EIP 开发手册

Version 7.1

广州宏天软件股份有限公司

www.hotent.com

各章节内容简介：

第 1 章 概述：平台简介

第 2 章 构建开发环境：如何构建 EIP 平台开发环境

第 3 章 平台架构：技术框架、开发模式等介绍

第 4 章 快速开始：以简单的案例入手、直接体验如何使用平台

第 5 章 组件开发：详细介绍如何开发各类组件

第 6 章 工作流程：流程开发和管理

第 7 章 专题：综合功能和独立功能的讲解

第 8 章 案例分析：结合案例深入了解平台运行机制

第 9 章 应用定制：介绍针对具体项目时，可能需要作出的调整

第 10 章 配置文件：配置文件详解

第 11 章 微服务五合一部署：如何以一个应用来部署 EIP 系统

第 12 章 开发管理：基于 EIP 开发的项目管理参考

目录

版本记录

序号	作者	版本号	修改内容	时间
1	何一帆	V1.0	初始化文档	
2	黎扬贵	V1.1	添加 国际化 说明 添加 统一日志 说明 添加 单点登录 说明	2018-09-14
3	黎扬贵		添加 认证机制 添加 权限控制	2018-09-21
4	何一帆	V1.2	添加了应用开发的章节 完善了消息队列、内存数据库的内容	2019-06-24
5	何一帆	V1.3	完善工作流章节、配置文件、微服务五合一部署、开发管理章节	2019-07-02
6	何一帆	V1.4	完善了案例分析章节，更新了 URL 表单章节	2019-07-03
7	何一帆	V1.5	添加了 7.3.3 章节数据级别的权限控制	2019-07-24

1 概述

EIP（Enterprise Information Platform）是基于 Java 的企业信息平台，平台有以下多种角色：

- **流程中心**

流程集中式管理，统一的用户组织架构，统一的流程审批，统一的流程消息。业务数据由各业务模块或业务系统管理，业务系统与流程中心之间通过 Restful 接口集成。

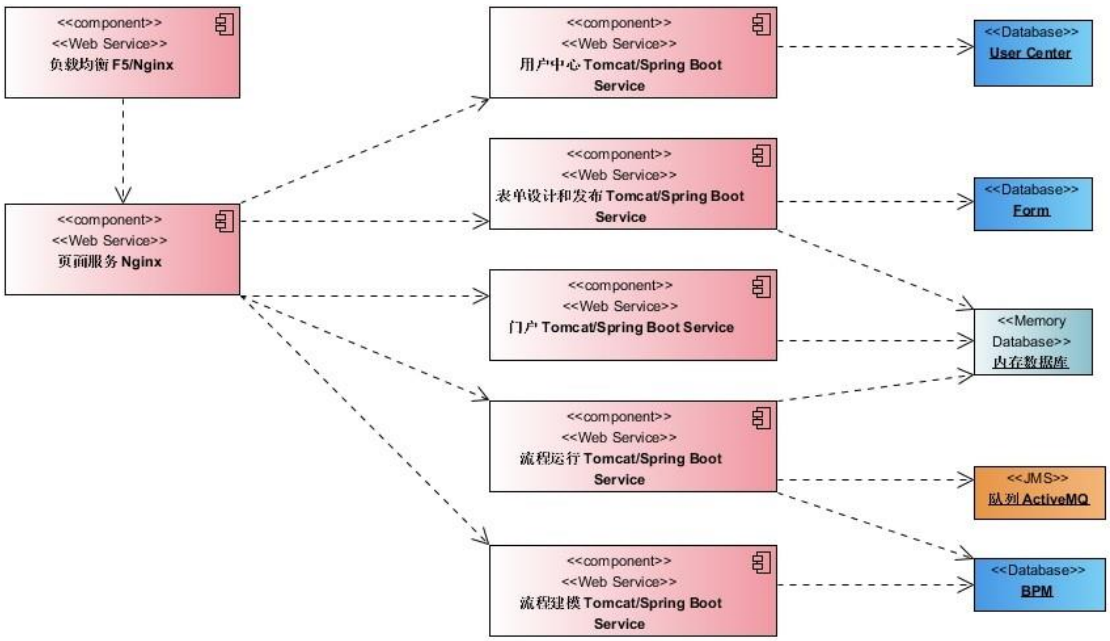
- **门户集成平台**

提供可视化门户布局、门户栏目设计，通过门户栏目可以方便快捷的集成各个系统的数据进行统一的展示、提供统一入口等。

- **快速开发平台**

提供业务功能快速开发，以可插拔组件为核心实现业务构建自动化，在可视化环境中创建可观察、可管理的企业级应用。

1.1 总体结构图



1. 负载均衡 F5/Nginx
   通过 F5 或者 Nginx 提供负载均衡的服务，在部署了多个页面服务时，根据各服务的压力适应性的分发请求到相应的节点上。
2. 页面服务 Nginx
   提供页面服务及后台服务转发，页面服务即系统的前端页面（包括 html、JavaScript、css、图片等资源）；后台服务转发即对后台的服务（在本平台中按照 restful 标准提供后台服务）进行转发，当部署了 API 网关时可将后台服务转发的功能转移到 API 网关中来实现。
3. 用户中心 Tomcat/Spring Boot Service
   提供统一的用户组织管理功能。
4. 表单设计和发布 Tomcat/Spring Boot Service
   提供业务建模、表单设计、表单运行发布的功能。
5. 门户 Tomcat/Spring Boot Service
   提供栏目管理、布局管理的功能。
6. 流程建模 Tomcat/Spring Boot Service
   提供流程设计、流程配置的功能。
7. 流程运行 Tomcat/Spring Boot Service
   提供流程发起、待办查询、任务审批、实例管理、历史查询的功能。

名词	角色	说明
F5	负载均衡	硬件实现的负载均衡
Nginx	负载均衡	软件实现的负载均衡
	Web 服务器	Apache 也可以代替它作为 Web 服务器
	转发服务器	
Spring Boot Service	Java 应用微服务	微服务可以直接在安装了 jdk 的物理机/虚拟机上部署运行； 微服务可以在 Spring Cloud 或 Dubbo 搭建的微服务运行环境中部署； 微服务也可以结合 docker 容器来部署，通过 docker 镜像的实例化实现部署，可以做到自动伸缩。

1.2 运行环境

- JDK1.8 及以上
- 数据库：MySQL5.7 及以上、Oracle 9i 及以上、SQLServer 2008 及以上

1.3 开发技能要求

- 熟悉 Java 编程的基本概念：面向对象、接口、继承等；
- 了解关系型数据库的 SQL 语法；
- 有一定的 Html、JavaScript、CSS 基础；
- 了解 Maven 构建项目的基础知识

2 构建开发环境

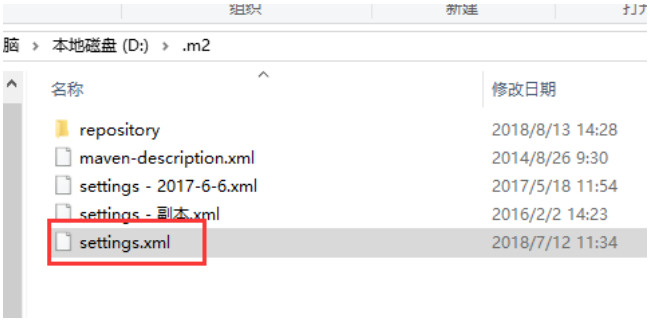
2.1 准备开发环境

- **安装 JDK**

安装 JDK1.8 或以上版本，并配置 JAVA_HOME 变量。

- **配置 Maven**

安装 Maven 并配置好本地仓库的位置，建议在配置文件中添加阿里云的镜像地址提升 Maven 构建时的效率。



修改如上图所示的配置文件 settings.xml：

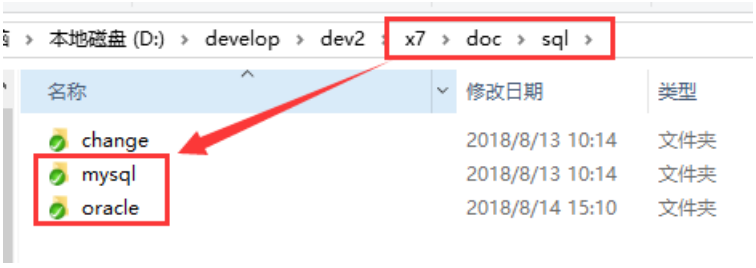
```
<?xml version="1.0" encoding="UTF-8"?>
<settings>
      <localRepository>D:/.m2/repository</localRepository>
      <mirrors>
            <mirror>
                  <id>internal-repository</id>
                  <name>Internal Repository Manager</name>

<url>http://maven.aliyun.com/nexus/content/groups/public/</url>
                  <mirrorOf>central</mirrorOf>
            </mirror>
            <mirror>
                  <id>internal-repository-thirdparty</id>
                  <name>internal-repository-thirdparty</name>

<url>http://maven.aliyun.com/nexus/content/groups/public/</url>
                  <mirrorOf>thirdparty</mirrorOf>
            </mirror>
      </mirrors>
</settings>
```

- **安装开发 IDE**

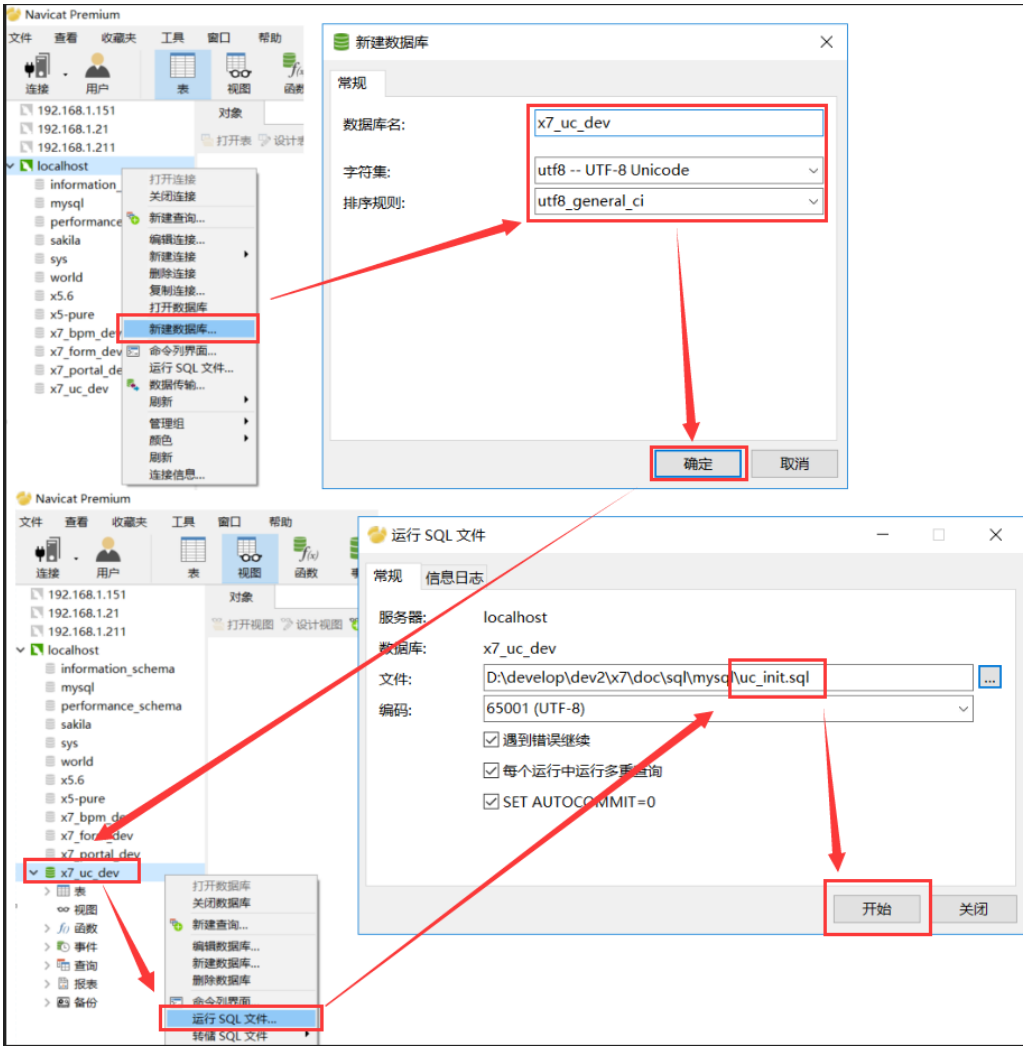
安装开发 IDE 工具并配置好 Java 环境和 Maven 环境，常见的开发工具有 Eclipse、MyEclipse、NetBeans、IntelliJ IDEA。

- **安装数据库**

安装数据库并初始化 EIP 的四个数据库，数据库的初始化脚本在如下的目录中：

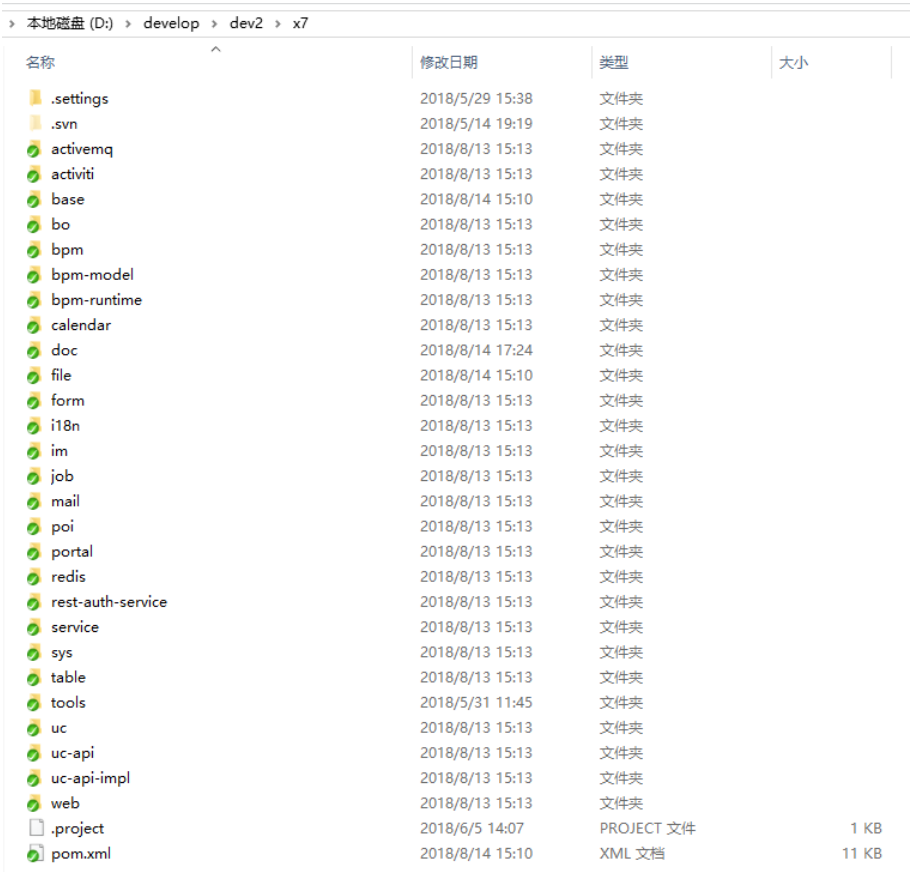


按照如下步骤新建数据库并初始化数据，注意数据库的字符集设置为 UTF8，总共有四个数据库需要创建：uc、form、bpm、portal。

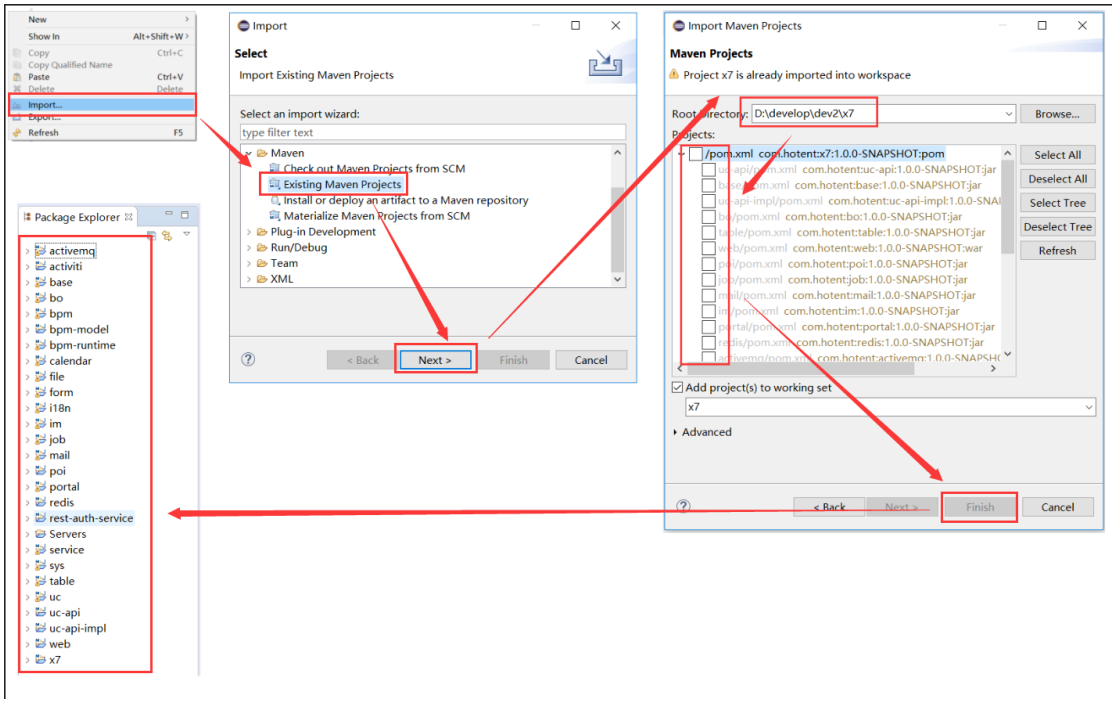


2.2 导入 EIP 项目代码

如下图所示，为 EIP 的整个项目代码结构：



通过如下步骤将整个项目导入到开发工具中：

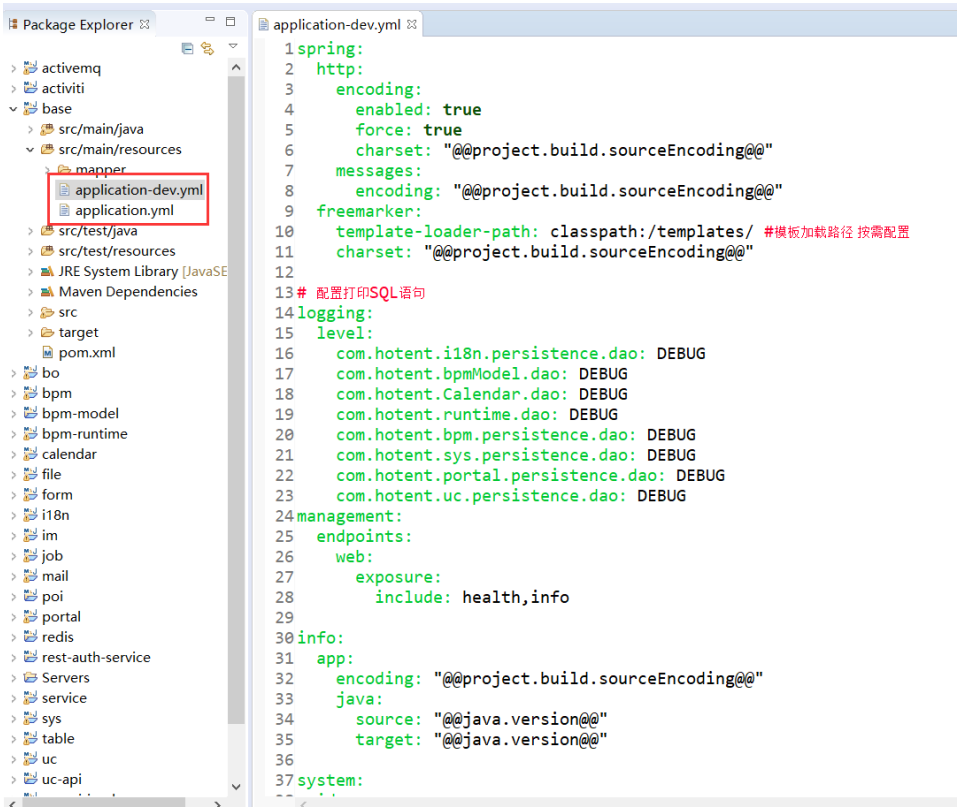


2.3 修改配置文件

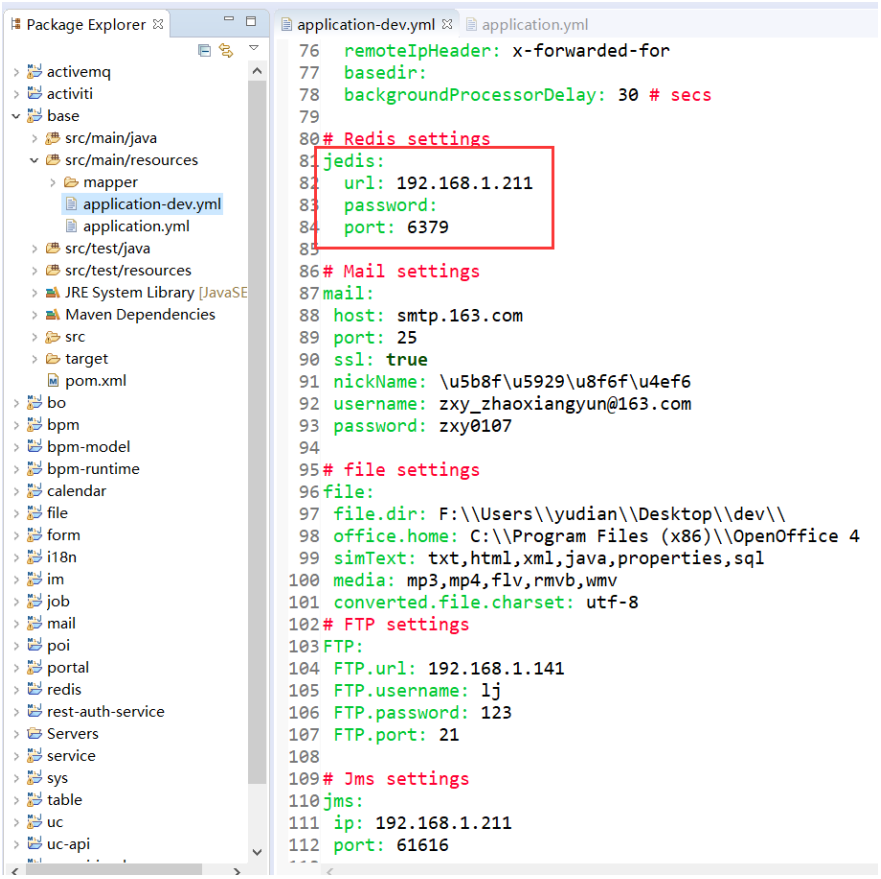
导入项目时会自动加载 Maven 依赖，这可能需要一些时间，等所有的依赖加载完成以后，修改配置文件，配置文件主要有三类：base 配置、服务配置和前端配置。

- **Base 配置**

Base 配置是通用配置，这里的配置五个微服务都会用到，如下图所示：

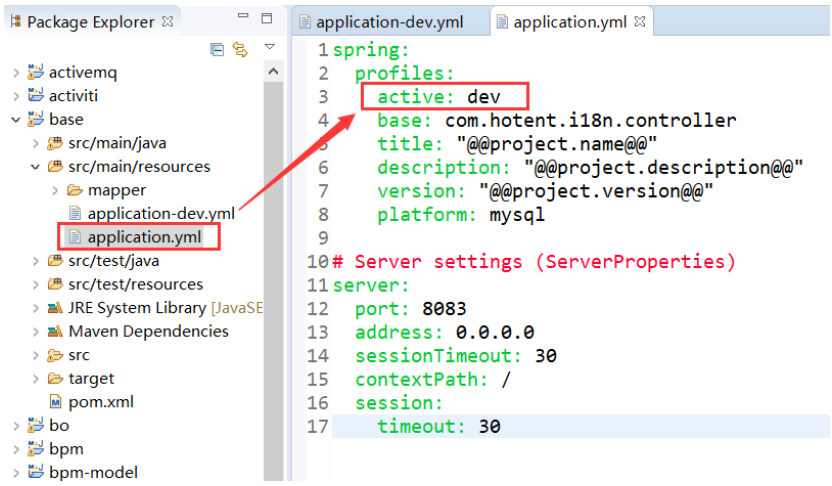


这个配置文件中配置了很多属性及依赖的中间件，其中 redis 地址的配置是必须配置准确才能正常启动系统的，如下图所示，其他的配置根据具体情况进行修改。



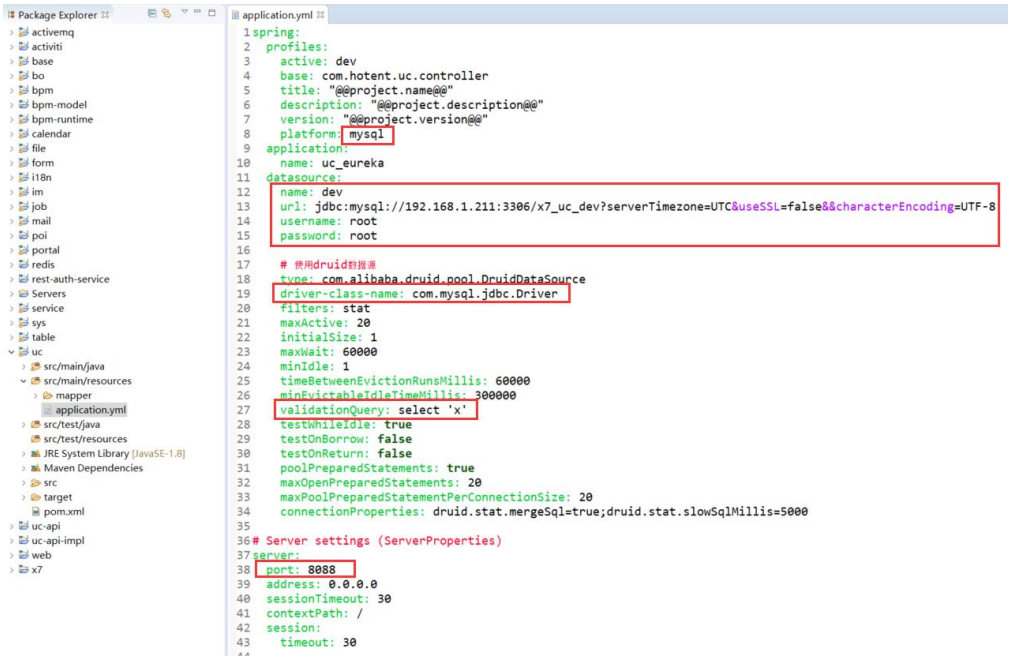
在实际的开发过程中可以创建多个配置文件，例如：application-dev.yml 和 application-test.yml，前者用在开发环境后者用在测试环境。

配置文件使用 application.yml 中的 spring.profiles.active 属性来切换或者在运行微服务时通过命令 java -jar uc.jar –spring.profiles.active=test 来切换。



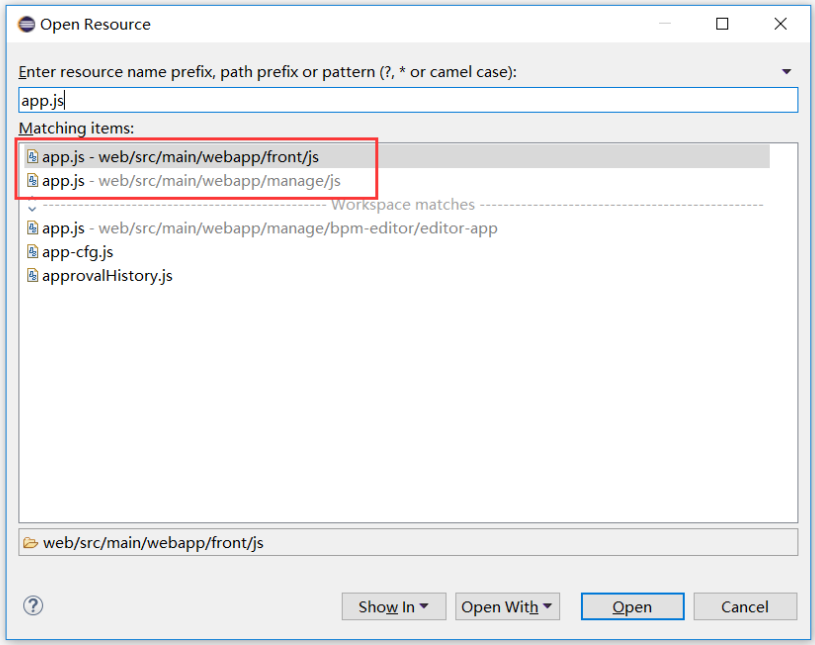
- **服务配置**

服务配置是指五个微服务的专属配置，例如 uc 服务的配置文件如下所示，核心配置属性为数据库类型、地址、用户名、密码、驱动程序、端口等属性：



- **前端配置**

因为 EIP 为前后分离架构，所以前端再访问后端微服务时需要指定 IP/域名和端口，如下图所示，注意前端分为管理端和应用端，相应的配置文件也有两份。



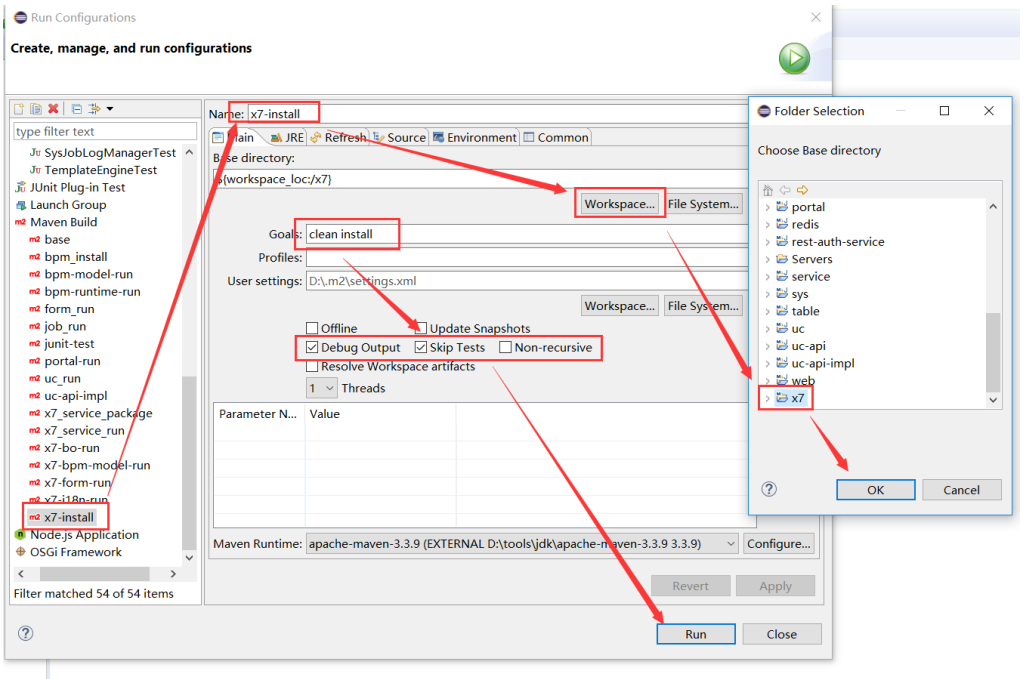
app.js

```
(function () {
    // 返回后台的context path
    window.getContext = function(){
        return {
            web : 'http://127.0.0.1:8080/manage',
            portal: 'http://127.0.0.1:8084',
            bpmRunTime: 'http://127.0.0.1:8086',
            bpmModel: 'http://127.0.0.1:8087',
            uc:'http://127.0.0.1:8088',
            form:'http://127.0.0.1:8082'
        };
    }

    angular.module('eip', [
        'ui.router',                    // Routing
        'oc.lazyLoad',                  // ocLazyLoad
        'ui.bootstrap',                 // Ui Bootstrap
        'pascalprecht.translate',       // Angular Translate
        'ngIdle',                       // Idle timer
        'ngSanitize',                   // ngSanitize
        'toaster',                      // toaster
        'ngStorage',                    // localStorage
        'ncy-angular-breadcrumb'        // breadcrumb
    ])
})();
```

2.4 启动项目

- **项目编译**

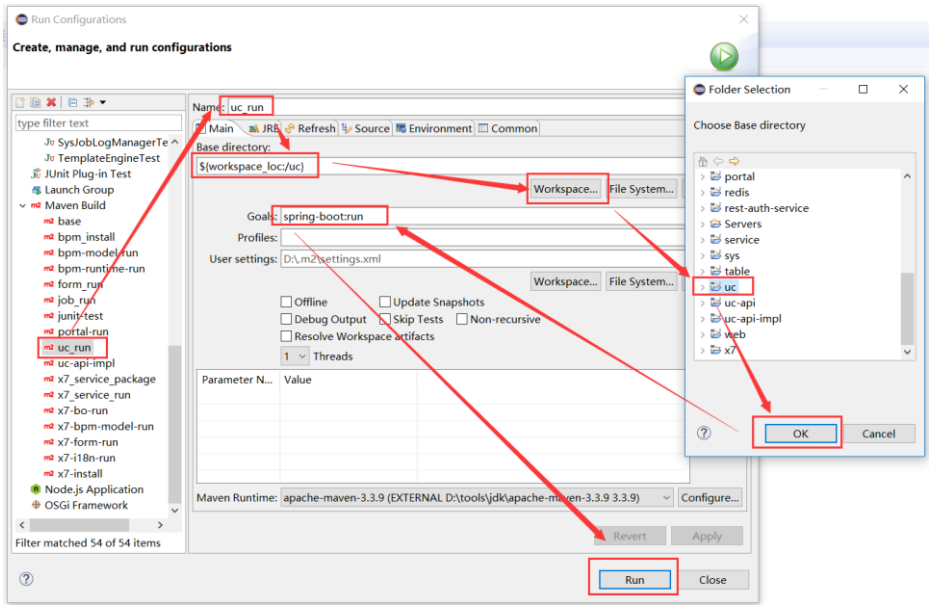
如下图所示，对整个 EIP 项目执行 maven install 命令即可与完成对整个项目的编译和发布。在开发过程中，修改了任何类库包的代码和配置时都需要重新编译和发布。



- **启动微服务**

编译好整个项目以后，需要启动五个微服务，以 uc 为例，按照如下步骤来

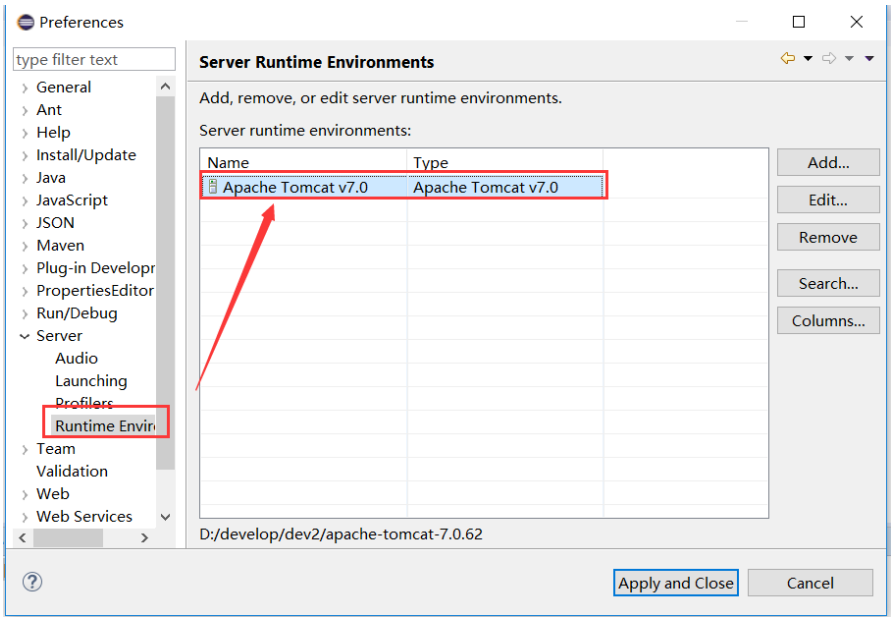
启动微服务：



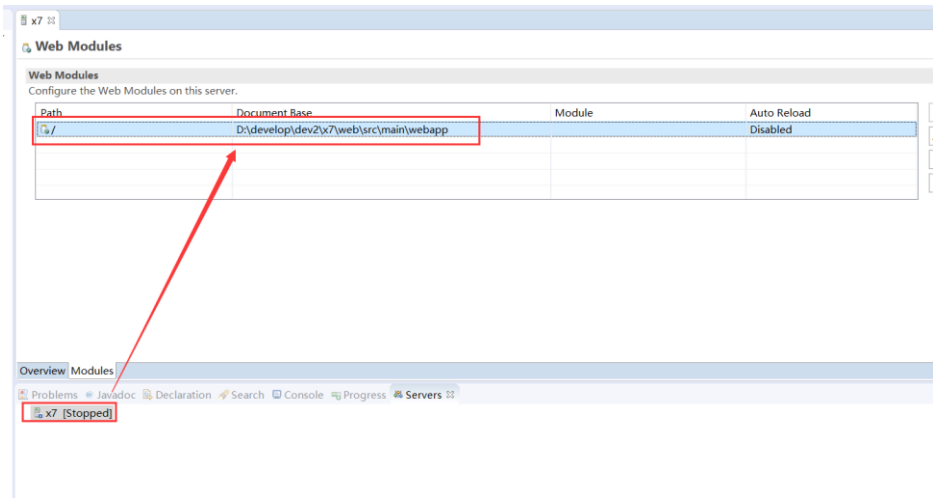
➢ **启动页面服务**

页面服务为纯 html 项目，无需依赖 Java 运行环境，所以我们可以使用任何 web 服务器来启动，这里为了方便，我们仍然使用 Eclipse 中的配置 Tomcat 的方式来启动，过程如下：

首先，确认开发工具中配置了 Tomcat 环境。

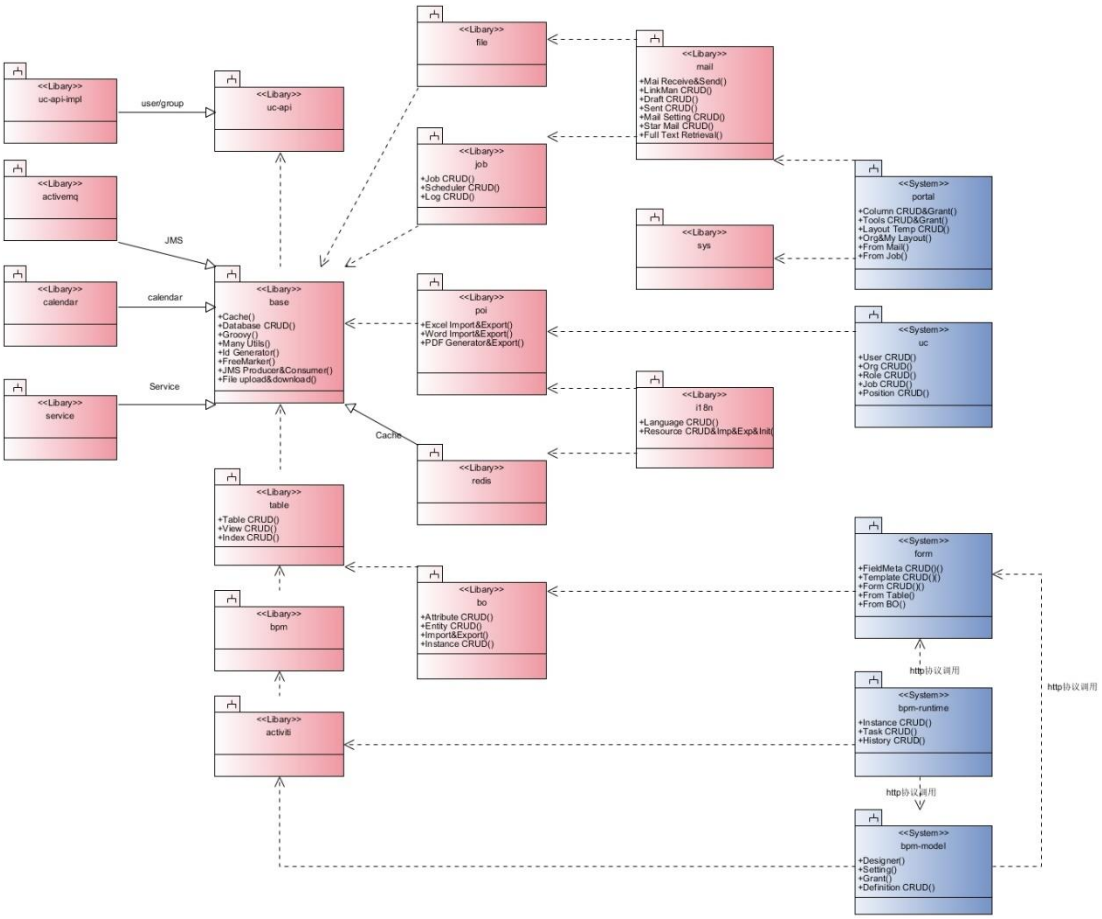


然后在 Servers 中添加一个新的 server 并将发布目录指向 web 项目中的 webapp 目录，接着启动这个 server。现在我们可以在浏览器端访问 EIP 平台了。


x7
Web Modules
Web Modules
Configure the Web Modules on this server.
Path
Document Base
Module
Auto Reload
/
D:\develop\dev2\x7\web\src\main\webapp
Disabled
Overview
Modules
Problems
Javadoc
Declaration
Search
Console
Progress
Servers
x7 [Stopped]

3 平台架构

3.1 模块划分



在上图中，红色模块为类库包，蓝色的为系统包。在下表中通过对比几个方面来描述两者之间的区别与联系。

	类库包（红色）	系统包（蓝色）
角色	不实现具体的业务功能，每个类库包只提供相对单一的通用功能，例如：缓存处理、附件处理、物理表操作等等。	提供具体的业务功能，比如设计一张表单、配置一个流程、添加一个用户等等。
标准	按照 Java interface 提供接口或者直接提供实现类供其他包使用	按照 http 标准提供接口服务
原则	1. 可以依赖另外一个类库包，不能依赖系统包； 2. 可以实现另外一个类库包中的接口； 3. 依赖关系只能单向，不能相互依赖，也不能循环依赖。	1. 只能依赖类库包，不能依赖另外的系统包； 2. 需要用到其他系统包的接口服务时，通过调用 http 服务实现。

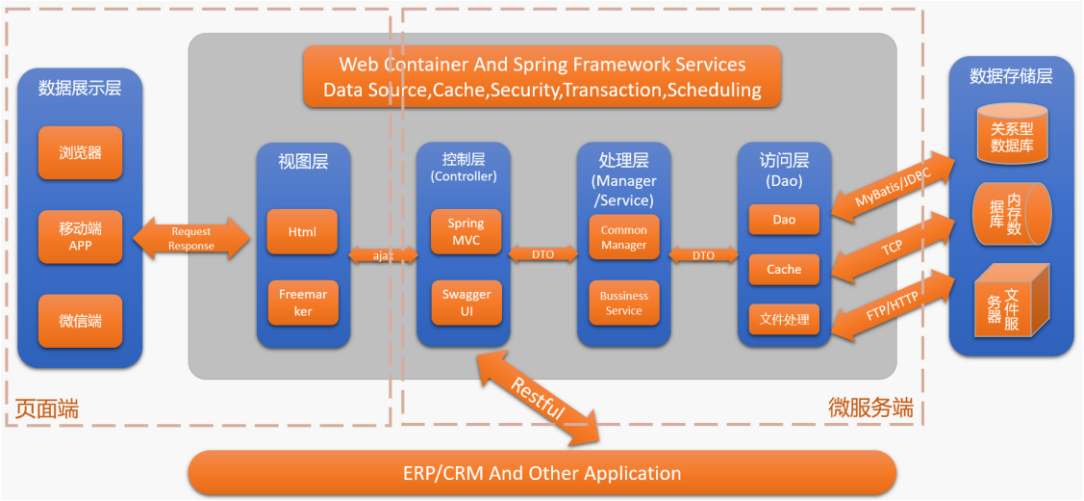
建议	1. 每个包职责相对单一； 2. 一个通用功能有多种实现方式时，定义为接口，通过独立的包来实现接口。	有一些类库包的部分功能也需要发布为 http 接口服务，可以在类库包中实现功能（实现 rest controller），在系统包中装配（扫描对应的 controller）

3.2 技术框架

整个平台采用微服务架构，使用 Spring Boot 2.X 作为基础框架，其他主要框架如下表所示：

框架	版本	说明
Spring Boot 系列	2.0.1.Release	微服务基础框架
Spring Cloud 系列	2.0.1.Release	微服务治理框架
Spring Web 系列	5.0.5.Release	Web 框架
MyBatis	3.4.5	ORM 框架
PageHelper	5.1.3	分页框架
Druid	1.1.8	数据库连接池
Jackson	2.9.5	Json 框架
Groovy	2.1.6	动态脚本框架
Logback	1.2.3	日志框架
Freemarker	2.3.28	模板框架
Swagger	2.8.0	API 文档工具框架

3.3 分层结构



3.3.1 微服务端

微服务端采用了标准的 Spring MVC 三层结构，由 dao manager controller 标准的三层构成，另外在类库包中对外提供的接口封装了一层 service 方便其他类库包或系统包引用。

- **实体层**

如下所示，实体类均继承自 BaseModel，并指定泛型主键类型，实体类必须使用 swagger 的注解@ApiModel 说明该实体类的作用，所有的属性都用注解@ApiModelProperty 进行描述，对于必填的属性需要在注解中注明 required=true，对于有可选值得属性可以指定 allowableValues 进行值范围约束。

```java
/**
 * bo定义
 *
 * @company 广州宏天软件股份有限公司
 * @author heyifan
 * @email heyf@jee-soft.cn
 * @date 2018年4月12日
 */
@ApiModel(description="bo定义")
public class BoDef extends BaseModel<String> {
    private static final long serialVersionUID = 1L;

    @ApiModelProperty(name="id", notes="主键")
    protected String id="";

    @ApiModelProperty(name="alias", notes="别名", required=true)
    protected String alias="";

    @ApiModelProperty(name="description", notes="描述")
    protected String description="";

    @ApiModelProperty(name="boEnt", notes="业务实体")
    protected BoEnt boEnt = new BoEnt();

    @ApiModelProperty(name="supportDb", notes="是否支持数据库(0:不支持  1:支持)", allowableValues="0,1")
    protected Short supportDb = 0;

    @ApiModelProperty(name="categoryId", notes="所属分类ID", required=true)
    protected String categoryId = "";

    @ApiModelProperty(name="status", notes="状态(normal:正常 forbidden:禁用)", allowableValues="normal,forbidden")
    protected String status = "normal";

    @ApiModelProperty(name="deployed", notes="是否发布(0:未发布 1:已发布)", allowableValues="0,1")
    protected Short deployed = 0;
}
```

