1 考勤管理

1.1 归档列表

考勤的归档列表是为了查询考勤的历史记录已经每个员工的当月考勤情况,对数据做备份。

（1）controller

```
//查询年度范围内，每月的考勤总览数据
@RequestMapping(value = "/reports/year" ,method = RequestMethod.GET)
public Result findReports(String year) throws Exception {
    List<ArchiveMonthly> list =
archiveService.findReportsByYears(year,companyId);
    return new Result(ResultCode.SUCCESS,list);
}

//查询当月考勤归档的详细数据
@RequestMapping(value = "/reports/{id}" ,method = RequestMethod.GET)
public Result findReportsById(@PathVariable String id) throws Exception {
    List<ArchiveMonthlyInfo> list = archiveService.findReportsById(id);
    return new Result(ResultCode.SUCCESS,list);
}
```

（2）service

```
//根据年份和企业id查询归档总表数据
public List<ArchiveMonthly> findReportsByYears(String year, String companyId) {

    return atteArchiveMonthlyDao.
        findByCompanyIdAndArchiveYearOrderByArchiveMonthDesc(companyId,year);
}
//根据归档总表数据查询明细列表
public List<ArchiveMonthlyInfo> findReportsById(String id) {
    return archiveMonthlyInfoDao.findByAtteArchiveMonthlyId(id);
}
```

（3）dao

```
//查询某一年的归档列表
List<ArchiveMonthly> findByCompanyIdAndArchiveYearOrderByArchiveMonthDesc(
    String  companyId, String archiveYear);
//根据归档列表查询月归档详情
List<ArchiveMonthlyInfo> findByAtteArchiveMonthlyId(String
atteArchiveMonthlyId);
```

1.2 查询已归档考勤

为了方便页面和其他微服务（薪资）的调用，需要根据用户id和考勤月份进行数据查询

（1）controller

```
RequestMethod.GET)
public Result historysData(@PathVariable  String userId,@PathVariable  String
yearMonth) throws Exception {
    ArchiveMonthlyInfo info = archiveService.findUserArchiveDetail(userId,
yearMonth);
    return new Result(ResultCode.SUCCESS,info);
}
```

（2）service

```
//查询用户的归档数据
public ArchiveMonthlyInfo findUserArchiveDetail(String userId, String yearMonth)
{
    return archiveMonthlyInfoDao.findByUserIdAndArchiveDate(userId,yearMonth);
}
```

（3）dao

```
//查询用户的归档数据
ArchiveMonthlyInfo findByUserIdAndArchiveDate(String userId, String yearMonth);
```

2 薪资管理

薪资管理是指企业制定的合理的工资发放制度及系统，包括不同员工的薪资标准、薪资的明确组成部分、发放薪资的政策、薪资发放办法和原则、对该员工工作评价制度和薪资评价制度等。薪资管理针对不同的企业有不同的模式，薪资管理是企业管理的重要组成部分。

2.1 需求分析

序号	姓名	手机	工号	聘用形式	部门	入职时间	工资基数	津贴方案	操作
1	itcast	13800000002	9002	正式	test1	2018-11-02	10	通用方案	调薪 查看
2	zbz	13800000003	111	正式	测试部	2018-11-04	40	通用方案	调薪 查看
3	ll	13800000004	1111	正式	测试部	2018-12-02	6	通用方案	调薪 查看
4	a01	13400000001	1001	正式	开发部	2018-01-01	200	通用方案	调薪 查看
5	a02	13400000002	1002	正式	开发部	2018-01-01	0	通用方案	定薪 查看
6	test001	13500000001	2001	正式	开发部	2018-01-01	14	通用方案	调薪 查看
7	test002	13500000002	2002	正式	开发部	2018-01-01	0	通用方案	定薪 查看
8	test003	13500000003	2003	正式	开发部	2018-01-01	20	通用方案	调薪 查看

对于薪资模块，所涉及的功能有：

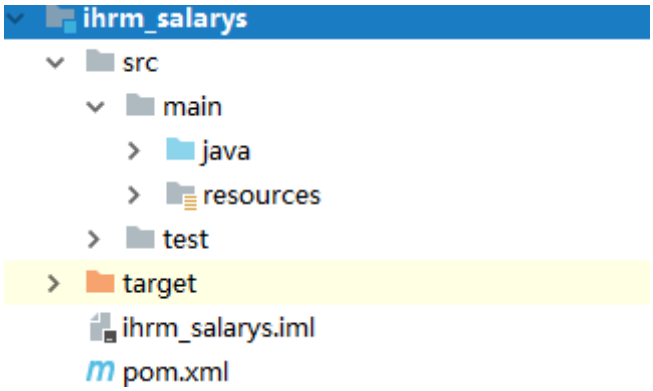
- 展示当月薪资列表
  - 展示员工当月工资技术与福利津贴方案。
- 员工调薪/顶薪
  - 调薪：发起调薪申请且通过后，可以设置员工新的薪资数据
- 查看员工薪资详情

