

Activiti7 流程管理项目核心技术栈

后台技术：Activiti 7、SpringBoot 2.5、SpringSecurity、MyBatis-Plus 3.3.1、Lombok、MySQL、Swagger

前端技术：vue、element-ui、axios、vue-router、vuex 等

Activiti7-Vue 前端项目环境搭建

安装 Node.js & NPM

NPM 全称 Node Package Manager，它是 JavaScript 的包管理工具，并且是 Node.js 平台的默认包管理工具。通过 NPM 可以安装、共享、分发代码，管理项目依赖关系。

其实我们可以把 NPM 理解为前端的 Maven。我们通过 npm 可以很方便地安装与下载 js 库，管理前端工程。

最新版本的 Node.js 已经集成了 npm 工具，所以必须首先在本机安装 Node 环境。

Node.js 官网下载地址：

- 英文网：<https://nodejs.org/en/download/>
- 中文网：<http://nodejs.cn/download/>

1. 安装Node.js与配置NPM环境变量

参考：04-配套软件\Nodejs&NPM安装包\Nodejs安装教程和NPM环境变量配置.pdf

装完成后，查看当前 nodejs 与 npm 版本

```
C:\Users\Administrator>node -v
v10.15.3

C:\Users\Administrator>npm -v
6.4.1
```

2. 配置NPM 淘宝镜像加速

1. 查看当前使用的镜像地址

```
npm get registry
```

2. 配置淘宝镜像地址

```
npm config set registry https://registry.npm.taobao.org
```

3. 安装下载模块

```
npm install <Module Name>
```

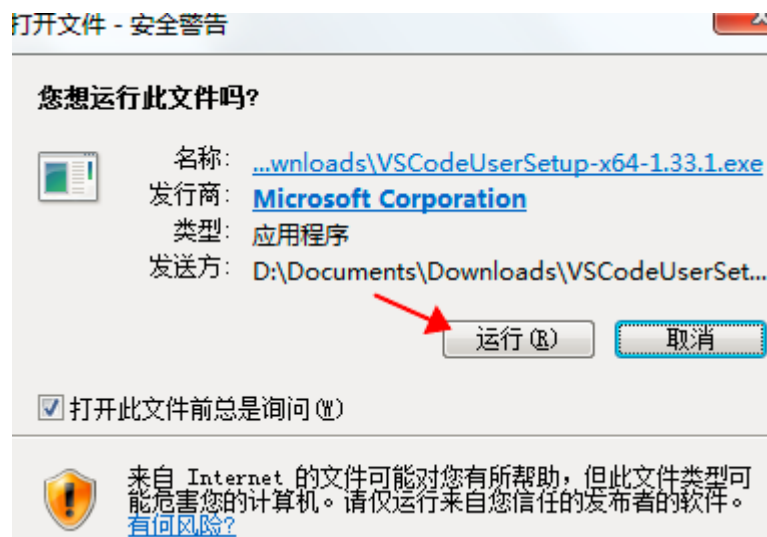
比如: `npm install jquery@2.2.0`

安装 VS Code 开发工具

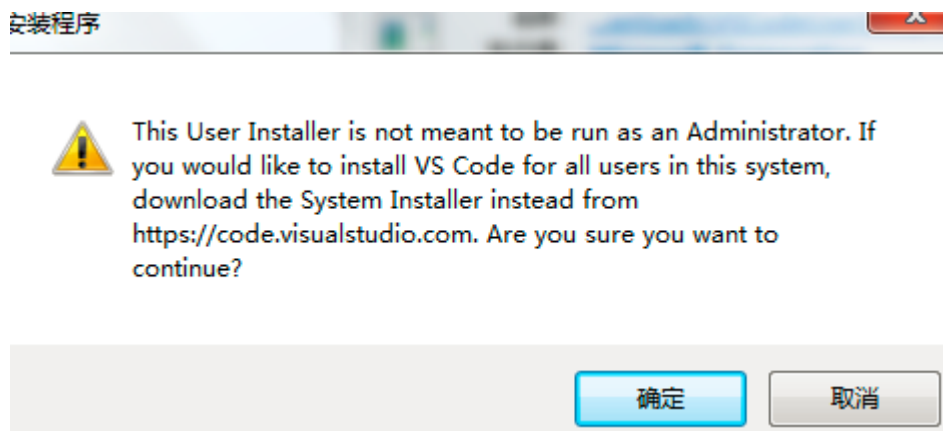
VS Code 下载地址: <https://www.visualstudio.com/>

安装包位于: 04-配套软件\VSCodeUserSetup-x64-1.33.1.exe

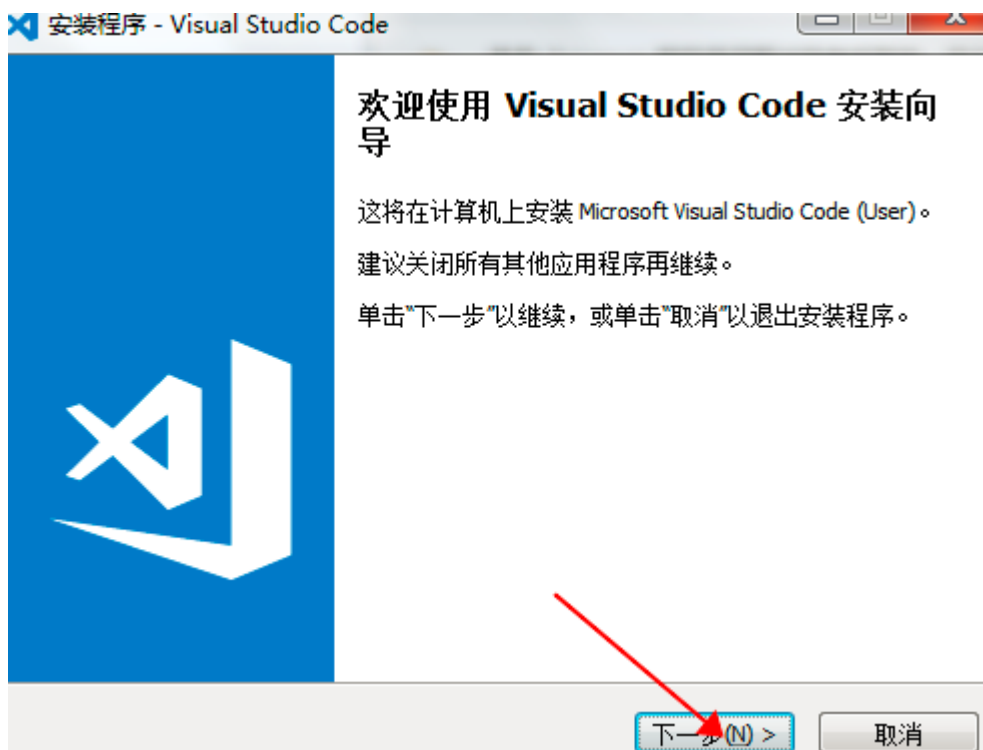
双击安装包，打开点击 运行



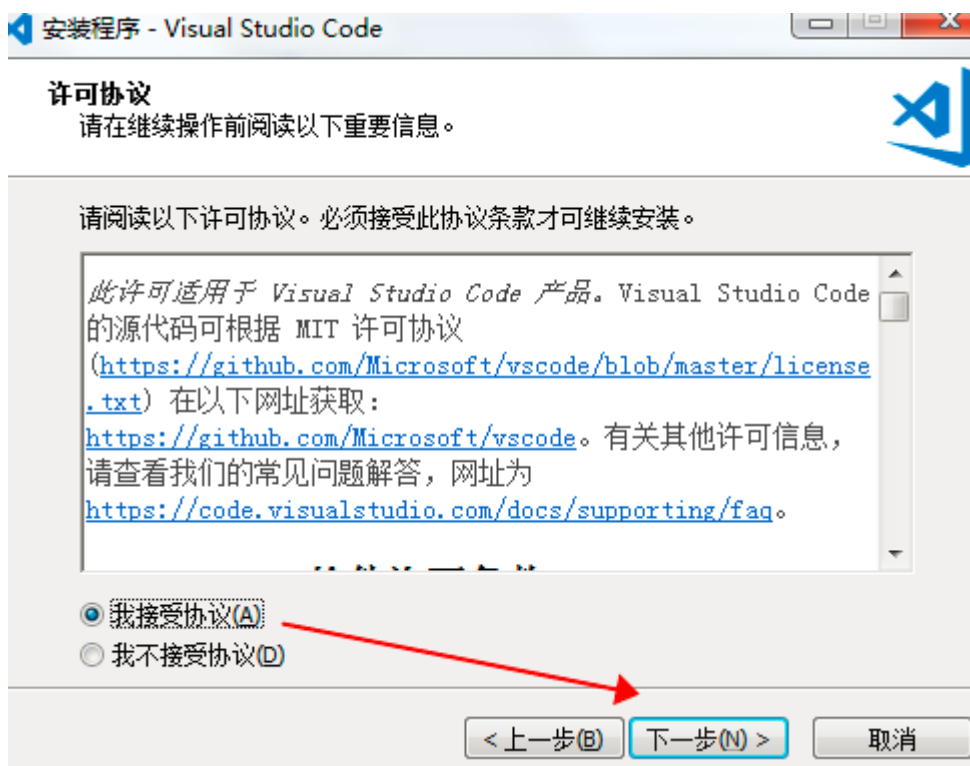
点击 确定



点击 下一步

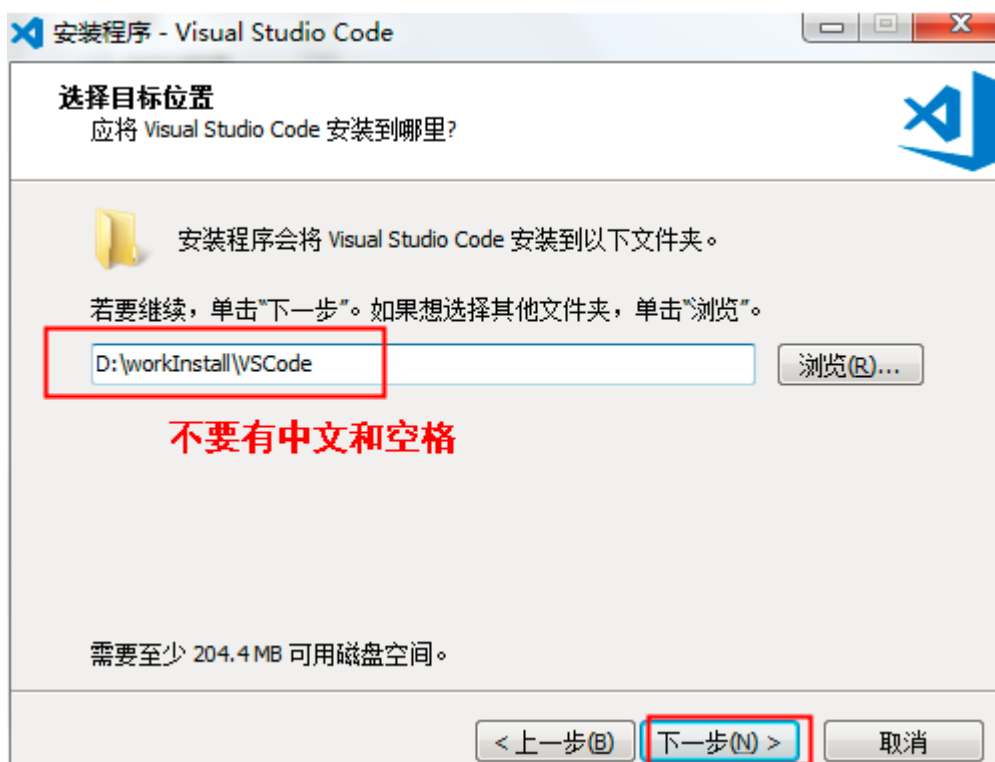


勾选 我接受协议 > 下一步

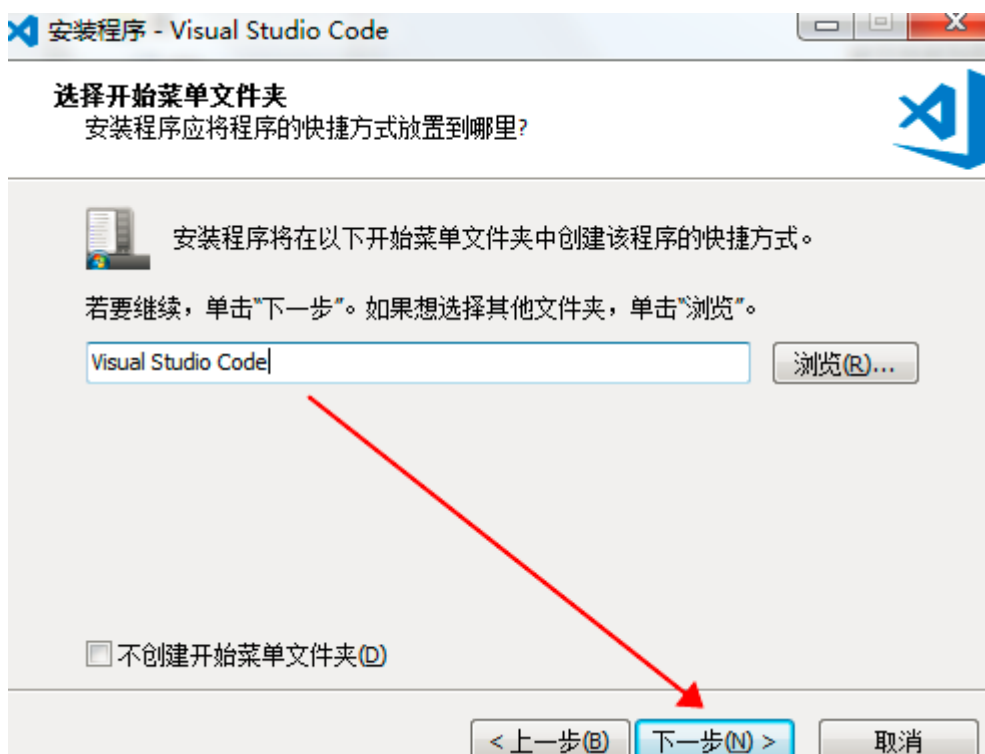


指定安装路径

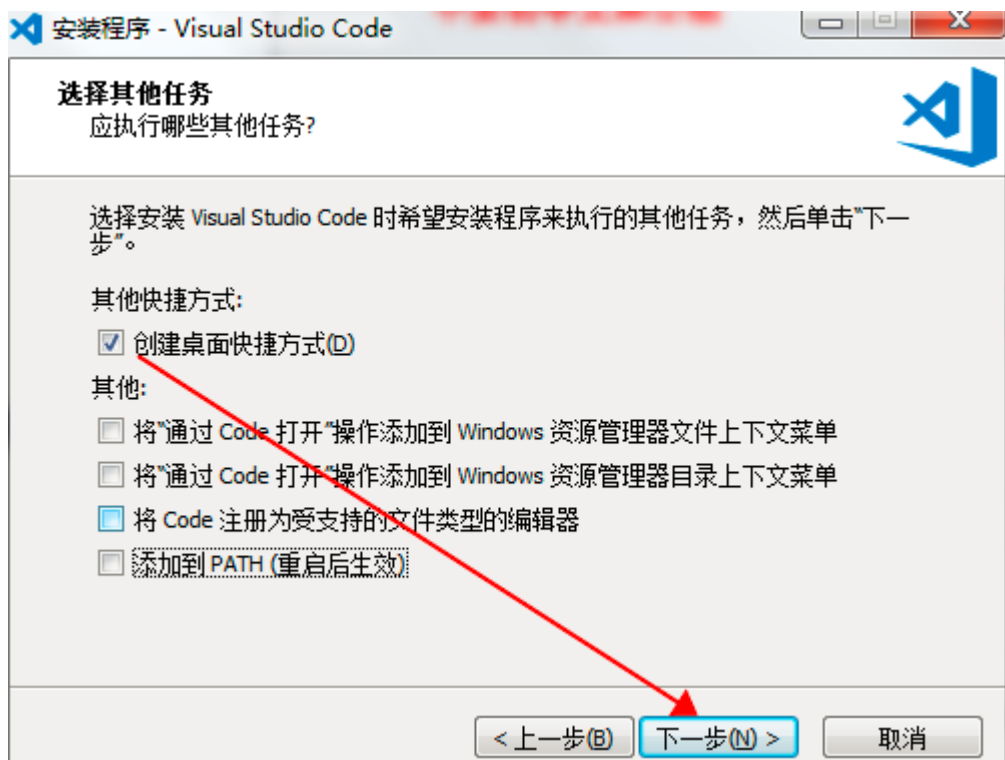
- 路径不要有空格和中文



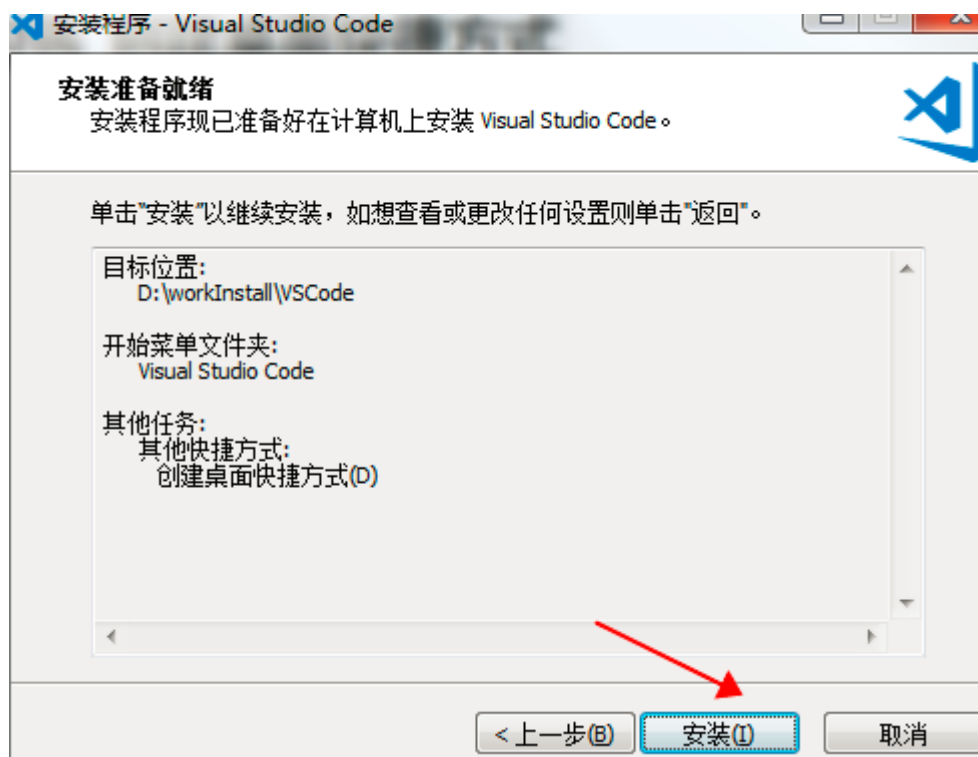
快捷方式，默认即可



勾选 创建桌面快捷方式



点击 **安装**



会进入安装页面，等待一会即安装成功

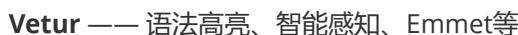
使用 VS Code

VS Code 使用手册: <https://code.visualstudio.com/docs>

安装完成后，我们看到的Visual Studio Code界面如下，当然不同的系统界面边框略有不同，基本布局如图：



1. 安装 Vue 插件，安装后有快捷提示，在 扩展栏 的搜索框中输入插件名



Auto Rename Tag —— 自动完成另一侧标签的同步修改

Path Intellisense —— 自动路径补全

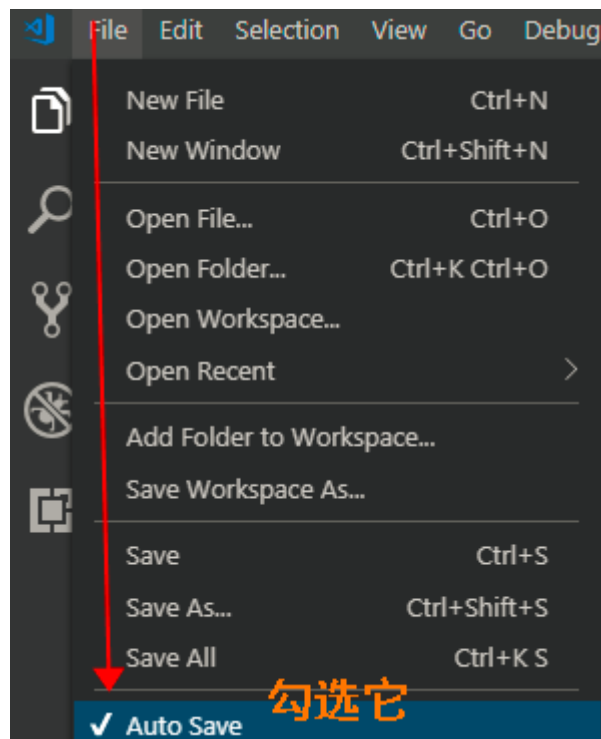
HTML CSS Support —— 让 html 标签上写class 智能提示当前项目所支持的样式，安装后有快捷提示

2. 汉化 vscode 插件 `chinese` , 安装后重启 vscode



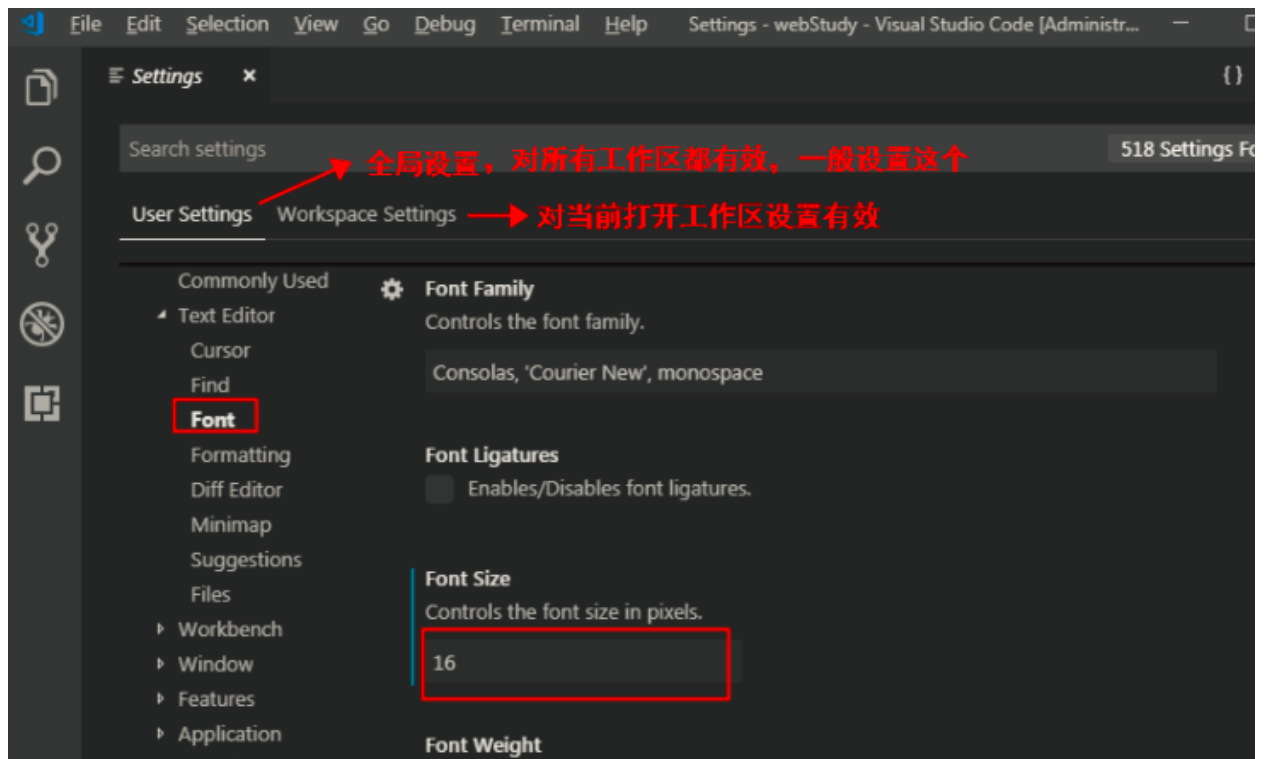
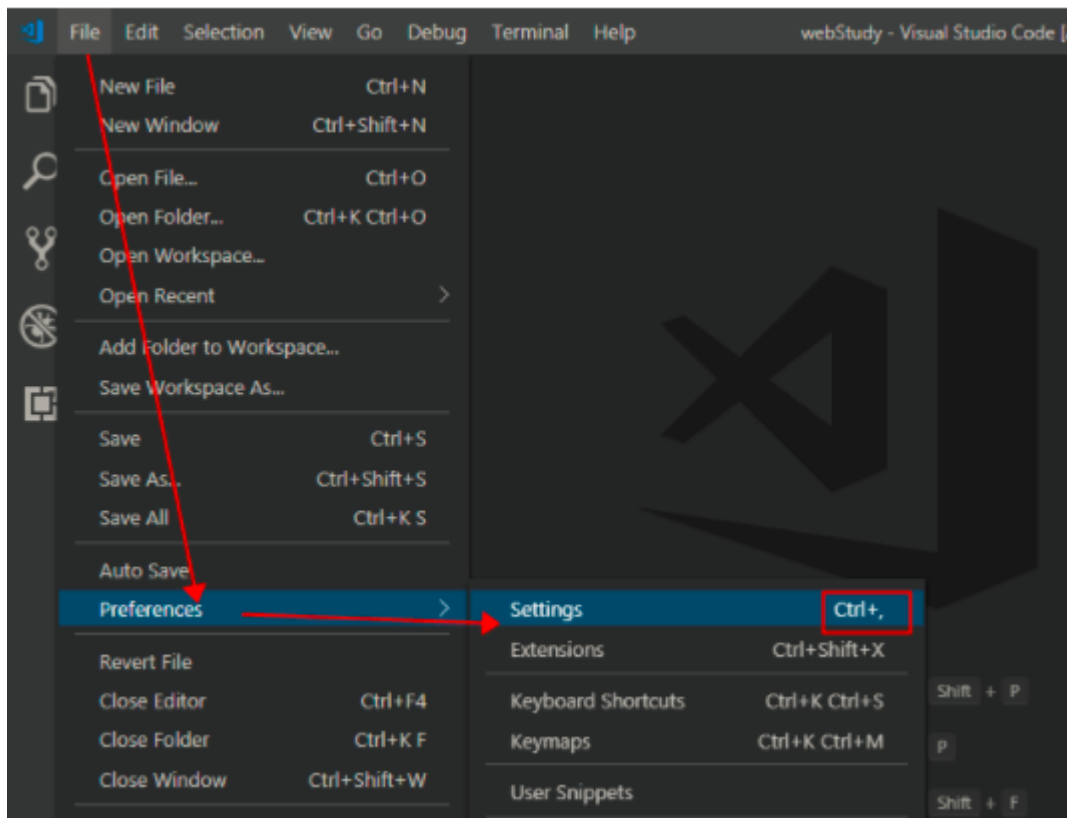
设置