- **dao 层**

mapper 文件的命名方式为 实体类名 Mapper.xml 的格式，统一放在每个包的 src/main/resources/mapper 目录下面。mapper 文件中 namespace 指向 dao 类，dao 的命名方式为实体类名 Dao.java，如下图所示:

dao 文件为一个接口，不是一个实现类，这个接口继承自 MyBatisDao，在 MyBatisDao 中提供了基础的增删改查方法。dao 有两个泛型需要指定，第一个泛型是主键的类型，第二个泛型是对应的实体类。

```java
/**
 * 流程定义处理接口
 *
 * @company 广州宏天软件股份有限公司
 * @author heyifan
 * @email heyf@jee-soft.cn
 * @date Apr 28, 2018
 */
public interface BoDefDao extends MyBatisDao<String, BoDef> {
    /**
     * 根据别名获取定义
     * @param alias    别名
     * @return         返回bo定义
     */
    BoDef getByAlias(String alias);

    /**
     * 通过别名删除定义
     * @param alias    别名
     * @return         返回删除了几个定义
     */
    int removeByAlias(String alias);
}
```

- **manager 层**

manager 分为接口和实现类，manager 的命名方式为实体类名 Manager.java，例如 BoDefManager 接口和 BoDefManagerImpl 实现类，其中接口扩展自 Manager，实现类继承自 AbstractManagerImpl 类并实现 BoDefManager 接口。在接口和实现类中均需指定两个泛型，第一个为主键的数据类型，第二个为对应的实体类。

```java
/**
 * 业务实体定义属性 处理接口
 *
 * @company 广州宏天软件股份有限公司
 * @author heyifan
 * @email heyf@jee-soft.cn
 * @date 2018年4月12日
 */
public interface BoAttributeManager extends Manager<String, BoAttribute> {

    /**
     * 根据实体ID获取BO属性
     * @param entId    实体ID
     * @return         返回bo属性列表
     */
    List<BoAttribute> getByEntId(String entId);

    /**
     * 根据实体ID删除属性。
     * @param entId    实体ID
     */
    void removeByEntId(String entId);
}
```

- **controller 层**

controller 层的命名方式为实体类 Controller.java，继承自 BaseController，类

上有三个注解：

- @RestController 表明其是一个 restful 控制类
- @RequestMapping("/bo/def/v1/") 标明了其映射的路径
- @Api(tags="BoDefController") 用以生成 swagger 在线接口文档

在每一个方法上需要添加两个注解:

- @RequestMapping(value="list", method=RequestMethod.POST, produces={"application/json; charset=utf-8" }) 该方法对应的地址，该方法对应的 restful 请求类型及支持的数据格式
- @ApiOperation(value = "获取 bo 定义列表（带分页信息）", httpMethod = "POST", notes = "获取 bo 定义列表" 用于生成 swagger 文档

列表方法使用 QueryFilter 作为入参，该参数封装了通用查询和分页信息；返回值为 PageList<?>，该参数的泛型为对应的实体类，在该返回值中包含了分页信息和列表数据。

方法可以直接 throws 异常信息，异常消息的处理和记录会统一在 BaseController 中进行。

```
/**
 * bo定义控制器
 *
 * @company 广州宏天软件股份有限公司
 * @author heyifan
 * @email heyf@jee-soft.cn
 * @date 2018年4月18日
 */
@RestController
@RequestMapping("/bo/def/v1/")
@Api(tags="BoDefController")
public class BoDefController extends BaseController {
    @Resource
    BoDefManager boDefManager;
    @Resource
    BoEntRelManager boEntRelManager;
    @Resource
    BoEntManager boEntManager;
    @Resource
    BoDataHandler boDataHandler;

    @RequestMapping(value="list", method=RequestMethod.POST, produces={"application/json; charset=utf-8" })
    @ApiOperation(value = "获取bo定义列表（带分页信息）", httpMethod = "POST", notes = "获取bo定义列表")
    public PageList<BoDef> listJson(@ApiParam(name="queryFilter",value="通用查询对象")@RequestBody QueryFilter queryFilter) throws Exception {
        return boDefManager.query(queryFilter);
    }

    @RequestMapping(value="detail",method=RequestMethod.GET, produces = { "application/json; charset=utf-8" })
    @ApiOperation(value = "获取bo定义详情", httpMethod = "GET", notes = "获取bo定义详情")
    public Object detail(@ApiParam(name="alias",value="bo定义别名", required = true) @RequestParam String alias) throws Exception{
        return boDefManager.getPureByAlias(alias);
    }

    @RequestMapping(value="save",method=RequestMethod.POST, produces = { "application/json; charset=utf-8" })
    @ApiOperation(value = "添加bo定义", httpMethod = "POST", notes = "添加bo定义")
    public Object save(@ApiParam(name="json",value="bo定义的json数据", required = true) @RequestBody String json) throws Exception{
        boDefManager.save(json);
        return modelMap("添加成功");
    }

    @RequestMapping(value="remove",method=RequestMethod.DELETE, produces = { "application/json; charset=utf-8" })
    @ApiOperation(value = "删除bo定义", httpMethod = "DELETE", notes = "删除bo定义")
    public Object remove(@ApiParam(name="alias",value="bo定义别名", required = true) @RequestParam String alias) throws Exception{
        int removeItems = boDefManager.removeByAlias(alias);
        Map<String, Object> map = modelMap("删除成功");
        map.put("count", removeItems);
        return map;
    }
}
```

- **特别说明**

1. QueryFilter 和 PageBean 等辅助查询对象只到 manager 层，不允许在 dao 层中使用；
2. PageBean 和 Page 都是分页辅助类，前者用作分页查询使用，后者为返回分页信息使用，构建分页查询条件时使用 PageBean，不允许使用 Page;
3. 实体类需要转为 json 数据，json 数据也需要实例化为实体类，整个项目使用 jackson 作为 json 处理框架，不允许再引入 gson 或 fastjson;
4. 实体类与 json 的转换过程中，有属性不参与转换时在该属性的 get 方法上添加@JsonIgnore 注解;

```
@JsonIgnore
public BoEnt getBoEnt() {
 return boEnt;
}
```

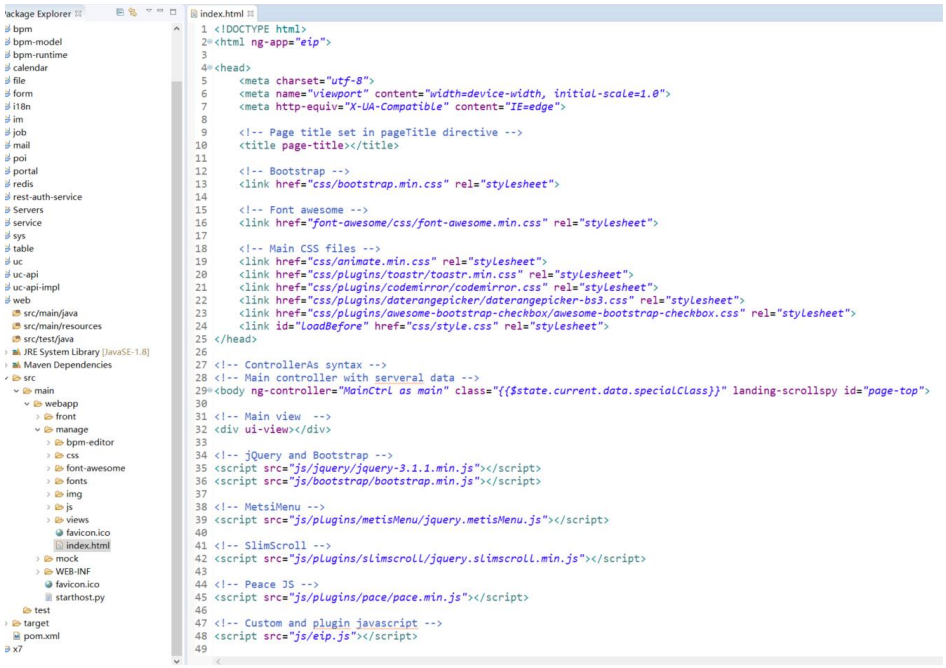
5. 实体类与 json 转换中，有的属性需要另外命名为其他名称时使用@JsonProperty 注解。

```
@JsonProperty("attrs")
public List<BoAttribute> getBoAttrList() {
 return boAttrList;
}
```

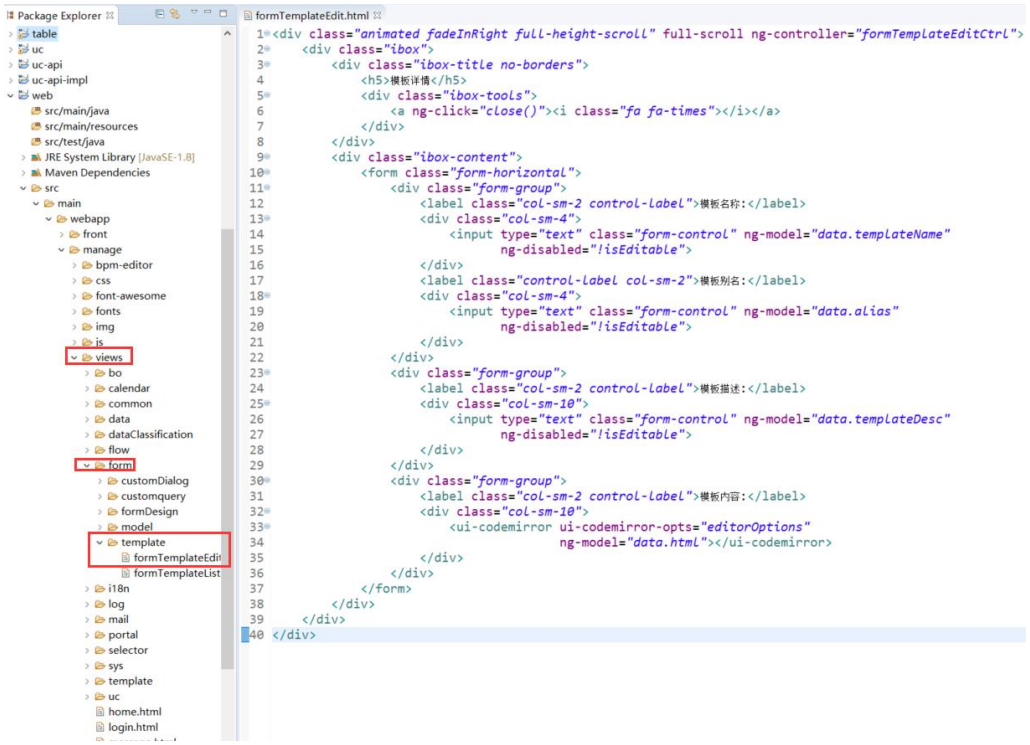
3.3.2 页面端

前端页面采用单页面应用的结构，样式框架采用 Bootstrap3 + Font-awesome 图标库。JavaScript 框架采用了 AngularJs 1.5 + jQuery 3.1.1。

整个系统只有一个 index.html 是完整的页面，通用的 css 文件和 js 文件在该页面引入，如下图所示：



其他的功能页面只有 html 片段，通过 ui-router 实现前端路由。功能页面的 html 代码如下图所示。



在 ui-router 中注册路由时，可以指定页面需要加载的 js、css 资源，这里指定的资源采用懒加载的方式，只有访问到该页面的时候才会加载这些资源。

```
.state('form.templateEdit', {
    url: "/templateEdit",
    templateUrl: "views/form/template/formTemplateEdit.html",
    data: { pageTitle: '表单模板编辑', specialClass: 'fixed-sidebar' },
    resolve: {
        loadPlugin: function ($ocLazyLoad) {
            return $ocLazyLoad.load([
                {
                    name: 'form',
                    files: ['js/form/formControllers.js']
                },
                {
                    name: 'ui.codemirror',
                    files: ['js/plugins/ui-codemirror/ui-codemirror.min.js']
                }
            ]);
        }
    }
})
```

4 快速开始

本章将实现两个简单案例，来直接体验使用 EIP 开发的基本过程。

4.1 办公用品库存管理

在微服务架构的体系里面，开发业务功能时应该按照业务需求切分为不同的微服务，例如开发办公用品管理时可以构建一个办公管理的微服务，不过本章节只是一个快速示例，所以我们在已有的微服务：门户(portal)项目中开发这个小功能。至于如果构建一个新的微服务，我们将在后面的章节应用定制中详细描述。

4.1.1 生成物理表

按照如下结构在数据库 x7_portal_dev 中创建一张物理表：办公用品表

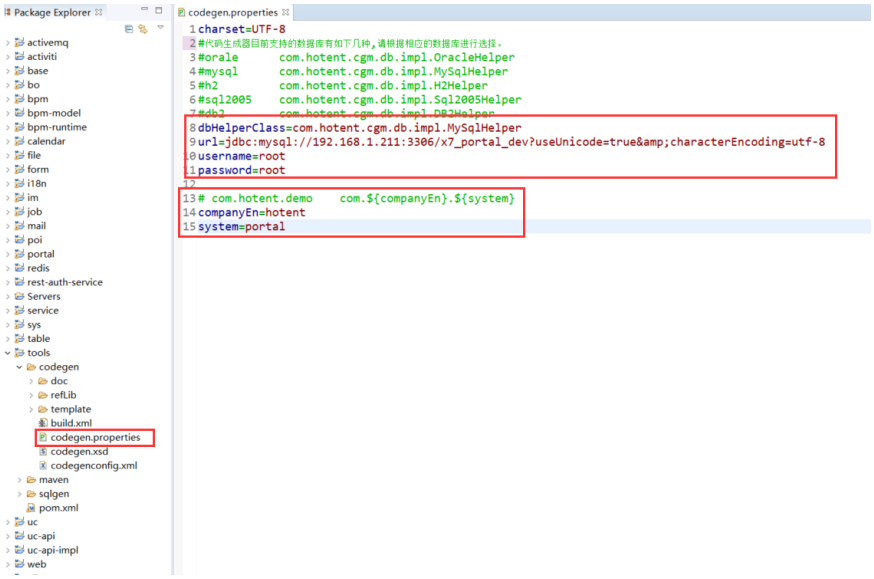
```
CREATE TABLE office_supplies (
    ID_    varchar(64) NOT NULL COMMENT '主键',
    NAME_    varchar(255) NULL COMMENT '名称',
    TYPE_    varchar(64) NULL COMMENT '类型',
    UNIT_    varchar(40) NULL COMMENT '计量单位',
    AMOUT_    integer(20) NULL COMMENT '库存数量',
    WORTH_    decimal(20,2) NOT NULL COMMENT '单件价值',
    PRIMARY KEY (ID_)
) COMMENT='办公用品';
```

4.1.2 代码生成

创建了物理表以后，通过代码生成器来生成代码，找到 tools 项目中的 codegen 目录。

- **基础配置**

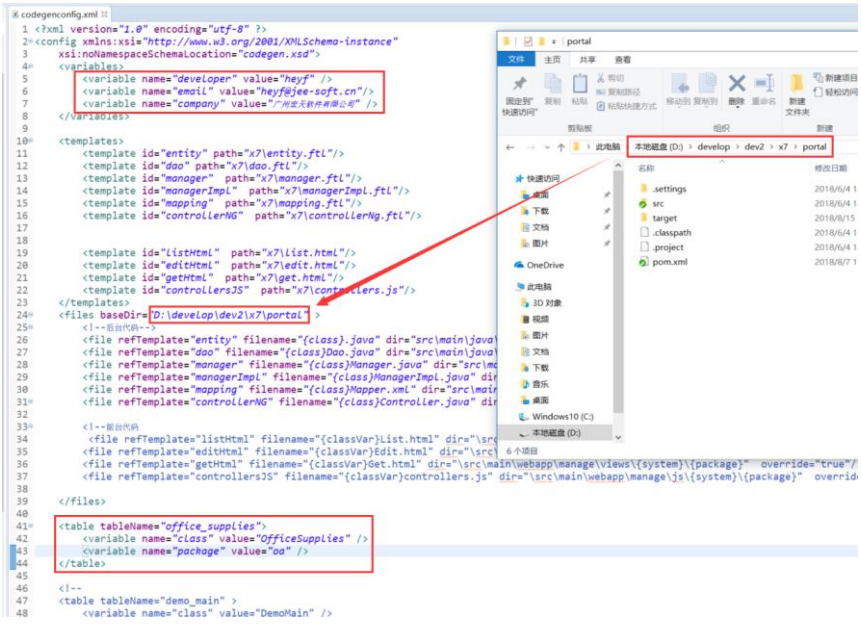
如下图所示，是代码生成的基础配置，这里需要配置数据库的连接信息、公司简称、项目简称等信息。



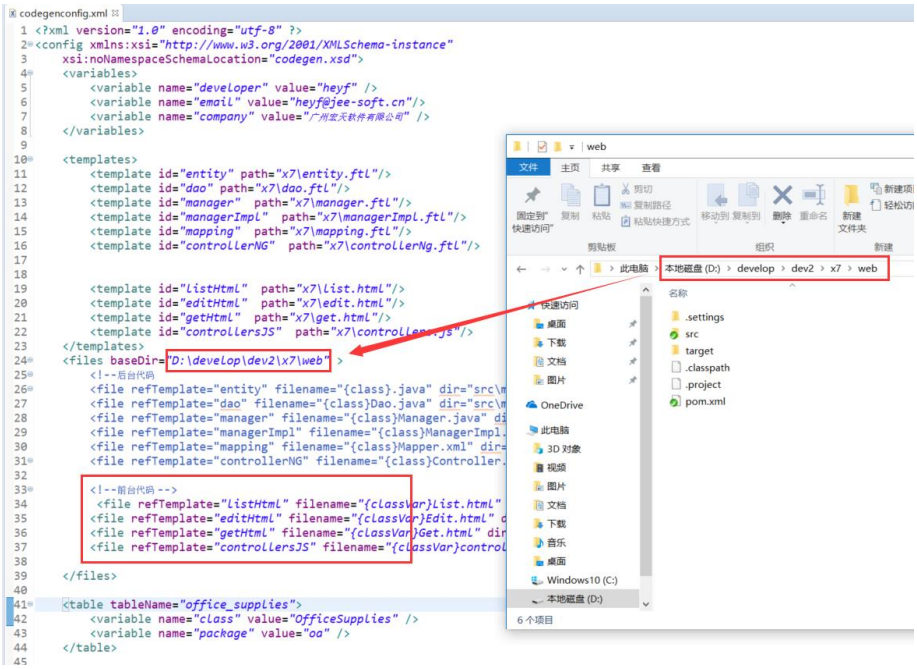
➢ 代码生成配置

代码的生成分为后台代码和前台代码，因为前后分离的设计模式，所以代码的生成需要分开生成，分别生成到不同的目录中。

如下图所示，在配置文件中修改开发人员信息、生成到指定目录、连接的数据库表、类名、包名等信息。

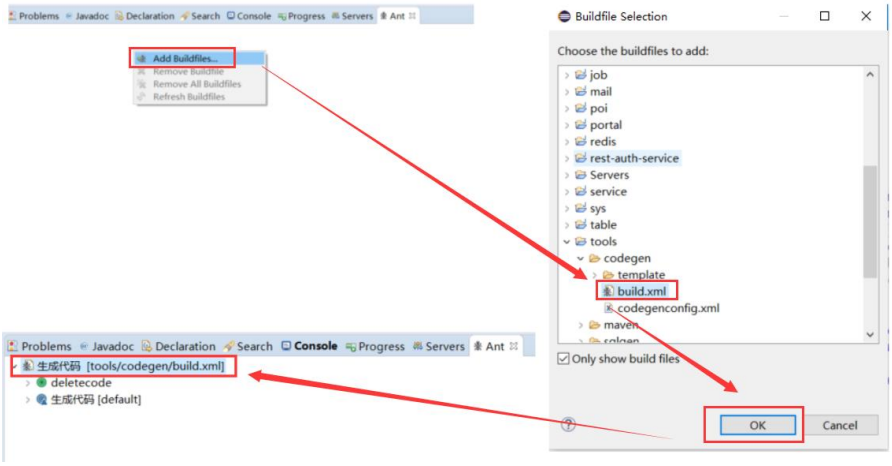


相应的，前端代码生成时按照如下图所示修改配置：

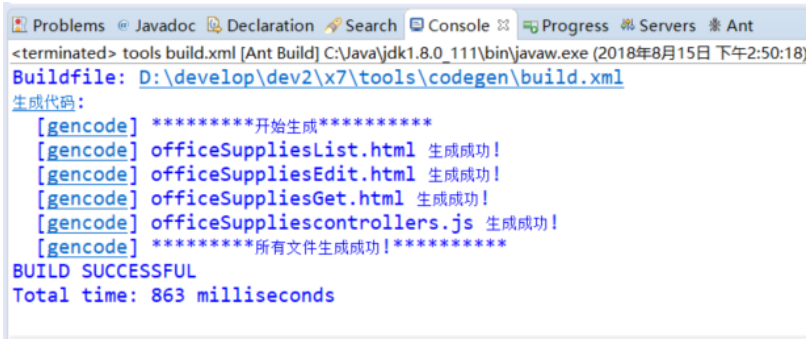


- **生成代码**

修改好配置以后，打开 ant 脚本面板，将 codegen 目录下的 build.xml 文件添加到 ant 面板。



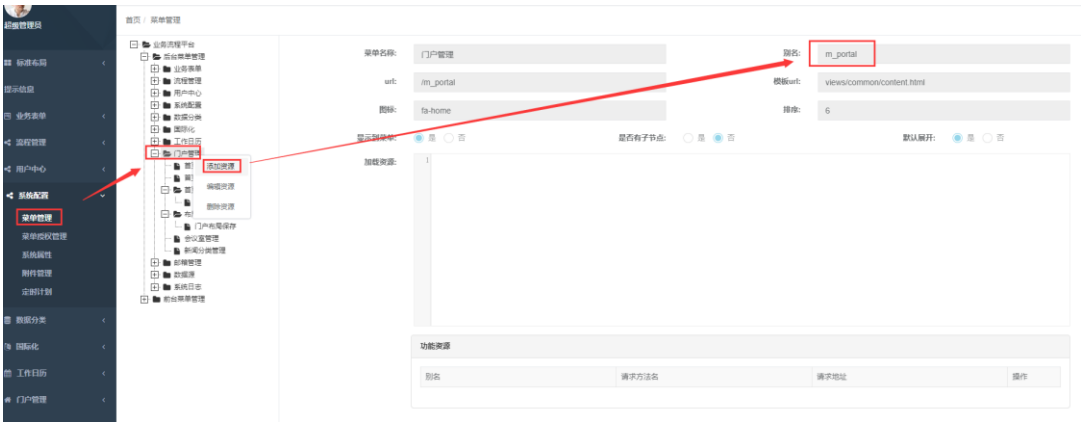
双击生成代码的选项即可进行生成代码的操作，通过控制台面板可以查看生成结果。在代码生成的目标目录，刷新就可以查看到新生成的代码文件。



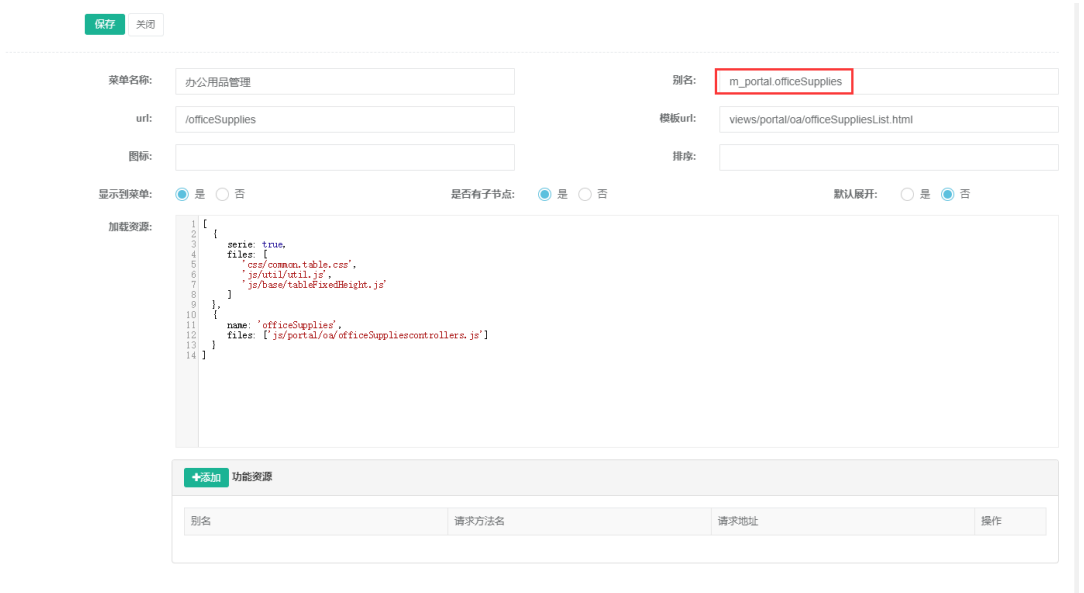
4.1.3 配置路由

代码生成以后，后台的微服务需要重启，前台服务不需要重启，但是在调整 Html、JavaScript 代码的过程中要注意浏览器端的缓存。

路由的注册需要在系统的菜单管理中进行，如下图所示，例如我要将办公用品管理放置在门户管理这个目录下，则右键门户管理这个选项，在弹出的右键菜单中选择添加资源，注意门户管理的别名是 m_portal。



选择添加资源时打开了一个编辑界面，按照如下格式填写办公用品的资源信息，注意别名以上级资源的别名为前缀的，中间用‘.’连接。



模板 URL 填写上代码生成器生成的 list 页面地址，在加载资源中 common.table.css 和 tableFixedHeight.js 是列表页面的通用资源，固定的引入即可，下面的 officeSuppliescontrollers.js 是代码生成器生成的当前功能页面所对应的 js

文件，而且是按照 AngularJs 的规则生成的 js 代码，所以在引入该 js 文件时必须要指定其 name，name 的值可以查看该 js 文件获得，如下图所示。

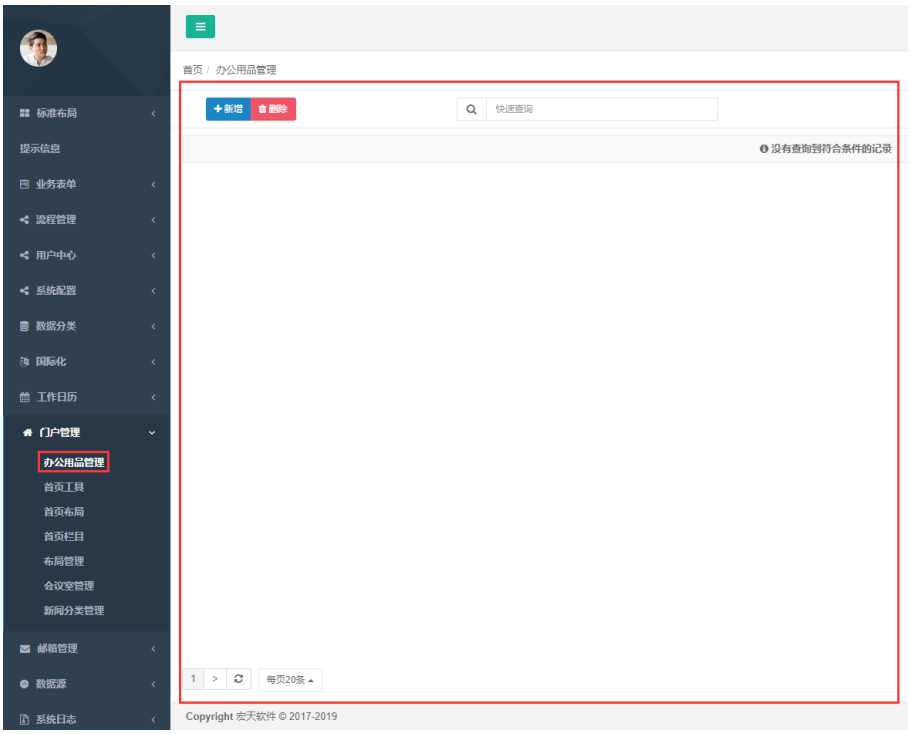
officeSuppliesList.html officeSuppliescontrollers.js

```
            });
        }
    }

}

//  办公用品 编辑明细
function officeSuppliesEditCtrl($scope, $compile, baseService,dialogService){
    //关闭
    $scope.close = function(){
        dialogService.closeSidebar();
    }
    $scope.title = $scope.pageParam.title;

    $scope.data = {
    }

    if($scope.pageParam && $scope.pageParam.id){
        baseService
        .get("${portal}/portal/officeSupplies/v1/get/"+$scope.pageParam.id)
        .then(function(response) {
            $scope.data = response;
        });
    }

    //保存
    $scope.save = function(){
            baseService
            .post("${portal}/portal/officeSupplies/v1/save",$scope.data)
            .then(function(response) {
                if(!response.state){
                    dialogService.fail(response.message);
                }else{
                    dialogService.success(response.message);
                    $scope.close();
                }
            });
    }
}

/**
 *
 * Pass all functions into module
 */
angular.module('officeSupplies', [])
    .controller('officeSuppliesEditCtrl', officeSuppliesEditCtrl)
    .controller('officeSuppliesListCtrl', officeSuppliesListCtrl)

```

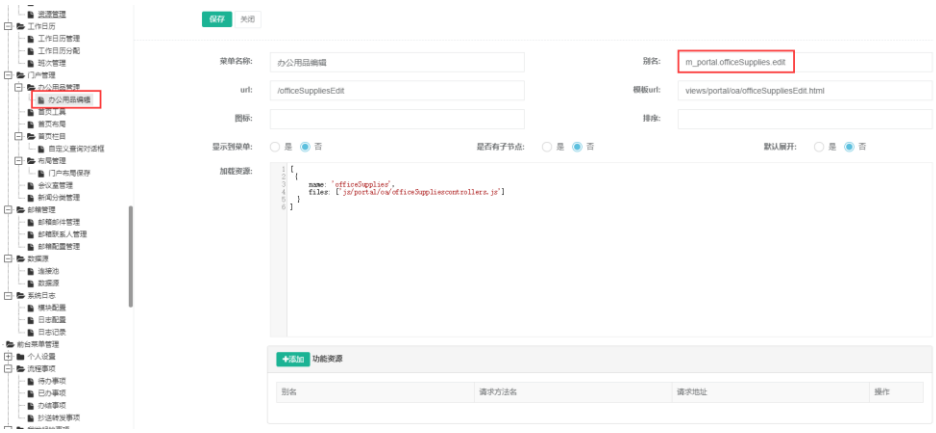
4.1.4 访问新功能

配置好路由以后，我们可以访问新的功能了，但是因为菜单资源是在登录时从后端获取并缓存起来的，所以要查看新的功能菜单，我们需要退出并重新登录系统。

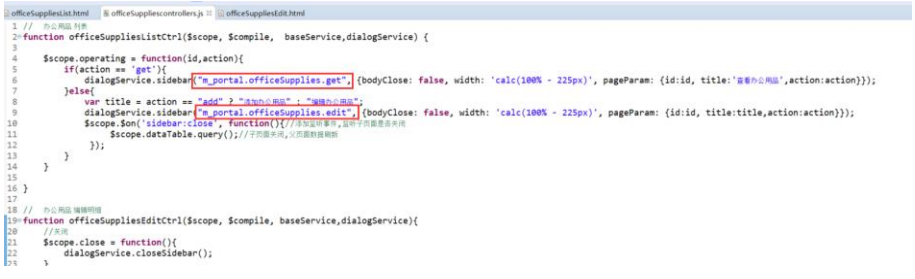
如下图所示，我们重新登录以后可以看到门户管理下面新增了一个办公用品管理的菜单，点击即可打开办公用品管理界面。



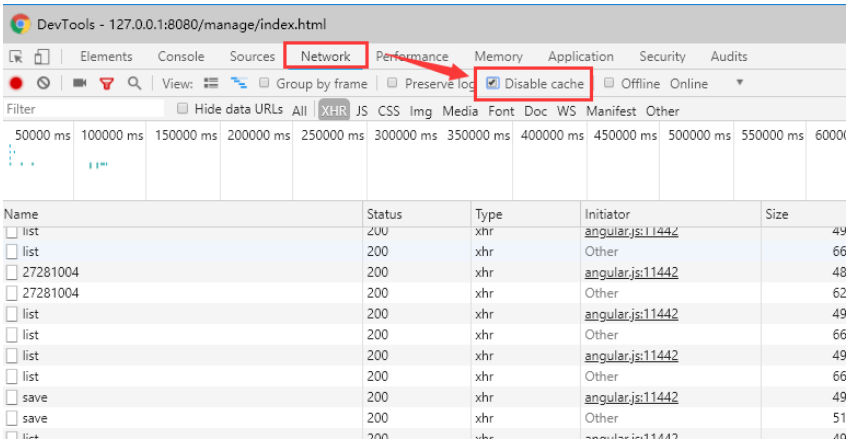
在数据库表为空的情况下是没有数据可以显示的，我们需要通过新增数据的功能录入一些数据，但是现在新增页面还未注册路由，所以点击新增是没有反应的，我们继续回到菜单管理中添加新增和查看详情的资源。



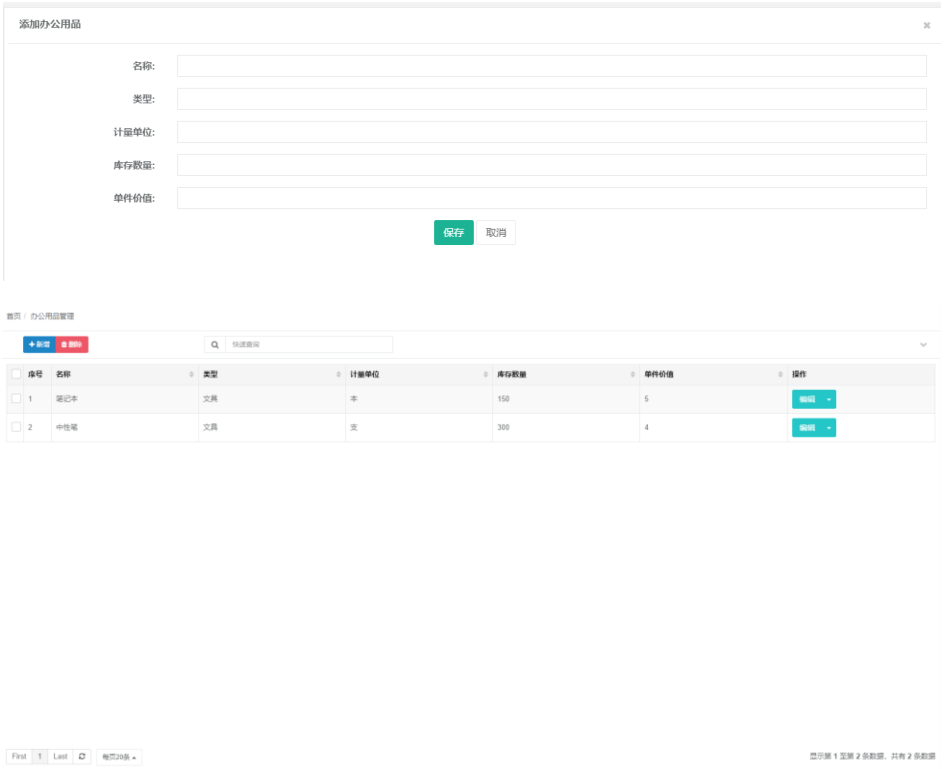
特别注意：这里注册的编辑页面的别名是 m_portal.officeSupplies.edit，而在列表页面点击添加和编辑按钮时是通过这个别名来找对应的资源的，所以需要将这个别名与 js 文件中调用的别名保持一致，如下图所示：



现在同样的退出系统重新登录，注意清理浏览器缓存，因为我们刚才修改的 js 文件可能会被浏览器缓存了。如果开发时使用的是 Chrome 浏览器，我们打开 F12 的调试窗口，在 network 那一栏中勾选 Disable cache 选项，则只要我们不关闭这个调试窗口，则浏览器就不会缓存任何 js 和 html 文件。



如下图所示，我们点击新增按钮即可打开编辑界面了，填写一些测试数据并保存，就可以在列表页面中查看到刚才新增的数据了。



至此，我们就完成了一个简单功能模块的开发，更多的功能我们可以在这些生成的代码上进行调整来实现。

4.2 办公用品领用流程

在上面的例子中，我们演示了开发一个功能模块，实现了基本的增删改查功能，现在我们演示开发一个功能模块并与流程进行对接，实现对办公用品的领用流程。

4.2.1 实现业务功能开发

开发办公用品领用的增删改查功能与上面的例子一致，这里不再复述，办公用品领用的表结构如下。

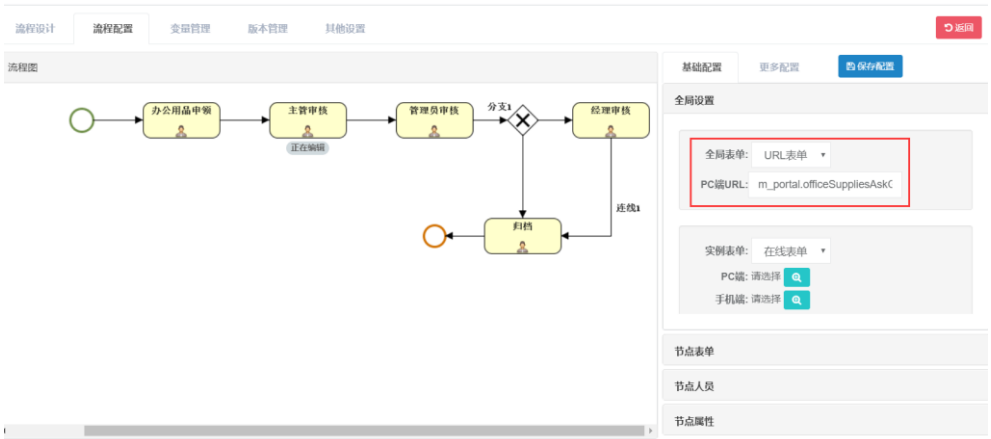
```
CREATE TABLE office_supplies_ask (
    ID_ varchar(64) NOT NULL COMMENT '主键',
    SUPPLIES_ID_ varchar(64) NOT NULL COMMENT '办公用品 ID',
    SUPPLIES_NAME_ varchar(255) NULL COMMENT '办公用品名称',
    ASK_MAN_ID_ varchar(64) NOT NULL COMMENT '领用人 ID',
    ASK_MAN_NAME_ varchar(255) NULL COMMENT '领用人姓名',
    AMOUT_ varchar(255) NULL COMMENT '数量',
    FLOW_INSTANCE_ID_ varchar(255) NULL COMMENT '关联流程实例 ID',
    STATUS_ smallint NULL COMMENT '状态',
    PRIMARY KEY (ID_)
) COMMENT='办公用品领用记录';
```

最终实现后的效果如下图所示：

4.2.2 配置流程

在流程定义中添加一个新的流程：办公用品领用流程，完成流程图设计、流程节点人员设置等基础配置。

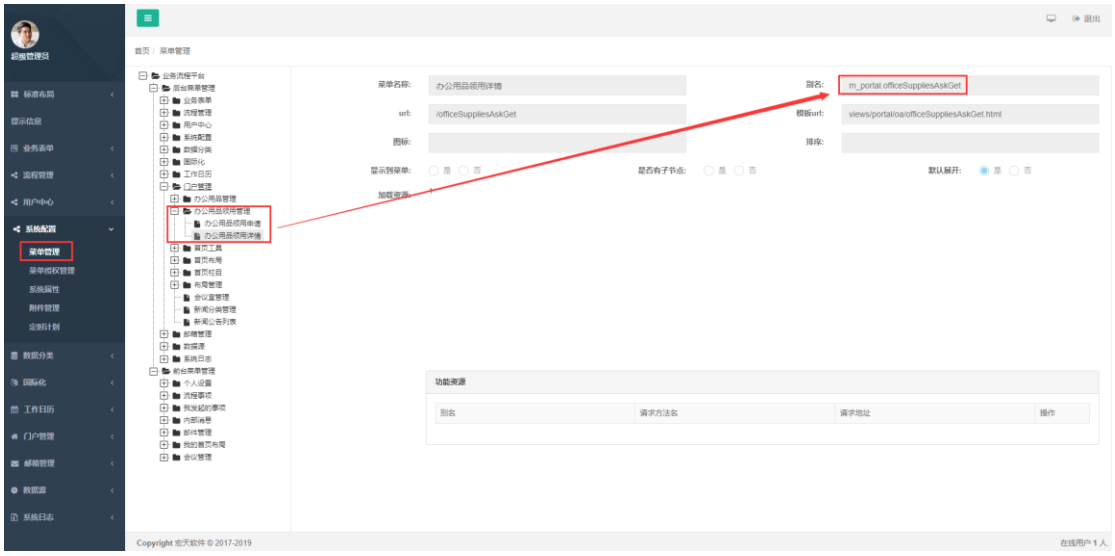
设置流程表单时，我们使用 URL 表单，URL 地址配置为我们的业务功能的页面地址，从而实现流程审批时能够通过 URL 地址加载表单界面，如下图所示：



URL 表单的配置有两种情况：

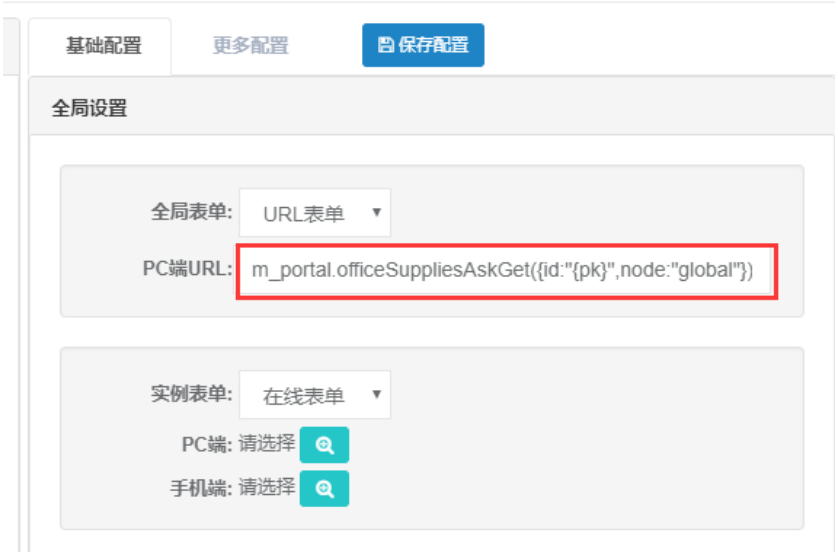
- **EIP 系统页面**

即当前 EIP 系统的页面，使用 EIP 的路由机制注册的页面，比如这里我们配置为办公用品领用的编辑或查看页面（注意这里是把办公用品领用这个功能注册在后台菜单中了，如果要把这个功能开放给终端用户使用，需要将这个页面注册到前台菜单中）。



在 URL 表单中我们将地址配置为相应的别名即可，如下图所示，在别名后面

通过括号可以带上参数，参数的格式为 JSON 格式：{id:”{pk}”,instanceId:”{instId}”}



这里的关键参数有两个：{pk}和{instId}，流程启动后，这两个变量分别会被替换成业务数据主键和流程实例 ID，这个替换的动作是在后台 java 代码中完成的，替换的代码如下所示：

AbstractFormService.java

```
475
476     private String getUrl(BpmProcessInstance instance, String url){
477
478         Map<String, String> map = new HashMap<String, String>();
479
480         if (ActionCmd.DATA_MODE_PK.equals(instance.getDataMode())){
481             map.put("pk", instance.getBizKey());
482             map.put("instId", instance.getId());
483         }
484         else{
485             List<BpmBusLink> list = bpmBusLinkManager.getByInstId(instance.getId());
486             for (BpmBusLink link : list){
487                 map.put(link.getFormIdentify(), link.getBusinesskeyStr());
488             }
489         }
490         url = replaceStr(url, map);
491
492         return url;
493     }
494
495     /**
496      * 替换字符串。
```

➢ **外部系统页面**

外部系统页面，则配置为完整的地址，例如 http://www.hotent.com?id={pk}，参数可以通过地址参数来传递。

4.2.3 业务中添加启动流程的功能

在办公用品领用的编辑页面添加一个按钮，如下图所示：

这个申请按钮对应的代码如下所示：

```
    </div>
    <div class="form-group">
        <label class="col-sm-2 control-label">关联流程实例ID:</label>
        <div class="col-sm-10">
            <input aria-required="true" class="form-control input-sm" type="text" ng-model="data.flowInstanceId"   ht-va
        </div>
    </div>
    <div class="form-group">
        <label class="col-sm-2 control-label">状态:</label>
        <div class="col-sm-10">
            <input aria-required="true" class="form-control input-sm" type="text" ng-model="data.status"   ht-validate="
        </div>
    </div>
    <div class="form-group" ng-if="!pageParam.node">
        <div class="col-sm-8 col-sm-offset-2" align="center">
            <button class="btn btn-primary" type="button" ng-click="startFlow()">发起申请</button>
            <button class="btn btn-info" type="submit">保存</button>
            <button class="btn btn-white" type="button" ng-click="close()">取消</button>
        </div>
    </div>
</form>
</div>
```

发送 Ajax 请求的代码基本和保存的代码是一致的，只是调用的后台的接口不同。

officeSuppliesAskEdit.html　officeSuppliesAskcontrollers.js

```
//发起流程申请
$scope.startFlow = function(){
    if($scope.askForm.$invalid){
        dialogService.error("请正确填写表单数据");
        return false;
    }
    baseService
    .post("${portal}/portal/officeSuppliesAsk/v1/startFlow", $scope.data)
    .then(function(response) {
        if(!response.state){
            dialogService.fail(response.message);
        }else{
            dialogService.success(response.message);
            $scope.close();
        }
    });
    return true;
}

//保存
$scope.save = function(){
    if($scope.askForm.$invalid){
        dialogService.error("请正确填写表单数据");
        return false;
    }
    baseService
    .post("${portal}/portal/officeSuppliesAsk/v1/save",$scope.data)
    .then(function(response) {
        if(!response.state){
            dialogService.fail(response.message);
        }else{
            dialogService.success(response.message);
            $scope.close();
        }
    });
    return true;
}
```

