

day06-点赞系统

我们已经实现了学习辅助中的互动问答功能，不过存在一个问题，仅仅靠老师来给学生回答问题存在一些弊端：

- 老师可能忙不过来
- 难以调动所有学员互动热情
- 互动的氛围感较差

因此，产品提出了新的需求：

当热心用户或者老师给学生回答了问题以后，所有学员可以给自己心仪的回答点赞，点赞越高，排名也越靠前。

这样一来，用户回答和评论的欲望就会增加，网站的活跃度也会越来越高。

点赞功能是社交、电商等几乎所有的互联网项目中都广泛使用。虽然看起来简单，不过蕴含的技术方案和手段还是比较多的。

今天，我们就一起来揭开点赞功能的神秘面纱。

1.需求分析

点赞功能与其它功能不同，没有复杂的原型和需求，仅仅是一个点赞、取消点赞的操作。所以，今天我们就不需要从原型图来分析，而是仅仅从这个功能的实现方案来思考。

1.1.业务需求

首先我们来分析整理一下点赞业务的需求，一个通用点赞系统需要满足下列特性：

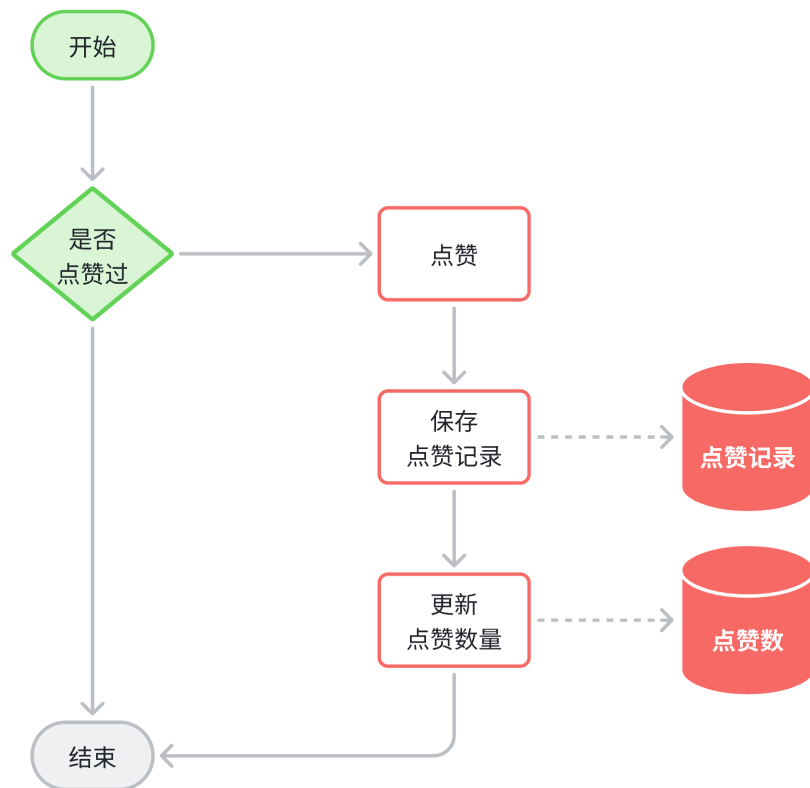


- 通用：点赞业务在设计的时候不要与业务系统耦合，必须同时支持不同业务的点赞功能
- 独立：点赞功能是独立系统，并且不依赖其它服务。这样才具备可迁移性。
- 并发：一些热点业务点赞会很多，所以点赞功能必须支持高并发
- 安全：要做好并发安全控制，避免重复点赞

1.2.实现思路

要保证安全，避免重复点赞，我们就必须保存每一次点赞记录。只有这样在下次用户点赞时我们才能查询数据，判断是否是重复点赞。同时，因为业务方经常需要根据点赞数量排序，因此每个业务的点赞数量也需要记录下来。

综上，点赞的基本思路如下：

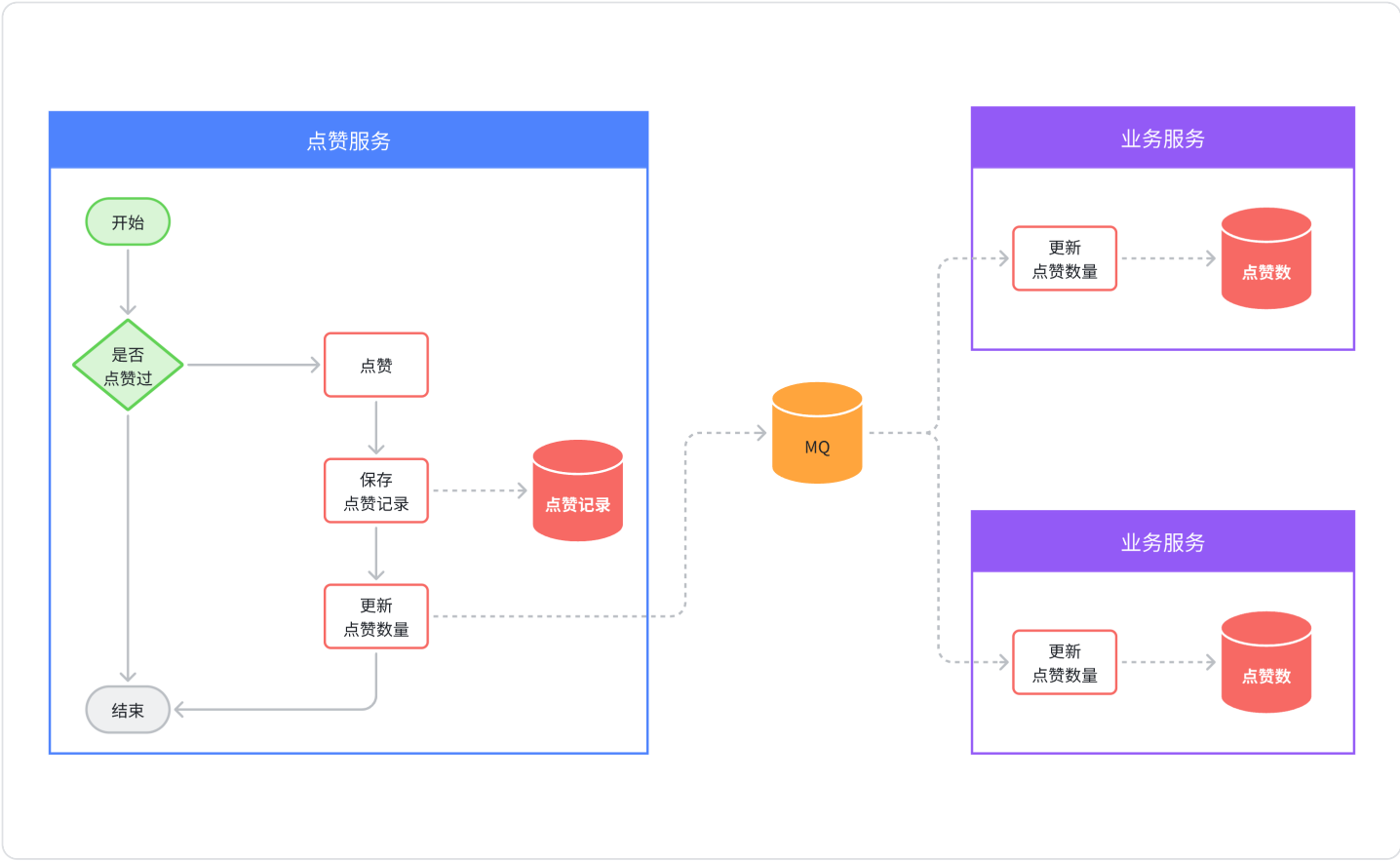


但问题来了，我们说过点赞服务必须独立，因此必须抽取为一个**独立服务**。多个其它微服务业务的点赞数据都有点赞系统来维护。但是问题来了：

如果业务方需要根据点赞数排序，就必须在数据库中维护点赞数字段。但是点赞系统无法修改其它业务服务的数据库，否则就出现了业务耦合。该怎么办呢？

点赞系统可以在点赞数变更时，通过MQ通知业务方，这样业务方就可以更新自己的点赞数量了。并且还避免了点赞系统与业务方的耦合。

于是，实现思路变成了这样：



2.数据结构

点赞的数据结构分两部分，一是**点赞记录**，二是与业务关联的**点赞数**。

点赞数自然是与具体业务表关联在一起记录，比如互动问答的点赞，自然是在问答表中记录点赞数。学员笔记点赞，自然是在笔记表中记录点赞数。

在之前实现互动问答的时候，我们已经给回答表设计了点赞数字段了：

#	名称	数据类型	注释	长度/集合	无符...	允许 NULL	默认
1	id	BIGINT	互动问题的回答id	19	<input type="checkbox"/>	<input type="checkbox"/>	无默认值
2	question_id	BIGINT	互动问题问题id	19	<input type="checkbox"/>	<input type="checkbox"/>	无默认值
3	answer_id	BIGINT	回复的上级回答id	19	<input type="checkbox"/>	<input checked="" type="checkbox"/>	'0'
4	user_id	BIGINT	回答者id	19	<input type="checkbox"/>	<input type="checkbox"/>	无默认值
5	content	VARCHAR	回答内容	255	<input type="checkbox"/>	<input type="checkbox"/>	无默认值
6	target_user_id	BIGINT	回复的目标用户id	19	<input type="checkbox"/>	<input checked="" type="checkbox"/>	'0'
7	target_reply_id	BIGINT	回复的目标回复id	19	<input type="checkbox"/>	<input checked="" type="checkbox"/>	'0'
8	reply_times	INT	评论数量	10	<input type="checkbox"/>	<input type="checkbox"/>	'0'
9	liked_times	INT	点赞数量	10	<input type="checkbox"/>	<input type="checkbox"/>	'0'
10	hidden	BIT	是否被隐藏，默认false	1	<input type="checkbox"/>	<input type="checkbox"/>	'b\0\'
11	anonymity	BIT	是否匿名，默认false	1	<input type="checkbox"/>	<input type="checkbox"/>	'b\0\'
12	create_time	DATETIME	创建时间		<input type="checkbox"/>	<input type="checkbox"/>	CURRENT_TIMESTAMP
13	update_time	DATETIME	更新时间		<input type="checkbox"/>	<input type="checkbox"/>	CURRENT_TIMESTAMP O...

其它业务也是类似的。

因此，本节我们只需要实现点赞记录的表结构设计即可。

2.1.ER图

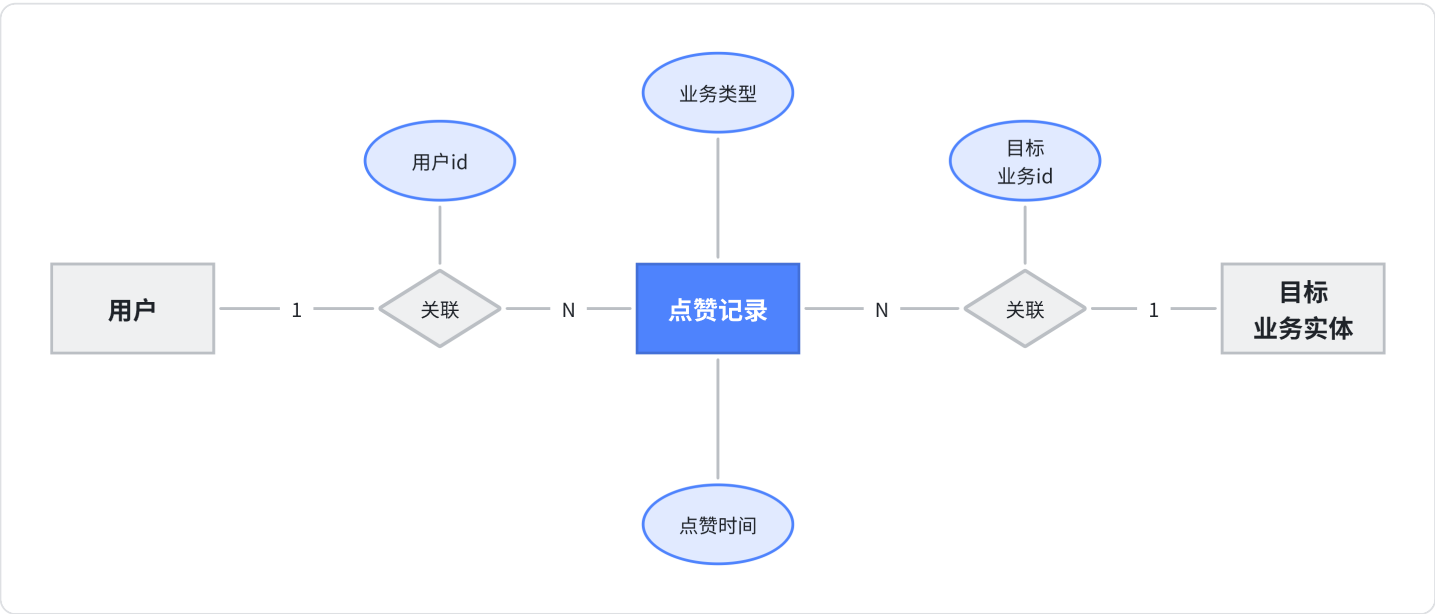
点赞记录本质就是记录**谁给什么内容点了赞**，所以核心属性包括：

- 点赞目标id
- 点赞人id

不过点赞的内容多种多样，为了加以区分，我们还需要把点赞内的类型记录下来：

- 点赞对象类型（为了通用性）

当然还有点赞时间，综上对应的数据库ER图如下：



2.2.表结构

由于点赞系统是独立于其它业务的，这里我们需要创建一个新的数据库：`tj_remark`

```
1 CREATE DATABASE tj_remark CHARACTER SET 'utf8mb4';
```

然后在ER图基础上，加上一些通用属性，点赞记录表结构如下：

```
1 CREATE TABLE IF NOT EXISTS `liked_record` (
```

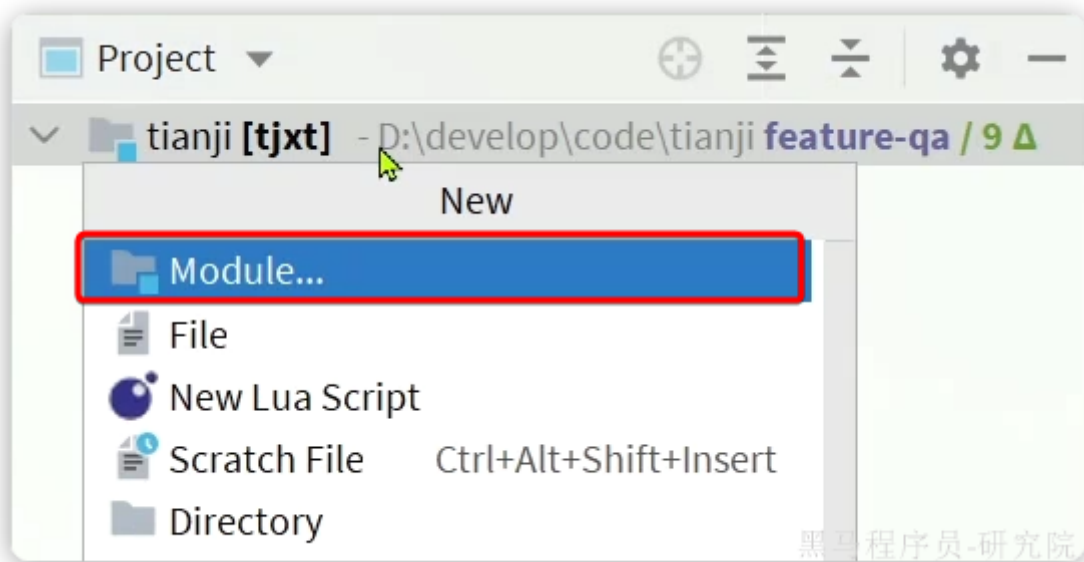
```
2  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '主键id',
3  `user_id` bigint NOT NULL COMMENT '用户id',
4  `biz_id` bigint NOT NULL COMMENT '点赞的业务id',
5  `biz_type` VARCHAR(16) NOT NULL COMMENT '点赞的业务类型',
6  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
7  `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
  CURRENT_TIMESTAMP COMMENT '更新时间',
8  PRIMARY KEY (`id`),
9  UNIQUE KEY `idx_biz_user` (`biz_id`,`user_id`)
10 ) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8mb4
  COLLATE=utf8mb4_0900_ai_ci COMMENT='点赞记录表';
11
```

