

# 2-后端开发

## 本节重点

开发面试刷题平台后端，包括：

- 需求分析
- 库表设计
- 后端项目初始化（万用模板讲解）
- 后端基础功能开发（增删改查）
- 后端核心业务流程开发

### 一、需求分析

目标是明确要做的需求，并且给需求设置优先级，从而明确开发计划。

### 项目功能梳理

#### 基础功能

- 用户模块
- 
- 用户注册
- 用户登录（账号密码）
- 【管理员】管理用户 - 增删改查
- 题库模块
- 
- 查看题库列表
- 查看题库详情（展示题库下的题目）
- 【管理员】管理题库 - 增删改查
- 题目模块
- 
- 题目搜索

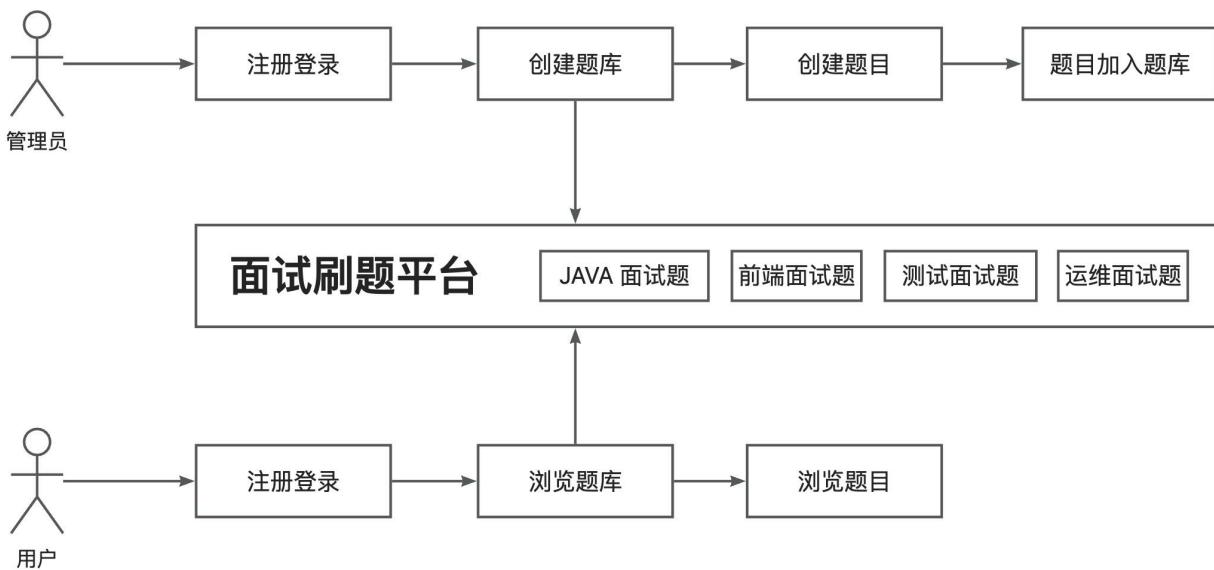
- 查看题目详情（进入刷题页面）
- 【管理员】管理题目 - 增删改查（比如按照题库查询题目、修改题目所属题库等）

## 高级功能

- 题目批量管理
- 
- 【管理员】批量向题库添加题目
- 【管理员】批量从题库移除题目
- 【管理员】批量删除题目
- 分词题目搜索
- 用户刷题记录日历图
- 自动缓存热门题目
- 网站流量控制和熔断
- 动态 IP 黑白名单过滤
- 同端登录冲突检测
- 分级题目反爬虫策略

## 核心业务流程

如下图：



## 需求优先级

根据核心业务业务流程，明确需求开发的优先级。

- P0 为核心，非做不可

- P1 为重点功能，最好做
- P2 为实用功能，有空就做
- P3 可做可不做，时间充裕再做

排好优先级的需求列表如下，其实用表格的形式整理更好：

## 基础功能 (均为 P0)

- 用户模块
  - 用户注册
- 用户登录 (账号密码)
- 【管理员】管理用户 - 增删改查
- 题库模块
  - 查看题库列表
- 查看题库详情 (展示题库下的题目)
- 【管理员】管理题库 - 增删改查
- 题目模块
  - 题目搜索
- 查看题目详情 (进入刷题页面)
- 【管理员】管理题目 - 增删改查 (比如按照题库查询题目、修改题目所属题库等)

## 高级功能 (均为 P1 ~ P2)

- 题目批量管理 P1
- - 【管理员】批量向题库添加题目
- 【管理员】批量从题库移除题目
- 【管理员】批量删除题目
- 分词题目搜索 P1
- 用户刷题记录日历图 P1
- 自动缓存热门题目 P2

- 网站流量控制和熔断 P2
- 动态 IP 黑白名单过滤 P2
- 同端登录冲突检测 P2
- 分级题目反爬虫策略 P2

排好优先级后，后端和前端同学就可以根据优先级去设计接口和页面了。

企业中一般使用专业的系统或者表格来管理需求。下图仅为示例，使用腾讯文档的智能表实现：



The screenshot shows a screenshot of a Tencent Document intelligent table titled "AI Answering Platform [Fish Answer] Requirement Scheduling Table". The table has the following columns: ①开始日期 (Start Date), ②项目模块 (Module), ③工作任务 (Task), ④负责人 (Responsible Person), ⑤优先级 (Priority), ⑥类型 (Type), ⑦工作详情 (Work Details), ⑧工作进度 (Work Progress), and ⑨相关链接 (Related Links). The table lists 22 requirements, each with a date from May 9, 2024, and a module like "User Module" or "Questionnaire Module". The "Work Progress" column shows values like 0%, 0.5%, or 100%.