- 数据列表：显示当月员工的所有薪资数据（包含薪资税前总额，扣款金额，社保，公积金金额，税后总金额等）
- 归档：当月薪资制作完成且无误后可以进行薪资数据的记录归档

2.2 搭建环境

2.2.1 环境搭建

（1）创建模块`ihrm_salarys`



导入依赖：

```
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
            <groupId>com.ihrm</groupId>
            <artifactId>ihrm_common</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>com.ihrm</groupId>
            <artifactId>ihrm_common_model</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-openfeign</artifactId>
        </dependency>
    </dependencies>
```

（2）创建启动类

```
@EnableFeignClients
public class SalarysApplication {
    /**
     * 启动方法
     */
    public static void main(String[] args) {
        SpringApplication.run(SalarysApplication.class, args);
    }


    @Bean
    public IdWorker idWorker() {
        return new IdWorker();
    }
}
```

（3）配置application.yml

```
#spring配置
spring:
  #1.应用配置
  application:
    name: ihrm-salarys #指定服务名
  #3.JPA
  jpa:
    database: MySQL
    show-sql: true
    open-in-view: true
  #2.数据库连接池
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ihrm?useUnicode=true&characterEncoding=utf8
    username: root
    password: 111111
    hikari:
      maximum-pool-size: 2
  redis:
    host: 127.0.0.1
    port: 6379
#注册到eureka的服务地址
eureka:
  client:
    service-url:
      defaultZone: http://localhost:6868/eureka/
  instance:
    preferIpAddress : true
    instance-id: ${spring.cloud.client.ip-
address}:${spring.application.name}:${server.port}
#服务配置
server:
  port: 9007
```

（4）配置网关

```
        serviceId: ihrm-salarys #指定Eureka注册中心中的服务id
        strip-prefix: false
        sentiviteHeaders:
        customSensitiveHeaders: true
```

至此，薪资模块的环境搭建工作就已经完成了。由于薪资模块，考勤模块，社保模块三者的业务逻辑和接口逻辑大同小异，我们本次课只重点关注数据归档。其他的功能（考勤列表，定薪/调薪，考勤详情）这里直接使用准备好的代码即可。

2.2.2 导入代码

将资料中准备的代码导入到工程中，步骤略。。。

2.2.3 考勤列表

（1）controller

前端页面post请求到list接口。为了简单直接将参数封装为map并获取。

```
@RequestMapping(value = "/list", method = RequestMethod.POST)
public Result list(@RequestBody Map map) {
    //1.获取请求参数,page,size
    Integer page = (Integer)map.get("page");
    Integer pageSize = (Integer)map.get("pageSize");
    //2.调用service查询
    PageResult pr = salaryService.findAll(page,pageSize,companyId);
    return new Result(ResultCode.SUCCESS, pr);
}
```

（2）service

service调用dao进行查询

```
public PageResult findAll(Integer page, Integer pageSize, String companyId) {
    Page page1 = userSalaryDao.findPage(companyId, new PageRequest(page - 1,
pageSize));
    return new PageResult(page1.getTotalElements(),page1.getContent());
}
```

（3）dao

由于列表查询需要进行多表操作，这里通过自定义sql语句的形式进行分页查询

```
        "bu.in_service_status inServiceStatus,bu.department_name
departmentName,bu.department_id departmentId,bu.time_of_entry timeOfEntry ," +
        "bu.form_of_employment formOfEmployment ,sauss.current_basic_salary
currentBasicSalary,sauss.current_post_wage currentPostWage from bs_user bu LEFT
JOIN sa_user_salary sauss ON bu.id=sauss.user_id WHERE bu.company_id = ?1",
        countQuery = "select count(*) from bs_user bu LEFT JOIN sa_user_salary
sauss ON bu.id=sauss.user_id WHERE bu.company_id = ?1"
      )
Page<Map> findPage(String companyId, PageRequest pageRequest);
```

@Query注解

nativeQuery ： 是否本地SQL查询（true（是）| false（JPQL语句））

value： 查询的SQL语句

countQuery ： 分页查询中需要的查询总记录数的SQL语句

2.2.4 定薪/调薪

定薪 ： 值刚刚入职的员工还没有设定薪资标准需要进行定薪（设置员工的试用期薪资以及转正之后的薪资）

调薪 ： 修改已入职员工薪资

（1）Controller

```
//修改员工薪资（调薪）
@RequestMapping(value = "/modify/{userId}", method = RequestMethod.POST)
public Result modify(@RequestBody UserSalary userSalary) throws Exception {
    salaryService.saveUserSalaryChange(userSalary);
    return new Result(ResultCode.SUCCESS);
}


//设置员工薪资（定薪）
@RequestMapping(value = "/init/{userId}", method = RequestMethod.POST)
public Result init(@RequestBody UserSalary userSalary) {
    salaryService.saveUserSalary(userSalary);
    return new Result(ResultCode.SUCCESS);
}
```