1. 开启自动保存文件（不需要手动按 `Ctrl + S` 保存）



2. 字体大小设置，点击如图菜单：

打开 Settings 快捷键: `ctrl + ,`



快捷键

官方快捷键: <https://code.visualstudio.com/docs/getstarted/keybindings>

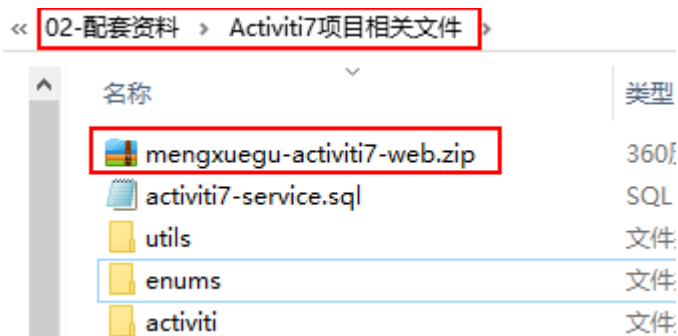
快捷键pdf文档: <https://code.visualstudio.com/shortcuts/keyboard-shortcuts-windows.pdf>

| 快捷键 | 作用 |
|-----------------------|---|
| Ctrl + Shift + F | 会激活这个工具栏的全局搜索功能 |
| Ctrl + F | 局部搜索, 搜索当前文件中的内容 |
| Ctrl + G | 输入行号可以跳转到指定的行 |
| ! + Tab | 空白html文件里输入一个英文感叹号 <code>!</code> , 然后按 <code>Tab</code> 键会生成html模板页面 |
| 元素或属性名+Tab | 会自动生成 |
| Shift + Alt + F | 格式化代码 |
| ctrl+/ shift+alt+A | 单行注释 多行注释 |
| alt+up/down | 移动行 |
| shift + alt +up/down | 复制当前行 |
| ctrl + b | 显示/隐藏左侧工具栏 |
| shift + ctrl + k | 删除当前行 |
| ctrl + x | 剪切当前行或剪切选中内容 |
| ctrl + ~ | 控制台终端显示与隐藏 |
| F2 | 重命名文件名 |

导入前端项目 mengxuegu-activiti7-web

1. vscode导入mengxuegu-activiti7-web

找到 02-配套资料\Activiti7项目相关文件\mengxuegu-activiti7-web.zip 进行解压



2. 下载依赖

```
npm install
```

下载如果有警告或错误，直接忽略，先运行下面命令看是否正常启动，可以启动即可。

如果报错，将错误复制百度找解决方案。

3. 运行

```
npm run dev
```

4. 访问 <http://localhost:9528>

Activiti7 项目后端环境搭建

创建数据库和导入业务表

1. 创建数据库 `activiti7-service`

2. 导入 5 张项目业务表

- 找到sql文件 `02-配套资料\Activiti7项目相关文件\activiti7-service.sql`，导入到数据库中

`sys_user` 用户表的密码均为：`123456`

```
mxg_business_status
mxg_leave
mxg_loan
mxg_process_config
sys_user
```

创建 `mengxuegu-activiti7-service` 模块

IDEA 创建 `mengxuegu-activiti7-service` 模块

添加依赖坐标 `pom.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mengxuegu</groupId>
  <artifactId>mengxuegu-activiti7-service</artifactId>
  <version>1.0-SNAPSHOT</version>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.0</version>
    <relativePath/>
  </parent>
```

```
<properties>
  <activiti.version>7.1.0.M5</activiti.version>
  <mybatis-plus.version>3.3.1</mybatis-plus.version>
</properties>

<dependencies>
  <!-- 依赖管理-->
  <dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-dependencies</artifactId>
    <version>${activiti.version}</version>
    <type>pom</type>
  </dependency>

  <!-- Activiti -->
  <dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-spring-boot-starter</artifactId>
    <version>${activiti.version}</version>
    <!--上面用了mybatis-plus, 要排除这个-->
    <exclusions>
      <exclusion>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
  <!-- java代码绘activiti流程图 -->
  <dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-image-generator</artifactId>
    <version>${activiti.version}</version>
  </dependency>
  <!-- activiti json转换器-->
  <dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-json-converter</artifactId>
    <version>${activiti.version}</version>
  </dependency>
  <!-- svn转换png图片工具 -->
  <dependency>
    <groupId>org.apache.xmlgraphics</groupId>
    <artifactId>batik-all</artifactId>
    <version>1.10</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
  <!-- web启动器 -->
  <dependency>

    <groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<!--mybatis-plus启动器-->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>${mybatis-plus.version}</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.75</version>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <scope>provided</scope>
</dependency>

<!-- swagger 接口文档-->
<dependency>
    <groupId>com.spring4all</groupId>
    <artifactId>swagger-spring-boot-starter</artifactId>
    <version>1.9.1.RELEASE</version>
</dependency>
<!-- springboot2.3.1后要单独引入，swagger才可以用 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>

</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

```
<resources>
  <resource>
    <!--
      编译时，默认情况下不会将 mapper.xml文件编译进去，
      src/main/java 资源文件的路径，
      **/*.xml 需要编译打包的文件类型是xml文件，
    -->
    <directory>src/main/java</directory>
    <includes>
      <include>**/*.xml</include>
    </includes>
  </resource>
  <resource>
    <directory>src/main/resources</directory>
  </resource>
</resources>
</build>

</project>
```

创建配置 application.yml

创建配置 application.yml，注意：

- 更改数据库名：activiti7-service
- 关于包名：entities 和 mapper 以自己的完整包名为准

```
server:
  port: 9090
  servlet:
    context-path: /workflow

spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://127.0.0.1:3306/activiti7-service?
nullCatalogMeansCurrent=true&useUnicode=true&characterEncoding=utf8&useSSL=false&serverTimezone=
GMT%2B8&allowMultiQueries=true
    username: root
    password: root

# activiti配置
activiti:
  #自动更新数据库结构
  # true: 适用开发环境，默认值。activiti会对数据库中所有表进行更新操作。如果表不存在，则自动创建
  # false: 适用生产环境。activiti在启动时，对比数据库中保存的版本，如果没有表或者版本不匹配，将抛出
异常
  # create_drop: 在activiti启动时创建表，在关闭时删除表（必须手动关闭引擎，才能删除表）
  # drop-create: 在activiti启动时删除原来的旧表，然后在创建新表（不需要手动关闭引擎）

  database-schema-update: true
```

```
# activiti7与springboot整合后默认不创建历史表，需要手动开启
db-history-used: true
# 记录历史等级 可配置的历史级别有none, activity, audit, full
# none: 不保存任何的历史数据，因此，在流程执行过程中，这是最高效的。
# activity: 级别高于none，保存流程实例与流程行为，其他数据不保存。
# audit: 除activity级别会保存的数据外，还会保存全部的流程任务及其属性。
# full: 保存历史数据的最高级别，除了会保存audit级别的数据外，还会保存其他全部流程相关的细节数据，
包括一些流程参数等。
history-level: full
# 是否自动检查resources下的processes目录的流程定义文件
check-process-definitions: false
# smtp服务器地址
mail-server-host: smtp.qq.com
# SSL端口号
mail-server-port: 465
# 开启ssl协议
mail-server-use-ssl: true
# 默认的邮件发送地址（发送人），如果activiti流程定义中没有指定发送人，则取这个值
mail-server-default-from: w736486962@qq.com
# 邮件的用户名
mail-server-user-name: w736486962@qq.com
# qq的smtp服务相关的授权码
mail-server-password: xxx填写自己qq邮箱的smtp授权码
# 关闭不自动添加部署数据 SpringAutoDeployment
deployment-mode: never-fail

# 日志级别是debug才能显示SQL日志
logging:
  level:
    org.activiti.engine.impl.persistence.entity: debug
    # 注意：写自己的 mapper 所在完整包名
    com.mengxuegu.workflow.mapper: debug

mybatis-plus:
  type-aliases-package: com.mengxuegu.workflow.entities
  # xxxMapper.xml 路径
  mapper-locations: classpath*:com/mengxuegu/workflow/mapper/**/*.xml
```

-  第14章05节a-创建基于Activiti7-M5版的模块.trec (264.68M/330.85M, 会话超时, 请重新登录)
 重试  删除
-  第14章02节a-Activiti7整合SpringSecurity.trec (0B/231.76M, 会话超时, 请重新登录)
 重试  删除
-  第14章01节a-介绍 Activiti7 新特性API.trec (82.37M, 中转站内部错误-2)
 重试  删除
-  第13章06节a-测试邮件任务流程.trec (0B/208.18M, 会话超时, 请重新登录)
 重试  删除
-  第12章07节a-什么是包含网关.trec (0B/53.16M, 会话超时, 请重新登录)
 重试  删除
-  第12章04节a-并行网关概要和应用场景.trec (0B/66.74M, 会话超时, 请重新登录)
 重试  删除
-  第12章08节a-绘制包含网关流程定义模型.trec (0B/96.67M, 会话超时, 请重新登录)
 重试  删除
-  第13章02节a-Activiti边界定时器事件实现.trec (0B/499.78M)
 删除 文件扫描中36%
-  第12章03节a-完成排他网关审批流程.trec (304.98M/381.22M, 等待上传)
 删除
-  第12章01节a-排他网关流程定义模型绘制.trec (142.24M/177.80M, 等待上传)
 删除
-  第12章02节a-解决更新流程定义Key和部署启动排他网关流程实例.trec (254.57M, 等待上传)
 删除

整合 MyBatis-plus 分页插件

创建 MyBatis-plus 配置类 `com.mengxuegu.workflow.config.MyBatisPlusConfig`，将分页插件添加到容器

```
package com.mengxuegu.workflow.config;

import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.transaction.annotation.EnableTransactionManagement;

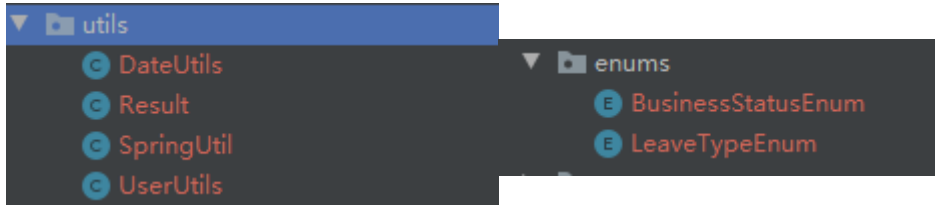
@EnableTransactionManagement // 开启事务管理
@MapperScan("com.mengxuegu.workflow.mapper") // 扫描mapper接口
@Configuration
public class MyBatisPlusConfig {

    /**
     * 分页插件
     * @return
     */
    @Bean
    public PaginationInterceptor paginationInterceptor() {
```

```
        return new PaginationInterceptor();  
    }  
  
}
```

添加工具类和业务枚举类

将 02-配套资料/Activiti7项目相关文件/ 目录下的 enums 和 utils 拷贝到 com.mengxuegu.workflow 包下



创建启动类和开启swagger接口文档

创建启动类 com.mengxuegu.workflow.WorkFlowApplication

- @EnableSwagger2Doc 开启swagger接口文档，后面需要使用接口文档地址（未认证会被拦截，登录后才可访问，后面完善）：
<http://localhost:9090/workflow/swagger-ui.html>
user/密码控制台打印的

```
package com.mengxuegu.workflow;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@EnableSwagger2Doc // 开启swagger接口文档  
@SpringBootApplication  
public class WorkFlowApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(WorkFlowApplication.class, args);  
    }  
  
}
```

启动项目和完善字段

运行启动类，启动项目

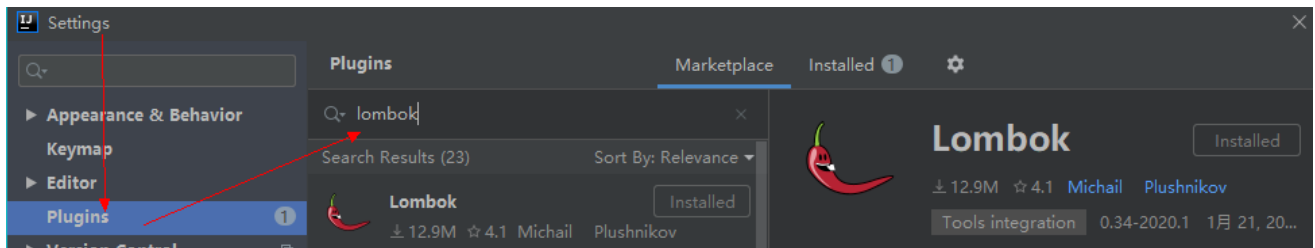
- 启动后，查看数据库是否自动创建 activiti 相关数据表

- 向 act_re_deployment 表增加字段：VERSION_、PROJECT_RELEASE_VERSION_

```
ALTER TABLE ACT_RE_DEPLOYMENT ADD COLUMN VERSION_ VARCHAR(255);  
ALTER TABLE ACT_RE_DEPLOYMENT ADD COLUMN PROJECT_RELEASE_VERSION_ VARCHAR(255);
```

整合 SpringSecurity 完成身份认证

settings中安装 Lombok 插件，使用lombok的 @Data 等注解才不会报错，编译后才有setter, getter方法。



创建 SysUser 实体类

创建 com.mengxuegu.workflow.entities.SysUser

```
package com.mengxuegu.workflow.entities;  
  
import com.baomidou.mybatisplus.annotation.IdType;  
import com.baomidou.mybatisplus.annotation.TableField;  
import com.baomidou.mybatisplus.annotation.TableId;  
import com.baomidou.mybatisplus.annotation.TableName;  
import io.swagger.annotations.ApiModel;  
import io.swagger.annotations.ApiModelProperty;  
import lombok.Data;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
  
import java.util.Collection;  
import java.util.Set;  
  
/**  
 * @Author: 梦学谷 www.mengxuegu.com  
 */  
@Data  
@ApiModel("用户表")  
@TableName("sys_user")  
public class SysUser implements UserDetails {  
  
    @TableId(value = "id", type = IdType.ASSIGN_ID)  
    private String id;  
  
    @ApiModelProperty("用户名")  
    private String username;
```

```
@ApiModelProperty("密码需要通过加密后存储，默认：123456")
private String password;

@ApiModelProperty("昵称")
private String nickName;

@ApiModelProperty("头像")
private String imageUrl;

@ApiModelProperty("封装用户权限")
@TableField(exist = false) // 不是表中字段
private Set<GrantedAuthority> authorities;

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return authorities;
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return true;
}
}
```

创建SysUserMapper接口

创建SysUserMapper接口 `com.mengxuegu.workflow.mapper.SysUserMapper`

```
package com.mengxuegu.workflow.mapper;  
  
import com.baomidou.mybatisplus.core.mapper.BaseMapper;  
import com.mengxuegu.workflow.entities.SysUser;  
  
public interface SysUserMapper extends BaseMapper<SysUser> {  
  
}
```

创建SysUserMapper.xml文件

创建 SysUserMapper.xml 文件 `com.mengxuegu.workflow.mapper.xml.SysUserMapper.xml`

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
<mapper namespace="com.mengxuegu.workflow.mapper.SysUserMapper">  
  
</mapper>
```

创建 ISysUserService 接口

创建 ISysUserService 接口: `com.mengxuegu.workflow.service.ISysUserService`

```
package com.mengxuegu.workflow.service;  
  
import com.baomidou.mybatisplus.extension.service.IService;  
import com.mengxuegu.workflow.entities.SysUser;  
  
public interface ISysUserService extends IService<SysUser> {  
  
    SysUser findByUsername(String username);  
}
```

创建 SysUserService 实现类

创建 ISysUserService 接口的 SysUserService 实现类 `com.mengxuegu.workflow.service.impl.SysUserService`

- 类上 `@Service` 注解不要少了

```
package com.mengxuegu.workflow.service.impl;
```

```
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.mengxuegu.workflow.entities.SysUser;
import com.mengxuegu.workflow.mapper.SysUserMapper;
import com.mengxuegu.workflow.service.ISysUserService;
import org.apache.commons.lang3.StringUtils;
import org.springframework.stereotype.Service;

@Service // 不要少了
public class SysUserService extends ServiceImpl<SysUserMapper, SysUser> implements
ISysUserService {

    @Override
    public SysUser findByUsername(String username) {
        if(StringUtils.isEmpty(username)) {
            return null;
        }

        QueryWrapper<SysUser> wrapper = new QueryWrapper();
        wrapper.eq("username", username);

        // baseMapper 对应的是就是 SysUserMapper
        return baseMapper.selectOne(wrapper);
    }
}
```

创建 CustomUserDetailsService

创建 CustomUserDetailsService 实现 UserDetailsService，SpringSecurity 认证时会调用此类来进行校验用户是否存在，存在返回用户信息。

完整类名：`com.mengxuegu.workflow.security.CustomUserDetailsService`

```
package com.mengxuegu.workflow.security;

import com.mengxuegu.workflow.entities.SysUser;
import com.mengxuegu.workflow.service.ISysUserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Component;

import java.util.HashSet;
import java.util.Set;

@Component("customUserDetailsService")
```

```
public class CustomUserDetailsService implements UserDetailsService {

    @Autowired
    private ISysUserService sysUserService;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        SysUser sysUser = sysUserService.findByUsername(username);
        if(sysUser == null) {
            throw new UsernameNotFoundException("用户名或密码错误");
        }

        // 用户拥有角色 ACTIVITI_USER, 候选组 MANAGER_TEAM
        Set<GrantedAuthority> authorities = new HashSet<>();
        authorities.add(new SimpleGrantedAuthority("ROLE_ACTIVITI_USER"));
        authorities.add(new SimpleGrantedAuthority("GROUP_MANAGER_TEAM"));
        sysUser.setAuthorities(authorities);
        return sysUser;
    }
}
```

创建认证成功处理器

1. 认证成功后，响应JSON字符串给前端

com.mengxuegu.workflow.security.CustomAuthenticationSuccessHandler

```
package com.mengxuegu.workflow.security;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.mengxuegu.workflow.utils.Result;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.SavedRequestAwareAuthenticationSuccessHandler;
import org.springframework.stereotype.Component;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 认证成功处理器 响应json
 * @Author: 梦学谷 www.mengxuegu.com
 */
@Component

public class CustomAuthenticationSuccessHandler extends
```

```
SavedRequestAwareAuthenticationSuccessHandler {

    @Autowired
    private ObjectMapper objectMapper;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
                                       HttpServletResponse response, Authentication
authentication) throws IOException, ServletException {

        // 认证成功后，响应JSON字符串
        response.setContentType("application/json;charset=UTF-8");
        response.getWriter().write(objectMapper.writeValueAsString( Result.ok("认证成功")
));
    }

}
```

创建认证失败处理器

1. 认证失败响应JSON字符串给前端

com.mengxuegu.workflow.security.CustomAuthenticationFailureHandler

```
package com.mengxuegu.workflow.security;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.mengxuegu.workflow.utils.Result;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.authentication.SimpleUrlAuthenticationFailureHandler;
import org.springframework.stereotype.Component;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 处理失败认证的，响应json
 * @Author: 梦学谷 www.mengxuegu.com
 */
@Component // 不要少了
public class CustomAuthenticationFailureHandler extends
SimpleUrlAuthenticationFailureHandler {

    @Autowired
    private ObjectMapper objectMapper;
```

```
@Override
public void onAuthenticationFailure(HttpServletRequest request,
                                    HttpServletResponse response,
                                    AuthenticationException exception) throws IOException, ServletException {
    // 认证失败响应JSON字符串,
    Result result = Result.build(HttpStatus.UNAUTHORIZED.value(),
    exception.getMessage());
    response.setContentType("application/json;charset=UTF-8");
    response.getWriter().write(objectMapper.writeValueAsString(result));
}
}
```

创建退出成功处理器

```
package com.mengxuegu.workflow.security;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.mengxuegu.workflow.utils.Result;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.logout.LogoutSuccessHandler;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 退出成功处理器:响应json
 */
@Component
public class CustomLogoutSuccessHandler implements LogoutSuccessHandler {

    @Autowired
    private ObjectMapper objectMapper;

    @Override
    public void onLogoutSuccess(HttpServletRequest request,
                                HttpServletResponse response,
                                Authentication authentication) throws IOException {
        // 退出成功, 响应结果
        response.setContentType("application/json;charset=UTF-8");
        response.getWriter().write(objectMapper.writeValueAsString( Result.ok("退出成功") ));
    }
}
```

创建认证配置类

创建认证配置类整合成功/失败处理器，和权限拦截等配置。

com.mengxuegu.workflow.security.SpringSecurityConfig

```
package com.mengxuegu.workflow.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.AuthenticationFailureHandler;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import org.springframework.security.web.authentication.logout.LogoutSuccessHandler;

/**
 * @Author: 梦学谷 www.mengxuegu.com
 */
@Configuration
@EnableWebSecurity // 开启 springsecurity 认证配置
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private AuthenticationSuccessHandler authenticationSuccessHandler;

    @Autowired
    private AuthenticationFailureHandler authenticationFailureHandler;

    @Autowired
    private LogoutSuccessHandler logoutSuccessHandler;

    @Autowired
    UserDetailsService customUserDetailsService;

    @Bean
    public PasswordEncoder passwordEncoder() {
        // 明文+随机盐值 加密存储
        return new BCryptPasswordEncoder();
    }
}
```



```
/**
 * 认证管理器：
 * 1. 认证信息（用户名，密码）
 */
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(customUserDetailsService);
}

/**
 * 资源权限配置
 */
@Override
protected void configure(HttpSecurity http) throws Exception {
    // 校验手机验证码过滤器
    http.formLogin() // 表单登录方式
        .loginProcessingUrl("/user/login")
        .successHandler(authenticationSuccessHandler)
        .failureHandler(authenticationFailureHandler)
        .and()
        .logout()
        .logoutUrl("/user/logout") // 退出请求路径
        .logoutSuccessHandler(logoutSuccessHandler)
        .and()
        .authorizeRequests() // 授权请求
        .anyRequest().authenticated()
        .and()
        .csrf().disable() // 关闭跨站请求伪造
    ;// 注意不要少了分号
}
}
```

测试

1. 启动项目
2. 访问 <http://localhost:9090/workflow/login>



A login form titled "Please sign in". It contains a text input field with the value "meng", a password input field with masked characters ".....", and a blue "Sign in" button.

3. 访问swagger 接口文档: <http://localhost:9090/workflow/swagger-ui.html>

如果访问后，弹出如下窗口，说明启动类上没有添加 `@EnableSwagger2Doc`，需要加上它。

localhost:9090 显示

Unable to infer base url. This is common when using dynamic servlet registration or when the API is behind an API Gateway. The base url is the root of where all the swagger resources are served. For e.g. if the api is available at <http://example.org/api/v2/api-docs> then the base url is <http://example.org/api/>. Please enter the location manually:

<http://localhost:9090/workflow/swagger-ui.html>

确定

取消

4. 上面登录页面是 SpringSecurity 提供后，而我们是前后端分离项目，登录页面是前端编写好的。

所以我们要单独测试登录接口 `/user/login`，

上面点击 Sign In 登录，以POST方式提交到 <http://localhost:9090/workflow/user/login>

请求参数名: username、password

打开 `04-配套软件/Postman-win64-5.5.2-.Setup.exe` 测试提交登录

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:9090/workflow/user/login
- Body Type:** form-data
- Body Data:**

| Key | Value |
|----------|--------|
| username | meng |
| password | 123456 |
- Response:**

```
{  "code": 20000,  "message": "操作成功",  "data": "认证成功"}
```

5. <http://localhost:9090/workflow/user/logout>

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:9090/workflow/user/logout
- Authorization:** Inherit auth from parent
- Body Type:** (Not explicitly shown, but the response is JSON)
- Response:**

```
{  "code": 20000,  "message": "操作成功",  "data": "退出成功"}
```

6. 测试前端项目请求登录、退出功能

流程定义模型管理

创建分页请求基础类

注意：getPage 方法是返回 `new Page<T>(current, size)`；需要指定参数，课上没有传，自己加上。

```
package com.mengxuegu.workflow.req;

import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

import java.io.Serializable;

@Data
@ApiModel(value = "分页请求基础类")
public class BaseRequest<T> implements Serializable {

    @ApiModelProperty(value = "页码", required = true)
    private int current;

    @ApiModelProperty(value = "每页显示多少条", required = true)
    private int size;

