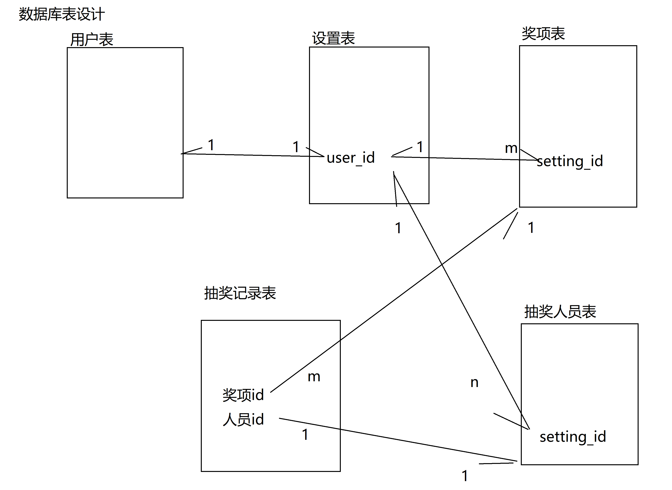
# 一.数据库表的设计



## Q:为什么设置一个设计表？

1对1关联的表，其实可以只使用一张表保存所有字段

但是可能出现业务扩展，方便系统扩展用，所以设计表时，可能考虑1对1设计

（比如一个本来是一个用户可以使用设置抽奖系统，如果之后要改成这个抽奖系统一个学校里面的用户都可以设置，如果我们没有设置表，就很难办。有设置表的话，我们只需要在将设置表和学校进行绑定即可，学校和用户1对多，只要是一个学校的都可以进行设置）

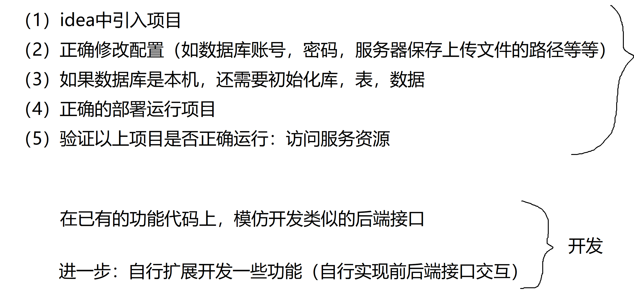
# 二.大致模板



* base：
* BaseMapper—所有Mapper的父接口（泛型），根据主键进行增删查改的操作
* JSONResponse—统一的响应体格式
* config
* interceptor—前后端拦截器（具体的拦截逻辑，以及拦截后做什么，这里前后端的拦截器写成了一个，通过判断url区分前端还是后端）
* web-RequestResponseBodyMethodProcessorWrapper—统一格式封装
* AppConfig—配置拦截器拦截的路径，以及所有后端路径前面都加上/api
* ExceptionAdvice—异常的统一处理（分自定义异常和非自定义异常）
* exception
* AppException—自定义异常，有一个code字段用于存放错误码，错误信息的话，父类RuntimeException里面可以存放
* tool—工具，像是Mybatis代码自动生成工具