## 二、库表设计

对应需求分析中的功能梳理的模块，有 4 张核心表。

库名：mianshiya

数据库初始化文件：见下方

每个表鱼皮都给出了基础版和可扩展字段，便于大家学习。

### 1、用户表

#### 核心设计

用户表的核心是用户登录凭证（账号密码）和个人信息，SQL 如下：

```

1  -- 用户表
2  create table if not exists user
3  (
4      id          bigint auto_increment comment 'id' primary key,
5      userAccount  varchar(256)           not null comment '账号'
6      userPassword  varchar(512)           not null comment '密码'
7      unionId      varchar(256)           null comment '微信开放
8      mpOpenId     varchar(256)           null comment '公众号op
9      userName     varchar(256)           null comment '用户名称
10     userAvatar   varchar(1024)          null comment '用户头像
11     userProfile  varchar(512)           null comment '用户简介
12     userRole     varchar(256) default 'user'      not null comment '用户
13     editTime     datetime   default CURRENT_TIMESTAMP not null comment '编辑
14     createTime    datetime   default CURRENT_TIMESTAMP not null comment '创建
15     updateTime    datetime   default CURRENT_TIMESTAMP not null on update CU
16     isDelete     tinyint    default 0           not null comment '是否
17     index idx_unionId (unionId)
18  ) comment '用户' collate = utf8mb4_unicode_ci;

```

其中，unionId、mpOpenId 是为了实现公众号登录的，也可以省略。每个微信用户在同一家公司（主体）的unionId 是唯一的，在同一个公众号的mpOpenId 是唯一的。

editTime 和 updateTime 的区别是：editTime 表示用户编辑个人信息的时间（需要业务代码来更新），而 updateTime 表示这条用户记录任何字段发生修改的时间（由数据库自动更新）。

## 扩展设计

1) 如果要实现会员功能，可以对表进行如下扩展：

1. 给 userRole 字段新增枚举值 `vip`，表示会员用户，可根据该值判断用户权限
2. 新增会员过期时间字段，可用于记录会员有效期
3. 新增会员兑换码字段，可用于记录会员的开通方式
4. 新增会员编号字段，可便于定位用户并提供额外服务，并增加会员归属感

对应的 SQL 如下：

```

1  vipExpireTime datetime    null comment '会员过期时间',
2  vipCode        varchar(128) null comment '会员兑换码',
3  vipNumber      bigint      null comment '会员编号'

```

2) 如果要实现用户邀请功能，可以对表进行如下扩展：

- 新增 shareCode 分享码字段，用于记录每个用户的唯一邀请标识，可拼接到邀请网址后面，比如 <https://mianshiya.com/?shareCode=xxx>
- 新增 inviteUser 字段，用于记录该用户被哪个用户邀请了，可通过这个字段查询某用户邀请的用户列表。

对应的 SQL 如下：

```
1  shareCode      varchar(20)  DEFAULT NULL COMMENT '分享码',
2  inviteUser     bigint       DEFAULT NULL COMMENT '邀请用户 id'
```

## 2、题库表

### 核心设计

题库表的核心是题库信息（标题、描述、图片），SQL 如下：

```
1  -- 题库表
2  create table if not exists question_bank
3  (
4      id          bigint auto_increment comment 'id' primary key,
5      title        varchar(256)           null comment '标题',
6      description  text                null comment '描述',
7      picture      varchar(2048)          null comment '图片',
8      userId       bigint              not null comment '创建用户',
9      editTime     datetime default CURRENT_TIMESTAMP not null comment '编辑时间',
10     createTime    datetime default CURRENT_TIMESTAMP not null comment '创建时间',
11     updateTime   datetime default CURRENT_TIMESTAMP not null on update CURRENT_TIMESTAMP comment '更新时间',
12     isDelete     tinyint  default 0      not null comment '是否删除',
13     index idx_title (title)
14  ) comment '题库' collate = utf8mb4_unicode_ci;
```

其中，picture 存储的是图片的 url 地址，而不是完整图片文件。通过 userId 和用户表关联，在本项目中只有管理员才能创建题库。

由于用户很可能按照标题搜索题库，所以给 title 字段增加索引。

### 扩展设计

1) 如果要实现题库审核功能，可以对表进行如下扩展：

- 新增审核状态字段，用枚举值表示待审核、通过和拒绝

2. 新增审核信息字段，用于记录未过审的原因等

3. 新增审核人 id 字段，便于审计操作。比如出现了违规内容过审的情况，可以追责到审核人。

4. 新增审核时间字段，也是便于审计。

对应的 SQL 如下：

```
1  reviewStatus  int      default 0  not null comment '状态: 0-待审核, 1-通过, 2-拒
2  reviewMessage varchar(512)      null comment '审核信息',
3  reviewerId    bigint      null comment '审核人 id',
4  reviewTime     datetime    null comment '审核时间'
```

2) 如果要实现题库排序功能，可以新增整型的优先级字段，并且根据该字段排序。

该字段还可以用于快速实现题库精选和置顶功能，比如优先级 = 1000 的题库表示精选，优先级 = 10000 的题库表示置顶。

对应的 SQL 如下：

```
1  priority  int  default 0  not null comment '优先级'
```

3) 如果要实现题库浏览功能，可以新增题库浏览数字段，每次进入题目详情页时该字段的值 +1，还可以根据该字段对题库进行排序。

对应的 SQL 如下：

```
1  viewNum  int  default 0  not null comment '浏览量'
```

如果要实现用户浏览数（同一个用户浏览数最多 +1），还需要额外的题库浏览记录表。

### 3、题目表

题目表的核心是题目信息（标题、详细内容、标签），SQL 如下：

```

1  -- 题目表
2  create table if not exists question
3  (
4      id      bigint auto_increment comment 'id' primary key,
5      title   varchar(256)           null comment '标题',
6      content  text                null comment '内容',
7      tags     varchar(1024)          null comment '标签列表 (json)'
8      answer   text                null comment '推荐答案',
9      userId   bigint              not null comment '创建用户 id',
10     editTime  datetime default CURRENT_TIMESTAMP not null comment '编辑时间',
11     createTime datetime default CURRENT_TIMESTAMP not null comment '创建时间',
12     updateTime datetime default CURRENT_TIMESTAMP not null on update CURRENT_TIMESTAMP comment '最后更新时间',
13     isDelete  tinyint default 0   not null comment '是否删除',
14     index idx_title (title),
15     index idx_userId (userId)
16  ) comment '题目' collate = utf8mb4_unicode_ci;

```

几个重点：

1. 题目标题 title 和题目创建人 userId 是常用的题目搜索条件，所以添加索引提升查询性能。
2. 题目可能有多个标签，为了简化设计，没有采用关联表，而是以 JSON 数组字符串的方式存储，比如 `["Java", "Python"]`。
3. 题目内容（详情）和题目答案可能很长，所以使用 text 类型。

