Quiz后端-1、spring项目初始化与数据库设置

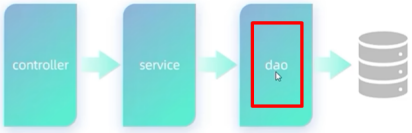
日期：2025.07.19 作者：tfzhang

后端主要提供用户与题目的注册、查询、删除等操作。使用到的开发工具：

- IDEA2024; 关注公众号"青椒工具"，发送"IDEA"，获取windows下的IDEA安装包
- mysql 5.7；关注公众号"青椒工具"，发送"mysql"，获取windows下的mysql5.7安装包；

后端采用java实现，项目采用spring初始化，并进行数据库设计。数据库采用mysql，涉及到后端技术栈：

- java
- spring
- springmvc
- mybatis：一款优秀的持久层框架，用于简化JDBC的开发；



- springboot

这些工具都可以在spring初始化时安装。

本项目使用mysql数据库，所以我们要先安装mysql数据库，其安装过程及资源可参考如下的视频：

```
https://www.bilibili.com/video/BV1Ra4y1y7Hs/?
spm_id_from=333.999.0.0&vd_source=b0e6d0da66db457c6afda440766d8139
```

为了让Maven管理器，下载包更快，可以将当前系统中的maven包源地址设置为阿里云镜像：

```
<mirrors>
    <mirror>
        <id>aliyun</id>
        <name>Aliyun Maven Mirror</name>
        <url>https://maven.aliyun.com/repository/public</url>
        <mirrorOf>central</mirrorOf>
    </mirror>
</mirrors>
```