    /**
     * activiti 分页
     * @return
     */
    public Integer getFirstResult() {
        return (current-1) * size;
    }

    /**
     * mybatis-plus分页
     * @return
     */
    public Page<T> getPage() {
        return new Page<T>(current, size);
    }
}
```

注意：getPage 方法是返回 `new Page<T>(current, size)`；需要指定参数，课上没有传，自己加上。

创建模型请求类

1. 创建一个条件查询模型请求类 com.mengxuegu.workflow.req.ModelREQ

```
package com.mengxuegu.workflow.req;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

@Data
@ApiModel("查询模型请求类")
public class ModelREQ extends BaseRequest {

    @ApiModelProperty("模型名称")
    private String name;

    @ApiModelProperty("模型标识key")
    private String key;

}
```

2. 创建新增模型请求类 com.mengxuegu.workflow.req.ModelAddREQ

```
package com.mengxuegu.workflow.req;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

@Data
@ApiModel("新增模型请求类")
public class ModelAddREQ extends ModelREQ {

    @ApiModelProperty("描述")
    private String description;

}
```

创建Activiti统一服务类

创建 ActivitiService 类，用于统一注入Activiti相关服务接口，然后其他Service继承这个类

com.mengxuegu.workflow.service.impl.ActivitiService

```
package com.mengxuegu.workflow.service.impl;

import com.fasterxml.jackson.databind.ObjectMapper;
```

```
import org.activiti.api.process.runtime.ProcessRuntime;
import org.activiti.api.task.runtime.TaskRuntime;
import org.activiti.engine.*;
import org.springframework.beans.factory.annotation.Autowired;

public class ActivitiService {

    @Autowired
    public ObjectMapper objectMapper;

    @Autowired
    public TaskService taskService;

    @Autowired
    public RuntimeService runtimeService;

    @Autowired
    public HistoryService historyService;

    @Autowired
    public RepositoryService repositoryService;

    /**
     * 内部最终调用repositoryService和runtimeService相关API。
     * 需要ACTIVITI_USER权限
     */
    @Autowired
    public ProcessRuntime processRuntime;

    /**
     * 类内部调用taskService
     * 需要ACTIVITI_USER权限
     */
    @Autowired
    public TaskRuntime taskRuntime;
}
```

新增流程定义模型数据

创建IModelService接口

创建流程定义模型管理服务接口 com.mengxuegu.workflow.service.IModelService

```
package com.mengxuegu.workflow.service;

import com.mengxuegu.workflow.req.ModelAddREQ;
import com.mengxuegu.workflow.utils.Result;

public interface IModelService {

    /**
     * 新增模型基本信息
     */
    Result add(ModelAddREQ req) throws Exception;
}
```

创建 ModelService 实现类

实现新增流程定义模型业务逻辑 com.mengxuegu.workflow.service.impl.ModelService

```
package com.mengxuegu.workflow.service.impl;

import com.fasterxml.jackson.databind.node.ObjectNode;
import com.mengxuegu.workflow.req.ModelAddREQ;
import com.mengxuegu.workflow.service.IModelService;
import com.mengxuegu.workflow.utils.Result;
import org.activiti.editor.constants.ModelDataJsonConstants;
import org.activiti.engine.repository.Model;
import org.springframework.stereotype.Service;

@Service
public class ModelService extends ActivitiService implements IModelService {

    public Result add(ModelAddREQ req) throws Exception {
        //初始化一个空模型
        Model model = repositoryService.newModel();
        model.setName(req.getName());
        model.setKey(req.getKey());
        model.setVersion(0);

        // 封装模型json对象
        ObjectNode modelObjectNode = objectMapper.createObjectNode();
        modelObjectNode.put(ModelDataJsonConstants.MODEL_NAME, req.getName());
        modelObjectNode.put(ModelDataJsonConstants.MODEL_REVISION, 0);
        modelObjectNode.put(ModelDataJsonConstants.MODEL_DESCRIPTION, req.getDescription());
        model.setMetaInfo(modelObjectNode.toString());
        // 存储模型对象 (表 ACT_RE_MODEL )
        repositoryService.saveModel(model);

        // 封装模型对象基础数据json串 {"id":"canvas","resourceId":"canvas","stencilset":
        {"namespace":"http://b3mn.org/stencilset/bpmn2.0#"},"properties":{"process_id":"未定义"}}
        ObjectNode editorNode = objectMapper.createObjectNode();
        //editorNode.put("resourceId", "canvas");
    }
}
```

```
ObjectNode stencilSetNode = objectMapper.createObjectNode();
stencilSetNode.put("namespace", "http://b3mn.org/stencilset/bpmn2.0#");
editorNode.replace("stencilset", stencilSetNode);
// 标识key
ObjectNode propertiesNode = objectMapper.createObjectNode();
propertiesNode.put("process_id", req.getKey());
editorNode.replace("properties", propertiesNode);

// 存储模型对象基础数据 (表 ACT_GE_BYTEARRAY )
repositoryService.addModelEditorSource(model.getId(),
editorNode.toString().getBytes("utf-8"));
return Result.ok(model.getId());
}
}
```

创建流程定义模型控制类 ModelController

```
package com.mengxuegu.workflow.controller;

import com.mengxuegu.workflow.req.ModelAddREQ;
import com.mengxuegu.workflow.service.IModelService;
import com.mengxuegu.workflow.utils.Result;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@Api("流程定义模型管理")
@Slf4j
@RestController
@RequestMapping("/model")
public class ModelController {

    @Autowired
    IModelService modelService;

    @ApiOperation("新增流程定义模型数据")
    @PostMapping
    public Result add(@RequestBody ModelAddREQ req) {
        try {
            return modelService.add(req);
        } catch (Exception e) {
            e.printStackTrace();
            log.error("创建模型失败: " + e.getMessage());

            return Result.error("创建模型失败");
        }
    }
}
```



```
}  
}  
  
}
```

测试

1. 重启项目
2. 访问Swagger接口文档: <http://localhost:9090/workflow/swagger-ui.html#/>
访问后会跳转登录页，进行登录后，再重新访问上面地址
3. 找到如下接口，添加数据进行测试

POST **/model** 新增流程定义模型数据

Parameters

| Name | Description |
|---------------------------------|---|
| req * required (body) | req |
| | <div>Example Value Model</div> <div><pre>{ "description": "梦学谷测试流程 ", "key": "test", "name": "测试流程" }</pre></div> |

Server response

| Code | Details |
|------|--|
| 200 | <div>Response body</div> <div><pre>{ "code": 20000, "message": "操作成功", "data": "3d47578f-e154-11eb-b424-2c337a6d7e1d" }</pre></div> |

4. 查看数据库 act_re_model表数据

| 对象 | act_re_model @activiti7-ser... | | | |
|-------------------|--------------------------------|-------|------|-----------|
| 开始事务 | 备注 | 筛选 | 排序 | 导入 导出 |
| ID_ | REV_ | NAME_ | KEY_ | CATEGORY_ |
| 3d47578f-e154-11e | 2 | 测试流程 | test | (Null) |

5. 在前端项目测试，提交数据

新增空白流程模型

* 模型名称: 测试流程模型2 7/20

* 标识Key: test2 5/20

备注: 测试流程模型2

提交 取消

| 开始事务 | 备注 | 筛选 | 排序 | 导入 |
|-------------------|------|-------------|------|----|
| ID_ | REV_ | NAME_ | KEY_ | |
| 3d47578f-e154-11e | 2 | 测试流程 | test | |
| 945542e1-e154-11e | 2 | 测试流程模型test2 | | |

注意:

提交后，如果页面提示：Network Error，则退出后再重新登录，再进行新增操作。

提交成功后，页面会有两个提示：

- 1、提交成功
- 2、提示404（原因：提交成功会调用查询模型列表接口，这个接口目前没有开发，所以提示404）

条件分页查询流程定义模型列表

IModelService 添加查询抽象方法

```
/**
 * 条件分页查询流程定义模型列表数据
 */
Result getModelList(ModelReq req);
```

ModelService 类中实现查询功能

```
public Result getModelList(ModelREQ req) {
    ModelQuery query = repositoryService.createModelQuery();
    if(StringUtils.isEmpty(req.getName())) {
        query.modelNameLike("%"+req.getName()+"%");
    }

    if(StringUtils.isEmpty(req.getKey())) {
        query.modelKey(req.getKey());
    }

    // 按创建时间降序排列
    query.orderByCreateTime().desc();
    // 开始查询 (第几条开始, 查询多少条)
    List<Model> modelList = query.listPage(req.getFirstResult(), req.getSize());

    // 用于前端显示页面, 总记录数
    long total = query.count();

    // 转换结果给前端展示
    List<Map<String, Object>> records = new ArrayList<>();
    for (Model model : modelList) {
        HashMap<String, Object> data = new HashMap<>();
        data.put("id", model.getId()); // 模型id
        data.put("name", model.getName()); //模型名称
        data.put("key", model.getKey()); //
        data.put("version", model.getVersion()); //版本号
        // 模型描述
        String desc = JSONObject.parseObject(model.getMetaInfo())
            .getString(ModelDataJsonConstants.MODEL_DESCRIPTION);
        data.put("description", desc);
        //创建时间
        data.put("createDate", DateUtils.format(model.getCreateTime()));
        //更新时间
        data.put("updateDate", DateUtils.format(model.getLastUpdateTime()));
        records.add(data);
    }
    // 封闭响应结果
    Map<String, Object> result = new HashMap<>();
    result.put("total", total);
    result.put("records", records);
    return Result.ok(result);
}
```

ModelController 添加查询API方法

```
@ApiOperation("条件分页查询流程定义模型列表数据")
@PostMapping("/list")
public Result modelList(@RequestBody ModelREQ req) {
    try {
        return modelService.getModelList(req);
    } catch (Exception e) {
        e.printStackTrace();
        log.error("查询流程定义模型列表数据失败: " + e.getMessage());
        return Result.error("查询模型列表失败");
    }
}
```

测试

1. 重启项目
2. 在前端项目测试，请求：<http://localhost:9528/#/workflow/model>

首页

工作流程

模型管理

流程管理

+ 新增流程模型

| 序号 | 模型名称 | 标识Key | 版本号 | 备注说明 |
|----|---------|-------|------|---------|
| 1 | 测试流程模型2 | test2 | v0.0 | 测试流程模型2 |
| 2 | 测试流程 | test | v0.0 | 梦学谷测试流程 |

注意：

提交后，如果页面提示：Network Error，则退出后再重新登录，再进行新增操作

处理未认证响应302状态码问题

注意：前端发送请求时，如果页面提示：Network Error，

原因：是重启后台项目 Session失效了，你再请求需要认证才可以访问的接口，SpringSecurity 会默认重定向到 AuthenticationEntryPoint 接口响应302状态。而前端无法捕获到 302，从而前端无法跳转到登录页面。

首页

工作流程

模型管理

流程管理

Dashboard / 工作流程 / 模型管理

Network Error

模型名称 标识Key

+ 新增流程模型

| 序号 | 模型名称 | 标识Key | 版本号 | 备注说明 |
|----|------|-------|-----|------|
|----|------|-------|-----|------|

解决：前端无法捕获302，我们就重新实现响应JSON 字符串 {code: 50008}，让前端接收处理。

1. 创建 com.mengxuegu.workflow.security.CustomAuthenticationEntryPoint

```
package com.mengxuegu.workflow.security;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.mengxuegu.workflow.utils.Result;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 在前后端分离项目中，使用 SpringSecurity 后在未登录的情况下，请求后台需要认证后才可以访问的接口，
 * SpringSecurity 会默认重定向到 AuthenticationEntryPoint 接口响应302状态。
 * 而前端无法捕获302，所以我们重新实现响应JSON，让前端接收处理。
 */
@Component
public class CustomAuthenticationEntryPoint implements AuthenticationEntryPoint {

    @Autowired
    ObjectMapper objectMapper;

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response,
        AuthenticationException authException) throws IOException, ServletException {
        // 未认证JSON字符串，
        Result result = Result.build(50008, "请先登录再访问!");
        response.setContentType("application/json;charset=UTF-8");
        response.getWriter().write(objectMapper.writeValueAsString(result));
    }
}
```

1. 在 com.mengxuegu.workflow.security.SpringSecurityConfig 操作如下：

- 注入 AuthenticationEntryPoint 实例
- configure(HttpSecurity http) 方法中绑定实例

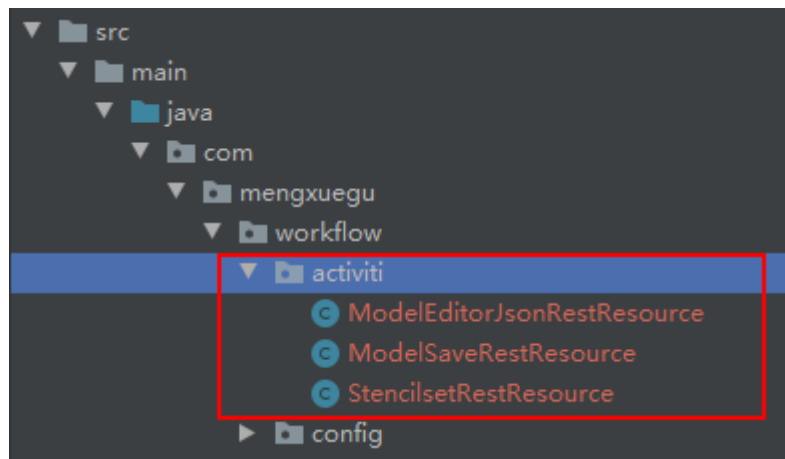
```
@Autowired
private AuthenticationEntryPoint authenticationEntryPoint;
```

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    // 校验手机验证码过滤器
    http.formLogin() // 表单登录方式
        .loginProcessingUrl("/user/login")
        .successHandler(authenticationSuccessHandler)
        .failureHandler(authenticationFailureHandler)
        .and()
        .logout()
        .logoutUrl("/user/logout") // 退出请求路径
        .logoutSuccessHandler(logoutSuccessHandler)
        .and()
        .authorizeRequests() // 授权请求
        // 其他请求，要通过认证才可以访问
        .anyRequest().authenticated()
        .and()
        .csrf().disable() // 关闭跨站请求伪造
        .exceptionHandling() // 处理前端无法捕获302 ++++++
        .authenticationEntryPoint(authenticationEntryPoint)
    ;// 注意不要少了分号
}
```

2. 测试：启动项目，访问前端，发现认证过期跳转到登录页，

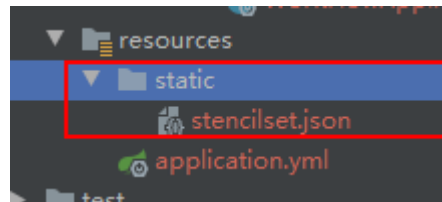
后台整合流程模型设计器 Activiti Modeler

1. 拷贝 Activiti 设计器的后台相关处理类到 `mengxuegu-activiti7-service` 项目中



2. 在 `mengxuegu-activiti7-service` 项目的 `resources` 下创建 `static` 目录，然后将汉化文件 `stencilset.json` 拷贝到 `resources\static` 目录下

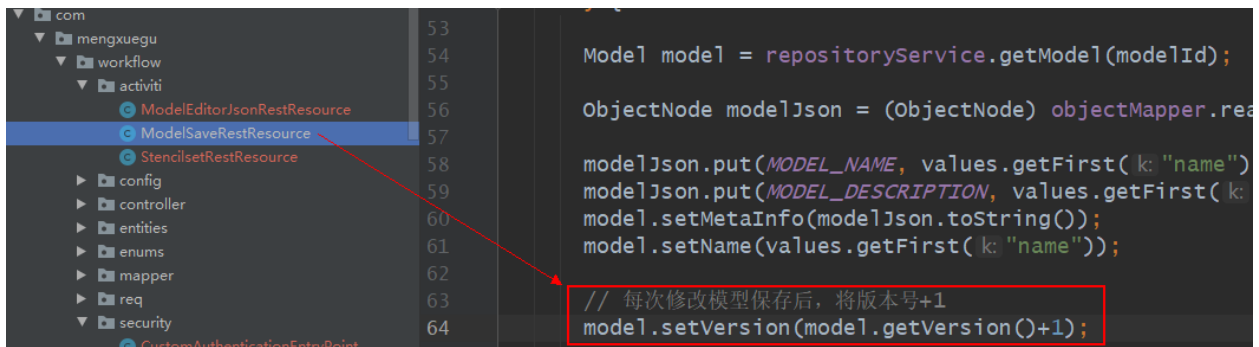
文件位于：02-配套资料\activiti modeler模型设计器\modeler汉化\stencilset.json



3. 重新设计流程模型保存后，将版本号+1

com.mengxuegu.workflow.activiti.ModelSaveRestResource#saveModel

```
// 每次修改模型保存后，将版本号+1
model.setVersion(model.getVersion()+1);
```



4. 重启项目, 点击 设计流程, 是否可以成功打开设计页面, 简单设计后保存, 查看版本号是+1了

| 操作 | |
|----|--------------------|
| 4 | 设计流程 部署流程 导出ZIP 删除 |
| 3 | 设计流程 部署流程 导出ZIP 删除 |

Activiti工作流在线流程设计器



设计流程定义模型-请假流程模型

1. 新增请假申请流程模型

新增空白流程模型

* 模型名称:

请假流程模型

60

* 标识Key:

leaveProcess

120

备注:

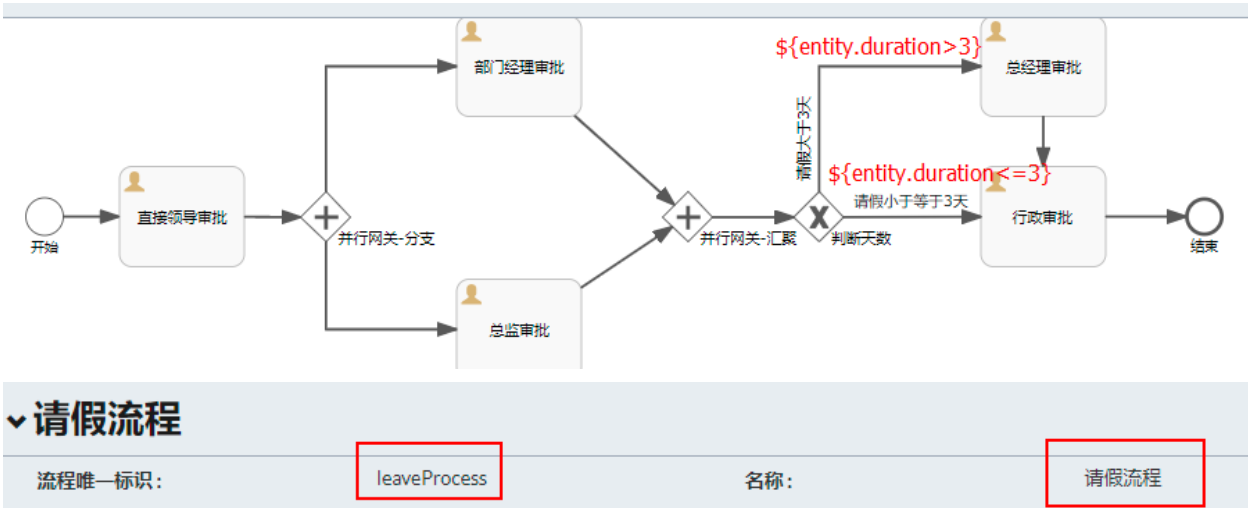
请假流程模型

提交

取消

2. 设计请假申请流程模型

- 注意：
- 排他网关出口连线上指定判断天数表达式，
 - 人工任务节点上不用指定办理人或者候选人，后面会动态分配



3. 保存后，查看版本号+1

| 序号 | 模型名称 | 标识Key | 版本号 | 备注说明 |
|----|--------|--------------|------|--------|
| 1 | 请假流程模型 | leaveProcess | v1.0 | 请假流程模型 |

部署流程定义

IModelService 添加部署流程抽象方法

```
/**
 * 通过流程定义模型ID部署流程定义
 * @throws Exception
 */
Result deploy(String modelId) throws Exception;
```

ModelService 类中实现部署流程

```
/**
 * 通过流程定义模型ID部署流程定义
 * @throws Exception
 */
public Result deploy(String modelId) throws Exception {
    // 1. 查询流程定义模型json字节码
    byte[] jsonBytes = repositoryService.getModelEditorSource(modelId);
    if(jsonBytes == null) {
        return Result.error("模型数据为空，请先设计流程定义模型，再进行部署");
    }
    // 将json字节码转为 xml 字节码，
    byte[] xmlBytes = bpmnJsonXmlBytes(jsonBytes);
    if(xmlBytes == null) {
        return Result.error("数据模型不符合要求，请至少设计一条主线程");
    }
    // 2. 查询流程定义模型的图片
    byte[] pngBytes = repositoryService.getModelEditorSourceExtra(modelId);

    // 查询模型的基本信息
    Model model = repositoryService.getModel(modelId);

    // xml资源的名称，对应act_ge_bytearray表中的name_字段
    String processName = model.getName() + ".bpmn20.xml";
    // 图片资源名称，对应act_ge_bytearray表中的name_字段
    String pngName = model.getName() + "." + model.getKey() + ".png";

    // 3. 调用部署相关的api方法进行部署流程定义
    Deployment deployment = repositoryService.createDeployment()
        .name(model.getName()) // 部署名称
        .addString(processName, new String(xmlBytes, "UTF-8")) // bpmn20.xml资源
        .addBytes(pngName, pngBytes) // png资源
        .deploy();

    // 更新 部署id 到流程定义模型数据表中
    model.setDeploymentId(deployment.getId());
}
```

```
repositoryService.saveModel(model);

return Result.ok();
}

private byte[] bpmnJsonXmlBytes(byte[] jsonBytes) throws IOException {
    if(jsonBytes == null) {
        return null;
    }

    // 1. json字节码转成 BpmnModel 对象
    JsonNode jsonNode = objectMapper.readTree(jsonBytes);
    BpmnModel bpmnModel = new BpmnJsonConverter().convertToBpmnModel(jsonNode);

    if(bpmnModel.getProcesses().size() == 0) {
        return null;
    }
    // 2. BpmnModel 对象转为xml字节码
    byte[] xmlBytes = new BpmnXMLConverter().convertToXML(bpmnModel);
    return xmlBytes;
}
```

ModelController 添加部署API方法

```
@ApiOperation("通过流程定义模型ID部署流程定义")
@PostMapping("/deploy/{modelId}")
public Result deploy(@PathVariable("modelId")String modelId) {
    try {
        return modelService.deploy(modelId);
    } catch (Exception e) {
        e.printStackTrace();
        log.error("通过流程定义模型ID部署流程定义失败: " + e.getMessage());
        return Result.error("部署流程定义失败");
    }
}
```

测试

1. 重启后台项目
2. 点击 请假流程模型 数据的 部署流程 按钮,



3. 如果报错如下:

```
SQL: insert into ACT_RE_DEPLOYMENT(ID_, NAME_, CATEGORY_, KEY_, TENANT_ID_, DEPLOY_TIME_, ENG
Cause: java.sql.SQLException: Unknown column 'VERSION_' in 'field list'
at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
at java.base/java.lang.Thread.run(Thread.java:844)
```

解决如下，再重启点击 部署流程

```
ALTER TABLE ACT_RE_DEPLOYMENT ADD COLUMN VERSION_ VARCHAR(255);
ALTER TABLE ACT_RE_DEPLOYMENT ADD COLUMN PROJECT_RELEASE_VERSION_ VARCHAR(255);
```

4. 观察数据库部署表中是否有部署数据

| ID_ | REV_ | CATEGORY_ | NAME_ | KEY_ | VERSION_ | DEPLOYMENT_ID_ | RESOURCE_NAME_ | DGRM_RESOURCE_ |
|----------------------|------|-----------------|-------|----------|----------|---------------------|-------------------------|----------------|
| leaveProcess:1:baafc | 1 | http://www.acti | 请假流程 | leavePro | | 1 baa453ce-e165-11e | 请假流程模型.bpmn;请假流程模型.leav | |

导出流程定义模型zip压缩包

IModelService 添加导出模型zip抽象方法

```
/**
 * 导出模型图zip压缩包(.bpmn20.xml流程图和.png图片资源)
 */
void exportZip(String modelId, HttpServletResponse response);
```

ModelService 类中实现导出模型zip压缩包

```
/**
 * 导出流程定义模型zip压缩包(.bpmn20.xml流程图和.png图片资源)
 */
public void exportZip(String modelId, HttpServletResponse response){
    ZipOutputStream zipos = null;
    try {
        // 实例化zip压缩对象输出流
```

```
zipos = new ZipOutputStream(response.getOutputStream());
// 压缩包文件名
String zipName = "模型不存在";

// 1. 查询模型基本信息
Model model = repositoryService.getModel(modelId);
if(model != null) {
    // 2. 查询流程定义模型的json字节码
    byte[] bpmnJsonBytes = repositoryService.getModelEditorSource(modelId);
    // 2.1 将json字节码转换为xml字节码
    byte[] xmlBytes = bpmnJsonXmlBytes(bpmnJsonBytes);
    if(xmlBytes == null) {
        zipName = "模型数据为空-请先设计流程定义模型，再导出";
    }else {
        // 压缩包文件名
        zipName = model.getName() + "." + model.getKey() + ".zip";
        // 将xml添加到压缩包中(指定xml文件名: 请假流程.bpmn20.xml )
        zipos.putNextEntry(new ZipEntry(model.getName() + ".bpmn20.xml"));
        // 将xml写入压缩流
        zipos.write(xmlBytes);

        // 3. 查询流程定义模型的图片字节码
        byte[] pngBytes = repositoryService.getModelEditorSourceExtra(modelId);
        if(pngBytes != null) {
            // 图片文件名 (请假流程.leaveProcess.png)
            zipos.putNextEntry(
                new ZipEntry(model.getName() + "." + model.getKey() + ".png")
            );
            zipos.write(pngBytes);
        }
    }
}

response.setContentType("application/octet-stream");
// 防止中文名乱码, 要编码
response.setHeader("Content-Disposition",
    "attachment; filename=" + URLEncoder.encode(zipName, "UTF-8") + ".zip");
// 刷出响应流
response.flushBuffer();
} catch (IOException e) {
    e.printStackTrace();
}finally {
    if(zipos != null) {
        try {
            zipos.closeEntry();
            zipos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ModelController 添加部署API方法