## 扩展设计

题目表有很多可以扩展的方法，下面举一些例子。

- 1) 如果要实现题目审核功能，可以参考上述题库审核功能，新增 4 个字段即可：

```

1  reviewStatus  int      default 0  not null comment '状态: 0-待审核, 1-通过, 2-拒绝',
2  reviewMessage varchar(512)      null comment '审核信息',
3  reviewerId    bigint           null comment '审核人 id',
4  reviewTime    datetime          null comment '审核时间'

```

- 2) 可能有很多评价题目的指标，比如浏览数、点赞数、收藏数，参考字段如下：

```

1  viewNum      int      default 0  not null comment '浏览量',
2  thumbNum     int      default 0  not null comment '点赞数',
3  favourNum    int      default 0  not null comment '收藏数'

```

3) 如果要实现题目排序、精选和置顶功能，可以参考上述题库表的设计，新增整型的优先级字段，并且根据该字段排序。对应的 SQL 如下：

```
1 priority int default 0 not null comment '优先级'
```

4) 如果题目是从其他网站或途径获取到的，担心有版权风险，可以增加题目来源字段。最简单的实现方式就是直接存来源名称：

```
1 source varchar(512) null comment '题目来源'
```

5) 如果想设置部分题目仅会员可见，可以给题目表加上一个“是否仅会员可见”的字段，本质上是个布尔类型，用 1 表示仅会员可见。参考 SQL 如下：

```
1 needVip tinyint default 0 not null comment '仅会员可见(1 表示仅会员可见)'
```

## 4、题库题目关系表

### 核心设计

由于一个题库可以有多个题目，一个题目可以属于多个题库，所以需要关联表来实现。

实现基础功能的 SQL 如下：

```
1 -- 题库题目表(硬删除)
2 create table if not exists question_bank_question
3 (
4     id          bigint auto_increment comment 'id' primary key,
5     questionBankId bigint          not null comment '题库',
6     questionId   bigint          not null comment '题目',
7     userId       bigint          not null comment '创建用户',
8     createTime    datetime default CURRENT_TIMESTAMP not null comment '创建时间',
9     updateTime    datetime default CURRENT_TIMESTAMP not null on update CURRENT_TIMESTAMP,
10    UNIQUE (questionBankId, questionId)
11 ) comment '题库题目' collate = utf8mb4_unicode_ci;
```

几个重点：

1. 上述代码中的 userId 表示添加题目到题库的用户 id，仅管理员可操作
2. 由于关联表中的数据记录并没有那么重要（一般由管理员维护），所以直接采用硬删除的方式，如果将题目移出题库，直接删掉对应的数据即可。按照这种设计，createTime 就是题目

加入到题库的时间。

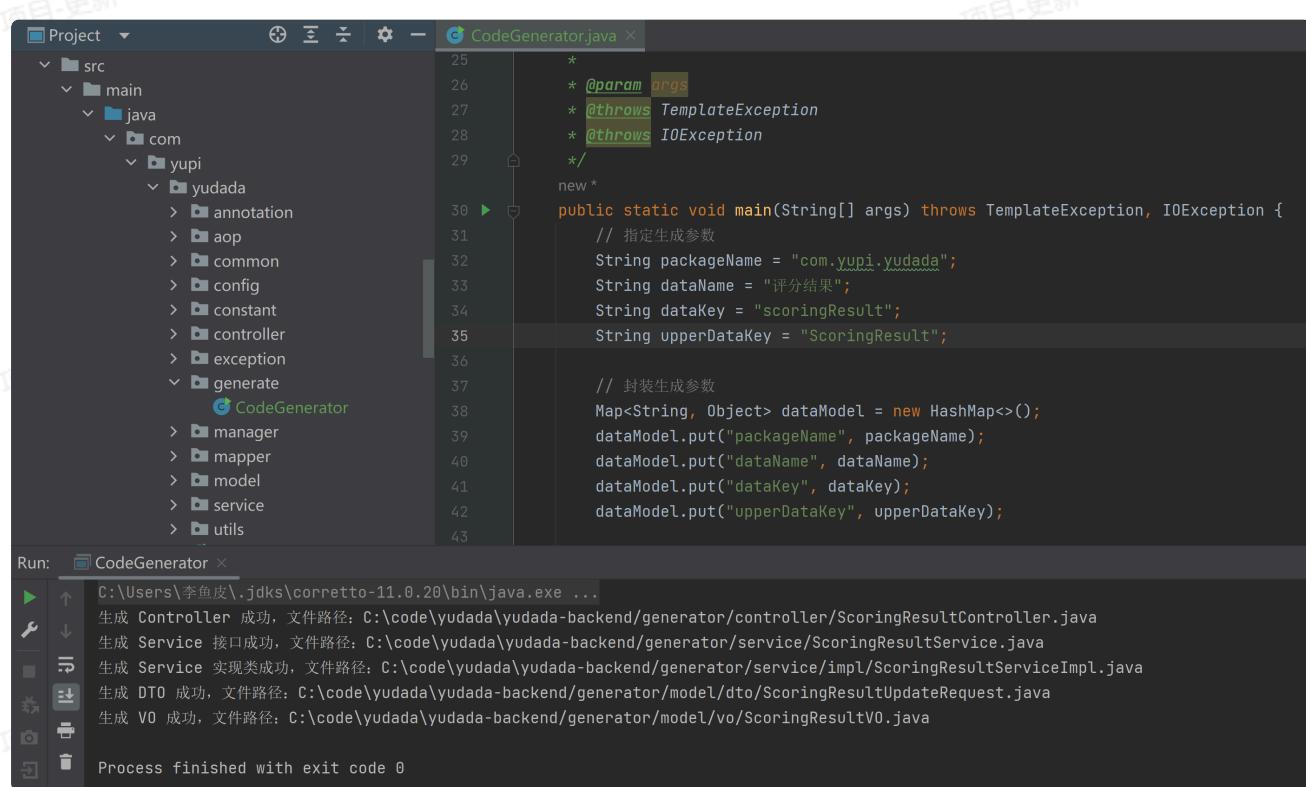
3. 通过给题库 id 和题目 id 添加 **联合唯一索引**，防止题目被重复添加到同一个题库中。而且要注意，将 questionBankId 放到前面，因为数据库中的查询大多是基于 questionBankId 进行的，比如查找某个题库中的所有问题，或者在一个题库中查找特定问题，将 questionBankId 放在前面符合查询模式，会使得这些查询更加高效（索引的最左前缀原则）。