2.3.代码生成

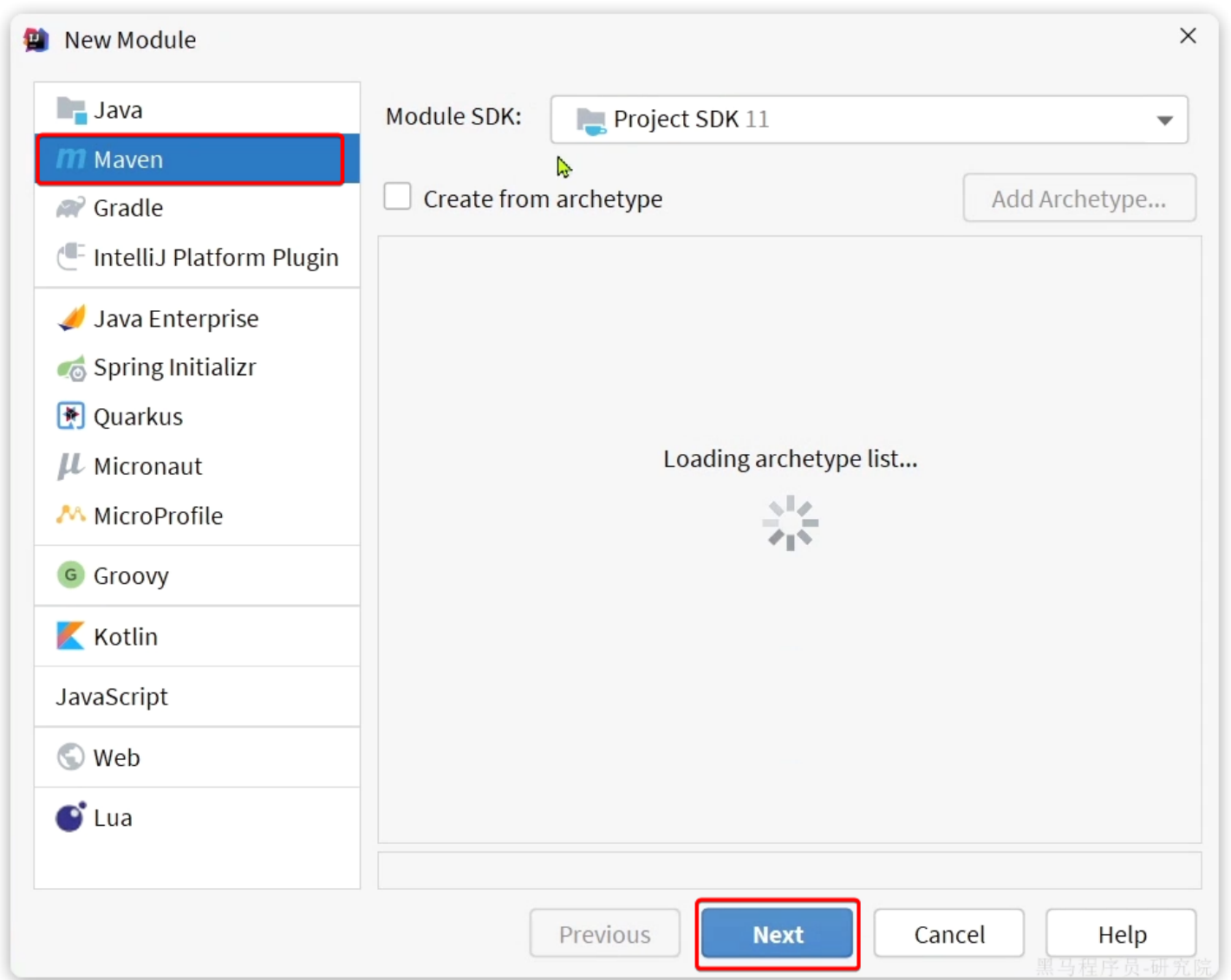
由于点赞系统是一个独立微服务，我们需要创建一个新的微服务模块。

2.3.1.创建微服务

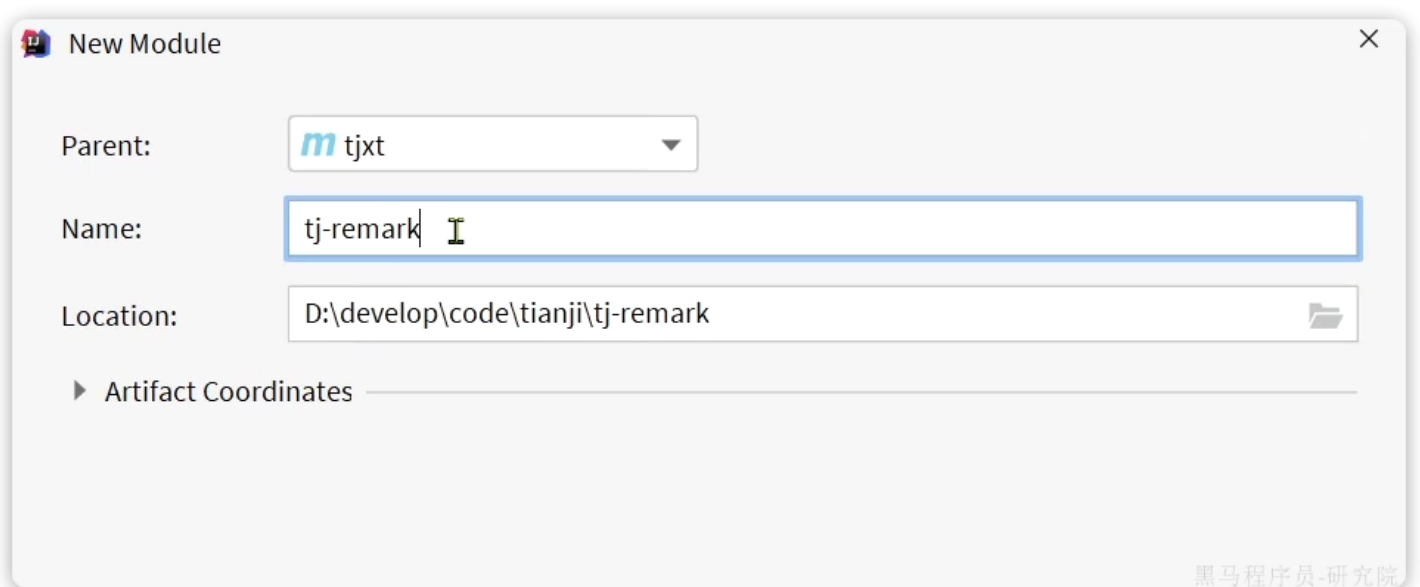
创建模块：



选择maven模块：



填写项目名称：



在pom.xml中填入依赖：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <parent>
6         <artifactId>tjxt</artifactId>
7         <groupId>com.tianji</groupId>
8         <version>1.0.0</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>tj-remark</artifactId>
13
14    <properties>
15        <maven.compiler.source>11</maven.compiler.source>
16        <maven.compiler.target>11</maven.compiler.target>
17    </properties>
18    <dependencies>
19        <!--auth-sdk-->
20        <dependency>
21            <groupId>com.tianji</groupId>
22            <artifactId>tj-auth-resource-sdk</artifactId>
23            <version>1.0.0</version>
24        </dependency>
25        <!--api-->
26        <dependency>
27            <groupId>com.tianji</groupId>
28            <artifactId>tj-api</artifactId>
29            <version>1.0.0</version>
30        </dependency>
31        <!--web-->
32        <dependency>
33            <groupId>org.springframework.boot</groupId>
34            <artifactId>spring-boot-starter-web</artifactId>
35        </dependency>
36        <!--mybatis-->
37        <dependency>
38            <groupId>com.baomidou</groupId>
39            <artifactId>mybatis-plus-boot-starter</artifactId>
40        </dependency>
41        <dependency>
42            <groupId>mysql</groupId>
43            <artifactId>mysql-connector-java</artifactId>
44        </dependency>
45        <!--Redis-->
46        <dependency>
```



```
47         <groupId>org.springframework.boot</groupId>
48         <artifactId>spring-boot-starter-data-redis</artifactId>
49     </dependency>
50     <!--discovery-->
51     <dependency>
52         <groupId>com.alibaba.cloud</groupId>
53         <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
54     </dependency>
55     <!--config-->
56     <dependency>
57         <groupId>com.alibaba.cloud</groupId>
58         <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
59     </dependency>
60     <!--mq-->
61     <dependency>
62         <groupId>org.springframework.boot</groupId>
63         <artifactId>spring-boot-starter-amqp</artifactId>
64     </dependency>
65     <!--loadbalancer-->
66     <dependency>
67         <groupId>org.springframework.cloud</groupId>
68         <artifactId>spring-cloud-starter-loadbalancer</artifactId>
69     </dependency>
70 </dependencies>
71 <build>
72     <finalName>${project.artifactId}</finalName>
73     <plugins>
74         <plugin>
75             <groupId>org.springframework.boot</groupId>
76             <artifactId>spring-boot-maven-plugin</artifactId>
77             <executions>
78                 <execution>
79                     <goals>
80                         <goal>build-info</goal>
81                     </goals>
82                 </execution>
83             </executions>
84             <configuration>
85                 <mainClass>com.tianji.remark.RemarkApplication</mainClass>
86             </configuration>
87         </plugin>
88     </plugins>
89 </build>
90 </project>
```

然后是配置文件：bootstrap.yml

```
1 server:
2   port: 8091  #端口
3   tomcat:
4     uri-encoding: UTF-8  #服务编码
5 spring:
6   profiles:
7     active: dev
8   application:
9     name: remark-service
10  cloud:
11    nacos:
12      config:
13        file-extension: yaml
14        shared-configs: # 共享配置
15          - data-id: shared-spring.yaml # 共享spring配置
16            refresh: false
17          - data-id: shared-redis.yaml # 共享redis配置
18            refresh: false
19          - data-id: shared-mybatis.yaml # 共享mybatis配置
20            refresh: false
21          - data-id: shared-logs.yaml # 共享日志配置
22            refresh: false
23          - data-id: shared-feign.yaml # 共享feign配置
24            refresh: false
25          - data-id: shared-mq.yaml # 共享mq配置
26            refresh: false
27  tj:
28    swagger:
29      enable: true
30      enableResponseWrap: true
31      package-path: com.tianji.remark.controller
32      title: 天机学堂 - 评价中心接口文档
33      description: 该服务包含评价、点赞等功能
34      contact-name: 传智教育·研究院
35      contact-url: http://www.itcast.cn/
36      contact-email: zhanghuyi@itcast.cn
37      version: v1.0
38  jdbc:
39    database: tj_remark
40  auth:
41    resource:
42      enable: true # 登录拦截功能
```

接着是 `bootstrap-dev.yml` :

```
1 spring:
2   cloud:
3     nacos:
4       server-addr: 192.168.150.101:8848 # nacos注册中心
5       discovery:
6         namespace: f923fb34-cb0a-4c06-8fca-ad61ea61a3f0
7         group: DEFAULT_GROUP
8         ip: 192.168.150.101
9   logging:
10    level:
11      com.tianji: debug
```

然后是 `bootstrap-local.yml` :

```
1 spring:
2   cloud:
3     nacos:
4       server-addr: 192.168.150.101:8848 # nacos注册中心
5       discovery:
6         namespace: f923fb34-cb0a-4c06-8fca-ad61ea61a3f0
7         group: DEFAULT_GROUP
8         ip: 192.168.150.1
9   logging:
10    level:
11      com.tianji: debug
```

最后，新建一个启动类：

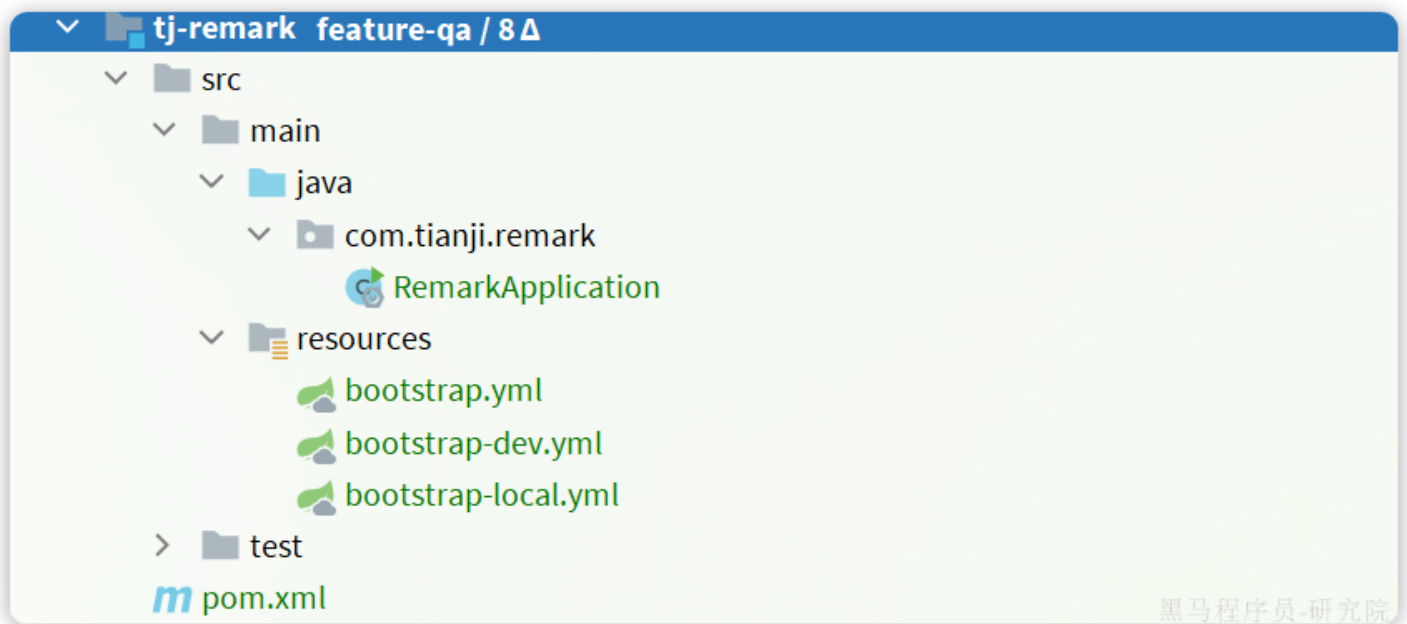
```
1 package com.tianji.remark;
2
3 import lombok.extern.slf4j.Slf4j;
4 import org.mybatis.spring.annotation.MapperScan;
5 import org.springframework.boot.SpringApplication;
6 import org.springframework.boot.autoconfigure.SpringBootApplication;
7 import org.springframework.boot.builder.SpringApplicationBuilder;
8 import org.springframework.core.env.Environment;
9 import org.springframework.scheduling.annotation.EnableScheduling;
10
```