2.2.5 福利津贴设置

不同的企业对员工福利津贴有不同的需求，这里也支持设置


```
/**
     * 获取企业计薪及津贴设置
     */
@RequestMapping(value = "/settings", method = RequestMethod.GET)
public Result getSettings() throws Exception {
    Settings settings = settingsService.findById(companyId);
    return new Result(ResultCode.SUCCESS, settings);
}

/**
     * 保存企业计薪及津贴设置
     */
@RequestMapping(value = "/settings", method = RequestMethod.POST)
public Result saveSettings(@RequestBody Settings settings) throws Exception {
    settings.setCompanyId(companyId);
    settingsService.save(settings);
    return new Result(ResultCode.SUCCESS);
}
```

2.3 数据归档

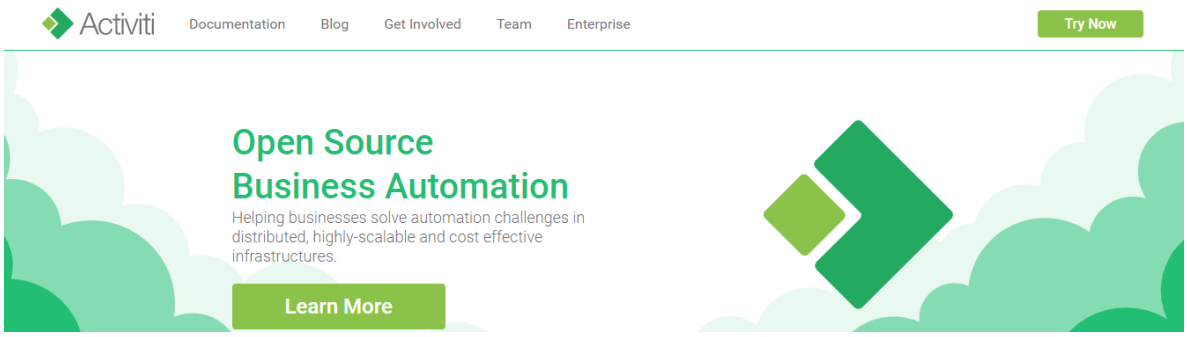
列表展示与数据归档是薪资中比较重要的组成部分。涉及到员工的薪资统计与计算

3 工作流概述

3.1 什么是工作流

工作流（workflow）就是工作流程的计算模型，即将工作流程中的工作如何前后组织在一起的逻辑和规则在计算机中以恰当的模型进行表示并对其实施计算。它主要解决的是“使在多个参与者之间按照某种预定义的规则传递文档、信息或任务的过程自动进行，从而实现某个预期的业务目标，或者促使此目标的实现”

3.2 activiti工作流概述



Activiti是一个开源的工作流引擎，它实现了BPMN 2.0规范，可以发布设计好的流程定义，并通过api进行流程调度。Activiti 作为一个遵从 Apache 许可的工作流和业务流程管理开源平台，其核心是基于 Java 的超快速、超稳定的 BPMN2.0 流程引擎，强调流程服务的可嵌入性和可扩展性，同时更加强调面向业务人员。Activiti 流程引擎重点关注在系统开发的易用性和轻量性上。每一项 BPM 业务功能 Activiti 流程引擎都以服务的形式提供给开发人员。通过使用这些服务，开发人员能够构建出功能丰富、轻便且高效的 BPM 应用程序。

