WebIDE的后端，采用java + SpringBoot + MybatisPlus + redis + mysql进行开发。SpringBoot快速开发应用简单易上手也是老生常谈的话题了。Redis缓存可以支持并发量比较大的情况仍能获得较好的读写性能，mysql（innodb）支持事务的特性能更好的保证数据持久化。

## 一、数据库设计

要设计一个应用，首先从数据库进行设计。整个WebIDE分为两大系统，依次为账户系统（用户权限控制），文件系统（运行核心逻辑）。两个系统相互独立，因此可以采用SpringCloud微服务的架构。

账户系统和文件系统相互隔离，在必要时进行服务调用即可。因此，这两个服务的数据库也是可以隔离开的。把负责账户权限控制的服务数据库命名为ide\_admin,文件服务的数据库命名为ide\_file\_system。

Ide\_admin库下应有两张表：sys\_user、sys\_user\_token。Ide\_file\_system库下数据表有如下：ide\_file\_table（文件表）、ide\_project\_table（工程表）、ide\_user\_table（用户表）、template\_file\_table（模板文件表）、\_template\_table（模板表）。

为了使系统的数据层次清晰、防止数据冗余、更新异常、插入异常和删除异常，数据库应满足第三范式。

账户服务的数据表sys\_user设计如下：



出于对系统用户的安全考虑，为每个用户增加了token表，token每次登录时自动生成，增加一个过期时间。

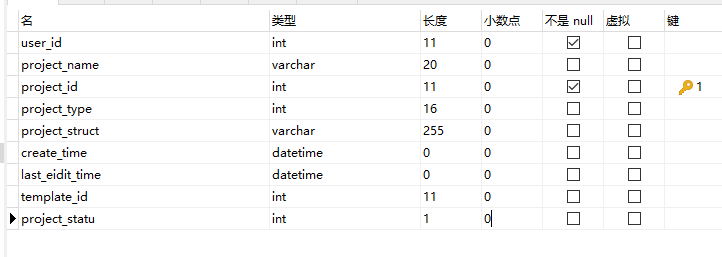


文件服务的数据表如下：

Ide\_file\_table：应包含文件的id，所属项目的id，以及文件的详细信息（文件类型，后缀，路径等）



Ide\_project\_table:应包含项目所属的用户id、项目类型、所用模板、以及编辑时间等详细信息。



Ide\_user\_table:应包含与账户服务对应的信息，如用户名、用户id、创建时间等。

Template\_file\_table:文件服务提供的项目模板数据表。该表和ide\_file\_table的数据大体一致，只因为模板数据需要固化，因此单独开设一张数据表。



Template\_table:该表用于表述模板类别关系，主要靠type字段识别是何种类型的模板（vue、react、原生）



## 二、系统详细设计

2．1短信校验注册（该功能需要在阿里云购买短信服务，并且使用用户独立的appcode进行校验）

核心逻辑是getCodeByPhone方法，它位于SignUpService接口中。

@PostMapping("/sys/getSignUpPhoneCheck")  
public R getSignUpPhoneCheck(@PathParam("phone") String phone){  
 String code = signUpService.getCodeByPhone(phone);  
 if(code.equals(""))  
 return R.*error*("验证码5分钟内有效，请勿重复发送");  
 signUpService.sendCheckMsg(phone,code);  
 return R.*ok*().put("phone",phone);  
}

下面是getCodeByPhone的实现（阿里巴巴官方文档有详细说明）。

public String getCodeByPhone(String phone) {  
 String code ="";  
 Object o = redisTemplate.opsForValue().get(phone);  
 if(o==null){  
 for(int i=0;i<5;++i){  
 code += (int) (Math.*random*()\*10);  
 }  
 // redisTemplate.opsForValue().set(phone,code);  
 redisTemplate.opsForValue().set(phone,code,600, TimeUnit.*SECONDS*);  
 }  
 return code;  
}

