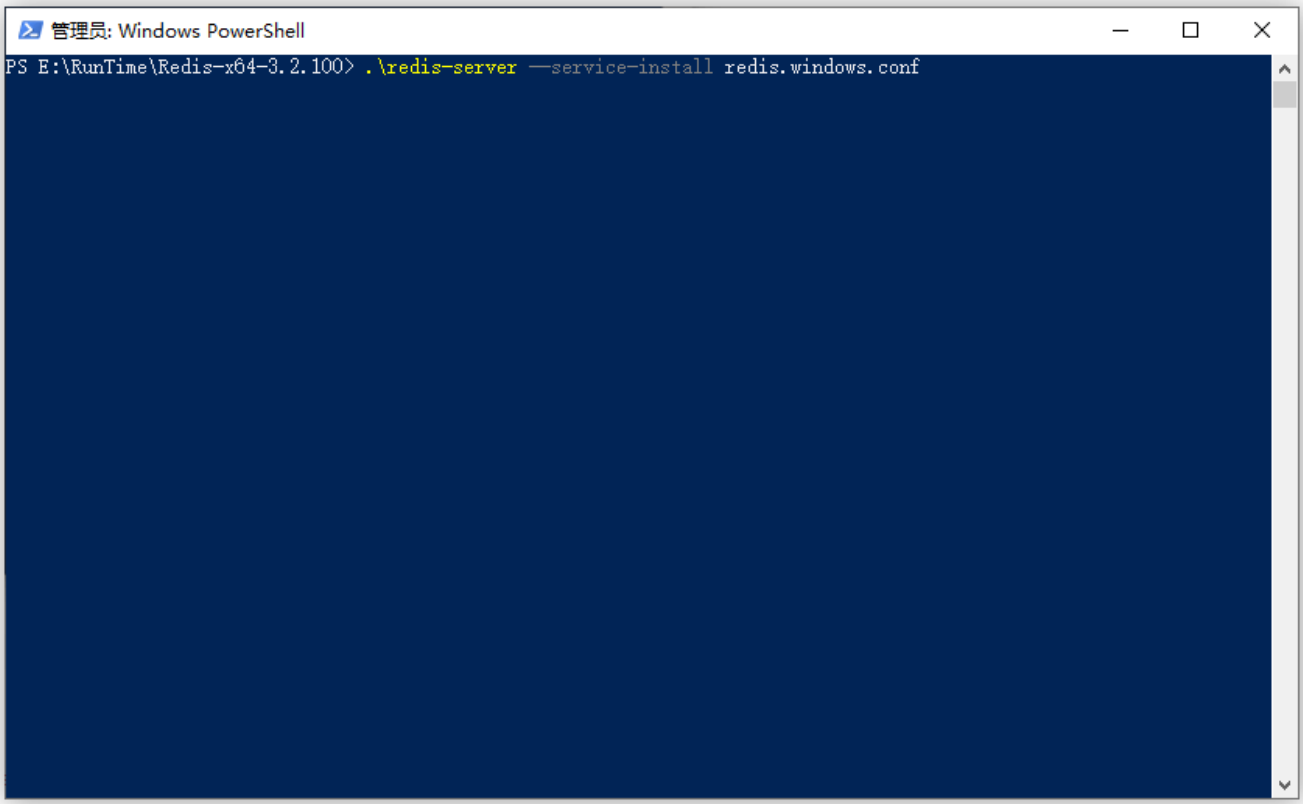
下载并解压

打开 `https://github.com/microsoftarchive/redis/releases` ，下载 `3.2.100` 版本

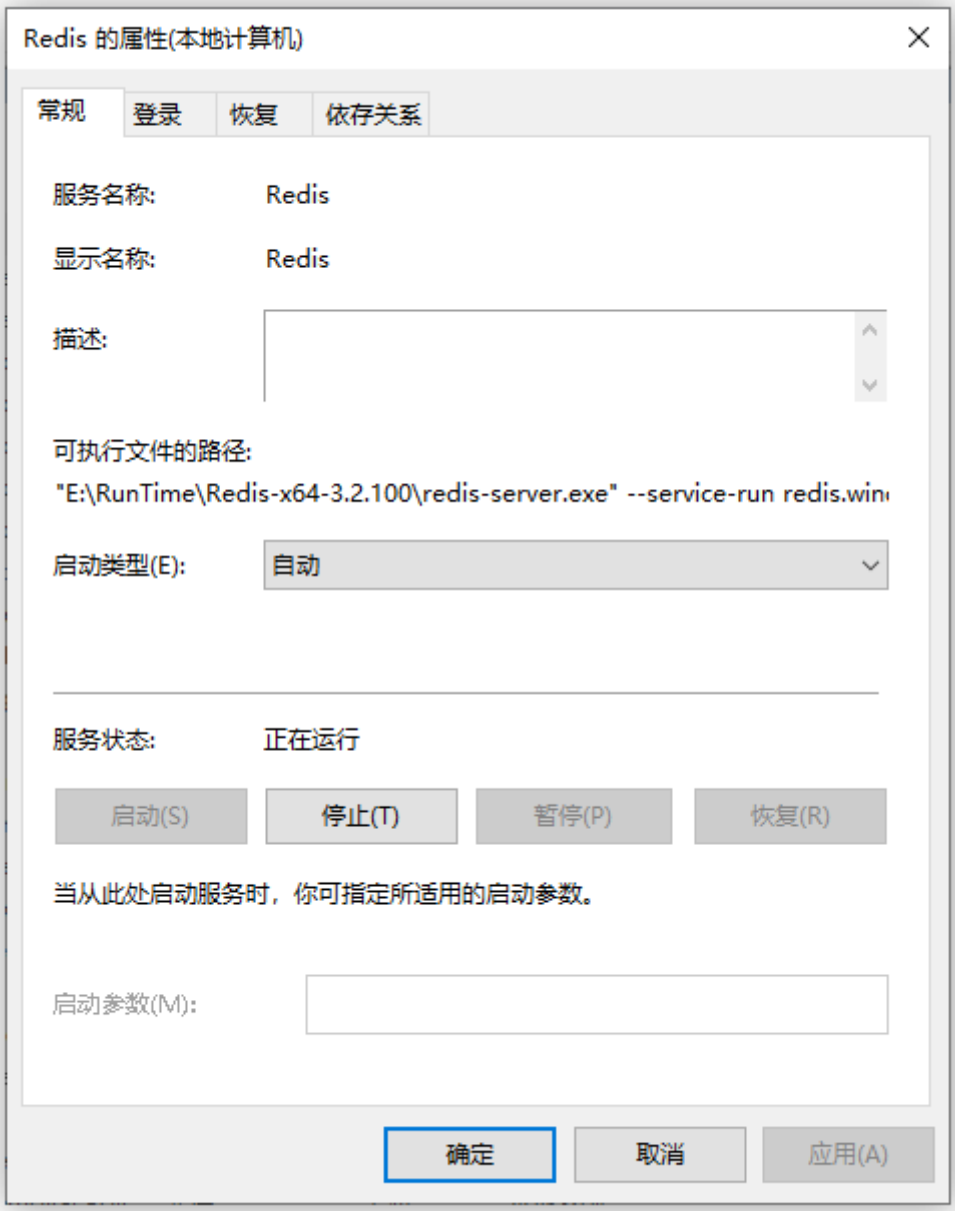


配置并设置为服务

1） 打开命令行(`Windows PowerShell`)，输入 `.\redis-server --service-install redis.windows.conf`



2） 右击 `电脑` - `管理` - `服务和应用程序` - `服务` 启动服务

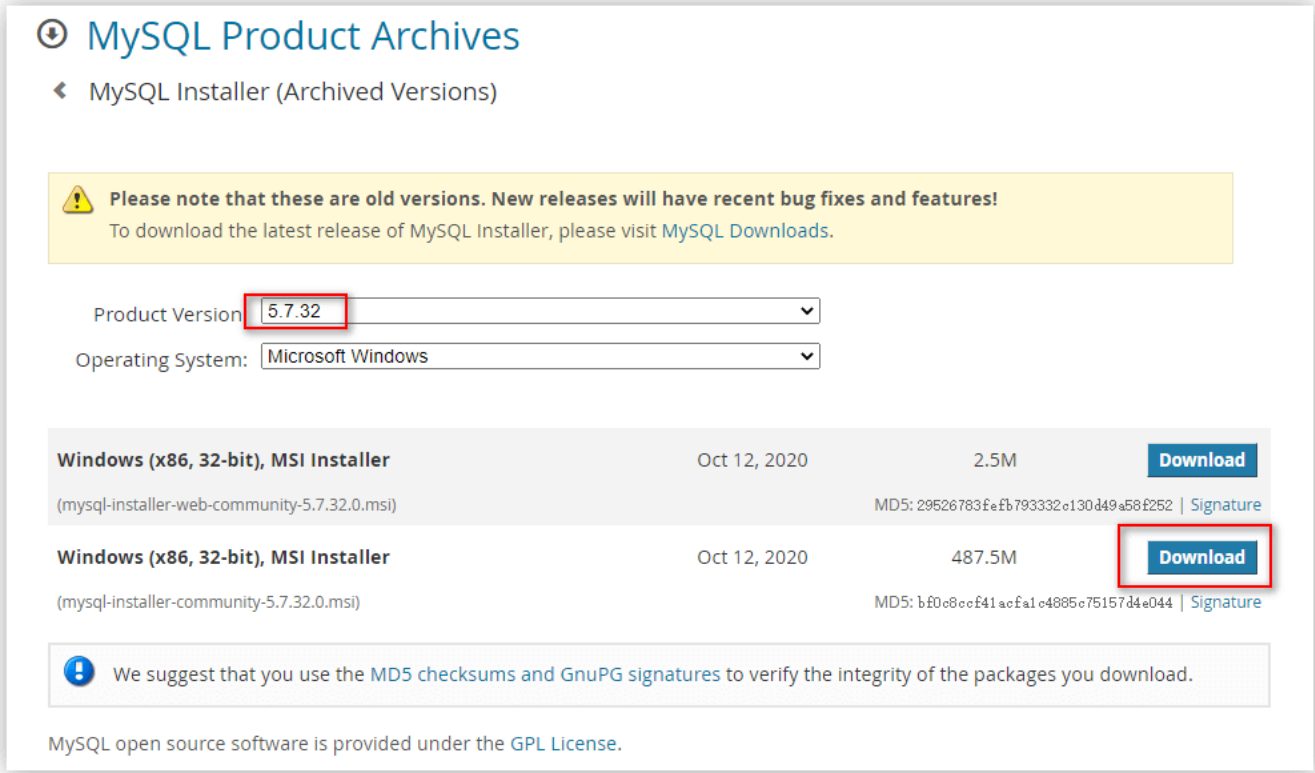


3） `Redis` 常用的指令

- 卸载服务：`redis-server --service-uninstall`
- 开启服务：`redis-server --service-start`
- 停止服务：`redis-server --service-stop`

MySQL 5.7.x

打开 `https://downloads.mysql.com/archives/installer/` ,下载 `MySQL5.7.x` ,按提示安装即可



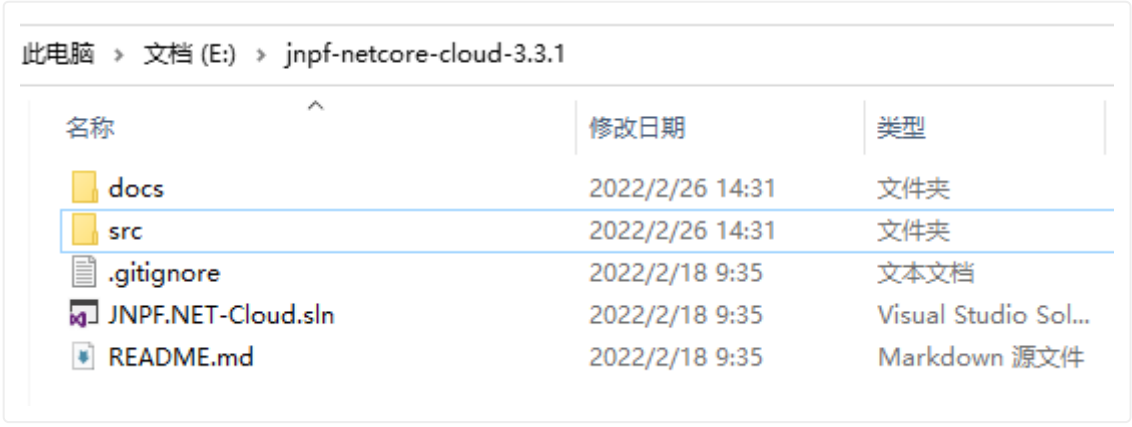
SQLServer 2012

下载 `SQL Server 2012 Developer` （全功能免费版本，许可在非生产环境下用作开发和测试数据库），按提示安装即

本地项目运行

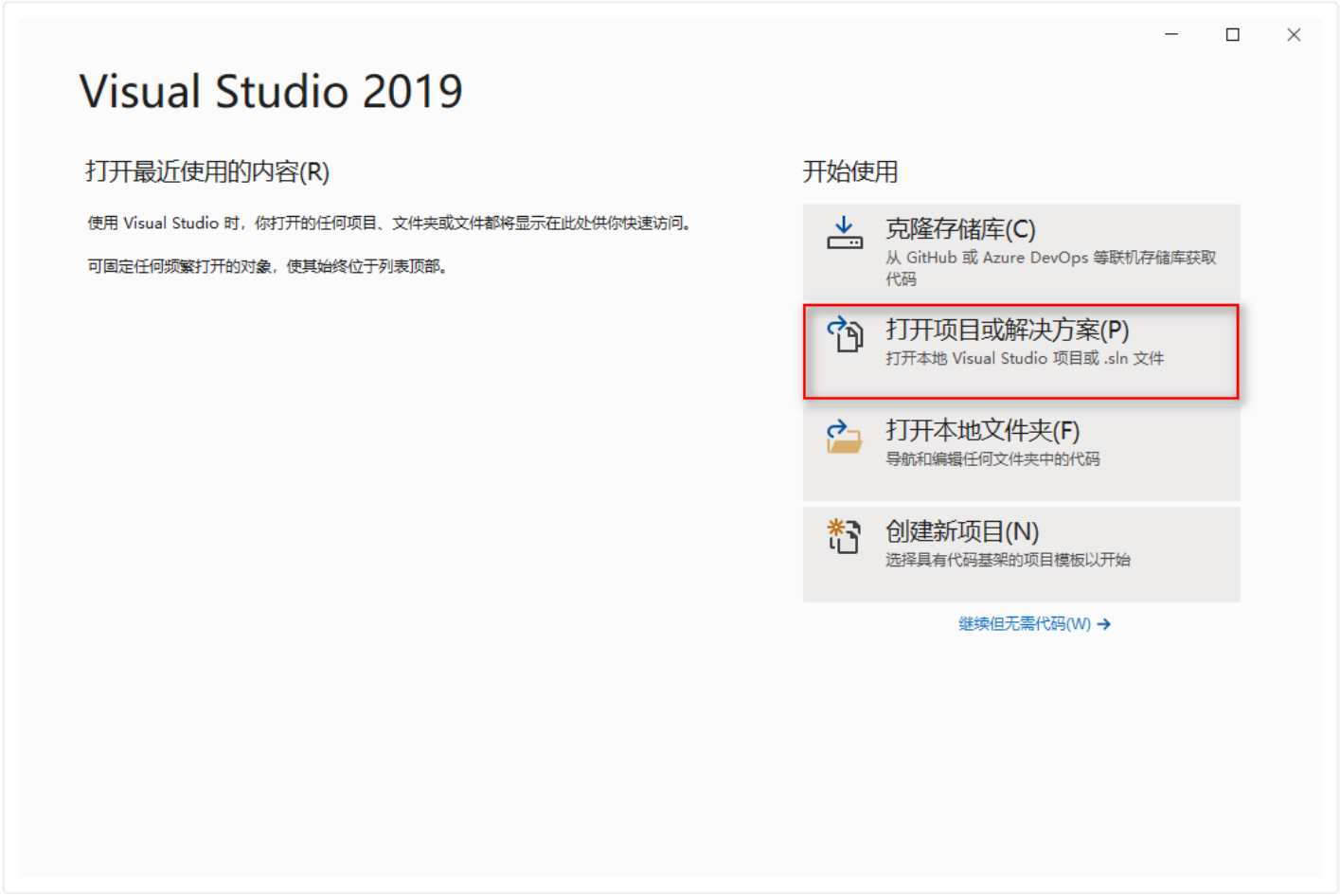
环境准备

> 参考 `.NET Cloud` -环境准备
>
> ## 项目导入
>
> 方式一：直接双击项目根目录下的 `JNPF.NET-Cloud.sln` 文件

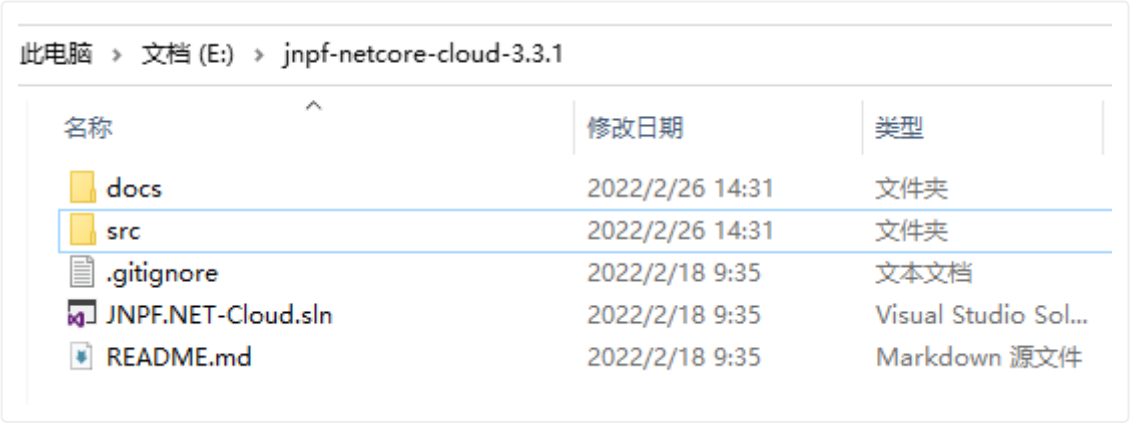


方式二：

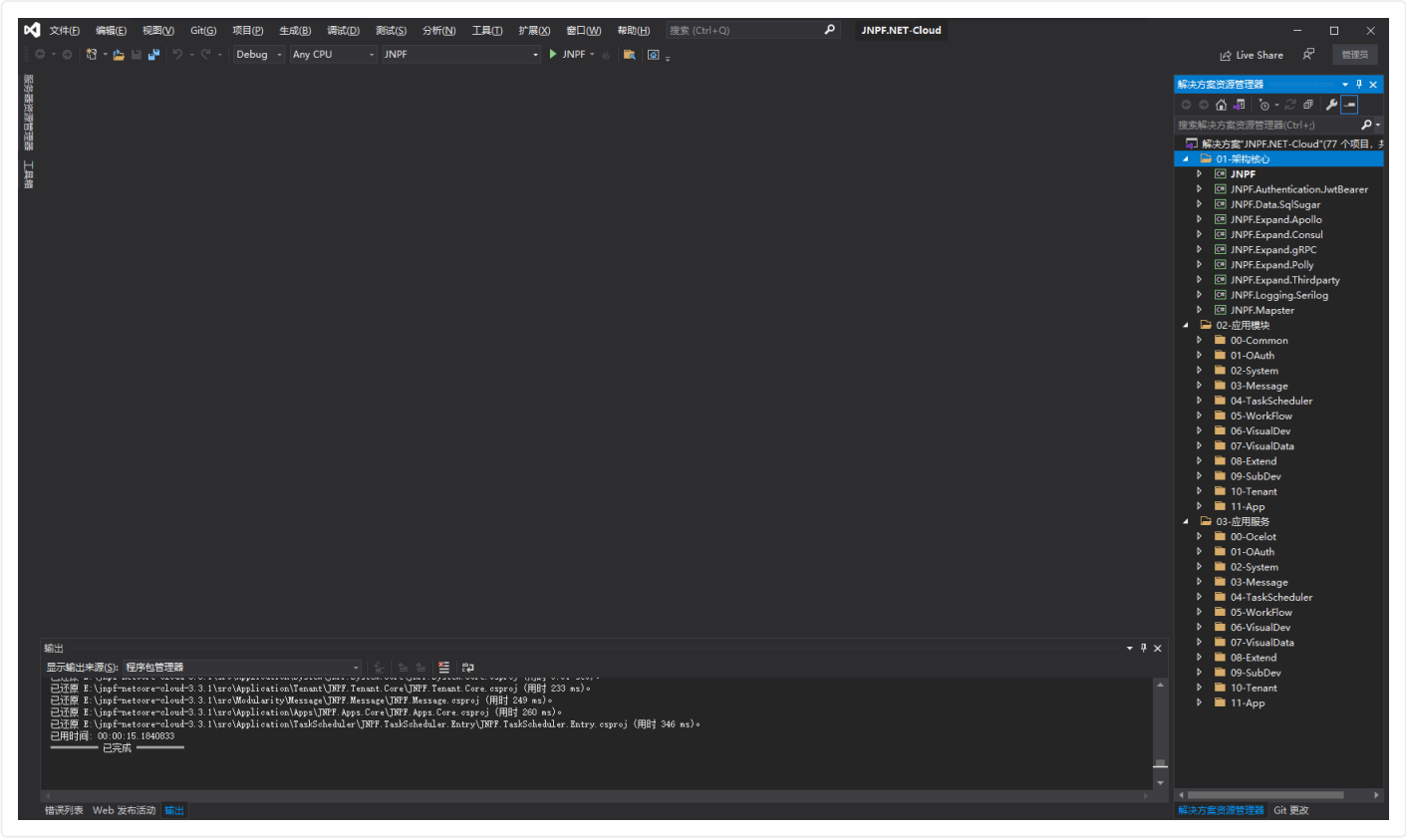
- 打开 `Visual Studio 2019` ,打开项目或解决方案，如下图



- 选择项目根目录下的 `JNPF.NET-Cloud.sln` 文件

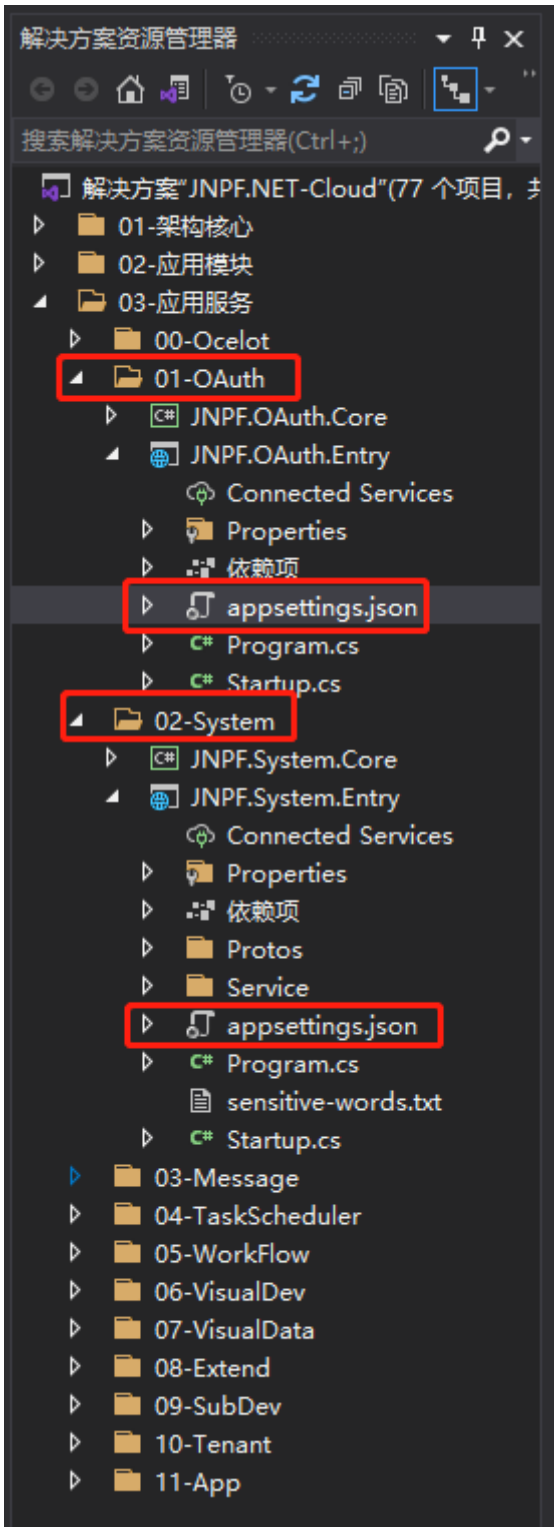


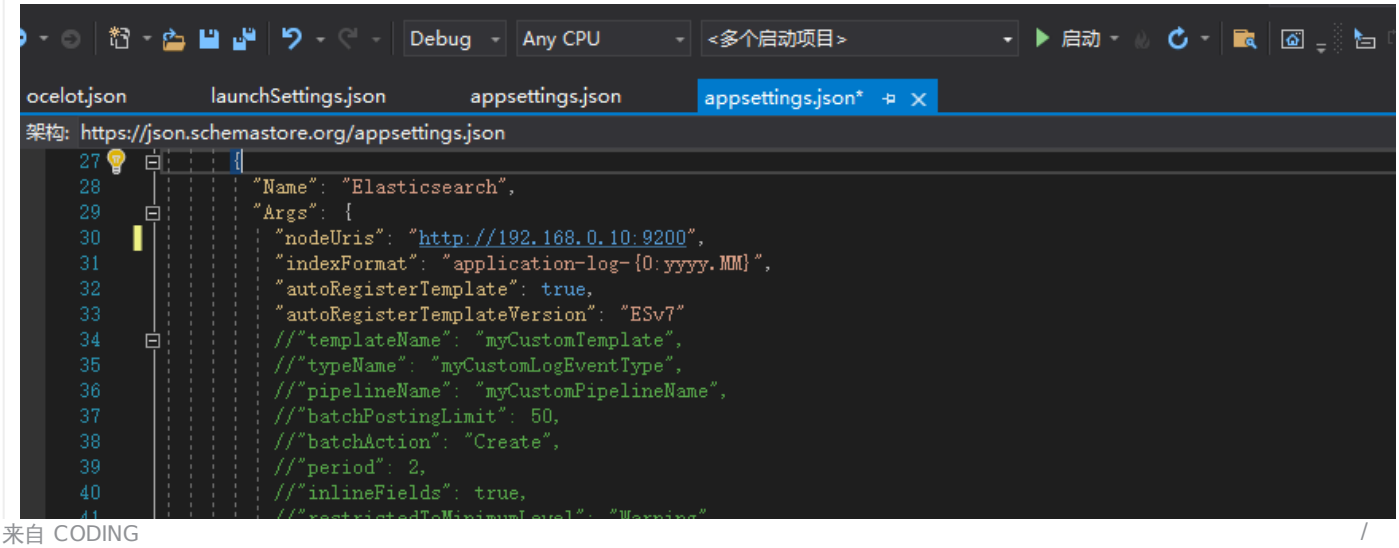
- 打开后，如下图界面



项目运行

1. 打开每个服务的 `appsettings.json` 修改相关配置


解决方案资源管理器
搜索解决方案资源管理器(Ctrl+;)
解决方案"JNPF.NET-Cloud"(77 个项目，共
01-架构核心
02-应用模块
03-应用服务
00-Ocelot
01-OAuth
JNPF.OAuth.Core
JNPF.OAuth.Entry
Connected Services
Properties
依赖项
appsettings.json
Program.cs
Startup.cs
02-System
JNPF.System.Core
JNPF.System.Entry
Connected Services
Properties
依赖项
Protos
Service
appsettings.json
Program.cs
sensitive-words.txt
Startup.cs
03-Message
04-TaskScheduler
05-WorkFlow
06-VisualDev
07-VisualData
08-Extend
09-SubDev
10-Tenant
11-App


Debug
Any CPU
<多个启动项目>
启动
ocelot.json
launchSettings.json
appsettings.json
appsettings.json*
架构: https://json.schemastore.org/appsettings.json
{
"Name": "Elasticsearch",
"Args": {
"nodeUris": "http://192.168.0.10:9200",
"indexFormat": "application-log-{0:yyyy.MM}",
"autoRegisterTemplate": true,
"autoRegisterTemplateVersion": "ESv7"
//"templateName": "myCustomTemplate",
//"typeName": "myCustomLogEventType",
//"pipelineName": "myCustomPipelineName",
//"batchPostingLimit": 50,
//"batchAction": "Create",
//"period": 2,
//"inlineFields": true,

```
//"bufferBaseFilename": "C:/Temp/docker-elk-serilog-web-buffer",
//"bufferFileSizeLimitBytes": 5242880,
//"bufferLogShippingInterval": 5000,
//"bufferRetainedInvalidPayloadsLimitBytes": 5000,
//"bufferFileCountLimit": 31,
//"connectionGlobalHeaders": "Authorization=Bearer SOME-TOKEN;OtherHeader=OTHER-HEADER-VALUE",
//"connectionTimeout": 5,
//"emitEventFailure": "WriteToSelfLog",
//"queueSizeLimit": "100000",
//"overwriteTemplate": false,
//"registerTemplateFailure": "IndexAnyway",
//"deadLetterIndexName": "deadletter-{0:yyyy.MM}",
//"numberOfShards": 20,
//"numberOfReplicas": 10,
//"templateCustomSettings": [ { "index.mapping.total_fields.limit": "10000000" } ],
//"formatProvider": "My.Namespace.MyFormatProvider, My.Assembly.Name",
//"connection": "My.Namespace.MyConnection, My.Assembly.Name",
//"serializer": "My.Namespace.MySerializer, My.Assembly.Name",
//"connectionPool": "My.Namespace.MyConnectionPool, My.Assembly.Name",
//"customFormatter": "My.Namespace.MyCustomFormatter, My.Assembly.Name",
//"customDurableFormatter": "My.Namespace.MyCustomDurableFormatter, My.Assembly.Name",
//"failureSink": "My.Namespace.MyFailureSink, My.Assembly.Name"
}
}
]
```

appsettings.json*

架构: https://json.schemastore.org/appsettings.json

```
},
"Consul": {
  "ServiceName": "OAuth",
  "ServiceIP": "localhost",
  "ServicePort": 5001,
  "ServiceHealthCheck": "https://192.168.0.10:5001/HealthCheck",
  "ConsulAddress": "http://127.0.0.1:8500"
},
//Apollo的配置
"Apollo": {
  "AppId": "JNPF.NET.Cloud",
  "Env": "DEV",
  "MetaServer": "http://192.168.0.10:8080",
  "ConfigServer": [ "http://192.168.0.10:8080" ],
  "Namespaces": [ "JNPF.NET", "application.json", "application" ]
}
}
```

每个服务端口不一样

```
"Name": "Elasticsearch",
"Args": {
  "nodeUris": "http://192.168.0.10:9200",
  "indexFormat": "application-log-{0:yyyy.MM}",
  "autoRegisterTemplate": true,
  "autoRegisterTemplateVersion": "ESv7"
  //"templateName": "myCustomTemplate",
  //"typeName": "myCustomLogEventType",
  //"pipelineName": "myCustomPipelineName",
  //"batchPostingLimit": 50,
  //"batchAction": "Create",
  //"period": 2,
  //"inlineFields": true,
  //"restrictedToMinimumLevel": "Warning",
```

```
          //"bufferBaseFilename": "C:/Temp/docker-elk-serilog-web-buffer",
          //"bufferFileSizeLimitBytes": 5242880,
          //"bufferLogShippingInterval": 5000,
          //"bufferRetainedInvalidPayloadsLimitBytes": 5000,
          //"bufferFileCountLimit": 31,
          //"connectionGlobalHeaders": "Authorization=Bearer SOME-TOKEN;OtherHeader=
OTHER-HEADER-VALUE",
          //"connectionTimeout": 5,
          //"emitEventFailure": "WriteToSelfLog",
          //"queueSizeLimit": "100000",
          //"overwriteTemplate": false,
          //"registerTemplateFailure": "IndexAnyway",
          //"deadLetterIndexName": "deadletter-{0:yyyy.MM}",
          //"numberOfShards": 20,
          //"numberOfReplicas": 10,
          //"templateCustomSettings": [ { "index.mapping.total_fields.limit": "10000
000" } ],
          //"formatProvider": "My.Namespace.MyFormatProvider, My.Assembly.Name",
          //"connection": "My.Namespace.MyConnection, My.Assembly.Name",
          //"serializer": "My.Namespace.MySerializer, My.Assembly.Name",
          //"connectionPool": "My.Namespace.MyConnectionPool, My.Assembly.Name",
          //"customFormatter": "My.Namespace.MyCustomFormatter, My.Assembly.Name",
          //"customDurableFormatter": "My.Namespace.MyCustomDurableFormatter, My.Ass
embly.Name",
          //"failureSink": "My.Namespace.MyFailureSink, My.Assembly.Name"
        }
```

```
  "Consul": {
    "ServiceName": "OAuth",
    "ServiceIP": "localhost",
    "ServicePort": 5001,
    "ServiceHealthCheck": "https://192.168.0.10:5001/HealthCheck",
    "ConsulAddress": "http://127.0.0.1:8500"
  },
```

```
  //Apollo的配置
  "Apollo": {
    "AppId": "JNPF.NET.Cloud",
    "Env": "DEV",
    "MetaServer": "http://192.168.0.10:8080",
    "ConfigServer": [ "http://192.168.0.10:8080" ],
    "Namespaces": [ "JNPF.NET", "application.json", "application" ]
  }
```

2.部署Apollo

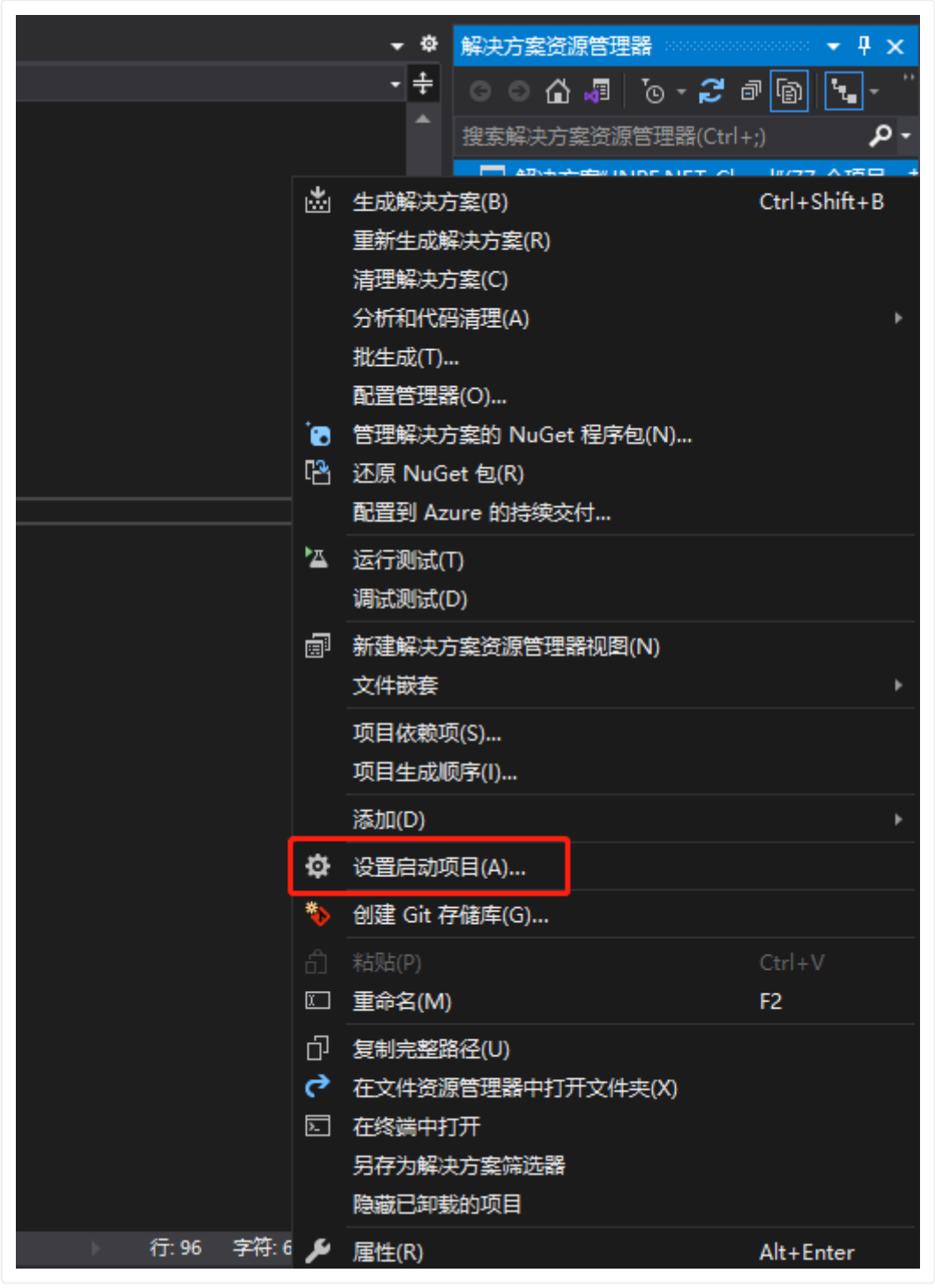
详情请看：服务器环境部署 -> Apollo部署文档

3.部署consul

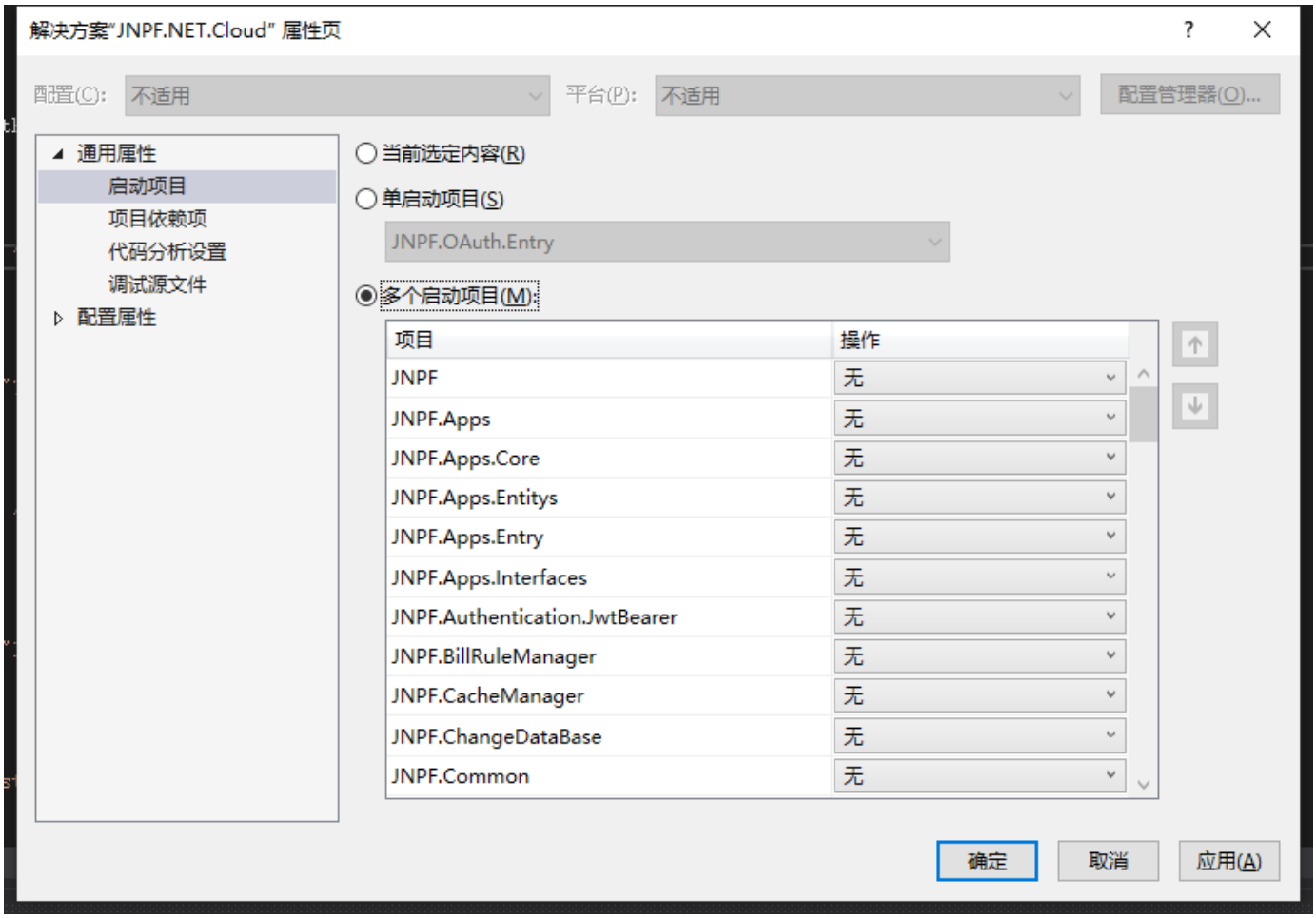
详情请看：服务器环境部署 -> consul部署文档

4.设置启动服务

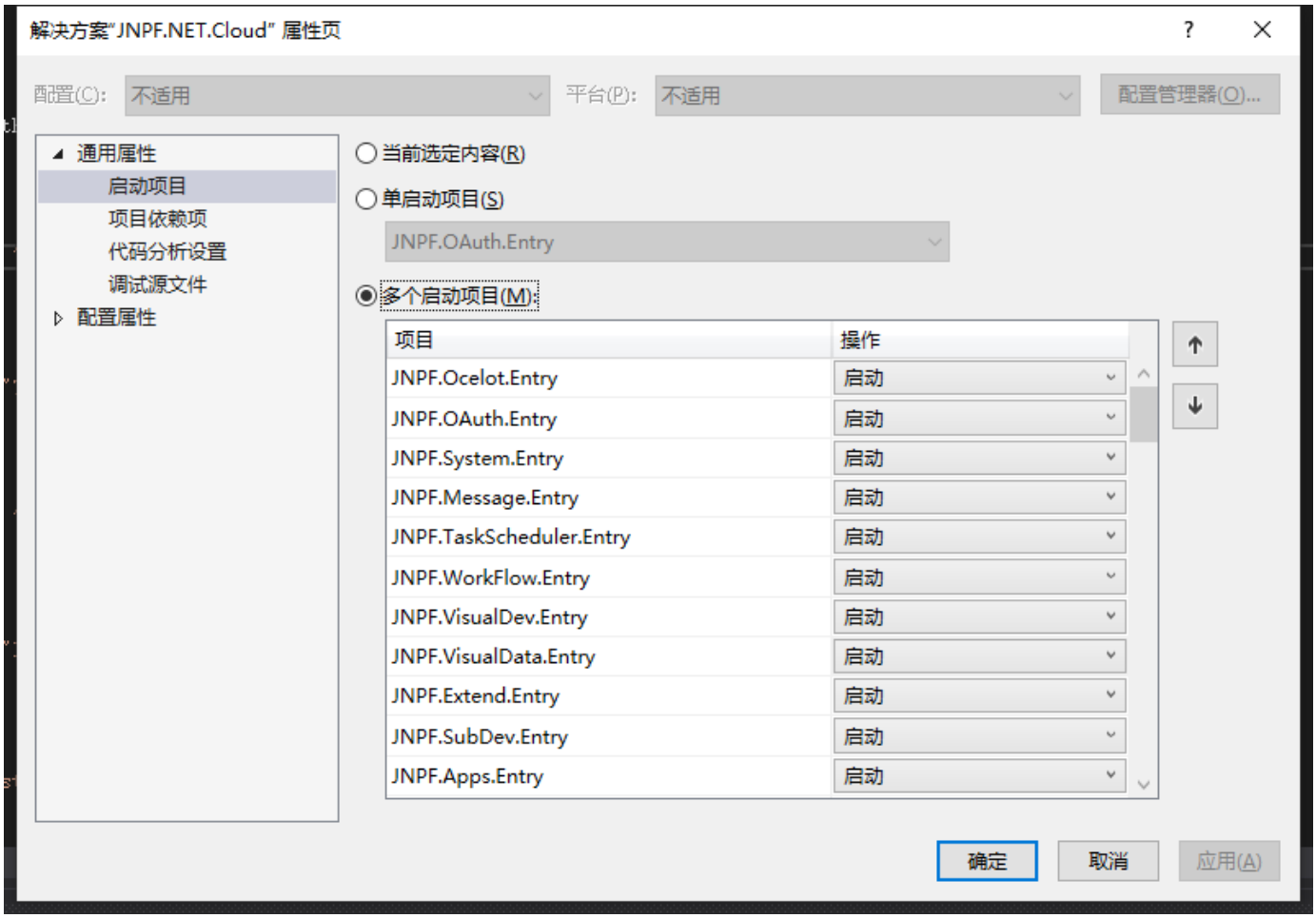
- 右击解决方案,选择设置启动项目



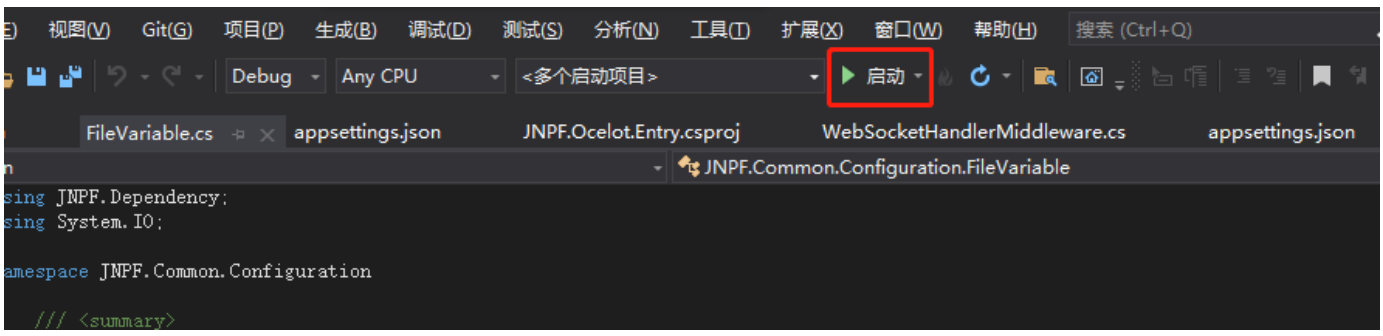
- 此时，弹窗窗口如下显示


解决方案"JNPF.NET.Cloud" 属性页
配置(C): 不适用
平台(P): 不适用
配置管理器(O)...
通用属性
启动项目
项目依赖项
代码分析设置
调试源文件
配置属性
当前选定内容(R)
单启动项目(S)
JNPF.OAuth.Entry
多个启动项目(M):
项目
操作
JNPF 无
JNPF.Apps 无
JNPF.Apps.Core 无
JNPF.Apps.Entitys 无
JNPF.Apps.Entry 无
JNPF.Apps.Interfaces 无
JNPF.Authentication.JwtBearer 无
JNPF.BillRuleManager 无
JNPF.CacheManager 无
JNPF.ChangeDataBase 无
JNPF.Common 无
确定
取消
应用(A)

-如下进行设置



- 点击启动



服务器环境部署

Linux环境部署

> 本文档环境基于 CentOS-8.2.2004
>
> **硬件配置参考**

> 本文档中的硬件参考配置为最低配置要求

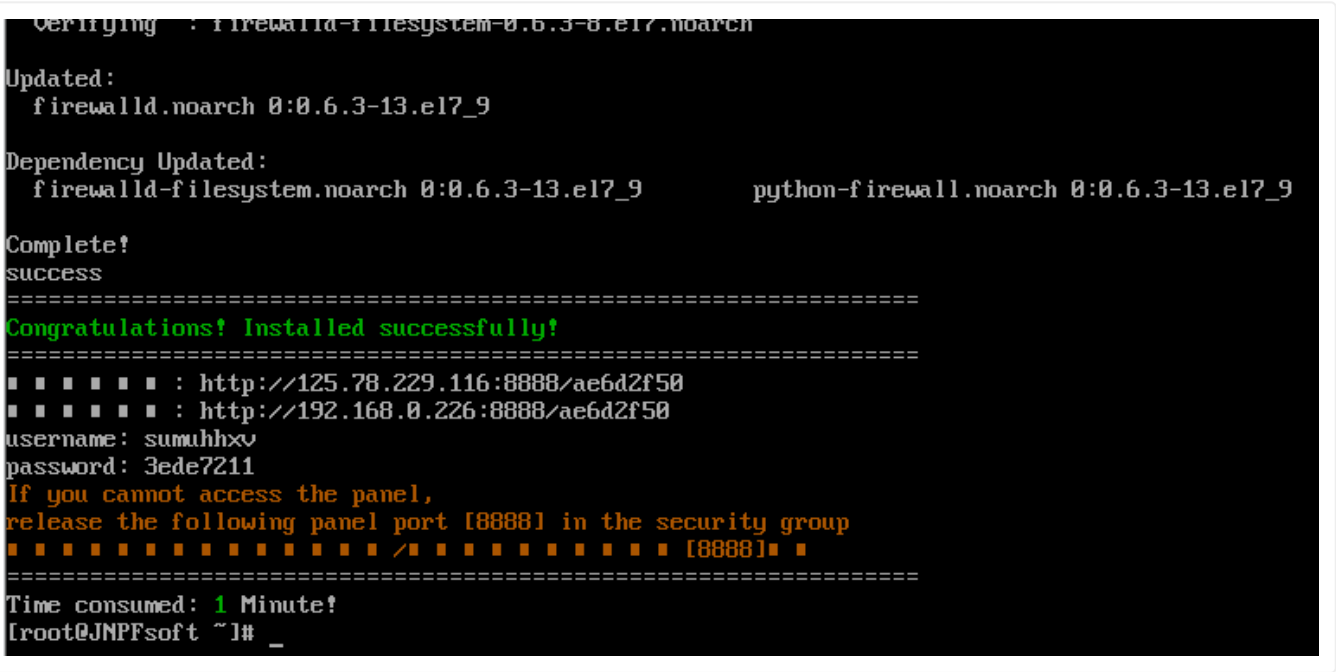
项目	CPU	内存	硬盘	描述
单体服务	2C	4G	30G	
微服务	2C	32G	50G	一般一个服务内存占用1G-3G

1. 安装宝塔

- 官网：https://www.bt.cn/
- 安装方法：

```
yum install -y wget && wget -O install.sh http://download.bt.cn/install/install_6.0.sh && sh install.sh
```

按提示安装成功后，会显示管理地址、用户名及密码（请务必妥善保管）

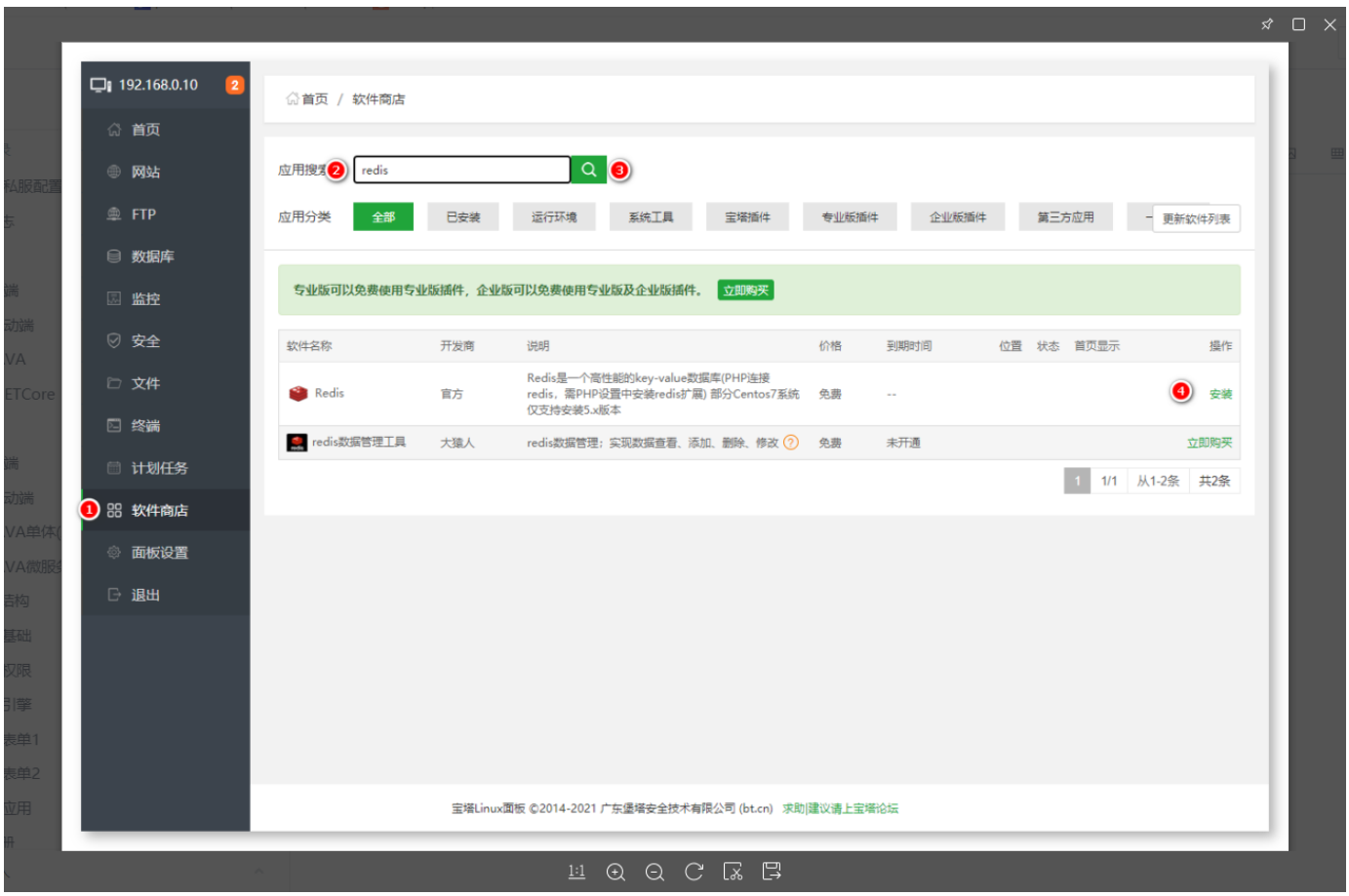


(1)进入宝塔Linux面板

① 选择安装Nginx和MySQL(JAVA环境),如下图：

② 安装Redis

在软件商店搜索Redis并安装



③ 因宝塔软件商店没提供JDK，需手动安装JDK，具体操作如下
首先上传JDK安装包至服务器，本文档中是存放在/usr/local/jnpf/，然后执行以下操作（可以coding云盘下载）

```
# 1.进入安装包所在目录
cd /usr/local/jnpf

# 2.解压
tar -zxvf jdk-8u261-linux-x64.tar.gz

# 3.打开配置文件
vi /etc/profile

# 4.JAVA环境变量配置，在最后面输入以下内容
JAVA_HOME=/usr/local/jnpf/jdk1.8.0_261
JRE_HOME=/usr/local/jnpf/jdk1.8.0_261/jre
CLASSPATH=.:$JAVA_HOME/lib:/dt.jar:$JAVA_HOME/lib/tools.jar
PATH=$PATH:$JAVA_HOME/bin

# 5.使配置生效
source /etc/profile

# 6.查看安装结果
java -version
```

④ 如果在命令行中显示以下信息，说明已安装成功

```
java version "1.8.0_261"
Java(TM) SE Runtime Environment (build 1.8.0_261-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.261-b12, mixed mode)
```

2. 安装.NET SDK

（1）官网链接：https://docs.microsoft.com/zh-cn/dotnet/core/install/linux-centos
（2）在根目录下命令安装：

```
sudo rpm -Uvh https://packages.microsoft.com/config/centos/7/packages-microsoft-prod.rpm

sudo yum install dotnet-sdk-6.0

sudo yum install aspnetcore-runtime-6.0
```

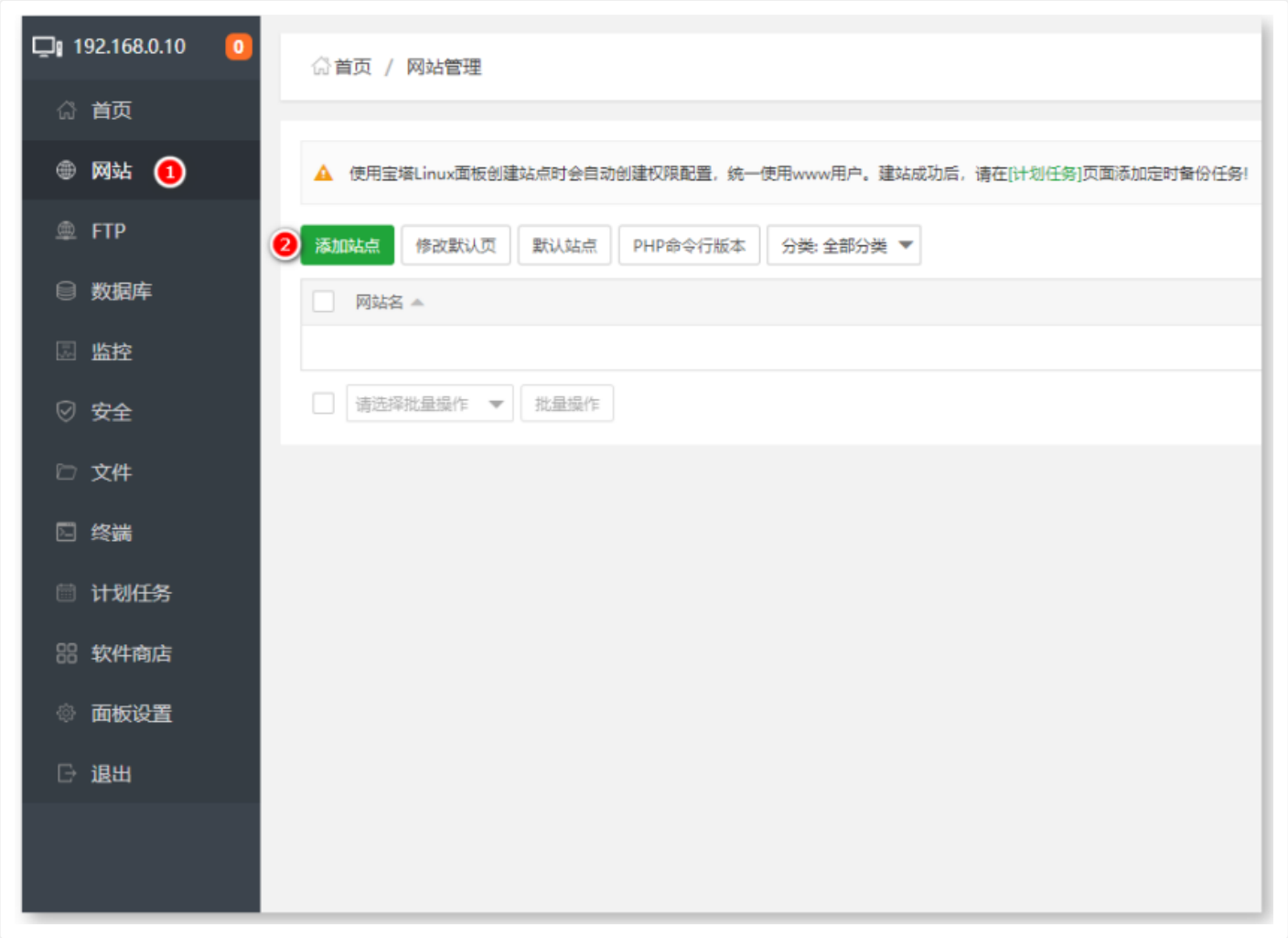
安装完成后，输入：

```
dotnet -version
```

查看版本号成功即安装成功。

3. 配置站点

(1)进入网站，添加站点



(2)站点配置如下

(3)添加成功后，我们先配置前端的配置文件

具体配置如下：

1.http网址的转发配置，如下：

```
server
{
  listen 80;
  server_name 192.168.0.10;
  index index.php index.html index.htm default.php default.htm default.html;
  root /www/wwwroot/jnpf-web;

  #JNPF-Start
  # 前端伪静态配置
  # 主项目前端
  location / {
    try_files $uri $uri/ /index.html;
  }
```

```
    # 大屏前端
    location /DataV {
      try_files $uri $uri/ /DataV/index.html;
    }
    #设置上传文件的大小
    client_max_body_size 100m;
    #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    client_header_buffer_size 128k;
    #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
    large_client_header_buffers 4 128k;
    #指定允许跨域的方法，*代表所有
    add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';
    # 预检命令的缓存，如果不缓存每次会发送两次请求
    add_header Access-Control-Max-Age 3600;
    #带cookie请求需要加上这个字段，并设置为true
    add_header Access-Control-Allow-Credentials true;
    #  表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #  $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #  表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;

    # 后端接口(按实际情况修改端口)
    location /api/ {
      proxy_pass http://localhost:5000/api/;
    }
    # webscket接口(按实际情况修改端口)
    location /websocket {
      proxy_pass http://localhost:5000/api/message/websocket;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
      proxy_read_timeout 600s;
    }
    # 报表设计接口配置(按实际情况修改端口)
    location /ReportServer/ {
      proxy_pass http://localhost:30007/;
    }
    # 文件预览服务
    location /FileServer {
```

```
    proxy_pass http://localhost:30090;
  }
  # 解决文件预览服务无法加载js,css问题
  location ~ /FileServer/*.*\.(js|css)?$ {
                proxy_pass http://localhost:30090;
  }
#JNPF-End
}
```

2.https网址的转发配置，如下：

```
server
{
    listen 80;
    listen 443 ssl http2;
    server_name 192.168.0.10;
    index index.php index.html index.htm default.php default.htm default.html;
    root /www/wwwroot/jnpf-web;

    #SSL-START SSL相关配置，请勿删除或修改下一行带注释的404规则
    #error_page 404/404.html;
    #HTTP_TO_HTTPS_START
    if ($server_port !~ 443){
        rewrite ^(/.*)$ https://$host$1 permanent;
    }
    #HTTP_TO_HTTPS_END
    ssl_certificate    /www/server/panel/vhost/cert/192.168.0.10/fullchain.pem;
    ssl_certificate_key    /www/server/panel/vhost/cert/192.168.0.10/privkey.pem;
    ssl_protocols TLSv1.1 TLSv1.2 TLSv1.3;
    ssl_ciphers EECDH+CHACHA20:EECDH+CHACHA20-draft:EECDH+AES128:RSA+AES128:EECDH+AE
S256:RSA+AES256:EECDH+3DES:RSA+3DES:!MD5;
    ssl_prefer_server_ciphers on;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;
    add_header Strict-Transport-Security "max-age=31536000";
    error_page 497  https://$host$request_uri;


    #SSL-END

    #ERROR-PAGE-START  错误页配置，可以注释、删除或修改
    #error_page 404 /404.html;
    #error_page 502 /502.html;
    #ERROR-PAGE-END
```

```
    #PHP-INFO-START  PHP引用配置，可以注释或修改
    include enable-php-00.conf;
    #PHP-INFO-END

    #REWRITE-START URL重写规则引用,修改后将导致面板设置的伪静态规则失效
    include /www/server/panel/vhost/rewrite/192.168.0.10;
    #REWRITE-END

    #禁止访问的文件或目录
    location ~ ^/(\.user.ini|\.htaccess|\.git|\.svn|\.project|LICENSE|README.md)
    {
        return 404;
    }

    #一键申请SSL证书验证目录相关设置
    location ~ \.well-known{
        allow all;
    }

    # location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
    # {
    #     expires      30d;
    #     error_log /dev/null;
    #     access_log /dev/null;
    # }

    # JNPF-START
    #设置上传文件的大小
    # client_max_body_size 100m;

    # #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    # #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    # client_header_buffer_size 128k;

    # #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
    # large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    # 预检命令的缓存，如果不缓存每次会发送两次请求
    #带cookie请求需要加上这个字段，并设置为true
```

```
    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号     #
表示请求头的字段 动态获取
    # 前端主项目(jnpf-web)伪静态
    location / {
      try_files $uri $uri/ /index.html;
    }

    # 前端大屏(jnpf-web-datascreen)伪静态
    location /DataV {
      try_files $uri $uri/ /DataV/index.html;
    }

    # 主项目
    location /api/ {
      proxy_pass http://localhost:5000;
    }

    location /websocket {
      proxy_pass http://localhost:5000/api/message/websocket;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
      proxy_read_timeout 600s;
    }

    # 报表（jnpf-datareport）接口
    location /ReportServer/ {
     proxy_pass http://localhost:30007/;
    }

    # 文件预览（jnpf-file-preview）
    location /FileServer {
      proxy_pass http://localhost:30090;
    }

    location ~ /FileServer/*.*\.(js|css)?$ {
                        proxy_pass http://localhost:30090;
    }
    # JNPF-End

    location ~ .*\.(js|css)?$
    {
        expires      12h;
        error_log /dev/null;
```

```
        access_log /dev/null;
    }
    access_log  /www/wwwlogs/netcore.log;
    error_log  /www/wwwlogs/netcore.error.log;
}
```

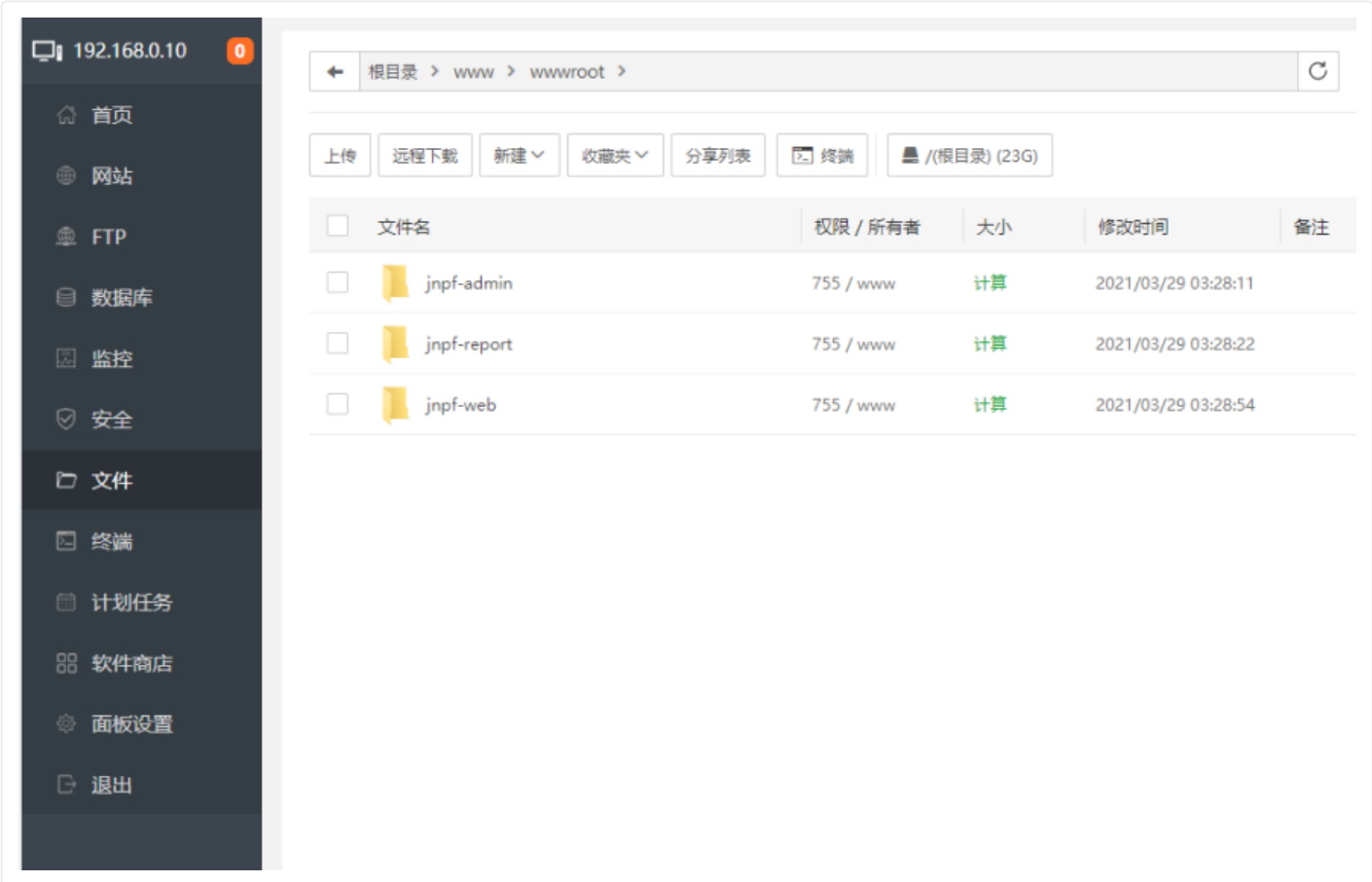
3.APP的接口转发配置，如下：

```
    #JNPF-Start

    #设置上传文件的大小
    client_max_body_size 100m;

    #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    client_header_buffer_size 128k;

    #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
    large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';

    # 预检命令的缓存，如果不缓存每次会发送两次请求
    add_header Access-Control-Max-Age 3600;

    #带cookie请求需要加上这个字段，并设置为true
    add_header Access-Control-Allow-Credentials true;

    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #    表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;


    # 伪静态
    location / {
      try_files $uri $uri/ /index.html;
    }
```

```
# 后端接口
location /api/ {
  proxy_pass http://localhost:5000;
}

# websocket
location /websocket {
  proxy_pass http://localhost:5000/api/message/websocket;
  proxy_http_version 1.1;
  proxy_set_header Upgrade $http_upgrade;
  proxy_set_header Connection "upgrade";
  proxy_read_timeout 600s;
}
#JNPF-End
```

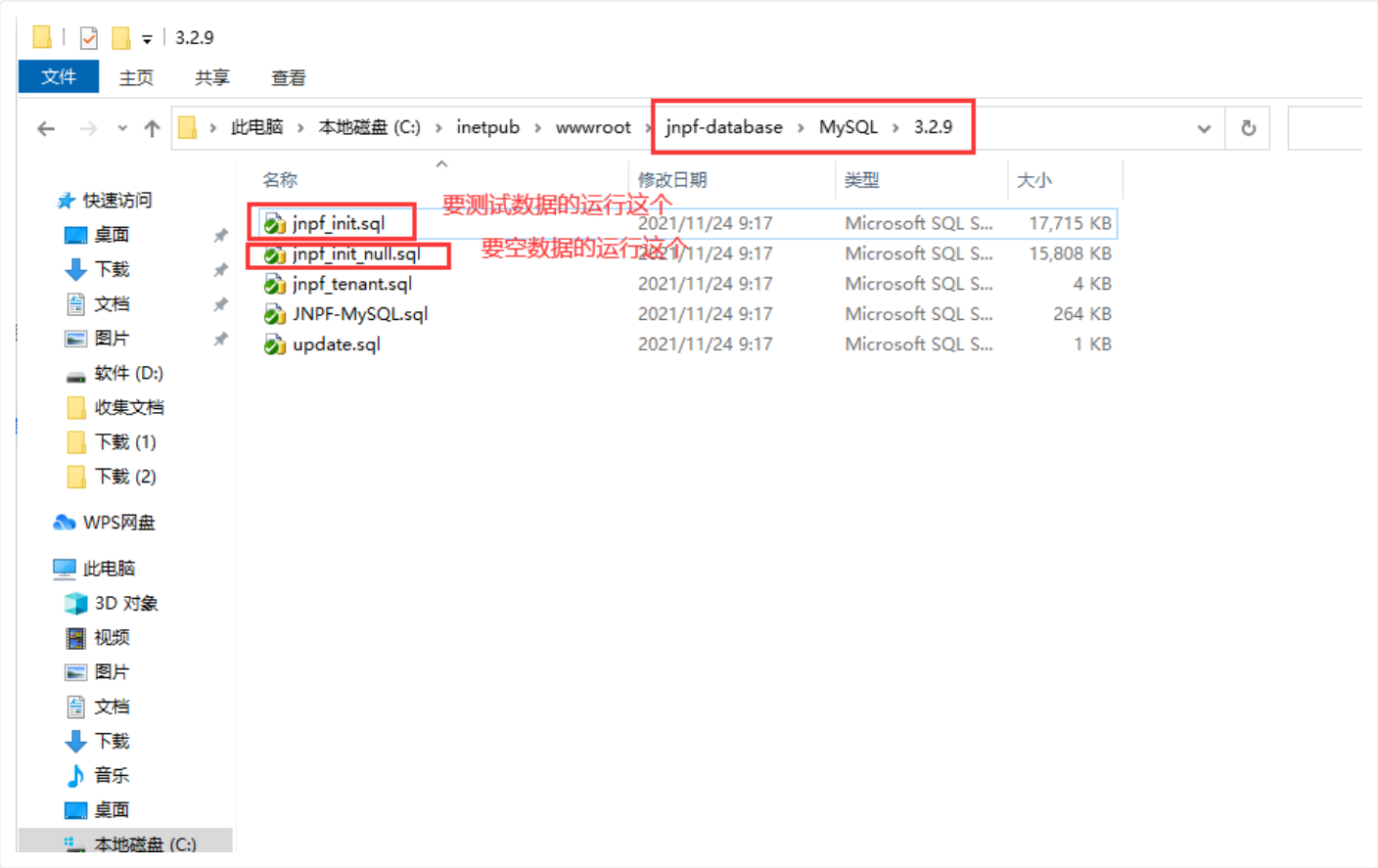
(4)目录建立

进入文件管理，按上文提供的目录约定创建对应目录



4.MySql创建数据库，创建用户，密码要记住

导入如下SQL文件



5.前端配置

主项目前端

配置
在项目根目录打开.env.staging(测试环境配置)，部署生产环境请打开.env.product文件

```
# 测试默认配置
ENV = 'staging'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.10'
# websocket接口
VUE_APP_BASE_WSS = 'ws://192.168.0.10/websocket'
```

```
# 生产环境默认配置
ENV = 'production'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.10'
# websocket接口
VUE_APP_BASE_WSS = 'ws://192.168.0.10/websocket'
```

构建

```
# 构建测试环境
npm run build:staging

# 构建生产环境
npm run build
```

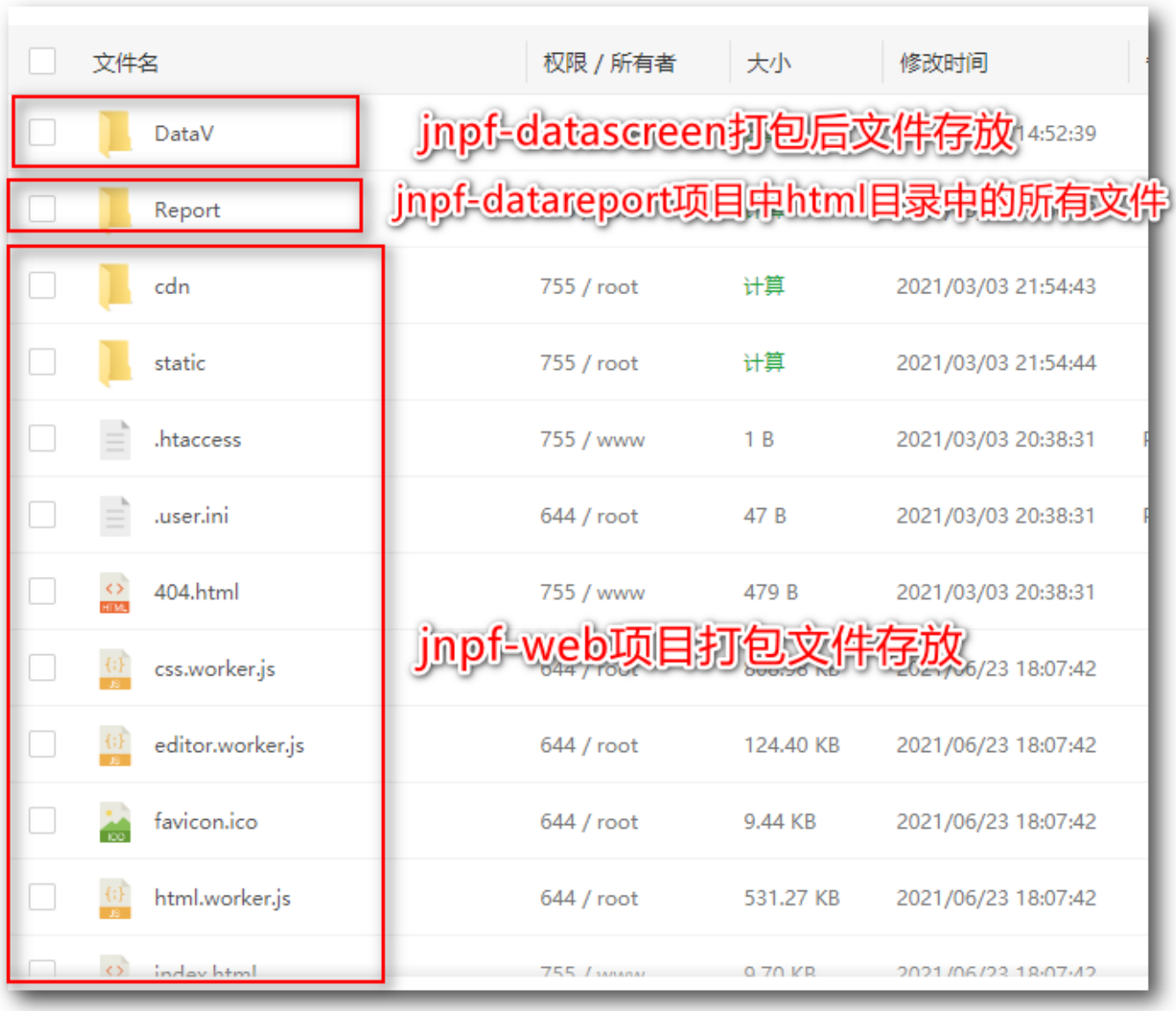
发布到服务器

- 方式1：压缩dist目录，上传到服务器/www/wwwroot/jnpf-web并解压
- 方式2：调整构建指令如下

```
npm run build:staging
scp -P 1802 -r  ./dist/* ssh root@192.168.0.10:/www/wwwroot/jnpf-web
```

6.报表前端部署

前端部署结构说明




```
├── jnpf-web                    # 假设这个目录是存放测试或生产环境的前端
│   ├── DataV                   # 大屏(`jnpf-datascreen`)打包后文件存放目录
│   ├── Report                  # 报表(`jnpf-datareport`)html下的文件
│   └── 主项目前端打包后的文件    # 主项目(`jnpf-web`)打包后存放在根目录
```

前端

- 将 `jnpf-web-datareport` 下的 `html` 文件夹中的所有拷贝到 `jnpf-web` 中的 `Report` 目录下, 如 `Report` 目录不存在请手动建立
- 接口配置
  - 打开 `/Report/index.html` ,做如下修改

```
# 在index.html文件中第24行开始
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.10/ReportServer";
// 报表前端
```

```
window._contextPath = "http://192.168.0.10/Report";
// 主项目接口地址
window._mainServer = "http://192.168.0.10";
</script>
```

- 打开 `/Report/preview.html` ,做如下修改,

```
# 在preview.html文件中第86行
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.10/ReportServer";
</script>
```

- 打开 `/Report/searchform.html` ,做如下修改,

```
# 在searchform.html文件中
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.10/ReportServer";
// 报表前端
window._contextPath = "http://192.168.0.10/Report";
</script>
```

- 配置说明:
  - 本示例中， `http://192.168.0.10` 为项目在测试环境中访问入口，在部署中根据实际情况调整;
  - 报表接口中， `ReportServer` 为虚拟目录，需要在Nginx增加相关配置，具体配置如下:

```
# 数据报表接口配置
  location /ReportServer/ {
     proxy_pass http://localhost:30007/;
  }
```

7.大屏前端部署

配置

在项目根目录打开 `.env.staging` (测试环境配置)，部署生产环境请打开 `.env.product` 文件

```
# 测试默认配置
ENV = 'staging'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.247'
```

```
# 生产环境默认配置
ENV = 'production'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.247'
```

构建

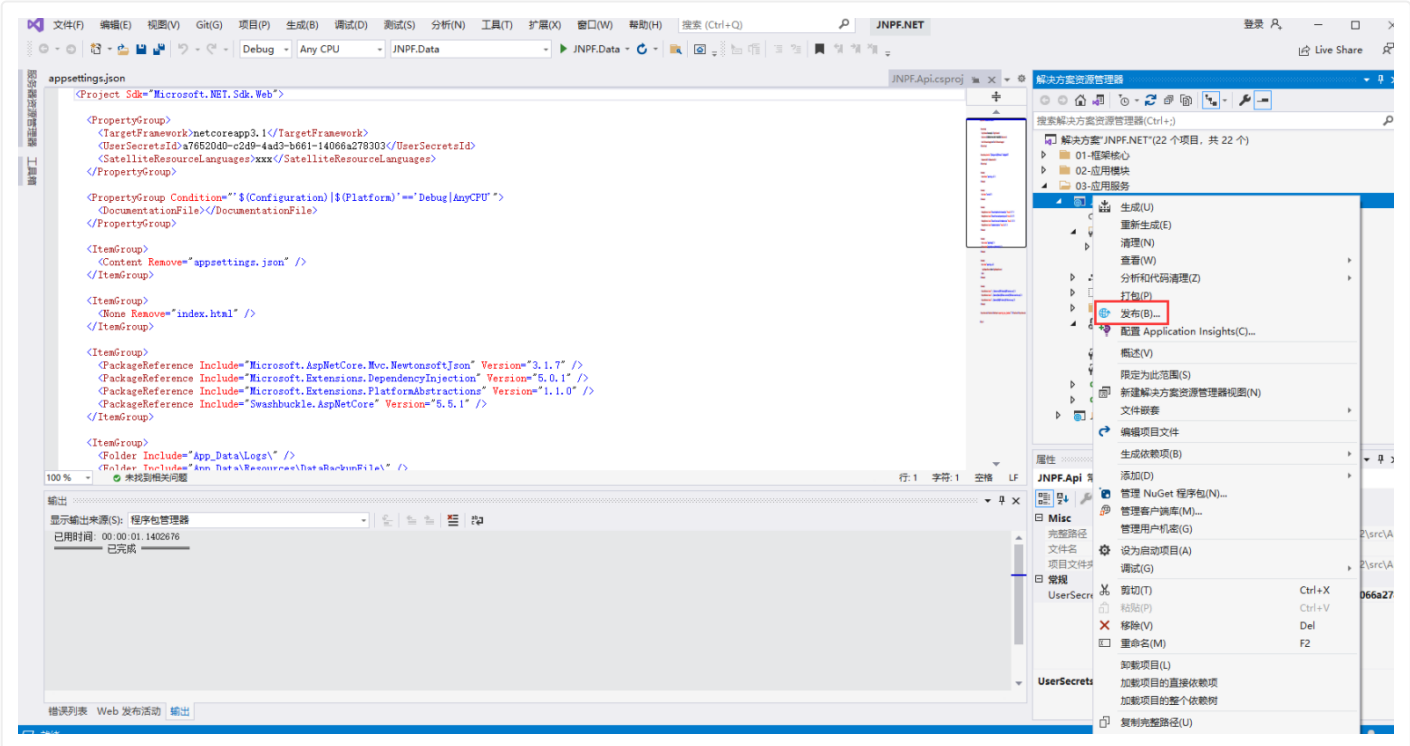
```
# 构建测试环境
yarn build:staging

# 构建生产环境
yarn build
```

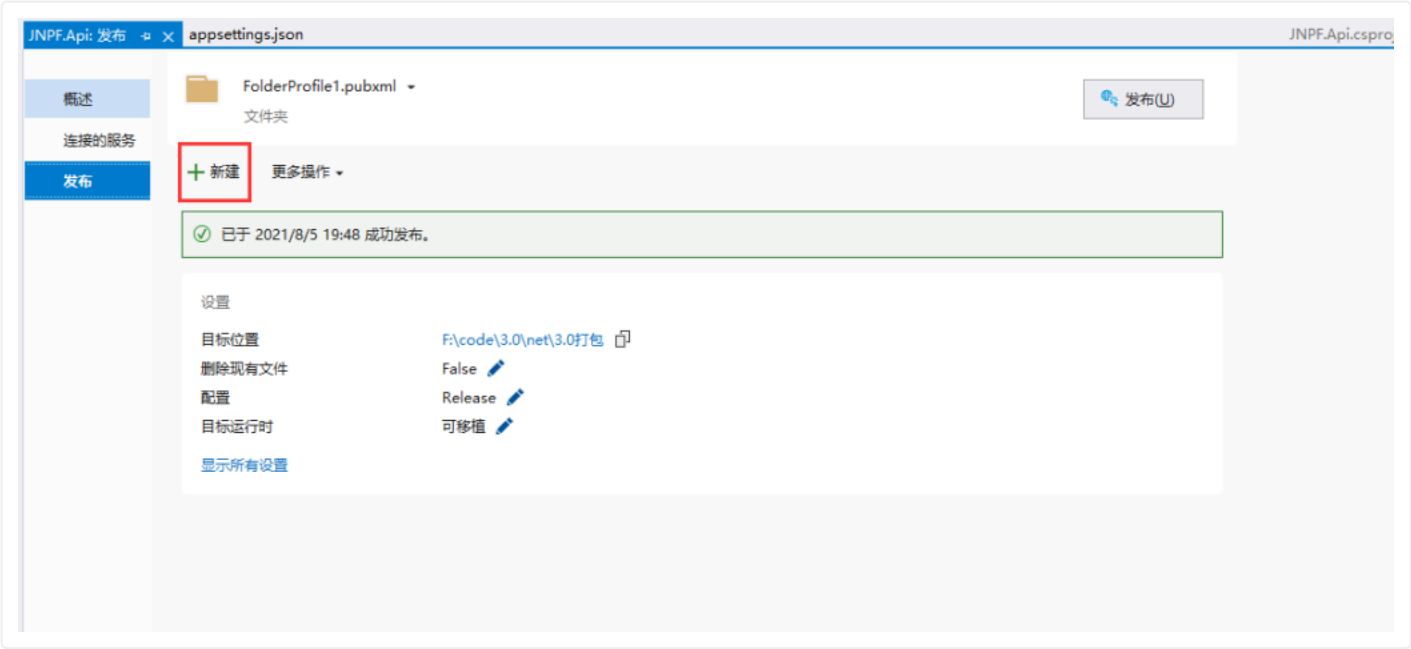
发布到服务器

- 压缩dist目录，上传到服务器C:\wwwroot\jnpf-web\DataV并解压

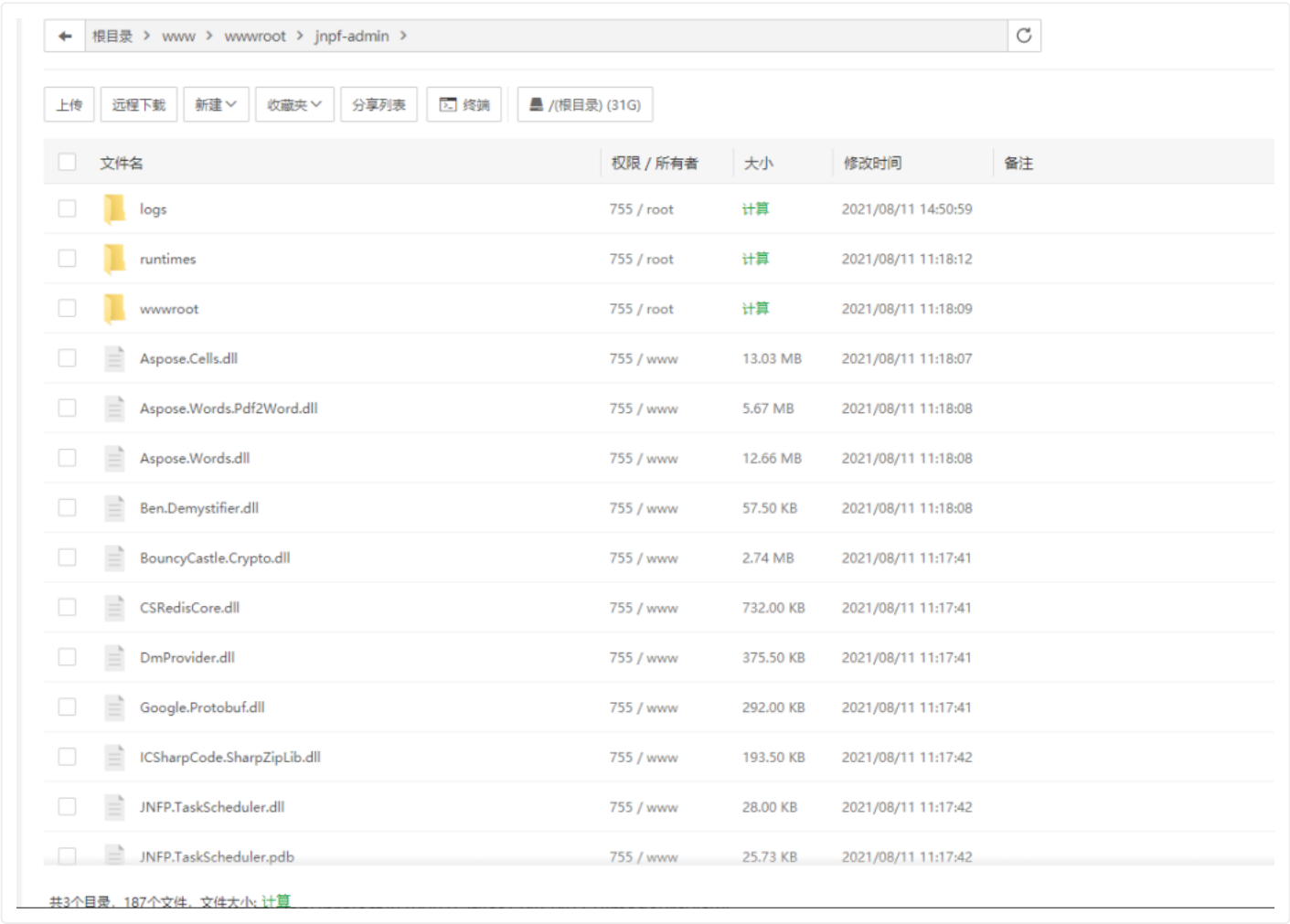
8.后端配置

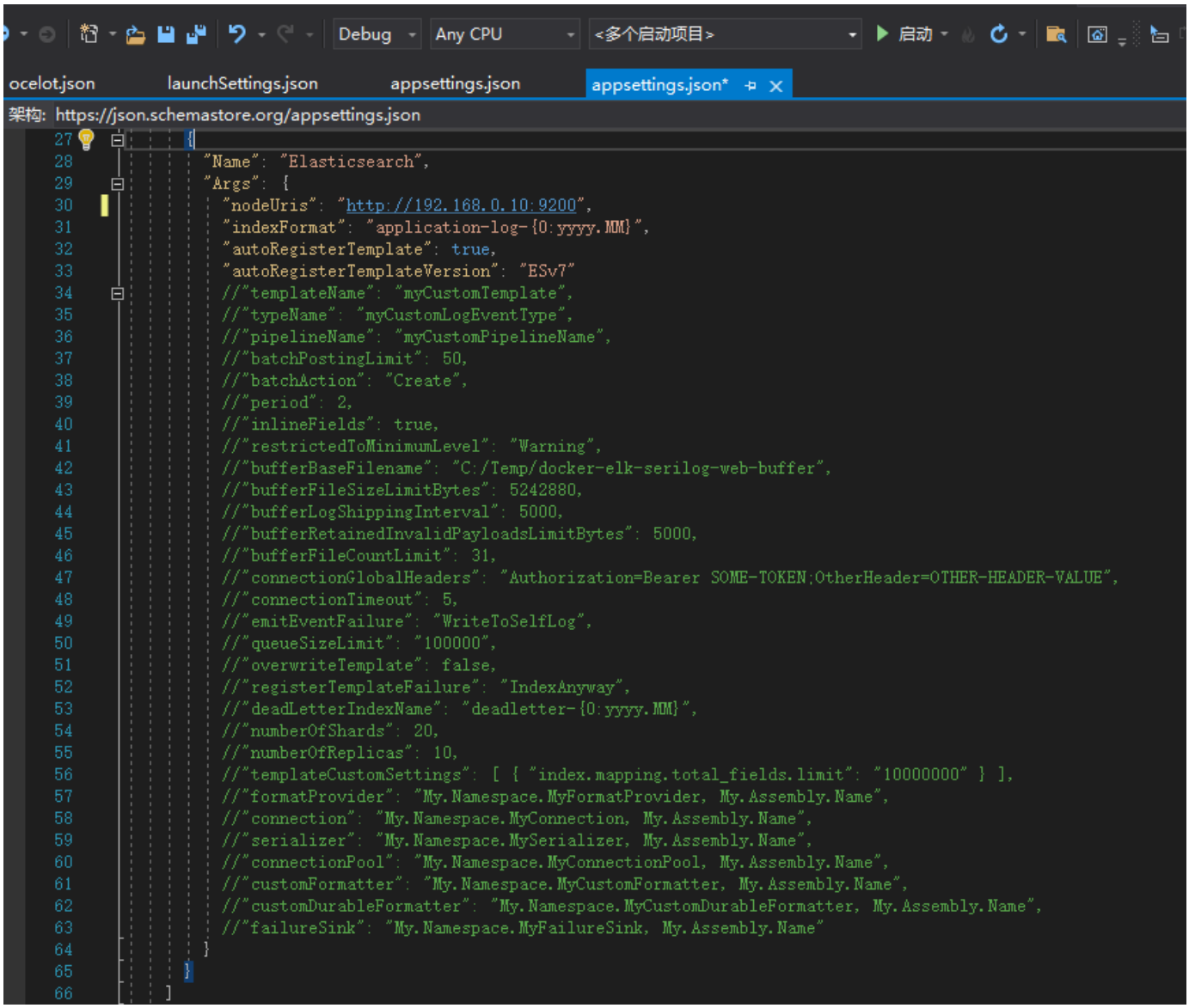
Visual studio 2019打开后端项目，应用服务里面的每一个服务 右键选择“发布”





发布文件拷贝指定目录（新建文件夹），修改appsetting.josn 配置：






```
        "Name": "Elasticsearch",
        "Args": {
          "nodeUris": "http://192.168.0.10:9200",
```

```
          "indexFormat": "application-log-{0:yyyy.MM}",
          "autoRegisterTemplate": true,
          "autoRegisterTemplateVersion": "ESv7"
          //"templateName": "myCustomTemplate",
          //"typeName": "myCustomLogEventType",
          //"pipelineName": "myCustomPipelineName",
          //"batchPostingLimit": 50,
          //"batchAction": "Create",
          //"period": 2,
          //"inlineFields": true,
          //"restrictedToMinimumLevel": "Warning",
          //"bufferBaseFilename": "C:/Temp/docker-elk-serilog-web-buffer",
          //"bufferFileSizeLimitBytes": 5242880,
          //"bufferLogShippingInterval": 5000,
          //"bufferRetainedInvalidPayloadsLimitBytes": 5000,
          //"bufferFileCountLimit": 31,
          //"connectionGlobalHeaders": "Authorization=Bearer SOME-TOKEN;OtherHeader=
OTHER-HEADER-VALUE",
          //"connectionTimeout": 5,
          //"emitEventFailure": "WriteToSelfLog",
          //"queueSizeLimit": "100000",
          //"overwriteTemplate": false,
          //"registerTemplateFailure": "IndexAnyway",
          //"deadLetterIndexName": "deadletter-{0:yyyy.MM}",
          //"numberOfShards": 20,
          //"numberOfReplicas": 10,
          //"templateCustomSettings": [ { "index.mapping.total_fields.limit": "10000
000" } ],
          //"formatProvider": "My.Namespace.MyFormatProvider, My.Assembly.Name",
          //"connection": "My.Namespace.MyConnection, My.Assembly.Name",
          //"serializer": "My.Namespace.MySerializer, My.Assembly.Name",
          //"connectionPool": "My.Namespace.MyConnectionPool, My.Assembly.Name",
          //"customFormatter": "My.Namespace.MyCustomFormatter, My.Assembly.Name",
          //"customDurableFormatter": "My.Namespace.MyCustomDurableFormatter, My.Ass
embly.Name",
          //"failureSink": "My.Namespace.MyFailureSink, My.Assembly.Name"
        }
```

```
  "Consul": {
    "ServiceName": "OAuth",
    "ServiceIP": "localhost",
    "ServicePort": 5001,
    "ServiceHealthCheck": "http://192.168.0.10:5001/HealthCheck",
    "ConsulAddress": "http://127.0.0.1:8500"
  },
```

```
//Apollo的配置
"Apollo": {
  "AppId": "JNPF.NET.Cloud",
  "Env": "DEV",
  "MetaServer": "http://192.168.0.10:8080",
  "ConfigServer": [ "http://192.168.0.10:8080" ],
  "Namespaces": [ "JNPF.NET", "application.json", "application" ]
}
```

部署Apollo

详情请看：服务器环境部署 -> Apollo部署文档

部署consul

详情请看：服务器环境部署 -> consul部署文档

后端运行

- 前台命令运行：

```
dotnet JNPF.Ocelot.Entry.dll --urls=http://*:5000
#网关服务运行命令
```

```
dotnet JNPF.Oauth.Entry.dll --urls=http://*:5001
#鉴权服务运行命令
```

```
dotnet JNPF.System.Entry.dll --urls=http://*:5002
#System服务运行命令
```

```
dotnet JNPF.Message.Entry.dll --urls=http://*:5003
#消息服务运行命令
```

```
dotnet JNPF.TaskScheduler.Entry.dll --urls=http://*:5004
#系统调度服务运行命令
```

```
dotnet JNPF.WorkFlow.Entry.dll --urls=http://*:5005
#WorkFlow服务运行命令
```

```
dotnet JNPF.VisualDev.Entry.dll --urls=http://*:5006
#VisualDev服务运行命令
```

```
dotnet JNPF.VisualData.Entry.dll --urls=http://*:5007
#大屏服务运行命令
```

```
dotnet JNPF.Extend.Entry.dll --urls=http://*:5008
#扩展服务运行命令
```

```
dotnet JNPF.SubDev.Entry.dll --urls=http://*:5009
#SubDev服务运行命令
```

```
dotnet JNPF.Apps.Entry.dll --urls=http://*:5011
#App服务运行命令
```

- 后台运行：

```
nohup dotnet JNPF.Ocelot.Entry.dll --urls="http://*:5000" --environment=Delopmen
t > /dev/null 2>&1 &
#网关服务运行命令
```

```
nohup dotnet JNPF.Oauth.Entry.dll --urls="http://*:5001" --environment=Delopment
> /dev/null 2>&1 &
#鉴权服务运行命令
```

```
nohup dotnet JNPF.System.Entry.dll --urls="http://*:5002" --environment=Delopmen
t > /dev/null 2>&1 &
#System服务运行命令
```

```
nohup dotnet JNPF.Message.Entry.dll --urls="http://*:5003" --environment=Delopme
nt > /dev/null 2>&1 &
#消息服务运行命令
```

```
nohup dotnet JNPF.TaskScheduler.Entry.dll --urls="http://*:5004" --environment=D
elopment > /dev/null 2>&1 &
#系统调度服务运行命令
```

```
nohup dotnet JNPF.WorkFlow.Entry.dll --urls="http://*:5005" --environment=Delopm
ent > /dev/null 2>&1 &
#WorkFlow服务运行命令
```

```
nohup dotnet JNPF.VisualDev.Entry.dll --urls="http://*:5006" --environment=Delop
ment > /dev/null 2>&1 &
#VisualDev服务运行命令
```

```
nohup dotnet JNPF.VisualData.Entry.dll --urls="http://*:5007" --environment=Delo
pment > /dev/null 2>&1 &
#大屏服务运行命令
```

```
nohup dotnet JNPF.Extend.Entry.dll --urls="http://*:5008" --environment=Delopmen
t > /dev/null 2>&1 &
#扩展服务运行命令
```

```
nohup dotnet JNPF.SubDev.Entry.dll --urls="http://*:5009" --environment=Delopmen
t > /dev/null 2>&1 &
#SubDev服务运行命令
```

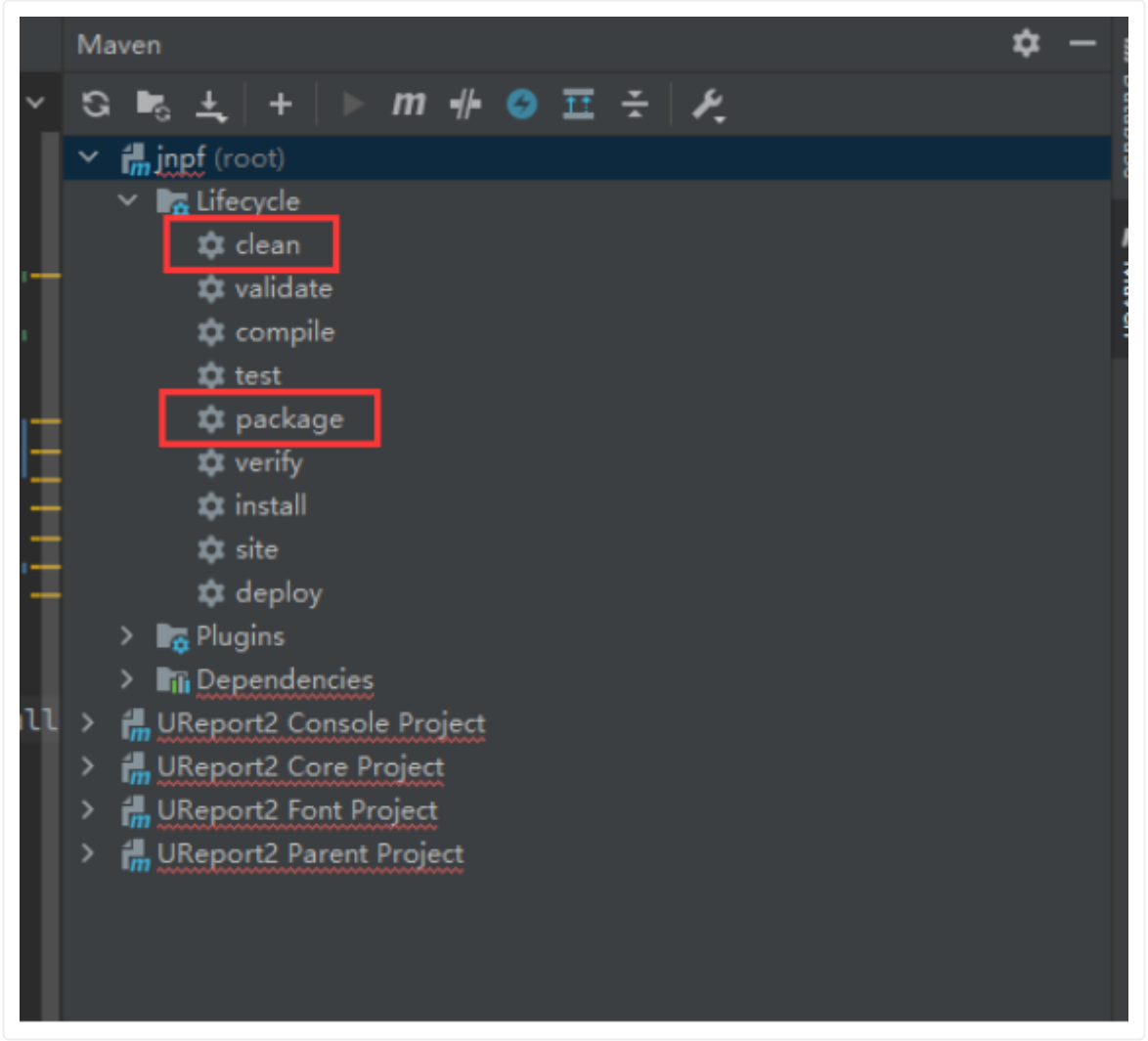
```
nohup dotnet JNPF.Apps.Entry.dll --urls="http://*:5011" --environment=Delopment
 > /dev/null 2>&1 &
#App服务运行命令
```

- 停止运行：

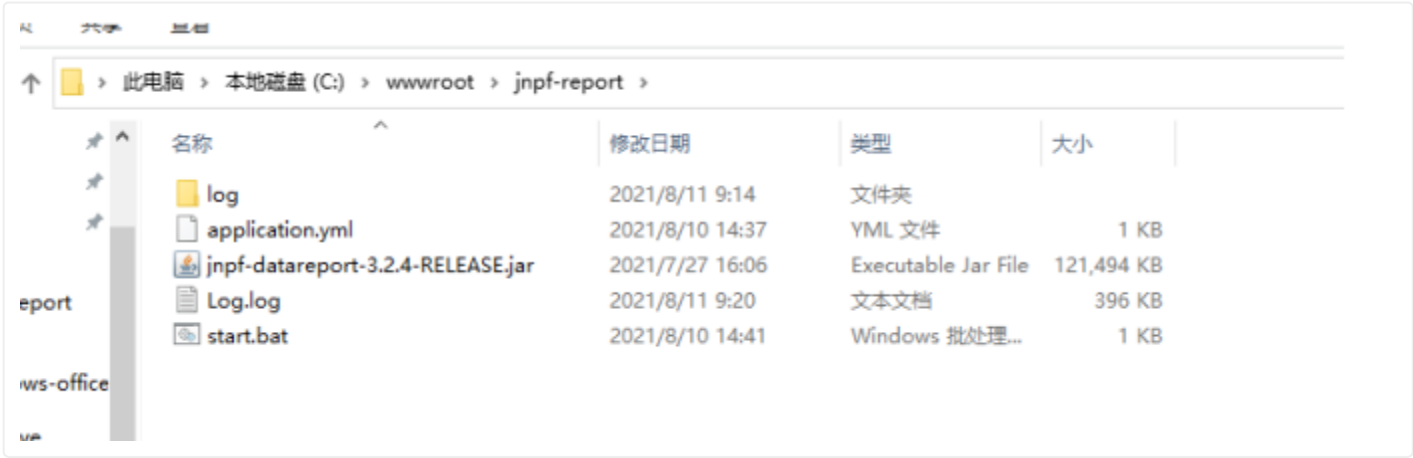
```
kill -9 $(netstat -nlp | grep 5000 | awk '{print $7}' | awk -F"/" '{ print $1 }'
)
```

9.报表后端项目打包

Intellij IDEA打开报表后端项目jnpf-datareport,打包



上传指服务器指定目录下（可以新建）



打开application.Yml文件，修改数据库配置并保存：

```yaml
# 配置端口
server:
  port: 30007
  max-http-header-size: 102400
  compression:
    enabled: true
    min-response-size: 102400
spring:
  # 表空间(Oracle)
  tableSpace: JNPFCLOUD
  datasource:
    # MySQL配置
    druid:
      dbinit: jnpf_init
      dbname: jnpf_init
      dbnull: jnpf_init
      url: jdbc:mysql://localhost:3306/{dbName}?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8
      username: root
      password: 123456
      driver-class-name: com.mysql.cj.jdbc.Driver

  #Redis配置
  redis:
    database: 1
    host: localhost
    port: 6379
    password:
    timeout: 3000
    lettuce:
      pool:
        max-active: 8 # 连接池最大连接数
        max-wait: -1ms  # 连接池最大阻塞等待时间（使用负值表示没有限制）
        min-idle: 0 # 连接池中的最小空闲连接
        max-idle: 8 # 连接池中的最大空闲连接
logging:
  level:
    root: info
    com.bstek.ureport.console: debug
  file:
    path: log
config:
```

MySQL配置：

```yaml
    druid:
      dbinit: jnpf_init
      dbname: jnpf_init
      dbnull: jnpf_init
      url: jdbc:mysql://localhost:3306/{dbName}?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8
      username: root
      password: 123456
      driver-class-name: com.mysql.cj.jdbc.Driver
```

SQLServer配置:

```yaml
    druid:
      dbinit: jnpf_init
      dbname: jnpf_init
      dbnull: jnpf_init
      url: jdbc:sqlserver://localhost:1433;DatabaseName={dbName}
      username: sa
      password: JNPF@2020
      driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
```