具体地，在Windows: `C:\Users\你的用户名\.m2\settings.xml` 文件中，输入上述内容。

关于Maven的详情，可以参考：Day04-01. maven-课程介绍*哔哩哔哩*bilibili

1、spring初始化

直接在IDEA开发工具进行初始化，

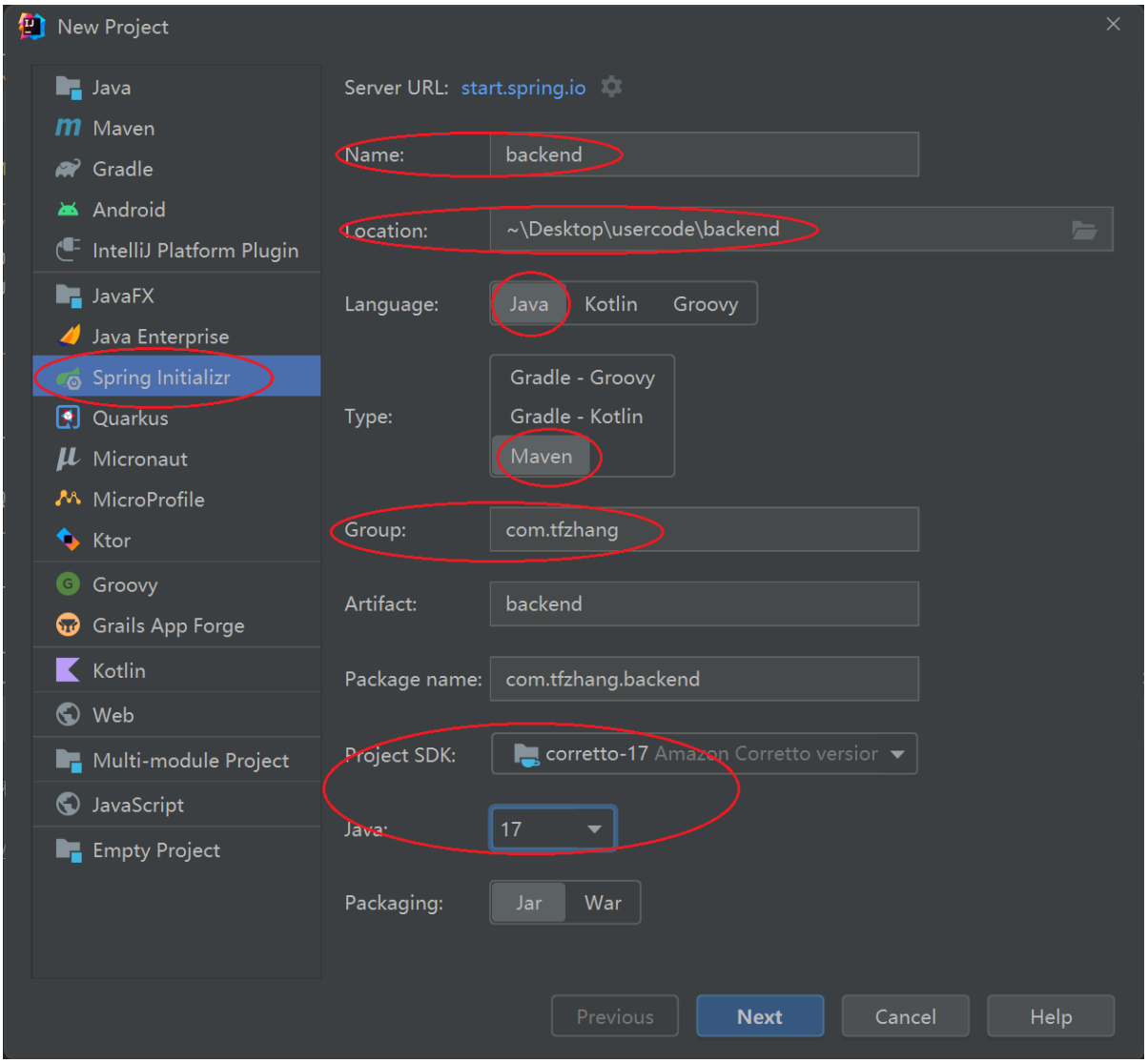


图1 spring项目初始化