可以看出，当redis中没有短信信息时，则生成一个5位随机数字，并且保存10分钟。如果已经存在，则返回重复发送提示。

public void sendCheckMsg(String phone,String code) {  
 String host = "https://gyytz.market.alicloudapi.com";  
 String path = "/sms/smsSend";  
 String method = "POST";  
 String appcode = "ba0498627e6a4c28843e7ebdcb524df1";  
 Map<String, String> headers = new HashMap<String, String>();  
 //最后在header中的格式(中间是英文空格)为Authorization:APPCODE 83359fd73fe94948385f570e3c139105  
 headers.put("Authorization", "APPCODE " + appcode);  
 Map<String, String> querys = new HashMap<String, String>();  
 querys.put("mobile", phone);  
 querys.put("param", "\*\*code\*\*:"+code+",\*\*minute\*\*:10");  
 querys.put("smsSignId", "2e65b1bb3d054466b82f0c9d125465e2");  
 querys.put("templateId", "47990cc6d3ca42e2bbaad2dd06371238");  
 Map<String, String> bodys = new HashMap<String, String>();  
 try {  
 */\*\*  
 \* 重要提示如下:  
 \* HttpUtils请从  
 \* https://github.com/aliyun/api-gateway-demo-sign-java/blob/master/src/main/java/com/aliyun/api/gateway/demo/util/HttpUtils.java  
 \* 下载  
 \*  
 \* 相应的依赖请参照  
 \* https://github.com/aliyun/api-gateway-demo-sign-java/blob/master/pom.xml  
 \*/* HttpResponse response = HttpUtils.*doPost*(host, path, method, headers, querys, bodys);  
 System.*out*.println(response.toString());  
 //获取response的body  
 //System.out.println(EntityUtils.toString(response.getEntity()));  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
}

2.2登录校验

前端会对登录信息进行必要的格式校验，后端也需要（前端防君子，后端防小人）。

后端对登录信息的校验主要包括格式校验以及长度校验，图片验证码（用户体验不好，但是相对安全）校验。

对于图片验证码校验，在用户进入登录页时，首先需要生成图片校验码，然后把这个校验码和用户信息一起传入账户服务，再进行判断（如果图片校验码校验失败，则不进行后面的用户验证，直接返回失败信息）。

*/\*\*  
 \* 验证码  
 \*/*@GetMapping("captcha.jpg")  
public void captcha(HttpServletResponse response, String uuid) throws IOException {  
 response.setHeader("Cache-Control", "no-store, no-cache");  
 response.setContentType("image/jpeg");  
 //获取图片验证码  
 BufferedImage image = sysCaptchaService.getCaptcha(uuid);  
 ServletOutputStream out = response.getOutputStream();  
 ImageIO.*write*(image, "jpg", out);  
 IOUtils.*closeQuietly*(out);  
}

对于用户信息校验，首先给非空字段标上注解，验证非空，然后对邮箱、手机号等字段写入正则校验。

@Data  
@TableName("sys\_user")  
public class SysUserEntity implements Serializable {  
 private static final long *serialVersionUID* = 1L;  
  
 */\*\*  
 \* 用户ID  
 \*/* @TableId  
 private Long userId;  
  
 */\*\*  
 \* 用户名  
 \*/* @NotBlank(message = "用户名不能为空", groups = {AddGroup.class, UpdateGroup.class})  
 private String username;  
  
 */\*\*  
 \* 密码  
 \*/* @NotBlank(message = "密码不能为空", groups = AddGroup.class)  
 private String password;  
  
 */\*\*  
 \* 盐  
 \*/* private String salt;  
  
 */\*\*  
 \* 邮箱  
 \*/* @NotBlank(message = "邮箱不能为空", groups = {AddGroup.class, UpdateGroup.class})  
 @Email(message = "邮箱格式不正确", groups = {AddGroup.class, UpdateGroup.class})  
 private String email;  
  
 */\*\*  
 \* 手机号  
 \*/* private String mobile;  
  
 */\*\*  
 \* 状态 0：禁用 1：正常  
 \*/* private Integer status;  
  
 */\*\*  
 \* 角色ID列表  
 \*/* @TableField(exist = false)  
 private List<Long> roleIdList;  
  
 */\*\*  
 \* 创建者ID  
 \*/* private Long createUserId;  
  
 */\*\*  
 \* 创建时间  
 \*/* private Date createTime;  
  
}

其中校验组Group如下：

public static void validateEntity(Object object, Class<?>... groups)  
 throws RRException {  
 Set<ConstraintViolation<Object>> constraintViolations = *validator*.validate(object, groups);  
 if (!constraintViolations.isEmpty()) {  
 StringBuilder msg = new StringBuilder();  
 for (ConstraintViolation<Object> constraint : constraintViolations) {  
 msg.append(constraint.getMessage()).append("<br>");  
 }  
 throw new RRException(msg.toString());  
 }  
}