Oracle配置:

```
  # 表空间(Oracle)
  tableSpace: JNPFCLOUD
```

```
    druid:
      dbinit: JNPFCLOUD
      dbname: JNPFCLOUD
      dbnull: JNPFCLOUD
      url: jdbc:oracle:thin:@192.168.0.10:1521:{dbName}
      username: JNPFCLOUD
      password: JNPFCLOUD
      driver-class-name: oracle.jdbc.driver.OracleDriver
```

运行.jar文件:

```
nohup java -jar -Xmx2000m -Xms2000m -Xmn1000m -Xss256k -XX:+HeapDumpOnOutOfMemoryErr
or jnpf-datareport-3.2.11-RELEASE.jar >Log.log 2>&1 &
```

注：新建文本，输入命令后，修改后缀为.sh，用sh命令运行。
停止运行:

```
kill -9 $(netstat -nlp | grep 30000 | awk '{print $7}' | awk -F"/" '{ print $1 }')
```

查看日志运行成功即可使用报表模块。

10.文档预览部署

环境要求

- JDK1.8+
- OpenOffice或LiberOffice(Windows下已内置，CentOS或Ubuntu下会自动下载安装，MacOS下需要自行安装)

部署运行

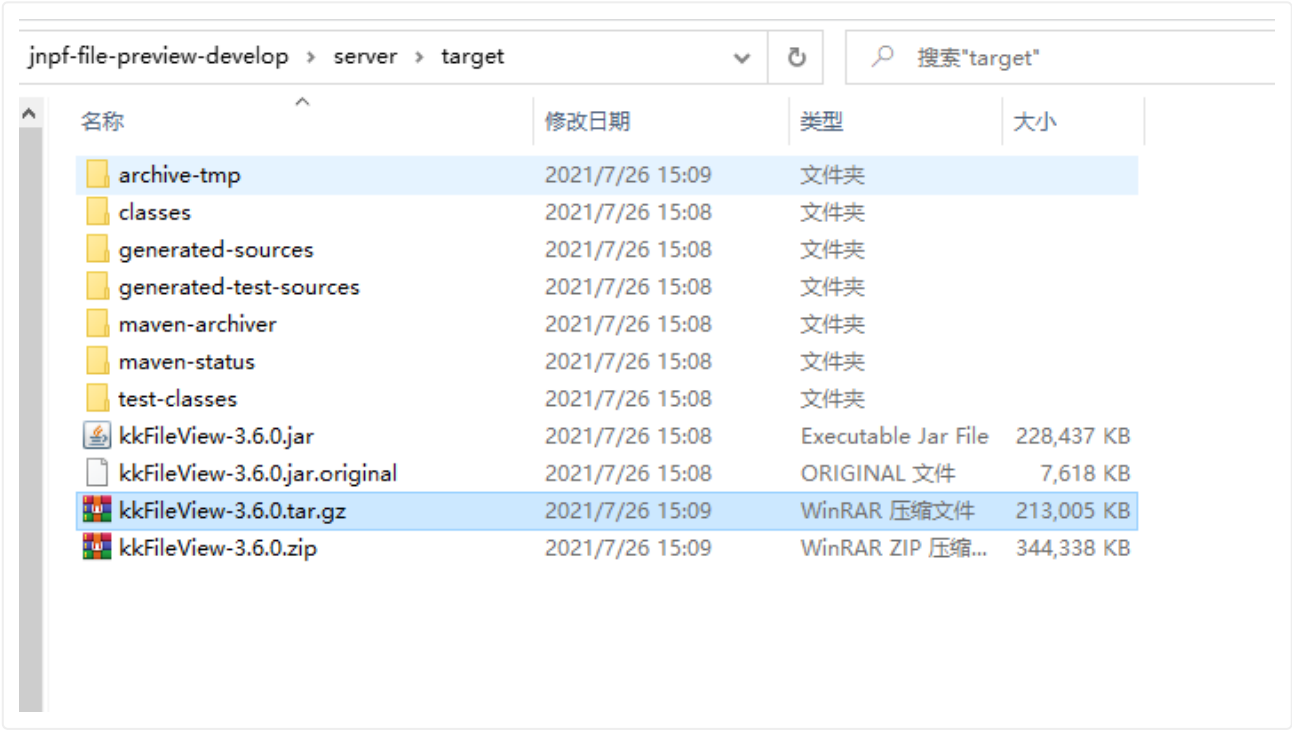
1、开发环境

- a.IDEA导入项目
- b.调整配置，打开 `server/src/main/config/application.properties`
- c.启动项目， `server/src/main/java/cn/keking/ServerMain`
- d.打开 `http://localhost:30090` 测试页面

2、测试生产环境

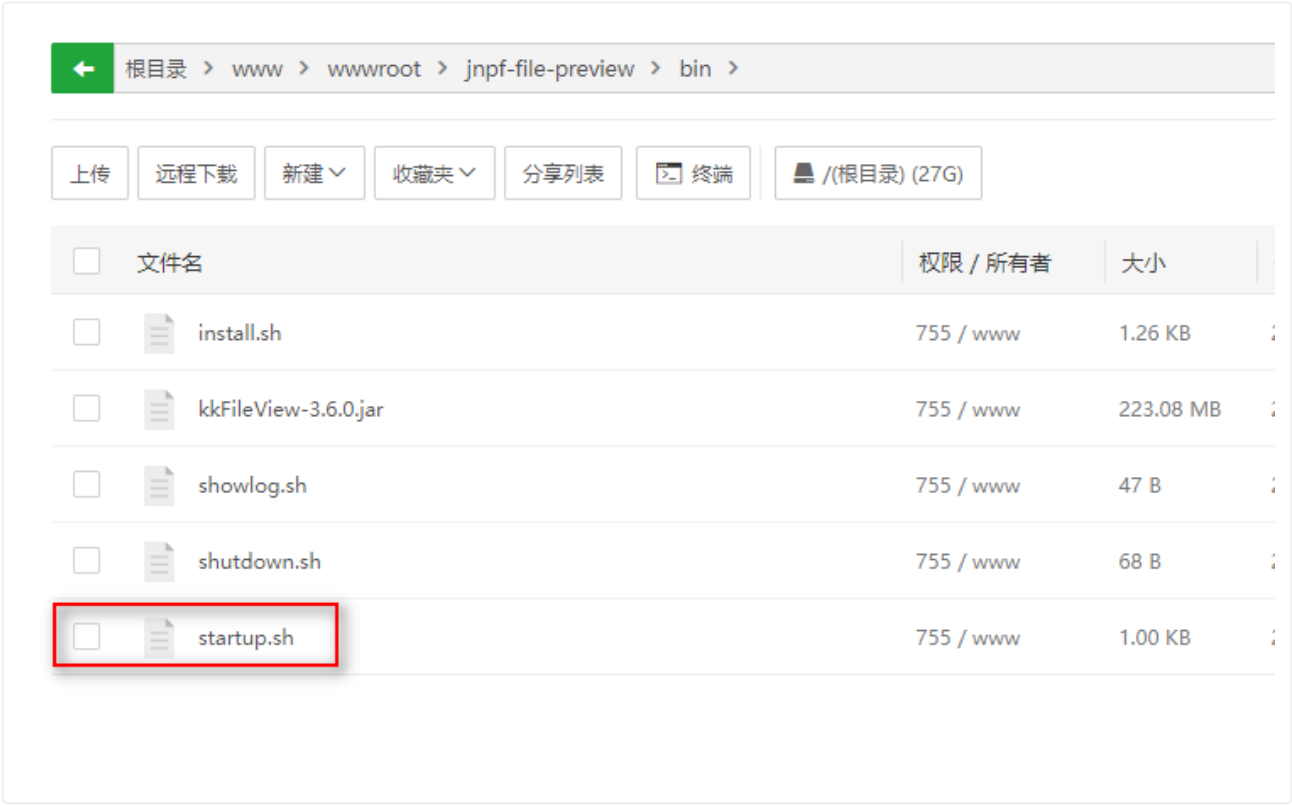
- a.打包

  - 打开 `\server\target` 主要有以下几个文件
  - kkFileView-xxx.jar(一般用于更新)
  - kkFileView-xxx.zip(windows环境下首次部署)
  - kkFileView-xxx.tar.gz(Linux环境下首次部署)



- b.上传至服务器

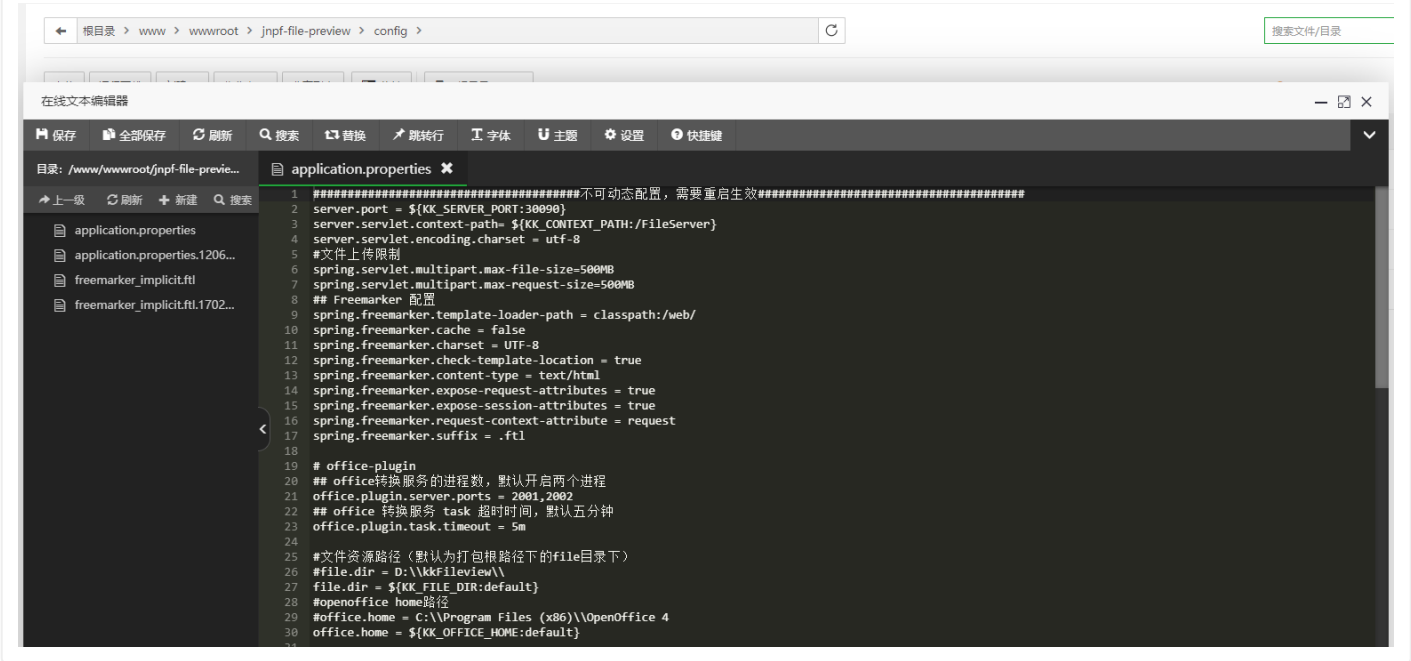
- 打开解压后文件夹的bin目录，运行startup脚本(首次部署，之后更新及维护都在bin目录下)



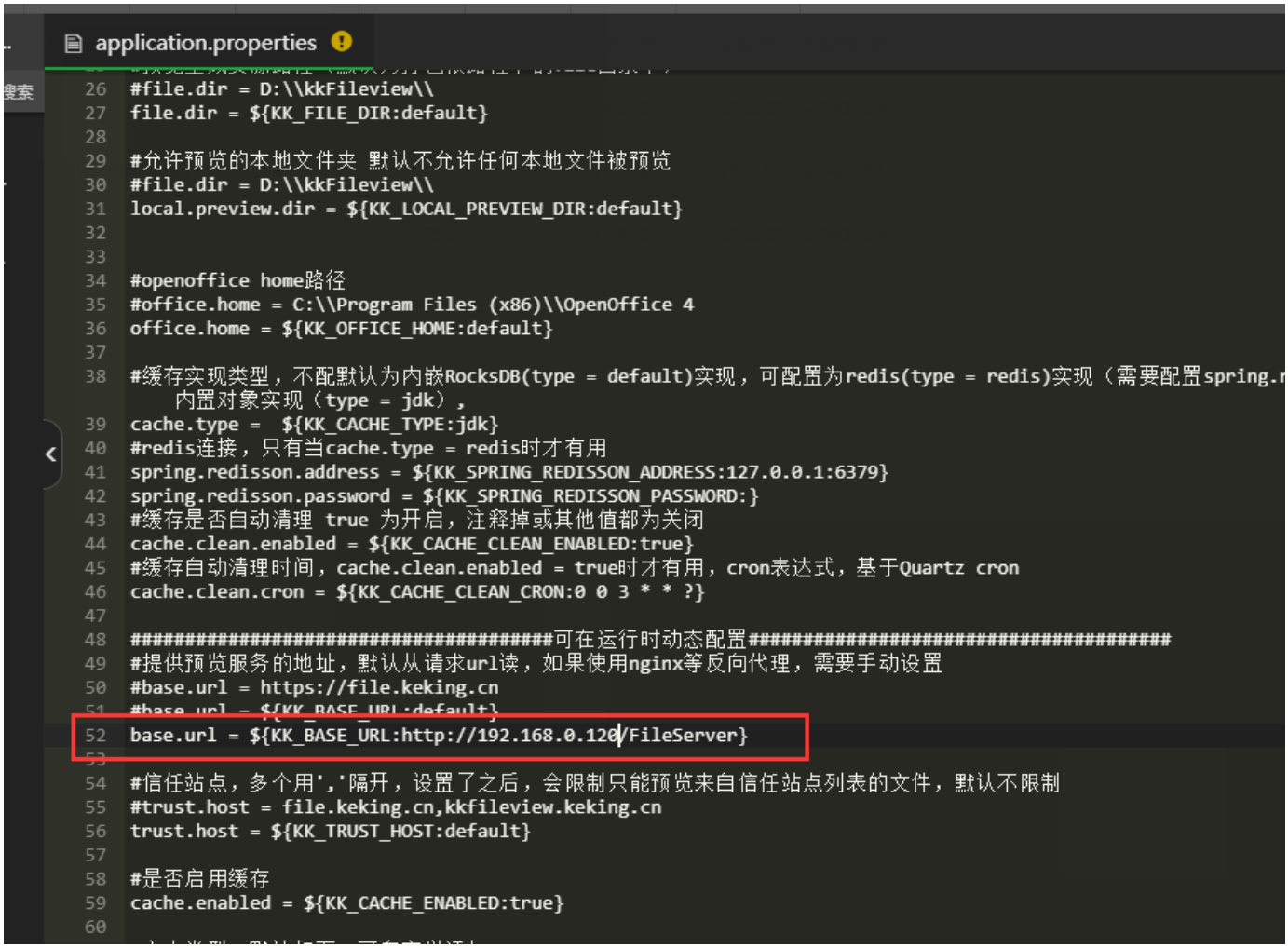
- 修改文件预览配置文件如下两项,打开服务器上的 `kkFileView-xxx/config/application.properties`（第3行和第45行）

修改成:

```
server.servlet.context-path= ${KK_CONTEXT_PATH:/FileServer}
base.url = ${KK_BASE_URL:http://192.168.0.10/FileServer}
```



新版只需修改第52行：



- c.Nginx配置说明
  例如nginx的访问地址为 `https://netcore.jnpfsoft.com` ,文件预览部署在内网192.168.0.10服务器上，需要在nginx中添加反向代理如下

```
# 文件预览服务
location /FileServer {
  proxy_pass 192.168.0.10:30090;
}

# 解决文件预览服务无法加载js,css问题
location ~ /FileServer/*.*\.(js|css)?$ {
  proxy_pass 192.168.0.10:30090;
}
```

使用指南

1、单文件预览

```
// 要预览文件的访问地址
let previewUrl = 'http://127.0.0.1:8080/file/test.txt'

// 使用
http://127.0.0.1:30090/onlinePreview?url='+encodeURIComponent(Base64.encode(previewUrl))
```

2、http/https下载流url预览

```
// 要预览文件的访问地址
let previewUrl = 'http://127.0.0.1:8080/filedownload?fileId=1'

// 使用
http://127.0.0.1:30090/onlinePreview?url='+encodeURIComponent(Base64.encode(previewUrl))
```

3、更多参考官方文档（`https://kkfileview.keking.cn/zh-cn/docs/usage.html`）

常见问题

1、预览乱码(字体问题)

大部分Linux系统上并没有预装中文字体或字体不全，需要把常用字体拷贝到Linux服务器上，具体操作如下： 下载如下字体包 `http://kkfileview.keking.cn/fonts.zip` 文件解压完整拷贝到Linux下的 `/usr/share/fonts` 目录。然后依次执行 `mkfontscale`、`mkfontdir`、`fc-cache` 使字体生效

> 特别说明：安装字体前确保Linux服务器已安装mkfontscale、fontconfig，如未安装运行以下命令

CentOS运行如下命令

```
# 使mkfontscale和mkfontdir命令正常运行
yum install mkfontscale

# 使fc-cache命令正常运行
yum install fontconfig
```

Ubuntu运行如下命令

```
# 使mkfontscale和mkfontdir命令正常运行
sudo apt-get install ttf-mscorefonts-installer

# 使fc-cache命令正常运行
sudo apt-get install fontconfig
```

2、更多问题请参考官方文档说明(`https://kkfileview.keking.cn/zh-cn/docs/faq.html`)

11.常见问题

（1）mysql数据库执行查询命令：（解决只有三个菜单问题）

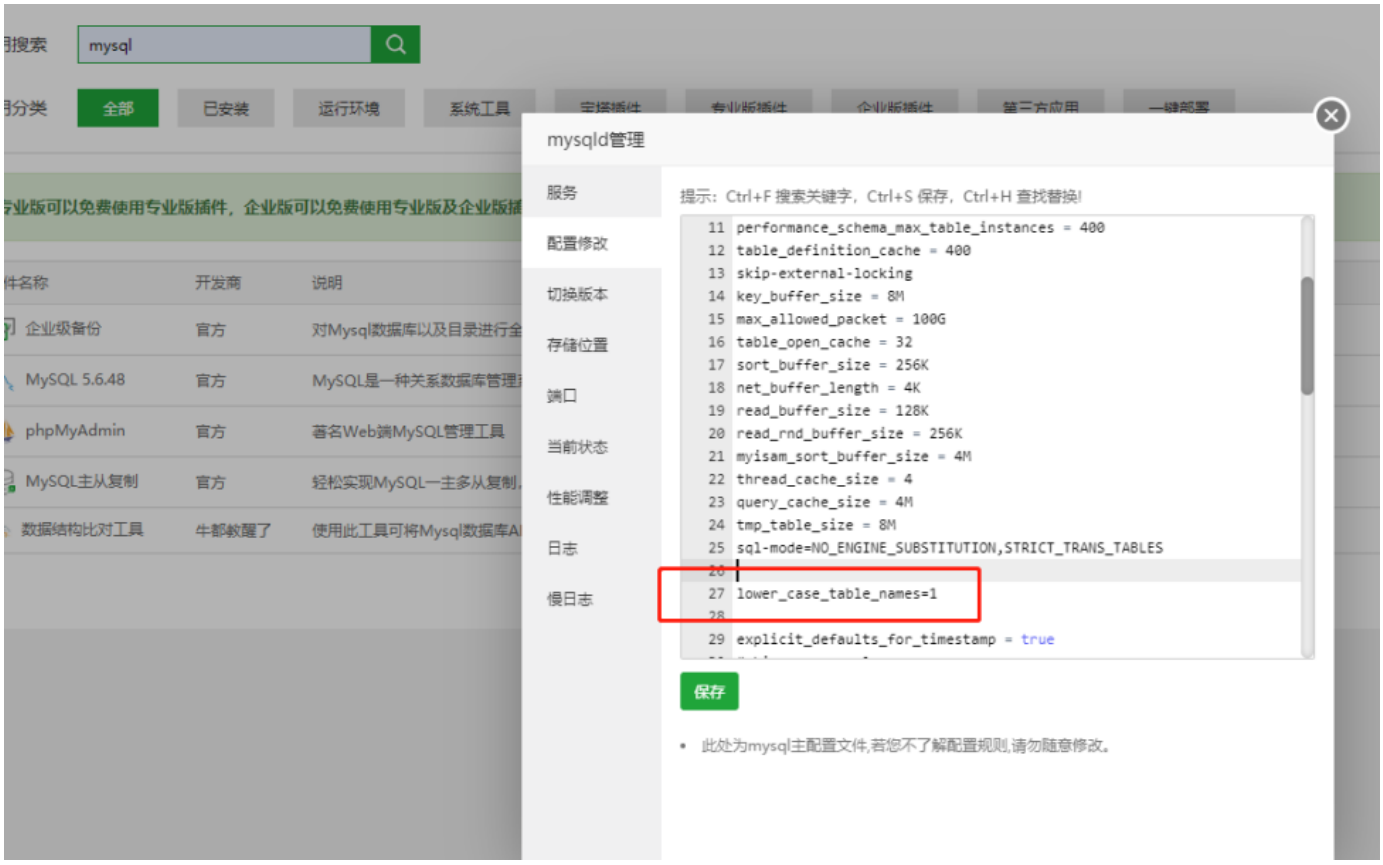
```
update base_module set F_DeleteMark=null
```

（2）数据库添加my.cnf：

```
lower_case_table_names = 1
```

解决mysql大小写转换问题。

```
【时间】2021-08-11 13:41:49,661
【等级】ERR
【消息】SqlSugar.SqlSugarException: English Message : Connection open error . Unable to connect to any of the specified MySQL hosts.
Chinese Message :  连接数据库过程中发生错误，检查服务器是否正常连接字符串是否正确，实在找不到原因请先Google错误信息：Unable to connect to any of the specified MySQL hosts..
   at SqlSugar.AdoProvider.GetDataReaderAsync(String sql, SugarParameter[] parameters)
   at SqlSugar.QueryableProvider`1.GetDataAsync[TResult](KeyValuePair`2 sqlObj)
   at SqlSugar.QueryableProvider`1._ToListAsync[TResult]()
   at SqlSugar.QueryableProvider`1.FirstAsync()
   at SqlSugar.QueryableProvider`1.FirstAsync(Expression`1 expression)
   at SqlSugar.SqlSugarRepository`1.FirstOrDefaultAsync(Expression`1 whereExpression) in F:\code\3.2net\jnpf-netcore-master\src\Infrastructure\JNPF.Data.SqlSugar\Repositories\SqlSugarRepository.cs:line 226
   at JNPF.System.Service.Permission.UsersService.GetInfoByAccount(String account) in F:\code\3.2net\jnpf-netcore-master\src\Modularity\System\JNPF.System\Service\Permission\UsersService.cs:line 544
   at JNPF.OAuth.Service.OAuthService.Login(LoginInput input) in F:\code\3.2net\jnpf-netcore-master\src\Modularity\OAuth\JNPF.OAuth\Service\OAuthService.cs:line 123
```



解决出现死循环问题：base_user：F_ManagerId清空。

```
09-aa7f-c0a6890e4546','f8c18307-065c-4dd3-a6c6-a2b95ba17ed6','f96d5799-c894-4f00-81dc-4caa97e294e4',
'fa0719d2-4a8b-4bac-ba2b-b65fbc286c1b','3bbad457-663e-4314-9b10-33c67c35cbcb','048657c1-ecb1-4e5d-ac
ed-6126ceae779e','0a85de84-4de0-4317-9767-25f95fdd9b1e','0c1d9463-221f-49ba-8e38-28c1f1df0c4b','146f
6c3a-9eaa-4295-817f-8bf4e19b1a20','1adea3ff-8a25-420b-a628-c5cb78fd51c0','1d91dd85-989e-4252-8d7f-e4
2e04bc6b3d','22736bf4-f2d5-4081-9fb4-ca43d6235f56','25e10fa4-7c68-4bc4-b751-92fa54f24c5b','26fe6815-
20a2-4be7-93b4-25e195d22833','27c57c5a-4c50-4572-954b-f9dafbbbca89','2ab1ce27-0903-4e09-8a40-a1b0f1e
8066c','317549d0-c4e5-400d-9ee5-72066869b1b3','02ed7f16-1337-454d-ac47-3250551a1588','3d855aed-83af-
4fe3-92fd-3d105a3ebd02','474f8b7a-17ce-466e-8b14-4ce53d610d40','4c33d1a2-759d-4b6a-8a37-b5e7249fce24
','52d997ed-9a1e-45de-ad89-6d4f542a321a','639528cd-4299-48ae-854a-2b7a33384f96','65ca58f2-ee3a-478f-
9343-da4a932ee143','69e88d0c-2bf0-45de-bfb9-7e023a78bb21','75b1d3f0-737e-46f6-afb1-c096c230de49','75
cd6e33-afa0-42c1-aec4-c8a0bf4c0d71','7d4ecaef-6be5-4ee5-bd9b-696d3f7dec23','7f04a44f-533d-46a8-b3ab-
f23f488a57dc','8c410beb-f10d-4dd6-941f-1f9c63962c28')    AND ( `F_DeleteMark` IS NULL )ORDER BY `F_S
ORTCODE` ASC
SELECT `F_Id` FROM `BASE_USER`  WHERE `F_MANAGERID` IN ('bd8c8775-5a47-4d06-952e-7c4cfe86d13e','964b
44e3-b813-432a-8d05-cf6c36056b0d','969b6be5-c9c2-432b-b952-7953c93577b7','99e7703f-f0e0-48ab-8d6d-82
1a4da6e8bc','a0923414-c5d3-4bee-8ab1-b03d879a227c','a4a0a204-ebbe-4428-accf-0c123abdd983','admin','a
fe0a868-3d70-4c6b-8200-58898c0e4e72','b0afc597-c533-4b21-950a-7f33081b3413','b10cfe40-6855-4934-8ea2
-47171e9ec9ad','b77b8f5b-1a57-4c4a-8233-7dad385cc611','bae583b0-a120-4a4d-9892-287a0ef0210b','bbb00a
3b-abbd-4e30-8b83-0ae0b06d5623','91e531c1-eba1-45e3-8243-2fd1cc84dde9','c76ecfed-30c1-4e3d-8517-18de
ad5e59ee','cf22426e-227d-40a3-8f9f-b45106934e04','e0d9ff45-4b86-4139-a8a4-548e01d34996','e10c139d-3e
e3-4b19-9b5b-93459ae31628','e3bccece-4c8f-49c3-86fb-feb9ba143726','e4582489-a568-4527-b31d-7b011c089
340','eb3ea438-2f7e-47d9-81a9-30e73ff3b909','f3322d81-a007-4e28-9c43-55489ef1d3db','f77c6a01-e3e5-4d
09-aa7f-c0a6890e4546','f8c18307-065c-4dd3-a6c6-a2b95ba17ed6','f96d5799-c894-4f00-81dc-4caa97e294e4',
'fa0719d2-4a8b-4bac-ba2b-b65fbc286c1b','3bbad457-663e-4314-9b10-33c67c35cbcb','048657c1-ecb1-4e5d-ac
ed-6126ceae779e','0a85de84-4de0-4317-9767-25f95fdd9b1e','0c1d9463-221f-49ba-8e38-28c1f1df0c4b','146f
6c3a-9eaa-4295-817f-8bf4e19b1a20','1adea3ff-8a25-420b-a628-c5cb78fd51c0','1d91dd85-989e-4252-8d7f-e4
2e04bc6b3d','22736bf4-f2d5-4081-9fb4-ca43d6235f56','25e10fa4-7c68-4bc4-b751-92fa54f24c5b','26fe6815-
20a2-4be7-93b4-25e195d22833','27c57c5a-4c50-4572-954b-f9dafbbbca89','2ab1ce27-0903-4e09-8a40-a1b0f1e
8066c','317549d0-c4e5-400d-9ee5-72066869b1b3','02ed7f16-1337-454d-ac47-3250551a1588','3d855aed-83af-
4fe3-92fd-3d105a3ebd02','474f8b7a-17ce-466e-8b14-4ce53d610d40','4c33d1a2-759d-4b6a-8a37-b5e7249fce24
','52d997ed-9a1e-45de-ad89-6d4f542a321a','639528cd-4299-48ae-854a-2b7a33384f96','65ca58f2-ee3a-478f-
9343-da4a932ee143','69e88d0c-2bf0-45de-bfb9-7e023a78bb21','75b1d3f0-737e-46f6-afb1-c096c230de49','75
cd6e33-afa0-42c1-aec4-c8a0bf4c0d71','7d4ecaef-6be5-4ee5-bd9b-696d3f7dec23','7f04a44f-533d-46a8-b3ab-
f23f488a57dc','8c410beb-f10d-4dd6-9
```

解决Dotnet无法预览问题：

appsettings.json

```
      "app_id": "",
      "app_key": "",
      "redirect_uri": "",
      "scope": "snsapi_userinfo"
    }
  },
  "JNPF_App": {
    "CodeAreasName": "SubDev",
    //系统文件路径(末尾必须带斜杆)
    "SystemPath": "//www//wwwroot//resources//",
    //微信公众号允许上传文件类型
    "MPUploadFileType": "bmp,png,jpeg,jpg,gif,mp3,wma,wav,amr,mp4",
    //微信允许上传文件类型
    "WeChatUploadFileType": "jpg,png,doc,docx,ppt,pptx,xls,xlsx,pdf,txt,rar,zip,csv,amr,mp4",
    //允许图片类型
    "AllowUploadImageType": "jpg,gif,png,bmp,jpeg,tiff,psd,swf,svg,pcx,dxf,wmf,emf,lic,eps,tga",
    //允许上传文件类型
    "AllowUploadFileType": "jpg,gif,png,bmp,jpeg,doc,docx,ppt,pptx,xls,xlsx,pdf,txt,rar,zip,csv",
    "Domain": "http://192.168.0.186",
    "YOZO": {
      "domain": "http://dcsapi.com/",
      "domainKey": "57462250284462899305150"
    },
    //Minio
    "BucketName": "jnpfsoftoss",
    //文件存储类型(本地:local,MinIo:minio,阿里云:aliyun-oss,腾讯云:tencent-cos)
    "FileStoreType": "local",
    //================== 系统错误邮件报告反馈相关 ============================== -->
    //软件的错误报告
    "ErrorReport": "false",
    //软件的错误报告发给谁
    "ErrorReportTo": "yinmaisoft@163.com"
  }
```

附录：

Windows环境部署

项目说明

> 以下为 `JNPF 3.3.x` .NET项目命名规则

项目名	说明
`JNPF.Net.Could`	.NET主项目

环境要求

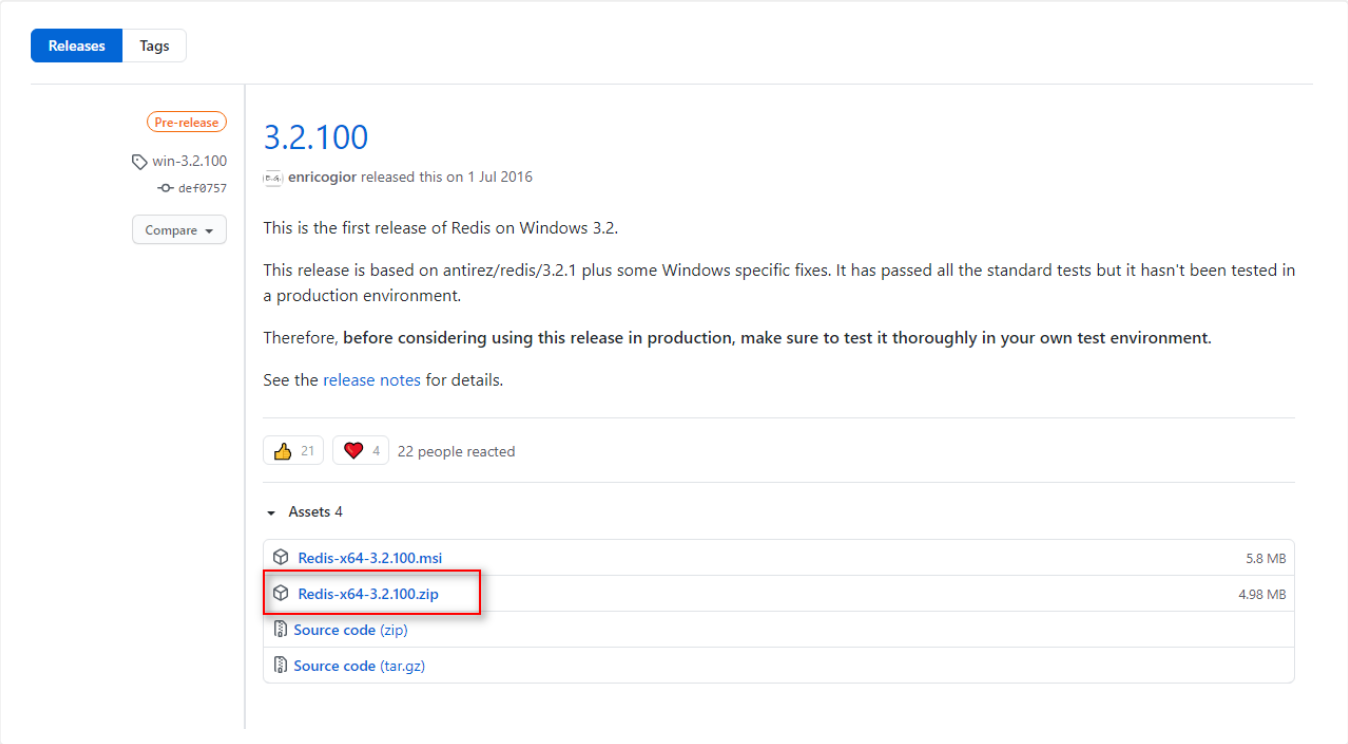
基础要求

> 环境具体安装教程

环境	推荐版本	说明
Nginx		
JDK	1.8.x	JAVA环境依赖(需配置环境变量)
SDK	最新版	.NET环境依赖
Redis	3.2.100(Windows)/6.0.x(Linux,Mac)	
MySQL	5.7.x+	数据库任选一
SQLServer	2012+	数据库任选一
Orale	11g+	数据库任选一

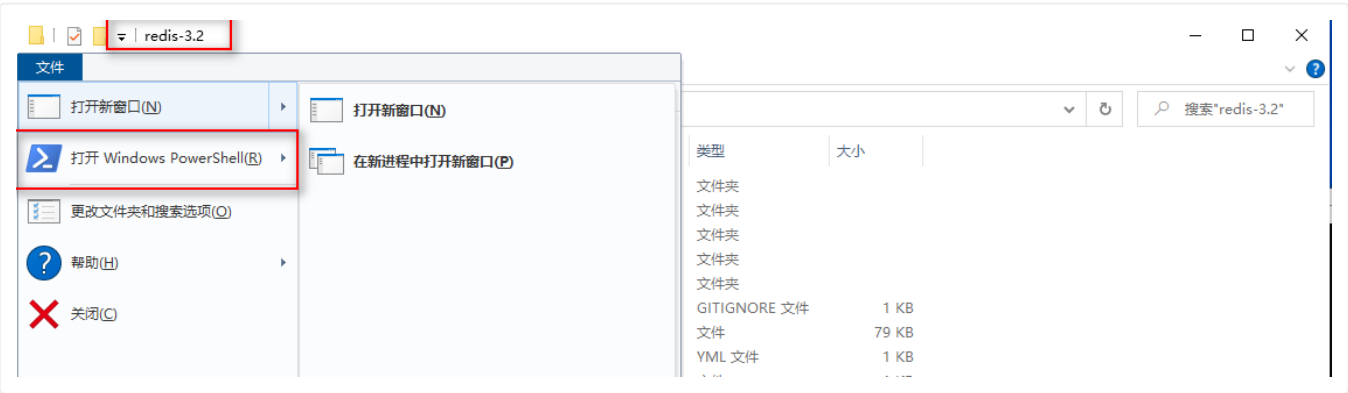
1. Redis安装

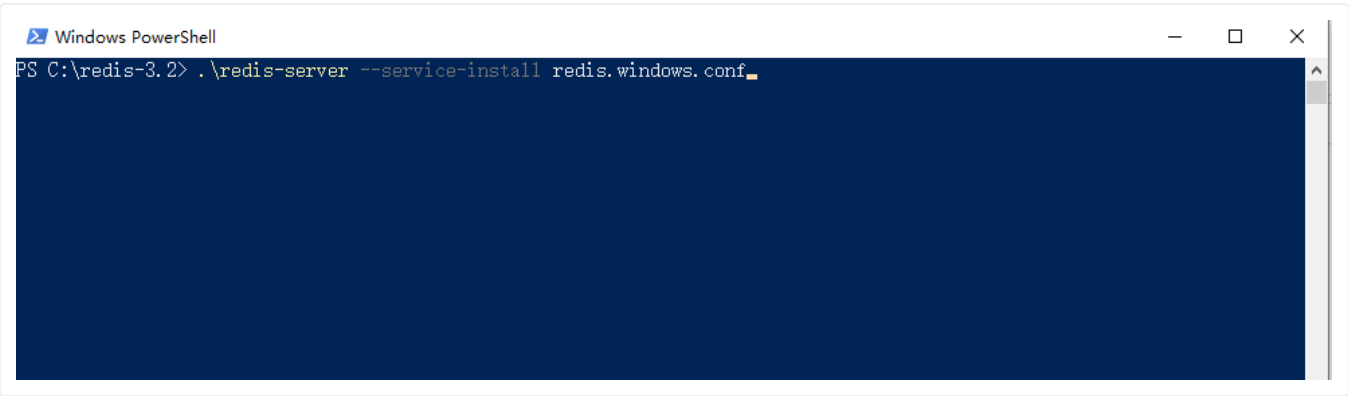
- Redis下载：https://github.com/microsoftarchive/redis/releases



配置并设置为服务

1） 打开命令行(`Windows PowerShell`)，输入.\redis-server --service-install redis.windows.conf





2） 右击 `电脑` – `管理` – `服务和应用程序` – `服务` 启动服务

3） `Redis` 常用的指令

- 卸载服务：`redis-server --service-uninstall`
- 开启服务：`redis-server --service-start`
- 停止服务：`redis-server --service-stop`

2. nginx安装

Nginx下载（Windows版）：http://nginx.org/en/download.html
启动命令

```
#启动命令:
start nginx.exe
#停止命令:
nginx.exe -s stop
#修改nginx.conf后不断服务重启:
nginx.exe -s reload
```

进入 `nginx-conf` 文件夹，修改 `nginx.conf` 配置文件：

1.http网址的转发配置，如下：

```
server
{
  listen 80;
  server_name 192.168.0.247;
  index index.php index.html index.htm default.php default.htm default.html;
  root c:\wwwroot\jnpf-web;

  #JNPF-Start
  # 前端伪静态配置
  # 主项目前端
  location / {
    try_files $uri $uri/ /index.html;
  }
  # 大屏前端
  location /DataV {
    try_files $uri $uri/ /DataV/index.html;
  }
  #设置上传文件的大小
  client_max_body_size 100m;
  #添加头部信息
  proxy_set_header Cookie $http_cookie;
  proxy_set_header X-Forwarded-Host $host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
  client_header_buffer_size 128k;
  #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
  large_client_header_buffers 4 128k;
  #指定允许跨域的方法，*代表所有
  add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';
  # 预检命令的缓存，如果不缓存每次会发送两次请求
  add_header Access-Control-Max-Age 3600;
  #带cookie请求需要加上这个字段，并设置为true
  add_header Access-Control-Allow-Credentials true;
```

```
    #  表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #  $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #  表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;

    # 后端接口(按实际情况修改端口)
    location /api/ {
      proxy_pass http://localhost:30000/api/;
    }
    # webscket接口(按实际情况修改端口)
    location /websocket {
      proxy_pass http://localhost:30000/api/message/websocket;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
      proxy_read_timeout 600s;
    }
    # 报表设计接口配置(按实际情况修改端口)
    location /ReportServer/ {
      proxy_pass http://localhost:30007/;
    }
    # 文件预览服务
    location /FileServer {
      proxy_pass http://localhost:30090;
    }
    # 解决文件预览服务无法加载js,css问题
    location ~ /FileServer/*.*\.(js|css)?$ {
                  proxy_pass http://localhost:30090;
    }
  #JNPF-End
  }
```

2.https网址的转发配置，如下：

```
server
{
    listen 80;
    listen 443 ssl http2;
    server_name 192.168.0.247;
    index index.php index.html index.htm default.php default.htm default.html;
    root c:\wwwroot\jnpf-web;

    #SSL-START SSL相关配置，请勿删除或修改下一行带注释的404规则
    #error_page 404/404.html;
```

```
    #HTTP_TO_HTTPS_START
    if ($server_port !~ 443){
        rewrite ^(/.*)$ https://$host$1 permanent;
    }
    #HTTP_TO_HTTPS_END
    ssl_certificate    c:\www\192.168.0.247\fullchain.pem;
    ssl_certificate_key    c:\www\192.168.0.247\privkey.pem;
    ssl_protocols TLSv1.1 TLSv1.2 TLSv1.3;
    ssl_ciphers EECDH+CHACHA20:EECDH+CHACHA20-draft:EECDH+AES128:RSA+AES128:EECDH+AE
S256:RSA+AES256:EECDH+3DES:RSA+3DES:!MD5;
    ssl_prefer_server_ciphers on;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;
    add_header Strict-Transport-Security "max-age=31536000";
    error_page 497  https://$host$request_uri;


    #SSL-END

    #ERROR-PAGE-START  错误页配置，可以注释、删除或修改
    #error_page 404 /404.html;
    #error_page 502 /502.html;
    #ERROR-PAGE-END

    #PHP-INFO-START  PHP引用配置，可以注释或修改
    include enable-php-00.conf;
    #PHP-INFO-END

    #REWRITE-START URL重写规则引用,修改后将导致面板设置的伪静态规则失效
    include c:\www\192.168.0.247;
    #REWRITE-END

    #禁止访问的文件或目录
    location ~ ^/(\.user.ini|\.htaccess|\.git|\.svn|\.project|LICENSE|README.md)
    {
        return 404;
    }

    #一键申请SSL证书验证目录相关设置
    location ~ \.well-known{
        allow all;
    }

    # location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
    # {
```

```
    #     expires      30d;
    #     error_log /dev/null;
    #     access_log /dev/null;
    # }

    # JNPF-START
    #设置上传文件的大小
    # client_max_body_size 100m;

    # #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    # #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    # client_header_buffer_size 128k;

    # #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
    # large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    # 预检命令的缓存，如果不缓存每次会发送两次请求
    #带cookie请求需要加上这个字段，并设置为true
    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号     #
表示请求头的字段 动态获取
    # 前端主项目(jnpf-web)伪静态
    location / {
      try_files $uri $uri/ /index.html;
    }

    # 前端大屏(jnpf-web-datascreen)伪静态
    location /DataV {
      try_files $uri $uri/ /DataV/index.html;
    }

    # 主项目
    location /api/ {
      proxy_pass http://localhost:30000;
    }

    location /websocket {
      proxy_pass http://localhost:30000/api/message/websocket;
      proxy_http_version 1.1;
```

```
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_read_timeout 600s;
      }

      # 报表（jnpf-datareport）接口
      location /ReportServer/ {
       proxy_pass http://localhost:30007/;
      }

      # 文件预览（jnpf-file-preview）
      location /FileServer {
        proxy_pass http://localhost:30090;
      }

      location ~ /FileServer/*.*\.(js|css)?$ {
                          proxy_pass http://localhost:30090;
      }
      # JNPF-End

      location ~ .*\.(js|css)?$
      {
          expires      12h;
          error_log /dev/null;
          access_log /dev/null;
      }
  }
```

3.APP的接口转发配置，如下：

```
    #JNPF-Start

    #设置上传文件的大小
    client_max_body_size 100m;

    #添加头部信息
    proxy_set_header Cookie $http_cookie;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    #请求头总长度大于128k时使用large_client_header_buffers设置的缓存区
    client_header_buffer_size 128k;

    #指令参数4为个数，128k为大小，默认是8k。申请4个128k。
```

```
    large_client_header_buffers 4 128k;

    #指定允许跨域的方法，*代表所有
    add_header Access-Control-Allow-Methods 'GET,PUT,POST,DELETE,OPTIONS';

    # 预检命令的缓存，如果不缓存每次会发送两次请求
    add_header Access-Control-Max-Age 3600;

    #带cookie请求需要加上这个字段，并设置为true
    add_header Access-Control-Allow-Credentials true;

    #    表示允许这个域跨域调用（客户端发送请求的域名和端口）
    #    $http_origin动态获取请求客户端请求的域    不用*的原因是带cookie的请求不支持*号
    add_header Access-Control-Allow-Origin $http_origin;
    #    表示请求头的字段 动态获取
    add_header Access-Control-Allow-Headers $http_access_control_request_headers;


    # 伪静态
    location / {
      try_files $uri $uri/ /index.html;
    }

    # 后端接口
    location /api/ {
      proxy_pass http://localhost:30000;
    }

    # websocket
    location /websocket {
      proxy_pass http://localhost:30000/api/message/websocket;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
      proxy_read_timeout 600s;
    }
    #JNPF-End
```

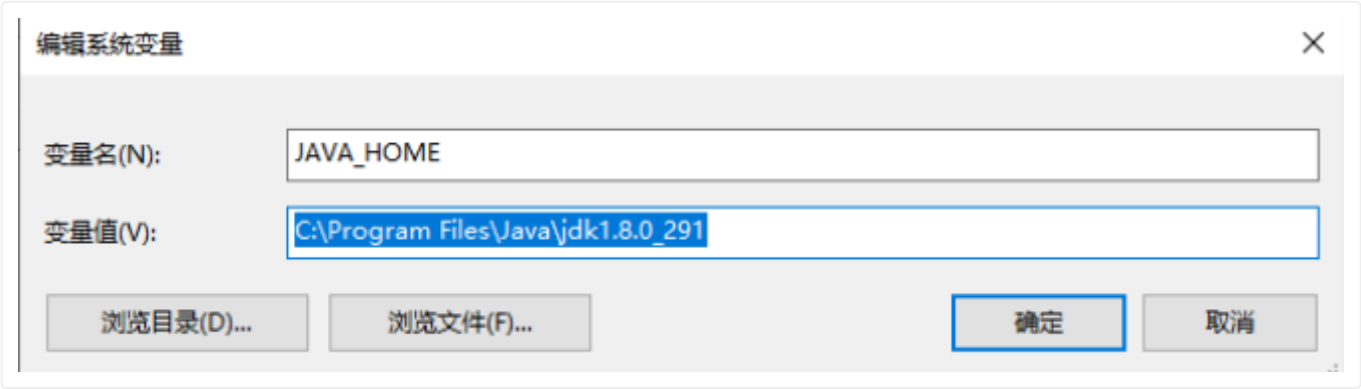
重启ngin.exe。

3. Java安装

1).JDK下载http://www.oracle.com/technetwork/java/javase/downloads/index.html
或者coding网盘下载(下载1.8.291版本)

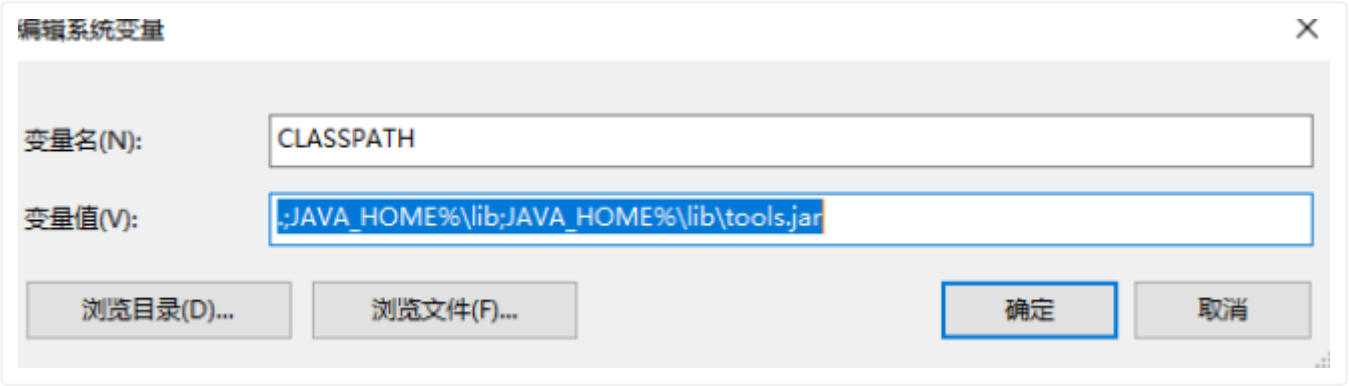
2).默认安装直至最后一步

3).点击系统变量下面的新建按钮，变量名JAVA_HOME（代表你的JDK安装路径），值对应的是你的JDK的安装路径。



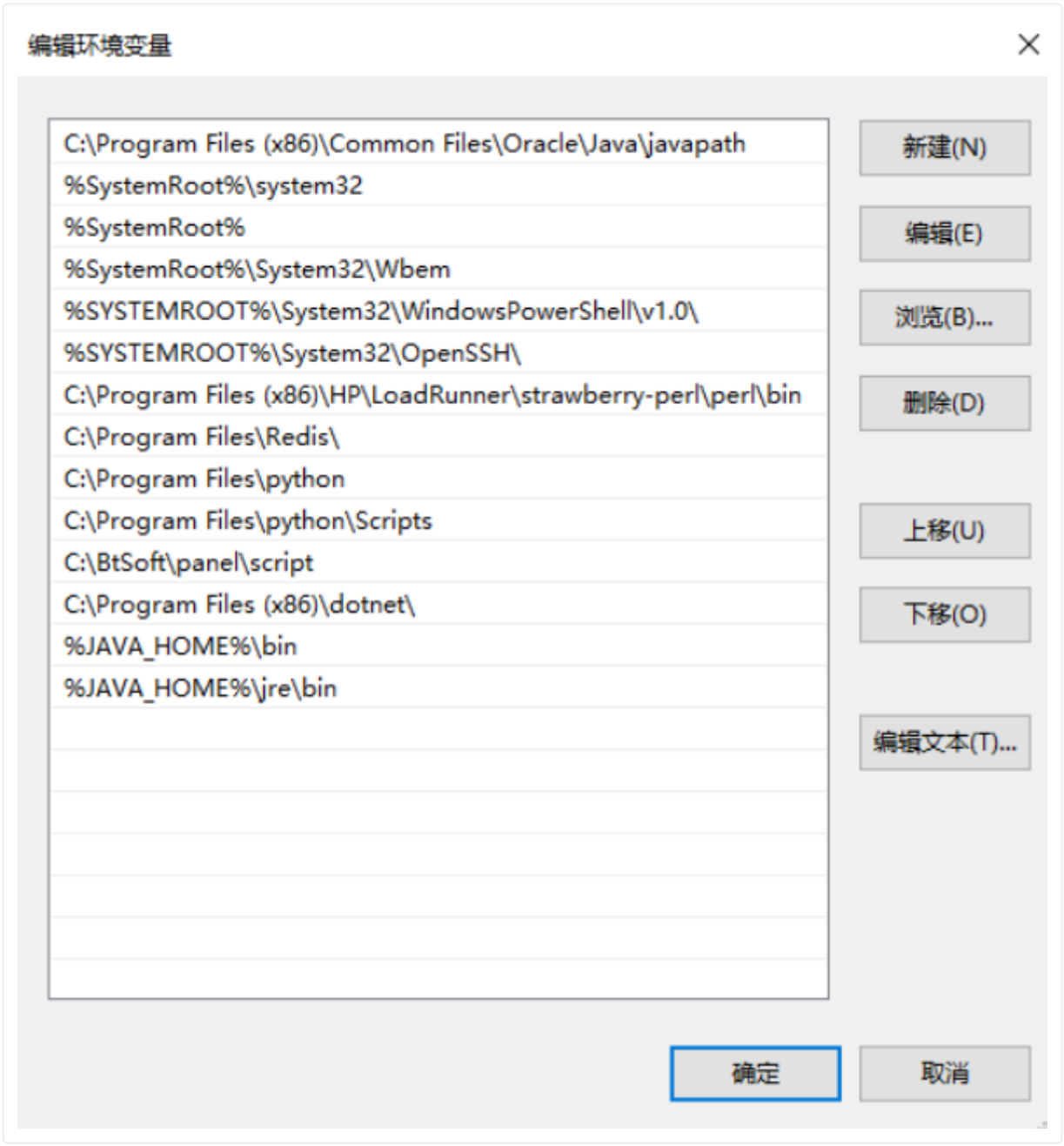
4).继续在系统变量里面新建一个CLASSPATH变量，其变量值如下图所示：

```
.;JAVA_HOME%\lib;JAVA_HOME%\lib\tools.jar
```



5).在你的系统变量里面找一个变量名是PATH的变量，需要在它的值域里面追加一段如下的代码：

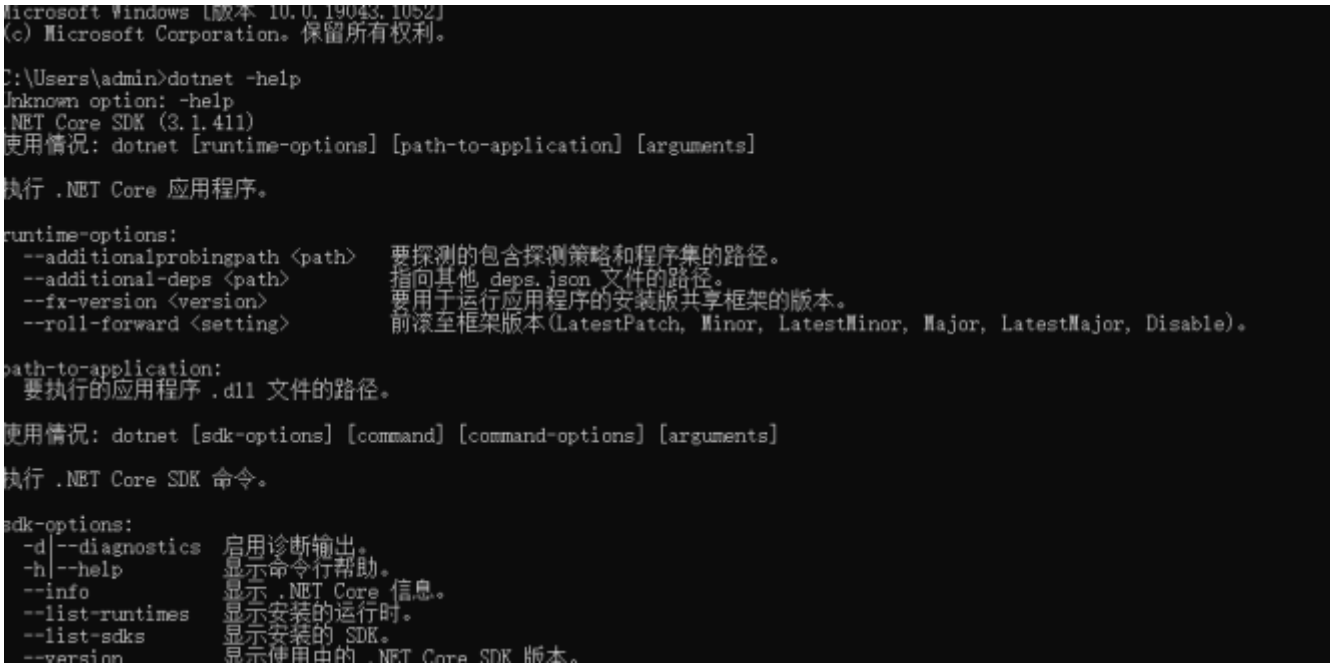
```
%JAVA_HOME%\bin
%JAVA_HOME%\jre\bin
```



6).测试自己所配置的环境变量是否正确，WINDOWS+R键，输入cmd，进入命令行界面，如下所示：

4.ASP.NET Core SDK安装

下载：https://dotnet.microsoft.com/download/dotnet/6.0
下载后默认安装直至最后一步。
WINDOWS+R键，输入cmd，命令窗口输入dotnet -help，如图显示安装成功。



注：liunx安装.net SDK工具：https://docs.microsoft.com/zh-cn/dotnet/core/install/linux-centos

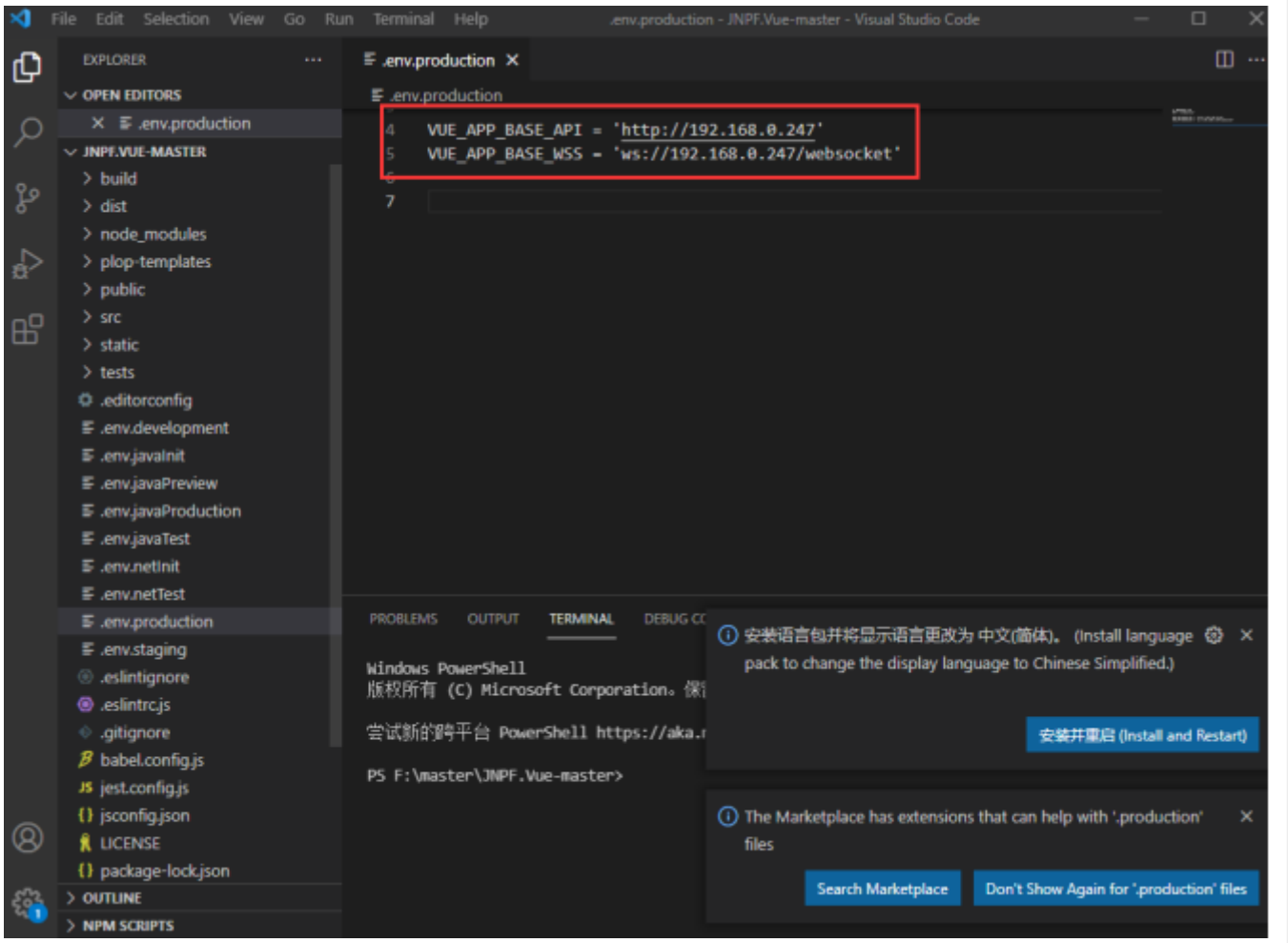
5.主项目前端项目部署

`Visual studio cood` 打开前端项目：

安装依赖

```
npm i
```

修改.env.production数据：



打包

```
npm run build
```

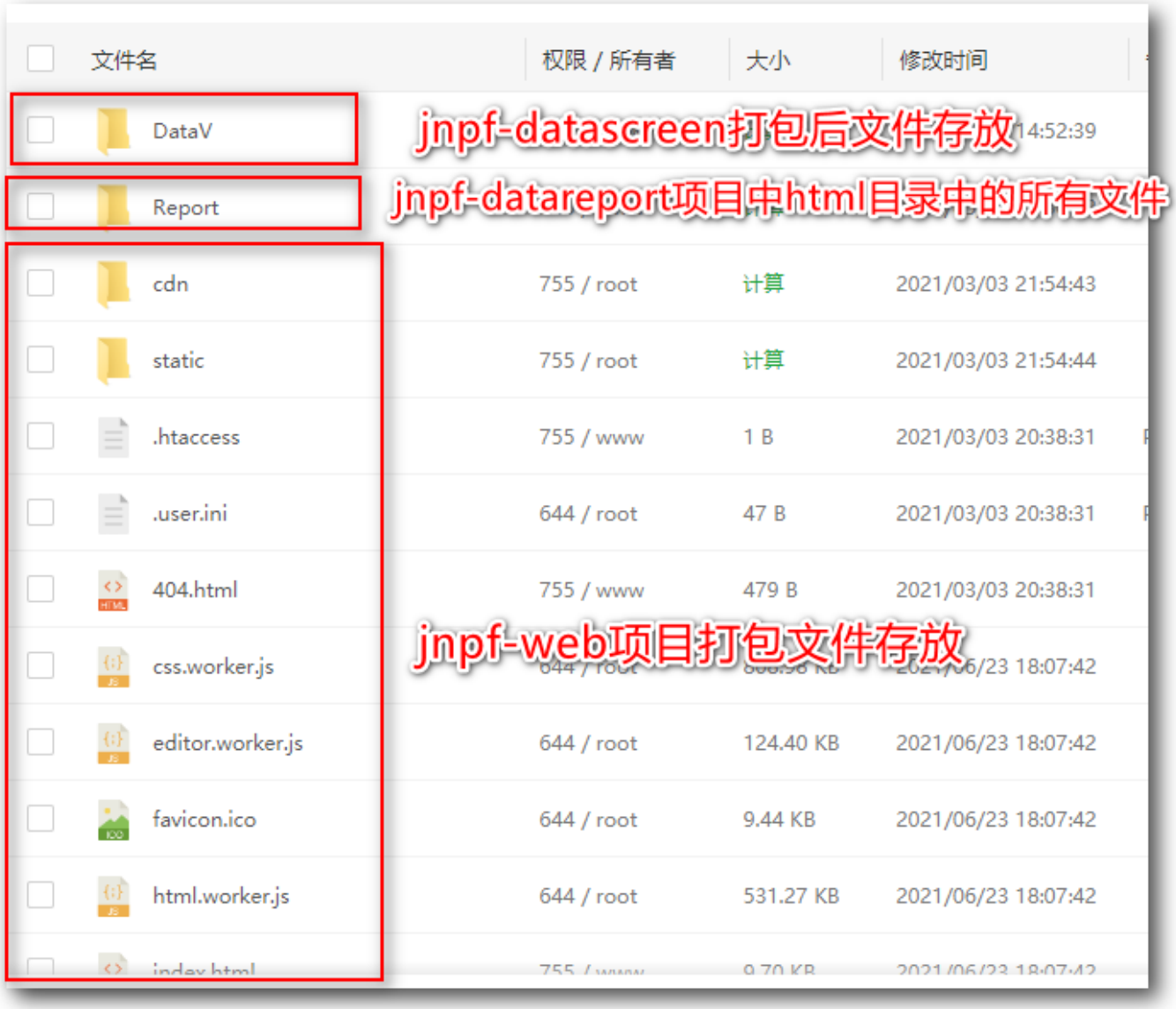
上传目录dist文件夹下全部文件到指定目录（或者配置nginx.cof）

```
server
{
    listen 80;
    listen 443 ssl http2;
    server_name 192.168.0.247;
    index index.php index.html index.htm default.p
    root c:\wwwroot\jnpf-web;
```

重启nginx访问即可。

6.报表前端部署

前端部署结构说明




```
├── jnpf-web                    # 假设这个目录是存放测试或生产环境的前端
│   ├── DataV                   # 大屏(`jnpf-datascreen`)打包后文件存放目录
│   ├── Report                  # 报表(`jnpf-datareport`)html下的文件
│   └── 主项目前端打包后的文件    # 主项目(`jnpf-web`)打包后存放在根目录
```

前端

- 将 `jnpf-web-datareport` 下的 `html` 文件夹中的所有拷贝到 `jnpf-web` 中的 `Report` 目录下, 如 `Report` 目录不存在请手动建立
- 接口配置
  - 打开 `/Report/index.html` ,做如下修改

```
# 在index.html文件中第24行开始
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.247/ReportServer";
// 报表前端
```

```
window._contextPath = "http://192.168.0.247/Report";
// 主项目接口地址
window._mainServer = "http://192.168.0.247";
</script>
```

- 打开 `/Report/preview.html` ,做如下修改,

```
# 在preview.html文件中第86行
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.247/ReportServer";
</script>
```

- 打开 `/Report/searchform.html` ,做如下修改,

```
# 在searchform.html文件中
<script type="text/javascript">
// 报表接口
window._server = "http://192.168.0.247/ReportServer";
// 报表前端
window._contextPath = "http://192.168.0.247/Report";
</script>
```

- 配置说明:
  - 本示例中， `http://192.168.0.0:247` 为项目在测试环境中访问入口，在部署中根据实际情况调整;
  - 报表接口中， `ReportServer` 为虚拟目录，需要在Nginx增加相关配置，具体配置如下:

```
# 数据报表接口配置
  location /ReportServer/ {
     proxy_pass http://localhost:30007/;
  }
```

7.大屏前端部署

配置

在项目根目录打开 `.env.staging` (测试环境配置)，部署生产环境请打开 `.env.product` 文件

```
# 测试默认配置
ENV = 'staging'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.247'
```

```
# 生产环境默认配置
ENV = 'production'

# 前端接口
VUE_APP_BASE_API = 'http://192.168.0.247'
```

构建

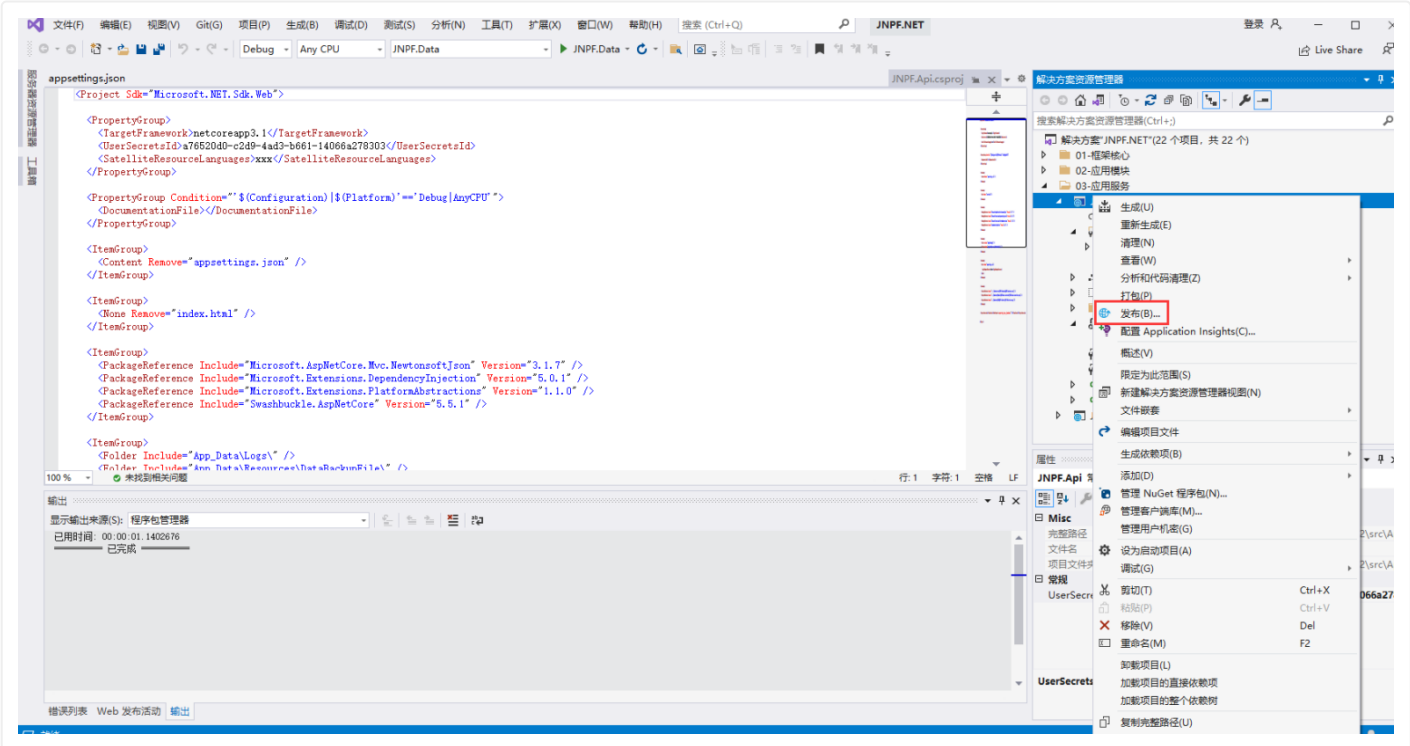
```
# 构建测试环境
yarn build:staging

# 构建生产环境
yarn build
```

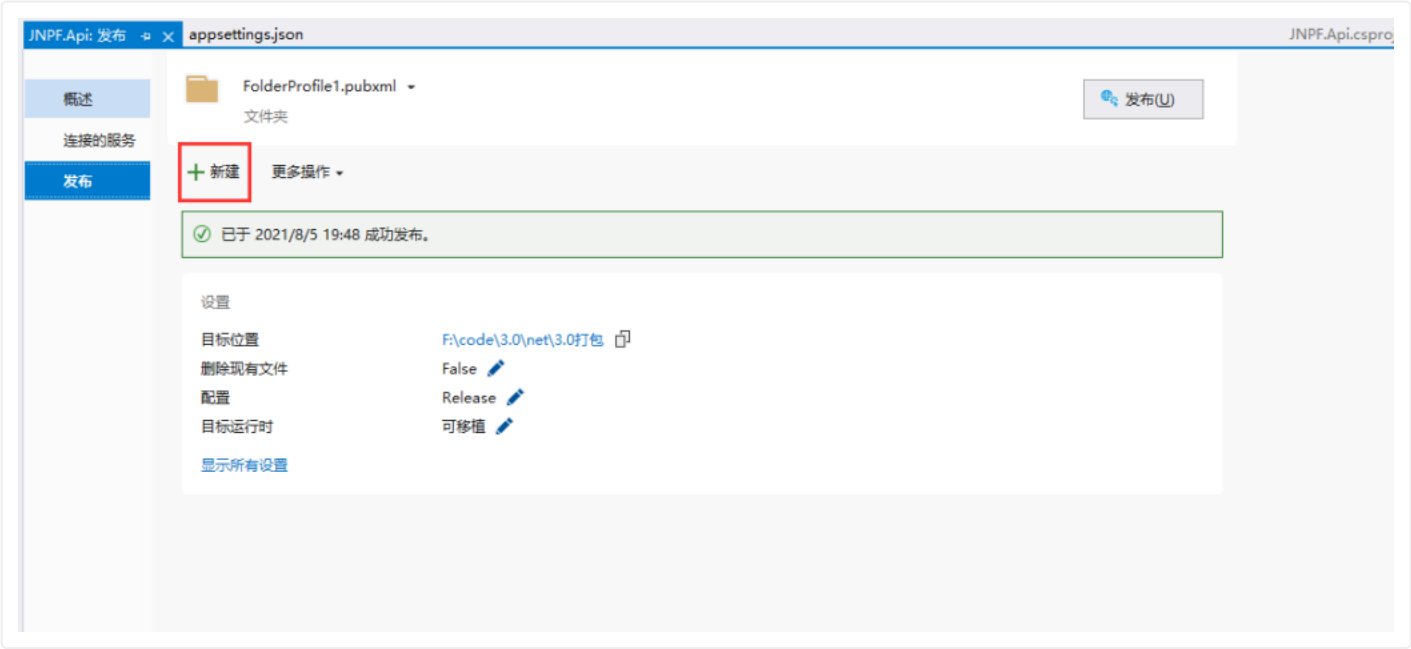
发布到服务器

- 压缩dist目录，上传到服务器C:\wwwroot\jnpf-web\DataV并解压

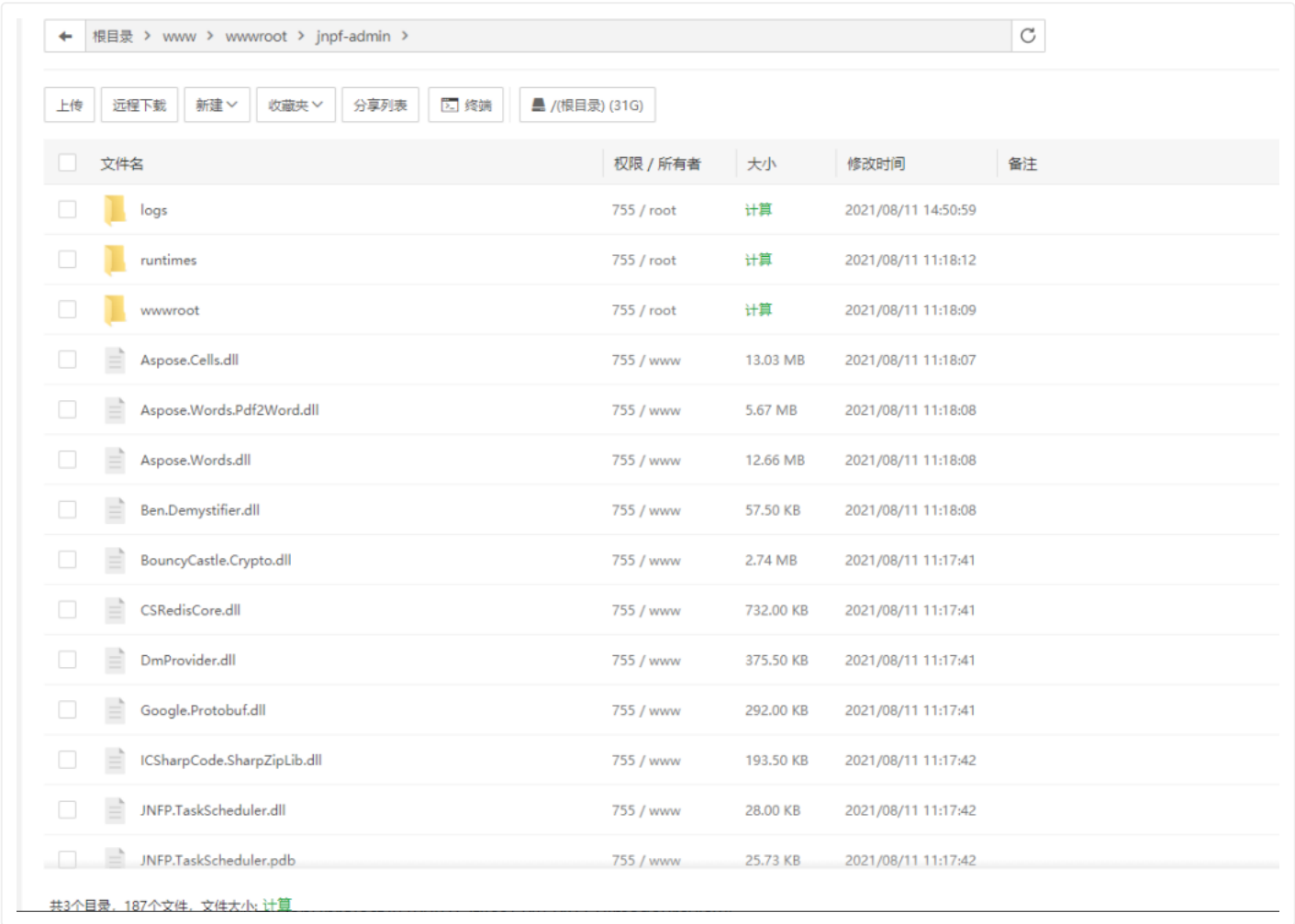
8.后端项目打包

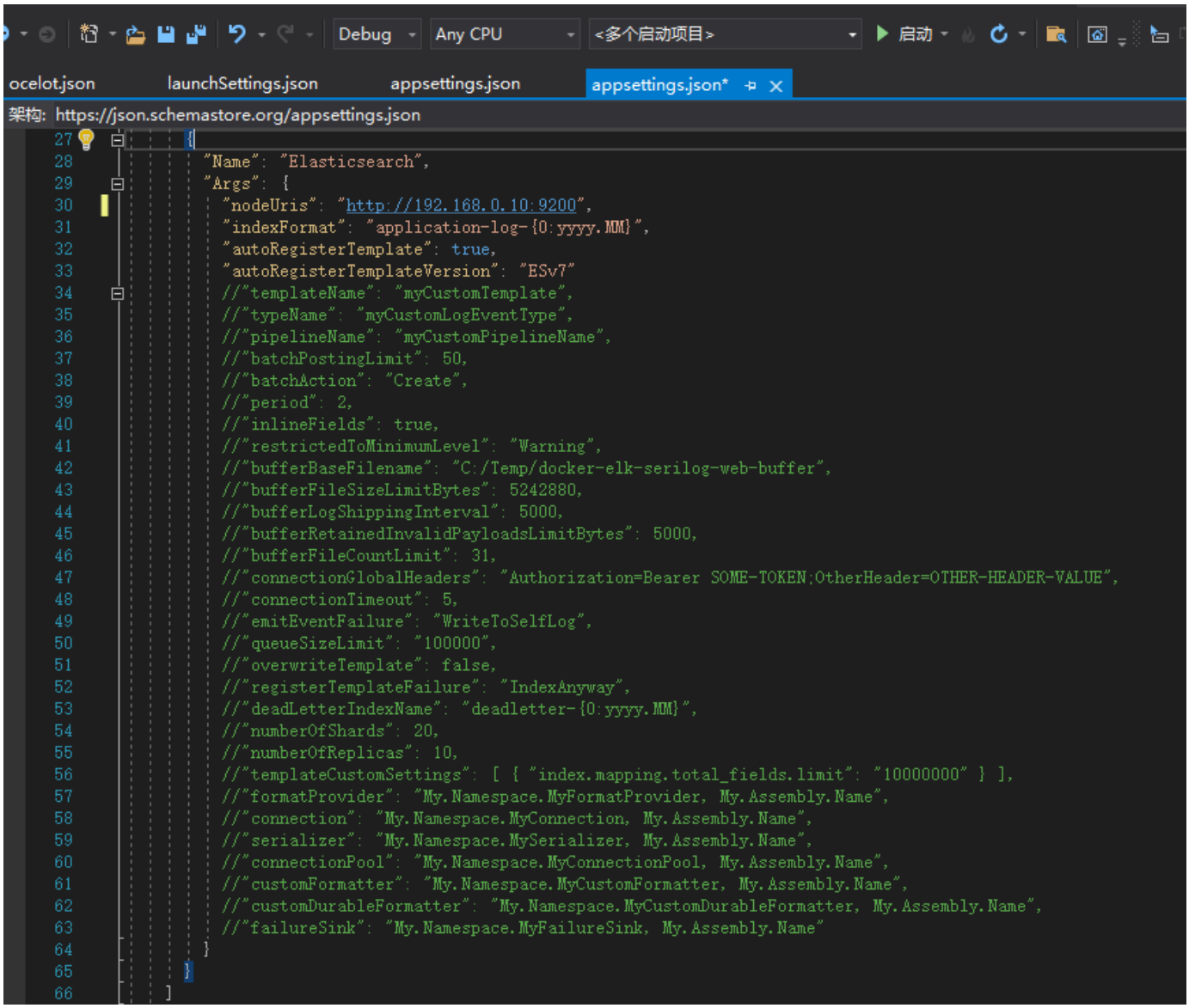
Visual studio 2019打开后端项目，应用服务里面的每一个服务 右键选择“发布”





发布文件拷贝指定目录（新建文件夹），修改appsetting.josn 配置：






```
        "Name": "Elasticsearch",
        "Args": {
          "nodeUris": "http://192.168.0.247:9200",
```

```
          "indexFormat": "application-log-{0:yyyy.MM}",
          "autoRegisterTemplate": true,
          "autoRegisterTemplateVersion": "ESv7"
          //"templateName": "myCustomTemplate",
          //"typeName": "myCustomLogEventType",
          //"pipelineName": "myCustomPipelineName",
          //"batchPostingLimit": 50,
          //"batchAction": "Create",
          //"period": 2,
          //"inlineFields": true,
          //"restrictedToMinimumLevel": "Warning",
          //"bufferBaseFilename": "C:/Temp/docker-elk-serilog-web-buffer",
          //"bufferFileSizeLimitBytes": 5242880,
          //"bufferLogShippingInterval": 5000,
          //"bufferRetainedInvalidPayloadsLimitBytes": 5000,
          //"bufferFileCountLimit": 31,
          //"connectionGlobalHeaders": "Authorization=Bearer SOME-TOKEN;OtherHeader=
OTHER-HEADER-VALUE",
          //"connectionTimeout": 5,
          //"emitEventFailure": "WriteToSelfLog",
          //"queueSizeLimit": "100000",
          //"overwriteTemplate": false,
          //"registerTemplateFailure": "IndexAnyway",
          //"deadLetterIndexName": "deadletter-{0:yyyy.MM}",
          //"numberOfShards": 20,
          //"numberOfReplicas": 10,
          //"templateCustomSettings": [ { "index.mapping.total_fields.limit": "10000
000" } ],
          //"formatProvider": "My.Namespace.MyFormatProvider, My.Assembly.Name",
          //"connection": "My.Namespace.MyConnection, My.Assembly.Name",
          //"serializer": "My.Namespace.MySerializer, My.Assembly.Name",
          //"connectionPool": "My.Namespace.MyConnectionPool, My.Assembly.Name",
          //"customFormatter": "My.Namespace.MyCustomFormatter, My.Assembly.Name",
          //"customDurableFormatter": "My.Namespace.MyCustomDurableFormatter, My.Ass
embly.Name",
          //"failureSink": "My.Namespace.MyFailureSink, My.Assembly.Name"
        }
```

```
  "Consul": {
    "ServiceName": "OAuth",
    "ServiceIP": "localhost",
    "ServicePort": 5001,
    "ServiceHealthCheck": "http://192.168.0.247:5001/HealthCheck",
    "ConsulAddress": "http://127.0.0.1:8500"
  },
```

```
//Apollo的配置
"Apollo": {
  "AppId": "JNPF.NET.Cloud",
  "Env": "DEV",
  "MetaServer": "http://192.168.0.247:8080",
  "ConfigServer": [ "http://192.168.0.247:8080" ],
  "Namespaces": [ "JNPF.NET", "application.json", "application" ]
}
```

部署Apollo

详情请看：服务器环境部署 -> Apollo部署文档

部署consul

详情请看：服务器环境部署 -> consul部署文档

后端运行

- 命令运行：

```
dotnet JNPF.Ocelot.Entry.dll --urls=http://*:5000
#网关服务运行命令
```

```
dotnet JNPF.Oauth.Entry.dll --urls=http://*:5001
#鉴权服务运行命令
```

```
dotnet JNPF.System.Entry.dll --urls=http://*:5002
#System服务运行命令
```

```
dotnet JNPF.Message.Entry.dll --urls=http://*:5003
#消息服务运行命令
```

```
dotnet JNPF.TaskScheduler.Entry.dll --urls=http://*:5004
#系统调度服务运行命令
```

```
dotnet JNPF.WorkFlow.Entry.dll --urls=http://*:5005
#WorkFlow服务运行命令
```

```
dotnet JNPF.VisualDev.Entry.dll --urls=http://*:5006
#VisualDev服务运行命令
```

```
dotnet JNPF.VisualData.Entry.dll --urls=http://*:5007
#大屏服务运行命令
```

```
dotnet JNPF.Extend.Entry.dll --urls=http://*:5008
#扩展服务运行命令
```

```
dotnet JNPF.SubDev.Entry.dll --urls=http://*:5009
#SubDev服务运行命令
```

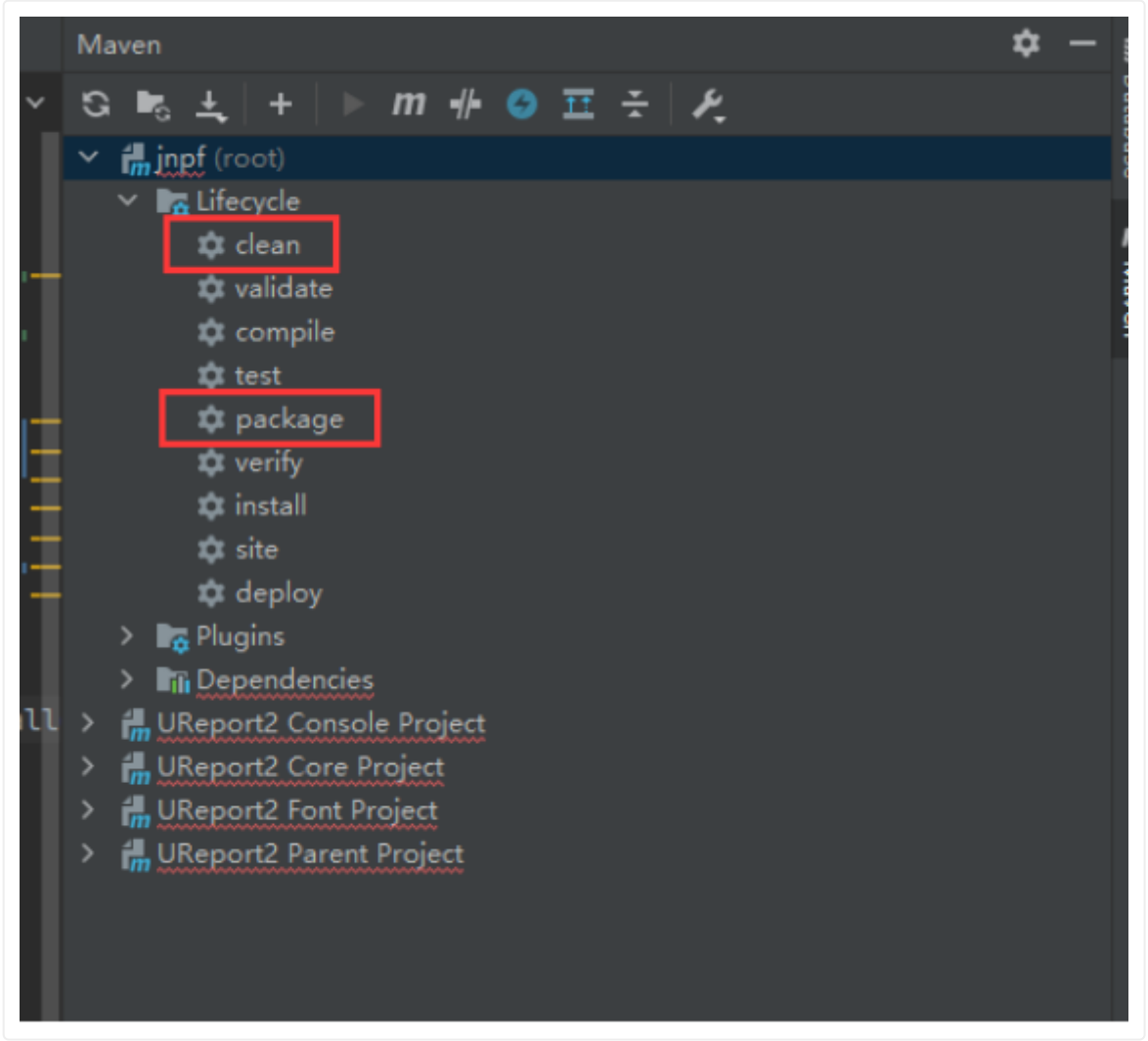
```
dotnet JNPF.Apps.Entry.dll --urls=http://*:5011
#App服务运行命令
```

- 停止运行：

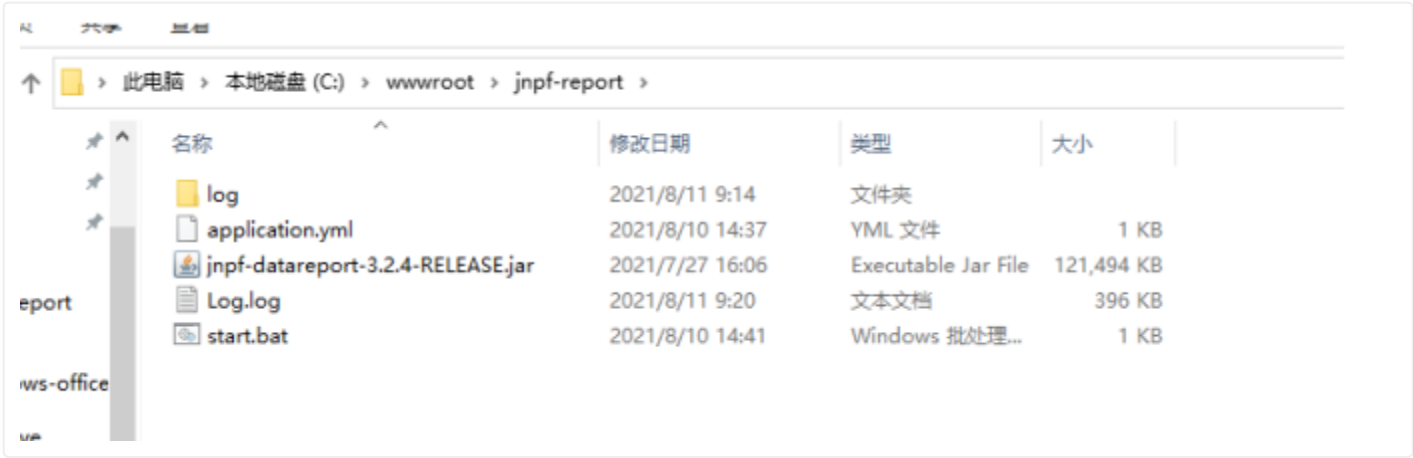
```
kill -9 $(netstat -nlp | grep 5000 | awk '{print $7}' | awk -F"/" '{ print $1 }'
)
```

9.报表后端项目打包

Intellij IDEA打开报表后端项目jnpf-datareport,打包



上传指服务器指定目录下（可以新建）



打开application.Yml文件，修改数据库配置并保存：

```yaml
# 配置端口
server:
  port: 30007
  max-http-header-size: 102400
  compression:
    enabled: true
    min-response-size: 102400
spring:
  # 表空间(Oracle)
  tableSpace: JNPFCLOUD
  datasource:
    # MySQL配置
    druid:
      dbinit: jnpf_init
      dbname: jnpf_init
      dbnull: jnpf_init
      url: jdbc:mysql://localhost:3306/{dbName}?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8
      username: root
      password: 123456
      driver-class-name: com.mysql.cj.jdbc.Driver

  #Redis配置
  redis:
    database: 1
    host: localhost
    port: 6379
    password:
    timeout: 3000
    lettuce:
      pool:
        max-active: 8 # 连接池最大连接数
        max-wait: -1ms  # 连接池最大阻塞等待时间（使用负值表示没有限制）
        min-idle: 0 # 连接池中的最小空闲连接
        max-idle: 8 # 连接池中的最大空闲连接
logging:
  level:
    root: info
    com.bstek.ureport.console: debug
  file:
    path: log
config:
```

MySQL配置：

```yaml
    druid:
      dbinit: jnpf_init
      dbname: jnpf_init
      dbnull: jnpf_init
      url: jdbc:mysql://localhost:3306/{dbName}?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8
      username: root
      password: 123456
      driver-class-name: com.mysql.cj.jdbc.Driver
```

SQLServer配置:

```yaml
    druid:
      dbinit: jnpf_init
      dbname: jnpf_init
      dbnull: jnpf_init
      url: jdbc:sqlserver://localhost:1433;DatabaseName={dbName}
      username: sa
      password: JNPF@2020
      driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
```

Oracle配置:

```
  # 表空间(Oracle)
  tableSpace: JNPFCLOUD
```

```
    druid:
      dbinit: JNPFCLOUD
      dbname: JNPFCLOUD
      dbnull: JNPFCLOUD
      url: jdbc:oracle:thin:@192.168.0.247:1521:{dbName}
      username: JNPFCLOUD
      password: JNPFCLOUD
      driver-class-name: oracle.jdbc.driver.OracleDriver
```

运行.jar文件:

```
java -jar -Dfile.encoding=utf-8 -Xmx500m -Xms500m jnpf-datareport-3.2.9-RELEASE.jar
 -> Log.log
```

注：新建文本，输入命令后，修改后缀为.bat，双击直接运行。
查看日志运行成功即可使用报表模块。

10.文档预览部署

环境要求

- JDK1.8+
- OpenOffice或LiberOffice(Windows下已内置)

部署运行

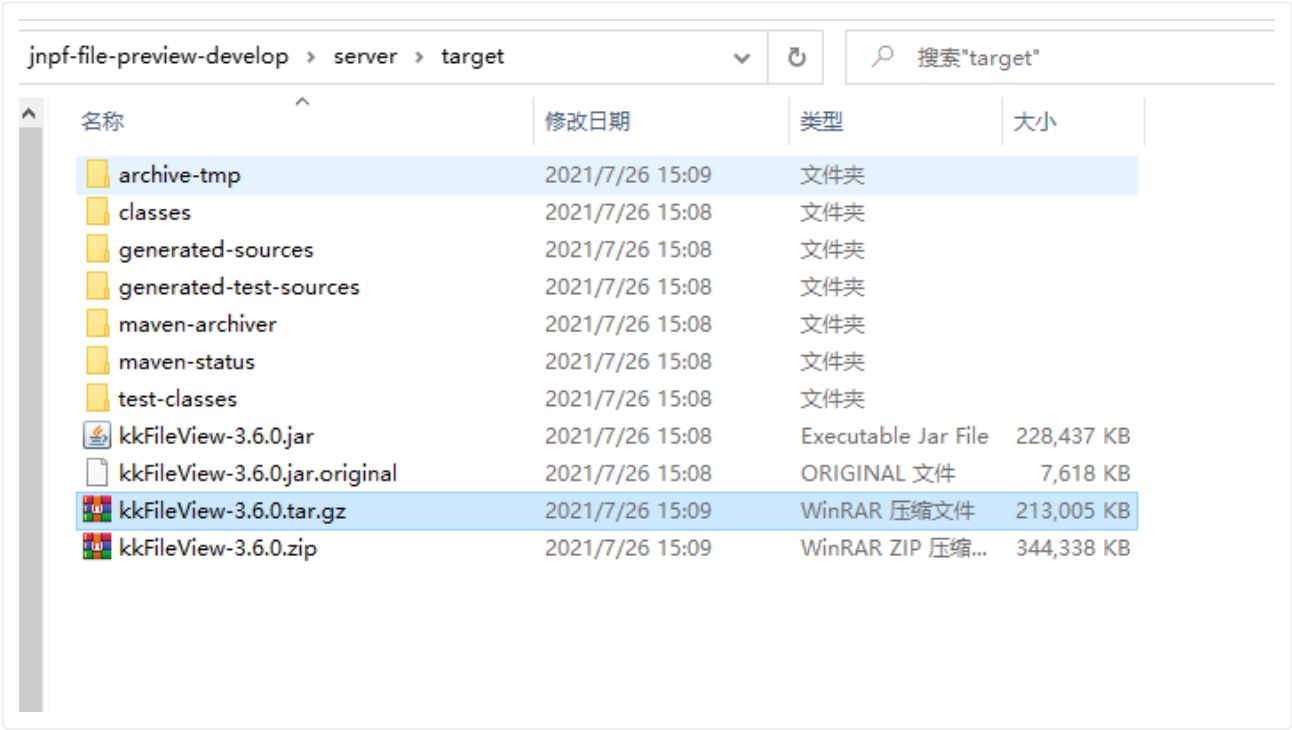
1、开发环境

- a.IDEA导入项目
- b.调整配置，打开 `server/src/main/config/application.properties`
- c.启动项目， `server/src/main/java/cn/keking/ServerMain`
- d.打开 `http://localhost:30090` 测试页面

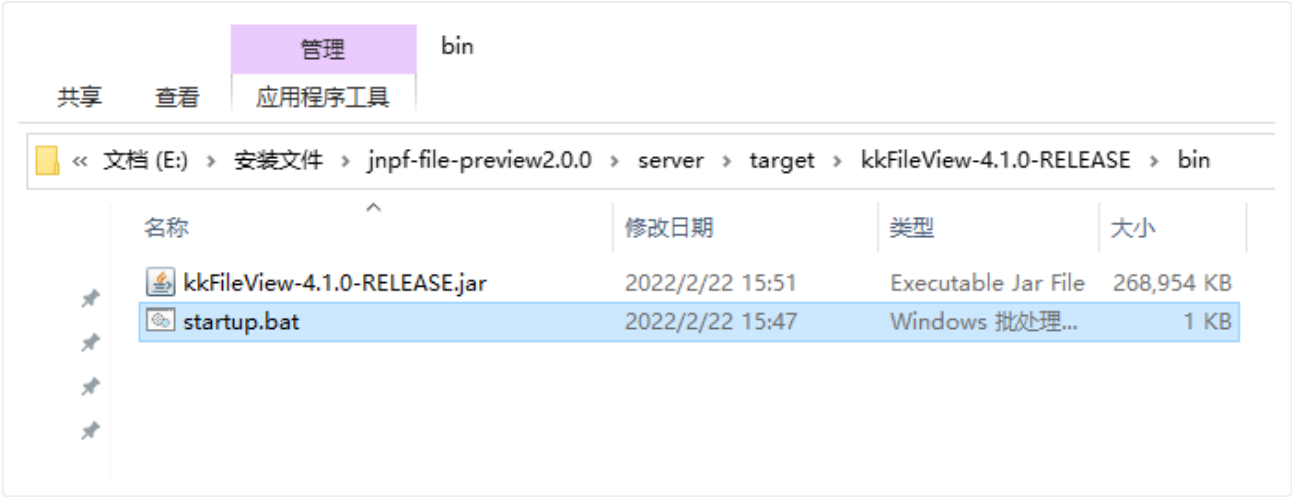
2、测试生产环境

- a.打包

  - 打开 `\server\target` 主要有以下几个文件
  - kkFileView-xxx.jar(一般用于更新)
  - kkFileView-xxx.zip(windows环境下首次部署)
  - kkFileView-xxx.tar.gz(Linux环境下首次部署)



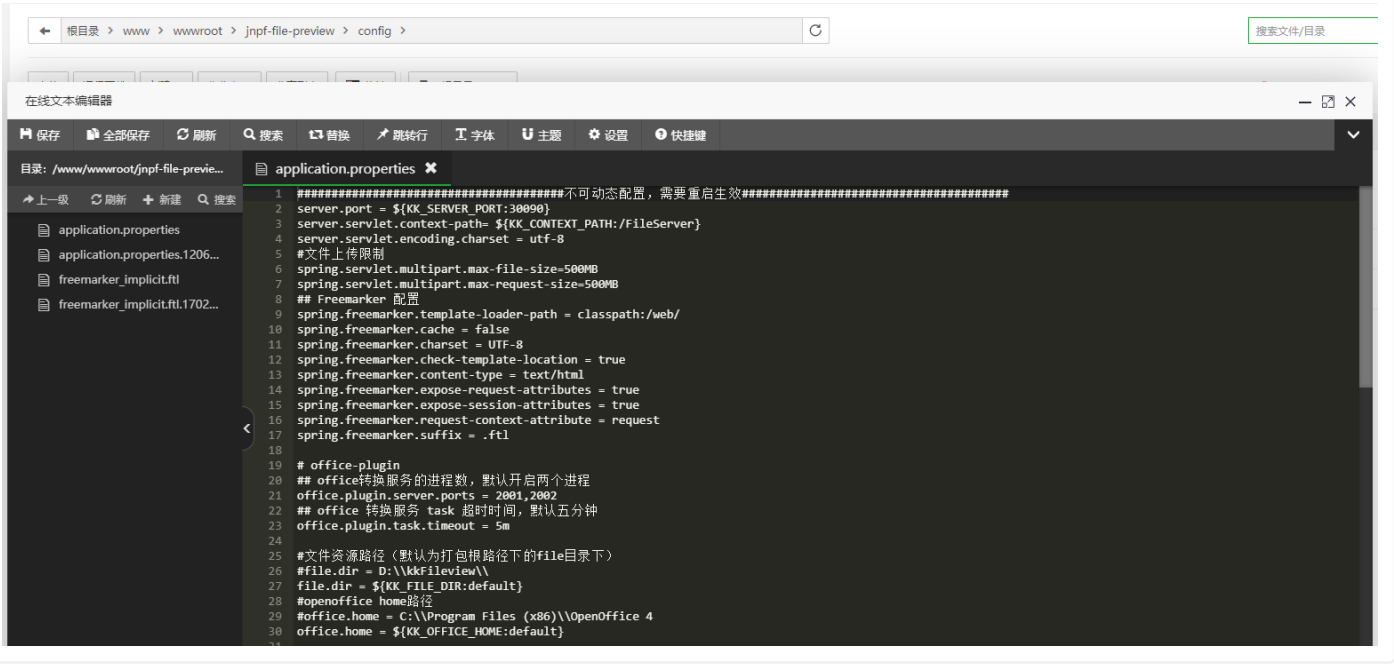
- b.上传至服务器
  - 打开解压后文件夹的bin目录，运行startup脚本(首次部署，之后更新及维护都在bin目录下)



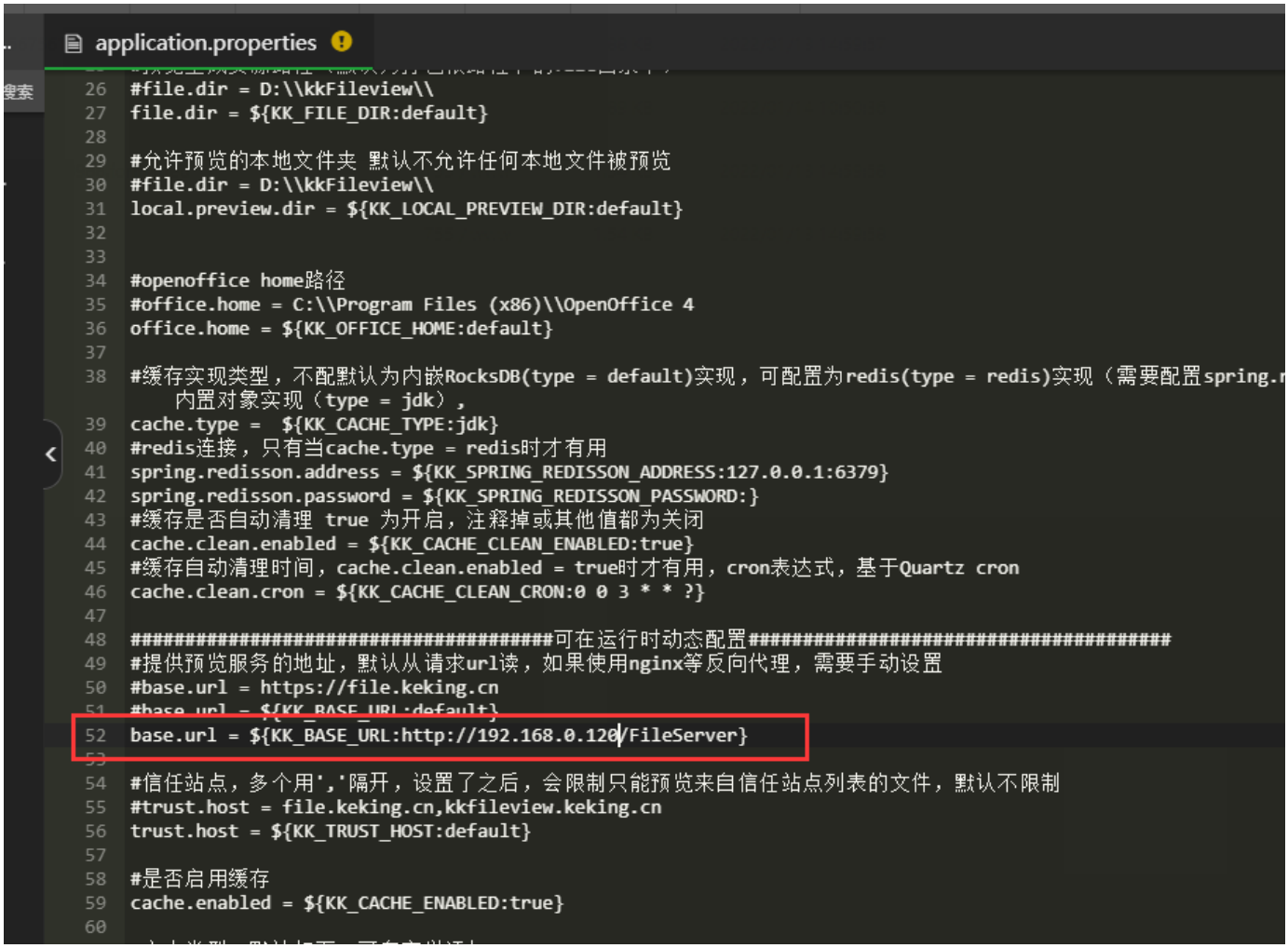
- 修改文件预览配置文件如下两项,打开服务器上的`kkFileView-xxx/config/application.properties`（第3行和第45行）

修改成:

```
server.servlet.context-path= ${KK_CONTEXT_PATH:/FileServer}
base.url = ${KK_BASE_URL:http://192.168.0.247/FileServer}
```



新版只需修改第52行：



- c.Nginx配置说明
  例如nginx的访问地址为 `https://netcore.jnpfsoft.com` ,文件预览部署在内网192.168.0.247服务器上，需要在nginx中添加反向代理如下

```
# 文件预览服务
location /FileServer {
  proxy_pass 192.168.0.247:30090;
}

# 解决文件预览服务无法加载js,css问题
location ~ /FileServer/*.*\.(js|css)?$ {
  proxy_pass 192.168.0.247:30090;
}
```

使用指南

1、单文件预览

```
// 要预览文件的访问地址
let previewUrl = 'http://127.0.0.1:8080/file/test.txt'

// 使用
http://127.0.0.1:30090/onlinePreview?url='+encodeURIComponent(Base64.encode(previewUrl))
```

2、http/https下载流url预览

```
// 要预览文件的访问地址
let previewUrl = 'http://127.0.0.1:8080/filedownload?fileId=1'

// 使用
http://127.0.0.1:30090/onlinePreview?url='+encodeURIComponent(Base64.encode(previewUrl))
```

3、更多参考官方文档（ `https://kkfileview.keking.cn/zh-cn/docs/usage.html` ）

11.常见问题

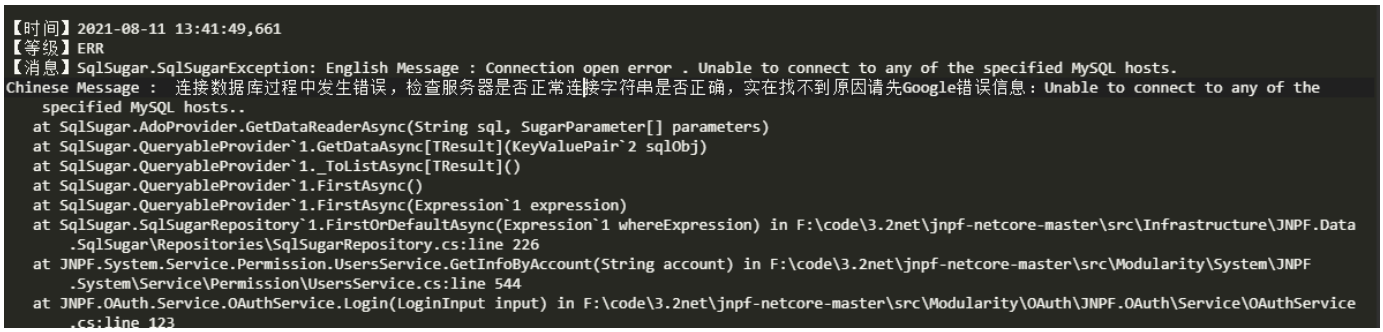
（1）mysql数据库执行查询命令：（解决只有三个菜单问题）

```
update base_module set F_DeleteMark=null
```