现在需要在后台添加一个 startFlow 的 restful 接口，后台的代码如下所示，与 save 接口相比只是多了一个注解@Workflow

officeSuppliesAskEdit.html officeSuppliesAskcontrollers.js OfficeSuppliesAskController.java

```
79      * @exception
80      */
81     @PostMapping(value="save")
82     @ApiOperation(value = "新增,更新办公用品领用记录数据", httpMethod = "POST", notes = "新增,更新办公用品领用记录数据")
83     public CommonResult<String> save(@ApiParam(name="officeSuppliesAsk",value="办公用品领用记录业务对象", required = true)@RequestBody Offic
84         String msg = "添加办公用品领用记录成功";
85         if(StringUtil.isEmpty(officeSuppliesAsk.getId())){
86             officeSuppliesAskManager.create(officeSuppliesAsk);
87         }else{
88             officeSuppliesAskManager.update(officeSuppliesAsk);
89              msg = "更新办公用品领用记录成功";
90         }
91         return new CommonResult<String>(msg);
92     }
93
94     /**
95      * 发起办公用品领用的流程申请
96      * @param officeSuppliesAsk
97      * @throws Exception
98      * @return
99      * @exception
100     */
101    @PostMapping(value="startFlow")
102    @ApiOperation(value = "发起办公用品领用的流程申请", httpMethod = "POST", notes = "发起办公用品领用的流程申请")
103    @Workflow(flowKey="bgyplylc", sysCode="oa", instanceIdField="flowInstanceId", varKeys= {"amout", "askManId"})
104    public CommonResult<String> startFlow(@ApiParam(name="officeSuppliesAsk",value="办公用品领用记录业务对象", required = true)@RequestBody
105        String msg = "申请成功";
106        officeSuppliesAskManager.create(officeSuppliesAsk);
107        return new CommonResult<String>(msg);
108    }
109
```

这个注解有四个参数，其中只有 flowKey 是必填的，由它来指定要启动的流程，其他参数的作用如下图所示。

```
19 @Target({ElementType.METHOD})
20 @Retention(RetentionPolicy.RUNTIME)
21 public @interface Workflow {
22     /**
23      * 流程定义key。
24      * @return
25      */
26     public String flowKey() default "";
27     /**
28      * 业务系统编码
29      * <pre>
30      * 当整个系统中很多模块都通过这种方式启动流程时，传递的业务主键可能会有重复的情况，添加一个业务系统编码可以避免主键重复。
31      * </pre>
32      * @return
33      */
34     public String sysCode() default "";
35     /**
36      * 实例ID关联字段
37      * <pre>
38      * 流程启动成功后会返回一个流程实例ID，该实例ID可以保存到业务数据中去，通过这个属性可以设置保存这个实例ID的字段。
39      * </pre>
40      * @return
41      */
42     public String instanceIdField() default "instanceId";
43     /**
44      * 变量keys集合
45      * <pre>
46      * 流程启动时可以传递变量集合，通过keys指定变量的值来自业务实体的哪些字段。
47      * </pre>
48      * @return
49      */
50     public String[] varKeys() default "";
51 }
```

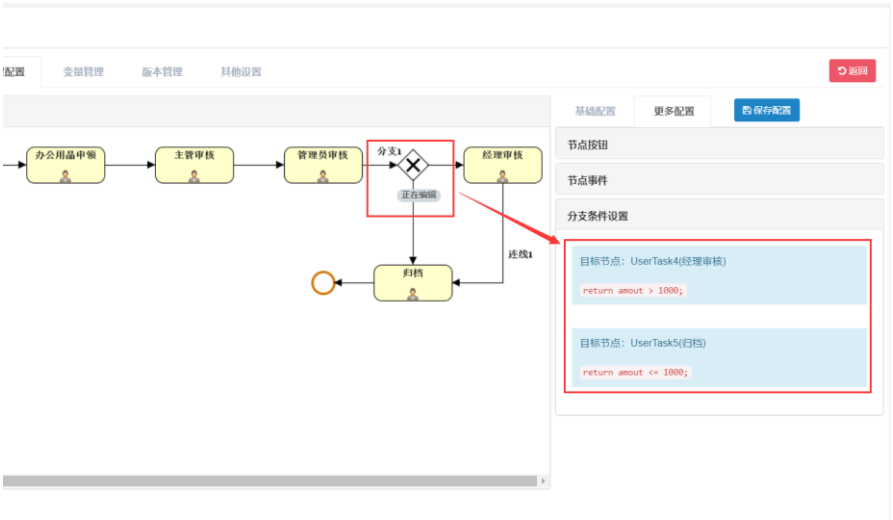
其中 varKeys 中定义的变量 key 必须同时满足以下两个条件时，才能成功的将业务变量传递给流程做为流程变量。

- 实体类中有这个属性

```java
@ApiModel(value = "OfficeSuppliesAsk",description = "办公用品领用记录")
public class OfficeSuppliesAsk extends BaseModel<String>{

    private static final long serialVersionUID = 1L;

    @ApiModelProperty(value="主键")
    protected String id;

    @ApiModelProperty(value="办公用品ID")
    protected String suppliesId;

    @ApiModelProperty(value="办公用品名称")
    protected String suppliesName;

    @ApiModelProperty(value="领用人ID")
    protected String askManId;

    @ApiModelProperty(value="领用人姓名")
    protected String askManName;

    @ApiModelProperty(value="数量")
    protected String amout;

    @ApiModelProperty(value="关联流程实例ID")
    protected String flowInstanceId;

    @ApiModelProperty(value="状态")
    protected Short status;

```

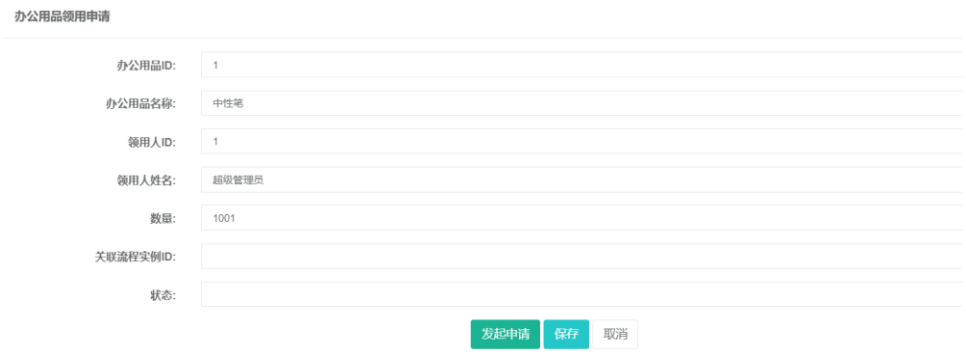
- 流程中定义了同名变量

流程变量的作用是可以用作流程分支条件的判断、跳转规则的判断等等，如下图所示：

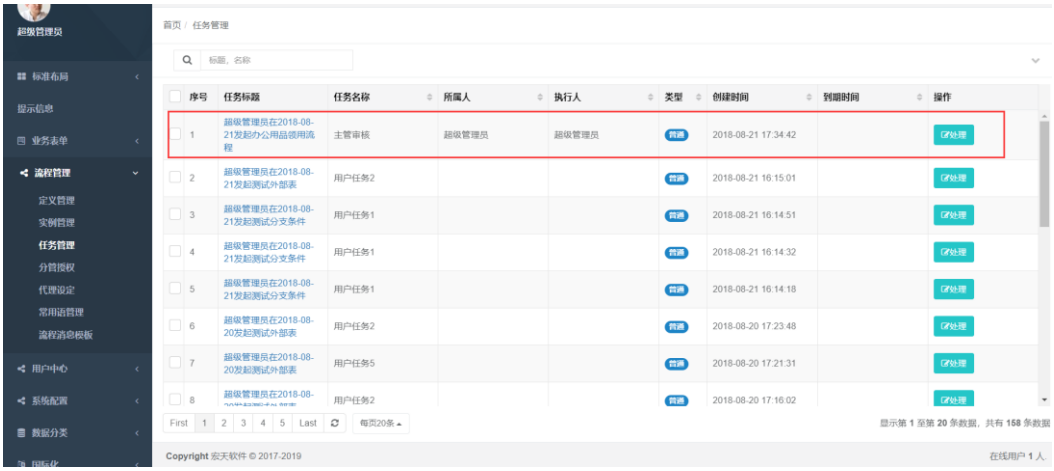


4.2.4 填写业务数据并发起流程申请

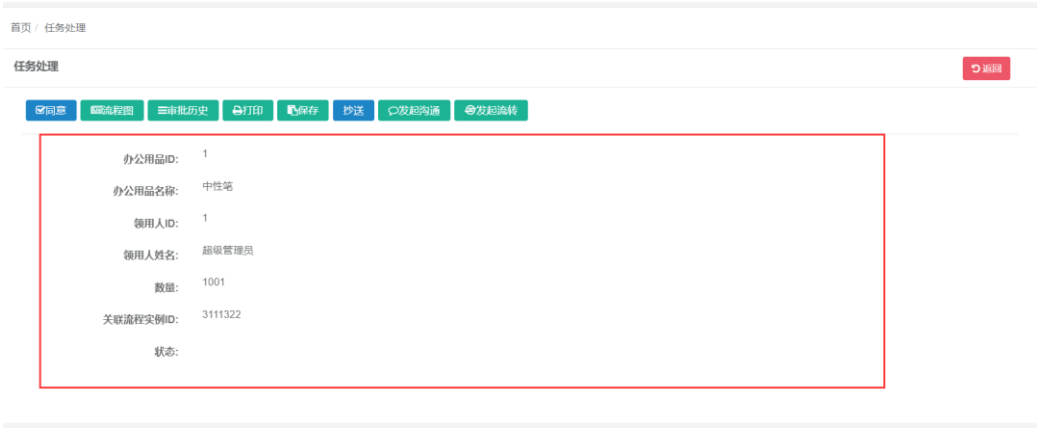
现在打开办公用品领用管理页面点击新增按钮，填写一些业务数据并点击发起申请按钮即可同时保存业务数据和启动流程了。



在流程管理中的任务管理界面即可查看刚才产生的流程任务。

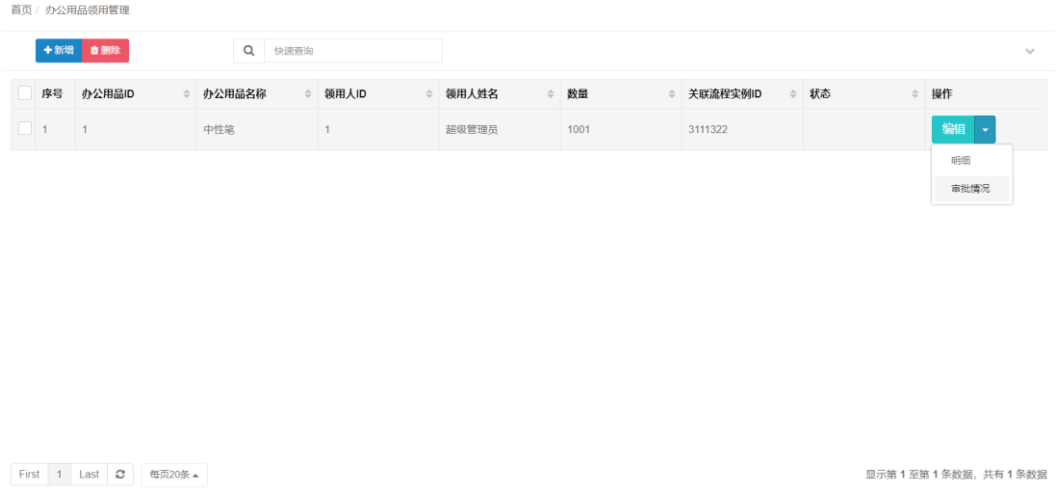


点击处理按钮即可对流程任务进行审批，任务审批界面的表单是来自业务系统的功能页面。



4.2.5 在业务管理页面查看流程审批情况

在业务管理页面上，用户需要查看这条业务数据所对应的审批情况，如下图所示：



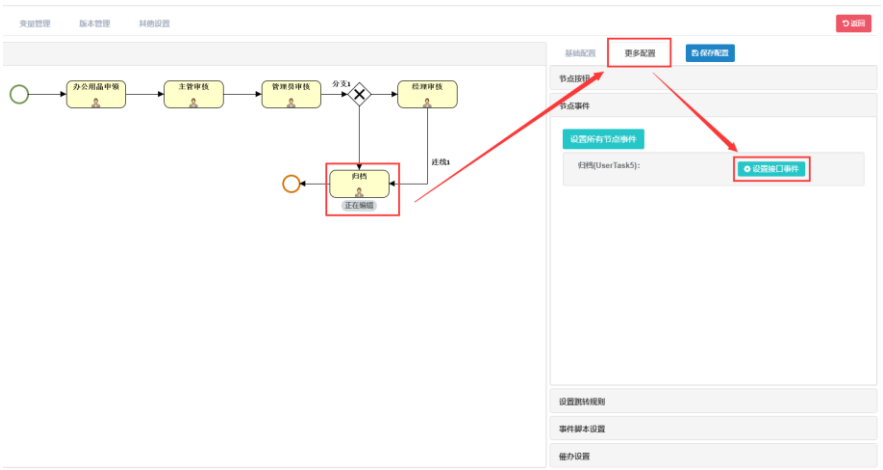
可以按照如下代码实现，添加这个按钮，而且可以通过 ng-if 来控制这个按钮是否显示出来。当这条业务数据有对应的流程实例 ID(这里是 flowInstanceId)时才显示这个按钮，而且点击这个按钮时也需要通过实例 ID 来查看实例详情。

officeSuppliesAskList.html

```
78                      <div class="checkbox">
79                          <input type="checkbox" ht-select="row">
80                          <label></label>
81                      </div>
82                  </td>
83                  <td width="50" ng-bind="$index+1" title="序号"></td>
84                  <td ht-field="row.suppliesId" title="办公用品ID" sortable="true"></td>
85                  <td ht-field="row.suppliesName" title="办公用品名称" sortable="true"></td>
86                  <td ht-field="row.askManId" title="领用人ID" sortable="true"></td>
87                  <td ht-field="row.askManName" title="领用人姓名" sortable="true"></td>
88                  <td ht-field="row.amout" title="数量" sortable="true"></td>
89                  <td ht-field="row.flowInstanceId" title="关联流程实例ID" sortable="true"></td>
90                  <td ht-field="row.status" title="状态" sortable="true"></td>
91                  <td title="操作">
92                      <div class="btn-group" uib-dropdown is-open="status.isopen">
93                          <button id="split-button" type="button" class="btn btn-info" ng-click="operating(row.id,'edit')">编辑<
94                          <button type="button" class="btn btn-info" uib-dropdown-toggle>
95                              <span class="caret"></span> <span class="sr-only">更多功能</span>
96                          </button>
97                          <ul class="dropdown-menu" uib-dropdown-menu role="menu" aria-labelledby="split-button">
98                              <li role="menuitem"><a href="javaScript:void(0)" ng-click="operating(row.id,'get')">明细</a></li>
99                              <li role="menuitem" ng-if="row.flowInstanceId">
100                                 <a ui-sref="flow.instanceDetail({id:row.flowInstanceId})">审批情况</a>
101                             </li>
102                         </ul>
103                     </div>
104                 </td>
105             </tr>
106             <tr class="no-row">
107                 <td colspan="10"><i class="fa fa-info-circle"></i> 没有查询到符合条件的记录</td>
108             </tr>
109             </tbody>
110         </table>
111         <div ht-pagination></div>
```

4.2.6 业务数据状态更新

如上图所示，当我们需要在业务数据中记录流程审批状态时，可以通过流程中的接口事件来实现。流程中接口的配置如下所示：



如上图所示，配置接口触发事件来调用业务模块中更新状态的 restful 接口。业务模块中的接口定义如下：

officeSuppliesAskList.html | OfficeSuppliesAskController.java | OfficeSuppliesAskMapper.xml | app.js | flowNodeEventSetting.html | DefaultRestfulService.java

```
113     /**
114      * 删除办公用品领用记录记录
115      * @param id
116      * @throws Exception
117      * @return
118      * @exception
119      */
120     @DeleteMapping(value="remove/{id}")
121     @ApiOperation(value = "删除办公用品领用记录记录", httpMethod = "DELETE", notes = "删除办公用品领用记录记录")
122     public CommonResult<String> remove(@ApiParam(name="id",value="业务主键", required = true)@PathVariable String id) throws Exception{
123         officeSuppliesAskManager.remove(id);
124         return new CommonResult<String>(true, "删除成功");
125     }
126
127     @DeleteMapping(value="updateStatusByInstanceId")
128     @ApiOperation(value = "更新办公用品领用记录的状态", httpMethod = "POST", notes = "更新办公用品领用记录的状态")
129     public CommonResult<String> updateStatusByInstanceId(@ApiParam(name="json",value="流程事件参数", required = true)@RequestBody String json) throws
130         JsonNode jsonNode = JsonUtil.toJsonNode(json);
131         officeSuppliesAskManager.updateStatusByInstanceId(jsonNode.get("instId").asText(), new Short("3"));
132         return new CommonResult<String>(true, "删除成功");
133     }
134
135     /**
136      * 批量删除办公用品领用记录记录
137      * @param ids
138      * @throws Exception
139      * @return
140      * @exception
141      */
142     @DeleteMapping(value="/removes")
143     @ApiOperation(value = "批量删除办公用品领用记录记录", httpMethod = "DELETE", notes = "批量删除办公用品领用记录记录")
```

在流程调用这个接口时，入参是 json 格式的字符串，包含的属性如下所示：

DefaultRestfulService.java

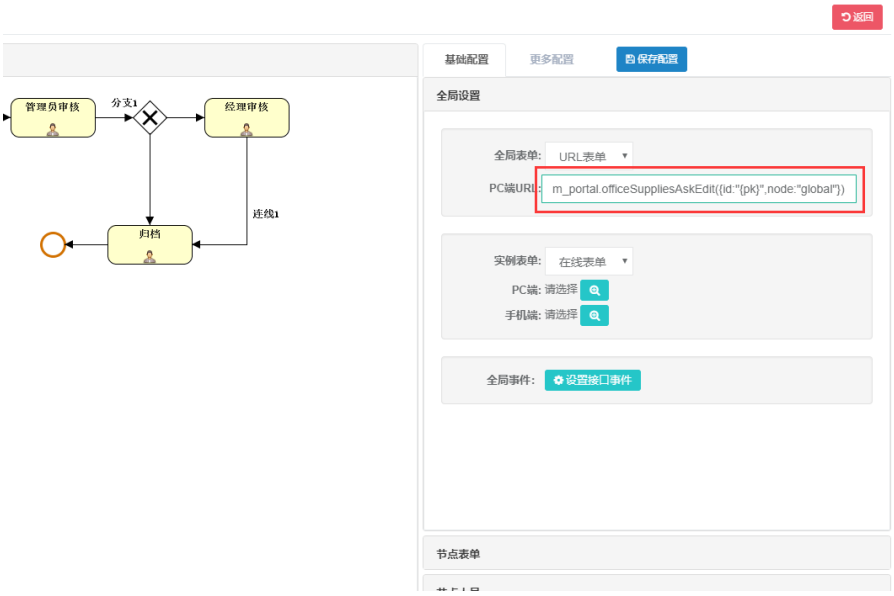
```
483         } catch (Exception e) {}
484         return list;
485     }
486
487     class RestfulParam{
488         private long timestamp;                          /*时间戳*/
489         private String procDefId;                        /*流程定义ID*/
490         private String flowKey;                          /*流程定义KEY*/
491         private String instId;                           /*流程实例ID*/
492         private String taskId;                           /*任务ID*/
493         private String nodeId;                           /*节点ID*/
494         private String nodeName;                         /*节点名称*/
495         private String eventType;                        /*事件类型*/
496         private String businesskey;                      /*业务主键*/
497         private String sysCode;                          /*业务系统编码*/
498         private String procDefName;                      /*流程名称*/
499         private BpmIdentityResult creator;               /*实例发起人*/
500         private BpmIdentityResult assignee;              /*任务执行人*/
501         private List<BpmIdentityResult> candidate;       /*任务候选人*/
502         private String actionName;                       /*节点处理类型*/
503         private String nodeType;                         /*节点类型*/
504         private LocalDateTime createTime;
505         private LocalDateTime completeTime;
506
507         public String getFlowKey() {
508             return flowKey;
509         }
```

4.2.7 URL 表单的数据处理

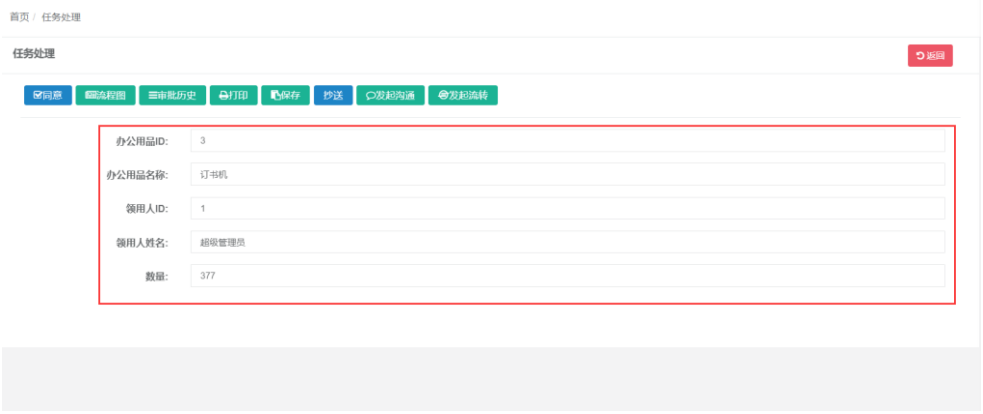
在配置流程时，我们配置的 URL 表单是一个查看详情页面，在整个流程审批过程中，用户只能查看表单数据，但是实际的需求中，某些流程节点上是需要对表单进行编辑的，那么如何实现对 URL 表单的编辑，下面依然分为两种情况来讨论：

- **EIP 系统页面**

首先修改流程中绑定的 URL 表单，从之前的绑定的详情页改为编辑页，如下图所示，这里设置的是全局表单，如果想要实现在某几个指定节点让表单可以编辑，可以通过设置全局表单为详情页面，而且具体的节点表单设置为编辑页面的方式实现。



现在我们打开任务处理界面就会看到表单处于可编辑的状态。



当我们修改了表单数据并点击同意或者保存按钮时，需要更新表单数据，这里会采用两层策略来实现对表单数据的保存。

- 通过 URL 表单中 Form 元素的 submit 方法

如下图所示，当 URL 表单中有 Form 元素，而且 Form 元素中配置了 ng-submit 事件时，表单数据的更新会以 submit 的方式提交。这层策略的优先级最高，会优先采用这种方式提交。

- 调用 URL 表单中指定的 js 方法

当表单没有 Form 元素，或者 Form 元素上没有配置 ng-submit 事件时，会获取表单所对应的$scope，判断$scope 下有没有 save 方法，如果有则调用该方法来更新数据。

特别注意，如果需要在保存 URL 表单数据的同时将业务数据更新给相应的流程变量，需要保证当前$scope 下的数据是以 data 变量来存放的，例如上述流程定义了两个流程变量：amout 和 askManId，在审批时如果修改了这两个变量的值，则当保存 URL 表单数据的同时，如果还需要将这两个变量的值相应的传递给流程变量，则需要确保这两个变量的值存放在$scope.data 下面，即能通过 $scope.data.amout 和$scope.data.askManId 能访问到这两个属性。

- **外部系统页面**

外部系统的表单，因为受跨域机制的影响，只能通过 iframe 中父子页面使用 message 通讯的方式来实现。任务处理界面作为父页面会发送一个消息给子页面，在子页面可以监听这个消息，当收到这个消息时自己做表单数据更新的操作。子页面监听的代码如下：

```
<script>
    var parentOrigin = 'http://127.0.0.1:8080';
    window.addEventListener("message", function(e){
        if(e.data == 'save' && e.origin.startsWith(parentOrigin)){
            //TODO 保存表单数据到后台
            //保存成功，则返回如下数据
            var result = {state: true, data: {amout: 1100}};
            //保存失败，则返回下面数据
            // var result = {state: false, message: "表单保存失败"};
            window.parent.postMessage(result, parentOrigin);
        }
    }, false );
</script>
</body>
```

同样的，如果需要同步更新表单数据到流程变量中，需要将表单数据传递给父页面，传递的格式如上图所示。

在 EIP 系统中对 URL 表单的处理逻辑在以下代码中：

services.js

```
1715            /**
1716             * 保存URL表单的数据
1717             */
1718            saveUrlFormData: function(){
1719                var htFrameForm = angular.element("div[ht-frame-form]"),
1720                    children = htFrameForm.children(),
1721                    deferred = $q.defer();
1722
1723                if(children && children.length == 1){
1724                    var child = children[0];
1725
1726                    // 通过iframe加载外部表单时
1727                    if(child.tagName=='IFRAME'){
1728                        var frameObj = child.contentWindow;
1729                        var handler = function(e){
1730                            if(child.src.startsWith(e.origin)){
1731                                var result = e.data;
1732                                if(result){
1733                                    if(result.state){
1734                                        deferred.resolve(result.data);
1735                                    }
1736                                    else{
1737                                        deferred.reject(result.message);
1738                                    }
1739                                }
1740                                else{
1741                                    deferred.reject("URL表单未返回正确的数据");
1742                                }
1743                            }
1744                            window.removeEventListener("message", handler);
1745                        }
1746                        window.addEventListener("message", handler);
1747                        // 通过iframe之间message通讯的机制来交互
1748                        frameObj.postMessage("save", child.src);
1749                        // 最多等待1.5秒，如果iframe子页面未返回表单数据，则开始处理流程数据
1750                        $timeout(function(){
1751                            deferred.resolve();
1752                        }, 1500);
1753                    }
1754                    // 使用EIP系统内的表单时
1755                    else{
1756                        child = angular.element(child);
1757                        var form = child.find("form");
1758                        // 表单中有form元素且该元素上有ng-submit事件监听
1759                        if(form && form.length > 0 && form.attr('ng-submit')){
1760                            var formName = form.attr('name'),
1761                                formScope = form.scope();
1762                            if(formName && formScope && formScope[formName].$invalid){
1763                                deferred.reject("表单校验错误，保存失败");
```

4.2.8 URL 表单的字段权限控制

在流程审批过程中，根据流程处于不同的节点，需要对表单上的字段实现隐

藏、只读、编辑、必填这样的权限控制，可以通过 URL 地址上的 node 这个参数来实现。

在流程配置中，不同节点上配置表单时，可以设置 node 为不同的值，例如配置全局表单时：m_portal.officeSuppliesAskEdit({id:"{pk}",node:"global"})这里的 node 参数值为 global。

在使用 EIP 系统表单时，传过来的参数会出现在$scope.pageParam 属性上，例如按照上面的参数格式，我们就可以分别取到$scope.pageParam.id 和 $scope.pageParam.node，前者为业务数据主键，后者为当前流程所处的环节。

officeSuppliesAskcontrollers.js officeSuppliesAskEdit.html

```
15
16 //  办公用品领用记录 编辑明细
17 function officeSuppliesAskEditCtrl($scope, $compile, baseService, dialogService){
18     //关闭
19     $scope.close = function(){
20         dialogService.closeSidebar();
21     }
22
23     $scope.data = {
24     }
25
26     if($scope.pageParam && $scope.pageParam.id){
27         baseService
28         .get("${portal}/portal/officeSuppliesAsk/v1/get/"+$scope.pageParam.id)
29         .then(function(response) {
30             $scope.data = response;
31         });
32     }
33
34     //发起流程申请
35     $scope.startFlow = function(){
```

另外，通过$scope.pageParam.node 参数我们可以实现一些显示上的优化，例如，同样是编辑页面，当我们在业务视图中打开时，需要显示标题栏和下方的按钮组，而在任务处理界面显示时需要隐藏标题栏和按钮组，这个效果可以通过 $scope.pageParam.node 来实现。

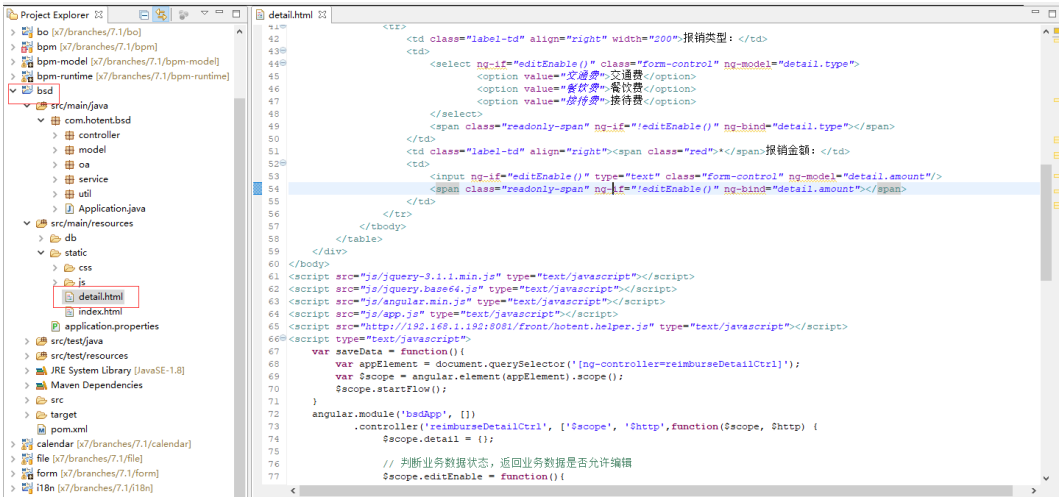
officeSuppliesAskcontrollers.js officeSuppliesAskEdit.html

```
<div class="animated fadeInRight" ng-controller="officeSuppliesAskEditCtrl">
    <div class="row wrapper">
        <div class="ibox">
            <div class="ibox-title no-borders" ng-if="!pageParam.node">
                <h5>办公用品领用申请</h5>
                <div class="ibox-tools">
                    <a ng-click="close()"><i class="fa fa-times"></i></a>
                </div>
            </div>
            <div class="ibox-content">
                <form class="form-horizontal" name="askForm" ng-submit="save()">
                    <div class="form-group">
                        <label class="col-sm-2 control-label">办公用品ID:</label>
                        <div class="col-sm-10">
                            <input aria-required="true" class="form-control input-sm" type="text" ng-model="data.supplie
                        </div>
                    </div>
                    <div class="form-group">
                        <label class="col-sm-2 control-label">办公用品名称:</label>
                        <div class="col-sm-10">
                            <input aria-required="true" class="form-control input-sm" type="text" ng-model="data.supplie
                        </div>
                    </div>
                    <div class="form-group">
                        <label class="col-sm-2 control-label">领用人ID:</label>
                        <div class="col-sm-10">
                            <input aria-required="true" class="form-control input-sm" type="text" ng-model="data.askManI
                        </div>
                    </div>
                    <div class="form-group">
                        <label class="col-sm-2 control-label">领用人姓名:</label>
                        <div class="col-sm-10">
                            <input aria-required="true" class="form-control input-sm" type="text" ng-model="data.askManN
                        </div>
                    </div>
                    <div class="form-group">
                        <label class="col-sm-2 control-label">数量:</label>
                        <div class="col-sm-10">
                            <input aria-required="true" class="form-control input-sm" type="text" ng-model="data.amout"
                        </div>
                    </div>
                    <div class="form-group" ng-if="!pageParam.node">
                        <div class="col-sm-8 col-sm-offset-2" align="center">
                            <button class="btn btn-primary" type="button" ng-click="startFlow()">发起申请</button>
                            <button class="btn btn-info" type="submit">保存</button>
                            <button class="btn btn-white" type="button" ng-click="close()">取消</button>
                        </div>
                    </div>
                </form>
```

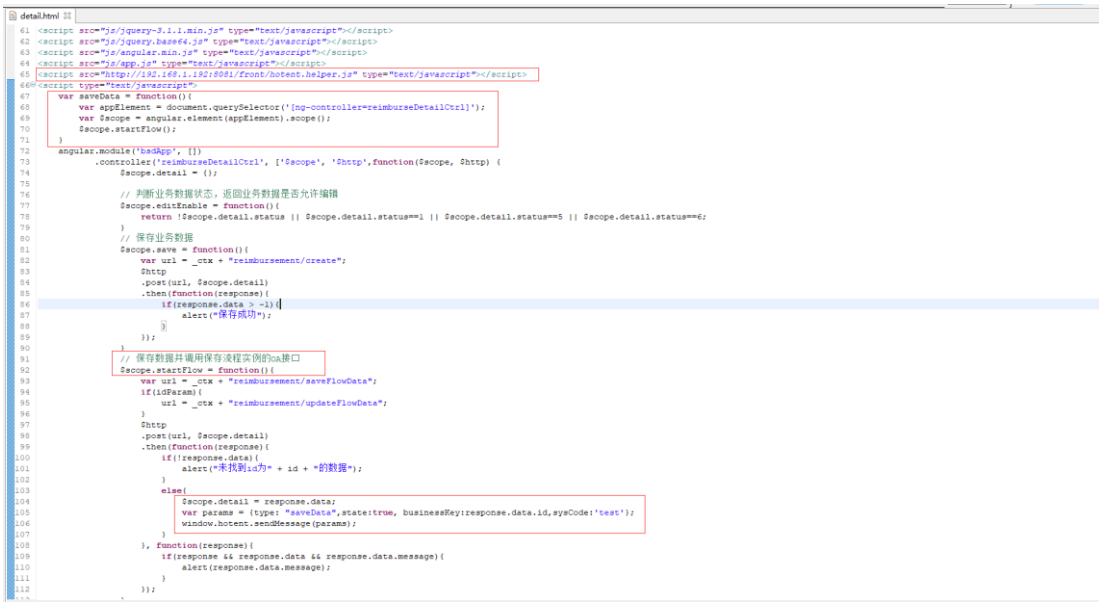
而对于外部系统 URL 表单来说，这个参数是处于 URL 参数中的，通过 js 代码可以获取到这个参数，然后自行实现相应逻辑。

4.3 流程使用外部系统 url 表单

4.3.1 开发外部系统页面



外部系统的表单，因为受跨域机制的影响，只能通过 iframe 中父子页面使用 message 通讯的方式来实现。任务处理界面作为父页面会发送一个消息给子页面，在子页面可以监听这个消息，当收到这个消息时子页面完成表单数据更新的操作。子页面监听消息需要引入流程系统一个 JS 文件（里面封装了常用交互的监听机制）且在子页面提供对应方法：

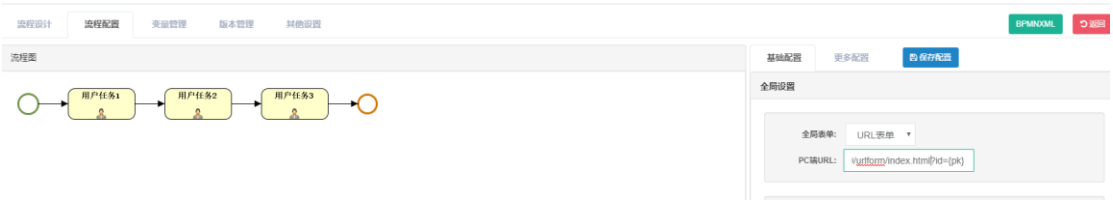


截图中的示例为发起流程或处理任务时业务系统表单与流程系统的对接，对接场景是：当在流程发起页面或任务处理页面点击提交或处理按钮时，流程系统判断当前表单是否为 URL 表单，如果是，则会发送消息给业务系统页面，事件类型为：saveData，hotent.helper.js 监听到该事件后，会调用业务系统页面的 saveData 方法，即示例中最终调用了 startFlow 方法。startFlow 方法保存当前业务数据后，将返回业务主键和业务系统编码并将信息反馈给流程系统。以下是被引入文件 hotent.helper.js 中封装的部分方法

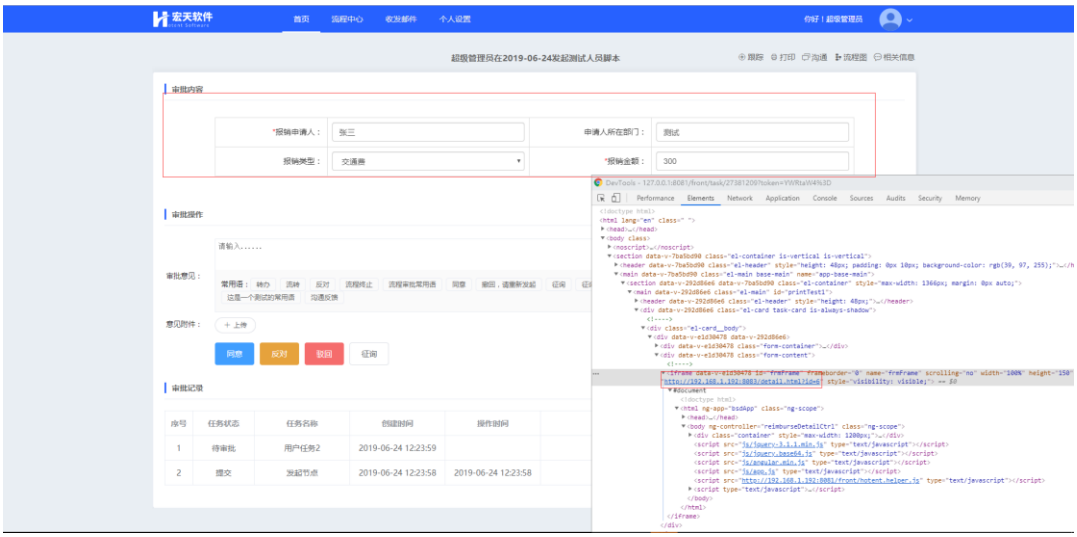
```
JS hotent.helper.js ×
public ▸ JS hotent.helper.js ▸ <function> ▸ hotent.addListener("message") callback
    }
    // 添加事件监听
    hotent.addListener = function(name, fn){
        if(!name || name.constructor!=String){
            throw "name could not be empty and must be String.";
        }
        if(!fn || fn.constructor!=Function){
            throw "fn could not be empty and must be Function.";
        }
        var eventMethod = hotent.isModern ? "addEventListener" : "attachEvent",
            eventer = window[eventMethod];

        var match = name.match(/^on(\w+)$/);
        if (match && match.length==2) {
            name = match[1];
        }
        eventer(hotent.isModern ? name : "on" + name, fn);
    }
    // 发送消息给父页面
    hotent.sendMessage = function(params){
        window.parent && window.parent.postMessage(params, "*");
    }

    // 监听父页面发送过来的message
    hotent.addListener("message", function(e){
        var type = e.data || 0;
        switch(type) {
            case "getHeight":   /*获取页面高度*/
                hotent.getHeight();
                break;
            case "saveData":    /*保存页面数据*/
            case "modifyForm":  /*打开新页面编辑数据*/
            case "printForm":   /*打印*/
            case "validForm":   /*验证数据*/
                hotent.invoke(type);
                break;
        }
    });

    // 调用页面定义的方法
    hotent.invoke = function(methodName){
        var r = window[methodName];
        if(!r || r.constructor!=Function){
            throw "页面未提供方法：" + methodName;
        }
        else{
            r.apply();
        }
    }
```

4.3.2 流程配置

http://www.hotent.org:8880/urlform/index.html?id={pk}



{pk} 占位符是存放 url 表单数据的主键值，在打开审批页面时，{pk}会被替换成相应的主键值打开详情页面，如下图所示。


宏天软件
首页
流程中心
收发邮件
个人设置
你好！超级管理员
超级管理员在2019-06-24发起测试人员脚本
跟踪
打印
沟通
流程图
相关信息
审批内容
*报销申请人：
张三
申请人所在部门：
测试
报销类型：
交通费
*报销金额：
300
审批操作
请输入.....
审批意见：
常用语：
同意
反对
驳回
征询
意见附件：
上传
审批记录
序号
任务状态
任务名称
创建时间
操作时间
1
待审批
用户任务2
2019-06-24 12:23:59
2
提交
发起节点
2019-06-24 12:23:58
2019-06-24 12:23:58
Performance
Elements
Network
Application
Console
Sources
Audits
Security
Memory

5 组件开发

EIP 系统提供了一些组件，用以在开发过程中简洁、快速的实现业务功能，而且使用统一的组件可以规范我们各个功能的实现，方便后期的维护和调整。

5.1 数据列表

5.1.1 依赖的文件

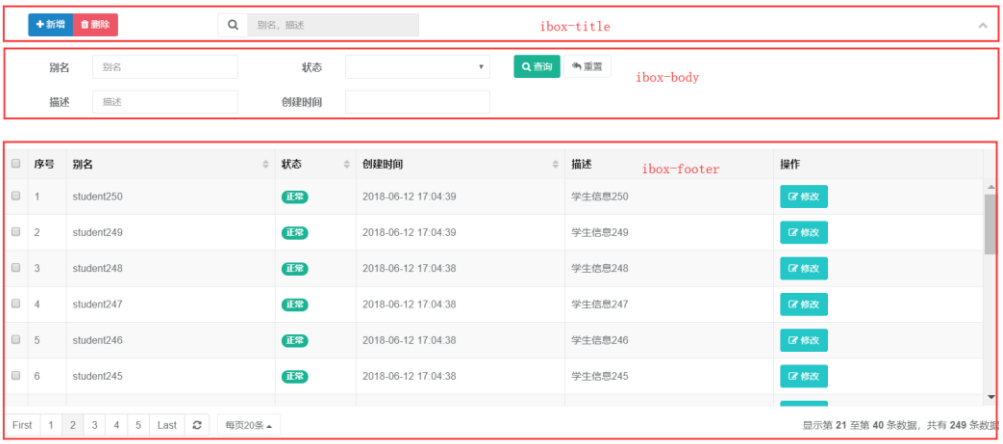
如下图，数据列表依赖的资源文件，前者控制数据列表的样式，后者实现数据列表固定高度的 js 运算。

```
[
    {
        serie: true,
        files: [
          'css/common.table.css',
          'js/base/tableFixedHeight.js'
        ]
    }
]
```

5.1.2 标准数据列表的 html 内容

```
<div class="white-bg border-left animated fadeInRight" ng-controller="bootstrapTableCtrl">
    <div class="ibox" ht-data-table="dataTable" url="http://127.0.0.1:8080/bo/def/v1/list">
        <div class="ibox-title">
            <div class="col-md-10 ">
                <div class="col-md-3 btn-group tools-panel">
                    <button type="button" class="btn btn-success btn-sm"
                        ng-click="test()">
                        <i class="fa fa-plus"></i> 新增
                    </button>
                    <button class="btn btn-danger btn-sm remove"
                            ht-remove-array url="http://127.0.0.1:8080/bo/def/v1/removes">
                        <i class="fa fa-trash"></i> 删除
                    </button>
                </div>
                <div class="input-group col-md-3 tools-panel">
                    <span class="input-group-addon">
                        <i class="fa fa-search"></i>
                    </span>
                    <input type="text" name="search" class="form-control input-sm" placeholder="别名，描述"
                           title="通过别名，描述查询" ht-quick-search="alias,description">
                </div>
            </div>
            <div class="col-md-2" table-tools></div>
        </div>
        <div class="ibox-content" style="display:none;">
            <form class="form-horizontal">
                <div class="form-group">
                    <label class="col-md-1 control-label">别名</label>
                    <div class="col-md-2">
                        <input type="text" placeholder="别名" ht-query="alias"
                               class="form-control input-sm">
                    </div>
                    <label class="col-md-1 control-label">状态</label>
                    <div class="col-md-2">
                        <select class="form-control input-sm" ht-query="status" operation="equal">
                            <option value=""></option>
                            <option value="normal">正常</option>
                            <option value="forbidden">禁用</option>
                        </select>
                    </div>
                    <div class="col-md-2">
                        <button class="btn btn-primary btn-sm" type="button"
                                ht-search><i class="fa fa-search"></i> 查询</button>
                        <button class="btn btn-default btn-sm" type="button"
                                ht-reset><i class="fa fa-reply-all"></i> 重置</button>
                    </div>
                </div>
                <div class="form-group">
                    <label class="col-md-1 control-label">描述</label>
                    <div class="col-md-2">
                        <input type="text" placeholder="描述" ht-query="description" class="form-control input-sm">
                    </div>
                    <label class="col-md-1 control-label">创建时间</label>
                    <div class="col-md-2">
                        <input date-range-picker clearable="true" class="form-control input-sm date-picker" type="text"
                               ht-query="createTime" ng-model="createTime" operation="between"/>
                    </div>
                </div>
            </form>
        </div>
        <div class="ibox-footer">
            <table class="table table-bordered table-striped table-hover" ht-table>
                <tbody>
                    <tr ng-repeat="row in dataTable.rows track by $index">
                        <td width="30" class="center" selectable="true">
                            <input class="select-checkbox" type="checkbox" ht-select="row">
                        </td>
                        <td width="50" ng-bind="$index+1" title="序号"></td>
                        <td ht-field="row.alias" title="别名" sortable="true"></td>
                        <td class="col-md-1" ng-switch="row.status" sortable="true" title="状态">
                            <span class="badge badge-primary" ng-switch-when="normal">正常</span>
                        </td>
                        <td ht-field="row.createTime | date:'yyyy-MM-dd hh:mm:ss'" title="创建时间" sortable="true"></td>
                        <td ht-field="row.description" title="描述"></td>
                        <td title="操作">
                            <button class="btn btn-info btn-sm"><i class="fa fa-edit"></i> 修改</button>
                        </td>
                    </tr>
                    <tr class="no-row">
                        <td colspan="7"><i class="fa fa-info-circle"></i> 没有查询到符合条件的记录</td>
                    </tr>
                </tbody>
            </table>
            <div ht-pagination></div>
        </div>
    </div>
</div>
```

如上图所示，是一份标准的数据列表对应的 html 代码，其页面效果如下图所示：



整个列表页面分为三个区域

- ibox-title 为列表头部，放置了功能按钮，快速搜索框，展开折起按钮
- ibox-body 为列表搜索区域，提供列表的精确搜索面板
- ibox-footer 为列表内容区域，提供列表数据、分页按钮、数据条数统计

5.1.3 各指令详细介绍

➢ 列表源指令

```
<div class="ibox" ht-data-table="dataTable" url="http://127.0.0.1:8080/bo/def/v1/list"></div>
```

列表源指令为必选指令，在该 div 元素内的所有其他指令均依赖它，其相当于 angularJs 的 ng-controller，指令属性如下：

- ht-data-table 必填，组件指令，值为实例化名称，在当前页面的 controller 中可以通过$scope.dataTable 访问该实例。注意当一个页面中有多个 ht-data-table 时，实例名称不能相同
- url 必填，列表页面所对应的数据源
- target 选填，列表组件为固定高度、固定列头，行数据部分滚动的效果，当列表相对于哪一个父组件固定高度时可以通过 target 属性来指定，该属性的值为指定元素的 class，比如：target="ibox-body"
- adjust 选填，列表固定高度时的修正值，例如:adjust="-80" adjust="77"

- **批量删除指令**

```
<button class="btn btn-danger btn-sm remove"
        ht-remove-array url="http://127.0.0.1:8080/bo/def/v1/removes">
    <i class="fa fa-trash"></i> 删除
</button>
```