非空校验：

public static void isBlank(String str, String message) {  
 if (StringUtils.*isBlank*(str)) {  
 throw new RRException(message);  
 }  
}  
  
public static void isNull(Object object, String message) {  
 if (object == null) {  
 throw new RRException(message);  
 }  
}

2.3文件树生成

文件树生成主要是通过广度优先的遍历文件夹来实现。传入项目的id,然后去ide\_file\_table中递归查找即可（最终结果是返回一个ProjectVo,这个Vo就是前端界面上展示的文件树）。值得注意的是，在递归查找结束时，应当对ProjectVo内部进行一个排序，使得文件夹在文件的上方，符合日常习惯。

public ProjectVo getPageByProjectId(Integer id) {  
 QueryWrapper<FileTableEntity> mapper = new QueryWrapper<>();  
 mapper.eq("project\_id",id);  
  
 List<FileTableEntity> fileTableEntities = fileTableDao.selectList(mapper);  
  
 ProjectVo projectVo = new ProjectVo();  
 projectVo.setProject\_id(id);  
 List<ProjectContent> childsFilefolder = new ArrayList<>();  
 List<ProjectContent> childsFile = new ArrayList<>();  
 for(int i=0;i<fileTableEntities.size();++i){  
 if(fileTableEntities.get(i).getPid()==0){  
 if(fileTableEntities.get(i).getFileType().equals("package"))  
 childsFilefolder.add(new ProjectContent(fileTableEntities.get(i)));  
 else childsFile.add(new ProjectContent(fileTableEntities.get(i)));  
 }  
 }  
 for(int i=0;i<childsFile.size();++i){  
 childsFilefolder.add(childsFile.get(i));  
 };  
 projectVo.setChildren(childsFilefolder);  
 for(int i=0;i<projectVo.getChildren().size();++i){  
 setAllChilds(projectVo.getChildren().get(i),fileTableEntities);  
 }  
  
 return projectVo;  
}

2.4项目及文件删除

项目删除需要递归删除项目空间下所有文件，而文件删除只需要删除单个文件。项目删除还会涉及到其它表的内容，比如ide\_project\_table。

public String deleteFileFolder(FileTableEntity fileTableEntity) {  
 if(fileTableEntity.getFileType().equals("package")){  
 FileTableEntity entity = new FileTableEntity();  
 entity.setProjectId(fileTableEntity.getProjectId());  
 //entity.set  
 List<Integer> ids = new ArrayList<>();  
 QueryWrapper<FileTableEntity> queryWrapper = new QueryWrapper<>();  
 queryWrapper.eq("project\_id",fileTableEntity.getProjectId());  
 List<FileTableEntity> fileTableEntities = fileTableDao.selectList(queryWrapper);  
 Queue<FileTableEntity> que = new LinkedList<>();  
 que.offer(fileTableEntity);  
 while(!que.isEmpty()){  
 FileTableEntity temp = que.poll();  
 Integer deleteId = temp.getId();  
 ids.add(deleteId);  
 for(int i=0;i<fileTableEntities.size();++i){  
 if(deleteId==(int)fileTableEntities.get(i).getPid()){  
 que.offer(fileTableEntities.get(i));  
 }  
 }  
 }  
 try {  
 fileTableDao.deleteBatchIds(ids);  
 }catch (Exception e){  
 return "Unknow error!";  
 }  
 return null;  
 }  
 else if(fileTableEntity.getFileType().equals("file")){  
 try {  
 fileTableDao.deleteById(fileTableEntity.getId());  
 return null;  
 }catch (Exception e){  
 return "Unknow error!";  
 }  
 }  
 return null;  
}

2.5、模板生成项目

模板生成项目时，只需根据模板id去相应的模板文件夹内拷贝模板项目即可。每次生成的模板项目初始文件目录及文件内容是相同的。

@Transactional  
@Override  
public void createFileByTemplate(ProjectTableEntity projectTableEntity) {  
 QueryWrapper<TemplateFileTableEntity> queryWrapper = new QueryWrapper<>();  
 queryWrapper.eq("template\_id",projectTableEntity.getTemplateId());  
 List<TemplateFileTableEntity> templateFileTableEntities = templateFileTableDao.selectList(queryWrapper);  
 List<FileTableEntity> fileTableEntities = new ArrayList<>();  
 templateFileTableEntities.forEach(item->{  
 fileTableEntities.add(copyTemplateToFile(new FileTableEntity(),item));  
 });  
 Queue<FileTableEntity> que = new LinkedList<>();  
 fileTableEntities.forEach(item->{  
 item.setProjectId(projectTableEntity.getProjectId());  
 fileTableDao.insert(item);  
 if(item.getPid()==0)  
 que.offer(item);  
 });  
 while(!que.isEmpty()){  
 FileTableEntity tempFile = que.poll();  
 fileTableEntities.forEach(item->{  
 if(tempFile.getVirtulId()==(int)item.getPid()){  
 item.setPid(tempFile.getId());//tempFile.getId()  
 que.offer(item);  
 }  
 });  
 }  
 fileTableEntities.forEach(item->{  
 fileTableDao.updateById(item);  
 });  
}