## 扩展设计

1) 如果要对题库内的题目进行排序, 可以增加题目顺序字段(整型)。对应的 SQL 如下:

```
1     questionOrder  int  default 0  not null comment '题目顺序（题号）'
```

需要注意，如果要实现任意移动题目顺序的功能，可能每次要更新多条记录的顺序，比较影响性能。如果追求更高性能的话，可以先在内存中计算出要变更的题目顺序，以减少更新的记录数。比如将第 100 题移动到第 98 题，只需要修改几条记录的顺序，不影响前面的题目。



### 三、后端项目初始化

## 1) 后端万用模板介绍, 核心能力讲解

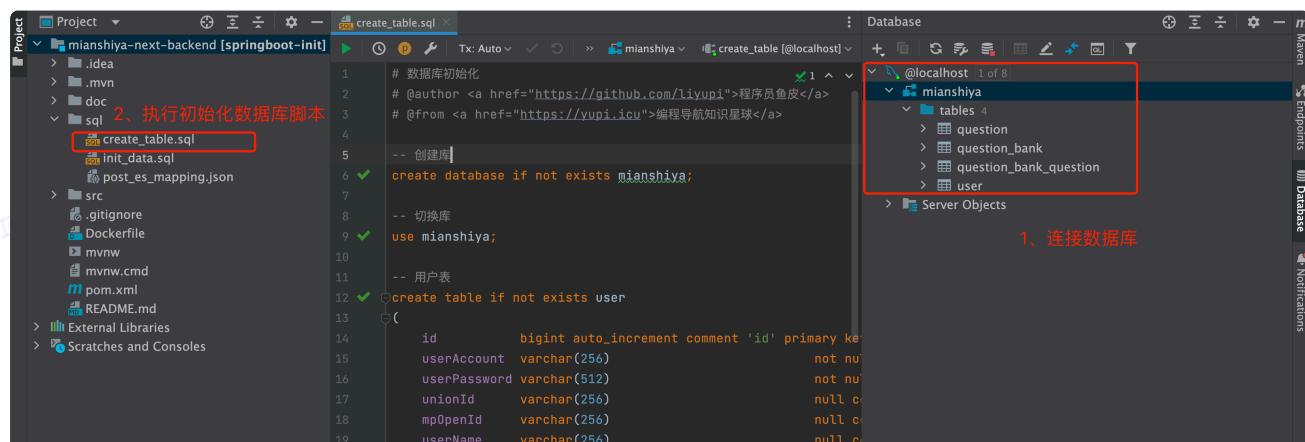
可以参考万用模板项目的 README.md 文件来了解模板的功能和启动方法。

后端万用模板介绍和源码下载: <https://www.codenav.cn/course/1826803928691945473/section/1826872096558985217?contentType=text&type=<https://www.codenav.cn/course/1826803928691945473/section/1826872096558985217?contentType=text&type=>>

## 2) 准备依赖

- MySQL 必须准备 8.x 版本
- Redis 默认不开启 (后续教程会用到)
- Elasticsearch 默认不开启 (后续教程会用到)
- COS 对象存储, 不影响运行

## 3) 执行模板的初始化 SQL 脚本, 创建数据库表:



The screenshot shows the MySQL Workbench interface. On the left, the 'Project' panel displays the 'mianshiya-next-backend [springboot-init]' project structure, including files like '.idea', '.mvn', 'doc', and 'sql'. The 'create\_table.sql' file is selected and shown in the central editor. The SQL script creates a database 'mianshiya' and several tables: 'question', 'question\_bank', 'question\_bank\_question', and 'user'. On the right, the 'Database' panel shows the 'mianshiya' database structure with the same four tables. A red box highlights the 'mianshiya' database in the tree view. A red line at the bottom right of the interface is labeled '1、连接数据库' (1. Connect to the database).

然后可以尝试运行项目。

## 4) 模板改造

1. 全局替换模块名, 由 `springboot-init` 改为 `mianshiya-next-backend` (或者自定义)
2. 全局修改包名, 重构 `com.yupi.springbootinit` + 全局替换 `springbootinit` 为 `mianshiya`
3. 按需移除不必要的模块, 比如 Elasticsearch、微信开发、表格处理、定时任务相关代码。不移除也不影响项目运行。
4. 执行本项目的数据库初始化文件, 修改 `application.yml` 配置, 更改数据库等配置为自己的 (比如将 `my_db` 改为 `mianshiya`)

自己做项目的话, 一定要用 Git 进行托管, 可以看到改动的文件记录。而且修改代码后可以提交一下代码, 万一改错了也可以很方便地回滚。

5) 启动项目，执行 MainApplication 主类文件即可。

推荐用 Debug 来启动项目，启动成功后，可以通过内置的 Swagger 接口文档来分析接口的请求参数和响应。

地址：<http://localhost:8101/api/doc.html#/home>

[<http://localhost:8101/api/doc.html#/home>](http://localhost:8101/api/doc.html#/home)

## 四、后端基础开发 - 增删改查

根据需求分析阶段整理好的项目功能，基础开发就是完成几张核心数据库表的增删改查，**先不包含任何复杂的业务逻辑。**

每个模块都要遵循如下的开发流程：

### 1、数据访问层代码生成

使用 MyBatisX 代码生成插件快速得到 mapper 和数据库实体类，生成后移动到项目对应位置 (mapper 和 model.entity 包) 。

Generate Options

module path	/Users/yupi/Code/mianshiya-next-backend		
base package	generator	encoding	UTF-8
base path	src/main/java	ignore field prefix	
relative package	domain	ignore field suffix	
extra class suffix		class name strategy	<input checked="" type="radio"/> camel <input type="radio"/> same as tablename
tableName	className		
question	Question		
question_bank	QuestionBank		
question_bank_question	QuestionBankQuestion		
user	User		