剩下的就是前端界面了，自己写的html页面放在了public里面，第三方框架以及一些js和css放在static里面

# 三.项目要求

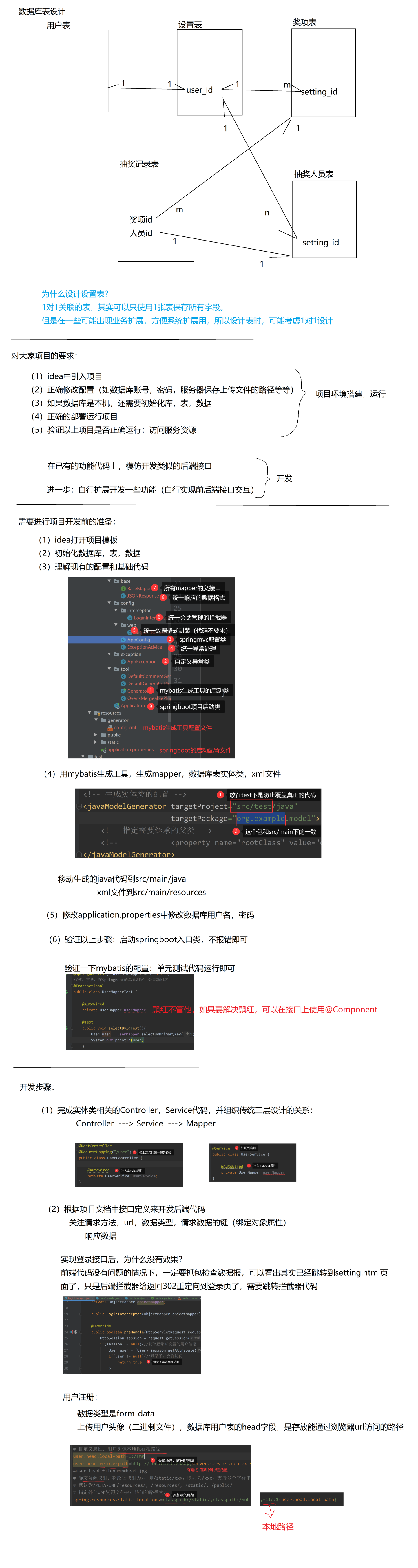


后端：比如我有一个注册功能，你要做一个类似的登录功能；我这里有一个修改的功能，你要把我的删除功能做出来

再进一步就是前后端全部自己弄，自己新加一个功能上去

# 四.开发步骤

## 1.开发前准备



## 2.开发步骤（注意设计的思路，先大局后细节）

### （1）.创建Controller包和Service包

1.根据我们的实体类，为每个实体类创建对应的Controller和Service

2.如图组织传统三层设计的关系并加上注解（Controller里面注入Service，Service里面注入Mapper）—autowired注入

首先是Controller整个类要加上@RestController表示返回类型为对象，@RequestMapping映射的路径（该业务的统一前缀）。

然后是Service的类上面加上Service注解

3.在之后方法上加注解的时候想一想这个方法参数的数据类型是什么，要是json的，对应的方法参数前面就要加上@RequestBody，其他的就不用加，默认使用pojo



### （2）.实现用户登录

##### 1）.Controller

1.根据文档，

* 我们可以知道登录的请求使用的是post请求，方法前面加上@postMapping
* 请求的数据类型是json类型的，方法参数传入的user前面加上@RequestBody

2.我们根据前端返回的用户对象的用户名去查，返回一个用户对象

* 要是找不到，抛异常
* 找到了通过数据库返回对象里面的密码和前端返回的user对象里面的密码校验，要是密码错，抛异常
* 密码正确，将用户信息保存到session里面，方法参数里面加上HttpServletRequest，然后从里面获取session，并且把数据库里面返回的用户对象设置进去（因为信息更全，前端返回用户对象只有账号和密码）就行。（注意这里设置的键要和拦截器里面取出来的键一样，因为拦截器是从session里面去取出user对象来判断用户是否登录的）