2.6代码下载（打包）

代码下载就需要把用户当前项目的所有文件进行查找并且压缩为zip返回给前端，因此需要获得项目的根路径，然后通过Ziputils类进行递归压缩文件即可。返回时，通过流的方式返回，修改SpringBoot控制器传入的默认对象的返回方式response.setContentType，即可流式返回。

@RequestMapping("/downloadProject")  
public R downloadProject(HttpServletRequest httpServletRequest, HttpServletResponse response,@RequestParam("projectId")Integer projectId){  
 String fileAndPackage = fileTableService.createFileAndPackage(projectId);  
 String realPath = fileAndPackage;  
 //String realPath="D:\\WebIDE\\ide\\userFileZIP\\312349.zip";  
 // String realPath = this.getServletContext().getRealPath("/download/1.JPG");  
 //2.获取要下载的文件名  
 String fileName = realPath.substring(realPath.lastIndexOf("\\")+1);  
 //3.设置content-disposition响应头控制浏览器以下载的形式打开文件  
 response.setContentType("application/octet-stream;charset=UTF-8");  
 response.setHeader("content-disposition", "attachment;filename="+fileName);  
 //4.获取要下载的文件输入流  
 InputStream in = null;  
 try {  
 in = new FileInputStream(realPath);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 int len = 0;  
 //5.创建数据缓冲区  
 byte[] buffer = new byte[1024];  
 //6.通过response对象获取OutputStream流  
 OutputStream out = null;  
 try {  
 out = response.getOutputStream();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 //7.将FileInputStream流写入到buffer缓冲区  
 while (true) {  
 try {  
 if (!((len = in.read(buffer)) > 0)) break;  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 //8.使用OutputStream将缓冲区的数据输出到客户端浏览器  
 try {  
 out.write(buffer,0,len);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 try {  
 in.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 // BufferedInputStream bins = new BufferedInputStream(fileAndPackage);//放到缓冲流里面  
 // OutputStream outs = response.getOutputStream();//获取文件输出IO流  
 return R.*ok*().put("fileName",fileName);  
}

递归压缩方法如下（递归出口就是当前文件夹为空）：

private static void compress(File sourceFile, ZipOutputStream zos, String name,  
 boolean KeepDirStructure) throws Exception {  
 byte[] buf = new byte[*BUFFER\_SIZE*];  
 if (sourceFile.isFile()) {  
 // 向zip输出流中添加一个zip实体，构造器中name为zip实体的文件的名字  
 zos.putNextEntry(new ZipEntry(name));  
 // copy文件到zip输出流中  
 int len;  
 FileInputStream in = new FileInputStream(sourceFile);  
 while ((len = in.read(buf)) != -1) {  
 zos.write(buf, 0, len);  
 }  
 // Complete the entry  
 zos.closeEntry();  
 in.close();  
 } else {  
 File[] listFiles = sourceFile.listFiles();  
 if (listFiles == null || listFiles.length == 0) {  
 // 需要保留原来的文件结构时,需要对空文件夹进行处理  
 if (KeepDirStructure) {  
 // 空文件夹的处理  
 zos.putNextEntry(new ZipEntry(name + "/"));  
 // 没有文件，不需要文件的copy  
 zos.closeEntry();  
 }  
 } else {  
 for (File file : listFiles) {  
 // 判断是否需要保留原来的文件结构  
 if (KeepDirStructure) {  
 // 注意：file.getName()前面需要带上父文件夹的名字加一斜杠,  
 // 不然最后压缩包中就不能保留原来的文件结构,即：所有文件都跑到压缩包根目录下了  
 *compress*(file, zos, name + "/" + file.getName(), KeepDirStructure);  
 } else {  
 *compress*(file, zos, file.getName(), KeepDirStructure);  
 }  
 }  
 }  
 }  
}