（2）数据库添加my.cnf：

```
lower_case_table_names = 1
```

解决mysql大小写转换问题。



解决出现死循环问题：base_user：F_ManagerId清空。

解决Dotnet无法预览问题：

appsettings.json

```
      "app_id": "",
      "app_key": "",
      "redirect_uri": "",
      "scope": "snsapi_userinfo"
    }
  },
  "JNPF_App": {
    "CodeAreasName": "SubDev",
    //系统文件路径(末尾必须带斜杆)
    "SystemPath": "//www//wwwroot//resources//",
    //微信公众号允许上传文件类型
    "MPUploadFileType": "bmp,png,jpeg,jpg,gif,mp3,wma,wav,amr,mp4",
    //微信允许上传文件类型
    "WeChatUploadFileType": "jpg,png,doc,docx,ppt,pptx,xls,xlsx,pdf,txt,rar,zip,csv,amr,mp4",
    //允许图片类型
    "AllowUploadImageType": "jpg,gif,png,bmp,jpeg,tiff,psd,swf,svg,pcx,dxf,wmf,emf,lic,eps,tga",
    //允许上传文件类型
    "AllowUploadFileType": "jpg,gif,png,bmp,jpeg,doc,docx,ppt,pptx,xls,xlsx,pdf,txt,rar,zip,csv",
    "Domain": "http://192.168.0.186",
    "YOZO": {
      "domain": "http://dcsapi.com/",
      "domainKey": "57462250284462899305150"
    },
    //Minio
    "BucketName": "jnpfsoftoss",
    //文件存储类型(本地:local,MinIo:minio,阿里云:aliyun-oss,腾讯云:tencent-cos)
    "FileStoreType": "local",
    //================== 系统错误邮件报告反馈相关 ============================== -->
    //软件的错误报告
    "ErrorReport": "false",
    //软件的错误报告发给谁
    "ErrorReportTo": "yinmaisoft@163.com"
  }
```

附录

在线预览字体乱码

启动脚本添加

```
-Dfile.encoding=UTF-8
```

注：命令窗口不可关闭

Apollo部署文档

> Apollo 官方文档：https://www.apolloconfig.com/#/zh/deployment/quick-start?id=%e4%b8%80%e3%80%81%e5%87%86%e5%a4%87%e5%b7%a5%e4%bd%9c

> 启动 Apollo 需要有bash环境，Windows用户请安装Git Bash，建议使用最新版本

准备工作

1.1 Java运行环境

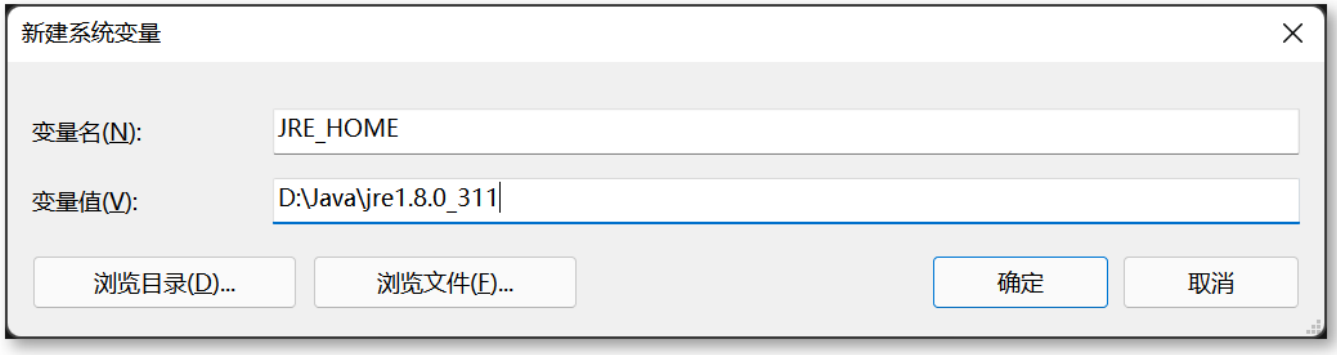
- Apollo服务端：1.8+
- Apollo客户端：1.8+

1.1.1 windows下安装

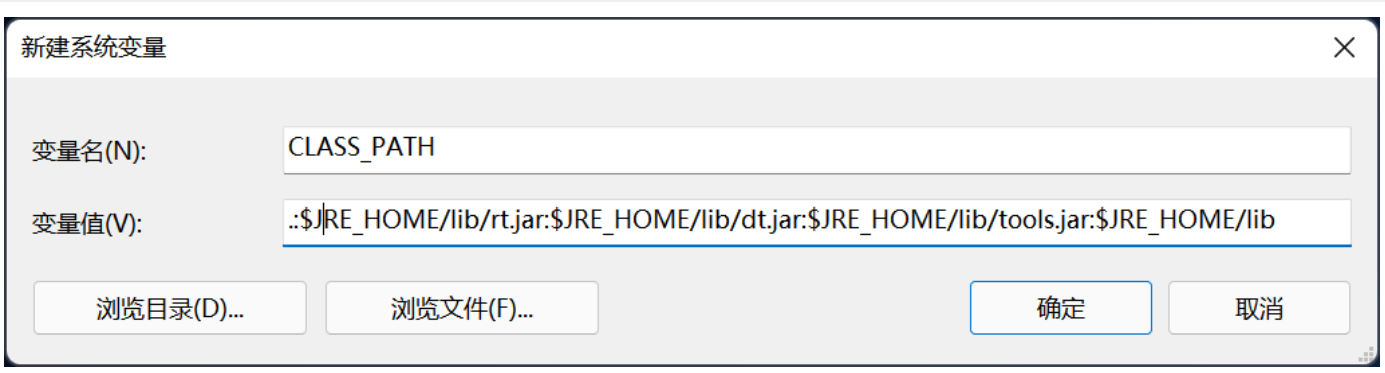
登录Coding并打开 `项目` - `JNPF开发文档` - `文件网盘` - `JAVA环境` - `运行环境` - `windows-x64` 下载 `jre-8u311-windows-x64.exe`

配置环境变量：

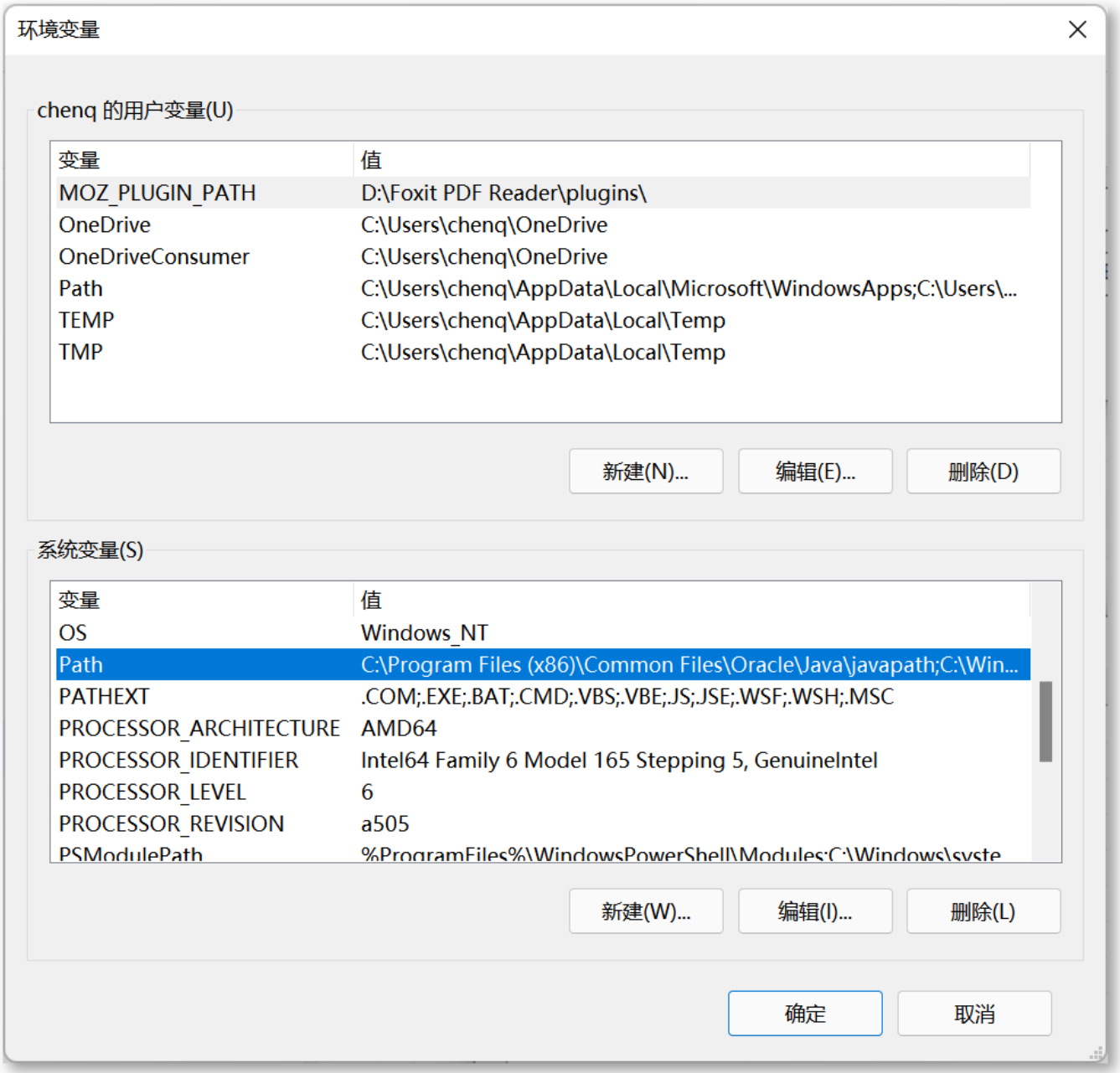
1. 新建系统变量
   - 变量名：JRE_HOME
   - 变量值：D:\Java\jre1.8.0_311

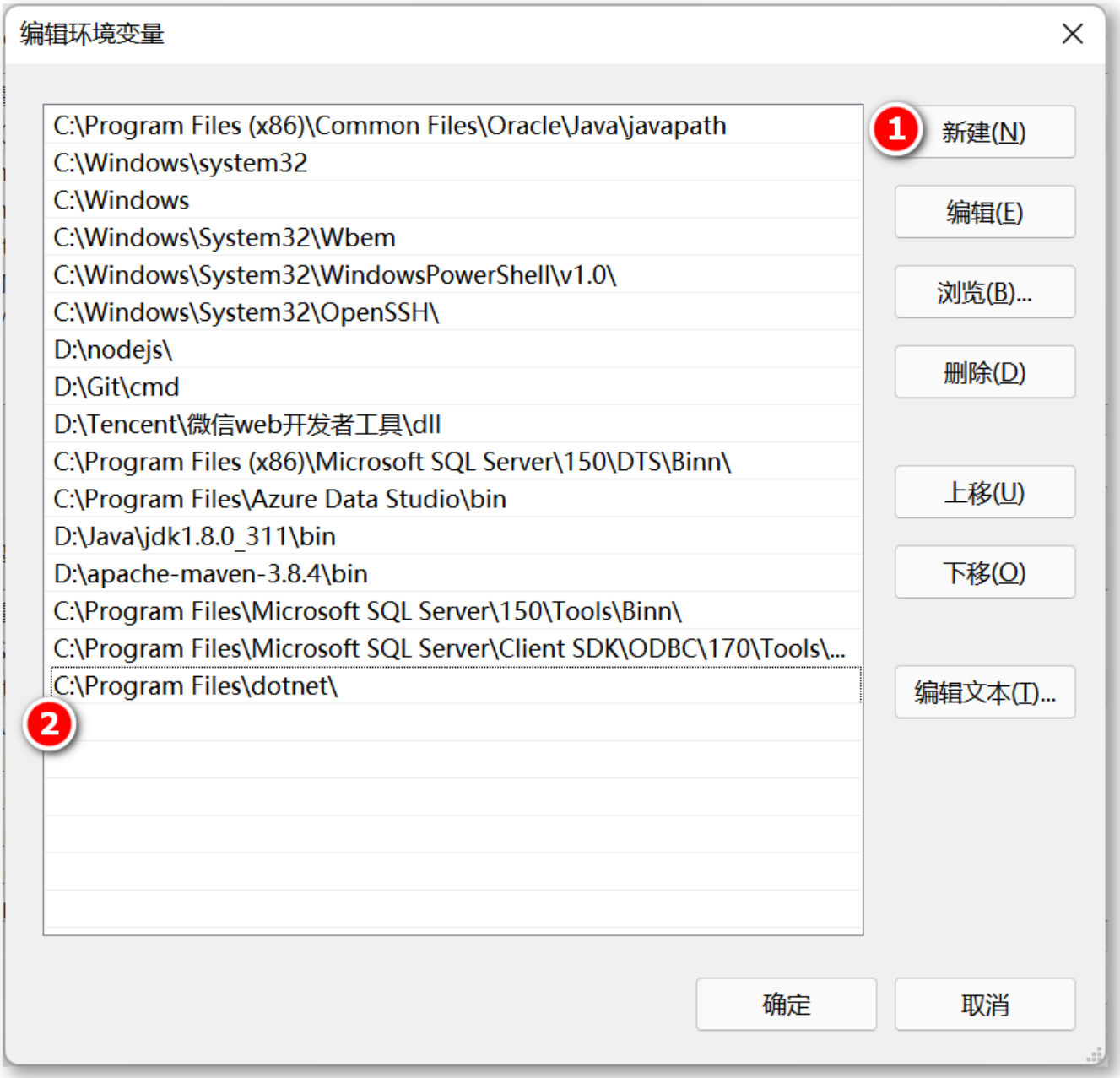


1. 新建系统变量
   变量名：`CLASS_PATH`
   变量
   值：`.:$JRE_HOME/lib/rt.jar:$JRE_HOME/lib/dt.jar:$JRE_HOME/lib/tools.jar:$JRE_HOME/lib`



1. 双击 `系统变量` 中的 `Path` ，点击新建，输入 `%JRE_HOME%\bin`


环境变量
chenq 的用户变量(U)
变量
值
MOZ_PLUGIN_PATH
D:\Foxit PDF Reader\plugins\
OneDrive
C:\Users\chenq\OneDrive
OneDriveConsumer
C:\Users\chenq\OneDrive
Path
C:\Users\chenq\AppData\Local\Microsoft\WindowsApps;C:\Users\...
TEMP
C:\Users\chenq\AppData\Local\Temp
TMP
C:\Users\chenq\AppData\Local\Temp
新建(N)...
编辑(E)...
删除(D)
系统变量(S)
变量
值
OS
Windows_NT
Path
C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Win...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE
AMD64
PROCESSOR_IDENTIFIER
Intel64 Family 6 Model 165 Stepping 5, GenuineIntel
PROCESSOR_LEVEL
6
PROCESSOR_REVISION
a505
PSModulePath
%ProgramFiles%\WindowsPowerShell\Modules;C:\Windows\syste
新建(W)...
编辑(I)...
删除(L)
确定
取消



1.1.2 Linux下安装

登录Coding并打开 `项目` - `JNPF开发文档` - `文件网盘` - `JAVA环境` - `运行环境` - `linux-x64` 下载 `jre-8u311-linux-x64.tar.gz`

将下载的文件上传至服务器并解压：

```
tar -zxvf jre-8u311-linux-x64.tar.gz
```

配置环境变量：

```
# 1.打开配置文件
vi /etc/profile

# 2.在配置文件的最后，复制以下内容，注意路径
```

```
JRE_HOME=/usr/local/src/jre1.8.0_311
CLASSPATH=.:$JRE_HOME/lib/rt.jar:$JRE_HOME/lib/dt.jar:$JRE_HOME/lib/tools.jar:$JRE_HOME/lib
PATH=$PATH:$JRE_HOME/bin:$JRE_HOME/bin
export JRE_HOME PATH CLASSPATH
```

使环境生效

```
source /etc/profile
```

1.1.3

在配置好后，可以通过如下命令检查：

```
java -version
```

样例输出：

```
java version "1.8.0_311"
Java(TM) SE Runtime Environment (build 1.8.0_311-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.311-b11, mixed mode)
```

1.2 MySQL

- 版本要求：5.7+

1.3 下载Apollo运行包

登录Coding并打开 `项目` - `JNPF开发文档` - `文件网盘` - `.NET微服务中间件` - `apollo-server.zip` 保存并解压

安装步骤

2.1 创建数据库

Apollo服务端共需要两个数据库： `ApolloConfigDB` 和 `ApolloPortalDB` ，在解压出来的 `sql` 文件夹中的 `apolloconfigdb.sql` 和 `apolloportaldb.sql` 分别导到相应的数据库中。

导入成功后，可以通过执行以下sql语句来验证：

```
# ApolloConfigDB库中执行以下脚本验证
select `NamespaceId`, `Key`, `Value`, `Comment` from ApolloConfigDB.Item;

# ApolloPortalDB库中执行以下脚本验证
select `Id`, `AppId`, `Name` from ApolloPortalDB.App;
```

2.2 配置数据库连接信息

用文本编辑器打开 `demo.sh` (windows系统下不建议用记事本打开，可以用 `Notepad++` 等代码编辑器打开),修改ApolloPortalDB和ApolloConfigDB相关的数据库连接串信息。

> 注意：填入的用户需要具备对ApolloPortalDB和ApolloConfigDB数据的读写权限。

```
# apollo config db info（如果没有密码，留空即可）
apollo_config_db_url="jdbc:mysql://localhost:3306/ApolloConfigDB?useUnicode=true&cha
racterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8"
apollo_config_db_username=root
apollo_config_db_password=123456

# apollo portal db info（如果没有密码，留空即可）
apollo_portal_db_url="jdbc:mysql://localhost:3306/ApolloPortalDB?useUnicode=true&cha
racterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8"
apollo_portal_db_username=root
apollo_portal_db_password=123456
```

启动Apollo配置中心

> 注意：在启动Apollo配置中心前，请确保 `8070` ， `8080` ， `8090` 端口没有被使用

执行启动脚本：

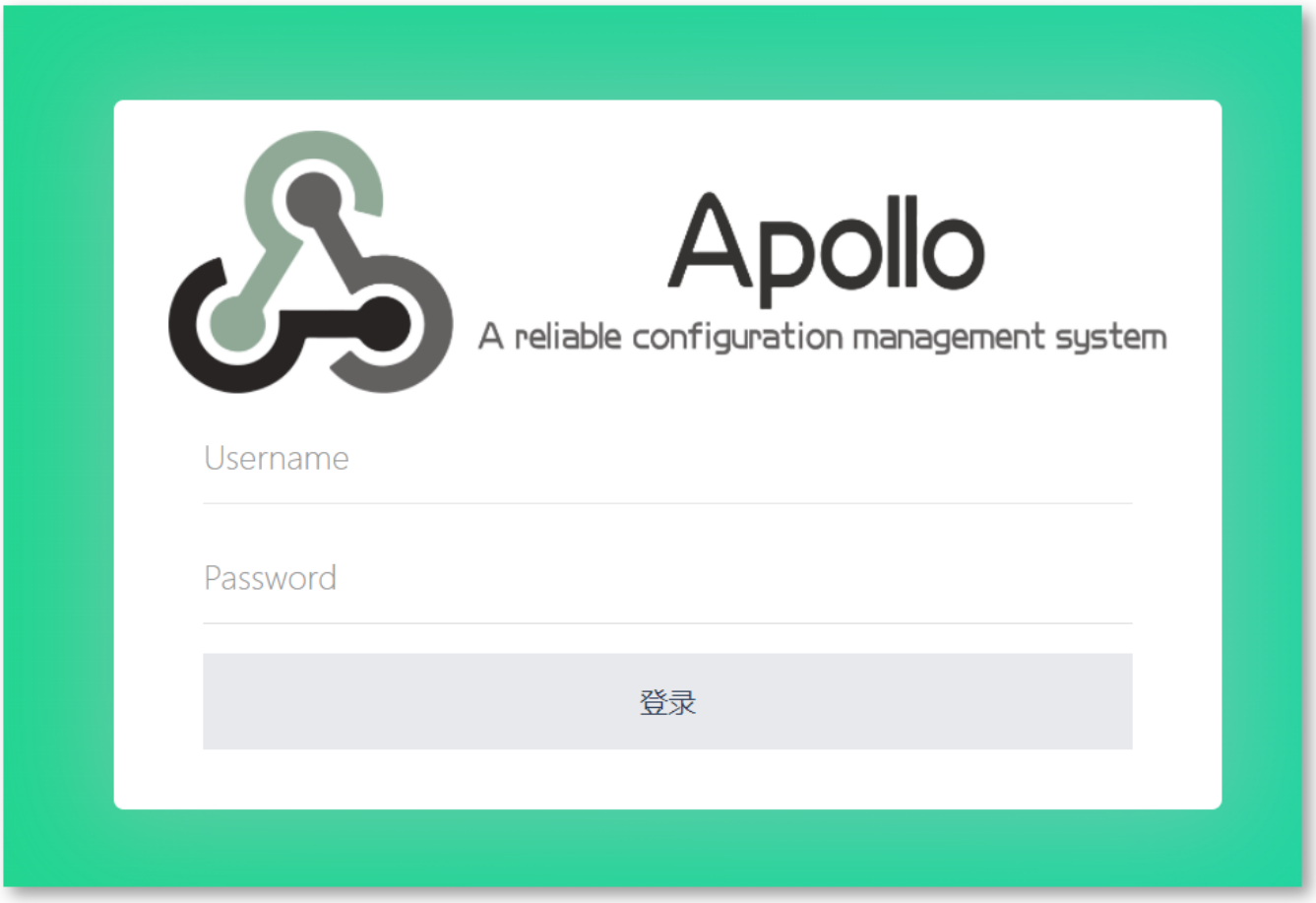
```
./demo.sh start
```

当看到如下输出后，就说明启动成功了！

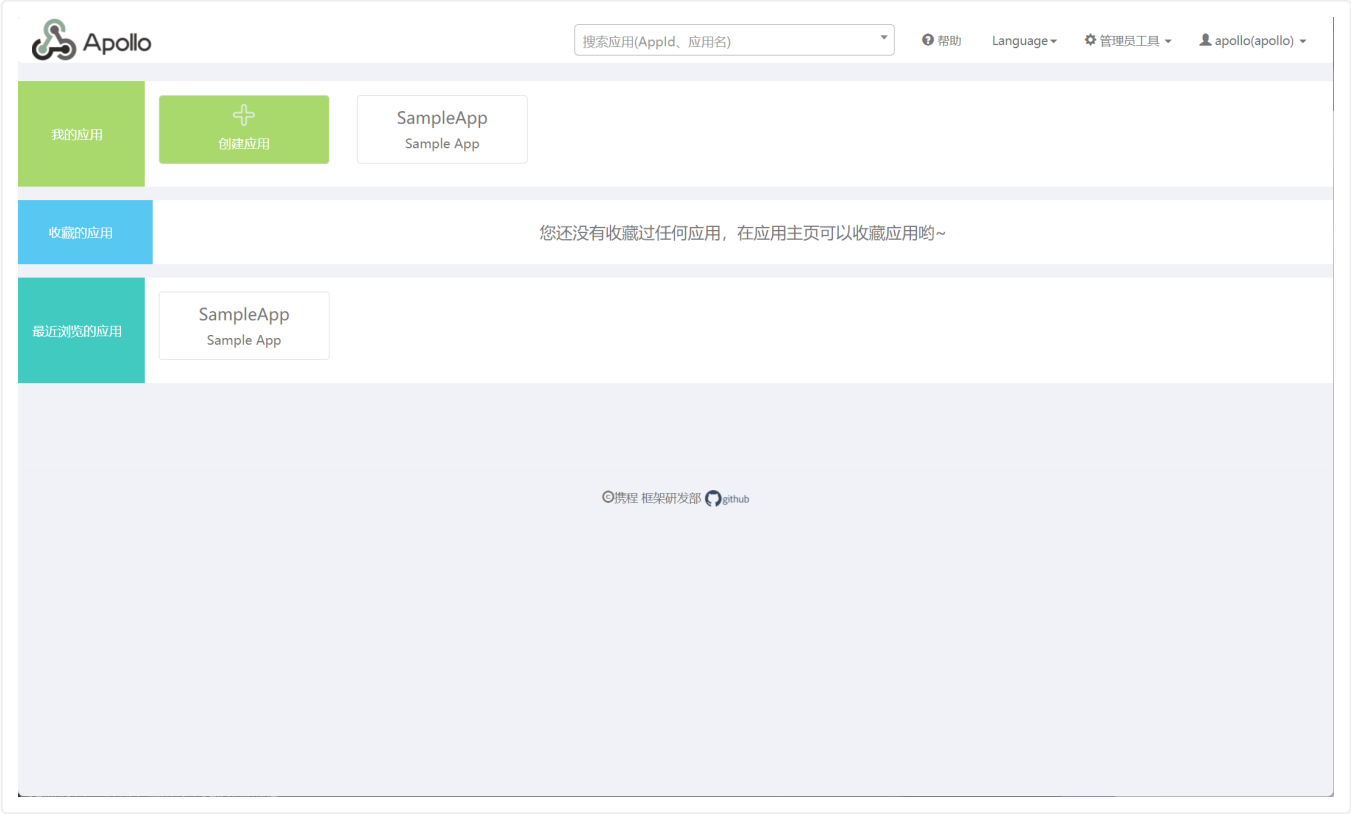
```
==== starting service ====
Service logging file is ./service/apollo-service.log
Started [10768]
Waiting for config service startup.......
Config service started. You may visit http://localhost:8080 for service status now!
Waiting for admin service startup....
Admin service started
==== starting portal ====
Portal logging file is ./portal/apollo-portal.log
Started [10846]
Waiting for portal startup......
Portal started. You can visit http://localhost:8070 now!
```

使用Apollo配置中心

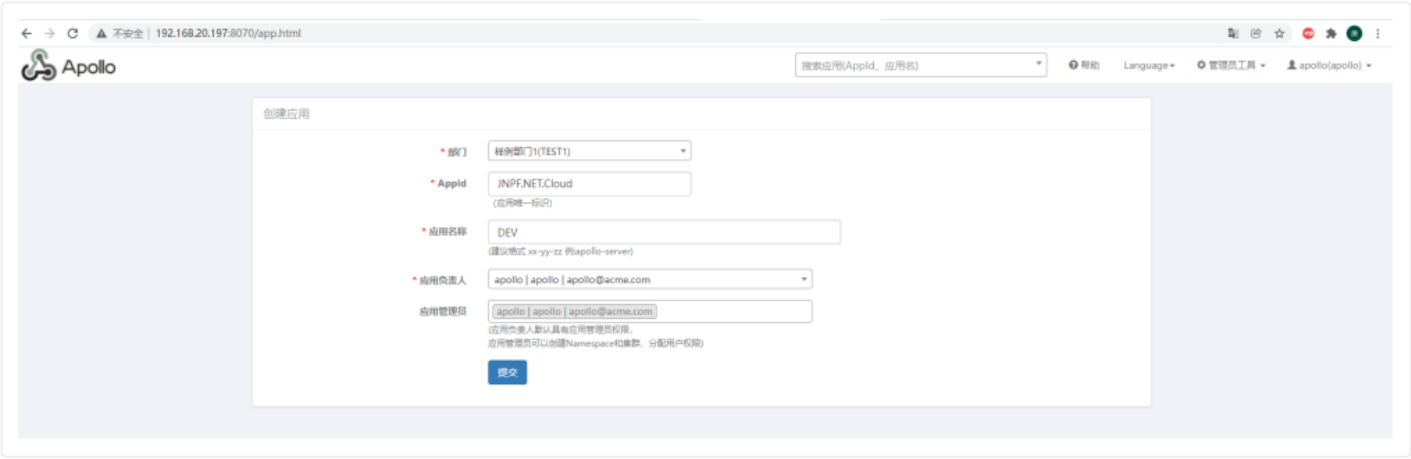
1. 打开Apollo配置中心
    - 本地：`http://localhost:8070`
    - 服务器：`http://ip:8070`



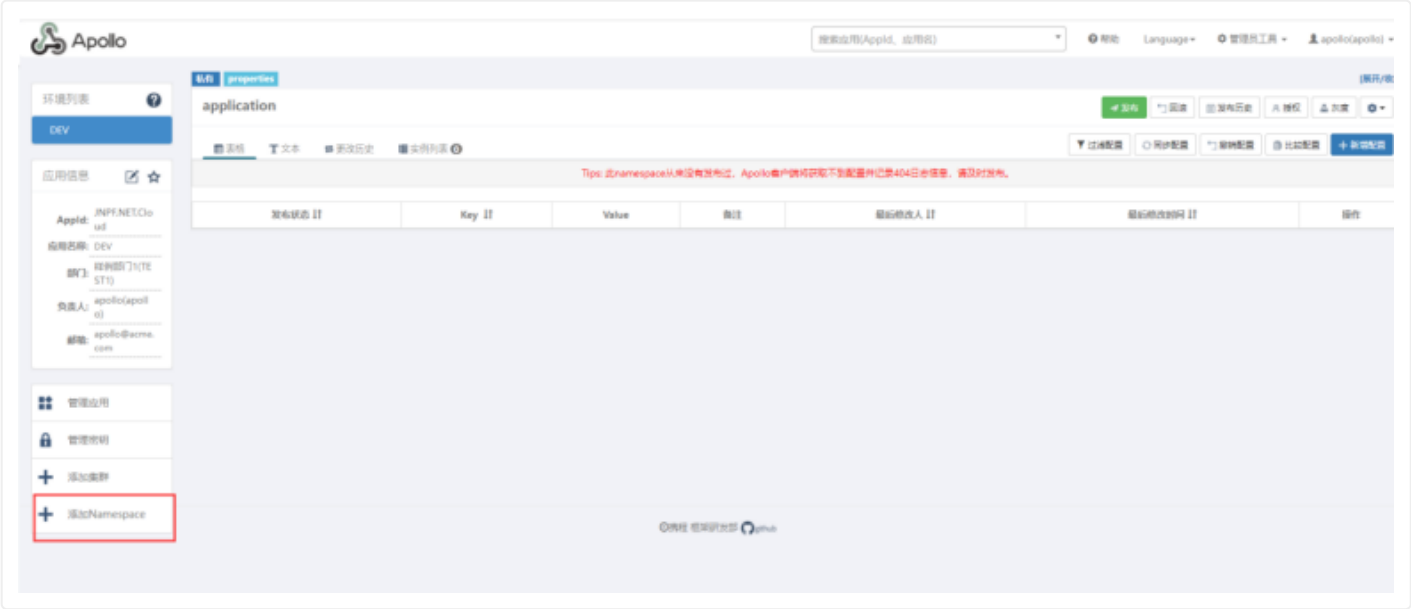
1. 输入用户名 `apollo` ，密码 `admin` 后登录



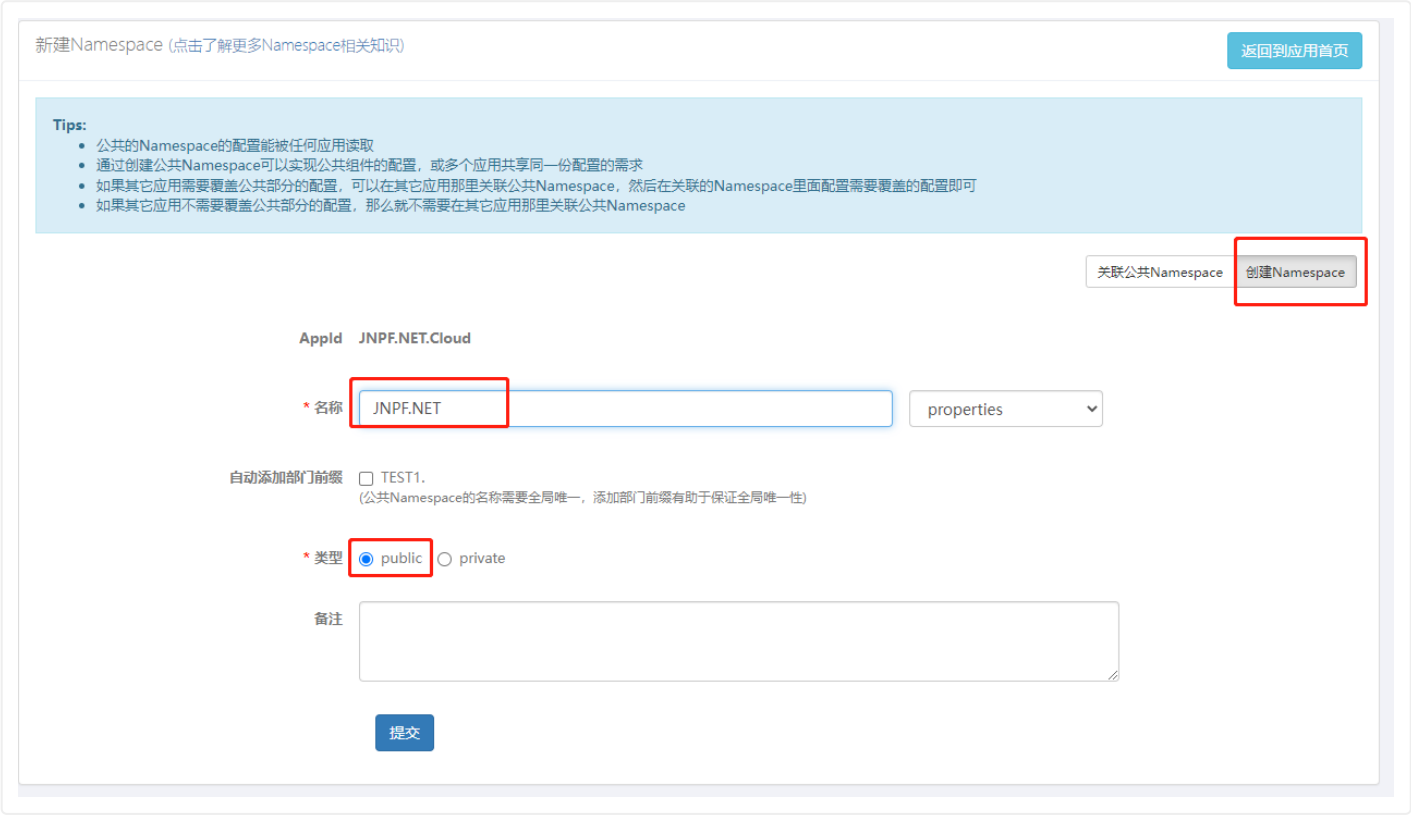
3.点创建应用，进去用，如下图填写



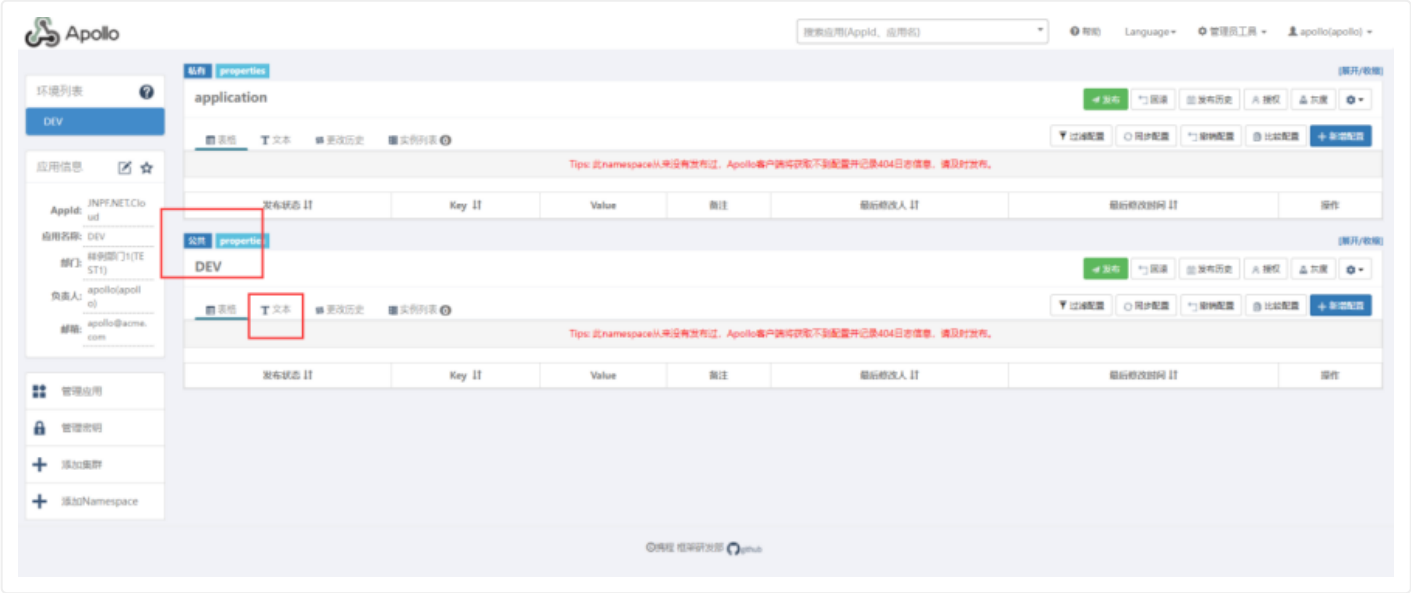
4.创建完应用，点 添加Namespace



5.Namespace配置，必须是公共的，如下图填写



提交后如下图



6.配置系统参数

在文本里面填写下面这份配置

```
ConnectionStrings:ConfigId = default
ConnectionStrings:DBName = jnpf_cloud
ConnectionStrings:DBType = MySql
ConnectionStrings:DefaultConnection = Server=192.168.10.25;Database={0};Uid=root;Pwd=Dfjl@58772387;
ConnectionStrings:Host = 192.168.10.25
ConnectionStrings:Port = 3306
ConnectionStrings:UserName = root
ConnectionStrings:Password = Dfjl@58772387
AppSettings:InjectMiniProfiler = true
AppSettings:InjectSpecificationDocument = true
JWTSettings:ValidateIssuerSigningKey = true
JWTSettings:IssuerSigningKey = 7k5yOxSMHVdYjs61gkgUY3W9DHbgk7tokaZlP3QIlfk34D1H7jYEOcLybClW1aKl
JWTSettings:ValidateIssuer = true
JWTSettings:ValidIssuer = yinmaisoft
JWTSettings:ValidateAudience = true
JWTSettings:ValidAudience = yinmaisoft
JWTSettings:ValidateLifetime = true
JWTSettings:ExpiredTime = 1440
JWTSettings:ClockSkew = 5
JNPF_App:CodeAreasName = ["SubDev","Test"]
JNPF_App:SystemPath = D:\jnpf-resources
JNPF_App:MPUploadFileType = [ "bmp", "png", "jpeg", "jpg", "gif", "mp3", "wma", "wav", "amr", "mp4" ]
JNPF_App:WeChatUploadFileType = [ "jpg", "png", "doc", "docx", "ppt", "pptx", "xls", "xlsx", "pdf", "txt", "rar", "zip", "csv", "amr", "mp4" ]
```

```
JNPF_App:AllowUploadImageType = ["jpg", "gif", "png", "bmp", "jpeg", "tiff", "psd", "swf", "svg", "pcx", "dxf", "wmf", "emf", "ic", "eps", "tga"]
JNPF_App:AllowUploadFileType = ["jpg", "mp3", "gif", "png", "bmp", "jpeg", "doc", "docx", "ppt", "pptx", "xls", "xlsx", "pdf", "txt", "rar", "zip", "csv"]
JNPF_App:Domain = http://127.0.0.1:5000
JNPF_App:YOZO:domain = http://dcsapi.com/
JNPF_App:YOZO:domainKey = 57462250284462899305150
JNPF_App:DefaultMailTo = yinmaisoft@163.com
JNPF_App:AppVersion = 3.4.1
JNPF_App:YOZO:UploadAPI = http://dmc.yozocloud.cn/api/file/http?fileUrl={0}&appId={1}&sign={2}
JNPF_App:YOZO:DownloadAPI = http://eic.yozocloud.cn/api/view/file?fileVersionId={0}&appId={1}&sign={2}
JNPF_App:YOZO:AppId = yozoAQh5dPSt6063
JNPF_App:YOZO:AppKey = 6365bfbd733fce644fd7ac0aaeca
JNPF_App:PreviewType = kkfile
CorsAccessorSettings:PolicyName = JNPFCorsAccessor
OSS:BucketName = jnpfsoftoss
OSS:Provider = Invalid
OSS:Endpoint = 192.168.0.60:9000
OSS:AccessKey = minioadmin
OSS:SecretKey = minioadmin
OSS:IsEnableHttps = false
OSS:IsEnableCache = true
Tenant:MultiTenancy = false
Tenant:MultiTenancyDBInterFace = http://127.0.0.1:5000/api/Saas/Tenant/DbContent/
ConsulClient:IP = localhost
ConsulClient:Port = 8500
ConsulClient:Datacenter = dc1
JNPF_APP:KKFileDomain = http://127.0.0.1:30090/FileServer
JNPF_APP:ErrorReport = false
JNPF_APP:ErrorReportTo = yinmaisoft@163.com
Cache:CacheType = RedisCache
Cache:RedisConnectionString = 127.0.0.1:6379,defaultDatabase=1,password=,connectTimeout=1000,connectRetry=1,syncTimeout=1000
JNPF_App:FileStoreType = local
JNPF_App:SystemPath = //www//wwwroot//resources
JNPF_App:SpecialString = [ "/", "<", ">", "|", "?", "\\", ":", "\"", "*" ]
```

7.点发布

常见问题

注：系统参数如果有修改，要停止Apollo服务，删除Apollo缓存，再启动Apollo服务
Apollo本地缓存文件位于：
Linux环境：/opt/data/{appId}/config-cache
Windows环境：C:\opt\data{appId}\config-cache

consul部署文档

> 打开 https://www.consul.io/downloads 下载相应版本

Windows 环境下运行

```
./consul agent -dev -ui -node=consul-dev
```

Linux环境下运行

```
nohup ./consul agent -dev -ui -node=consul-dev -client=0.0.0.0 > Log.log 2>&1 &
```

常见问题

> 服务注册进 consul 时，请注意服务运行命令 http 和 https 的区别，运行https需要安装ssl证书才能注册健康服务

> http 对应 ws ， https 对应 wss

用户手册

平台主题

导航模块

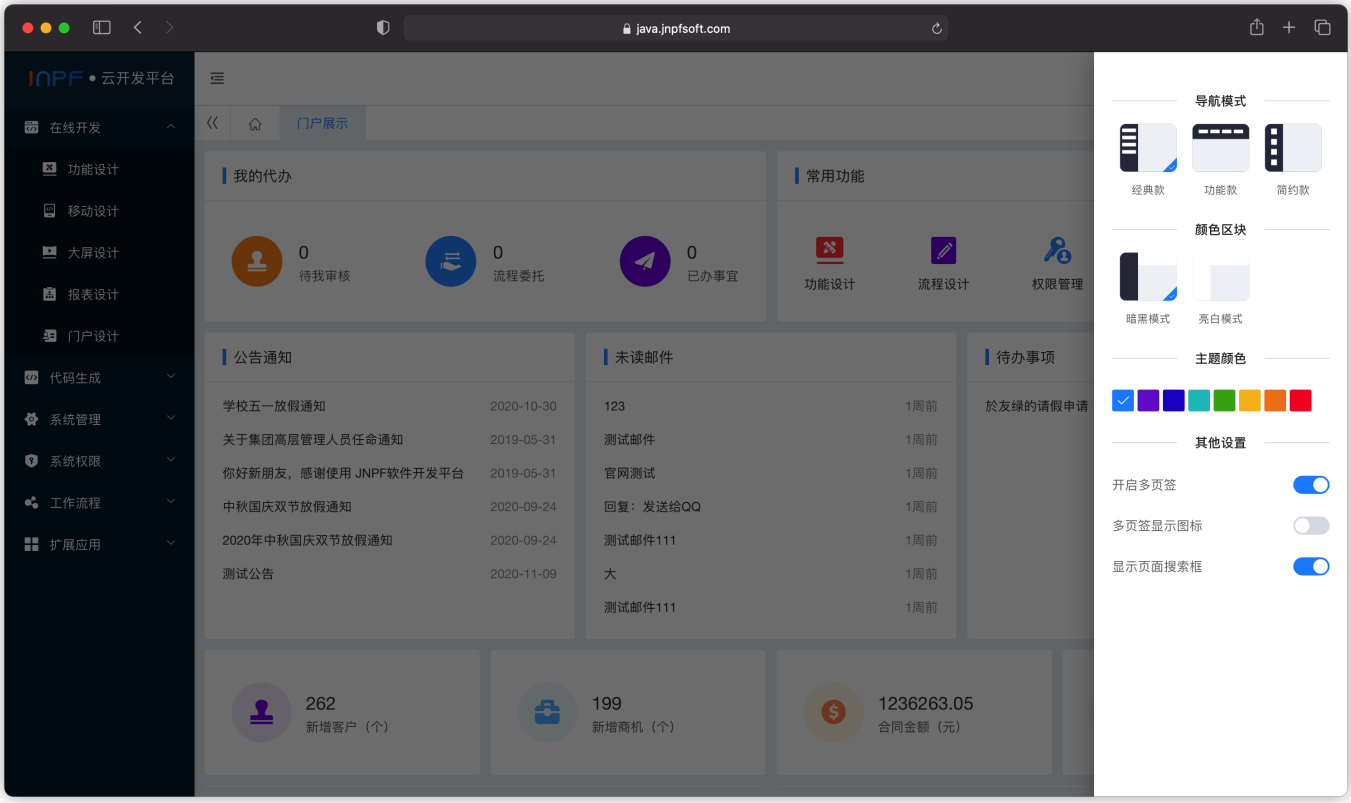
经典款、功能款、简约款可选择；

颜色区块

暗黑模式、亮白模式可选择；

主题颜色

有经典蓝、玫紫、湛蓝、海洋、生机、丰收、阳橙热情可选择；



在线开发

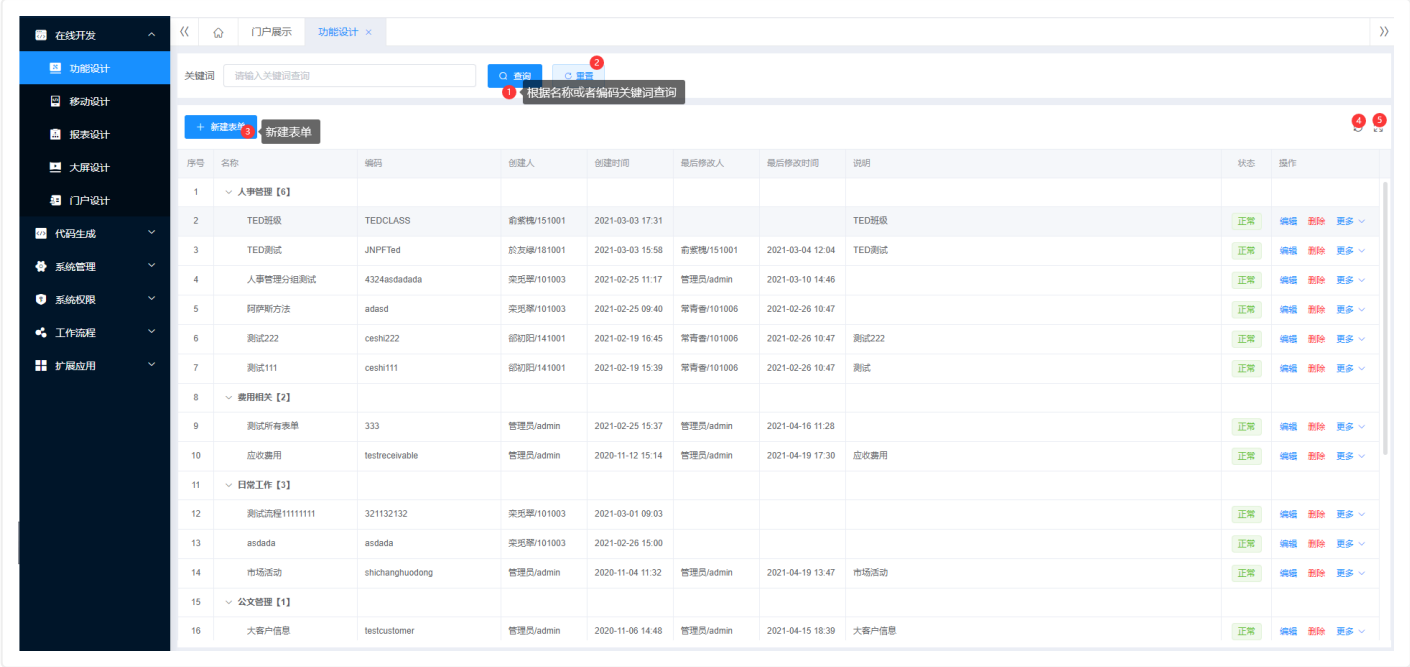
功能设计

功能描述

零代码生成PC端功能，通过拖拽式操作界面，有基础控件、高级控件、系统控件等、可选择配置表单属性、组件属性等，列表设计有查询字段、列表字段、列表属性等选择设置，自动生成可视化应用。

操作步骤

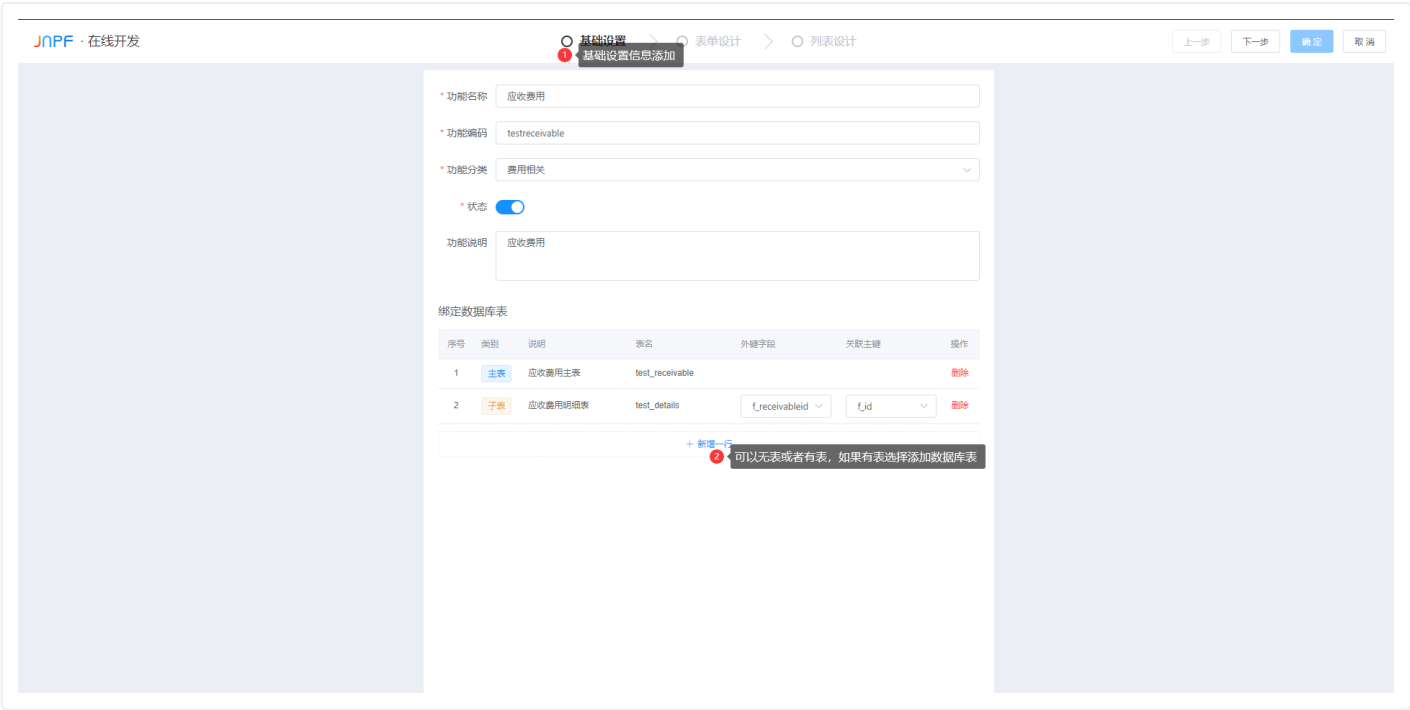
在在线开发目录下操作在功能设计，进入【功能设计】页面有查询、重置、刷新、新建表单、全屏等功能；如下图所示：



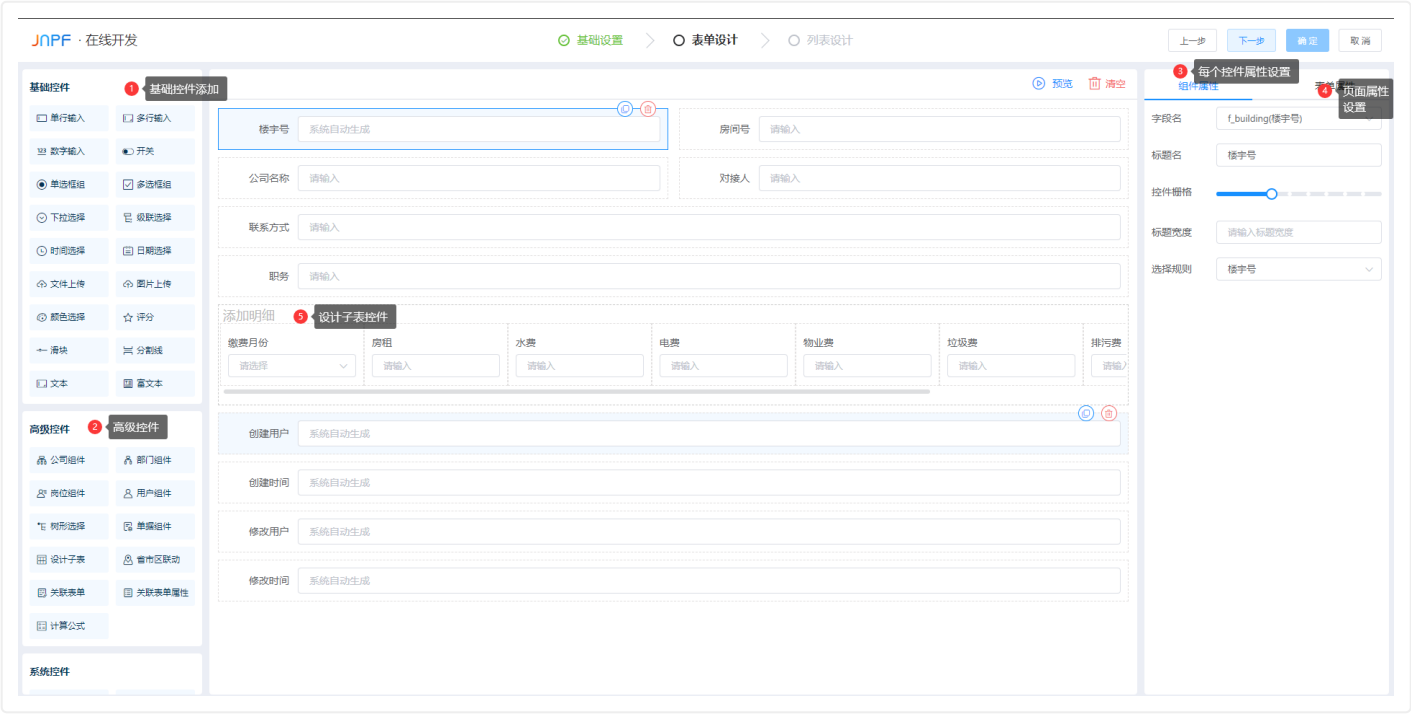
ⅰ刷新当前页面；

ⅱ在当前页面输入名称查询；

ⅲ鼠标点击该页面左上角新建表单功能，进入新建表单页面，进行字段填写、红色星号表示必填项；基础设置填写一些功能基础信息和是否绑定数据库表后，确认点击“下一步”进行表单设计或者点击取消按钮离开该页面、当前数据未保存。如下图所示：

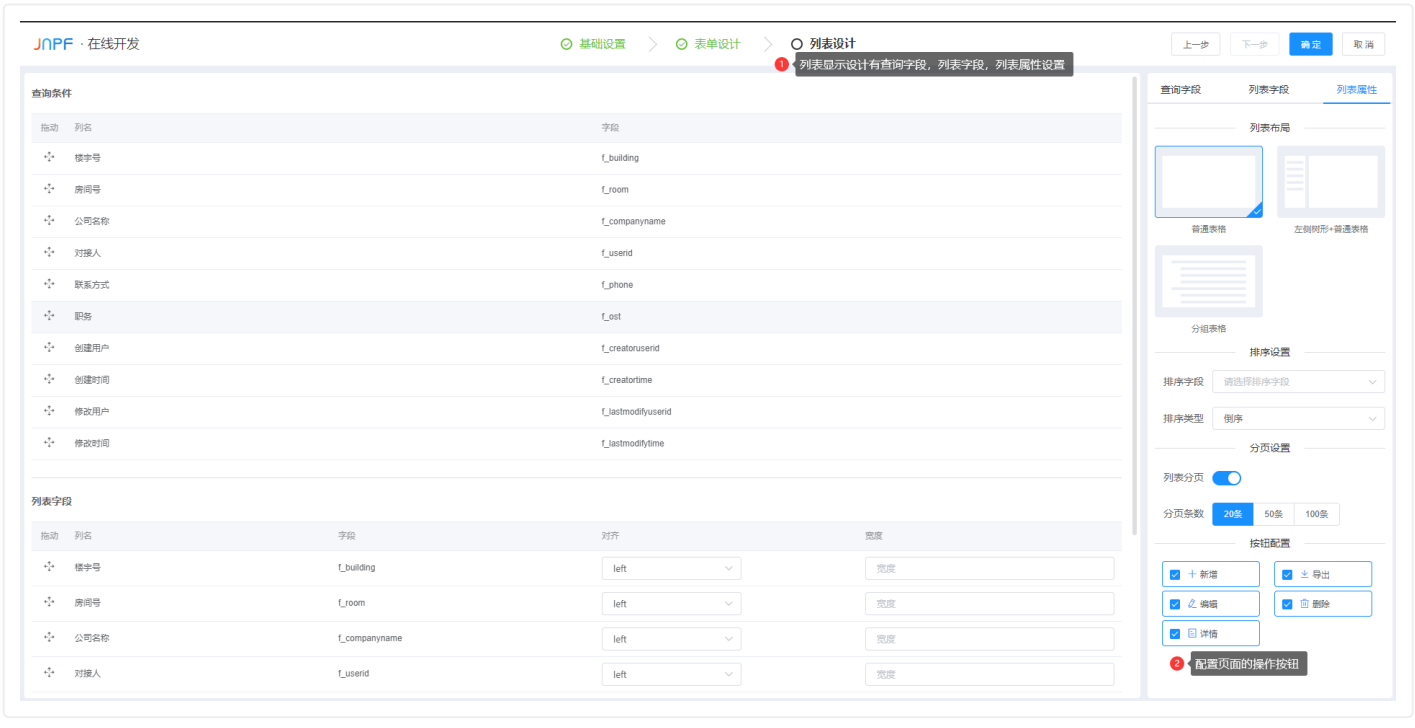


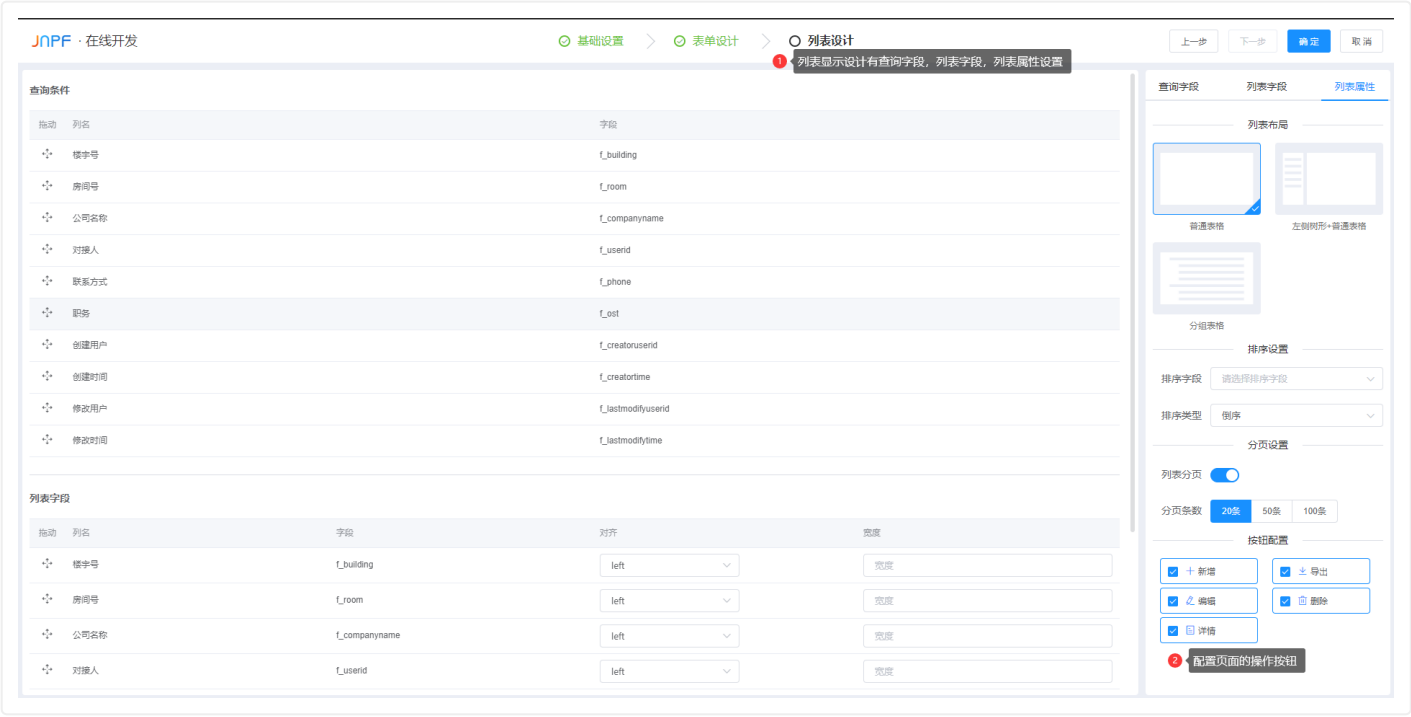
表单设计是根据用户需求设计界面功能，确认点击“下一步”进行列表设计或者点击取消按钮离开该页面、当前数据未保存。如下图所示：


JNPF · 在线开发
基础设置
表单设计
列表设计
上一步
下一步
确定
取消
基础控件
1 基础控件添加
单行输入
多行输入
数字输入
开关
单选框组
多选框组
下拉选择
级联选择
时间选择
日期选择
文件上传
图片上传
颜色选择
评分
滑块
分割线
文本
富文本
高级控件
2 高级控件
公司组件
部门组件
岗位组件
用户组件
树形选择
单据组件
设计子表
省市区联动
关联表单
关联表单属性
计算公式
系统控件
预览
清空
3 每个控件属性设置
组件属性
表单属性
4 页面属性设置
楼宇号
系统自动生成
房间号
请输入
公司名称
对接人
联系方式
职务
添加明细
5 设计子表控件
缴费月份
房租
水费
电费
物业费
垃圾费
排污费
创建用户
创建时间
修改用户
修改时间
字段名
f_building(楼宇号)
标题名
控件栅格
标题宽度
选择规则


组件属性
表单属性
表单尺寸
中等
较小
迷你
标签对齐
左对齐
右对齐
顶部对齐
标题宽度
100
栅格间隔
15
弹窗类型
普通弹窗
表单样式
默认风格
表单宽度
600px
取消按钮文本
取 消
确定按钮文本
确 定

列表设计是根据用户设计的功能显示列表属性、列表字段、查询字段等设计，点击“确定”保存这个功能或者点击取消按钮离开该页面、当前数据未保存。如下图所示：

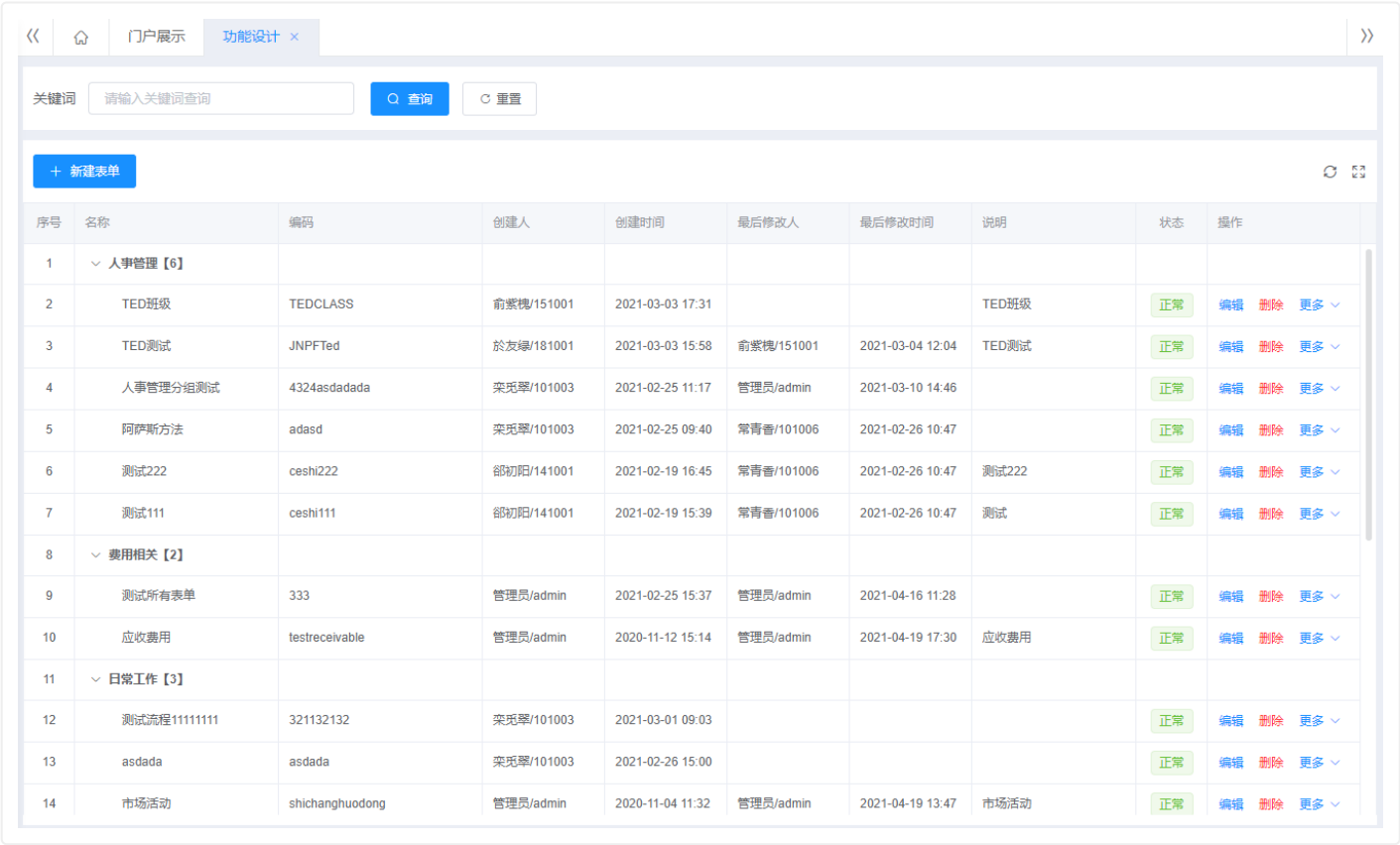




在功能设计页面可以对数据编辑、删除、更多（复制、预览）功能操作，如下图所示：

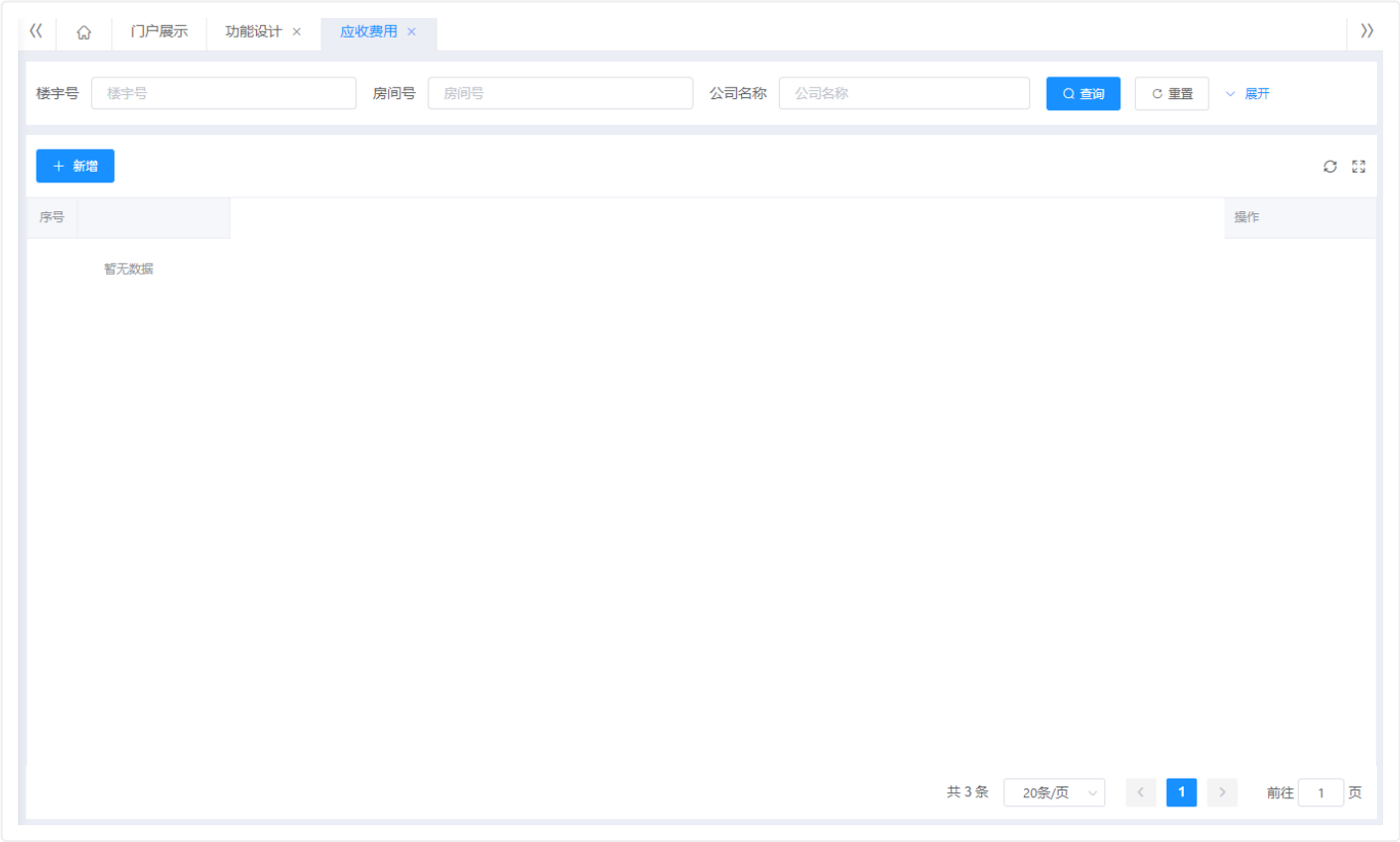
ⅰ点击“删除”按钮可以直接删除这个功能设计成功；

ⅱ操作更多下复制或预览，复制这个功能设计或者预览功能设计页面；



在菜单管理里面添加这个功能表单功能菜单，然后就可以直接看到自己设计的表单页面，如下图所示：


编辑菜单
* 分类
Web
* 上级
功能示例
* 名称
应收费用
* 编码
yingshou
* 图标
icon-ym icon-ym-darg
选择
* 类型
功能
* 关联
应收费用
排序
0
状态
说明
取 消
确 定



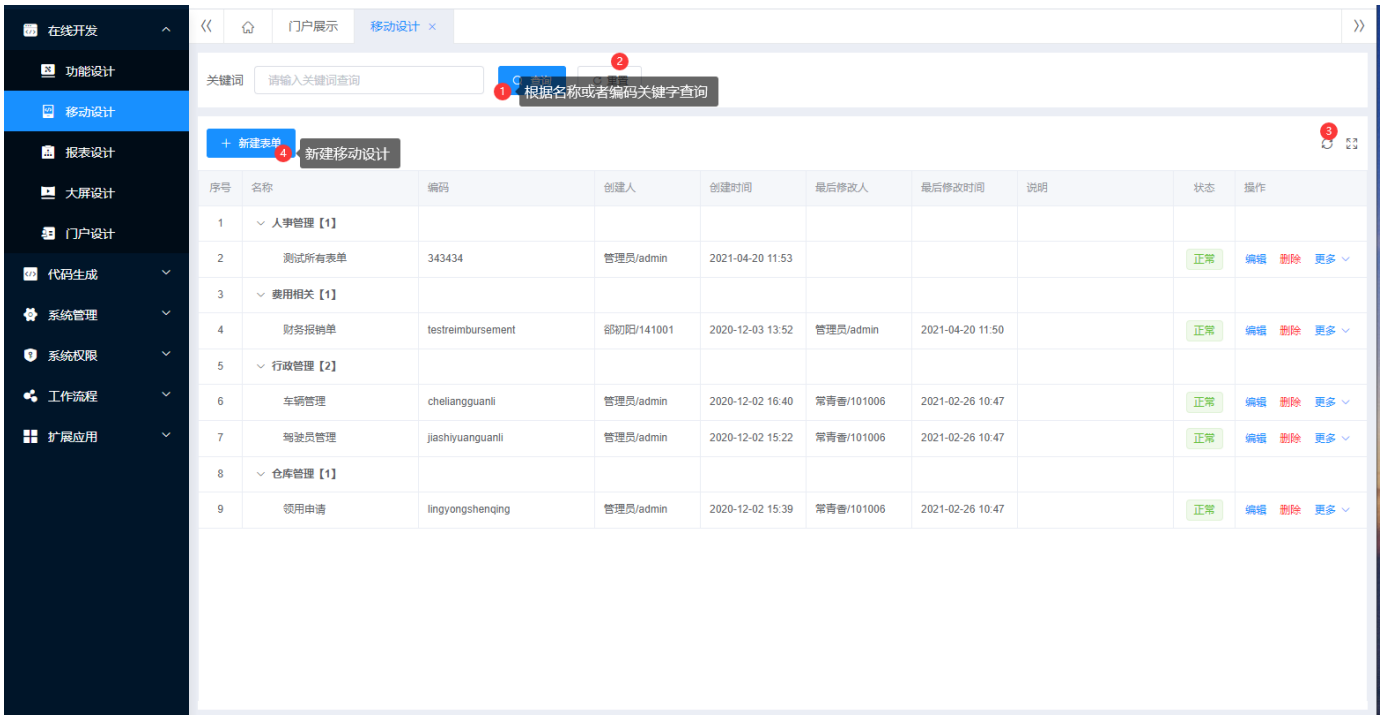
移动设计

功能描述

零代码生成移动端功能，通过拖拽式操作界面，有基础控件、高级控件、系统控件等、可选择配置表单属性、组件属性等，列表设计有排序字段、查询字段、列表字段、列表属性等选择设置，自动生成可视化应用。

操作步骤

在在线开发目录下操作在移动设计，进入【移动设计】页面有查询、重置、刷新、新建表单、全屏等功能；如下图所示：

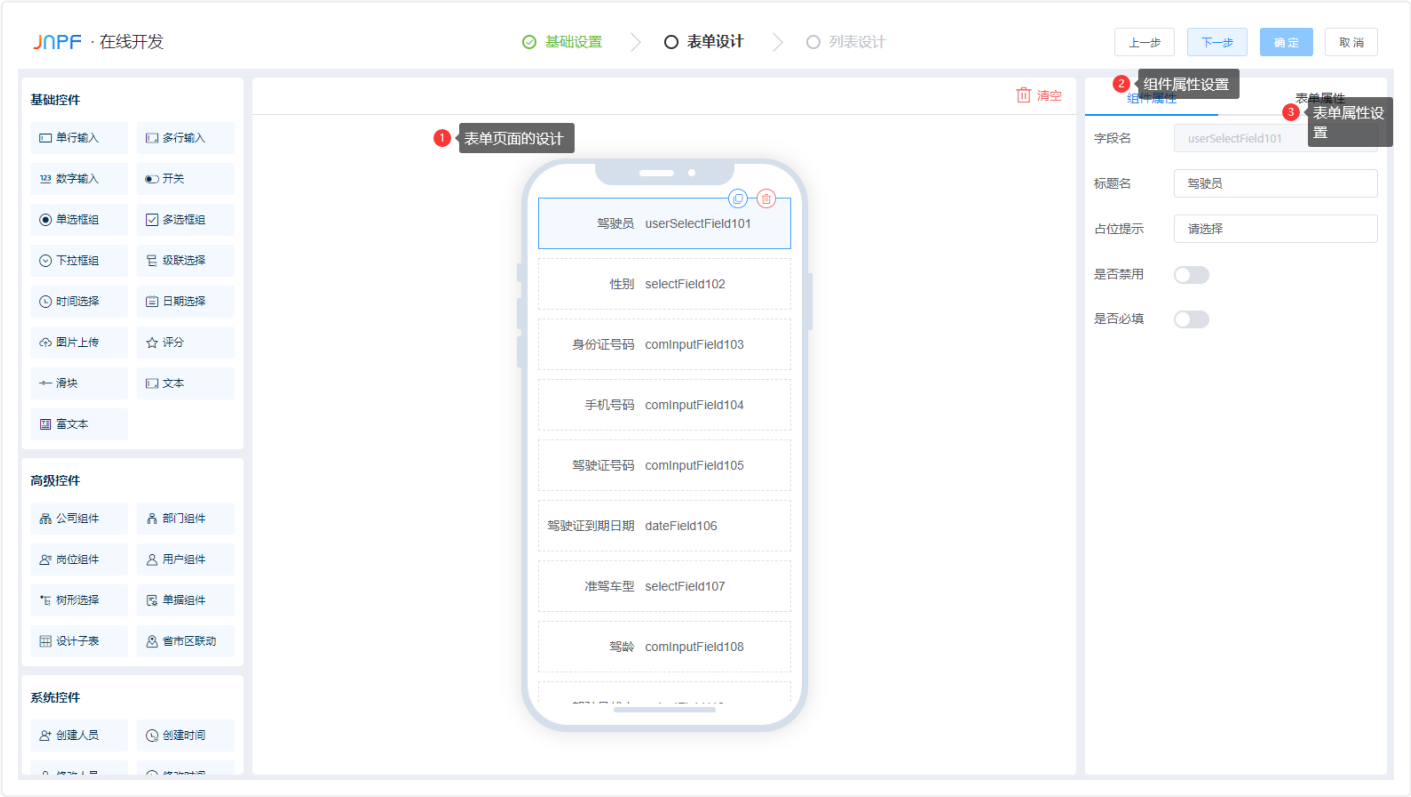


ⅰ刷新当前页面；

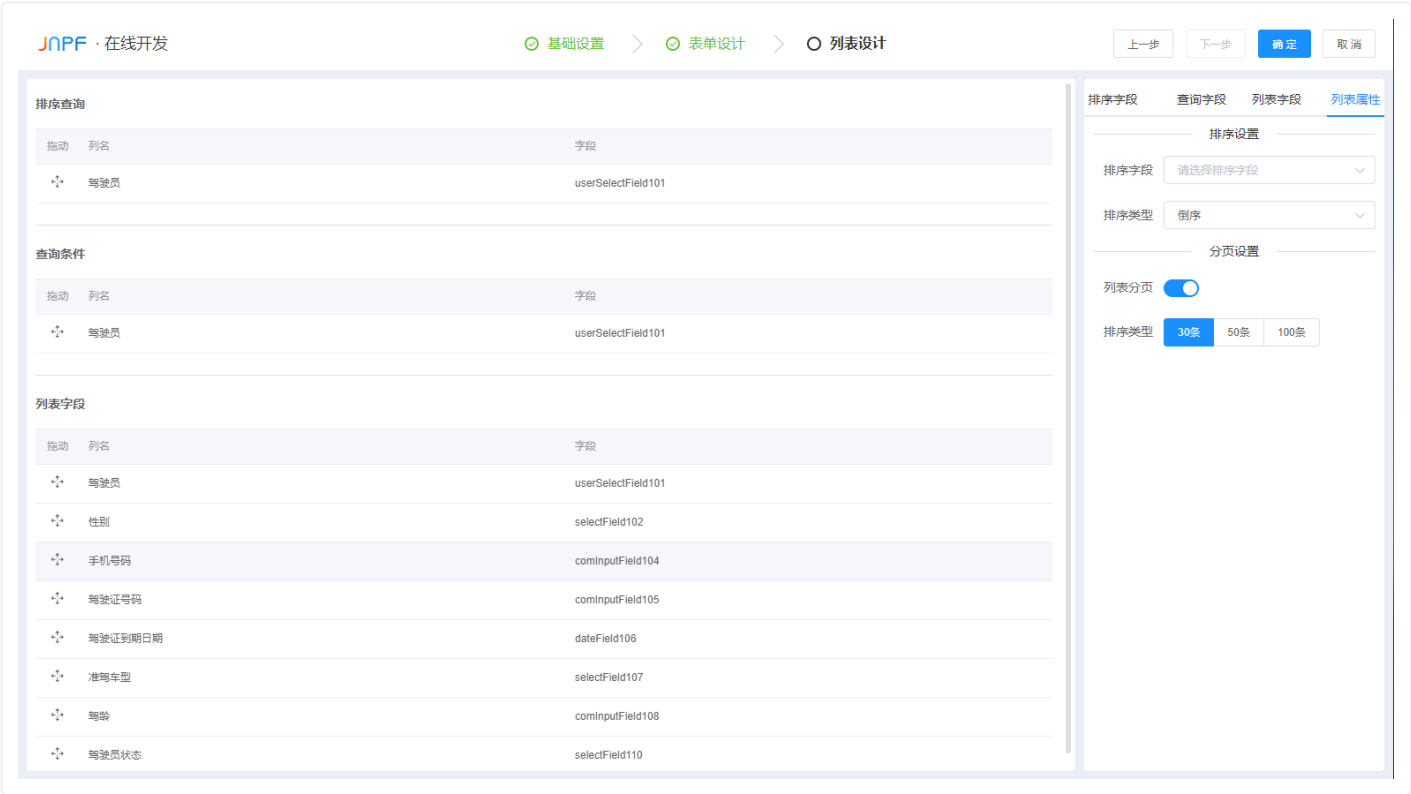
ⅱ在当前页面输入名称查询；

ⅲ鼠标点击该页面左上角新建表单功能，进入新建表单页面，进行基础设置填写一些功能基础信息和是否绑定数据库表后，确认点击“下一步”进行表单设计或者点击取消按钮离开该页面、当前数据未保存。红色星号表示必填项；如下图所示：

表单设计是根据用户需求设计界面功能，确认点击“下一步”进行列表设计或者点击取消按钮离开该页面、当前数据未保存。如下图所示：



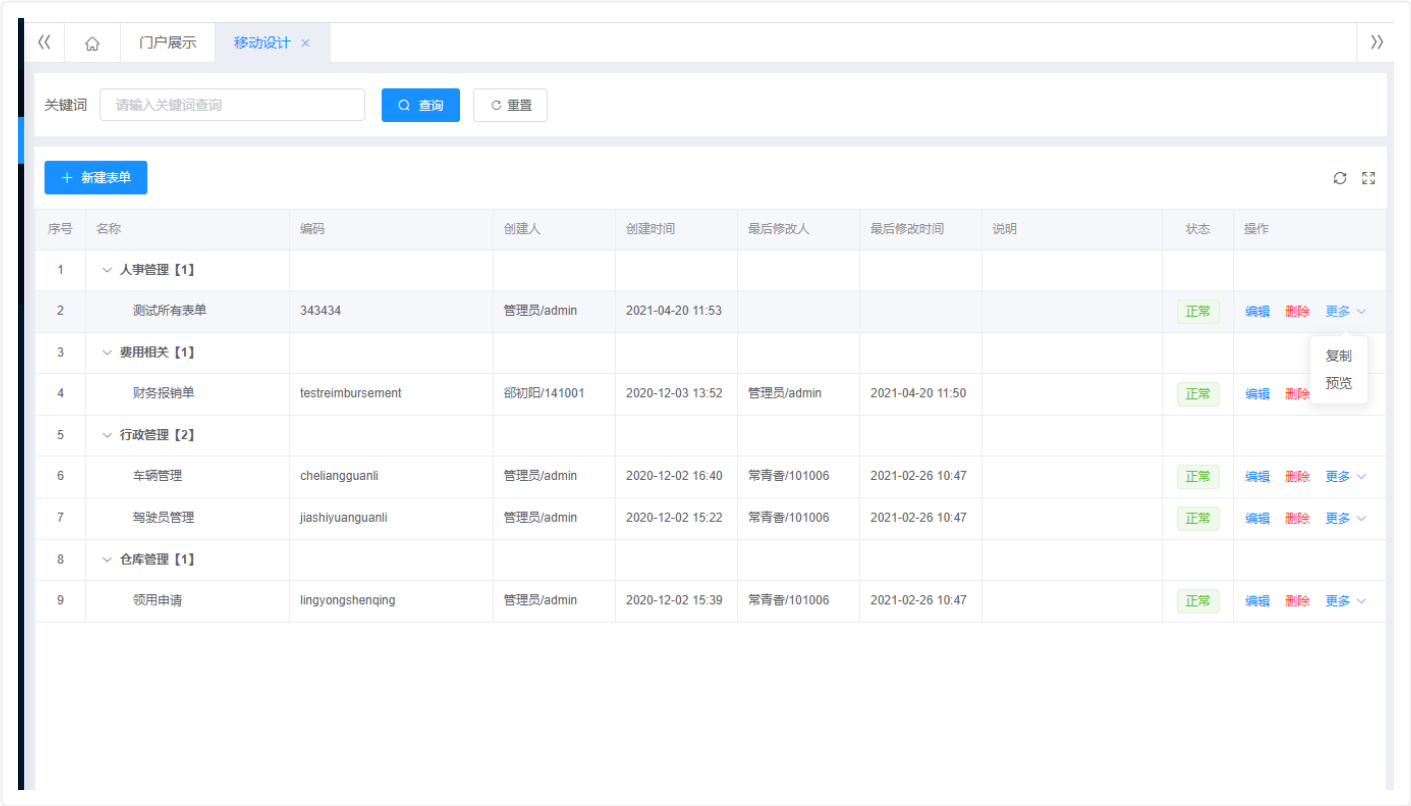
列表设计是根据用户设计的功能显示列表属性、列表字段、查询字段等设计，点击“确定”保存这个功能或者点击取消按钮离开该页面、当前数据未保存。如下图所示：



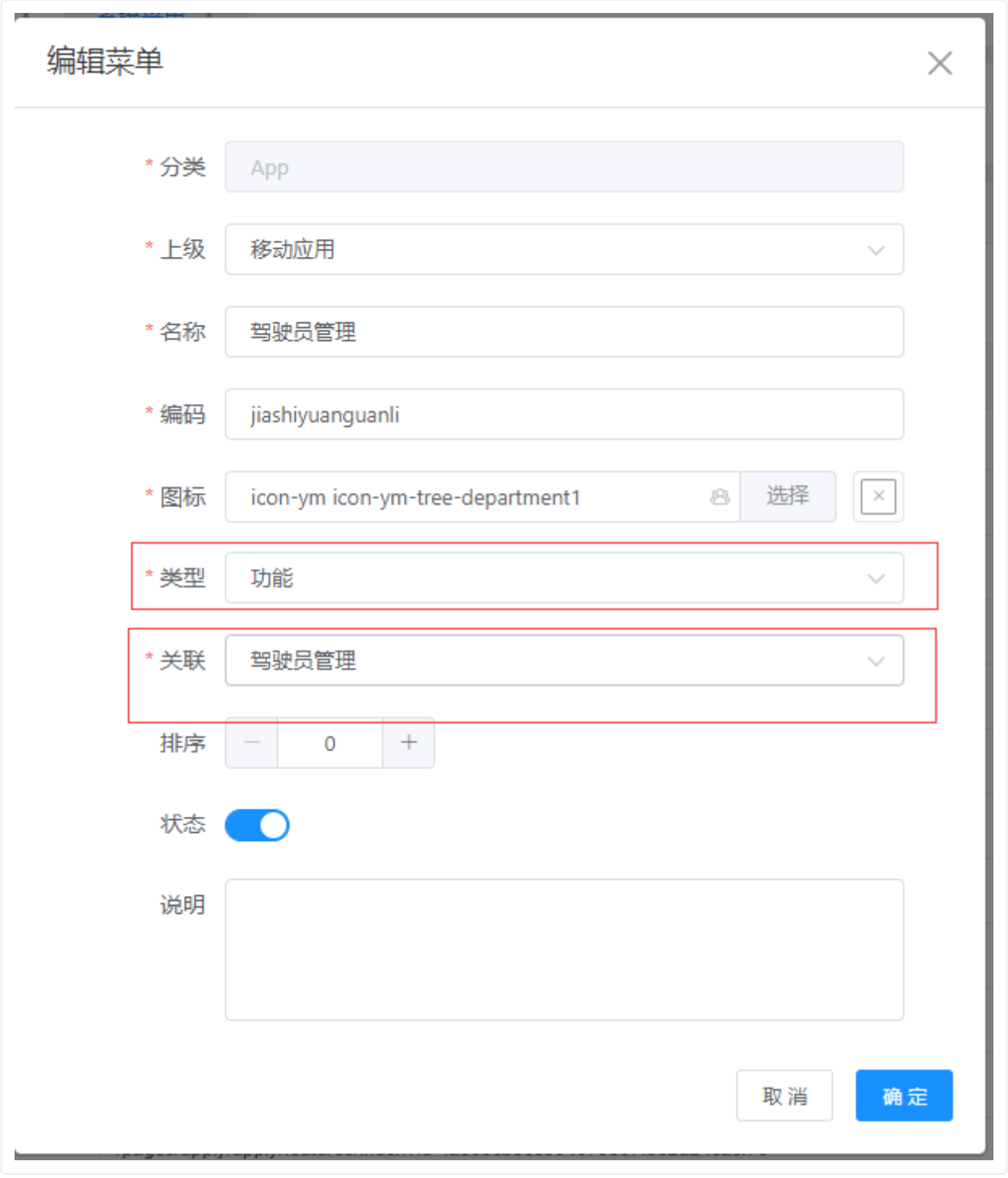
在移动设计页面可以对数据编辑、删除、更多（复制、预览）功能操作，如下图所示：

ⅰ点击“删除”按钮可以直接删除这个移动设计成功；

ⅱ操作更多下复制或预览，复制这个功能设计或者预览移动设计页面；



在菜单管理里面添加app移动设计功能菜单，然后就可以在App端显示自己设计的移动，如下图所示：



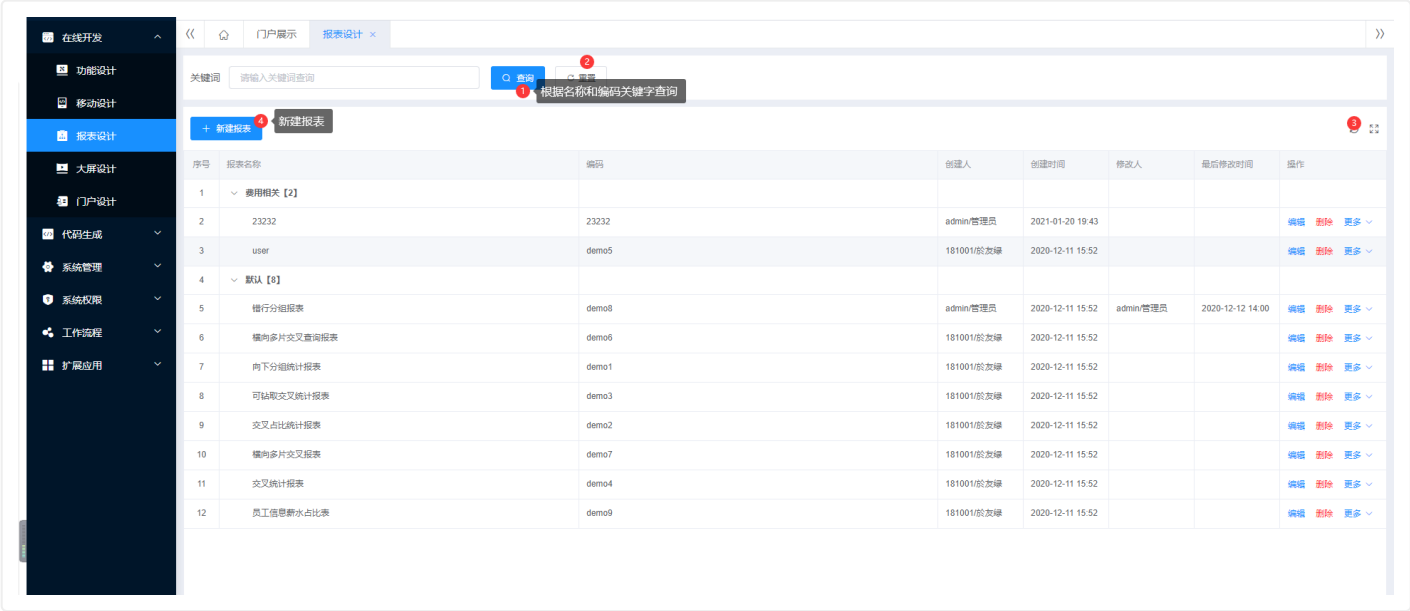
报表设计

功能描述

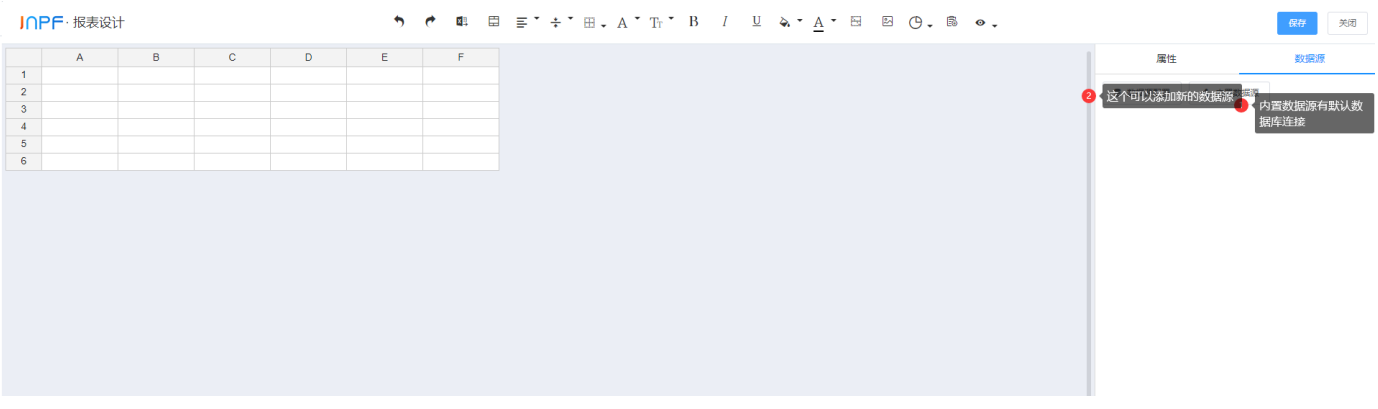
支持多源、横纵向分组交叉等常见功能的轻量级报表设计器，支持向导式创建报表项目，支持单元格的字体、字号、颜色、底色、对齐、框线等可视化设置，可完成各种复杂报表的设计制作。

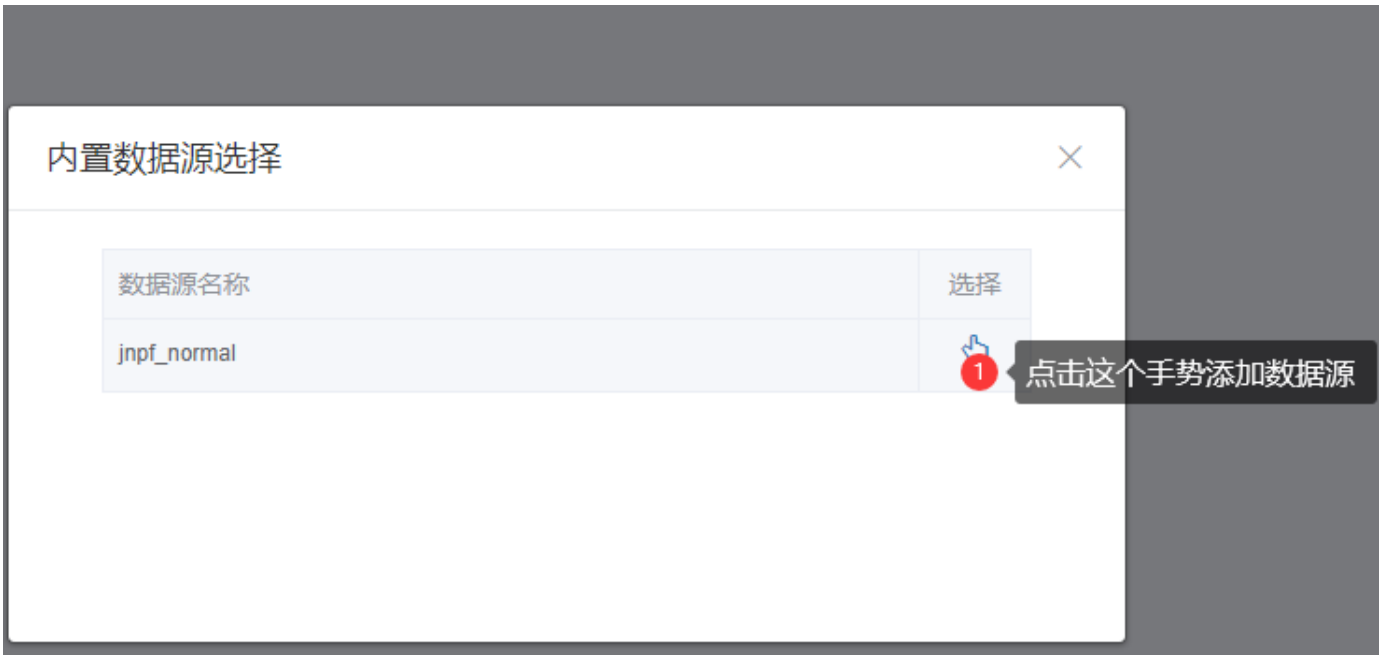
操作步骤

在在线开发目录下操作在报表设计，进入【报表设计】查询、重置、刷新、新建报表、全屏等功能；如下图所示：



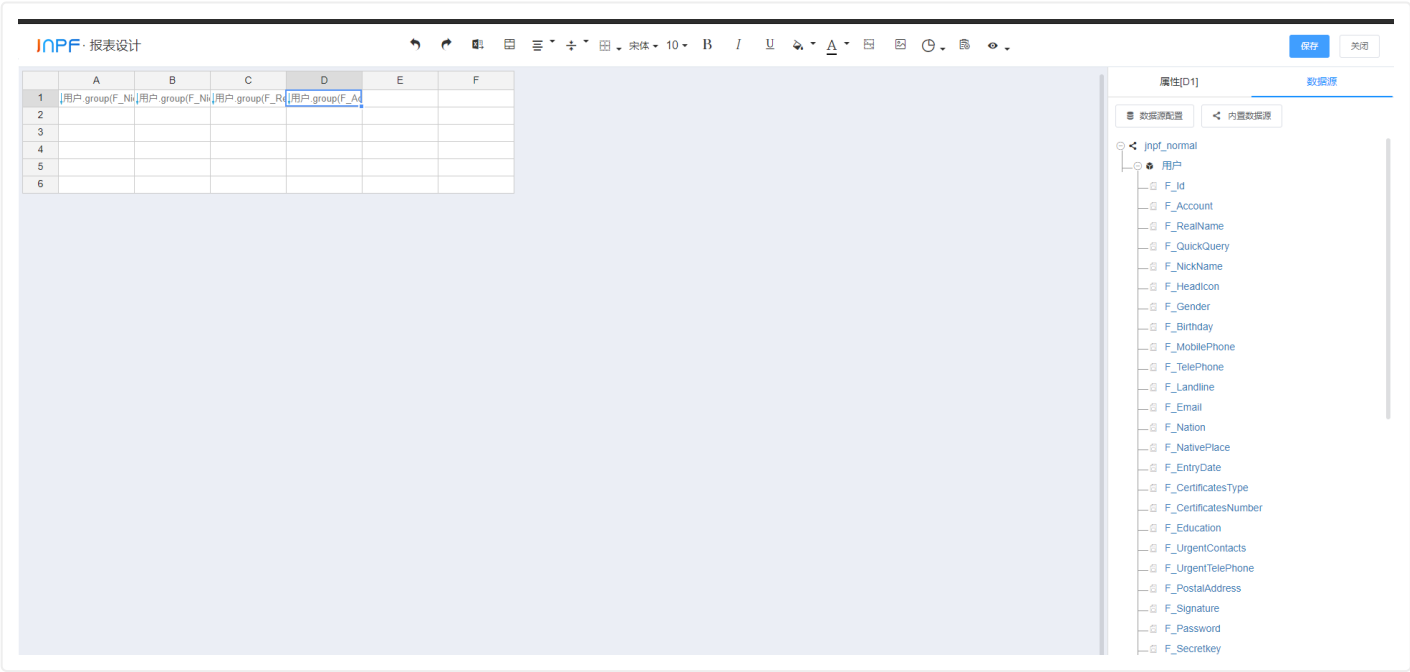
鼠标点击该页面左上角新建报表功能，进入新建报表页面，进行内置数据源选择、添加数据集设置；如下图所示：






属性
数据源
数据源配置
内置数据源
jnpf_normal
添加数据集
删除

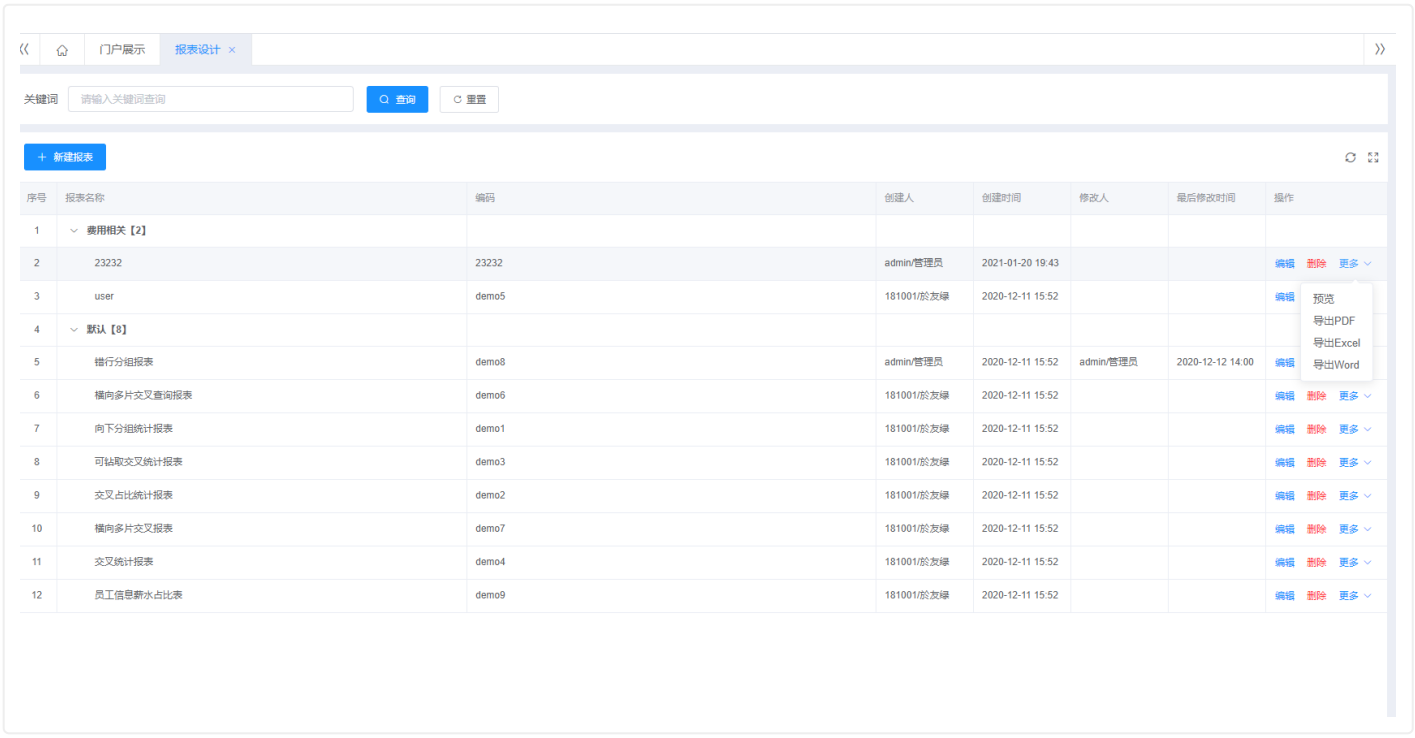
报表设计是根据用户需求设计报表功能，确认点击“保存”进行报表设计或者点击关闭按钮离开该页面、当前数据未保存。如下图所示：