3.3 ihrm中的流程介绍

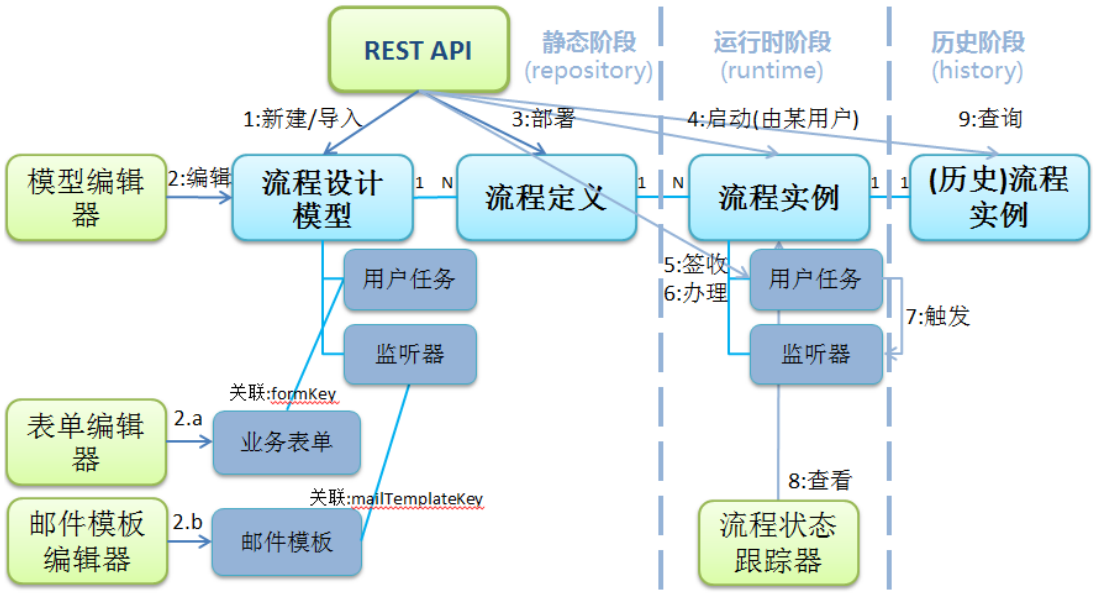
3.3.1 概述

业务流是项目中最为核心的部分，为了能够将业务流程进一步规范化出来，在我们的 SaaS-IHRM项目中也可以将日常业务规范化处理，这样我们就可以使用所学的工作流引擎来完成操作。我们的审批中心模块：

- 请假/调休流程
  - 请假流程先有HR专员进行行行审批，之后由部门门负责人人进行行行审批，在此处判断请假天数，如在3天内，则直接通过即可，如超过3天则需要总裁办任意人人员进行行行审批。
- 离职流程
  - 转正需要员工工自自己己提交申请，提交申请后由部门门负责人人进行行行审批，审批通过后由人人事部进行行行审批，之后由人人事部负责人人审批，之后由财务部进行行行审批（任意人人员），通过后流程结束。
- 加班流程
  - 加班需要员工工自自行行行提交，提交后由部门门负责人人进行行行审批，之后由人人事部进行行行审批（任意一一人人）之后完成审批。

3.3.2 开发流程

在IHRM系统中，所有的流程全部使用Activiti 工作流引擎进行管理。那么在这种基于前后端分离的项目，我们应该怎样进行开发呢？这就设计到Activiti 工作流开发的几个声明周期。



- **静态阶段** ： 包含流程设计和流程部署
- **运行时阶段**： 用户发起流程，各责任人对流程进行审核驳回等操作，Activiti 自动的根据流程状态
- **历史阶段**：对于以关闭或审核通过的历史流程，进行查询的管理

综上所知：在项目中使用Activiti工作流引擎进行流程控制。一般而言可以分为三步

1. 流程制作
2. 流程执行：工作流已进入“运行中”阶段
3. 查询历史流程

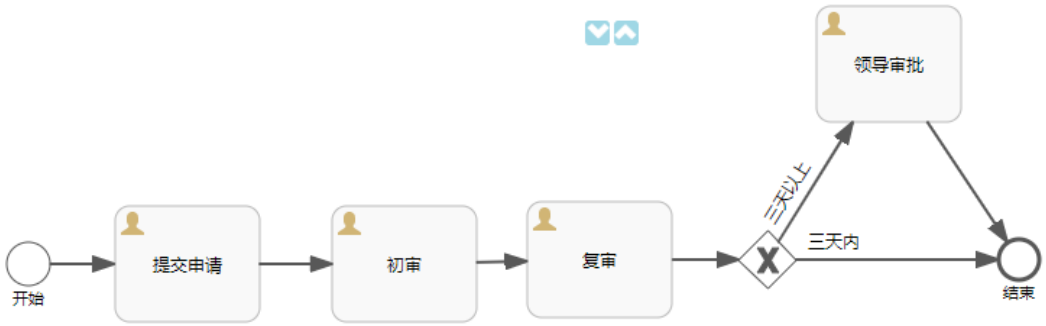
那接下来先以请假为例完成第一个工作，即制作流程。

4 流程制作

本次项目，我们先以请假流程来进行分析和讲解。

请假的业务，是我们最为熟悉的一套业务了，主要包括请假当事人，以及他的直属领导，或更高层的领导来逐个进行审批，最后完成一个请假的流程。每个公司的请假流程细节上可能存在差异，但总体流程都差不多。下面我们一起来看一个简版的请假流程，如下：

- 当事人发起请假申请
- 直属领导根据实际情况进行审核，如果没有问题就通过，流程继续向后执行。
  如果有问题就可以不通过，流程就终止。
- 直属领导审批通过之后，进入人事部门复审阶段。对于三天内请假可以复审结束请假完成
- 对于三天以上的请假需要进行内部领导审批，结束之后放个请假完成