此处值得注意的是Project SDK是选择jdk版本，你选择好后，IDEA会自动帮你下载安装，不需要自己额外到系统中安装java jdk。

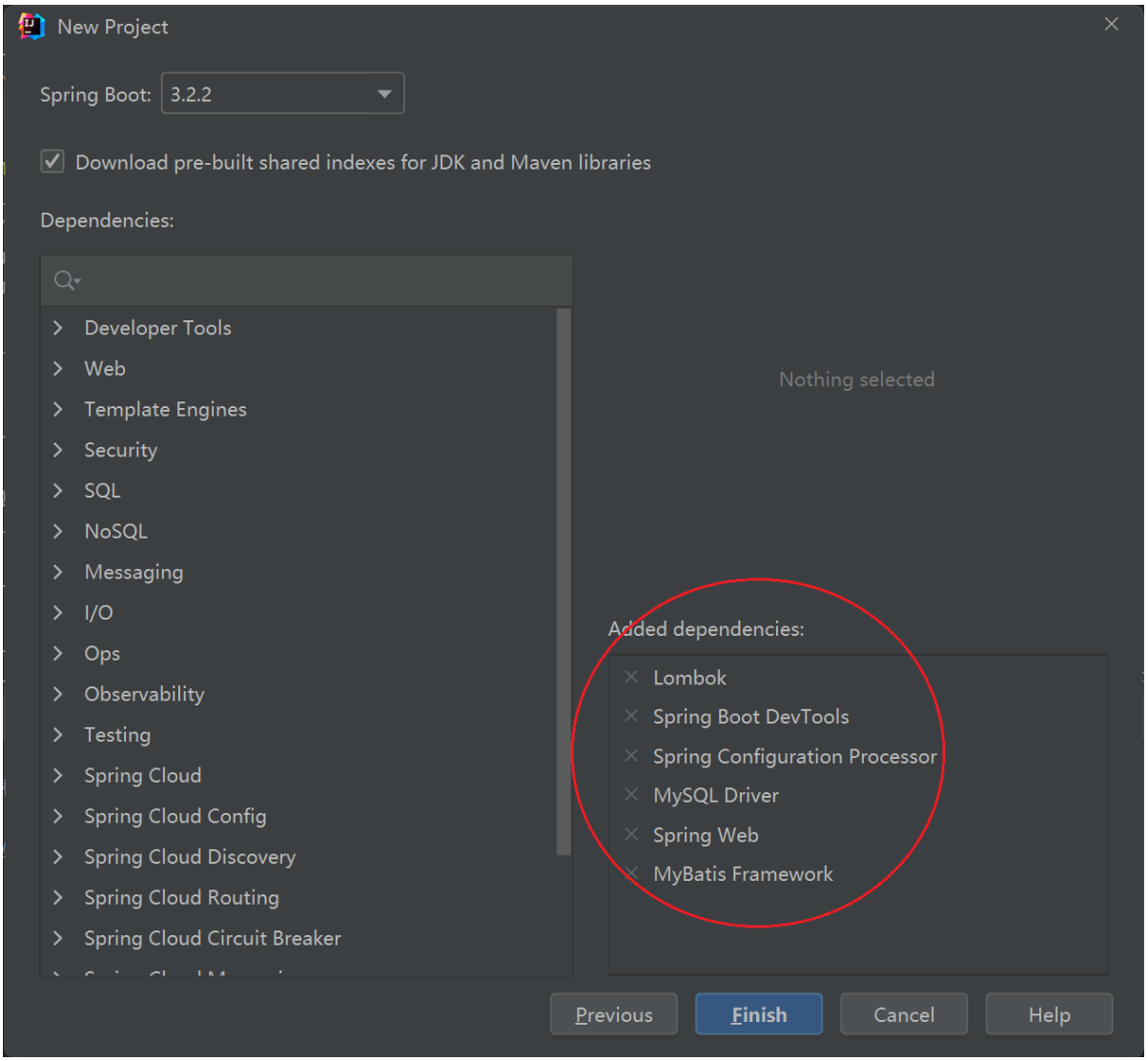


图2 spring boot需要安装的依赖

这些依赖，可以通过左上角的"搜索框"查找，添加。简单介绍下：

- Lombok：自动添加get和set方法；
- Mysql Driver：链接数据库的驱动；
- Spring web: 支持网络访问接口；
- MyBatis Framework：操作数据库；