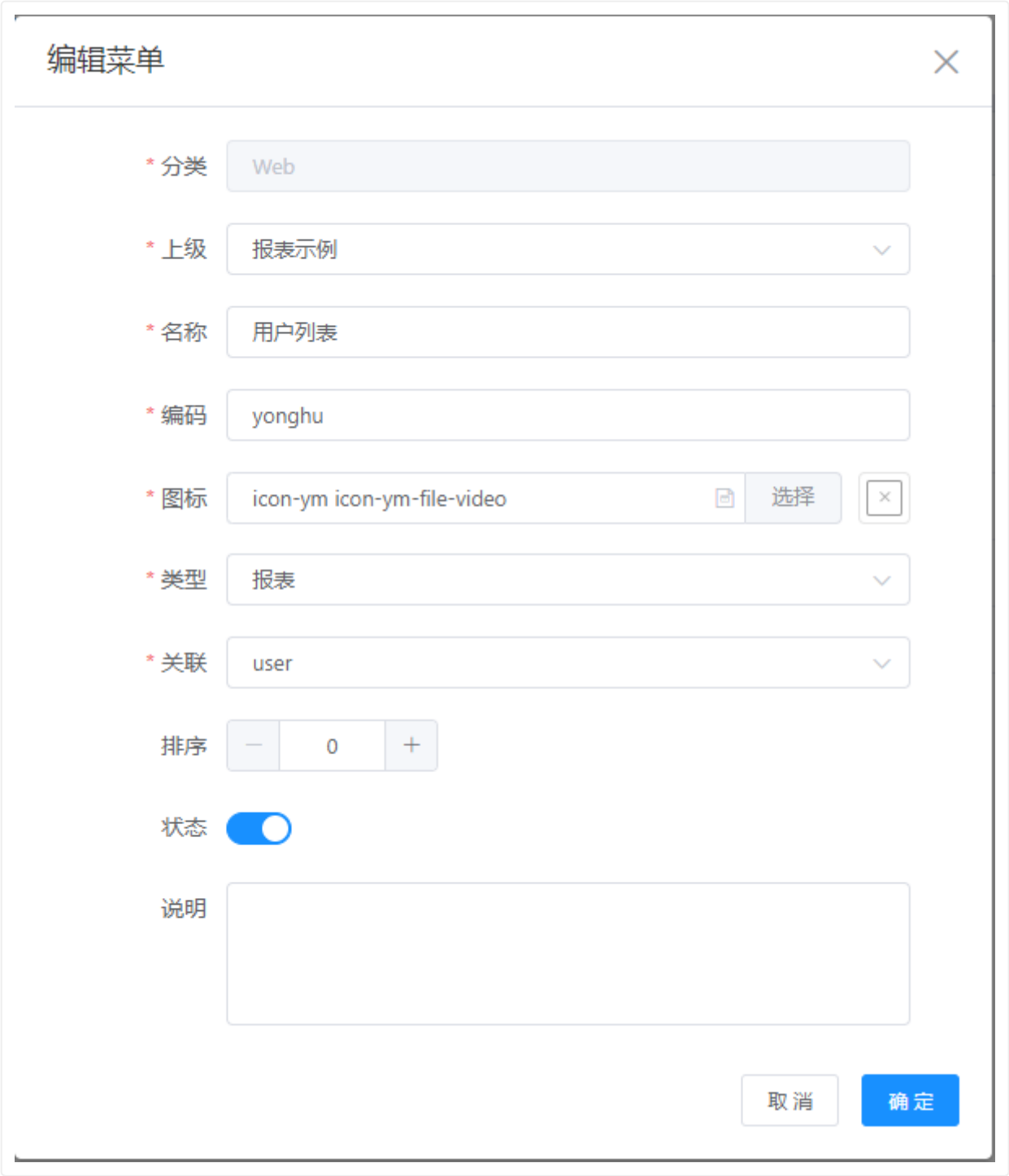
在报表设计页面可以对数据编辑、删除、更多等功能操作，如下图所示：

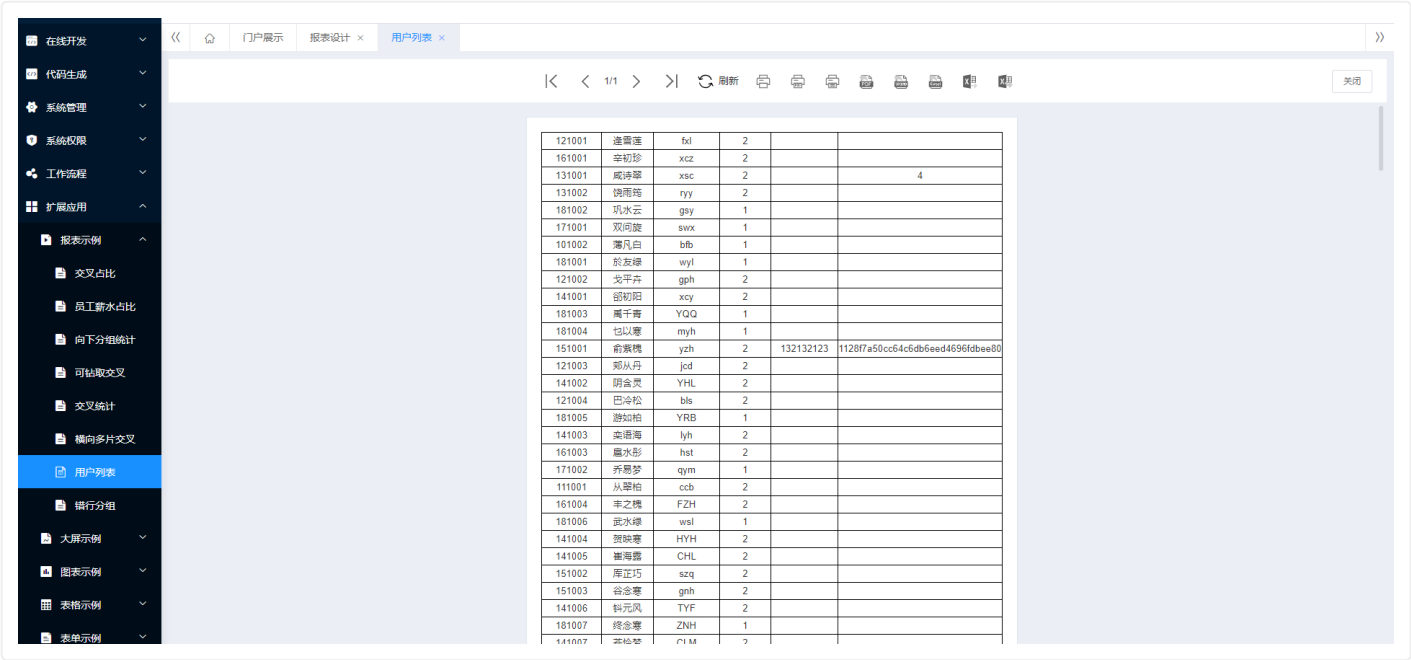
ⅰ 点击“删除”按钮可以直接删除这个报表设计成功；

ⅱ 操作更多下预览或者导出Word，预览报表设计页面或者导出Word文档；



在菜单管理里面添加这个报表菜单，然后就可以直接看到自己设计的报表页面，如下图所示：


编辑菜单
* 分类
Web
* 上级
报表示例
* 名称
用户列表
* 编码
yonghu
* 图标
icon-ym icon-ym-file-video
选择
* 类型
报表
* 关联
user
排序
0
状态
说明
取 消
确 定



121001	连雪莲	fxl	2		
161001	辛初珍	xcz	2		
131001	咸诗翠	xsc	2		4
131002	饶雨筠	ryy	2		
181002	巩水云	gsy	1		
171001	双问旋	swx	1		
101002	薄凡白	bfb	1		
181001	於友绿	wyl	1		
121002	戈平卉	gph	2		
141001	邵初阳	xcy	2		
181003	属千青	YQQ	1		
181004	乜以寒	myh	1		
151001	俞紫槐	yzh	2	132132123	1128f7a50cc64c6db6eed4696fdbee80
121003	郑从丹	jcd	2		
141002	阴含灵	YHL	2		
121004	巴冷松	bls	2		
181005	游如柏	YRB	1		
141003	栾语海	lyh	2		
161003	扈水彤	hst	2		
171002	乔易梦	qym	1		
111001	从翠柏	ccb	2		
161004	丰之槐	FZH	2		
181006	武水绿	wsl	1		
141004	贺映寒	HYH	2		
141005	崔海露	CHL	2		
151002	库正巧	szq	2		
151003	谷念寒	gnh	2		
141006	斜元风	TYF	2		
181007	终念寒	ZNH	1		
141007	乔松芳	CLM	2		

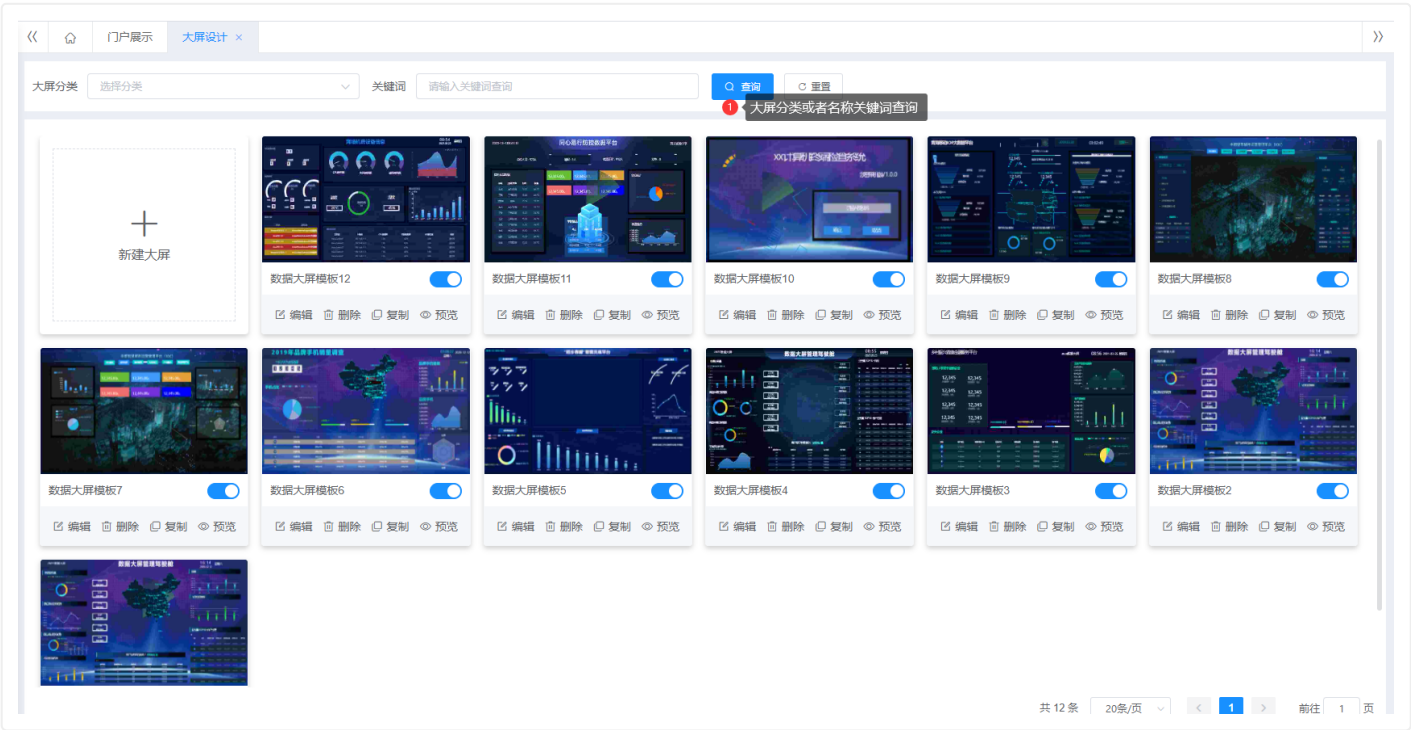
大屏设计

功能描述

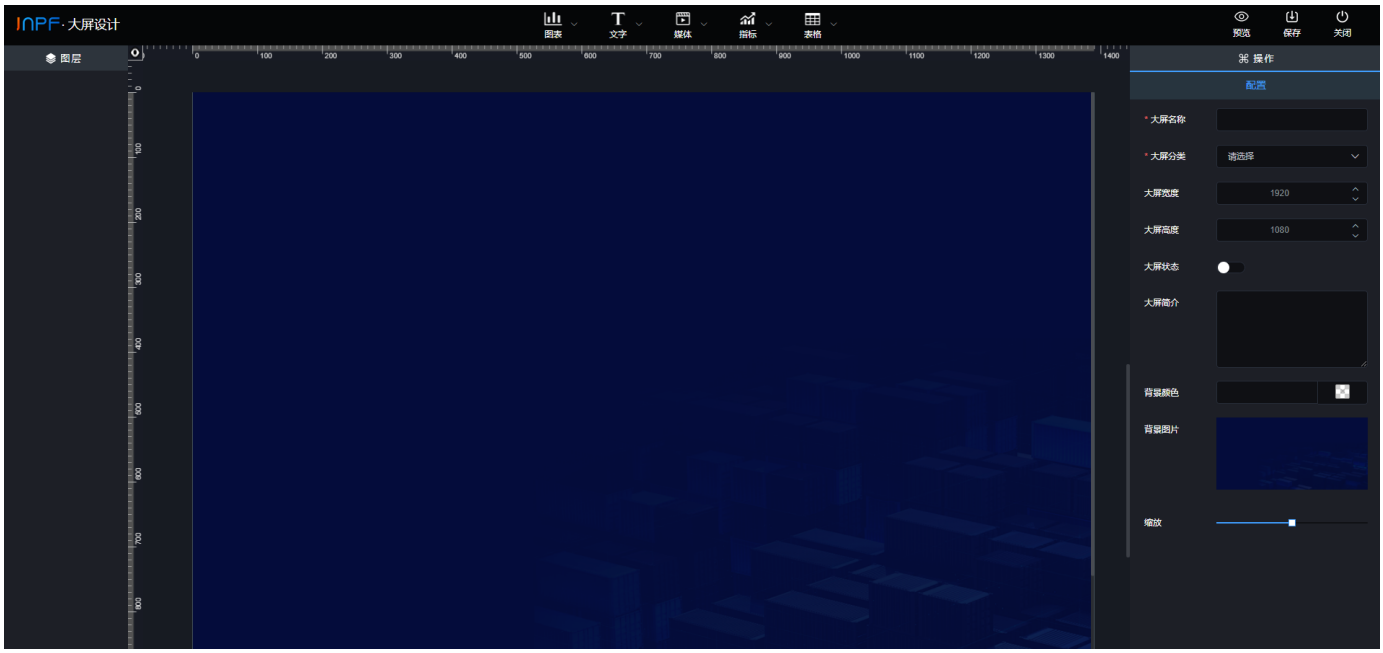
自由布局页面，拖拽控件至页面中，针对不同的控件进行内容设值和数据绑定，所见即所得的实现大屏可视化页面开发。

操作步骤

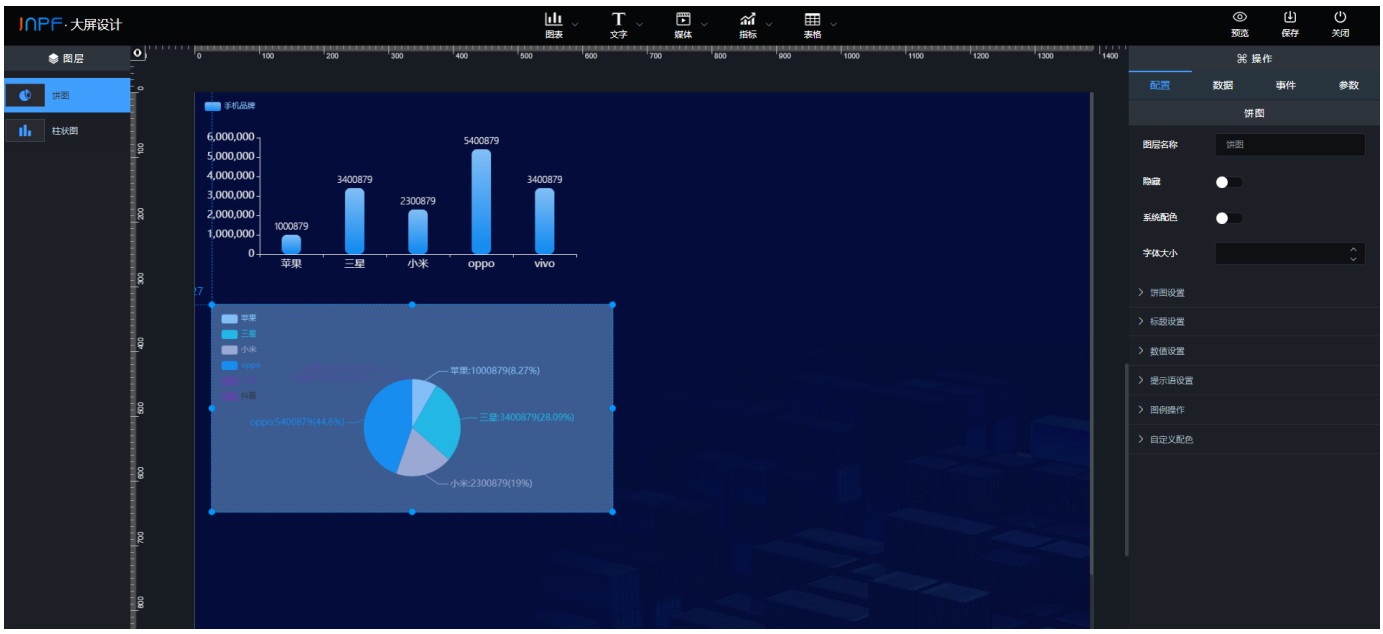
在在线开发目录下操作在大屏设计，进入【大屏设计】页面有查询、重置、刷新、新建大屏、全屏、编辑、复制等功能；如下图所示：



点击该页面中新建大屏功能，进入新建大屏页面，进行字段填写、图表、文字、媒体、指标、表格、红色星号表示必填项；如下图所示：



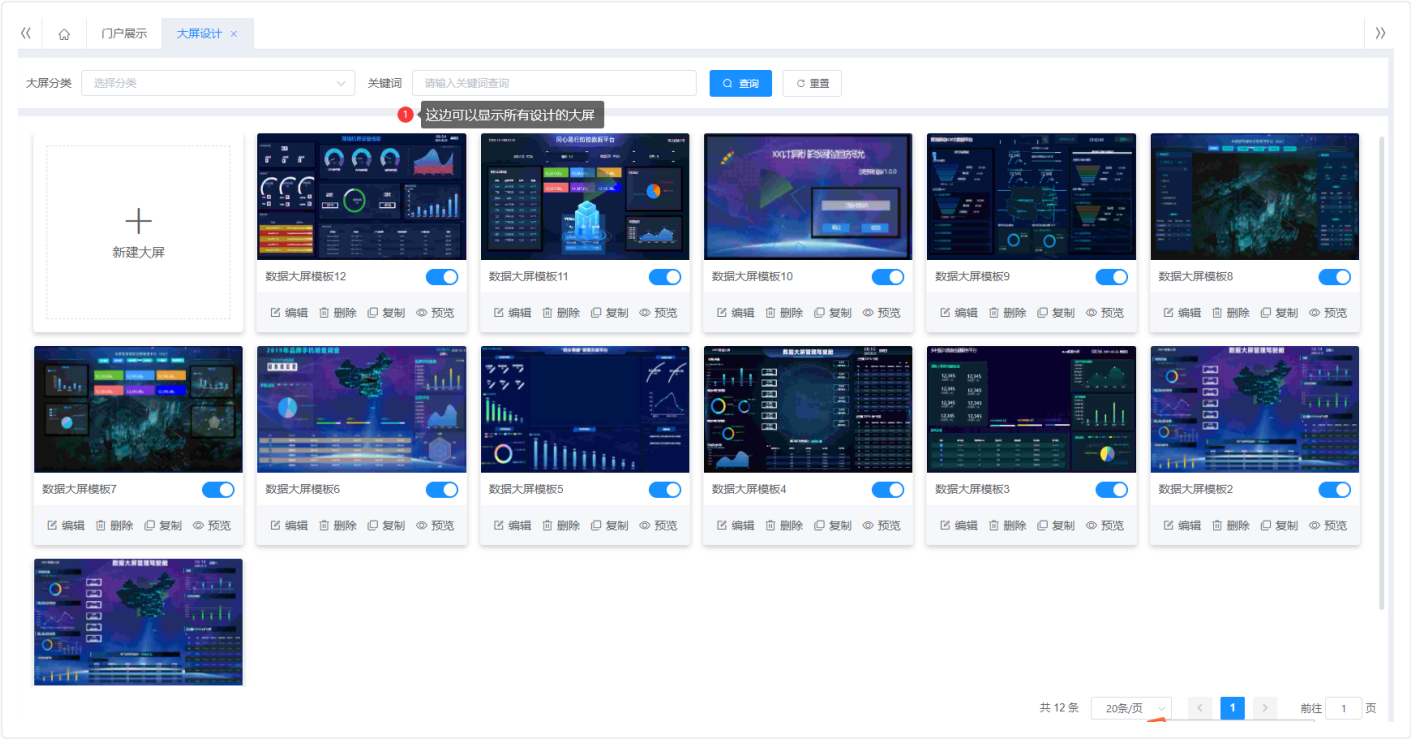
大屏设计是根据用户需求设计大屏功能，确认点击“保存”进行大屏设计或者点击关闭按钮离开该页面、当前数据未保存，或者预览大屏。如下图所示：



在大屏设计页面可以对数据编辑、删除、复制、预览等功能操作，如下图所示：

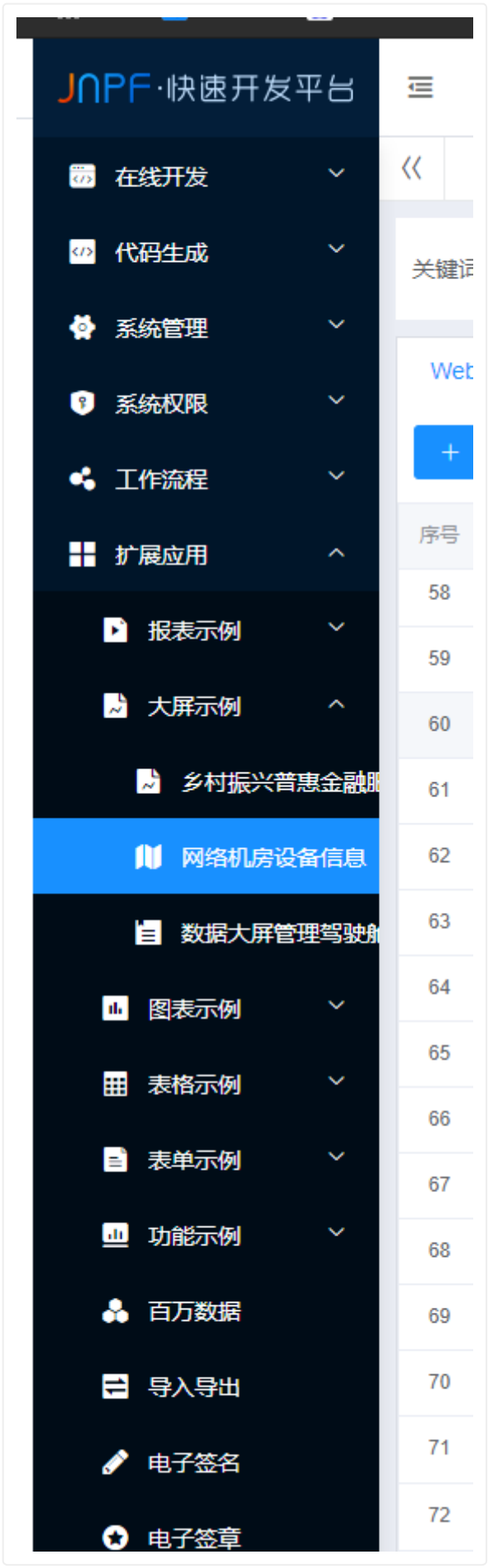
ⅰ点击“删除”按钮可以直接删除这个大屏设计成功；

ⅱ操作预览或者复制，预览大屏设计页面或者复制一份大屏；



在菜单管理里面添加这个大屏菜单，然后就可以直接看到自己设计的大屏页面，如下图所示：


编辑菜单
* 分类
Web
* 上级
大屏示例
* 名称
网络机房设备信息
* 编码
121212
* 图标
icon-ym icon-ym-mapExample
选择
* 类型
大屏
* 关联
数据大屏模板12
排序
0
状态
说明
取 消
确 定


JNPF·快速开发平台
在线开发
代码生成
系统管理
系统权限
工作流程
扩展应用
报表示例
大屏示例
乡村振兴普惠金融
网络机房设备信息
数据大屏管理驾驶
图表示例
表格示例
表单示例
功能示例
百万数据
导入导出
电子签名
电子签章
关键词
序号
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72

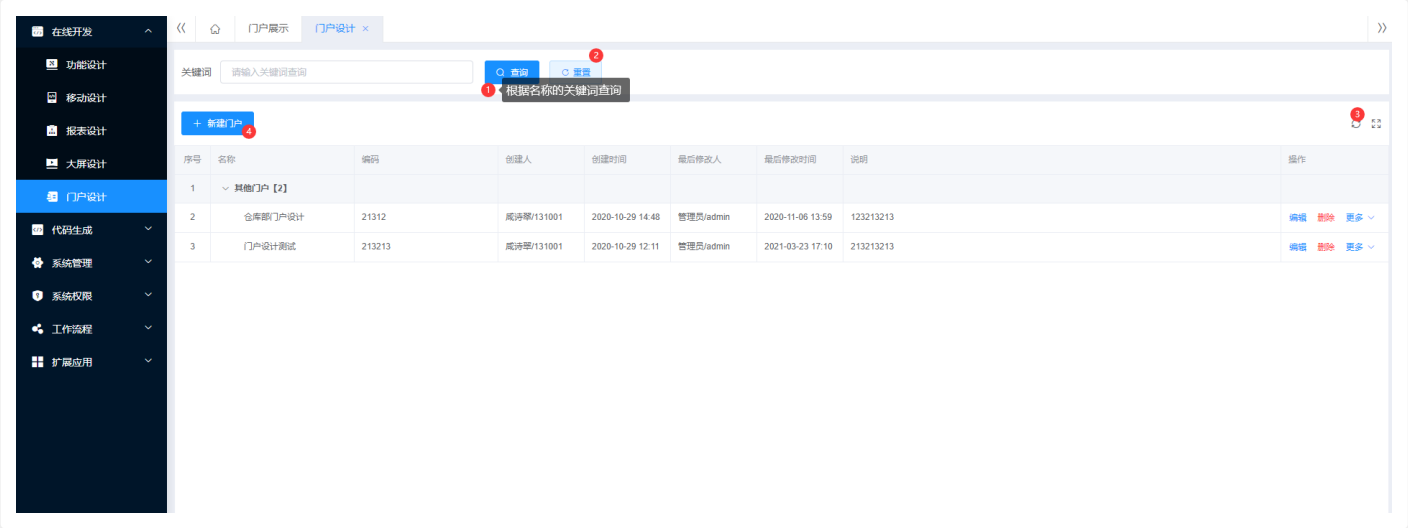
门户设计

功能描述

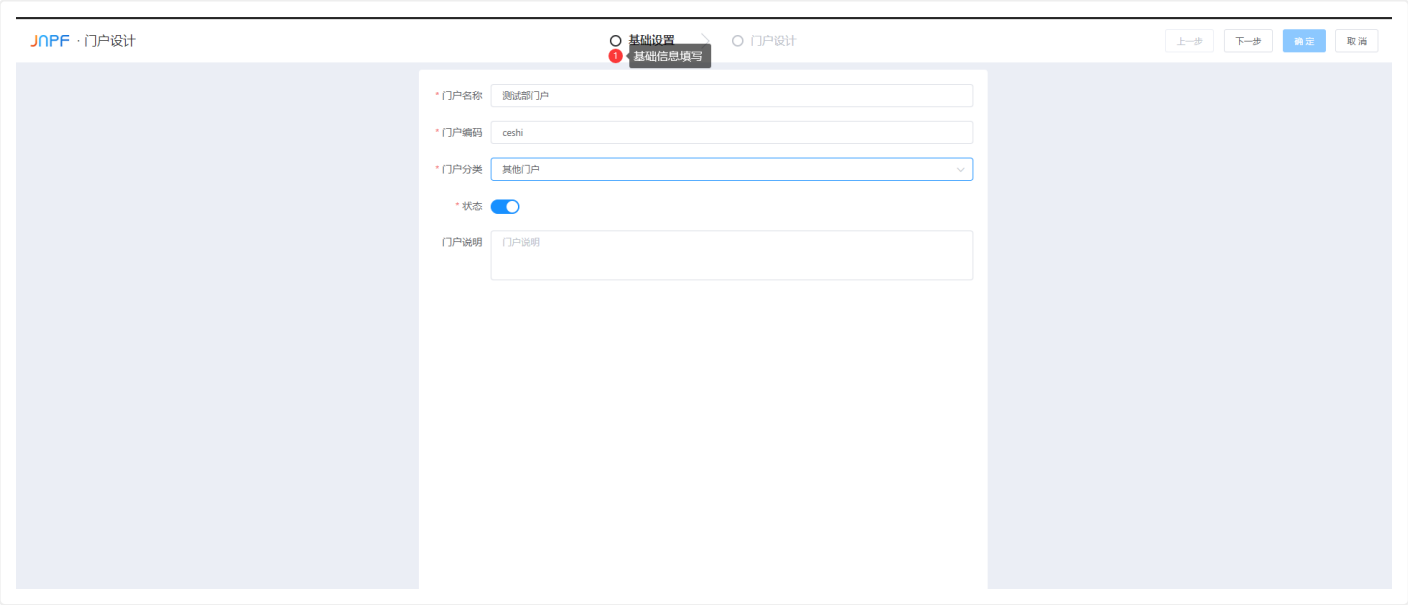
提供了多样式的门户模板，可以自由拖拽组件到画布上进行配置和布局，通过简单的拖动和设置，设计个性化的门户页面。

操作步骤

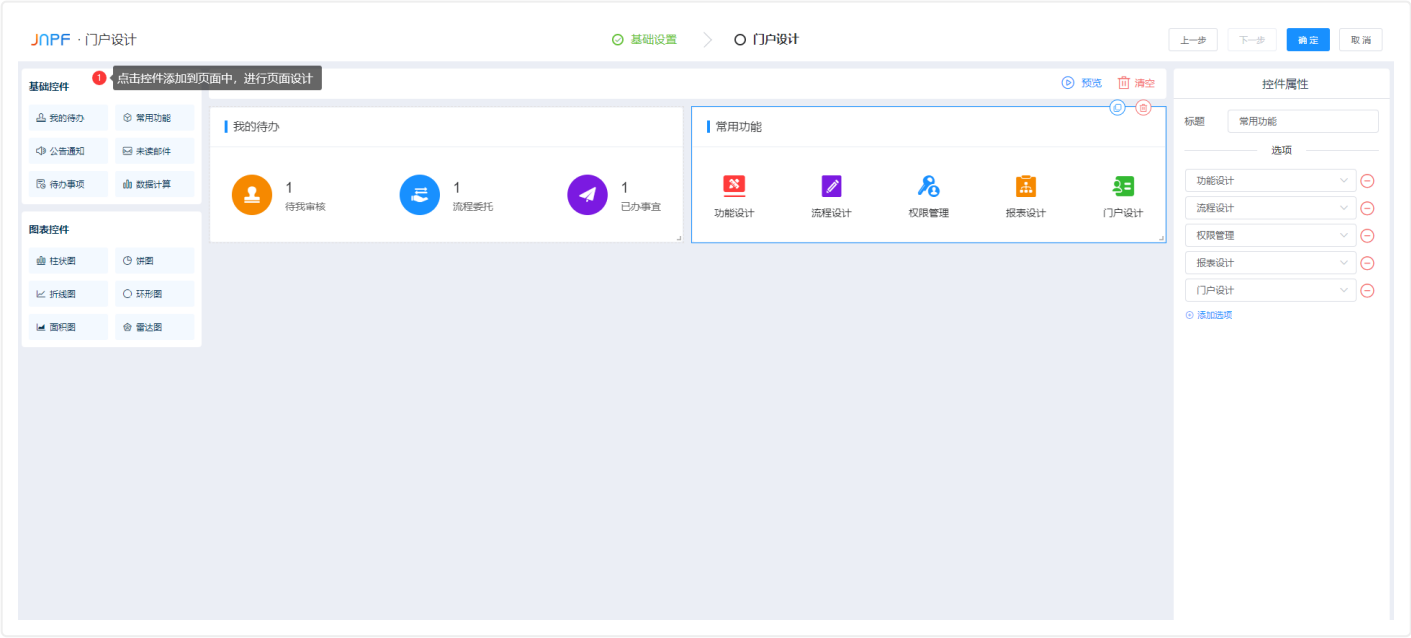
在在线开发目录下操作在门户设计，进入【门户设计】页面有查询、重置、刷新、新建门户、全屏、编辑等功能；如下图所示：



点击该页面左上角新建门户功能，进入新建门户设计页面，进行基础字段填写、红色星号表示必填项；如下图所示：



门户设计是根据用户需求设计门户设计功能，确认点击“下一步”进行门户设计或者点击取消按钮离开该页面、当前数据未保存。如下图所示：

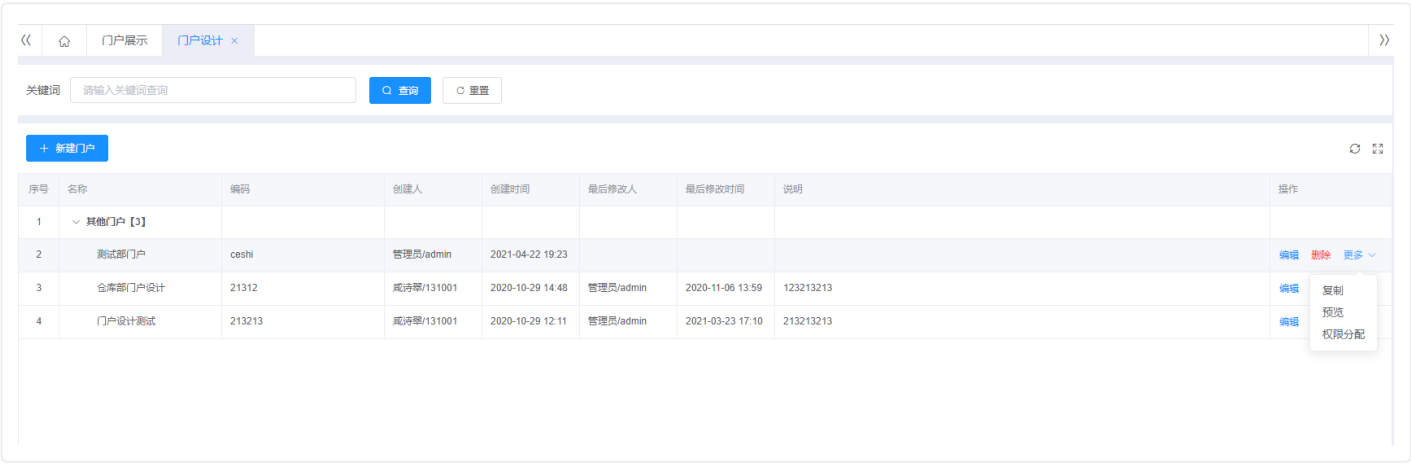


在门户设计页面可以对数据编辑、删除、更多（复制、预览、权限分配）功能操作，如下图所示：

ⅰ点击“删除”按钮可以直接删除这个移动设计成功；

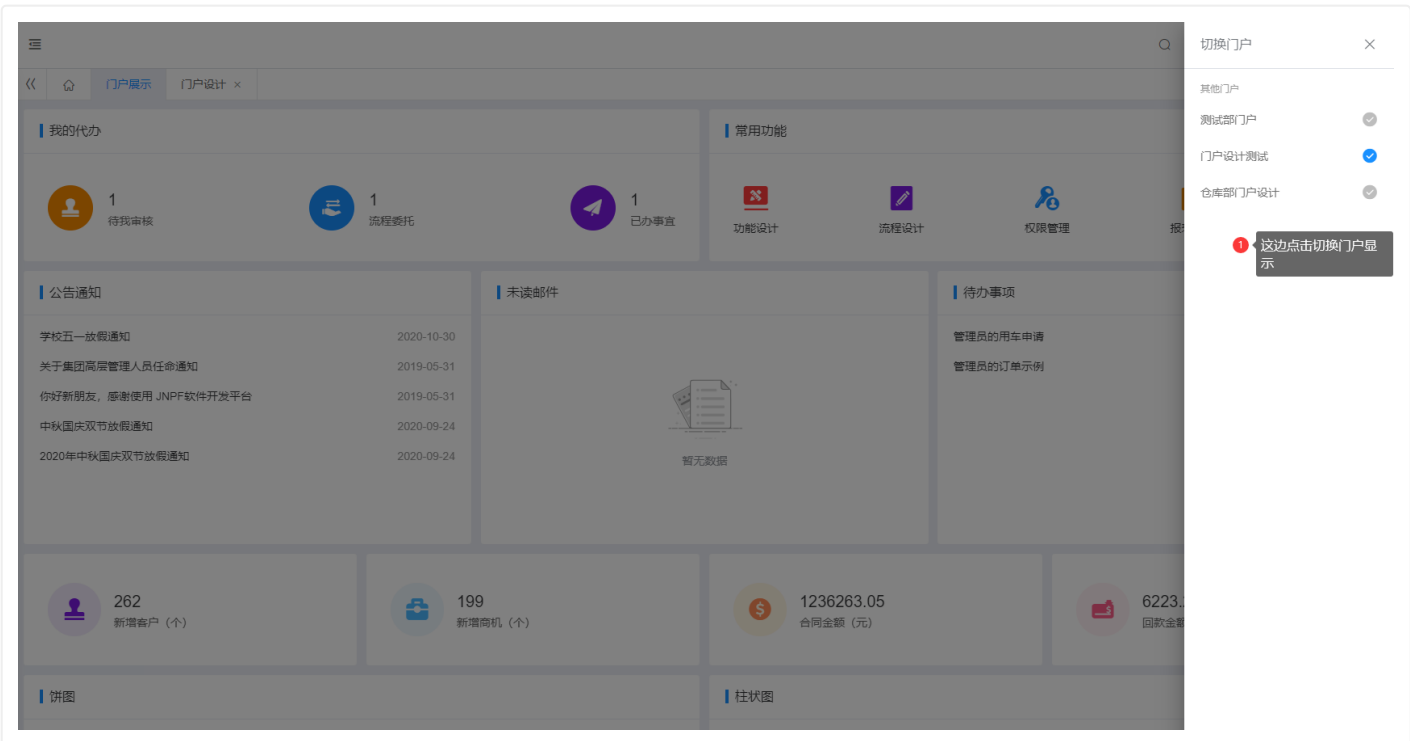
ⅱ操作更多下复制或预览，复制这个功能设计或者预览移动设计页面；

ⅲ操作权限配置，是选择那些角色可以使用这个门户；



在平台上有一个门户展示，右边这个按钮点击，

可以选择配置这个用户的门户。



代码生成

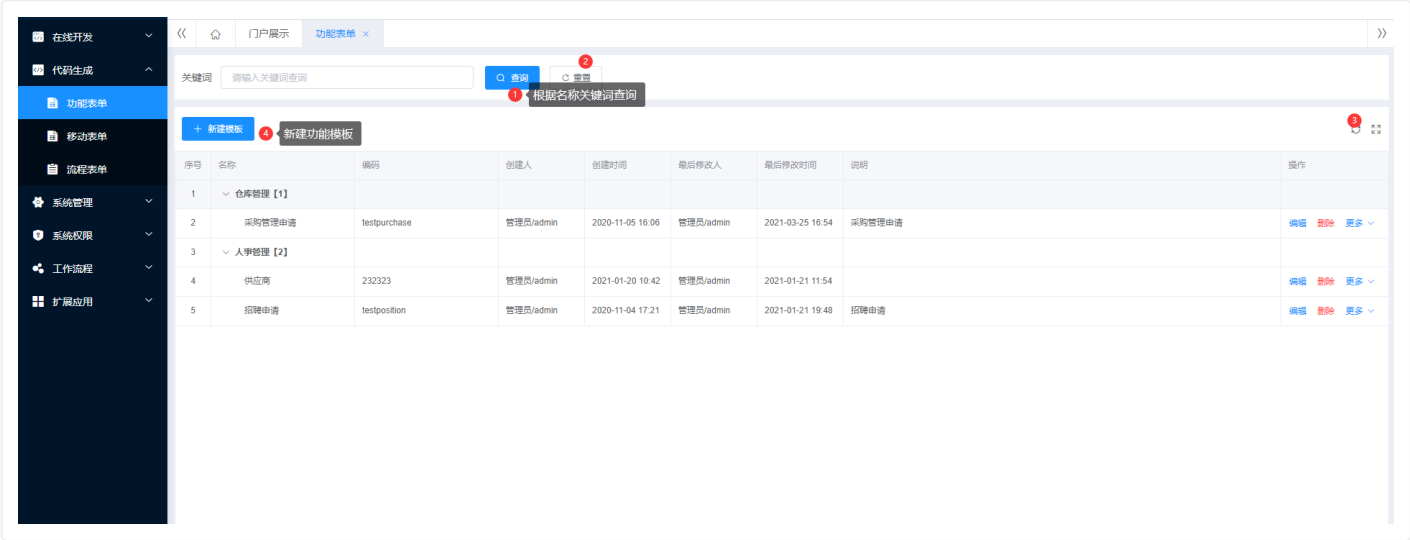
功能表单

功能描述

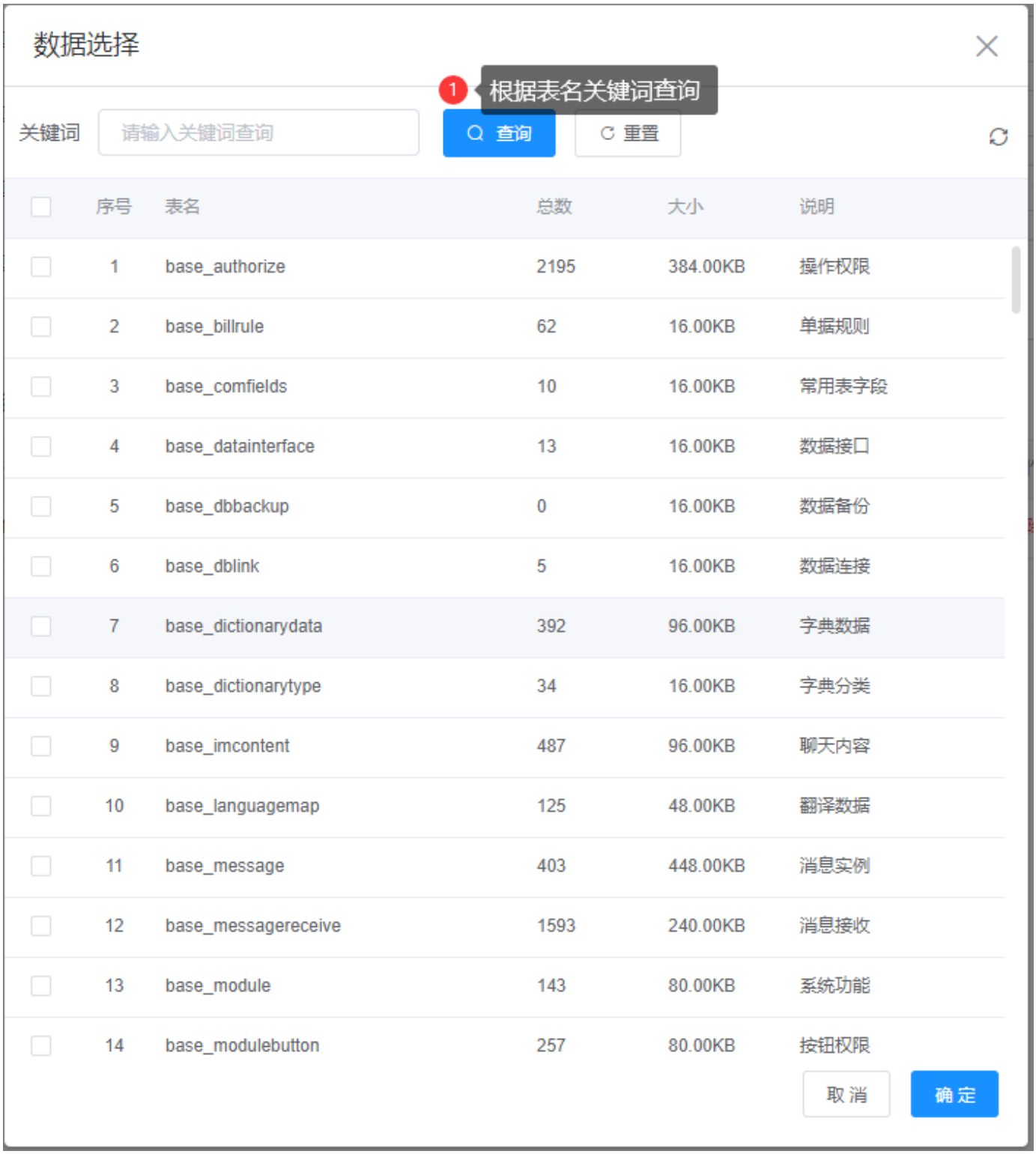
功能模块中选择单表或多表形式进行数据关联，在可视化功能设计的基础上，可下载或查看源代码进行二次开发。

操作步骤

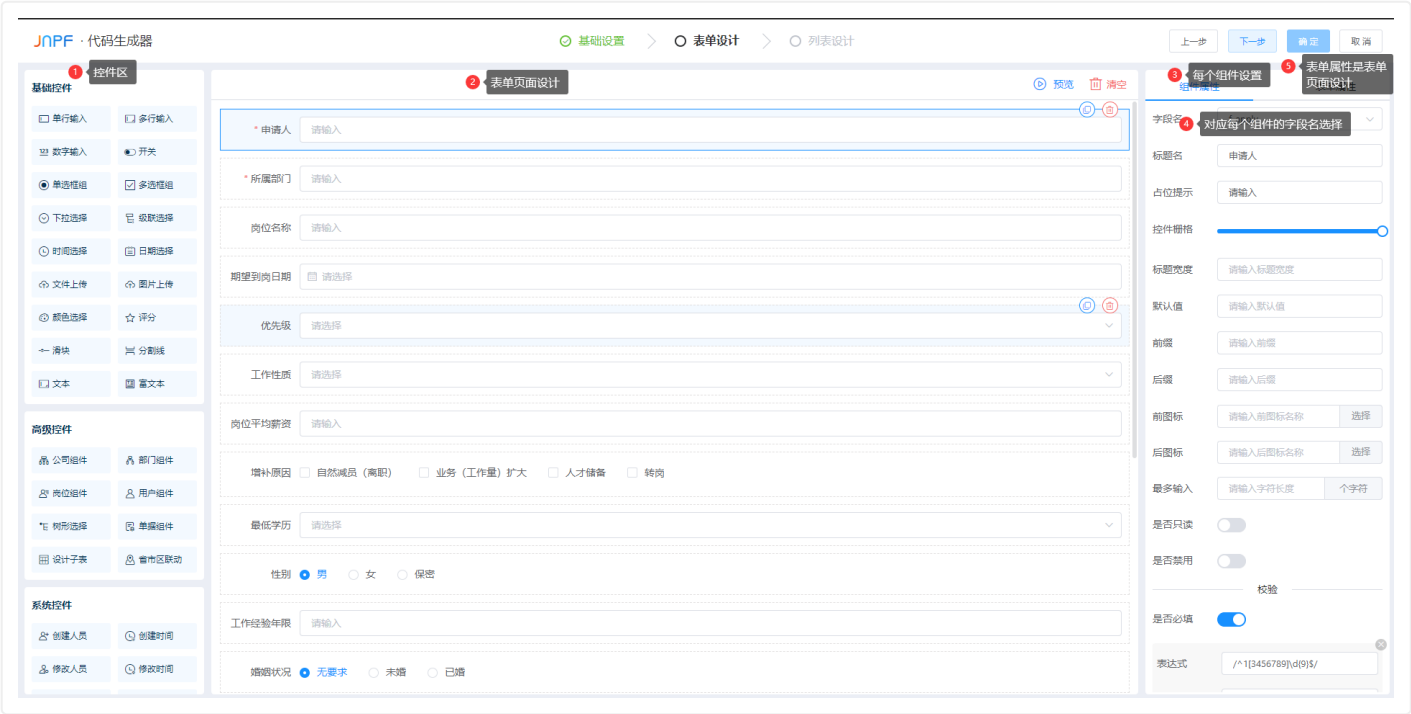
在代码生成目录下操作在功能表单，进入【功能表单】页面有查询、重置、刷新、新建模板、全屏等功能；如下图所示：



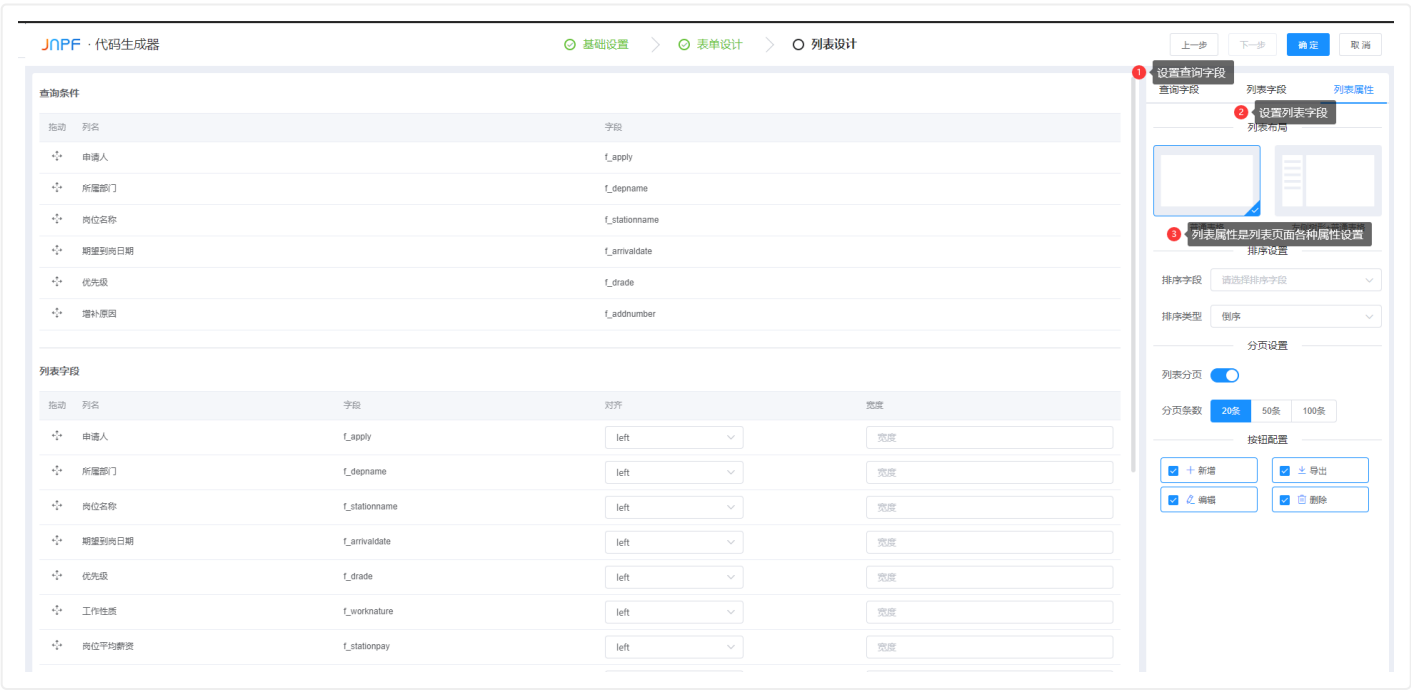
点击该页面左上角新建模板功能，进入新建代码生成器页面，进行字段填写、红色星号表示必填项；基础设置填写一些功能基础信息和绑定数据库表(单表或者多表)后，确认点击“下一步”进行表单设计或者点击取消按钮离开该页面、当前数据未保存。如下图所示：



表单设计是根据用户需求设计界面绑定数据字段，确认点击“下一步”进行列表设计或者点击取消按钮离开该页面、当前数据未保存。如下图所示：



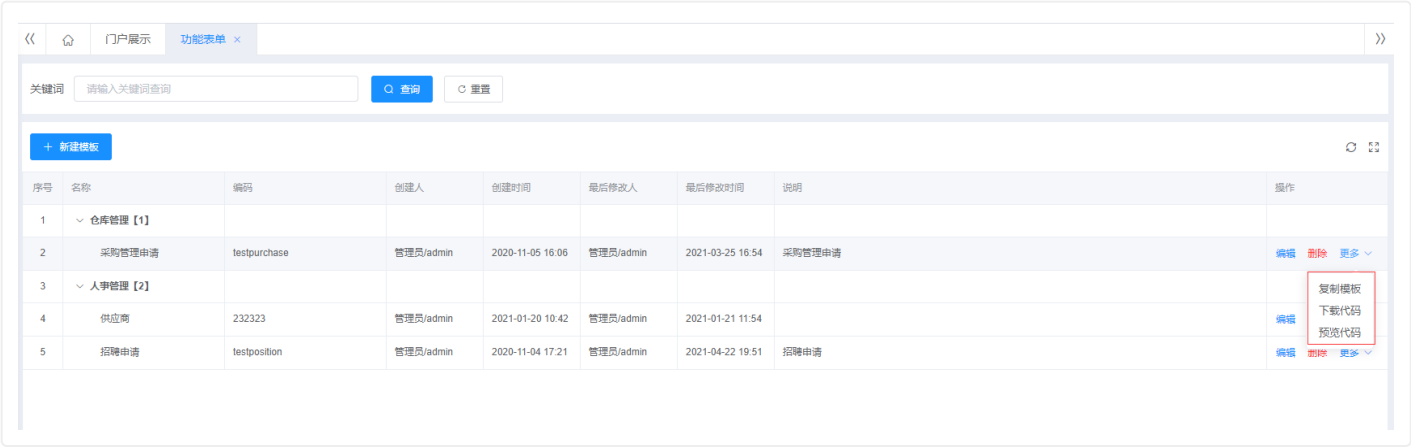
列表设计是根据用户设计的功能显示列表属性、列表字段、查询字段、排序字段等设计，点击“确定”保存这个功能表单或者点击取消按钮离开该页面、当前数据未保存。如下图所示：

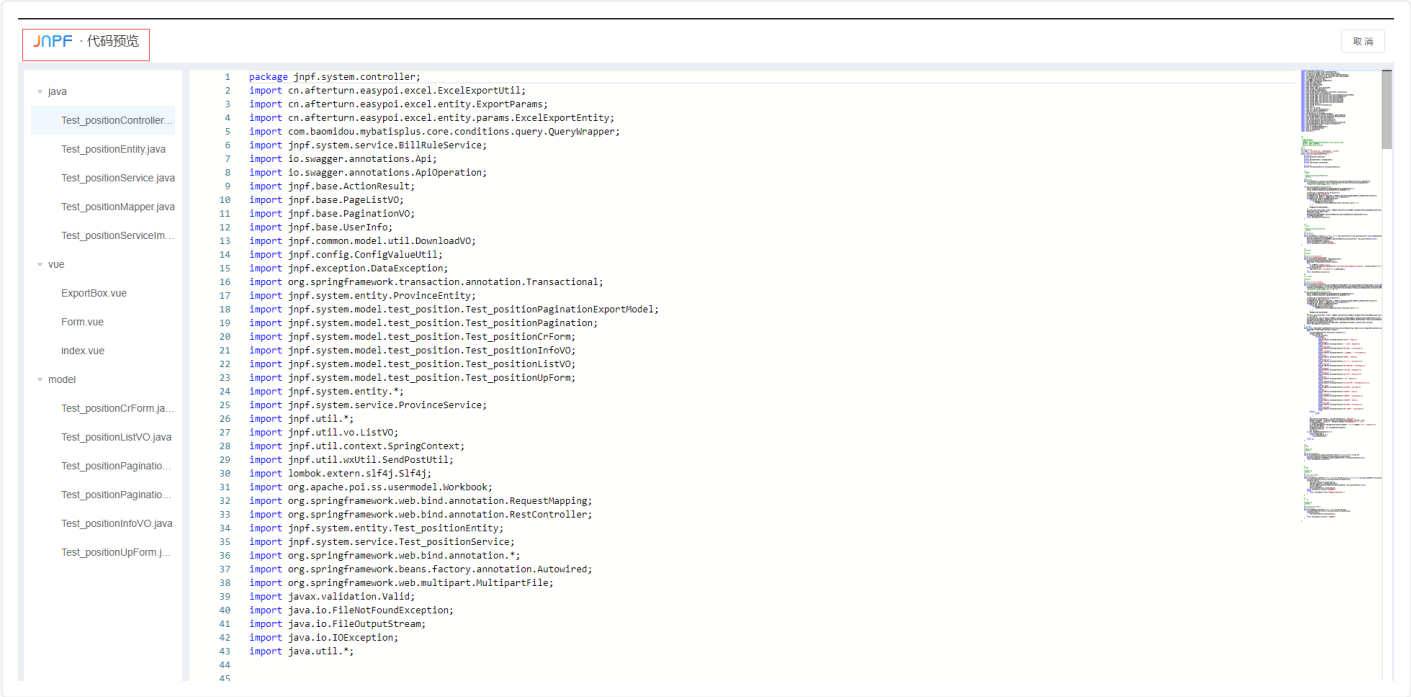


在功能设计页面可以对数据编辑、删除、更多（复制模板、下载代码、预览代码）功能操作，如下图所示：

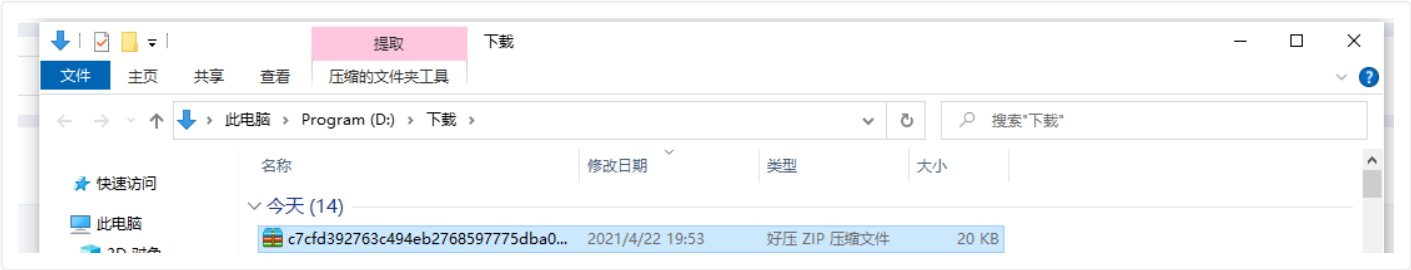
ⅰ 点击“删除”按钮可以直接删除这个功能表单成功；

ⅱ 操作更多下复制模板、下载代码、预览代码，复制这个功能表单或者预览功能表单页面；





可进行下载代码操作，在下载任务中显示下载文件名、保存到下载地址打开文件或打开文件夹位置。如下图所示：



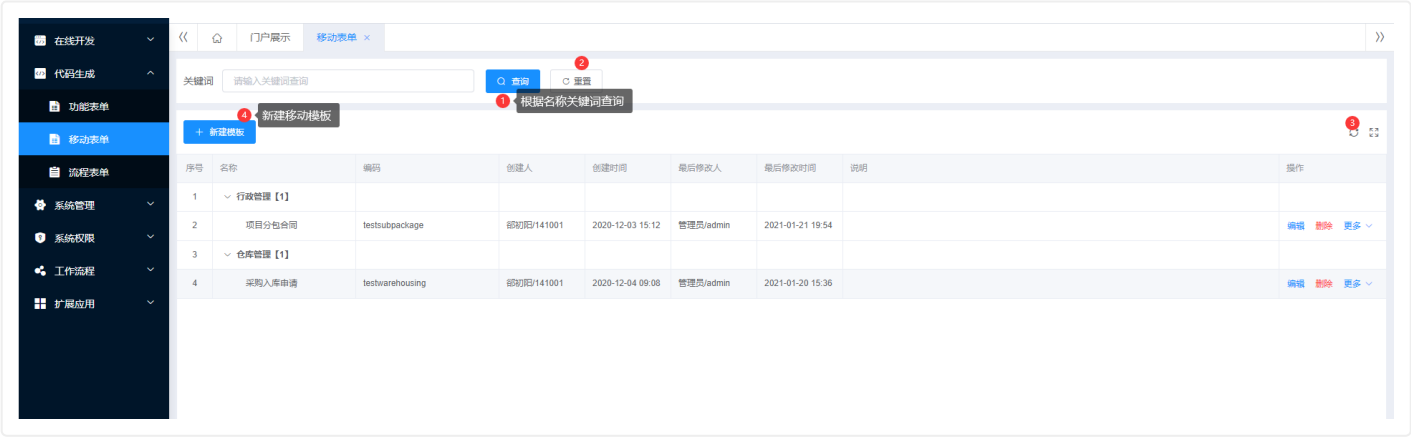
移动表单

功能描述

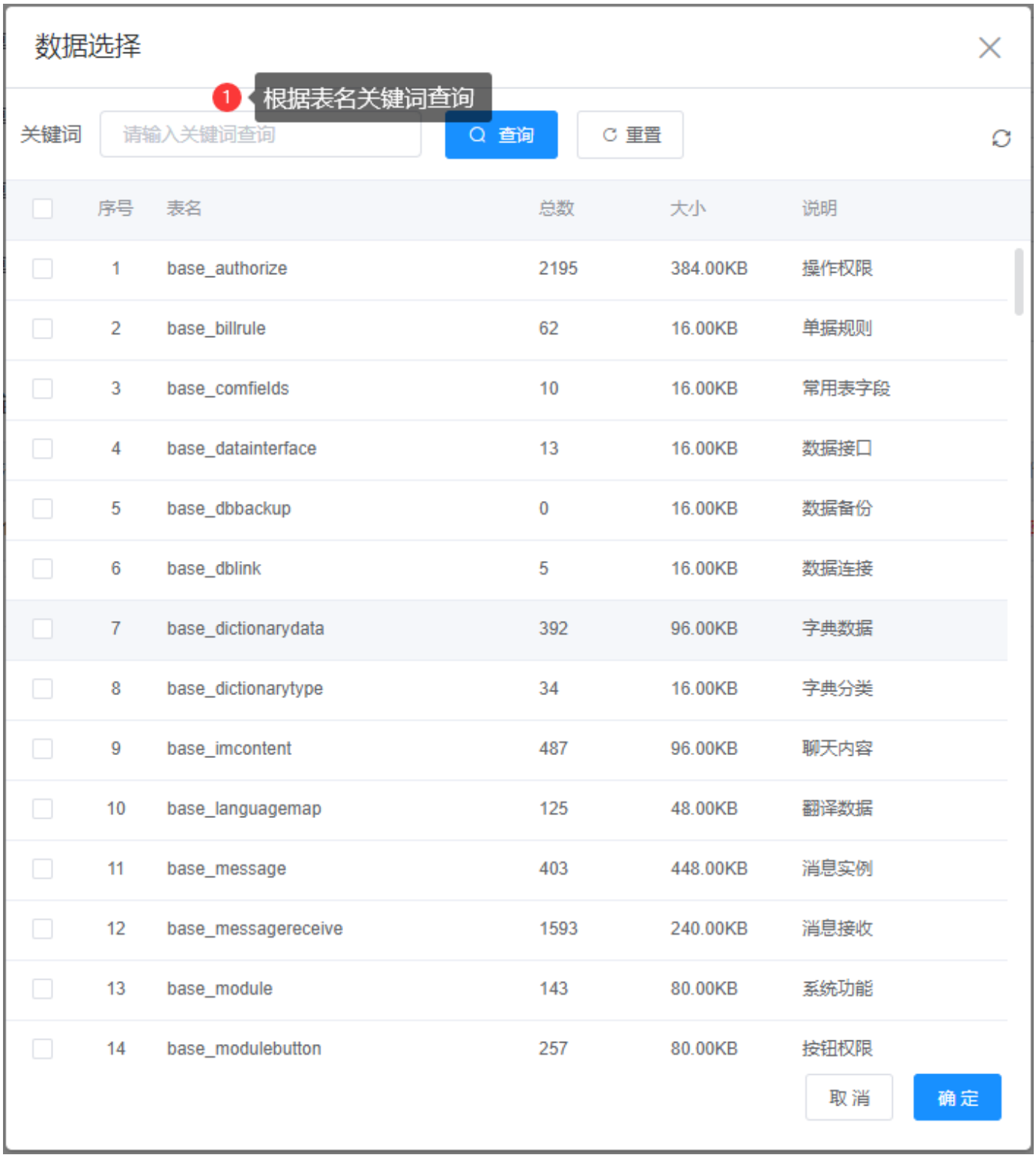
移动模块中选择单表或多表形式进行数据关联，在可视化移动设计的基础上，可下载或查看源代码进行二次开发。

操作步骤

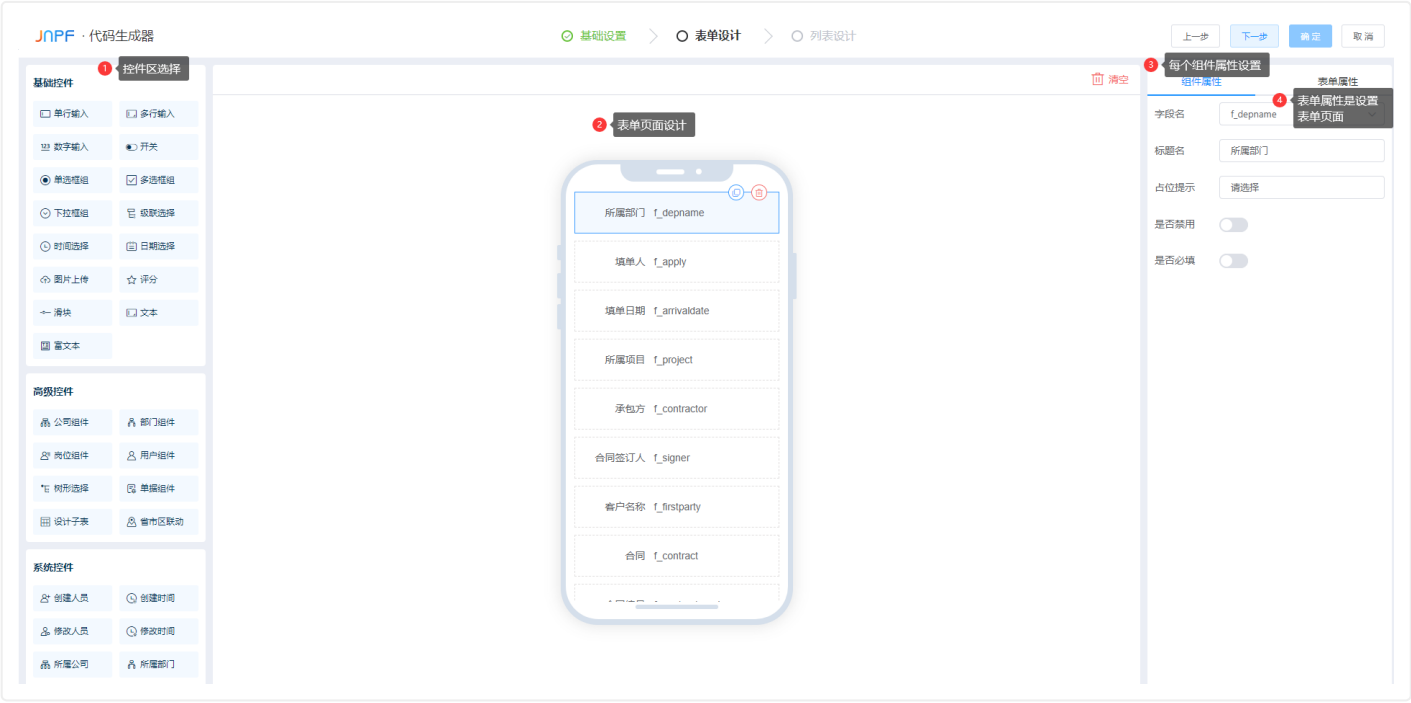
在代码生成目录下操作在移动表单，进入【移动表单】页面有查询、重置、刷新、新建模板、全屏等功能；如下图所示：



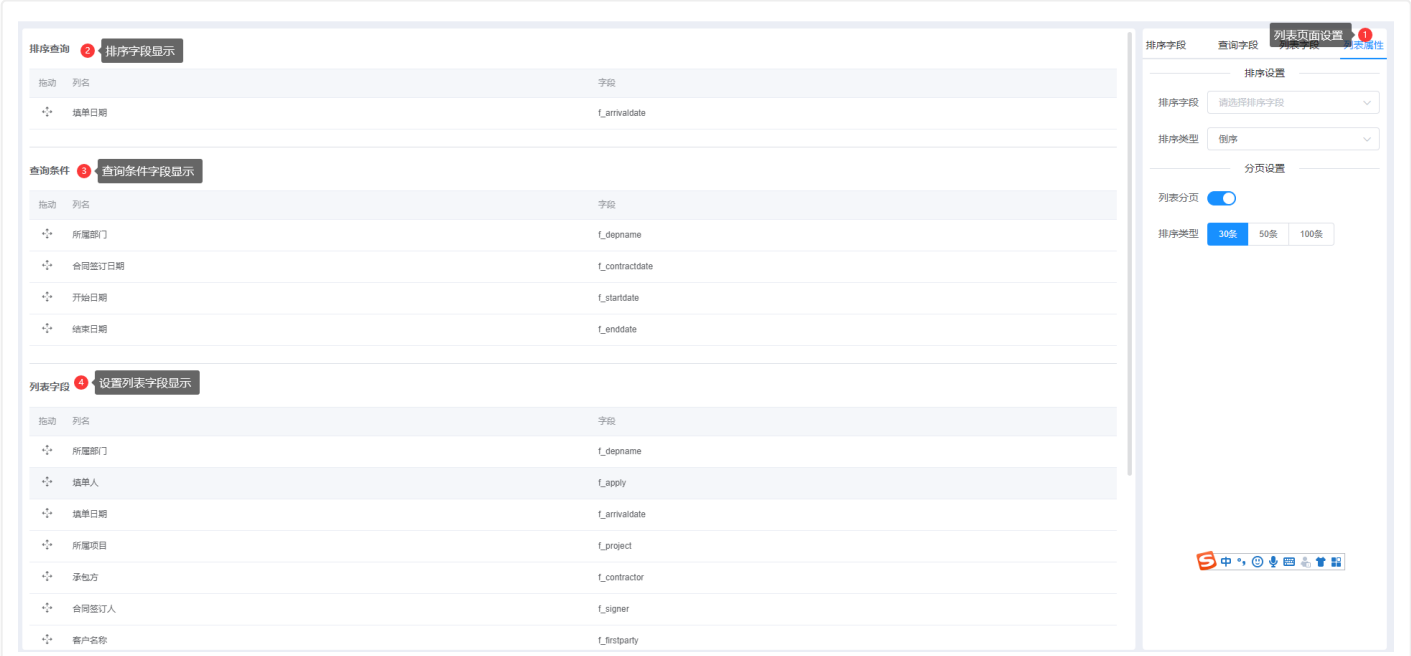
点击该页面左上角新建模板功能，进入新建代码生成器页面，进行字段填写、红色星号表示必填项；基础设置填写一些功能基础信息和绑定数据库表(单表或者多表)后，确认点击“下一步”进行表单设计或者点击取消按钮离开该页面、当前数据未保存。如下图所示：



表单设计是根据用户需求设计界面绑定数据字段，确认点击“下一步”进行列表设计或者点击取消按钮离开该页面、当前数据未保存。如下图所示：



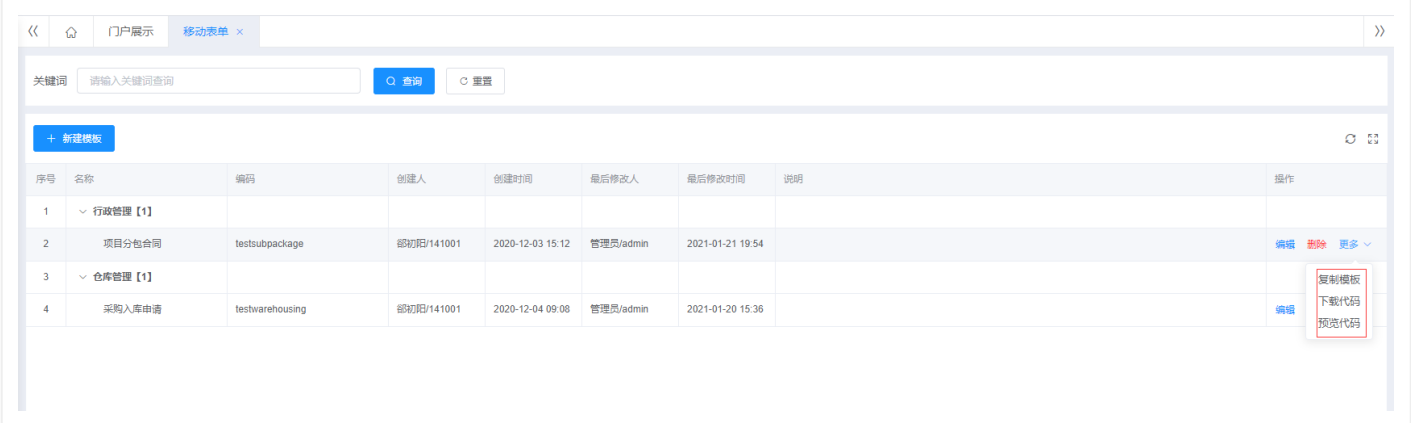
列表设计是根据用户设计的功能显示列表属性、列表字段、查询字段、排序字段等设计，点击“确定”保存这个功能表单或者点击取消按钮离开该页面、当前数据未保存。如下图所示：



在功能设计页面可以对数据编辑、删除、更多（复制模板、下载代码、预览代码）功能操作，如下图所示：

ⅰ点击“删除”按钮可以直接删除这个移动表单成功；

ⅱ操作更多下复制模板、下载代码、预览代码，复制这个功能表单或者预览功能表单页面；



可进行下载代码操作，在下载任务中显示下载文件名、保存到下载地址打开文件或打开文件夹位置。如下图所示：

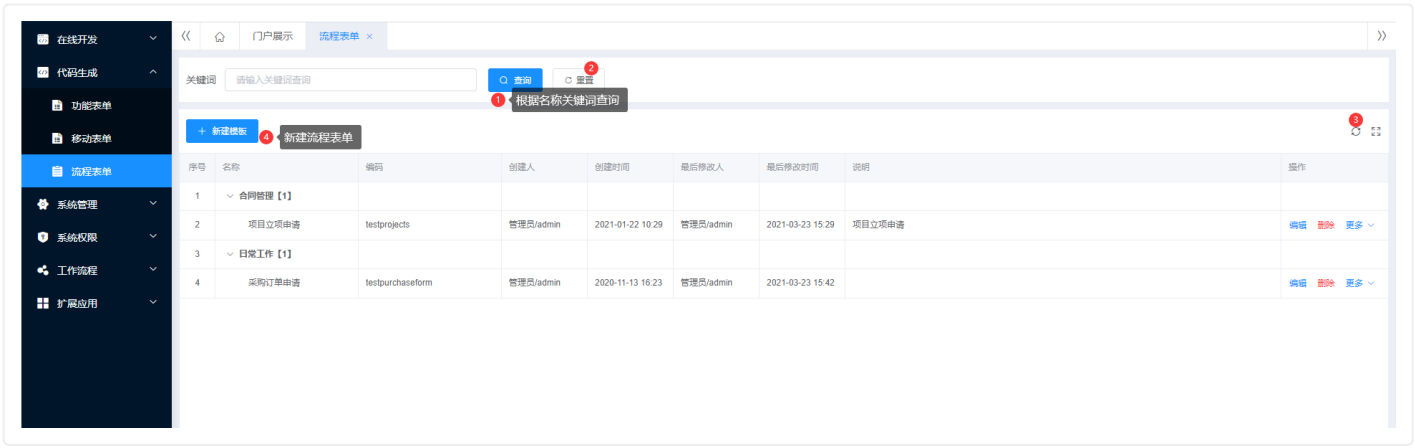
流程表单

功能描述

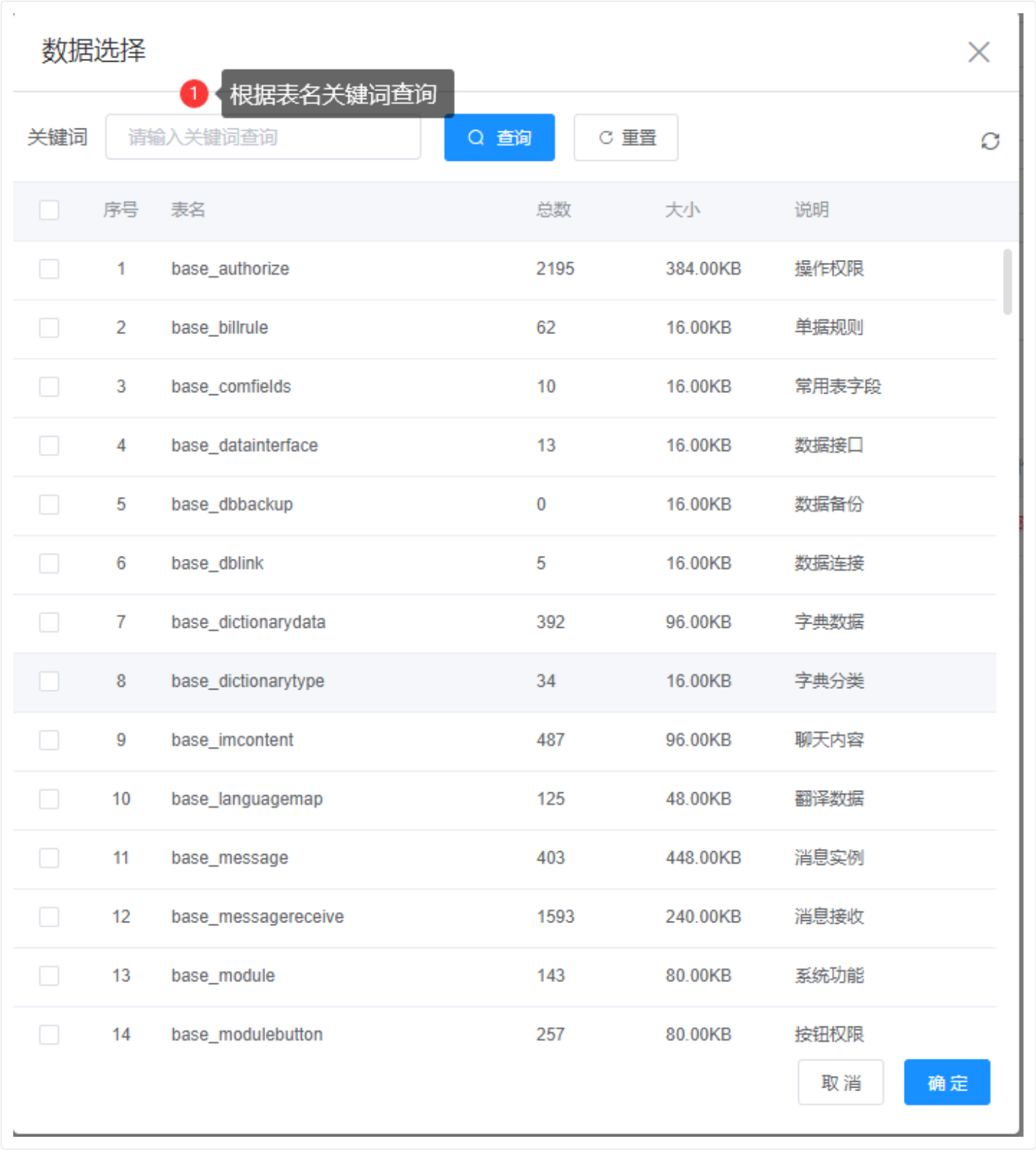
流程表单模块中选择单表或多表形式进行数据关联，在可视化流程设计的基础上，可下载或查看源代码进行二次开发。

操作步骤

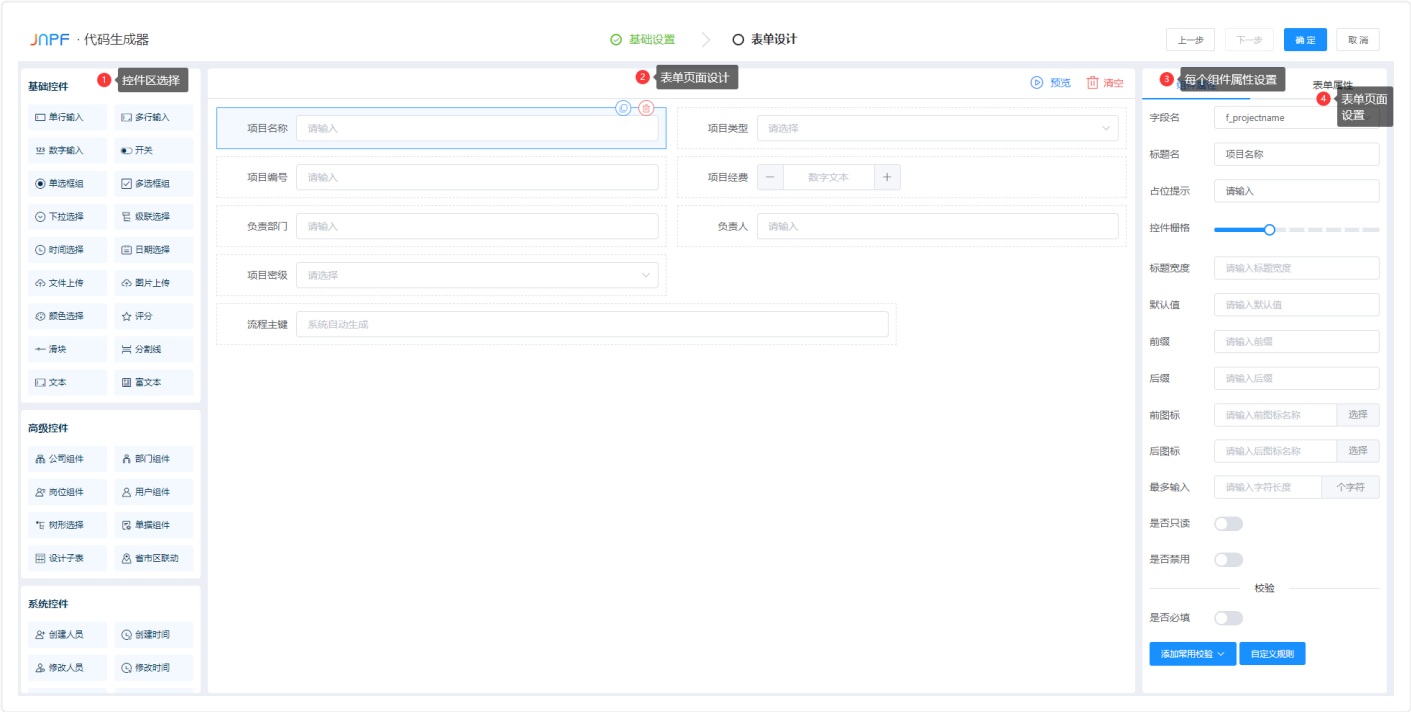
在代码生成目录下操作在流程表单，进入【流程表单】页面有查询、重置、刷新、新建模板、全屏等功能；如下图所示：



点击该页面左上角新建模板功能，进入新建代码生成器页面，进行字段填写、红色星号表示必填项；基础设置填写一些功能基础信息和绑定数据库表(单表或者多表)后，确认点击“下一步”进行表单设计或者点击取消按钮离开该页面、当前数据未保存。如下图所示：



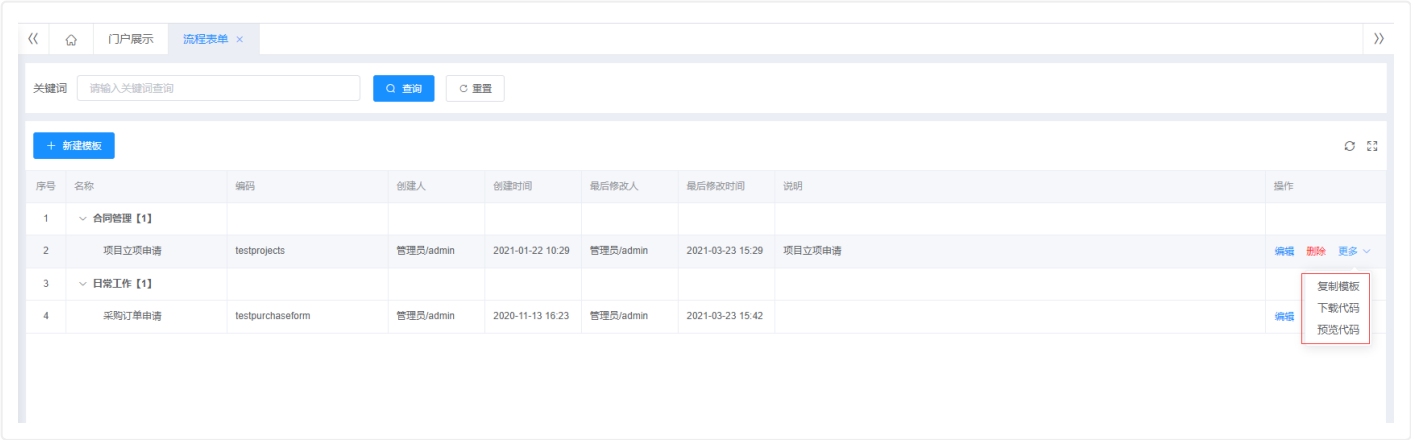
表单设计是根据用户需求设计界面绑定数据字段，，点击“确定”保存这个功能表单或者点击取消按钮离开该页面、当前数据未保存。如下图所示：

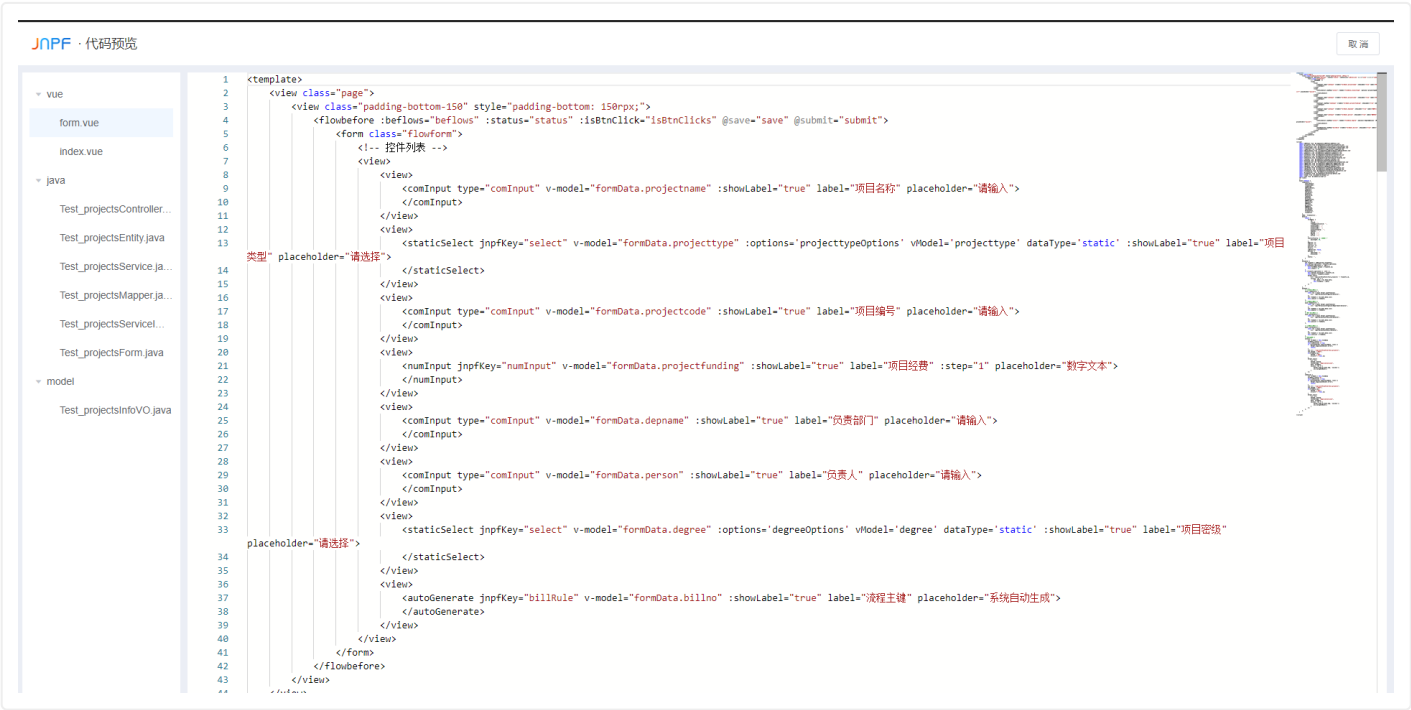


在功能设计页面可以对数据编辑、删除、更多（复制模板、下载代码、预览代码）功能操作，如下图所示：

ⅰ点击“删除”按钮可以直接删除这个功能设计成功；

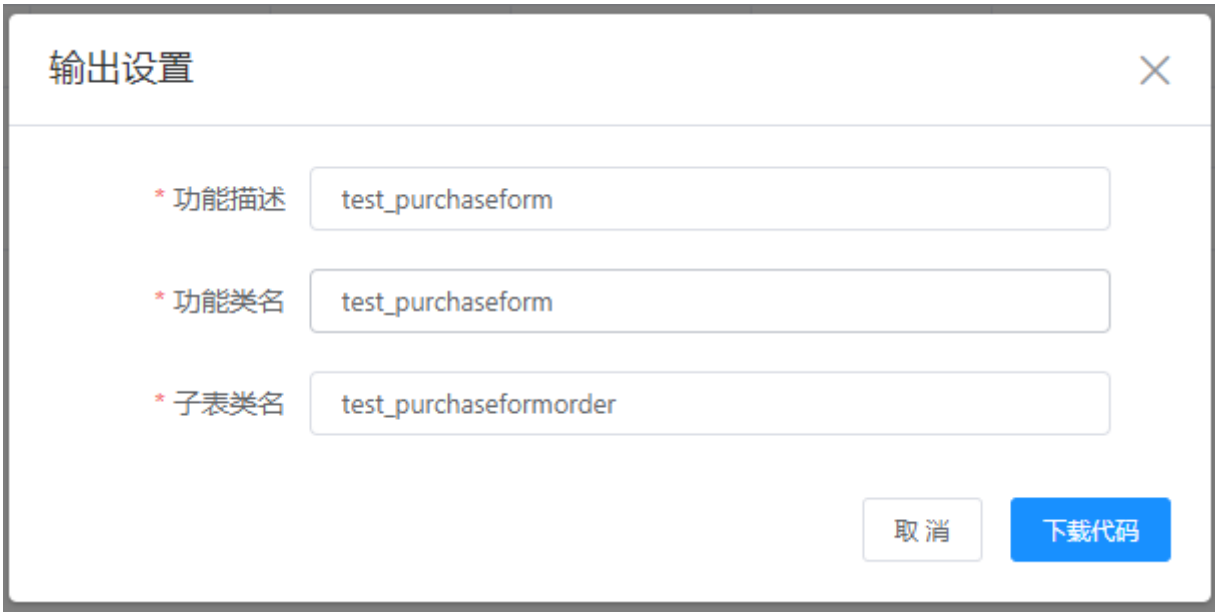
ⅱ操作更多下复制模板、下载代码、预览代码，复制这个功能表单或者预览功能表单页面；如下图所示：

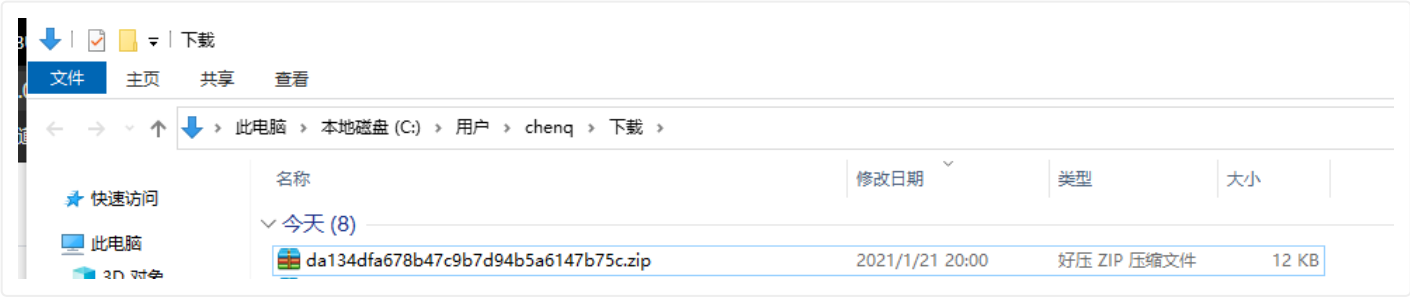




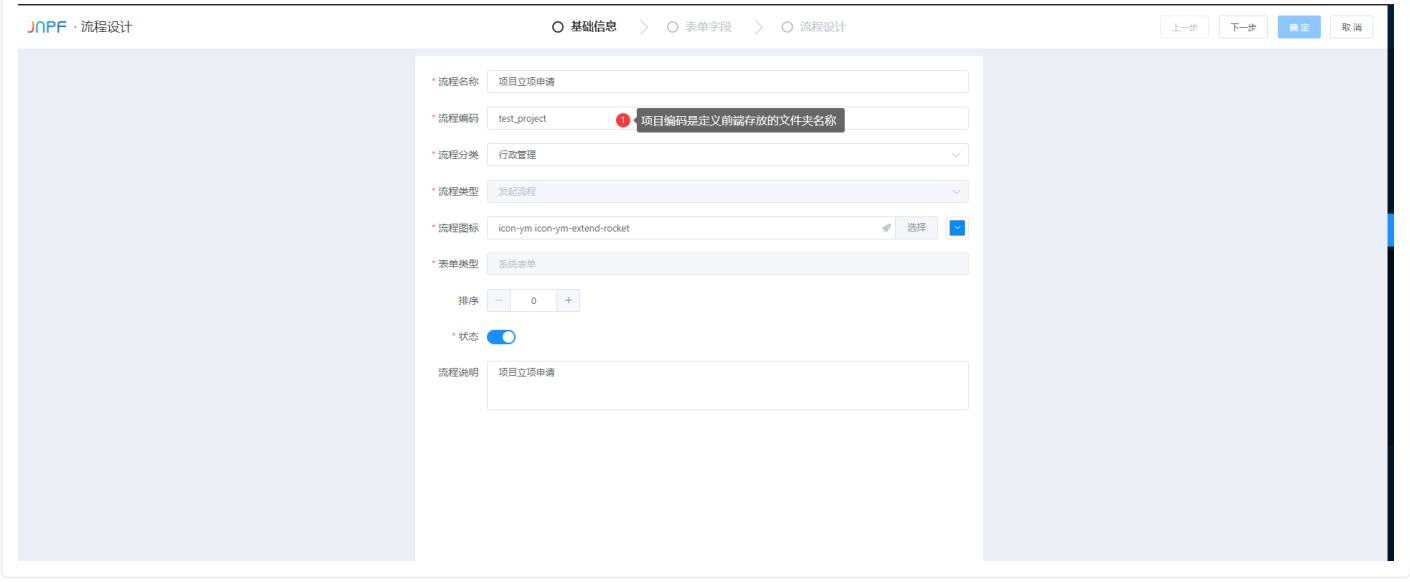

```
<template>
    <view class="page">
        <view class="padding-bottom-150" style="padding-bottom: 150rpx;">
            <flowbefore :beflows="beflows" :status="status" :isBtnClick="isBtnClicks" @save="save" @submit="submit">
                <form class="flowform">
                    <!-- 控件列表 -->
                    <view>
                        <view>
                            <comInput type="comInput" v-model="formData.projectname" :showLabel="true" label="项目名称" placeholder="请输入">
                            </comInput>
                        </view>
                        <view>
                            <staticSelect jnpfKey="select" v-model="formData.projecttype" :options='projecttypeOptions' vModel='projecttype' dataType='static' :showLabel="true" label="项目
类型" placeholder="请选择">
                            </staticSelect>
                        </view>
                        <view>
                            <comInput type="comInput" v-model="formData.projectcode" :showLabel="true" label="项目编号" placeholder="请输入">
                            </comInput>
                        </view>
                        <view>
                            <numInput jnpfKey="numInput" v-model="formData.projectfunding" :showLabel="true" label="项目经费" :step="1" placeholder="数字文本">
                            </numInput>
                        </view>
                        <view>
                            <comInput type="comInput" v-model="formData.depname" :showLabel="true" label="负责部门" placeholder="请输入">
                            </comInput>
                        </view>
                        <view>
                            <comInput type="comInput" v-model="formData.person" :showLabel="true" label="负责人" placeholder="请输入">
                            </comInput>
                        </view>
                        <view>
                            <staticSelect jnpfKey="select" v-model="formData.degree" :options='degreeOptions' vModel='degree' dataType='static' :showLabel="true" label="项目密级"
placeholder="请选择">
                            </staticSelect>
                        </view>
                        <view>
                            <autoGenerate jnpfKey="billRule" v-model="formData.billno" :showLabel="true" label="流程主键" placeholder="系统自动生成">
                            </autoGenerate>
                        </view>
                    </view>
                </form>
            </flowbefore>
        </view>
```

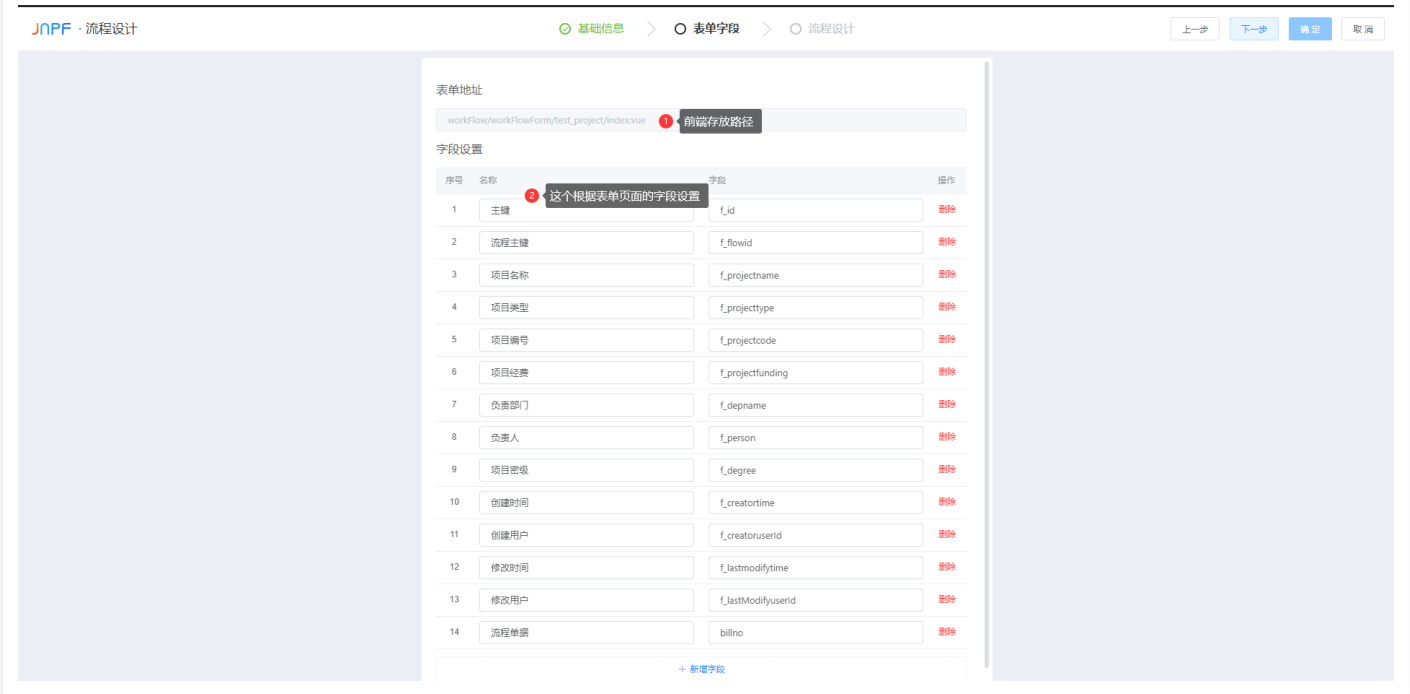
可进行下载代码操作，在下载任务中显示下载文件名、保存到下载地址打开文件或打开文件夹位置。如下图所示：