```

11 import java.net.InetAddress;
12 import java.net.UnknownHostException;
13
14 @Slf4j
15 @EnableScheduling
16 @SpringBootApplication
17 @MapperScan("com.tianji.remark.mapper")
18 public class RemarkApplication {
19     public static void main(String[] args) throws UnknownHostException {
20         SpringApplication app = new
SpringApplicationBuilder(RemarkApplication.class).build(args);
21         Environment env = app.run(args).getEnvironment();
22         String protocol = "http";
23         if (env.getProperty("server.ssl.key-store") != null) {
24             protocol = "https";
25         }
26         log.info("--/\n-----
-----\n\t" +
27             "Application '{}' is running! Access URLs:\n\t" +
28             "Local: \t\t{}://localhost:{}\n\t" +
29             "External: \t{}://{}:{}\n\t" +
30             "Profile(s): \t{}" +
31             "\n-----",
32             env.getProperty("spring.application.name"),
33             protocol,
34             env.getProperty("server.port"),
35             protocol,
36             InetAddress.getLocalHost().getHostAddress(),
37             env.getProperty("server.port"),
38             env.getActiveProfiles());
39     }
40 }

```

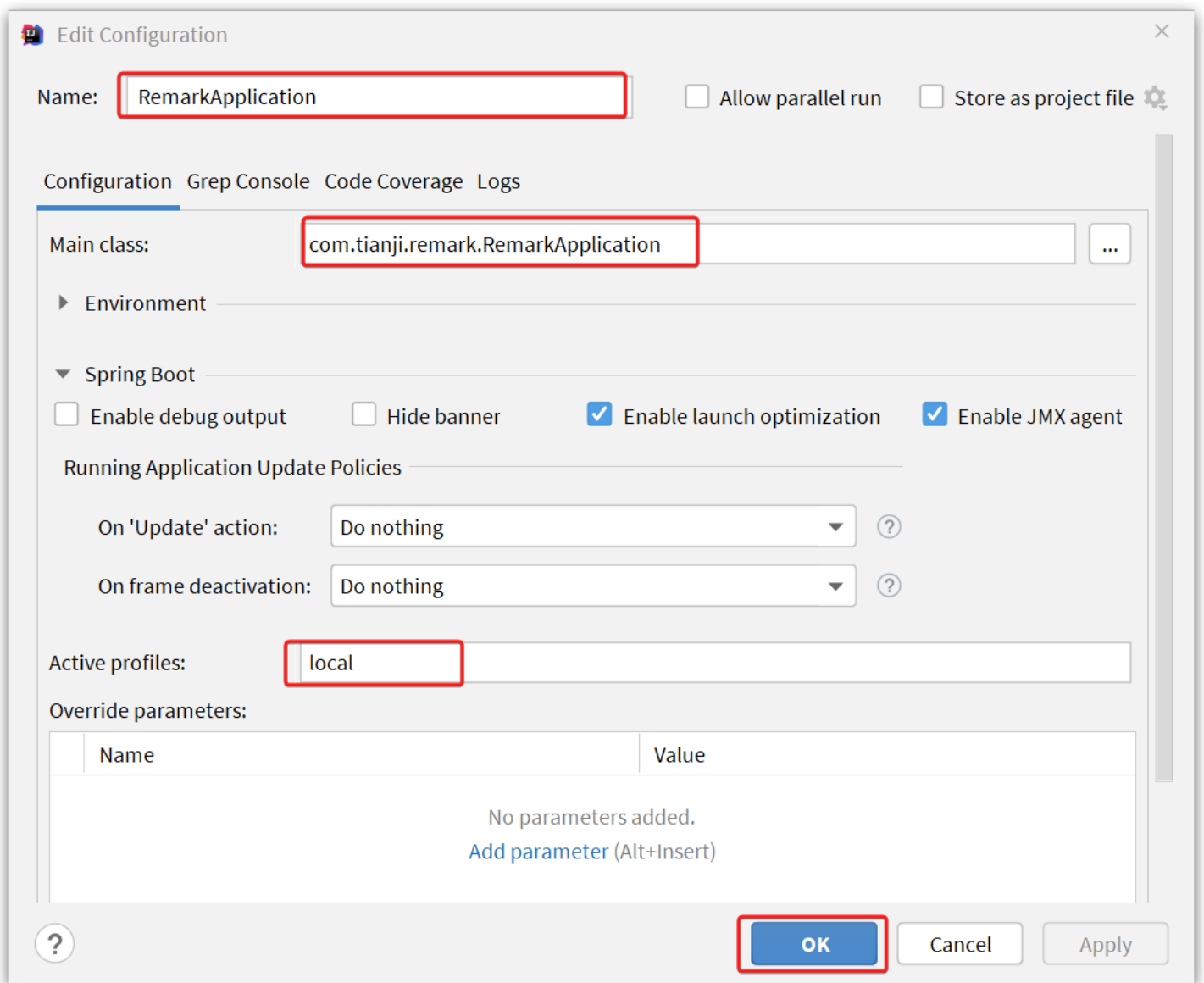
项目结构：



微服务搭建后，一定不要忘了在网关配置服务路由，找到 `tj-gateway` 服务的 `bootstrap.yml` 文件，添加以下内容：

```
1
2 # ... 其它略
3 spring:
4     # ... 其它略
5     cloud:
6         # ... 其它略
7         gateway:
8             routes:
9                 - id: rs
10                   uri: lb://remark-service
11                   predicates:
12                       - Path=/rs/**
13             # ... 其它略
14         default-filters:
15             - StripPrefix=1
16 # ... 其它略
```

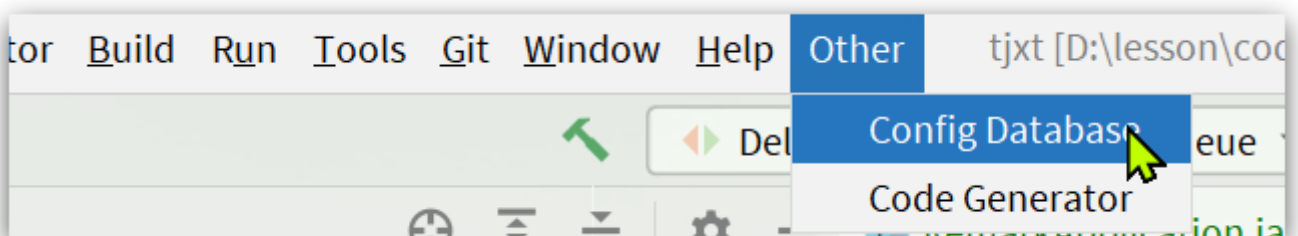
为了方便本地启动测试，最后给remark-service添加一个SpringBoot启动项：



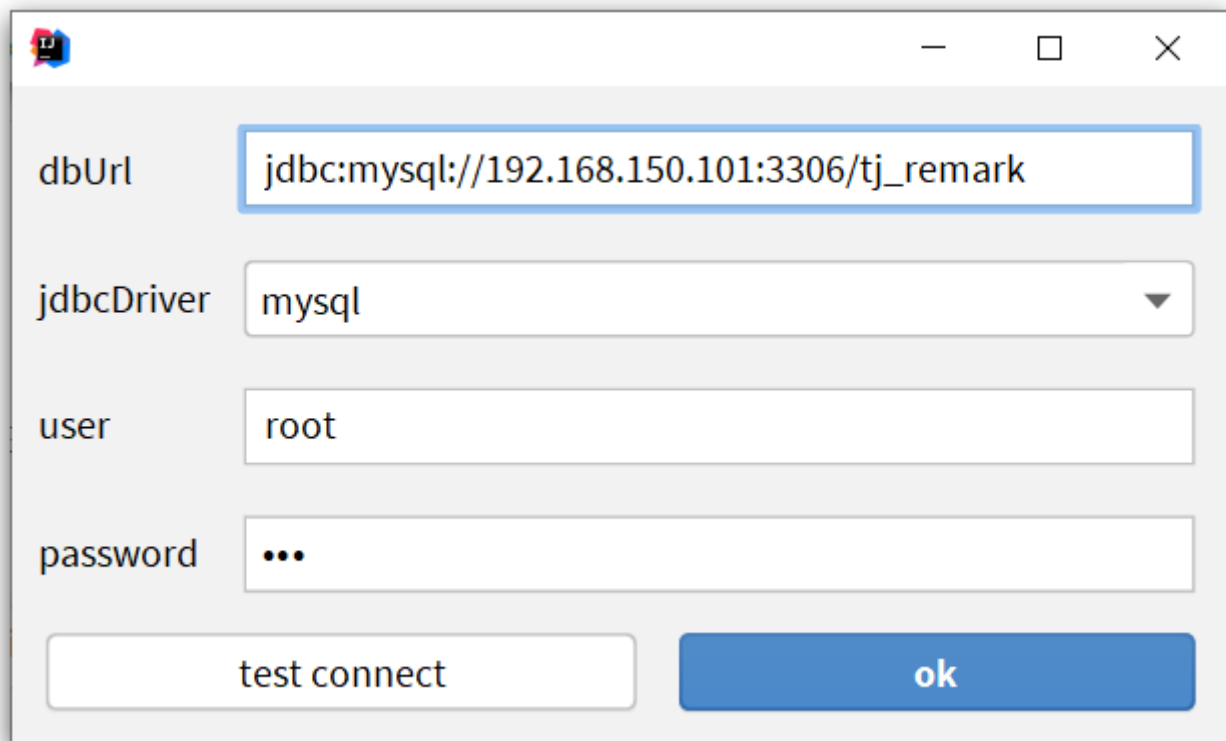
2.3.2.代码生成

利用MybatisPlus的插件生成实体、mapper、service、controller等代码。

注意要先配置数据库地址：



填写数据库信息：



A dialog box for configuring a database connection. It contains four input fields: 'dbUrl' with the value 'jdbc:mysql://192.168.150.101:3306/tj_remark', 'jdbcDriver' with a dropdown menu showing 'mysql', 'user' with the value 'root', and 'password' with three dots indicating a masked password. At the bottom, there are two buttons: 'test connect' and 'ok'.

dbUrl: jdbc:mysql://192.168.150.101:3306/tj_remark

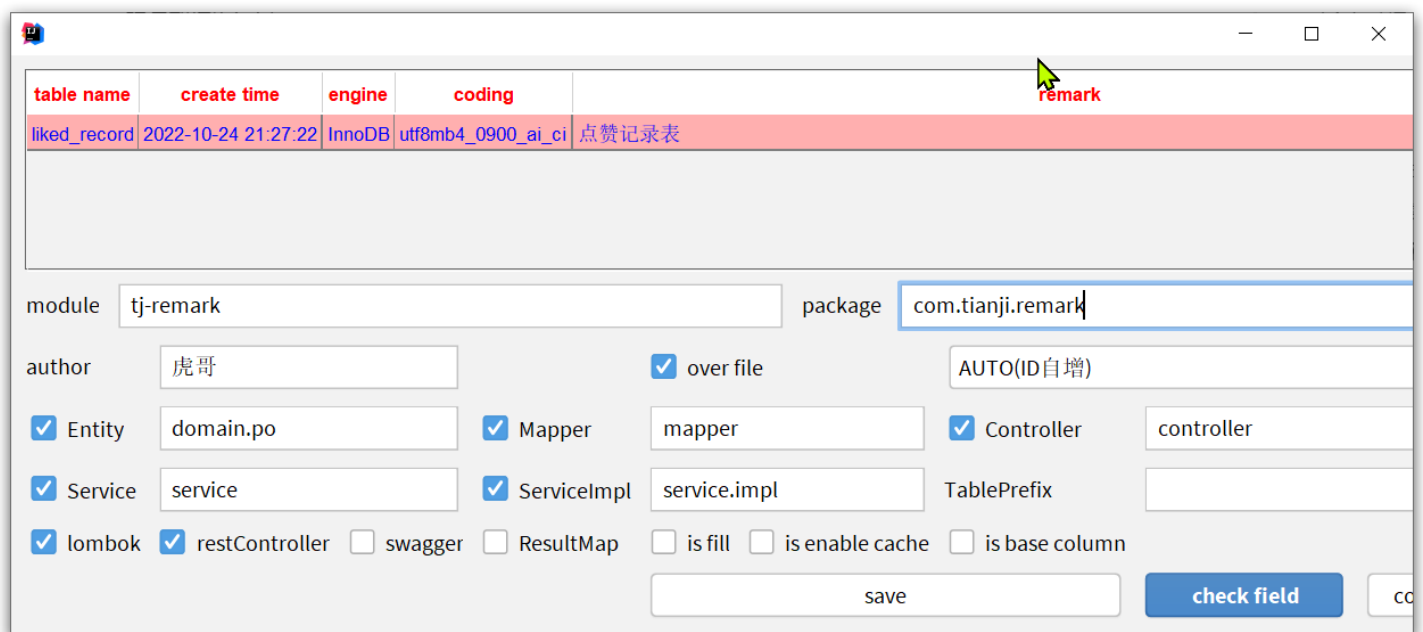
jdbcDriver: mysql

user: root

password: ...

test connect ok

然后生成代码：



A dialog box for configuring code generation. It features a table at the top with columns: 'table name', 'create time', 'engine', 'coding', and 'remark'. The first row of data is highlighted in red and contains the values: 'liked_record', '2022-10-24 21:27:22', 'InnoDB', 'utf8mb4_0900_ai_ci', and '点赞记录表'. Below the table, there are several input fields and checkboxes. The 'module' field is 'tj-remark' and the 'package' field is 'com.tianji.remark'. The 'author' field is '虎哥'. There are checkboxes for 'Entity', 'Service', 'lombok', 'restController', 'Mapper', 'ServiceImpl', 'Controller', and 'over file'. The 'Entity' checkbox is checked. The 'Service' checkbox is checked. The 'lombok' checkbox is checked. The 'restController' checkbox is checked. The 'Mapper' checkbox is checked. The 'ServiceImpl' checkbox is checked. The 'Controller' checkbox is checked. The 'over file' checkbox is checked. The 'TablePrefix' field is empty. The 'is fill' checkbox is unchecked. The 'is enable cache' checkbox is unchecked. The 'is base column' checkbox is unchecked. At the bottom, there are two buttons: 'save' and 'check field'.

table name	create time	engine	coding	remark
liked_record	2022-10-24 21:27:22	InnoDB	utf8mb4_0900_ai_ci	点赞记录表

module: tj-remark package: com.tianji.remark

author: 虎哥

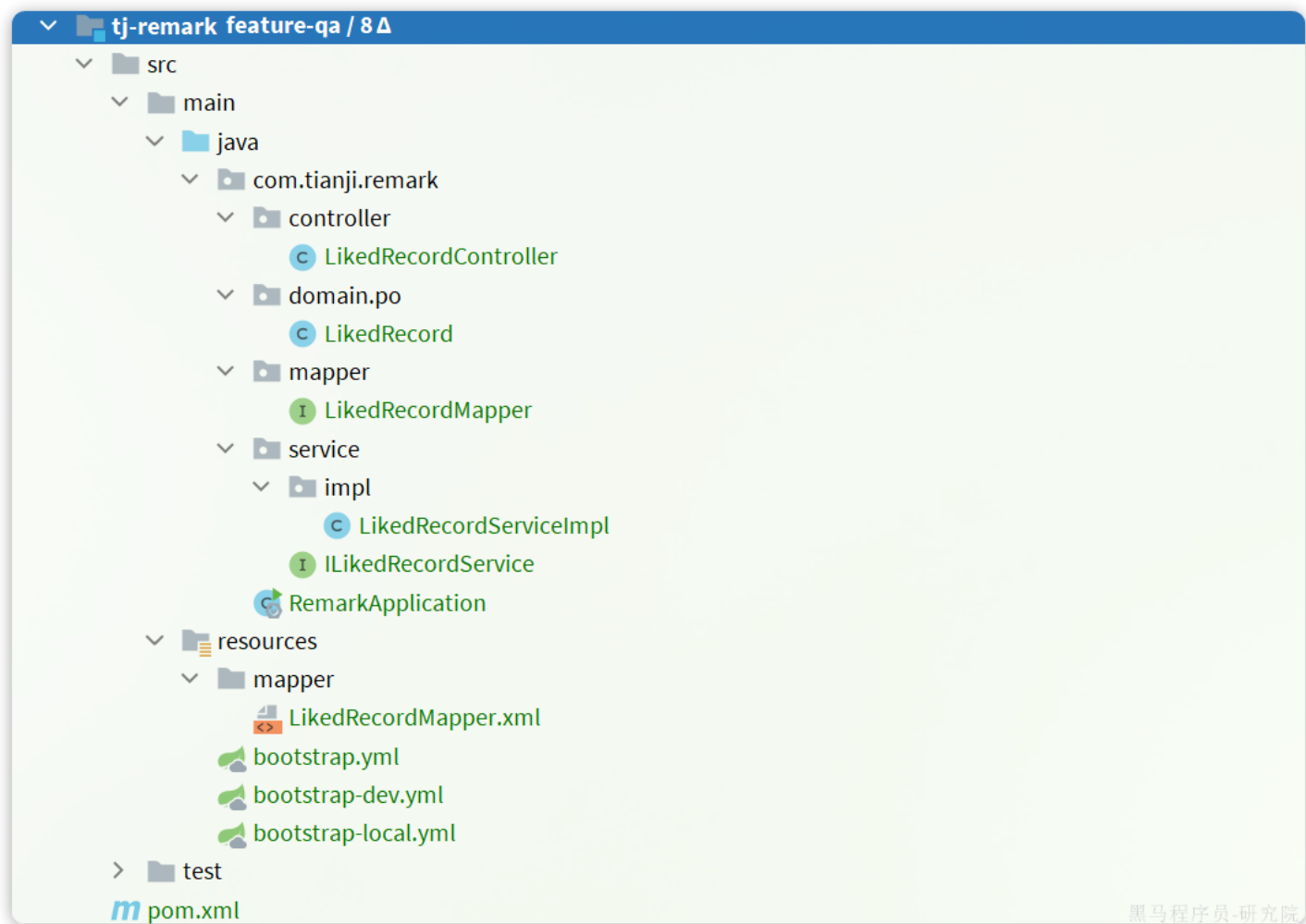
☒ Entity domain.po ☒ Mapper mapper ☒ Controller controller

☒ Service service ☒ ServiceImpl service.impl TablePrefix

☒ lombok ☒ restController ☐ swagger ☐ ResultMap ☐ is fill ☐ is enable cache ☐ is base column

save check field

代码结构如下：



3.实现点赞功能

从表面来看，点赞功能要实现的接口就是一个点赞接口。不过仔细观察所有的点赞页面，你会发现点赞按钮有灰色和点亮两种状态。

也就是说我们还需要实现查询用户点赞状态的接口，这样前端才能根据点赞状态渲染不同效果。因此我们要实现的接口包括：

- 点赞/取消点赞
- 根据多个业务id批量查询用户是否点赞多个业务

3.1.点赞或取消点赞

3.1.1.接口信息

当用户点击点赞按钮的时候，第一次点击是点赞，按钮会高亮；第二次点击是取消，点赞按钮变灰：

全部回答 (23)



匿名用户001

建议你把主要精力放在Spring全家桶、Mybatis 等框架，MySQL数据库、Redis缓存，Docker、Git 上，这是 Java 后端开发中使用频率最高的东西。也就是说，你学好了这些东西之后（通过一个实战项目串联起来），应该就可以找到一份中小厂的 Java 开发工作了。

2022.07.12 18:08

评论 2

点赞 3



匿名用户002 评论了 匿名用户001

自学Java难不难？该如何高效入门呢？网上资源又多又杂，是不是感觉无从下手？这里小编给大家整理了一份关于Java的学习资料，帮助大家构建Java语言的完整体系，不妨花上几分钟耐心阅读。

2022.07.12 18:08

评论 1

点赞 3



匿名用户001 回复了 匿名用户002

有很多小伙伴对JavaWeb、Spring、SpringMVC、SpringBoot等技术概念上有些混淆，那我们针对这几个概念，来详细解释一下这些技术的区别和联系。

2022.07.12 18:08

评论

已赞 3

黑马程序员-研究院

从后台实现来看，点赞就是新增一条点赞记录，取消就是删除这条记录。为了方便前端交互，这两个合并为一个接口即可。

因此，请求参数首先要包含点赞有关的数据，并且要标记是点赞还是取消：

- 点赞的目标业务id：bizId
- 谁在点赞（就是登陆用户，可以不用提交）
- 点赞还是取消

除此以外，我们之前说过，在问答、笔记等功能中都会出现点赞功能，所以点赞必须具备通用性。因此还需要在提交一个参数标记点赞的类型：

- 点赞目标的类型

返回值有两种设计：

- 方案一：无返回值，200就是成功，页面直接把点赞数+1展示给用户即可
- 方案二：返回点赞数量，页面渲染

这里推荐使用方案一，因为每次统计点赞数量也有很大的性能消耗。

综上，按照Restful风格设计，接口信息如下：

接口说明	用户可以给自己喜欢的内容点赞，也可以取消点赞
请求方式	POST
请求路径	/likes
请求参数格式	<pre>1 { 2 "bizId": "1578558664933920770", // 点赞业务id 3 "bizType": 1, // 点赞业务类型, 1: 问答; 2: 笔记; .. 4 "liked": true, // 是否点赞, true: 点赞, false: 取消 5 }</pre>
返回值格式	无

3.1.2.实体

请求参数需要定义一个DTO实体类来接收，在课前资料已经提供了：



3.1.3.代码实现

首先是 `tj-remark` 的 `com.tianji.remark.controller.LikedRecordController`：