点击Finish后，IDEA会自动去下载需要的依赖包，需要一定的时间，完成下载后如图3所示。

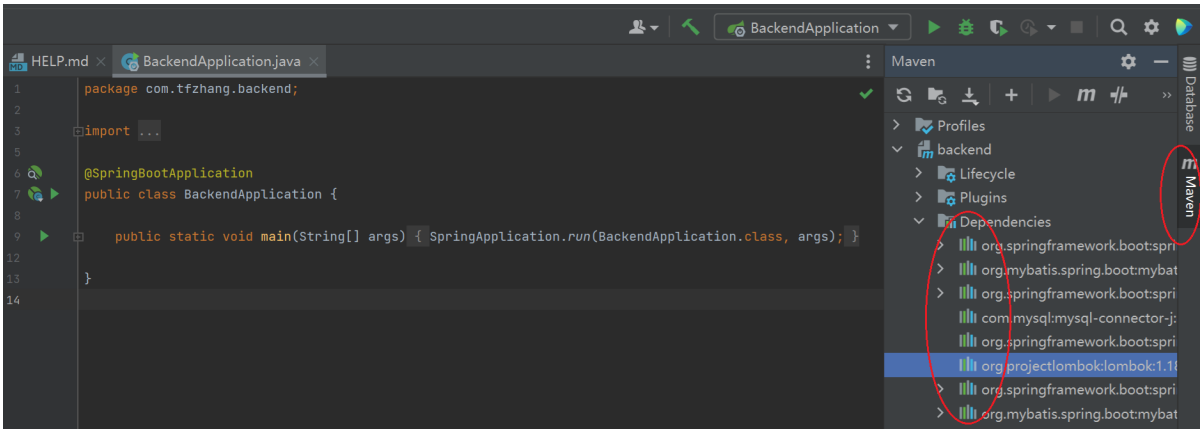


图3 点击右侧的maven展示依赖包

图4是IDEA初始化后的项目目录，简单扼要介绍下：

- src: 主要存放源代码， main是业务代码，test是测试代码；
- pom.xml: 依赖包的配置文件，如果要增加新的依赖包，可以在该文件中配置；

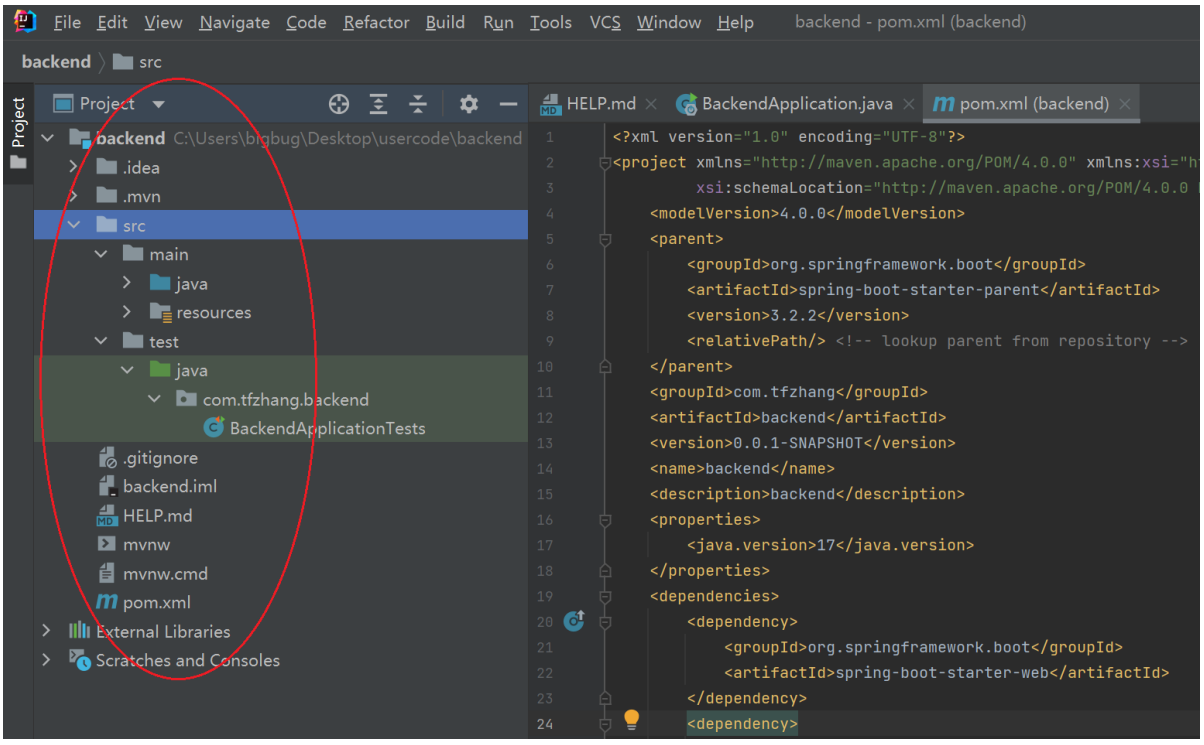


图4 IDEA初始化的项目文件