把代码放在本地项目里面运行起来，在流程设计功能这边新建一个系统表单，如下图所示





设置完成上面步骤，一个流程使用起来。

系统管理

系统配置

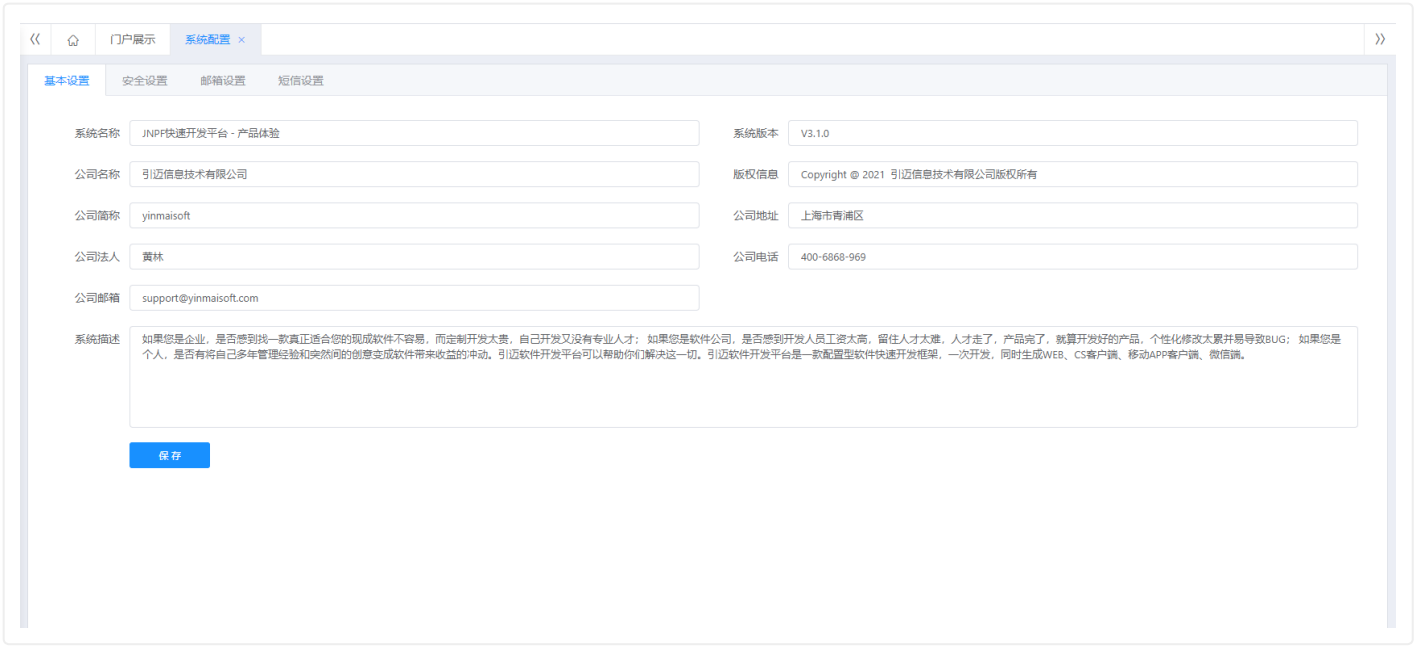
功能描述

在该系统中配置一系列基本信息，使大家对系统和公司的了解。

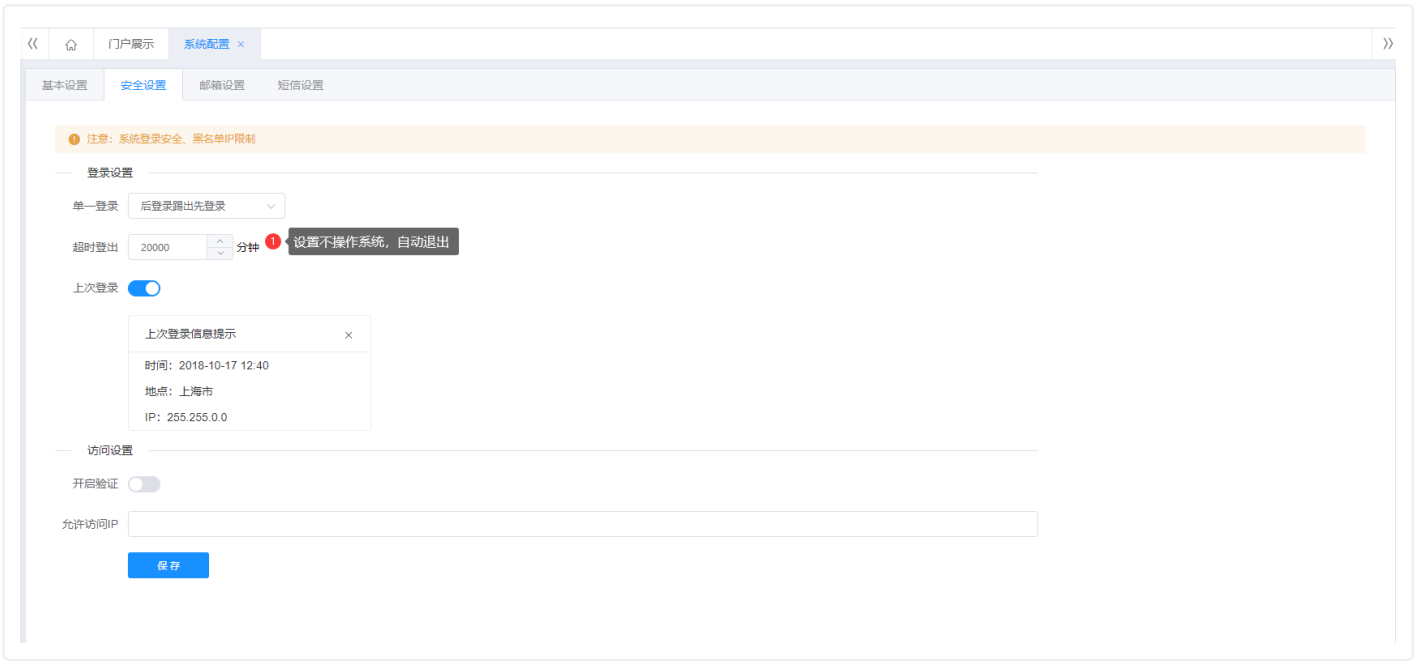
操作步骤

在系统管理目录下操作在系统配置，进入【系统配置】页面需要填写系统基础设置、安全设置、邮箱设置、短信设置功能界面填写。

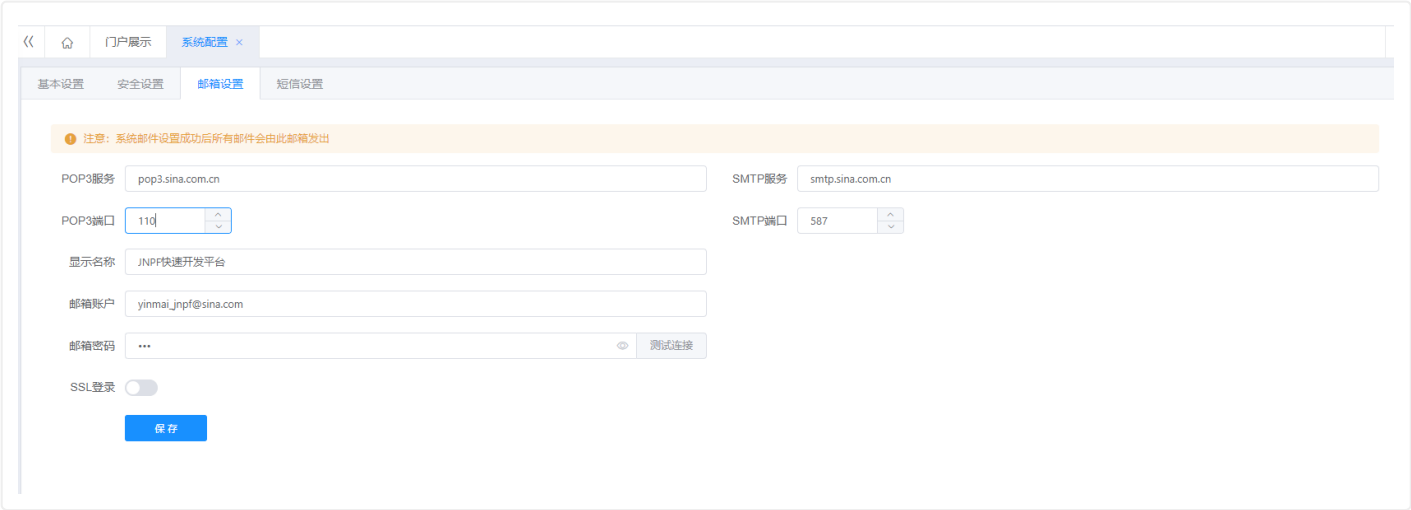
基础设置信息填写如下图所示：



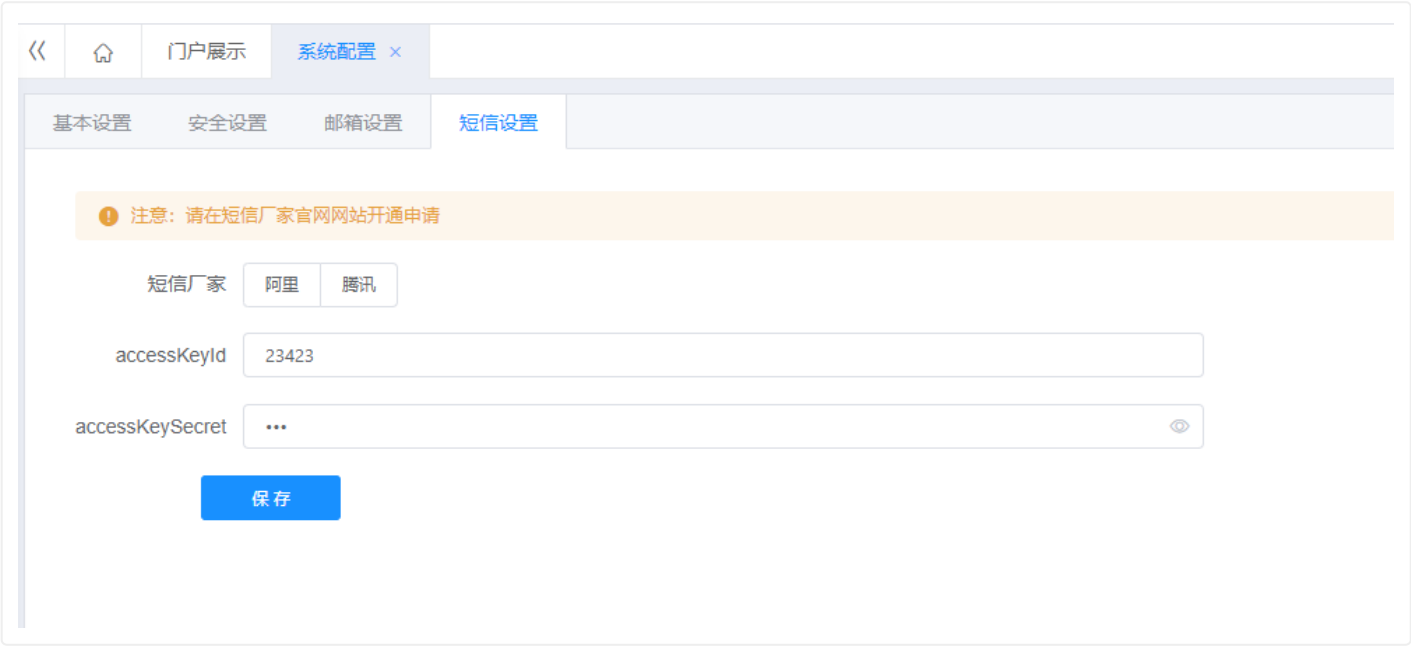
安全设置单词登录时间设置、上次登录地方以及应许访问IP地址；如下图所示：



邮箱设置进行POP3服务、POP3端口、SMTP服务、SMTP端口、显示名称、账户和密码、是否登录等信息数据填写完点击保存按钮，如下图所示：



短信设置页面，如下图所示：



第三方配置

短信配置

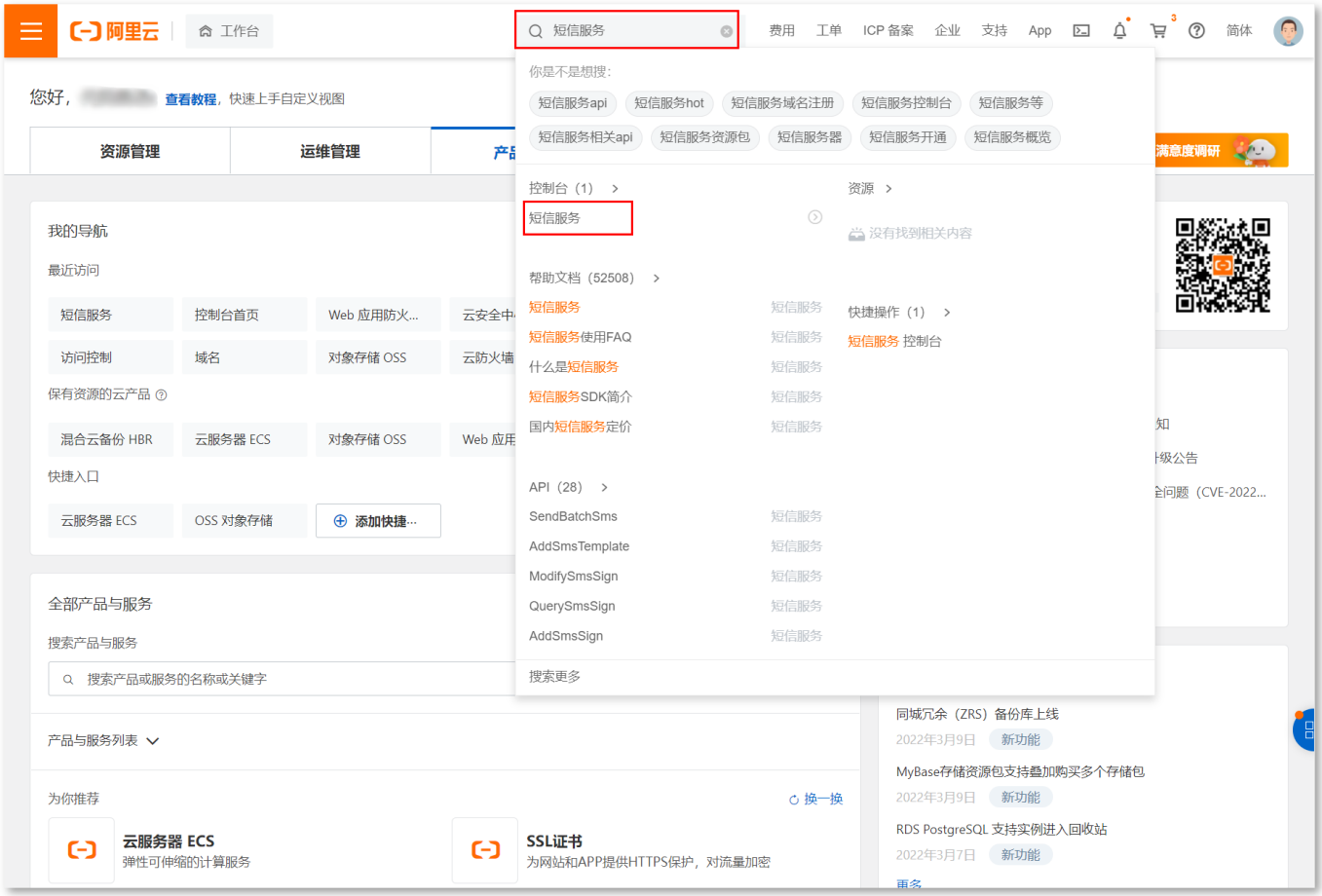
> JNPF快速开发平台支持【阿里云短信】和【腾讯云短信】

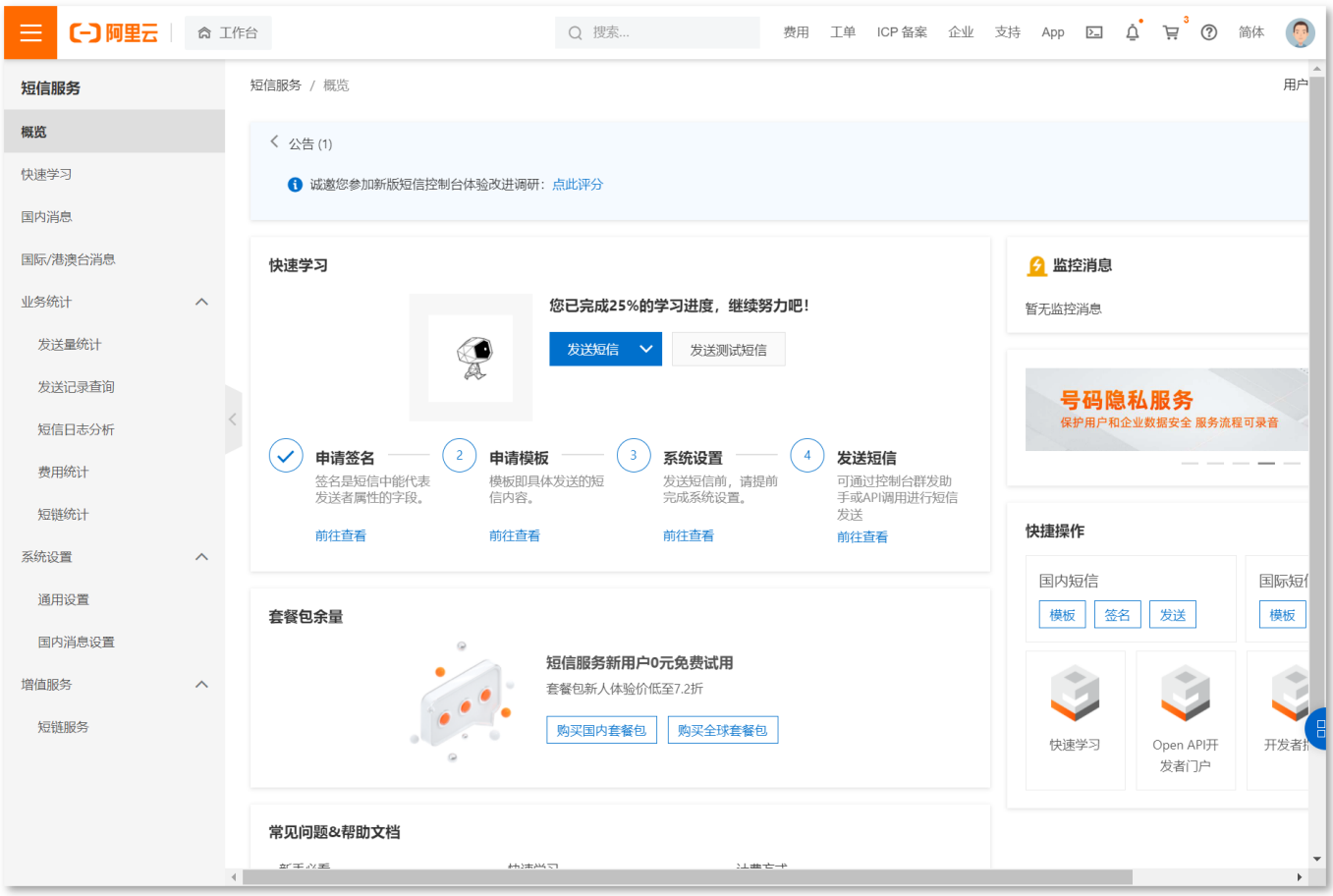
阿里短信配置

一、 登录阿里云（https:///cloud.tencent.com)控制台

二、在控制台首页搜索【短信服务】，进入【短信服务】短信服务概览页面

如果未开通【短信服务】，系统会提示开通【短信服务】

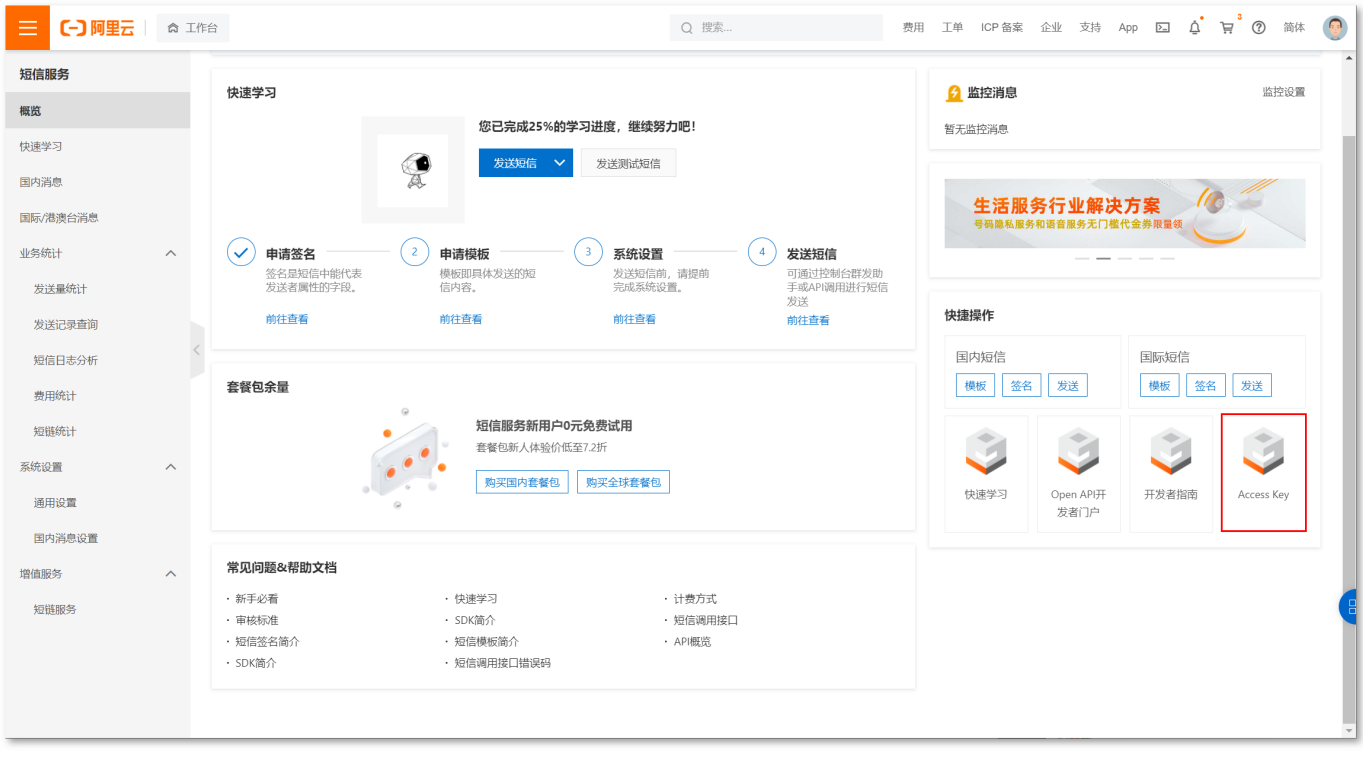




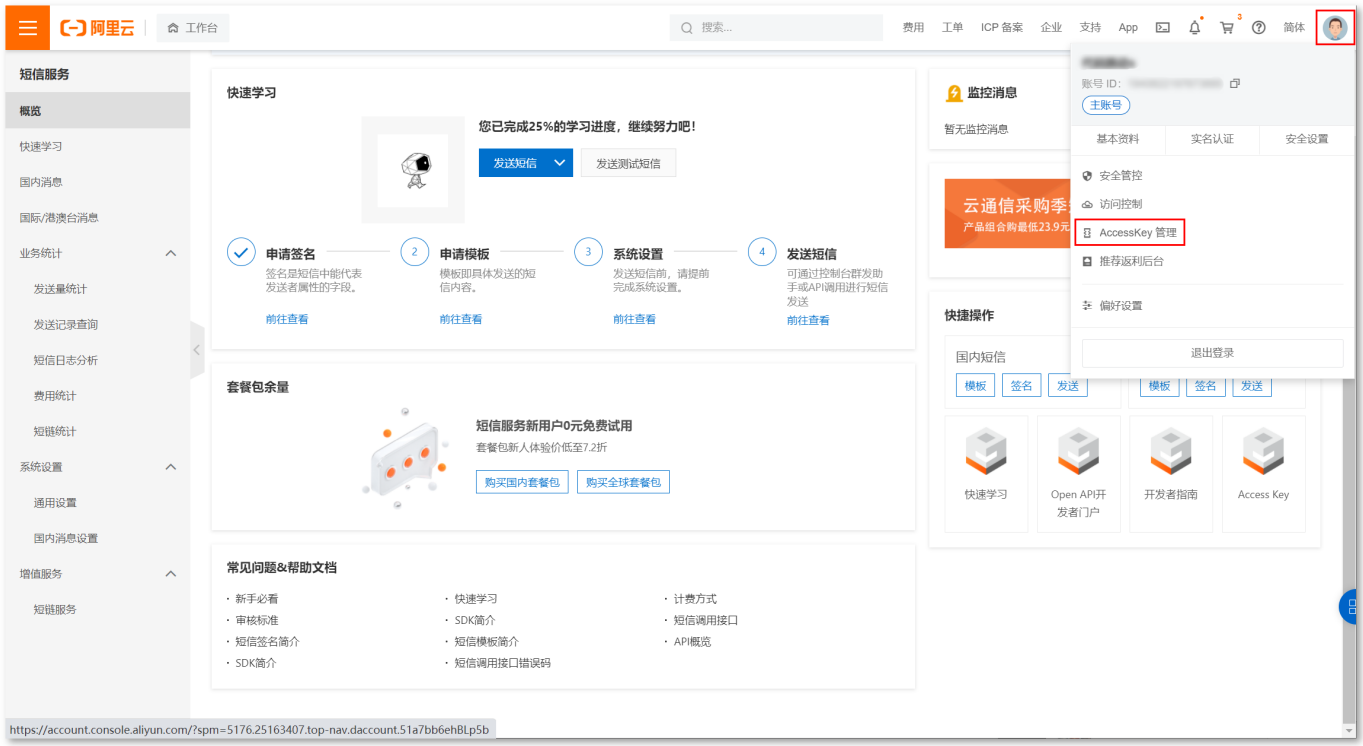
三、创建AccessKey

可以从两个入口创建 `AccessKey`

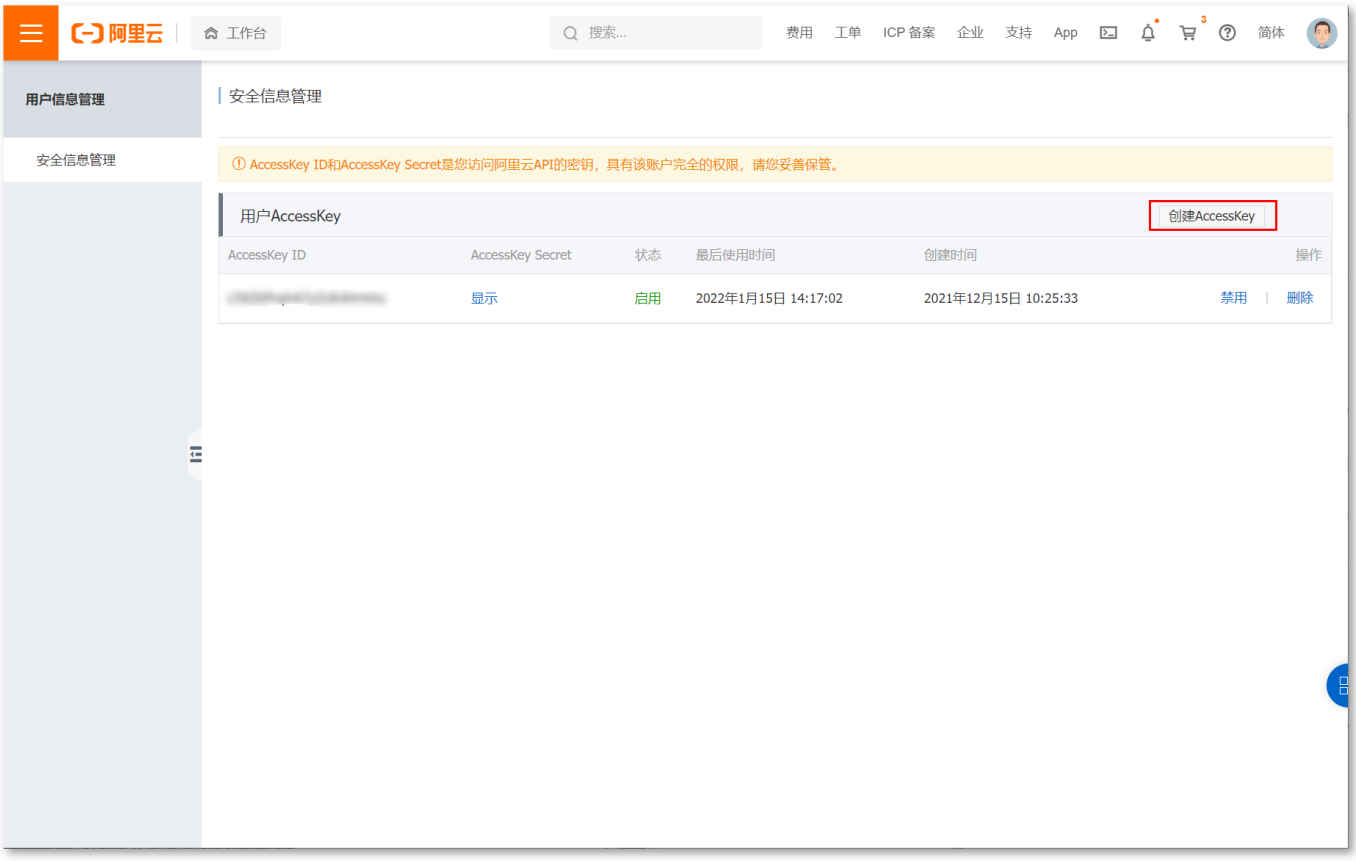
1、从短信服务概览页面



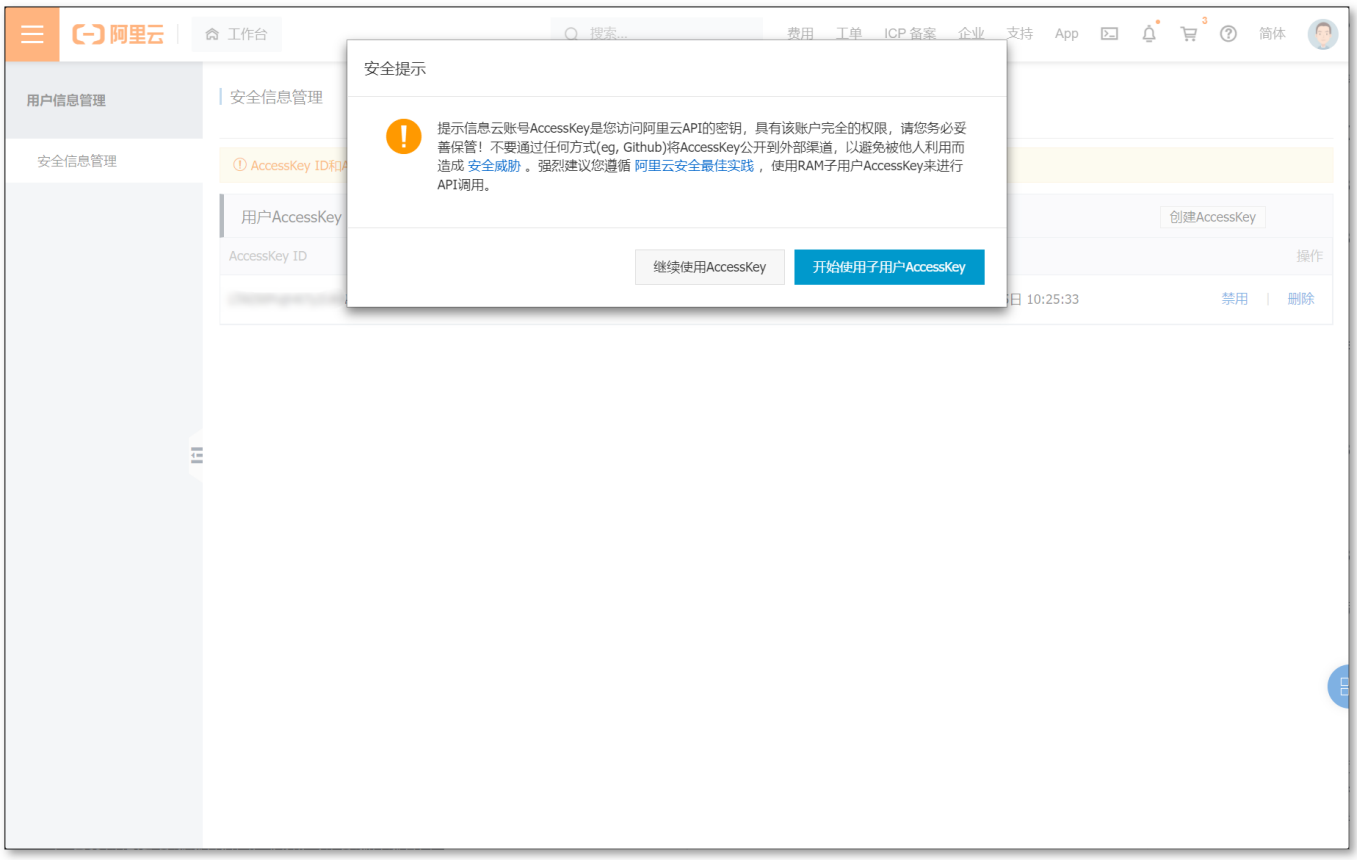
2、鼠标移到右上角头像，找到【AccessKey管理】



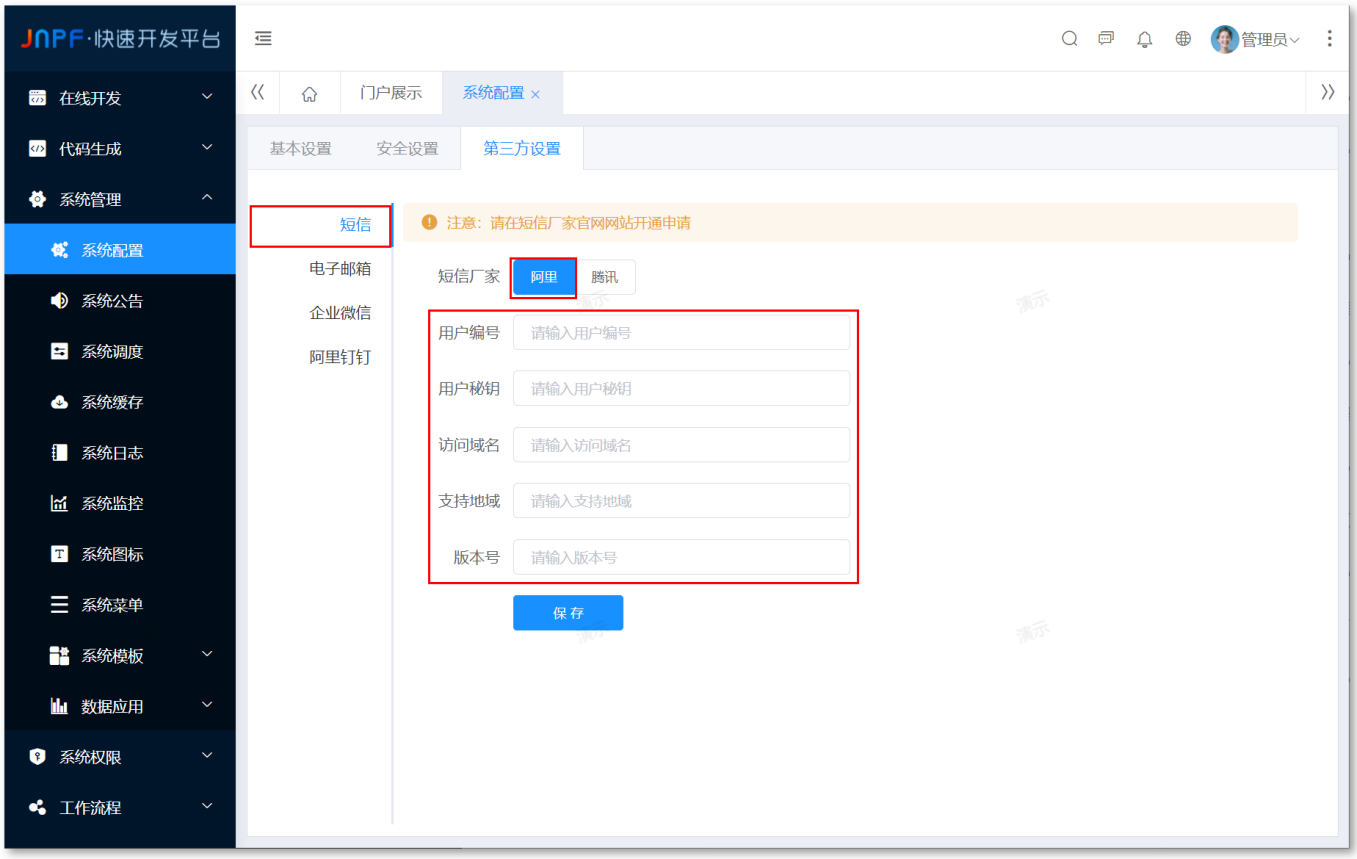
进入【AccessKey管理】界面后，点击右侧 `创建AccessKey`



> 如果出现以下提示，可以根据实际情况选择



四、页面配置



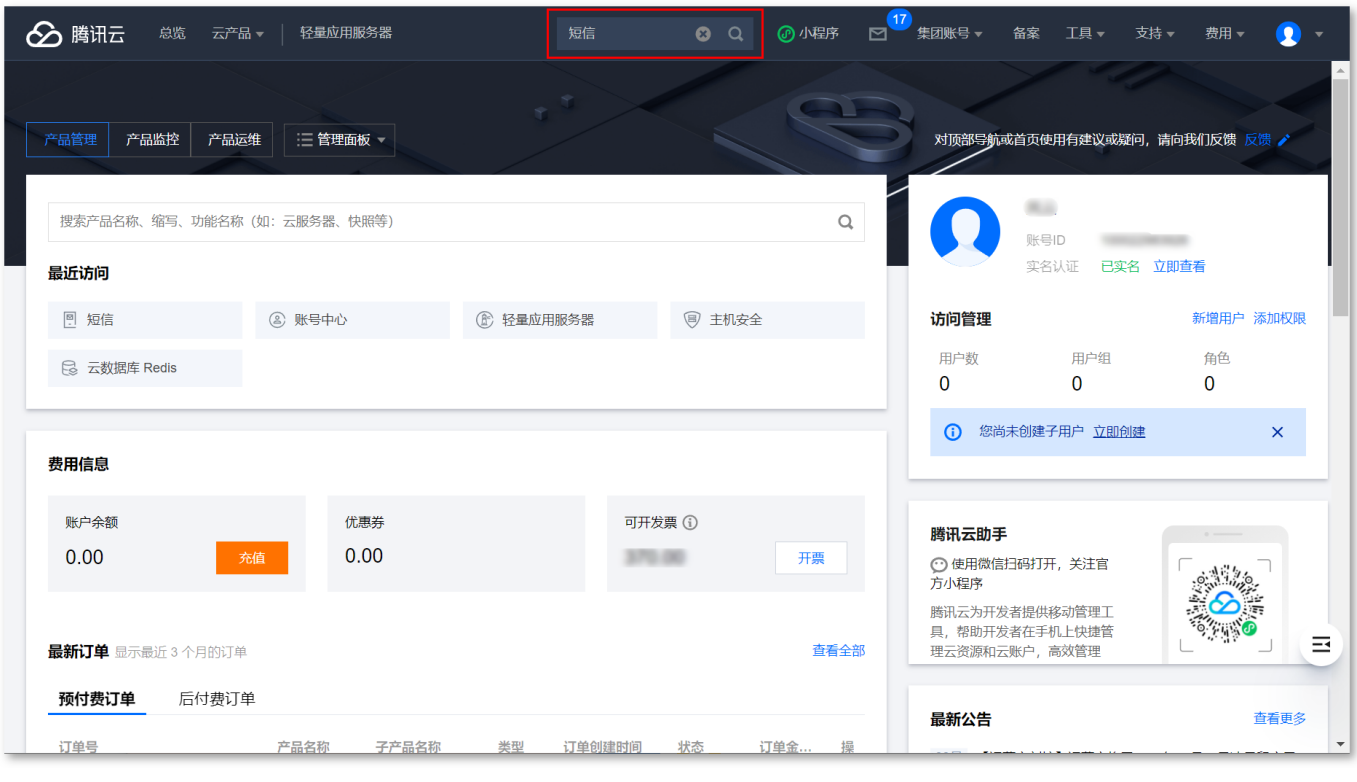
配置项	配置值

用户编号	【AccessKey管理】中的 `AccessKey ID`
用户秘钥	【AccessKey管理】中的 `AccessKey Secret`
访问域名	无需填写
支持地域	无需填写
版本号	无需填写

腾讯云短信配置

一、 登录腾讯云（https:///cloud.tencent.com)进入控制台首页

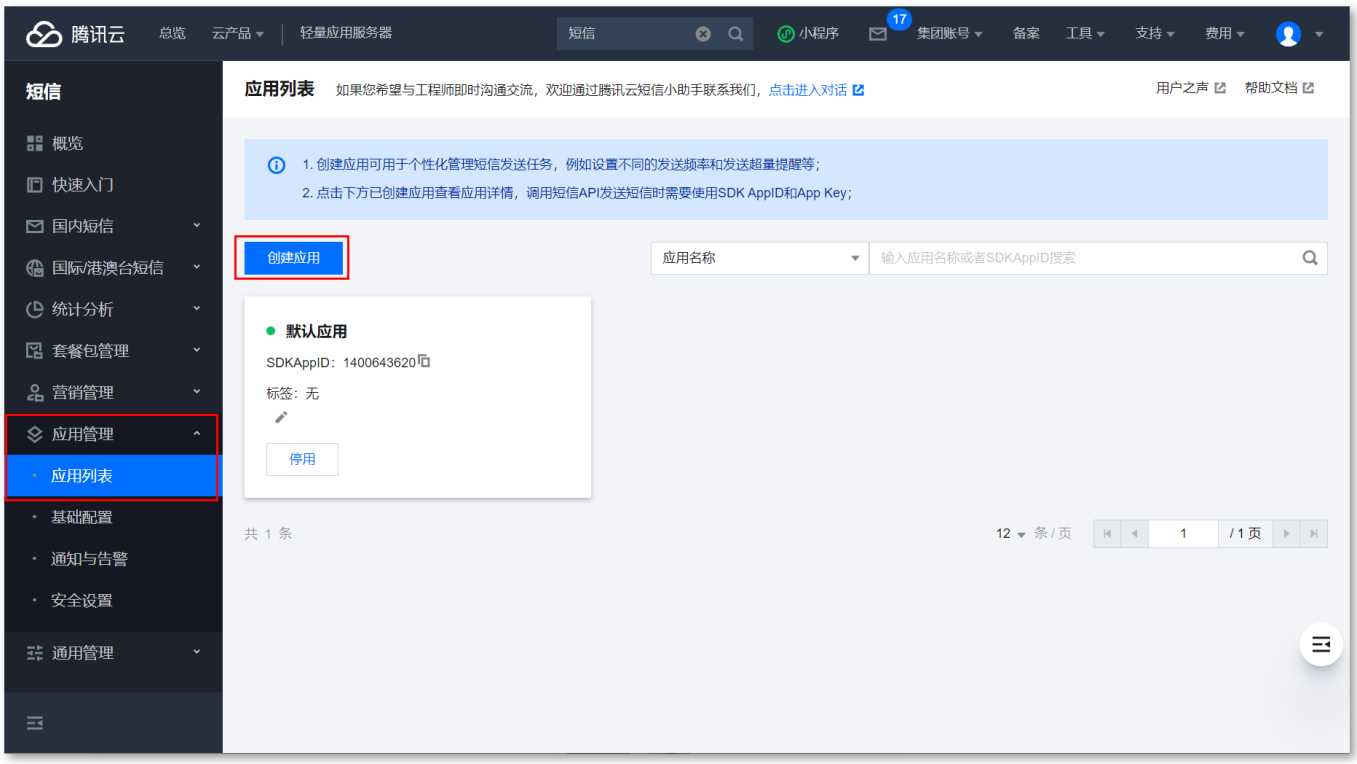
二、在控制台首页搜索【短信】，进入【腾讯短信服务】页面

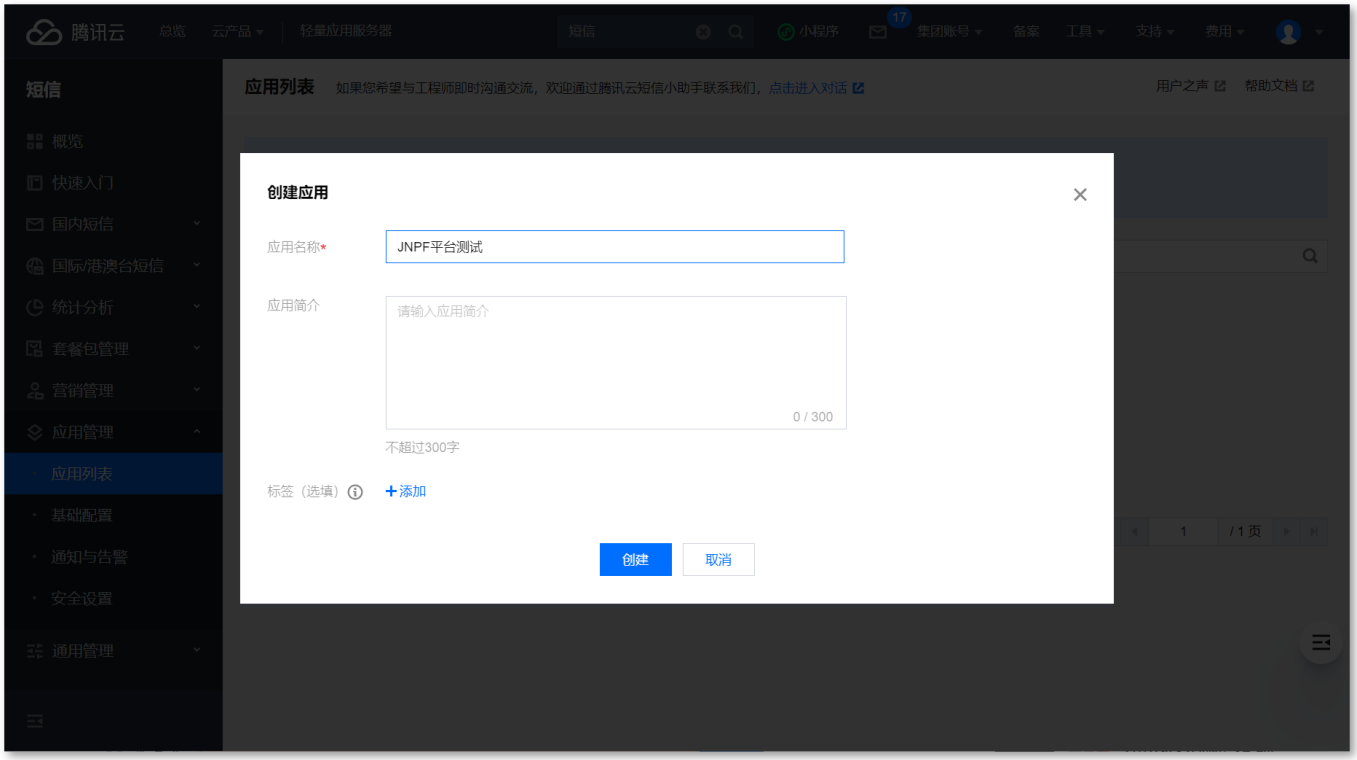


三、创建应用

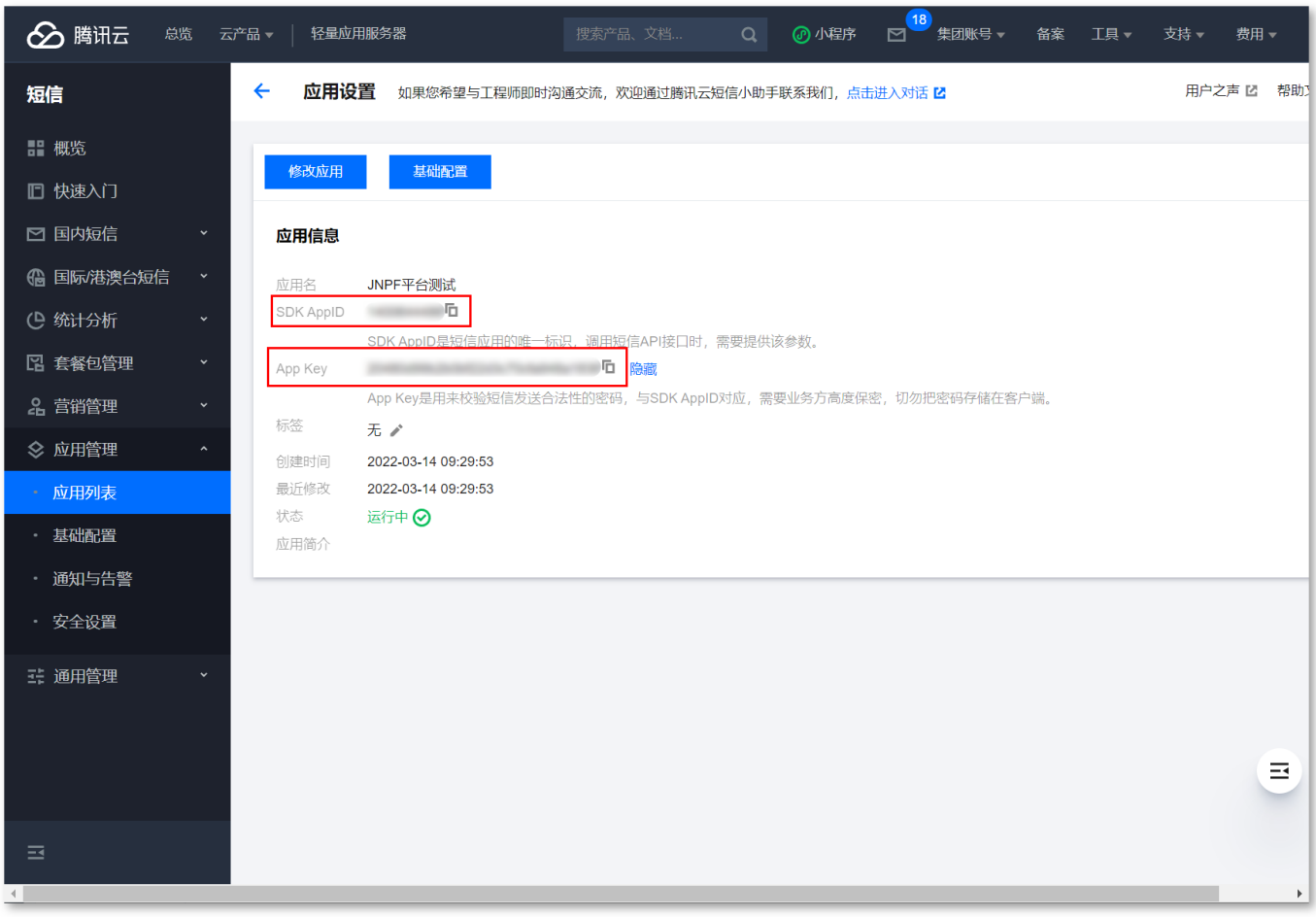
> 系统默认已创建一个应用，可以直接用，也可以重新创建

点击左侧菜单【应用管理】-【应用列表】页面的【创建应用】

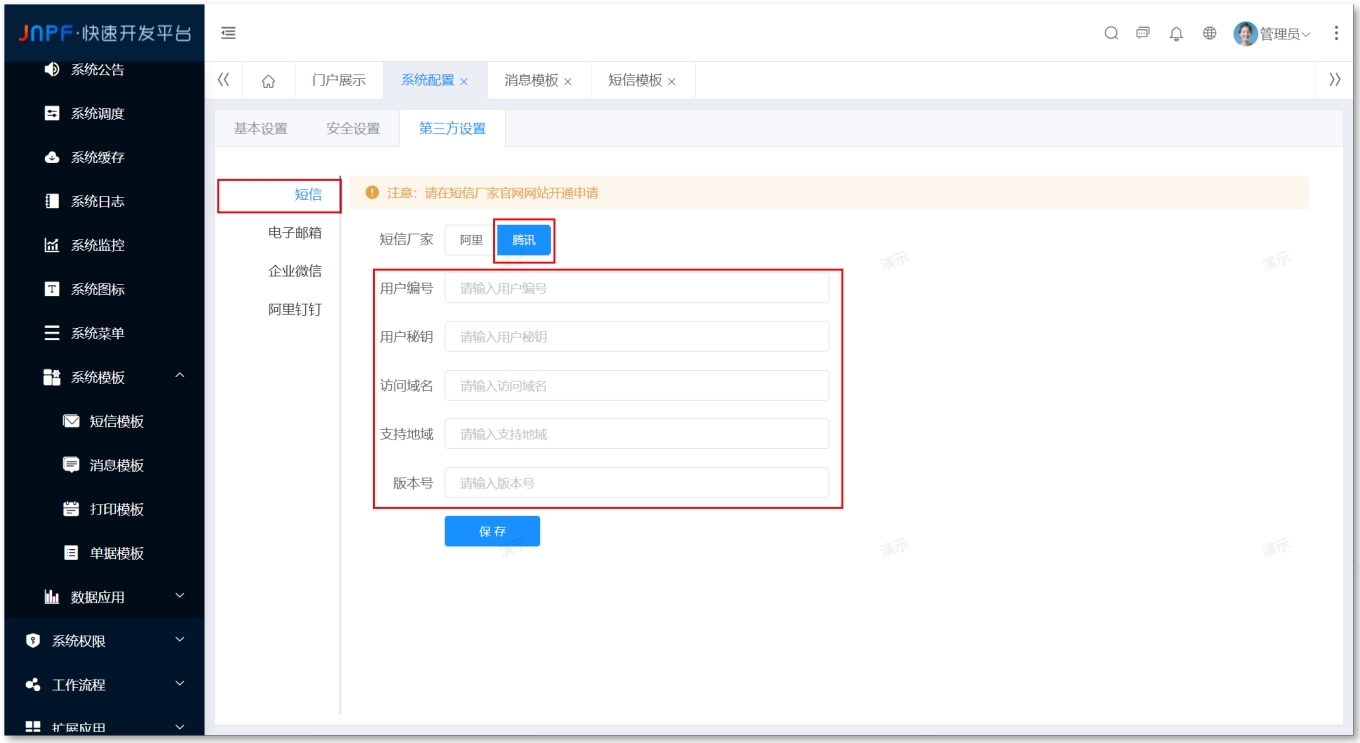




创建成功后，点击刚创建的应用查看详情，先记录下 `SDK AppID` 和 `App Key` 。



四、页面配置



配置项	配置值
用户编号	【应用管理】-【应用列表】应用中的 `SDK AppID`
用户秘钥	【应用管理】-【应用列表】应用中的 `App Key`
访问域名	可不填写，API 支持就近地域接入，近地域接入域名为 sms.tencentcloudapi.com ，也支持指定地域域名访问，例如广州地域的域名为 sms.ap-guangzhou.tencentcloudapi.com 。参考官方文档中的【接入地域】(https://cloud.tencent.com/document/api/382/52070)，注意：对时延敏感的业务，建议指定带地域的域名。
支持地域	取值如： `ap-beijing` （华北地区(北京)），地域列表：https://cloud.tencent.com/document/api/382/52071#.E5.9C.B0.E5.9F.9F.E5.88.97.E8.A1.A8
版本号	无需填写

电子邮箱

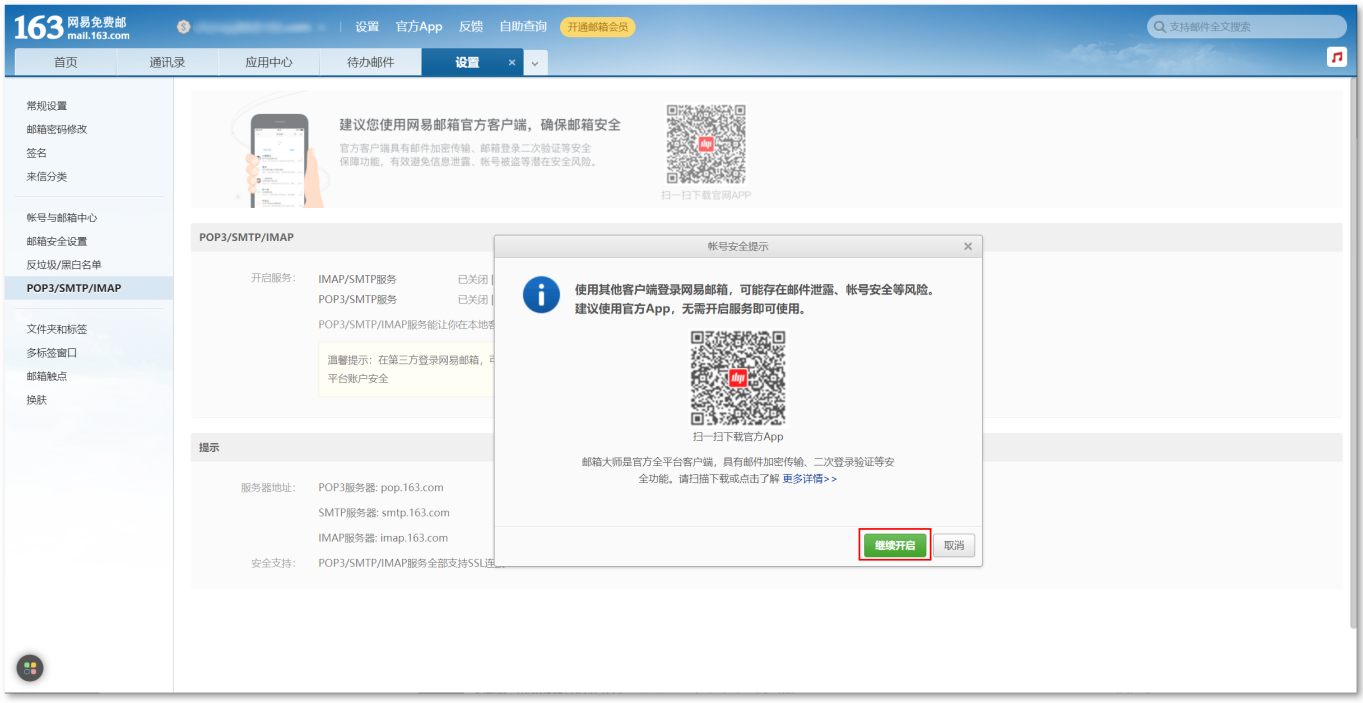
以163邮箱为例，操作步骤如下

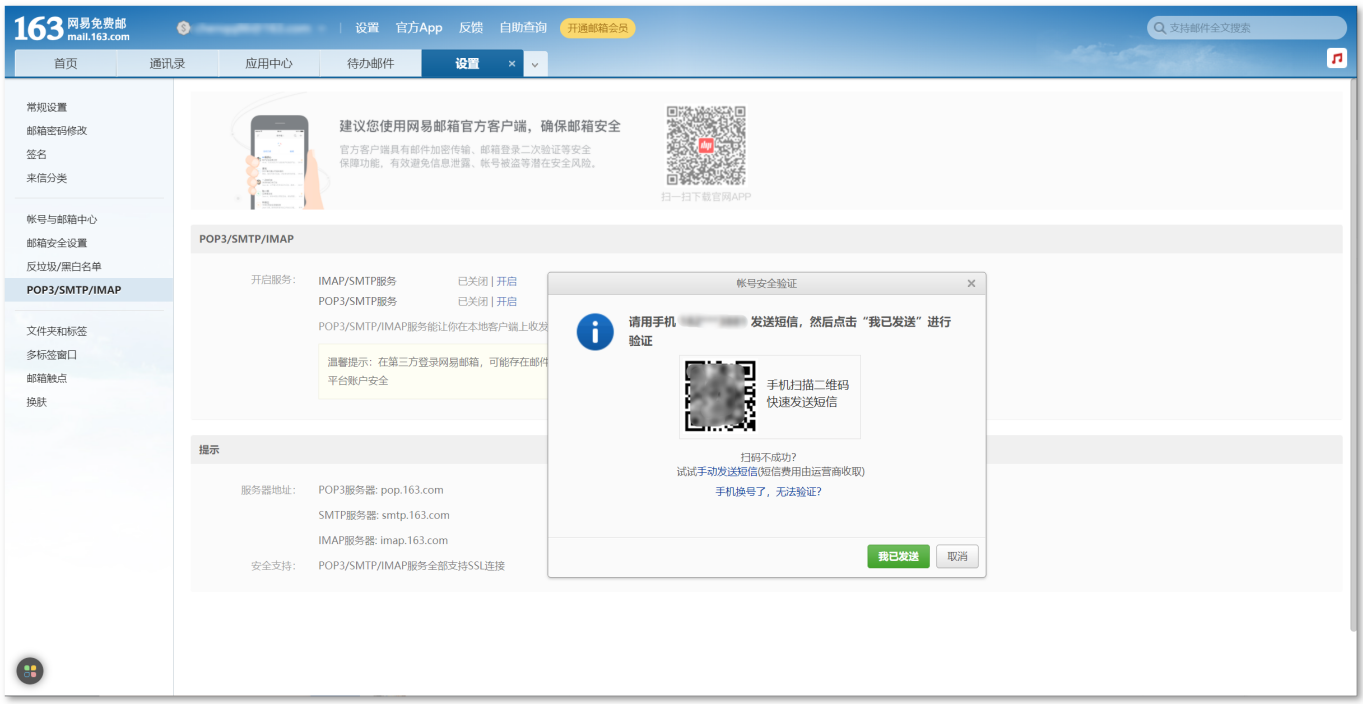
一、 登录网页版邮箱（https://email.163.com),进入邮箱首页。

二、点击上方设置，选择POP/SMTP/IMAP选项。

三、在客户端协议界面，选择`POP3/SMTP服务`后面的`开启`

四、在新弹出的弹窗中，点击继续开启，扫码页面您可以选择扫码发送短信，或者点击下方“手动发送短信”。（如果发送5分钟后系统依旧提示未收到短信，请联系移动运营商核实短信发送情况。）





五、点击 我已发送 后，如果系统检测到用户成功发送短信，则会提示您的客户端授权码（自动生成一串（16位字母组合）唯一随机授权密码），为了最大程度保证用户授权密码使用安全，一个授权码在开启后网页上只出现一次，但是一个授权码可以同时设置多个客户端。

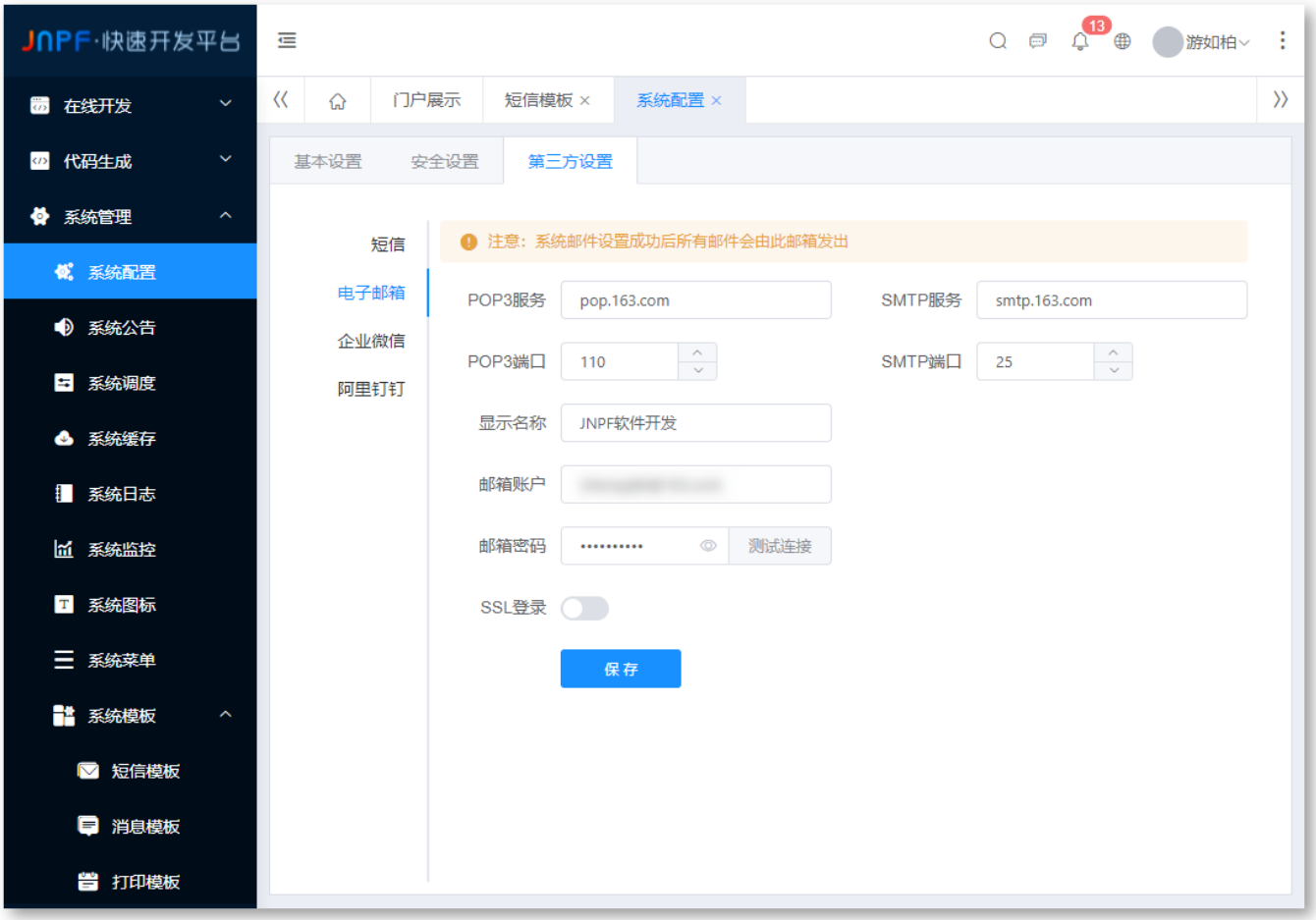
六、163邮箱服务器地址，信息如下：

> 其他邮箱服务商的服务器地址可以从服务商的帮助文档中获取

协议类型	协议功能	服务器地址	非SSL端口号	SSL端口号
SMTP	发送邮件	smtp.163.com	25	465
POP	接收邮件	pop.163.com	110	995
IMAP	接收邮件	imap.163.com	143	993

七、最终在页面上的配置如下：

> 这里的邮箱密码为获取到的 客户端授权码 ，保存前可以先点击 测试连接 查看是否正常



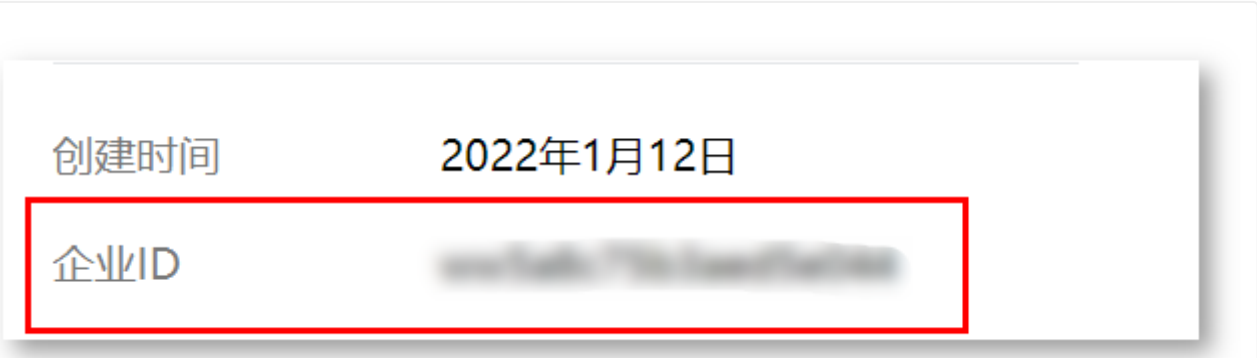
企业微信

一、 打开企业微信官网（https://work.weixin.qq.com），点击上方的【企业登录】扫码登录企业微信

二、获取企业ID

点击顶部导航【我的企业】

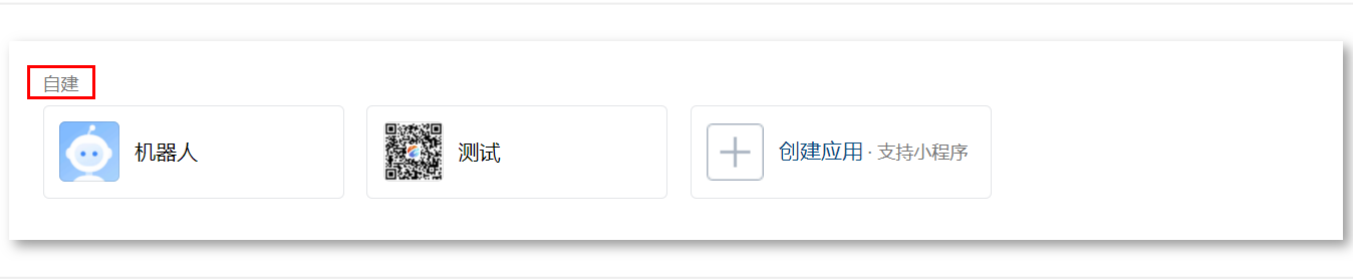
在【我的企业】页面拉到最下面获取到 `企业ID` 并记录下来



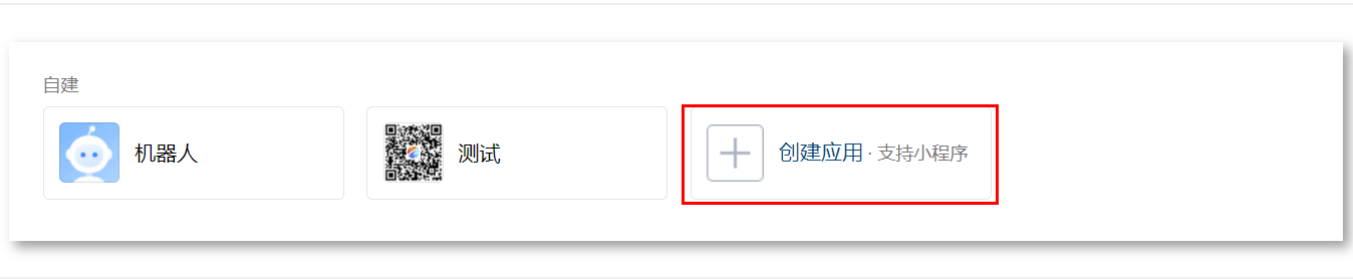
三、获取自建应用ID和密钥

点击顶部导航【应用管理】

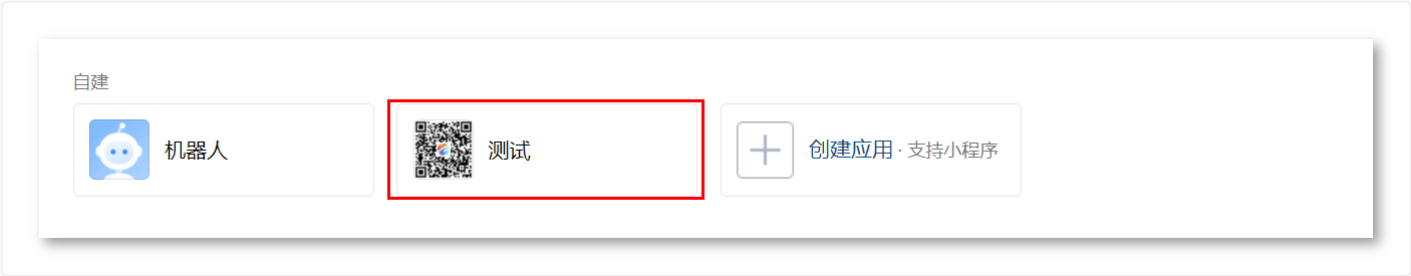
拉到最底找到【自建】



1、如果没有自建应用，先点击【创建应用】，按提示填写表单，`可见范围` 可设置此应用哪些员工可见，



2、如果已经有有自建应用，点击应用名称，



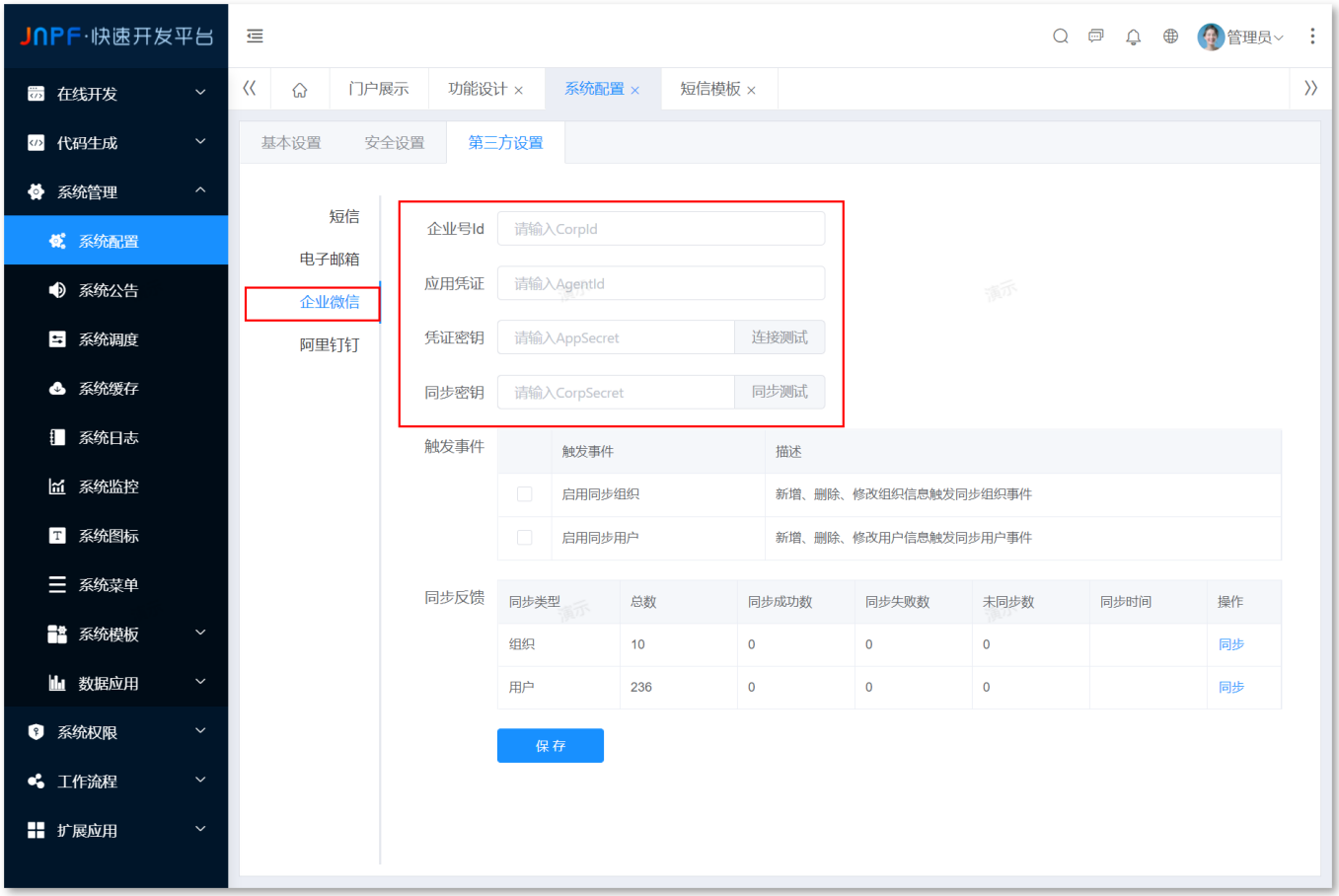
进入应用详情后，可查看应用详细信息及配置，这里我们需要把 `AgentId` 和 `Secret` 记录下来

四、获取通讯录同步密钥

点击顶部导航【管理工具】，在【管理工具】页面找到【通讯录同步】并点击进去

在【通讯录同步】页面可获取 `Secret` 同步密钥

五、页面配置



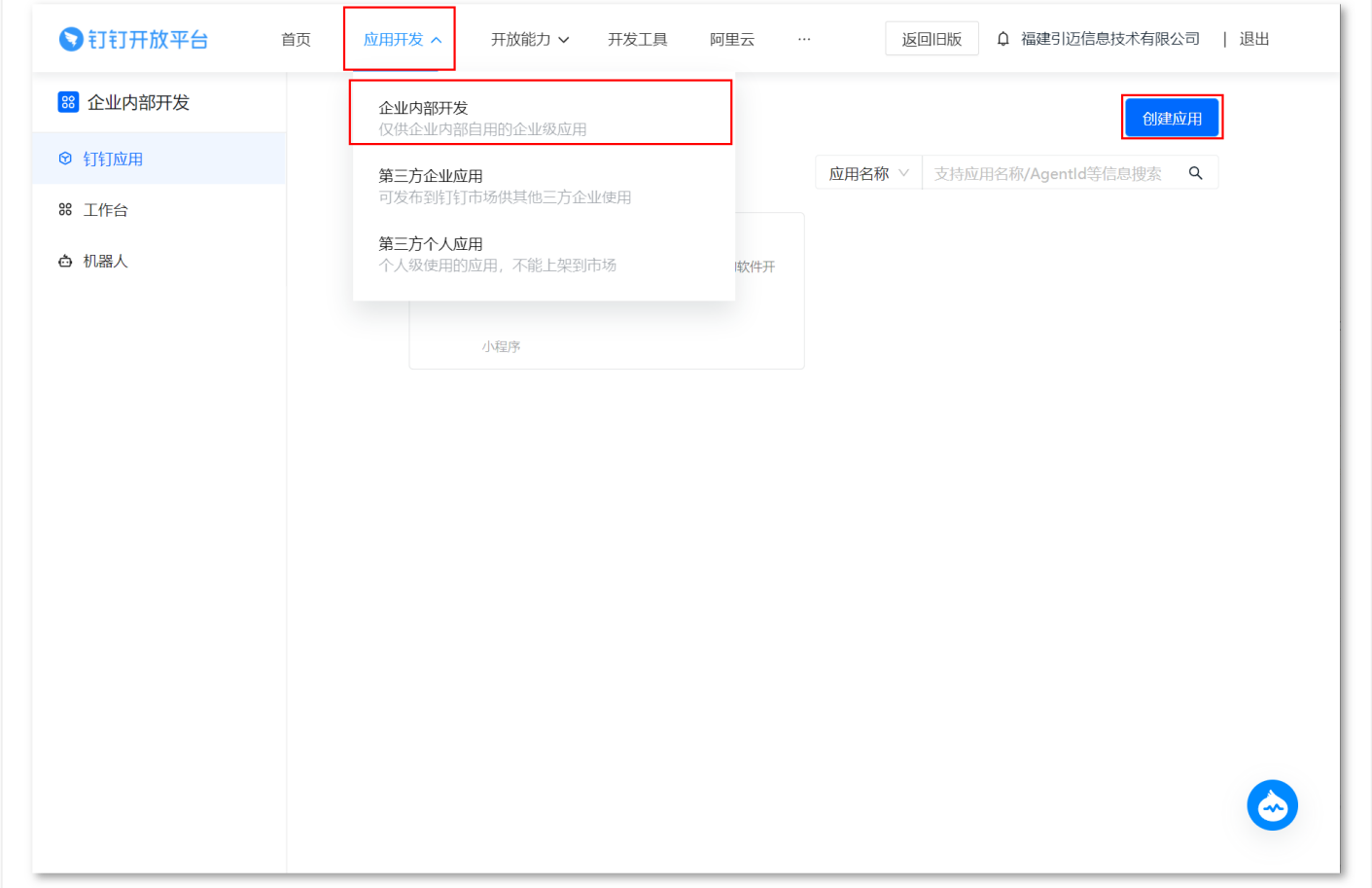
配置项	配置值
企业号Id	【我的企业】获取 `企业ID`
应用凭证	【应用管理】页面， `自建` 应用中获取 `AgentId`
凭证密钥	【应用管理】页面， `自建` 应用中获取 `Secret`
同步密钥	【应用工具】【通讯录同步】页面中获取 `Secret`
触发事件	可配置 `启用同步组织` 和 `启用同步用户` 的触发事件
同步反馈	统计同步信息

阿里钉钉

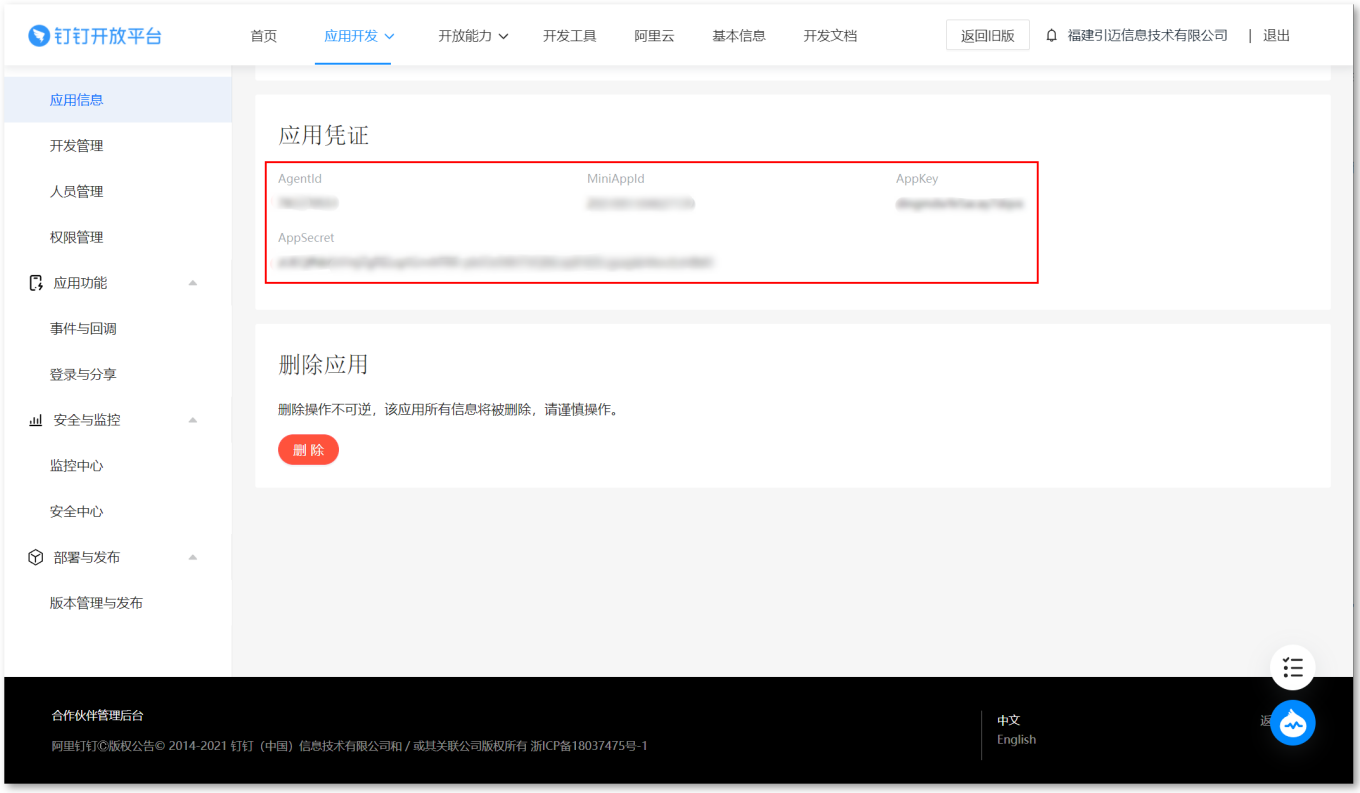
一、 登录钉钉开放平台（https://open-dev.dingtalk.com)控制台

二、获取AgentId、AppKey、AppSecret

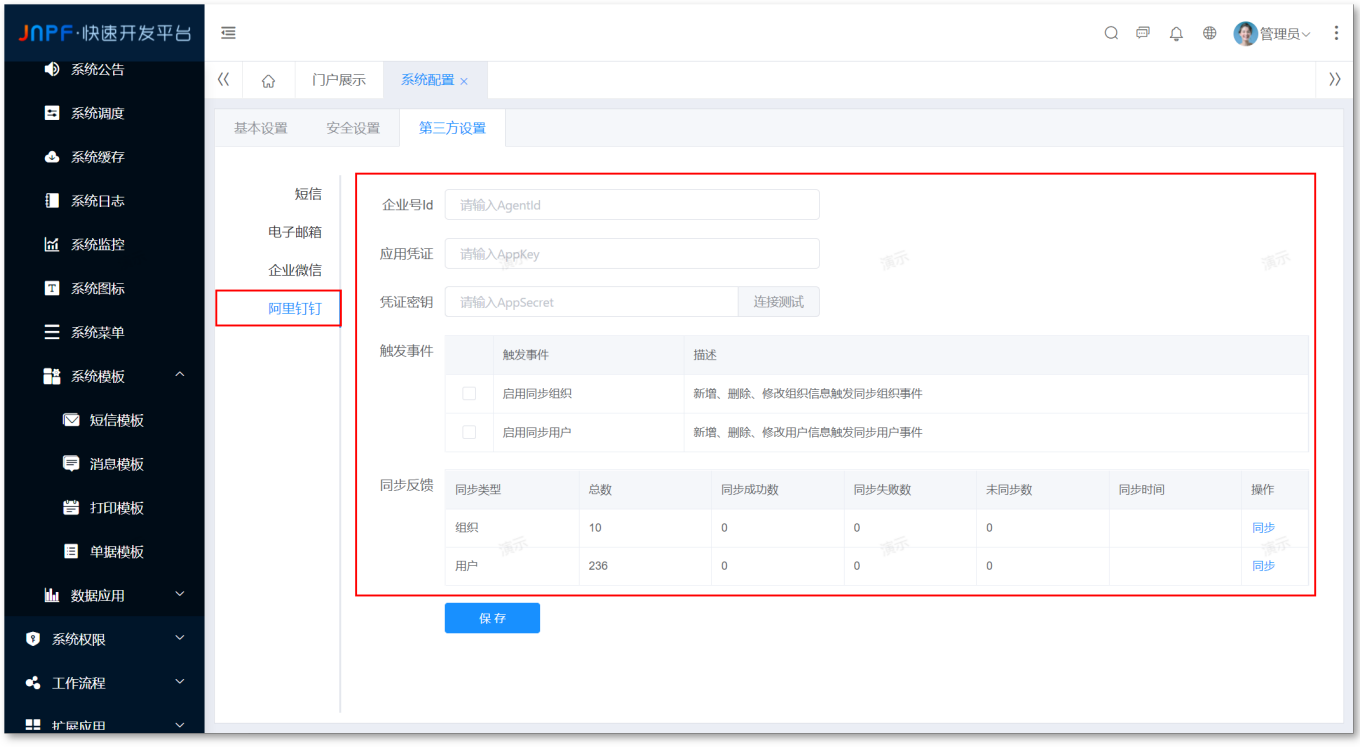
顶部导航【应用开发】-【企业内部开发】，点击【创建应用】



按提示规范填写表单后获取到如下信息



三、页面设置



配置项	配置值
企业号Id	【应用开发】-【企业内部开发】应用详情中的 `AgentId`
应用凭证	【应用开发】-【企业内部开发】应用详情中的 `AppKey`

凭证密钥	【应用开发】- 【企业内部开发】应用详情中的 `AppSecret`
触发事件	可配置 启用同步组织 和 启用同步用户 的触发事件
同步反馈	统计同步信息

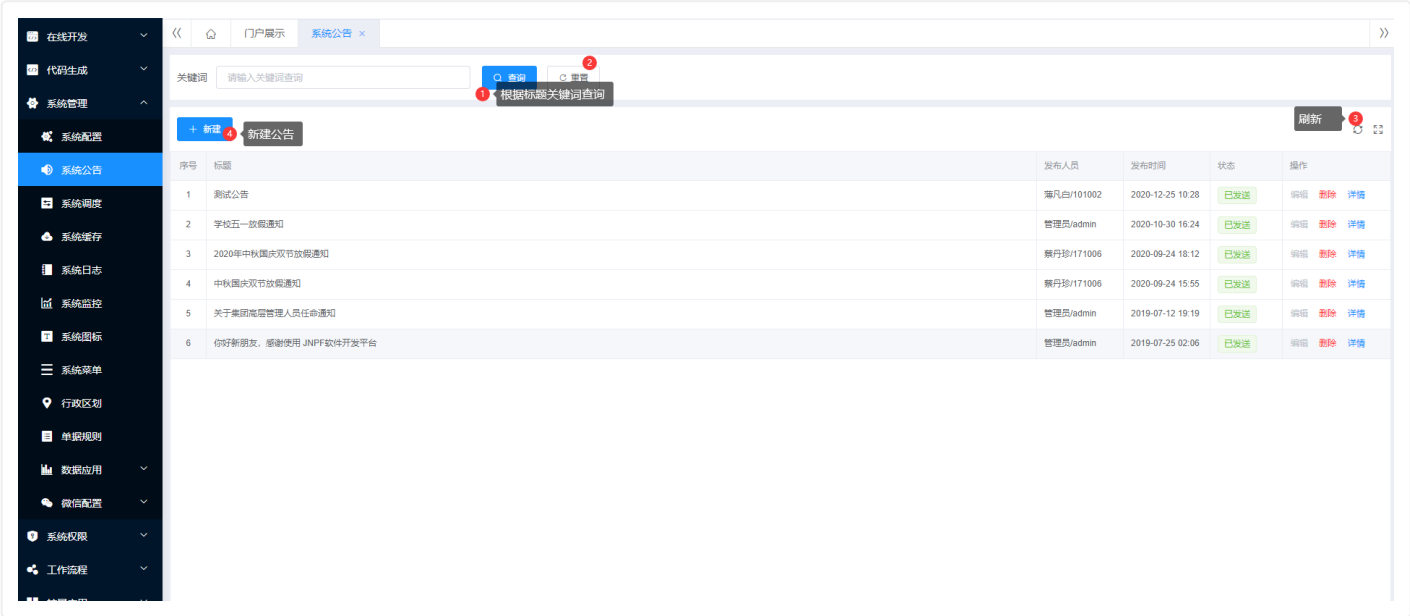
系统公告

功能描述

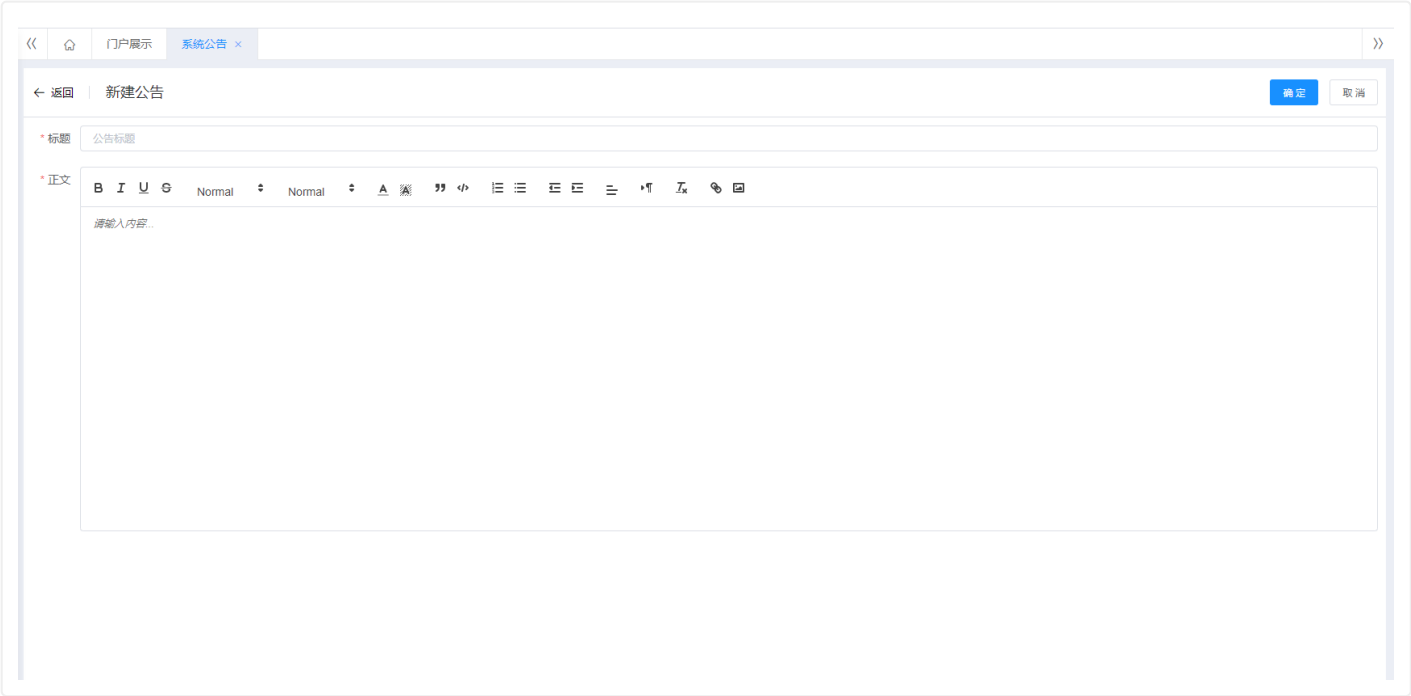
系统公告是发布一条公告或者通知，可及时传达到企业每个人手中，避免时间和地点距离消息传达延误。

操作步骤

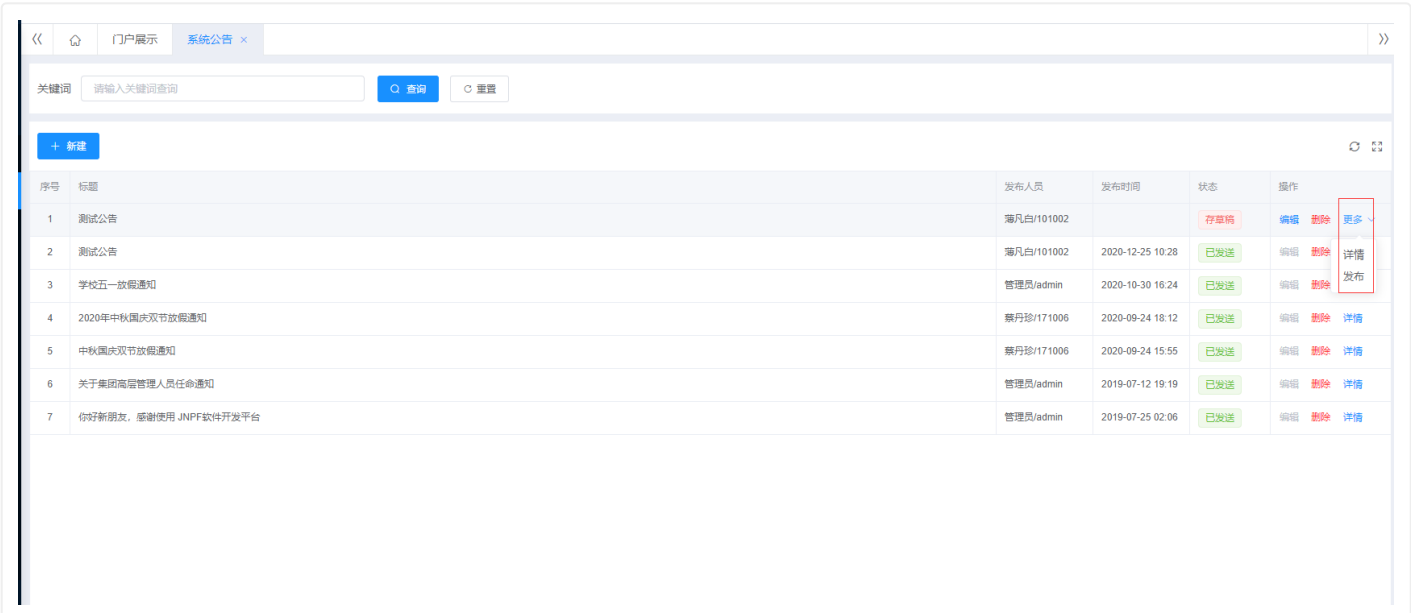
系统管理目录下操作在系统公告，进入【系统公告页】面查询、重置、刷新、全屏、新建公告功能；如下图所示：



在系统公告页面左上角点击新建按钮、进入创建公告页面，请输入标题、正文，正文上面对发送正文内容可以进行简单的格式调整，对文字加粗、倾斜、文字添加颜色、表格等等。如下图所示：



在系统公告页面对一条信息有编辑、删除、更多（详情、发布）功能，如下图所示：

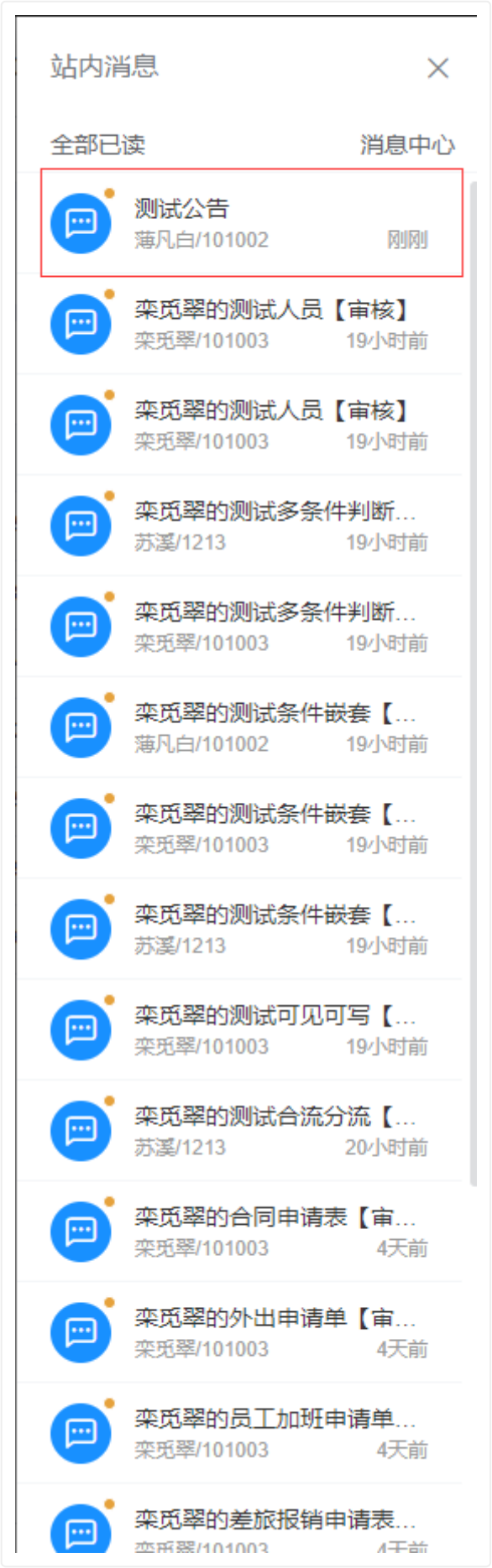


点击编辑，进入编辑页面、修改需要修改的内容，点击确认按钮保存该条数据或者点击取消按钮离开该页面。

点击删除提示“此操作将永久删除该数据，是否继续？”，如果是点击确认按钮删除此条数据、或者取消关闭该页面这条数据依然存在。

点击详情进入查看公告页面显示创建人员、创建日期以及内容，需要离开公告页面操作关闭按钮。如下图所示：

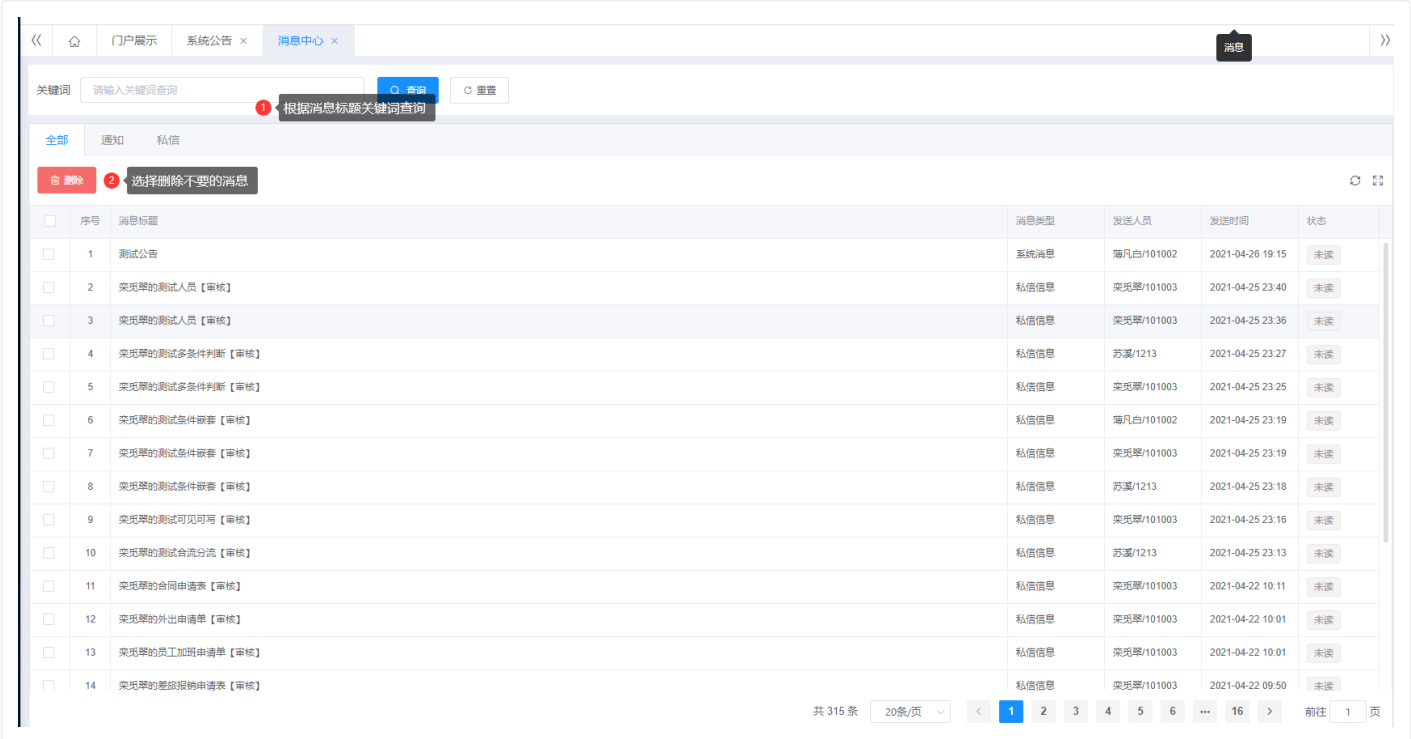
系统公告信息填写完、确认无误后操作发布按钮，当前这条数据状态显示发送、同时也推送给其他账号站内消息中，在站内消息中存储所有人员发布的系统公告信息；如下图所示：



在站内消息列表点击“全部已读”和“消息中心”功能显示；如下图所示：

当站内消息页面操作全部消息已读时提示“您确定全部为已读状态、是否继续？”，操作确定按钮时在站内消息全部显示已读状态、或者操作取消按钮站内消息保持原样（有新消息标志和消息读过标志）。

在站内消息页面操作消息中心功能显示有查询功能、可以对一条数据进行删除、查看当条信息详细；如下图所示：



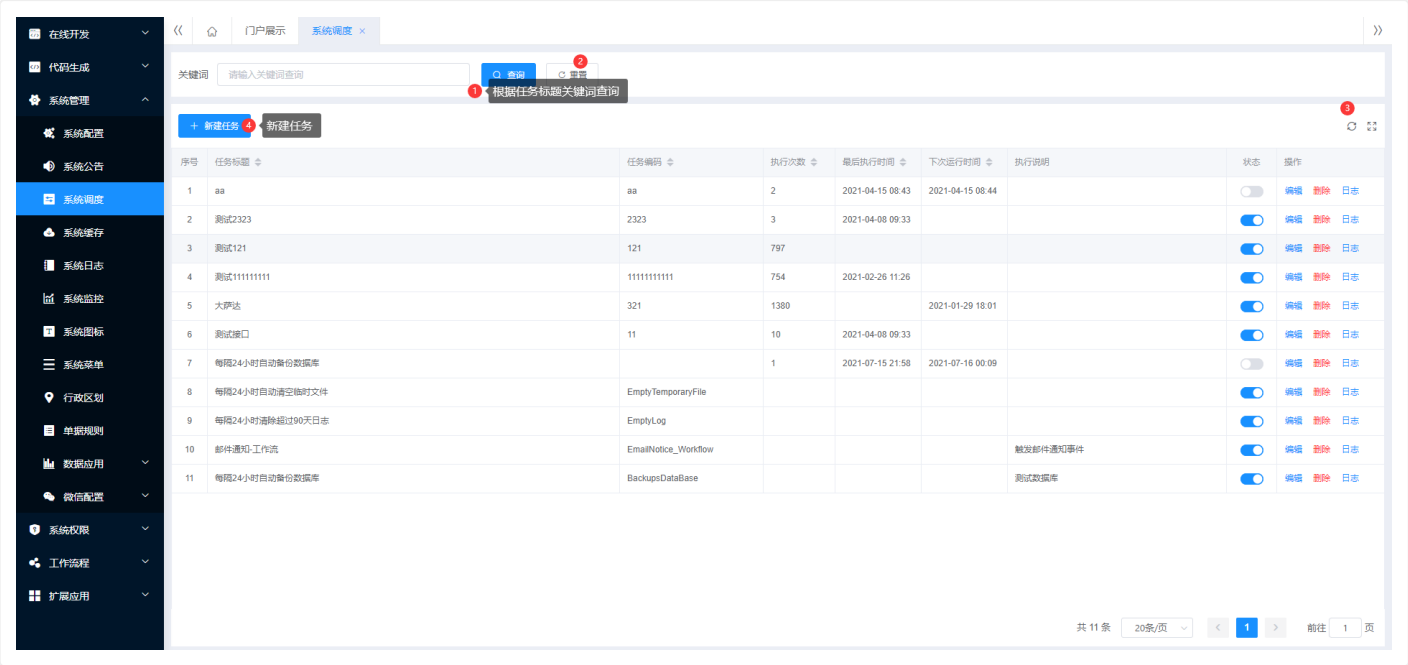
系统调度

功能描述

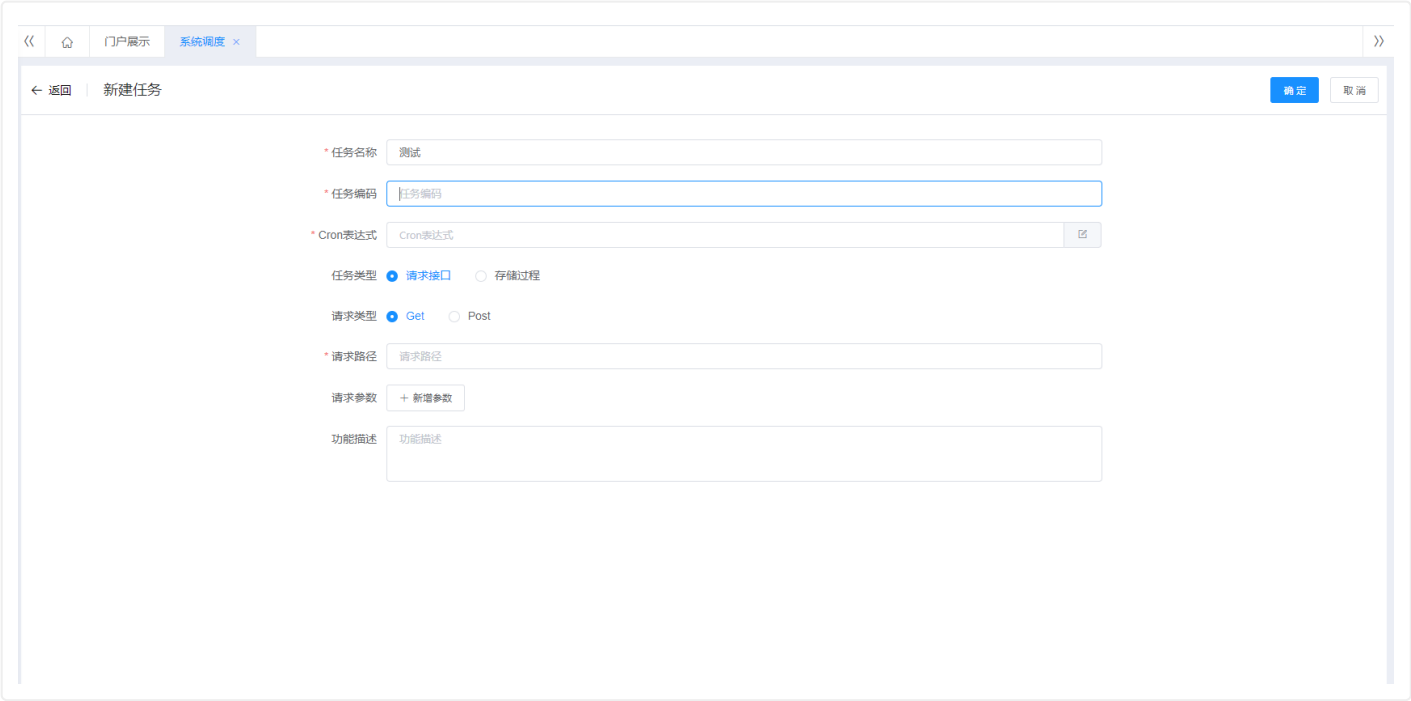
基于给定时间点，给定时间间隔或者给定执行次数自动执行任务，任务管理界面，任务对象集中管理，操作人性化。

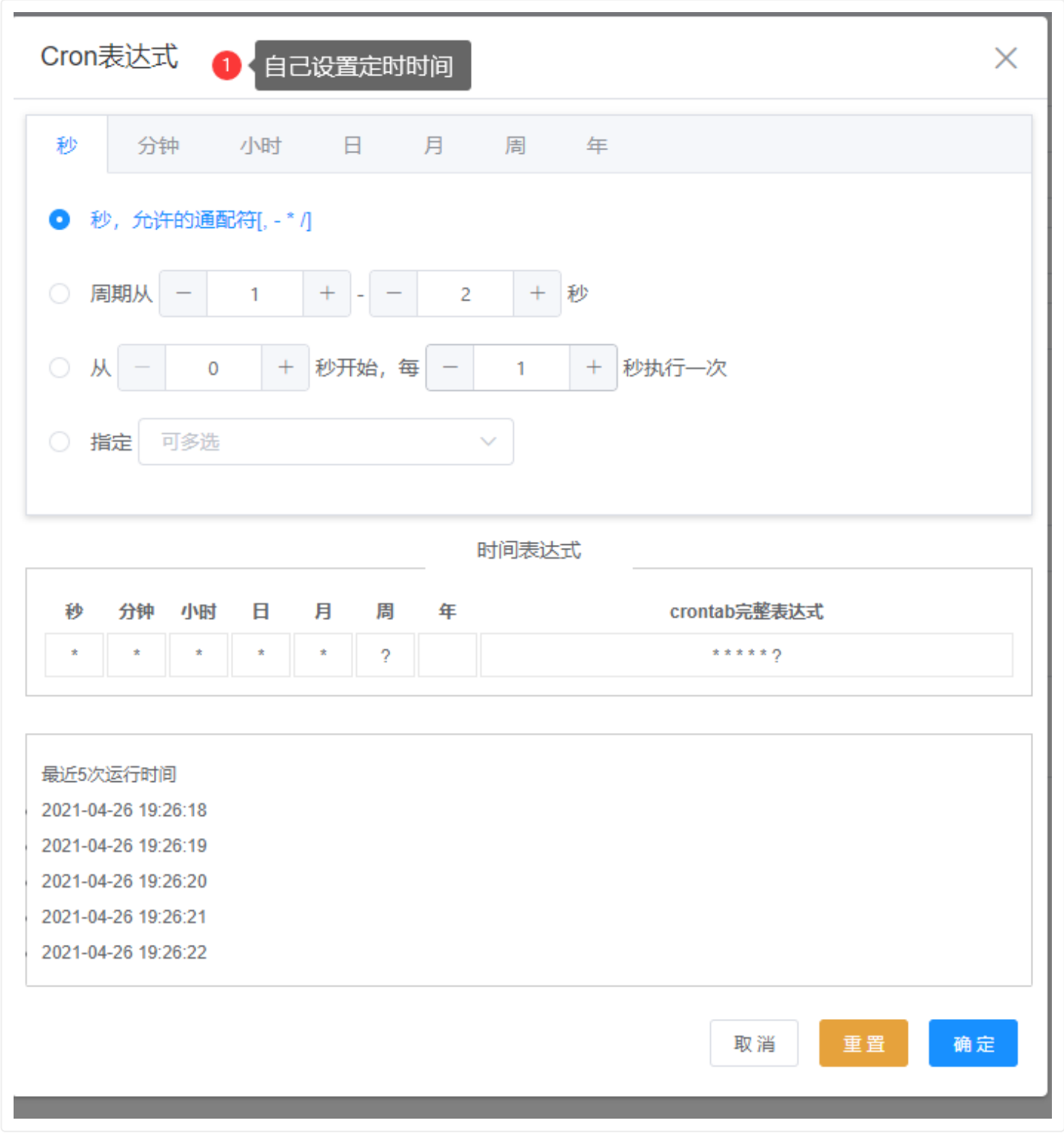
操作步骤

系统管理目录下操作系统调度，进入【系统调度】页面有查询、重置、刷新、新建任务等功能操作；如下图所示：



在系统调度页面新建任务，在新建任务页面红色星号表示必填项，信息填写完点击确定按钮当前数据保存成功或者取消按钮离开该页面、当前数据未保存；如下图所示：


门户展示
系统调度
返回
新建任务
确定
取消
* 任务名称
测试
* 任务编码
任务编码
* Cron表达式
Cron表达式
任务类型
请求接口
存储过程
请求类型
Get
Post
* 请求路径
请求路径
请求参数
+ 新增参数
功能描述
功能描述

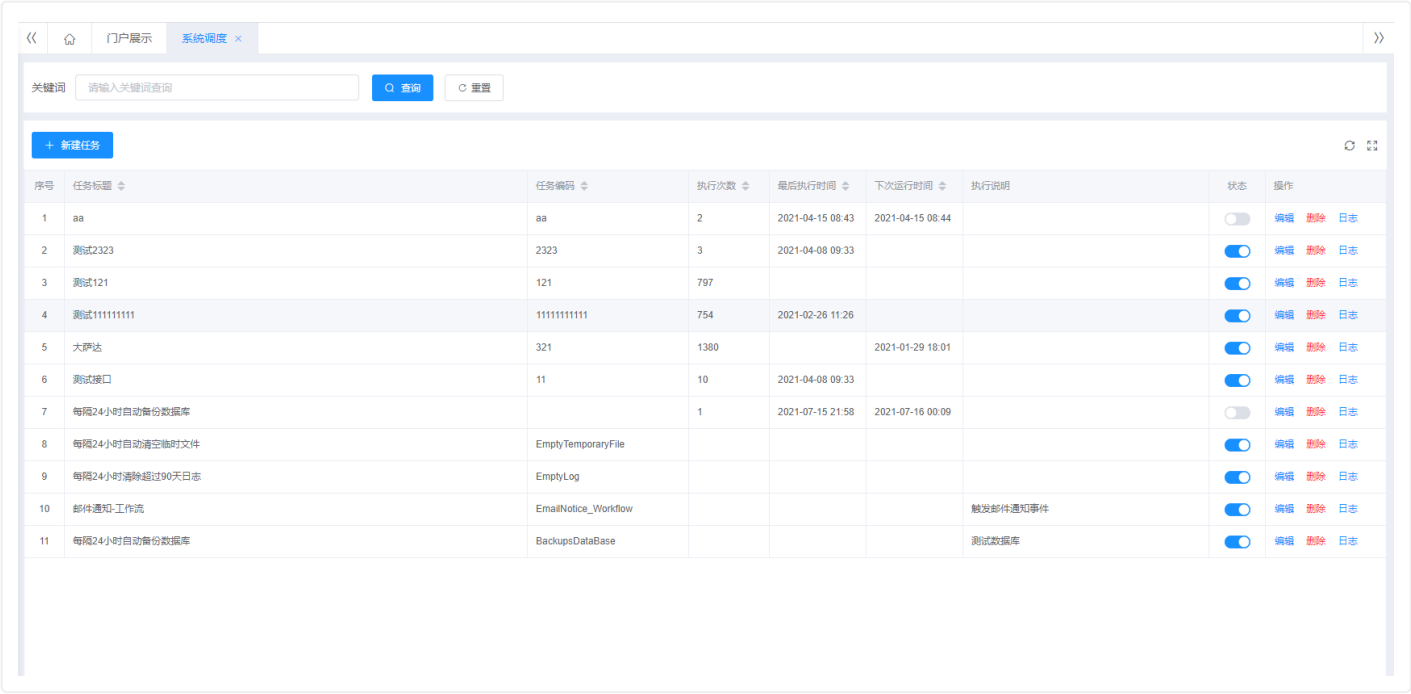


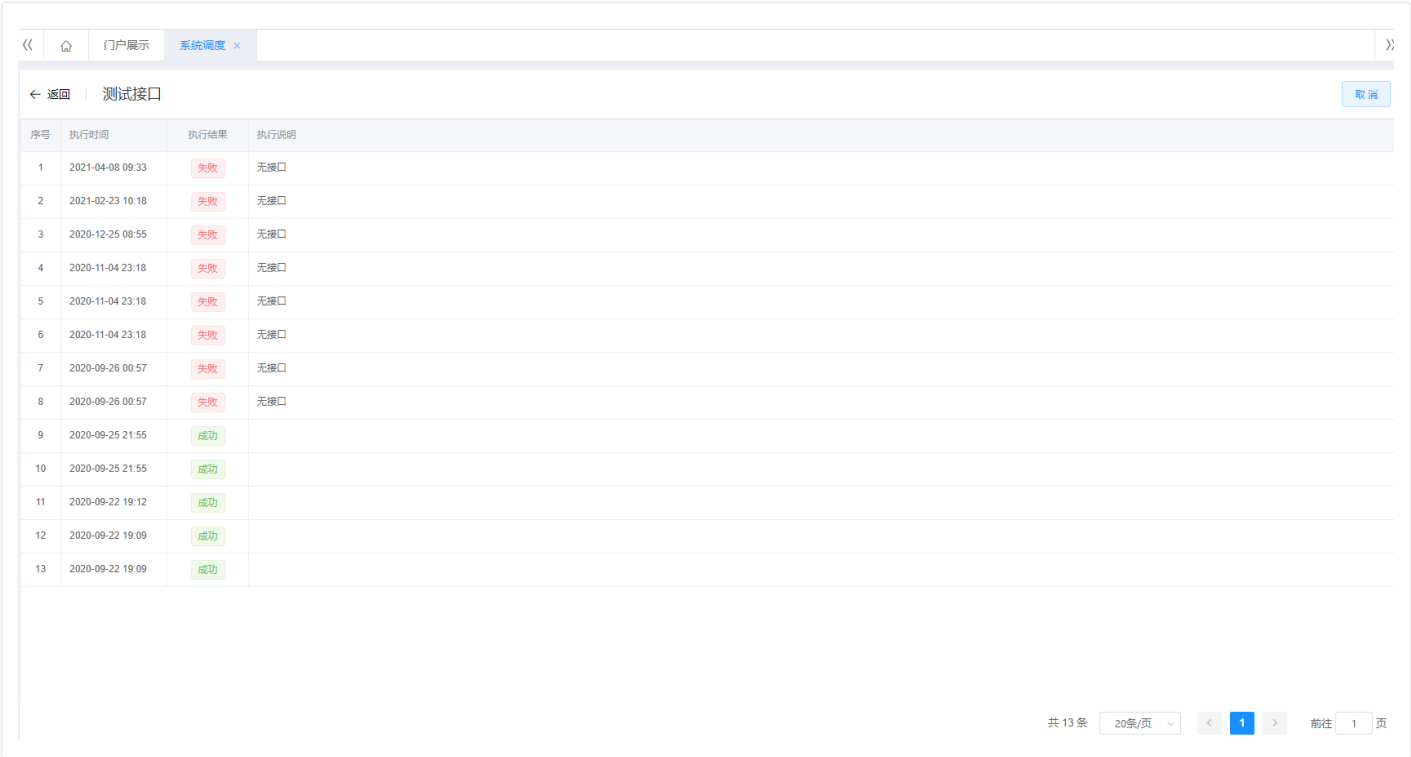
在任务调度页面对一条数据有编辑、删除、日志、点击执行状态（停止、启用功能）；如下图所示：