4.2 流程制作工具

对于Activiti工作流而言，制作流程是不可缺少的一环。制作工作流程往往可以通过以下几种方式完成：

- 手写xml文件
- 基于IDEA中的actiBMP插件
- 通过java编程式生成xml文件
- 基于Activiti官方提供的流程制作web工具

我们这里采用的是官方提供的activiti-app工具包，完成流程制作的环节。

4.2.1 下载与安装

（1）下载

略

（2）部署

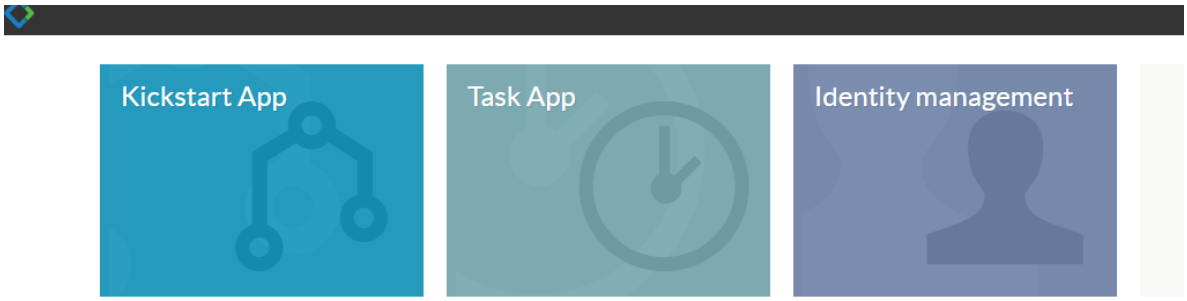
直接将资料中的`activiti-app.war`放置到tomcat的`webapps`中，启动tomcat即可运行