```
1 package com.tianji.remark.controller;
2
3 import com.tianji.remark.domain.dto.LikeRecordFormDTO;
4 import com.tianji.remark.service.ILikedRecordService;
5 import io.swagger.annotations.Api;
6 import io.swagger.annotations.ApiOperation;
7 import lombok.RequiredArgsConstructor;
8 import org.springframework.web.bind.annotation.*;
9
```

```

10 import javax.validation.Valid;
11 import java.util.List;
12 import java.util.Set;
13
14 /**
15  * <p>
16  * 点赞记录表 控制器
17  * </p>
18  */
19 @RestController
20 @RequiredArgsConstructor
21 @RequestMapping("/likes")
22 @Api(tags = "点赞业务相关接口")
23 public class LikedRecordController {
24
25     private final ILikedRecordService likedRecordService;
26
27     @PostMapping
28     @ApiOperation("点赞或取消点赞")
29     public void addLikeRecord(@Valid @RequestBody LikeRecordFormDTO recordDTO)
30     {
31         likedRecordService.addLikeRecord(recordDTO);
32     }
33 }

```

然后是 `tj-remark` 的 `com.tianji.remark.service.ILikedRecordService` :

```

1 public interface ILikedRecordService extends IService<LikedRecord> {
2
3     void addLikeRecord(LikeRecordFormDTO recordFormDTO);
4 }

```

最后是 `tj-remark` 的实现类

`com.tianji.remark.service.impl.LikedRecordServiceImpl` :

```

1 @Service
2 public class LikedRecordServiceImpl extends ServiceImpl<LikedRecordMapper,
    LikedRecord> implements ILikedRecordService {
3
4     @Override
5     public void addLikeRecord(LikeRecordFormDTO recordFormDTO) {

```

```
6      // TODO 实现点赞或取消点赞
7      }
8  }
```

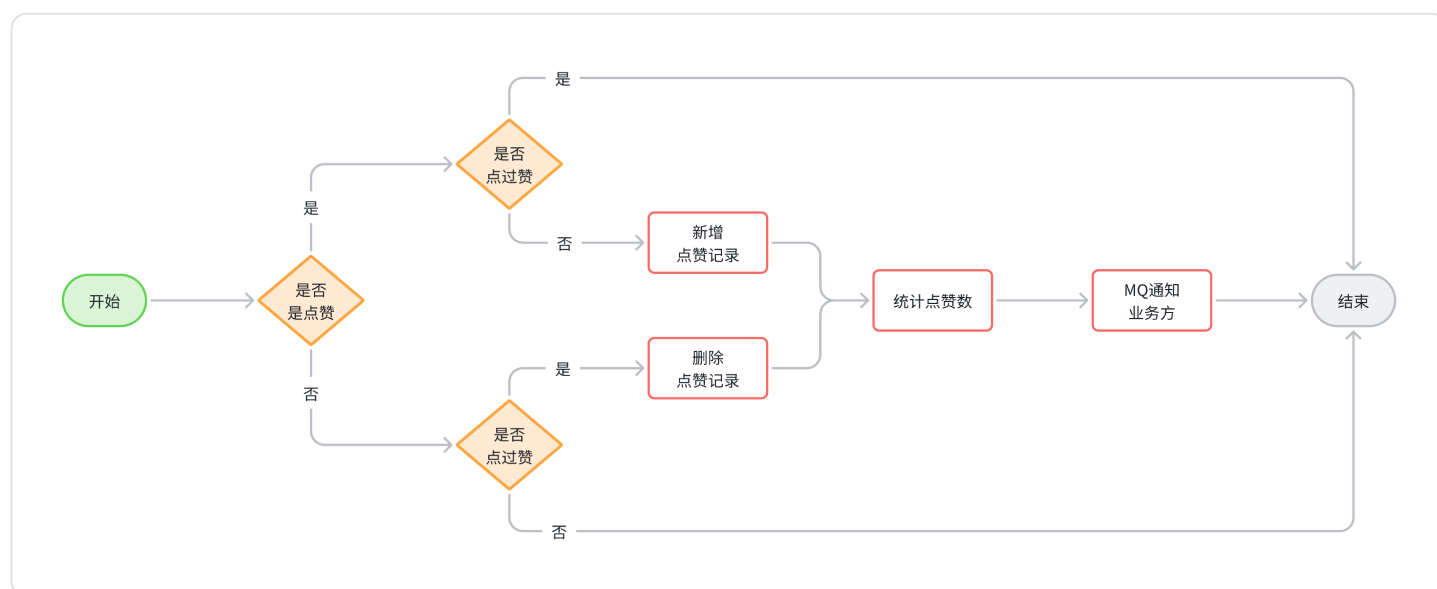
3.1.4.业务流程

我们先梳理一下点赞业务的几点需求：

- 点赞就新增一条点赞记录，取消点赞就删除记录
- 用户不能重复点赞
- 点赞数由具体的业务方保存，需要通知业务方更新点赞数

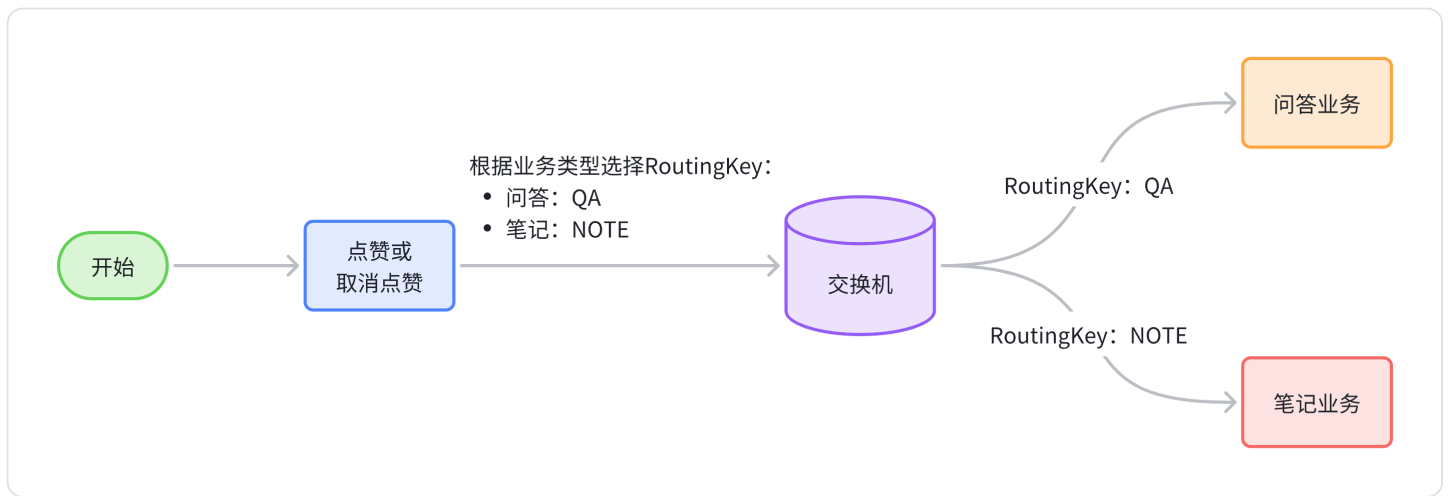
由于业务方的类型很多，比如互动问答、笔记、课程等。所以通知方式必须是**低耦合**的，这里建议使用MQ来实现。

当点赞或取消点赞后，点赞数发生变化，我们就发送MQ通知。整体业务流程如图：



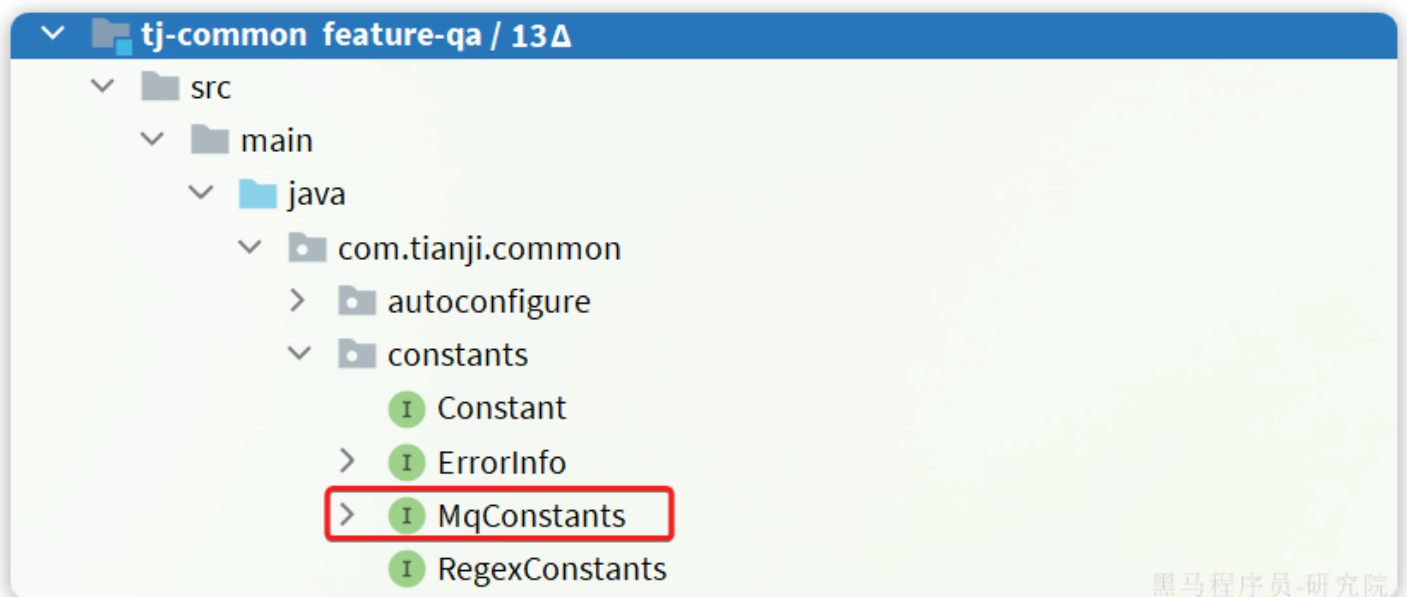
需要注意的是，由于**每次点赞的业务类型不同**，所以**没有必要通知到所有业务方**，而是**仅仅通知与当前点赞业务关联的业务方即可**。

在RabbitMQ中，利用TOPIC类型的交换机，结合不同的RoutingKey，可以实现通知对象的变化。我们需要让不同的业务方监听不同的RoutingKey，然后发送通知时根据点赞类型不同，发送不同RoutingKey：



当然，真实的RoutingKey不一定如图中所示，这里只是做一个示意。

其实在tj-common中，我们已经定义了MQ的常量：



并且定义了点赞有关的 `Exchange` 和 `RoutingKey` 常量：