ⅰ 对一条数据操作执行状态显示灰色按钮，当前这条数据停止状态是不可以使用；

ⅱ 选择在执行状态显示灰色按钮数据、点击操作启动，在执行状态显示蓝色按钮就正常；

ⅲ 点击日志，可以查看执行的日志列表





系统缓存

功能描述

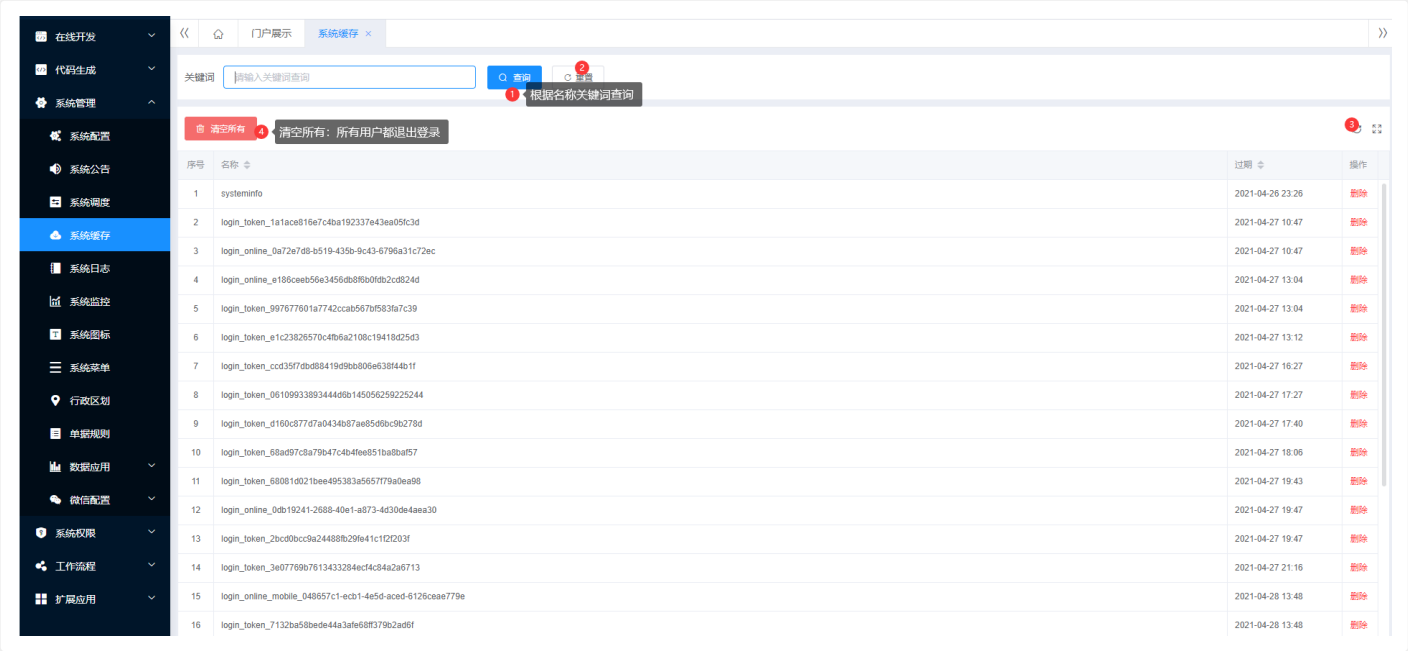
实现系统缓存数据管理，可以通过平台更直观查看所有缓存数据。

操作步骤

系统管理目录下操作系统缓存，进入【系统缓存】页面有查询、重置、刷新、清空所有功能；如下图所示：

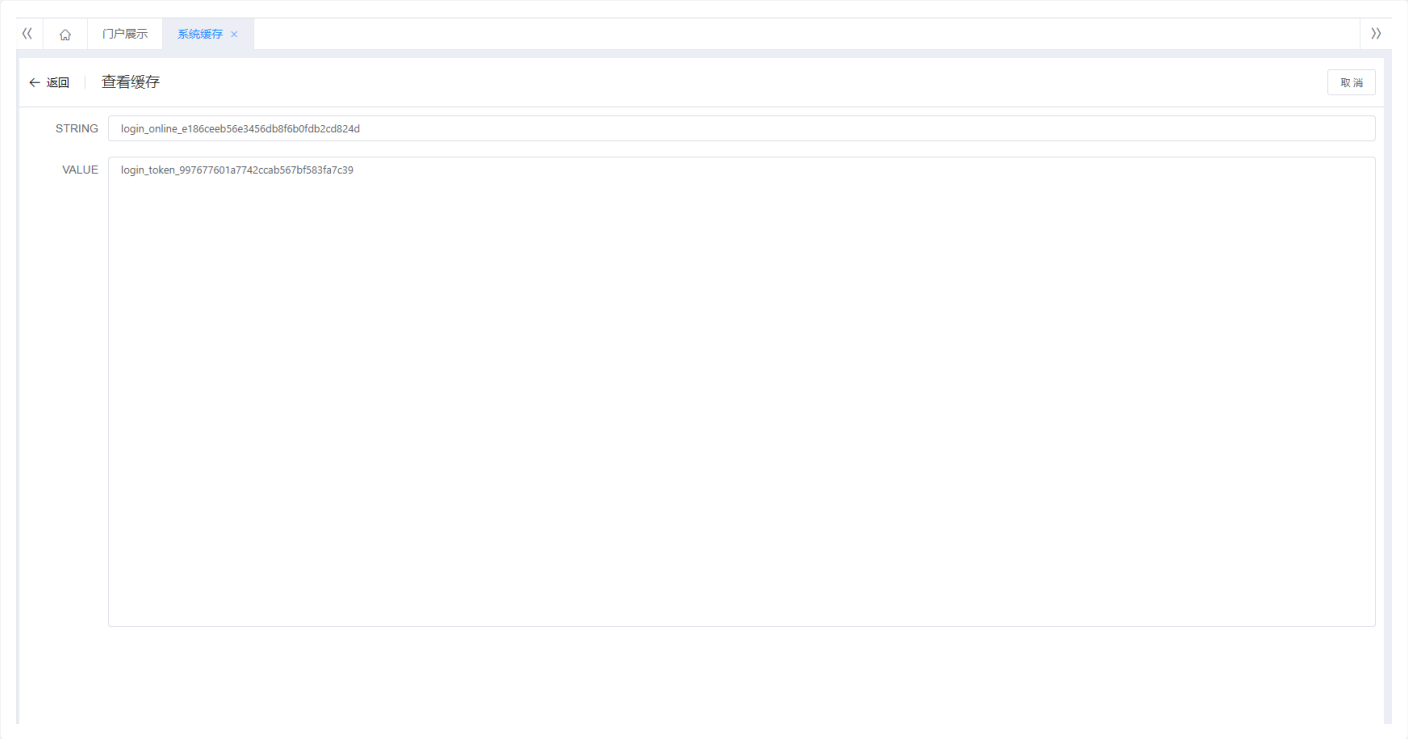
ⅰ 在缓存管理页面操作需谨慎清空所有按钮时、自动会清空所有在平台中登录的数据，导致其它登录账户自动退出、需要再次登录才能使用。

ⅱ 在缓存管理页面可以对一条数据操作删除（红色字）、表示删除当前该条数据；



在缓存管理页面选择名称栏目点击一下表示查看缓存信息详细；

在查看缓存页面点击左上角的取消按钮表示关闭当前页面跳转到缓存管理页面；如下图所示：



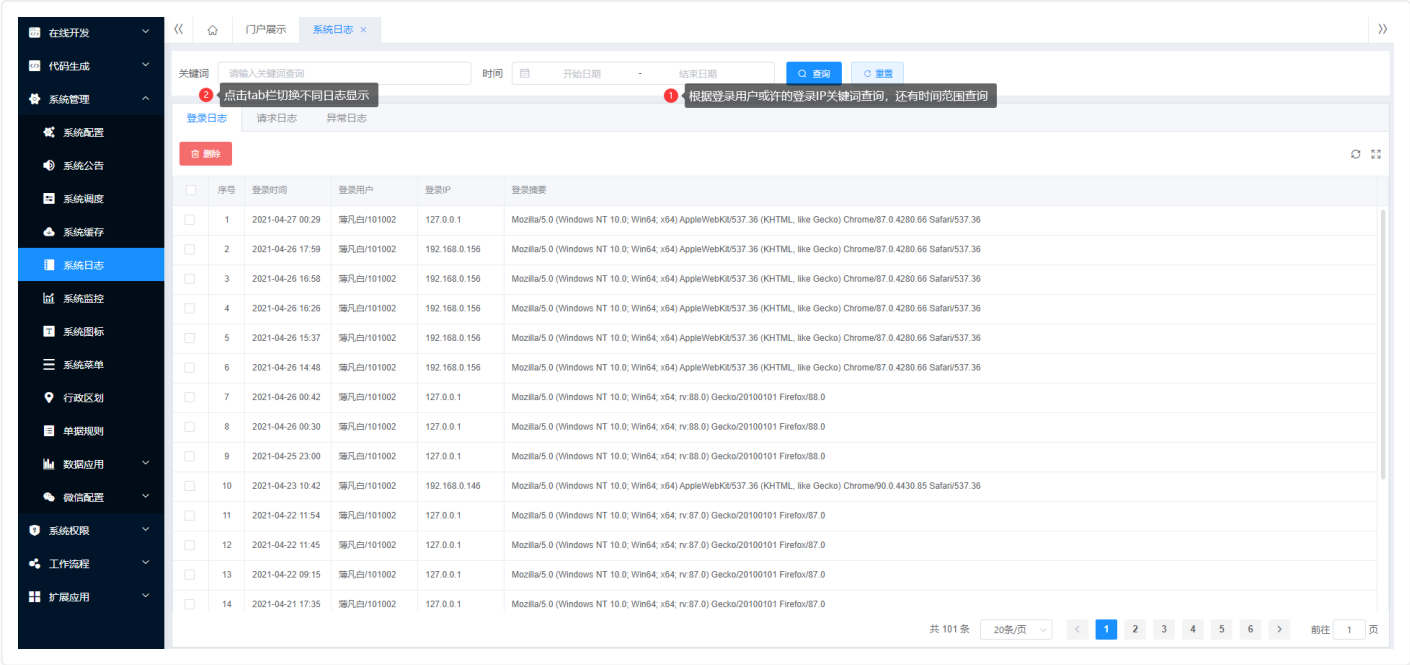
系统日志

功能描述

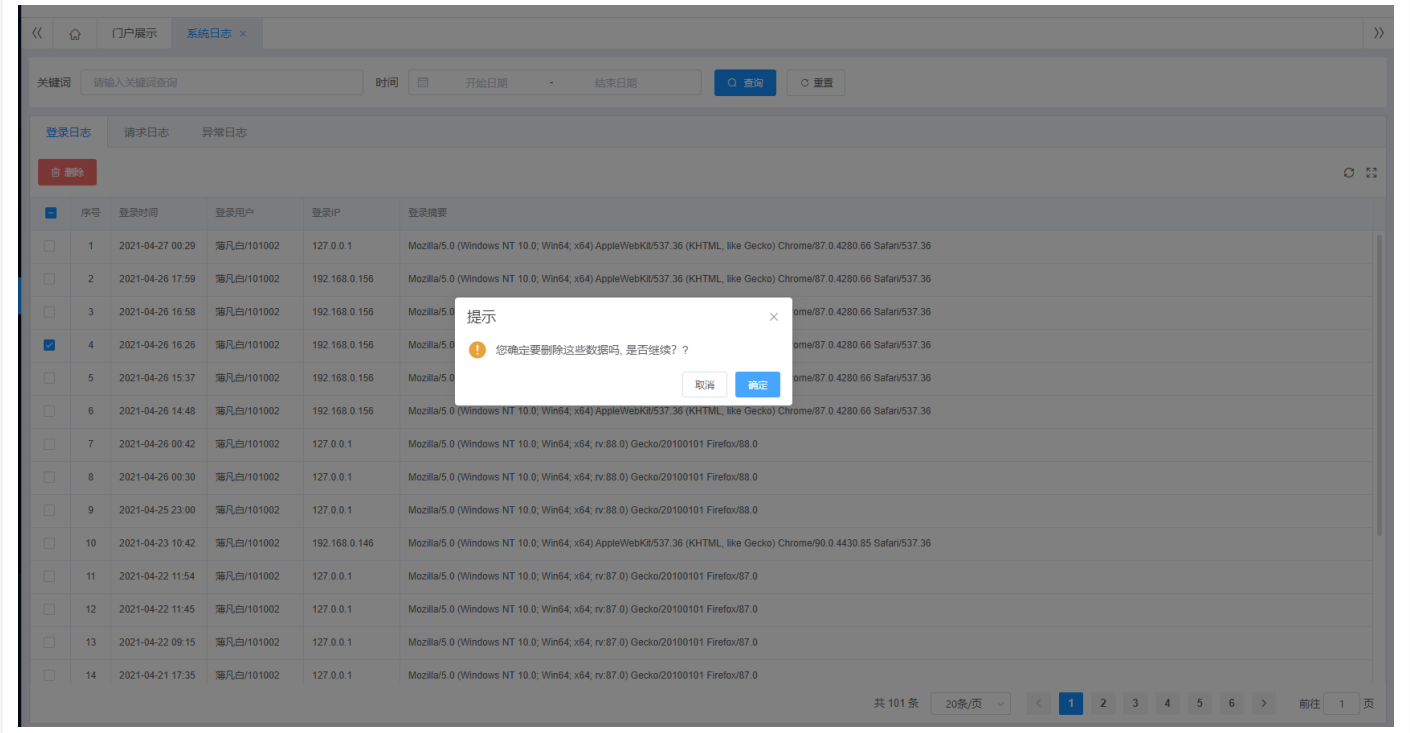
系统日志是记录每个用户操作系统一些日志。

操作步骤

系统管理目录下操作系统日志，进入【系统日志】页面，输入关键字进行查询功能、时间查询的系统日志信息，分别可以查看登录日志、请求日志、异常日志等信息可查看，对每一个账号登录访问进行监控。如下图所示：



在系统日志页面可以选择多条数据或一条数据进行删除，这时删除提示“您确定删除当前数据吗，是否继续？”。当点击确认按钮执行删除选择的数据，点击取消按钮离开删除提示页面、选择删除的数据还存在，没有执行删除语句。如下图所示：



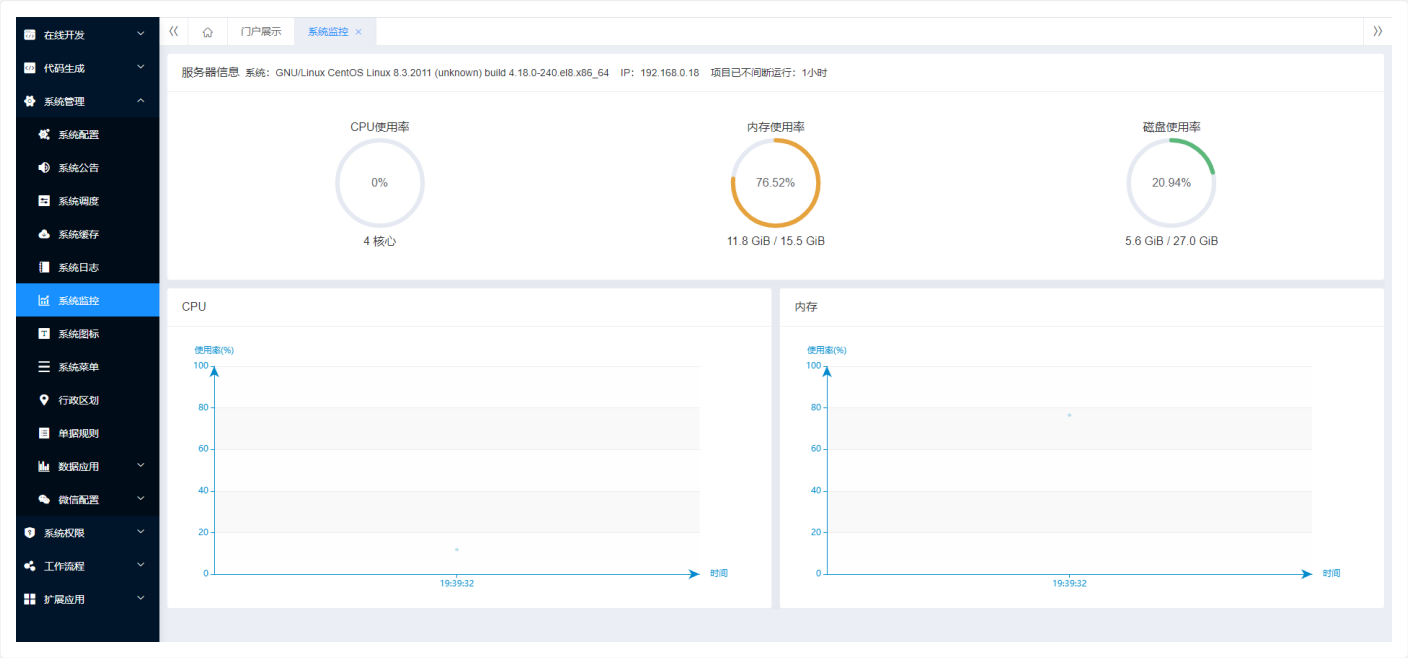
系统监控

功能描述

实现系统监控，直接查看服务器信息，实时监控服务器CPU、内存、磁盘等使用率

操作步骤

在系统管理目录下面打开系统监控，进入【系统监控】页面主要显示服务器CPU、内存、磁盘等使用率。如下图所示：



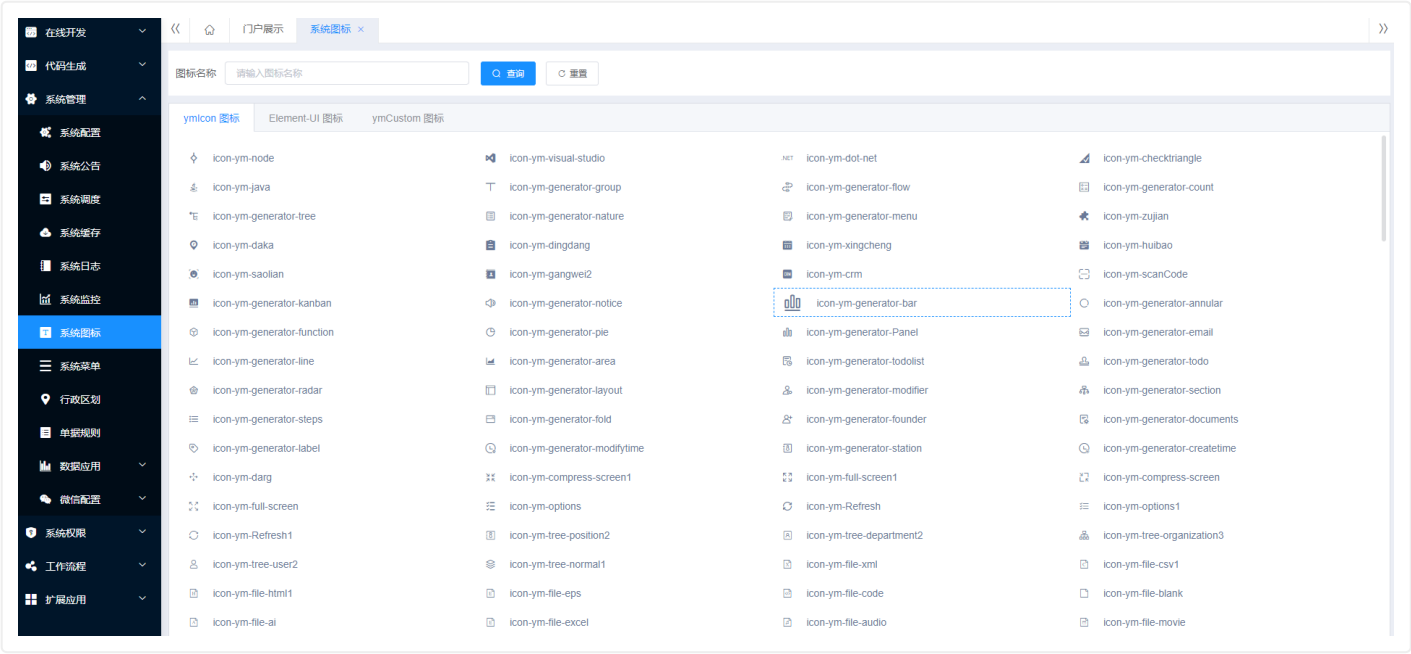
系统图标

功能描述

实现系统图标里面有各种图标符号应用库，为了方便在平台中调用

操作步骤

系统管理目录下系统图标，进入【系统图标】页面里面有各种图标符号应用库，方便在平台中调用。可以输入图标名称关键字进行查询功能。如下图所示：



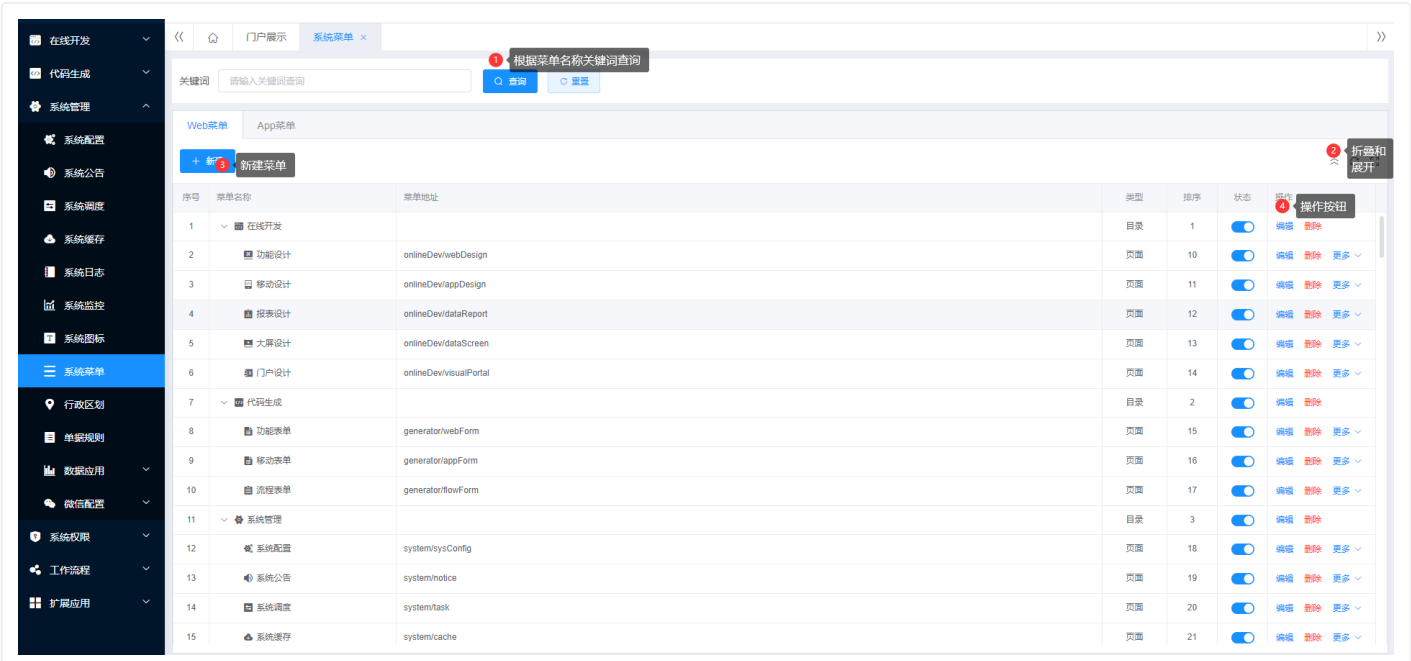
系统菜单

功能描述

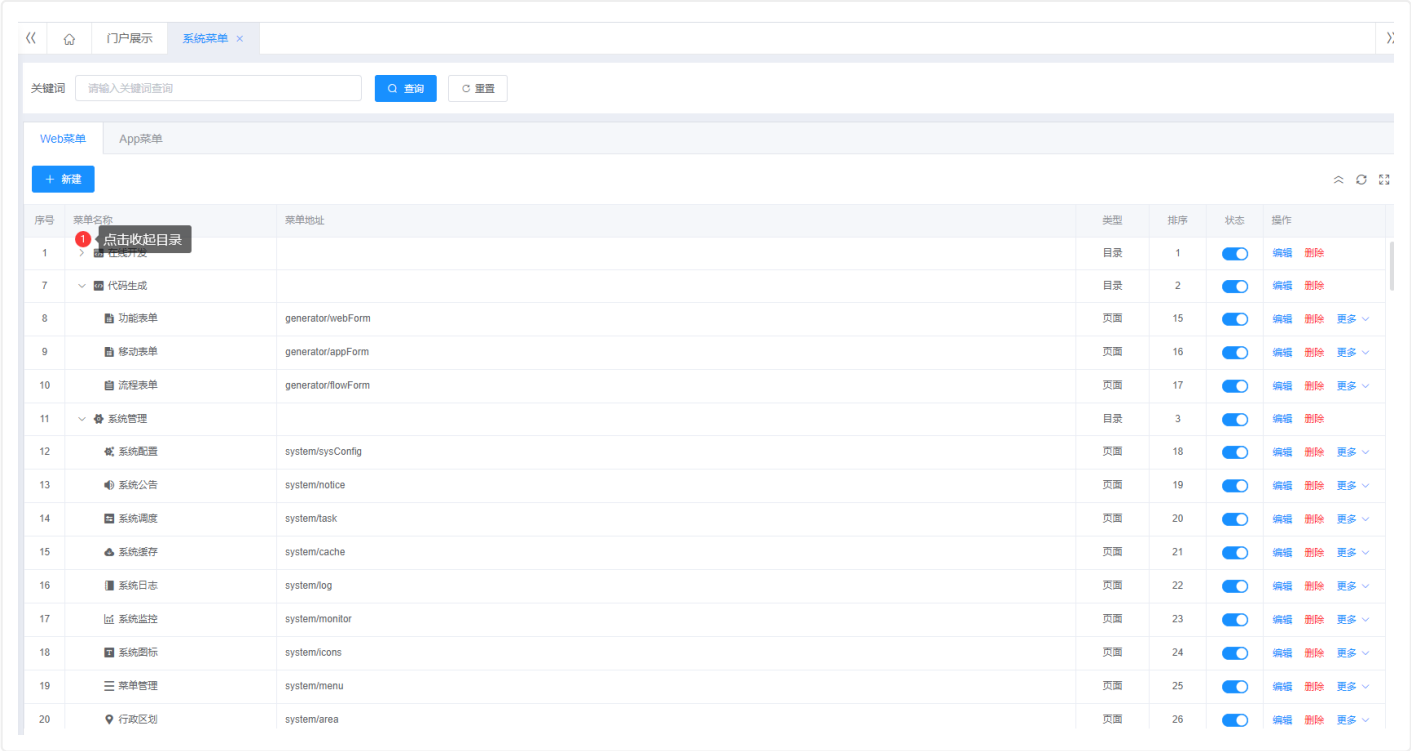
实现全模块的统一标签管理，实现菜单多种分类管理，支持用户多样化应用。实现系统层面的整合、管理层面的融合。

操作步骤

系统管理目录下系统菜单，进入【系统菜单】页面页面分成Web菜单和App菜单两种类型。进入菜单管理有查询、刷新、折叠、展开、全屏、新建菜单等功能，对一条菜单管理数据进行编辑、删除、更多等功能。在菜单管理页面输入菜单名称进行查询，在当前页面进行刷新和新建菜单功能。如下图所示：



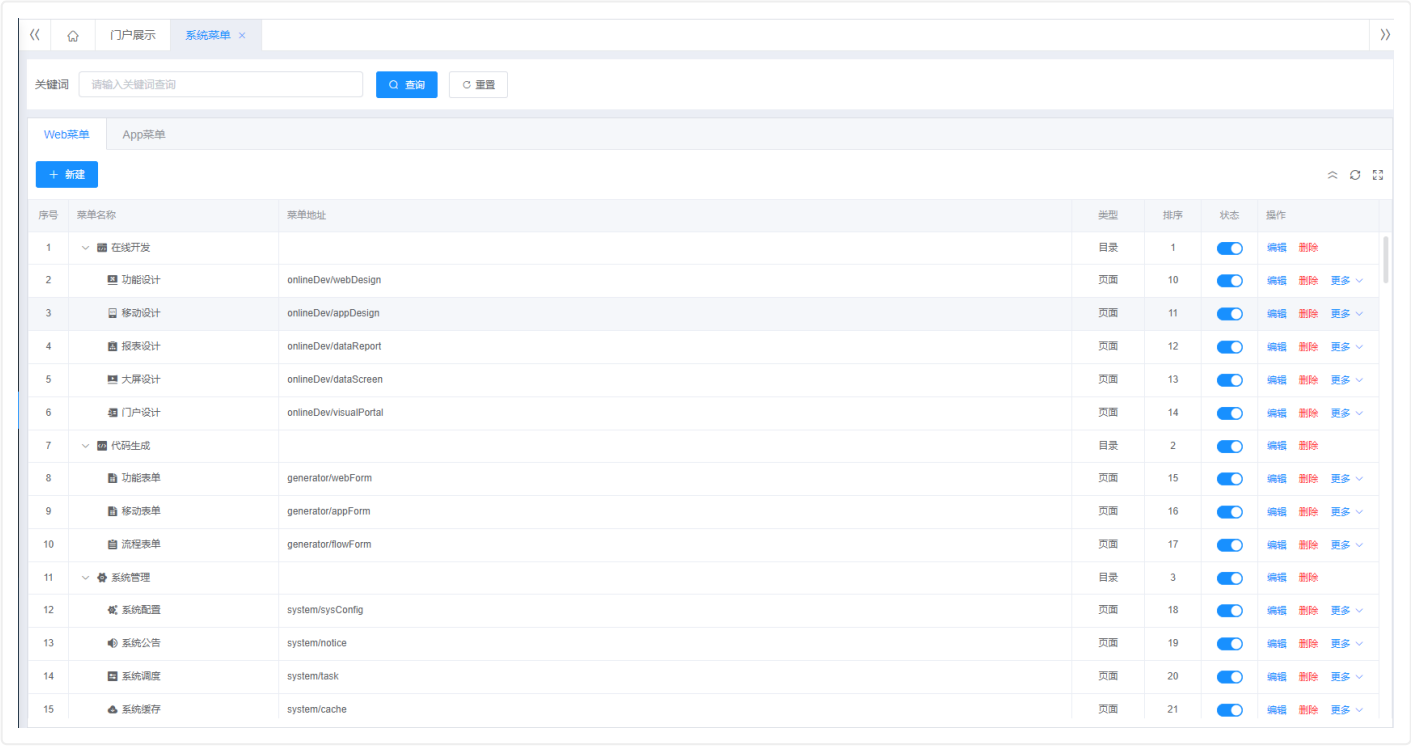
当鼠标选择“>”符号方向右边表示自动收起该目录下的子目录，当鼠标选择“﹀”符号方向下边表示自动展开该目录下的子目录。如下图所示：



a . 在菜单管理页面添加Web菜单，当在创建菜单页面进行下拉框选择上级菜单、菜单类型，填写菜单名称、编码、图标、类型（目录、页面、功能、报表、字典、大屏）、排序、说明，菜单状态（开启或关闭），该页面数据填写完、点击确定按钮保存该条数据或者取消按钮表示离开该页面。如下图所示：

b. 在菜单管理页面添加App菜单，当在创建菜单页面进行下拉框选择上级菜单、菜单类型，填写菜单名称、编码、图标、类型（目录、页面、功能）、排序、说明，菜单状态（开启或关闭），该页面数据填写完、点击确定按钮保存该条数据或者取消按钮表示离开该页面。如下图所示：

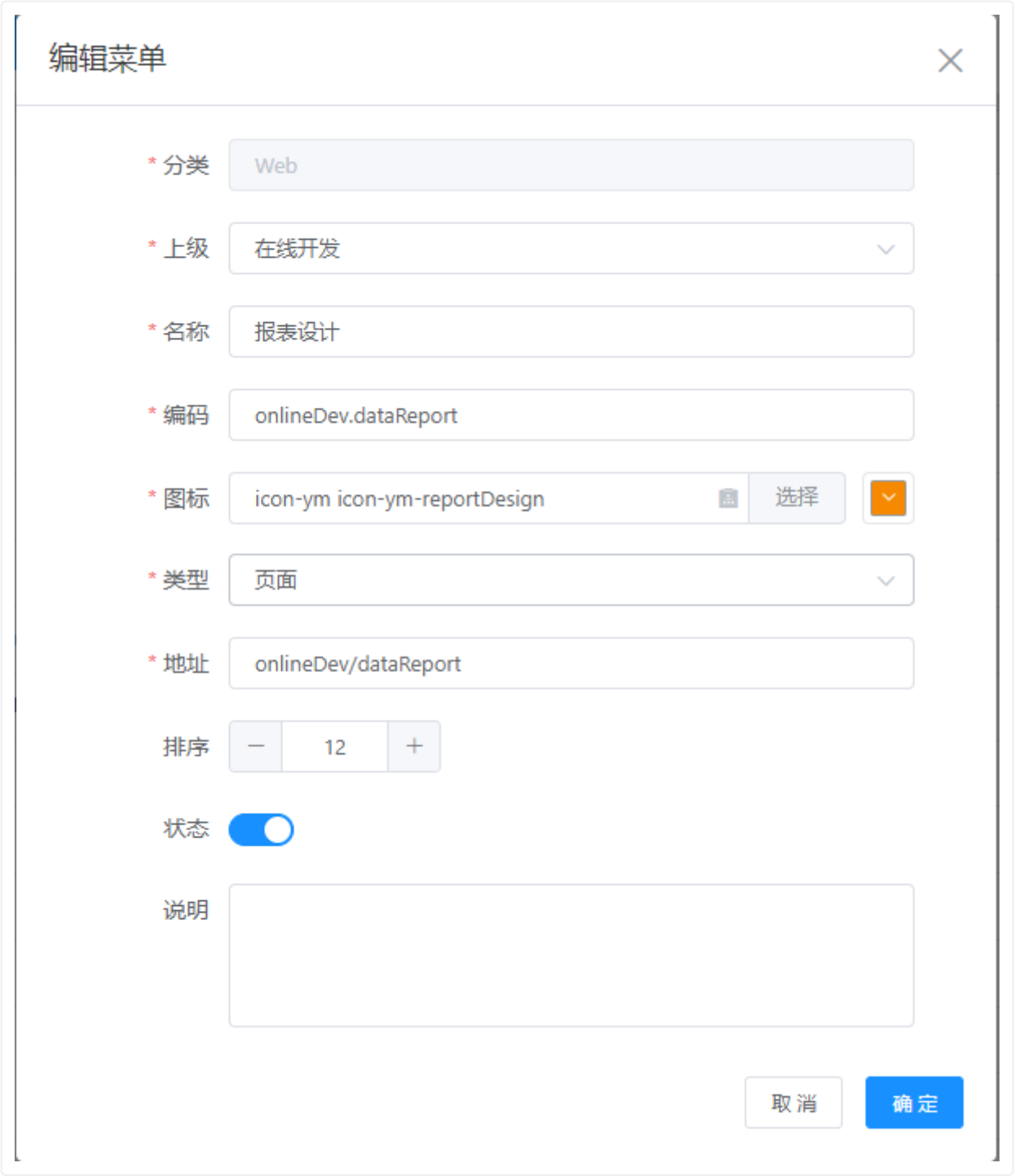
在菜单管理页面对一条数据进行编辑、删除、更多功能操作，如下图所示：



ⅰ在该页面对一条数据进行编辑，进入编辑页面、修改需要修改的地方，点击确认按钮保存该条数据或者点击取消按钮离开该页面。

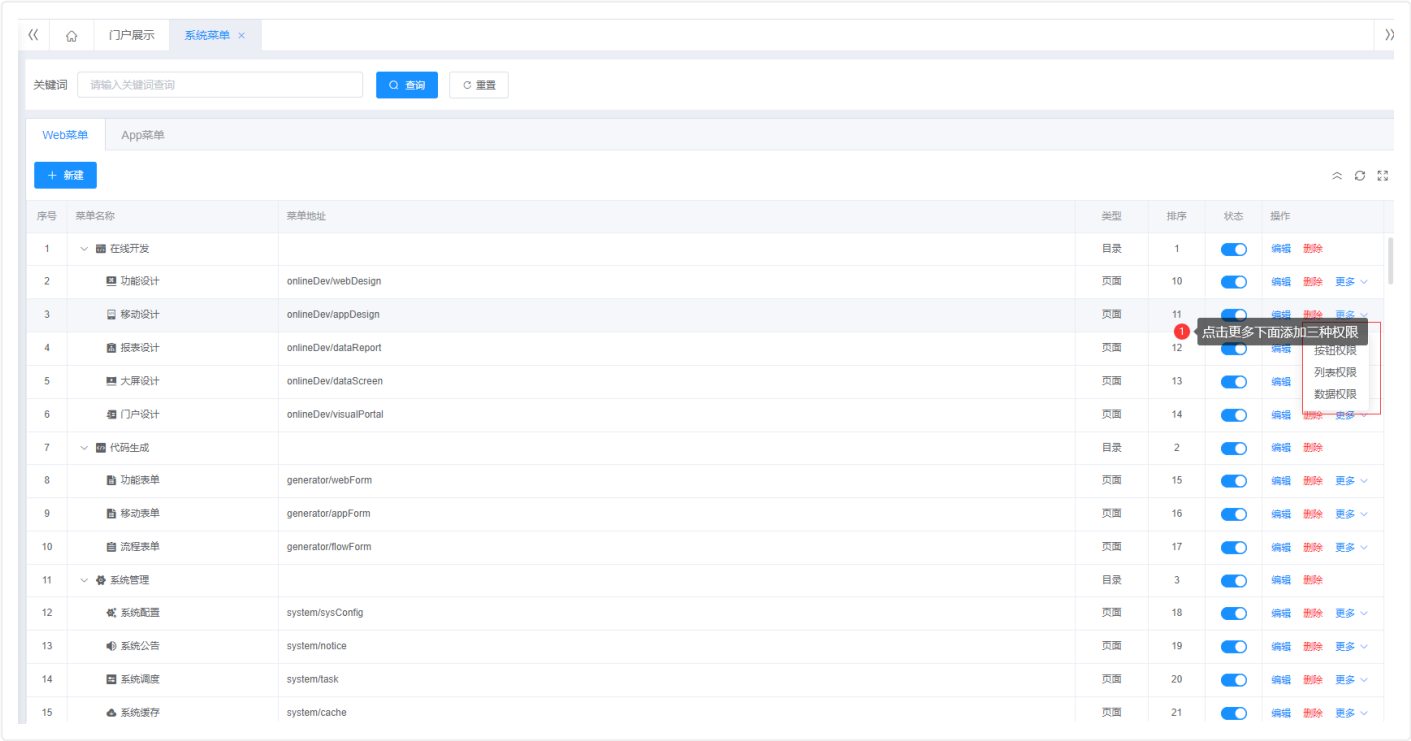
ⅱ在菜单管理页面只能选择子菜单点击删除、该条数据才能删除成功，选择该页面的主目菜单点击删除会提示您“此目录不引用不能够删除”，如果您必须要删除主目录，请先删除完下面的子目录、在删除主目录可以删除成功。

ⅲ菜单显示按照排序字段排序。如下图所示



在菜单管理中添加类型（页面、功能）时，更多按钮下面（按钮权限、列表权限、数据权限）

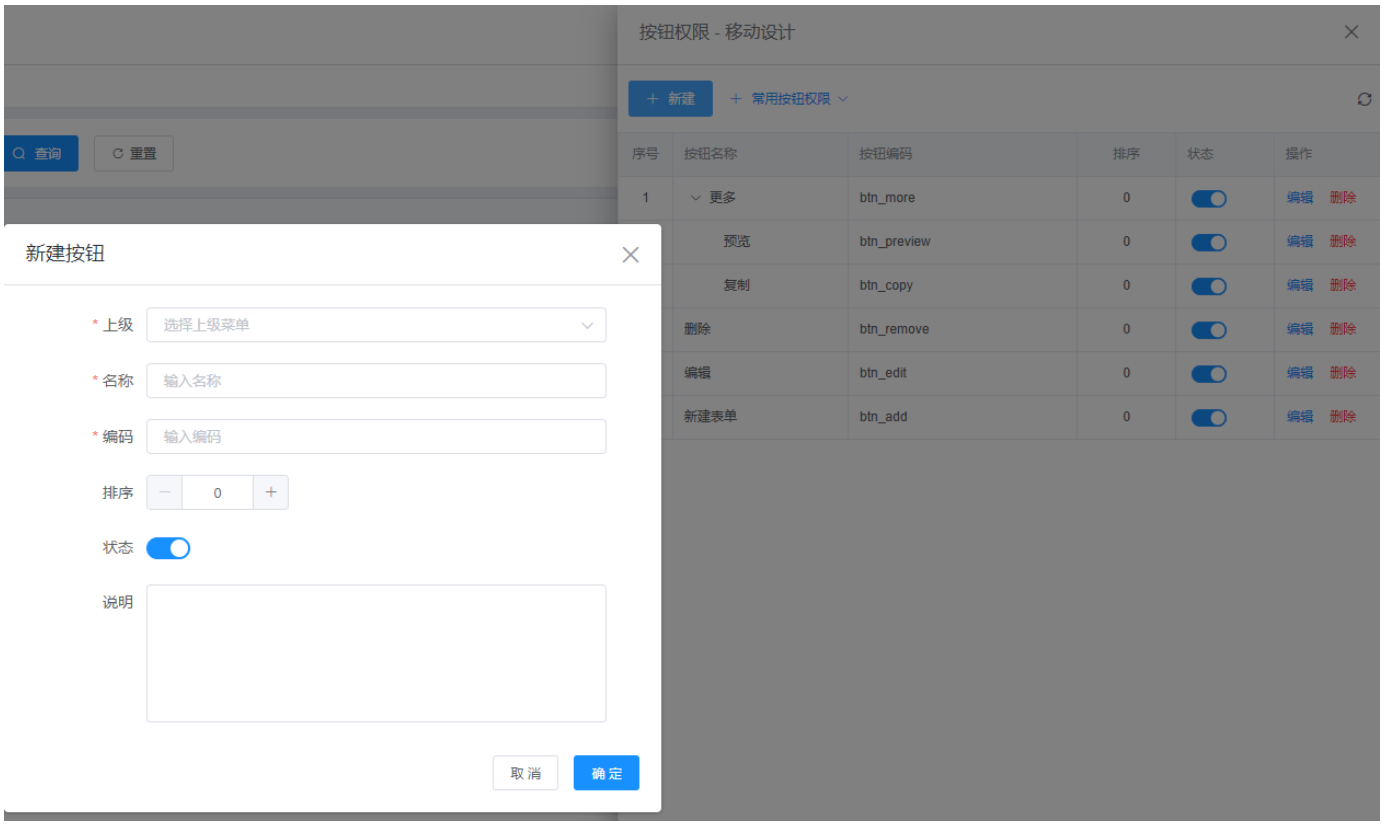
a.在菜单管理列表，点击更多，选择“按钮权限”显示出按钮权限列表。如下图所示：



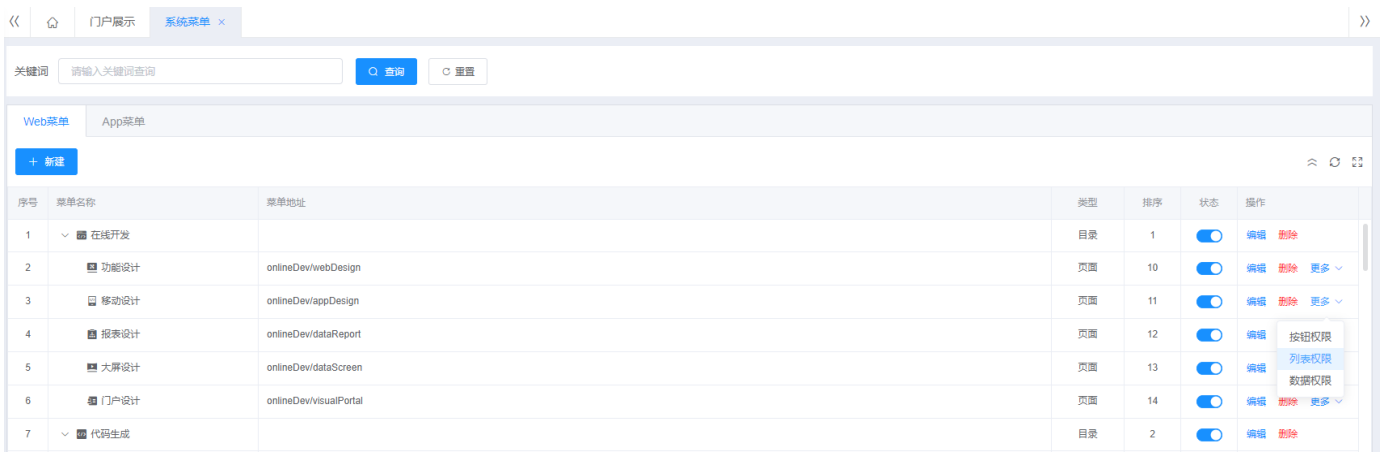
ⅰ按钮列表页面新建页面，进行上级、名称和编号填写完毕，点击确认按钮表示当前新建数据保存成功在功能分类中可以查看，或取消按钮表示当前新建数据未被保存、自动关闭当前页面。如下图所示：；

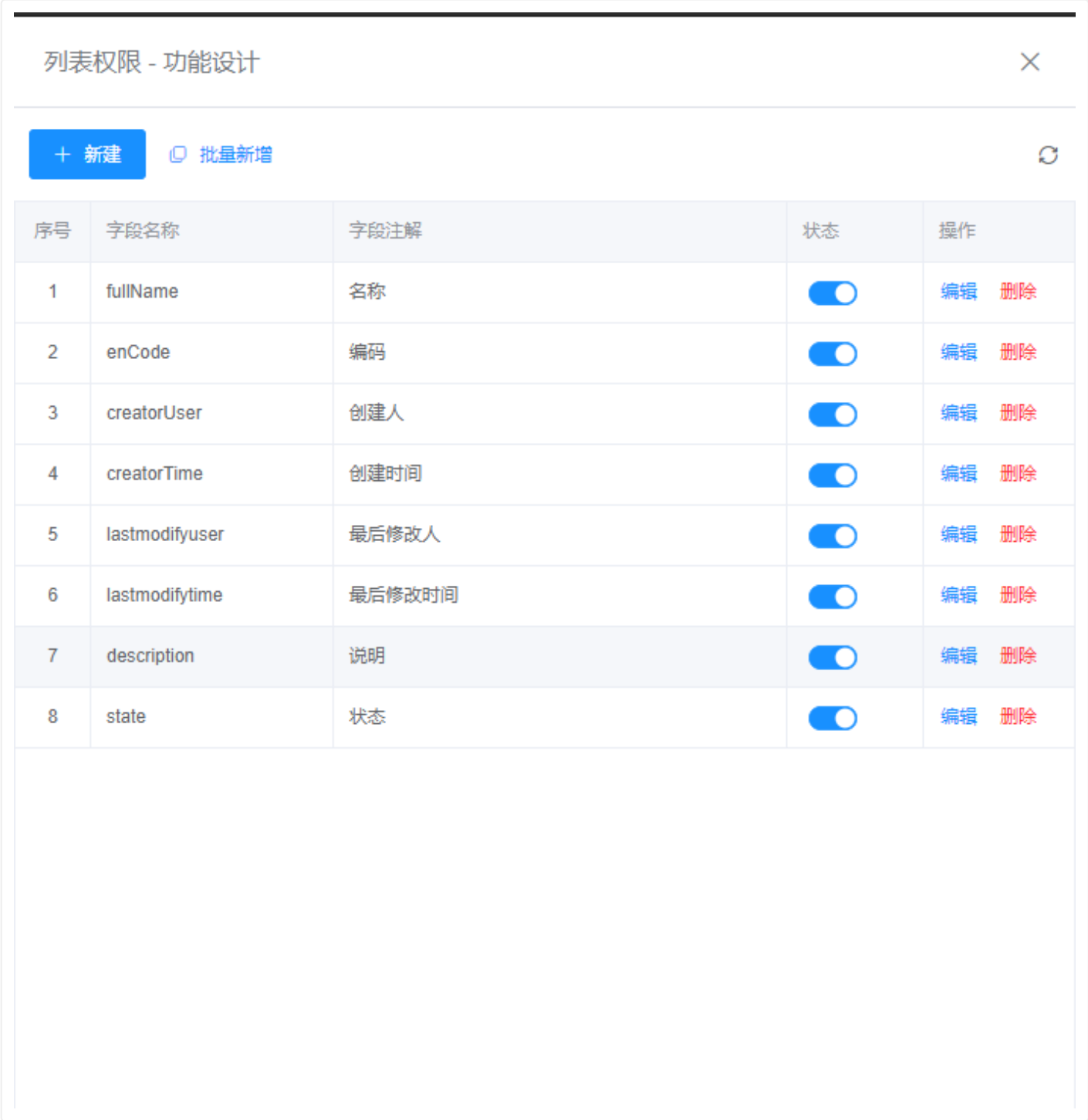
ⅱ对一条数据进行点击编辑按钮、数据编辑修改完毕操作确认保存数据或者关闭离开当前页面数据未被保存；

iii可以对直接删除成功。



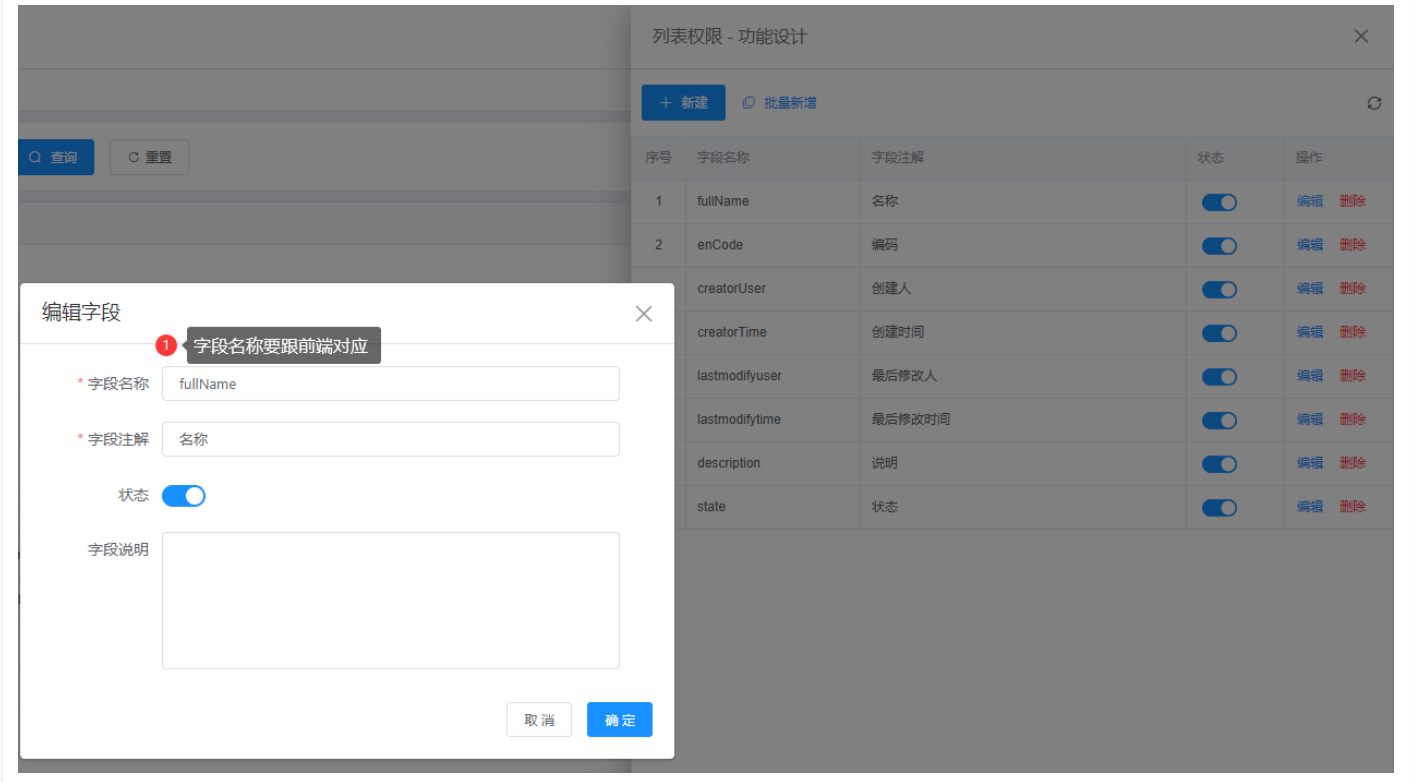
b.在菜单管理列表点击更多，选择“列表权限”显示出列表权限列表。进入列表权限页面有新建、批量新增、刷新等功能，如下图所示：





在列表权限页面左上角点击新建出现创建字段名称，创建字段注解、设置字段状态等等信息填写完点击确定按钮保存或者关闭离开该页面。如下图所示：

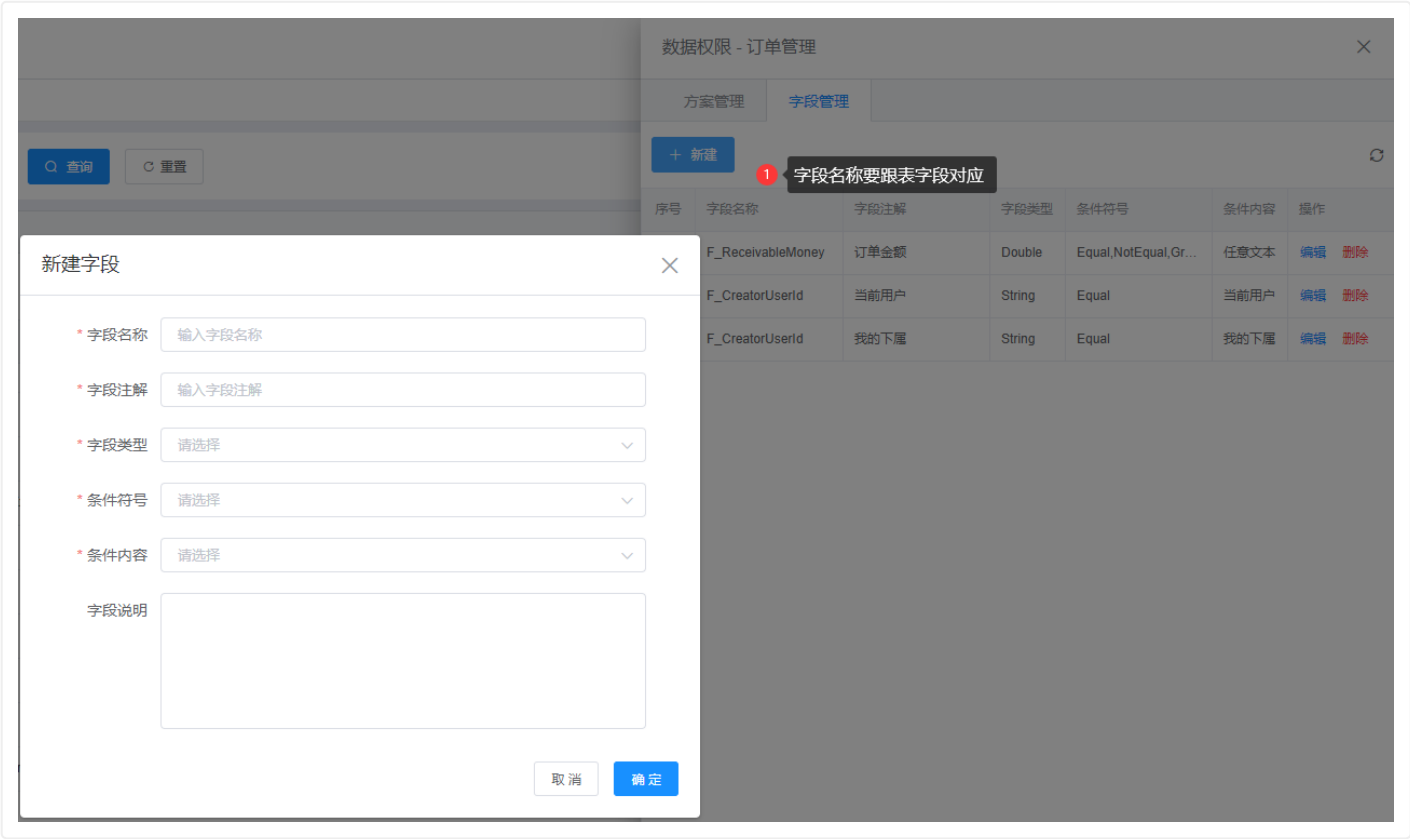
在列表管理页面对一条数据进行编辑、删除功能。如下图所示：



c.在菜单管理列表点击更多，选择“数据权限”显示出按钮权限列表。进入数据权限管理列表页面，该页面有刷新、创建字段管理、创建方案管理。点击创建字段管理、在创建字段管理页面字段名称、字段类型、条件符号等一系列信息填写完，点击确定按钮保存当前数据保存成功、显示在字段管理页面，或者关闭离开该页面、当前新建字段数据未被保存。如下图所示：

ⅰ 刷新当前页面；

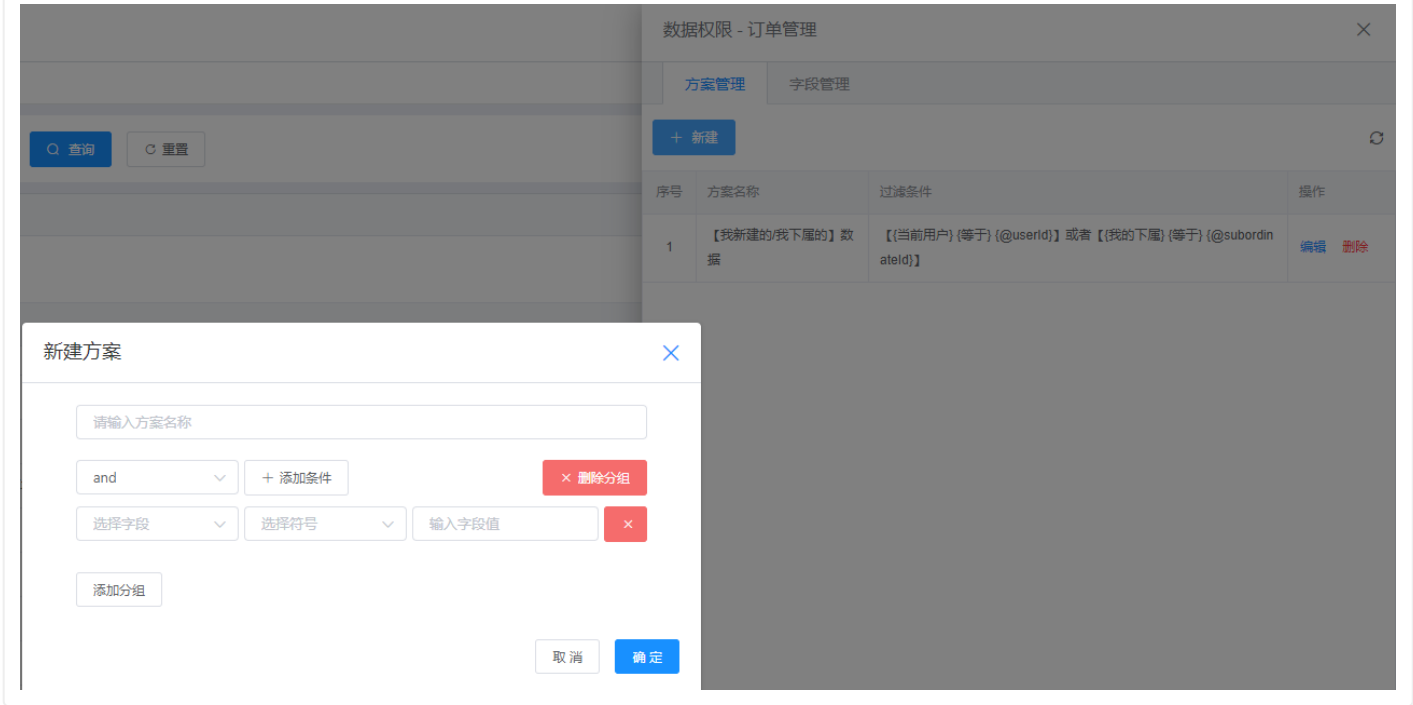
ⅱ 在字段管理新建字段后在设置方案管理；



在数据权限管理选择字段管理对一条数据有编辑、删除、刷新功能；如下图所示：

在方案管理页面有刷新、新建方案等功能，新建方案页面；如下图所示：

ⅰ刷新当前页面；



行政区划

功能描述

形成一个整体化体系，便于平台个性化开发引用模板，每个省、市区域级别等级以树形结构体现，区域名称、编号、说明等有地可查。

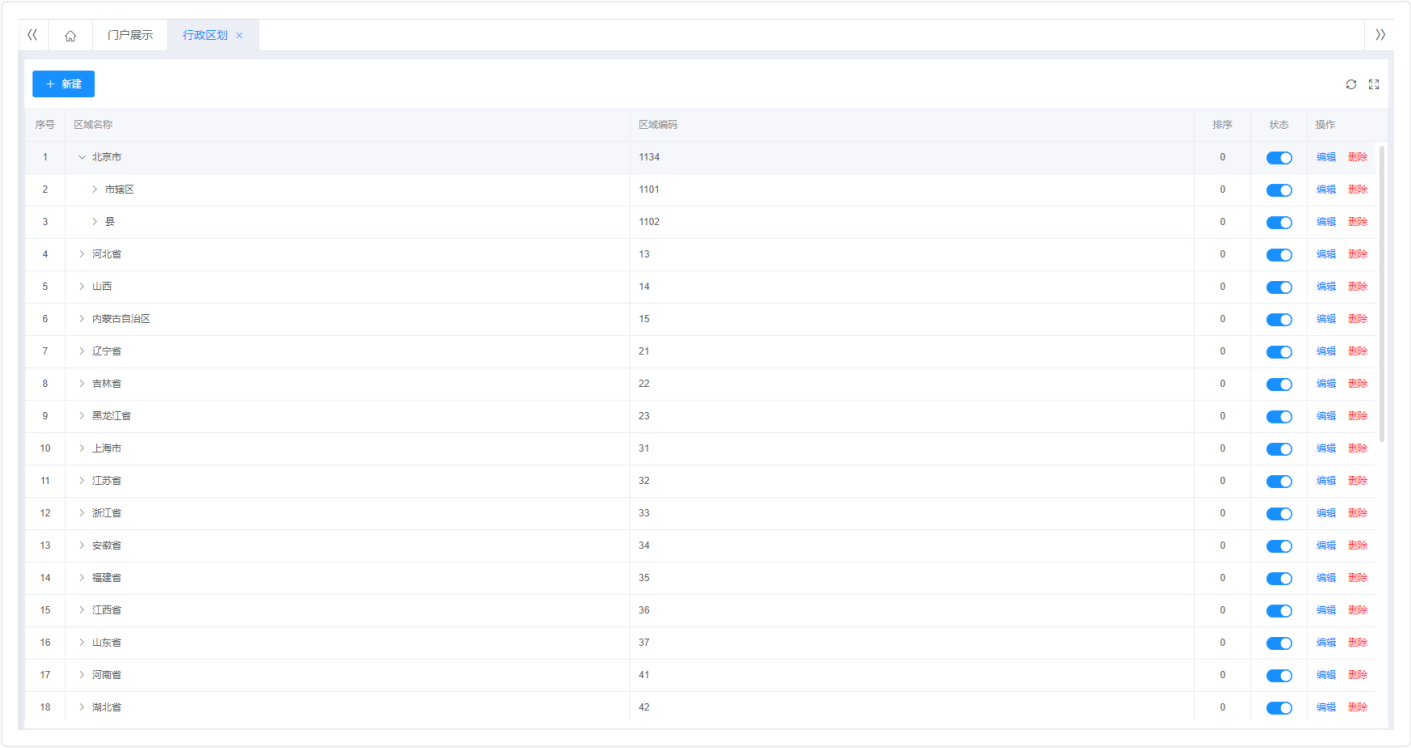
操作步骤

系统管理目录打开行政区划，进入【行政区划】页面有全屏、刷新、新建区域功能；如下图所示：

在行政区划页面新增区域上级（下拉框选择）、区域名称、区域编码、区域状态、说明等数据填写完毕，操作确定按钮表示保存当前数据在行政区划页面显示，或者取消按钮离开新建页面当前数据未被保存；如下图所示；

在行政区划页面选择对一条数据进行编辑、删除功能，如下图所示：

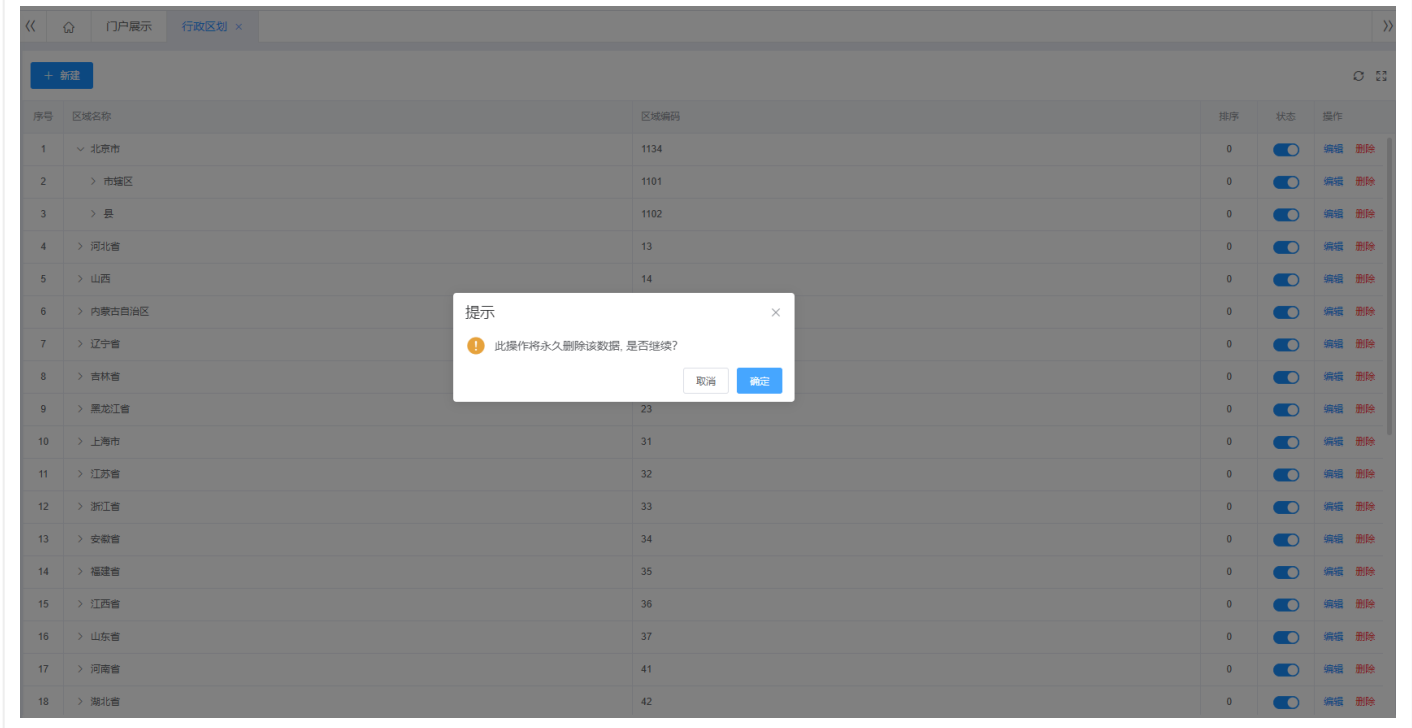
当鼠标选择>符号方向右边表示自动收起该目录下的子目录，当鼠标选择﹀方向下边表示自动展开该目录下的子目录。



ⅰ点击编辑按钮，进入编辑页面、修改需要修改的地方，点击确认按钮保存该条数据或者点击取消按钮离开该页面。

ⅱ点击删除提示“此操作将永久删除该数据，是否继续？”，如果是点击确认按钮删除此条数据或者取消关闭该页面、这条数据依然存在。如下图所示：


编辑区域
* 区域上级
顶级节点
* 区域名称
北京市
* 区域编码
1134
排序
0
状态
区域说明
取 消
确 定


门户展示
行政区划
新建
序号
区域名称
区域编码
排序
状态
操作
提示
此操作将永久删除该数据, 是否继续?
取消
确定

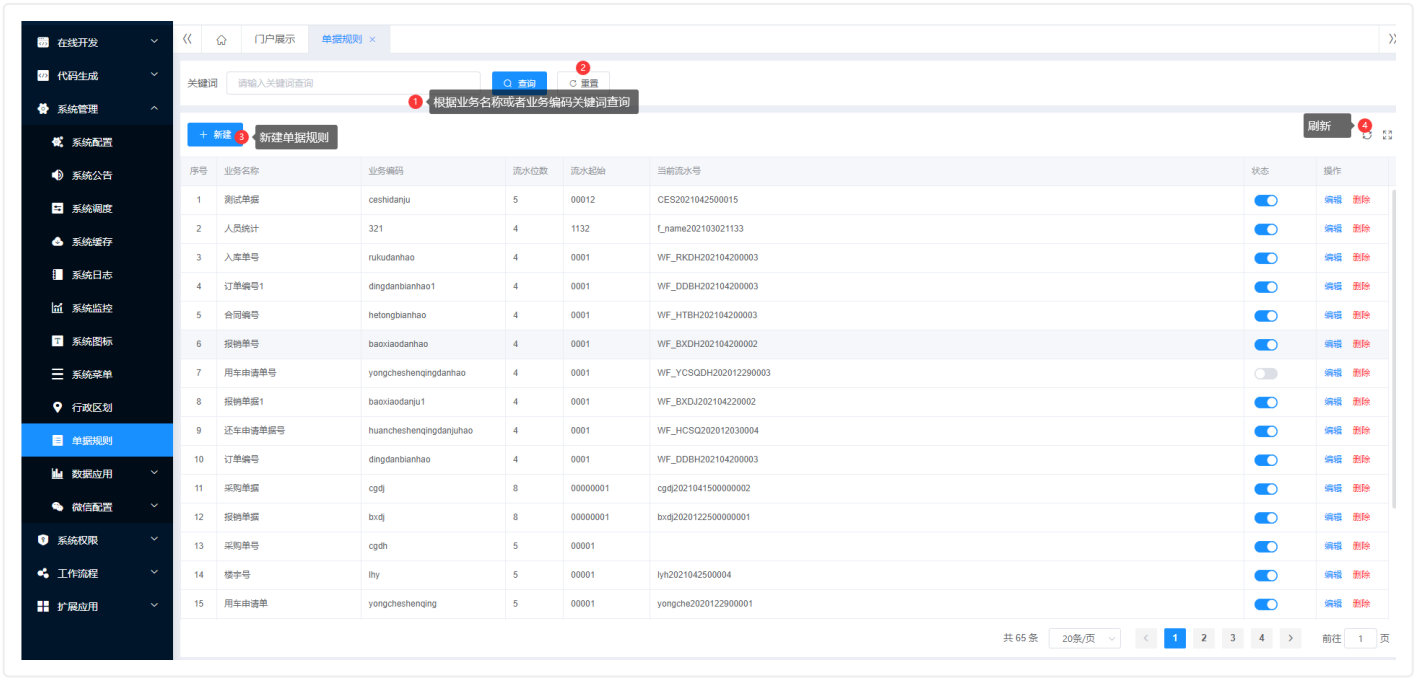
单据规则

功能描述

实现单据规则定义，为了工作流程系统表单定义的单据规则，以前缀+日期+流水号的规则生成单据编号，流水号位数，起始号可自定义。

操作步骤

系统管理目录打开单据规则，进入【单据规则】页面有查询、重置、刷新、新建单据等功能；如下图所示：



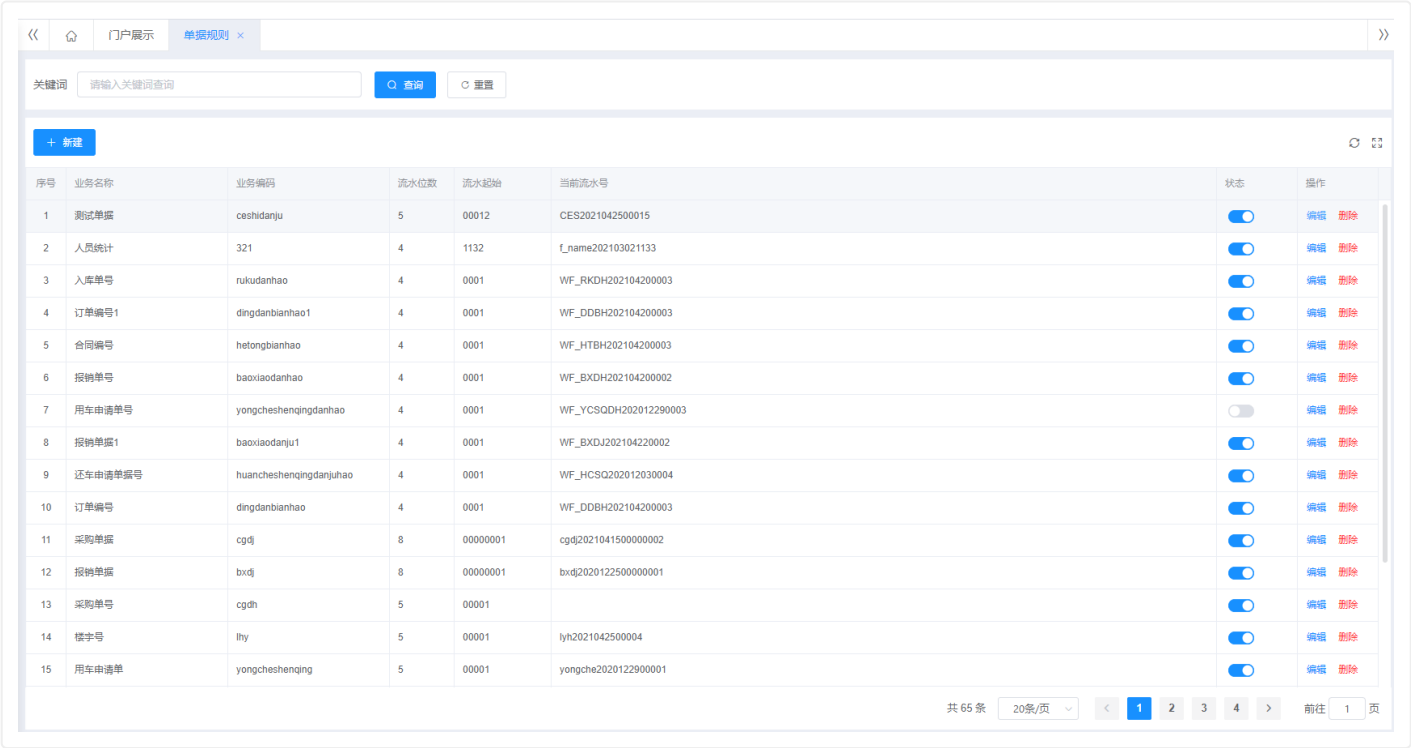
在单据规则页面左上角打开新建，进入新建单据页面、填写业务名称、编号、流水前缀、日期、位数、起始、范例、状态、说明，创建单据数据填写完点击确定按钮保存该条数据或者取消按钮离开该页面。如下图所示:

在单据规则页面对一条数据进行辑、删除、流水状态修改功能，如下图所示：

ⅰ 点击编辑按钮，进入编辑页面、修改需要修改的数据，点击确认按钮保存该条数据或者点击取消按钮离开该页面。

ⅱ 点击删除按钮提示“此操作将永久删除该数据，是否继续？”，如果是点击确认按钮删除此条数据或者取消关闭该页面、这条数据依然存在，如果当前删除的数据被应用不能够直接删除。

ⅲ 单据规则页面对一条数据流水状态按钮进行开启或关闭，如下图所示：



数据应用

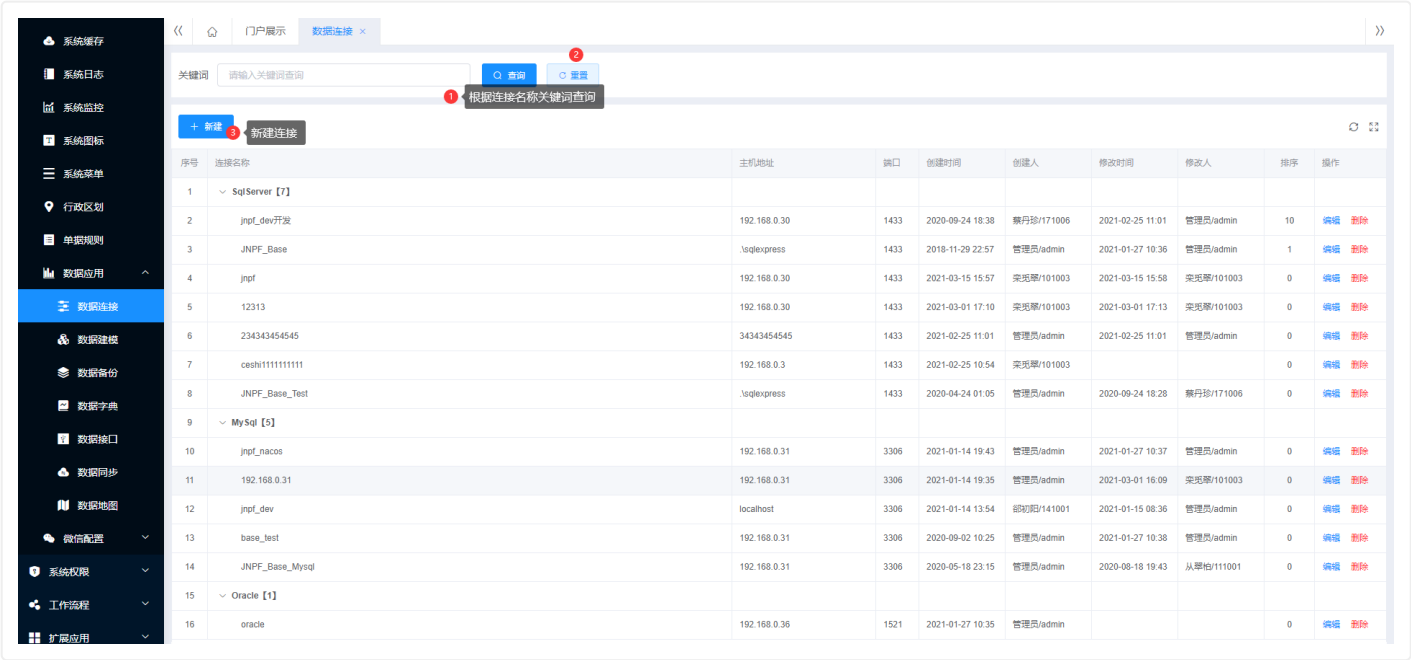
数据连接

功能描述

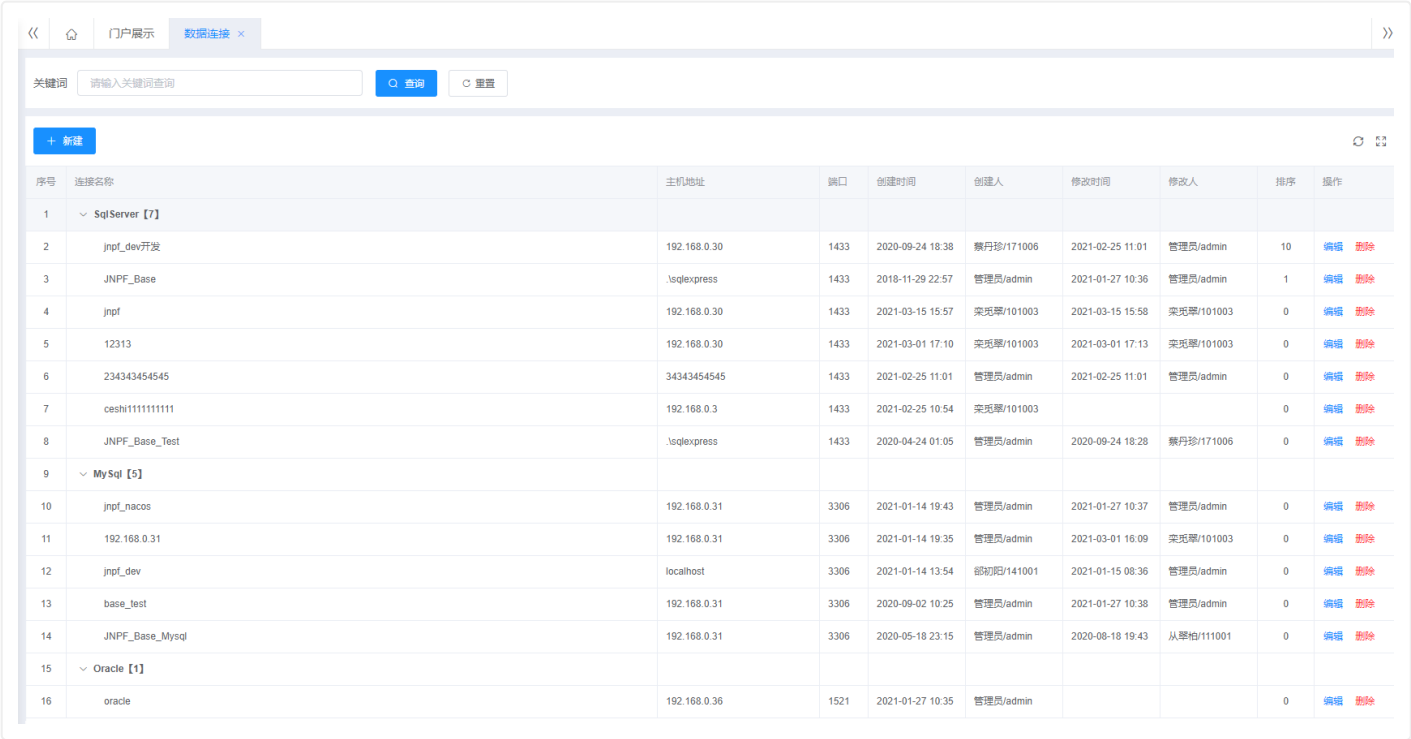
创建数据库连接，配置完成后可测试数据库是否连接成功，可连接不同（SQL、MySql、Oracle）数据库。

操作步骤

在系统管理目录下打开数据应用中数据连接，进入【数据连接】页面有查询、重置、刷新、新建数据连接，在新建连接名称数据类型（Sqlserver/MySql/Qracle）连接字符串等信息，点击确认按钮当前数据保存成功或者取消按钮离开该页面、当前数据未保存；如下图所示：



在数据连接页面对一条数据进行编辑、删除、排序功能，如下图所示：



序号	连接名称	主机地址	端口	创建时间	创建人	修改时间	修改人	排序	操作
1	SqlServer【7】								
2	jnpf_dev开发	192.168.0.30	1433	2020-09-24 18:38	蔡丹珍/171006	2021-02-25 11:01	管理员/admin	10	编辑 删除
3	JNPF_Base	.\sqlexpress	1433	2018-11-29 22:57	管理员/admin	2021-01-27 10:36	管理员/admin	1	编辑 删除
4	jnpf	192.168.0.30	1433	2021-03-15 15:57	栾觅翠/101003	2021-03-15 15:58	栾觅翠/101003	0	编辑 删除
5	12313	192.168.0.30	1433	2021-03-01 17:10	栾觅翠/101003	2021-03-01 17:13	栾觅翠/101003	0	编辑 删除
6	234343454545	34343454545	1433	2021-02-25 11:01	管理员/admin	2021-02-25 11:01	管理员/admin	0	编辑 删除
7	ceshi1111111111	192.168.0.3	1433	2021-02-25 10:54	栾觅翠/101003			0	编辑 删除
8	JNPF_Base_Test	.\sqlexpress	1433	2020-04-24 01:05	管理员/admin	2020-09-24 18:28	蔡丹珍/171006	0	编辑 删除
9	MySql【5】								
10	jnpf_nacos	192.168.0.31	3306	2021-01-14 19:43	管理员/admin	2021-01-27 10:37	管理员/admin	0	编辑 删除
11	192.168.0.31	192.168.0.31	3306	2021-01-14 19:35	管理员/admin	2021-03-01 16:09	栾觅翠/101003	0	编辑 删除
12	jnpf_dev	localhost	3306	2021-01-14 13:54	邵初阳/141001	2021-01-15 08:36	管理员/admin	0	编辑 删除
13	base_test	192.168.0.31	3306	2020-09-02 10:25	管理员/admin	2021-01-27 10:38	管理员/admin	0	编辑 删除
14	JNPF_Base_Mysql	192.168.0.31	3306	2020-05-18 23:15	管理员/admin	2020-08-18 19:43	从琴柏/111001	0	编辑 删除
15	Oracle【1】								
16	oracle	192.168.0.36	1521	2021-01-27 10:35	管理员/admin			0	编辑 删除

数据建模

功能描述

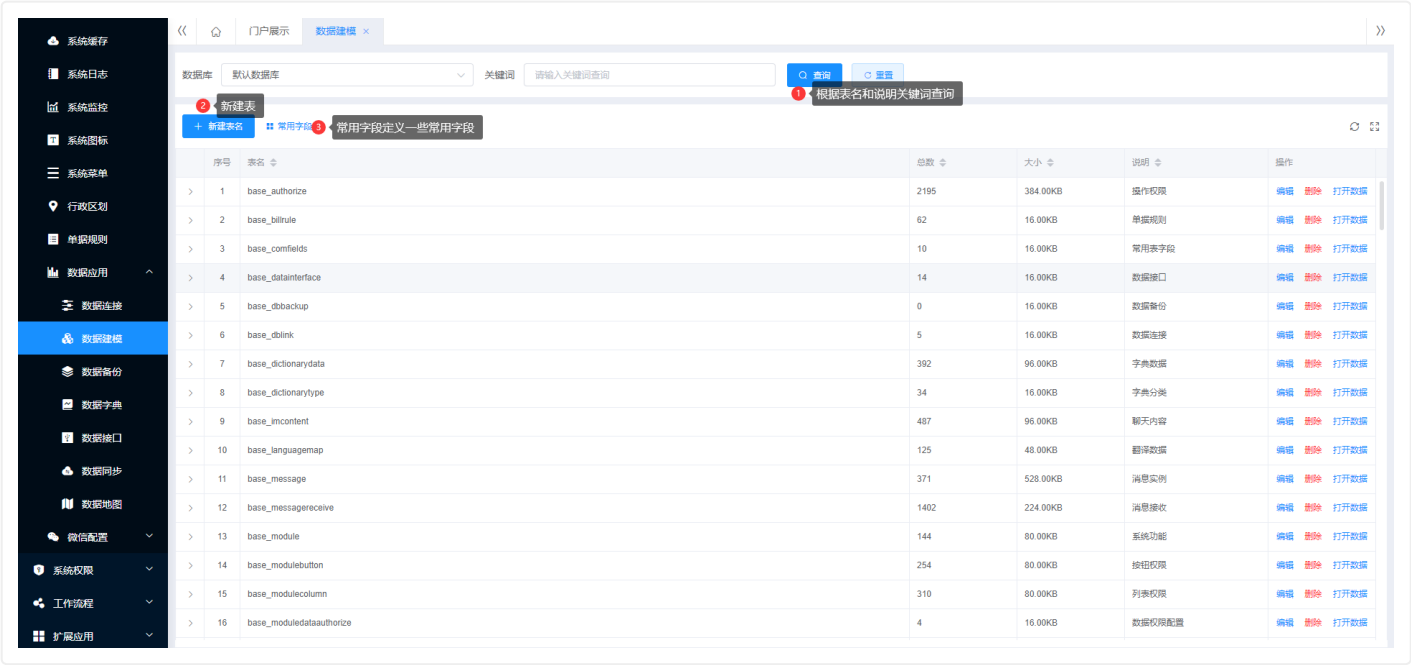
用来描述数据库中基本表的设计，主要包括字段名、数据类型、主键、外键等描述表的属性的内容。

操作步骤

在系统管理目录下打开数据应用中数据建模，进入数据建模页面有查询、刷新、新建表名、常用字段等功能；如下图所示：

ⅰ刷新当前页面；

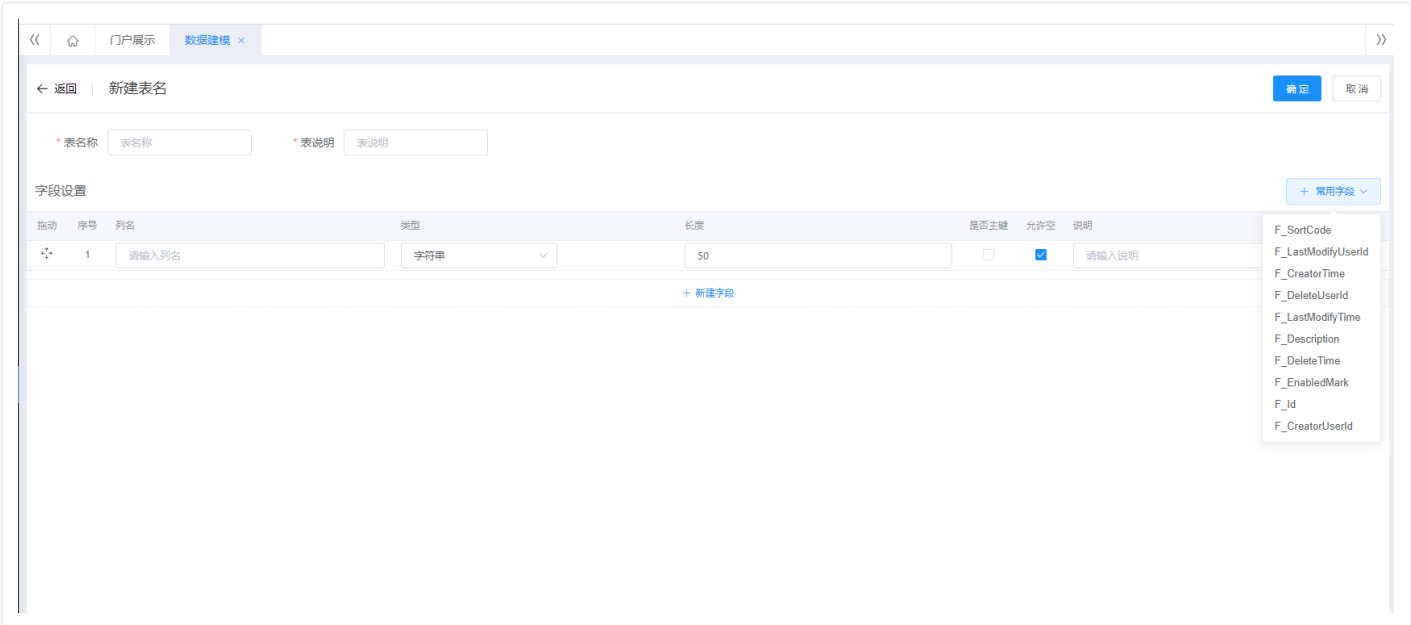
ⅱ在当前页面下拉框选择条件在输入表名进行查询；



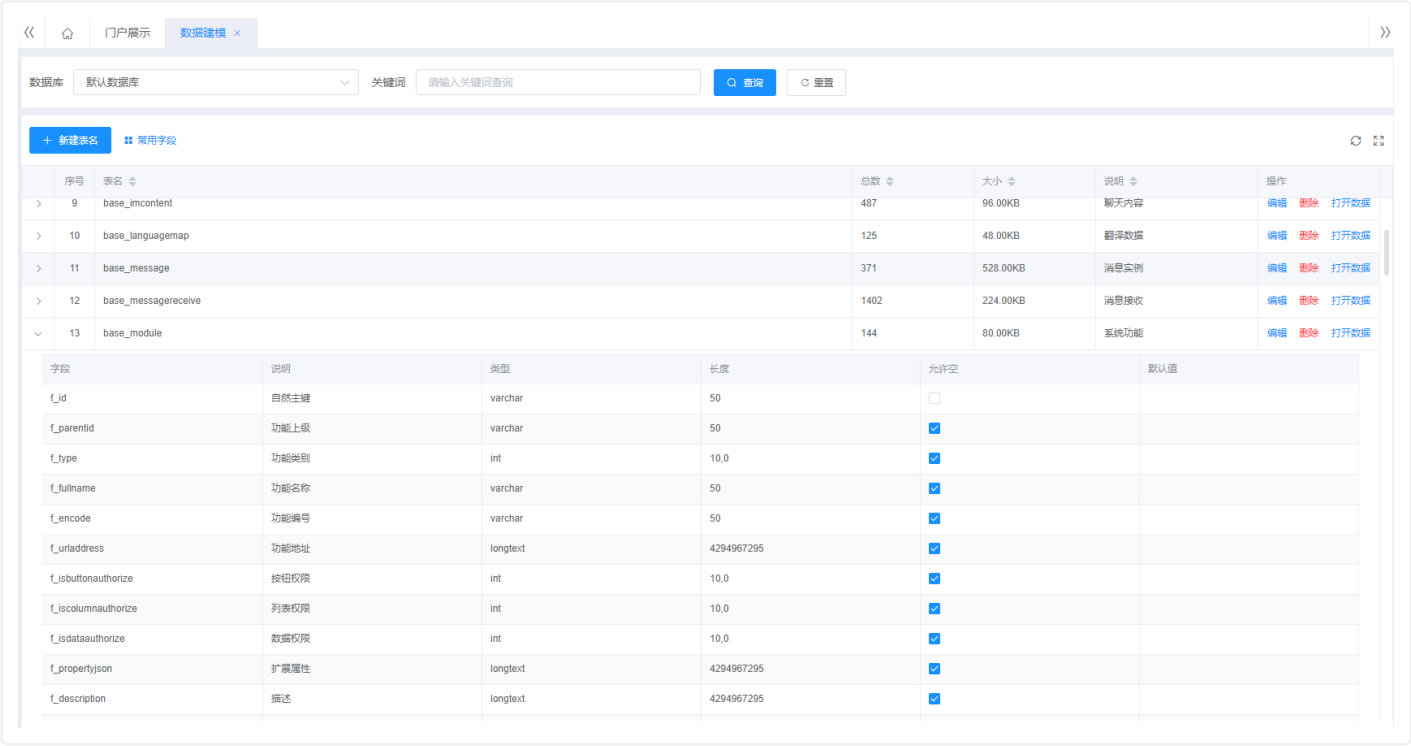
在数据建模页面打开新建表名，在该页面必填项表名称、表说明，字段设置里面可以进行字段新增、常用字段、是否主键、删除操作，这样使用起来方便快捷；如果数据填写完毕点击确认按钮当前数据保存成功或者消息按钮离开该页面、当前数据未保存；如下图所示：

ⅰ字段设置中“新建字段”表示新增当前行；

ⅱ字段设置中“删除”表示删除当前行；



当在数据建模页面对一条数据进行编辑、删除、打开数据以及查看子目录功能；如下图所示：

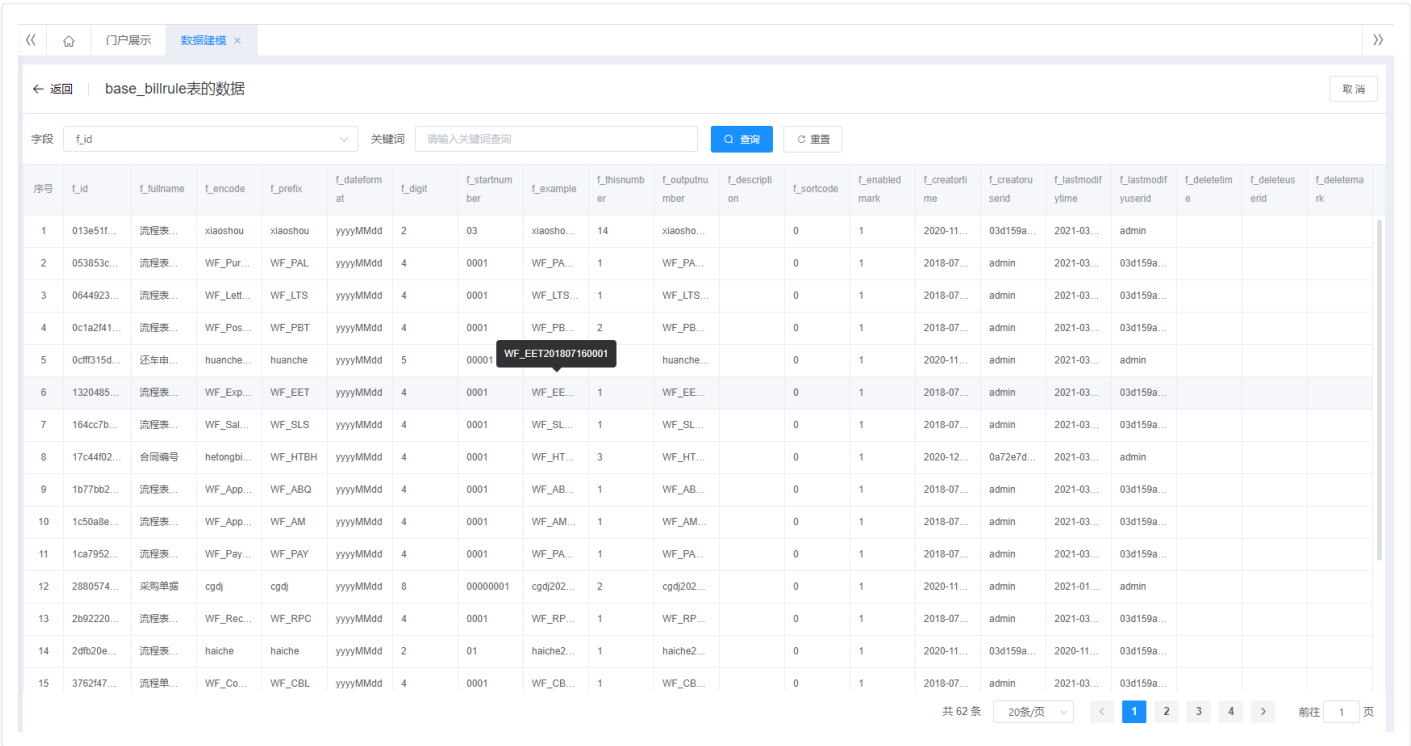


ⅰ选择当前数据如果被应用后就不能够直接删除成功；

ⅱ选择表名点击操作三角形显示该目录下的子目录信息；

ⅲ当对一条数据操作打开数据按钮时，自己进入当前选择表数据页面；

如下图所示



在当前表的数据页面下拉框选择搜索条件（f_id、f_fullname等等）然后输入条件下的关键字进行查询，然而当前查询的数据跳转在页面；如下图所示：

ⅰ如果清空当前输入关键字点击查询、数据返回当前这张表全部数据页面；

ii 操作当前表数据关闭按钮时、当前这张表数据页面自己关闭跳转到数据建模页面；

数据字典

功能描述

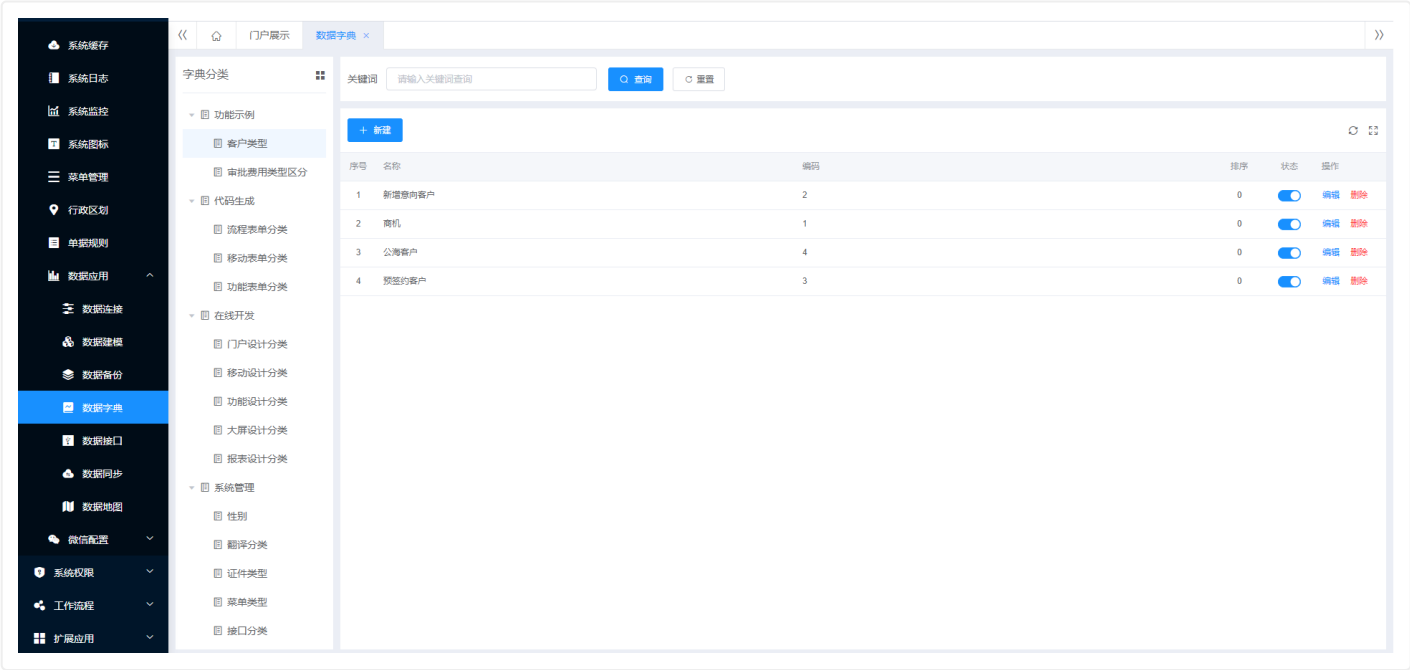
系统中所有数据元素的定义的集合。

操作步骤

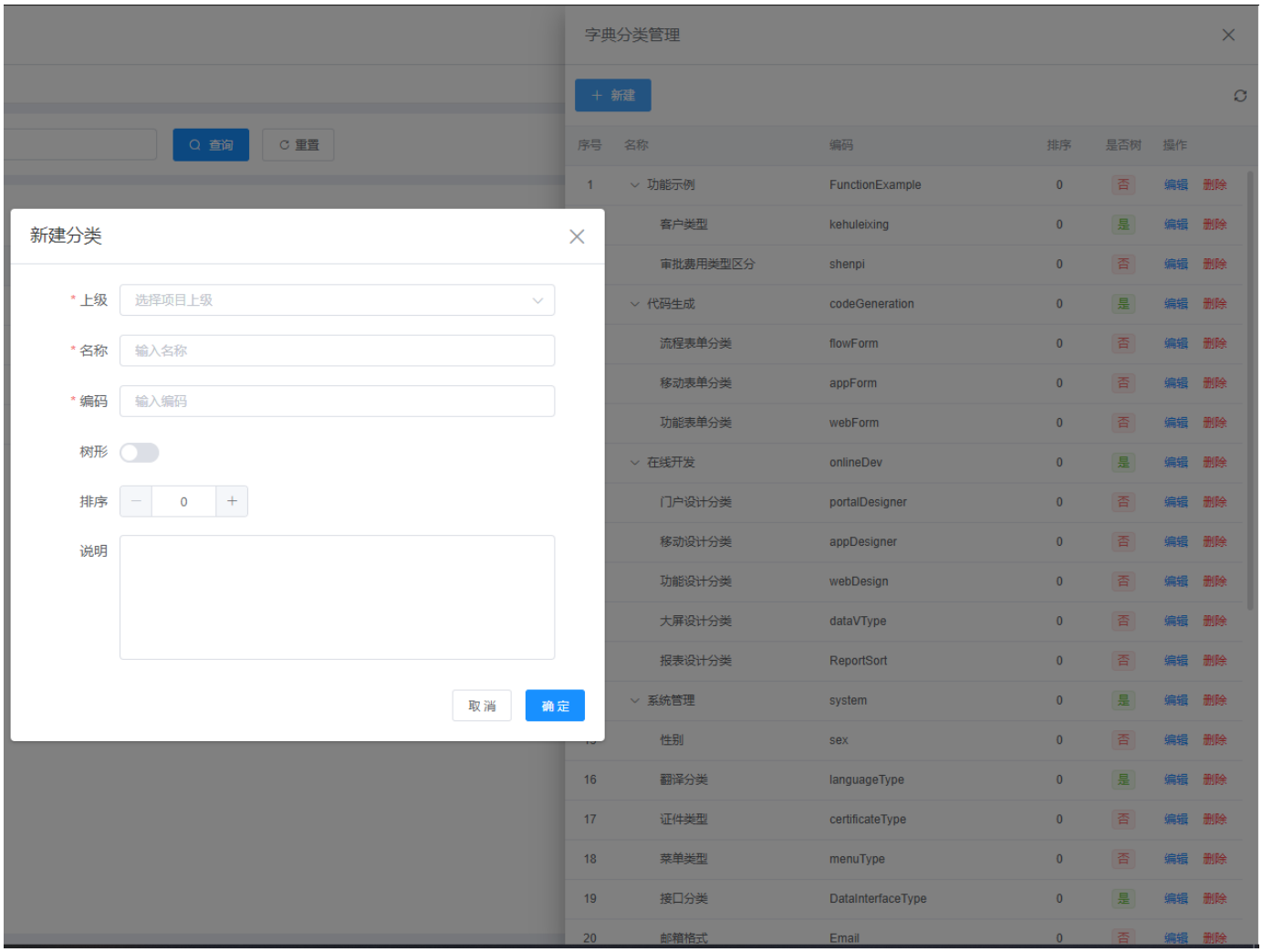
在系统管理目录下打开数据应用中数据字典，进入【数据字典】页面有字典分类和字典列表，在数据字典列表页面有查询、刷新、全屏、新建等功能。如下图所示：

i 刷新数据字典当前页面；

ii 在数据字典页面输入字典名称或者编码进行查询；

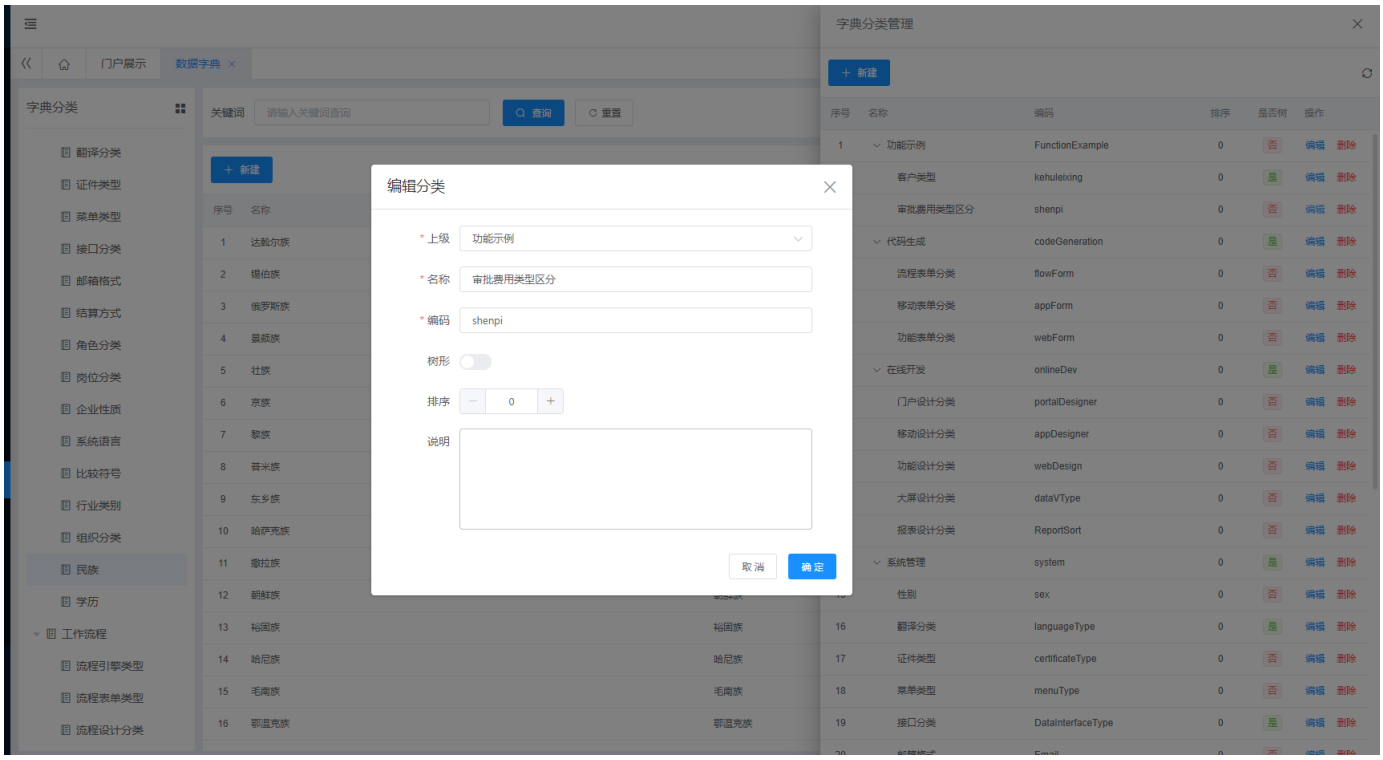


a.在字典分类栏目旁边“正方形”符号点击出现字典分类管理页面，在该页面点击“新建”按钮出现新建分类页面有选择下拉框选择上级填写名称、编号、说明、选择树形状态等，该页面必填信息填写完点击确认按钮保存该条数据或者取消按钮离开该页面。如下图示：