2、连接数据库及数据库设计

完成数据库安装后，我们使用IDEA连接数据库，并初始化数据库。

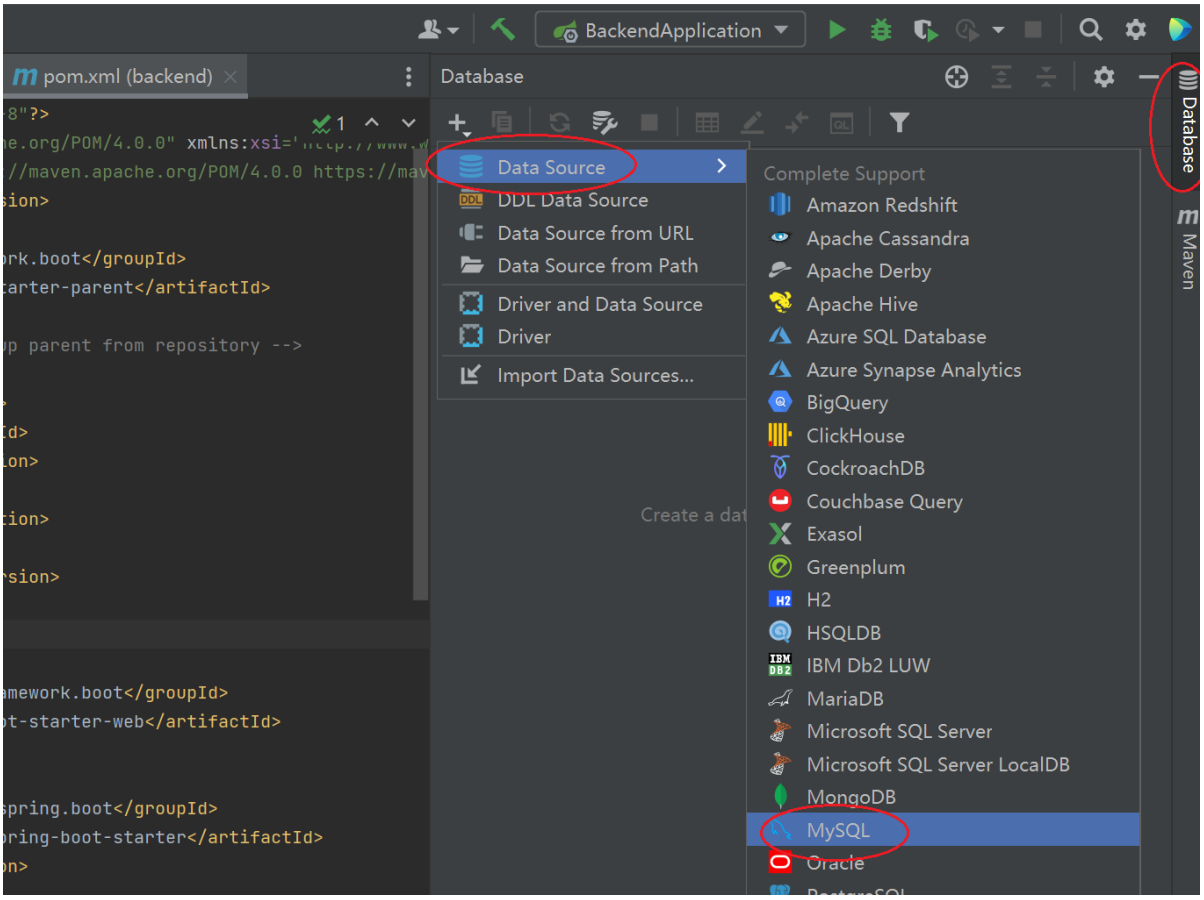


图5 IDEA连接mysql数据库

填写数据库用户和密码，登录数据库，图6所示：

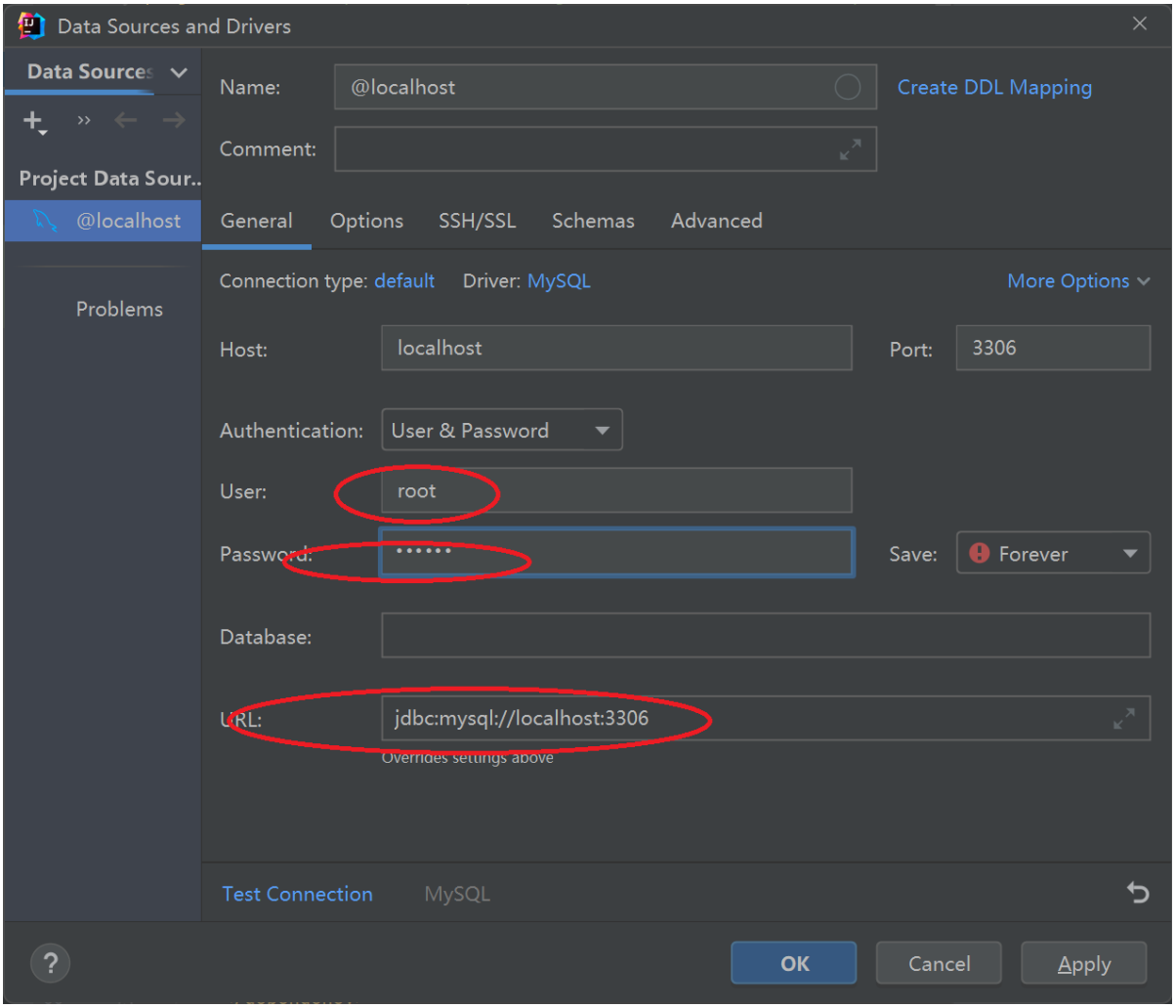


图6 填写数据库账户名和密码

数据库的root用户和密码是你在安装mysql数据库是设定的，另外，由于是本地数据库，所以URL地址默认是：

```
jdbc:mysql://localhost:3306
```