```

public interface MqConstants {
    interface Exchange{

        /*点赞记录有关的交换机*/
        String LIKE_RECORD_EXCHANGE = "like.record.topic";

        /*问答有关的交换机*/
        interface Key{

            /*点赞的RoutingKey*/
            String LIKED_TIMES_KEY_TEMPLATE = "{}.times.changed";
            /* 问答*/
            String QA_LIKED_TIMES_KEY = "QA.times.changed";
            /*笔记*/
            String NOTE_LIKED_TIMES_KEY = "NOTE.times.changed";
        }
    }
}

```

黑马程序员-研究院

其中的 `RoutingKey` 只是一个模板，其中 `{}` 部分是占位符，不同业务类型就填写不同的具体值。

3.1.5.实现完整业务

首先我们需要定义一个MQ通知的消息体，由于这个消息体会在各个相关微服务中使用，需要定义到公用的模块中，这里我们定义到 `tj-api` 模块：

```

tj-api feature-qa / 13Δ
├── src
│   └── main
│       └── java
│           └── com.tianji.api
│               ├── annotations
│               ├── cache
│               ├── client
│               ├── config
│               ├── constants
│               ├── dto
│               │   ├── auth
│               │   ├── course
│               │   ├── exam
│               │   ├── learning
│               │   ├── remark
│               │   └── LikedTimesDTO
│               ├── sms

```

黑马程序员-研究院

具体代码如下：

```
1 @Data
2 @NoArgsConstructor
3 @AllArgsConstructor
4 public class LikedTimesDTO {
5     /**
6      * 点赞的业务id
7      */
8     private Long bizId;
9     /**
10     * 总的点赞次数
11     */
12     private Integer likedTimes;
13 }
```

在课前资料中已经提供了：

新加卷 (D:) > 课程资料 > 天机学堂 > 课件 > day06-点赞系统 > 资料 >

名称	类型	大小
 LikedTimesDTO.java	Java 源文件	1 KB
 LikeRecordFormDTO.java	Java 源文件	1 KB
 tj_remark.sql	SQL 源文件	1 KB

黑马程序员-研究院

然后是 `com.tianji.remark.service.impl.LikedRecordServiceImpl` 完整的业务逻辑：

```
1 package com.tianji.remark.service.impl;
2
3 import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
4 import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
5 import com.tianji.common.autoconfigure.mq.RabbitMqHelper;
6 import com.tianji.common.utils.StringUtils;
7 import com.tianji.common.utils.UserContext;
8 import com.tianji.remark.domain.dto.LikeRecordFormDTO;
9 import com.tianji.remark.domain.po.LikedRecord;
10 import com.tianji.remark.mapper.LikedRecordMapper;
11 import com.tianji.remark.service.ILikedRecordService;
12 import lombok.RequiredArgsConstructor;
13
```

```
14 import java.util.List;
15 import java.util.Set;
16 import java.util.stream.Collectors;
17
18 import static
    com.tianji.common.constants.MqConstants.Exchange.LIKE_RECORD_EXCHANGE;
19 import static
    com.tianji.common.constants.MqConstants.Key.LIKED_TIMES_KEY_TEMPLATE;
20
21 /**
22  * <p>
23  * 点赞记录表 服务实现类
24  * </p>
25  */
26 @Service
27 @RequiredArgsConstructor
28 public class LikedRecordServiceImpl extends ServiceImpl<LikedRecordMapper,
    LikedRecord> implements ILikedRecordService {
29
30     private final RabbitMqHelper mqHelper;
31
32     @Override
33     public void addLikeRecord(LikeRecordFormDTO recordDTO) {
34         // 1.基于前端的参数,判断是执行点赞还是取消点赞
35         boolean success = recordDTO.getLiked() ? like(recordDTO) :
    unlike(recordDTO);
36         // 2.判断是否执行成功,如果失败,则直接结束
37         if (!success) {
38             return;
39         }
40         // 3.如果执行成功,统计点赞总数
41         Integer likedTimes = lambdaQuery()
42             .eq(LikedRecord::getBizId, recordDTO.getBizId())
43             .count();
44         // 4.发送MQ通知
45         mqHelper.send(
46             LIKE_RECORD_EXCHANGE,
47             StringUtils.format(LIKED_TIMES_KEY_TEMPLATE,
    recordDTO.getBizType()),
48             LikedTimesDTO.of(recordDTO.getBizId(), likedTimes));
49     }
50
51     private boolean unlike(LikeRecordFormDTO recordDTO) {
52         return remove(new QueryWrapper<LikedRecord>().lambda()
53             .eq(LikedRecord::getUserId, UserContext.getUser())
54             .eq(LikedRecord::getBizId, recordDTO.getBizId()));
55     }
56 }
```



```

56
57     private boolean like(LikeRecordFormDTO recordDTO) {
58         Long userId = UserContext.getUser();
59         // 1.查询点赞记录
60         Integer count = lambdaQuery()
61             .eq(LikedRecord::getUserId, userId)
62             .eq(LikedRecord::getBizId, recordDTO.getBizId())
63             .count();
64         // 2.判断是否存在, 如果已经存在, 直接结束
65         if (count > 0) {
66             return false;
67         }
68         // 3.如果不存在, 直接新增
69         LikedRecord r = new LikedRecord();
70         r.setUserId(userId);
71         r.setBizId(recordDTO.getBizId());
72         r.setBizType(recordDTO.getBizType());
73         save(r);
74         return true;
75     }
76 }

```

3.2.批量查询点赞状态

由于这个接口是供其它微服务调用, 实现完成接口后, 还需要定义对应的FeignClient

3.2.1.接口信息

这里是查询多个业务的点赞状态, 因此请求参数自然是业务id的集合。由于是查询当前用户的点赞状态, 因此无需传递用户信息。

经过筛选判断后, 我们把点赞过的业务id集合返回即可。

综上, 按照Restful来设计该接口, 接口信息如下:

接口说明	查询当前用户是否点赞了指定的业务
请求方式	GET
请求路径	/likes/list
请求参数格式	请求数据类型: application/x-www-form-urlencoded

	例如: bizIds=1,2,3 代表业务id集合
返回值格式	<pre>1 [2 "业务id1", "业务id2", "业务id3", "业务id4" 3]</pre>

3.3.2.代码

首先是 `tj-remark` 的 `com.tianji.remark.controller.LikedRecordController` :

```
1 package com.tianji.remark.controller;  
2  
3 import com.tianji.remark.domain.dto.LikeRecordFormDTO;  
4 import com.tianji.remark.service.ILikedRecordService;  
5 import io.swagger.annotations.Api;  
6 import io.swagger.annotations.ApiOperation;  
7 import lombok.RequiredArgsConstructor;  
8 import org.springframework.web.bind.annotation.*;  
9  
10 import javax.validation.Valid;  
11 import java.util.List;  
12 import java.util.Set;  
13  
14 /**  
15  * <p>  
16  * 点赞记录表 控制器  
17  * </p>  
18  */  
19 @RestController  
20 @RequiredArgsConstructor  
21 @RequestMapping("/likes")  
22 @Api(tags = "点赞业务相关接口")  
23 public class LikedRecordController {  
24  
25     private final ILikedRecordService likedRecordService;  
26  
27     @PostMapping  
28     @ApiOperation("点赞或取消点赞")  
29     public void addLikeRecord(@Valid @RequestBody LikeRecordFormDTO recordDTO)  
30     {
```

```

30         likedRecordService.addLikeRecord(recordDTO);
31     }
32
33     @GetMapping("list")
34     @ApiOperation("查询指定业务id的点赞状态")
35     public Set<Long> isBizLiked(@RequestParam("bizIds") List<Long> bizIds){
36         return likedRecordService.isBizLiked(bizIds);
37     }
38 }

```

然后是 `tj-remark` 的 `com.tianji.remark.service.ILikedRecordService` :

```

1 package com.tianji.remark.service;
2
3 import com.tianji.remark.domain.dto.LikeRecordFormDTO;
4 import com.tianji.remark.domain.po.LikedRecord;
5 import com.baomidou.mybatisplus.extension.service.IService;
6
7 import java.util.List;
8 import java.util.Set;
9
10 /**
11  * <p>
12  * 点赞记录表 服务类
13  * </p>
14  */
15 public interface ILikedRecordService extends IService<LikedRecord> {
16
17     void addLikeRecord(LikeRecordFormDTO recordDTO);
18
19     Set<Long> isBizLiked(List<Long> bizIds);
20 }

```

最后是 `tj-remark` 的实现类

`com.tianji.remark.service.impl.LikedRecordServiceImpl` :

```

1 @Override
2 public Set<Long> isBizLiked(List<Long> bizIds) {
3     // 1. 获取登录用户id
4     Long userId = UserContext.getUser();
5     // 2. 查询点赞状态
6     List<LikedRecord> list = lambdaQuery()
7         .in(LikedRecord::getBizId, bizIds)

```

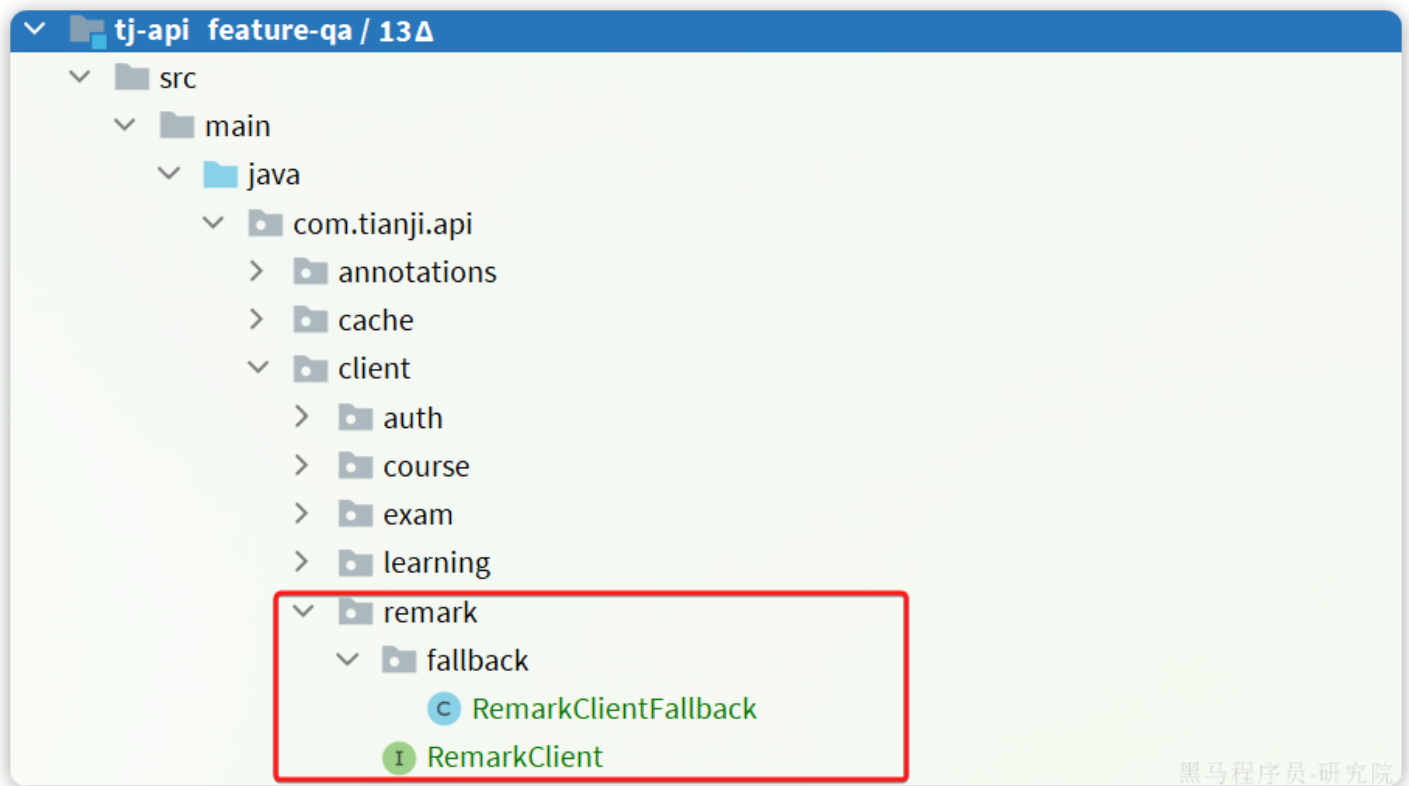
```