在字典分类管理页面里面有编辑、删除功能。如下图所示：

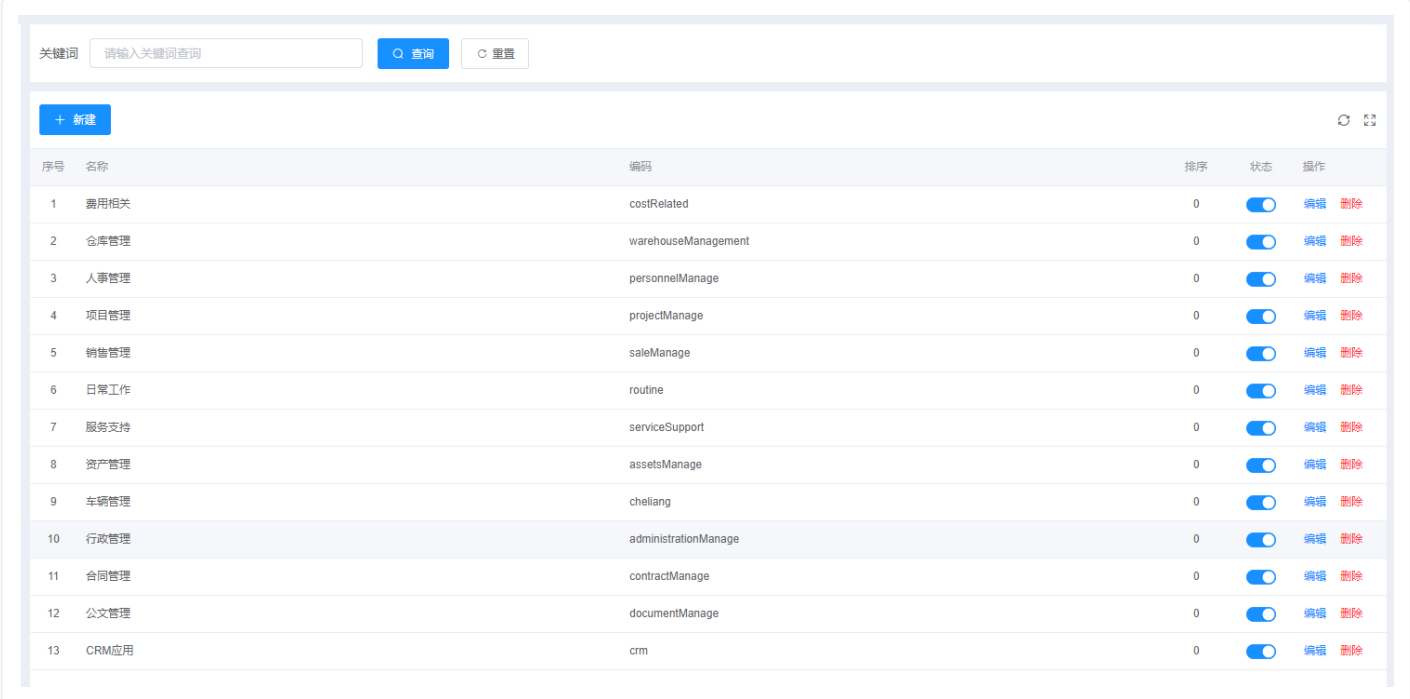
在数据字典分类页面对一条数据进行编辑，进入编辑页面、修改需要修改的地方，点击确认按钮保存该条数据或者点击取消按钮离开该页面当前数据未被保存。如下图所示：



ⅰ在数据字典分类页面对数据点击删除、该条数据没有被应用才能删除成功，选择该页面的主目菜单点击删除会提示您“此目录不引用不能够删除”不能够直接删除，如果您必须要删除目录，请先删除被应用的目录、在删除该录才能删除成功。

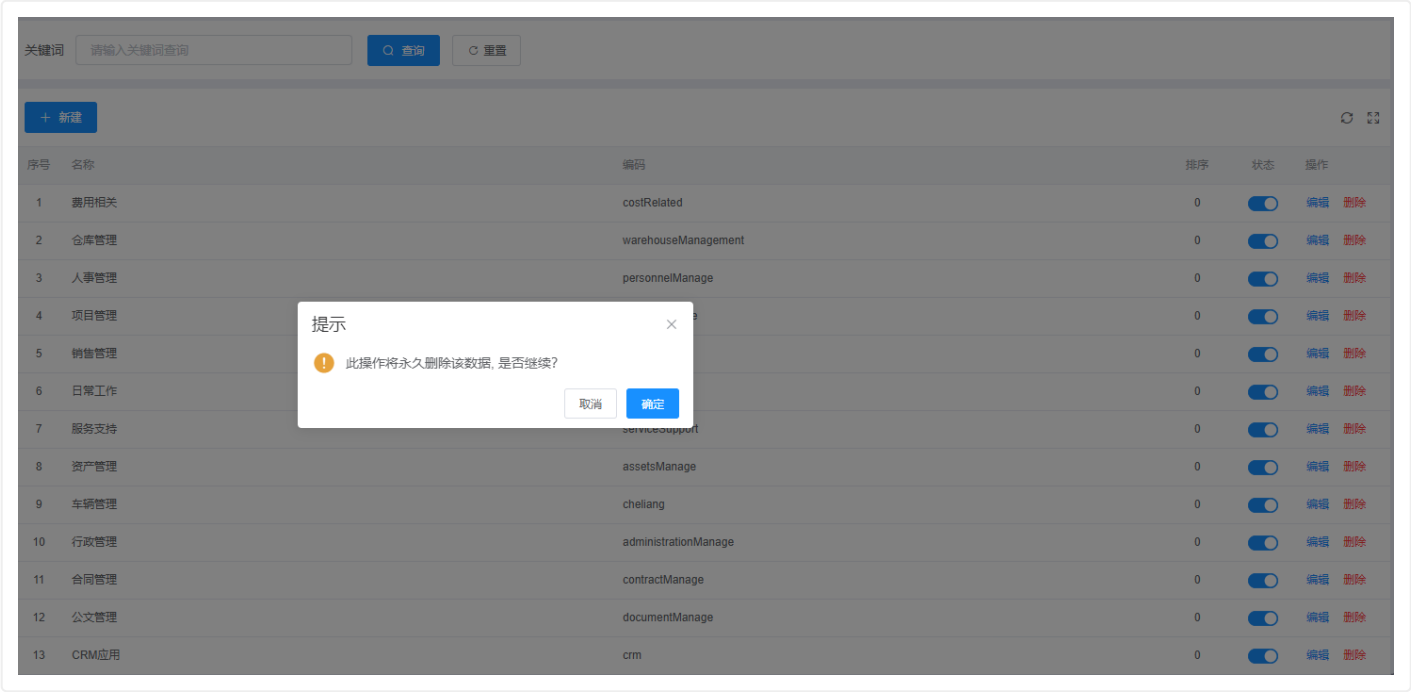
b.字典列表页面操作添加数据、进入创建字典页面，下拉框选择字典上级、填写字典名称、代码、说明、状态、排序，当数据填写完点击确定按钮保存新增该条数据或者取消按钮离开该页面。如下图所示：

在字典列表页面对一条数据进行编辑、删除、状态按钮修改功能，如下图所示：



ⅰ在该页面对一条数据进行编辑，进入编辑页面、修改需要修改的地方，点击确认按钮保存该条数据或者取消关闭按钮离开该页面。

ii 在字典列表页面选择条数据点击删除提示“此操作将永久删除该数据，是否继续？”，如果是点击确认按钮删除此条数据或者取消关闭该页面、这条数据依然存在。如下图所示：



字典列表页面显示顺序可以根据排序字段排序。

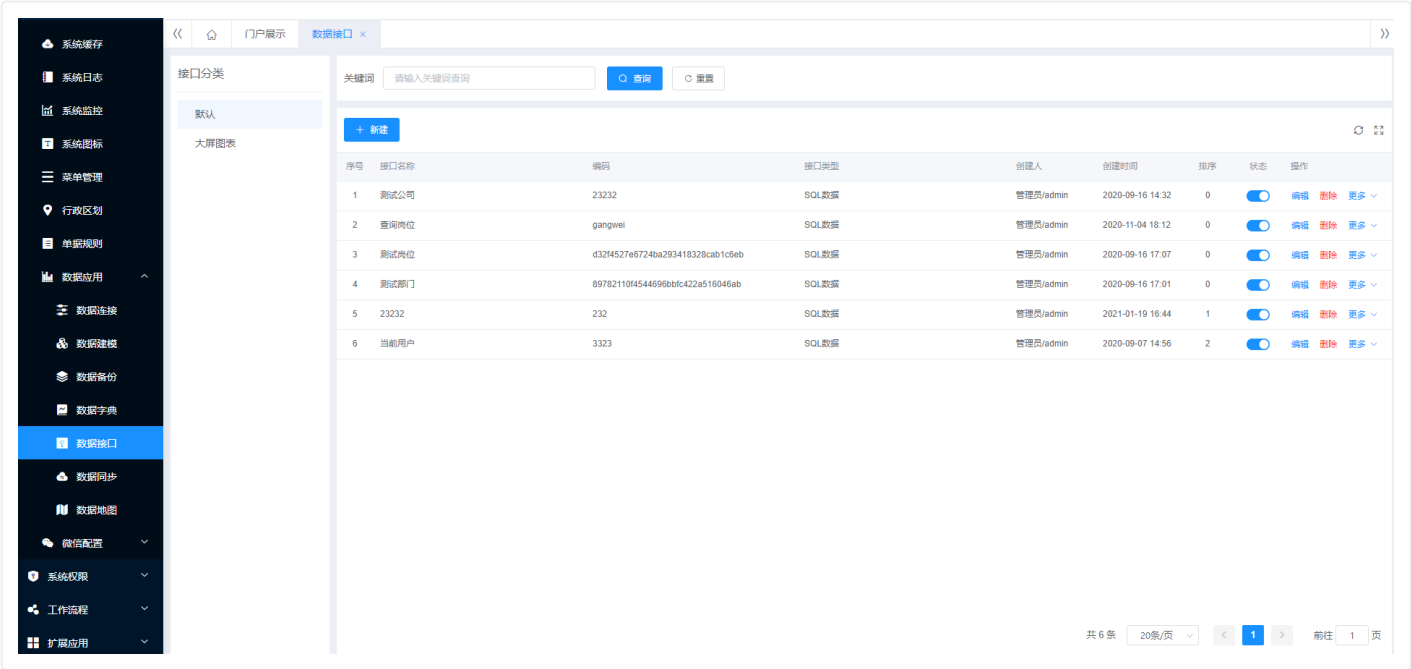
数据接口

功能描述

第三方数据接口配置对接。

操作步骤

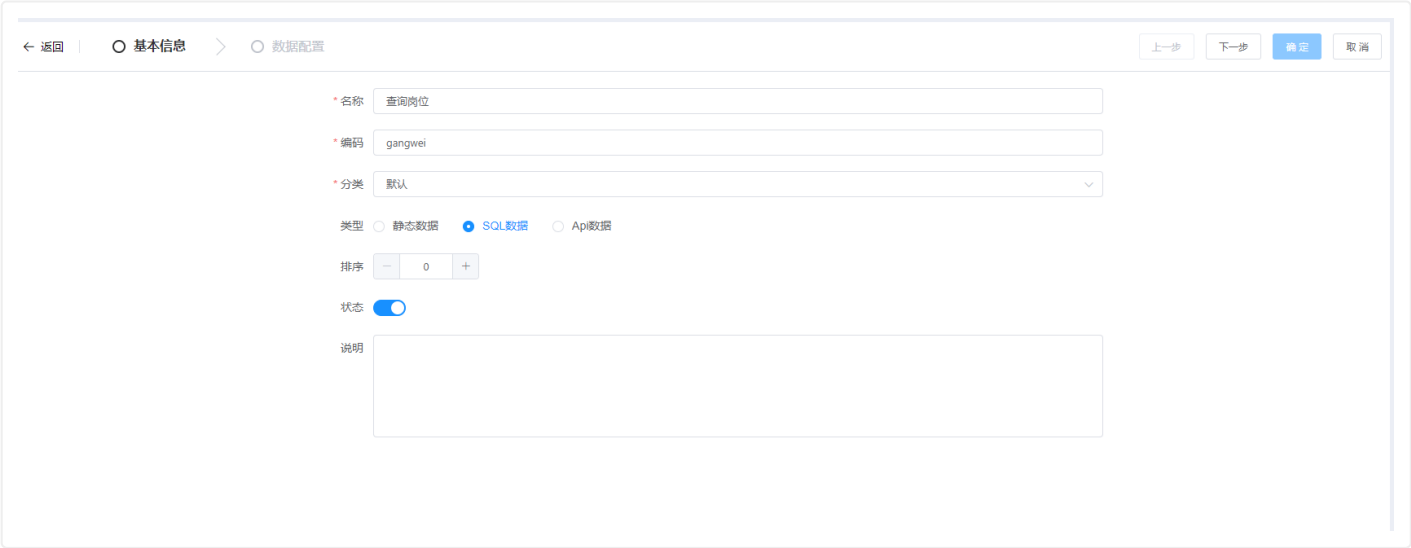
在系统管理目录下打开数据应用中数据接口，进入【数据接口】页面有查询、重置、刷新、新建、全屏等功能；如下图所示：



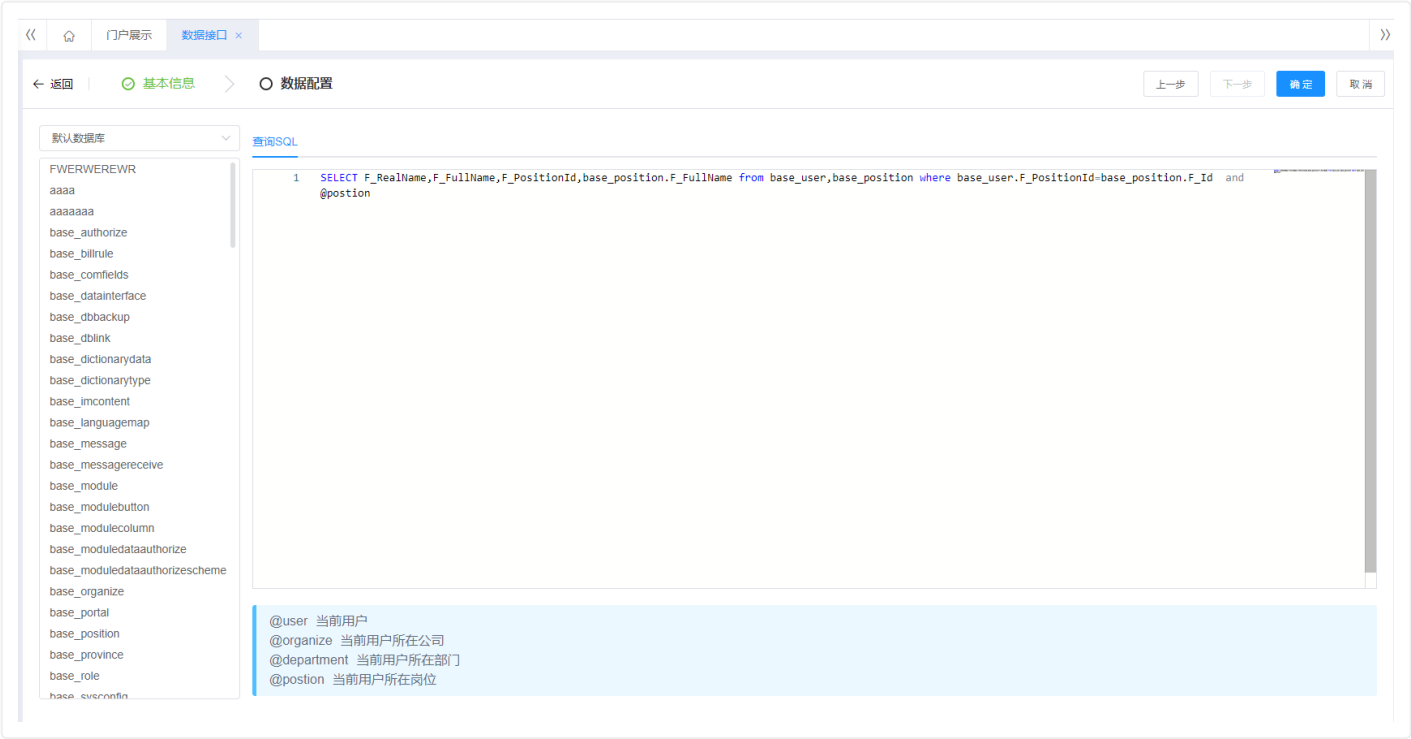
ⅰ刷新当前页面；

ⅱ在当前页面输入接口名称进行搜索；

ⅲ鼠标点击该页面左上角新建，进入新建数据接口页面，进行字段填写、红色星号表示必填项；基础信息填写一些功能基础信息后，确认点击“下一步”进行数据配置或者点击取消按钮离开该页面、当前数据未保存。如下图所示：



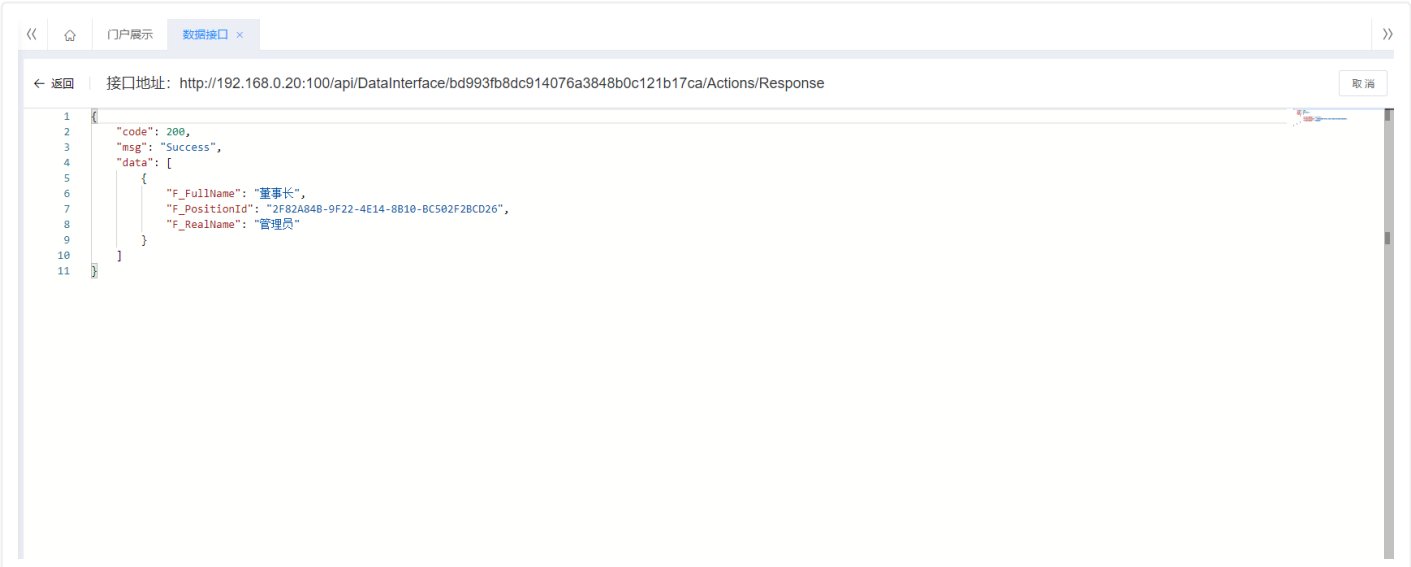
数据配置是根据用户需求编写SQL数据、静态数据、API数据，确认点击“上一步”进行基本信息修改或者点击取消按钮离开该页面、当前数据未保存。点击“确定”保存这个数据接口，如下图所示：



在数据接口页面可以对数据编辑、删除、更多（预览）功能等操作，如下图所示：

ⅰ点击“删除”按钮可以直接删除这个数据接口成功；

ⅱ操作更多下预览，预览数据接口效果；



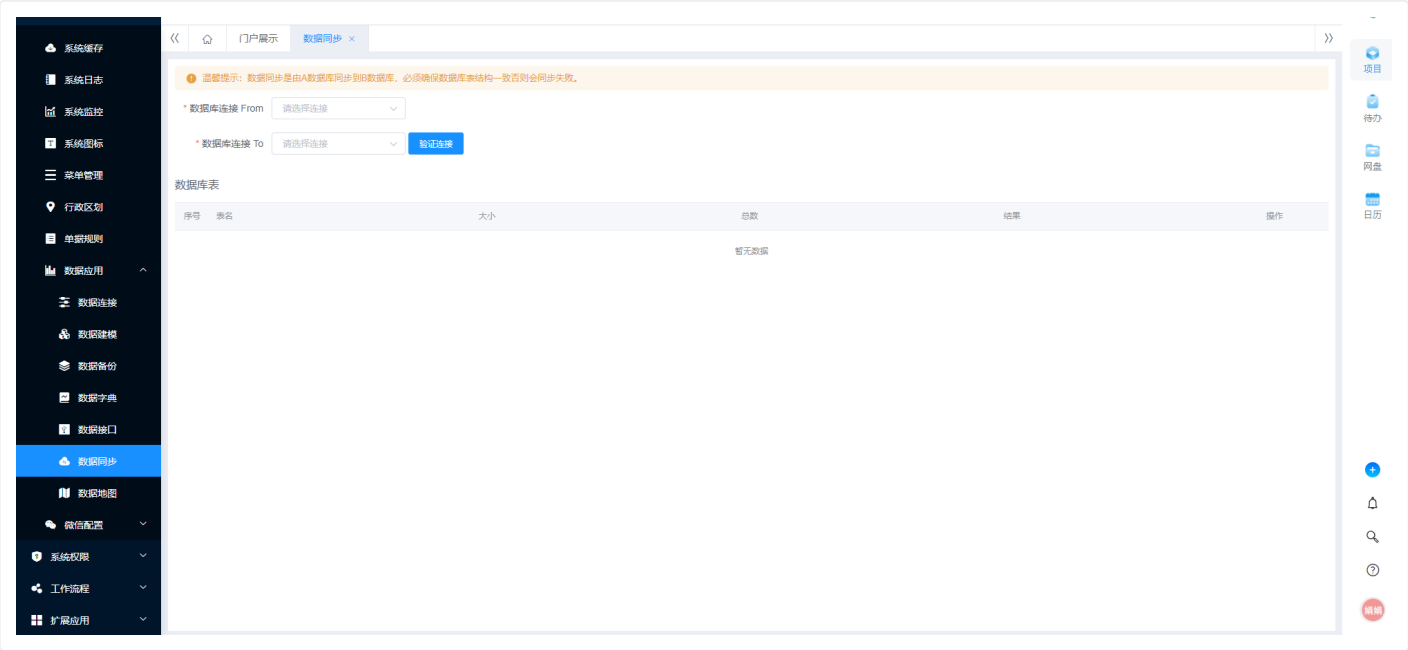
数据同步

功能描述

数据库结构一致的前提下，选择数据库连接From、数据库连接To操作验证连接功能，可将A数据库同步到B数据库

操作步骤

在系统管理目录下打开数据应用中数据同步，进入【数据同步】页面操作数据同步温馨提示：数据同步是由A数据库同步到B数据库，必须确保数据库表结构一致否则会同步失败。如下图所示：



连接数据类型有SqlServer、MySql、Qracle数据库；

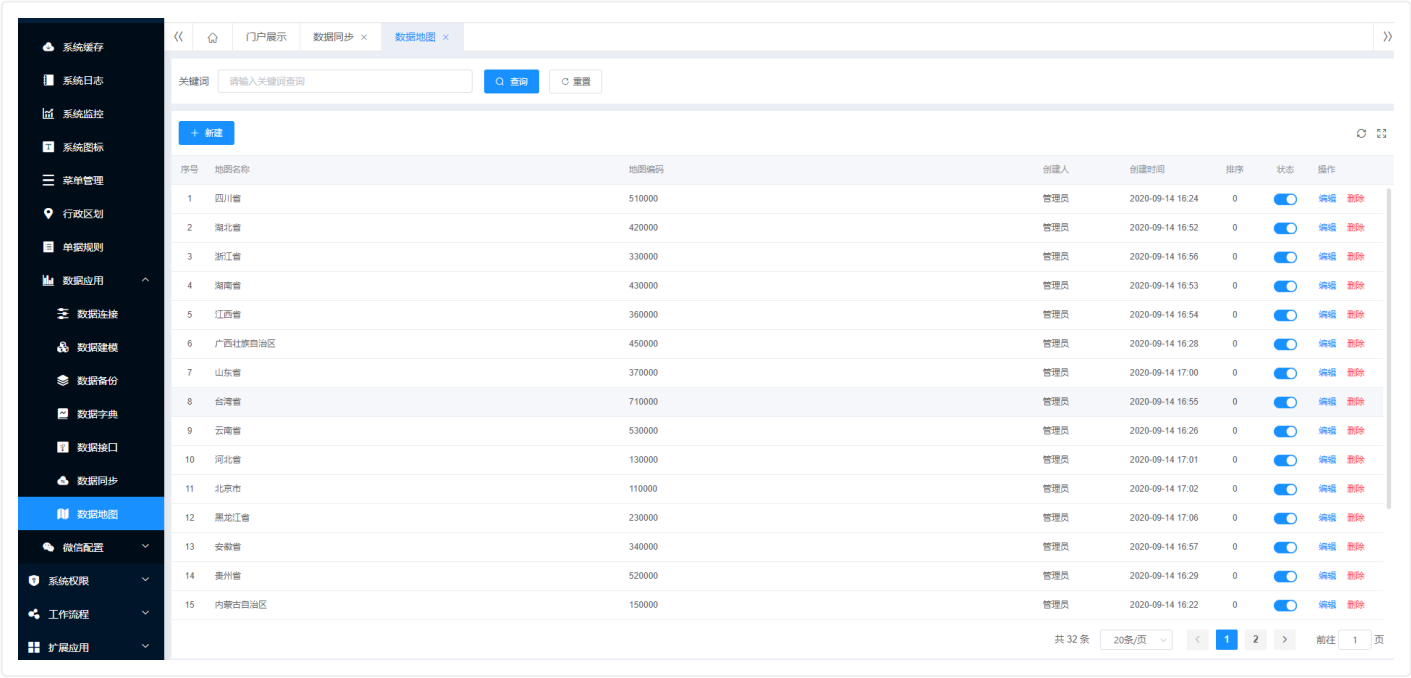
数据地图

功能描述

大屏设计引用数据地图的定义

操作步骤

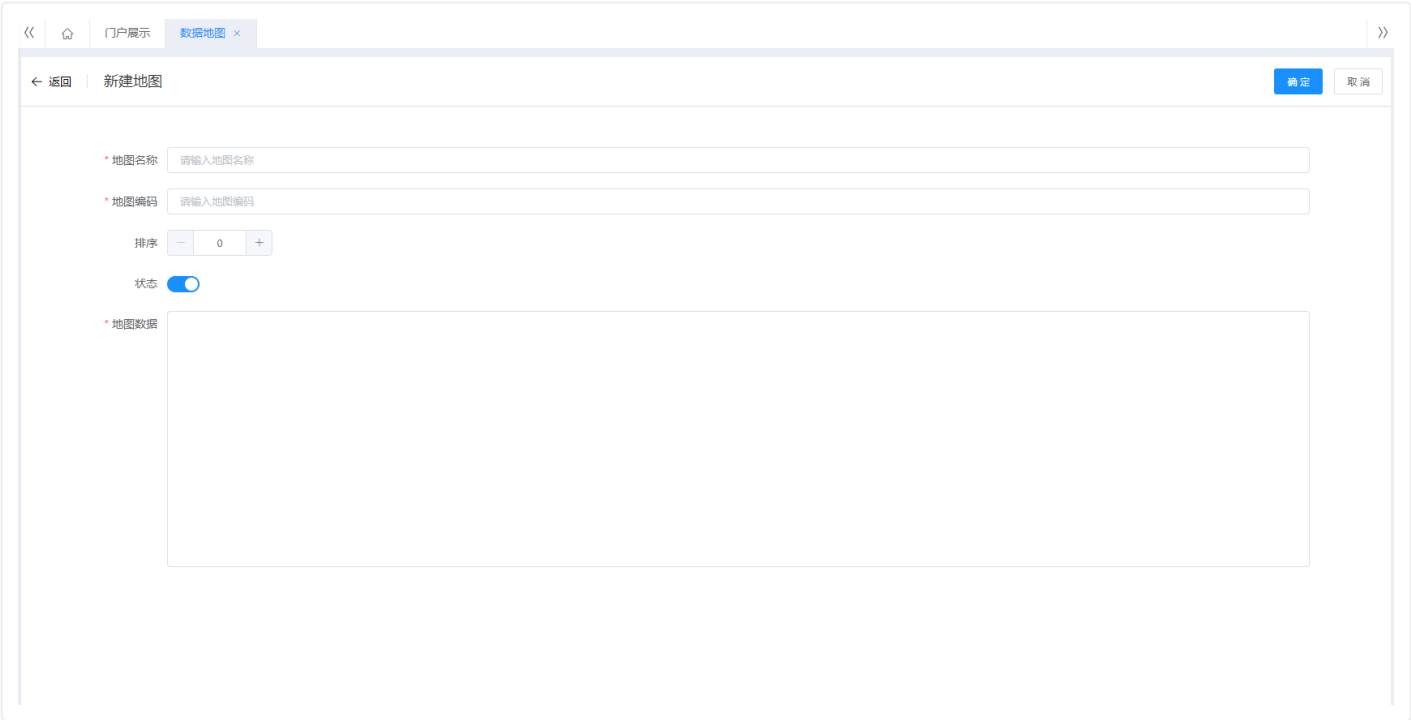
在系统管理目录下打开数据应用中数据地图，进入【数据地图】页面有查询、重置、刷新、新建模板、全屏等功能；如下图所示：



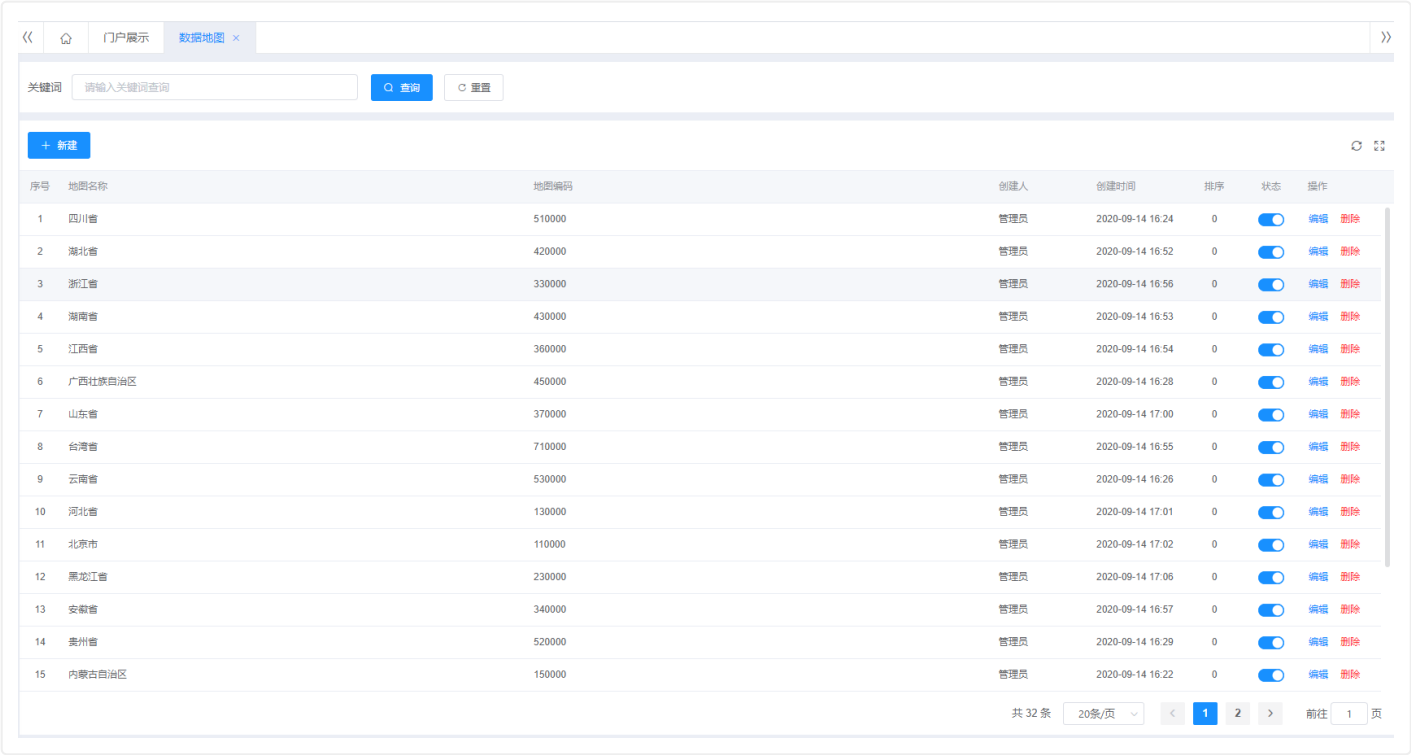
ⅰ刷新当前页面；

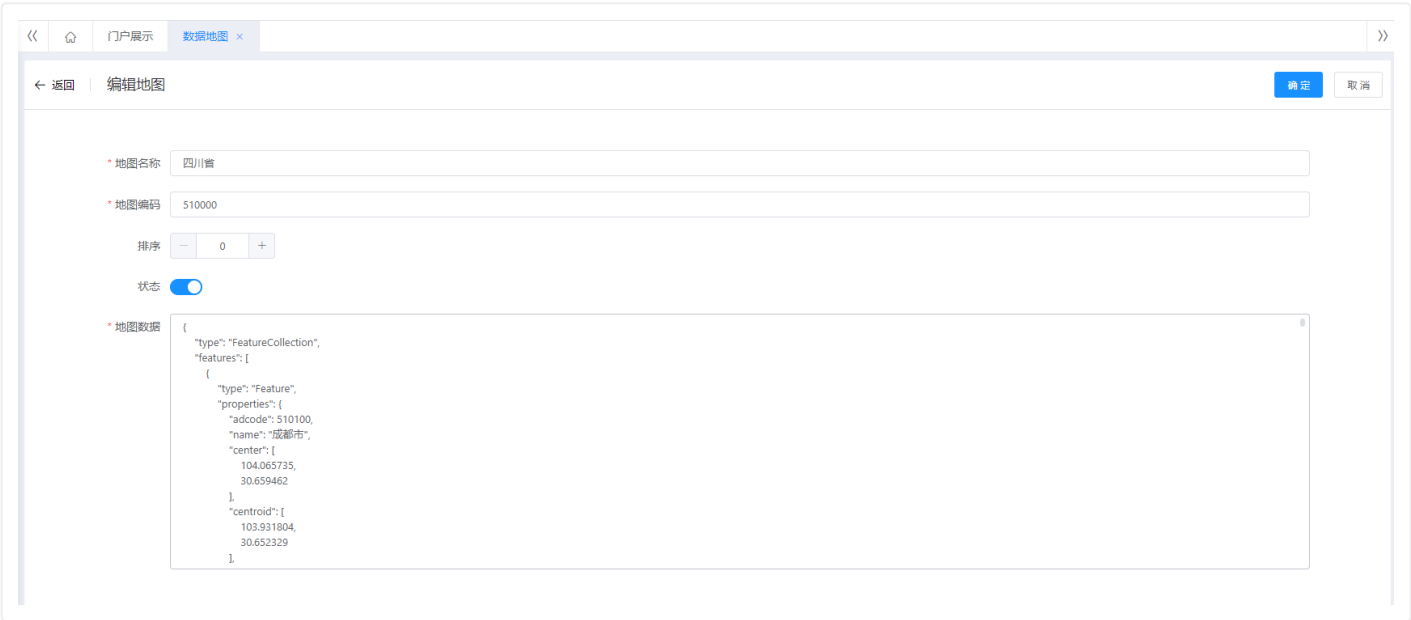
ⅱ在当前页面输入地图名称进行搜索；

ⅲ鼠标点击该页面左上角新建，进入新建数据地图页面，进行字段填写、红色星号表示必填项；点击确定按钮保存地图信息或者点击取消按钮离开该页面、当前数据未保存。如下图所示：



在数据地图页面可以对一条数据进行编辑、删除、启用或者停止等功能；如下图所示：


门户展示
数据地图
关键词
请输入关键词查询
查询
重置
新建
序号 地图名称 地图编码 创建人 创建时间 排序 状态 操作
1 四川省 510000 管理员 2020-09-14 16:24 0 编辑 删除
2 湖北省 420000 管理员 2020-09-14 16:52 0 编辑 删除
3 浙江省 330000 管理员 2020-09-14 16:56 0 编辑 删除
4 湖南省 430000 管理员 2020-09-14 16:53 0 编辑 删除
5 江西省 360000 管理员 2020-09-14 16:54 0 编辑 删除
6 广西壮族自治区 450000 管理员 2020-09-14 16:28 0 编辑 删除
7 山东省 370000 管理员 2020-09-14 17:00 0 编辑 删除
8 台湾省 710000 管理员 2020-09-14 16:55 0 编辑 删除
9 云南省 530000 管理员 2020-09-14 16:26 0 编辑 删除
10 河北省 130000 管理员 2020-09-14 17:01 0 编辑 删除
11 北京市 110000 管理员 2020-09-14 17:02 0 编辑 删除
12 黑龙江省 230000 管理员 2020-09-14 17:06 0 编辑 删除
13 安徽省 340000 管理员 2020-09-14 16:57 0 编辑 删除
14 贵州省 520000 管理员 2020-09-14 16:29 0 编辑 删除
15 内蒙古自治区 150000 管理员 2020-09-14 16:22 0 编辑 删除
共 32 条
20条/页
前往
页


门户展示
数据地图
返回
编辑地图
确定
取消
地图名称
四川省
地图编码
510000
排序
状态
地图数据
{
"type": "FeatureCollection",
"features": [
{
"type": "Feature",
"properties": {
"adcode": 510100,
"name": "成都市",
"center": [
104.065735,
30.659462
],
"centroid": [
103.931804,
30.652329
],


提示
此操作将永久删除该数据, 是否继续?
取消
确定

微信配置

公众号配置

公众号菜单

公众号用户

公众号消息

公众号素材

企业号配置

企业号组织

企业号用户

企业号消息

系统权限

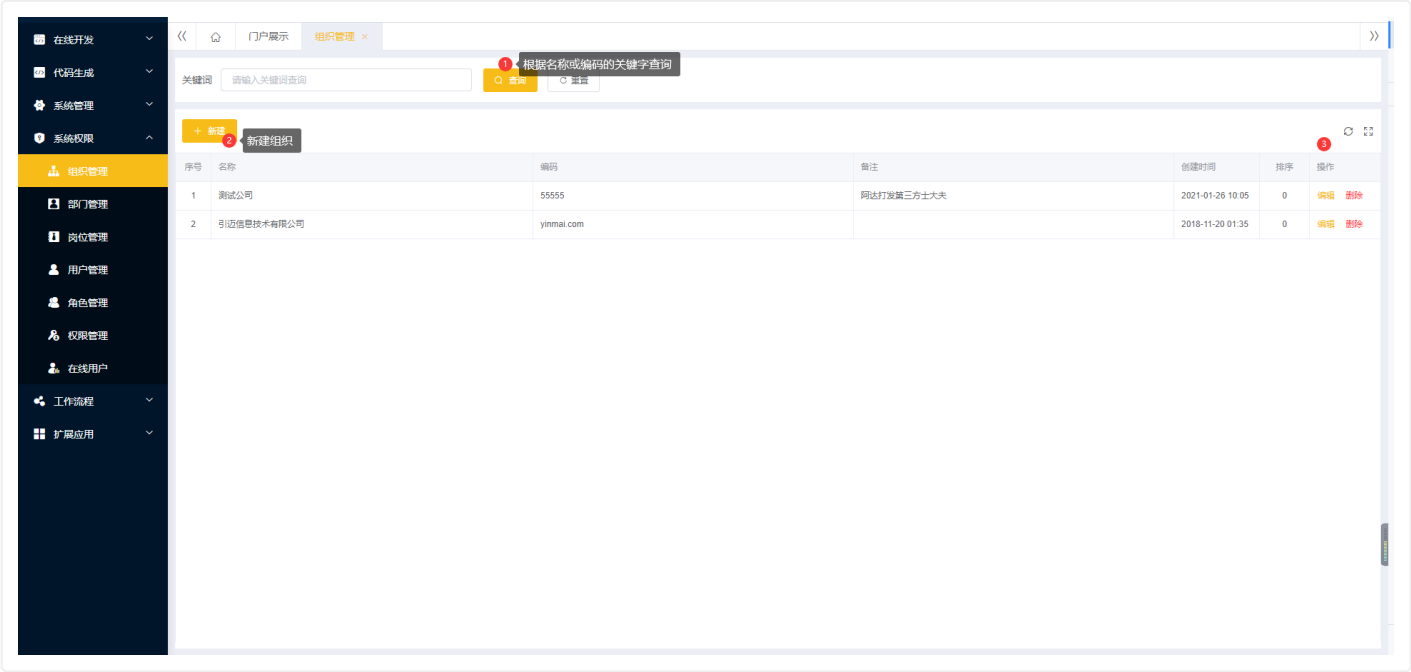
组织管理

功能描述

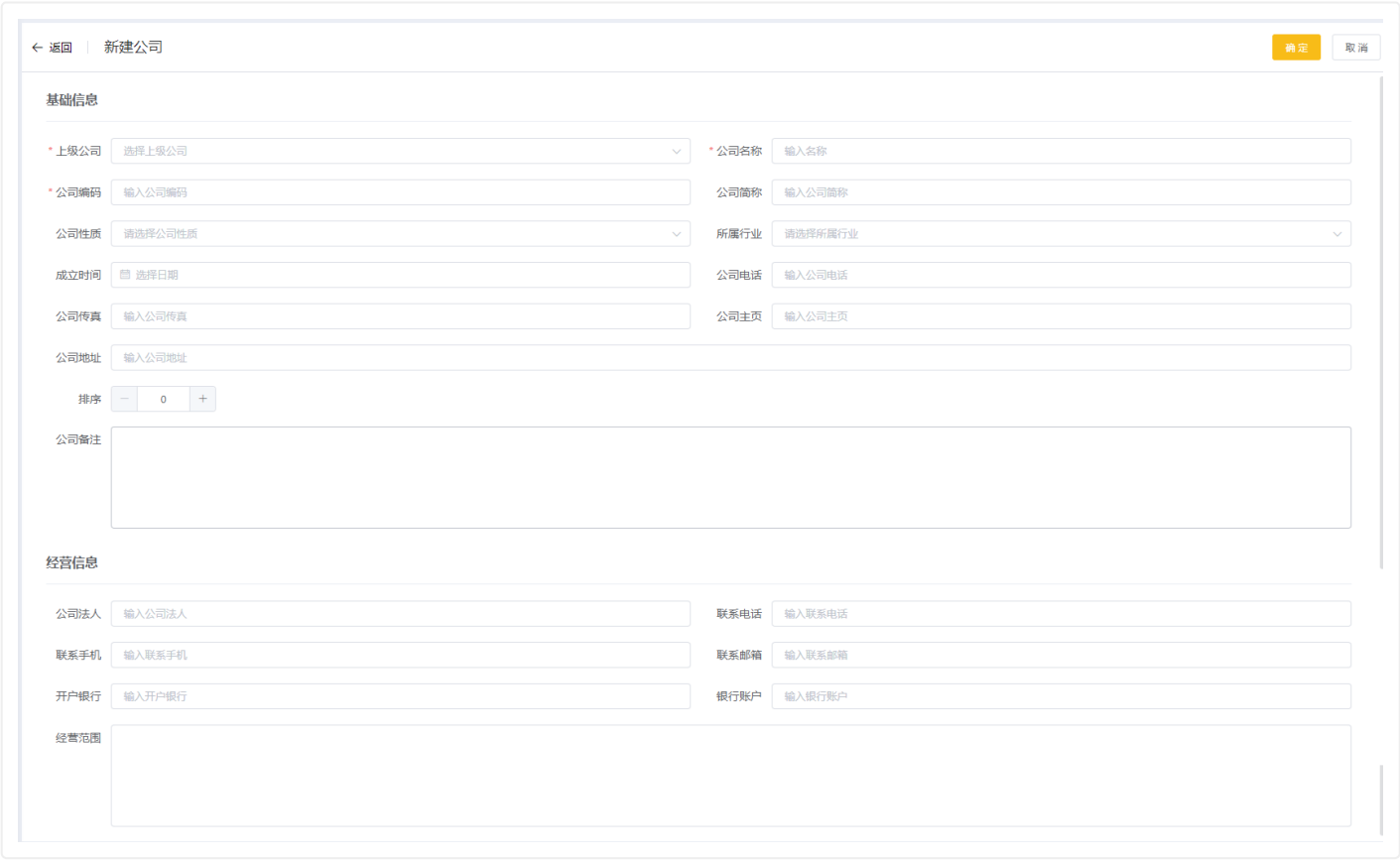
支持多租户Saas架构设计分析，创建多个公司，可以对组织机构进行管理

操作步骤

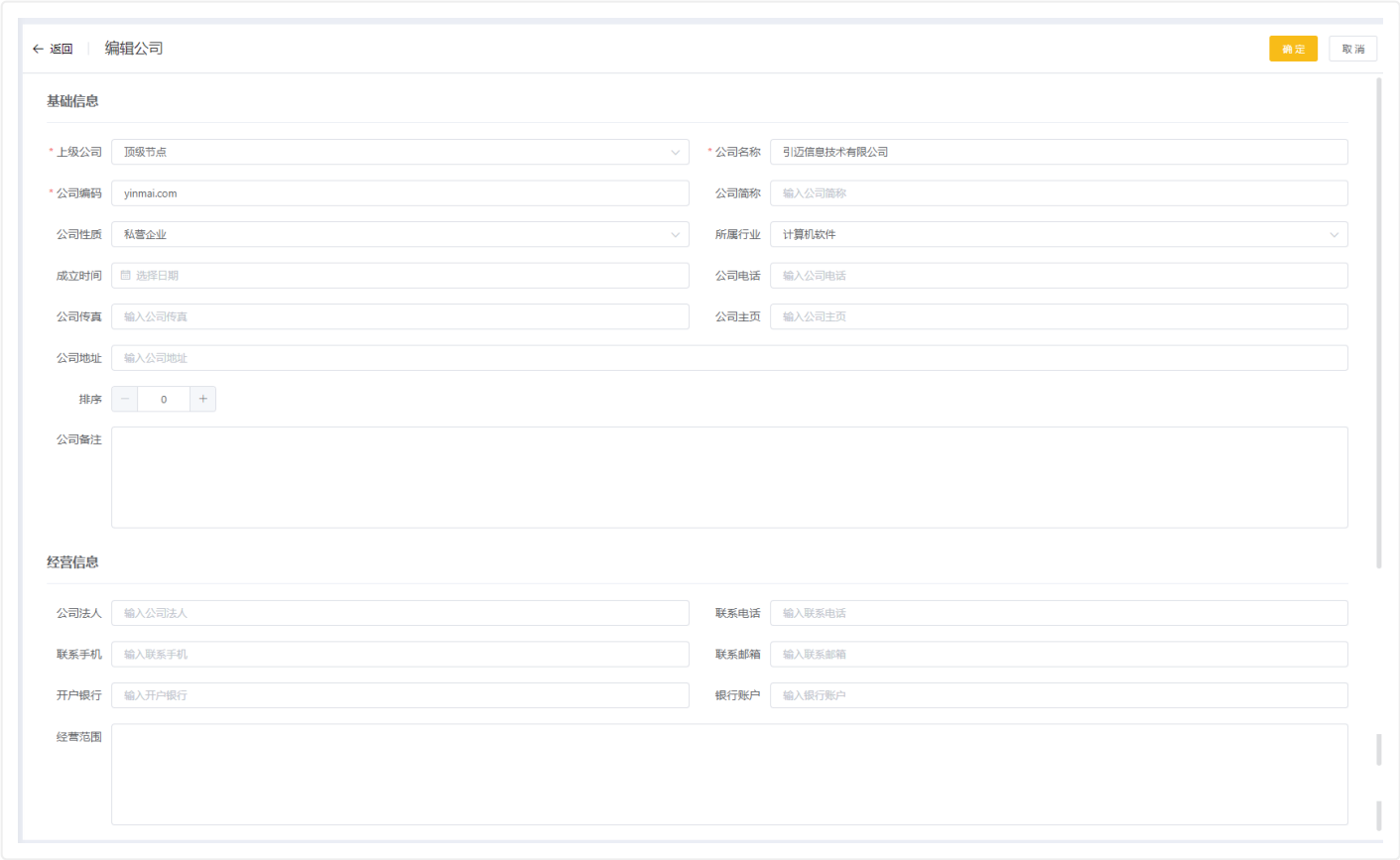
在系统权限目录下操作在组织管理，进入【组织管理】页面查询、重置、刷新、全屏、新建功能。如下图所示：



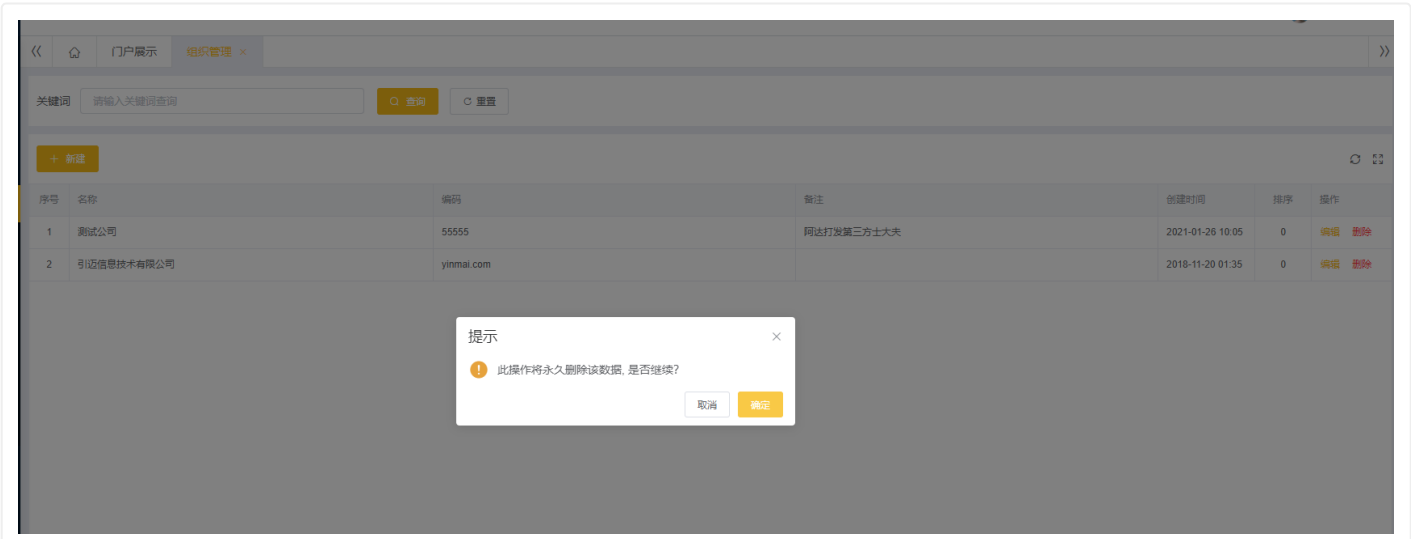
新建公司：基础信息模块必填字段有上级公司、编号、性质、名称、简称、所属行业，其它模块表示完善信息有成立时间、传真号码、企业电话、主业、地址、备注、以及企业状态等。经营信息模块填写企业经营信息分别有企业法人、开户银行、银行账户、联系电话、手机、邮箱、经营范围等。以上信息填写完毕点击确认键表示保存新建公司信息、在组织管理页面可查看，或者取消键离开该页面、数据未被保存。如下图所示：



在组织管理页面点击编辑按钮。进入编辑当前页面修改内容，如果修改完毕需要保存点击确定键，当数据保存成功在组织管理页面显示，如果不需要保存点击取消键、离开编辑页面。



在组织管理页面点击编辑按钮，会提示“此操作将永久删除该数据，是否继续？”,如果继续点击确认键、执行删除语句，在组织列表页面该条数据不存在。如果不删除点击取消按钮，离开删除提示页面、组织列表数据依然存在。如果组织下已经有关联部门，该组织就不能删除。



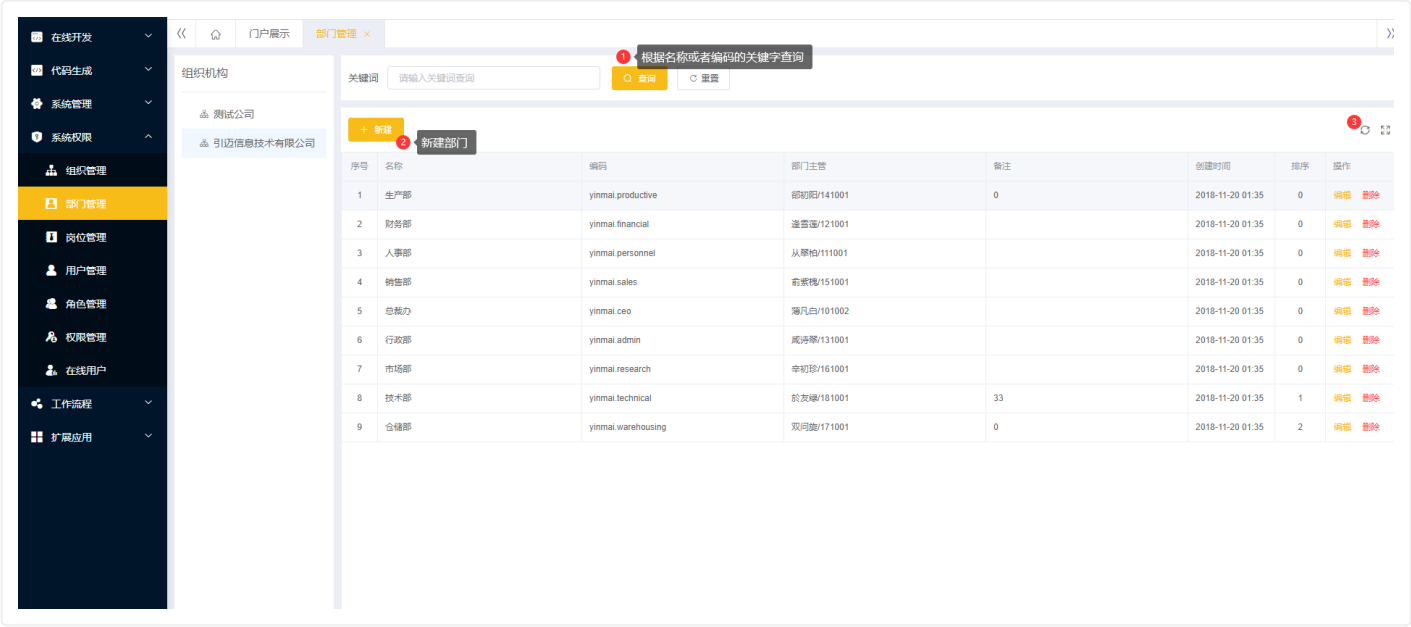
部门管理

功能描述

新增部门时关联对应组织机构，部门管理通过树形显示。

操作步骤

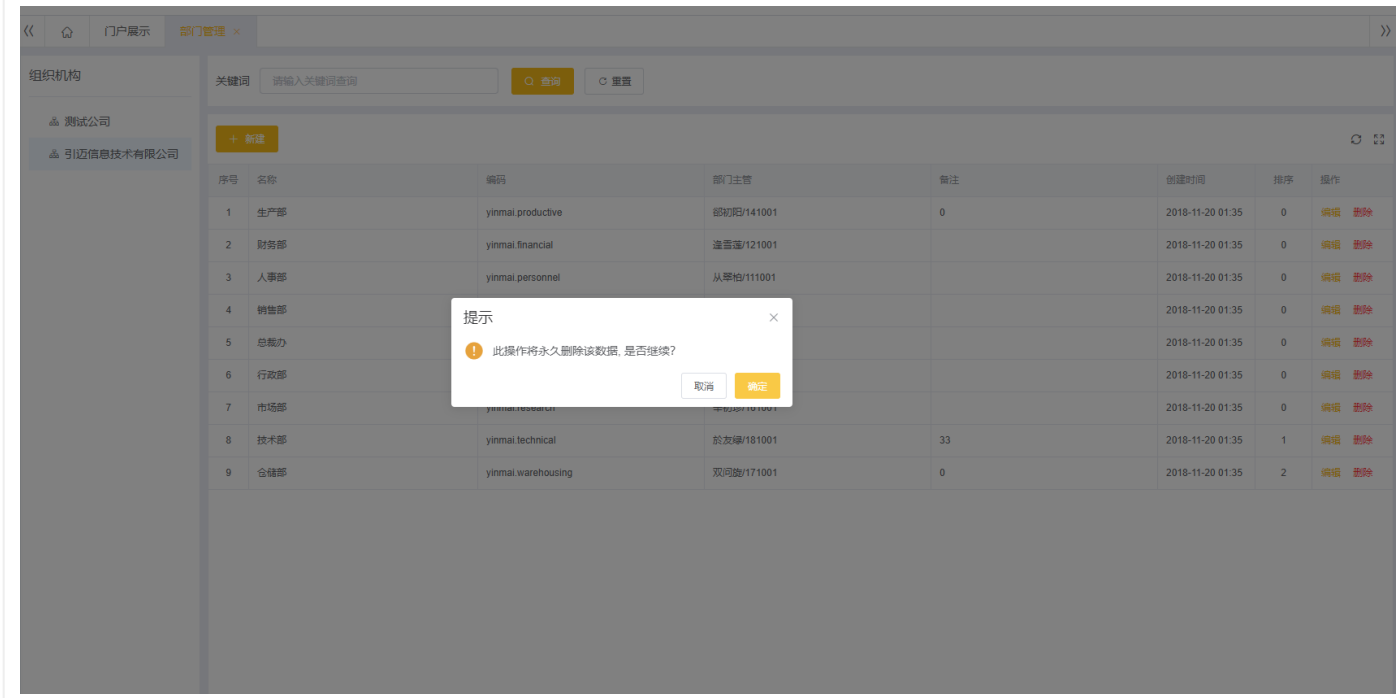
在系统权限目录下操作在部门管理，进入【部门管理】页面查询、重置、刷新、全屏、新建功能。如下图所示：



创建部门，进入新建部门页面填写表格当中需要填写创建部门信息有部门上级、名称、编码、部门经理、状态、说明等资料方便查看。如下图所示：

在部门管理页面点击编辑按钮。进入编辑当前页面修改内容，如果修改完毕需要保存点击确定键，当数据保存成功在组织管理页面显示，如果不需要保存点击取消键、离开编辑页面。

在部门管理页面点击删除按钮提示“此操作将永久删除该数据，是否继续？”，如果继续点击确认键、执行删除语句，在部门列表页面该条数据不存在。如果不删除点击取消按钮，离开删除提示页面部门列表数据依然存在。



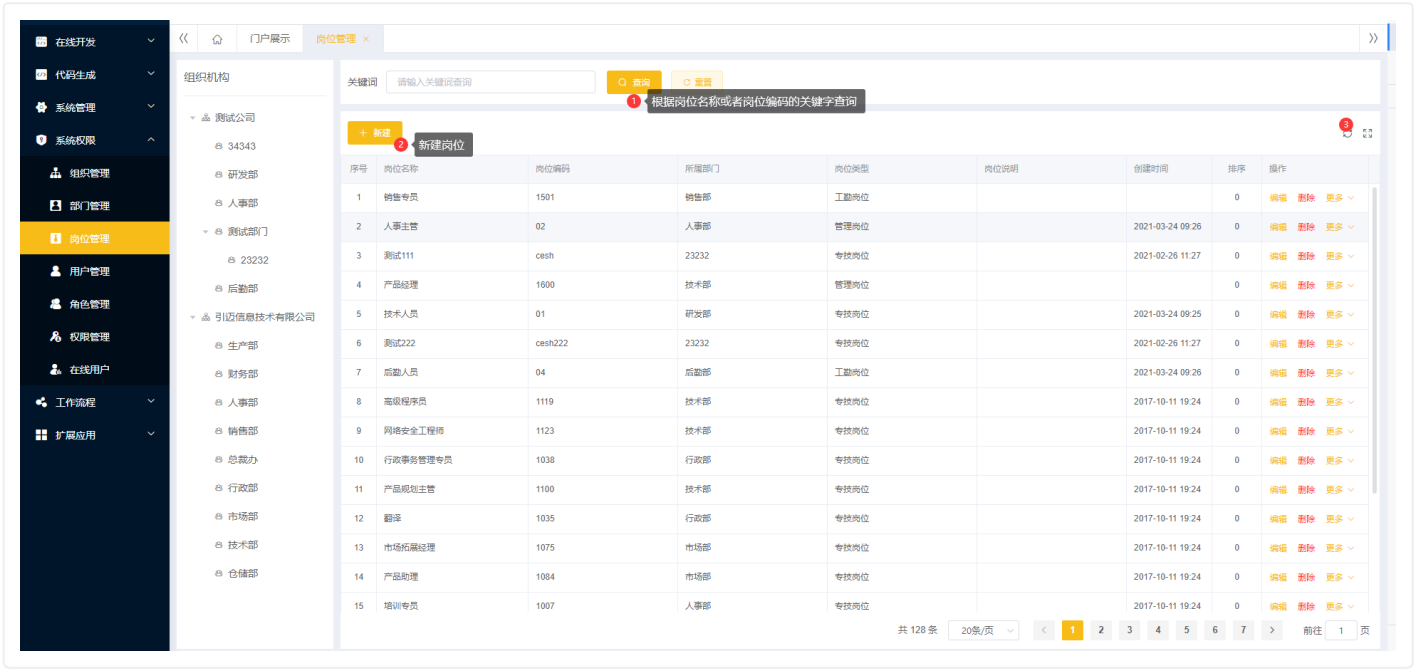
岗位管理

功能描述

新增岗位时关联部门，通过多层级联的方式展示岗位数据，岗位设置明确各岗位职责。

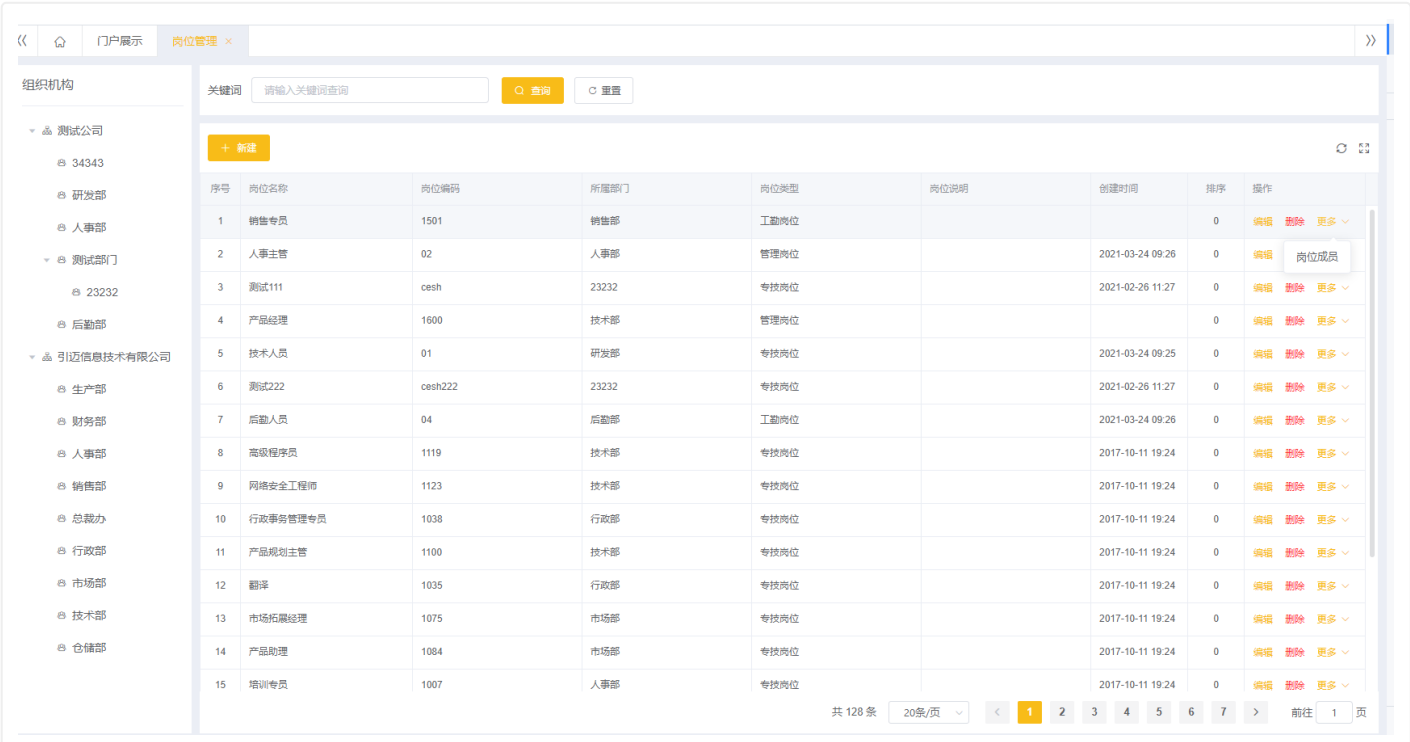
操作步骤

在系统权限目录下操作在岗位管理，进入【岗位管理】页面查询、重置、刷新、全屏、新建功能。如下图所示：



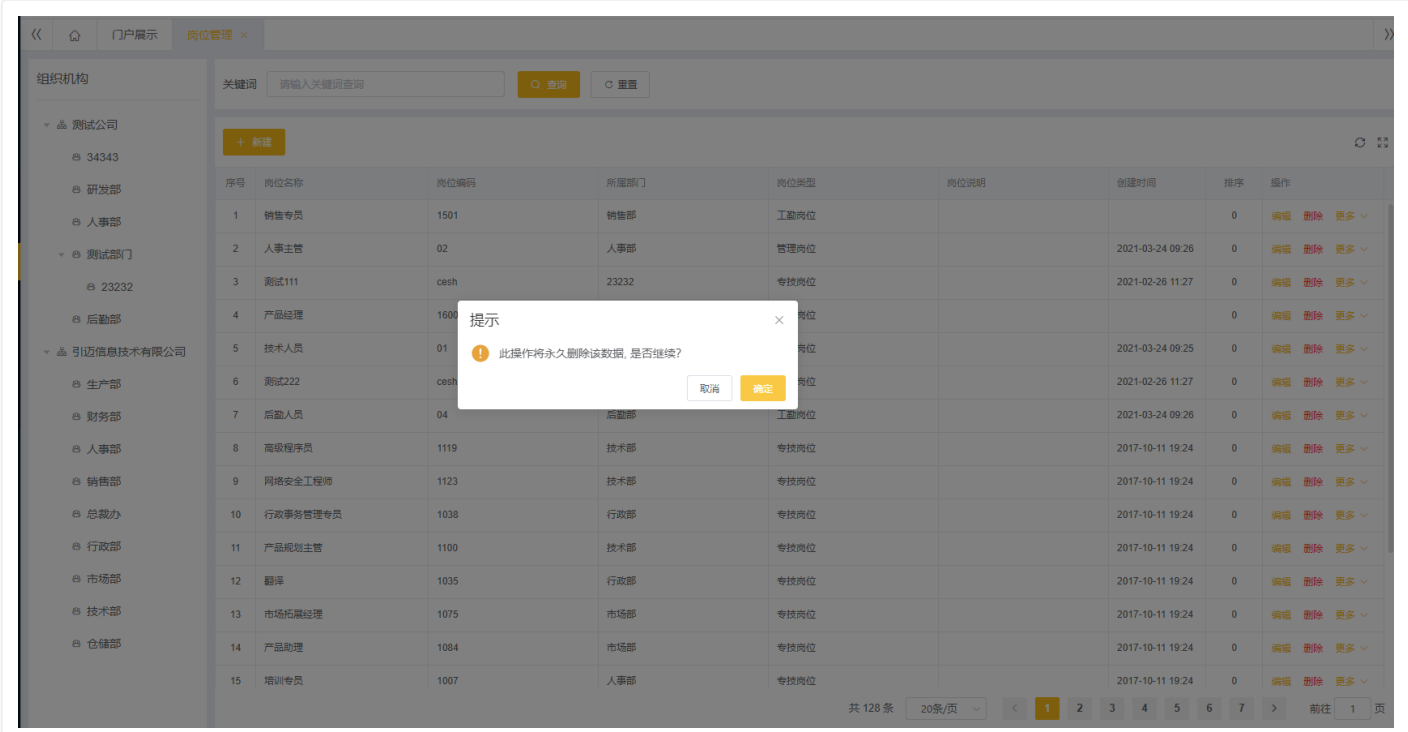
岗位管理页面点击新建按钮，进入新建岗位页面、下拉框选择所属机构、岗位类型，请填写岗位名称、编码、备注，选择岗位状态等信息填写完毕。操作确定键保存当前数据在岗位管理中显示，或者操作取消键离开该页面，数据未被保存。如下图所示：

岗位管理页面有编辑、删除、更多（岗位成员）等功能。如下图所示：



点击编辑按钮，进入编辑岗位页面、修改列表中的内容，操作确定按钮保存当前修改数据，操作取消按钮离开该页面放弃保存。如下图所示：

点击删除按钮提示“此操作将永久删除该数据，是否继续？”，如果继续点击确认执行删除语句，在岗位管理页面该条数据不存在。如果不删除点击取消按钮，离开删除提示页面、岗位管理页面数据依然存在。



岗位成员功能：在岗位管理页面点击更多选择岗位成员、进入岗位成员管理页面，显示搜索、删除、清空列表、确

定功能。如下图所示：

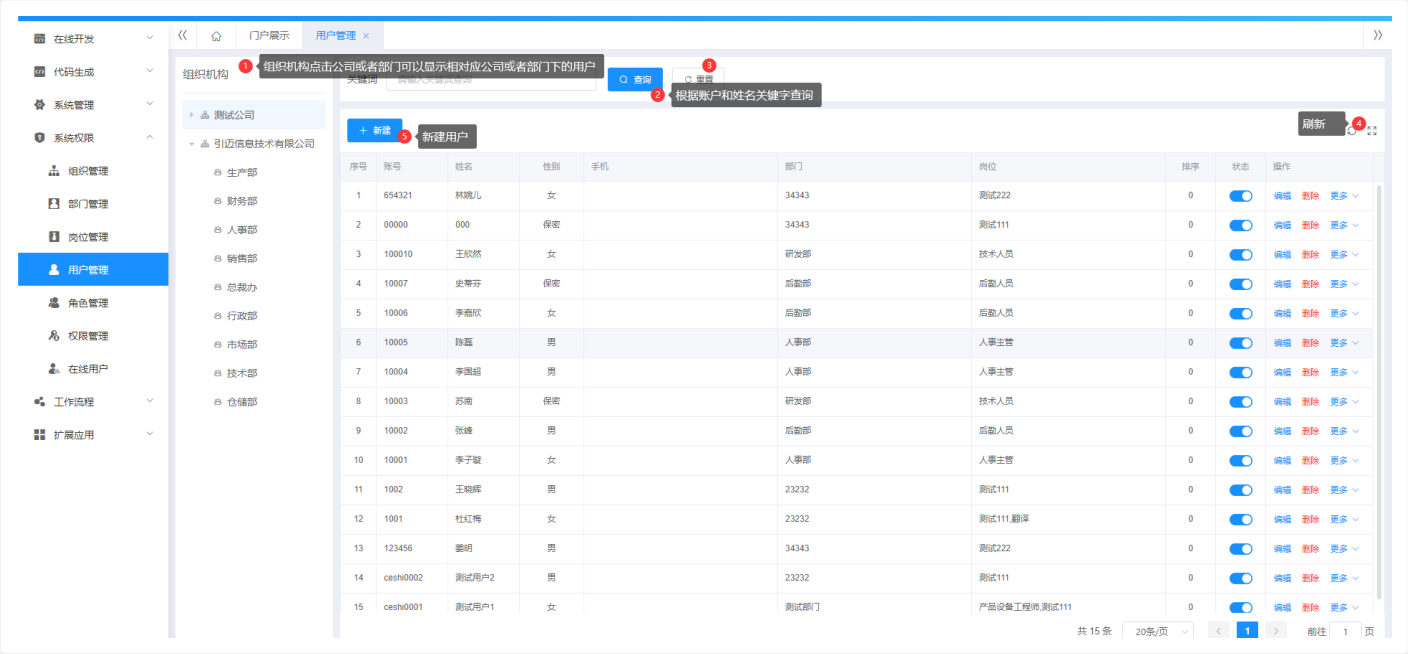
用户管理

功能描述

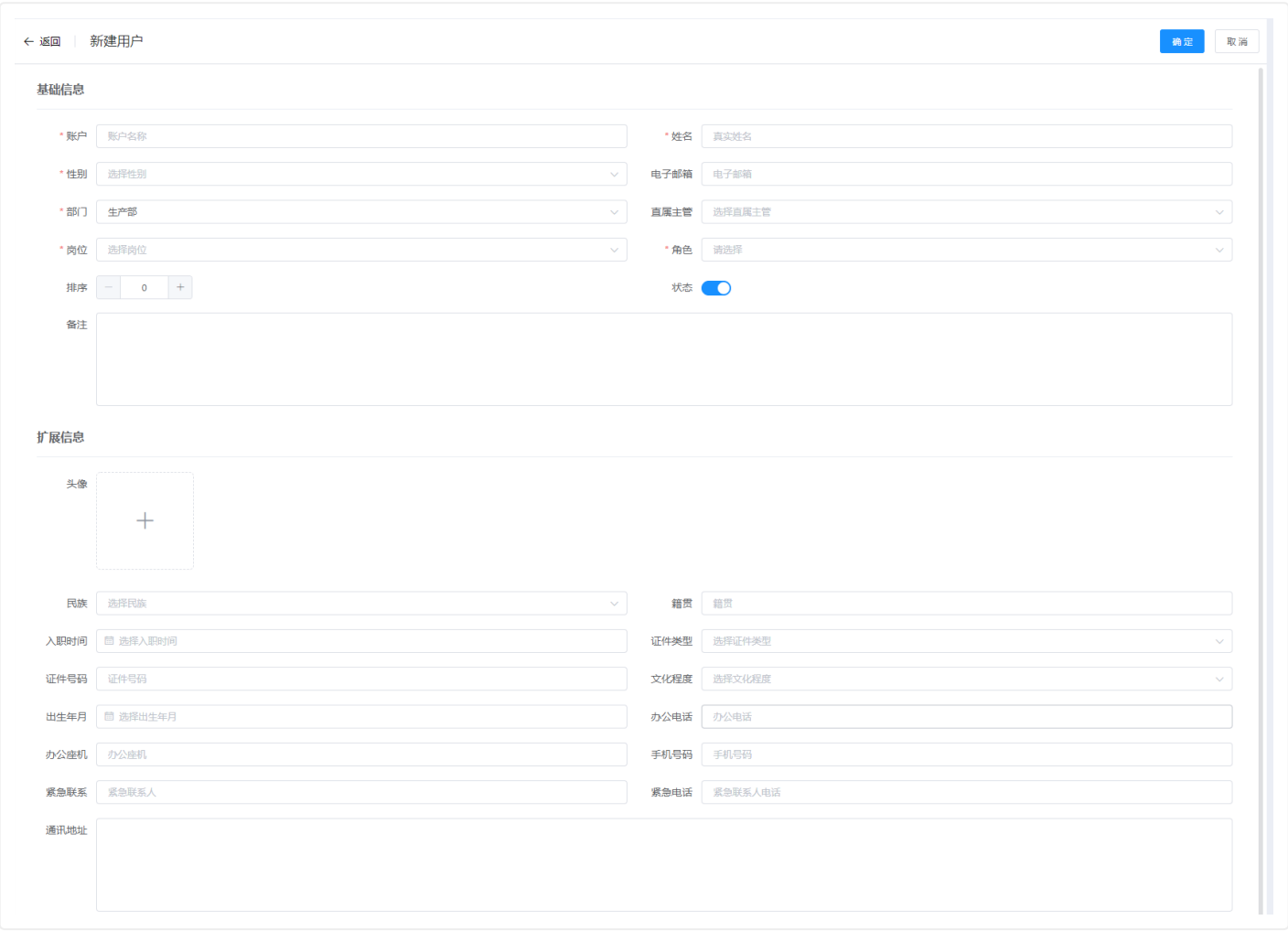
主要是功能系统的使用者，新增用户需维护用户基本信息，并关联对应的部门及岗位

操作步骤

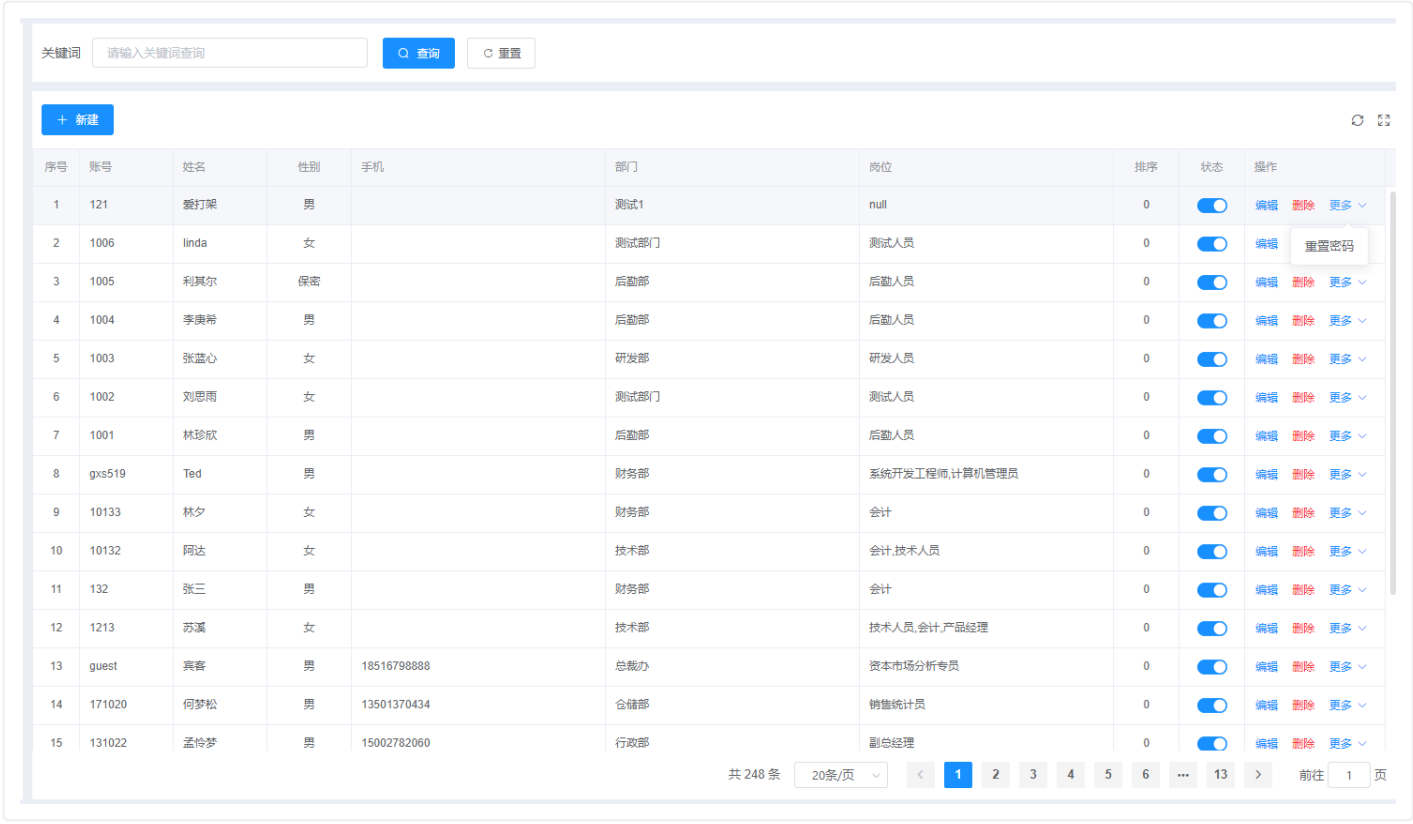
在系统权限目录下操作在用户管理，进入【用户管理】页面是由组织机构里面是一个树形结构、可以收缩和展开，用户列表页面有查询、重置、刷新、全屏、新建等功能。如下图所示：



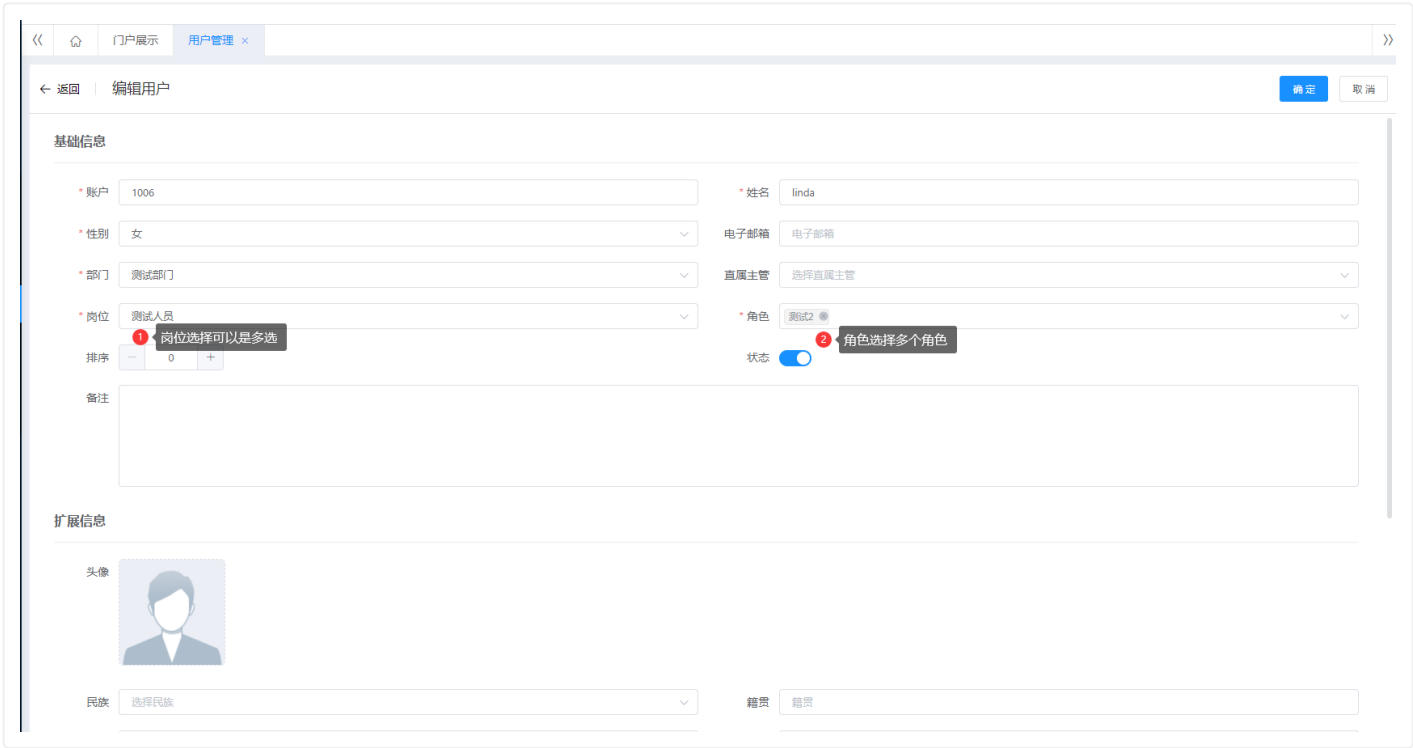
在组织机构中选择部门，给这个部门新增用户成员，在新建用户页面分为基本信息和扩展信息栏，新建用户基础信息 账户、姓名、性别、部门、直属主管、岗位、角色、备注，扩展信息 头像、民族、籍贯、入职日期、证件类型、证件号码、文化程度、出生年月等等。填写完点击确定表示保存成功或者取消按钮离开该有面、数据未被保存。如下图所示：



在用户列表页面，可以进行编辑、删除、用户状态修改，更多（重置密码限设置）功能操作。如下图所示：
执行删除时、当前数据没有被应用时可直接删除成功，如果当前数据被应用后就不能删除成功、会提示当前数据被应用不能删除。

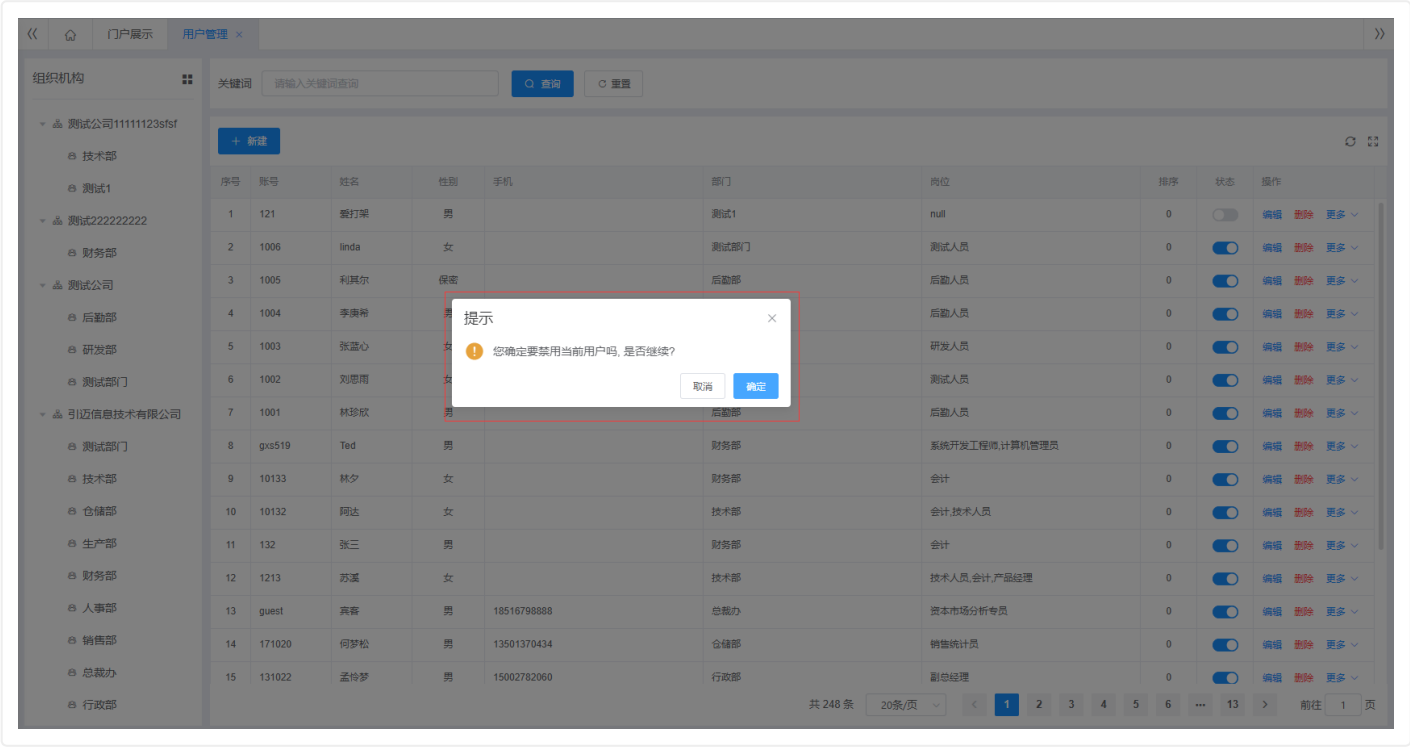


点击编辑按钮，进入编辑用户功能页面，修改当前用户信息完，点击确定键保存该条数据，操作取消按钮离开该页面、修改的内容未保存。如下图所示：



点击更多下重置密码按钮，进入重置密码页面、请重复输入两次新密码，操作确定按钮保存当前重置新密码，或者点击取消按钮离开该页面，当前数据未被保存。如下图所示：

点击状态列表的按钮进行账户启用或者停用操作，对用户操作控制后状态栏中按钮蓝色表示正常、按钮灰色表示停用，注意：用户状态正常状态下该用户账号可以登录使用，如果是在停用状态下该用户不能登录。如下图所示：



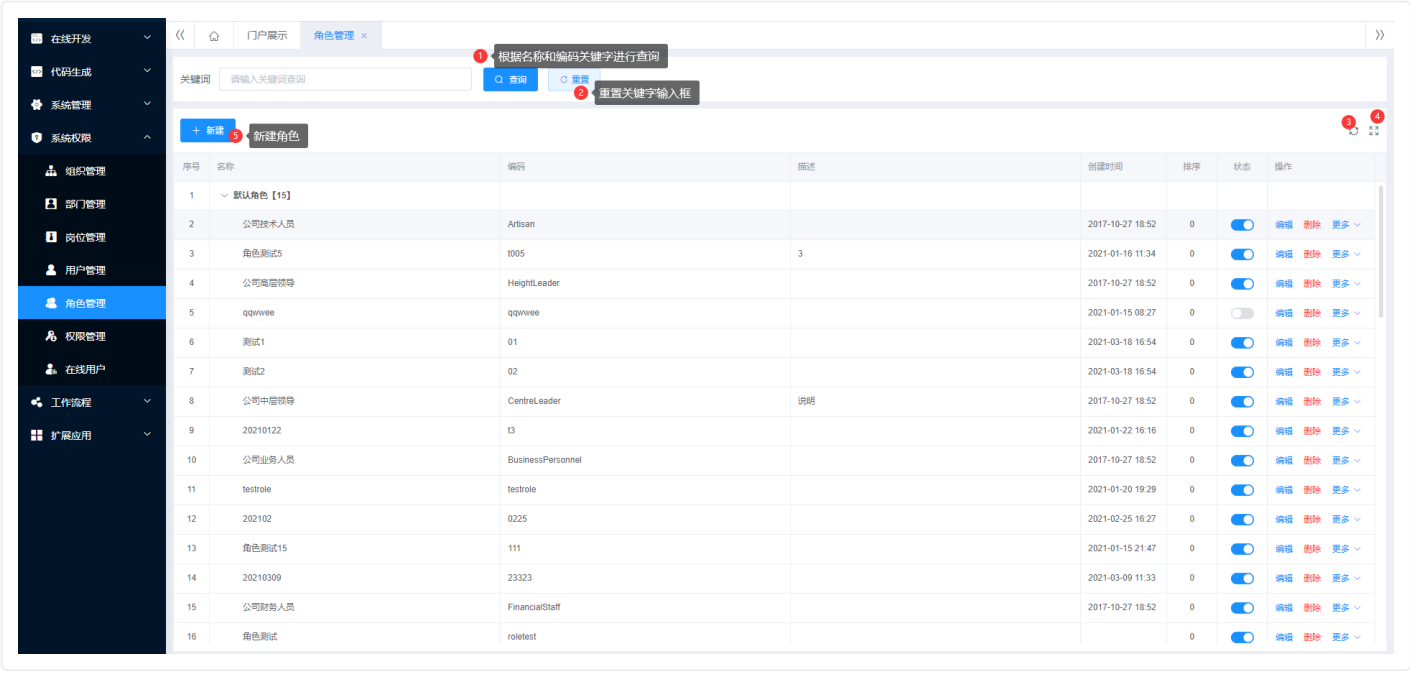
角色管理

功能描述

基于业务管理需求而预先在系统中设定好的角色标签，多角色权限叠加。每个角色对应明确的系统权限，其所拥有的系统权限一般不会随意更改，并且角色也不会随着用户的被添加和被移除而进行改变，相较于用户管理而言更加稳定

操作步骤

在系统权限目录下操作在角色管理，进入【角色管理】页面查询、重置、刷新、全屏、新建角色功能。如下图所示：

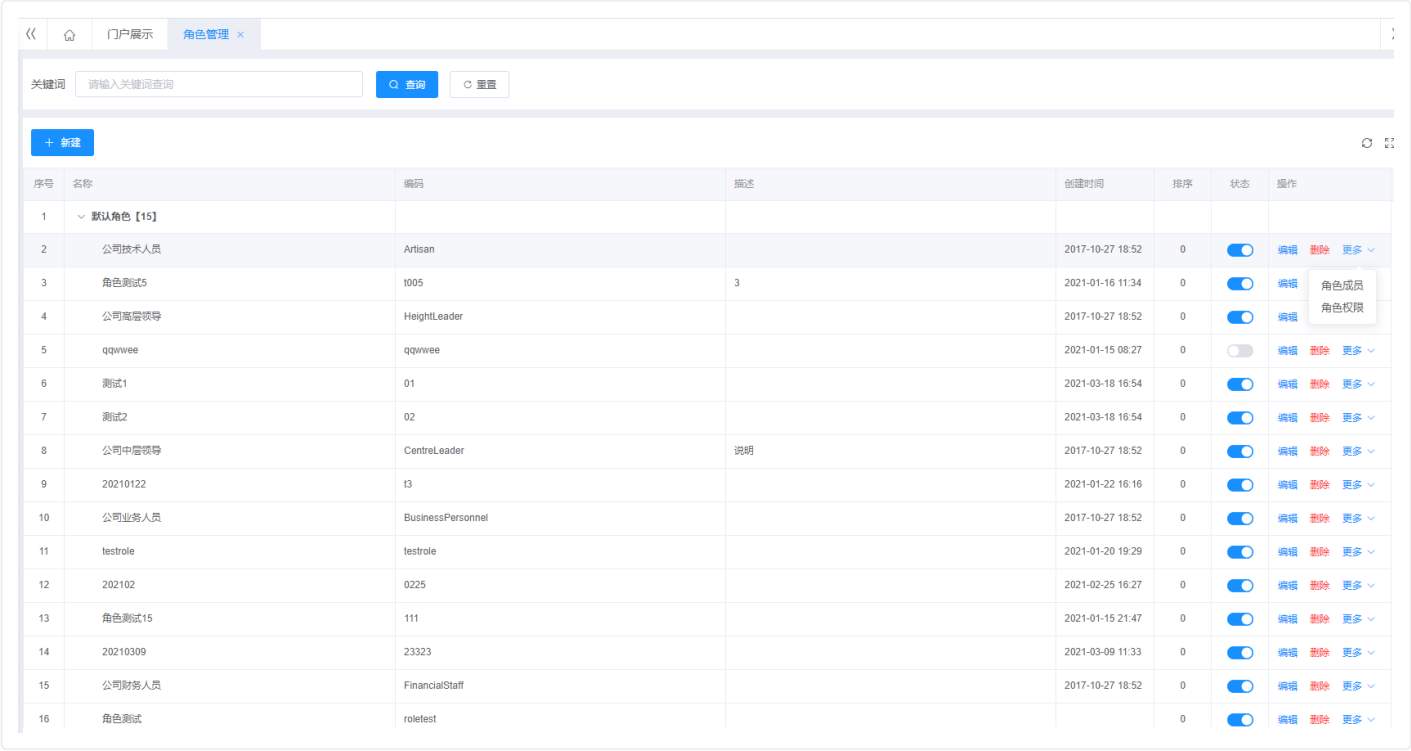


点击新建按钮进入新建角色页面，新建角色内容有角色名称、角色编码、角色类型、角色状态（no/open）、角色说明信息，填写完点击确定表示保存当前数据在角色管理页面显示，或者取消离开当前页面、数据未被保存。如下图所示：

在角色管理页面有编辑、删除、以及更多（角色成员、角色权限）功能可操作。如下图所示：

ⅰ在角色管理页面点击编辑按钮，进入编辑页面修改完数据点击确定表示保存当前数据、或者点击取消表示离开该页面当前数据未被保存；

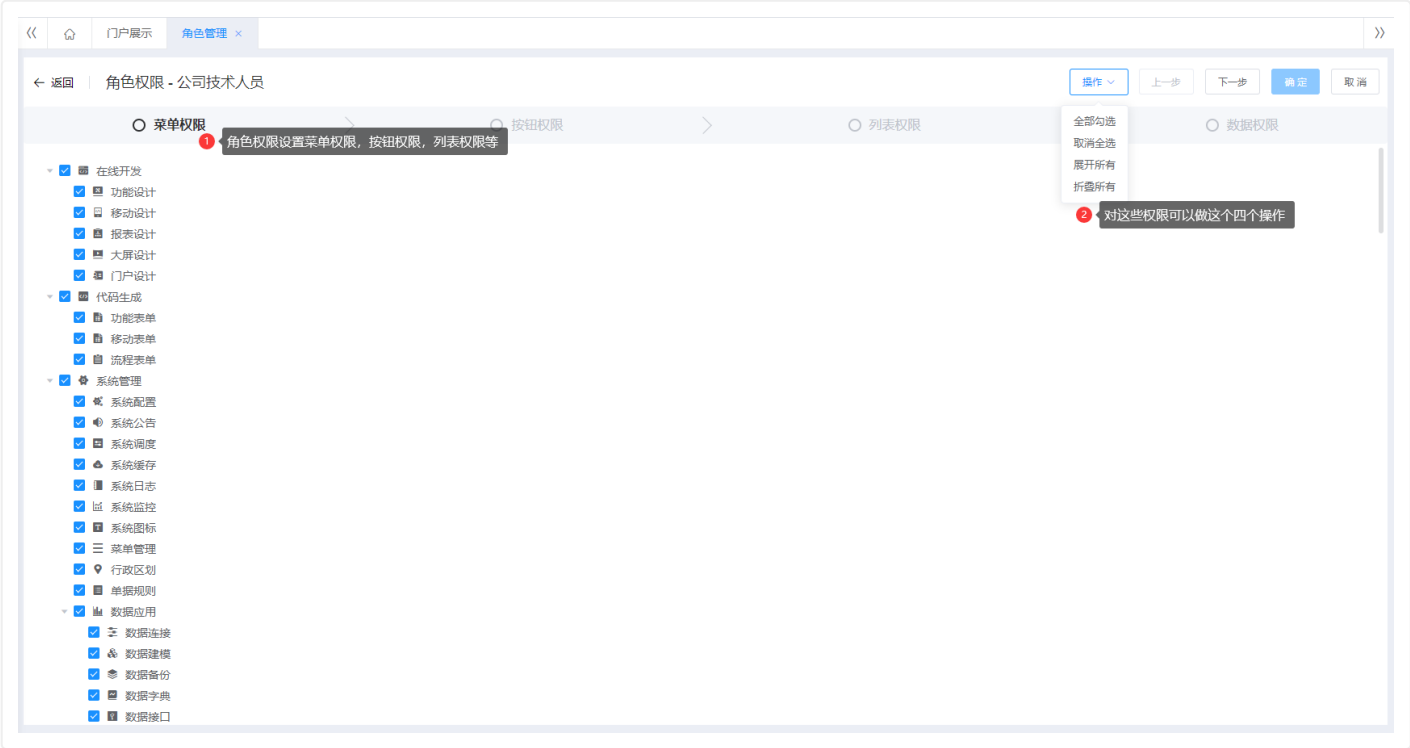
ⅱ在角色管理页面点击删除按钮，当这条数据未被其它应用可直接删除、如果被应用就不能够直接删除；



在角色管理页面点击更多下面角色成员进行设置，在角色成员管理页面有添加成员、搜索、删除等功能，搜索在角色成员管理页面请输入账户或姓名进行搜索。如下图所示：

在角色管理页面点击更多下面角色权限进行设置，在角色权限页面对角色进行菜单权限、按钮权限、列表权限、数据权限等详细操作分配，勾选平台下菜单权限表示分配给当前角色权限直到下一步……可以操作确定按钮保存时表

示完成当前角色权限设置。如果关闭角色权限页面操作取消按钮；如下图所示：



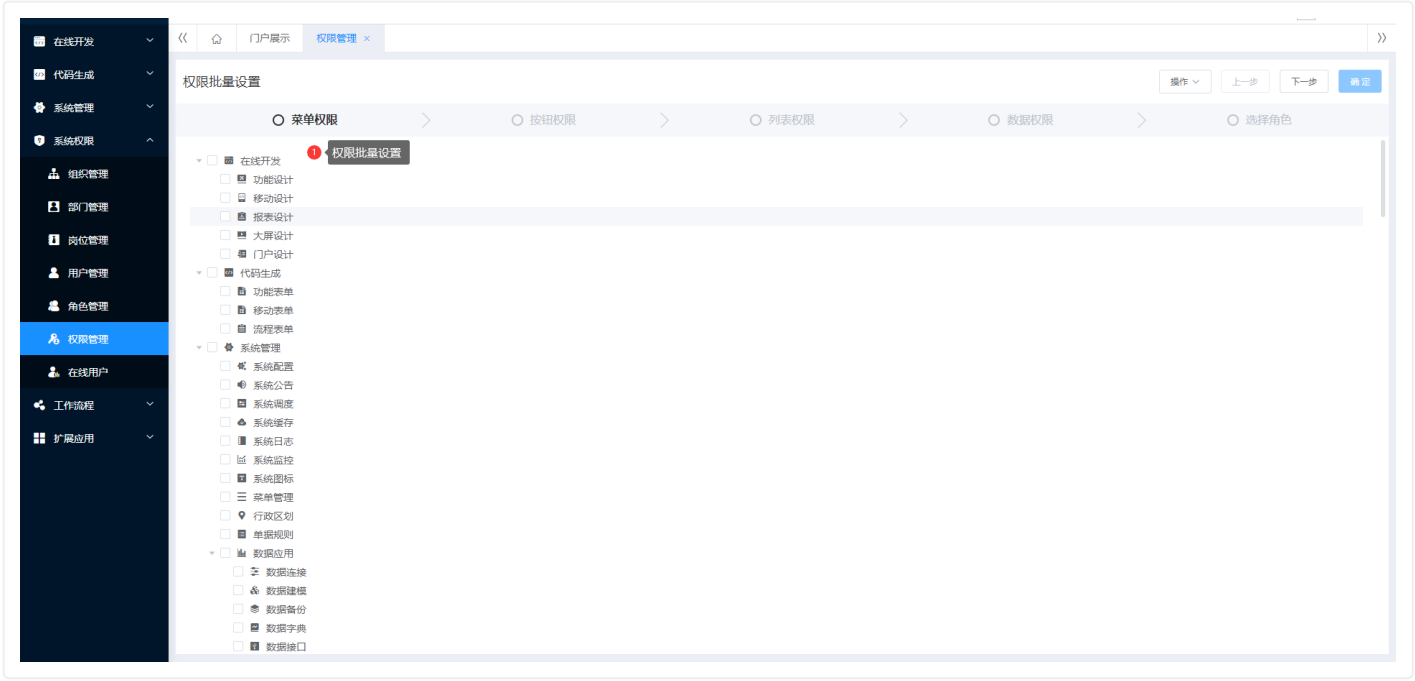
权限管理

功能描述

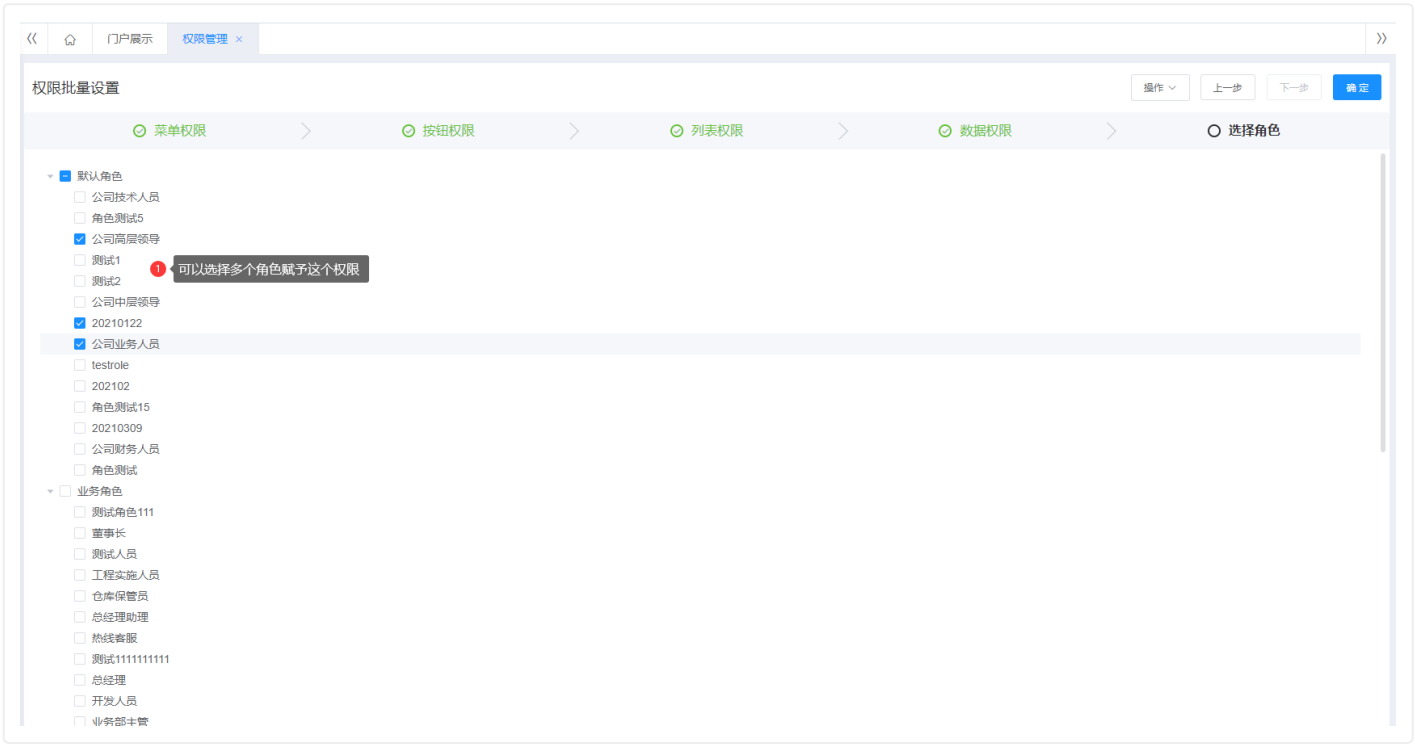
可从菜单权限、按钮权限、列表权限、数据权限等四个不同颗粒度等级来授予单个或者多个角色权限；当某一功能权限授权给用户时，也就相当于为该用户开通了可以操作某个目标功能的许可权