鼠标选中mysql的localhost，new->schema，创建名为quiz的数据库：

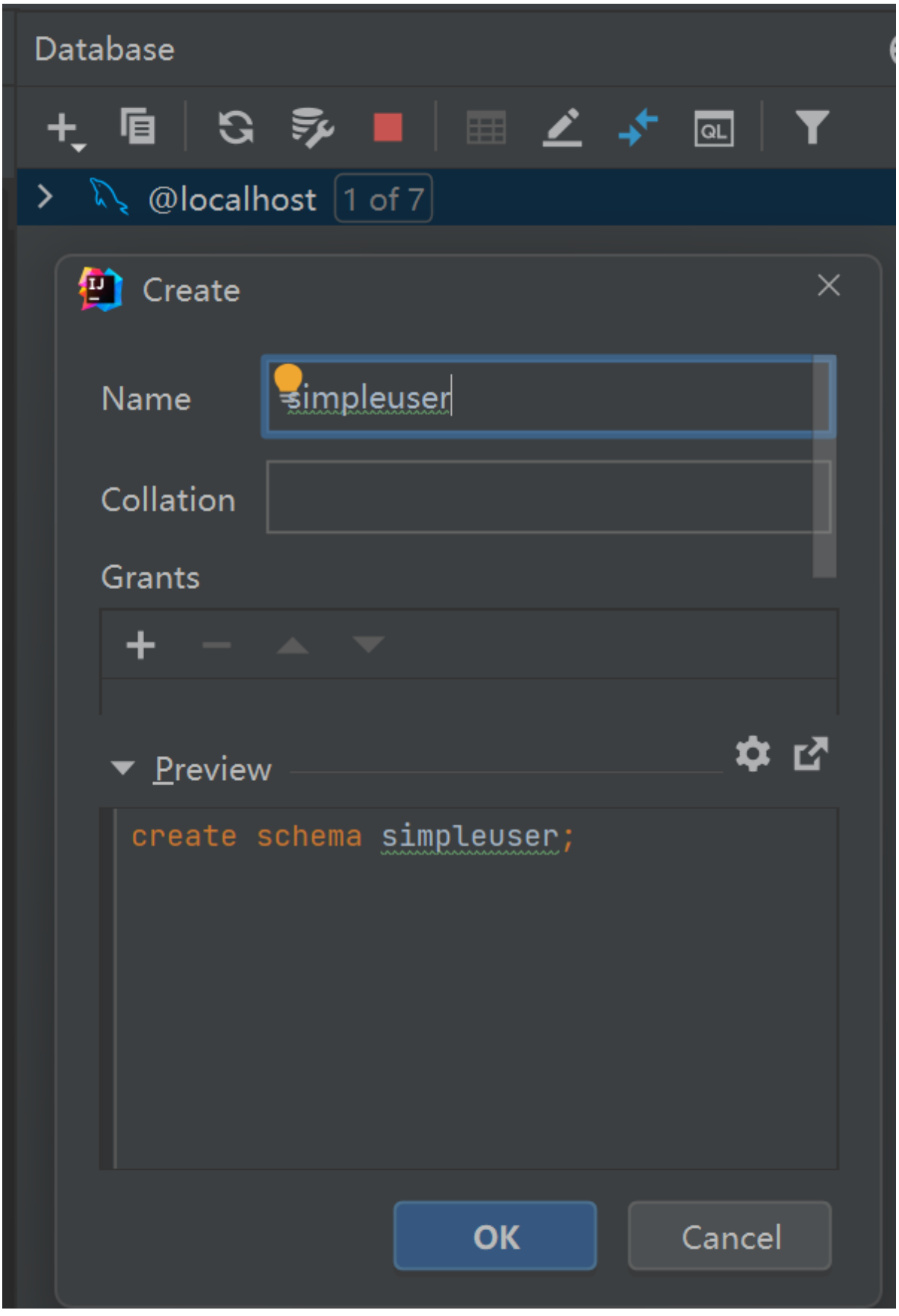


图7 创建数据库

3、MyBatis操作数据库：

步骤1：创建示例数据库

快速地创建一个mytest数据库，然后参照文档创建user表，并插入数据；

```
DROP TABLE IF EXISTS `user`;
```

```sql
CREATE TABLE `user`
(
    id BIGINT NOT NULL COMMENT '主键ID',
    name VARCHAR(30) NULL DEFAULT NULL COMMENT '姓名',
    age INT NULL DEFAULT NULL COMMENT '年龄',
    email VARCHAR(50) NULL DEFAULT NULL COMMENT '邮箱',
    PRIMARY KEY (id)
);

DELETE FROM `user`;

INSERT INTO `user` (id, name, age, email) VALUES
(1, 'Jone', 18, 'test1@baomidou.com'),
(2, 'Jack', 20, 'test2@baomidou.com'),
(3, 'Tom', 28, 'test3@baomidou.com'),
(4, 'Sandy', 21, 'test4@baomidou.com'),
(5, 'Billie', 24, 'test5@baomidou.com');
```

步骤2：创建数据库对应的实体类

在com.tfzhang.quiz目录下创建model目录，其中添加User.java文件，并输入文档中的内容：

```java
package com.tfzhang.quiz.model;

import lombok.Data;

@Data
public class User {
    private Long id;
    private String name;
    private Integer age;
    private String email;
}
```

步骤3：创建MyBatis配置文件

将项目中的application.properites配置修改为application.yml，即只改后缀，主要原因是yml后缀的配置文件书写更灵活；参照mybatis文档，书写配置文件：

```yaml
spring:
  application:
    name:quiz
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/mytest
    username: root
    password: 123456
server:
  port: 8080
```

注意：数据库采用我们为本次测试创建的mytest据库；

步骤4：编写SQL语句

在com.tfzhang.quiz目录下创建mapper目录，创建UserMapper这个接口；

```
package com.tfzhang.quiz.mapper;

import com.tfzhang.quiz.model.User;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import java.util.List;

@Mapper //运行时，框架会自动实现实现类对象，并将对象实例交由IoC容器管理；
public interface UserMapper{

    //查询全部用户；
    @Select("select * from user")
    public List<User> list();
}
```

需要注意的问题：

- 这里我们只需要定义interface，不需要写实现类；因为使用了Mapper标签；
- list()方法与sql语句的“select * from user"绑定；通过@Select标签；