勾选任意条数的数据后，点击该按钮可以执行删除操作，组件属性如下：

- ht-remove-array 必填，组件指令，无需带值
- url 必填，删除操作对应的后台地址
- remove-key 选填，默认为 id，通过列表数据中的该 key 执行删除操作
- paramKey 选填，默认为 ids，请求发送到后台时声明的参数名

- **快速搜索指令**

```
<input type="text" name="search" class="form-control input-sm" placeholder="别名，描述"
       title="通过别名，描述查询" ht-quick-search="alias,description">
```

通过该组件可以快速查询列表数据，支持同时对多个列进行查询，快速搜索会自动执行查询不需要点击查询按钮或敲击 Enter 键，组件属性如下：

- `ht-quick-search` 必填，组件指令，其值为对哪些列进行查询，比如这里是对 `alias` 和 `description`，多个列名之间逗号分隔

- **展开折起指令**

```
<div class="col-md-2" table-tools></div>
```

显示为一个三角形箭头，点击可以显示/隐藏精确搜索面板，组件属性：

- table-tools 必填，组件指令

- **精确搜索指令**

```
<input type="text" placeholder="别名" ht-query="alias"
       class="form-control input-sm">
```

对列表进行精确搜索的搜索框，属性如下：

- ht-query 必填，组件指令，其值为要搜索的列名
- operation 选填，默认值 like，搜索条件匹配操作符，可选值有:equal, equal_ignore_case, less, great,less_equal, great_equal, not_equal, like, left_like, right_like, is_null, notnull,in, between

- relation 选填，默认值 and，搜索条件与其他条件的组合关系，可选值有:and,or,exclusion 精确搜索需要点击查询按钮，该按钮可以通过指令 ht-search 触发搜索操作，示例代码如下:

```
<button class="btn btn-primary btn-sm" type="button"  ht-search><i class="fa fa-search"></i> 查询</button>
```

- 另外提供了重置按钮，该按钮可以通过指令 ht-reset 实现重置操作,示例代码如下：

```
<button class="btn btn-default btn-sm" type="button"  ht-reset><i class="fa fa-reply-all"></i> 重置</button>
```

➢ **表格指令**

```
<table class="table table-bordered table-striped table-hover" ht-table></table>
```

表格组件，显示列表数据的表格，属性如下：

- ht-table 必填，组件指令
- initialize 选填，默认值 true，是否初始化查询，当设置该值为 false 时，表格不会默认加载数据，可以在 controller 中添加如下代码手动加载数据

```
$scope.$on("table:ready", function(t, m){
  m.query();
});
```

- 或者用$scope.dataTable.query()来加载数据，但是注意，因为 ht-table 的加载需要一点时间，所以直接在 controller 中执行$scope.dataTable.query()时可能因为$scope.dataTable 还未实例化导致 query 失败，所以要么在$scope.$on("table:ready")中查询，要么在 controller 中定义的方法里面通过$scope.dataTable.query()来查询。

➢ **表格行指令**

```
<tr ng-repeat="row in dataTable.rows track by $index"></tr>
```

表格列组件，通过组件设置列头及列数据，属性如下：

- ng-repeat 必填，组件指令，该指令为 angularJs 自带指令，遍历列表数据，dataTable.rows 为列表数据，dataTable 为列表组件 ht-data-table 的实例名，rows 为固定写法，返回的列表数据会绑定到 rows 属性上。row 为每一行数据，在下面的列组件中会用到，$index 为索引，表示当前是第几行数据，注意索引从 0 开始。

- **表格列指令**

```
<td ht-field="row.alias" title="别名" sortable="true"></td>
```

表格列组件，通过组件设置列头及列数据，属性如下：

- ht-field 必填，组件指令，该列绑定的字段，值格式为 row.列名，row 与行组件中的 row 呼应
- title 必填，列头名称
- sortable 选填，是否支持排序
- width 选填，列宽度，绝对宽度，值为纯数字，单位 px
- class 选填，列样式，可以通过该属性设置相对宽度，如 class="col-md-2"，即为按照 bootstrap 布局规则占据总宽度的 2/12

注意，列组件中有两个特殊列：

- 行复选框 该列可以不设置 ht-field 及 title 属性，行复选框的标准写法如下，其中属性 selectable 属性为是否显示全选的复选框，ht-select 指令为勾选单行数据的指令

```
<td width="30" class="center" selectable="true">
  <div class="checkbox">
      <input type="checkbox" ht-select="row"><label></label>
  </div>
</td>
```

- 行数 该列可以不设置 ht-field 属性，通过该列在当前分页显示行数据的序号，标准写法如下：

```
<td width="50" ng-bind="$index+1" title="序号"></td>
```

- **分页指令**

```
<div ht-pagination></div>
```

通过该组件可以实现对列表数据进行分页查询，不配置该组件时为全列表数据查询，组件属性如下：

- ht-pagination 必填，组件指令
- ht-page-size 选填，默认值 20，每页条数
- ht-displayed-pages 选填，默认值 5，最多显示的分页页签数量

- ht-show-total 选填，默认值 true，是否统计总数，当设置为 false，不会对列表数据统计总条数，当数据量较大时可以提升性能
- page-changed 选填，分页事件，翻页时触发该事件，通过如下代码可以添加对该事件的监听：

```
<div ht-pagination page-changed="pageChange($pagination)"></div>
```

- 注意参数$pagination 为固定写法，该事件的监听函数可以声明在 controller 中，如下所示:

```
$scope.pageChange = function(p){
  console.info(p);
}
```

- $pagination 会映射到入参 p 上，该参数的值为如下：

```
{
  currentPage: 1,        /*当前页*/
  displayedPages: 5,    /*显示的最大分页页签*/
  pageSize: 20,        /*每一页的数量*/
  show: true,           /*是否分页*/
  showTotal: true,    /*是否显示总条数*/
  pageResult: {
      startRow: 1,    /*总记录中当前开始行*/
      endRow: 20,       /*总记录中当前结束行*/
      page: 1,        /*当前页*/
      pageSize: 20,    /*每一页的数量*/
      show: true,       /*是否分页*/
      totalCount: 249,/*总记录数*/
      totalPages: 13    /*总页数*/
  }
}
```

- page-size-changed 选填，每页条数调整事件，通过如下代码可以监听该事件，该事件的监听函数可以申明在 controller 中，参数为$pagination，为固定写法，参数内容与 pageChanged 事件的参数一致。

```
<div ht-pagination page-size-changed="pageSizeChange($pagination)"></div>
```

5.1.4 数据列表的方法

- query() 列表查询的方法
- addQuery(param) 添加查询参数，param 格式如下

```
{
  "property": "property",
  "value": "value",
  "operation": "operation",
  "relation": "relation"
}
```

参数 param 各个属性的说明如下：

- property 字段名，具有唯一性，后添加的查询参数会覆盖前面添加的 property 相同的查询参数
- value 查询值，可以是字符串、对象、数组
- operation 操作符号，默认为 like，可选值：["equal", "equal_ignore_case", "less", "great","less_equal", "great_equal", "not_equal", "like", "left_like", "right_like", "is_null", "notnull","in", "between"]
- relation 查询参数间的组合关系，默认为 and，可选值：["and", "or", "exclusion"]

- clearQuery(property) 清除查询参数，传入 property 时则清除该查询参数，不传时清空所有查询参数
- getQuerySize() 获取当前的查询参数个数
- hasSelectedRow() 查询当前列表是否有已选中数据
- reset() 重置查询条件并执行一次查询
- selectRow() 获取所有已选中数据，以列表返回
- selectRowIds(options) 获取所有已选中数据，返回其主键数组，主键默认为 id，可以通过传入 options 来指定主键，例如：

```
{"selectKey":"pk"}
```

- removeArray(options) 删除选中数据，options 为删除参数

```
{
  "url": "url",
  "removeKey": "removeKey",
  "paramKey": "paramKey"
}
```

各个属性的说明如下：

- url 必填，删除操作对应的后台地址
- removeKey 列表数据中通过哪个字段来执行删除操作，默认为 id
- paramKey 请求后台时，通过什么参数名来传递值，默认为 ids，例如:/data/remove?ids=1,2,3

5.1.5 数据列表的事件

列表页有六个事件：

- table:ready 表格初始化完成事件
- dataTable:ready 列表组件初始化完成事件
- dataTable:query:begin 列表开始查询事件
- dataTable:query:complete 列表完成查询事件
- dataTable:query:error 列表出错事件
- dataTable:query:reset 列表搜索条件重置事件

对事件的监听可以通过在 controller 中添加监听函数实现，例如：

```
$scope.$on("dataTable:query:begin", function(t, d){
    if(d.name!==$scope.dataTable.name){
        return;
    }
});
```

如上面的代码所示，监听了列表开始查询的事件，事件有两个参数

- t 为 angularJs 广播事件的固定参数
- d 为 dataTable 实例

注意：因为有可能在一个 controller 会存在一个以上的 dataTable,而广播事件是通过$rootScope 发布的，所以可能会监听到其他 dataTable 的事件，通过判断 dataTable 的 name 可以区分是否为所需监听的事件。

为了避免频繁的请求后台数据，当一个列表正在查询数据时，即触发了 dataTable:query:begin 还未触发 dataTable:query:complete 时，列表不响应 query() 请求，这种状态下 dataTable.inquiring 属性的值为 true，通过该属性我们可以实现一些效果，比如将查询按钮，重置按钮的状态置为不可用，查询时显示正在查询...的过渡效果。

5.2 树组件

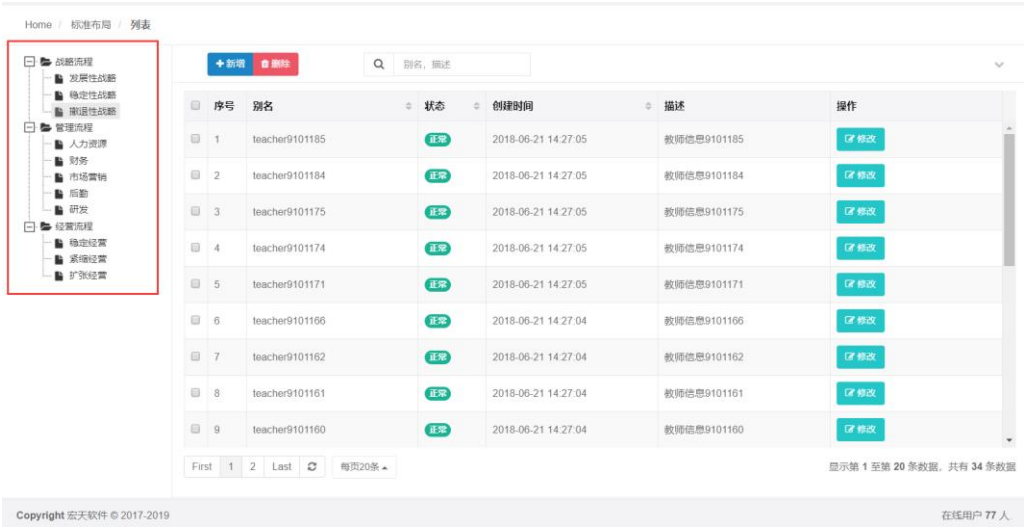
5.2.1 依赖的文件

```
{
    serie: true,
    files: [
          'css/plugins/zTree/metroStyle.css',
          'js/plugins/zTree/jquery.ztree.min.js'
    ]
},
{
    name: 'ngZtree',
    files: ['js/plugins/zTree/ng-ztree.js']
}
```

文件	说明
metroStyle.css	默认使用 metro 风格的树组件样式
jquery.ztree.min.js	zTree 库
ng-ztree.js	基于 zTree 的 AngularJs 组件库

5.2.2 树组件应用

在左树右列表的页面中，左边以树组件展示数据分类、状态的，右边展示位数据列表，当点击左边某条数据时，右边的列表过滤为对应分类或状态的数据，效果如下：



该页面对应的 html 代码:

```
<div class="white-bg border-left animated fadeInRight" ng-controller="treeTableCtrl">
    <div class="col-md-2 no-padding">
        <div class="ibox">
            <div class="ibox-content tree-content">
                <div class="full-height-scroll" full-scroll>
                    <ul ht-tree="treeConfig" ng-model="treeData" tree="treeInstance"
                        tree-events="onClick:tree_click;onCheck:tree_check"></ul>
                </div>
            </div>
        </div>
    </div>
    <div class="col-md-10 no-padding">
        <div class="ibox" ht-data-table="dataTable" url="http://127.0.0.1:8080/bo/def/v1/list">
            ...
        </div>
    </div>
</div>
```

上述 html 代码中，在 ul 元素中配置了树组件，各参数说明如下:

- ht-tree 必填，指令属性，其值为 zTree 的初始化 setting，配置项较多，详情请查看 zTree 官网的 API 文档
- ng-model 必填，树组件对应的列表数据
- tree 必填，树组件初始化后的实例对象
- tree-events 选填，事件绑定按照 name:callback 的方式申明，name 为 zTree 支持的回调事件名称，callback 为定义的回调函数，zTree 支持的回调事件很多，详情查看 zTree 官网的 API 文档

对应的 javascript 代码：

```
function treeTableCtrl($scope, baseService){
    $scope.treeConfig = {};
    $scope.treeData = [];

    baseService
    .get("http://127.0.0.1:8081/mock/tree_data.json")
    .then(function(response){
        $scope.treeData = response;
    });

    //监听列表重置事件
    $scope.$on("dataTable:query:reset", function(t, d){
        if(d.name!==$scope.dataTable.name){
            return;
        }
        //列表重置时，清空树组件中的已选项
        $scope.treeInstance.cancelSelectedNode();
    });

    $scope.tree_click = function(e, i, n){
        //将当前选中项作为查询条件
        $scope.dataTable.addQuery({property: 'categoryId', operation: 'equal', value: n.id});
        //执行列表查询
        $scope.dataTable.query();
    }

    $scope.tree_check = function(e, i, n){
        console.info(n);
    }
}
```

5.2.3 树右键菜单

可以通过以下代码为树组件添加右键菜单：

```
<ul ht-tree="treeConfig" ng-model="treeData" tree="treeInstance"
                            tree-context-menu="contextMenu" before-right-click="beforeRightClick(treeNode)"
                            menu-click="menuClick(menu, treeNode)"></ul>
```

- tree-context-menu 必填，值为右键菜单项数组，在 controller 中指定 contextMenu 的值，例如$scope.contextMenu=['添加子项','删除目录','-','授权']，多个菜单项用逗号分隔，-为分割线
- before-right-click 选填，右键点击前事件，通过该事件可以实现不同节点显示不同的右键菜单项，例如在 controller 中通过如下代码实现父节点与子节点有不同的菜单项:

```
// 右键菜单点击前事件
$scope.beforeRightClick = function(treeNode){
  if(treeNode.isParent){
      $scope.contextMenu = ['添加子项', '删除目录', '-', '授权'];
  }
  else{
      $scope.contextMenu = ['编辑', '删除'];
  }
}
```

- menu-click 选填，点击右键菜单项的事件，该事件有两个参数 menu 和 treeNode，前者为当前点击的菜单项，后者为右键选中的树节点。根据这两个参数可以执行不同菜单项的对应逻辑，如下面的代码所示，在该方法中返回 false 时，不会关闭右键菜单。

```
// 点击某个右键菜单项
$scope.menuClick = function(menu, treeNode){
  switch(menu){
      case '添加子项':
          break;
      case '删除目录':
          break;
      case '授权':
          break;
      case '编辑':
          break;
      case '删除':
          break;
  }
}
```

5.3 提示信息

提示信息使用统一的 dialogService 服务来实现，该服务作为 AngularJs 的 Service 来定义的，使用时需要先注入，按照如下格式：

controllers.js

```
68
69  function messageCtrl($scope, dialogService, baseService, toaster,$timeout){
70      //选择国家化资源按钮
71      $scope.resources = function() {
72          //跳转选择国际化资源页面
73          dialogService.page("i18n-search", {pageParam:{id:1}})
74              .then(function(result){
75                  dialogService.msg("回传的数据:" + JSON.stringify(result));
76              });
77      }
78      $scope.messageSuccess = function(){
79          dialogService.success("操作成功").then(function(){
80              dialogService.msg("这是回调");
81          });
82      }
83
84      $scope.messagePrompt = function(){
85          dialogService.msg("已完成，但有些记录没有成功删除");
86      }
87
88      $scope.messageWarn = function(){
89          dialogService.warn("该数据已被关联，无法删除");
90      }
91
92      $scope.dialogError = function(){
93          dialogService.fail("操作失败").then(function(){
94              dialogService.msg("错误消息的回调");
95          });
96      }
97
98      $scope.dialogConfirm = function(){
```

- **成功**

```
dialogService.success("操作成功").then(function(){
    alert("这是回调");
});
```

- **提示**

```
dialogService.msg("已完成，但有些记录没有成功删除").then(function(){
    alert("这是回调");
});
```

- **警告**

```
dialogService.warn("该数据已被关联，无法删除").then(function(){
    alert("这是回调");
});
```

- **错误**

```
dialogService.fail("操作失败").then(function(){
    alert("错误消息的回调");
});
```

- **确认**

```
dialogService.confirm("确认要删除吗?").then(function(){
    alert("您选择了确定");
}, function(){
    alert("您选择了取消");
});
```

- **Tooltip**

```
<label class="fa fa-question-circle-o" tooltip-placement="left"
      uib-tooltip="一个人生命中的自然与必然事件，由前世的作为所决定。含有善恶、苦乐果报之意味，亦即与因果关系相结合的一种持续作用力。">  Karma</label>
```

- **Toaster**

成功消息

```
toaster.success({ body:"Hi, welcome to Inspinia. This is example of Toastr notification box."});
```

警告消息

```
toaster.warning({ title: "Title example", body:"This is example of Toastr notification box."});
```

自定义消息

```
toaster.pop({
    type: 'error',
    title: 'Title example',
    body: 'This is example of Toastr notification box.',
    showCloseButton: true,
    timeout: 1000
});
```

5.4 对话框

需要通过弹框显示某个页面时，可以通过以下代码实现（同样需要注入 dialogService 服务）：

```
dialogService.page("bo-selector", {pageParam:{id:1}})
             .then(function(result){
                 dialogService.msg("回传的数据:" + JSON.stringify(result));
             });
```

上述代码中，值 bo-selector 为注册到 ui-router 中的状态路由，如下:

```
.state('bo-selector', {
    url: "/bo-selector",
    templateUrl: "views/selector/bo-selector.html",
    data: {pageTitle: '选择业务对象'},
    resolve: {
        loadPlugin: function ($ocLazyLoad) {
            return $ocLazyLoad.load([
                {
                    serie: true,
                    files: [
                          'js/plugins/daterangepicker/daterangepicker.js',
                          'css/plugins/daterangepicker/daterangepicker-bs3.css',
                          'css/common.table.css',
                          'js/util/util.js',
                          'js/base/tableFixedHeight.js'
                    ]
                },
                {
                    name: 'selector',
                    files: ['js/selector/selector-controllers.js']
                }
            ]);
        }
    }
})
```

通过参数{pageParam: {id: 1}}可以传递值到 bo-selector 中访问，在 bo-selector.html 页面所对应的 controller 中可以访问该 pageParam 属性，另外在子页面中定义$scope.pageSure 方法会在点击对话框的确定按钮时调用该方法，而该方法的返回值会返回到父页面的回调方法中。

```
function boSelectorCtrl($scope, dialogService){
    // 获取父页面传递过来的值
    $scope.id = $scope.pageParam.id;

    // 响应对话框的确定按钮，并返回值到父页面
    $scope.pageSure = function(){
        return $scope.dataTable.selectRow();
    }
}
```

5.5 侧边栏

通过侧边栏也可以显示另一个页面，调用的代码如下：

```
dialogService.sidebar("bo-detail", {bodyClose: false, width: '300px', pageParam: {name: '业务对象'}});
```

这个方法中，第一个参数 bo-detail 同样为在 ui-router 中注册的状态，第二个参数是配置对象，各配置属性介绍如下：

- bodyClose 选填，默认为 true，默认为点击非侧边栏区域时会关闭侧边栏，设置为 false 时不会关闭侧边栏
- width 选填，默认为 360px，侧边栏的宽度，可以设置为固定值也可以设

置为百分比，如：300px,25%,calc(100% - 250px)

- pageParam 选填，传递到子页面的参数

另外可以通过 dialogService.closeSidebar()方法来关闭侧边栏，整个平台侧边栏一次只能打开一个，所以在主页面或者子页面调用这个关闭方法都会关闭当前侧边栏。

6 工作流

6.1 流程术语

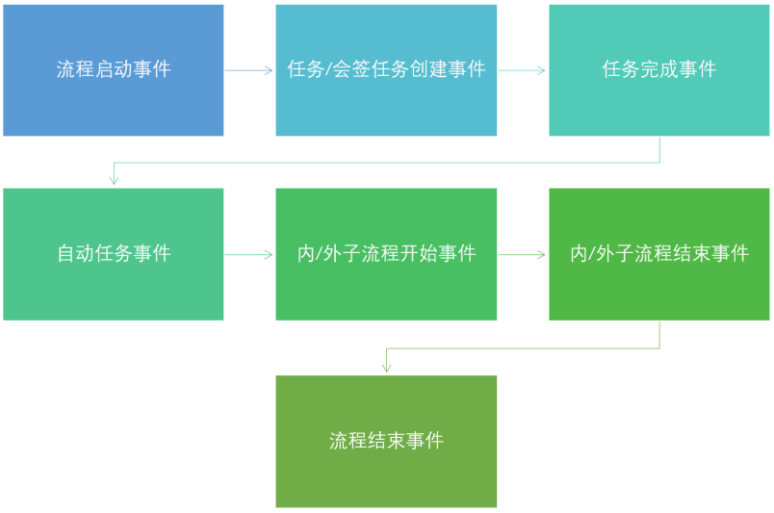
序号	术语	描述
1	流程定义	流程定义又称流程模型，是用来描述业务过程的规定性文档，一个流程主要由一系列的活动和转移组成。流程定义需要遵从特定的语法规范。ACTIVITI 支持 BPMN2.0 标准规范。流程定义有工作流引擎负责解释执行。
2	流程实例	流程实例是在流程根据流程定义产生的实例，是实例化流程定义。我们说一条流程执行完毕，意思就是流程实例生命周期结束。
3	流程任务	流程任务是指当启动流程实例以后，流程走到用户任务或会签任务时产生的需要用户审批的待办。
4	流程元素	在 BPMN 标准中，流程元素包括：事件、活动、网关、泳池。
5	流程环节	完成一个流程的设计以后，流程定义中的每一个元素，我们称为这个流程的一个环节。
6	发起人	发起流程实例的用户称为发起人，或者称为流程实例的创建人。
7	执行人	当一个流程任务分配给一个用户来处理时，这个人就是这个任务的执行人。
8	候选人	当一个流程任务分配给多个人来处理时，这些人就是这个任务的候选人。

6.2 流程扩展

在平台中我们对 Activiti 扩展，主要是扩展了流程的监听器，自动任务节点等。

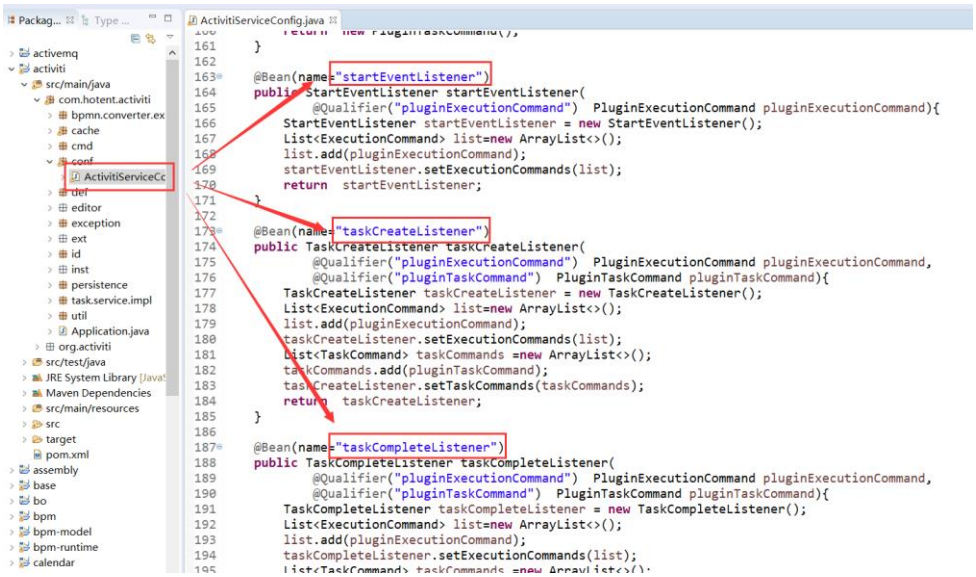
Activiti 流程定义为一个 xml 文件，我们可以在平台中选择流程定义 BPMNXML 按钮来查看这份 xml 文件。

我们对流程进行了扩展，添加了一系列通用的事件监听器，这些监听器的执行顺序如下图所示：



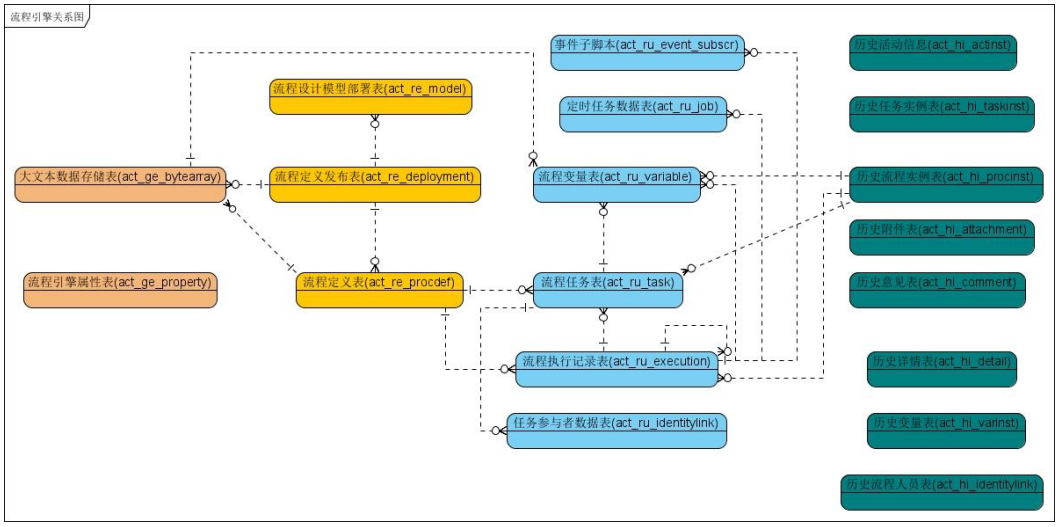
序号	事件	事件监听器
1	流程启动事件	startEventListener
2	任务创建事件	taskCreateListener
3	任务完成事件	taskCompleteListener
4	会签任务创建事件	taskSignCreateListener
5	自动任务事件	customServiceTask
6	流程结束事件	endEventListener
7	外部子流程开始事件	callSubProcessStartListener
8	外部子流程结束事件	callSubProcessEndListener
9	子流程开始事件	subProcessStartListener
10	子流程结束事件	subProcessEndListener

在代码中我们可以找到这些事件监听器的注册类：



6.3 流程定义扩展

Activiti 流程引擎提供了流程流转的基本功能，但是涉及到一些具体的操作，流程引擎没有满足相关功能。我们在引擎的基础上做了很多的扩展，来满足这些功能需求。



如上图所示，为 Activiti 流程引擎的表结构，其中最左侧部分的两张表位存储信息表、三张黄色的表为流程定义表、六张浅蓝色的表为流程运行时的表、最右侧的八张绿色的表为历史表。

为了扩展更多的流程功能，我们添加了一张表 BPM_DEFINITION 来保存流程定义的数据。

字段名	字段类型	备注

DEF_ID_	VARCHAR	流程定义 ID
NAME_	VARCHAR	流程名称
DEF_KEY_	VARCHAR	流程定义 KEY
DESC_	VARCHAR	流程描述
TYPE_ID_	VARCHAR	所属分类 ID
STATUS_	VARCHAR	状态。草稿、发布、禁用
TEST_STATUS_	VARCHAR	测试状态
BPMN_DEF_ID_	VARCHAR	BPMN - 流程定义 ID
BPMN_DEPLOY_ID_	VARCHAR	BPMN - 流程发布 ID
VERSION_	INT	版本 - 当前版本号
MAIN_DEF_ID_	VARCHAR	版本 - 主版本流程 ID
IS_MAIN_	CHAR	版本 - 是否主版本
REASON_	VARCHAR	版本 - 变更理由
CREATE_BY_	VARCHAR	创建人 ID
CREATE_TIME_	DATETIME	创建时间
CREATE_ORG_ID_	VARCHAR	创建者所属组织 ID
UPDATE_BY_	VARCHAR	更新人 ID
UPDATE_TIME_	DATETIME	更新时间
DESIGNER_	VARCHAR	设计器类型
SUPPORT_MOBILE_	INT	支持手机表单

这张表所对应的实体类是 DefaultBpmDefinition，其提供的流程定义接口定义在 BpmDefinitionManager 中。

6.4 流程设计器

EIP 提供 Html5 的流程设计器，这个设计器设计出来的流程定义会以 Json 格式提交给后台的发布或保存流程的接口，在后台以 bpm_def_data 这张表来保存流程定义的文件。

这张表有两个关键字段：bpmnXml 和 defJson，其中 defJson 存放的是 Html5 设计出来的 json 数据，而 bpmnXml 中需要存放的是 Activiti 所需的遵循 BPMN2.0 规范的 xml 格式数据，这里就需要将 json 转换为 xml 格式的，我们使用 DefTransform 接口来转换，对于 json 格式，我们提供了 WebDefTransform 这个实现类。

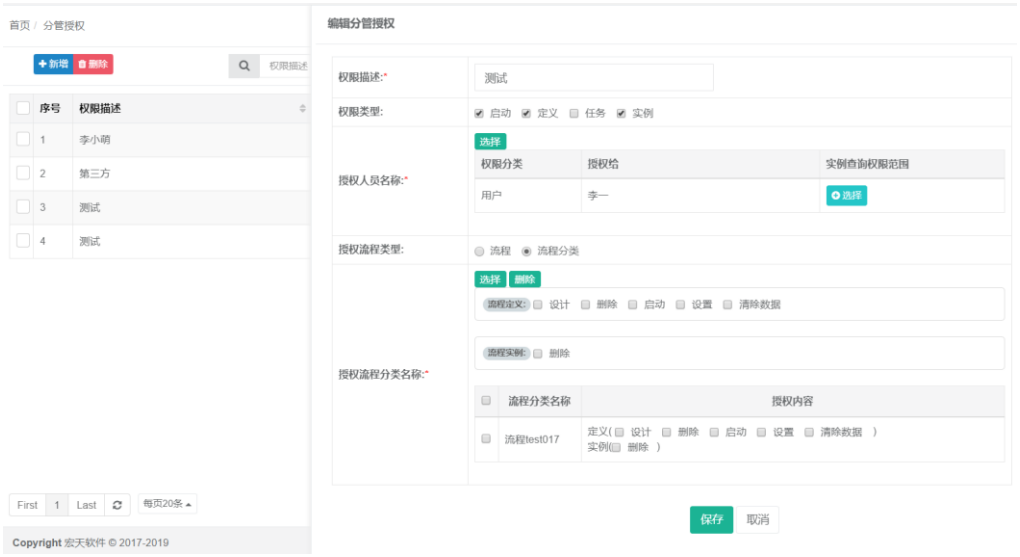
BpmDefDataMapper.xml BpmDefData.java BpmDefinitionManagerImpl.java DefTransform.java WebDefTransform.java

```java
package com.hotent.activiti.def.impl;

import java.util.Map;

public class WebDefTransform implements DefTransform {

    @Override
    public String convert(String id, String name, String designerXml) {
        String bpmnXml = "";
        try {
            JsonNode modelNode =  new ObjectMapper().readTree(designerXml);
            BpmnModel bpmnModel = new HtBpmnJsonConverter().convertToBpmnModel(modelNode);
            BpmnXMLConverter xmlConverter = new BpmnXMLConverter();
            byte[] bpmnBytes = xmlConverter.convertToXML(bpmnModel);
            bpmnXml = new String(bpmnBytes,"utf-8");
        } catch (Exception e) {
        }
        return bpmnXml;
    }

    @Override
    public String converConditionXml(String nodeId, Map<String, String> map,
            String xml) {
        return "";
    }
}
```

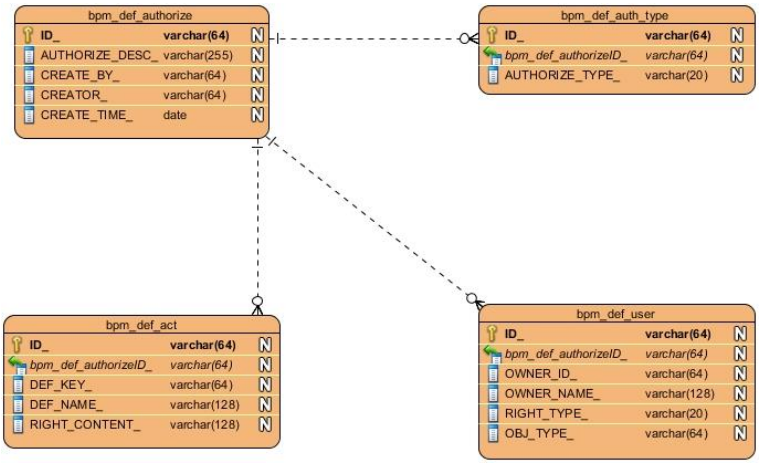
6.5 流程授权

流程设计和配置完成以后，需要发布给用户来使用，或者需要将流程的管理权限分配出去，这里就需要通过流程授权来实现。

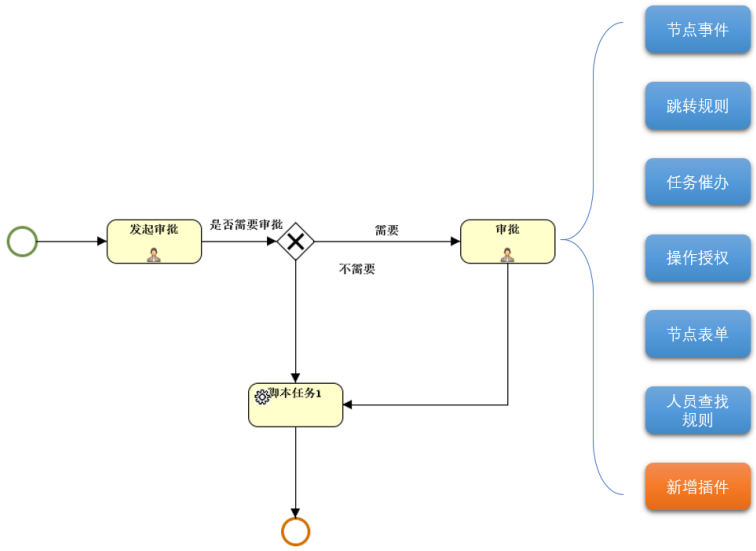
流程授权界面如下图所示：



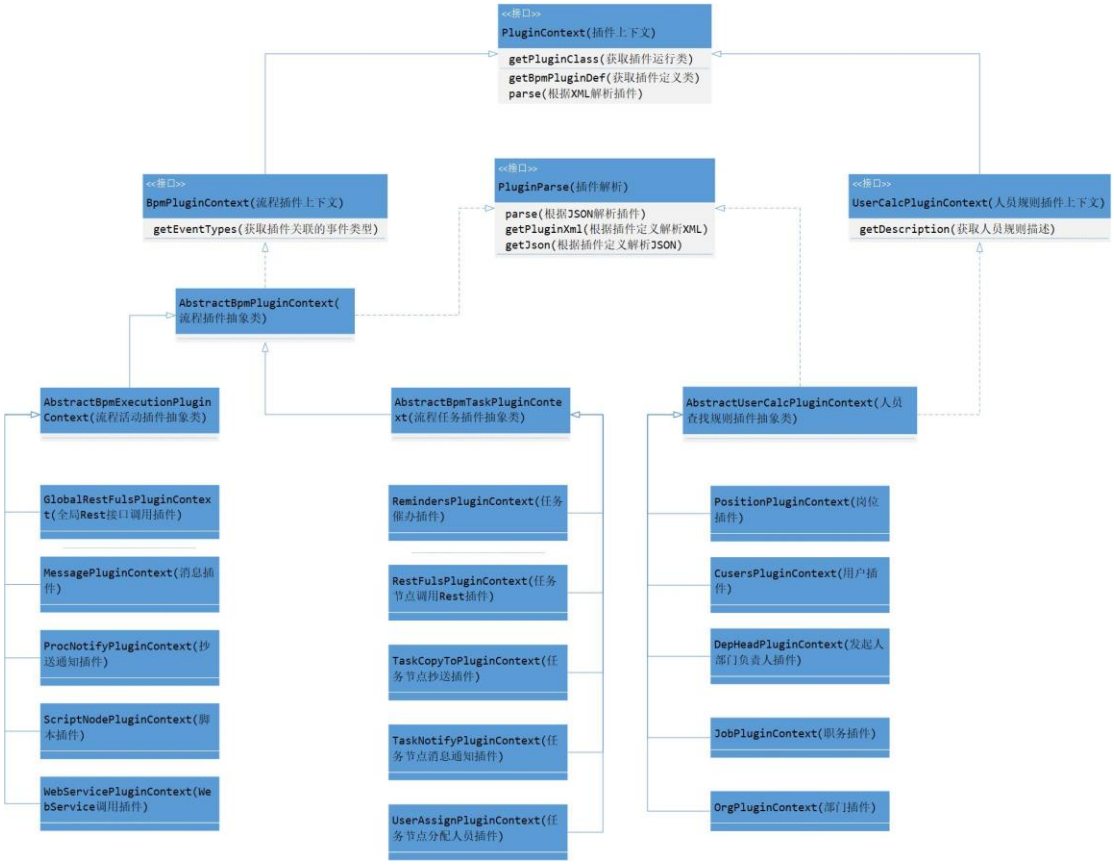
存放这些授权数据的表如下所示：



6.6 流程插件开发



如图所示，EIP 中使用插件来实现流程的扩展功能。系统中包含三种类型的插件：execution（放置所有执行类插件）、task（放置所有任务类插件）、usercalc（放置所有用户计算策略插件），其类图如下：



6.6.1 插件的配置

以 GlobalRestFulsPluginContext 插件为例，在如下图所示的配置界面完成插件的配置。

在提交保存操作时，会将这些配置内容提交到后台，后台的代码会通过如下图所示的过程完成插件的解析和配置保存。

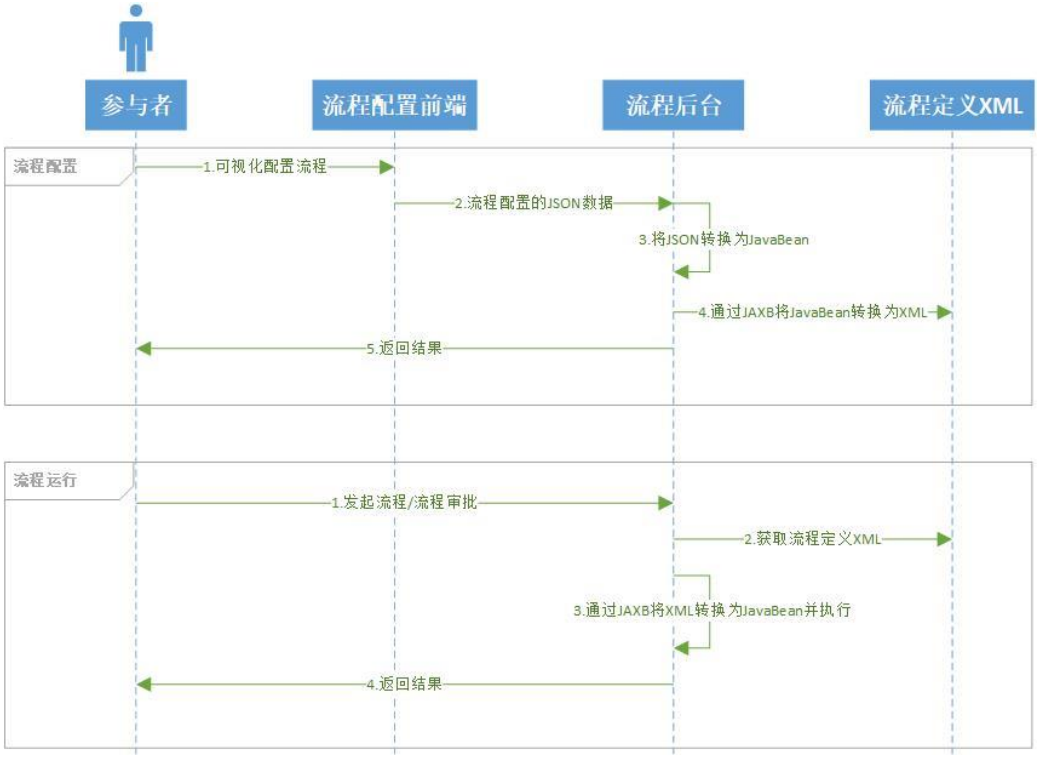
RestFulsPluginContext.java | NodeController.java | AbstractBpmPluginContext.java

```
//保存全局restful插件
try {
    ObjectNode settingJson =(ObjectNode) JsonUtil.toJsonNode(defSettingJson);
    String globalRestfulStr = settingJson.get("globalRestfuls").toString();
    if(StringUtil.isNotEmpty(globalRestfulStr)||settingJson.get("globalRestfuls").size()>0){
        GlobalRestFulsPluginContext globalContext = new GlobalRestFulsPluginContext();
        globalContext.parse(globalRestfulStr);
        BpmProcessDef<BpmProcessDefExt> bpmProcessDefExt = bpmDefinitionAccessor.getBpmProcessDef(defId);
        DefaultBpmProcessDefExt defExt = (DefaultBpmProcessDefExt) bpmProcessDefExt.getProcessDefExt();
        List<BpmPluginContext> plugins = changeOnePluginContextForSave(defExt.getPluginContextList(),globalContext);
        PluginsBpmDefXmlHandler bpmDefXmlHandler = (PluginsBpmDefXmlHandler) AppUtil.getBean(PluginsBpmDefXmlHandler.class);
        bpmDefXmlHandler.saveNodeXml(defId, null, plugins);
    }
} catch (Exception e) {
```

最终，在流程对应的 XML 文件中，会以如下图所示的片段保存这个插件配置。

```
<ext:extProperties>
    <ext:subjectRule/>
    <ext:description/>
    <ext:startNotifyType/>
    <ext:archiveNotifyType/>
    <ext:skipFirstNode/>
    <ext:firstNodeUserAssign/>
    <ext:skipSameUser/>
    <ext:allowCopyTo/>
    <ext:allowTransTo/>
    <ext:useMainForm/>
    <ext:allowReference/>
    <ext:allowRefCounts/>
</ext:extProperties><ext:globalForm><ext:form formValue="jmtest111" name="建模test111" parentFlowKey="local_" type="inner"/></ext:globalForm><ext:boList saveMode="database">
<ext:boDef isRequired="0" key="tets111" name="tets111"/>
</ext:boList><ext:extPlugins>
<globalRestFuls xmlns="http://www.jee-soft.cn/bpm/plugins/execution/globalRestFuls">
    <globalRestful params="" outPutScript="" parentDefKey="">
        <url>http://127.0.0.1:8009/demo/test</url>
        <desc>流程事件接口测试</desc>
        <header>1:1=</header>
        <invokeMode>1</invokeMode>
        <callTime>startEvent,endEvent</callTime>
        <callNodes/>
    </globalRestful>
</globalRestFuls>
</ext:extPlugins></ext:extProcess>
</ext:definitions>
```

6.6.2 插件的解析



如上图所示，在流程的配置和流程运行时，会涉及到插件的解析，通过 Jaxb 框架来实现 JavaBean 与 XML 之间的相互转换。

GlobalRestFulsPluginContext.java

```
42
43    @SuppressWarnings("rawtypes")
44    @Override
45    public Class<? extends RunTimePlugin> getPluginClass() {
46        return (Class<? extends RunTimePlugin>) GlobalRestfulInvokePlugin.class;
47    }
48
49    @Override
50    public String getPluginXml() {
51        GlobalRestfulInvokePluginDef pluginDef = (GlobalRestfulInvokePluginDef) this.getBpmPluginDef();
52        String xml = "";
53        try {
54            if(BeanUtils.isEmpty(pluginDef.getRestfulList()))return xml;
55            xml = JAXBUtil.marshall(GlobalRestfulInvokePluginDef.getRestfulExt(pluginDef), ObjectFactory.class);
56            xml = xml.replace("encoding=\"utf-8\"", "encoding=\"UTF-8\"");
57            xml = xml.replace("<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>\n", "");
58        } catch (JAXBException e) {
59            e.printStackTrace();
60        }
61        return xml;
62    }
63
```

GlobalRestFulsPluginContext.java

```
89          def.setRestfulList(restFuls);
90          return def;
91      }
92
93      @Override
94      protected BpmPluginDef parseElement(Element element) {
95          String xml = XmlUtil.getXML(element);
96          GlobalRestfulInvokePluginDef def ;
97           try {
98               GlobalRestFuls restfuls = (GlobalRestFuls) JAXBUtil.unmarshall(xml,ObjectFactory.class);
99              def =GlobalRestfulInvokePluginDef.getRestfuls(restfuls);
100             return def;
101         } catch (Exception e) {
102             e.printStackTrace();
103         }
104         return null;
105     }
106
107
108
```

6.6.3 插件的执行

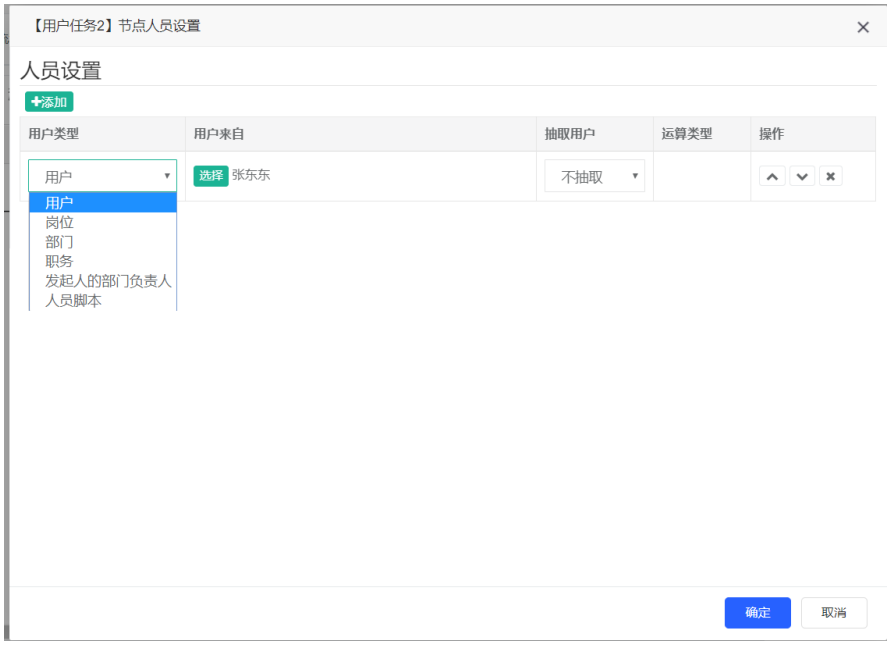
插件的执行通过流程事件的监听处理器来触发，如下图所示，流程启动事件、任务创建事件、任务完成事件、流程结束事件等事件中都会将 ExecutionCommand 设置到事件处理器中。