3.最后返回null即可，这里不需要自己设置响应的格式，响应头等，是因为在我们调用这个api的时候web服务器自动帮我们加上去了

##### 2）.Service

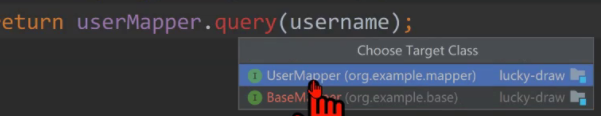
使用工具生成方法体，我们移动的query函数点击create即可。

根据username查找一个user对象，直接returnMapper.方法

**public User query(String username) {  
 return userMapper.query(username);  
}**

##### 3）.Mapper

一样的道理，使用工具生成方法体



直接生成成功了。

**@Mapper  
@Component  
public interface UserMapper extends BaseMapper<User> {  
 User query(String username);  
}**

这里不加上@Component会飘红报错，不过不影响。

###### 对应的xml

我们在Resources包下的Mapper包下的对应的xml文件里面配置好SQL语句，注意这里我们自己加的SQL要放在最后边，前面那些是默认生成的，不去动它

##### 4）.一些问题

1.我们要多使用抓包工具去判断一些错误：

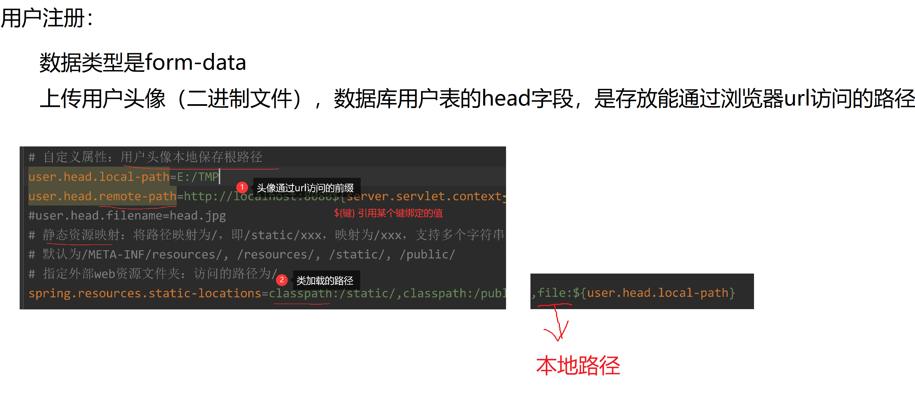


2.如果我们想重新回到登录界面，但是不改变url，也没有实现注销功能的情况下，我们可以将浏览器里面的cookie删除

### （3）.用户注册

##### 1）.Controller

1.上传的头像是二进制文件，但是数据库里面存的head是一个url



2.注意请求数据绑定user对象中的属性，及headFile变量名，要一致

请求的数据类型是form表单类型，所以我们user直接用pojo对象，headFile用二进制文件类型MultipartFile。现在user里面head空的，之后我们在service里面把上传的二进制文件保存在本地，然后把url设置到user对象里面。

**@PostMapping("/register")  
public Object register(User user, MultipartFile headFile){  
 userService.register(user, headFile);  
 return null;  
}**

3.调用service里面的register方法，直接用工具生成即可。

注意：controller主要是用来接收请求，对格式的判断，判空等，然后返回响应；具体的业务是放在service里面的。

##### 2）.Service

1.大致流程

* 如果有上传用户头像，需要保存在服务端本地，并且部署好，能通过url访问
* 用户头像的url，设置到user对象的head属性
* 用户注册，插入用户数据到数据库
* 用户注册并登陆后，跳转到设置页面，页面初始化就查询设置表数据绑定的奖项，人员信息，所以注册时，再插入一条和用户绑定的设置数据

2.这些内容为什么写在service里面呢，因为controller主要是用来接收请求，对格式的判断，判空等，然后返回响应；具体的业务是放在service里面的。

###### 1.配置本地文件路径

**#servlet上下文路径  
#server.servlet.context-path=/lucky-draw  
# 自定义属性：用户头像本地保存根路径  
user.head.local-path=D:/a\_zebra/0\_a\_Java/Project/a-project-data/lucky-draw  
user.head.remote-path=http://localhost:8080${server.servlet.context-path:}  
#user.head.filename=head.jpg  
# 静态资源映射：将路径映射为/，即/static/xxx，映射为/xxx，支持多个字符串，逗号间隔  
# 默认为/META-INF/resources/, /resources/, /static/, /public/  
# 指定外部web资源文件夹：访问的路径为/  
spring.resources.static-locations=classpath:/static/,classpath:/public/,file:${user.head.local-path}**

1.local-path表示的是本地用来保存文件的位置

2.remote-path表示的是ip+端口号+上下文路径，这里通过${}去获取上下文路径

3.然后配置静态资源映射，将/static/xxx,/public/xxx都配置为/xxx，同理，对应的D:/a\_zebra/0...../xxx映射为/xxx

4.将本地路径和相对路径注入到service中

**@Value("${user.head.local-path}")  
private String headLocalPath;  
  
@Value("${user.head.remote-path}")  
private String headRemotePath;**

###### 2.生成文件路径并保存文件

1.经过之前的配置，我们本地的路径就是headlocalPath+uri,远程访问路径就是headRemotePath+uri，所以我们需要为每个图片生成一个唯一的不重复的uri。保存时，需要考虑localPath后的相对路径是在并发情况下唯一，这里我们使用UUID，它能保证并发情况下唯一。

2.首先我们随机生成一个UUID，作为一个文件夹名，然后生成uri，/uuid/图片名称，名称通过headFile获取

**String uuid = UUID.randomUUID().toString();  
//使用上传时的图片名称保存到本地  
String uri = "/" + uuid + "/" + headFile.getOriginalFilename();  
//本地就是保存在headLocalPath+uri，http访问就是headRemotePath+uri  
String localPath = headLocalPath + uri;**

3.根据本地路径生成一个文件夹对象，然后获取它的父文件夹对象，如果父文件夹没有创建，我们就创建一个，最后将文件transfer也就是保存到file对象。然后将远程访问路径设置到user里面。