Previous Cancel Next

Generate Options

annotation	<input type="radio"/> None	<input checked="" type="radio"/> Mybatis-Plus 3																					
	<input type="radio"/> Mybatis-Plus 2	<input type="radio"/> JPA																					
options	<input checked="" type="checkbox"/> Comment	<input type="checkbox"/> toString/hashCode>equals																					
	<input checked="" type="checkbox"/> Actual Column	<input type="checkbox"/> Actual Column Annotation																					
	<input checked="" type="checkbox"/> Model	<input checked="" type="checkbox"/> Lombok																					
template	<input type="radio"/> custom-model-swagger	<input type="radio"/> default-all	<input type="radio"/> default-empty																				
	<input type="radio"/> mybatis-plus2	<input checked="" type="radio"/> mybatis-plus3	<input type="checkbox"/> JSR310: Date API																				
<table border="1"> <thead> <tr> <th>config name</th> <th>module path</th> <th>base path</th> <th>package name</th> </tr> </thead> <tbody> <tr> <td>mapperXml</td> <td>/Users/yupi/Code/mian...</td> <td>src/main/resources</td> <td>mapper</td> </tr> <tr> <td>serviceImpl</td> <td>/Users/yupi/Code/mian...</td> <td>src/main/java</td> <td>generator.service.impl</td> </tr> <tr> <td>mapperInterface</td> <td>/Users/yupi/Code/mian...</td> <td>src/main/java</td> <td>generator.mapper</td> </tr> <tr> <td>serviceInterface</td> <td>/Users/yupi/Code/mian...</td> <td>src/main/java</td> <td>generator.service</td> </tr> </tbody> </table>				config name	module path	base path	package name	mapperXml	/Users/yupi/Code/mian...	src/main/resources	mapper	serviceImpl	/Users/yupi/Code/mian...	src/main/java	generator.service.impl	mapperInterface	/Users/yupi/Code/mian...	src/main/java	generator.mapper	serviceInterface	/Users/yupi/Code/mian...	src/main/java	generator.service
config name	module path	base path	package name																				
mapperXml	/Users/yupi/Code/mian...	src/main/resources	mapper																				
serviceImpl	/Users/yupi/Code/mian...	src/main/java	generator.service.impl																				
mapperInterface	/Users/yupi/Code/mian...	src/main/java	generator.mapper																				
serviceInterface	/Users/yupi/Code/mian...	src/main/java	generator.service																				

Previous Cancel Finish

然后更改生成的数据库实体类的字段配置，指定主键策略和逻辑删除。

比如题目表，id 默认是连续生成的，容易被爬虫抓取，所以更换策略为 `ASSIGN_ID` 雪花算法生成。示例代码：

```

1  public class Question implements Serializable {
2      /**
3      * id (要指定主键策略)
4      */
5      @TableId(type = IdType.ASSIGN_ID)
6      private Long id;
7
8      // ...
9
10 /**
11 * 是否删除 (逻辑删除)
12 */
13 @TableLogic
14 private Integer isDelete;
15 }

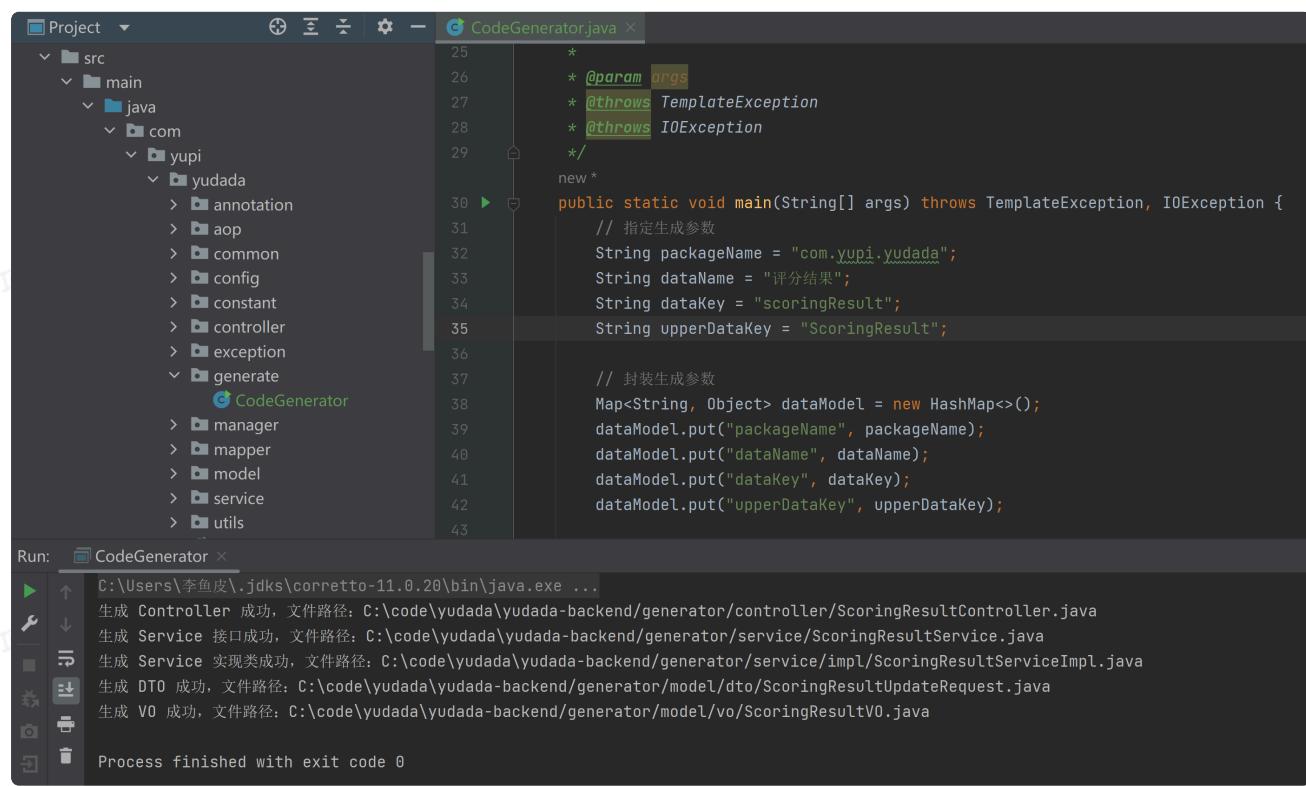
```

使用框架的过程中，有任何疑问，都可以在官方文档查阅，比如了解 MyBatis Plus 的主键生成注解：<https://baomidou.com/reference/annotation/#tableid>