GlobalRestfulInvokePlugin.java | ActivitiServiceConfig.java | EndEventListener.java | AbstractExecutionListener.java

```
166         StartEventListener startEventListener = new StartEventListener();
167         List<ExecutionCommand> list=new ArrayList<>();
168         list.add(pluginExecutionCommand);
169         startEventListener.setExecutionCommands(list);
170         return  startEventListener;
171     }
172
173     @Bean(name="taskCreateListener")
174     public TaskCreateListener taskCreateListener(
175             @Qualifier("pluginExecutionCommand")  PluginExecutionCommand pluginExecutionCommand,
176             @Qualifier("pluginTaskCommand")  PluginTaskCommand pluginTaskCommand){
177         TaskCreateListener taskCreateListener = new TaskCreateListener();
178         List<ExecutionCommand> list=new ArrayList<>();
179         list.add(pluginExecutionCommand);
180         taskCreateListener.setExecutionCommands(list);
181         List<TaskCommand> taskCommands =new ArrayList<>();
182         taskCommands.add(pluginTaskCommand);
183         taskCreateListener.setTaskCommands(taskCommands);
184         return  taskCreateListener;
185     }
186
187     @Bean(name="taskCompleteListener")
188     public TaskCompleteListener taskCompleteListener(
189             @Qualifier("pluginExecutionCommand")  PluginExecutionCommand pluginExecutionCommand,
190             @Qualifier("pluginTaskCommand")  PluginTaskCommand pluginTaskCommand){
191         TaskCompleteListener taskCompleteListener = new TaskCompleteListener();
192         List<ExecutionCommand> list=new ArrayList<>();
193         list.add(pluginExecutionCommand);
194         taskCompleteListener.setExecutionCommands(list);
195         List<TaskCommand> taskCommands =new ArrayList<>();
196         taskCommands.add(pluginTaskCommand);
197         taskCompleteListener.setTaskCommands(taskCommands);
198         return  taskCompleteListener;
199     }
200
```

```java
GlobalRestfulInvokePlugin.java  ActivitiServiceConfig.java  EndEventListener.java  AbstractExecutionListener.java
     * @throws Exception
     */
    public abstract void afterPluginExecute(BpmDelegateExecution bpmDelegateExecution) throws Exception;

    @Override
    public void notify(DelegateExecution delegateExecution) throws Exception {
        //转换数据
        BpmDelegateExecution bpmDelegateExecution = BpmDelegateFactory
                .getBpmDelegateExecution(delegateExecution);
        //在插件之前执行逻辑
        beforePluginExecute(bpmDelegateExecution);
        //执行插件（在触发子类执行之前的插件，对应getBeforeTriggerEventType）
        if(executionCommands!=null && getBeforeTriggerEventType()!=null){
            for(ExecutionCommand cmd:executionCommands){
                cmd.execute(getBeforeTriggerEventType(), bpmDelegateExecution);
            }
        }
        //触发子类执行
        triggerExecute(bpmDelegateExecution);
        //执行插件（在触发子类执行之后的插件，对应getAfterTriggerEventType）
        if(executionCommands!=null && getAfterTriggerEventType()!=null){
            for(ExecutionCommand cmd:executionCommands){
                cmd.execute(getAfterTriggerEventType(), bpmDelegateExecution);
            }
        }
        //在插件全部执行完之后执行逻辑
        afterPluginExecute(bpmDelegateExecution);
        //执行脚本
        exeEventScript(bpmDelegateExecution);

    }
```

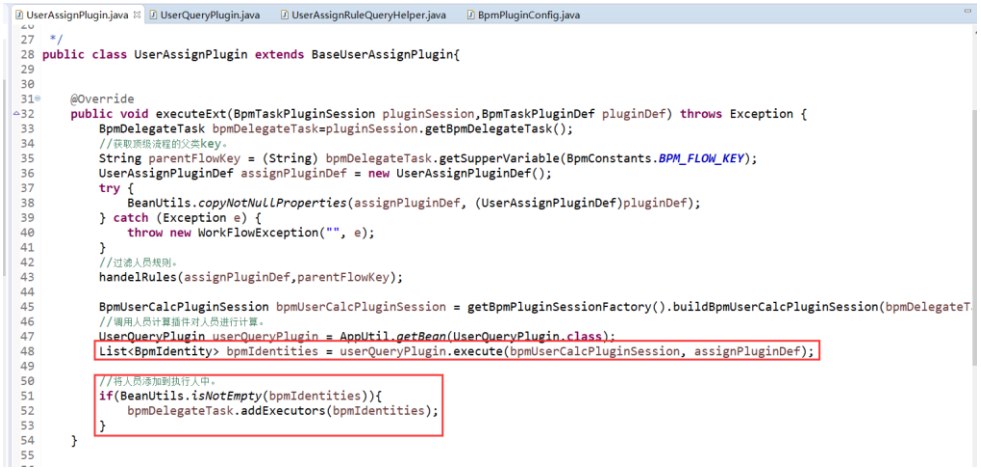
6.7 节点人员



如上图所示，节点人员的可选策略默认包含了以上选项，这些选项由如下的代码配置来限定。

这些策略可以根据需求再进行扩展，扩展的方式按照上一章节介绍的流程插件开发方式进行。不过我们也提供了人员脚本的插件，通过人员脚本可以解决绝大部分的人员查找需求。

在流程运行过程中，产生了待办任务以后，将这个待办分配给用户时，会通过以下代码完成待办任务的分配。



6.8 流程跳转

如果流程节点上配置了跳转规则，在该节点对应的待办任务审批时，会按照从上至下的顺序依次判定跳转规则是否成立，成立时就会跳转到规则指定的目标节点。

如下图所示，在下面的代码中对跳转的规则进行计算，如果返回了目标节点，则流程会跳转到该目标节点。

```java
// 3.计算跳转规则。
calcJumpRules(bpmTask,cmd);
}

/**
 * 计算跳转规则。
 * @param bpmTask
 * @param cmd
 * void
 * @throws Exception
 */
private void calcJumpRules(BpmTask bpmTask,DefaultTaskFinishCmd cmd) throws Exception{
    String nodeId = bpmTask.getNodeId();
    String defId = bpmTask.getProcDefId();
    UserTaskNodeDef bpmNodeDef = (UserTaskNodeDef) bpmDefinitionAccessor.getBpmNodeDef(defId, nodeId);
    List<DefaultJumpRule> ruleList = bpmNodeDef.getJumpRuleList();
    if (BeanUtils.isEmpty(ruleList)) return ;

    Map<String, Object> vars = natTaskService.getVariables(bpmTask.getTaskId());
    //加入实例对象。
    Map<String, ObjectNode> boMap= BpmContextUtil.getBoFromContext();

    vars.putAll(cmd.getVariables());
    vars.putAll(boMap);
    String targetNode = jumpRuleCalc.eval(ruleList, vars);
    if (StringUtil.isNotEmpty(targetNode)){
        cmd.setDestination(targetNode);
    }
}
```

6.9 流程事件脚本

流程提供了事件切入口来执行脚本，这些事件包括：流程启动、任务创建、会签任务创建、任务完成、流程结束、子流程启动、子流程结束这些。

如下图所示，分别是任务类、活动类两类的监听器来触发事件脚本的执行。脚本执行过程中的上下文变量，通过 vars 变量传入。

```java
protected abstract ScriptType getScriptType();

private void exeEventScript(BpmDelegateTask delegateTask) throws Exception{
    String bpmnDefId=delegateTask.getBpmnDefId();
    String defId =bpmDefinitionService.getDefIdByBpmnDefId(bpmnDefId);
    String nodeId=delegateTask.getTaskDefinitionKey();
    BpmNodeDef nodeDef= bpmDefinitionAccessor.getBpmNodeDef(defId, nodeId);

    ScriptType scriptType= getScriptType();
    String script=nodeDef.getScripts().get(scriptType);
    if(StringUtil.isEmpty(script)) return;


    Map<String, Object> vars=delegateTask.getVariables();
    ActionCmd cmd= ContextThreadUtil.getActionCmd();

    Map<String,ObjectNode> boMap= BpmContextUtil.getBoFromContext();
    if(BeanUtils.isNotEmpty(boMap)){
        vars.putAll(boMap);
    }

    vars.put("nodeDef", nodeDef);
    vars.put("task", delegateTask);
    vars.put("cmd", cmd);

    try {
        groovyScriptEngine.execute(script, vars);
    } catch (BusinessException e) {
        throw new WorkFlowException(e.getMessage());
    } catch (Exception e) {
        StringBuffer sb = new StringBuffer();
        sb.append("<br/><br/>流程在节点："+nodeDef.getName()+"("+nodeDef.getNodeId()+")执行"+scriptType.get
        sb.append("<br/>请联系管理员！");
        sb.append("<br/>可能原因为："+e.getMessage());
        sb.append("<br/>执行脚本为："+script);
        sb.append("脚本变量："+vars.toString());
        throw new WorkFlowException(sb.toString());
    }
```

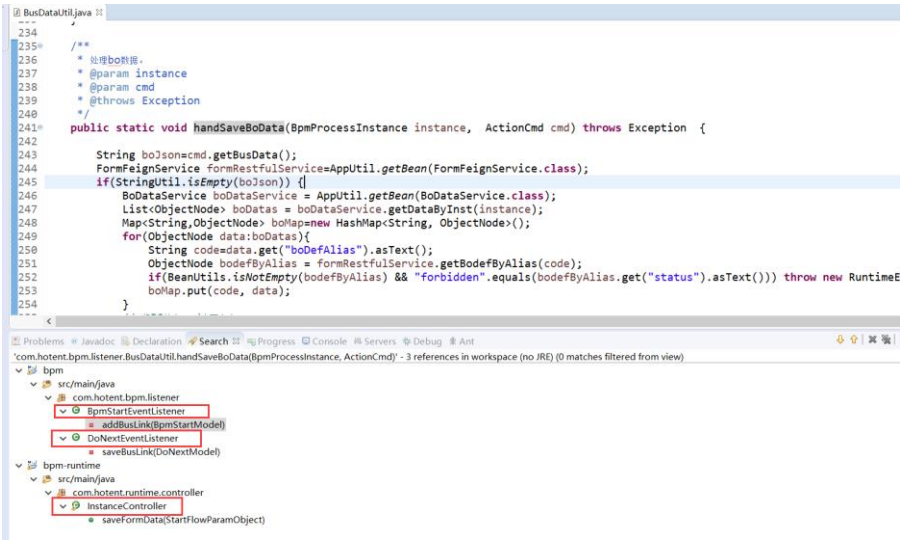
```java
AbstractTaskListener.java  AbstractExecutionListener.java
     * @throws Exception
     */
    private void exeEventScript(BpmDelegateExecution bpmDelegateExecution) throws Exception{
        String bpmnDefId=bpmDelegateExecution.getBpmnDefId();
        String defId =bpmDefinitionService.getDefIdByBpmnDefId(bpmnDefId);
        String nodeId=bpmDelegateExecution.getNodeId();
        if(StringUtil.isEmpty(nodeId)){
            return;
        }
        BpmNodeDef nodeDef= bpmDefinitionAccessor.getBpmNodeDef(defId, nodeId);

        ScriptType scriptType= getScriptType();
        String script=nodeDef.getScripts().get(scriptType);
        if(StringUtil.isEmpty(script)) return;

        Map<String, Object> vars=bpmDelegateExecution.getVariables();
        //流程实例ID

        ActionCmd cmd= ContextThreadUtil.getActionCmd();
        //BO上下文
        Map<String,ObjectNode> boMap= BpmContextUtil.getBoFromContext();
        if(BeanUtils.isNotEmpty(boMap)){
            vars.putAll(boMap);
        }

        vars.put("nodeDef", nodeDef);
        vars.put("execution", bpmDelegateExecution);
        vars.put("cmd", cmd);
        groovyScriptEngine.execute(script, vars);
    }
```

6.10 流程表单

表单的设置与保存通过如下代码完成。

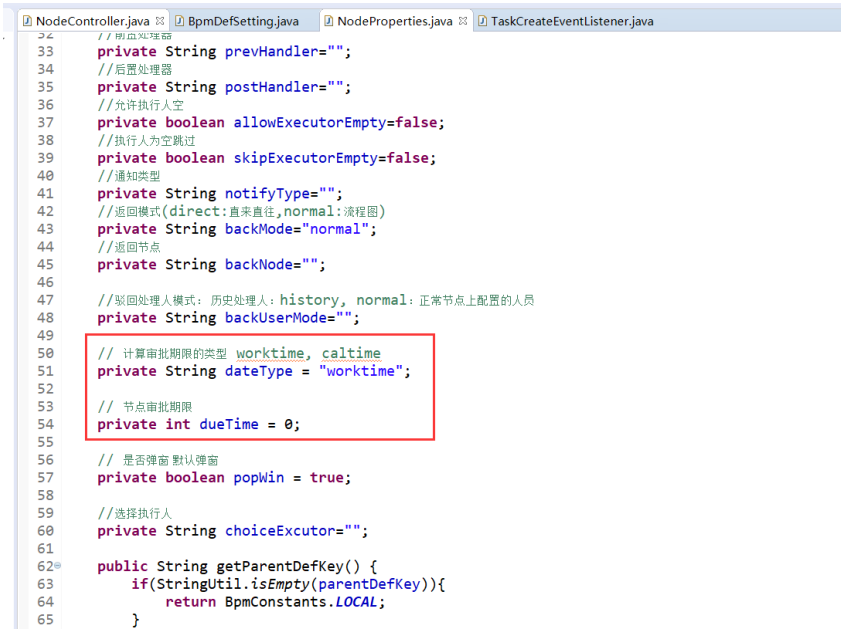
```java
NodeController.java  BpmDefSetting.java  NodeProperties.java
        }
        //保存表单
        BpmDefSettingBpmDefXmlHandler bpmDefSettingBpmDefXmlHandler = AppUtil.getBean(BpmDefSettingBpmDefXmlHandler.class);
        try {
            Map<String,Object> map = new HashMap<>();
            map.put("id", defId);
            map.put("rev", vo.getRev());
            DefaultBpmDefinition defaultBpmDefinition1 = bpmDefinitionManager.getBpmDefinitionByRev(map);
            if(BeanUtils.isNotEmpty(defaultBpmDefinition1)) {
                defSettingJson = dealRestFulHeader(defSettingJson);
                BpmDefSetting bpmDefSetting = JsonUtil.toBean(defSettingJson, BpmDefSetting.class);

                Set<String> formKeys = new HashSet<>();

                if (BeanUtils.isNotEmpty(bpmDefSetting)) {
                    if (BeanUtils.isNotEmpty(bpmDefSetting.getGlobalForm()))
                        formKeys.add(bpmDefSetting.getGlobalForm().getFormValue());
                    if (BeanUtils.isNotEmpty(bpmDefSetting.getGlobalMobileForm()))
                        formKeys.add(bpmDefSetting.getGlobalMobileForm().getFormValue());
                    if (BeanUtils.isNotEmpty(bpmDefSetting.getInstForm()))
                        formKeys.add(bpmDefSetting.getInstForm().getFormValue());
                    if (BeanUtils.isNotEmpty(bpmDefSetting.getInstMobileForm()))
                        formKeys.add(bpmDefSetting.getInstMobileForm().getFormValue());
                    if (BeanUtils.isNotEmpty(bpmDefSetting.getNodeForms())) {
                        List<Form> nodeForms = bpmDefSetting.getNodeForms();
                        for (Form form : nodeForms) {
                            if (BeanUtils.isNotEmpty(form)) formKeys.add(form.getFormValue());
                        }
                    }
                }
                Map<String, ObjectNode> boMap = new HashMap<>();
                FormFeignService formRestfulService = AppUtil.getBean(FormFeignService.class);
                for (Iterator<String> iterator = formKeys.iterator(); iterator.hasNext(); ) {
                    String formkey = iterator.next();
                    if (StringUtil.isNotEmpty(formkey)) {
                        List<ObjectNode> list = formRestfulService.getFormBoLists(formkey);
                        for (ObjectNode objectNode : list) {
```

流程绑定了表单以后，在流程启动、待办审批、保存表单数据时，会对表单的数据进行保存和更新的处理，如下图所示。



6.11 审批时限

审批时限的设置通过下列属性进行设置。



在设置了审批时限以后，在产生待办任务时，会根据审批时限类型及时长计算这个待办任务的到期时间，并将其设置到任务到期时间这个字段上。

```java
int dueTimeMin = 0;
String dateTpye = "";
if( nodeProperties.getDueTime()!=0 ){
    dueTimeMin = nodeProperties.getDueTime();
    dateTpye = nodeProperties.getDateType();
}else{
    dueTimeMin = defExt.getExtProperties().getDueTime();
    dateTpye = defExt.getExtProperties().getDateType();
}

if(dueTimeMin==0) return;

if("caltime".equals(dateTpye)){
    dueTime = TimeUtil.getLocalDateTimeByMills(TimeUtil.getNextTime(TimeUtil.MINUTE, dueTimeMin,TimeUtil.getTimeMillis(bpmTask.getCreateTime())));
}else{
    // 设置第一个执行人的工作日
    if(BeanUtils.isNotEmpty(identityList)){
        BpmIdentity bpmIdentity = identityList.get(0);
        if(BpmIdentity.TYPE_USER.equals(bpmIdentity.getType())){
            userId = bpmIdentity.getId();
            userName = bpmIdentity.getName();
            ObjectNode params=JsonUtil.getMapper().createObjectNode();
            params.put("userId", identityList.get(0).getId());
            if(BeanUtils.isNotEmpty(bpmTask.getCreateTime())){
                params.put("startTime", DateFormatUtil.formaDatetTime(bpmTask.getCreateTime()));
            }
            params.put("time", dueTimeMin);
            String dueTimeStr = PortalFeignService.getEndTimeByUser(params);
            dueTime = DateFormatUtil.parse(dueTimeStr);
        }
    }
}

BpmTaskDueTime bpmTaskDueTime = new BpmTaskDueTime();
bpmTaskDueTime.setId(UniqueIdUtil.getSuid());
bpmTaskDueTime.setDateType(dateTpye);
bpmTaskDueTime.setDueTime(dueTimeMin);
bpmTaskDueTime.setRemainingTime(dueTimeMin);
bpmTaskDueTime.setExpirationDate(dueTime);
bpmTaskDueTime.setInstId(bpmTask.getProcInstId());
bpmTaskDueTime.setTaskId(bpmTask.getTaskId());
bpmTaskDueTime.setStartTime(bpmTask.getCreateTime());
bpmTaskDueTime.setUserId(userId);
bpmTaskDueTime.setUserName(userName);
bpmTaskDueTime.setIsNew((short)1);
bpmTaskDueTime.setStatus((short)0);

bpmTaskDueTimeManager.create(bpmTaskDueTime);
```

6.12 流程催办

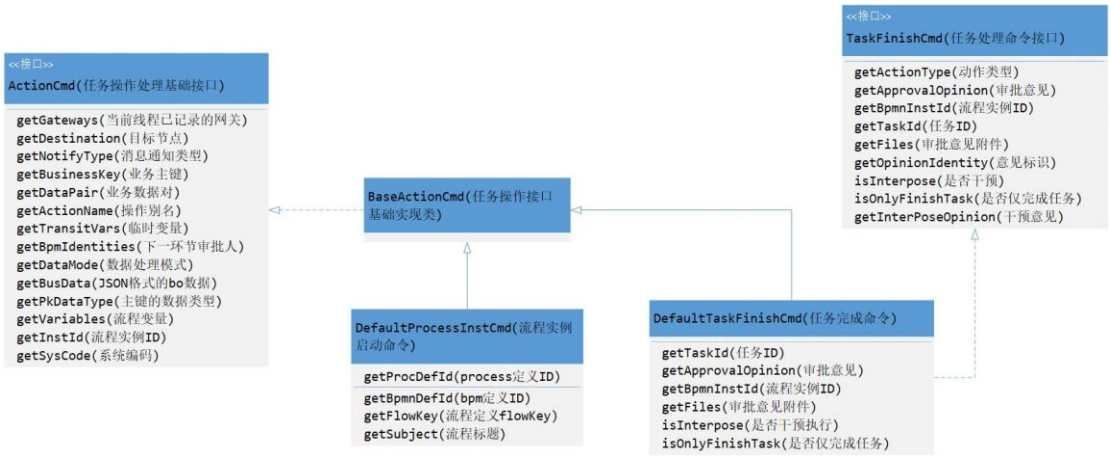
流程的催办现在由定时任务来触发，在触发时会查询当前需要进行催办的数据，并关联到对应的待办任务，对相应的待办审批人发送催办消息，如果符合了到期规则，则会相应的执行到期动作。

```java
@Override
public void executeTaskReminderJob() throws Exception {
    //获取当前时间要出发的所有催办项
    List<BpmTaskReminder> reminderList = this.getTriggerReminders();


    for (BpmTaskReminder reminder : reminderList) {
        DefaultBpmTask task = bpmTaskManager.get(reminder.getTaskId());

        if(BeanUtils.isEmpty(task)){
            this.remove(reminder.getId());
            continue;
        }

        task.setIdentityList(bpmIdentityService.searchByNode(task.getProcInstId(),task.getNodeId()));

        // 如果执行到期 动作任务被处理。则继续
        boolean taskComplate = executeDueAction(reminder,task);

        //发送催办
        if(reminder.getIsSendMsg()==1) sendMsg(reminder,task);

        if(taskComplate) continue;

        //处理预警
        handleWarning(reminder,task);

        // 如果预警为无动作，无催办，无预警。则删除改催办
        if(BpmTaskReminder.TASK_DUE_ACTION_NO_ACTION.equals(reminder.getDueAction())
                && (reminder.getIsSendMsg()==0 || reminder.getMsgCount()<=0)
                && StringUtil.isEmpty(reminder.getWarningset())){
            this.remove(reminder.getId());
            return;
        }
        // 更新催办
        this.update(reminder);
    }
```

6.13 流程操作按钮

序号	操作名称	别名	操作说明	功能分类	是否默认
1	启动	startFlow		起草	是
2	同意	agree		待办	是
3	流程图	flowImage		实例	是
4	打印	print		实例	是
5	保存草稿	saveDraft		起草	是
6	抄送	instanceTrans	抄送时产生知会任务，审批记录中挂载到当前的审批环节。	待办	否
7	沟通	startCommu	沟通时产生知会任务，沟通发起时，流程实例处理哪个环节，审批记录中就挂载到这个环节。	实例	是
8	流转	startTrans		待办	否
9	驳回	reject	驳回分为驳回上一步、驳回发起人、驳回指定节点。	待办	是
10	反对	oppose	只在会签时出现。	待办	是
11	终止	endProcess		实例	否
12	反馈	commu	对沟通进行反馈，注意：对于沟通任务，不做反馈也不会影响流程的审批。	待办	是
13	转办	delegate		待办	是
14	锁定和解锁	lockUnlock	当节点为用户任务，而且有多个审批候选人时，显示该按钮。	待办	是
15	任务延期	taskDelay		待办	否
16	征询	inqu	当前待办任务流转到指定的被征询人那里，被征询人处理后回到本节点。	待办	是
17	回复	inqu_reply	对征询进行回复。	待办	是
18	跟踪	follow	跟踪指定节点，当指定节点完成审批时产生知会任务。	实例	是
19	加签	addSign		待办	是

流程的操作通过构建操作命令来实现，命令对象的类图如下所示：



在执行不同的操作时，通过不同的参数构建不同的命令对象来执行相应的流程操作。以启动流程为例，构建启动流程的命令并执行流程发起的操作，如下图所示：

```
575
576    @RequestMapping(value = "startForm", method = RequestMethod.POST, produces = {"application/json; charset=utf-8"})
577    @ApiOperation(value = "业务数据模板中 启动流程实例", httpMethod = "POST", notes = "业务数据模板中 启动流程实例")
578    public CommonResult<String> startForm(
579            @ApiParam(required = true, name = "defId", value = "流程定义id") @RequestParam String defId,
580            @ApiParam(required = true, name = "businessKey", value = "业务主键") @RequestParam String businessKey,
581            @ApiParam(required = true, name = "boAlias", value = "bo定义alias") @RequestParam String boAlias) throws Exception{
582        try{
583            DefaultProcessInstCmd cmd= new DefaultProcessInstCmd();
584            cmd.setProcDefId(defId);
585            cmd.setDataMode(ActionCmd.DATA_MODE_BO);
586            cmd.setActionName("startFlow");
587            // 获取编辑数据
588            ObjectNode formRestParams=JsonUtil.getMapper().createObjectNode();
589            formRestParams.put("saveType", "database");
590            formRestParams.put("boid", businessKey);
591            formRestParams.put("code", boAlias);
592            ObjectNode boData = formRestfulService.getBodataById(formRestParams);
593            cmd.setBusData(JsonUtil.toJson(BoDataUtil.hanlerData(Arrays.asList(boData))));
594            processInstanceService.startProcessInst(cmd);
595            return new CommonResult<String>(true, "流程启动成功", "");
596        }
597        catch (Exception e) {
598            return new CommonResult<String>(false, "流程启动失败："+e.getMessage(), "");
599        }
600    }
601
```

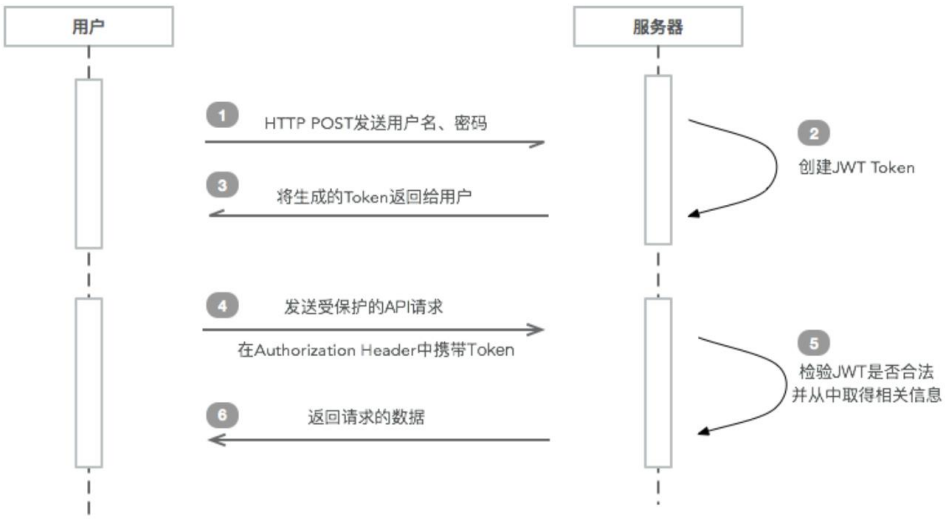
7 专题

7.1 认证机制

7.1.1.1 Jwt 认证机制

JSON Web Token（JWT）是一个非常轻巧的规范，这个规范允许我们使用 JWT 在用户和服务器之间传递安全可靠的信息。它是基于 RFC 7519 标准定义的一种可以安全传输的小巧和自包含的 JSON 对象。由于数据是使用数字签名的，所以是可信任的和安全的。JWT 可以使用 HMAC 算法对 secret 进行加密或者使用 RSA 的公钥私钥对其进行签名。

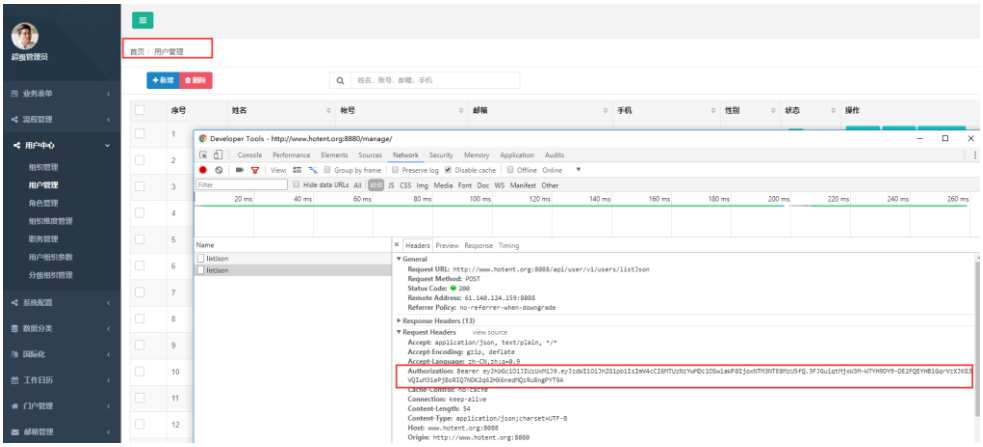
- **认证时序图**



如上图所示：

1. 用户导航到登录页，输入用户名和密码，进行登录
2. 服务器对登录用户进行认证，如果认证通过，根据用户的信息和 JWT 的生成规则生成 JWT Token
3. 服务器将该 Token 字符串返回
4. 客户端得到 Token 信息，将 Token 存储在 localStorage、sessionStorage 或 cookie 等存储形式中。
5. 当用户请求服务器 API 时，在请求的 Header 中加入 Authorization：Token。
6. 服务端对此 Token 进行校验如下图，如果合法就解析其中内容，根据其拥有的权限和自己的业务逻辑给出响应结果，如果不通过，返回 HTTP 401。

7. 用户进入系统，获得请求资源



```
28 public class JwtAuthorizationTokenFilter extends OncePerRequestFilter {
29     private final Logger logger = LoggerFactory.getLogger(this.getClass());
30
31     private UserDetailsService userDetailsService;
32     private JwtTokenHandler jwtTokenHandler;
33     private String tokenHeader;
34
35     public JwtAuthorizationTokenFilter(UserDetailsService userDetailsService, JwtTokenHandler jwtTokenHandler, String tokenHea
36         this.userDetailsService = userDetailsService;
37         this.jwtTokenHandler = jwtTokenHandler;
38         this.tokenHeader = tokenHeader;
39     }
40
41     @Override
42     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws Servle
43         logger.debug("processing authentication for '{}'", request.getRequestURL());
44
45         final String requestHeader = request.getHeader(this.tokenHeader);
46
47         String username = null;
48         String authToken = null;
49         if (requestHeader != null && requestHeader.startsWith("Bearer ")) {
50             authToken = requestHeader.substring(7);
51             try {
52                 username = jwtTokenHandler.getUsernameFromToken(authToken);
53             } catch (IllegalArgumentException e) {
54                 logger.error("an error occured during getting username from token", e);
55
56                 if(isAjax(request)){
57                     throw new BaseException(HttpStatus.UNAUTHORIZED, HttpStatus.UNAUTHORIZED.value(), "从token中获取的账号错误");
58                 }else{
59                     response.sendError(HttpStatus.UNAUTHORIZED.value(), e.getMessage());
60                 }
61
62             } catch (ExpiredJwtException e) {
```

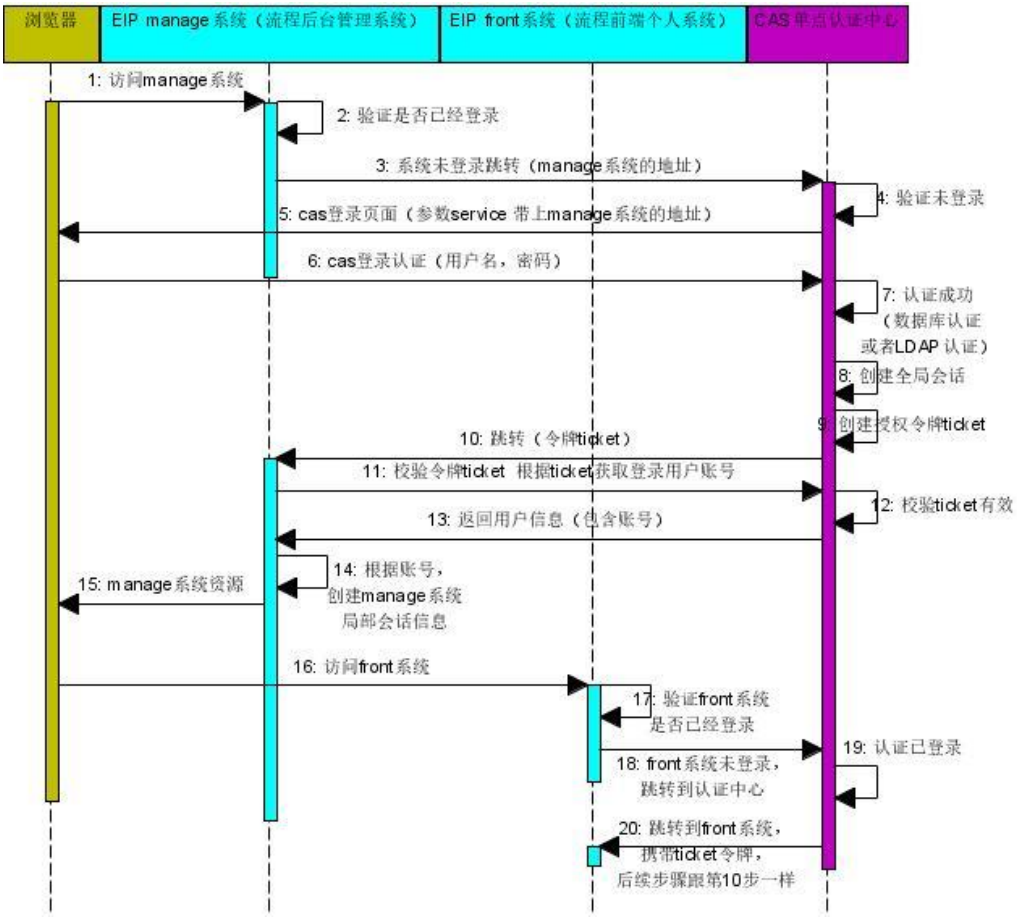
➢ 相应代码

模块	文件	说明
base	com.hotent.base.controller.AuthenticationRestController.java	createAuthenticationToken 方法认证用户，认证通过返回 jwttoken 到前端
web	manage\js\services.js	Login 方法进行了用户登录请求，登录成功后将返回的信息（包含了 jwt token）保存到$sessionStorage 中，并且给请求都设置默认的 Authorization
base	com.hotent.base.conf.FeignConfig.java	服务之间的调用通过将用户请求的 header 中的信息 Authorization，设置到 feign 请求中
base	com.hotent.base.filter.	服务端对此 Token 进行校验(包含

	JwtAuthorizationTokenFilter.java	两种 token　jwt 和 basic)

7.2　单点登录

单点集成原理



本系统集成了 cas 单点登录，由于系统是采用前后端分离的系统架构，在这里采用的是 restful 接口来进行单点登录认证的，所以单点服务器需要添加 jar 来支持 restful 接口认证。

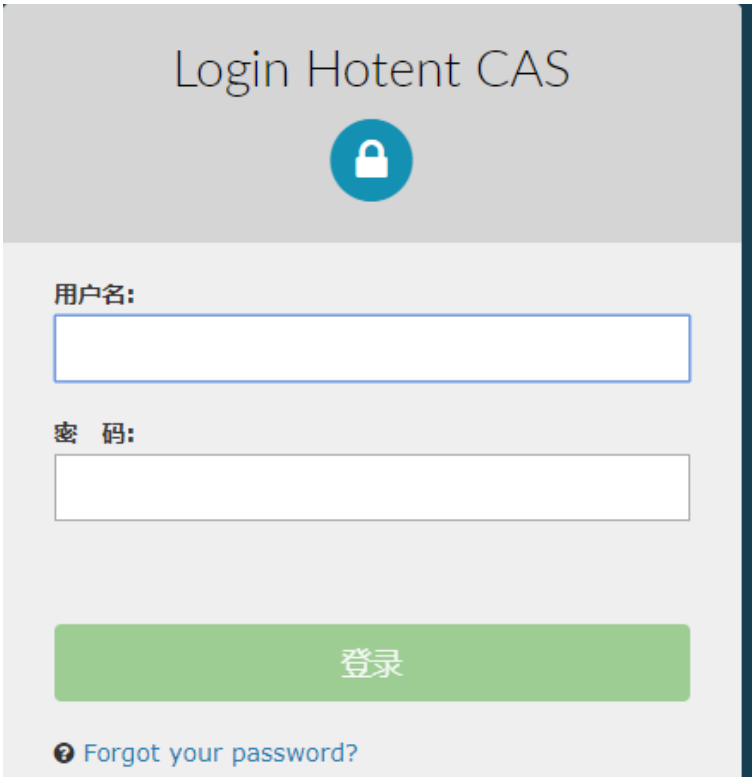
在此以 5.2.6 版本讲解，　需要给 cas server 添加支持 restful 接口认证的 jar 如上图。在本系统是通过 restful 接口去单点服务器认证用户，如果用户认证成功

则返回 jwt token 给前端作为下次请求的认证信息。

在 application-dev.yml 中配置单点登录信息，cas. useSecurityCas 的值为 false / true false 值不使用单点登录， true 使用单点登录。cas.server.url 为单点登录的服务器地址

```
106 # 单点登录配置
107 sso:
108   enable: true
109   mode: basic
110   cas:
111     url: http://www.hotent.org:7080/cas     #CAS服务地址
112   oauth:
113     url: http://192.168.1.211:8093  #oauth2服务地址
114     loginPath: /oauth/authorize
115     tokenPath: /oauth/token
116     checkPath: /oauth/check_token
117     clientId: eip7
118     secret: flossy
119
```

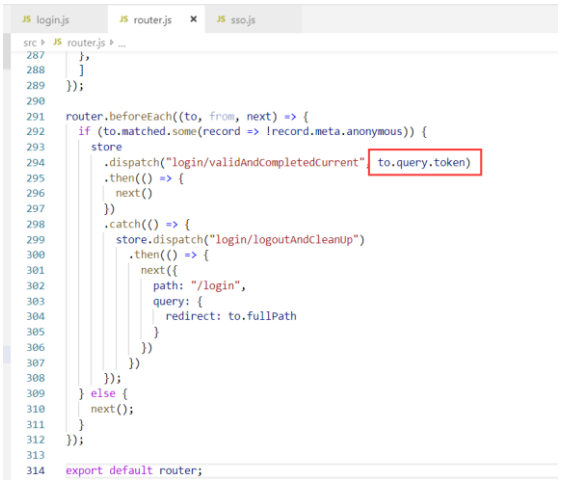
在前端需要改在的地方，当判断用户还没有登录系统时，前端会请求后台，获取 sso.enable 配置，如果为 false，这直接路由到原来的登录页面，如果为 true，则重定向到单点登录的服务器中。



登录后再重定向到前端页面，逻辑代码在 sso.js 中，sso.js 作为前端的统一入口，通过以下逻辑代码可以实现单点登录的跳转。

```
// 返回后台的context path
window.domainName = 'http://127.0.0.1:8088';
window.ueditor = 'http://127.0.0.1:8088';
window.context = {
  portal: 'http://127.0.0.1:8088',
  bpmRunTime: 'http://127.0.0.1:8088',
  bpmModel: 'http://127.0.0.1:8088',
  uc: 'http://127.0.0.1:8088',
  form: 'http://127.0.0.1:8088',
  ssoUrl: 'http://www.hotent.com:7080'
};

var currentUser = window.sessionStorage.getItem("currentUser"), //当前如果已经登录了，则session中可以获取到当前登录用户
  ticket = getUrlParam("token");                                 //如果是从单点登录服务器跳回来时，在url中会携带token字符串

//当前既没有登录，也没有token信息时，重定向到单点登录页面
if (!currentUser && !ticket) {
  var redirect_uri = window.location.href;
  redirect_uri = encodeURIComponent(redirect_uri);
  var ssoUrl = window.context['ssoUrl'];
  window.location.href = ssoUrl + "/cas/login?redirect_uri=" + redirect_uri;
}
```

完成单点登录后，在 URL 地址中会携带 token 字符串，此时会进入 router 的路由守卫，如下图所示，token 会被提交到 validAndCompletedCurrent 方法中。



```
router.beforeEach((to, from, next) => {
  if (to.matched.some(record => !record.meta.anonymous)) {
    store
      .dispatch("login/validAndCompletedCurrent", to.query.token)
      .then(() => {
        next()
      })
      .catch(() => {
        store.dispatch("login/logoutAndCleanUp")
          .then(() => {
            next({
              path: "/login",
              query: {
                redirect: to.fullPath
              }
            })
          })
      });
  } else {
    next();
  }
});

export default router;
```

在 validAndCompletedCurrent 方法中，会根据 token 是否有值来判断是否调用后台的 Sso 方法完成单点认证并获取到 jwt。

```
validAndCompletedCurrent({ commit, state }, token) {
    return new Promise((resolve, reject) => {
        if (state.currentUser) {
            resolve();
        }
        else {
            const user = sessionStorage.getItem("currentUser")
            if (user) {
                commit("setCurrentUser", JSON.parse(user))
                resolve();
            }
            else if (token) {
                uc.basicSso(token, user => {
                    sessionStorage.setItem("currentUser", JSON.stringify(user));
                    commit("setCurrentUser", user);
                    resolve();
                }, () => {
                    reject();
                });
            }
            else {
                reject()
            }
        }
    })
},
```

7.3 权限控制

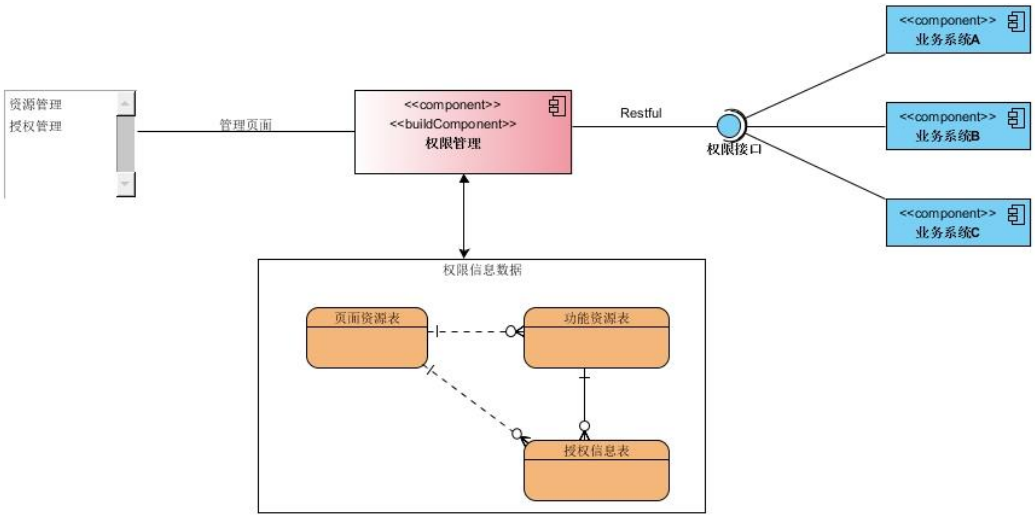
7.3.1 后台权限控制

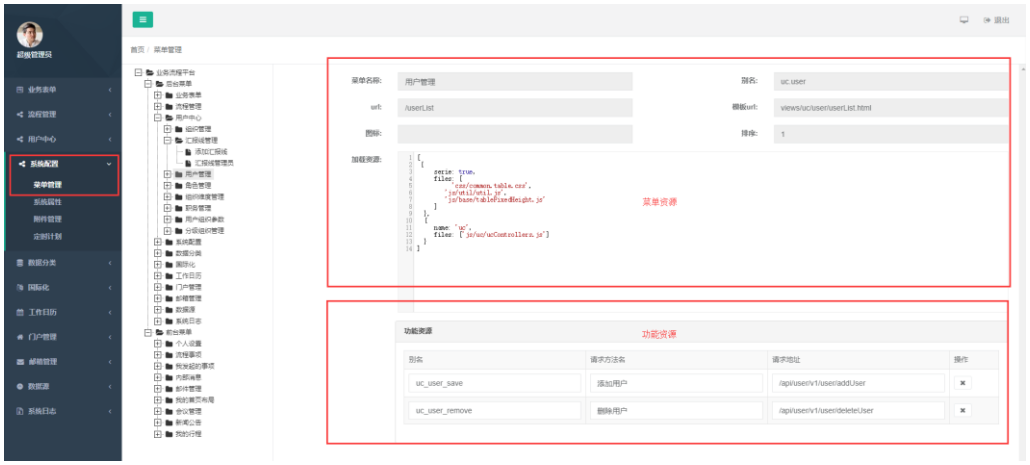
将权限管理设计为独立应用，对所有的微服务的资源，和授权统一在 portal 服务中统一维护。在 base 模块中通过接口获取权限信息后实现权限控制。