```
@ApiOperation("导出流程定义模型zip压缩包")
@GetMapping("/export/zip/{modelId}")
public void exportZip(@PathVariable("modelId") String modelId,
    HttpServletResponse response) {
    modelService.exportZip(modelId, response);
}
```

测试

1. 重启后台项目
2. 前端重新登录
3. 点击 **导出ZIP**，会导出一个zip压缩包



删除流程定义模型

1. 在 ModelController 添加API方法 `deleteModel`：删除流程定义模型

```
@Autowired
RepositoryService repositoryService;

@ApiOperation("删除流程定义模型")
@DeleteMapping("/{modelId}")
public Result deleteModel(@PathVariable("modelId") String modelId){
    repositoryService.deleteModel(modelId);
    return Result.ok();
}
```

2. 重启项目，重新登录。
3. 点击 **删除** 按钮，会删除流程定义模型（不管此模型有没有被部署过流程定义都可以删除，因为部署好的流程定义，会生成在部署时流程定义的xml和png资源）

设计流程 部署流程 导出ZIP 删除

? 您确定删除【测试流程】吗?

取消

确定

流程定义管理

条件分页查询流程定义数据

查询部署的流程定义数据列表,如果有多个相同标识key的流程时, 只查询其最新版本

创建流程定义请求类

创建查询流程定义条件请求类 com.mengxuegu.workflow.req.ProcDefiREQ

```
package com.mengxuegu.workflow.req;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

@Data
@ApiModel("流程定义请求类")
public class ProcDefiREQ extends BaseRequest {

    @ApiModelProperty("流程名称")
    private String name;

    @ApiModelProperty("标识key")
    private String key;

}
```

创建 IProcessDefinitionService 接口

创建流程定义管理服务接口 com.mengxuegu.workflow.service.IProcessDefinitionService

```
package com.mengxuegu.workflow.service;

import com.mengxuegu.workflow.req.ProcDefiREQ;
```

```
import com.mengxuegu.workflow.utils.Result;

/**
 * 流程定义管理
 */
public interface IProcessDefinitionService {

    /**
     * 条件分页查询已部署的流程定义数据
     */
    Result getProcDefiList(ProcDefiREQ req);

}
```

创建 ProcessDefinitionService 实现类

实现条件分页查询流程定义 com.mengxuegu.workflow.service.impl.ProcessDefinitionService

```
package com.mengxuegu.workflow.service.impl;

import com.mengxuegu.workflow.req.ProcDefiREQ;
import com.mengxuegu.workflow.service.IProcessDefinitionService;
import com.mengxuegu.workflow.utils.DateUtils;
import com.mengxuegu.workflow.utils.Result;
import org.activiti.engine.repository.Deployment;
import org.activiti.engine.repository.ProcessDefinition;
import org.activiti.engine.repository.ProcessDefinitionQuery;
import org.apache.commons.lang3.StringUtils;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Service
public class ProcessDefinitionService extends ActivitiService implements
IProcessDefinitionService {

    /**
     * 查询部署的流程定义数据列表,如果有多个相同标识key的流程时,只查询其最新版本
     * @return
     */
    public Result getProcDefiList(ProcDefiREQ req) {
        // 1. 获取 ProcessDefinitionQuery
        ProcessDefinitionQuery query = repositoryService.createProcessDefinitionQuery();
        // 条件查询

        if(StringUtils.isEmpty(req.getName())) {
```

```
        query.processDefinitionNameLike("%" + req.getName() + "%");
    }

    if(StringUtils.isEmpty(req.getKey())) {
        query.processDefinitionKeyLike( "%" + req.getKey() + "%");
    }

    // 有多个相同标识key的流程时，只查询其最新版本
    List<ProcessDefinition> definitionList = query.latestVersion()
        .listPage(req.getFirstResult(), req.getSize());

    // 用于前端显示页面，总记录数
    long total = query.count();

    // 封装响应结果
    List<Map<String, Object>> records = new ArrayList<>();
    for (ProcessDefinition pd: definitionList) {
        Map<String, Object> data = new HashMap<>();
        data.put("id", pd.getId());
        data.put("name", pd.getName());
        data.put("key", pd.getKey());
        data.put("version", pd.getVersion());
        // 流程定义状态
        data.put("state", pd.isSuspended() ? "已暂停" : "已启动");
        // xml文件名
        data.put("xmlName", pd.getResourceName());
        // png 文件名
        data.put("pngName", pd.getDiagramResourceName());

        // 查询部署时间
        String deploymentId = pd.getDeploymentId();
        data.put("deploymentId", deploymentId);
        Deployment deployment = repositoryService.createDeploymentQuery()
            .deploymentId(deploymentId).singleResult();
        data.put("deploymentTime", DateUtils.format(deployment.getDeploymentTime()));

        // TODO 查询流程定义与业务关系配置信息

        records.add(data);
    }

    Map<String, Object> result = new HashMap<>();
    result.put("total", total);
    result.put("records", records);
    return Result.ok(result);
}
```

创建 ProcessDefinitionController 控制层


```
package com.mengxuegu.workflow.controller;

import com.mengxuegu.workflow.req.ProcDefiREQ;
import com.mengxuegu.workflow.service.IProcessDefinitionService;
import com.mengxuegu.workflow.utils.Result;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@Api("流程定义管理")
@Slf4j
@RestController
@RequestMapping("/process")
public class ProcessDefinitionController {

    @Autowired
    private IProcessDefinitionService processDefinitionService;

    @ApiOperation("查询部署的流程定义数据列表,如果有多个相同标识key的流程时,只查询其最新版本")
    @PostMapping("/list")
    public Result getProcDefiList(@RequestBody ProcDefiREQ req) {
        return processDefinitionService.getProcDefiList(req);
    }
}
```

测试

1. 重启后台项目
2. 前端重新登录
3. 访问 <http://localhost:9528/#/workflow/process>



说明: 当有多个相同标识Key的流程时,只显示其最新版本流程。

| 序号 | 流程名称 | 标识Key | 状态 | 流程XML | 流程图片 |
|----|------|--------------|-----|--------------------|-------------------------|
| 1 | 请假流程 | leaveProcess | 已启动 | 请假流程模型 bpmn20.xml | 请假流程模型.leaveProcess.png |
| 2 | | test2 | 已启动 | 测试流程模型2 bpmn20.xml | 测试流程模型2.test2.png |

流程定义与业务配置管理

需求说明

为了后期扩展方便，部署的不同类型流程定义可灵活的与对应业务进行绑定，绑定数据保存在 `mxg_process_config` 表中。

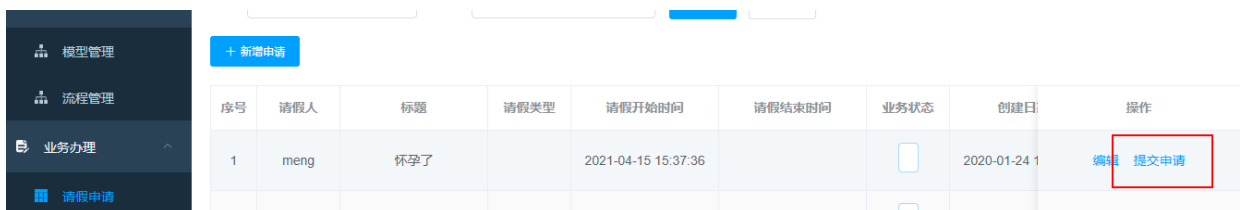
| | |
|----------------|----------|
| id | varchar |
| process_key | varchar |
| business_route | varchar |
| form_name | varchar |
| create_date | datetime |
| update_date | datetime |

绑定的有：

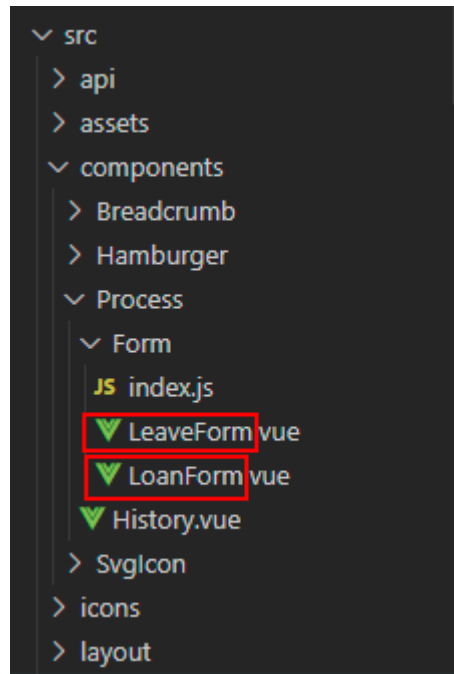
1. 绑定查询业务数据列表的路由名，对应路由名参见路由配置：src\router\index.js 中配置的name值
如：请假申请列表的路由名是：Leave



作用：是在对应业务管理中，提交申请时（启动流程实例），知道对应的是哪个流程定义。



- 绑定业务数据表单组件名



作用：用于查询提交的申请（流程实例）的审批历史记录时，知道此流程实例关联的业务表单是哪个，知道是哪个表单后，就可以使用这个表单组件来展示数据。

创建流程业务关系实体类

1. 创建流程业务关系实体类 com.mengxuegu.workflow.entities.ProcessConfig

```
package com.mengxuegu.workflow.entities;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import com.mengxuegu.workflow.utils.DateUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;
```

```
import java.io.Serializable;
import java.util.Date;

@Data
@ApiModel("流程定义配置实体")
@TableName("mxg_process_config")
public class ProcessConfig implements Serializable {

    @TableId(value = "id", type = IdType.ASSIGN_ID)
    private String id;

    @ApiModelProperty("流程KEY")
    private String processKey;

    @ApiModelProperty("业务申请路由名")
    private String businessRoute;

    @ApiModelProperty("关联表单组件名")
    private String formName;

    @ApiModelProperty("创建时间")
    private Date createDate;

    @ApiModelProperty("更新时间")
    private Date updateDate;

    public String getCreateDateStr() {
        if(createDate == null) {
            return "";
        }
        return DateUtils.format(createDate);
    }

    public String getUpdateDateStr() {
        if(updateDate == null) {
            return "";
        }
        return DateUtils.format(updateDate);
    }
}
```

创建接口 ProcessConfigMapper

创建 com.mengxuegu.workflow.mapper.ProcessConfigMapper

```
package com.mengxuegu.workflow.mapper;  
  
import com.baomidou.mybatisplus.core.mapper.BaseMapper;  
import com.mengxuegu.workflow.entities.ProcessConfig;  
  
public interface ProcessConfigMapper extends BaseMapper<ProcessConfig> {  
  
}  

```

创建文件 ProcessConfigMapper.xml

创建 com.mengxuegu.workflow.mapper.xml.ProcessConfigMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
<mapper namespace="com.mengxuegu.workflow.mapper.ProcessConfigMapper">  
  
</mapper>  

```

创建接口 IProcessConfigService

创建 com.mengxuegu.workflow.service.IProcessConfigService,

定义通过流程标识key，查询流程配置数据。

```
package com.mengxuegu.workflow.service;  
  
import com.baomidou.mybatisplus.extension.service.IService;  
import com.mengxuegu.workflow.entities.ProcessConfig;  
import com.mengxuegu.workflow.utils.Result;  
  
public interface IProcessConfigService extends IService<ProcessConfig> {  
  
    /**  
     * 通过流程key查询配置数据  
     */  
    ProcessConfig getByProcessKey(String processKey);  
  
    /**  
     * 通过流程key删除流程定义  
     * @param processKey 流程key  
     * @return  
     */  
    Result deleteByProcessKey(String processKey);  
  
}
```

创建实现类 ProcessConfigService

创建业务配置实现类 com.mengxuegu.workflow.service.impl.ProcessConfigService

实现业务配置逻辑代码：

```
package com.mengxuegu.workflow.service.impl;

import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.mengxuegu.workflow.entities.ProcessConfig;
import com.mengxuegu.workflow.mapper.ProcessConfigMapper;
import com.mengxuegu.workflow.service.IProcessConfigService;
import com.mengxuegu.workflow.utils.Result;
import org.springframework.stereotype.Service;

@Service
public class ProcessConfigService extends ServiceImpl<ProcessConfigMapper, ProcessConfig>
implements IProcessConfigService {

    @Override
    public ProcessConfig getByProcessKey(String processKey) {
        QueryWrapper<ProcessConfig> query = new QueryWrapper<>();
        query.eq("process_key", processKey);
        return baseMapper.selectOne(query);
    }

    @Override
    public Result deleteByProcessKey(String processKey) {
        QueryWrapper<ProcessConfig> query = new QueryWrapper<>();
        query.eq("process_key", processKey);
        baseMapper.delete(query);
        return Result.ok();
    }
}
```

创建业务配置控制层 ProcessConfigController

1. 流程定义列表中的 **流程配置** 按钮，需要先查询原流程配置信息回显配置窗口，如果没有原配置信息，则新增流程配置，否则更新流程配置。

```
package com.mengxuegu.workflow.controller;

import com.mengxuegu.workflow.entities.ProcessConfig;
import com.mengxuegu.workflow.service.IProcessConfigService;
```

```
import com.mengxuegu.workflow.utils.Result;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import org.activiti.engine.RuntimeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@Api("流程配置控制层")
@RequestMapping("/processConfig")
@RestController
public class ProcessConfigController {

    @Autowired
    IProcessConfigService processConfigService;

    @ApiOperation("根据流程Key查询流程配置")
    @GetMapping("/{processKey}")
    public Result view(@PathVariable String processKey) {
        return Result.ok( processConfigService.getByProcessKey(processKey) );
    }

    @ApiOperation("新增或更新流程配置")
    @PutMapping
    public Result saveOrUpdate(@RequestBody ProcessConfig processConfig) {
        boolean b = processConfigService.saveOrUpdate(processConfig);
        if(b) {
            return Result.ok();
        } else {
            return Result.error("操作失败");
        }
    }

}
```

查询流程配置信息

完善 `ProcessDefinitionService#getProcDefiList` 方法查询业务配置关系信息

- 注入 `ProcessConfigService`
- 调用通过key查询: `processConfigService.getByProcessKey(pd.getKey());`

```
@Autowired
private ProcessConfigService processConfigService;

/**
 * 查询部署的流程定义数据列表,如果有多个相同标识key的流程时, 只查询其最新版本
 * @return
 */
public Result getProcDefiList(ProcDefiReq req) {
```

```
// 1. 获取 ProcessDefinitionQuery
ProcessDefinitionQuery query = repositoryService.createProcessDefinitionQuery();
// 条件查询
if(StringUtils.isNotEmpty(req.getName())) {
    query.processDefinitionNameLike("%" + req.getName() + "%");
}

if(StringUtils.isNotEmpty(req.getKey())) {
    query.processDefinitionKeyLike( "%" + req.getKey() + "%");
}

// 有多个相同标识key的流程时，只查询其最新版本
List<ProcessDefinition> definitionList = query.latestVersion()
    .listPage(req.getFirstResult(), req.getSize());

// 用于前端显示页面，总记录数
long total = query.count();

// 封闭响应结果
List<Map<String, Object>> records = new ArrayList<>();
for (ProcessDefinition pd: definitionList) {
    Map<String, Object> data = new HashMap<>();
    data.put("id", pd.getId());
    data.put("name", pd.getName());
    data.put("key", pd.getKey());
    data.put("version", pd.getVersion());
    // 流程定义状态
    data.put("state", pd.isSuspended() ? "已暂停" : "已启动");
    // xml文件名
    data.put("xmlName", pd.getResourceName());
    // png 文件名
    data.put("pngName", pd.getDiagramResourceName());

    // 查询部署时间
    String deploymentId = pd.getDeploymentId();
    data.put("deploymentId", deploymentId);
    Deployment deployment = repositoryService.createDeploymentQuery()
        .deploymentId(deploymentId).singleResult();
    data.put("deploymentTime", DateUtils.format(deployment.getDeploymentTime()));

    // 查询流程配置信息
    ProcessConfig processConfig = processConfigService.getByProcessKey(pd.getKey());
    if(processConfig != null) {
        // 业务路由名
        data.put("businessRoute", processConfig.getBusinessRoute());
        // 表单组件名
        data.put("formName", processConfig.getFormName());
    }

    records.add(data);
}

Map<String, Object> result = new HashMap<>();
```



```
result.put("total", total);  
result.put("records", records);  
return Result.ok(result);  
}
```

测试

1. 重启后台
2. 前端重新登录
3. 访问流程管理查询流程定义数据 <http://localhost:9528/#/workflow/process>
4. 点击 **流程配置** 按钮，进行配置后提交，

流程配置

* 关联业务路由名: Leave 5/30 ✓

* 关联表单组件名: LeaveForm 9/30 ✓

提交 取消

5. 查询列表中是否有显示配置信息

工作流程
模型管理
流程管理
业务办理
个人任务

流程名称 请输入名称 标识Key 请输入标识Key 查询 重置

部署流程文件

说明: 当有多个相同标识Key的流程时, 只显示其最新版本流程。

| 状态 | 流程XML | 流程图片 | 版本号 | 关联业务路由名 | 关联表单组件名 |
|-----|-------------------|-------------------------|------|---------|-----------|
| 已启动 | 请假流程模型 bpmn20.xml | 请假流程模型 leaveProcess.png | v1.0 | Leave | LeaveForm |

挂起或激活流程定义

添加抽象方法 IProcessDefinitionService

添加抽象方法 com.mengxuegu.workflow.service.IProcessDefinitionService#updateProcDefiState

```
/**
 * 通过流程定义id，挂起或激活流程定义
 * @param procInstId 流程实例id
 * @return
 */
Result updateProcDefiState(String procInstId);
```

添加实现方法 ProcessDefinitionService

添加实现方法 com.mengxuegu.workflow.service.impl.ProcessDefinitionService#updateProcDefiState

```
public Result updateProcDefiState(String procDefiId) {
    // 流程定义对象
    ProcessDefinition processDefinition = repositoryService.createProcessDefinitionQuery()
        .processDefinitionId(procDefiId)
        .singleResult();

    // 获取当前状态是否为：挂起
    boolean suspended = processDefinition.isSuspended();
    if (suspended) {
        // 如果状态是：挂起，将状态更新为：激活
        repositoryService.activateProcessDefinitionById(procDefiId, true, null);
    } else {
        // 如果状态是：激活，将状态更新为：挂起
        repositoryService.suspendProcessDefinitionById(procDefiId, true, null);
    }
    return Result.ok();
}
```

添加API方法 ProcessDefinitionController

添加激活挂起流程定义 API方法

com.mengxuegu.workflow.controller.ProcessDefinitionController#updateProcessDefinitionState

```
@ApiOperation("通过流程定义id，挂起或激活流程定义")
@PutMapping("/state/{procDefiId}")
public Result updateProcessDefinitionState(@PathVariable String procDefiId) {
    return processDefinitionService.updateProcDefiState(procDefiId);
}
```

删除流程定义

添加抽象方法 IProcessDefinitionService

添加抽象方法 com.mengxuegu.workflow.service.IProcessDefinitionService#deleteDeployment

```
/**
 * 根据部署ID删除流程部署信息和删除流程配置信息
 * @param deploymentId 部署id
 * @param key 流程key
 * @return
 */
Result deleteDeployment(String deploymentId, String key);
```

添加实现方法 ProcessDefinitionService

添加实现方法 com.mengxuegu.workflow.service.impl.ProcessDefinitionService#deleteDeployment

```
public Result deleteDeployment(String deploymentId, String key) {
    // 不带级联的删除：如果有正在执行的流程，则删除失败抛出异常；不会删除 ACT_HI_和 历史表数据
    repositoryService.deleteDeployment(deploymentId);

    List<ProcessDefinition> list = repositoryService.createProcessDefinitionQuery()
        .processDefinitionKey(key).list();

    // 没有流程定义了，删除流程配置
    if(CollectionUtils.isEmpty(list)) {
        processConfigService.deleteByProcessKey(key);
    }
    return Result.ok();
}
```

添加API方法 ProcessDefinitionController

添加删除流程定义 API方法

com.mengxuegu.workflow.controller.ProcessDefinitionController#deleteDeployment

```
@ApiOperation("根据部署ID删除流程部署信息")
@DeleteMapping("/{deploymentId}")
public Result deleteDeployment(@PathVariable String deploymentId,
    @RequestParam String key) {
    try {
        return processDefinitionService.deleteDeployment(deploymentId, key);
    } catch (Exception e) {
        e.printStackTrace();
        String message = e.getMessage();
        log.error("根据部署ID删除流程，异常:{}", message);
        if(StringUtils.contains(message, "a foreign key constraint fails")) {
            return Result.error("有正在执行的流程实例，不允许删除");
        } else {
```

```
        return Result.error("删除失败，原因：" + e.getMessage());  
    }  
}  
}
```

流程定义xml文件和预览png图片

```
@Autowired  
RepositoryService repositoryService;  
  
@ApiOperation("导出下载流程文件(.bpmn20.xml流程图或.png图片资源)")  
@GetMapping("/export/{type}/{definitionId}")  
public void exportFile(@PathVariable("type") String type,  
    @PathVariable("definitionId") String definitionId,  
    HttpServletResponse response) {  
    try {  
        // 查询流程定义数据  
        ProcessDefinition processDefinition =  
repositoryService.getProcessDefinition(definitionId);  
        // 文件输入流  
        String filename = "文件不存在";  
        if("xml".equals(type)) {  
            // xml 类型  
            response.setContentType("application/xml");  
            // xml 文件名  
            filename = processDefinition.getResourceName();  
        }else if("png".equals(type)) {  
            // png 类型  
            response.setContentType("image/png");  
            // png 图片名  
            filename = processDefinition.getDiagramResourceName();  
        }  
  
        // 获取对应文件输入流  
        InputStream inputStream =  
repositoryService.getResourceAsStream(processDefinition.getDeploymentId(), filename);  
  
        // 防止文件名中文乱码，要进行编码  
        response.setHeader("Content-Disposition", "attachment; filename=" +  
URLLEncoder.encode(filename, "UTF-8"));  
  
        // 这句必须放到setHeader下面，否则10K以上的无法导出  
        IOUtils.copy(inputStream, response.getOutputStream());  
        response.flushBuffer();  
    } catch (Exception e) {  
        //e.printStackTrace();  
        log.error("导出失败: {}", e.getMessage());  
    }  
}
```

上传zip与xml与bpmn文件部署流程定义

```
@ApiOperation("上传 .zip 或 .bpmn 或 .bpmn20.xml 流程文件并完成部署")
@PostMapping("/file/deploy")
public Result deployByFile(@RequestParam("file") MultipartFile file) {
    try {
        // 文件名+后缀名
        String filename = file.getOriginalFilename();
        // 文件后缀名
        String suffix = filename.substring(filename.lastIndexOf(".") + 1).toUpperCase();

        InputStream input = file.getInputStream();
        DeploymentBuilder deployment = repositoryService.createDeployment();
        if("ZIP".equals(suffix)) {
            // zip
            deployment.addZipInputStream(new ZipInputStream(input));
        } else {
            // xml 或 bpmn
            deployment.addInputStream(filename, input);
        }
        // 部署名称
        deployment.name(filename.substring(0, filename.lastIndexOf(".")));

        // 开始部署
        deployment.deploy();
        return Result.ok();
    } catch (IOException e) {
        e.printStackTrace();
        log.error("部署失败: " + e.getMessage());
        return Result.error("部署失败");
    }
}
```

请假申请管理

新增请假申请

需求:

新增的请假申请（启动流程实例）数据，我们使用业务状态关系表来记录提交申请的实时状态。

单独使用一张关系表记录状态，不单单是针对请假申请，而是为了扩展其他不同流程时，也在这张表记录所有不同流程申请的实时状态。

业务状态: 0已撤回, 1待提交, 2处理中, 3已完成, 4已作废, 5已删除

请假申请实体

```
package com.mengxuegu.workflow.entities;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import com.mengxuegu.workflow.enums.LeaveTypeEnum;
import com.mengxuegu.workflow.enums.BusinessStatusEnum;
import com.mengxuegu.workflow.utils.DateUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

import java.io.Serializable;
import java.util.Date;

@Data
@ApiModel("请假申请实体")
@TableName("mxg_leave")
public class Leave implements Serializable {

    @TableId(value = "id", type = IdType.ASSIGN_ID)
    private String id;

    @ApiModelProperty("申请人用户名")
    private String username;

    @ApiModelProperty("请假类型: 1病假, 2事假, 3年假, 4婚假, 5产假, 6丧假, 7探亲, 8调休, 9其他")
    private Integer leaveType;

    /**
     * 用于前端展示名称
     */
    public String getLeaveTypeStr() {
        if(this.leaveType == null) {
            return "";
        }
        return LeaveTypeEnum.getEnumByCode(this.leaveType).getDesc();
    }

    @ApiModelProperty("标题")
    private String title;

    @ApiModelProperty("请假事由")
    private String leaveReason;

    @ApiModelProperty("请假开始时间")
    private Date startDate;

    @ApiModelProperty("请假结束时间")
}
```

```
private Date endDate;

@ApiModelProperty("请假时长, 单位: 天")
private Double duration;

@ApiModelProperty("应急工作委托人")
private String principal;

@ApiModelProperty("休息期间联系人电话")
private String contactPhone;

@ApiModelProperty("创建时间")
private Date createDate;

@ApiModelProperty("更新时间")
private Date updateDate;

@TableField(exist = false)
@ApiModelProperty("流程实例id")
private String processInstanceId;

@TableField(exist = false)
@ApiModelProperty("业务状态")
private Integer status;
public String getStatusStr() {
    if(this.status == null) {
        return "";
    }
    return BusinessStatusEnum.getEumByCode(this.status).getDesc();
}

public String getStartDateStr() {
    if(startDate == null) {
        return "";
    }
    return DateUtils.format(startDate);
}

public String getEndDateStr() {
    if(endDate == null) {
        return "";
    }
    return DateUtils.format(endDate);
}

public String getCreateDateStr() {
    if(createDate == null) {
        return "";
    }
    return DateUtils.format(createDate);
}

public String getUpdateDateStr() {
```

```
        if(updateDate == null) {  
            return "";  
        }  
        return DateUtils.format(updateDate);  
    }  
}
```