## 2、业务逻辑代码生成

使用万用模板的代码生成器工具（CodeGenerator）生成代码，包括：Controller、Service 接口和实现类、数据模型包装类。

修改生成参数即可使用（用户表可以不生成，因为万用模板已经自带了用户业务逻辑代码）：



```

25      *
26      * @param args
27      * @throws TemplateException
28      * @throws IOException
29      */
30      public static void main(String[] args) throws TemplateException, IOException {
31          // 指定生成参数
32          String packageName = "com.yupi.yudada";
33          String dataName = "评分结果";
34          String dataKey = "scoringResult";
35          String upperDataKey = "ScoringResult";
36
37          // 封装生成参数
38          Map<String, Object> dataModel = new HashMap<>();
39          dataModel.put("packageName", packageName);
40          dataModel.put("dataName", dataName);
41          dataModel.put("dataKey", dataKey);
42          dataModel.put("upperDataKey", upperDataKey);
43
44      }

```

Run: CodeGenerator

- 生成 Controller 成功，文件路径: C:\code\yudada\yudada-backend\generator\controller\ScoringResultController.java
- 生成 Service 接口成功，文件路径: C:\code\yudada\yudada-backend\generator\service\ScoringResultService.java
- 生成 Service 实现类成功，文件路径: C:\code\yudada\yudada-backend\generator\service\impl\ScoringResultServiceImpl.java
- 生成 DTO 成功，文件路径: C:\code\yudada\yudada-backend\generator\model\DTO\ScoringResultUpdateRequest.java
- 生成 VO 成功，文件路径: C:\code\yudada\yudada-backend\generator\model\vo\ScoringResultVO.java

7954字

比如修改配置代码如下，生成题库相关的业务逻辑代码：

```
1 // 指定生成参数
2 String packageName = "com.yupi.mianshiya";
3 String dataName = "题库";
4 String dataKey = "questionBank";
5 // 注意，这里需要保持首字母大写
6 String upperDataKey = "QuestionBank";
```

生成之后，将生成的文件从 generator 包移动到对应的位置即可，包括 controller、service、model.dto、model.vo。

想了解生成代码的原理，自己也做一个生成器，可以学习鱼皮编程导航的 [代码生成器共享平台项目 <https://www.code-nav.cn/course/1790980795074654209>](https://www.code-nav.cn/course/1790980795074654209)。

### 3、数据模型开发

需要编写数据模型包装类（dto 包的请求类和 vo 包的视图类）、JSON 结构对应的类、枚举类。

**其中，包装类需要根据前端实际传递的请求参数或需要的响应结果自行修改。**

以创建题目请求包装类 QuestionAddRequest 为例，需要保留前端需要的字段，并且将 JSON 字符串字段转换为前端更好理解的数据类型（比如 tags 由 String 转为 List），还有一些由后端自动生成的字段（比如 userId、createTime）就不用写到类里了。代码如下：

```
1  @Data
2  public class QuestionAddRequest implements Serializable {
3
4      /**
5       * 标题
6       */
7      private String title;
8
9      /**
10     * 内容
11     */
12     private String content;
13
14     /**
15     * 标签列表
16     */
17     private List<String> tags;
18
19     /**
20     * 推荐答案
21     */
22     private String answer;
23
24     private static final long serialVersionUID = 1L;
25 }
```

其他的请求包装类同理，目前我们只实现基本的增删改查包装类，需要仔细地依次过一遍生成的代码，不需要用到的请求类可以移除。比如移除 QuestionBankQuestionEditRequest 类，因为不需要让用户编辑题库和题目的关系。

 小技巧：可以结合具体的业务、实体类（比如 Question）、以及创建表的 DDL 语句去编写请求包装类。

## 扩展

目前的表设计不涉及枚举字段的编写。

如果需要给题目增加审核功能，那么审核状态就是一个枚举字段。作为示例，给大家提供审核状态枚举类代码：

```
1 package com.yupi.yudada.model.enums;
2
3 import cn.hutool.core.util.ObjectUtil;
4
5 import java.util.Arrays;
6 import java.util.List;
7 import java.util.stream.Collectors;
8
9 /**
10  * 审核状态枚举
11  *
12  * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
13  *
14  * @from <a href="https://www.code-nav.cn">编程导航学习圈</a>
15  *
16  */
17 public enum ReviewStatusEnum {
18
19     REVIEWING("待审核", 0),
20     PASS("通过", 1),
21     REJECT("拒绝", 2);
22
23     private final String text;
24
25     private final int value;
26
27     ReviewStatusEnum(String text, int value) {
28         this.text = text;
29         this.value = value;
30     }
31
32     /**
33      * 根据 value 获取枚举
34      *
35      * @param value
36      * @return
37      */
38     public static ReviewStatusEnum getEnumByValue(Integer value) {
39         if (ObjectUtil.isEmpty(value)) {
40             return null;
41         }
42         for (ReviewStatusEnum anEnum : ReviewStatusEnum.values()) {
43             if (anEnum.value == value) {
44                 return anEnum;
45             }
46         }
47         return null;
48     }
49
50     /**
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
995
996
997
997
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1
```

```
51     * 获取值列表
52     *
53     * @return
54     */
55     public static List<Integer> getValues() {
56         return Arrays.stream(values()).map(item -> item.value).collect(Collectors.toList());
57     }
58
59     public int getValue() {
60         return value;
61     }
62
63     public String getText() {
64         return text;
65     }
66 }
```

其中，`getEnumByValue` 是通过循环找到枚举值，可以用 Map 缓存所有枚举值来加速查找。

## 4、接口开发

修改生成的 Controller 接口，不需要包含业务逻辑，处理字段不一致的问题、并且将无用的接口移除掉，能跑通就行。

生成的 Controller 接口结构都是一致的，只需要理解一个 Controller，其他的都很简单。

## 5、服务开发

修改生成的 Service 接口和实现类，处理字段不一致的问题，略微调整数据校验、查询条件等代码，能跑通就行。

生成的 Service 结构都是一致的，只需要理解一个，其他的都很简单。

## 6、Swagger 接口文档测试

运行项目，通过访问 Swagger 接口文档来测试增删改查接口能否正常执行。

地址：[<http://localhost:8101/api/doc.html#/home>](http://localhost:8101/api/doc.html#/home)

# 五、后端核心业务流程开发

# 核心业务接口梳理

根据之前整理的功能列表，分析核心业务流程需要调用的接口：

## 1) 用户模块

- 用户注册：已完成  万用模板自带
- 用户登录（账号密码）：已完成  万用模板自带
- 【管理员】管理用户 - 增删改查：已完成  万用模板自带

## 2) 题库模块

- 查看题库列表：分页获取题库接口，⌚ 已通过生成器生成，需要确认
- 查看题库详情（展示题库下的题目）：⌚ 根据 id 获取题库详情接口，需要开发
- 【管理员】管理题库 - 增删改查：⌚ 已通过生成器生成，需要确认