（3）访问

浏览器访问http://ip:port/activiti-app，即可访问页面制作工具

输入默认的用户名/密码即可登录并制作流程： admin/test

4.2.2 面板介绍

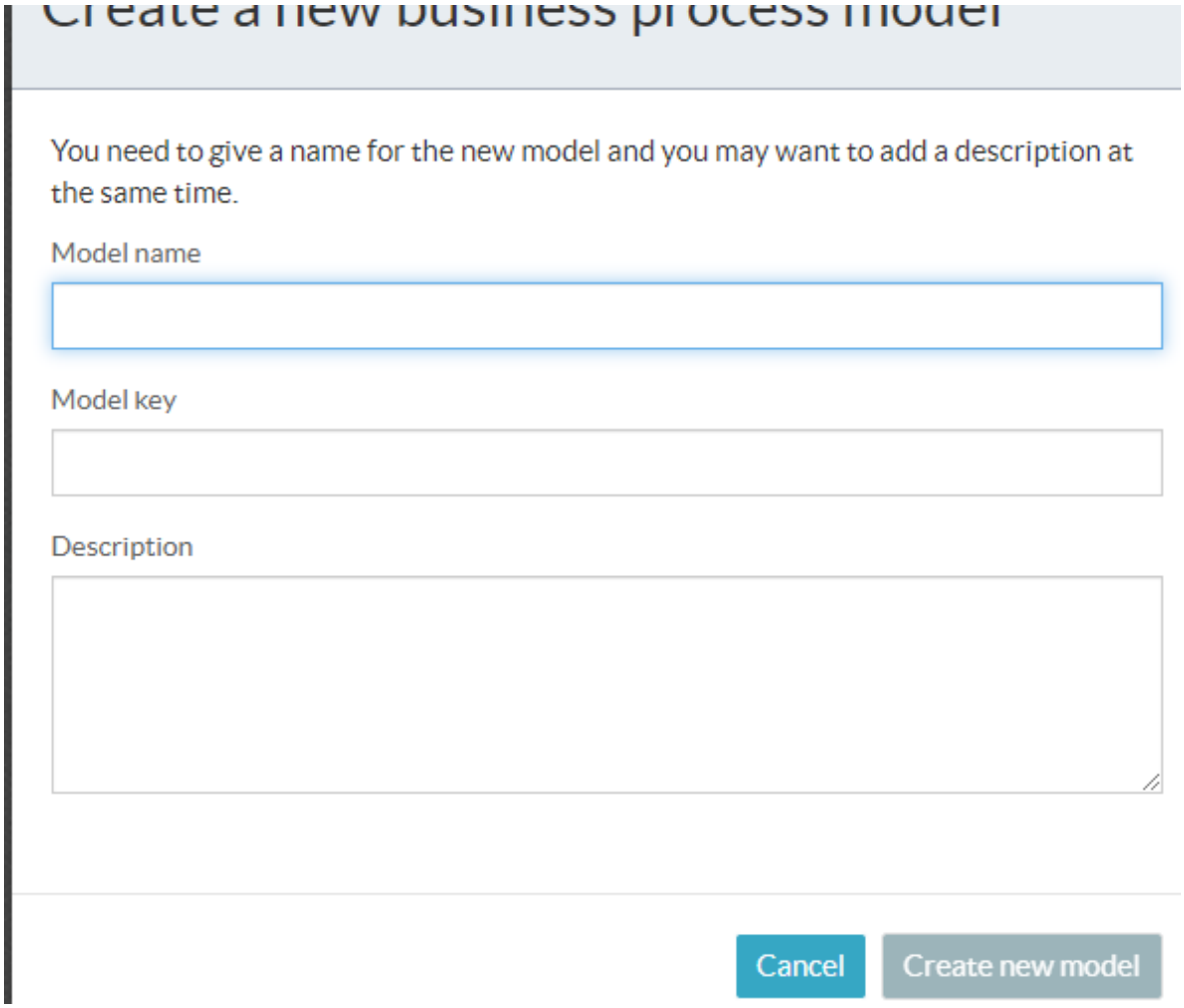


主界面的三个菜单主要承担以下功能：

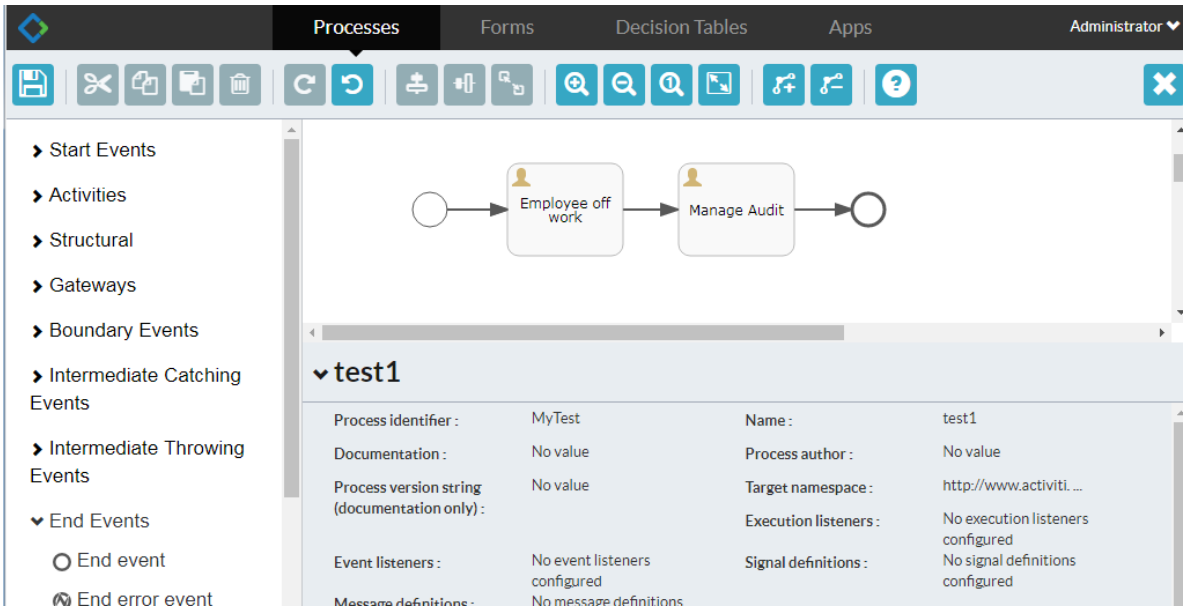
- Kickstart App：主要用于流程模型管理、表单管理及应用（App）管理，一个应用可以包含多个流程模型，应用可发布给其他用户使用。
- Task App：用于管理整个activiti-app的任务，在该功能里面也可以启动流程。
- Idenity management：身份信息管理，可以管理用户、用户组等数据。