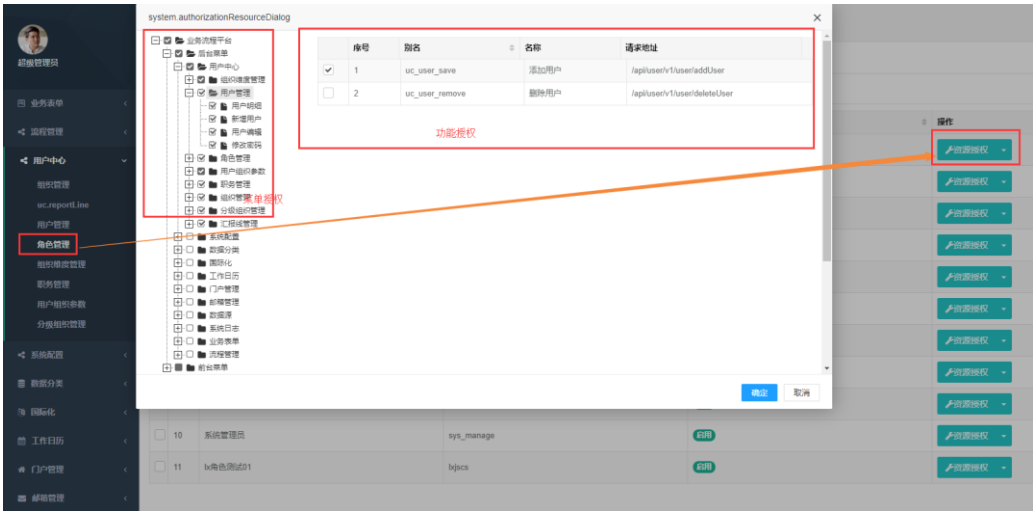
- **架构设计**

如下图所示，权限管理提供界面维护两方面的信息。

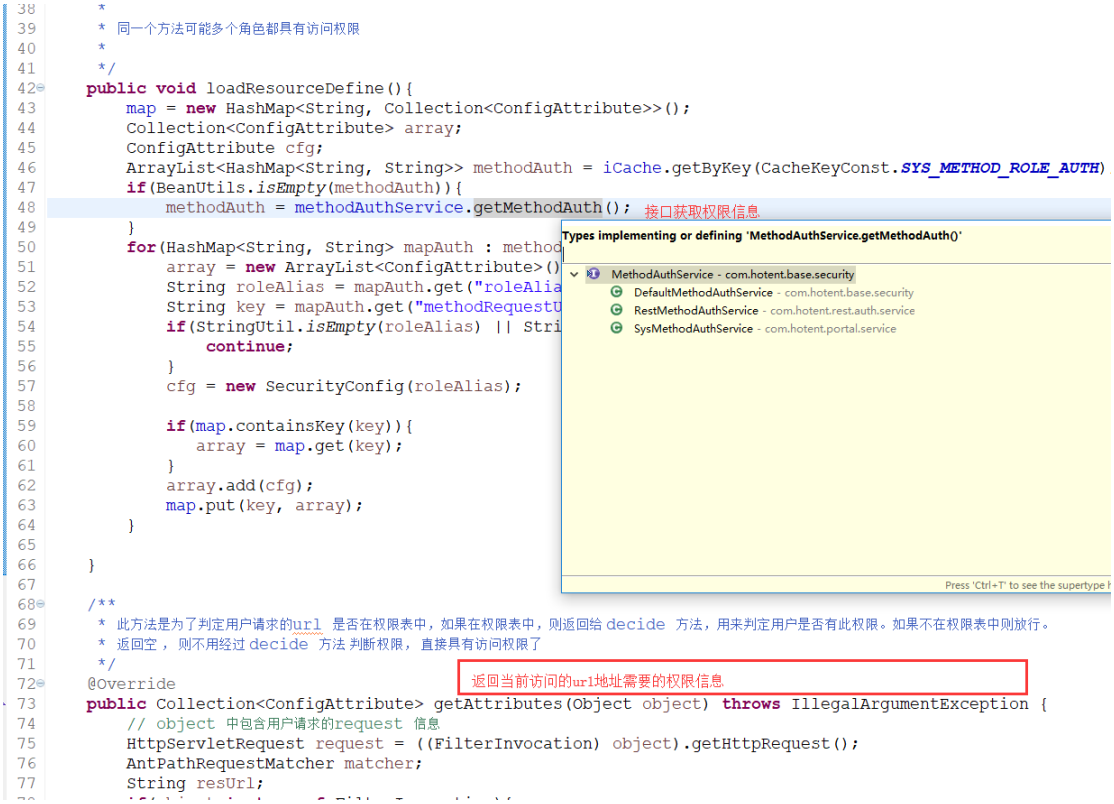
- 资源信息，包括页面资源和功能资源
- 授权信息， 主要是维护角色和资源信息的映射关系，另外以接口方式提供权限信息查询。







base\src\main\java\com\hotent\base\security\HtInvocationSecurityMetadataSourceService.java　该类是返回当前请求地址，需要哪些角色的权限才能请求当前接口请求的。



base\src\main\java\com\hotent\base\security\HtDecisionManager.java　决策类，用于判断当前用户是否有权限访问当前的接口（后台请求地址）

```java
public class HtDecisionManager implements AccessDecisionManager {

    @SuppressWarnings("unchecked")
    @Override
    public void decide(Authentication authentication, Object object,Collection<ConfigAttribute> configAttributes)
            throws AccessDeniedException, InsufficientAuthenticationException {

        if (BeanUtils.isEmpty(configAttributes))
            return;

        Collection<GrantedAuthority> authorities = (Collection<GrantedAuthority>) authentication
                .getAuthorities();

        for (GrantedAuthority grantedAuthority : authorities) {
            // 超级管理员
            if (PlatformConsts.ROLE_SUPER.equals(grantedAuthority.getAuthority())) {
                return;
            }
        }

        for (ConfigAttribute configAttribute : configAttributes) {
            if (StringUtil.isEmpty(configAttribute.getAttribute())||
                PlatformConsts.PERMIT_All.equals(configAttribute.getAttribute())) {
                // 有权限
                return;
            }
        }

        for (GrantedAuthority grantedAuthority : authorities) {
            for (ConfigAttribute configAttribute : configAttributes) {
                if (grantedAuthority.getAuthority().equals(
                        configAttribute.getAttribute())) {
                    // 有权限
                    return;
                }
            }
```

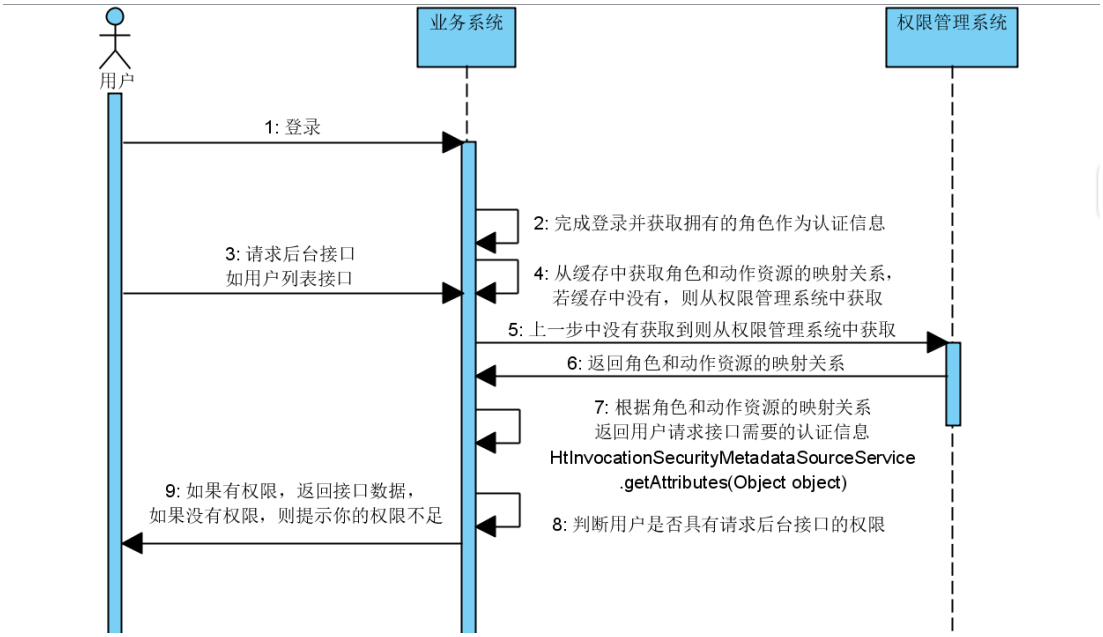
当前用户信息　请求信息　接口需要的权限信息

如果是超级管理员具有权限

如果当前用户具有接口访问需要的权限， 则有权限

➢ 时序说明

如下图所示，对权限控制的过程进行了说明，在第 6 步时实现了对前端显示的权限控制，同时权限信息已缓存到业务系统中，为了实现更安全的权限控制，在后续的用户操作中，仍可在后端对用户的动作进行权限判断。



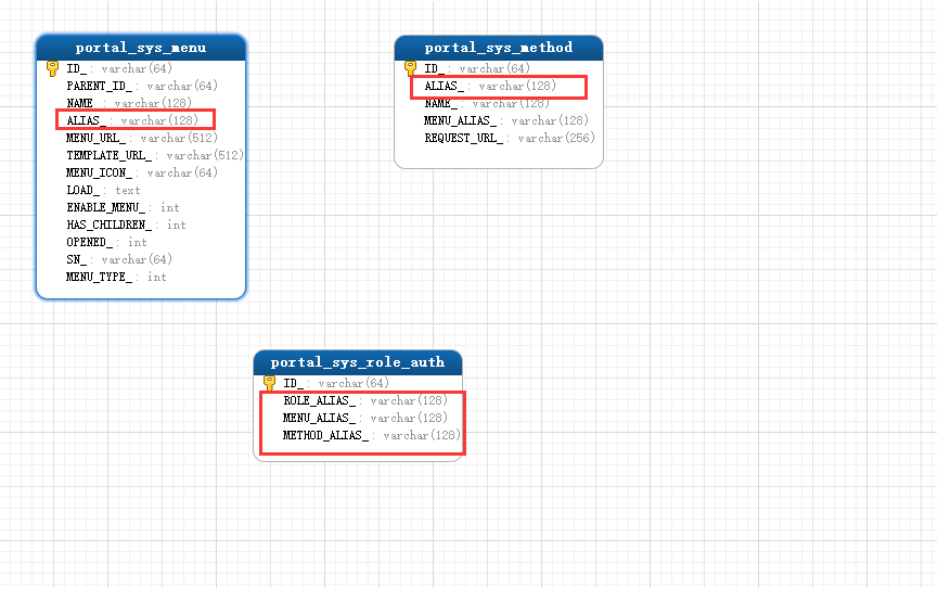
➢ 数据库设计

如下图所示，设计了三张表来存放权限信息。

- 页面资源表 页面资源进行严格权限控制，即未授权的页面资源，用户无

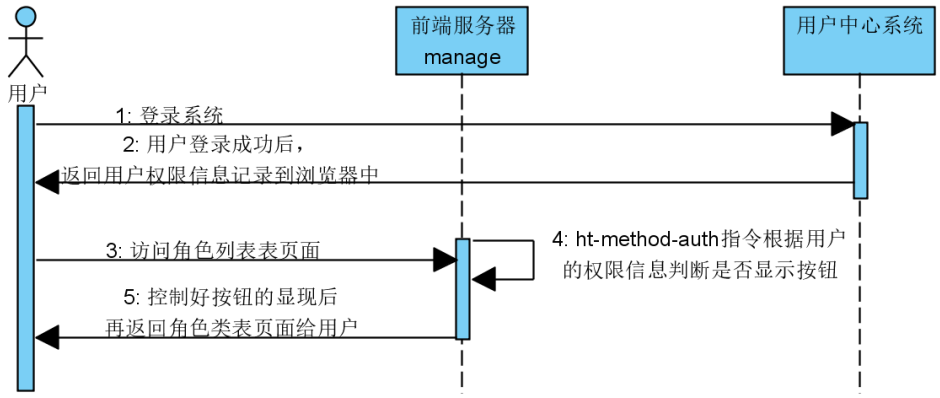
法访问，用户登录时通过接口返回用户的页面资源信息。

- 功能资源表 功能资源进行松散权限控制，即未维护到功能资源表中的功能，用户都可以访问，只有维护到功能资源表中的功能才进行权限验证，拥有权限的角色才允许访问。
- 授权信息表 授权信息表中的页面资源别名和功能资源别名分别来自页面资源表和功能资源表和角色类表中的别名。



7.3.2 前端按钮权限控制

➢ **时序图**



- **设计以及相关代码**

根据后台权限控制的基础，实现前端权限按钮控制显示和隐藏。 用户登录后，返回用户权限信息到前端，前端再通过 angular 指令去判断用户是否有相应的功能权限，如 ht-method-auth=”uc_role_remove”,uc_role_remove 为功能资源的别名, ht-method-auth 指令会判断如果功能资源具有 uc_role_remove 资源配置，并且当前用户是没有 uc_role_remove 功能资源授权的角色则页面将隐藏功能按钮

roleList.html

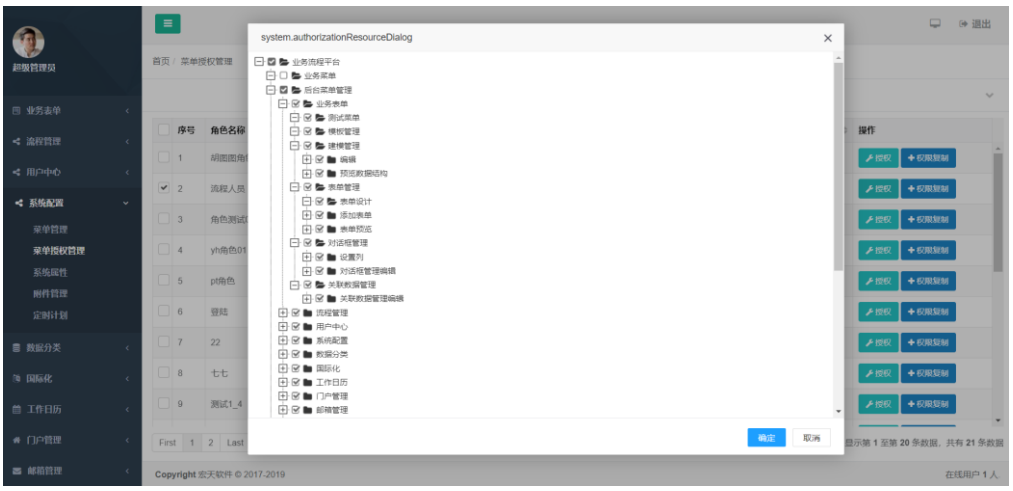
```
<div class="white-bg border-left animated fadeInRight" ng-controller="roleListCtrl">
    <div class="ibox" ht-data-table="roleListTable" url="${uc}/api/role/v1/roles/getRolePage">
        <div class="ibox-title no-borders">
            <div class="col-md-10 ">
                <div class="col-md-3 btn-group tools-panel">
                    <button type="button" class="btn btn-success btn-sm"
                            ng-click="operating('', 'add')" ht-method-auth="uc_role_save" >
                        <i class="fa fa-plus"></i> 新增
                    </button>
                    <button class="btn btn-danger btn-sm remove"
                            ht-remove-array url="${uc}/api/role/v1/role/deleteRoleByIds"
                            ht-method-auth="uc_role_remove" >
                        <i class="fa fa-trash"></i> 删除
                    </button>
                </div>
```

```
/**
 * 按钮权限指令 控制是否显示
 */
function htMethodAuth($sessionStorage){
    return {
        restrict:"AE",
        scope:{
            htMethodAuth:"@"
        },
        compile:function(tELe ,tAttrs,transcludeFn){
                //进行编译后的dom操作
                return{
                        pre:function(scope, iElement, iAttrs, controller){
                            // 在子元素被链接之前执行
                            // 在这里进行Dom转换不安全
                        },
                        post:function(scope, iElement, iAttrs, controller){
                            // 在子元素被链接之后执行
                            var methodAuth = $sessionStorage.methodAuth;
                            var allMethod = methodAuth.allMethod  || [] ;系统所配置的所有的功能资源 ["uc_role_remove","uc_role_get"]
                            var curUserMethod =  methodAuth.curUserMethod  || [] ; 当前用户具有的功能权限，如
                            // 无权限则隐藏按钮                                    ["uc_role_remove","uc_role_save"]
                            if( !curUserMethod.contains(scope.htMethodAuth) && allMethod.contains(scope.htMethodAuth) ){
                                $(iElement).hide(); 当前用户没有 改指令指定的权限 并且 已经配置在功能资源中的，则认为是没有权限的，隐藏按钮，否则限制按钮
                            }

                        }
                    }
        }
    };
}
```

文件	说明
web\src\main\webapp\manage\js\directives.js	htMethodAuth 方法实现了按钮权限的控制
web\src\main\webapp\manage\js\services.js	getCurrentUserMethodAuth 方法实现了用户登录后获取用户的权限信息，用于给 ht-method-auth 指令做判断使用

7.3.3 数据级别的权限控制

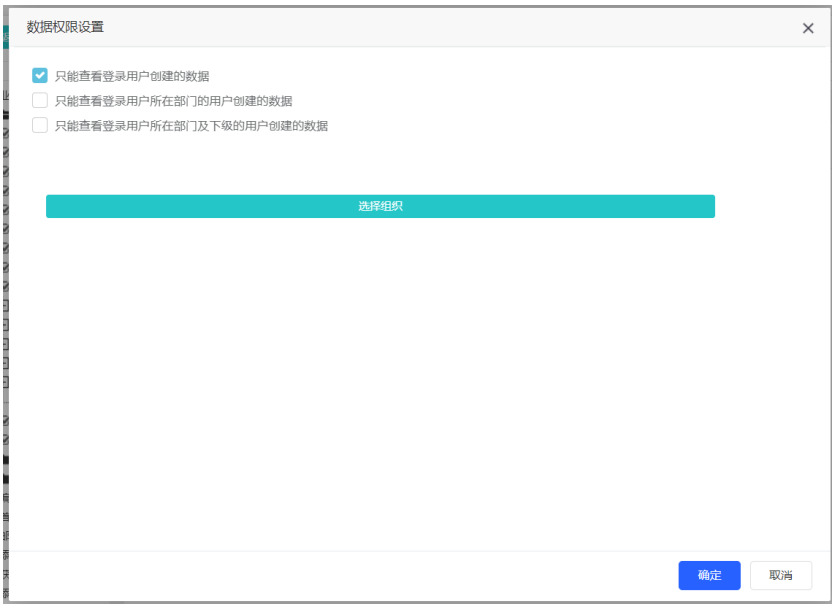
系统已经实现了对页面资源、功能资源的维护，通过角色与资源的授权，可以实现不同的用户登录系统时，根据其拥有的不同角色来控制其能访问的页面和能使用的功能。



现在要更进一步，做到数据级别的权限控制，例如：现在有技术部工程师、技术部经理两个角色的用户，他们都需要查看报销列表这个页面，但是要控制工

程师只能看到本人的数据，经理能看到技术部的所有数据，这就是数据级别的权限控制。

如下图所示，在维护权限映射时，映射有四种策略：本人、本部门、本部门及下属部门、指定部门。根据不同的角色选择不同的授权策略，四种策略任选其一。



- **表结构支持**

要实现数据权限控制，需要表结构中有相应字段来支持，例如上面的四种策略需要记录这条数据的创建人、创建人所属部门才能实现。

EIP 系统中的所有实体类均继承于 BaseMode，我们在设计 BaseMode 实体时已经设计了一些公共字段，如下图所示：

BaseModel.java

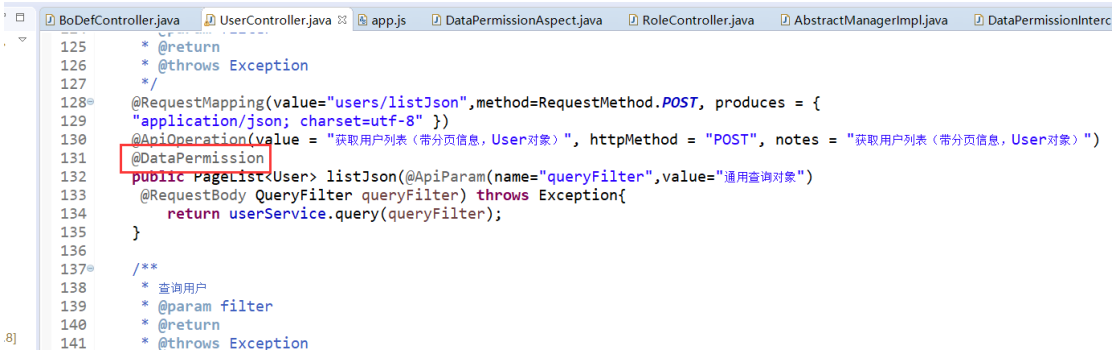
```
1  package com.hotent.base.model;
2
3  import java.io.Serializable;
9
10 /**
11  * 基础实体类
12  *
13  * @company 广州宏天软件股份有限公司
14  * @author heyifan
15  * @email heyf@jee-soft.cn
16  * @date 2018年4月4日
17  */
18 @ApiModel(description="基础实体类")
19 @XmlAccessorType(XmlAccessType.FIELD)
20 public abstract class BaseModel<PK extends Serializable> implements Serializable{
21     private static final long serialVersionUID = 3796984803158565007L;
22     @ApiModelProperty(name="createBy",notes="创建人ID")
23     protected String createBy;
24     @ApiModelProperty(name="updateBy",notes="更新人ID")
25     protected String updateBy;
26     @ApiModelProperty(name="createTime",notes="创建时间")
27     protected LocalDateTime createTime;
28     @ApiModelProperty(name="updateTime",notes="更新时间")
29     protected LocalDateTime updateTime;
30     @ApiModelProperty(name="createOrgId",notes="创建组织ID")
31     protected String createOrgId;
32
33     public final String getCreateBy() {
34         return createBy;
35     }
```

在开发新的业务功能时，我们只需要在设计物理表时，添加这两个列就可以了，注意列的命名，按照 EIP 系统的命名规范，数据库列名为：create_by_和 create_org_id_，这样在代码生成以后无需修改 mapper 文件，如果不按照这个命名则调整 mapper 文件也可以。

- **后台代码支持**

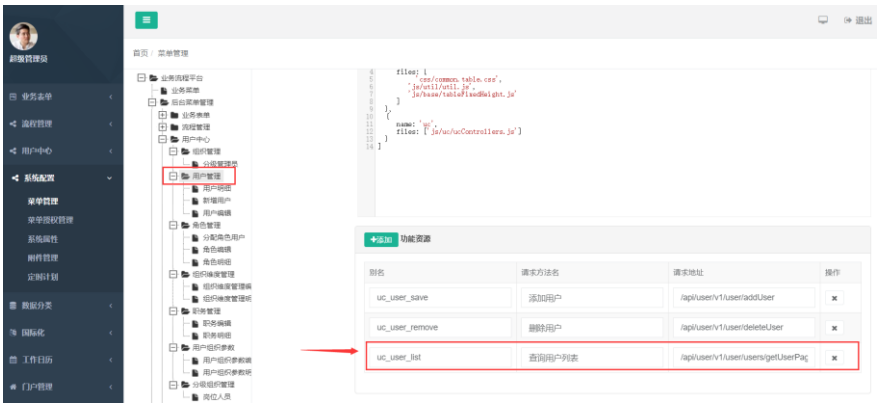
因为数据权限控制是需要牺牲一部分性能的，而大部分模块是不需要数据权限控制的，所以我们按照约定优于配置的原则，对于需要做数据权限控制的模块添加上注解来实现数据权限控制，其他未添加注解的模块不做数据权限控制。

例如，对于用户角色列表这个方法，如果要实现数据权限控制，我们需要在这个方法上添加 DataPermission 注解。添加了这个注解以后我们会通过 AOP 来实现数据权限控制，AOP 的特性让我们只需要编写通用的控制逻辑，后续开发新的业务模块时，我们唯一需要修改的代码就是添加这个注解即可。



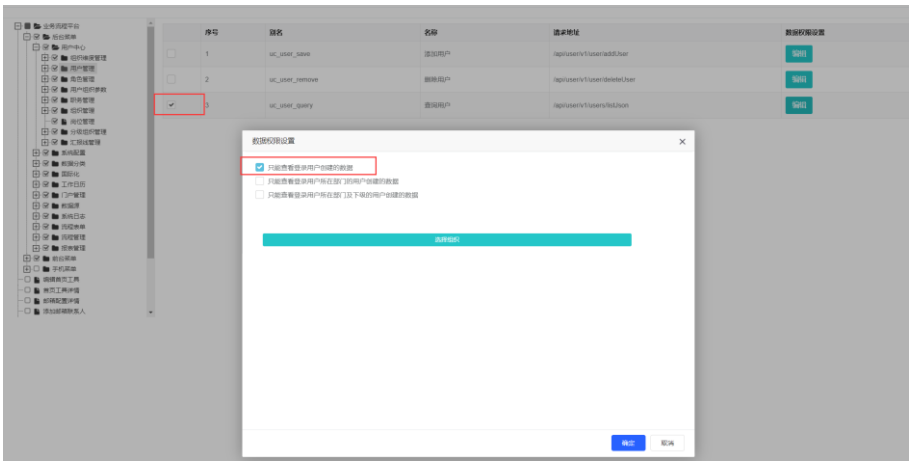
- **资源维护**

在资源维护界面，我们维护一个列表页面时，只维护了其 html 页面，对应的后台 restful 接口地址没有维护进来，所以需要实现数据权限控制，我们还需要在列表页面中添加一个功能资源指向这个后台功能。



- **角色授权资源**

维护了功能资源以后，在角色授权资源界面，可以针对功能资源进行数据权限控制授权。



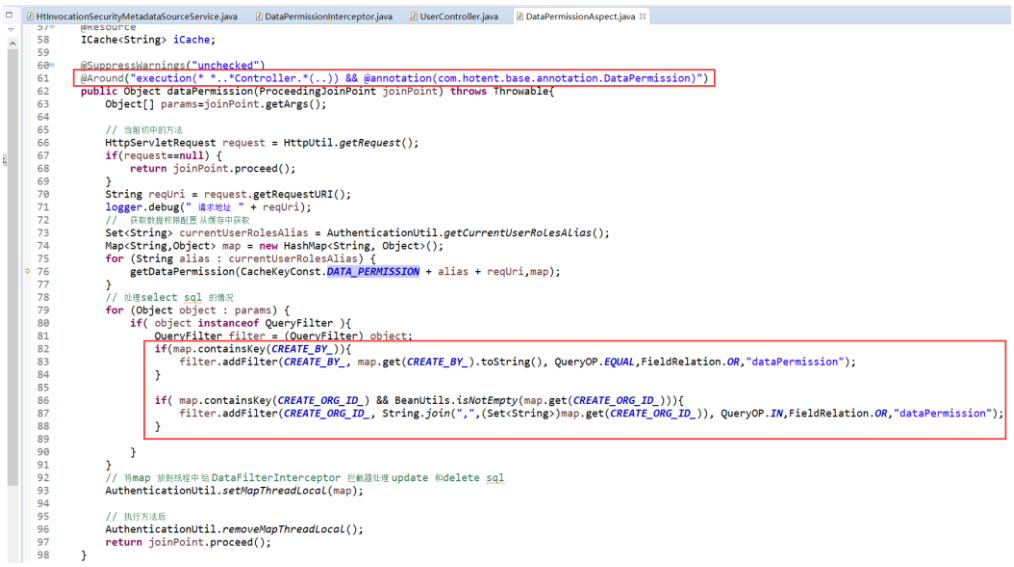
- **代码实现**

首先，如下图所示，系统会初始化角色的资源授权关系到缓存中。

HtInvocationSecurityMetadataSourceService.java | DataPermissionInterceptor.java

```
1  package com.hotent.base.security;
2
3* import java.util.ArrayList;
26
27 @Service
28 public class HtInvocationSecurityMetadataSourceService  implements
29         FilterInvocationSecurityMetadataSource {
30
31     // security认证对象的线程变量 Authentication
32     private static ThreadLocal<HashMap<String, Collection<ConfigAttribute>>> mapThreadLocal = new ThreadLocal<HashMap<String, Collecti
33
34*    @Resource
35     private ICache<ArrayList<HashMap<String,String>>> iCache;
36*    @Resource
37     private ICache<String> iCacheDataPermission;
38*    @Resource
39     private  MethodAuthService methodAuthService;
40
41
42*    /**
43      * 加载权限表中所有权限
44      *
45      * 同一个方法可能多个角色都具有访问权限
46      *
47      */
48*    public void loadResourceDefine(){
49         HashMap<String, Collection<ConfigAttribute>> map = getMapThreadLocal();
50         Collection<ConfigAttribute> array;
51         ConfigAttribute cfg;
52         ArrayList<HashMap<String, String>> methodAuth = iCache.getByKey(CacheKeyConst.SYS_METHOD_ROLE_AUTH);
53         if(BeanUtils.isEmpty(methodAuth)){
54             methodAuth = methodAuthService.getMethodAuth();
55         }
56         for(HashMap<String, String> mapAuth : methodAuth) {
57             array = new ArrayList<ConfigAttribute>();
58             String roleAlias = mapAuth.get("roleAlias");
59             String key = mapAuth.get("methodRequestUrl");
60             if(StringUtil.isEmpty(roleAlias) || StringUtil.isEmpty(key) ) {
61                 continue;
62             }
63             cfg = new SecurityConfig(roleAlias);
64
65             if( mapAuth.containsKey("dataPermission") && StringUtil.isNotEmpty(mapAuth.get("dataPermission")) ){
66                 iCacheDataPermission.add(CacheKeyConst.DATA_PERMISSION+roleAlias+key, mapAuth.get("dataPermission"));
67             }
68
69             if(map.containsKey(key)){
70                 array = map.get(key);
71             }
72             array.add(cfg);
73             map.put(key, array);
74         }
75
```

当用户访问某个 Controller 层的方法时，如果这个方法上添加了 DataPermission 注解，则会被 Aop 方法拦截，被拦截到的请求，如果传递了 QueryFilter 对象，则会在这个查询对象中根据数据过滤规则添加新的过滤条件，例如：添加 createBy 等于当前用户、createOrgId 等于当前用户所属的组织等。



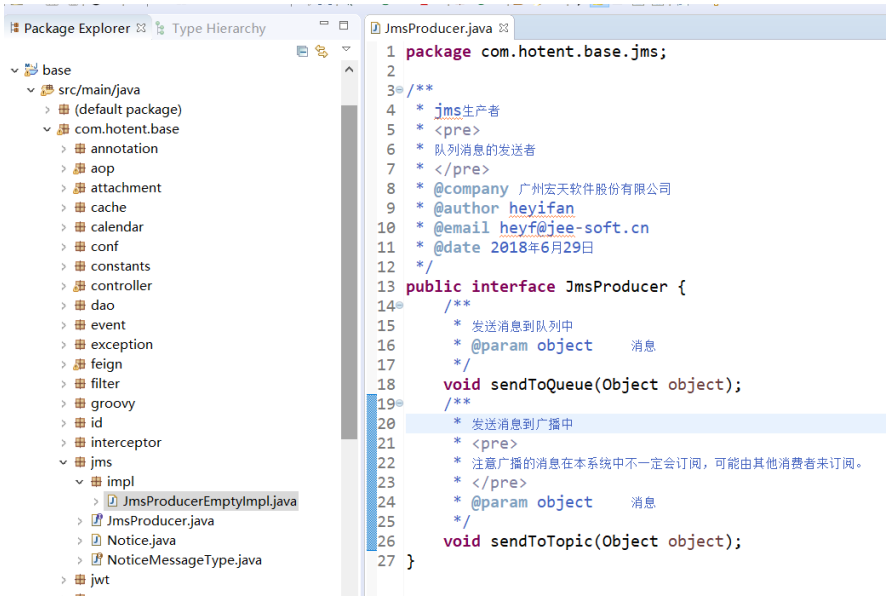
7.4 菜单管理和路由注册

7.5 消息队列

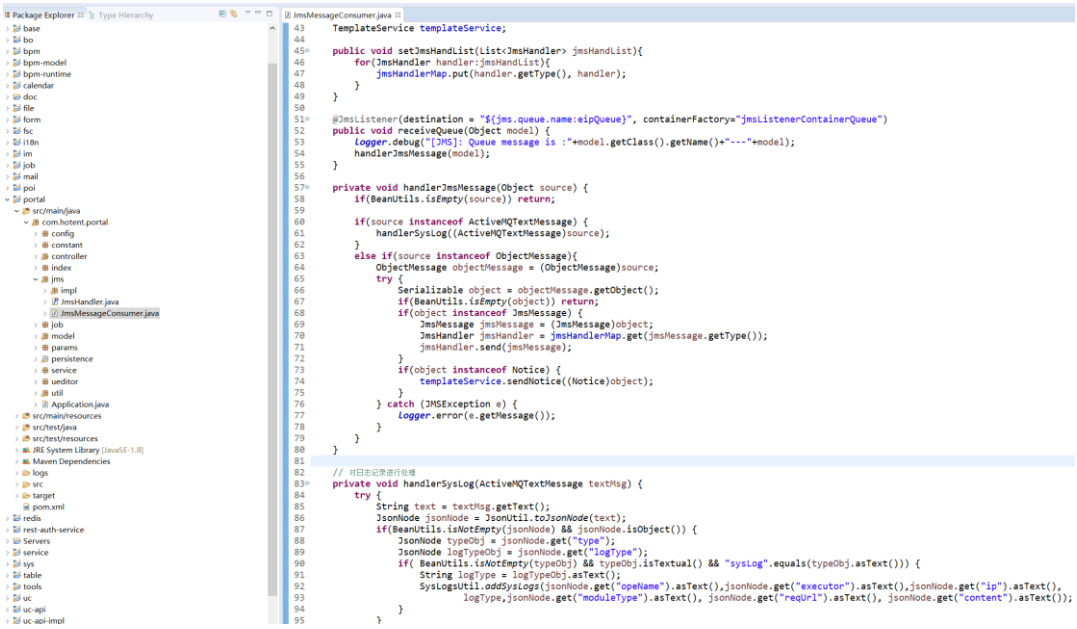
EIP7 中集成了消息队列，目前队列在 EIP7 中的用途主要有两种：一种是记录日志（包括操作日志和异常日志），另一种是流程中发送流程消息。

MQ 分为生产者和消费者，我们将生产者的接口定义在 base 模块中，如下图所示，这样在所有的微服务中都可以调用接口来发送消息到 MQ 中。

当然，前提条件是这个微服务引入了接口的实现类，EIP7 默认提供了 activemq 作为实现，也可以更换为其他的中间件作为实现类。为了设计上更加松耦合，这个 JmsProducer 的接口我们提供了一个空实现类 JmsProducerEmptyImpl，这样在微服务中不引入 activemq 这个类库包也可以正常运行，只不过消息没有正确的发送到 mq 队列而已。



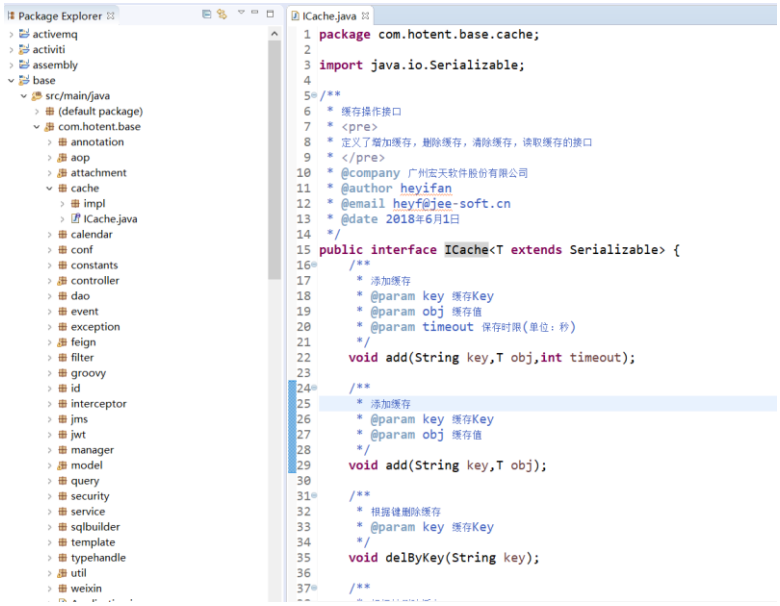
MQ 的消费者定义在 portal 这个微服务中，如下图所示，目前在消费者中针对上述两种消息分别进行处理。对于流程消息通过 JmsHandler 找到对应的处理器来处理（邮件、手机短信、微信消息），对于系统日志直接记录到系统日志表中。



7.6 内存数据库

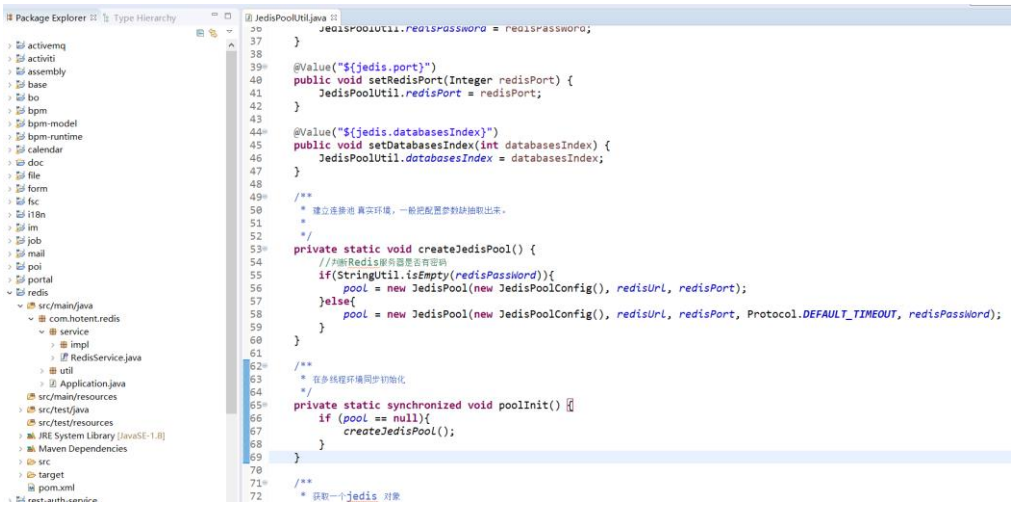
EIP7 中为了尽量提升系统的性能，将很多数据缓存到内存数据库中，不过这些数据都是热数据，即可以随时清理的数据（程序读不到内存数据库中的数据时会再次从数据库中加载数据并缓存起来）。

内存数据库的操作接口定义在 base 模块中，如下图所示，同样的，为了避免对 ICache 接口的实现类产生强依赖，我们提供了 MemoryCache 作为默认实现类，但是要注意 MemoryCache 以 Map 来存放数据，只在当前单个微服务中生效。



另外，为了实现对缓存数据的批量操作，我们继承 ICache 接口在 redis 这个模块中定义了 RedisService 这个接口，在 EIP7 中主要对国际化的资源进行批量操作。而基于 Redis 内存数据库作为中间件的实现类，是 EIP7 系统提供的默认实现，同样的，我们可以更改为其他的中间件作为实现类。

另外要注意，默认的 redis 连接为单节点的，在生产环境中，如要保证系统的高可用，需要将 redis 部署为集群模式（哨兵模式），那么对应的连接 redis 的代码需要做相应的修改，采用哨兵模式来连接 redis。



7.7 统一日志

系统统一记录了所有用户的操作的日志，方便管理员查看请求记录。

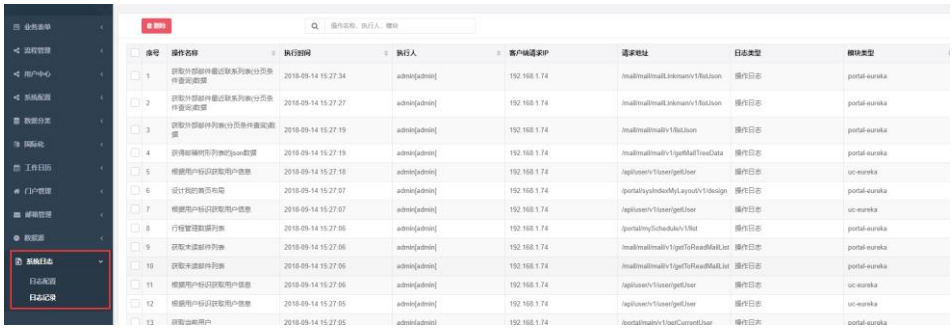
- **实现方案**

在系统中，添加了 AOP 切面 com.hotent.base.aop.SysLogsAspect 类， 拦截所有的请求，并且具有@ApiOperation 注解的请求，拦截到的请求在 sysLogs 方法中记录用户的操作日志。

```
@Aspect
@Component
public class SysLogsAspect {

    private Logger logger = LoggerFactory.getLogger(SysLogsAspect.class);

    @Resource
    PortalFeignService portalFeignService;
    @Resource
    ICache<HashMap<String,String>> iCache;

    private static String moduleType = "uc-eureka" ;

    @Value("${spring.application.name}")
    public void setDbType(String param){
        moduleType = param;
    }

    @Around("execution(* *..*Controller.*(..)) && @annotation(io.swagger.annotations.ApiOperation)")
    public Object sysLogs(ProceedingJoinPoint joinPoint) throws Throwable{
```

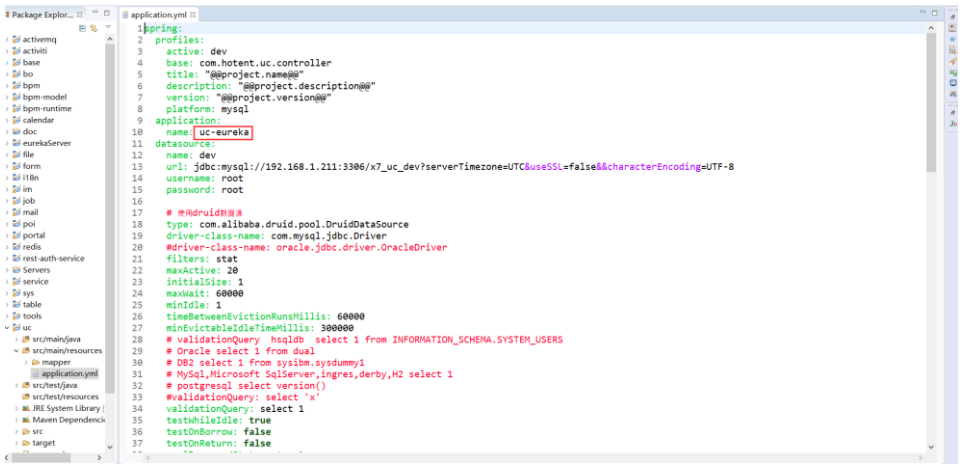
- **已有微服务**

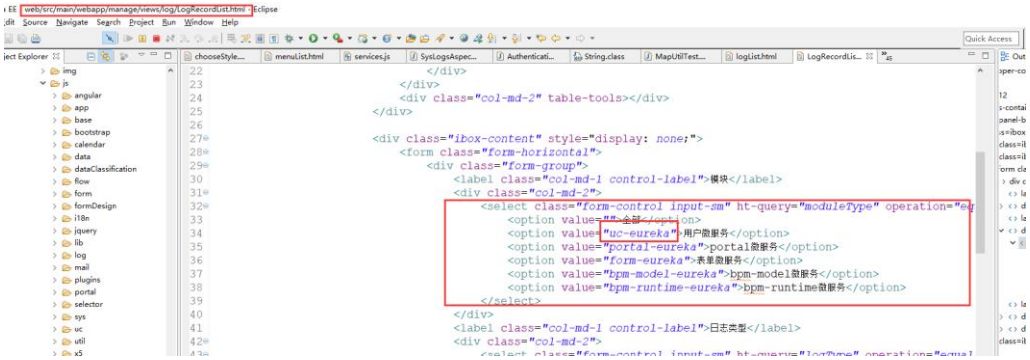
在 uc,portal,form,bpm-model,bpm-runtime 这五个微服务中， 只需要在 Controllers 层添加@ApiOperation 注解，就会将用户的操作记录日志到系统日志表 portal_sys_logs，在系统日志中可以查看如下图



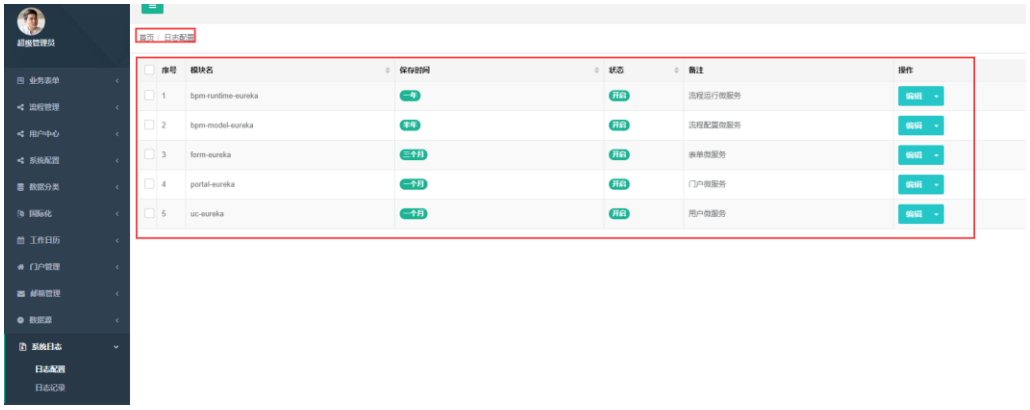
- **新加的微服务**

处理和已有微服务的处理方式一样（在 controllers 层代码添加@ApiOperation 注解）还需要修改 web\src\main\webapp\manage\views\log\LogRecordList.html 页面，在页面上添加搜索条件，该条件为新增的微服务的名称，用于分微服务来归类系统的操作日志。





修改完页面后需要在数据库表 portal_sys_logs_settings 插入一条数据，该表时用来配置日志信息的， 配置日志是否开始记录日志功能，日志保存时间。



如： INSERT INTO portal_sys_logs_settings (ID_, MODULE_TYPE_, STATUS_, SAVE_DAYS_, REMARK_) VALUES ('5', 'bpm-runtime-eureka', '1', '365', '流程运行微服务')。

bpm-runtime-eureka （MODULE_TYPE_）为新的微服务的名称

spring.application.name

- **添加定时任务删除系统操作日志**

系统运行时间长了，系统日志表的数据将会越来越多，这张表的查询也将会

变得越来越慢，这种情况需要定时地去清理一些以前的操作日志。在系统中，是通过配置定时任务来删除系统操作日志的。

- **添加定时任务**

  com.hotent.portal.job.SysLogsJob

- **添加计划任务**

可以添加每个一周清理一次操作日志，会根据系统的存留时间来清楚系统日志。

7.8 国际化

7.8.1 国际化实现方案

前端页面，js 文件是使用 angular 实现国际化方案。

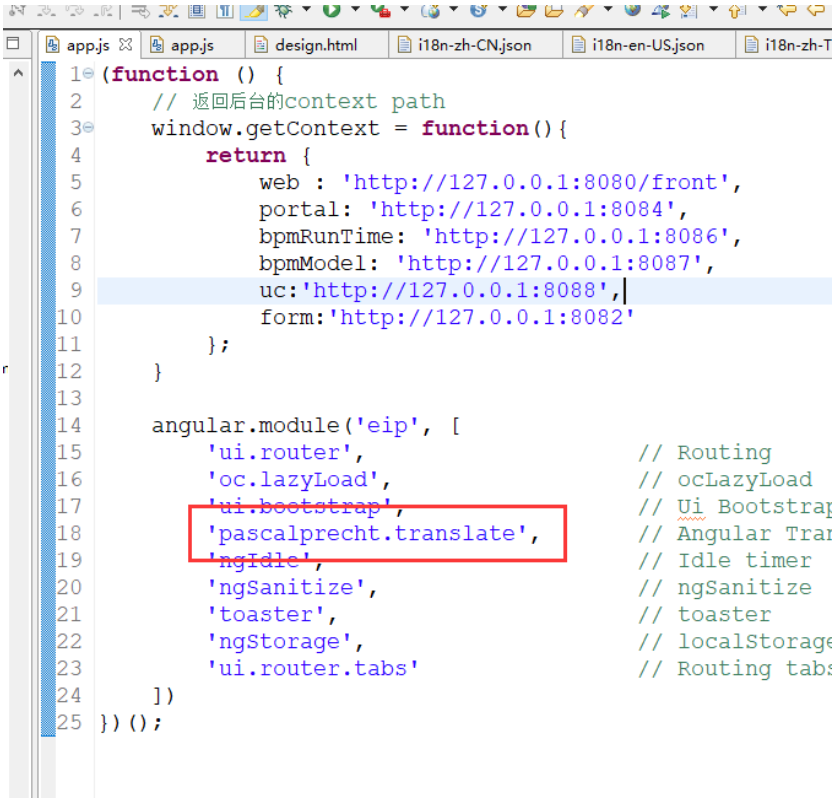
- **引入相关文件**

需要在首页 index.html 页面引入 angular-translate.min.js, angular-translate-loader-static-files.min.js,如下图

```
<!-- moment -->
<script src="js/plugins/moment/moment.min.js"></script>

<!-- Main Angular scripts-->
<script src="js/angular/angular.min.js"></script>
<script src="js/angular/angular-sanitize.min.js"></script>
<script src="js/angular/angular-translate.min.js"></script>
<script src="js/angular/angular-translate-loader-static-files.min.js"></script>
<script src="js/angular/angular-ui-router.min.js"></script>
<script src="js/angular/angular-ui-router-tabs.js"></script>
<script src="js/angular/ngStorage.min.js"></script>
<script src="js/bootstrap/ui-bootstrap-tpls-1.3.3.min.js"></script>
<script src="js/plugins/oclazyload/ocLazyLoad.min.js"></script>
<script src="js/plugins/angular-idle/angular-idle.js"></script>
<script src="js/plugins/toastr/toastr.min.js"></script>
```