## 3) 题目模块

- 题目搜索：分页获取题目接口，⌚ 已通过生成器生成，需要确认
- 查看题目详情（进入刷题页面）：⌚ 根据 id 获取题目详情接口，需要确认
- 【管理员】管理题目 - 增删改查：⌚ 已通过生成器生成，需要确认
- 【管理员】按照题库查询题目：⌚ 根据题库 id 获取题目列表，需要开发
- 【管理员】修改题目所属题库等：⌚ 修改题目所属题库接口，需要开发

然后通过接口文档来完整测试一遍业务流程即可。

## 1、确认和完善接口

### 1) 查看题库列表：分页获取题库接口

需要修改题库的校验规则（validQuestionBank）和查询条件（getQueryWrapper）

### 2) 题目搜索：分页获取题目接口

需要修改题库的校验规则（validQuestion）和查询条件（getQueryWrapper）

### 3) 【管理员】管理题库 - 增删改查

创建、删除、编辑接口需要补充仅管理员可用的注解

```
1  @AuthCheck(mustRole = UserConstant.ADMIN_ROLE)
```

### 4) 【管理员】管理题目 - 增删改查

创建、删除、编辑接口需要补充仅管理员可用的注解

### 5) 查看题目详情（进入刷题页面）：根据 id 获取题目详情接口

该接口正常，不用修改

## 2、根据题库查询题目列表接口

### 分析

需求：根据题库 id 查询题目列表

分析：由于同一个题库内的题目不会很多，为了简化实现，可以直接获取全部题目。注意，这个功能要抽象成 service 方法，便于后续的获取题库详情接口复用。

由于题目和题库是通过关联表维护关系的，所以在查询时，要先通过查询关联表得到题目 id，再根据 id 从题目表查询到题目的完整信息。

有 2 种实现方式：

1) 通过 SQL 的 join 联表查询

示例 SQL 如下：

```
1  SELECT
2      q.id,
3      q.title,
4      q.content,
5      q.tags,
6      q.answer,
7      q.userId,
8      q.createTime,
9      q.updateTime
10  FROM
11      question_bank_question qbq
12  JOIN
13      question q
14  ON
15      qbq.questionId = q.id
16  WHERE
17      qbq.questionBankId = ?; -- 在这里替换 ? 为具体的题库 id
```

2) 业务层分步查询

要先通过查询关联表得到题目 id，再把 id 放到集合中，根据 id 使用 in 查询从题目表查询到题目的完整信息。

此处选择第二种方式，因为关联逻辑并不复杂、数据量也不大，业务层实现更灵活，也便于组合其他条件去过滤题目列表。

注意，如果要对题目题库关联表和题目表同时进行过滤和分页查询，那么考虑使用 SQL 的 join 实现，因为业务层处理多表分页比较麻烦。

## 开发

1) 请求参数为 questionBankId，直接补充到题目查询请求类 QuestionQueryRequest：

```
1  public class QuestionQueryRequest extends PageRequest implements Serializable
2
3     // 省略其他字段。。
4
5     /**
6      * 题库 id
7      */
8     private Long questionBankId;
9 }
```

2) 该接口本质上还是查询题目列表，可以把题库 id 当做一个过滤题目的查询条件，所以应该在 QuestionService 中编写一个通用的分页获取题目列表的方法。

代码如下：

```

1  public Page<Question> listQuestionByPage(QuestionQueryRequest questionQueryRe
2      long current = questionQueryRequest.getCurrent();
3      long size = questionQueryRequest.getPageSize();
4      // 题目表的查询条件
5      QueryWrapper<Question> queryWrapper = this.getQueryWrapper(questionQueryR
6      // 根据题库查询题目列表接口
7      Long questionBankId = questionQueryRequest.getQuestionBankId();
8      if (questionBankId != null) {
9          // 查询题库内的题目 id
10         LambdaQueryWrapper<QuestionBankQuestion> lambdaQueryWrapper = Wrapper
11             .select(QuestionBankQuestion::getQuestionId)
12             .eq(QuestionBankQuestion::getQuestionBankId, questionBankId);
13         List<QuestionBankQuestion> questionList = questionBankQuestionService
14         if (CollUtil.isNotEmpty(questionList)) {
15             // 取出题目 id 集合
16             Set<Long> questionIdSet = questionList.stream()
17                 .map(QuestionBankQuestion::getQuestionId)
18                 .collect(Collectors.toSet());
19             // 复用原有题目表的查询条件
20             queryWrapper.in("id", questionIdSet);
21         }
22     }
23     // 查询数据库
24     Page<Question> questionPage = this.page(new Page<>(current, size), queryW
25     return questionPage;
26 }

```

上述代码中，有几个学习重点：

1. 查询关联表的时候，不要 select 所有字段，只取 questionId 就够了，可以提升性能。
2. 注意判断通过关联表查询到的题目数量，如果为空，不用再作为查询条件。
3. 从关联表查到的虽然只有一个字段，但返回的还是对象，所以需要使用 Lambda 表达式转换成题目 id 集合。
4. 把题库 id 通过关联表转换为了多个题目 id，巧妙地复用了原有的题目过滤条件 (QueryWrapper)，可以同时支持按照题库 id 和标题等其他条件来搜索题目。

### 3、获取题库详情接口

#### 分析

需求：根据题库 id 获取题库详情，同时可能需要查询到题库内的题目列表。

分析：可以用一个字段来控制是否要关联查询题目列表，前端可以根据不同的场景，给该字段传入不同的值。这样一来，对于有些不需要关联查询题目列表的页面，就能减少性能损耗。

实现过程比较简单，先从题库表查询出题库信息，然后复用上一步中已开发好的“根据题库 id 获得题库列表”的 Service 获取到题目列表，再封装返回值即可。

## 开发

1) 因为获取题库详情接口其实也是对题库的查询，所以可以给 QuestionBankQueryRequest 增加字段 needQueryQuestionList，用于控制是否要关联查询题目列表。默认为 false，表示不查询。

代码如下：

```
1  public class QuestionBankQueryRequest extends PageRequest implements Serializable {
2      // 省略其他字段...
3
4      /**
5      * 是否要关联查询题目列表
6      */
7      private boolean needQueryQuestionList;
8
9      private static final long serialVersionUID = 1L;
10 }
```