业务状态实体表

对应 mxg_business_status 表的实体类，用于管理提交申请（流程实例）的业务实时状态，方便我们管理业务状态。

```
package com.mengxuegu.workflow.entities;  
  
import com.baomidou.mybatisplus.annotation.TableId;  
import com.baomidou.mybatisplus.annotation.TableName;  
import io.swagger.annotations.ApiModel;  
import io.swagger.annotations.ApiModelProperty;  
import lombok.Data;  
  
import java.io.Serializable;  
import java.util.Date;  
  
@Data  
@ApiModel("业务状态实体")  
@TableName("mxg_business_status")  
public class BusinessStatus implements Serializable {  
  
    @TableId  
    @ApiModelProperty("业务ID")  
    private String businessKey;  
  
    @ApiModelProperty("流程实例ID")  
    private String processInstanceId;  
  
    @ApiModelProperty("流程状态: 0已撤回, 1待提交, 2处理中, 3已完成, 4已作废, 5已删除")  
    private Integer status;  
  
    @ApiModelProperty("创建时间")  
    private Date createDate;  
  
    @ApiModelProperty("更新时间")  
    private Date updateDate;  
  
}
```


创建接口 LeaveMapper

创建请假申请的 com.mengxuegu.workflow.mapper.LeaveMapper 接口

```
package com.mengxuegu.workflow.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.mengxuegu.workflow.entities.Leave;

public interface LeaveMapper extends BaseMapper<Leave> {

}
```

创建文件 LeaveMapper.xml

创建请假申请的 com.mengxuegu.workflow.mapper.xml.LeaveMapper.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.mengxuegu.workflow.mapper.LeaveMapper">

</mapper>
```

创建接口 BusinessStatusMapper

创建流程实例与业务状态关系表的 com.mengxuegu.workflow.mapper.BusinessStatusMapper接口

```
package com.mengxuegu.workflow.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.mengxuegu.workflow.entities.BusinessStatus;

public interface BusinessStatusMapper extends BaseMapper<BusinessStatus> {

}
```

创建文件 BusinessStatusMapper.xml

创建流程实例与业务状态关系表的 com.mengxuegu.workflow.mapper.xml.BusinessStatusMapper.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.mengxuegu.workflow.mapper.BusinessStatusMapper">

</mapper>
```

创建业务状态服务接口 IBusinessStatusService

创建业务状态服务接口 `com.mengxuegu.workflow.service.IBusinessStatusService`，添加一个新增业务状态方法 `add`。

```
package com.mengxuegu.workflow.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.mengxuegu.workflow.entities.BusinessStatus;

public interface IBusinessStatusService extends IService<BusinessStatus> {

    /**
     * 新增数据，状态：待处理
     * @param businessKey 业务id
     * @return
     */
    int add(String businessKey);

}
```

创建业务状态服务实现类 BusinessStatusService

创建业务状态服务实现类 `com.mengxuegu.workflow.service.impl.BusinessStatusService`，实现新增功能

```
package com.mengxuegu.workflow.service.impl;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.mengxuegu.workflow.entities.BusinessStatus;
import com.mengxuegu.workflow.enums.BusinessStatusEnum;
import com.mengxuegu.workflow.mapper.BusinessStatusMapper;
import com.mengxuegu.workflow.service.IBusinessStatusService;
import org.springframework.stereotype.Service;

@Service
public class BusinessStatusService extends ServiceImpl<BusinessStatusMapper, BusinessStatus>
implements IBusinessStatusService {

    /**
```

```
* 新增
* @param businessKey 业务id
* @return
*/
public int add(String businessKey) {
    BusinessStatus bs = new BusinessStatus();
    bs.setBusinessKey(businessKey);
    bs.setStatus(BusinessStatusEnum.WAIT.getCode());
    return baseMapper.insert(bs);
}

}
```

创建请假申请服务接口 ILeaveService

创建请假申请服务接口 com.mengxuegu.workflow.service.ILeaveService

```
package com.mengxuegu.workflow.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.mengxuegu.workflow.entities.Leave;
import com.mengxuegu.workflow.utils.Result;

public interface ILeaveService extends IService<Leave> {

    Result add(Leave leave);

}
```

创建请假申请服务实现类 LeaveService

创建请假申请服务实现类 com.mengxuegu.workflow.service.impl.LeaveService

- 保存到请假业务表
- 保存到业务状态表

```
package com.mengxuegu.workflow.service.impl;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.mengxuegu.workflow.entities.Leave;
import com.mengxuegu.workflow.mapper.LeaveMapper;
import com.mengxuegu.workflow.service.ILeaveService;
import com.mengxuegu.workflow.service.IBusinessStatusService;
import com.mengxuegu.workflow.utils.Result;
import com.mengxuegu.workflow.utils.UserUtils;
import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;
```

```
import org.springframework.transaction.annotation.Transactional;

@Service
public class LeaveService extends ServiceImpl<LeaveMapper, Leave> implements ILeaveService {

    @Autowired
    private IBusinessStatusService businessStatusService;

    @Transactional
    public Result add(Leave leave) {
        // 保存到请假业务表
        leave.setUsername(UserUtils.getUsername());
        int size = baseMapper.insert(leave);
        if(size == 1) {
            // 保存到业务流程关系中间表
            businessStatusService.add(leave.getId());
        }
        return Result.ok();
    }
}
```

创建请假申请控制层

创建请假申请控制层 com.mengxuegu.workflow.controller.LeaveController，提供新增api接口。

```
package com.mengxuegu.workflow.controller;

import com.mengxuegu.workflow.entities.Leave;
import com.mengxuegu.workflow.req.LeaveREQ;
import com.mengxuegu.workflow.service.ILeaveService;
import com.mengxuegu.workflow.utils.Result;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@Api("请假申请控制层")
@ResponseBody
@RestController
@RequestMapping("/leave")
public class LeaveController {

    @Autowired
    private ILeaveService leaveService;

    @PostMapping
    @ApiOperation("新增申请")
    public Result add(@RequestBody Leave leave) {

        return leaveService.add(leave);
    }
}
```

}

}

条件分页查询请假申请列表

需求：条件分页查询请假申请列表功能涉及到查询 mxg_leave 和 mxg_business_status

创建条件查询请假请求类

```
package com.mengxuegu.workflow.req;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

@Data
@ApiModel(value = "查询请假列表条件")
public class LeaveREQ extends BaseRequest {

    @ApiModelProperty("标题")
    private String title;

    @ApiModelProperty("流程状态")
    private Integer status;

    @ApiModelProperty("所属用户名")
    private String username;
}
```

添加Mapper层查询方法

在LeaveMapper添加条件分页查询请假列表抽象方法，

```
package com.mengxuegu.workflow.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.baomidou.mybatisplus.core.metadata.IPage;
import com.mengxuegu.workflow.entities.Leave;
import com.mengxuegu.workflow.req.LeaveREQ;
import org.apache.ibatis.annotations.Param;

public interface LeaveMapper extends BaseMapper<Leave> {

    /**
     * 查询请假和业务状态表数据列表
     * @param req
     */
}
```

```
* @return
*/
IPage<Leave> getLeaveAndStatusList(IPage page, @Param("req") LeaveREQ req);

}
```

添加Mapper.xml实现查询

在LeaveMapper.xml 实现条件分页查询请假列表， mxg_leave 和 mxg_business_status表关联查询

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.mengxuegu.workflow.mapper.LeaveMapper">

    <resultMap id="LeaveAndStatusList" type="com.mengxuegu.workflow.entities.Leave">
        <id column="id" property="id" />
        <result column="username" property="username" />
        <result column="duration" property="duration" />
        <result column="principal" property="principal" />
        <result column="contact_phone" property="contactPhone" />
        <result column="leave_type" property="leaveType" />
        <result column="title" property="title" />
        <result column="leave_reason" property="leaveReason" />
        <result column="start_date" property="startDate" />
        <result column="end_date" property="endDate" />
        <result column="create_date" property="createDate" />
        <result column="update_date" property="updateDate" />
        <result column="process_instance_id" property="processInstanceId" />
        <result column="status" property="status" />
    </resultMap>

    <select id="getLeaveAndStatusList" resultMap="LeaveAndStatusList">
        SELECT t1.*, t2.* FROM mxg_leave t1
        LEFT JOIN mxg_business_status t2
        ON t1.id = t2.business_key
        WHERE t1.username = #{req.username}
        <if test="req.title != null and req.title != ''">
            AND t1.title LIKE CONCAT('%', #{req.title}, '%')
        </if>
        <if test="req.status != null">
            AND t2.status = #{req.status}
        </if>
        ORDER BY t1.create_date DESC
    </select>

</mapper>
```

添加查询请假列表服务接口

添加条件分页查询请假服务接口方法 com.mengxuegu.workflow.service.ILeaveService#listPage

```
/**
 * 条件分页查询
 * @param req
 * @return
 */
Result listPage(LeaveREQ req);
```

添加查询请假列表服务实现

com.mengxuegu.workflow.service.impl.LeaveService#listPage

```
@Override
public Result listPage(LeaveREQ req) {
    if(StringUtils.isEmpty(req.getUsername())) {
        // 当前登录用户
        req.setUsername(UserUtils.getUsername());
    }
    IPage<Leave> page = baseMapper.getLeaveAndStatusList(req.getPage(), req);
    return Result.ok(page);
}
```

添加控制层API方法

添加查询请假列表API方法 com.mengxuegu.workflow.controller.LeaveController#listPage

```
@PostMapping("/list")
@ApiOperation("查询请假申请列表")
public Result listPage(@RequestBody LeaveREQ req) {
    return leaveService.listPage(req);
}
```

查询请假申请详情

需求：通过请假id查询详情信息，用于回显修改页面。

代码实现：

MyBatis-plus已经内置了查询方法，直接使用即可。

添加API方法：com.mengxuegu.workflow.controller.LeaveController#view

```
@GetMapping("/{id}")
@ApiOperation("查询请假详情信息")
public Result view(@PathVariable String id) {
    Leave leave = leaveService.getById(id);
    return Result.ok(leave);
}
```

提交修改请假信息

添加修改服务接口

添加 提交修改请假信息服务接口方法 com.mengxuegu.workflow.service.ILeaveService#update

```
/**
 * 更新请假信息
 */
Result update(L Leave leave);
```

添加修改服务实现

com.mengxuegu.workflow.service.impl.LeaveService#update

```
@Override
public Result update(L Leave leave) {
    if(leave == null || StringUtils.isEmpty(leave.getId())) {
        return Result.error("数据不合法");
    }
    // 查询原数据
    Leave entity = baseMapper.selectById(leave.getId());
    // 拷贝新数据
    BeanUtils.copyProperties(leave, entity);
    entity.setUpdateDate(new Date());
    baseMapper.updateById(entity);
    return Result.ok();
}
```

添加控制层API方法

添加修改请假信息的API方法 com.mengxuegu.workflow.controller.LeaveController#listPage

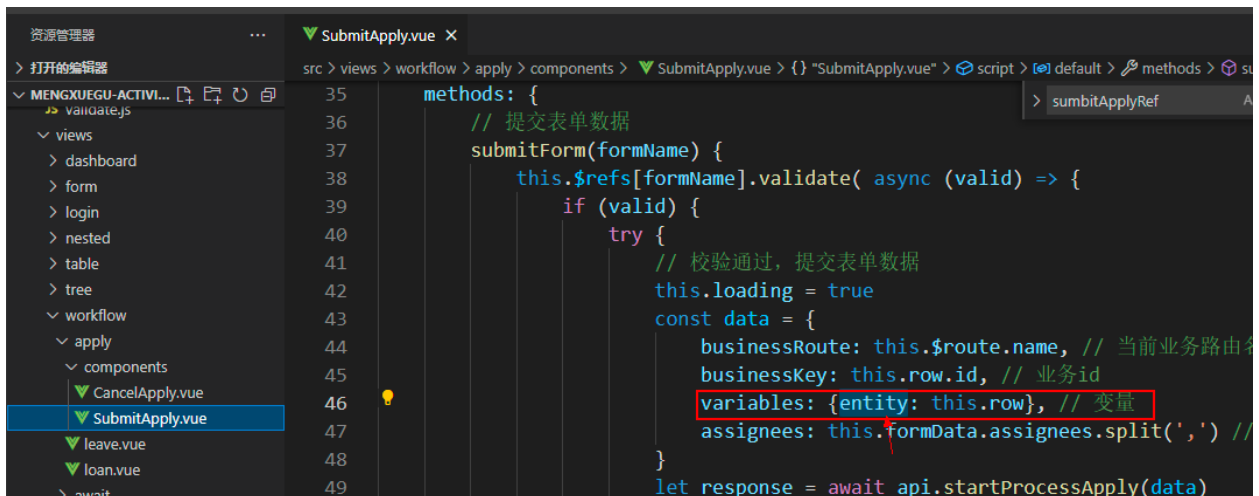

```
@PostMapping
@ApiOperation("修改详情信息")
public Result edit(@RequestBody Leave leave) {
    return leaveService.update(leave);
}
```

流程申请管理(流程实例)

提交申请(启动流程实例)-重新申请

需求

1. 请假流程定义中有流程变量 `entity.duration` 天数判断，而前端给后台时会传递一个流程变量对象 `variables`，其中有一个元素的 `key` 是 `entity`，`value` 是此申请表单数据；直接在后台使用 `Map` 类型接收到 `variables`，然后启动时设置为流程变量。



2. 后台通过当前业务列表页面的路由名（Leave）获取流程定义 `key` 和业务表单组件名，即查询流程配置信息表 `mxg_process_config`。
 - 有了流程定义 `key` 就可知道这次申请对应启动的实例是哪个流程定义；
 - 将业务表单组件名，设置到流程变量 `variables` 中，就可在任何节点中查询到表单数据进行渲染。
3. 指定第一节节点办理人或候选人 `assignees`
4. 更新流程实例业务状态码为：处理中

通过路由名查询流程配置信息

通过业务列表页面的前端名 查询流程配置信息

1. 添加查询接口方法

`com.mengxuegu.workflow.service.IProcessConfigService#getByBusinessRoute`

```
/**
 * 通过业务列表页面的前端名 查询流程配置信息
 * @param businessRoute 业务列表页面的前端名
 */
ProcessConfig getByBusinessRoute(String businessRoute);
```

2. 实现查询流程配置信息

com.mengxuegu.workflow.service.impl.ProcessConfigService#getByBusinessRoute

```
@Override
public ProcessConfig getByBusinessRoute(String businessRoute) {
    QueryWrapper<ProcessConfig> query = new QueryWrapper<>();
    query.eq("upper(business_route)", businessRoute.toUpperCase());
    List<ProcessConfig> list = baseMapper.selectList(query);
    if(CollectionUtils.isEmpty(list)) {
        return null;
    }
    return list.get(0);
}
```

更新流程实例业务状态码

当启动流程实例成功后，更新业务状态表的流程实例id和状态为：审批中

1. 添加更新状态接口方法

com.mengxuegu.workflow.service.IBusinessStatusService#updateState

```
/**
 * 根据业务id更新业务状态和流程实例id
 * @param businessKey 业务id
 * @param stateEnum 业务状态
 * @param procInstId 流程实例id
 */
Result updateState(String businessKey, BusinessStatusEnum stateEnum, String procInstId);

/**
 * 根据业务id更新业务状态
 * @param businessKey 业务id
 * @param stateEnum 业务状态
 */
Result updateState(String businessKey, BusinessStatusEnum stateEnum);
```

2. 添加更新状态实现方法

com.mengxuegu.workflow.service.impl.BusinessStatusService#updateState

```
@Override
public Result updateState(String businessKey,
                          BusinessStatusEnum stateEnum, String procInstId) {
    BusinessStatus bs = baseMapper.selectById(businessKey);
    // 状态
    bs.setStatus(stateEnum.getCode());
    bs.setUpdateDate(new Date());
    // 值为null,不会更新此字段
    if(procInstId != null) {
        bs.setProcessInstanceId(procInstId);
    }
    baseMapper.updateById(bs);
    return Result.ok();
}

@Override
public Result updateState(String businessKey, BusinessStatusEnum stateEnum) {
    return updateState(businessKey, stateEnum, null);
}
```

创建启动流程请求类

启动流程请求类用于启动流程实例时，接收前端提交的信息。

```
package com.mengxuegu.workflow.req;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

import java.io.Serializable;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Data
@ApiModel(value = "启动流程请求类")
public class StartREQ implements Serializable {

    @ApiModelProperty("业务列表页面的前端路由名")
    private String businessRoute;

    @ApiModelProperty("业务唯一值id")
    private String businessKey;

    @ApiModelProperty("节点任务办理人一位或多位")
    private List<String> assignees;

    @ApiModelProperty("流程变量")
    private Map<String, Object> variables;
```

```
public Map<String, Object> getVariables() {  
    // 为空时，创建空实例  
    return variables == null ? new HashMap<>() : variables;  
}  
  
}
```

创建启动流程实例服务接口

提交申请启动流程实例接口方法

创建 com.mengxuegu.workflow.service.IProcessInstanceService#startProcess

```
package com.mengxuegu.workflow.service;  
  
import com.mengxuegu.workflow.req.StartREQ;  
import com.mengxuegu.workflow.utils.Result;  
  
public interface IProcessInstanceService {  
    /**  
     * 提交申请启动流程实例  
     * @param req  
     * @return  
     */  
    Result startProcess(StartREQ req);  
}
```

添加启动流程实例服务实现

提交申请启动流程实例逻辑实现

创建 com.mengxuegu.workflow.service.impl.ProcessInstanceService#startProcess

```
package com.mengxuegu.workflow.service.impl;  
  
import com.mengxuegu.workflow.entities.ProcessConfig;  
import com.mengxuegu.workflow.enums.BusinessStatusEnum;  
import com.mengxuegu.workflow.req.StartREQ;  
import com.mengxuegu.workflow.service.IProcessInstanceService;  
import com.mengxuegu.workflow.utils.Result;  
import com.mengxuegu.workflow.utils.UserUtils;  
import org.activiti.engine.impl.identity.Authentication;  
import org.activiti.engine.runtime.ProcessInstance;  
import org.activiti.engine.task.Task;  
import org.apache.commons.lang3.StringUtils;  
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Service;
import org.springframework.util.CollectionUtils;

import java.util.List;
import java.util.Map;

@Service
public class ProcessInstanceService extends ActivitiService implements IProcessInstanceService {

    @Autowired
    private ProcessConfigService processConfigService;

    @Autowired
    private BusinessStatusService businessStatusService;

    @Override
    public Result startProcess(StartREQ req) {
        // 通过业务申请路由名，获取流程配置
        ProcessConfig processConfig =
processConfigService.getByBusinessRoute(req.getBusinessRoute());
        if(processConfig == null || StringUtils.isEmpty(processConfig.getProcessKey())) {
            return Result.error("未找到对应流程，请先配置流程");
        }

        // 接收前端传递的流程变量Map，其中有申请表单的 { entity: {业务表单实体数据} }
        Map<String, Object> variables = req.getVariables();
        // 将 `表单组件名`，添加到流程变量中
        variables.put("formName", processConfig.getFormName());

        // 任务办理人
        List<String> assignees = req.getAssignees();
        if(CollectionUtils.isEmpty(assignees)) {
            return Result.error("请指定审批人");
        }

        // 启动流程用户（保存在 act_hi_procinst 的 start_user_id 字段）
        Authentication.setAuthenticatedUserId(UserUtils.getUsername());

        // 开启流程实例（流程设计图唯一标识key， businessKey 业务ID， 流程变量
        ProcessInstance pi =
runtimeService.startProcessInstanceByKey(processConfig.getProcessKey(), req.getBusinessKey(),
variables);

        // 设置流程实例名称
        runtimeService.setProcessInstanceName(pi.getProcessInstanceId(),
pi.getProcessDefinitionName());

        // 查询当前流程实例的正在运行任务
        List<Task> taskList =
taskService.createTaskQuery().processInstanceId(pi.getId()).list();

        for (Task task : taskList) {

            // 分配第一个任务办理人
```

```
        if(assignees.size() == 1) {
            // 一位办理人，直接作为办理人
            taskService.setAssignee(task.getId(), assignees.get(0));
        }else {
            // 多位办理人，作为候选人
            for (String assignee : assignees) {
                taskService.addCandidateUser(task.getId(), assignee);
            }
        }
    }
}

// 更新业务状态表数据（业务key，业务状态，流程实例id）
return businessStatusService.updateState(req.getBusinessKey(),
                                           BusinessStatusEnum.PROCESS,
                                           pi.getProcessInstanceId());
}
}
```

创建启动流程实例控制层API方法

```
package com.mengxuegu.workflow.controller;

import com.mengxuegu.workflow.req.StartREQ;
import com.mengxuegu.workflow.service.IProcessInstanceService;
import com.mengxuegu.workflow.utils.Result;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@Api("流程实例管理")
@Slf4j
@RestController
@RequestMapping("/instance")
public class ProcessInstanceController {

    @Autowired
    private IProcessInstanceService processInstanceService;

    @ApiOperation(value = "提交申请，启动流程实例")
    @PostMapping("/start")
    public Result start(@RequestBody StartREQ req) {
        return processInstanceService.startProcess(req);
    }
}
```

重新申请

重新申请，在下面撤回申请后，可以重新提交进行申请，重新申请调用也是上面的接口。

撤回申请

当申请状态为 2 处理中，可进行撤回操作，撤回实际上将流程实例删除即可，然后再更新业务状态为已撤回。

添加撤回服务接口

添加撤回服务接口方法 com.mengxuegu.workflow.service.IProcessInstanceService#cancel

```
/**
 * 撤回申请
 * @param businessKey 业务id
 * @param procInstId 流程实例id
 * @param message 撤回原因
 * @return
 */
Result cancel(String businessKey, String procInstId, String message);
```

添加撤回服务实现

添加撤回服务实现方法 com.mengxuegu.workflow.service.impl.ProcessInstanceService#cancel

```
@Override
public Result cancel(String businessKey, String procInstId, String message) {
    // 撤回，删除当前流程实例
    runtimeService.deleteProcessInstance(procInstId,
        UserUtils.getUsername() + " 主动撤回了当前申请：" + message);

    // 删除历史记录
    historyService.deleteHistoricProcessInstance(procInstId);
    historyService.deleteHistoricTaskInstance(procInstId);

    // 更新业务状态：已撤回
    return businessStatusService.updateState(businessKey, BusinessStatusEnum.CANCEL, "");
}
```

添加撤回控制层API方法

添加撤回申请API方法 com.mengxuegu.workflow.controller.ProcessInstanceController#cancelApply

```
@ApiOperation("撤回申请")
@DeleteMapping("/cancel/apply")
public Result cancelApply(@RequestParam String businessKey,
                           @RequestParam String procInstId,
                           @RequestParam(defaultValue = "撤回成功") String message){
    return processInstanceService.cancel(businessKey, procInstId, message)
}
```

审批历史（业务单据和流程审批历史数据和审批历史流程图）

审批历史包含：业务单据 和 流程进度（流程审批历史数据和审批历史图）

业务单据

流程审批进度跟踪

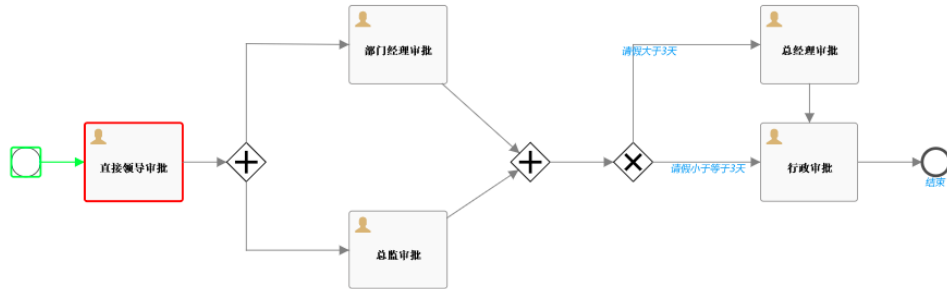
| 业务单据 | 流程进度 |
|---|------|
| <p>请假人: mengxuegu</p> <p>* 请假类型: <input type="radio"/> 病假 <input checked="" type="radio"/> 事假 <input type="radio"/> 年假 <input type="radio"/> 婚假 <input type="radio"/> 产假 <input type="radio"/> 丧假 <input type="radio"/> 探亲 <input type="radio"/> 调休 <input type="radio"/> 其他</p> <p>* 标题: <input type="text" value="有事"/> 2/100</p> <p>* 请假事由: <input type="text" value="有事有事有事"/> 6/500</p> <p>* 请假时间: <input type="text" value="2026-07-16 00:00"/> 至 <input type="text" value="2026-07-23 23:59"/></p> <p>* 请假时长: <input type="text" value="8"/> 天</p> <p>休息期间联系电话: <input type="text" value="18888888888"/> 休息期间应急工作委托人: <input type="text"/></p> | |

流程进度

流程审批进度跟踪

| | |
|------|------|
| 业务单据 | 流程进度 |
|------|------|

| 流程审批历史记录 | | | | | | |
|----------|--------|-----------|-----|------|---------------------|------|
| 序号 | 任务名称 | 办理人 | 状态 | 审批意见 | 开始时间 | 结束时间 |
| 1 | 直接领导审批 | mengxuegu | 待处理 | | 2021-07-11 16:59:35 | |



业务单据需求

- 通过实例id查询此申请（流程实例）所对应的申请表单组件名，就是获取在启动流程实例时设置的 formName 值。

目的：在审批历史页面，业务单据中动态渲染哪个表单组件。