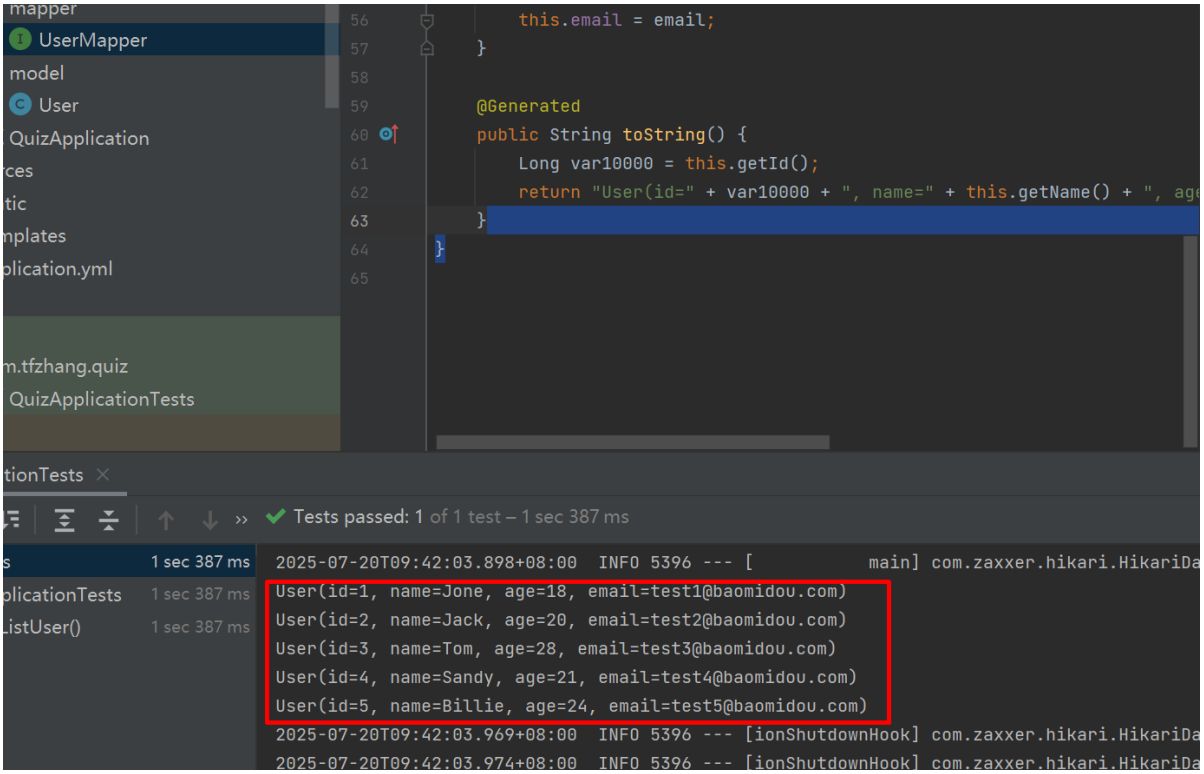
步骤5：编写测试类

将我们的测试list()的代码写在test.com.tfzhang.quiz.QuizApplicationTests.java中；

标签@Autowired，使用的是依赖注入的概念，具体参见：Day05-10. 分层解耦-分层解耦(IOC-DI引入)*哔哩哔哩*bilibili

```
package com.tfzhang.quiz;

import com.tfzhang.quiz.mapper.UserMapper;
import com.tfzhang.quiz.model.User;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

@SpringBootTest
class QuizApplicationTests {

    @Autowired  //依赖注入；
    private UserMapper userMapper;

    @Test
    public void testListUser(){
        List<User> userList = userMapper.list();
        userList.forEach(user->{
            System.out.println(user);
        });
```

```
    }
}
```



运行测试类后，可以在终端看到5个数据的输出。至此，MyBatis的测试成功。

Quiz后端-2、测试Spring contrller接收网络请求

日期：2025.07.19 作者：tfzhang

后端主要提供用户与题目的注册、查询、删除等操作。使用到的开发工具：

- IDEA2024; 关注公众号"青椒工具"，发送"IDEA"，获取windows下的IDEA安装包
- mysql 5.7；关注公众号"青椒工具"，发送"mysql"，获取windows下的mysql5.7安装包；

需求1：无参数请求

```
1、接收浏览器发起的/hello请求，给浏览器返回字符串"hello world"
```

步骤1：创建controller类

在com.tfzhang.quiz目录下创建controller这个目录，然后再创建HelloController这个类：

```
package com.tfzhang.quiz.controller;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController //表示当前类为请求处理类；
public class HelloController {

    @RequestMapping("/hello") //表示如果接收到浏览器的/hello请求，就执行下面的hello()方法；
    public String hello() {
        System.out.println("hello");
        return "Hello World";
    }
}
```