2) 封装题库详情响应类，补充题目列表分页。代码如下：

```
1  public class QuestionBankVO implements Serializable {
2      // 省略其他字段...
3
4      /**
5      * 题库里的题目列表（分页）
6      */
7      Page<Question> questionPage;
8 }
```

此处也可以改为补充列表而不是分页，根据自己后续的实际需求调整即可。

3) 在 QuestionBankController 中修改“根据 id 获得题库”封装类的接口，代码如下：

```

1  @GetMapping("/get/vo")
2  public BaseResponse<QuestionBankVO> getQuestionBankVOById(QuestionBankQueryRe
3      ThrowUtils.throwIf(questionBankQueryRequest == null, ErrorCode.PARAMS_ERR
4      Long id = questionBankQueryRequest.getId();
5      ThrowUtils.throwIf(id <= 0, ErrorCode.PARAMS_ERROR);
6      // 查询数据库
7      QuestionBank questionBank = questionBankService.getById(id);
8      ThrowUtils.throwIf(questionBank == null, ErrorCode.NOT_FOUND_ERROR);
9      // 查询题库封装类
10     QuestionBankVO questionBankVO = questionBankService.getQuestionBankVO(que
11     // 是否要关联查询题库下的题目列表
12     boolean needQueryQuestionList = questionBankQueryRequest.isNeedQueryQuest
13     if (needQueryQuestionList) {
14         QuestionQueryRequest questionQueryRequest = new QuestionQueryRequest(
15             questionQueryRequest.setQuestionBankId(id);
16             Page<Question> questionPage = questionService.listQuestionByPage(q
17             questionBankVO.setQuestionPage(questionPage);
18     }
19     // 获取封装类
20     return ResultUtils.success(questionBankVO);
21 }

```

注意，这段逻辑如果觉得过于复杂，也可以考虑封装到 `getQuestionBankVO` 方法中。

## 4、修改题目所属题库接口

### 分析

需求：根据题目 `id` 和题库 `id`，修改题目所属题库。

分析：修改的本质是添加和删除，是两个动作。

- 如果题目未加入到该题库，则管理员可以添加题目到题库，在题库题目关联表中加一条记录。
- 如果题目已加入到该题库，则管理员可以从题库移除题目，需要将关联表已有的记录删除。

所以需要分别开发 2 个接口：创建题库题目关联、移除题库题目关联。

由于题库题目关联表已经添加了 `题库 id`、`题目 id` 的唯一性约束，不用担心重复添加脏数据，做好异常处理即可。

### 创建题库题目关联

1) 之前已经通过生成器得到了创建关联的 Controller 接口代码，只需要补充数据不存在的校验。  
修改 `QuestionBankQuestionServiceImpl` 的 `validQuestionBankQuestion` 方法，进行校验。代码如下：

```
1  @Override
2  public void validQuestionBankQuestion(QuestionBankQuestion questionBankQuesti
3      ThrowUtils.throwIf(questionBankQuestion == null, ErrorCode.PARAMS_ERROR);
4      // 题目和题库必须存在
5      Long questionId = questionBankQuestion.getQuestionId();
6      if (questionId != null) {
7          Question question = questionService.getById(questionId);
8          ThrowUtils.throwIf(question == null, ErrorCode.NOT_FOUND_ERROR, "题目")
9      }
10     Long questionBankId = questionBankQuestion.getQuestionBankId();
11     if (questionBankId != null) {
12         QuestionBank questionBank = questionBankService.getById(questionBankI
13         ThrowUtils.throwIf(questionBank == null, ErrorCode.NOT_FOUND_ERROR, ""
14     }
15 }
```

因为 QuestionService 和 QuestionBankQuestionService 互相引用，会导致循环依赖问题，让项目无法启动。可以通过给 QuestionBankQuestionServiceImpl 中引用 QuestionService 的位置加上 `@Lazy` 注解解决。

```
1  @Resource
2  @Lazy
3  private QuestionService questionService;
```

2) 注意，需要给创建关联接口补充“仅管理员”的权限：

```
1  @PostMapping("/add")
2  @AuthCheck(mustRole = UserConstant.ADMIN_ROLE)
3  public BaseResponse<Long> addQuestionBankQuestion(
4      @RequestBody QuestionBankQuestionAddRequest questionBankQuestionAddRequest
5      HttpServletRequest request) {}
```

## 移除题库题目关联

1) 编写请求类：

```
1  *//**
2   * 删題題庫關係請求
3   */
4  @Data
5  public class QuestionBankQuestionRemoveRequest implements Serializable {
6      /**
7       * 題庫 id
8       */
9      private Long questionBankId;
10
11     /**
12      * 題目 id
13      */
14     private Long questionId;
15
16     private static final long serialVersionUID = 1L;
17 }
```

## 2) 编写接口:

```
1  @PostMapping("/remove")
2  @AuthCheck(mustRole = UserConstant.ADMIN_ROLE)
3  public BaseResponse<Boolean> removeQuestionBankQuestion(
4      @RequestBody QuestionBankQuestionRemoveRequest questionBankQuestionRe
5  ) {
6      // 参数校验
7      ThrowUtils.throwIf(questionBankQuestionRemoveRequest == null, ErrorCode.P
8      Long questionBankId = questionBankQuestionRemoveRequest.getQuestionBankId
9      Long questionId = questionBankQuestionRemoveRequest.getQuestionId();
10     ThrowUtils.throwIf(questionBankId == null || questionId == null, ErrorCode.
11     // 构造查询
12     LambdaQueryWrapper<QuestionBankQuestion> lambdaQueryWrapper = Wrappers.la
13         .eq(QuestionBankQuestion::getQuestionId, questionId)
14         .eq(QuestionBankQuestion::getQuestionBankId, questionBankId);
15     boolean result = questionBankQuestionService.remove(lambdaQueryWrapper);
16     return ResultUtils.success(result);
17 }
```

逻辑并不复杂，所以就直接在 Controller 编写了。有时间的话，可以再移动到 Service 中。

## 扩展

- 1) 如果用同样的参数多次调用“创建题库题目关联接口”，会因为数据库的唯一性约束导致 `org.springframework.dao.DuplicateKeyException` 异常，然后会被全局异常处理器处理返回“系统错误”。可以针对这种情况进行异常捕获，并优化报错文案，比如“不能重复加入”。