- **注入 translate 模块**

在 app.js 文件中的 eip module 中注入 pascalprecht.translate 模块，如下图：



- **在 config.js 中配置**

在 config.js 中的 config 方法添加使用如下方法， 需要在 config 方法中注入 $translateProvider

```
// 国际化资源
var lang = "i18n-zh-CN";
if (window.localStorage.getItem("lang")) {
    lang = window.localStorage.getItem("lang");
}
// 设置默认语言
$translateProvider.fallbackLanguage(lang);
//$translateProvider.preferredLanguage(lang);
$translateProvider.useStaticFilesLoader({
    prefix: 'js/i18n/',
    suffix: '.json'
});
```

- **在 controllers.js 中的 MainCtrl 设置切换语言的方法**

目前是在 front 实现了国际化， 和语言的切换

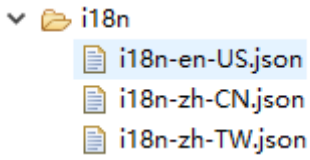
需要在 controllers.js 中的 MainCtrl 方法添加国际化的处理。去切换语言

```
112
113     baseService.post('${portal}/i18n/custom/i18nMessageType/v1/all').then(function(rep){
114        $scope.i18nMessageType = rep;
115     });                                  获取国际化语种类型 如 zh-CN, en-US
116
117     $scope.changeLanguage = function(language){
118         $translate.use("i18n-"+language);   使用何种语言 , 如 zh-CN
119         window.localStorage.setItem("lang", "i18n-"+language);
120         $http.defaults.headers.common["Accept-Language"] = language;
121         //  delete $sessionStorage.frontMenus; 无效
122         window.sessionStorage.removeItem("ngStorage-frontMenus");
123         // 刷新当前页面
124         window.location.reload();
125     }
126
```

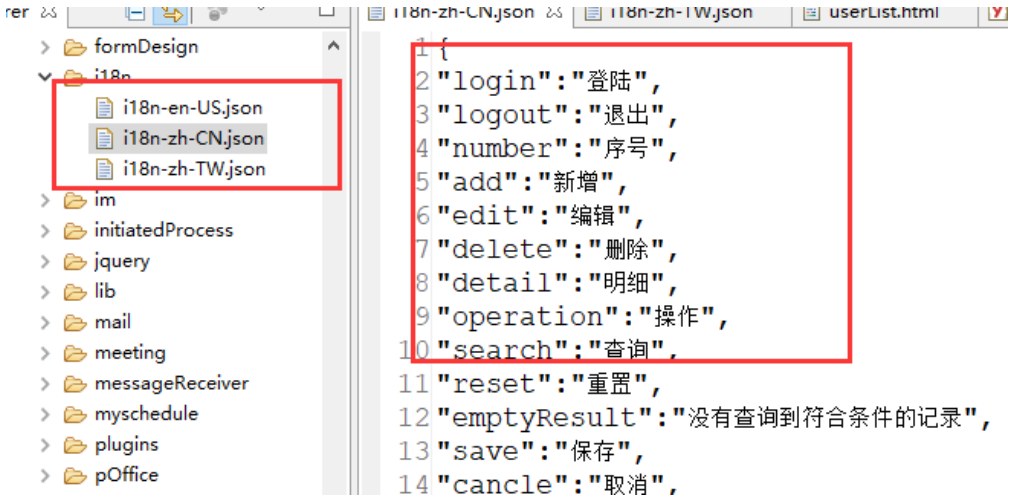
需要注意的是，切换语言使用$translate 的 use 方法说明使用哪个国际化的资源文件。 如 使用 i18n-zh-CN 文件 ， 改文件的路径是在 config.js 中配置的

```
//$translateProvider.preferredLanguage(lang
$translateProvider.useStaticFilesLoader({
    prefix: 'js/i18n/',
    suffix: '.json'
});
```

如当切换的语言问中文（zh-CN） ，则 js，html 的国际化资源将会获取 js/i18n/i18n-zh-CN.json 文件翻译系统



7.8.2 静态资源国际化维护



系统内置了三种语言的国际化， 简体中文（zh-CN），繁体中文(zh-TW)， 英文(en-US),国际化资源都是 json 结构的。

如果需要再添加一种语言，则需要在 国际化》语种管理 添加一条语种

如添加韩文（ko-KR） 然后在 js/i18n 文件夹中添加 i18n-ko-KR.json 文件，并且把 i18n-zh-CN.json 的内容复制到 i18n-ko-KR.json 文件中，然后 key 不变，修改 value 值为对应的韩文。

7.8.3 Html 页面如何使用国际化资源

在 html 页面中可以通过如下实现

{{ key | translate }} key 为国际化资源 json 文件的对应的 key 值

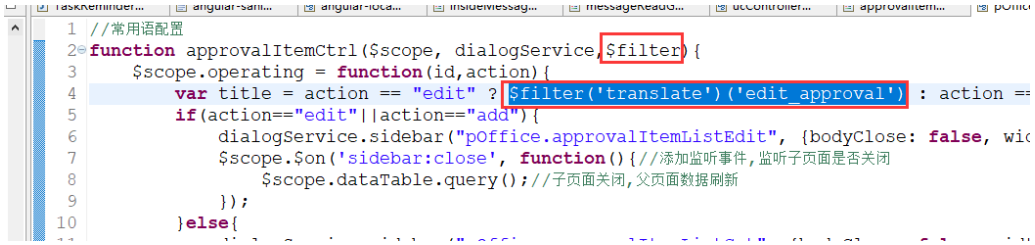
如：{{ 'login' | translate }} {{ 'logout' | translate }}

```
<li><a ui-sref="login"><i class="fa fa-sign-out"></i> {{'logout' | translate}}</a></li>

<button type="submit" class="btn btn-primary block full-width m-b"
        ng-disabled="vm.loading">{{'login' | translate}}</button>
```

可以参考 \web\src\main\webapp\front\views\common\chooseStyle.html 页面，切换皮肤功能页面

7.8.4 Js 如何使用国际化

在 js 方法中注入$filter,使用$filter 获取国际化资源如



```
//常用语配置
function approvalItemCtrl($scope, dialogService,$filter){
    $scope.operating = function(id,action){
        var title = action == "edit" ? $filter('translate')('edit_approval') : action ==
        if(action=="edit"||action=="add"){
            dialogService.sidebar("pOffice.approvalItemListEdit", {bodyClose: false, wi
            $scope.$on('sidebar:close', function(){//添加监听事件,监听子页面是否关闭
                $scope.dataTable.query();//子页面关闭,父页面数据刷新
            });
        }else{
```

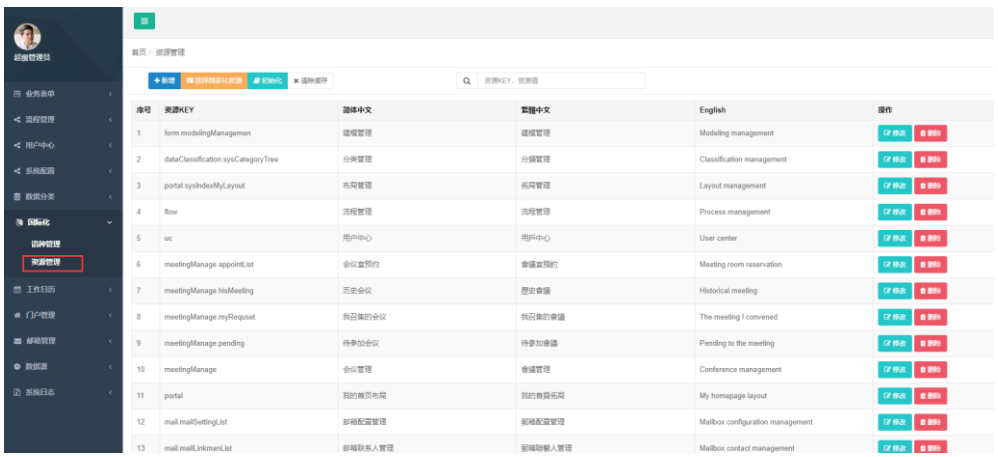
`$filter('translate')('edit_approval')`

使用 angular 的 filter 获取国际化资源。

7.8.5 Java 代码使用国际化

Java 代码使用的国际化资源时在资源管理中维护的，这些数据在数据库中有一份数据， 使用的数据全部从 redis 缓存中获取， 后台的代码返回的国际化数据通过 I18nUtil 类来获取国际化资源

```
226
227     //  菜单资源国际化
228     private List<SysMenu> i18nSysMenu(List<SysMenu> lists) {
229
230         List<String> i18nKey = new ArrayList<String>();
231
232         for (SysMenu sysMenu : lists) {
233             i18nKey.add(sysMenu.getAlias());
234         }
235         Map<String, String> messages = I18nUtil.getMessages(i18nKey, getLocale());
236         for (SysMenu sysMenu : lists) {
237             String key = sysMenu.getAlias();
238             if(messages.containsKey(key) && StringUtil.isNotEmpty(messages.get(key)) ){
239                 sysMenu.setName(messages.get(key));
240             }
241         }
242         return lists;
243     }
```



I18nUtil.getMessages(i18nKey, getLocale()); i18nkey，是要获取的国际化资源的可以，对应资源管理中的 资源 key, getLocale()获取当前语言。在前端切换语言时，会将语言存储到 localStroage 中，在 config.js 中的 run 方法中添加如下代码，用于设置当前请求接受何种语言。getLocale()方法将会获取语种。

```
}
// 默認支持語言
if(window.localStorage.getItem("lang")){
    $rootScope.lang = window.localStorage.getItem("lang").replace("i18n-","");
    $http.defaults.headers.common["Accept-Language"] = $rootScope.lang;
}
```

如果其他模块没有 I18nUtil 工具类，需要引入 i18n 模块

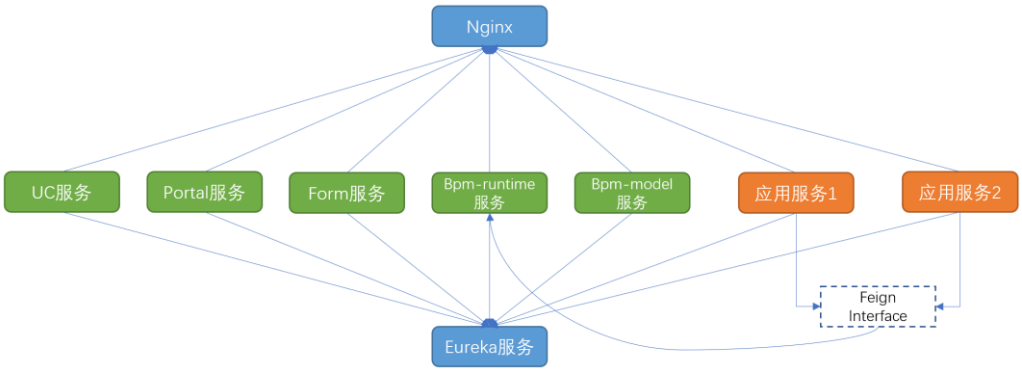
```
        </dependency>
        <!-- 其他模块使用I18nUtil的方法,获取国际化资源 -->
        <dependency>
            <groupId>com.hotent</groupId>
            <artifactId>i18n</artifactId>
            <version>${project.version}</version>
            <exclusions>
                <exclusion>
                    <groupId>com.hotent</groupId>
                    <artifactId>uc-api-impl</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
```

8 案例分析

8.1 微服务流程中心

如下图所示，EIP 作为微服务架构体系，可以采用微服务的方式来构建流程中心。流程微服务注册到 Eureka 中，各个应用服务通过 Feign Interface 来调用流程微服务。

这里的 Feign Interface 只需要定义接口方法，并声明接口要调用的微服务的名称，在实际调用时就会自动通过 Eureka 寻找可用的微服务节点并完成调用。而且 Feign 会自动继承认证信息，同步接口调用异常，避免分布式事务问题。



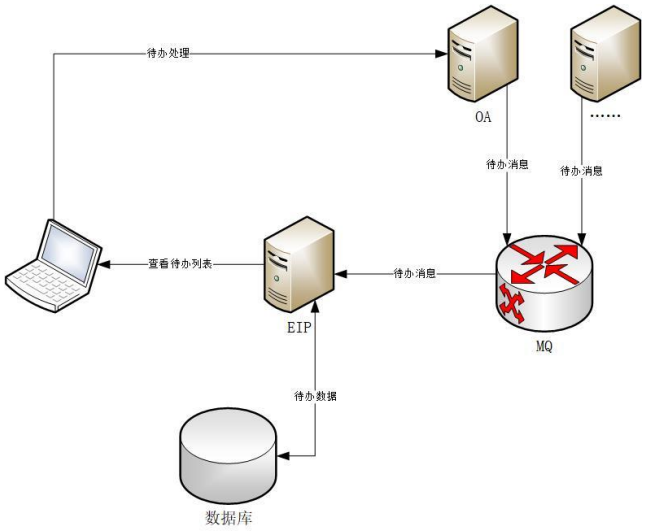
示例代码如下图所示，只需要定义这份 Feign Interface 在应用服务中，需要在业务逻辑中启动流程时，注入这个 Interface 然后调用 start 方法即可完成流程的启动。

BpmRuntimeFeignService.java

```
package com.hotent.base.feign;

import java.util.List;

/**
 * 流程运行Restful接口访问的代理类
 *
 * @company 广州宏天软件股份有限公司
 * @author heyifan
 * @email heyf@jee-soft.cn
 * @date 2018年8月21日
 */
@FeignClient(name="bpm-runtime-eureka",fallback=BpmRuntimeFeignServiceImpl.class, configuration=FeignConfig.class)
public interface BpmRuntimeFeignService {

    @RequestMapping(value = "/runtime/instance/v1/start", method = RequestMethod.POST)
    public ObjectNode start(@RequestBody ObjectNode startFlowParamObject) throws Exception;

    @RequestMapping(value = "/runtime/instance/v1/getByBusinesKey", method = RequestMethod.GET)
    public ObjectNode getByBusinesKey(@RequestParam(value="businessKey", required=true)String businessKey,@RequestParam(valu

    @RequestMapping(value = "/runtime/instance/v1/getDataByDefId", method = RequestMethod.GET)
    public List<ObjectNode> getDataByDefId(@RequestParam(value="defId", required=true) String defId) throws Exception;

    @RequestMapping(value = "/runtime/instance/v1/getDataByInst", method = RequestMethod.GET)
    public List<ObjectNode> getDataByInst(@RequestParam(value="instId", required=true) String instId) throws Exception;

    @RequestMapping(value = "/runtime/instance/v1/isSynchronize", method = RequestMethod.GET)
    public Boolean isSynchronize(@RequestParam(value="instId", required=true) String instId,@RequestParam(value="nodeIds", r
                                 @RequestParam(value="status", required=true) String status,@RequestParam(value="lastStatus"
                                 @RequestParam(value="lastNodeIds", required=true) String lastNodeIds) throws Exception;
}
```

8.2 统一待办门户

一个企业中，可能有多个系统都会产生待办任务，用户需要在各个系统中进行待办处理，操作繁琐、而且容易遗漏。通过构建统一的待办门户可以解决这个问题，如下图所示，将各个系统的待办任务通过 MQ 统一到一起展示。



各个业务系统产生待办、完成待办时，通过 MQ 将待办消息发送给 EIP 系统，EIP 会根据消息类型在数据库中对待办数据进行保存或删除等操作。用户在通过 EIP 的待办门户查询待办时，查看到的待办任务就是来自所有系统的待办任务，当他需要对某个待办进行查看或处理时，会跳转到对应的系统完成待办处理。

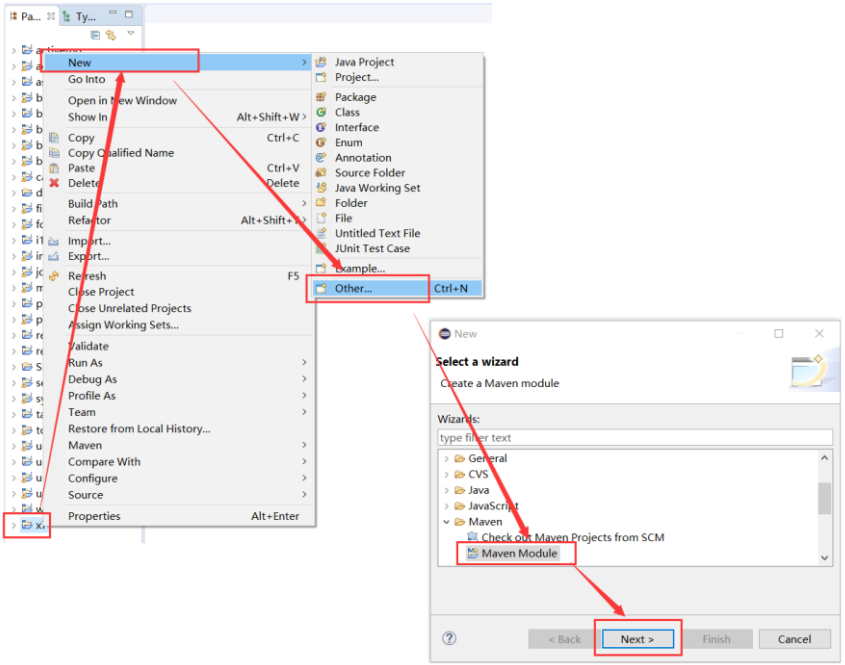
9 应用定制

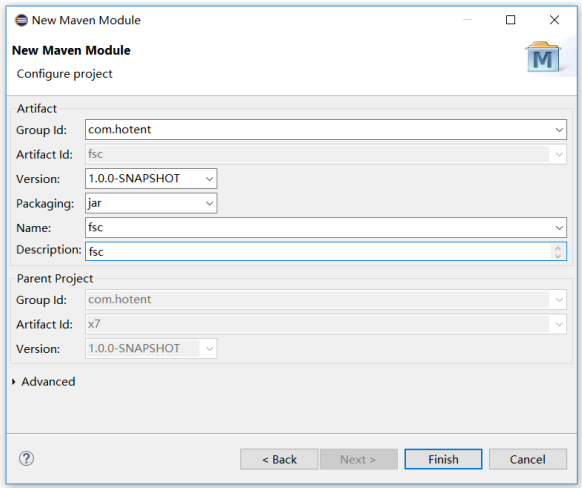
根据项目需求，可能需要将一些功能打包在一起，以一个独立的应用来发布，所以在这个章节，我们会介绍如何进行应用的定制开发。

9.1 定制微服务应用

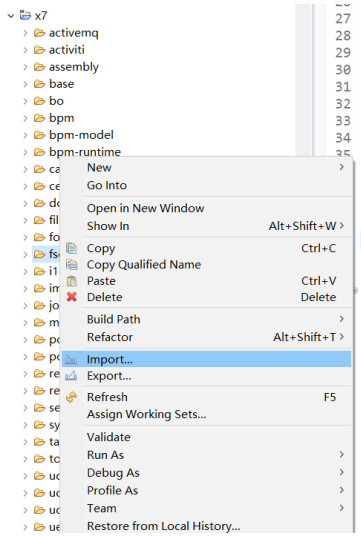
9.1.1 创建微服务

EIP7 自带的微服务为五个，但是以系统的类库包为基础，我们可以开发出第六个、第七个微服务作为业务应用。如下图所示，在项目根目录上右键选在新建一个 Maven Module，项目会自动作为 EIP7 的一个子模块创建。





点击 Finish 按钮后，新创建的 maven 模块会自动引入到 Package Explorer 中，如果没有自动引入，也可以在根目录中找到 fsc 目录，右键菜单中点击 import 完成引入。



9.1.2 Maven 依赖

新添加的 Maven 模块需要以 EIP7 的子模块定义在 pom.xml 的 modules 列表中，所以如下图所示，确认一下已经引入 fsc 这个模块，注意：maven 会按照 modules 中自上至下的顺序依次编译，所以确保 fsc 依赖的其他 maven 模块生命在 fsc 之上。

x7/pom.xml

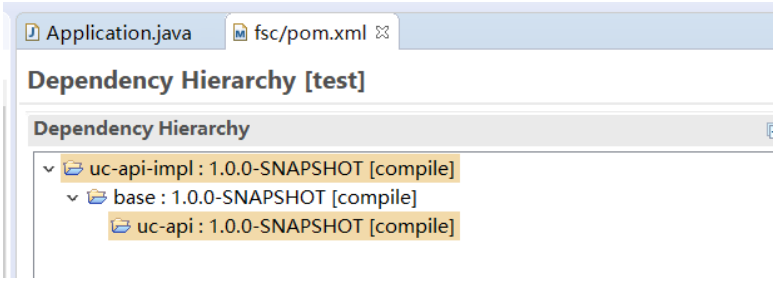
```
    <modules>
        <module>uc-api</module>
        <module>base</module>
        <module>uc-api-impl</module>
        <module>bo</module>
        <module>table</module>
        <module>poi</module>
        <module>job</module>
        <module>mail</module>
        <module>im</module>
        <module>service</module>
        <module>portal</module>
        <module>redis</module>
        <module>activemq</module>
        <module>i18n</module>
        <module>file</module>
        <module>form</module>
        <module>bpm</module>
        <module>bpm-model</module>
        <module>bpm-runtime</module>
        <module>uc</module>
        <module>activiti</module>
        <module>calendar</module>
        <module>sys</module>
        <module>rest-auth-service</module>
        <module>ueditor</module>
        <module>fsc</module>
<!--        <module>assembly</module> -->
    </modules>
```

对于 fsc 模块，其 pom.xml 文件如下图所示，依赖配置中，spring-boot-starter-test 为单元测试的依赖包。

Application.java | fsc/pom.xml

```
<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/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>com.hotent</groupId>
        <artifactId>x7</artifactId>
        <version>1.0.0-SNAPSHOT</version>
    </parent>
    <artifactId>fsc</artifactId>
    <name>fsc</name>
    <description>fsc</description>

    <dependencies>
        <!-- Spring boot 测试 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.hotent</groupId>
            <artifactId>uc-api-impl</artifactId>
            <version>${parent.version}</version>
        </dependency>
    </dependencies>

    <!-- 用于生成jar文件 -->
    <build>
        <finalName>fsc</finalName>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <mainClass>com.hotent.fsc.Application</mainClass>
                    <layout>JAR</layout>
                </configuration>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

uc-api-impl 为 eip7 的基础依赖包，其依赖关系如下图所示：



1. uc-api：这个包中定义了基础的用户、用户组接口模型，以及获取用户、用户组的相关接口；
2. base：封装了登录认证、数据库连接、基础的 CRUD 方法、缓存操作等等（详情请查看 API 简介章节）；
3. uc-api-impl：uc-api 的默认实现包，会通过 feign 调用微服务 uc 的相应接口来获取用户、用户组数据。

另外 build 命令中集成了 spring-boot-maven-plugin 插件，该插件提供微服务打包为可执行 jar 的作用，并定义了微服务应用的唯一入口，便于我们在开发工具中启动应用并进行开发及调试。

以上配置为微服务应用的基础 maven 配置，其余的依赖包或 build 插件根据模块的需求相应的添加。

com.hotent.fsc.Application 为微服务的唯一入口，其代码如下所示：

fsc/pom.xml | Application.java

```
1  package com.hotent.fsc;
2
3  import org.mybatis.spring.annotation.MapperScan;
4  import org.springframework.boot.SpringApplication;
5  import org.springframework.cloud.client.SpringCloudApplication;
6  import org.springframework.cloud.openfeign.EnableFeignClients;
7  import org.springframework.context.annotation.ComponentScan;
8  import org.springframework.context.annotation.Configuration;
9
10 @SpringCloudApplication
11 @Configuration
12 @MapperScan(basePackages={"com.hotent.**.dao"})
13 @ComponentScan({"com.hotent.*"})
14 @EnableFeignClients(basePackages = {"com.hotent.*"})
15 public class Application {
16     public static void main( String[] args )
17     {
18         SpringApplication.run(Application.class, args);
19     }
20 }
21
```

在这个类上面有 5 个注解，起作用分别如下：

1. SpringCloudApplication：表示这个应用是一个 Spring Cloud 应用；
2. Configuration：表示该类是一个配置类；
3. MapperScan：因为系统使用 MyBatis 作为框架，所以需要扫描 mapper 文

件，该配置指定扫描哪些 dao 所对应的 mapper 文件，如果应用中开发的 dao 存放在其他的 namespace 下，则相应的添加到这里，多个 namespace 以逗号分割；

4. ComponentScan：需要扫描到 spring 容器中的 java bean，同样的有不同 namespace 下的类需要扫描时，均需要声明在这里；

5. EnableFeignClients：表示系统是一个 feign 的客户端，需要扫描装配所有的 FeignClient 注解修饰的接口类。

9.1.3 配置文件修改

微服务需要定义一个名为 application.yml 的文件作为配置文件，最终该配置文件会和 base 模块中定义的 application-dev.yml 合并为一个文件作为该微服务的配置文件。当这两份配置中有相同的配置项时，以 application-dev.yml 中的为准。

base 模块中的 application-dev.yml 中定义了一些通用配置项及中间件的连接信息。

各个微服务模块中的 application.yml 中则定义了该微服务的独有的配置项，其配置如下图所示，注意红框标识出来的内容，需要根据不同的微服务修改这些配置（每一个配置项的作用请查看章节配置文件来了解详情）。

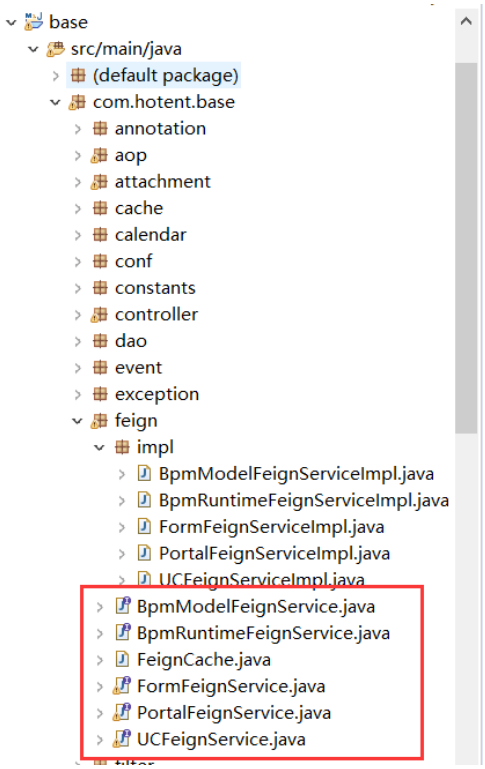
```yaml
spring:
  profiles:
    active: dev
    platform: mysql
    base: com.hotent.fsc.controller
  application:
    name: fsc-eureka
  datasource:
    name: dev
    url: jdbc:mysql://192.168.1.211:3306/X7_fsc_dev?serverTimezone=UTC&useSSL=false&&characterEncoding=UTF-8
    username: root
    password: root
    # 使用druid数据源
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.jdbc.Driver
    filters: stat
    maxActive: 20
    initialSize: 1
    maxWait: 60000
    minIdle: 1
    timeBetweenEvictionRunsMillis: 60000
    minEvictableIdleTimeMillis: 300000
    validationQuery: select 1 from dual
    testWhileIdle: true
    testOnBorrow: false
    testOnReturn: false
    poolPreparedStatements: true
    maxOpenPreparedStatements: 20
    maxPoolPreparedStatementPerConnectionSize: 20
    connectionProperties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000

# Server settings (ServerProperties)
server:
  port: 8083
  address: 0.0.0.0
  sessionTimeout: 30
  contextPath: /
  undertow:
    io-threads: 2
    worker-threads: 30
  session:
    timeout: 30
  compression:
    enabled: true
    mime-types: 'text/html,text/xml,text/plain,text/css,text/javascript,application/javascript,application/json'
    min-response-size: 1024

system:
  id:
    machineName: fsc

eureka:
  client:
    healthcheck:
      enabled:  true
    service-url:
      defaultZone: http://127.0.0.1:8761/eureka/
  instance:
    lease-expiration-duration-in-seconds: 30
    lease-renewal-interval-in-seconds: 10

# feign配置
feign:
  hystrix:
    enabled: true
  httpclient:
    enabled: true
  client:
    config:
      default:
        connectTimeout: 15000
        readTimeout: 30000
        loggerLevel: full

# 断路器配置
hystrix:
  metrics:
    enabled: true
  command:
    default:
      execution:
        isolation:
          strategy: SEMAPHORE   # THREAD  SEMAPHORE
          semaphore:
            maxConcurrentRequests: 200 # 默认情况下下面两个值都是10，也就是超过10个的并发会直接进入fallback方法，不会去真正请求
          thread:
            timeoutInMilliseconds: 60000 #缺省为1000
      fallback:
        isolation:
          semaphore:
            maxConcurrentRequests: 200   # 默认情况下下面两个值都是10，也就是超过10个的并发会直接进入fallback方法，不会去真正请求

# 负载均衡配置
ribbon:
  eureka:
    enabled: true
  connectTimeout: 15000
  readTimeout: 30000
  MaxAutoRetries: 0 #同一台实例最大重试次数，不包括首次调用
  MaxAutoRetriesNextServer: 1 #重试负载均衡其他实例的最大重试次数，不包括首次调用
  OkToRetryOnAllOperations: false #是否所有操作都重试
```

9.1.4 微服务之间的调用

微服务之间的相互调用通过 eureka 实现服务的自动发现与注册，通过 feign 来实现微服务间的接口调用。Feign 的接口定义在 base 模块中，如下图所示，以每个微服务定义一个接口，将微服务中可能被其他服务调用的接口声明在这个接口中，而且在 base 中定义了对于这个接口的默认实现（默认实现中直接抛出接

口调用异常，因为正常情况下，feign 会通过代理类调用到对应的微服务的接口，如果没有正常调用到就会调用到这个默认实现类）。

新开发的微服务，如果也有接口要给其他微服务调用，可以同样的在 base 中定义一个 feign 的接口。



9.2 应用后端开发

应用的后端开发，可以参考快速开始章节中的办公用品库存管理的开发样例。只不过生成出来的代码就放置在 fsc 这个新的微服务中了。

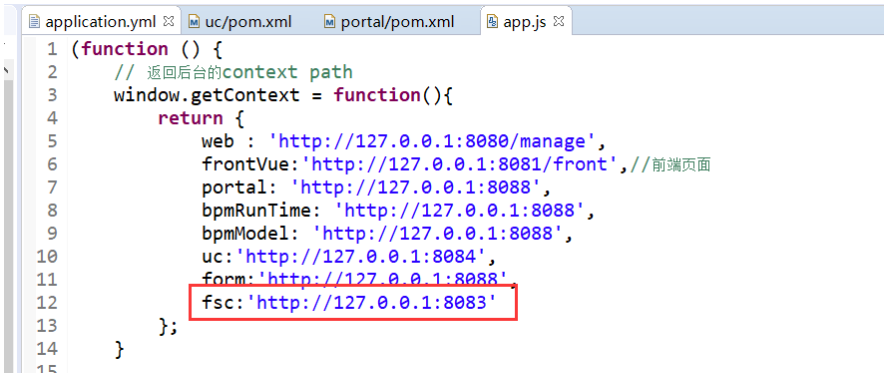
9.3 应用前端开发

应用的前端开发分为管理端和应用端。

9.3.1 管理端

管理端的技术栈是 AngularJs，其开发过程可以参照快速开始章节的办公用品库存管理的开发样例，唯一不同的地方是：样例中新开发的功能放置在 portal

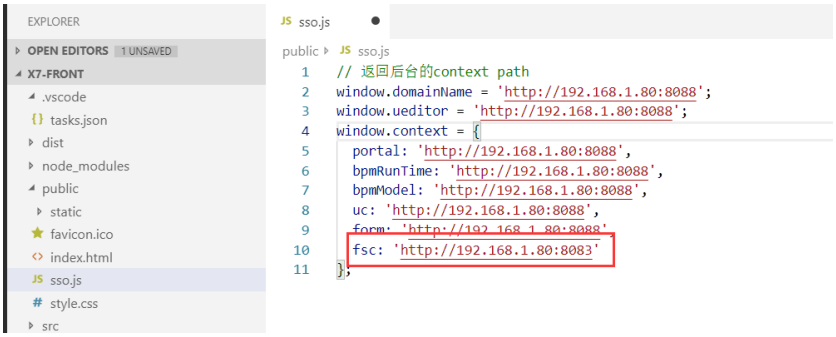
这个微服务中。这里生成出来的代码在 fsc 这个新的微服务中，所以需要修改前端调用后端的地址。如下图所示，在 app.js 中添加一个新的微服务的地址，而且在相应的前端 html 页面和 js 文件中，修改请求后台的 url 地址前缀为这个 fsc 的地址。



9.3.2 应用端

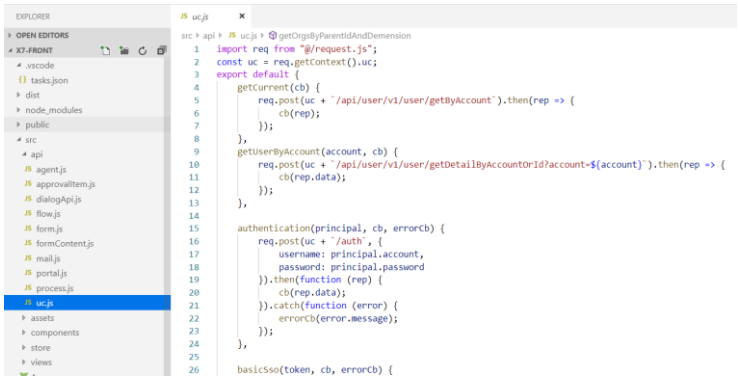
应用端的技术栈为 Vue，在 Vue 中要构建一个功能模块的增删改查页面，需要按照以下过程进行：

首先在如下图所示的 js 文件中声明一个新的后端微服务地址：



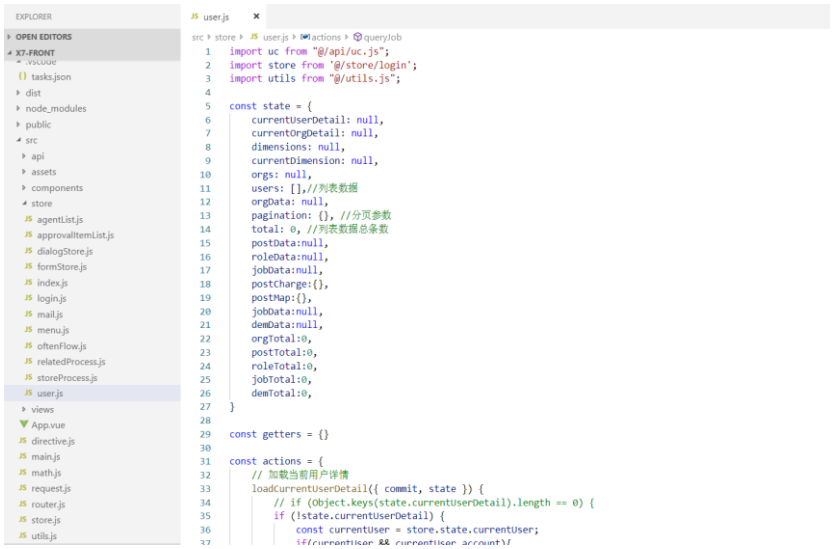
- **开发调用后端接口的 api**

如下图所示，仿照 uc.js 在 api 目录中添加一个新的 js 文件，将需要跟后台交互的接口定义在这个 js 文件中。

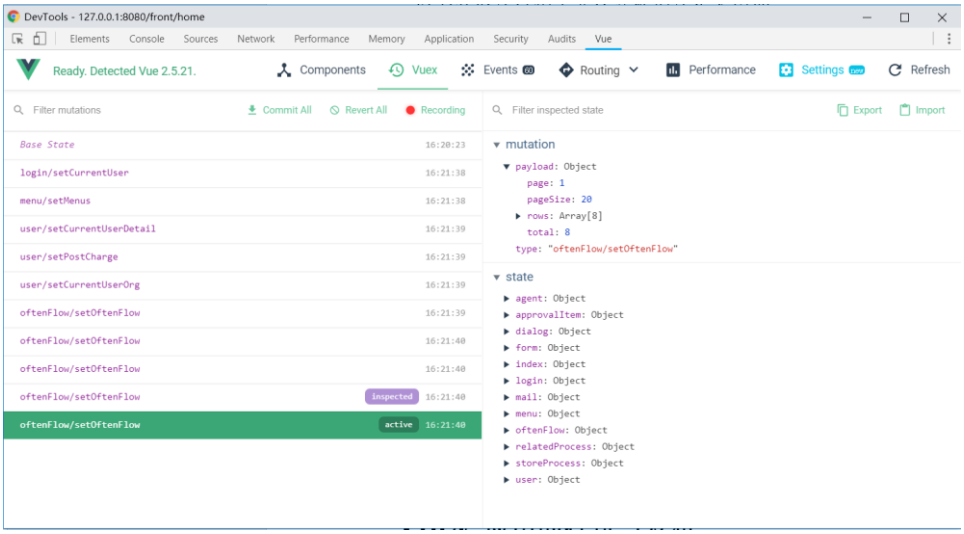


- **开发访问 api 的 vuex**

系统使用 vuex 来管理数据，如下图所示，仿照 user.js 开发 fsc 的 vuex，在 vuex 中通过调用上一步定义的 api 来与后台进行数据交互。注意 vuex 中的数据具有全局唯一性，而且在页面跳转时，vuex 中的数据会保存自己的当前状态，所以在开发的过程中要注意数据的生命周期。

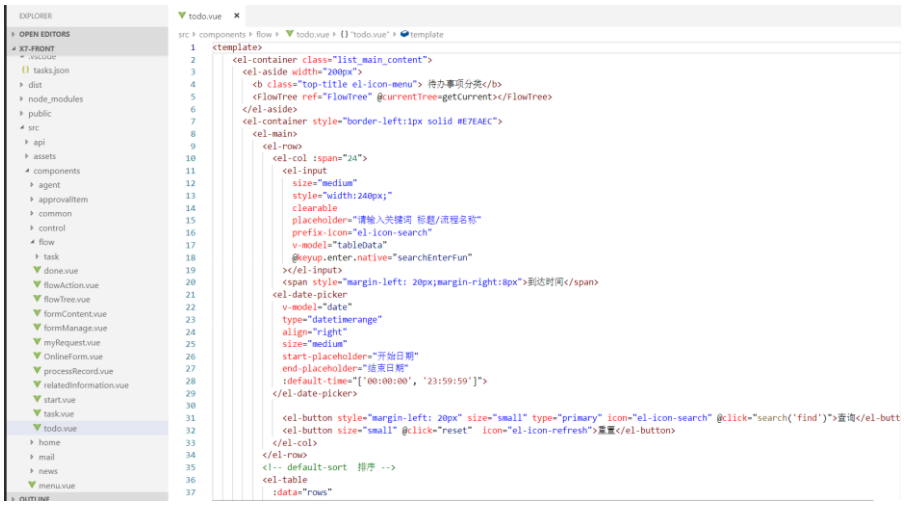


在开发调试过程中，建议安装 Vue 的 Chrome 插件，可以很方便直接的查看 vuex 中的数据。



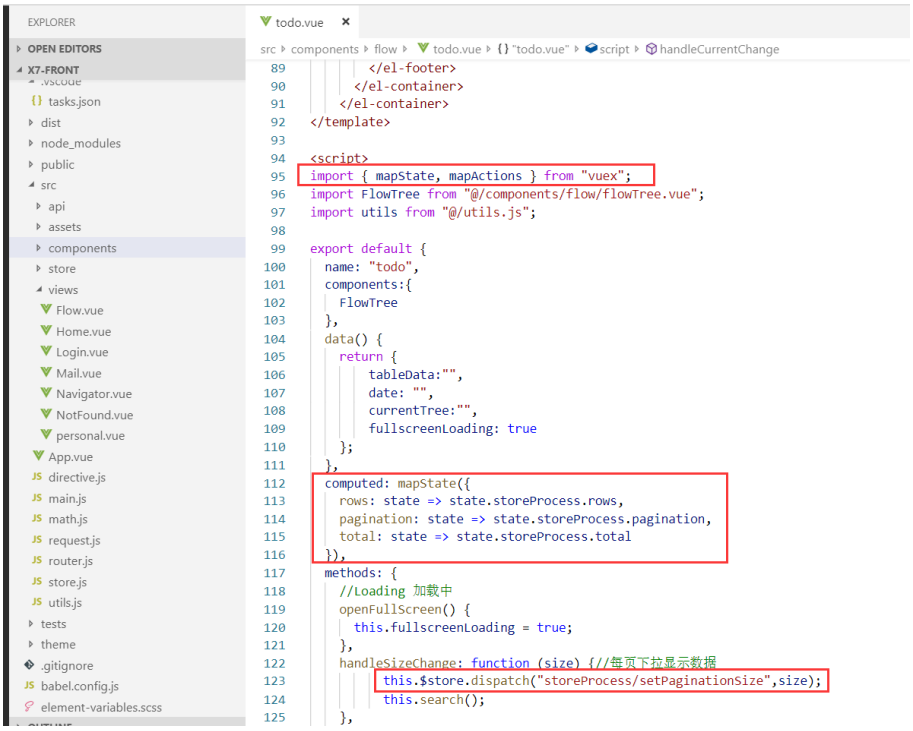
- **开发列表页面、编辑页面、详情页面**

Vue 的开发中很好的遵循的组件复用的原则，所以在开发各个功能模块时，对于可复用的组件或页面，我们可以定义在 components 目录中，在其他功能模块中可以复用这些组件或页面。而具体的列表页面、编辑页面、详情页面可以定义在 Views 目录下，如下图所示，为待办任务的列表页面。



- **挂载 vuex 到页面中**

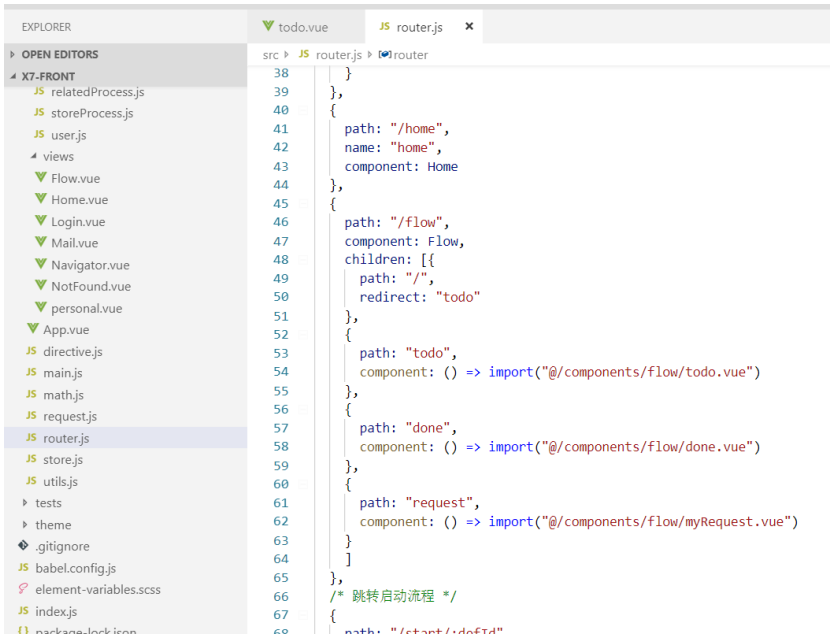
在页面中要挂载 Vuex 的数据，按照如下图所示的方式完成，首先从 vuex 中 import 出 mapState 和 mapActions。然后在 computed 中声明数据来自哪些名称空间下的哪些属性，当页面加载或者用户执行了某个操作时，通过 store 的 dispatch 命令来触发 Vuex 中的一个 mutations，这样会触发 Vuex 中的数据更新，同样的在页面上就会自动更新相应的数据（Vue 的开发核心理念就是这种单向流的数据状态管理模式）。