8         .eq(LikedRecord::getUserId, userId)
9         .list();
10    // 3.返回结果
11    return
    list.stream().map(LikedRecord::getBizId).collect(Collectors.toSet());
12 }

```

3.3.3.暴露Feign接口

由于该接口是给其它微服务调用的，所以必须暴露出Feign客户端，并且定义好fallback降级处理：我们在tj-api模块中定义一个客户端：



其中RemarkClient如下：

```

1 package com.tianji.api.client.remark;
2
3 import com.tianji.api.client.remark.fallback.RemarkClientFallback;
4 import org.springframework.cloud.openfeign.FeignClient;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RequestParam;
7
8 import java.util.Set;

```

```

9
10 @FeignClient(value = "remark-service", fallbackFactory =
    RemarkClientFallback.class)
11 public interface RemarkClient {
12     @GetMapping("/likes/list")
13     Set<Long> isBizLiked(@RequestParam("bizIds") Iterable<Long> bizIds);
14 }

```

对应的fallback逻辑:

```

1 package com.tianji.api.client.remark.fallback;
2
3 import com.tianji.api.client.remark.RemarkClient;
4 import com.tianji.common.utils.CollUtils;
5 import lombok.extern.slf4j.Slf4j;
6 import org.springframework.cloud.openfeign.FallbackFactory;
7
8 import java.util.Set;
9
10 @Slf4j
11 public class RemarkClientFallback implements FallbackFactory<RemarkClient> {
12
13     @Override
14     public RemarkClient create(Throwable cause) {
15         log.error("查询remark-service服务异常", cause);
16         return new RemarkClient() {
17
18             @Override
19             public Set<Long> isBizLiked(Iterable<Long> bizIds) {
20                 return CollUtils.emptySet();
21             }
22         };
23     }
24 }

```

由于 `RemarkClientFallback` 是定义在 `tj-api` 的 `com.tianji.api` 包, 由于每个微服务扫描包不一致。因此其它引用 `tj-api` 的微服务是无法通过扫描包加载到这个类的。

我们需要通过SpringBoot的自动加载机制来加载这些fallback类:



由于SpringBoot会在启动时读取 `/META-INF/spring.factories` 文件，我们只需要在该文件中指定了要加载

`FallbackConig` 类：

```
1 @Configuration
2 public class FallbackConfig {
3     @Bean
4     public LearningClientFallback learningClientFallback(){
5         return new LearningClientFallback();
6     }
7
8     @Bean
9     public TradeClientFallback tradeClientFallback(){
10         return new TradeClientFallback();
11     }
12
13     @Bean
14     public RemarkClientFallback remarkClientFallback(){
15         return new RemarkClientFallback();
16     }
17 }
```

这样所有在其中定义的fallback类都会被加载了。

3.3.3.改造查询回复接口

开发查询点赞状态接口的目的，是为了在查询用户回答和评论时，能看到当前用户是否点赞了。所以我们需要改造之前实现的分页查询回答或评论的接口。

首先找到 `tj-api` 中的

`com.tianji.learning.service.impl.InteractionReplyServiceImpl`，注入评价服务的Feign客户端：

```
@Service
@RequiredArgsConstructor
public class InteractionReplyServiceImpl extends ServiceImpl<InteractionReplyMapper, InteractionReply> {

    private final IInteractionQuestionService questionService;
    private final UserClient userClient;
    private final RemarkClient remarkClient;
```

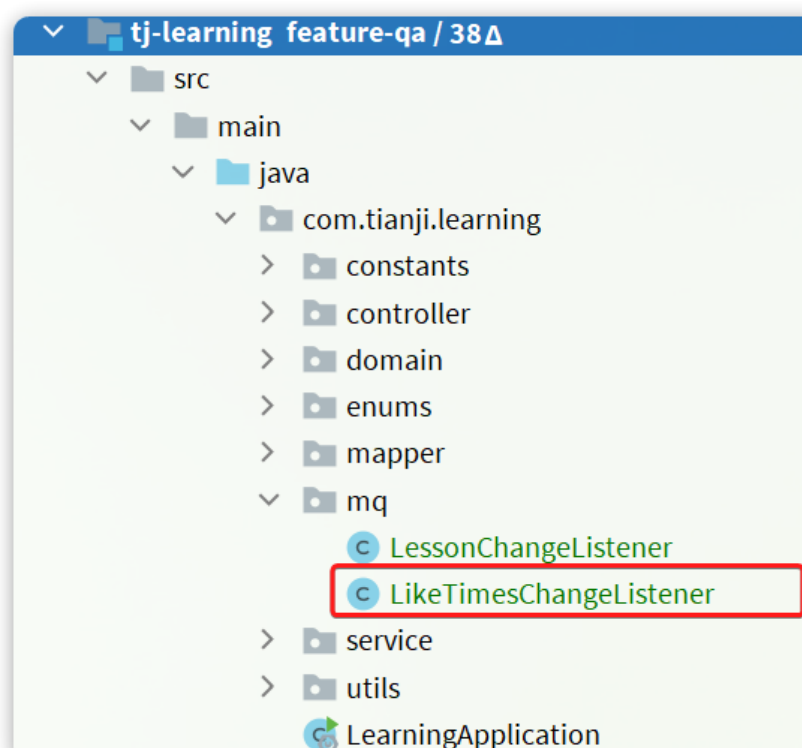
黑马程序员-研究院

然后改造分页查询回答的业务即可，由于分页查询回答是大家各自实现的，这部分改造也有大家来实现。

3.4.监听点赞变更的消息

既然点赞后会发送MQ消息通知业务服务，那么每一个有关的业务服务都应该监听点赞数变更的消息，更新本地的点赞数量。

例如互动问答，我们需要在 `tj-learning` 服务中定义MQ监听器：



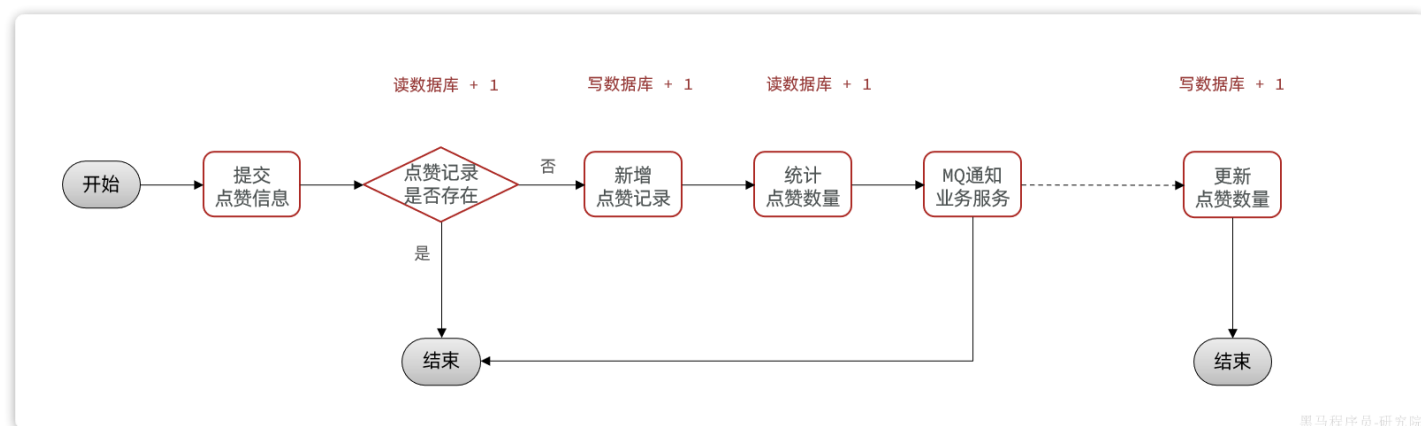
黑马程序员-研究院

具体代码如下：

```
1 package com.tianji.learning.mq;
2
3 import com.tianji.api.dto.remark.LikedTimesDTO;
4 import com.tianji.learning.domain.po.InteractionReply;
5 import com.tianji.learning.service.IInteractionReplyService;
6 import lombok.RequiredArgsConstructor;
7 import lombok.extern.slf4j.Slf4j;
8 import org.springframework.amqp.core.ExchangeTypes;
9 import org.springframework.amqp.rabbit.annotation.Exchange;
10 import org.springframework.amqp.rabbit.annotation.Queue;
11 import org.springframework.amqp.rabbit.annotation.QueueBinding;
12 import org.springframework.amqp.rabbit.annotation.RabbitListener;
13 import org.springframework.stereotype.Component;
14
15 import java.util.ArrayList;
16 import java.util.List;
17
18 import static
    com.tianji.common.constants.MqConstants.Exchange.LIKE_RECORD_EXCHANGE;
19 import static com.tianji.common.constants.MqConstants.Key.QA_LIKED_TIMES_KEY;
20
21 @Slf4j
22 @Component
23 @RequiredArgsConstructor
24 public class LikeTimesChangeListener {
25
26     private final IInteractionReplyService replyService;
27
28     @RabbitListener(bindings = @QueueBinding(
29         value = @Queue(name = "qa.liked.times.queue", durable = "true"),
30         exchange = @Exchange(name = LIKE_RECORD_EXCHANGE, type =
    ExchangeTypes.TOPIC),
31         key = QA_LIKED_TIMES_KEY
32     ))
33     public void listenReplyLikedTimesChange(LikedTimesDTO dto){
34         log.debug("监听到回答或评论{}的点赞数变更:{}", dto.getBizId(),
    dto.getLikedTimes());
35         InteractionReply r = new InteractionReply();
36         r.setId(dto.getBizId());
37         r.setLikedTimes(dto.getLikedTimes());
38         replyService.updateById(r);
39     }
40 }
```


4.点赞功能改进

虽然我们初步实现了点赞功能，不过有一个非常严重的问题，点赞业务包含多次数据库读写操作：



更重要的是，点赞操作波动较大，有可能会在短时间内访问量激增。例如有人非常频繁的点赞、取消点赞。这样就会给数据库带来非常大的压力。

怎么办呢？

4.1.改进思路分析

其实在实现提交学习记录的时候，我们就给大家分析过高并发问题的处理方案。点赞业务与提交播放记录类似，都是高并发写操作。

按照之前我们讲的，高并发写操作常见的优化手段有：

- 优化SQL和代码
- 变同步写为异步写
- 合并写请求

有同学可能会说，我们更新业务方点赞数量的时候，不就是利用MQ异步写来实现的吗？

没错，确实如此，虽然异步写减少了业务执行时间，降低了数据库写频率。不过此处更重要的是利用MQ来解耦。而且数据库的写次数没有减少，压力依然很大。

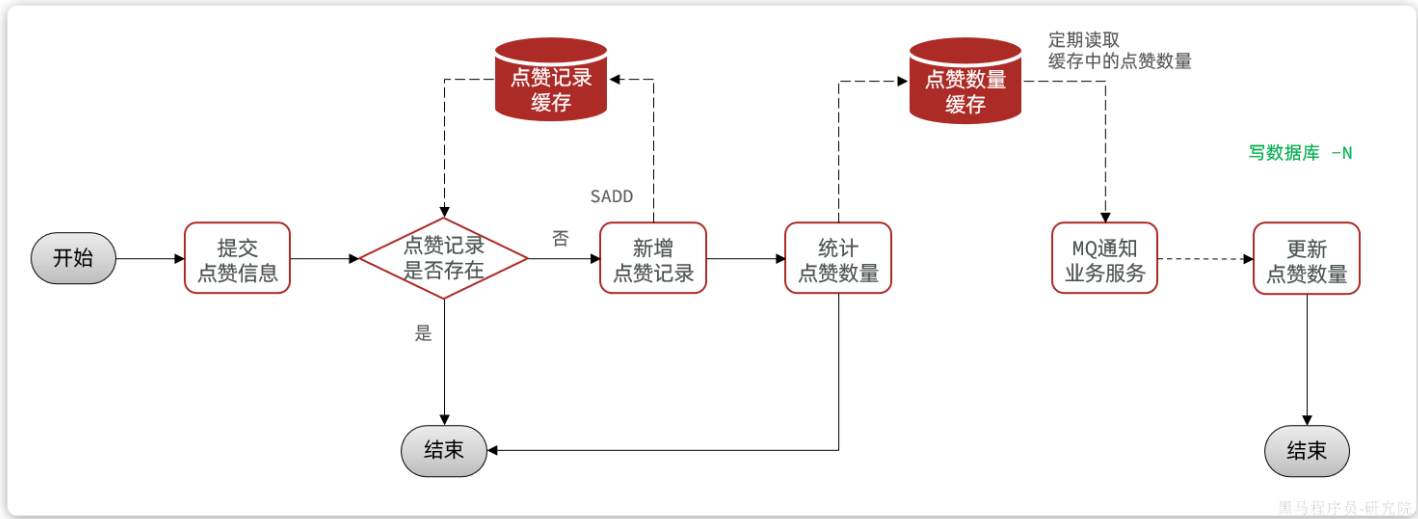
所以，我们应该像之前播放记录业务一样，采用合并写请求的方案。当然，现在的异步处理也保留，这样就兼顾了**异步写**、**合并写**的优势。

需要注意的是，合并写是有使用场景的，必须是对中间的N次写操作不敏感的情况下。点赞业务是否符合这一需求呢？

无论用户中间执行点赞、取消、再点赞、再取消多少次，点赞次数发生了多少次变化，业务方只关注最终的点赞结果即可：

- 用户是否点赞了
- 业务的总点赞次数

因此，点赞功能可以使用合并写方案。最终我们的点赞业务流程变成这样：



合并写请求有两个关键点要考虑：

- 数据如何缓存
- 缓存何时写入数据库

4.1.1.点赞数据缓存

点赞记录中最两个关键信息：

- 用户是否点赞
- 某业务的点赞总次数

这两个信息需要分别记录，也就是说我们需要在Redis中设计两种数据结构分别存储。

4.1.1.1.用户是否点赞

要知道某个用户是否点赞某个业务，就必须记录业务id以及给业务点赞的所有用户id。由于一个业务可以被很多用户点赞，显然是需要一个集合来记录。而Redis中的集合类型包含四种：

- List
- Set

- SortedSet
- Hash

而要判断用户是否点赞，就是判断存在且唯一。显然，Set集合是最合适的。我们可以用业务id为Key，创建Set集合，将点赞的所有用户保存其中，格式如下：

KEY (bizId)	VALUE(userId)
bizId:1	userId:1
	userId:2
	userId:3

可以使用Set集合的下列命令完成点赞功能：