**//不能直接保存，因为父文件夹可能还没创建  
File head = new File(localPath);  
File parent = head.getParentFile();  
if(!parent.exists()) parent.mkdirs();  
headFile.transferTo(head);  
//用户头像的url，设置到user对象的head属性  
user.setHead(headRemotePath+uri);**

4.将设置好所有属性的user插入到数据库中

5.插入一条和用户绑定的设置数据，之后我们登陆需要根据用户id将当前用户设置的奖项，以及存放的人员信息全部加载到界面上，所以我们要在setting表里面插入一条有用户id的数据

**Setting setting = new Setting();  
setting.setUserId(user.getId());  
setting.setBatchNumber(8);  
settingMapper.insertSelective(setting);**

这里的setBatchNumber表示每次抽奖的人数，这里我们默认设置8个

6.这里我们还要处理一下异常，抛出一个自定义异常即可（注意把e也带上）；整个方法前面还要加上一个Transactional事务（这里在user表里面插入了数据，后面再setting表里面也插入了数据；如果后面插入的时候出现了异常，那么前面的数据也不应该插入，所以使用事务，要么全部成功插入，一个抛异常就回滚），这样就算中间出现异常还是可以往下执行

### （4）.查询抽奖设置

##### 1）.Controller

0.首先注入对应的service

**@Autowired  
private SettingService settingService;**

1.根据文档，请求中没有数据，我们需要返回的除了我们之前setting里面的属性以外，对应的setting我们还需要有user，award和member，所以我们在setting实体类里面加上对应的属性

**private User user;  
private List<Award> awards;  
private List<Member> members;**

2.没有请求数据，所以我们要根据session里面的用户信息去查询这些数据，根据用户id去查，user我们这里已经有了，所以我们直接在这里把user设置进去即可，不需要去数据库里面查询了。

**//返回的字段是setting对象属性，缺少的属性先构造好  
//需要setting对象，可以通过user\_id从数据库查询  
//user\_id从session中的user获取  
User user = (User) session.getAttribute("user");  
Setting setting = settingService.query(user.getId());**

**setting.setUser(user);  
return setting;**

##### 2）.Service

1.注入三个Mapper

**@Autowired  
private SettingMapper settingMapper;  
  
@Autowired  
private AwardMapper awardMapper;  
  
@Autowired  
private MemberMapper memberMapper;**

2.

根据用户id在setting表里面查到setting，返回的是一个setting对象

根据settingid在award表里面查到award，并且设置到setting对象里面

根据settingid在member表里面查到member，并且设置到setting对象里面

**Setting setting = settingMapper.queryByUserId(userId);  
List<Award> awards = awardMapper.queryBySettingId(setting.getId());  
setting.setAwards(awards);  
List<Member> members = memberMapper.queryBySettingId(setting.getId());  
setting.setMembers(members);  
return setting;**

##### 3）.Mapper

直接用工具生成，剩下的就是写对应的xml，注意这个id和award的xml下面这个settingId是对应的，因为这里只有一个属性，所以属性名称可以不一样，但是如果有多个名称的话属性名必须相同。

**List<Award> queryBySettingId(Integer id);**

**where a.setting\_id=#{settingId}**

###### 对应的xml（include refid简便方式）

1.这个是之前用generator自动生成的部分，方便后面使用

**<sql id="Base\_Column\_List">  
 id, user\_id, batch\_number, create\_time  
</sql>**

2.写好id，绑定好结果集，然后我们写语句的时候就可以使用include refid，就不需要一个个属性去写了

**<select id="queryByUserId" resultMap="BaseResultMap">  
 select  
 <include refid="Base\_Column\_List" />  
 from setting  
 where user\_id=#{userId}  
</select>**

##### 4）.中间人员记录数据（使用外连接，collection）

我们点击去抽奖的界面，这个界面需要一个记录数据，记录每个奖项对应的获奖的人

1.在award实体类里面加上

**private List<Integer> luckyMemberIds;**

2.所以award表和member表是一对多的关系，record表和member表是一对一的关系，所以award表和record表也是一对多的关系，因此我们可以在awardMapper的xml文件里面使用collection标签，我们之后查询出来只有一个member\_id，所以也不需要设置前缀和结果集了，设置一下返回的数据类型，即可，这里的collection就对应的是luckyMemberIds数组，最后就能实现一对多方式