```
89            </el-footer>
90          </el-container>
91        </el-container>
92    </template>
93
94    <script>
95    import { mapState, mapActions } from "vuex";
96    import FlowTree from "@/components/flow/flowTree.vue";
97    import utils from "@/utils.js";
98
99    export default {
100     name: "todo",
101     components:{
102       FlowTree
103     },
104     data() {
105       return {
106           tableData:"",
107           date: "",
108           currentTree:"",
109           fullscreenLoading: true
110       };
111     },
112     computed: mapState({
113       rows: state => state.storeProcess.rows,
114       pagination: state => state.storeProcess.pagination,
115       total: state => state.storeProcess.total
116     }),
117     methods: {
118       //Loading 加载中
119       openFullScreen() {
120         this.fullscreenLoading = true;
121       },
122       handleSizeChange: function (size) {//每页下拉显示数据
123           this.$store.dispatch("storeProcess/setPaginationSize",size);
124           this.search();
125       },
```

- **注册页面到路由中**

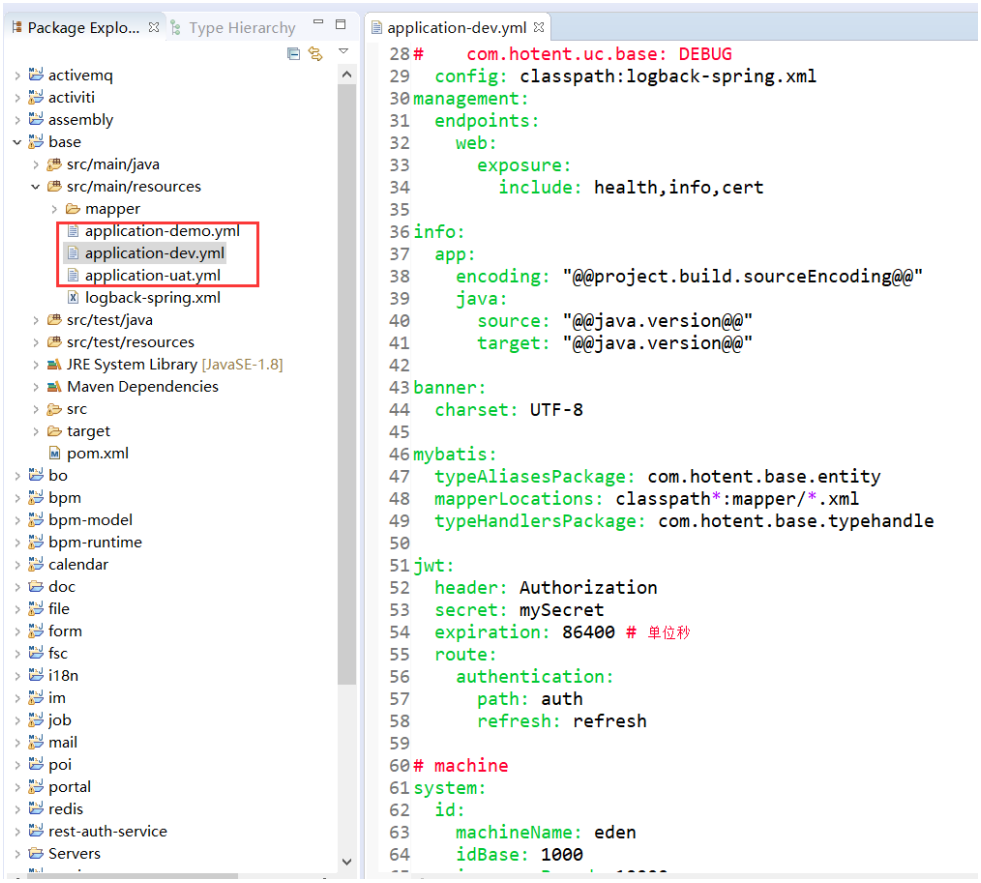
开发好的页面，需要在 router.js 中注册路由，如下图所示：


EXPLORER
todo.vue
router.js
OPEN EDITORS
X7-FRONT
relatedProcess.js
storeProcess.js
user.js
views
Flow.vue
Home.vue
Login.vue
Mail.vue
Navigator.vue
NotFound.vue
personal.vue
App.vue
directive.js
main.js
math.js
request.js
router.js
store.js
utils.js
tests
theme
.gitignore
babel.config.js
element-variables.scss
index.js
src
router.js
router
}
},
{
path: "/home",
name: "home",
component: Home
},
{
path: "/flow",
component: Flow,
children: [{
path: "/",
redirect: "todo"
},
{
path: "todo",
component: () => import("@/components/flow/todo.vue")
},
{
path: "done",
component: () => import("@/components/flow/done.vue")
},
{
path: "request",
component: () => import("@/components/flow/myRequest.vue")
}
]
},
/* 跳转启动流程 */
{

10 配置文件

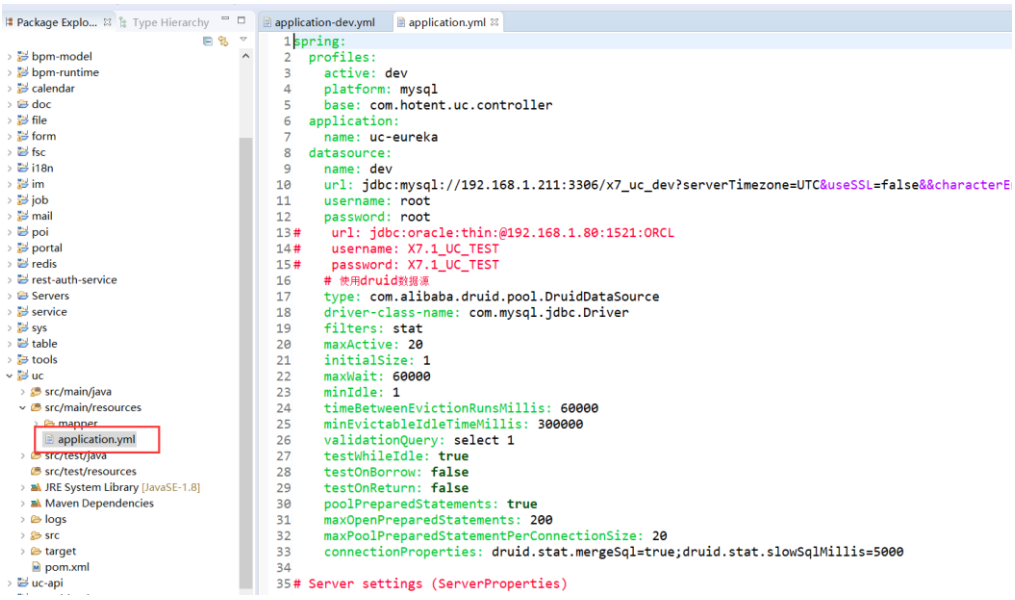
10.1 系统配置文件

系统的配置文件由两份构成，在 base 项目中定义了通用的配置项，如下图所示，application-xxx.yml 格式指定了在不同环境中应用的配置文件。



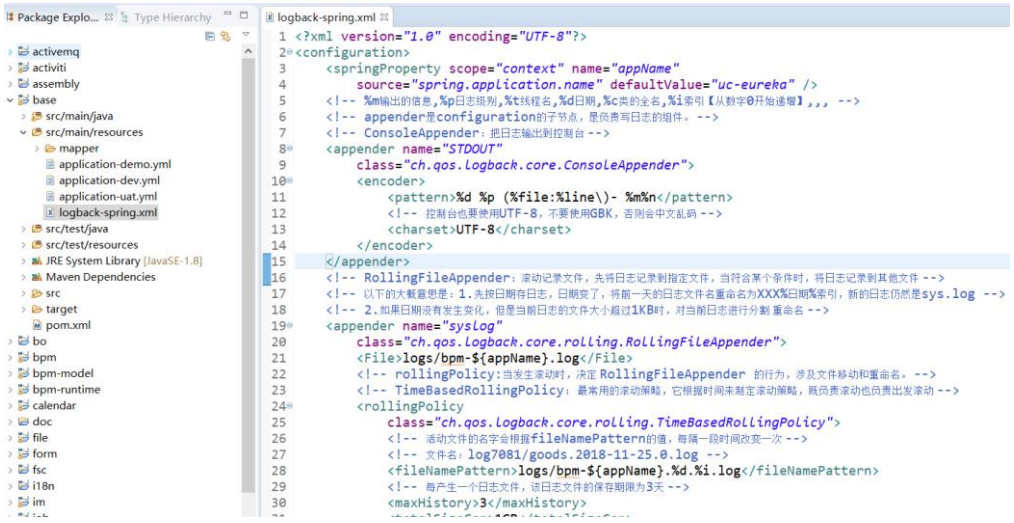
在具体的微服务中，定义 application.yml 来配置该微服务所需要的配置项。在运行这个微服务时，最终的配置文件会将 base 中的配置文件（多份配置文件选择哪份由 spring.profiles.active 属性来决定，例如下图中这个属性为 dev，则使用 base 中的 application-dev.yml）与当前微服务的配置文件合并为一份。

合并时，application-xxx.yml 的优先级会高一些，在两份配置文件中有相同配置项时，优先使用 application-xxx.yml 中的配置项。



10.2 日志配置

日志的配置通过 base 中的 logback-spring.xml 来进行配置。



10.3 SpringCloud 的优化配置

10.3.1Eureka 参数调整

如下图所示为 Eureka 的配置参数

1. enable-self-preservation 为自我保护模式，默认为 true，这里建议设置为 false。设置为 false 时关闭自我保护，自我保护模式其作用是当 eureka 中注册的微

服务大量下线时会触发，eureka 会认为这些微服务仍然在线，导致在 eureka 中看到这个微服务是正常的，但实际却调用不到。

2. eviction-interval-timer-in-ms 清理间隔，建议设置为 15 秒。

3. fetch-registry 是否从其他的 eureka 上获取所注册的微服务的信息，如果 eureka 部署了多个节点，建议设置为 true，而且 defaultZone 指向另外部署的 eureka 服务。

```yaml
server:
  port: 8761
eureka:
  server:
    # 设为false，关闭自我保护
    enable-self-preservation: false
    # 清理间隔（单位毫秒，默认是60*1000）
    eviction-interval-timer-in-ms: 15000
  instance:
    hostname: server1
  client:
    # 表示是否注册自身到eureka服务器
    #register-with-eureka: true
    # 是否从eureka上获取注册信息
    fetch-registry: true
    service-url:
      defaultZone: http://server2:8762/eureka/,http://server3:8763/eureka/
```

10.3.2 Feign 参数调整

在微服务中配置的 Feign 参数如下图所示：

1. connectTimeout 是连接超时时间，根据服务器的性能和网络状况来调整，在较好的情况下，建议设置为 5 秒，较差的情况下可以设置为 15 秒；

2. readTimeout 为读取数据的超时时间，建议设置为 connectTimeout 的两倍。

```yaml
# feign配置
feign:
  hystrix:
    enabled: true
  client:
    config:
      default:
        connectTimeout: 15000
        readTimeout: 30000
        loggerLevel: full
```

10.3.3Hystrix 参数调整

Hystrix 的参数配置如下图所示：

1. maxConcurrentRequests 为允许的最大并发请求量，根据系统的使用情况来调整，并发量较高时相应的调大这个参数。

2. timeoutInMilliseconds 为服务熔断的超时时间，特别注意这个参数和下面的 Ribbon 超时时间有关系，这个时间必须大于等于 Ribbon 超时时间乘以重试次数。

```
# 断路器配置
hystrix:
  metrics:
    enabled: true
  command:
    default:
      execution:
        isolation:
          strategy: SEMAPHORE   # THREAD  SEMAPHORE
          semaphore:
            maxConcurrentRequests: 200 # 默认情况下下面两个值都是10，也就是超过10个的并发会直接进入fallback方法，不会去真正请求
          thread:
            timeoutInMilliseconds: 60000 #缺省为1000
      fallback:
        isolation:
          semaphore:
            maxConcurrentRequests: 200   # 默认情况下下面两个值都是10，也就是超过10个的并发会直接进入fallback方法，不会去真正请求
```

10.3.4Ribbon 参数调整

Ribbon 是微服务间相互调用的负载均衡模块，其参数配置如下：

1. connectTimeout 和 readTimeout 为连接超时时间与读取超时时间，与上面的 Feign 参数一样，根据服务器性能和网络状况进行配置。

2. MaxAutoRetries 为调用某个节点超时后最大重试次数（可以看做调用第一个可用节点）。

3. MaxAutoRetriesNextServer 为调用另外一个可用节点的最大重试次数（可以看做换一个可用节点来重试，所以如果 Eureka 中的微服务可用节点只有一个时，这个参数是不生效的，因为没有另外一个可用节点来重试了）。

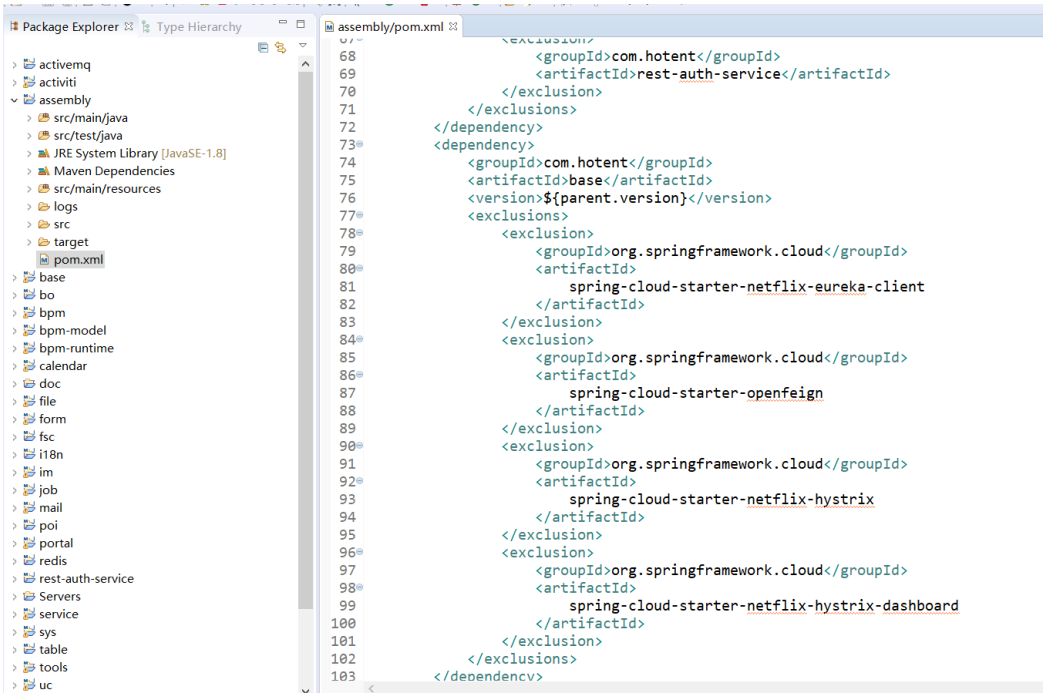
4. Hystrix 中的 timeoutInMilliseconds 的时间必须大于等于这里的 readTimeout × (MaxAutoRetries + MaxAutoRetriesNextServer)

```yaml
# 负载均衡配置
ribbon:
  eureka:
    enabled: true
  connectTimeout: 15000
  readTimeout: 30000
  MaxAutoRetries: 0 #同一台实例最大重试次数,不包括首次调用
  MaxAutoRetriesNextServer: 1 #重试负载均衡其他的实例最大重试次数,不包括首次调用
  OkToRetryOnAllOperations: false  #是否所有操作都重试

```

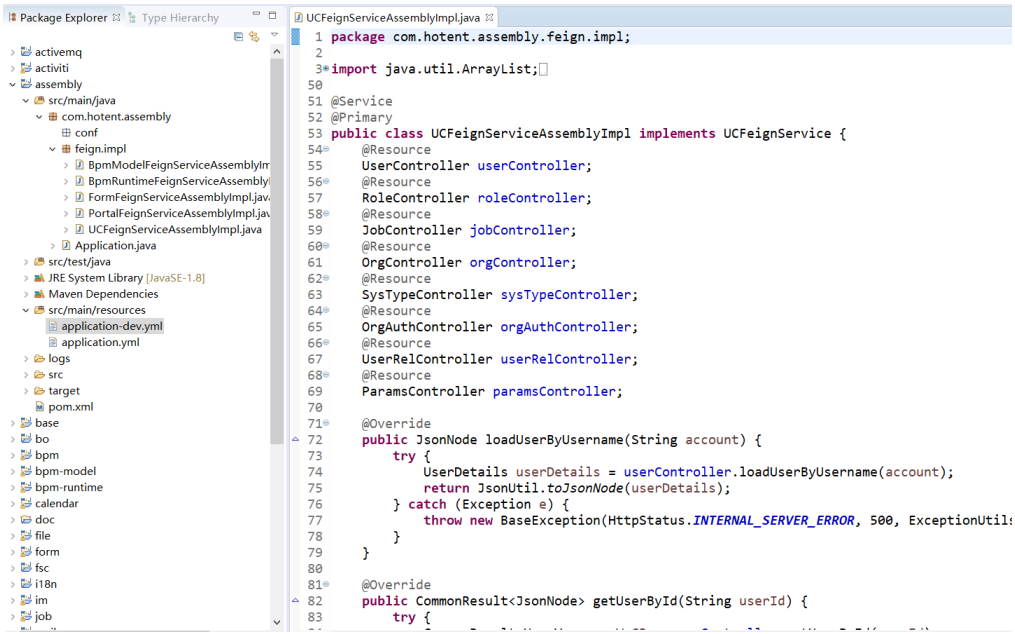
11 微服务五合一部署

在开发和部署时，按照五个微服务来分别部署运行会占用较多的内存，而且还需要部署 Eureka 服务实现微服务的注册与发现，所以不想以微服务来进行部署时，可以将五个服务合并为一个服务部署。五合一部署时，系统完全移除了 Spring Cloud 的依赖，不再需要 Eureka 服务。

如下图所示 assembly 项目为五合一的项目，其 pom.xml 中依赖了五个微服务，而且移除了 spring Cloud 相关的依赖。

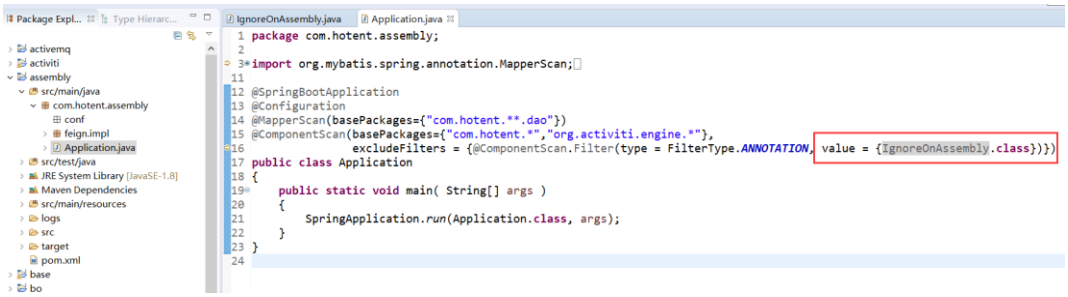


微服务之间的相互调用通过 Feign 完成，所以在 assembly 项目中还需要提供对 Feign 接口的实现类。在实现类中，直接注入各个微服务的 Controller 实例来实现相应的接口。



五合一时，只能连接一个数据库，所以在部署时需要将四个数据库初始化到一个数据库里面。

多个微服务合并到一个 assembly 项目中，有些实现类可能会存在冲突，我们通过定义了一个注解 IgnoreOnAssembly 来标记这些冲突的实现类，在 assembly 项目的入口类中，在扫描类时忽略被注解 IgnoreOnAssembly 标记的类。

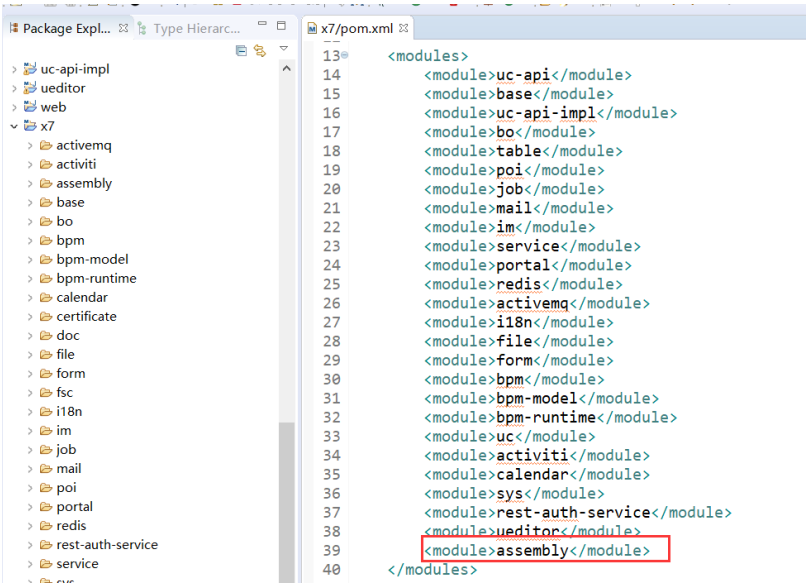


在打包时，我们需要修改五个微服务的 pom.xml 文件，注释掉生成 jar 包的部分。

uc/pom.xml

```
73                <exclusion>
74                    <groupId>com.hotent</groupId>
75                    <artifactId>uc-api-impl</artifactId>
76                </exclusion>
77            </exclusions>
78        </dependency>
79        <!-- service -->
80        <dependency>
81            <groupId>com.hotent</groupId>
82            <artifactId>service</artifactId>
83            <version>${project.version}</version>
84        </dependency>
85    </dependencies>
86
87    <!-- 用于生成jar文件
88    <build>
89        <finalName>uc</finalName>
90        <plugins>
91            <plugin>
92                <groupId>org.springframework.boot</groupId>
93                <artifactId>spring-boot-maven-plugin</artifactId>
94                <configuration>
95                    <mainClass>com.hotent.uc.Application</mainClass>
96                    <layout>JAR</layout>
97                </configuration>
98                <executions>
99                    <execution>
100                       <goals>
101                           <goal>repackage</goal>
102                       </goals>
103                   </execution>
104               </executions>
105           </plugin>
106       </plugins>
107   </build>-->
108 </project>
```

另外，确认在项目的根目录下 pom.xml 文件中，assembly 项目处于未注释掉的状态。



12 开发管理

12.1 编程规范

12.1.1重要性

编码规范对于程序员而言尤为重要，有以下几个原因：

- 一个软件的生命周期中，80%的花费在于维护
- 几乎没有任何一个软件，在其整个生命周期中，一直由最初的开发人员来维护。
- 编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码
- 如果你将源码作为产品发布，就需要确任它是否被很好的打包并且清晰无误，一如你已构建的其它任何产品

为了执行规范，每个软件开发人员**必须一致遵守编码规范**。

12.1.2注释

Java 程序有两类注释：文档注释(document comments)和实现注释(implementation comments)。

文档注释：由/** .../界定。文档注释可以通过 Javadoc 工具转换成 HTML 文件。

实现注释：也称普通注释。用以注释代码或者实现细节。普通注释是为了帮助开发者和阅读者更好地理解程序，不会出现在 API 文档中。其中，单行注释以“//”开头；多行注释以“/*”开头，以“*/”结束。

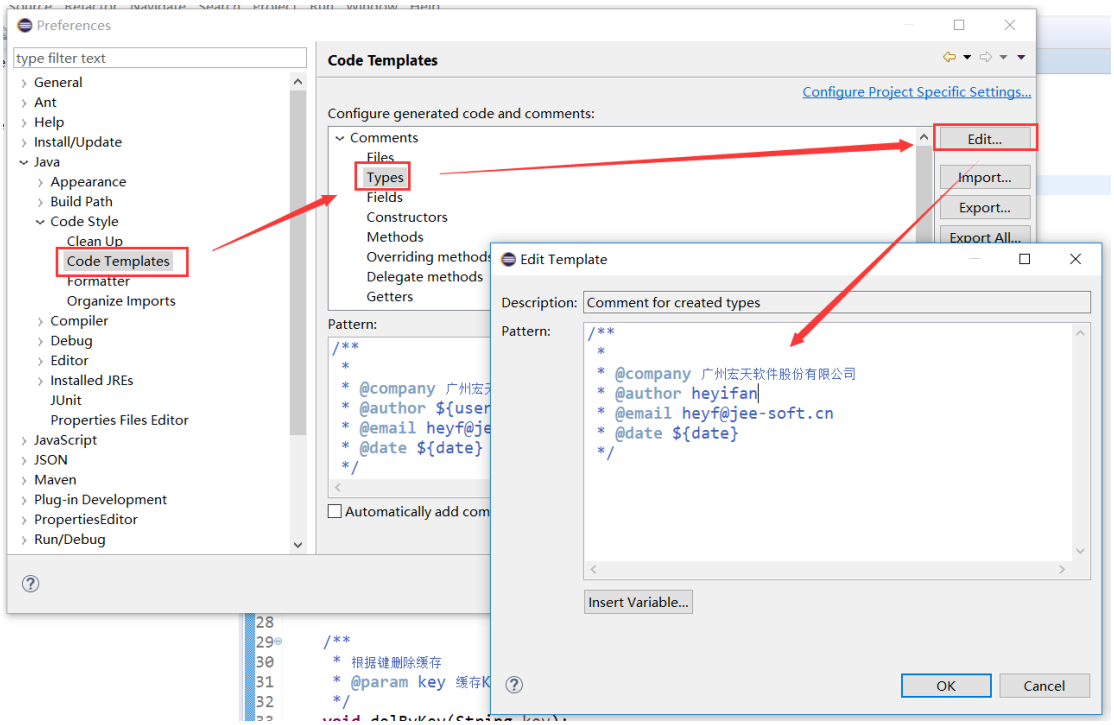
注意特别是以下代码必须写好注释：

- **接口注释、类注释，示例如下：**

```
/**
 * 缓存操作接口
 * <pre>
 * 定义了增加缓存，删除缓存，清除缓存，读取缓存的接口
 * </pre>
```

```
 * @company 广州宏天软件股份有限公司
 * @author  heyifan
 * @email   heyf@jee-soft.cn
 * @date    2018 年 4 月 9 日
 */
public interface ICache<T extends Serializable> {
}
```

使用 Eclipse 作为开发工具时，可以设置注释模板，设置了模板以后在接口或类文件头部输入 /**然后回车 Enter 即可生成注释，设置注释模板如下图所示：



如果生成的注释中，日期格式不是中文格式，修改 eclipse.ini 文件-Duser.language=zh-cn。

- **方法的注释，示例如下：**

```
/**
 * 添加缓存
 * @param key       缓存 Key
 * @param obj     缓存值
 * @param timeout 保存时限(单位：秒)
 */
void add(String key, T obj, int timeout);
/**
 * 将查询条件生成为 SQL 语句
 * @param isFirst    当前是否为第一个查询条件
 * @param clazz        查询条件所对应的实体类
 * @return        SQL 语句
```

```
 * @throws SystemException
 */
@SuppressWarnings("rawtypes")
public String toSql(Boolean isFirst, Class clazz) throws SystemException {
}
```

方法的作用，复杂时要配合<pre></pre> 或 标签做详述，各个参数的说明，返回值的说明。

接口中的方法均需注释，类中的 public protected 方法必须注释，private 方法除非望文生义，否则也需要注释。

12.1.3格式

或许你认为“让代码能工作”才是专业开发者的头等大事，但我希望我们的团队成员能够抛掉这种想法。

我们今天实现的所有功能，都有可能在交付客户后，给客户的开发人员进行修改和扩展，代码的可读性会对以后可能发生的修改行为产生深远的影响。

代码的整洁、一致性能够在第一次被阅读时就表现出我们对细节的关注，从而让客户感受到我们的专业。而反过来，如果他们看到的是一堆鬼画符般的代码，那他们多半会认为我们对项目的其它方面的细节也漠不关心，细节决定成败。

- **缩进**

4 个空格或者 1 个 tab 作为一个缩进排版的单位，每一个层级与上一层级相比往右缩进一单位。

```
private void initSqlValue(){
    if(hasInitValue){
        return;
    }
    hasInitValue = true;
    if (QueryOP.IN.equals(operation))
    {
        this.value = getInValueSql();
    }
    if(BeanUtils.isNotEmpty(value)){
        //TODO
    }
}
```

多个变量或参数用逗号分隔时，逗号后面留一个空格。

```
/**
 * 获取返回对象
 * @param result  执行结果:true > 成功, false > 失败
 * @param message 返回的消息
 * @return
 */
public Map<String, Object> modelMap(Boolean result, String message){
    Map<String, Object> modelMap = modelMap();
    modelMap.put("result", result);
    modelMap.put("message", message);
    return modelMap;
}
```

- **行长度**

一般情况避免一行的长度超过 80 个字符。

但实际开发中，也不需要限制在这个数量，超出几个字符问题不大，但如果超过 90 或 100，则应该使用换行了。

- **水平对齐**

对于一组声明中的变量名，或者一组赋值语句的右值。请尽量对齐。

```
protected String createBy;
protected String updateBy;
protected Date   createTime;
protected Date   updateTime;
protected String createOrgId;
```

- **换行**

多参数换行

```
private static void fitCubi(Point2D.Double[] d,
                int first,
                int last,
                Point2D.Double tHat1,
                Point2D.Double tHat2) {}
```

条件换行

```
if(! attributes.containsKey(key)
  || oldValue != newValue
  || (oldValue != null && newValue != null))
```

三元运算符换行

```
return (drawBound==null)
      ? new Rectangle2D.Double(0, 0, -1, -1)
      : (Rectangle2D.Double) drawBound.clone();
```

12.1.4命名

系统中命名使用英文单词或缩写，**不要使用拼音**。

命名格式主要使用驼峰式命名，即多个单词构成的名称，第一个单词第一个字母小写，其它的单词第一个字母大写，比如 orgName，processCmd 等等。

但对于一些方法的传入参数或局部变量则可以根据实际需要来选择驼峰式命名或下划线分隔，比如 weight_g（表示重量，按克计算），start_ms（开始时间，用毫秒 ms 表示）等。

- **命名的原则**

1. 选择专业的词下面提供了常用的单词和说明。单词的选择关键在于动词，名词比较容易选。

常用动词

英文	缩写	中文	英文	缩写	中文
query		查询	set		设置
find		查找	clear		清空
get		获取	dump		写入
display		显示	refresh		刷新
initialize	init	初始化	send		发送
manage		管理	receive	rec	接收
add		增加	retry		重试
edit		编辑	format		格式化
validate		验证	calculate	calc	计算
print		页面打印	sum		总计
create		创建	evaluate	eval	评估

英文	缩写	中文	英文	缩写	中文
update		更新	execute	exec	执行
save		保存	start		开启
remove		删除	launch/begin/open		发动/开始/开启
rebuild		重建	stop		停止
do		操作	run		运行
import		导入	fork		分支
export		导出	complete		完成
show		显示	modify		修改
delete		删除	finish		完成

助词

英文	缩写	中文	英文	缩写	中文
abstract		抽象的	current	curr	当前
first		第一个	last		最后一个
previous	prev	前一个	next		下一个

2. 避免泛泛的名字 避免使用如 abc、a1、a2、tmp、temp、retval、flag、foo 等等没有明显含义的名字。 除了某些特定的情况，如 for 循环中使用 i，冒泡算法中的 temp 等，但是绝大多数情况，请起一个有意义的名字。

3. 用具体的名字代替抽象的名字 比如 fruit（水果）对比 apple、orange；vehicle（交通工具）对比 car、bike 等； 除非抽象的名字用在抽象类或者接口中。

4. 为名字附加更多的信息 诸如 id、name、number、key 等等变量，如果不是处在实体类中，而是某个方法内的局部变量，那么这些名字的可读性很有

限。增加一些附加信息会避免误解。 如 orgId、orgName、repeat_number、orgKey 等等。

5. 增加内容的限定 有些名字具备一些默认的含义，比如 password，默认就是加密的密码，但是如果你希望传递的是明文的密码，那么最好加上一些限定会避免误解，比如 plaintext_password 或者 plaintextPassword。

6. 增加单位描述 这里主要针对数值，比如某个数值表示时、分、秒、毫秒、千克、克或者其它计量单位，那么单纯一个 weight、start、end 等等，不足以描述清楚，而且如果弄错了单位，可能导致其它的错误。这时候可以加上单位： 如：weight_kg、weight_g、start_ms、end_s 等等。

7. 关于缩写 不要随便缩写，很容易造成理解困难。 缩写遵循以下原则：

8. 有通用缩写采用通用缩写

9. 无通用缩写：

10. 单词小于等于 6 位，无需缩写

11. 当整个名称的单词小于或等于 3 个，无需缩写

12. 当整个名称的单词超出 3 个，使用单词的前四位字母作为缩写代替，然后在注释中说明

13. 关于长名字 长名字带来丰富信息的同时，某些开发人员会认为它难以输入。但现在的 IDE 已经提供了很多快捷方式。比如 Eclipse 的：

14. 代码模板（输入 syso，自动转成 System.out.println("");

15. 补全或提示：输入前几位字母，输入 `alt + /`，会自动补全或提示。

- **包命名**

包名称均为小写单词，格式如下：

```
com.[company].[system].[module]
```

如：com.hotent.base.cache
com.hotent.mail.linkman

- **类命名**

1. 类必须由大写字母开头而单词中的其他字母均为小写；

2. 如果类名称由多个单词组成，则每个单词的首字母均应为大写例如 PartyService；

3. 如果类名称中包含单词缩写，则这个所写词的每个字母均应大写，如：

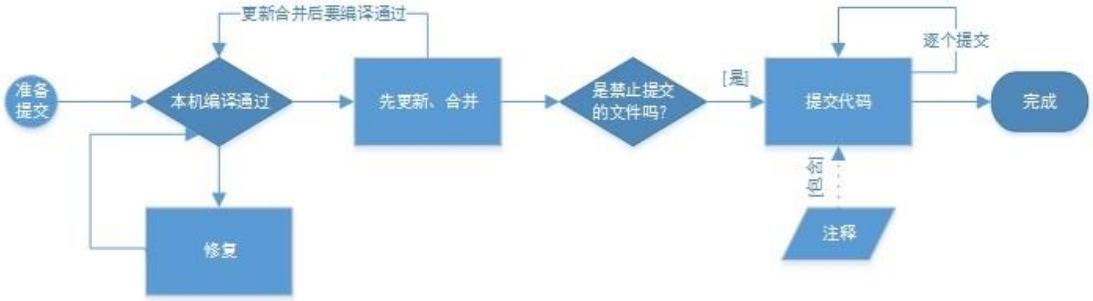
XMLUtils；

4. 由于类是设计用来代表对象的，所以在命名类时应尽量选择名词。长度不超过 18 位；
5. 实现类在该接口名称后面添加后缀 Impl，如 MyBatisDaoImpl；
6. 类方法命名采用动词+名词，如 sendMessage()。

12.1.5单元测试

- 程序员的价值在于和他人合作，开发出高质量的代码，而不是开发出一堆虫件（bugware）。
- 程序员必须对自己的代码质量负责，单元测试是对自己代码质量的基本承诺。
- 程序 = 单元测试（UT） + 代码（CODE）
- 对于组件化的项目，除了 1~2 个子项目是可以通过界面进行测试之外，其它的项目均是 java 代码。如果一个 bug 不能在单元测试阶段被发现，而是在集成测试，甚至是系统测试阶段才被发现，那么排查和修复成本将是非常高昂的。
- 不做好单元测试，就会影响团队其他人员的工作。
- 测试人员有权利对没有做过单元测试的代码说不。
- 其他的开发人员也有权利对没有做过单元测试的代码说不。
- 项目经理在日常管理工作中，单元测试是重要的考察部分，也是界定出错责任的关键依据。
- public 和 protected 的方法都有相应的 UT 方法。

12.2 版本控制



1. 严禁使用其他人的账号提交代码。
2. 准备阶段
3. 本机是否编译通过？
4. 本机不能编译通过，不允许做提交操作。
5. 这些代码自己是否能够看明白？
6. 提交的代码必须是原子性的，如完成了某个功能，修复了某个 bug，做了某个页面调整等。
7. 对于只是自己使用的代码，比如 private 方法，可以阶段性完成即提交，但最起码自己看得明白，不是垃圾代码。
8. 先更新、合并代码，每一次要提交前，先查看要提交的文件在 SVN 服务器上是否有修改。 如 NetBeans 就是右键点击要提交的目录，选择 Subversion 显示更改。
9. 更新合并后，是否编译通过？如编译不通过，则调整代码或更新编译依赖的其它的文件。
10. 禁止提交的文件
11. 本地 IDE 自动生成的文件；
12. 本地测试的文件，如 uploads 的图片、附件等。（这些目录应该设置为 svn ignore）
13. 配置文件的本地化修改。比如 app.properties，使用了本机的数据库。
14. 提交代码 —— 逐个文件提交！除了下面这些例外，其它的文件请逐个提交：
15. 第三方组件的新增或者升级。（注：我方做个性化重构时需要逐一提交）

16. 某个目录的文件全部是新增的。

17. 样式设计和设计图片。

18. 每次提交必须书写明晰的标注逐个文件提交，单独写标注，记录调整历史，方便其他开发人员更新和理解。

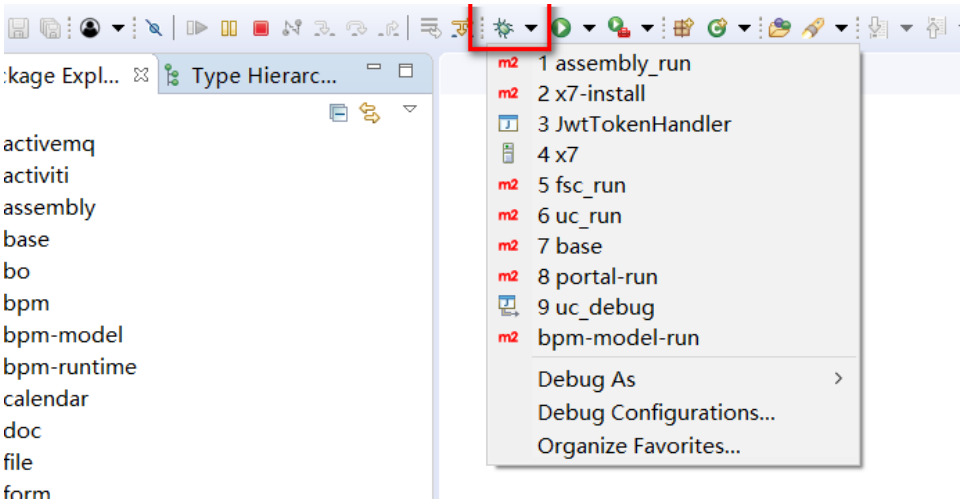
19. 描述本次提交的文件变更内容。

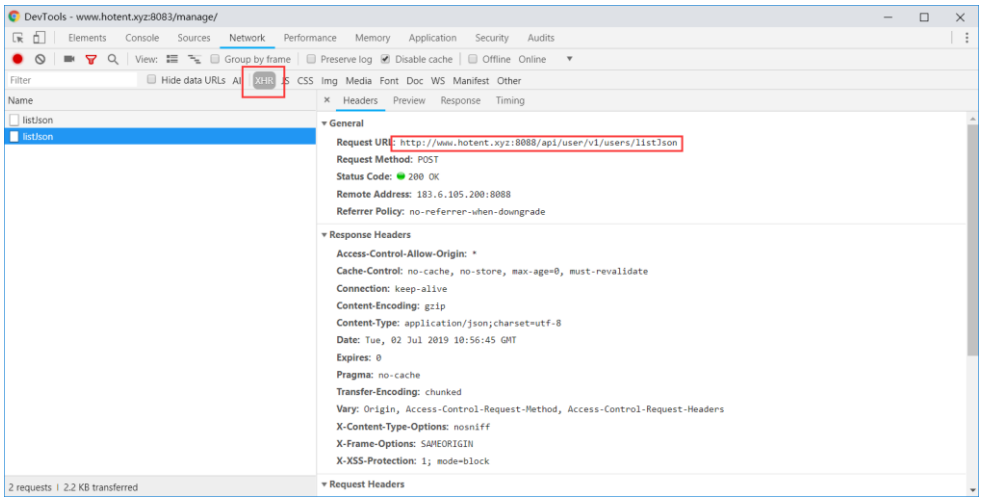
20. 如果修复了禅道上的 bug，请附上 BUG ID。

21. 发布版本周期要短 —— 管理员打 Tag 这要求项目经理将每个小版本的待开发功能控制在一定数量内，保证周期要短，这样才可以经常打 tag 发布版本。 步子要小，这样项目才更加可控，每个版本的稳定性才更能保证。

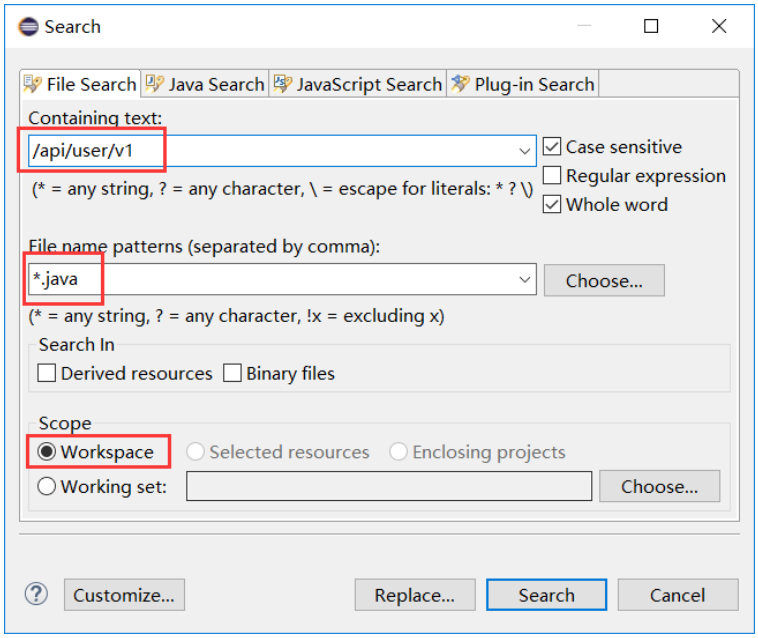
12.3 系统调试方法

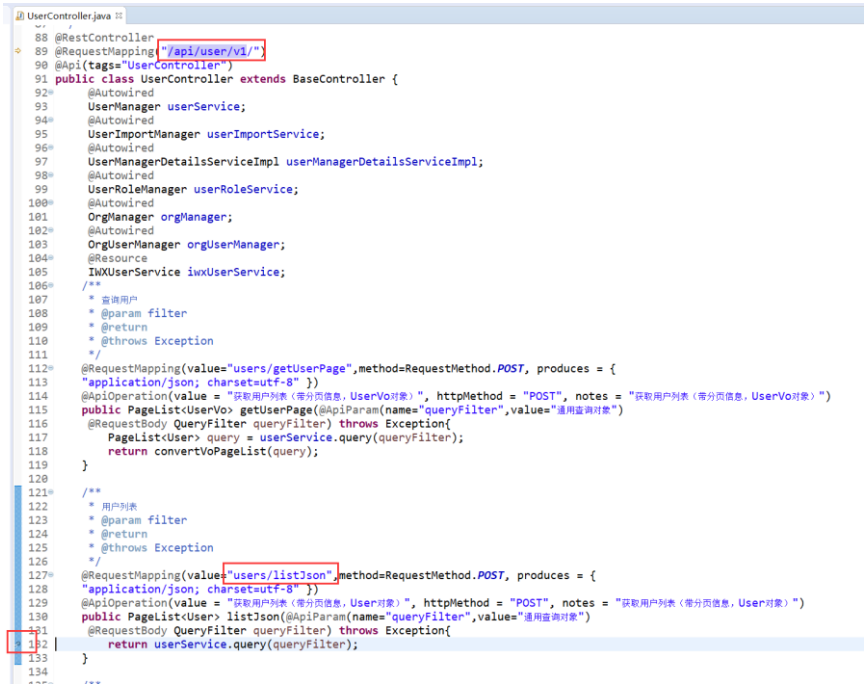
系统在开发调试过程中，可以通过以下方法进行前后端的调试。首先以调试模式运行项目，然后在前端通过浏览器的调试工具（以 Chrome 为例，F12 打开调试工具）找到页面请求的后端地址。





最后再通过搜索/api/user/v1 找到对应 controller 代码，在对应的方法处打上调试断点即可进入调试模式。

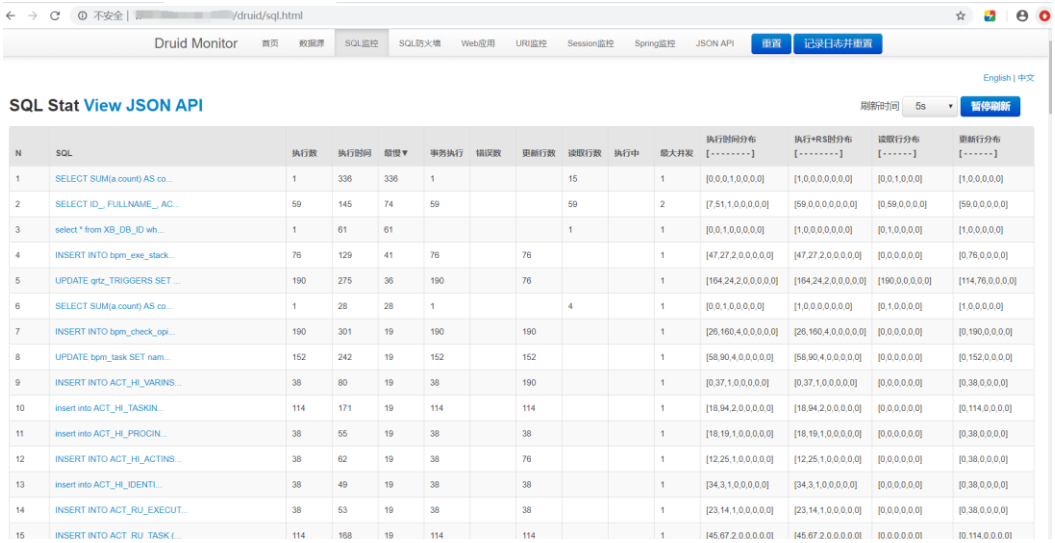




```java
UserController.java
@RestController
@RequestMapping("/api/user/v1/")
@Api(tags="UserController")
public class UserController extends BaseController {
    @Autowired
    UserManager userService;
    @Autowired
    UserImportManager userImportService;
    @Autowired
    UserManagerDetailsServiceImpl userManagerDetailsServiceImpl;
    @Autowired
    UserRoleManager userRoleService;
    @Autowired
    OrgManager orgManager;
    @Autowired
    OrgUserManager orgUserManager;
    @Resource
    IWXUserService iwxUserService;
   /**
    * 查询用户
    * @param filter
    * @return
    * @throws Exception
    */
   @RequestMapping(value="users/getUserPage",method=RequestMethod.POST, produces = {
   "application/json; charset=utf-8" })
   @ApiOperation(value = "获取用户列表（带分页信息，UserVo对象）", httpMethod = "POST", notes = "获取用户列表（带分页信息，UserVo对象）")
   public PageList<UserVo> getUserPage(@ApiParam(name="queryFilter",value="通用查询对象")
    @RequestBody QueryFilter queryFilter) throws Exception{
       PageList<User> query = userService.query(queryFilter);
       return convertVoPageList(query);
   }

   /**
    * 用户列表
    * @param filter
    * @return
    * @throws Exception
    */
   @RequestMapping(value="users/listJson",method=RequestMethod.POST, produces = {
   "application/json; charset=utf-8" })
   @ApiOperation(value = "获取用户列表（带分页信息，User对象）", httpMethod = "POST", notes = "获取用户列表（带分页信息，User对象）")
   public PageList<User> listJson(@ApiParam(name="queryFilter",value="通用查询对象")
    @RequestBody QueryFilter queryFilter) throws Exception{
       return userService.query(queryFilter);
   }
```

12.4 调优方法

系统在使用过程中，随着数据量的增加及新功能的开发，可能某些功能会出现性能的问题，例如加载数据很慢、提交和保存很慢等情况。这一类的问题通过 druid 来监视 SQL 的执行效率，通过以下访问地址可以打开 druid 的管理界面：http://ip:port/druid,如下图所示，输入 admin/admin 可以进入管理页面。



通过分析 SQL 监控页面的慢查询，可以找到待优化的 SQL，通过增减索引或者调整 SQL 语句的方式来进行优化。

另外针对数据量特别大的表，可以通过创建历史表，定期归集数据到历史表。

或者创建表分区的方式来优化。