通过`Kickstart App`即可完成流程模型的制作与管理

在主界面点击“Kickstart App”菜单，进入流程模型管理的主界面，点击“Create Process”按钮，弹出新建流程模型界面：

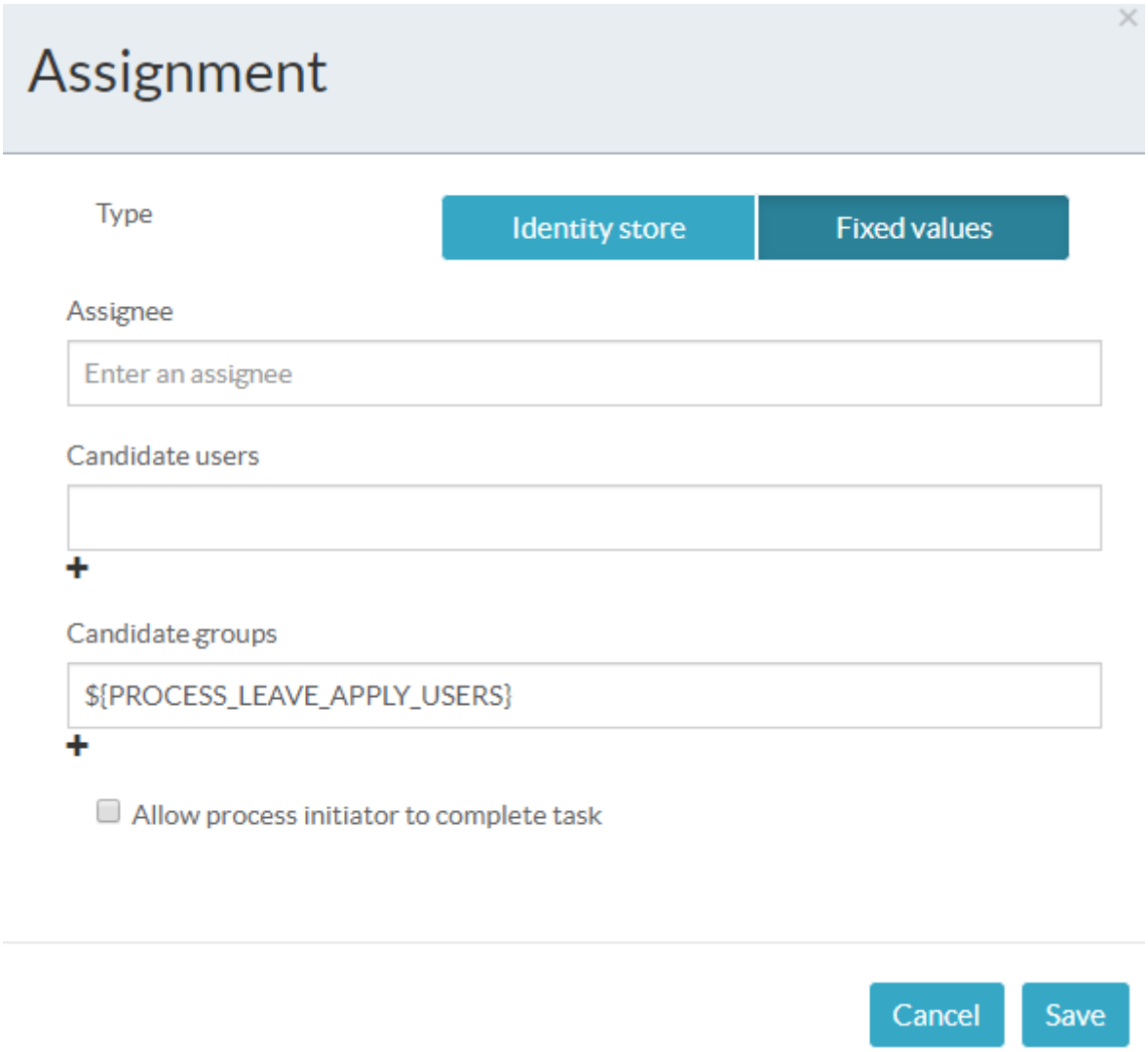


输入模型信息后，会进入流程模型设计界面，在流程设计界面中，只需要普通的鼠标拖拉操作，即可完成流程模型的定义，该编辑器也可以开放给业务人员使用。根据前面定义的请假流程，在编辑器中"拖拉"一下，定义请假流程模型：

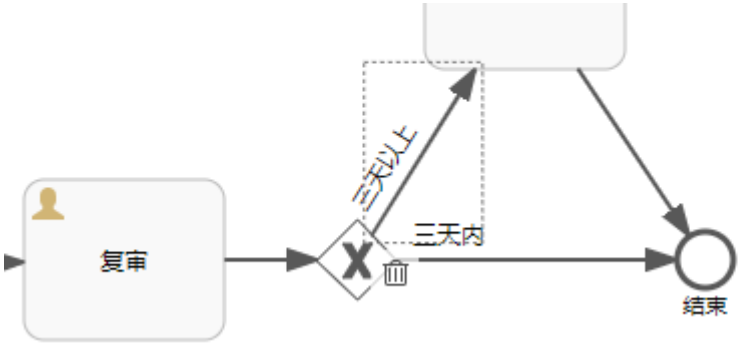


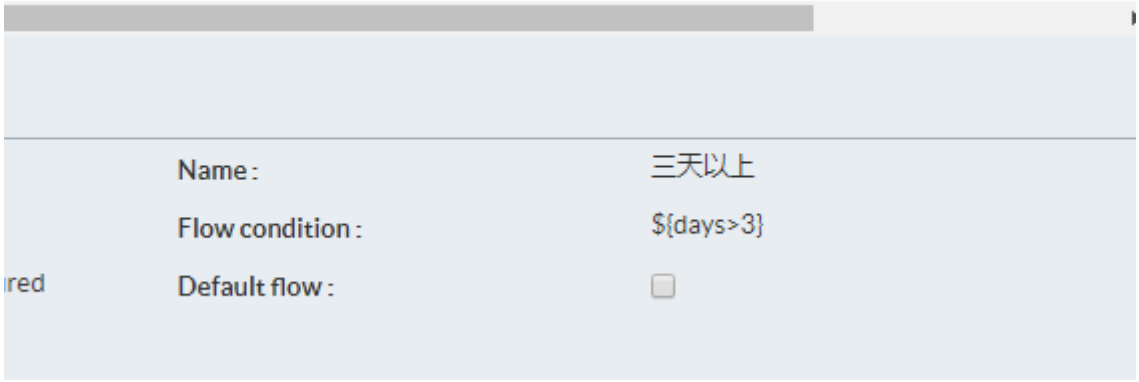
4.3 流程制作

行
者。



请假当事人发起请求时，先由HR专员进行审批，之后由部门负责人审批，此时请假天数会决定不同的流程走向（判断请假天数，如在3天内，则直接通过即可，如超过3天则需要总裁办任意人员进行审批），我们在此设置 days的流程变量来保存，days<=3时表示通过流程结束；days>3时，代表需要流转到更高一级单位进行审批