```
1 # 判断用户是否点赞
2 SISMEMBER bizId userId
3 # 点赞，如果返回1则代表点赞成功，返回0则代表点赞失败
4 SADD bizId userId
5 # 取消点赞，就是删除一个元素
6 SREM bizId userId
7 # 统计点赞总数
8 SCARD bizId
```

由于Redis本身具备持久化机制，AOF提供的数据可靠性已经能够满足点赞业务的安全需求，因此我们完全可以用Redis存储来代替数据库的点赞记录。

也就是说，用户的一切点赞行为，以及将来查询点赞状态我们可以都走Redis，不再使用数据库查询。

！ 有同学会担心，如果点赞数据非常庞大，达到数百亿，那么该怎么办呢？

大多数企业根本达不到这样的规模，如果真的达到也没有关系。这个时候我们可以将Redis与数据库结合。

- 先利用Redis来记录点赞状态
- 并且定期的将Redis中的点赞状态持久化到数据库
- 对于历史点赞记录，比如下架的课程、或者超过2年以上的访问量较低的数据都可以从redis移除，只保留在数据库中

- 当某个记录点赞时，优先去Redis查询并判断，如果Redis中不存在，再去查询数据库数据并缓存到Redis

4.1.1.2.点赞次数

由于点赞次数需要在业务方持久化存储到数据库，因此Redis只起到缓存作用即可。

由于需要记录业务id、业务类型、点赞数三个信息：

- 一个业务类型下包含多个业务id
- 每个业务id对应一个点赞数。

因此，我们可以把每一个业务类型作为一组，使用Redis的一个key，然后业务id作为键，点赞数作为值。这样的键值对集合，有两种结构都可以满足：

- Hash：传统键值对集合，无序
- SortedSet：基于Hash结构，并且增加了跳表。因此可排序，但更占用内存

如果是从节省内存角度来考虑，Hash结构无疑是最佳的选择；但是考虑到将来我们要从Redis读取点赞数，然后移除（避免重复处理）。为了保证线程安全，查询、移除操作必须具备原子性。而SortedSet则提供了几个移除并获取的功能，天生具备原子性。并且我们每隔一段时间就会将数据从Redis移除，并不会占用太多内存。因此，这里我们计划使用SortedSet结构。

格式如下：

KEY (bizType)	Member(bizId)	Score(likedTimes)
likes:qa	bizId:1001	10
	bizId:1002	5
likes:note	bizId:2001	9
	bizId:2002	21

当用户对某个业务点赞时，我们统计点赞总数，并将其缓存在Redis中。这样一来在一段时间内，不管有多少用户对该业务点赞（热点业务数据，比如某个微博大V），都只在Redis中修改点赞总数，无需修改数据库。

4.1.2.点赞数据入库

点赞数据写入缓存了，但是这里有一个新的问题：
何时把缓存的点赞数，通过MQ通知到业务方，持久化到业务方的数据库呢？

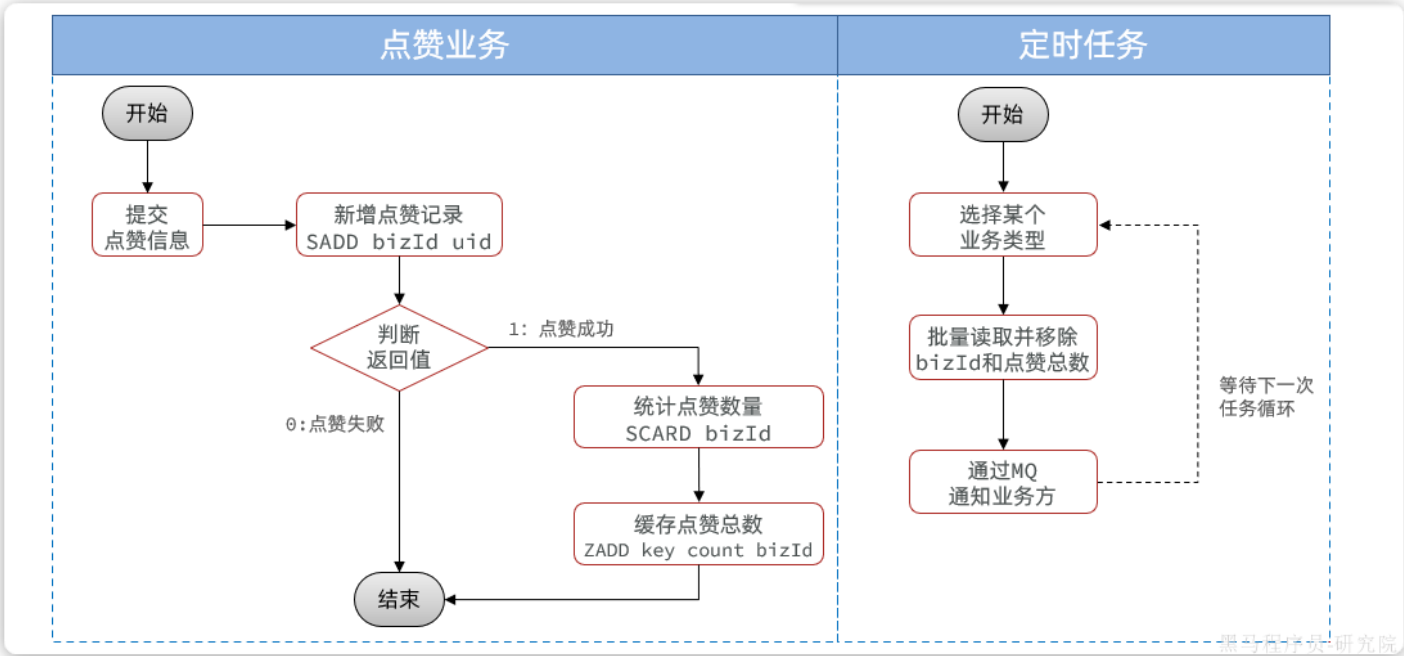
在之前的提交播放记录业务中，由于播放记录是定期每隔15秒发送一次请求，频率固定。因此我们可以通过接收到播放记录后延迟20秒检测数据变更来确定是否有新数据到达。

但是点赞则不然，用户何时点赞、点赞频率如何完全不确定。因此无法采用延迟检测这样的手段。怎么办？

事实上这也是大多数**合并写请求**业务面临的问题，而多数情况下，我们只能通过**定时任务**，定期将缓存的数据持久化到数据库中。

4.1.3.流程图

综上所述，基于Redis做写缓存后，点赞流程如下：



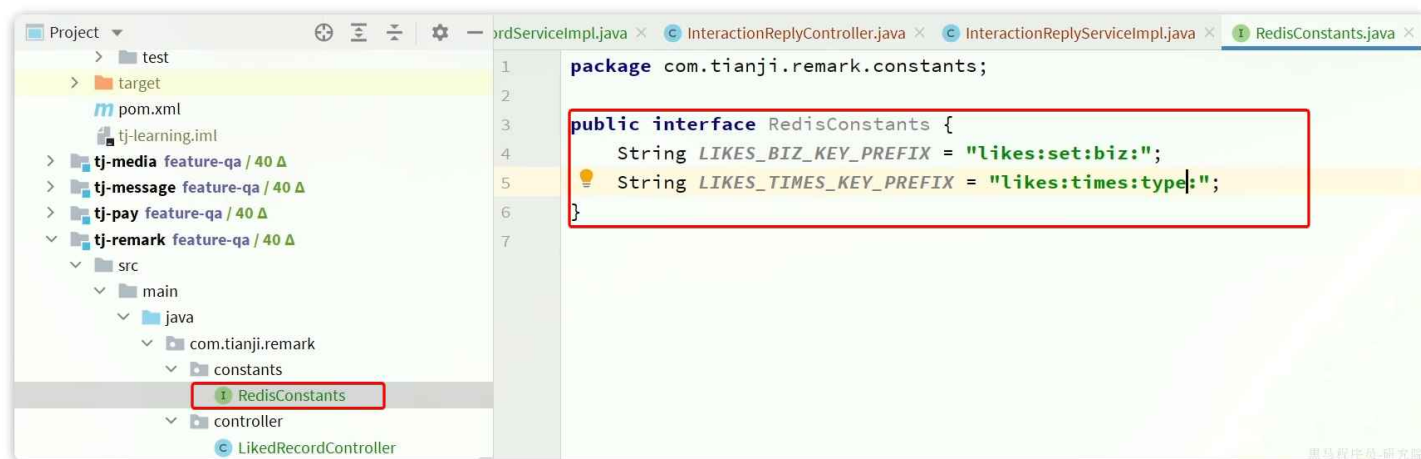
4.2.改造点赞逻辑

需要改造的内容包括：

- `tj-remark` 中所有点赞有关接口
 - 点赞接口
 - 查询单个点赞状态
 - 批量查询点赞状态

- `tj-remark` 处理点赞数据持久化的定时任务
- `tj-learning` 监听点赞数变更消息的业务

由于需要访问Redis，我们提前定义一个常量类，把Redis相关的Key定义为常量：



代码如下：

```
1 public interface RedisConstants {
2     /*给业务点赞的用户集合的KEY前缀，后缀是业务id*/
3     String LIKES_BIZ_KEY_PREFIX = "likes:set:biz:";
4     /*业务点赞数统计的KEY前缀，后缀是业务类型*/
5     String LIKES_TIMES_KEY_PREFIX = "likes:times:type:";
6 }
```

4.2.1.点赞接口

接下来，我们定义一个新的点赞业务实现类：

- ▼ **tj-remark**
 - ▼ **src**
 - ▼ **main**
 - ▼ **java**
 - ▼ **com.tianji.remark**
 - > **constants**
 - > **controller**
 - > **domain**
 - > **mapper**
 - ▼ **service**
 - ▼ **impl**
 - LikedRecordServiceImpl
 - LikedRecordServiceRedisImpl**
 - ILikedRecordService
- > **task**

黑马程序员-研究院

并将LikedRecordServiceImpl注释：

点赞记录表 服务实现类

Author: 虎哥

// @Service

@RequiredArgsConstructor

public class LikedRecordServiceImpl **extends** ServiceImpl<Li

private final RabbitMqHelper mqHelper;

黑马程序员-研究院

代码如下：

```

1 package com.tianji.remark.service.impl;
2
3 import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
4 import com.tianji.api.dto.remark.LikedTimesDTO;
5 import com.tianji.common.autoconfigure.mq.RabbitMqHelper;
6 import com.tianji.common.utils.CollUtils;
7 import com.tianji.common.utils.StringUtils;
8 import com.tianji.common.utils.UserContext;
9 import com.tianji.remark.constants.RedisConstants;

```

```

10 import com.tianji.remark.domain.dto.LikeRecordFormDTO;
11 import com.tianji.remark.domain.po.LikedRecord;
12 import com.tianji.remark.mapper.LikedRecordMapper;
13 import com.tianji.remark.service.ILikedRecordService;
14 import lombok.RequiredArgsConstructor;
15 import org.springframework.data.redis.connection.StringRedisConnection;
16 import org.springframework.data.redis.core.RedisCallback;
17 import org.springframework.data.redis.core.StringRedisTemplate;
18 import org.springframework.data.redis.core.ZSetOperations;
19 import org.springframework.stereotype.Service;
20
21 import java.util.ArrayList;
22 import java.util.List;
23 import java.util.Set;
24 import java.util.stream.Collectors;
25 import java.util.stream.IntStream;
26
27 import static
    com.tianji.common.constants.MqConstants.Exchange.LIKE_RECORD_EXCHANGE;
28 import static
    com.tianji.common.constants.MqConstants.Key.LIKED_TIMES_KEY_TEMPLATE;
29
30 /**
31  * <p>
32  * 点赞记录表 服务实现类
33  * </p>
34  */
35 @Service
36 @RequiredArgsConstructor
37 public class LikedRecordServiceRedisImpl extends
    ServiceImpl<LikedRecordMapper, LikedRecord> implements ILikedRecordService {
38
39     private final RabbitMqHelper mqHelper;
40     private final StringRedisTemplate redisTemplate;
41
42     @Override
43     public void addLikeRecord(LikeRecordFormDTO recordDTO) {
44         // 1.基于前端的参数,判断是执行点赞还是取消点赞
45         boolean success = recordDTO.getLiked() ? like(recordDTO) :
        unlike(recordDTO);
46         // 2.判断是否执行成功,如果失败,则直接结束
47         if (!success) {
48             return;
49         }
50         // 3.如果执行成功,统计点赞总数
51         Long likedTimes = redisTemplate.opsForSet()

```



```

52         .size(RedisConstants.LIKES_BIZ_KEY_PREFIX +
recordDTO.getBizId());
53         if (likedTimes == null) {
54             return;
55         }
56         // 4.缓存点总数到Redis
57         redisTemplate.opsForZSet().add(
58             RedisConstants.LIKES_TIMES_KEY_PREFIX + recordDTO.getBizType(),
59             recordDTO.getBizId().toString(),
60             likedTimes
61         );
62     }
63
64     private boolean unlike(LikeRecordFormDTO recordDTO) {
65         // 1.获取用户id
66         Long userId = UserContext.getUser();
67         // 2.获取Key
68         String key = RedisConstants.LIKES_BIZ_KEY_PREFIX +
recordDTO.getBizId();
69         // 3.执行SREM命令
70         Long result = redisTemplate.opsForSet().remove(key, userId.toString());
71         return result != null && result > 0;
72     }
73
74     private boolean like(LikeRecordFormDTO recordDTO) {
75         // 1.获取用户id
76         Long userId = UserContext.getUser();
77         // 2.获取Key
78         String key = RedisConstants.LIKES_BIZ_KEY_PREFIX +
recordDTO.getBizId();
79         // 3.执行SADD命令
80         Long result = redisTemplate.opsForSet().add(key, userId.toString());
81         return result != null && result > 0;
82     }
83 }

```

4.2.2.批量查询点赞状态统计

目前我们的Redis点赞记录数据结构如下：

KEY (bizId)	VALUE(userId)
bizId:1	userId:1

	userId:2
	userId:3

当我们判断某用户是否点赞时，需要使用下面命令：

```
1 # 判断用户是否点赞
2 SISMEMBER bizId userId
```

需要注意的是，这个命令只能判断**一个用户对某一个业务**的点赞状态。而我们的接口是要查询当前用户对多个业务的点赞状态。

因此，我们就需要多次调用 `SISMEMBER` 命令，也就需要向Redis多次发起网络请求，给网络带宽带来非常大的压力，影响业务性能。

那么，有没有办法能够一个命令完成多个业务点赞状态判断呢？

非常遗憾，答案是没有！只能多次执行 `SISMEMBER` 命令来判断。

不过，Redis中提供了一个功能，可以在一次请求中执行多个命令，实现批处理效果。这个功能就是 Pipeline

<https://redis.io/docs/manual/pipelining/>

Redis pipelining

How to optimize round-trip times by batching Redis commands

中文文档：

<https://www.redis.com.cn/topics/pipelining.html>

redis使用管道(Pipelining)提高查询速度

首页 Redis 命令 Redis 教程 Redis 客户端 Redis 中文文档 Redis 下载 Redis 模块 首页 Redis 教程 Redis 命令手册 Redis 客户端 中文文档 下载 模块 * redis使用管道(Pipelining)提高查询速度 * 请求/响应协议和RTT redis 管道(Pipelining) Redis是一种基于客...



不要在一次批处理中传输太多命令，否则单次命令占用带宽过多，会导致网络阻塞

Spring提供的RedisTemplate也具备pipeline功能，最终批量查询点赞状态功能实现如下：