**<collection property="luckyMemberIds" ofType="java.lang.Integer">  
 <result column="member\_id" />  
</collection>**

3.这里我们award关联record进行查询，要是使用内连接，那么当没有中奖记录的时候，连奖项都查询不出来，所以必须使用左外连接，可以保证award表的内容一定可以查询到。

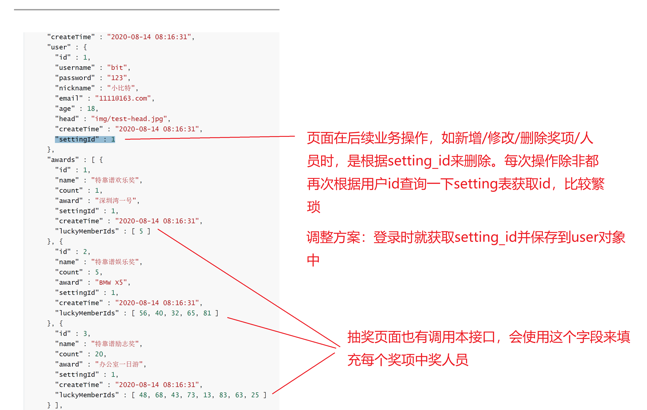
**from award a  
 left join record r  
 on a.id=r.award\_id  
 where a.setting\_id=#{settingId}**

4.查询的时候最后加上一个member.id就可以了。

**a.id,a.name,a.count,a.award,a.setting\_id,a.create\_time,r.member\_id**

文档里面这个settingid可能是用于抽完奖以后根据用户获取到settingid，方便后面的操作，这样就不需要每次都去查找了。

##### 5）.settingId和luckyMemberIds的作用



所以我们在userMapper.xml文件里面修改一下，把settingId也查询出来

**select  
 u.id,u.username,u.password,u.nickname,u.email,u.age,  
 u.head,u.create\_time,.id setting\_id  
 from user u  
 join setting s on u.id=s.user\_id  
 where u.username=#{username}  
</select>**

在user实体类里面加上settingId属性，然后在xml里面的结果集配置里面把数据库属性别名setting\_id和实体类属性名settingId绑定起来。

**<result column="setting\_id" property="settingId" />**

### （5）.注销功能

在userController里面直接把session remove掉即可

**@GetMapping("/logout")  
public Object logout(HttpSession session){  
 session.removeAttribute("user");  
 return null;  
}**

### （6）.修改抽奖人数（封装类型，不传入就是null，不修改）

我们自动生成的Mapper里面已经有根据主键更新字段值的接口了（根据主键修改对应的字段值，setting表里面只有一个主键，就是settingId，我们这里就是根据settingId来修改值的），传入一个setting对象，然后修改对应setting对象中的内容，如果有就会修改，没有不会修改（因为我们使用的都是封装类，所以不传入默认值就是null）。

工具生成service里面的内容，然后在service里面调用Mapper对应的接口。

**@GetMapping("/update")  
public Object update(HttpSession session, Integer batchNumber){  
 User user = (User) session.getAttribute("user");  
 Setting setting = new Setting();  
 setting.setId(user.getSettingId());  
 setting.setBatchNumber(batchNumber);  
 int n = settingService.update(setting);  
 return null;  
}**

### （7）.新增奖项（直接根据主键插入）

1.controller

请求类型是post，请求数据是application/json，所以使用@RequestBody注解，同时我们还要获取到settingId，进行新增奖项。

在award里面把settingId设置进去，然后就会根据主键插入值，我们这里实际上就是把award插入进去就可以了，这里是根据award里面的自增id插入一条条数据的

**User user = (User) session.getAttribute("user");  
award.setSettingId(user.getSettingId());  
int n = awardService.insert(award);  
return award.getId();**

2.service

直接调用生成的接口即可。

**return awardMapper.insertSelective(award);**

### （8）.修改奖项（根据主键修改，反正主键又改不到）

1.和新增奖项一样，post，json，可以使用生成的接口，传入一个award，会根据id修改有的值，这里我们修改的时候传入的award对象里面的id是肯定有的，新增的时候就已经确定了，而且是不能改的。