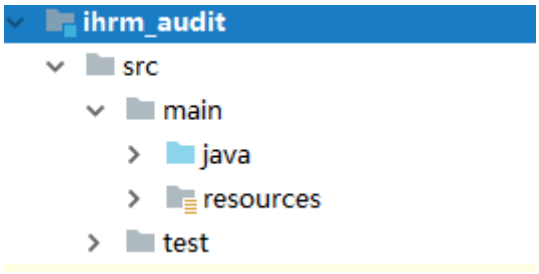


4.4 流程部署

通常情况下，一些流程是提前规定好的（比如我们的请假流程），此时我们可以提前在我们的流程设计器中将流程先设计出来，再将已设计好的流程通过文件上传方式，先上传到服务器，同时再将流程定义文件部署到 activti流程引擎中。

4.4.1 环境搭建

（1）创建模块`ihrm_audit`




```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
    <dependency>
        <groupId>com.ihrm</groupId>
        <artifactId>ihrm_common</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
```

```
        <artifactId>ihrm_common_model</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-openfeign</artifactId>
    </dependency>
    <dependency>
        <groupId>org.activiti</groupId>
        <artifactId>activiti-spring-boot-starter</artifactId>
        <version>7.0.0.Beta3</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>
    <!-- Activiti生成流程图 -->
    <dependency>
        <groupId>org.activiti</groupId>
        <artifactId>activiti-image-generator</artifactId>
        <version>7.1.0.M2</version>
        <exclusions>
            <exclusion>
                <groupId>commons-io</groupId>
                <artifactId>commons-io</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <!-- https://mvnrepository.com/artifact/commons-fileupload/commons-
fileupload -->
    <dependency>
        <groupId>commons-fileupload</groupId>
        <artifactId>commons-fileupload</artifactId>
        <version>1.3.1</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

（2）导入配置类和配置文件

将资料中准备好的配置文件和配置类导入到项目中

```
#spring配置
```

```
  application:
    name: ihrm-audit #指定服务名
  #2.数据库连接池
  datasource:
    act:
      jdbc-url: jdbc:mysql://localhost:3306/act?
useUnicode=true&characterEncoding=utf8&serverTimezone=GMT
      username : root
      password : 111111
      driver-class-name: com.mysql.jdbc.Driver
    ihrm:
      jdbc-url: jdbc:mysql://localhost:3306/ihrm?
useUnicode=true&characterEncoding=utf8&serverTimezone=GMT
      username : root
      password : 111111
      driver-class-name: com.mysql.jdbc.Driver
  redis:
    host: 127.0.0.1
    port: 6379
  #3.JPA
  jpa:
    database: MySQL
    show-sql: true
    open-in-view: true
  activiti:
    history-level: full
    db-history-used: true
eureka:
  client:
    service-url:
      defaultZone: http://localhost:6868/eureka/
  instance:
    preferIpAddress : true
    instance-id: ${spring.cloud.client.ip-
address}:${spring.application.name}:${server.port}
#服务配置
server:
  port: 9009
```

其中需要注意的是需要在项目中引入Activiti的数据库配置

（3）网关中加入相应配置

```
    ihrm-user-audit: #用户自助-审批
      path: /user/** #配置请求URL的请求规则
      serviceId: ihrm-audit #指定Eureka注册中心中的服务id
      strip-prefix: false
      sentiviteHeaders:
      customSensitiveHeaders: true
```

4.4.2 流程部署

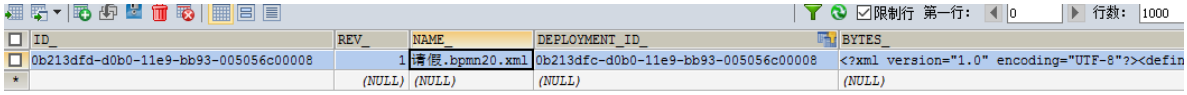
（1）编写控制器Controller

```
public Result deployProcess(@RequestParam("file") MultipartFile file) {
    processService.deployProcess(file);
    return new Result(ResultCode.SUCCESS);
}
```

（2）service代码编写

```
@Override
public void deployProcess(MultipartFile files) throws IOException {
    String fileName = files.getOriginalFilename();
    Deployment deploy = repositoryService.createDeployment()
        .addBytes(fileName, files.getBytes()).deploy();
    System.out.println(" 部署 ID:" + deploy.getId());
}
```

当部署成功后，我们可以查看 activiti数据库的 act_ge_bytearray表



ID_	REV_	NAME_	DEPLOYMENT_ID_	BYTES_
0b213dfd-d0b0-11e9-bb93-005056c00008	1	请假.bpmn20.xml	0b213dfc-d0b0-11e9-bb93-005056c00008	<?xml version="1.0" encoding="UTF-8"?><defin
	(NULL)	(NULL)	(NULL)	(NULL)