2.7在线预览方案

使用ProcessBuilder开辟新线程通过调用系统的批处理文件(后缀为.bat的文件)，处理项目文件（可以npm build或者npm start两种方式，根据同时用户在线数量决定用哪种方式）然后把生成的静态文件路径返回给前端即可，前端采用iframe进行展示。此外，可以手动写入bat文件内容，后续在线预览时，也会根据手动写入的bat文件内容决定预览方案。

public static String NpmProcessBuilder() {  
 ProcessBuilder processBuilder = new ProcessBuilder();  
// processBuilder.command("ping","127.0.0.1");  
 ///processBuilder.command("cd userFile");  
 processBuilder.command("run.bat");  
 // processBuilder.command("run.bat");  
 //将标准输入流和错误输入流合并，通过标准输入流读取信息  
  
 processBuilder.redirectErrorStream(true);  
 //System.exit(1);  
 try {  
 //启动进程  
 Process start = processBuilder.start();  
 //获取输入流  
 InputStream inputStream = start.getInputStream();  
 //转成字符输入流  
 InputStreamReader inputStreamReader = new InputStreamReader(inputStream, "utf-8");  
 int len = -1;  
 char[] c = new char[1024];  
 StringBuffer outputString = new StringBuffer();  
 //读取进程输入流中的内容  
 while ((len = inputStreamReader.read(c)) != -1) {  
 String s = new String(c, 0, len);  
 outputString.append(s);  
 //System.out.print(s);  
 //Thread.sleep(10000);  
 }  
 inputStream.close();  
 return outputString.toString();  
 } catch (IOException e) {  
 e.printStackTrace();  
 } //catch (InterruptedException e) {  
// e.printStackTrace();  
// }  
 return null;  
 }

.bat文件大致内容

cd userFile/template  
npm run dev

#WebIDE的核心大概就是这些，比较细节的地方后续完善

#需要完善的有：通过网关实现系统安全保证（所有用户仅知道网关地址，而不知道各个服务的真实ip，一定程度上保障了安全性）。

#服务调用、事务相关。

## 三、部分说明

请求返回对象R：R中有状态码，code和信息msg,实际上就是一个JSON对象，请求成功时，code为0，失败时附带msg.

*/\*\*  
 \* 返回数据  
 \*  
 \** ***@author*** *Mark sunlightcs@gmail.com  
 \*/*public class R extends HashMap<String, Object> {  
 private static final long *serialVersionUID* = 1L;  
  
 //利用fastjson进行逆转 可以指定域中key的名字  
 public <T> T getData(String key, TypeReference<T> typeReference) {  
 Object data = get(key);  
 String s = JSON.*toJSONString*(data);  
 T t = JSON.*parseObject*(s, typeReference);  
 return t;  
 }  
 //利用fastjson进行逆转  
 public <T> T getData(TypeReference<T> typeReference) {  
 Object data = get("data");  
 String s = JSON.*toJSONString*(data);  
 T t = JSON.*parseObject*(s, typeReference);  
 return t;  
 }  
 public R setData(Object data) {  
 put("data", data);  
 return this;  
 }  
 public R() {  
 put("code", 0);  
 put("msg", "success");  
 }  
 public static R error() {  
 return *error*(HttpStatus.*SC\_INTERNAL\_SERVER\_ERROR*, "未知异常，请联系管理员");  
 }  
 public static R error(String msg) {  
 return *error*(HttpStatus.*SC\_INTERNAL\_SERVER\_ERROR*, msg);  
 }  
 public static R error(int code, String msg) {  
 R r = new R();  
 r.put("code", code);  
 r.put("msg", msg);  
 return r;  
 }  
 public static R ok(String msg) {  
 R r = new R();  
 r.put("msg", msg);  
 return r;  
 }  
 public static R ok(Map<String, Object> map) {  
 R r = new R();  
 r.putAll(map);  
 return r;  
 }  
  
 public static R ok() {  
 return new R();  
 }  
 public R put(String key, Object value) {  
 super.put(key, value);  
 return this;  
 }  
 public Integer getCode() {  
 return (Integer) this.get("code");  
 }  
}

项目采用分布式框架SpringCloud、Nacos。使用Gateway gateway+shiro完成认证权限（shiro主要体现在访问哪些接口需要token验证，哪些接口可以跳过，例如，注册和获取登录校验图片时，是比不可能需要token的）一些不需要登录也能访问的页面也可以通过shiro验证。