```
1 @Override
2 public Set<Long> isBizLiked(List<Long> bizIds) {
3     // 1. 获取登录用户id
4     Long userId = UserContext.getUser();
5     // 2. 查询点赞状态
6     List<Object> objects =
7         redisTemplate.executePipelined((RedisCallback<Object>) connection -> {
8             StringRedisConnection src = (StringRedisConnection) connection;
9             for (Long bizId : bizIds) {
10                 String key = RedisConstants.LIKES_BIZ_KEY_PREFIX + bizId;
11                 src.sIsMember(key, userId.toString());
12             }
13             return null;
14         });
15     // 3. 返回结果
16     return IntStream.range(0, objects.size()) // 创建从0到集合size的流
17         .filter(i -> (boolean) objects.get(i)) // 遍历每个元素，保留结果为true
18         .mapToObj(bizIds::get) // 用角标i取bizIds中的对应数据，就是点赞过的id
19         .collect(Collectors.toSet()); // 收集
20 }
```

4.2.3.定时任务

点赞成功后，会更新点赞总数并写入Redis中。而我们需要定时读取这些点赞总数的变更数据，通过MQ发送给业务方。这就需要定时任务来实现了。

定时任务的实现方案有很多，简单的例如：

- SpringTask
- Quartz

还有一些依赖第三方服务的分布式任务框架：

- Elastic-Job
- XXL-Job

此处我们先使用简单的SpringTask来实现并测试效果。

首先，在 `tj-remark` 模块的 `RemarkApplication` 启动类上添加注解：

```

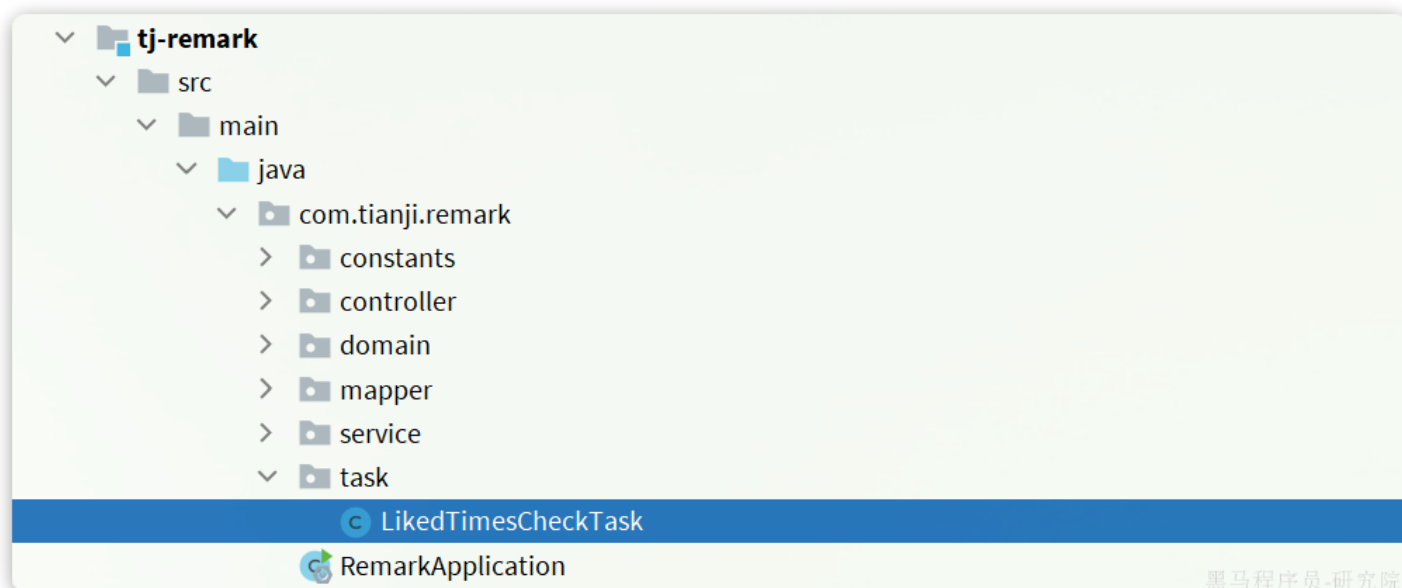
@EnableScheduling
@Slf4j
@MapperScan("com.tianji.remark.mapper")
@SpringBootApplication
public class RemarkApplication {

    public static void main(String[] args) { SpringApplication
}

```

其作用就是启用Spring的定时任务功能。

然后，定义一个定时任务处理器类：



代码如下：

```

1 package com.tianji.remark.task;
2
3 import com.tianji.remark.service.ILikedRecordService;
4 import lombok.RequiredArgsConstructor;
5 import org.springframework.scheduling.annotation.Scheduled;
6 import org.springframework.stereotype.Component;
7
8 import java.util.List;
9
10 @Component
11 @RequiredArgsConstructor

```

```

12 public class LikedTimesCheckTask {
13
14     private static final List<String> BIZ_TYPES = List.of("QA", "NOTE");
15     private static final int MAX_BIZ_SIZE = 30;
16
17     private final ILikedRecordService recordService;
18
19     @Scheduled(fixedDelay = 20000)
20     public void checkLikedTimes(){
21         for (String bizType : BIZ_TYPES) {
22             recordService.readLikedTimesAndSendMessage(bizType, MAX_BIZ_SIZE);
23         }
24     }
25 }

```

由于可能存在多个业务类型，不能厚此薄彼只处理部分业务。所以我们会遍历多种业务类型，分别处理。同时为了避免一次处理的业务过多，这里设定了每次处理的业务数量为30，当然这些都是可以调整的。

真正处理业务的逻辑封装到了 `ILikedRecordService` 中：

```

1 public interface ILikedRecordService extends IService<LikedRecord> {
2     // ... 略
3
4     void readLikedTimesAndSendMessage(String bizType, int maxBizSize);
5 }

```

其实现类：

```

1 @Override
2 public void readLikedTimesAndSendMessage(String bizType, int maxBizSize) {
3     // 1. 读取并移除Redis中缓存的点赞总数
4     String key = RedisConstants.LIKES_TIMES_KEY_PREFIX + bizType;
5     Set<ZSetOperations.TypedTuple<String>> tuples =
6         redisTemplate.opsForZSet().popMin(key, maxBizSize);
7     if (CollUtils.isEmpty(tuples)) {
8         return;
9     }
10    // 2. 数据转换
11    List<LikedTimesDTO> list = new ArrayList<>(tuples.size());
12    for (ZSetOperations.TypedTuple<String> tuple : tuples) {
13        String bizId = tuple.getValue();

```

```

13         Double likedTimes = tuple.getScore();
14         if (bizId == null || likedTimes == null) {
15             continue;
16         }
17         list.add(LikedTimesDTO.of(Long.valueOf(bizId), likedTimes.intValue()));
18     }
19     // 3. 发送MQ消息
20     mqHelper.send(
21         LIKE_RECORD_EXCHANGE,
22         StringUtils.format(LIKED_TIMES_KEY_TEMPLATE, bizType),
23         list);
24 }

```

4.2.4. 监听点赞数变更

需要注意的是，由于在定时任务中一次最多处理20条数据，这些数据就需要通过MQ一次发送到业务方，也就是说MQ的消息体变成了一个集合：

```

// 2. 数据转换
List<LikedTimesDTO> list = new ArrayList<>(tuples.size());
for (ZSetOperations.TypedTuple<String> tuple : tuples) {
    String bizId = tuple.getValue();
    Double likedTimes = tuple.getScore();
    if (bizId == null || likedTimes == null) {
        continue;
    }
    list.add(LikedTimesDTO.of(Long.valueOf(bizId), likedTimes.intValue()));
}
// 3. 发送MQ消息
mqHelper.send(
    LIKE_RECORD_EXCHANGE,
    StringUtils.format(LIKED_TIMES_KEY_TEMPLATE, bizType),
    list);

```

黑马程序员-研究院

因此，作为业务方，在监听MQ消息的时候也必须接收集合格式。

我们修改 `tj-learning` 中的类

`com.tianji.learning.mq.LikeTimesChangeListener`：

```

1 package com.tianji.learning.mq;
2
3 import com.tianji.api.dto.remark.LikedTimesDTO;
4 import com.tianji.learning.domain.po.InteractionReply;
5 import com.tianji.learning.service.IInteractionReplyService;

```

```

6  import lombok.RequiredArgsConstructor;
7  import lombok.extern.slf4j.Slf4j;
8  import org.springframework.amqp.core.ExchangeTypes;
9  import org.springframework.amqp.rabbit.annotation.Exchange;
10 import org.springframework.amqp.rabbit.annotation.Queue;
11 import org.springframework.amqp.rabbit.annotation.QueueBinding;
12 import org.springframework.amqp.rabbit.annotation.RabbitListener;
13 import org.springframework.stereotype.Component;
14
15 import java.util.ArrayList;
16 import java.util.List;
17
18 import static
    com.tianji.common.constants.MqConstants.Exchange.LIKE_RECORD_EXCHANGE;
19 import static com.tianji.common.constants.MqConstants.Key.QA_LIKED_TIMES_KEY;
20
21 @Slf4j
22 @Component
23 @RequiredArgsConstructor
24 public class LikeTimesChangeListener {
25
26     private final IInteractionReplyService replyService;
27
28     @RabbitListener(bindings = @QueueBinding(
29         value = @Queue(name = "qa.liked.times.queue", durable = "true"),
30         exchange = @Exchange(name = LIKE_RECORD_EXCHANGE, type =
31             ExchangeTypes.TOPIC),
32         key = QA_LIKED_TIMES_KEY
33     ))
34     public void listenReplyLikedTimesChange(List<LikedTimesDTO> likedTimesDTOs)
35     {
36         log.debug("监听到回答或评论的点赞数变更");
37
38         List<InteractionReply> list = new ArrayList<>(likedTimesDTOs.size());
39         for (LikedTimesDTO dto : likedTimesDTOs) {
40             InteractionReply r = new InteractionReply();
41             r.setId(dto.getBizId());
42             r.setLikedTimes(dto.getLikedTimes());
43             list.add(r);
44         }
45         replyService.updateBatchById(list);
46     }
47 }

```

5.练习

5.1.完善互动问答功能

在互动问答功能中，有一些与点赞有关的之前暂未实现，请补充完整

5.2.点赞业务类型的动态配置

目前，点赞业务类型是写死在代码中的。将其定义到配置文件，交给nacos管理，实现动态加载效果。并将业务中与点赞类型有关的“魔法值”去除，改为读取配置文件中的业务类型。

5.3.点赞记录持久化（难度较大，选做）

思考一下，如果要把点赞记录定期持久化到数据库，查询时再加载，该如何实现？讲解一下你的思路。

如果有能力的话自己尝试实现一下。

5.4.定时任务（难度较大，选做）

研究一下XXL-JOB这个定时任务框架，尝试利用它来代替SpringTask。

6.面试

面试官：看你项目中介绍，你负责点赞功能的设计和开发，那你能不能讲讲你们的点赞系统是如何设计的？



答：首先在设计之初我们分析了一下点赞业务可能需要的一些要求。

例如，在我们项目中需要用到点赞的业务不止一个，因此点赞系统必须具备通用性，独立性，不能跟具体业务耦合。

再比如，点赞业务可能会有较高的并发，我们要考虑到高并发写库的压力问题。

所以呢，我们在设计的时候，就将点赞功能抽离出来作为独立服务。当然这个服务中除了点赞功能以外，还有与之关联的评价功能，不过这部分我就没有参与了。在数据层面也会用业

务类型对不同点赞数据做隔离。

从具体实现上来说，为了减少数据库压力，我们会利用Redis来保存点赞记录、点赞数量信息。然后利用定时任务定期的将点赞数量同步给业务方，持久化到数据库中。

注意事项：回答时要先说自己的思考过程，再说具体设计，彰显你的逻辑清晰。设计的时候先不说细节，只说大概，停顿一下，吸引面试官去追问细节。如果面试官不追问，停顿一下后，自己接着说下面的

面试官追问：那你们Redis中具体使用了哪种数据结构呢？



答：我们使用了两种数据结构，set和zset

首先保存点赞记录，使用了set结构，key是业务类型+业务id，值是点赞过的用户id。当用户点赞时就 `SADD` 用户id进去，当用户取消点赞时就 `SREM` 删除用户id。当判断是否点赞时使用 `SISMEMBER` 即可。当要统计点赞数量时，只需要 `SCARD` 就行，而Redis的SET结构会在头信息中保存元素数量，因此SCARD直接读取该值，时间复杂度为 $O(1)$ ，性能非常好。

为什么不用用户id为key，业务id为值呢？如果用户量很大，可能出现BigKey？

您说的这个方案也是可以的，不过呢，考虑到我们的项目数据量并不会很大，我们不会有大V，因此点赞数量通常不会超过1000，因此不会出现BigKey。并且，由于我们采用了业务id为KEY，当我们要统计点赞数量时，可以直接使用SCARD来获取元素数量，无需额外保存，这是一个很大的优势。但如果是考虑到有大V的场景，有两种选择，一种还是应该选择您说的这种方案，另一种则是对用户id做hash分片，将大V的key拆分到多个KEY中，结构为[bizType:bizId:userId高8位]

不过这里存在一个问题，就是页面需要判断当前用户有没有对某些业务点赞。这个时候会传来多个业务id的集合，而SISMEMBER只能一次判断一个业务的点赞状态，要判断多个业务的点赞状态，就必须多次调用SISMEMBER命令，与Redis多次交互，这显然是不合适的。（此处略停顿，等待面试官追问，面试官可能会问“那你们怎么解决的”。如果没追问，自己接着说），所以呢我们就采用了Pipeline管道方式，这样就可以一次请求实现多个业务点赞状态的判断了。

面试官追问（可能会）：那你ZSET干什么用的？

💡 答：严格来说ZSET并不是用来实现点赞业务的，因为点赞只靠SET就能实现了。但是这里有一个问题，我们要定期将业务方的点赞总数通过MQ同步给业务方，并持久化到数据库。但是如果只有SET，我没办法知道哪些业务的点赞数发生了变化，需要同步到业务方。

因此，我们又添加了一个ZSET结构，用来记录点赞数变化的业务及对应的点赞总数。可以理解为一个待持久化的点赞任务队列。

每当业务被点赞，除了要缓存点赞记录，还要把业务id及点赞总数写入ZSET。这样定时任务开启时，只需要从ZSET中获取并移除数据，然后发送MQ给业务方，并持久化到数据库即可。

面试官追问（可能会，没追问就自己说）：那为什么一定要用ZSET结构，把更新过的业务扔到一个List中不行吗？

💡 答：扔到List结构中虽然也能实现，但是存在一些问题：

首先，假设定时任务每隔2分钟执行一次，一个业务如果在2分钟内多次被点赞，那就会多次向List中添加同一个业务及对应的点赞总数，数据库也要持久化多次。这显然是多余的，因为只有最后一次才是有效的。而使用ZSET则因为member的唯一性，多次添加会覆盖旧的点赞数量，最终也只会持久化一次。

（面试官可能说：“那就改为SET结构，SET中只放业务id，业务方收到MQ通知后再次查询不就行了。” 如果没问就自己往下说）

当然要解决这个问题，也可以用SET结构代替List，然后当业务被点赞时，只存业务id到SET并通知业务方。业务方接收到MQ通知后，根据id再次查询点赞总数从而避免多次更新的问题。但是这种做法会导致多次网络通信，增加系统网络负担。而ZSET则可以同时保存业务id及最新点赞数量，避免多次网络查询。

不过，并不是说ZSET方案就是完全没问题的，**毕竟ZSET底层是哈希结构+跳表**，对内存会有额外的占用。但是考虑到我们的定时任务每次会查询并删除ZSET数据，ZSET中的数据量始终会维持在一个较低级别，内存占用也是可以接受的。

注意：加黑的地方一定要说，彰显你对Redis底层数据结构和算法有深入了解。