```

mock
node_modules
public
src
  api
  assets
  components
    Breadcrumb
    Hamburger
    Process
    Form
      LeaveForm.vue
      LoanForm.vue
      History.vue
  <template>
  <el-dialog title="流程审批进度跟踪" :visible.sync="visible" align="center" wid
  <el-tabs style="min-height: 500px" type="border-card" >
  <el-tab-pane label="业务单据" v-loading="loading">
    <component :is="currProcessForm" :businessKey="businessKey"></comp
  </el-tab-pane>
  <el-tab-pane label="流程进度">
    <el-table :data="list" border stripe style="width: 90%" max-height="
    <el-table-column label="流程审批历史记录" align="center">
    <el-table-column type="index" label="序号" align="center" width
    <el-table-column prop="taskName" label="任务名称" align="center
    <el-table-column prop="assigneeName" label="办理人" align="cent
    <el-table-column prop="status" label="状态" align="center" ></e
  
```

- 通过 businessKey 业务ID 查询申请表单详情数据，渲染在表单组件上。

流程审批进度跟踪

业务单据

流程进度

请假人: mengxuegu

* 请假类型: ☒ 病假 ☐ 事假 ☐ 年假 ☐ 婚假 ☐ 产假 ☐ 丧假 ☐ 探亲 ☐ 调休 ☐ 其他

* 标题: 生病了1 4/100

* 请假事由: 生病了生病了 6/500

* 请假时间: 2027-07-08 00:00 至 2027-08-10 23:59

* 请假时长: 34 天

休息期间联系电话: 18888888888 休息期间应急工作委托人:

业务表单代码实现

1. 添加查询申请表单组件名的服务接口

com.mengxuegu.workflow.service.IProcessInstanceService#getFormNameByProcInstId

```
/**
 * 通过流程实例id获取申请表单
 * @param procInstId
 * @return
 */
Result getFormNameByProcInstId(String procInstId);
```

2. 实现查询申请表单组件名

com.mengxuegu.workflow.service.impl.ProcessInstanceService#getFormNameByProcInstId

```
@Override
public Result getFormNameByProcInstId(String procInstId) {
    // 通过历史流程实例查询，因为如果流程实例全部审批完后，
    // 正在运行的流程实例数据会被删除，只有历史中有
    HistoricProcessInstance hpi = historyService.createHistoricProcessInstanceQuery()
        .includeProcessVariables() // 查询流程变量
        .processInstanceId(procInstId).singleResult();

    // 再获取流程实例中的流程变量（启动流程实例时设置了）
    return Result.ok(hpi.getProcessVariables().get("formName"));
}
```

3. 添加查询申请表单组件名的控制层API请求接口

```
@ApiOperation("通过实例ID获取申请表单组件名")
@GetMapping("/form/name/{procInstId}")
public Result getFormName(@PathVariable String procInstId) {
    return processInstanceService.getFormNameByProcInstId(procInstId);
}
```

4. 完成上面后，审批历史页面的表单数据就会展示出来。（查询请假详情后台代码前面已编写过）

业务单据

流程进度

请假人: mengxuegu

* 请假类型: ☒ 病假 ☐ 事假 ☐ 年假 ☐ 婚假 ☐ 产假 ☐ 其他

* 标题: 生病了1

* 请假事由: 生病了生病了

* 请假时间: 2024-07-20 00:00 至 2024-08-27 23:59

* 请假时长: 39 天

休息期间联系电话: 1888888888

休息期间应急工作委托人:

查询流程审批历史数据

流程进度包含：审批历史数据和实时流程图，当前完成查询流程审批历史数据

1. 添加查询流程审批历史数据的服务接口

com.mengxuegu.workflow.service.IProcessInstanceService#getHistoryInfoList

```
/**
 * 通过流程实例Id查询审批历史信息
 * @param procInstId 流程实例id
 * @return
 */
Result getHistoryInfoList(String procInstId);
```

2. 实现查询流程审批历史数据

com.mengxuegu.workflow.service.impl.ProcessInstanceService#getHistoryInfoList

```
public Result getHistoryInfoList(String procInstId) {
    // 查询流程人工任务历史数据
    List<HistoricTaskInstance> list = historyService.createHistoricTaskInstanceQuery()
        .processInstanceId(procInstId)
        .orderByTaskCreateTime().asc()
        .list();
    List<Map<String, Object>> records = new ArrayList<>();

    for(HistoricTaskInstance hti: list){
        Map<String, Object> result = new HashMap<>();

        result.put("taskId", hti.getId()); // 任务ID
```

```
result.put("taskName", hti.getName()); // 任务名称
result.put("processInstanceId", hti.getProcessInstanceId()); // 流程实例ID
result.put("startTime", DateUtils.format(hti.getStartTime())); // 开始时间
result.put("endTime", DateUtils.format(hti.getEndTime())); // 结束时间
result.put("status", hti.getEndTime() == null ? "待处理" : "已处理"); // 状态
result.put("assignee", hti.getAssignee()); // 办理人

// 审批意见：撤回原因为空，查询审批意见
String message = hti.getDeleteReason();
if(StringUtils.isEmpty(message)) {
    List<Comment> taskComments = taskService.getTaskComments(hti.getId());
    System.out.println(taskComments);
    message = taskComments.stream()
        .map(m -> m.getFullMessage()).collect(Collectors.joining("。"));
}
result.put("message", message);
records.add(result);
}
return Result.ok(records);
}
```

3. 添加查询流程审批历史数据的控制层API请求接口

```
@ApiOperation("根据流程实例ID查询审批历史记录")
@GetMapping("/history/list")
public Result HistoryInfoList(@RequestParam String procInstId) {
    return processInstanceService.getHistoryInfoList(procInstId);
}
```

4. 完成上面后，流程审批历史记录就有数据了

| 流程审批历史记录 | | | | | | |
|----------|--------|-----------|-----|------|---------------------|------|
| 序号 | 任务名称 | 办理人 | 状态 | 审批意见 | 开始时间 | 结束时间 |
| 1 | 直接领导审批 | mengxuegu | 待处理 | | 2021-07-11 16:59:35 | |

流程实例审批历史流程图

后台绘制流程实例审批历史图（与流程定义图类似），审批历史图中会在完成审批的节点使用红色边框，正在审批节点使用绿色边框。

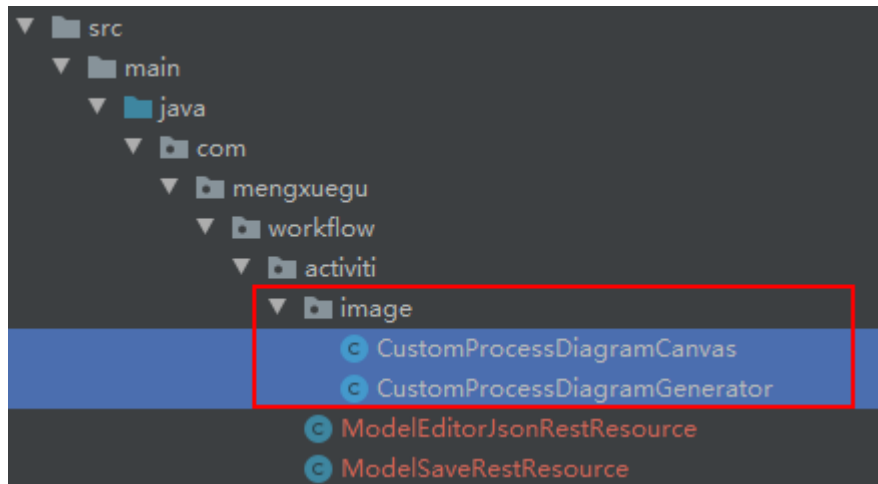
添加绘制流程图工具代码

添加绘制流程图代码，如下操作：

1. 复制 02-配套资料\Activiti7项目相关文件\activiti\image 的 image 目录，



2. 粘贴到后台项目的 `com.mengxuegu.workflow.activiti` 包下



绘制审批历史流程图实现

1. 添加获取流程实例审批历史图的服务接口

`com.mengxuegu.workflow.service.IProcessInstanceService#getHistoryProcessImage`

```
/**
 * 获取流程实例审批历史图
 * @param procInstId
 * @param response
 */
void getHistoryProcessImage(String procInstId, HttpServletResponse response);
```

2. 实现获取流程实例审批历史图

`com.mengxuegu.workflow.service.impl.ProcessInstanceService#getHistoryProcessImage`

```
public void getHistoryProcessImage(String procInstId, HttpServletResponse response) {
    InputStream imageStream = null;
    try {
        // 通过流程实例ID获取历史流程实例
        HistoricProcessInstance historicProcessInstance =
            historyService.createHistoricProcessInstanceQuery()
                .processInstanceId(procInstId).singleResult();

        // 通过流程实例ID获取流程中已经执行的节点，按照执行先后顺序排序
        List<HistoricActivityInstance> historicActivityInstanceList =
            historyService.createHistoricActivityInstanceQuery()
```

```
.processInstanceId(procInstId)
.orderByHistoricActivityInstanceStartTime().desc()
.list();

// 将已经执行的节点id放入高亮显示节点集合
List<String> highLightedActivityIdList = historicActivityInstanceList.stream()
    .map(HistoricActivityInstance::getActivityId)
    .collect(Collectors.toList());

// 通过流程实例ID获取流程中正在执行的节点
List<Execution> runningActivityInstanceList =
runtimeService.createExecutionQuery()
    .processInstanceId(procInstId)
    .list();
List<String> runningActivityIdList = new ArrayList<>();
for (Execution execution : runningActivityInstanceList) {
    if (!StringUtils.isEmpty(execution.getActivityId())) {
        runningActivityIdList.add(execution.getActivityId());
    }
}

// 获取流程定义Model对象
BpmnModel bpmnModel =
repositoryService.getBpmnModel(historicProcessInstance.getProcessDefinitionId());

// 创建流程图生成器
CustomProcessDiagramGenerator generator = new CustomProcessDiagramGenerator();
// 获取已经流经的流程线，需要高亮显示流程已经发生流转的线id集合
List<String> highLightedFlowsIds = generator.getHighLightedFlows(bpmnModel,
historicActivityInstanceList);

// 使用自定义配置获得流程图表生成器，并生成追踪图片字符流
imageStream = generator.generateDiagramCustom(bpmnModel,
    highLightedActivityIdList, runningActivityIdList, highLightedFlowsIds,
    "宋体", "微软雅黑", "黑体");

// 输出资源内容到相应对象
response.setContentType("image/svg+xml");
byte[] bytes = IOUtils.toByteArray(imageStream);
OutputStream outputStream = response.getOutputStream();
outputStream.write(bytes);
outputStream.flush();
outputStream.close();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (imageStream != null) {
        try {
            imageStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
}  
}
```

3. 添加查询流程审批历史数据的控制层API请求接口

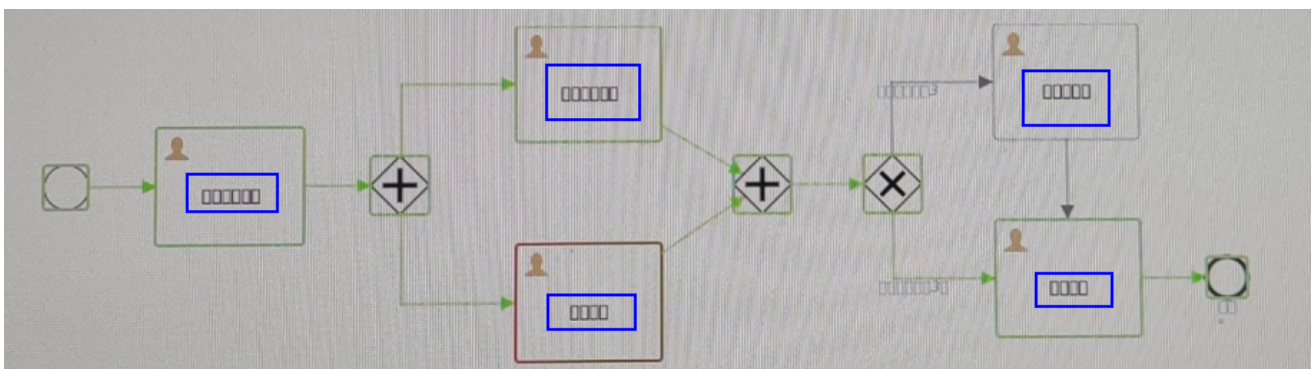
```
@ApiOperation("获取流程实例审批历史图")  
@GetMapping("/history/image")  
public void getHistoryProcessImage(@RequestParam String procInstId, HttpServletResponse  
response){  
    processInstanceService.getHistoryProcessImage(procInstId, response);  
}
```

4. 完成上面后，就可以查看到审批历史流程图



Linux部署activiti项目流程图中文乱码问题

问题：当项目如果部署到 Linux 环境，历史流程图中文可能乱码，如下图：



原因：

Linux 服务器没有安装中文字体，历史流程图是通过后端服务代码生成的，并且指定了字体，在 服务器上找不到对应字体，所以在生成图片时出现乱码。

进入字体目录，查看字体比较少

```
cd /usr/share/fonts  
ls
```

解决方法：

1. 安装中文字体

```
yum groupinstall 'fonts' -y
```

2. 安装好后，再查看安装好后的中文字体（上面一定要安装，没有执行安装命令可能也可以看下图效果）

```
locale -a |grep 'zh_CN'
```

```
[root@izwz9e73kbnrm5u9mww2m1z fonts]# locale -a |grep 'zh_CN'  
zh_CN  
zh_CN.gb18030  
zh_CN.gb2312  
zh_CN.gbk  
zh_CN.utf8
```

进入字体目录，查看多了很多字体

```
cd /usr/share/fonts  
ls
```

```
[root@izwz9e73kbnrm5u9mww2m1z fonts]# cd /usr/share/fonts  
[root@izwz9e73kbnrm5u9mww2m1z fonts]# ls  
CJKuni-uming liberation lohit-marathi open-sans stix  
dejavu lklug lohit-nepali overpass thai-scalable  
gnu-free lohit-assamese lohit-oriya paktype-naskh-basic ucs-miscfixed  
google-crosextra-caladea lohit-bengali lohit-punjabi paratype-pt-sans vlgothic  
google-crosextra-carlito lohit-devanagari lohit-tamil sil-abyssinica wqy-microhei  
google-noto-emoji lohit-gujarati lohit-telugu sil-nuosu wqy-zenhei  
jmolhari lohit-kannada madan sil-padauk  
khmeros lohit-malavalam nhn-nanum smc
```

3. 重新启动部署的项目。

个人任务管理

查询当前用户是办理人或候选人的待办任务

条件查询任务请求类

创建条件查询任务请求类 com.mengxuegu.workflow.req.TaskREQ

```
package com.mengxuegu.workflow.req;  
  
import io.swagger.annotations.ApiModel;
```



```
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

@Data
@ApiModel("条件查询任务请求类")
public class TaskREQ extends BaseRequest {

    @ApiModelProperty("任务名称")
    private String taskName;

}
```

实现查询待办任务逻辑

查询当前用户是办理人或候选人的待办任务

```
package com.mengxuegu.workflow.controller;

import com.mengxuegu.workflow.req.TaskREQ;
import com.mengxuegu.workflow.utils.DateUtils;
import com.mengxuegu.workflow.utils.Result;
import com.mengxuegu.workflow.utils.UserUtils;
import io.swagger.annotations.ApiOperation;
import lombok.extern.slf4j.Slf4j;
import org.activiti.engine.RuntimeService;
import org.activiti.engine.TaskService;
import org.activiti.engine.runtime.ProcessInstance;
import org.activiti.engine.task.Task;
import org.activiti.engine.task.TaskQuery;
import org.apache.commons.lang3.StringUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * 任务相关接口
 * 梦学谷 www.mengxuegu.com
 */
@Slf4j
@RestController
@RequestMapping("/task")
public class TaskController {
```

```
@Autowired
private TaskService taskService;

@Autowired
private RuntimeService runtimeService;

@ApiOperation("查询当前用户是办理人或候选人的待办任务")
@PostMapping("/list/wait")
public Result findWaitTask(@RequestBody TaskREQ req) {
    //办理人 (当前用户)
    String assignee = UserUtils.getUsername();
    try {
        TaskQuery query = taskService.createTaskQuery()
            .taskCandidateOrAssigned(assignee) // 作为办理人或候选人
            .orderByTaskCreateTime().desc();

        if (StringUtils.isNotEmpty(req.getTaskName())) {
            query.taskNameLike("%" + req.getTaskName() + "%");
        }

        List<Task> taskList = query.listPage(req.getFirstResult(), req.getSize());

        // 用于前端显示页面，总记录数
        long total = query.count();

        List<Map<String, Object>> records = new ArrayList<>();
        for (Task task : taskList) {
            Map<String, Object> result = new HashMap<>();
            // 任务ID
            result.put("taskId", task.getId());
            // 任务名称
            result.put("taskName", task.getName());
            // 状态
            result.put("processStatus", task.isSuspended() ? "已暂停": "已启动");
            // 任务的创建时间
            result.put("taskCreateTime", DateUtils.format(task.getCreateTime()));
            //任务的办理人: 为null, 说明当前是候选人, 不是办理人, 要先签收
            result.put("taskAssignee", task.getAssignee());
            // 流程实例ID
            result.put("processInstanceId", task.getProcessInstanceId());
            // 执行对象ID
            result.put("executionId", task.getExecutionId());
            // 流程定义ID
            result.put("processDefinitionId", task.getProcessDefinitionId());

            // 查询流程实例
            ProcessInstance pi = runtimeService.createProcessInstanceQuery()
                .processInstanceId(task.getProcessInstanceId()).singleResult();
            // 业务唯一标识
            result.put("businessKey", pi.getBusinessKey());
            // 获取发起人
            result.put("proposer", pi.getStartUserId());

            // 流程名称
```

```
        result.put("processName", pi.getProcessDefinitionName());
        // 版本号
        result.put("version", pi.getProcessDefinitionVersion());

        records.add(result);
    }

    Map<String, Object> result = new HashMap<>();
    result.put("total", total);
    result.put("records", records);
    return Result.ok("查询成功", result);
} catch (Exception e) {
    log.error("根据流程assignee查询当前人的个人任务,异常:{}", e);
    return Result.error("查询失败"+ e.getMessage());
}
}
```

完成任务-指定下一审批人

1. 获取目标节点：在任务审批通过前，先查询一下节点类型为人工任务的节点，因为人工任务才需要指定审批人；需要考虑到下一节点是网关等类型节点，它会自动处理，然后流向人工任务来等待办理。
2. 完成任务：指定了下一节点审批人后，完成任务审批时进行设置一下任务审批人。

获取目标节点(人工任务)

```
@Autowired
private RepositoryService repositoryService;

@ApiOperation("获取目标节点（下一节点）")
@GetMapping("/next/node")
public Result getNextNodeInfo(@RequestParam("taskId") String taskId) {
    Task task = taskService.createTaskQuery().taskId(taskId).singleResult();
    if(task == null) {
        return Result.error("任务不存在");
    }
    // 获取当前模型
    BpmnModel bpmnModel = repositoryService.getBpmnModel(task.getProcessDefinitionId());
    // 根据任务节点id获取当前节点
    FlowElement flowElement = bpmnModel.getFlowElement(task.getTaskDefinitionKey());
    // 封装下一个节点信息
    List<Map<String, Object>> nextNodes = new ArrayList<>();
    getNextNodes(flowElement, nextNodes);
    return Result.ok(nextNodes);
}

/**
```

```
* 判断当前节点的下一节点是人工任务的集合
* @param flowElement 当前节点
* @param nextNodes 下节点名称集合
* @return
*/
public void getNextNodes(FlowElement flowElement, List<Map<String, Object>> nextNodes) {
    // 获取当前节点的连线信息
    List<SequenceFlow> outgoingFlows = ((FlowNode)flowElement).getOutgoingFlows();
    for (SequenceFlow outgoingFlow : outgoingFlows) {
        // 下一节点
        FlowElement nextFlowElement = outgoingFlow.getTargetFlowElement();
        if(nextFlowElement instanceof EndEvent) {
            // 结束节点
            break;
        }else if(nextFlowElement instanceof UserTask) {
            Map<String, Object> node = new HashMap<>();
            // 用户任务，获取节点id和名称
            node.put("id", nextFlowElement.getId());
            node.put("name", nextFlowElement.getName());
            nextNodes.add(node);
        }else if(nextFlowElement instanceof ParallelGateway // 并行网关
            || nextFlowElement instanceof ExclusiveGateway) { // 排他网关
            getNextNodes(nextFlowElement, nextNodes);
        }
    }
}
```

创建完成任务请求类

com.mengxuegu.workflow.req.TaskCompleteREQ

```
package com.mengxuegu.workflow.req;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;
import org.apache.commons.lang3.StringUtils;

import java.io.Serializable;
import java.util.Map;

@Data
@ApiModel("完成任务请求类")
public class TaskCompleteREQ implements Serializable {

    @ApiModelProperty("任务Id")
    private String taskId;

    @ApiModelProperty("审批意见")
    private String message;
}
```

```
@ApiModelProperty("下个节点审批人, key: 节点Id, value: 审批人集合")
private Map<String, String> assigneeMap;

public String getMessage() {
    // 默认: 审批通过
    return StringUtils.isEmpty(message) ? "审批通过": message;
}

/**
 * 节点Id获取审批人集合
 * @param key
 * @return
 */
public String[] getAssignees(String key) {
    if(assigneeMap == null) {
        return null;
    }
    return assigneeMap.get(key).split(",");
}
}
```

完成任务审批

完成任务前，添加审批意见，完成后指定一下人工任务办理人。

```
@Autowired
private TaskRuntime taskRuntime;

@Autowired
private HistoryService historyService;

@Autowired
private BusinessStatusService businessStatusService;

@ApiOperation("完成任务")
@PostMapping("/complete")
public Result completeTask(@RequestBody TaskCompleteREQ req) {
    // 任务id
    String taskId = req.getTaskId();

    org.activiti.api.task.model.Task task = taskRuntime.task(taskId);

    if(task == null) {
        return Result.error("任务不存在或不是您办理的任务");
    }

    String procInstId = task.getProcessInstanceId();
```

```
// 处理意见(注意是使用 taskService)
taskService.addComment(taskId, procInstId, req.getMessage());

// 完成任务
/*taskService.complete(taskId);*/
taskRuntime.complete(TaskPayloadBuilder.complete().withTaskId(taskId).build());

// 查询下一个任务
List<Task> taskList =
    taskService.createTaskQuery().processInstanceId(procInstId).list();

// 没有任务了，表示到达最后结束节点
if (CollectionUtils.isEmpty(taskList)) {
    // 1. 获取流程实例拿到业务id,
    // 注意新api的Task对象有businessKey属性，但是没有值，
    // 并且是整个流程结束，所以查询历史实例获取businessKey
    HistoricProcessInstance hpi = historyService.createHistoricProcessInstanceQuery()
        .processInstanceId(procInstId).singleResult();
    // 2. 更新业务状态
    return businessStatusService.updateState(hpi.getBusinessKey(),
        BusinessStatusEnum.FINISH);
} else {
    // 有下一人工任务
    Map<String, String> assigneeMap = req.getAssigneeMap();
    if(assigneeMap == null) {
        // 没有分配处理人，直接结束删除流程实例
        return deleteProcessInstance(procInstId);
    }
    // 针对每个任务分配审批人
    for (Task t : taskList) {
        // 当前任务有审批人，则不设置新的审批人
        if(StringUtils.isNotEmpty(t.getAssignee())) {
            continue;
        }
        // 任务的节点id，获取对应审批人
        String[] assignees = req.getAssignees(t.getTaskDefinitionKey());
        if(ArrayUtils.isEmpty(assignees)) {
            // 如果下个节点未分配审批人为空：结束删除流程实例
            return deleteProcessInstance(procInstId);
        }
        // 分配第一个任务办理人
        if(assignees.length == 1) {
            // 一个审批人，直接作为办理人
            taskService.setAssignee(t.getId(), assignees[0]);
        } else {
            // 多个审批人，作为候选人
            for (String assignee : assignees) {
                taskService.addCandidateUser(t.getId(), assignee);
            }
        }
    }
}
```

```
        return Result.ok();
    }

    /**
     * 结束流程：删除流程实例，精力状态更新为取消状态
     * @param procInstId 流程实例id
     * @return
     */
    private Result deleteProcessInstance (String procInstId) {
        // 1. 删除流程实例
        runtimeService.deleteProcessInstance(procInstId, "审批节点未分配审批人，流程自动中断取消");
        // 2. 通过任务对象获取流程实例
        HistoricProcessInstance hpi = historyService.createHistoricProcessInstanceQuery()
            .processInstanceId(procInstId).singleResult();
        // 3. 更新业务状态
        businessStatusService.updateState(hpi.getBusinessKey(), BusinessStatusEnum.CANCEL);
        return Result.ok("审批节点未分配审批人，流程自动中断取消");
    }
}
```

签收（拾取）任务

当指定任务办理人时，可指定多个候选人（指定时使用英文逗号分隔），被指定的人进行签收任务，签收再进行审批。

提交申请

×

* 请输入审批人用户名

mengxuegu,meng

14/100



可输入用户名：【mengxuegu,meng,xue,gu】

可同时指定多个用户（用英文逗号分隔）

提交

取消

代码实现：