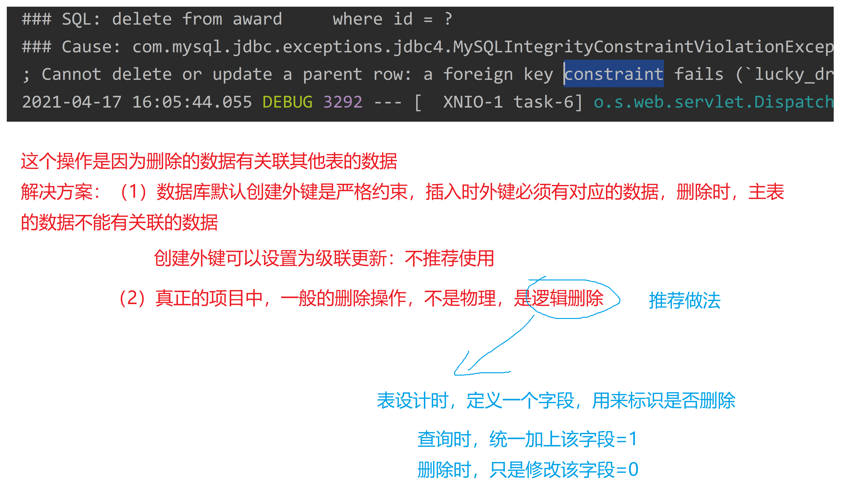
### （9）.删除奖项

1.注意文档里面请求的url是delete/4，传入参数为url上的内容，所以我们要加上pathVairable注解获取到url里面的id，然后根据id进行删除。

这里也不需要自己写xml，有写好的接口可以直接用。

**@GetMapping("/delete/{id}")  
public Object delete(@PathVariable Integer id){  
 int n = awardService.delete(id);  
 return null;  
}**

##### 删除报错（删除数据有关联别的表—记录表）



这里还有个bug，新增那里要返回奖品id才行，不然删除会出问题（可能删除就是根据这个id删除的，保存在前端了。要是新增没有返回id，会新增成功，但是删除会出现bug）

### （10）.新增，修改，删除抽奖人员

和前面的新增，修改，删除奖项基本一样。自己去看代码。

### （11）.抽奖（记得要加上@Param）

1.post，从url里面获取awardId，json传入一个list数组

**@PostMapping("/add/{awardId}")  
public Object add(@PathVariable Integer awardId,  
 @RequestBody List<Integer> memberIds){  
 int n = recordService.add(awardId, memberIds);  
 return null;  
}**

2.抽奖是在前端抽的，当前抽奖的所有信息在前端都有，后端只负责将抽奖完的结果放到数据库中。

3.这里要注意在Mapper里面的方法参数上要加上param注解

**int batchInsert(@Param("awardId") Integer awardId,  
 @Param("memberIds") List<Integer> memberIds);**

4.字段只有两个，所以直接写在后面就行，然后使用for each循环，循环的是memberids，一个awardId是同一个

**<insert id="batchInsert">  
 insert into record(member\_id, award\_id)  
 values  
 <foreach collection="memberIds" item="item" separator=",">  
 (#{item}, #{awardId})  
 </foreach>  
</insert>**

5.后端只是记录，抽奖在前端，没有任何算法，实现方式也很简单



### （12）.删除当前奖项某个获奖人员/删除当前奖项已获奖人员

1.根据人员id来删除，注意这里给id的方式是通过参数的形式给的，而不是url

直接用Integer id来进行接收即可



**@GetMapping("/delete/award")  
public Object deleteByAwardId(Integer id){  
 int n = recordService.deleteByAwardId(id);  
 return null;  
}**

2.根据人员删除和根据奖项删除可以用一个方法来实现，传入一个record对象，哪个不为空，我就根据什么删

**public int deleteByAwardId(Integer id) {  
 Record r = new Record();  
 r.setAwardId(id);  
 return recordMapper.delete(r);  
}**

**public int deleteByMemberId(Integer id) {  
 Record r = new Record();  
 r.setMemberId(id);  
 return recordMapper.delete(r);  
}**

3.直接使用if进行判断即可，哪个不为空，我就用哪一个删

**<delete id="delete">  
 delete from record  
 <trim prefix="where" prefixOverrides="and">  
 <if test="awardId != null">  
 and award\_id=#{awardId}  
 </if>  
 <if test="memberId != null">  
 and member\_id=#{memberId}  
 </if>  
 </trim>  
</delete>**

### （13）.重置抽奖结果

多表关联删除数据的语法自己可以去搜索一下，delete t1 from t1,t2表示删除t1表中的数据，但是where后面的条件可以使用t1和t2中的数据。