操作步骤

在系统权限目录下操作权限管理，进入【权限管理】页面批量设置分配权限页面，选择菜单权限、按钮权限、列表权限、数据权限等勾选操作，当数据勾选完点击确定按钮表示保存当前权限设置数据，或取消按钮表示离开当前页面、但是数据未被保存；如下图所示：



在权限管理页面操作批量设置进入批量分配权限页面，在选择角色可以多选。如下图所示：



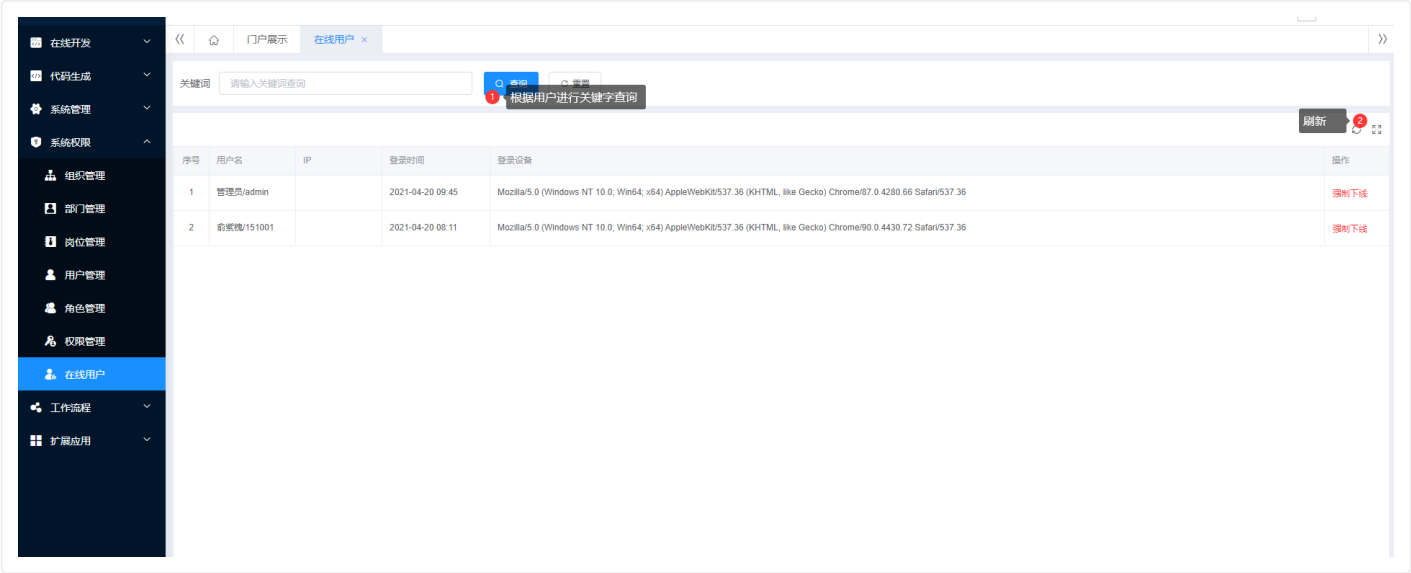
在线用户

功能描述

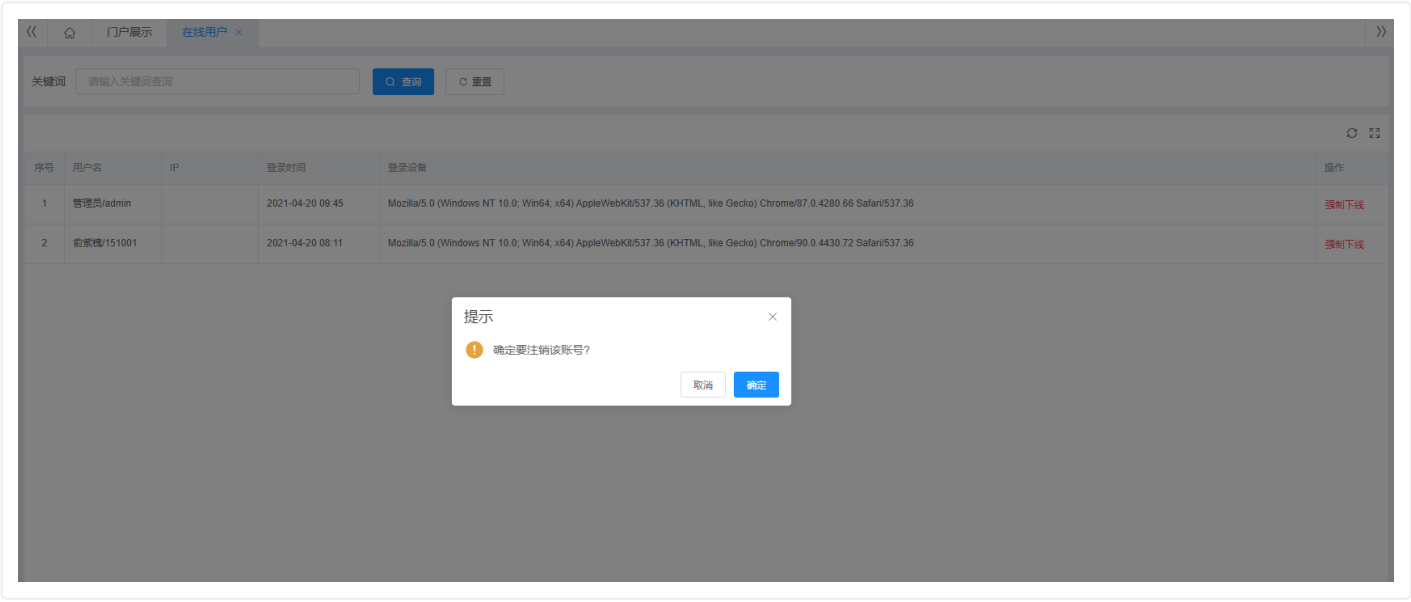
对登录进来的用户进行监督管理；对该在线用户查看IP地址、登录时间等等，以及要求强制下线操作

操作步骤

在系统权限目录下操作在线用户，进入【在线用户】页面有查询、刷新等功能。如下图所示：



点击强制下线按钮，当操作强制下线按钮提示“确定要注销该账户吗？”，如果继续就点击确定按钮、该在线用户强制下线，在线用户页面信息自动消失、对方登录的当前用户自动退出。如果操作取消按钮自动关闭系统提示框，在线用户信息依然存在。



工作流程

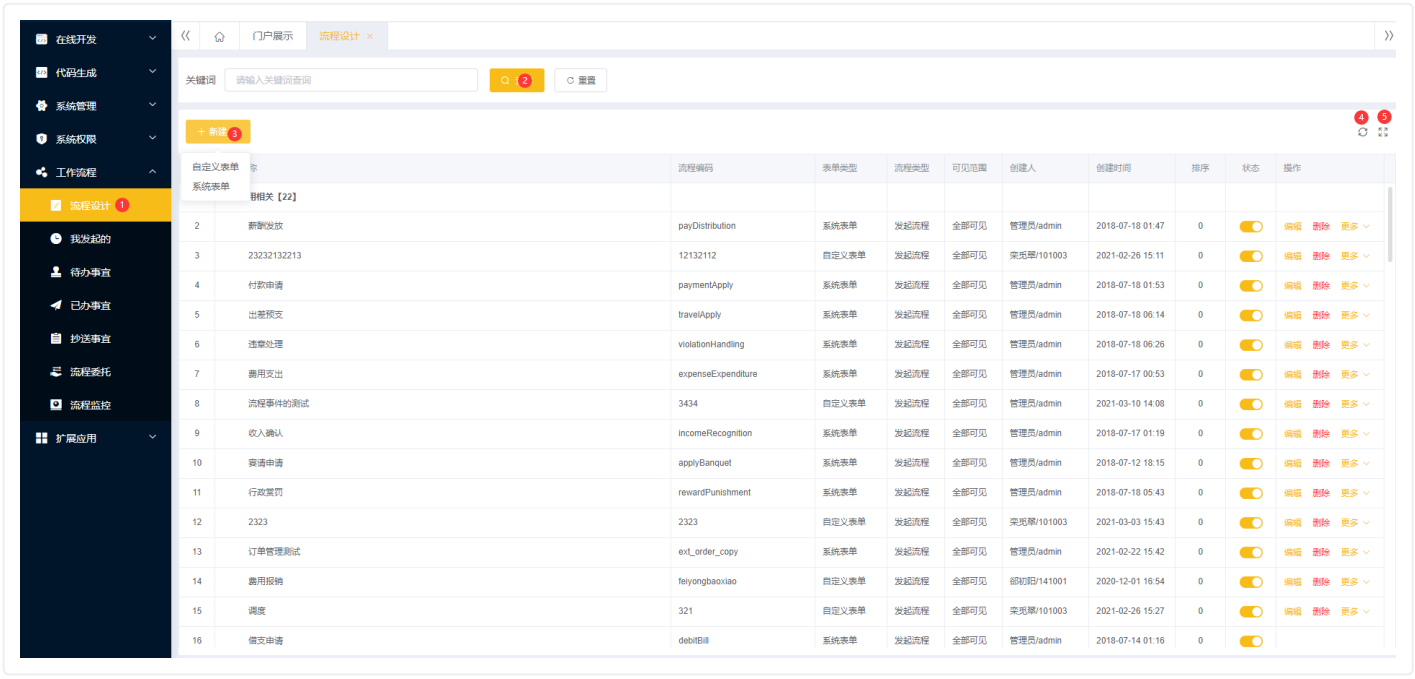
流程设计

功能描述

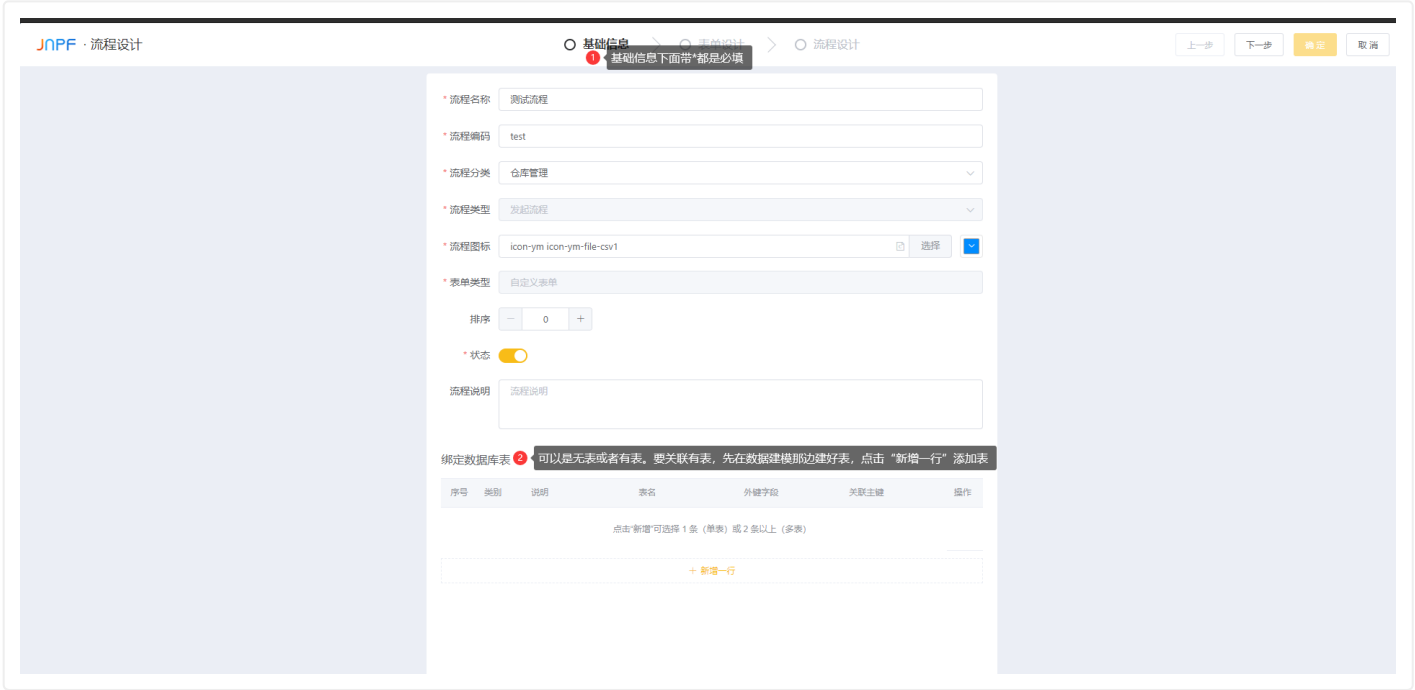
根据企业的需求创建工作流程设计，工作流程设计根据事项按不同类型进行自定义表单或者系统表单，还可以进行创建流程表单信息编辑、删除、复制流程、预览表单等操作；可以进行流程状态控制发布与停止。各部门工作流程设计可以通过每个部门不同审批流程设计不同审批流程走向。

操作步骤

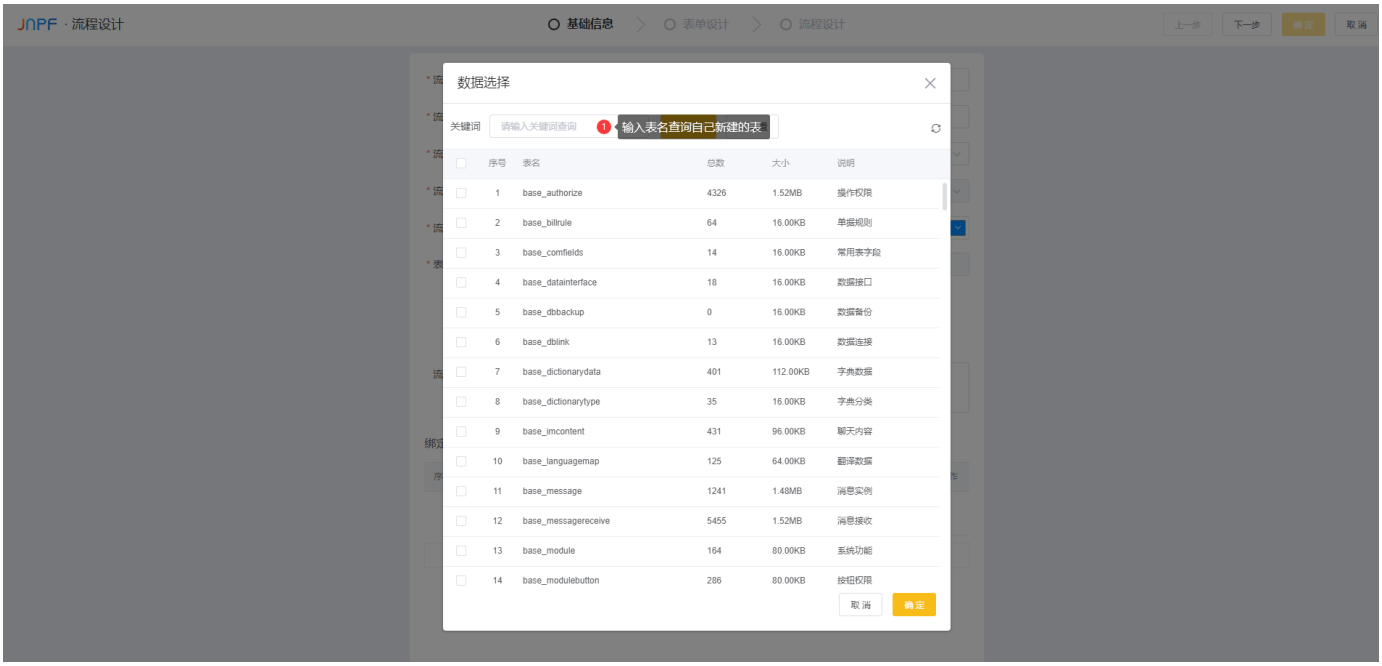
在工作流程目录下操作流程设计，进入【流程设计】页面可操作查询、刷新、全屏、新建流程功能。如下图所示：



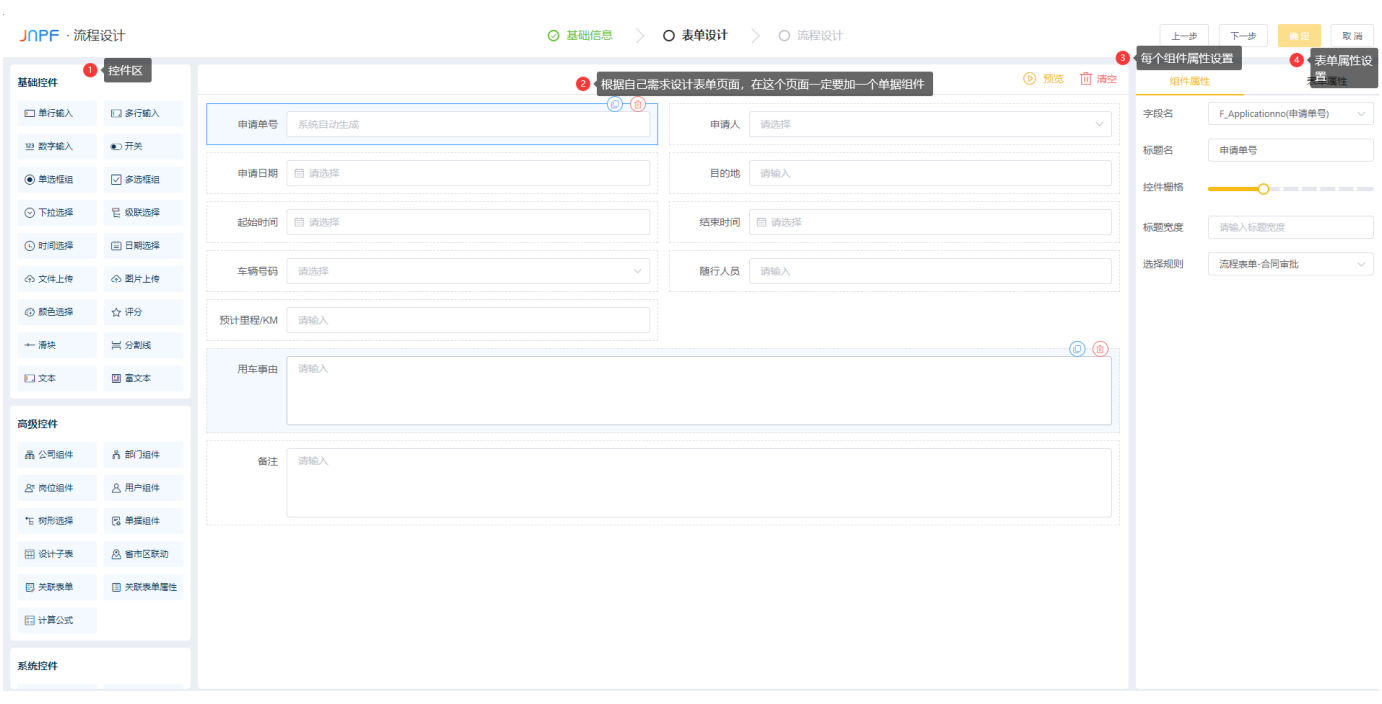
（1）点击新建“自定义表单”进入流程设计页面。如下图所示：



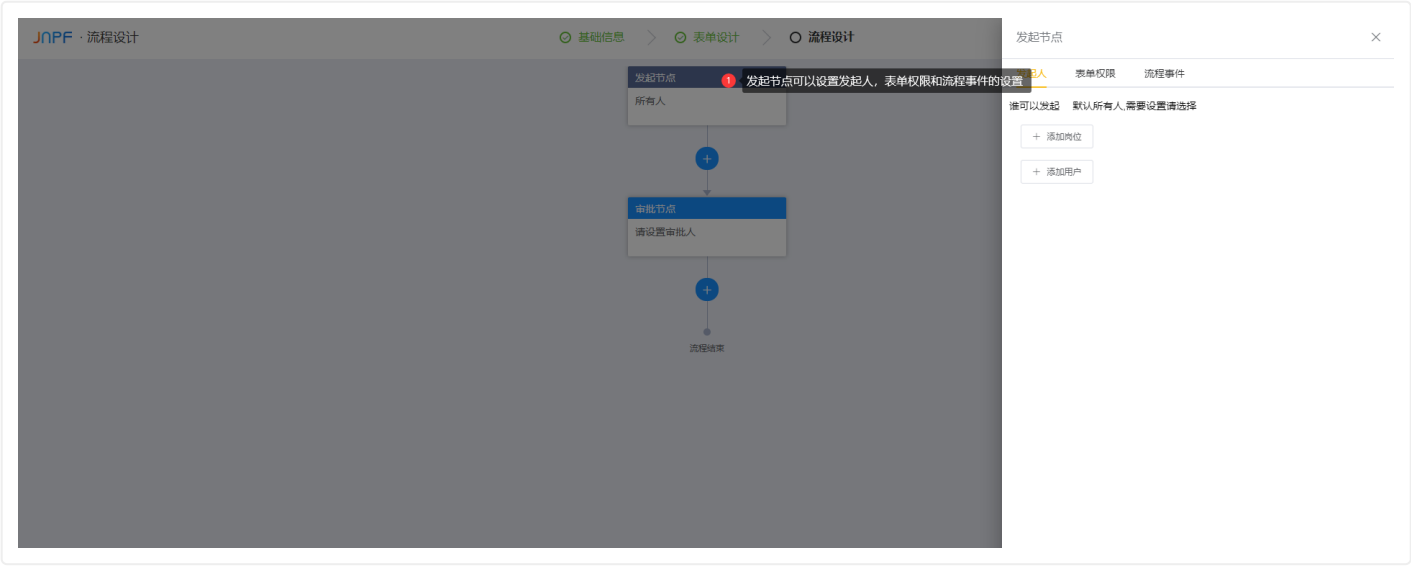
绑定数据库表，进入数据选择页面。如下图所示：



点击“下一步”进入表单设计页面，根据自己的业务流程设计表单设计。如下图所示：



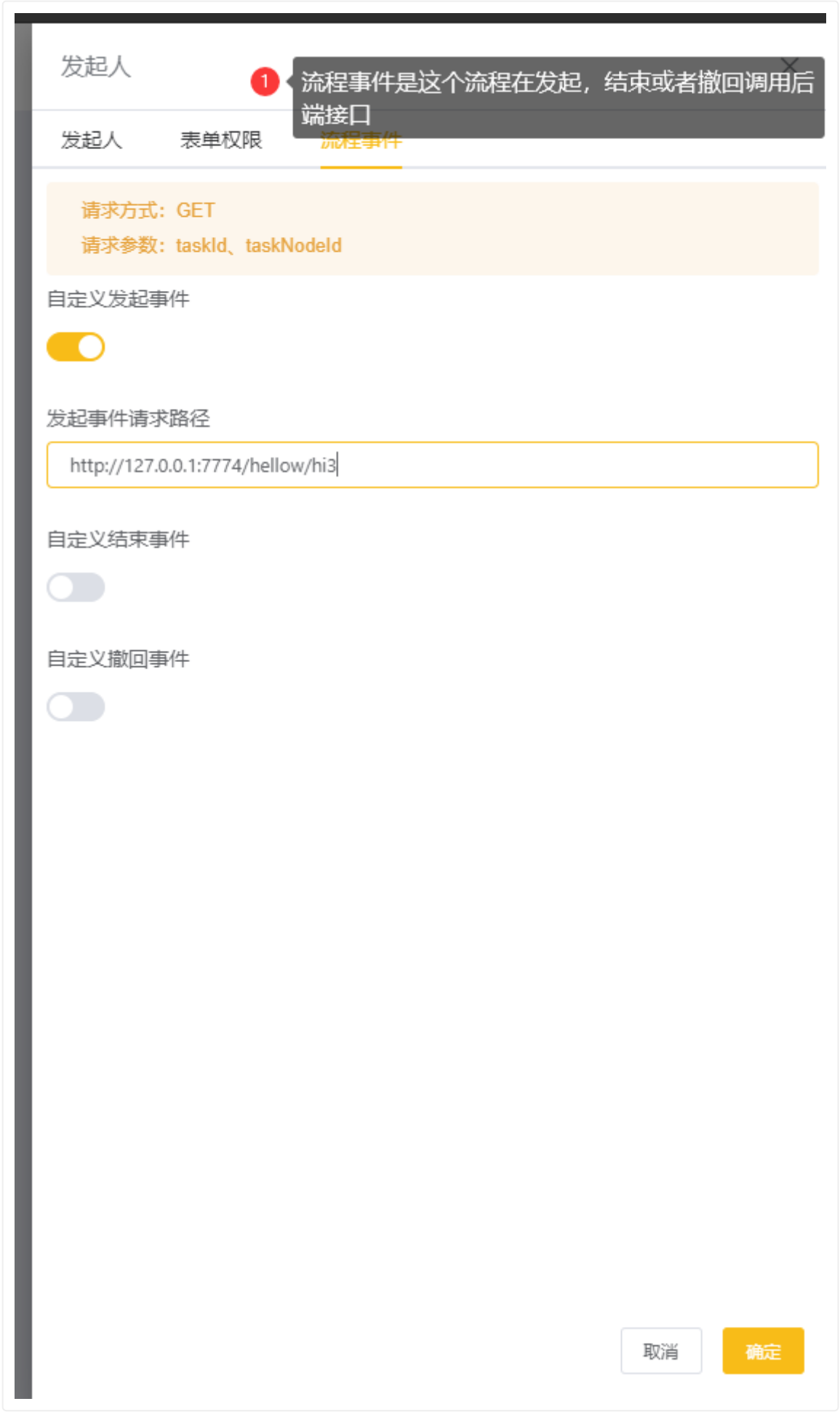
点击“上一步”进入基础信息页面设置或者点击“下一步”进入流程设计。也可以点击“取消”按钮就直接关闭这个流程设计。点击“下一步”进入流程设计页面如下图所示：


JNPF · 流程设计
基础信息
表单设计
流程设计
发起节点
所有人
发起节点可以设置发起人，表单权限和流程事件的设置
审批节点
请设置审批人
流程结束
发起节点
表单权限
流程事件
谁可以发起
默认所有人,需要设置请选择
添加岗位
添加用户

发起人说明，如下图：

发起人的表单权限，如下图：

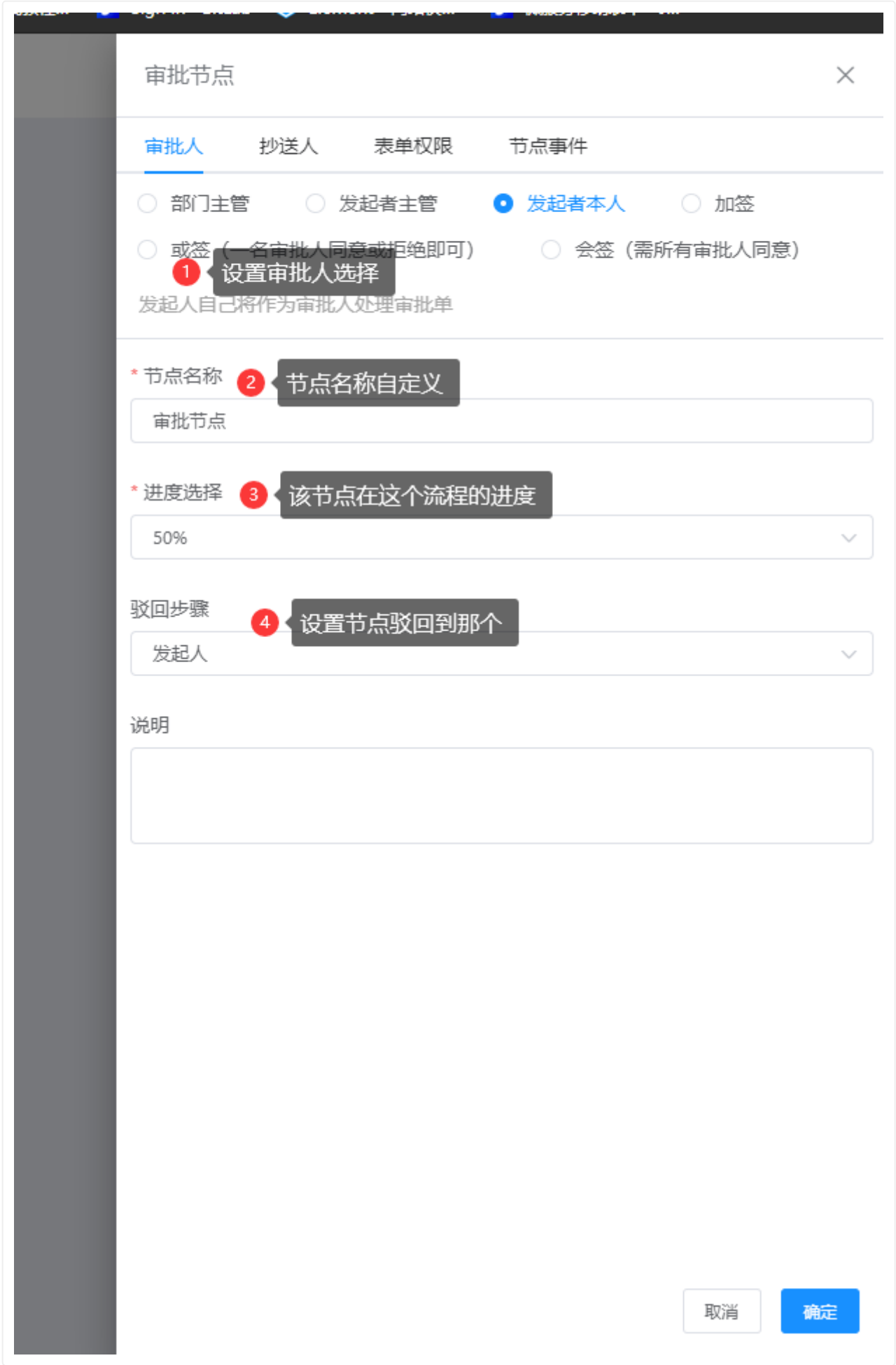
发起人的流程事件，如下图：



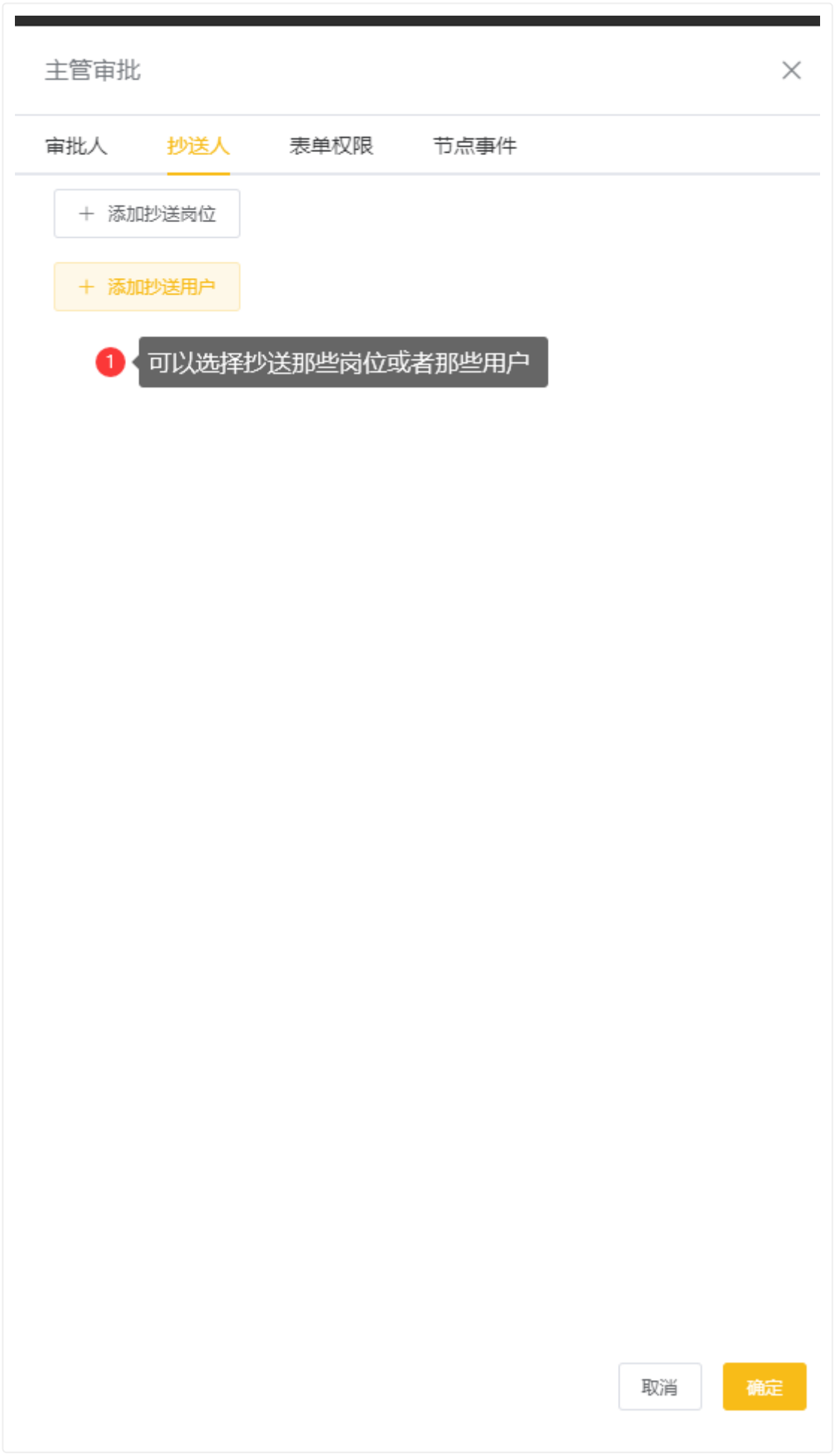
提供多种工作流程设计，可以根据自己需求进行设计，如下图：


JNPF · 流程设计
基础信息
表单设计
流程设计
上一步
下一步
确定
取消
100%
发起节点
所有人
审批节点
条件分支
分流/合流
定时器
审批节点
请设置审批人
工作流程设计采用可视化编辑，提供包括审批节点、条件分支、分流/合流、定时器等多形式流程设计。客户可以根据自己需求进行流程设计
流程结束

审批节点设置：，如下图：



审批节点的抄送人，如下图：

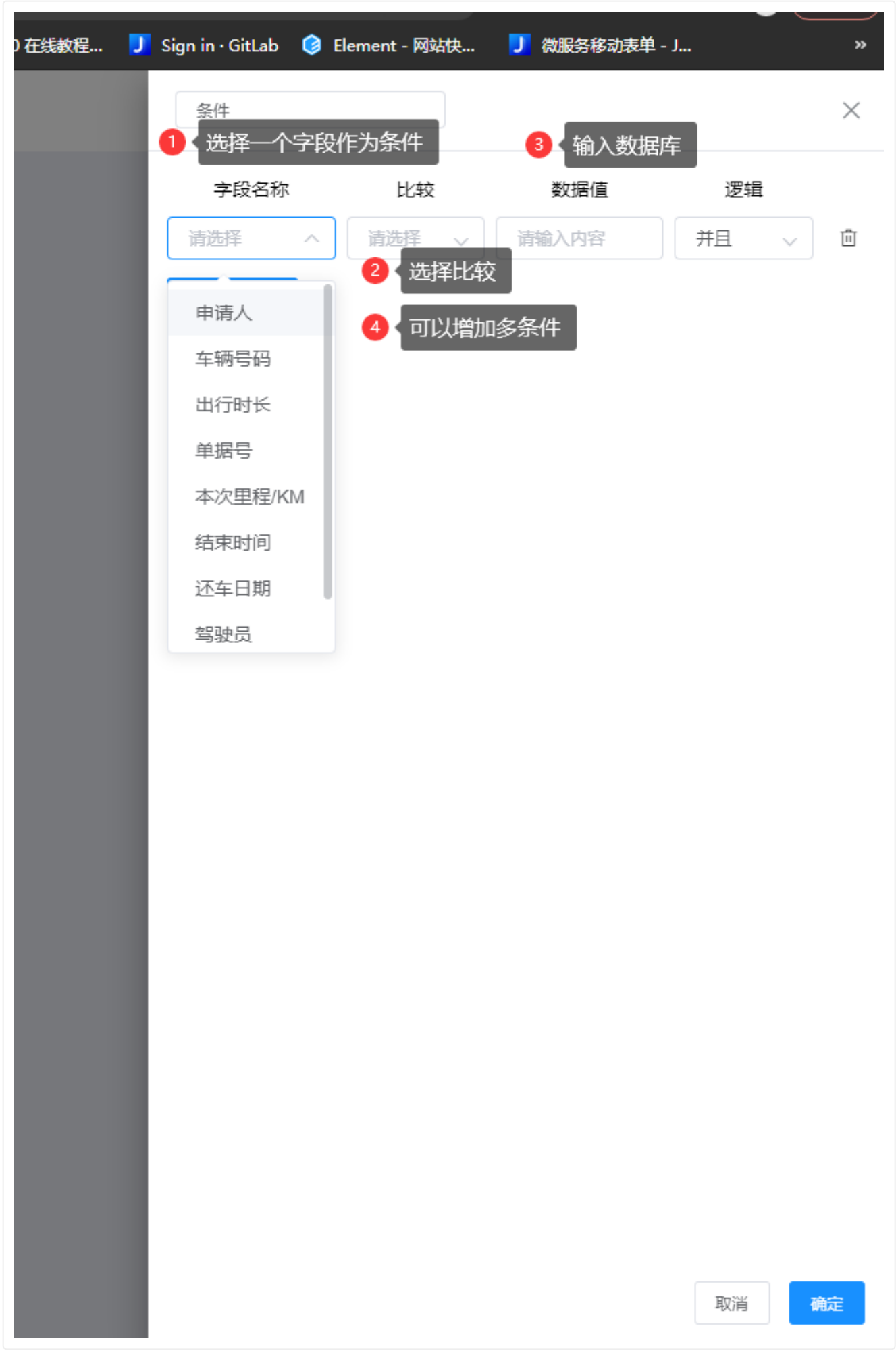


审批节点的表单权限，如下图：

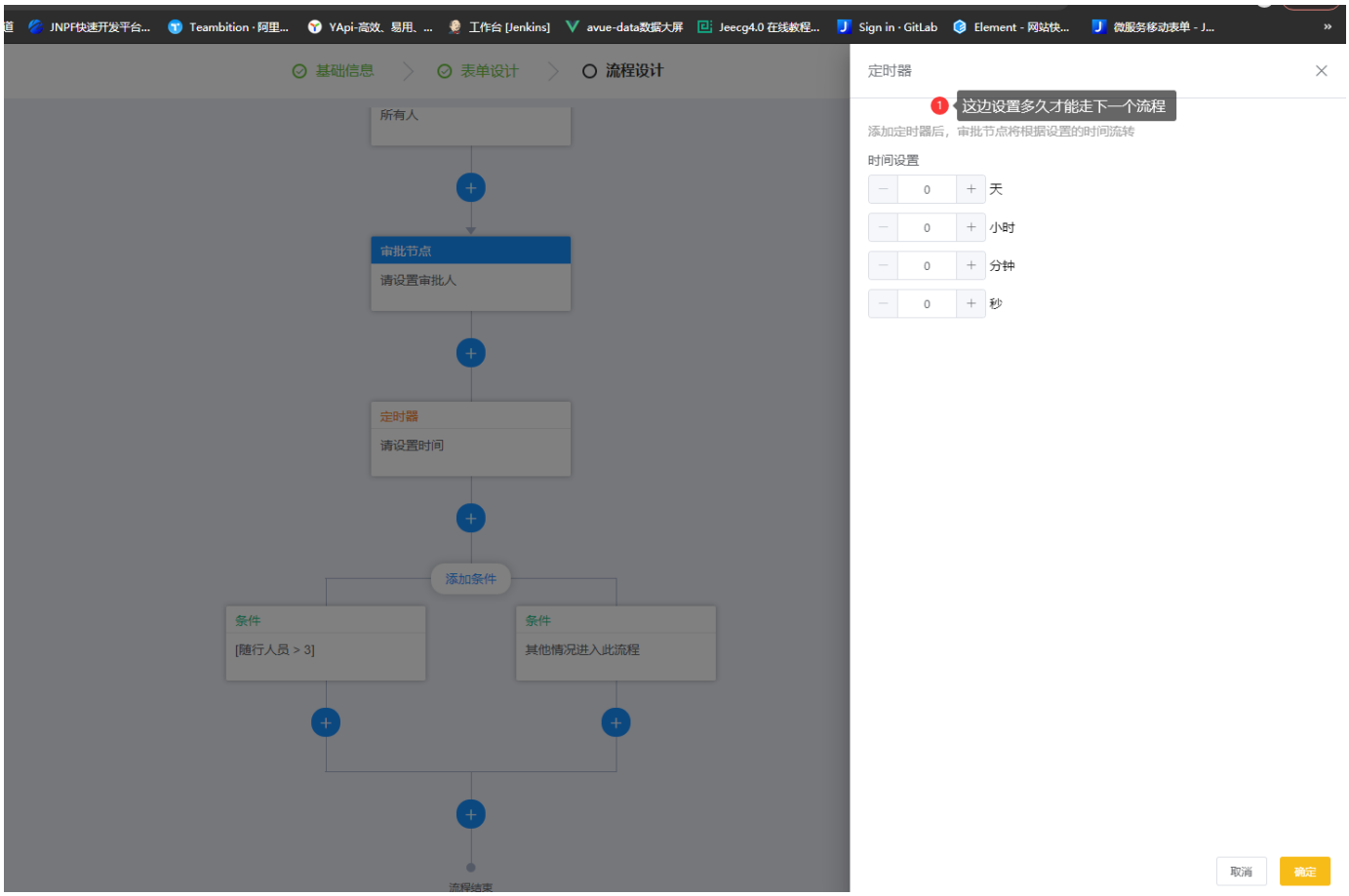
审批节点的节点事件，如下图：


主管审批
审批人
抄送人
表单权限
节点事件
请求方式：GET
请求参数：taskId、taskNodeId、handleStatus(撤回事件无此参数)
处理状态：0-拒绝、1-同意
自定义审批事件
1
审批节点的拒绝或者同意触发后端事件接口
自定义撤回事件
2
审批节点撤回触发后端事件接口
取消
确定

条件节点设置，如下图：


在线教程...
Sign in · GitLab
Element - 网站快...
微服务移动表单 - J...
条件
1 选择一个字段作为条件
3 输入数据库
字段名称
比较
数据值
逻辑
请选择
请选择
请输入内容
并且
2 选择比较
4 可以增加多条件
申请人
车辆号码
出行时长
单据号
本次里程/KM
结束时间
还车日期
驾驶员
取消
确定

定时器设置，如下图

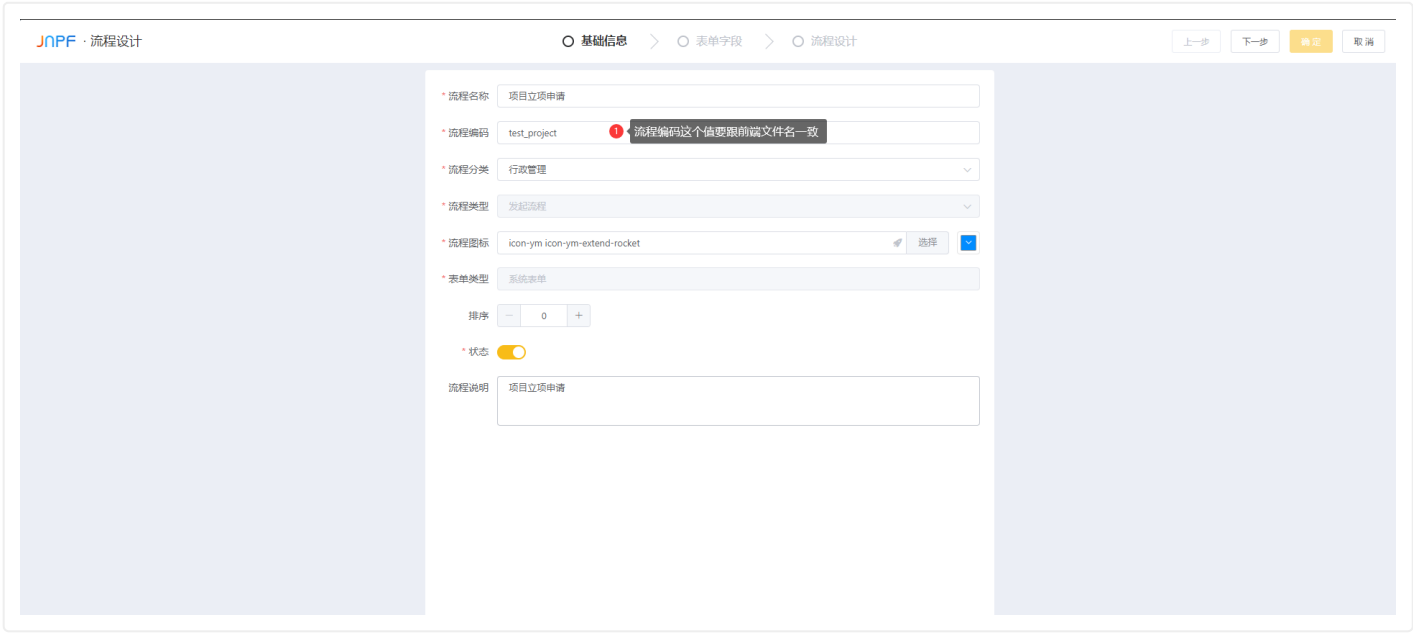

基础信息
表单设计
流程设计
所有人
审批节点
请设置审批人
定时器
请设置时间
添加条件
条件
[随行人员 > 3]
条件
其他情况进入此流程
流程结束
定时器
这边设置多久才能走下一个流程
添加定时器后，审批节点将根据设置的时间流转
时间设置
0
天
0
小时
0
分钟
0
秒
取消
确定

分流/合流设置，如下图：

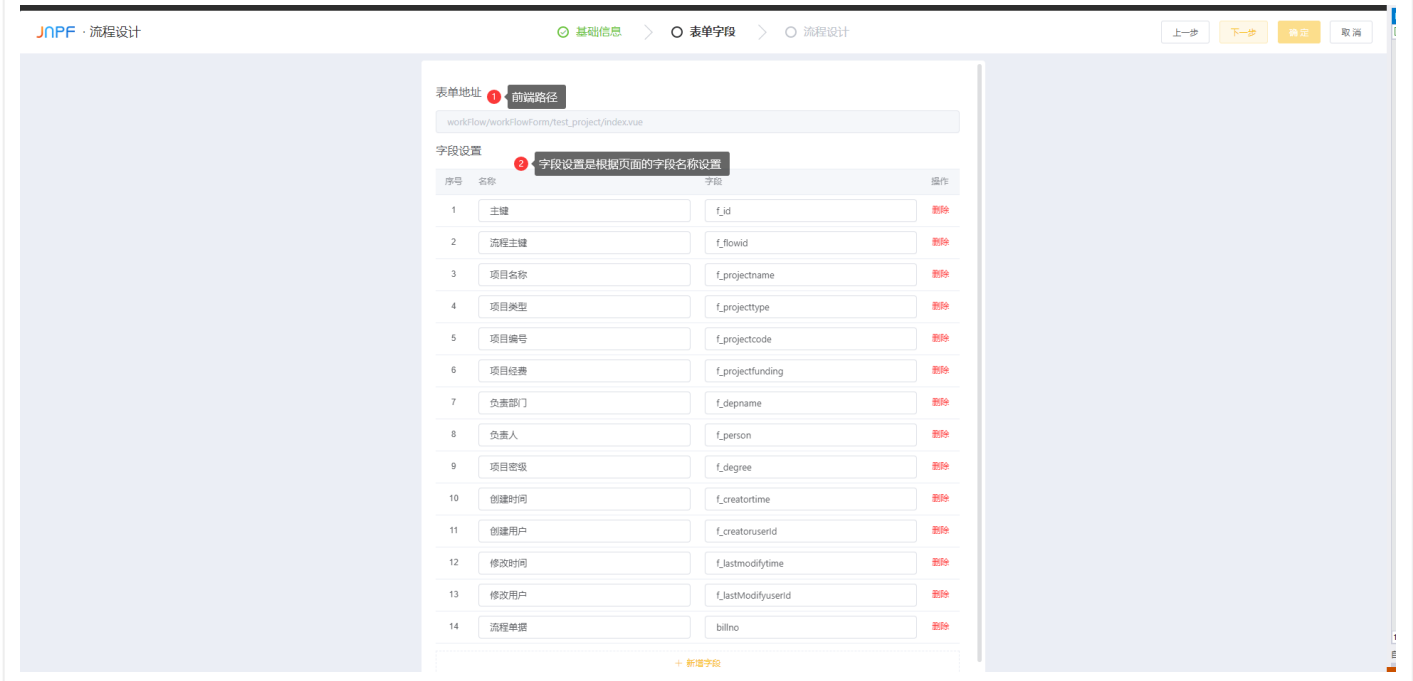

基础信息
表单设计
流程设计
所有人
审批节点
请设置审批人
"+"选择添加分流/合流节点
添加分流
审批节点
请设置审批人
审批节点
请设置审批人

设置好流程设计，点击“确定”保存这个功能表单或者点击取消按钮离开该页面，当前数据未保存。

（2）点击新建“系统表单”进入流程设计页面。系统表单是跟流程表单结合使用。如下图所示：

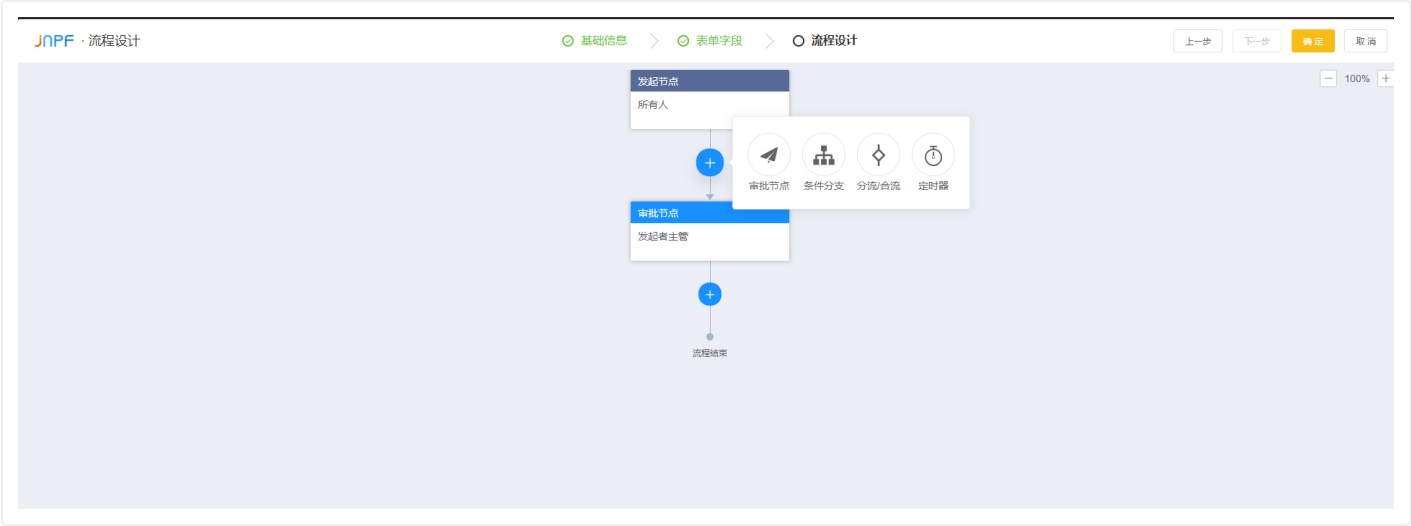


点击“下一步”进入表单字段页面，表单字段是根据用户在代码生成流程表单设计表进行数据字段，确认点击“下一步”进行流程设计或者点击取消按钮离开该页面、当前数据未保存。如下图：



点击“下一步”进入流程设计页面，流程设计是根据用户审批流程需要设计审批节点、添加条件、分流/合流和定时

器等，点击“确定”保存这个功能表单或者点击取消按钮离开该页面、当前数据未保存。如下图：



系统表单的流程设计跟自定义流程设计是一样。

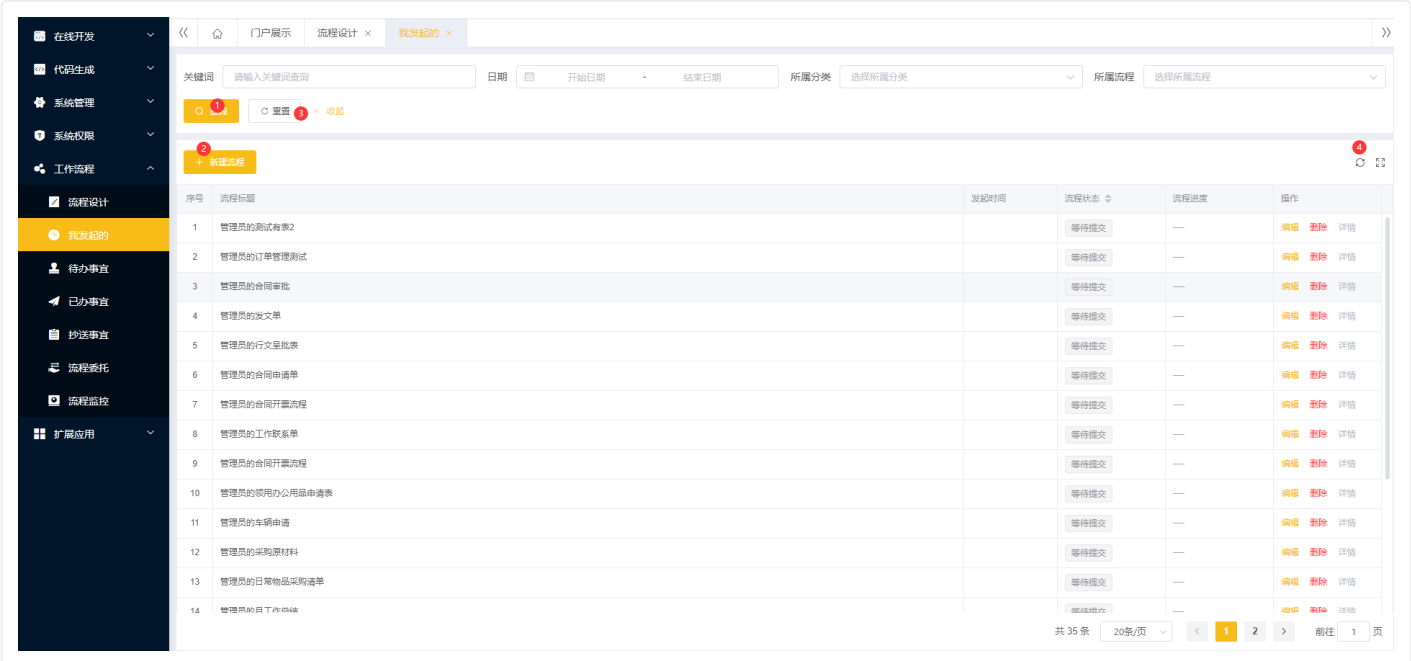
我发起的

功能描述

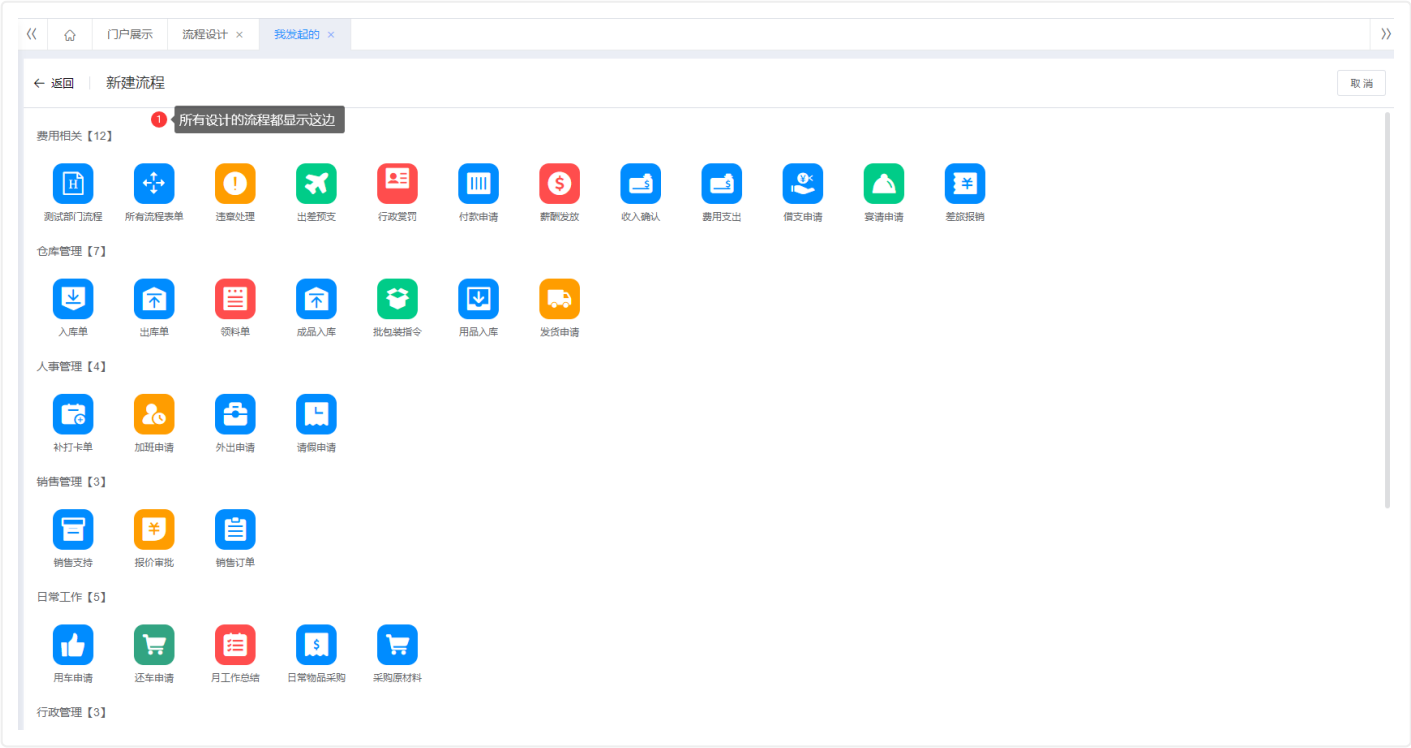
当前用户发起的流程实例。对各个办公流程规范化，使繁琐流程变为轻量级，同时展示出每个信息状态，让工作内容、步骤和程序有迹可查。

操作步骤

在我发起页面有查询（关键字、日期范围、所属分类、所属流程）进行查询，新建流程、重置、刷新等；如下图所示：



点击新建流程，进入新建流程页面，如下图：

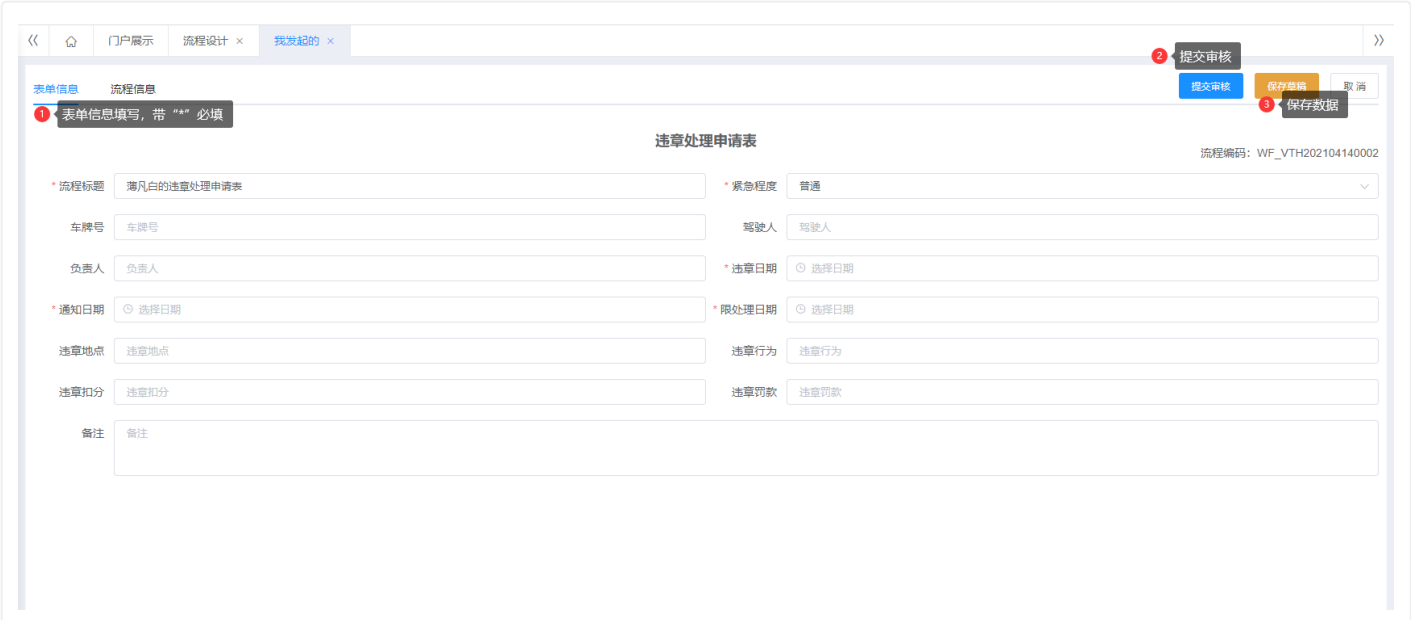


示例：点击发起一个“违章处理”流程，如下图：

内容填写完毕点击保存草稿按钮表示自动保存当前数据、可以再次修改、未被提交流程。

内容填写完毕点击提交审核按钮表示当前数据自动提交流程，不能编辑修改。

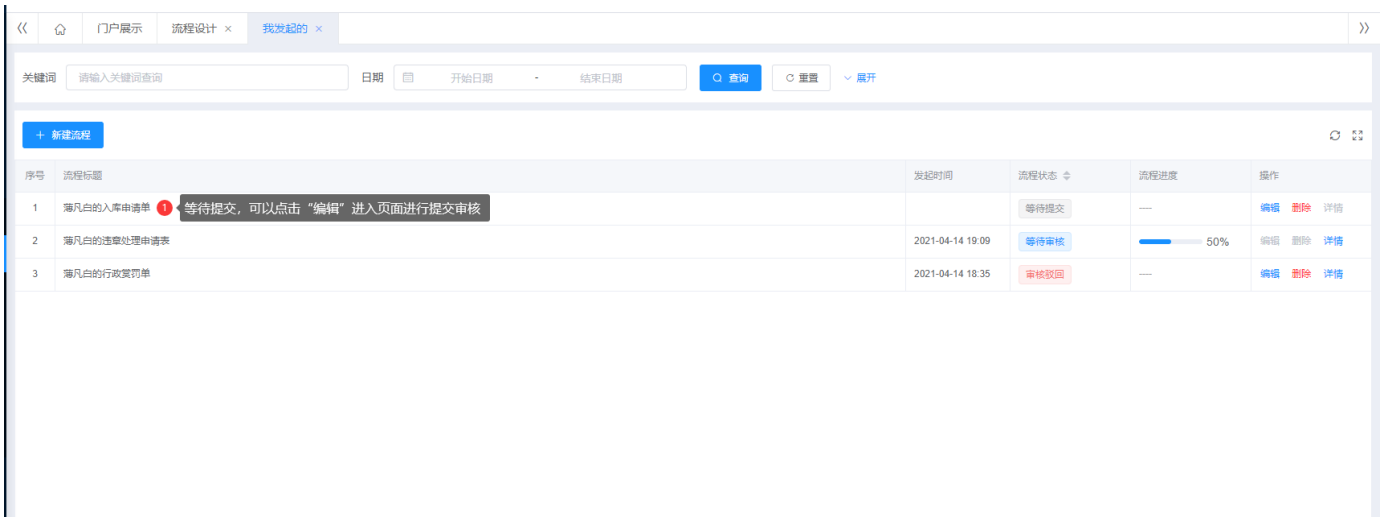
内容填写完毕点击取消按钮表示自动离开当前页面，当前数据未被保存。



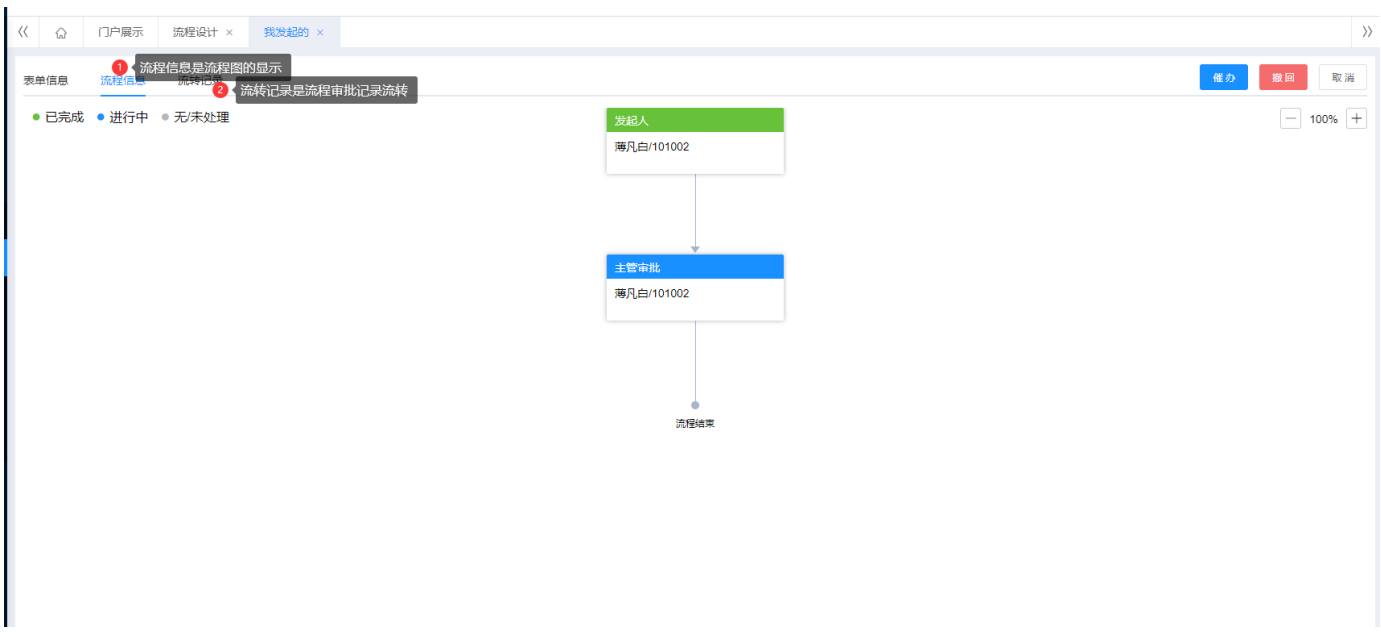
在我发起页面选择一条数据显示等待提交状态、可以进行当条数据操作编辑或删除功能操作，如下图所示：

编辑修改当前这条数据完、点击提交进行进度转变为等待审批状态。

点击删除按钮，删除提示“您确定要删除当前数据吗，是否继续？”如是点击确认键删除当前数据，如果操作取消按钮关闭当前提示框、该条数据未被自动删除，在我发起页面数据依然存在。



在我发起页面等待审核状态点击“详情”按钮、进入详情页面可以操作撤回、催办、取消功能，如下图所示：
选择等待审核状态数据点击查看数据内容，发现提交内容有错误，这时可以操作撤回（前提条件是未被审核过）当前这条数据，如果被审核过的数据只能进行查看操作。



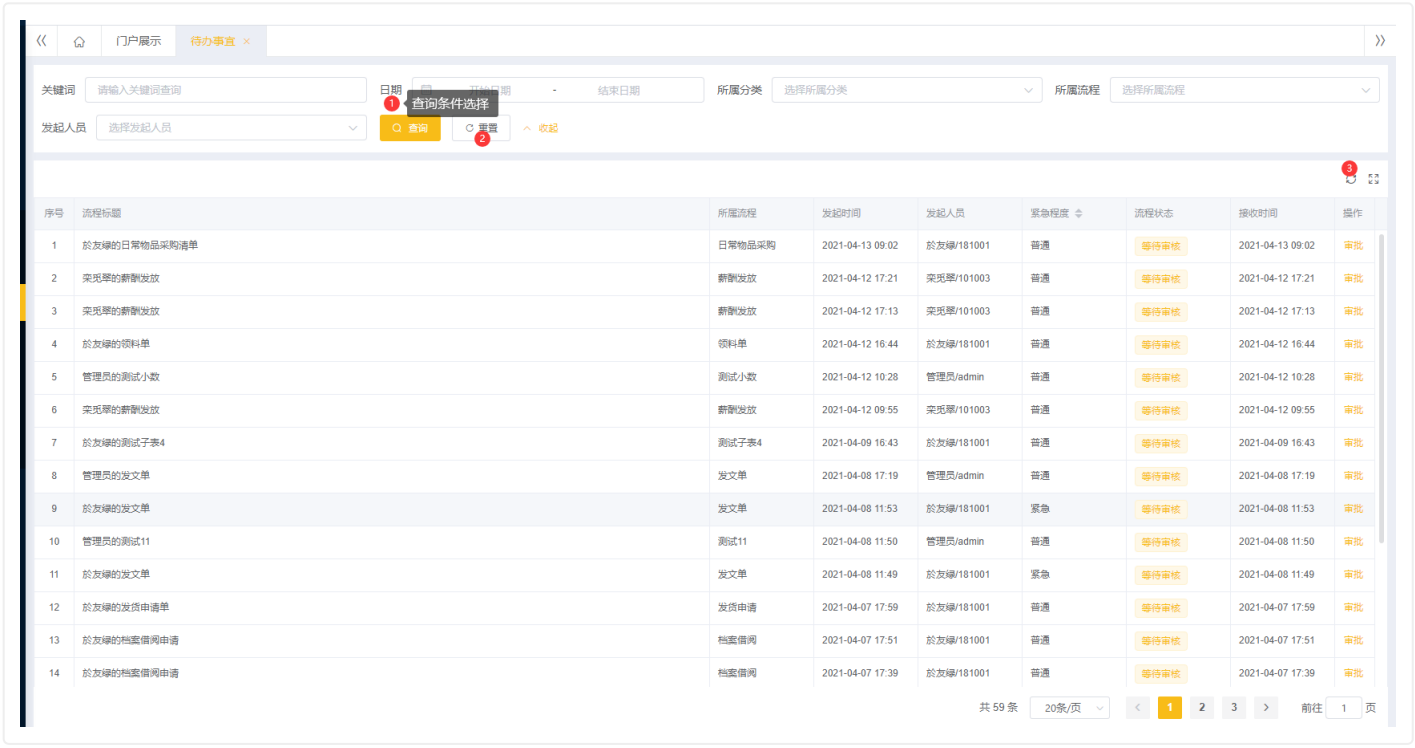
待办事宜

功能描述

需要用户处理的流程。日常待办提醒，提醒用户及时处理待办事宜。集中处理待办事项，可转办工作事项，适应不同的工作场景

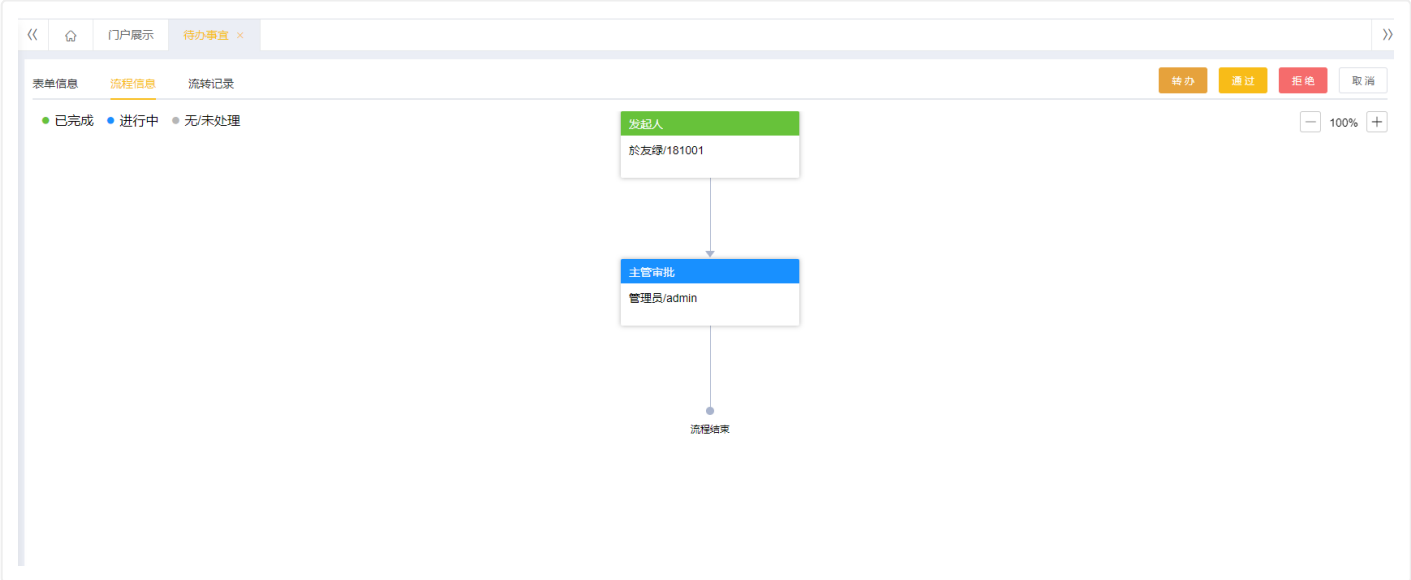
操作步骤

在待办事宜页面有查询（关键字、日期范围、所属分类、所属流程、发起人员）进行查询，重置、刷新等；如下图所示：

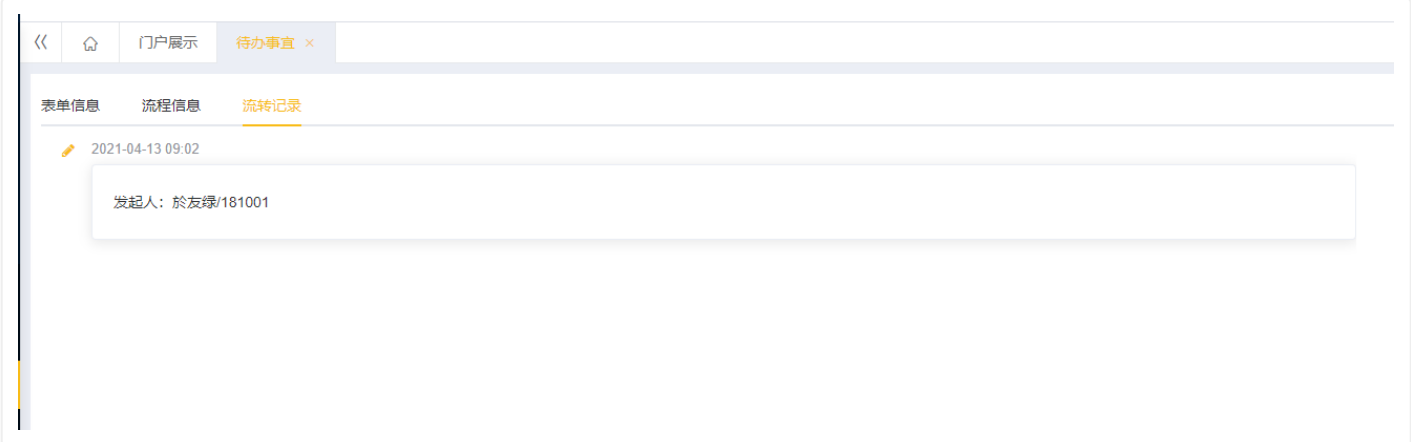


待办事宜操作列表点击“审批”进入审批页面。如下图所示：
在流程表单页面点击通过流程进入下一个阶段审批，流程反馈给申请人。
如果点击拒绝表示当前未通过，直接驳回到流程设计中设计的驳回节点。
如果点击转办可以把当前审批转办给别人来审批，转办人会有消息提醒。
操作取消按钮离开该页面，数据没有任何状态显示。

查看流程信息显示审批流程以及已完成、进行、无/未处理颜色显示表示。如下图所示：



流转记录显示，显示出有谁创建、创建时间、谁审批、申请时间。如下图所示：



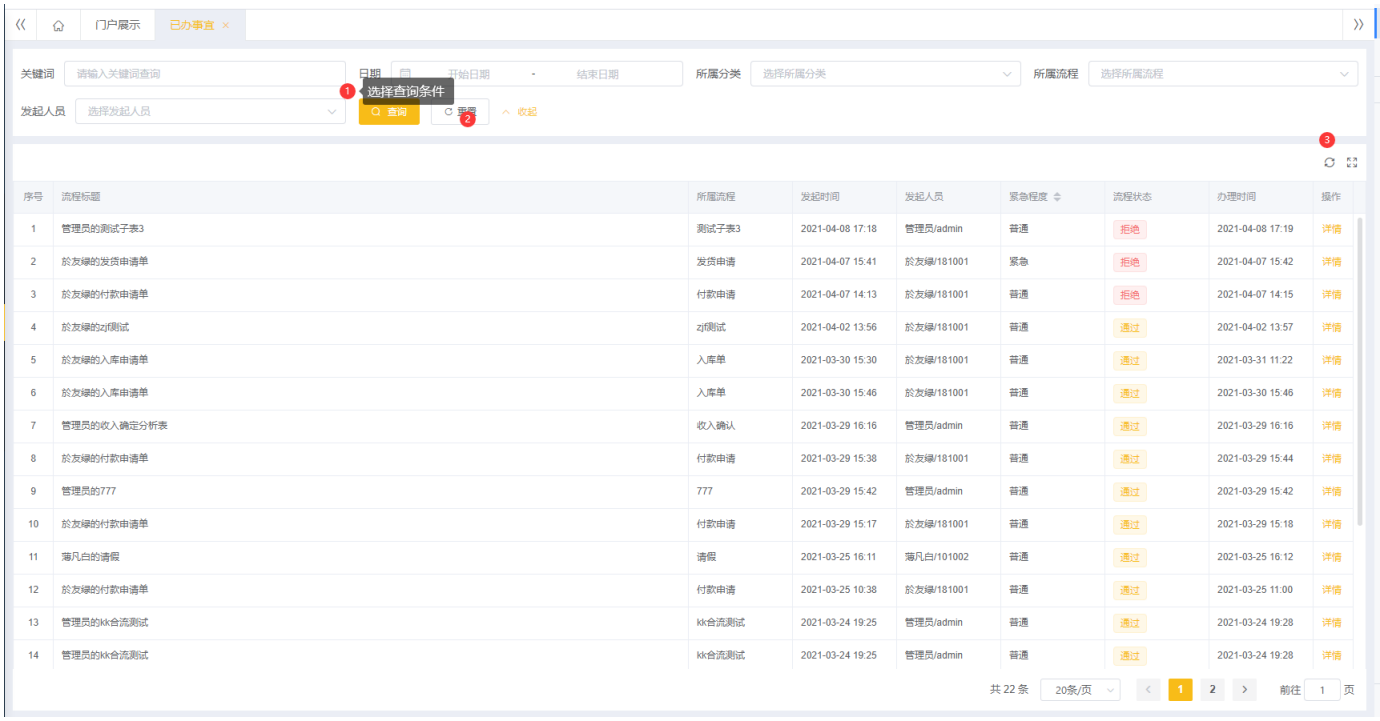
已办事宜

功能描述

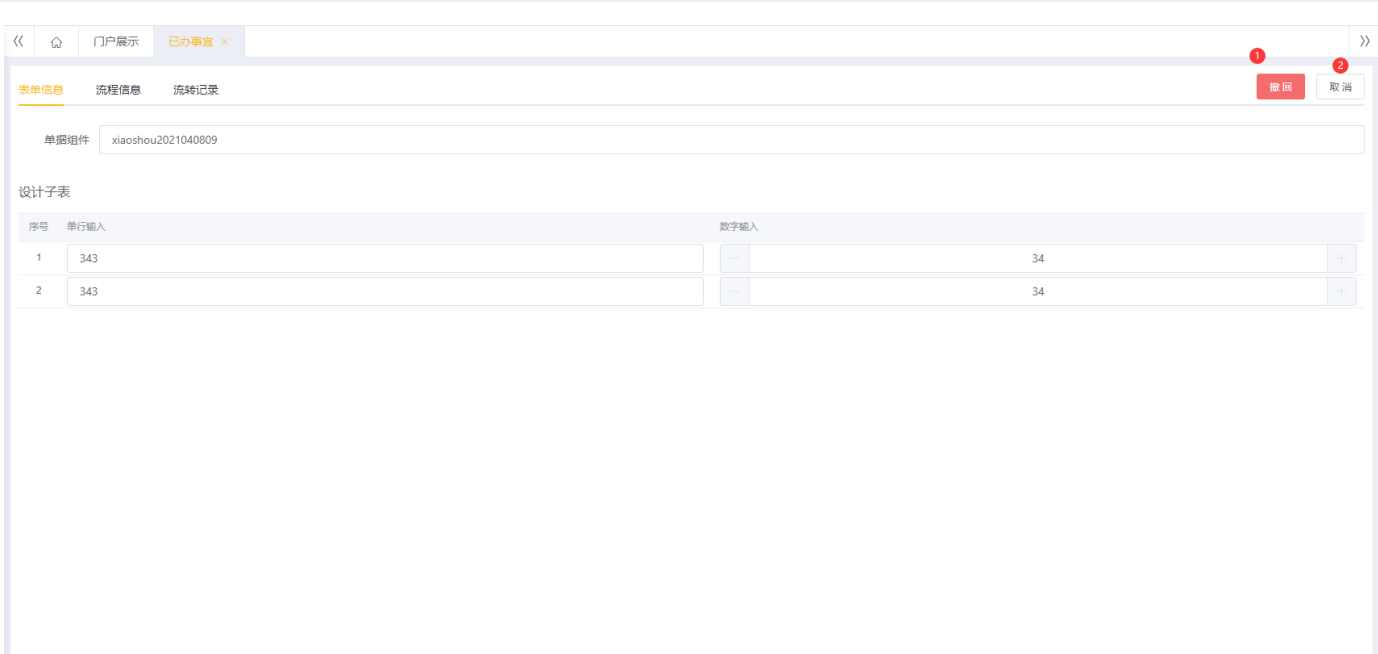
用户处理过的流程实例。业务记录有地可查，在当前页面查看详细流程表单、流程视图以及流转记录等，更直观查看流程记录

操作步骤

在已办事宜页面有查询（关键字、日期范围、所属分类、所属流程、发起人员）进行查询，重置、刷新等；如下图所示：

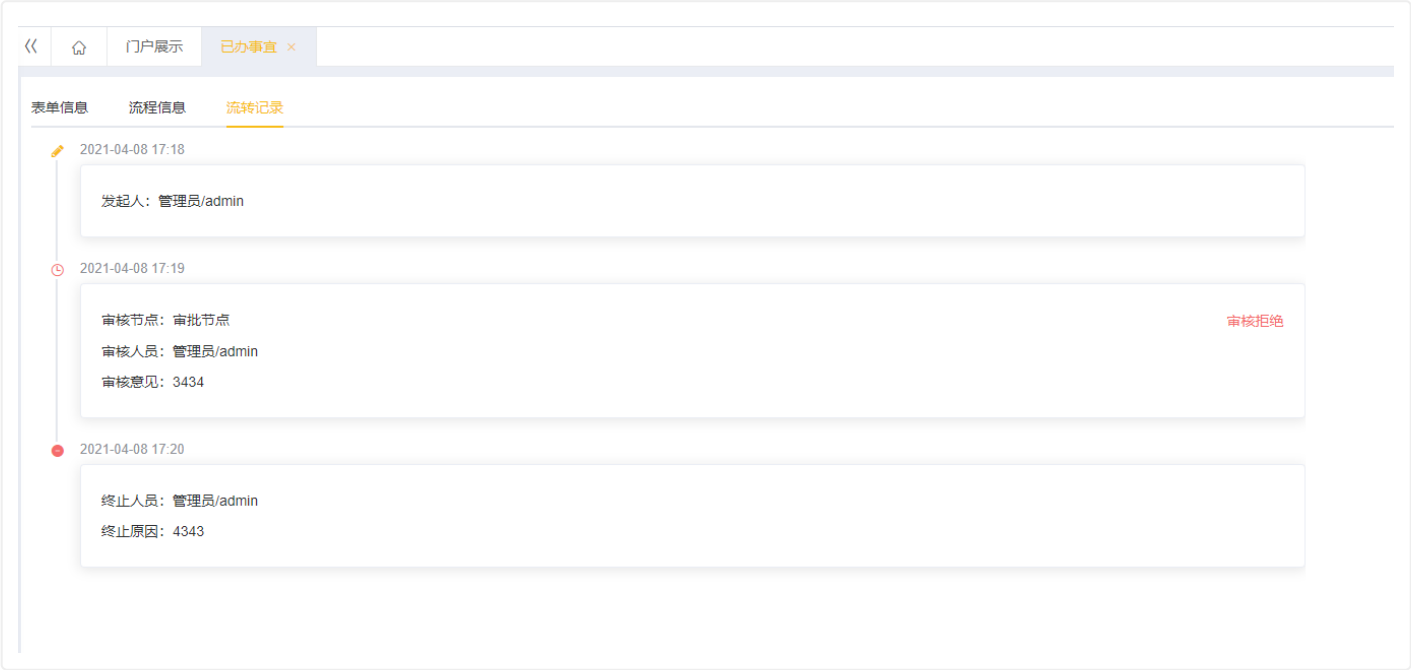


已办事宜操作列表点击“详情”进入审批页面。如下图所示：
在流程表单页面点击撤回表示:当前流程未审批完可以撤回数据成功，如果当前流程审批完不能够撤回当前这条数据。
操作取消按钮离开该页面，数据没有任何状态显示。



查看流程信息显示审批流程以及已完成、进行、无/未处理颜色显示表示。如下图所示：

流转记录显示，显示出有谁创建、创建时间、谁审批、申请时间。如下图所示：



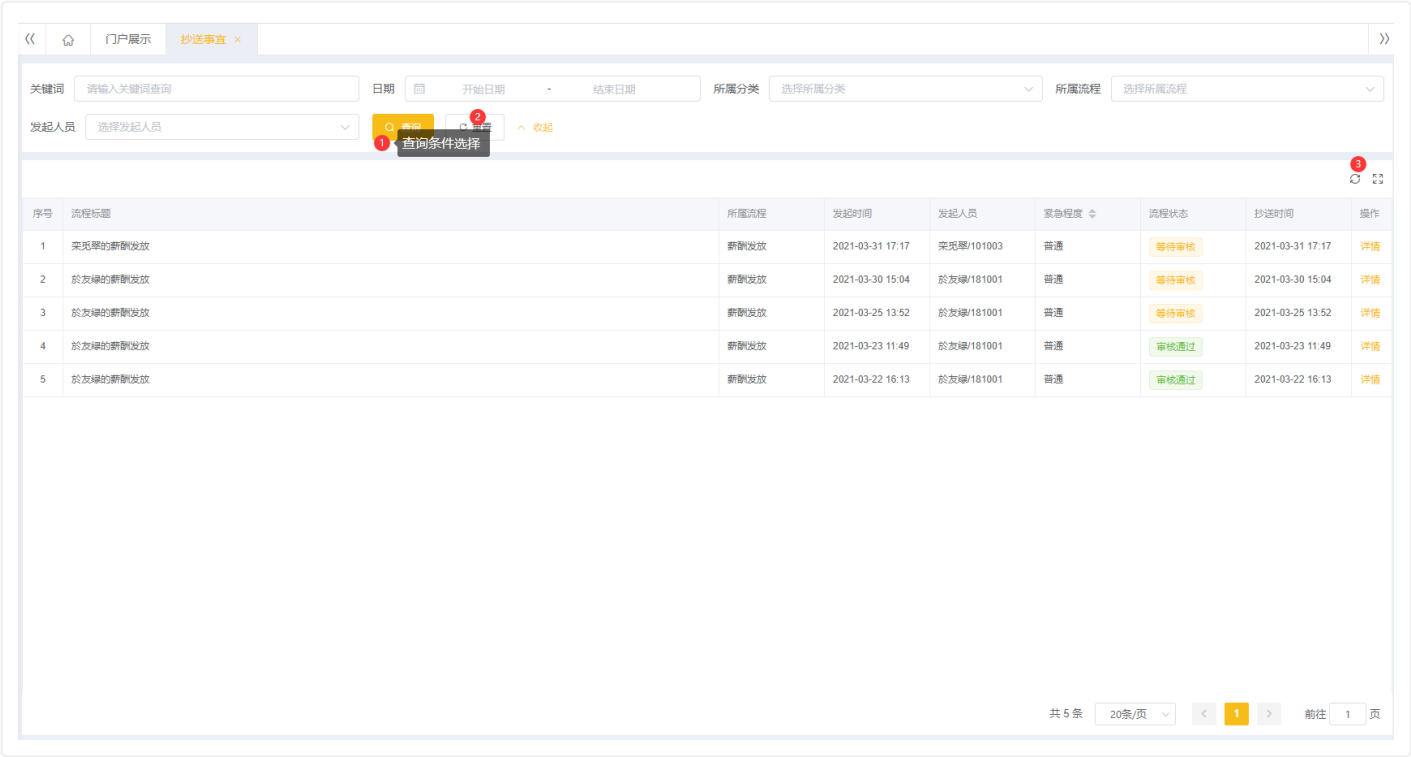
抄送事宜

功能描述

抄送给用户的流程实例。快速将当前事项信息同步到对应的负责人手中、方便查看

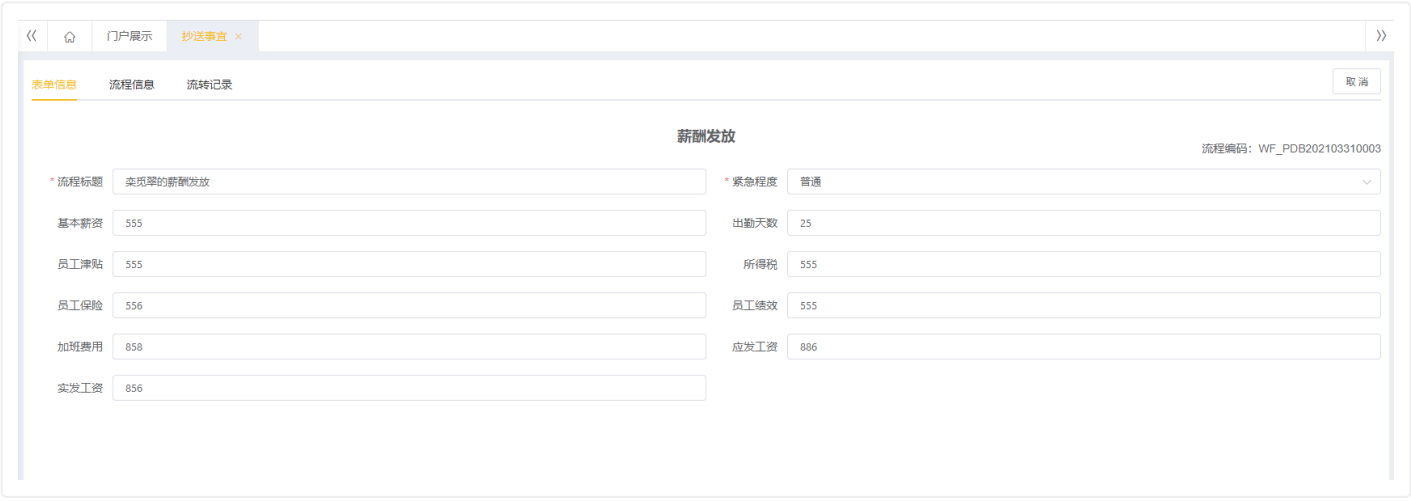
操作步骤

在抄送事宜页面有查询（关键字、日期范围、所属分类、所属流程、发起人员）进行查询，重置、刷新等；如下图所示：



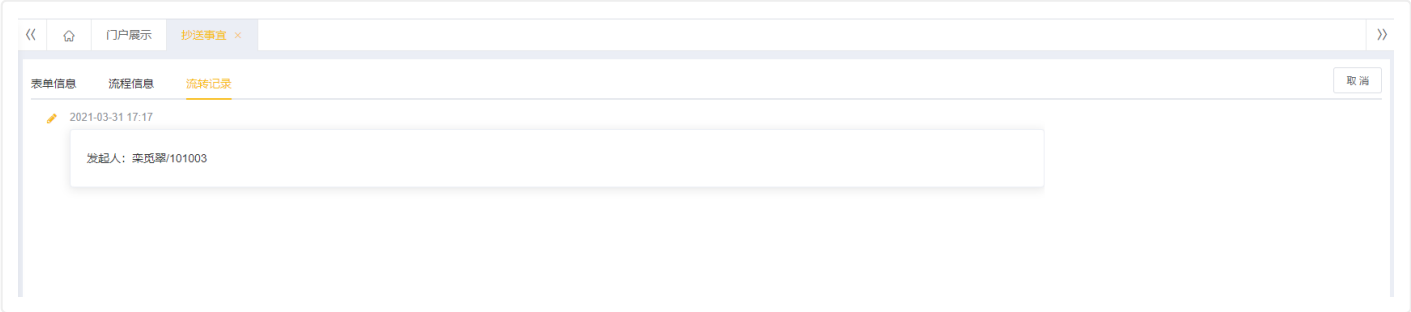
抄送事宜操作列表点击“详情”进入审批页面。如下图所示：

在流程表单页面点击取消按钮离开该页面，数据没有任何状态显示。



查看流程信息显示审批流程以及已完成、进行、无/未处理颜色显示表示。如下图所示：

流转记录显示，显示出有谁创建、创建时间、谁审批、申请时间。如下图所示：



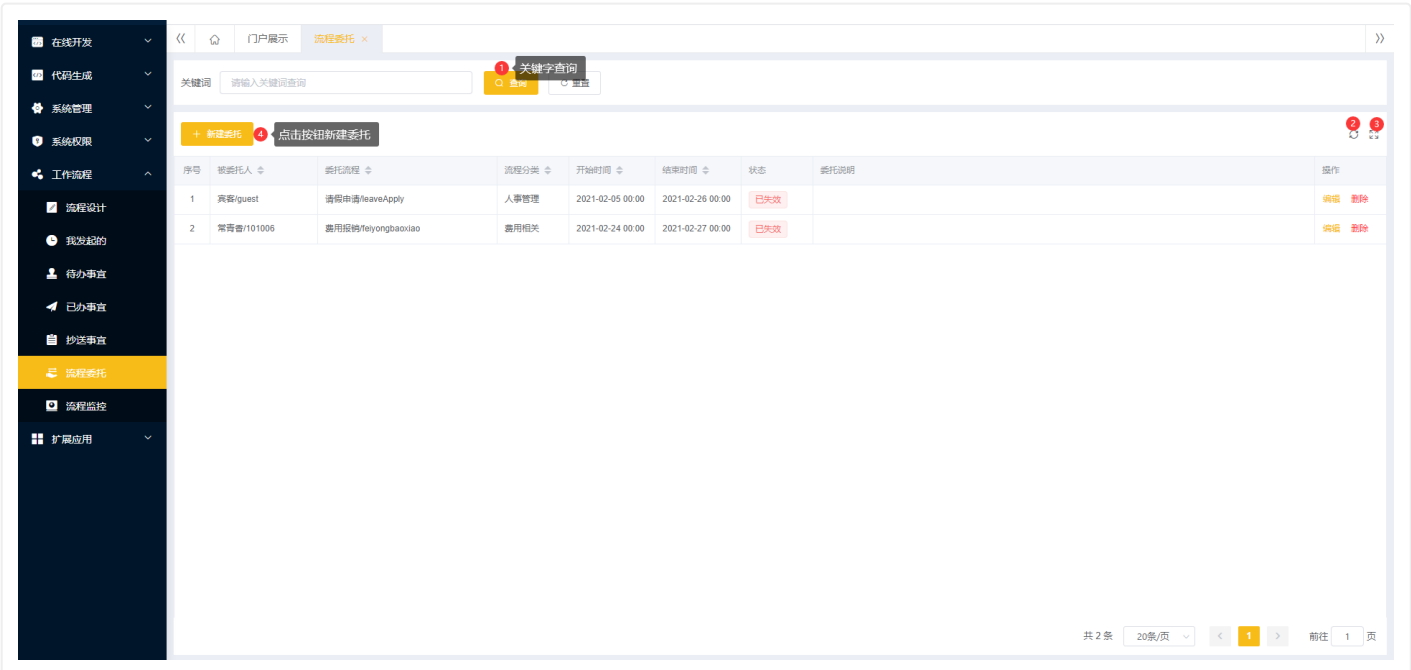
流程委托

功能描述

用户转交/代理给别人办理的任务。工作流程中的审批人出现某段时间不在岗的情况，需要指定代理人、时间、审批功能模块，代理人在指定时间内继承委托人审批权限

操作步骤

在工作流程目录下操作流程委托，进入【流程委托】页面可操作查询、刷新、全屏、新建委托功能。如下图所示：



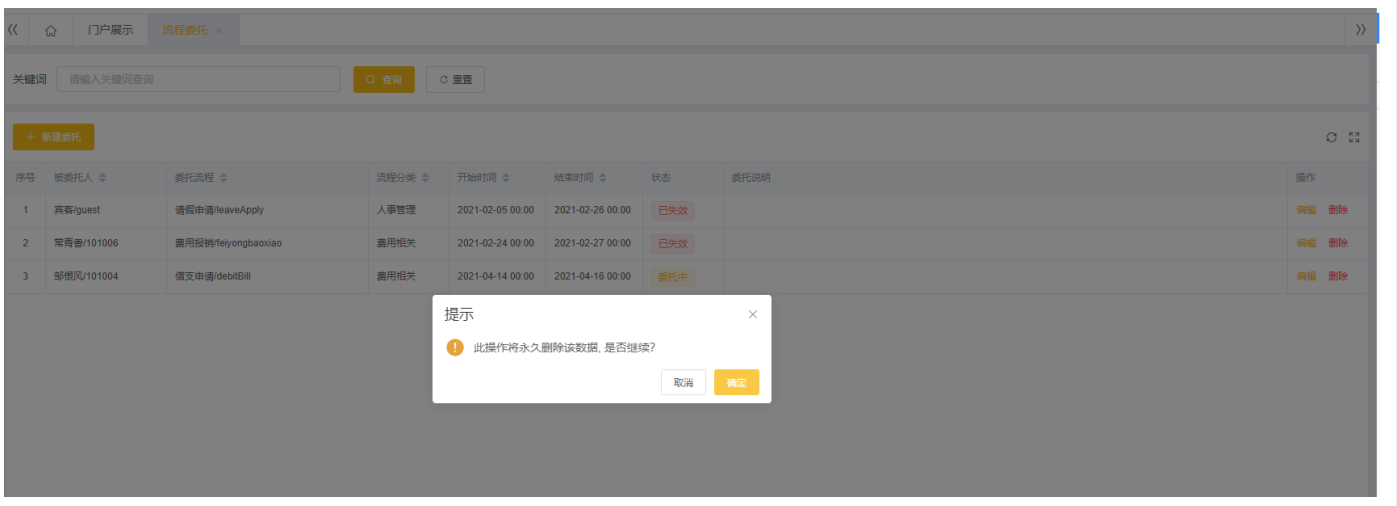
进入新建委托页面，下拉框选择被委托人、委托流程、开始时间、结束时间、备注填写委托说明；信息填写完毕点击确定按钮保存当前数据，关闭按钮自动关闭当前页面、当前时间未被保存。
当前被委托人在指定时间范围内可以临时代理审批工作流程权限。如下图所示：

在流程委托页面可以对数据进行编辑和删除功能，如下图所示：

选择点击编辑按钮，进入编辑页面修改数据，点击确定按钮修改保存成功；或者点击关闭按钮自动关闭当前页面、数据未被保存。

工作流程委托审批是否生效、在状态显示委托中表示委托人可审批，状态显示已失效表示委托人不能够审批该流程。

点击删除按钮提示“您确定要删除当前数据吗，是否继续”信息，如果需要删除点击确认按钮、删除当前这条数据成功；或者点击取消按钮当前窗体自动关闭、当前选择数据未被删除。如下图所示：



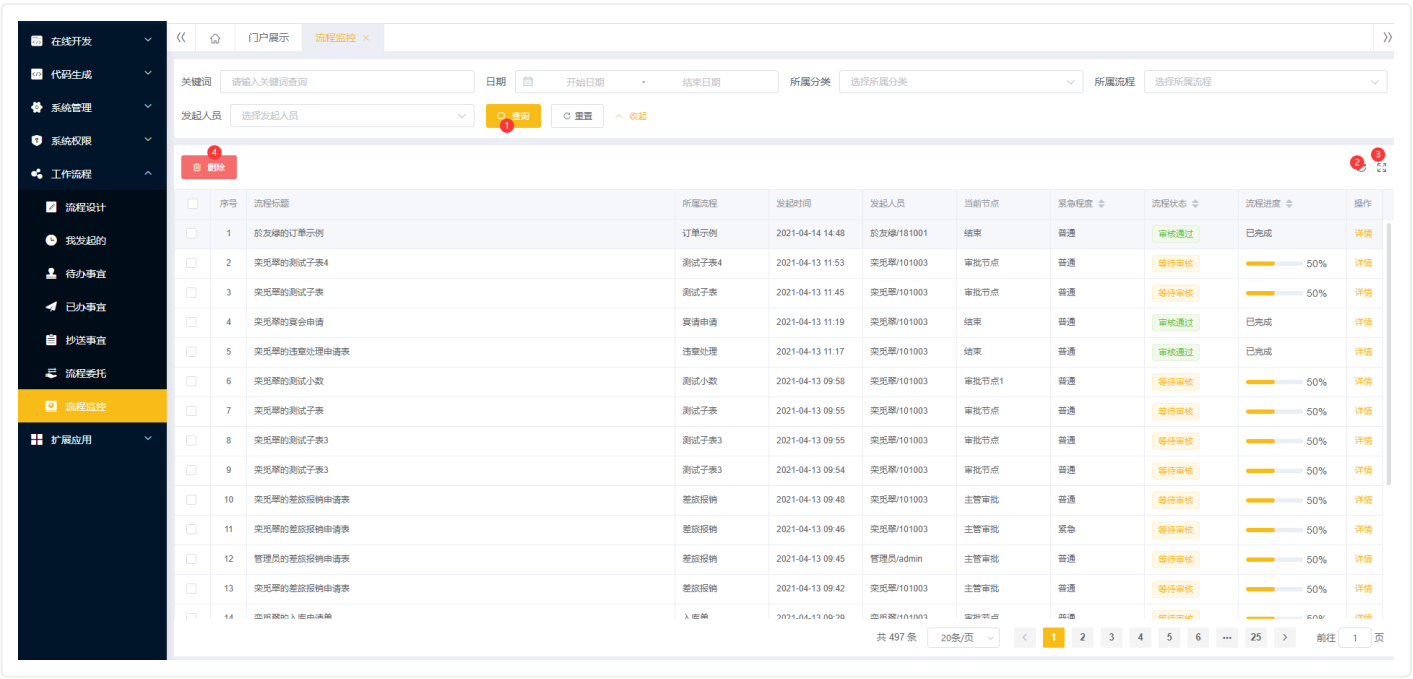
流程监控

功能描述

监控流程进度，可以查看不同状态流程的流程内容，流程进度及流转记录，同时操作中止流程功能。

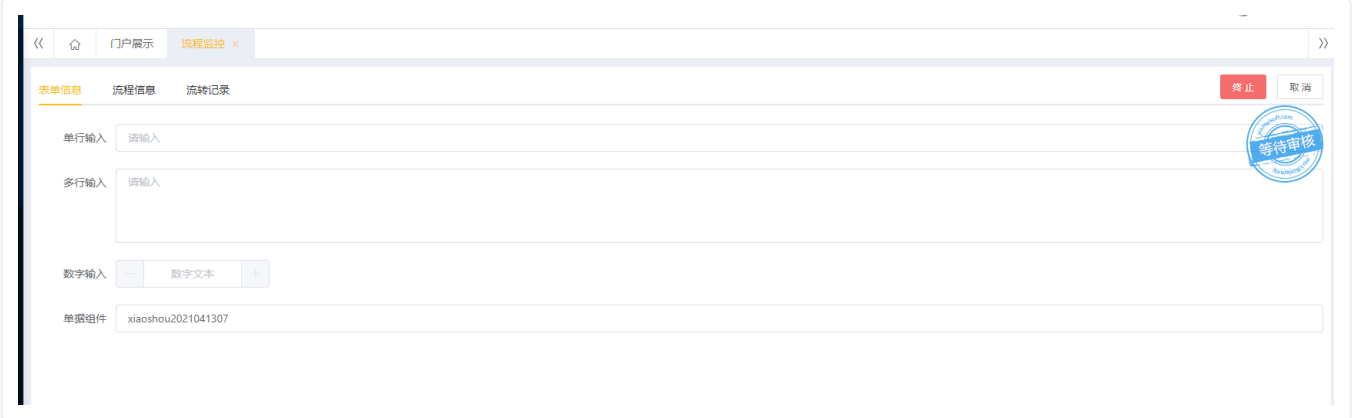
操作步骤

在工作流程目录下操作流程监控，进入【流程监控】页面可操作（关键字、日期范围、所属分类、所属流程、发起人员）进行查询、刷新、全屏、删除功能。如下图所示：



点击详情按钮进入详情页面，页面中有终止、取消功能，如下图所示：

1. 点击操作终止流程成功，当终止流程工作操作成功时流程状态显示流程状态从等待审核转变为审核终止。
2. 全部流程审核完毕时，点击详情进入详情页面不显示终止按钮，但是可以查看该条信息；



扩展应用

报表示例

大屏示例

图表示例

表格示例

表单示例

功能示例

百万数据

导入导出

电子签名

电子签章

日程安排

邮件收发

知识管理

文件预览

条码示例

打印示例

地图示例

订单管理

项目管理