```
@ApiOperation("签收(拾取)任务")
@PostMapping("/claim")
public Result claimTask(@RequestParam String taskId) {
    try {
        taskRuntime.claim(TaskPayloadBuilder.claim().withTaskId(taskId).build());
        return Result.ok();
    } catch (Exception e) {
        log.error("签收(拾取)任务,异常:{}", e);
        return Result.error("签收任务失败:"+e.getMessage());
    }
}
```

任务转办-把任务交给别人处理

```
@ApiOperation("任务转办, 把任务交给别人处理")
@PostMapping("/turn")
public Result turnTask(@RequestParam String taskId,
                      @RequestParam String assigneeUserKey) {
    try {
        org.activiti.api.task.model.Task task = taskRuntime.task(taskId);
        // 转办
        taskService.setAssignee(taskId, assigneeUserKey);
        // 处理意见
        String message = String.format("%s 转办任务【%s】给 %s 办理",
                                       UserUtils.getUsername(),
                                       task.getName(),
                                       assigneeUserKey );
        taskService.addComment(taskId, task.getProcessInstanceId(), message);
        return Result.ok(taskId);
    } catch (Exception e) {
        log.error("任务转办,异常:{}", e);
        return Result.error("转办任务失败:"+e.getMessage());
    }
}
```

驳回任务

驳回任务就是返回到前面审批过的节点:

1. 通过流程实例ID获取已完成历史任务节点，用于选择驳回到哪个节点
2. 实现驳回到指定节点
 - 取得当前节点的上一个节点的信息
 - 新建流向，由当前节点指向目标节点，并分配目标节点原办理人

获取已完成历史任务节点


```
/**
 * 通过流程实例ID获取已完成历史任务节点，用于驳回功能
 */
@GetMapping("/back/nodes")
public Result getBackNodes(@RequestParam("taskId") String taskId) {
    try {
        Task task = taskService.createTaskQuery()
            .taskId(taskId)
            .taskAssignee(UserUtils.getUsername())
            .singleResult();
        if(task == null) {
            return Result.error("不是该任务办理人或任务不存在");
        }

        // 查询已完成历史任务节点，且去重，
        // 注意：必须带任务id_值，不然查询出来的全部相同数据。而且原id每条数据都是不相同，没法去重，因此
        // 使用rand()随机数作为id
        String sql =
            "select rand() AS ID_, t2.* from ( select distinct t1.TASK_DEF_KEY_, t1.NAME_ from " +
            " ( select ID_, RES.TASK_DEF_KEY_, RES.NAME_, RES.START_TIME_, RES.END_TIME_ from " +
            "   ACT_HI_TASKINST RES " +
            "   WHERE RES.PROC_INST_ID_ = #{processInstanceId} and TASK_DEF_KEY_ != #{taskDefKey}" +
            "   and RES.END_TIME_ is not null order by RES.START_TIME_ asc ) t1 ) t2";
        List<HistoricTaskInstance> htiList =
            historyService.createNativeHistoricTaskInstanceQuery()
                .sql(sql)
                .parameter("processInstanceId", task.getProcessInstanceId())
                .parameter("taskDefKey", task.getTaskDefinitionKey())
                .list();

        List<Map<String, Object>> records = new ArrayList<>();
        for (HistoricTaskInstance hti : htiList) {
            Map<String, Object> data = new HashMap<>();
            data.put("activityId", hti.getTaskDefinitionKey()); // 节点id
            data.put("activityName", hti.getName()); // 节点名称
            records.add(data);
        }

        return Result.ok(records);
    } catch (Exception e) {
        return Result.error("查询失败"+ e.getMessage());
    }
}
```

实现驳回到指定节点（实现并行网关驳回）

实现驳回到指定节点，并行网关驳回在activiti中是没有直接提供驳回的，需要自己写业务逻辑。

- 取得当前节点信息
- 获取驳回的目标节点信息

- 要考虑并行网关：从选中的目标节点的上级节点（如：并行网关），找到其上级节点的所有子节点，并行网关就会有多条子节点
- 将当前节点出口指定为驳回的目标节点，（并行网关是多条）
- 完成当前节点任务，删除执行表 is_active_=0数据，不然并行汇聚不向后流转；删除其他并行任务，
- 分配目标节点原办理人。

```
@Autowired
ManagementService managementService;

/**
 * 1. 取得当前节点信息
 * 2. 获取驳回的目标节点信息
 *    要考虑并行网关：从选中的目标节点的上级节点（如：并行网关），找到其上级节点的所有子节点，并行
网关就会有多条子节点
 * 3. 将当前节点出口指定为驳回的目标节点，（并行网关是多条）
 * 4. 完成当前节点任务，删除执行表 is_active_=0数据，不然并行汇聚不向后流转；删除其他并行任务，
 * 5. 分配目标节点原办理人。
 */
@ApiOperation("驳回指定历史节点")
@PostMapping("/back")
public Result backProcess(@RequestParam String taskId,
                          @RequestParam String targetActivityId) throws Exception {
    try {
        Task task = taskService.createTaskQuery().taskId(taskId)
                                .taskAssignee(UserUtils.getUsername())
                                .singleResult();

        // 判断当前用户是否为该节点处理人
        if (task == null) {
            return Result.error("当前用户不是该节点办理人");
        }

        String procInstId = task.getProcessInstanceId();

        // 1. 获取流程模型实例
        BpmnModel bpmnModel = repositoryService.getBpmnModel(task.getProcessDefinitionId());

        // 2. 获取当前节点信息(强类型转换 FlowNode 获取连线)
        FlowNode curFlowNode = (FlowNode)
bpmnModel.getMainProcess().getFlowElement(task.getTaskDefinitionKey());

        // 3. 获取当前节点的原出口连线
        List<SequenceFlow> sequenceFlowList = curFlowNode.getOutgoingFlows();
        // 4. 临时存储原出口连线（用于驳回后还原回来）
        List<SequenceFlow> oriSequenceFlows = new ArrayList<>(sequenceFlowList);
        // 5. 清空原出口连线
        sequenceFlowList.clear();

        // 6. 获取驳回的目标节点信息
        FlowNode targetFlowNode = (FlowNode) bpmnModel.getFlowElement(targetActivityId);

        // 7. 获取驳回的新流向节点
```

// 通过获取目标节点的入口连线，通过入口连线获取目标节点的父节点，看父节点有几个出口连线，
// 如果父节点是并行或包含网关，出口连线就会有多条，否则一般是一条出口连线，此时出口连线的所有节点才是驳回的真实节点，
// 目的：解决并行网关多个出口情况，不使用驳回父节点找子节点，当驳回到并行网关下的任务节点时会汇聚不了直接结束。

```
List<SequenceFlow> incomingFlows = targetFlowNode.getIncomingFlows();
// 存储获取驳回的新的流向
List<SequenceFlow> allSequenceFlow = new ArrayList<>();
for (SequenceFlow incomingFlow : incomingFlows) {
    // 找到入口连线的源头（获取目标节点的父节点）
    FlowNode source = (FlowNode)incomingFlow.getSourceFlowElement();
    // 获取目标节点的父组件的所有出口，
    List<SequenceFlow> sequenceFlows;
    if(source instanceof ParallelGateway) { // 并行网关
        // 并行网关的出口有多条连线:根据目标入口连线的父节点的出口连线，其所有出口连线才是驳回的真实节点
        sequenceFlows = source.getOutgoingFlows();
    }else {
        // 其他类型，将目标入口作为当前节点的出口
        sequenceFlows = targetFlowNode.getIncomingFlows();
    }
    // 找到后把它添加到集合作为新方向
    allSequenceFlow.addAll(outgoingFlows);
}

// 8. 将新方向设置为当前节点出口
curFlowNode.setOutgoingFlows(allSequenceFlow);

// 9. 查询当前流程实例的所有任务，进行完成和删除任务，删除执行数据表is_active_=0的数据
List<Task> taskList =
taskService.createTaskQuery().processInstanceId(procInstId).list();
for (Task t : taskList) {
    // 如果是当前节点任务（并行网关后面多个任务），完成任务让流程流向新的驳回节点
    if(taskId.equals(t.getId())) {
        // 保存记录 『 mengxuegu 驳回任务 【行政审批】→【部门领导审批】 』
        String message = String.format(" 『 %s 驳回任务 【%s】→【%s】 』 ",
            task.getAssignee(), curFlowNode.getName(),
            targetFlowNode.getName());
        taskService.addComment(t.getId(), procInstId, message);
        // 完成任务（驳回完成）
        taskService.complete(t.getId());
        // 完成的这条任务在act_ru_execution表中对应的执行数据，is_active_ 会更新为0，
        // 更新后，会影响并行任务节点操作驳回，然后又流向回来时并行任务完成一个就会往后走，
        // 不会等其他并行任务完成，
        // 因为并行汇聚是通过判断并行任务的执行数据 is_active_=0，所以要把当前
        // is_active_=0 的数据删除，
        DeleteExecutionCommand deleteExecutionCMD = new
DeleteExecutionCommand(task.getId());
        managementService.executeCommand(deleteExecutionCMD);
    }else {
        // 删除非当前节点任务
        DeleteTaskCommand deleteTaskCMD = new DeleteTaskCommand(t.getId());
        managementService.executeCommand(deleteTaskCMD);
    }
}
```

```
    }  
}  
  
// 10. 实现驳回功能后，将当前节点方向恢复为原来正常的方向  
curFlowNode.setOutgoingFlows(oriSequenceFlows);  
  
// 11. 查询驳回后的新任务  
List<Task> newTaskList = taskService.createTaskQuery()  
    .processInstanceId(task.getProcessInstanceId())  
    .list();  
  
// 12. 设置新任务执行人  
for (Task newTask : newTaskList) {  
    // 取得之前目标节点的原执行人  
    HistoricTaskInstance oldTargetTask =  
historyService.createHistoricTaskInstanceQuery()  
        .taskDefinitionKey(newTask.getTaskDefinitionKey()) // 同一节点  
        .processInstanceId(task.getProcessInstanceId())  
        .finished() // 已完成任务  
        .orderByTaskCreateTime().desc() // 最新办理的前面  
        .list().get(0);  
    taskService.setAssignee(newTask.getId(), oldTargetTask.getAssignee());  
}  
  
    return Result.ok();  
} catch (Exception e) {  
    e.printStackTrace();  
    log.error("完成任务,异常:{", e.getMessage());  
    return Result.error("驳回任务失败: " + e.getMessage());  
}  
}
```

自定义 command 模型实现删除执行数据

流程引擎的内部执行模式是 command 命令模式，不管是启动流程，还是任务办理等等，都调用了command的execute 方法。

```
package com.mengxuegu.workflow.cmd;  
  
import org.activiti.engine.impl.interceptor.Command;  
import org.activiti.engine.impl.interceptor.CommandContext;  
import org.activiti.engine.impl.persistence.entity.ExecutionEntity;  
import org.activiti.engine.impl.persistence.entity.ExecutionEntityManager;  
  
import java.io.Serializable;  
import java.util.List;  
  
public class DeleteExecutionCommand implements Command<String>, Serializable {  
    /**  
     * 当前任务对应的 act_ru_execution 执行id  
     */  
}
```

```
*/
private String executionId;

public DeleteExecutionCommand(String executionId) {
    this.executionId = executionId;
}

@Override
public String execute(CommandContext commandContext) {
    ExecutionEntityManager executionEntityManager =
commandContext.getExecutionEntityManager();
    // 获取当前执行数据
    ExecutionEntity executionEntity = executionEntityManager.findById(executionId);
    // 通过当前执行数据的父执行，查询所有子执行数据
    List<ExecutionEntity> allChildrenExecution =
        executionEntityManager.collectChildren(executionEntity.getParent());
    for (ExecutionEntity entity : allChildrenExecution) {
        if(!entity.isActive()) {
            // 将is_active_=0的执行数据删除，不然重复流向并行任务后，重新审批并行任务，
            // 只要有一个节点完成（就是当前有2个并行任务，驳回前已经完成了1个任务），则会并行网关就
会汇聚往后走
            executionEntityManager.delete(entity);
        }
    }
    return null;
}
}
```

自定义 command 模型实现删除任务数据

直接通过 taskService.delete是无法直接删除的，不通过校验，我们自己写逻辑调用删除方法。

```
package com.mengxuegu.workflow.cmd;

import com.mengxuegu.workflow.utils.UserUtils;
import org.activiti.engine.impl.cmd.NeedsActiveTaskCmd;
import org.activiti.engine.impl.interceptor.CommandContext;
import org.activiti.engine.impl.persistence.entity.TaskEntity;
import org.activiti.engine.impl.persistence.entity.TaskEntityManager;

public class DeleteTaskCommand extends NeedsActiveTaskCmd<String>{

    public DeleteTaskCommand(String taskId) {
        super(taskId);
    }

    @Override
    protected String execute(CommandContext commandContext, TaskEntity task) {
```

```
TaskEntityManager taskEntityManager = commandContext.getTaskEntityManager();
// 删除当前任务，不会把执行表中的 is_active_更新为0，会将任务数据更新到历史任务实例表中
taskEntityManager.deleteTask(task,
    "["+task.getName()+"] 任务被" + UserUtils.getUsername() + "]" 驳回",
    false, false);
return null;
}
}
```

查询个人已办理任务

查询当前用户的已办理任务

com.mengxuegu.workflow.controller.TaskController#findCompleteTask

```
@ApiOperation("查询当前用户的已办理任务")
@PostMapping("/list/complete")
public Result findCompleteTask(@RequestBody TaskREQ req) {
    //办理人 (当前用户)
    String assignee = UserUtils.getUsername();
    try {
        HistoricTaskInstanceQuery query = historyService.createHistoricTaskInstanceQuery()
            .taskAssignee(assignee)
            .orderByTaskCreateTime()
            .desc()
            .finished(); // 任务已办理

        if (StringUtils.isNotEmpty(req.getTaskName())) {
            query.taskNameLike("%" + req.getTaskName() + "%");
        }
        // 分页查询
        List<HistoricTaskInstance> taskList =
            query.listPage(req.getFirstResult(), req.getSize());

        // 用于前端显示页面，总记录数
        long total = query.count();

        List<Map<String, Object>> records = new ArrayList<>();
        for (HistoricTaskInstance task : taskList) {
            Map<String, Object> result = new HashMap<>();
            // 任务ID
            result.put("taskId", task.getId());
            // 任务名称
            result.put("taskName", task.getName());
            // 任务的开始时间
            result.put("taskStartTime", DateUtils.format(task.getStartTime()));
            // 任务的结束时间
            result.put("taskEndTime", DateUtils.format(task.getEndTime()));
            // 任务的办理人
```

```
result.put("taskAssignee", task.getAssignee());
// 流程实例ID
result.put("processInstanceId", task.getProcessInstanceId());
// 流程定义ID
result.put("processDefinitionId", task.getProcessDefinitionId());

HistoricProcessInstance hpi = historyService.createHistoricProcessInstanceQuery()
    .processInstanceId(task.getProcessInstanceId())
    .singleResult();
// 业务唯一标识
result.put("businessKey", hpi.getBusinessKey());
// 获取发起人
result.put("proposer", hpi.getStartUserId());
// 流程名称
result.put("processName", hpi.getProcessDefinitionName());
// 版本号
result.put("version", hpi.getProcessDefinitionVersion());

records.add(result);
}

Map<String, Object> result = new HashMap<>();
result.put("total", total);
result.put("records", records);
return Result.ok("查询成功", result);
} catch (Exception e) {
    log.error("查询当前用户的已处理任务,异常:{", e);
    return Result.error("查询失败"+ e.getMessage());
}
}
```

运行中流程实例管理

查询正在运行中的流程实例

1. 创建查询流程实例条件请求类

```
package com.mengxuegu.workflow.req;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

@ApiModel("流程实例条件请求类")
@Data
public class ProcInstREQ extends BaseRequest{
```

```
@ApiModelProperty("流程名称")
private String processName;

@ApiModelProperty("任务发起人")
private String proposer;

}
```

2. 编写查询正在运行中的流程实例的服务接口

com.mengxuegu.workflow.service.IProcessInstanceService#getProcInstListRunning

```
/**
 * 查询正在运行中的流程实例
 */
Result getProcInstListRunning(ProcInstREQ req);
```

3. 编写查询正在运行中的流程实例的服务实现

```
public Result getProcInstListRunning(ProcInstREQ req) {
    ProcessInstanceQuery query = runtimeService.createProcessInstanceQuery();
    if(StringUtils.isNotEmpty(req.getProcessName())) {
        query.processInstanceNameLikeIgnoreCase("%"+ req.getProcessName() +"%");
    }
    if(StringUtils.isNotEmpty(req.getProposer())) {
        query.startedBy(req.getProposer());
    }
    List<ProcessInstance> instanceList = query.listPage(req.getFirstResult(),
        req.getSize());

    // 用于前端显示页面，总记录数
    long total = query.count();

    List<Map<String, Object>> records = new ArrayList<>();
    for (ProcessInstance instance : instanceList) {
        Map<String, Object> result = new HashMap<>();
        // 流程实例id
        result.put("processInstanceId", instance.getProcessInstanceId());
        // 流程实例名称
        result.put("processInstanceName", instance.getName());
        result.put("processKey", instance.getProcessDefinitionKey());
        // 流程定义版本号
        result.put("version", instance.getProcessDefinitionVersion());
        // 流程发起人
        result.put("proposer", instance.getStartUserId());
        // 流程状态
        result.put("processStatus", instance.isSuspended() ? "已暂停" : "已启动");
        // 业务唯一标识
        result.put("businessKey", instance.getBusinessKey());
        // 查询当前实例任务
        List<Task> taskList = taskService.createTaskQuery()
            .processInstanceId(instance.getProcessInstanceId()).list();
    }
}
```



```
String currTaskInfo = ""; // 当前任务
for (Task task : taskList) {
    currTaskInfo += "任务名【" + task.getName() + "】，办理人【" + task.getAssignee()
+ "】<br>";
}
result.put("currTaskInfo", currTaskInfo);
result.put("startTime", DateUtils.format(instance.getStartTime()));

records.add(result);
}
// 排序
Collections.sort(records, new Comparator<Map<String, Object>>() {
    @Override
    public int compare(Map<String, Object> m1, Map<String, Object> m2) {
        String date1 = (String)m1.get("startTime");
        String date2 = (String)m2.get("startTime");
        return date2.compareTo(date1);
    }
});

Map<String, Object> result = new HashMap<>();
result.put("total", total);
result.put("records", records);
return Result.ok(result);
}
```

4. 编写查询正在运行中的流程实例的控制API请求方法

```
@ApiOperation("查询正在运行中的流程实例")
@PostMapping("/list/running")
public Result getProcInstListRunning(@RequestBody ProcInstREQ req) {
    return processInstanceService.getProcInstListRunning(req);
}
```

挂起或激活单个流程实例

挂起（暂停）或激活(启动) 单个流程实例

com.mengxuegu.workflow.controller.ProcessInstanceController#updateProcInstState

```
@Autowired
private RuntimeService runtimeService;

@ApiOperation("挂起或激活单个流程实例")
@PutMapping("/state/{procInstId}")
public Result updateProcInstState(@PathVariable("procInstId") String procInstId) {
    // 查询流程实例对象
    ProcessInstance processInstance = runtimeService.createProcessInstanceQuery()

        .processInstanceId(procInstId).singleResult();
}
```

```
// 获取当前流程实例状态是否为：挂起（暂停）
boolean suspended = processInstance.isSuspended();

// 判断
if(suspended){
    // 如果状态是：挂起，将状态更新为：激活（启动）
    runtimeService.activateProcessInstanceById(procInstId);
}else{
    // 如果状态是：激活，将状态更新为：挂起（暂停）
    runtimeService.suspendProcessInstanceById(procInstId);
}
return Result.ok();
}
```

作废正在运行流程实例

作废流程实例，不删除历史记录

com.mengxuegu.workflow.controller.ProcessInstanceController#deleteProcInst

```
@Autowired
private IBusinessStatusService businessStatusService;

@ApiOperation("作废（删除）流程实例，不会删除历史记录")
@DeleteMapping("/{procInstId}")
public Result deleteProcInst(@PathVariable("procInstId") String procInstId) {
    // 查询流程实例
    ProcessInstance instance = runtimeService.createProcessInstanceQuery()
        .processInstanceId(procInstId).singleResult();
    if(instance == null) {
        return Result.error("流程实例不存在");
    }

    // 删除流程实例
    runtimeService.deleteProcessInstance(procInstId,
        UserUtils.getUsername()+" 作废了当前流程申请");

    // 更新流程业务状态
    return businessStatusService.updateState(instance.getBusinessKey(),
        BusinessStatusEnum.INVALID);
}
```

已结束流程实例管理

查询已结束流程实例

1. 编写查询已结束的流程实例服务接口

com.mengxuegu.workflow.service.IProcessInstanceService#getProcInstFinish

```
/**
 * 查询已结束的流程实例
 * @param req
 * @return
 */
Result getProcInstFinish(ProcInstREQ req);
```

2. 编写查询已结束的流程实例的服务实现

```
@Override
public Result getProcInstFinish(ProcInstREQ req) {
    HistoricProcessInstanceQuery query =
        historyService.createHistoricProcessInstanceQuery()
            .finished() // 已结束的
            .orderByProcessInstanceEndTime().desc();

    if(StringUtils.isNotEmpty(req.getProcessName())) {
        query.processInstanceNameLikeIgnoreCase(req.getProcessName());
    }
    if(StringUtils.isNotEmpty(req.getProposer())) {
        query.startedBy(req.getProposer());
    }

    List<HistoricProcessInstance> instanceList = query.listPage(req.getFirstResult(),
        req.getSize());
    long total = query.count();

    List<Map<String, Object>> records = new ArrayList<>();
    for (HistoricProcessInstance pi : instanceList) {
        Map<String, Object> result = new HashMap<>();
        result.put("processInstanceId", pi.getId());
        result.put("processInstanceName", pi.getName());
        result.put("processKey", pi.getProcessDefinitionKey());
        result.put("version", pi.getProcessDefinitionVersion());
        result.put("proposer", pi.getStartUserId());
        result.put("businessKey", pi.getBusinessKey());
        result.put("startTime", DateUtils.format(pi.getStartTime()));
        result.put("endTime", DateUtils.format(pi.getEndTime()));
        // 原因
        result.put("deleteReason", pi.getDeleteReason());

        // 业务状态
        BusinessStatus businessStatus = businessStatusService.getById(pi.getBusinessKey());
        if(businessStatus != null) {
            result.put("status",
                BusinessStatusEnum.getEnumByCode(businessStatus.getStatus()).getDesc());
        }

        records.add(result);
    }
}
```

```
Map<String, Object> result = new HashMap<>();
result.put("total", total);
result.put("records", records);

return Result.ok(result);
}
```

3. 查询已结束流程实例 com.mengxuegu.workflow.controller.ProcessInstanceController#getProcInstFinish

```
@ApiOperation("查询已结束的流程实例")
@PostMapping("/list/finish")
public Result getProcInstFinish(@RequestBody ProcInstREQ req) {
    return processInstanceService.getProcInstFinish(req);
}
```

删除历史流程实例和审批历史记录

1. 编写删除流程实例与历史记录服务接口

com.mengxuegu.workflow.service.IProcessInstanceService#deleteProcInstAndHistory

```
/**
 * 删除流程实例与历史记录
 * @param procInstId
 * @return
 */
Result deleteProcInstAndHistory(String procInstId);
```

2. 编写删除流程实例与历史记录的服务实现

```
@Override
public Result deleteProcInstAndHistory(String procInstId) {
    // 1. 查询历史流程实例
    HistoricProcessInstance instance = historyService.createHistoricProcessInstanceQuery()
        .processInstanceId(procInstId).singleResult();

    // 2. 删除历史流程实例
    historyService.deleteHistoricProcessInstance(procInstId);
    historyService.deleteHistoricTaskInstance(procInstId);

    // 3. 更新流程业务状态，注意：流程实例id传递一个空字符串""，不要是null，不然无法更新到
    return businessStatusService.updateState(instance.getBusinessKey(),
        BusinessStatusEnum.DELETE, "");
}
```

3. 删除历史流程实例 和 删除流程业务状态（将实例id变为空字符串""，不能null,null无法将值变为空）

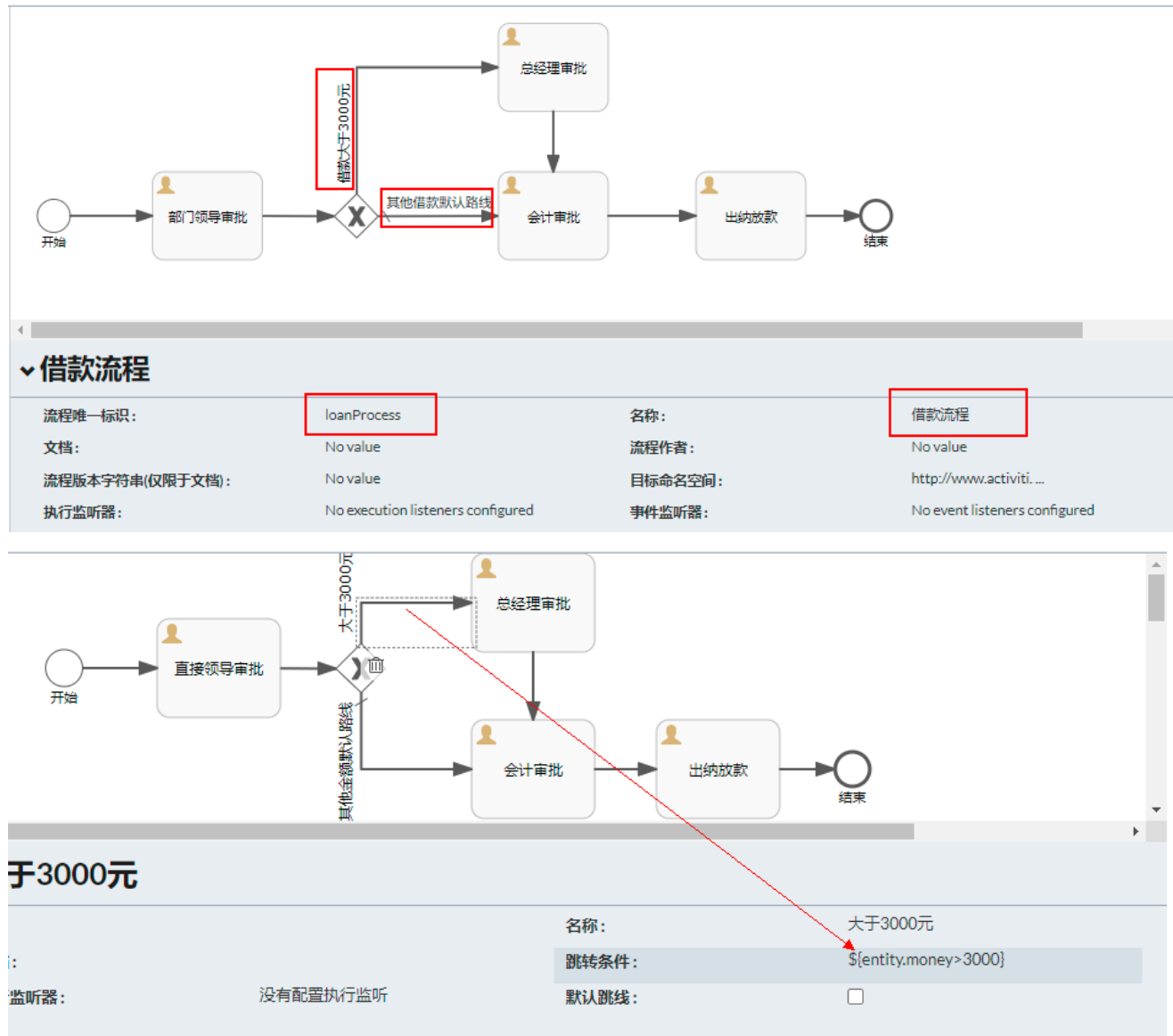
com.mengxuegu.workflow.controller.ProcessInstanceController#deleteProcInstAndHistory

```
@ApiOperation("删除已结束流程实例和历史记录")
@DeleteMapping("/history/{procInstId}")
public Result deleteProcInstAndHistory(@PathVariable String procInstId) {
    return processInstanceService.deleteProcInstAndHistory(procInstId);
}
```

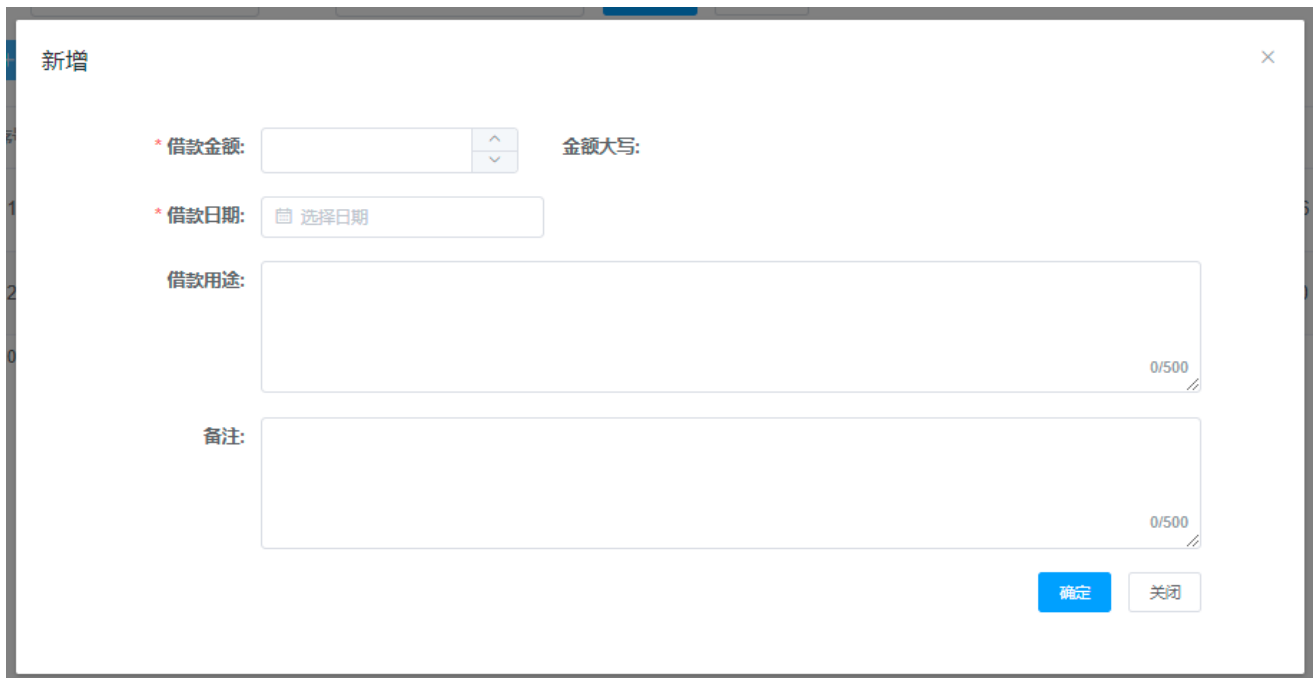
借款申请流程管理

分析借款流程

根据业务需求绘制借款流程定义模型



表单页面效果



新增

* 借款金额: 金额大写:

* 借款日期:

借款用途:

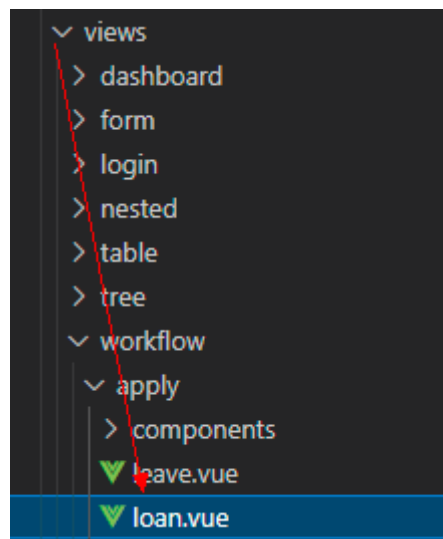
备注:

0/500 0/500

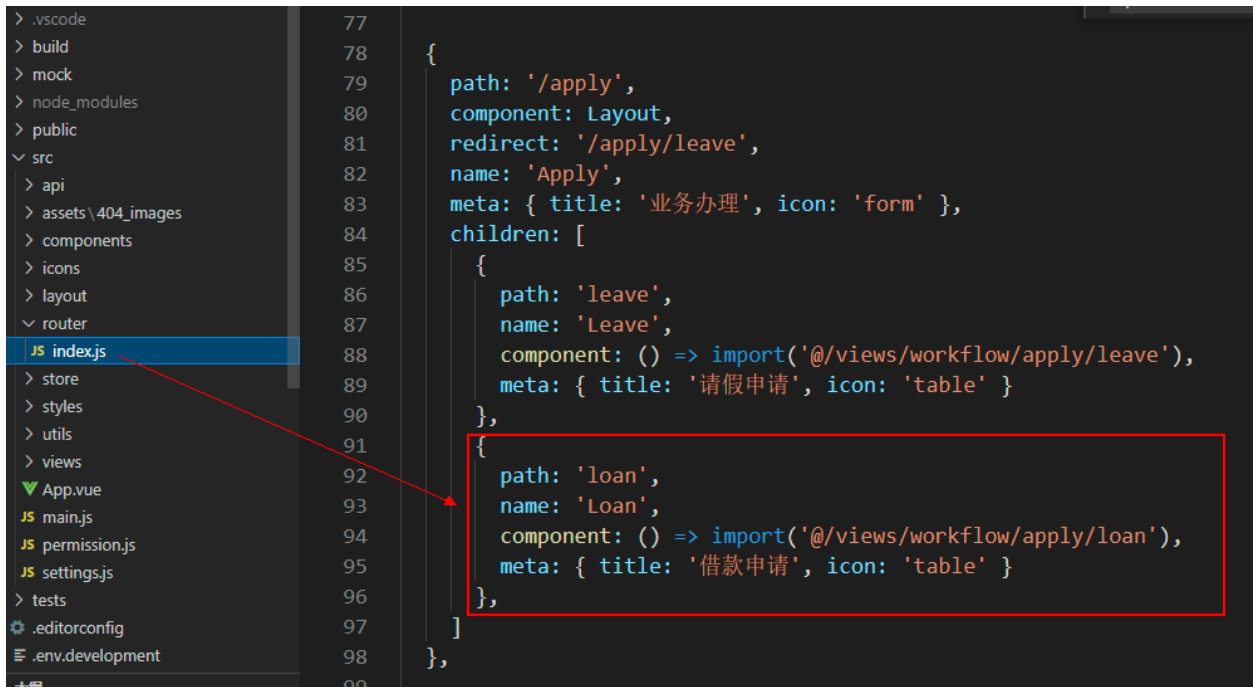
确定 关闭

前端页面开发流程

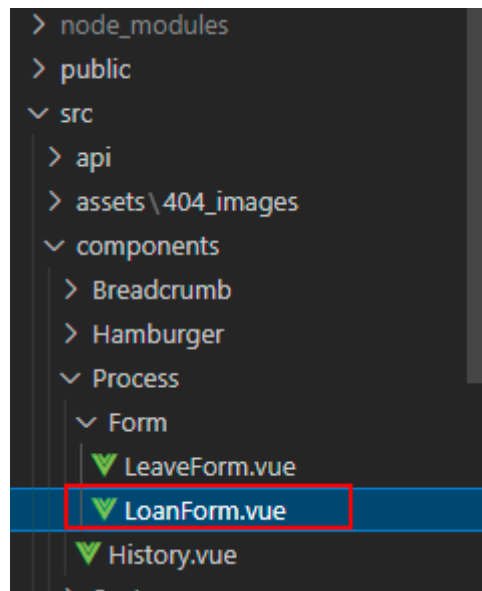
1. 创建借款查询列表组件，src\views\workflow\apply\loan.vue



2. 查询列表组件路由配置 src\router\index.js，记住配置中的 name 值 `Loan`，配置在借款流程定义中



3. 添加借款表单组件，记住表单组件名 `LoanForm`，配置在借款流程定义中，History.vue会动态加载它。



后端业务开发

数据库新建借款业务表 mxg_loan

| 字段 | 索引 | 外键 | 触发器 | 选项 | 注释 | SQL 预览 | |
|-------------|----|----|-----|----|----------|--------|-----|
| 名 | | | | | 类型 | 长度 | 小数点 |
| id | | | | | varchar | 255 | 0 |
| user_id | | | | | varchar | 40 | 0 |
| nick_name | | | | | varchar | 40 | 0 |
| money | | | | | double | 15 | 2 |
| purpose | | | | | varchar | 1000 | 0 |
| remark | | | | | varchar | 1000 | 0 |
| loan_date | | | | | datetime | 0 | 0 |
| create_date | | | | | datetime | 0 | 0 |
| update_date | | | | | datetime | 0 | 0 |

创建借款实体类

com.mengxuegu.workflow.entities.Leave

```
package com.mengxuegu.workflow.entities;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;
import com.mengxuegu.workflow.enums.LeaveTypeEnum;
import com.mengxuegu.workflow.enums.BusinessStatusEnum;
import com.mengxuegu.workflow.utils.DateUtils;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

import java.io.Serializable;
import java.util.Date;

@Data
@ApiModel("请假申请实体")
@TableName("mxg_leave")
public class Leave implements Serializable {

    @TableId(value = "id", type = IdType.ASSIGN_ID)
    private String id;

    @ApiModelProperty("申请人用户名")
    private String username;

    @ApiModelProperty("请假类型: 1病假, 2事假, 3年假, 4婚假, 5产假, 6丧假, 7探亲, 8调休, 9其他")
    private Integer leaveType;

    /**
     * 用于前端展示名称
     */
}
```



```
public String getLeaveTypeStr() {
    if(this.leaveType == null) return "";
    return LeaveTypeEnum.getEumByCode(this.leaveType).getDesc();
}

@ApiModelProperty("标题")
private String title;

@ApiModelProperty("请假事由")
private String leaveReason;

@ApiModelProperty("请假开始时间")
private Date startDate;

@ApiModelProperty("请假结束时间")
private Date endDate;

@ApiModelProperty("请假时长, 单位: 天")
private Double duration;

@ApiModelProperty("应急工作委托人")
private String principal;

@ApiModelProperty("休息期间联系人电话")
private String contactPhone;

@ApiModelProperty("创建时间")
private Date createDate;

@ApiModelProperty("更新时间")
private Date updateDate;

@TableField(exist = false)
@ApiModelProperty("流程实例id")
private String processInstanceId;

@TableField(exist = false)
@ApiModelProperty("流程状态")
private Integer status;
public String getStatusStr() {
    if(this.status == null) {
        return "";
    }
    return BusinessStatusEnum.getEumByCode(this.status).getDesc();
}

public String getStartDateStr() {
    if(startDate == null) {
        return "";
    }
    return DateUtils.format(startDate);
}
```

```
public String getEndDateStr() {
    if(endDate == null) {
        return "";
    }
    return DateUtils.format(endDate);
}

public String getCreateDateStr() {
    if(createDate == null) {
        return "";
    }
    return DateUtils.format(createDate);
}

public String getUpdateDateStr() {
    if(updateDate == null) {
        return "";
    }
    return DateUtils.format(updateDate);
}
}
```

创建条件查询借款请求类

```
package com.mengxuegu.workflow.req;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

@Data
@ApiModel("查询借款列表条件")
public class LoanREQ extends BaseRequest {

    @ApiModelProperty("用途")
    private String purpose;

    @ApiModelProperty("业务状态")
    private Integer status;

    @ApiModelProperty("所属的用户名")
    private String username;
}
```

创建LoanMapper接口

com.mengxuegu.workflow.mapper.LoanMapper

```
package com.mengxuegu.workflow.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.baomidou.mybatisplus.core.metadata.IPage;
import com.mengxuegu.workflow.entities.Leave;
import com.mengxuegu.workflow.entities.Loan;
import com.mengxuegu.workflow.req.LoanREQ;
import org.apache.ibatis.annotations.Param;

public interface LoanMapper extends BaseMapper<Loan> {

    /**
     * 查询借款和业务状态表数据列表
     * @param req
     * @return
     */
    IPage<Leave> getLoanAndStatusList(IPage page, @Param("req") LoanREQ req);

}
```

创建LoanMapper.xml文件

com.mengxuegu.workflow.mapper.xml.LoanMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.mengxuegu.workflow.mapper.LoanMapper">

    <resultMap id="LoanAndStatusList" type="com.mengxuegu.workflow.entities.Loan">
        <id column="id" property="id" />
        <result column="user_id" property="userId" />
        <result column="nick_name" property="nickName" />
        <result column="money" property="money" />
        <result column="purpose" property="purpose" />
        <result column="remark" property="remark" />
        <result column="load_date" property="loadDate" />
        <result column="create_date" property="createDate" />
        <result column="update_date" property="updateDate" />
        <result column="process_instance_id" property="processInstanceId" />
        <result column="status" property="status" />
    </resultMap>

    <select id="getLoanAndStatusList" resultMap="LoanAndStatusList">
        SELECT t1.*, t2.* FROM mxg_loan t1
        LEFT JOIN mxg_business_status t2
        ON t1.id = t2.business_key
        WHERE t1.user_id = #{req.username}
        <if test="req.purpose != null and req.purpose != ''">
            AND t1.purpose LIKE CONCAT('%', #{req.purpose}, '%')
        </if>
    </select>
</mapper>
```

```
</if>
<if test="req.status != null">
    AND t2.`status` = #{req.status}
</if>
ORDER BY t1.create_date DESC
</select>

</mapper>
```

创建 ILoanService 服务接口

com.mengxuegu.workflow.service.ILoanService

```
package com.mengxuegu.workflow.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.mengxuegu.workflow.entities.Loan;
import com.mengxuegu.workflow.req.LoanREQ;
import com.mengxuegu.workflow.utils.Result;

public interface ILoanService extends IService<Loan> {

    /**
     * 新增借款数据
     */
    Result add(Loan leave);

    /**
     * 条件分页查询借款列表数据
     */
    Result listPage(LoanREQ req);

    /**
     * 更新借款数据
     */
    Result update(Loan leave);
}
```

创建 LoanService 服务实现类

com.mengxuegu.workflow.service.impl.LoanService

```
package com.mengxuegu.workflow.service.impl;

import com.baomidou.mybatisplus.core.metadata.IPage;
import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.fasterxml.jackson.databind.ObjectMapper;
```

```
import com.mengxuegu.workflow.entities.Leave;
import com.mengxuegu.workflow.entities.Loan;
import com.mengxuegu.workflow.mapper.LoanMapper;
import com.mengxuegu.workflow.req.LoanREQ;
import com.mengxuegu.workflow.service.IBusinessStatusService;
import com.mengxuegu.workflow.service.ILoanService;
import com.mengxuegu.workflow.utils.Result;
import com.mengxuegu.workflow.utils.UserUtils;
import org.apache.commons.lang3.StringUtils;
import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.Date;

@Service
public class LoanService extends ServiceImpl<LoanMapper, Loan> implements ILoanService {

    @Autowired
    private IBusinessStatusService businessStatusService;

    @Transactional
    public Result add(Loan loan) {
        // 保存到请假业务表
        loan.setUserId(UserUtils.getUsername());
        int size = baseMapper.insert(loan);
        if(size == 1) {
            // 保存到业务流程关系中间表
            businessStatusService.add(loan.getId());
        }
        return Result.ok();
    }

    @Override
    public Result listPage(LoanREQ req) {
        if(StringUtils.isEmpty(req.getUsername())) {
            // 当前登录用户
            req.setUsername(UserUtils.getUsername());
        }
        IPage<Loan> page = baseMapper.getLoanAndStatusList(req.getPage(), req);
        return Result.ok(page);
    }

    @Override
    public Result update(Loan loan) {
        if(loan == null || StringUtils.isEmpty(loan.getId())) {
            return Result.error("数据不合法");
        }

        Loan entity = baseMapper.selectById(loan.getId());
        BeanUtils.copyProperties(loan, entity);

        entity.setUpdateDate(new Date());
    }
}
```

```
        baseMapper.updateById(entity);  
        return Result.ok();  
    }  
  
}
```

创建 LoanController 控制类

```
package com.mengxuegu.workflow.controller;  
  
import com.mengxuegu.workflow.entities.Loan;  
import com.mengxuegu.workflow.req.LoanREQ;  
import com.mengxuegu.workflow.service.ILoanService;  
import com.mengxuegu.workflow.utils.Result;  
import io.swagger.annotations.Api;  
import io.swagger.annotations.ApiOperation;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.*;  
  
@Api("借款申请控制层")  
@ResponseBody  
@RestController  
@RequestMapping("/loan")  
public class LoanController {  
  
    @Autowired  
    private ILoanService loanService;  
  
    @PostMapping  
    @ApiOperation("新增申请")  
    public Result add(@RequestBody Loan loan) {  
        return loanService.add(loan);  
    }  
  
    @PostMapping("/list")  
    @ApiOperation("查询申请列表")  
    public Result listPage(@RequestBody LoanREQ req) {  
        return loanService.listPage(req);  
    }  
  
    @GetMapping("/{id}")  
    @ApiOperation("查询详情信息")  
    public Result view(@PathVariable String id) {  
        Loan loan = loanService.getById(id);  
        return Result.ok(loan);  
    }  
  
    @PutMapping  
    @ApiOperation("修改详情信息")
```

```
public Result edit(@RequestBody Loan loan) {
    return loanService.update(loan);
}

}
```

绘制借款流程定义模型

根据业务需求绘制借款流程定义模型

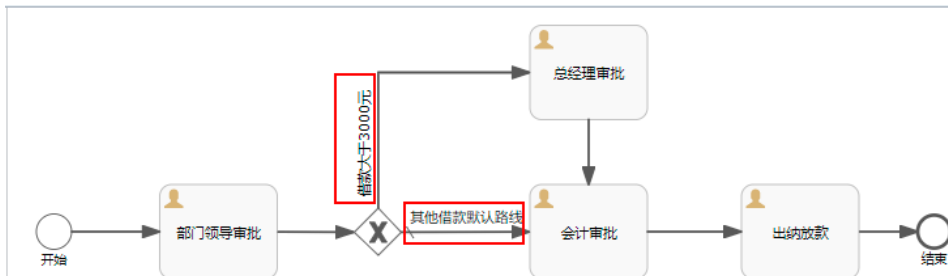
1. 新增模型数据

模型名称 标识Key

+ 新增流程模型

| 序号 | 模型名称 | 标识Key | 版本号 | 备注说明 |
|----|--------|-------------|------|--------|
| 1 | 借款流程模型 | loanProcess | v1.0 | 借款流程模型 |

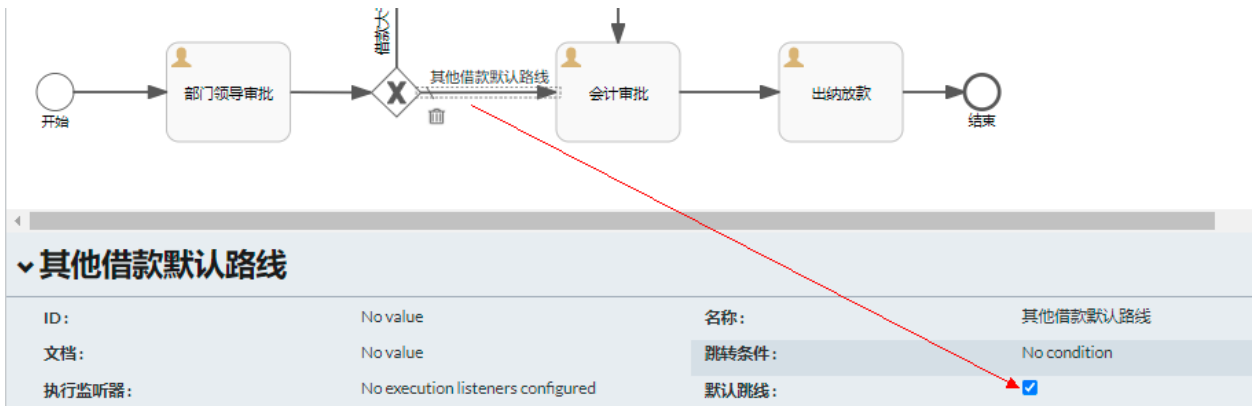
2. 绘制借款流程定义模型



借款流程

| | | | |
|-----------------|-----------------------------------|---------|-------------------------------|
| 流程唯一标识: | loanProcess | 名称: | 借款流程 |
| 文档: | No value | 流程作者: | No value |
| 流程版本字符串(仅限于文档): | No value | 目标命名空间: | http://www.activiti... |
| 执行监听器: | No execution listeners configured | 事件监听器: | No event listeners configured |

默认跳线如下:



记得点击右上角保存。

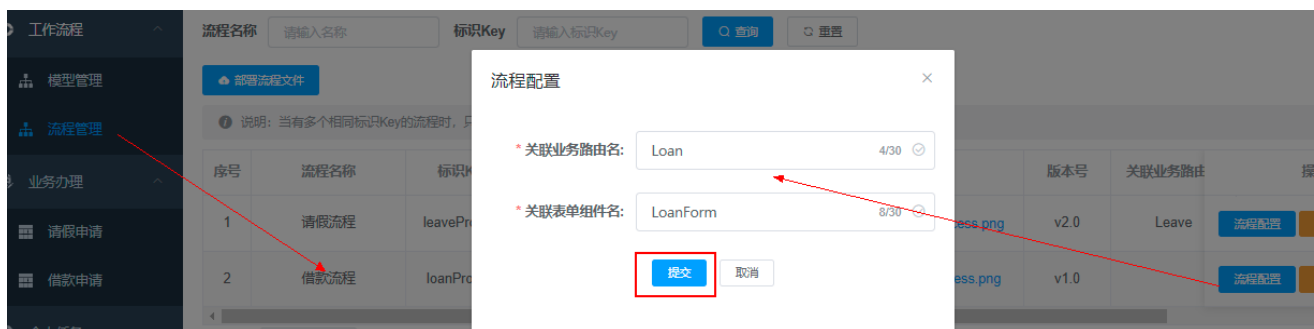
3. 点击 模型管理 页面的 部署流程 进行部署借款流程定义



配置流程定义

在 流程管理 页面中，找到部署好的借款流程定义，点击 流程配置 按钮，配置如下：

- 关联业务路由名: Loan
- 关联表单组件名: LoanForm



| 流程名称 | 标识Key | 状态 | 流程XML | 流程图片 | 版本号 | 关联业务路由名 | 关联表单组件名 |
|------|--------------|-----|-------------------|-------------------------|------|---------|-----------|
| 请假流程 | leaveProcess | 已启动 | 请假流程模型 bpmn20.xml | 请假流程模型 leaveProcess.png | v2.0 | Leave | LeaveForm |
| 借款流程 | loanProcess | 已启动 | 借款流程模型 bpmn20.xml | 借款流程模型 loanProcess.png | v1.0 | Loan | LoanForm |

测试

1. 新增借款申请，提交表单

3. 查看审批历史

仅供购买者学习，禁止盗版、转卖、传播课程