步骤2：启动函数，实现浏览器访问

postman, apifox

需求2：简单参数请求

```
1、接收浏览器发起的/simpleParam?name=tom&age=10请求，服务器获取name和age两个参数并打印到终端，同时返回ok。
```

步骤1：创建类处理方法

com.tfzhang.quiz.controller，在HelloController类中添加对应的类处理方法：

```
    @RequestMapping("/simpleParam") //表示如果接收到浏览器的/simpleParam，就执行下面的getParam()方法；
    public String getParam(String name, Integer age) {
        System.out.println(name+":"+age);
        return "ok";
    }
```

步骤2：使用apifox访问

同时使用GET和POST方法访问。

需求3：实体参数请求

1、来自浏览器发起请求的参数不止两个，如果参数的数量远大于2个，比如10个；
那么在请求处理方法中，就不能再一个个的写变量；最好可以封装到一个类中；那此处我们以需求2为例，实现如何采用实体类来获取参数。

步骤1：在model中创建SimpleUser这个类

```
public class SimpleUser{
    private String name;
    private Integer age;
    //...get,set,tostring等方法;
}
```

步骤2：创建类处理方法simpleUser

```
    @RequestMapping("/simpleUser") //表示如果接收到浏览器的/simpleUser，就执行下面的
getUser()方法;
    public String getUser(SimpleUser user) {
        System.out.println(user);
        return "ok";
    }
```

需求4：统一响应结果

修改上述的需求3的返回结果，让其返回所接收到的对象；

现在发现需求3和需求1返回的数据结果更异，于是出现需求4：

后端返回的数据呈现不同的返回形式，
    现在希望后台返回的数据格式可以一致。

步骤1：创建统一的返回类

在model目录下，创建一个Result返回类别:

```
public Class Result{
    private Integer code; //1是成功，0是失败;
    private String msg; //成功消息或者失败消息;
    private Object data; //放数据;
    //...get,set,toString
    //构造方法;
    //静态success方法和error方法;
}
```

步骤2：重构各个类处理方法

返回的结果设置为Result。

步骤3：测试查看结果

apifox

Quiz后端-3、实现插入新用户接口

日期：2025.07.19 作者：tfzhang

后端主要提供用户与题目的注册、查询、删除等操作。使用到的开发工具：

- IDEA2024; 关注公众号"青椒工具"，发送"IDEA"，获取windows下的IDEA安装包
- mysql 5.7；关注公众号"青椒工具"，发送"mysql"，获取windows下的mysql5.7安装包；

1、前情回顾：

- 1、Java访问数据库：MyBatis框架；
- 2、浏览器访问Java：Controller；

2、实现用户注册与添加：

Quiz的用户数据表设计：

```
DROP TABLE IF EXISTS user;

create table user
(
    id           bigint auto_increment primary key,
    userName     varchar(256)                       null comment '用户名',
    userPassword varchar(512)                       null comment '密码',
    updateTime   datetime default CURRENT_TIMESTAMP null,
    createTime   datetime default CURRENT_TIMESTAMP null,
    isDelete     tinyint                            null,
    userRole     int      default 0                 null comment '表示用户角色， 0
普通用户， 1 管理员'
)
    comment '用户表';
```

字段解释：

- isDelete是逻辑删除标志位，所有的删除数据不真正将数据从数据库删除，而是置位逻辑符；
- userRole是用户角色，用于区别管理员与普通用户；

步骤1：创建数据库

步骤2：创建数据库对应的实体类

对应上述的user数据库表，如下是对应的user实体类：

```
@Data
public class User {
    /** 用户ID */
    private Long id;

    /** 用户名 */
```

```
    private String userName;

    /* 密码*/
    private String userPassword;

    /* 是否删除 */
    private Integer isDelete;

    /* 用户角色：0-普通用户，1-管理员 */
    private Integer userRole;

    /* 创建时间 */
    private Date createTime;

    /*更新时间 */
    private Date updateTime;
}
```

步骤3：创建MyBatis配置文件

步骤4：编写添加用户的mapper

UserMapper类原来的代码：

```
package com.tfzhang.quiz.mapper;

import com.tfzhang.quiz.model.User;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;

import java.util.List;

@Mapper //运行时，框架会自动实现实现类对象，并将对象实例交由IoC容器管理；
public interface UserMapper{

    //查询全部用户；
    @Select("select * from user")
    public List<User> list();
}
```

现在要新增用户，对应的sql语句如下：

```
insert into user(userName, userPassword,isDelete,userRole,createTime,updateTime)
    values('tom', '123456', 0, 0, now(), now(),);
```

对应的Java代码：

```
    @Insert("insert into user(userName,
userPassword,isDelete,userRole,createTime, updateTime)"+
            "values(#{userName}, #{userPassword}, #{isDelete}, #{userRole}, #
{createTime}, #{updateTime})")
    @Options(useGeneratedKeys = true, keyProperty = "id")
    public int saveUser(User user);
```

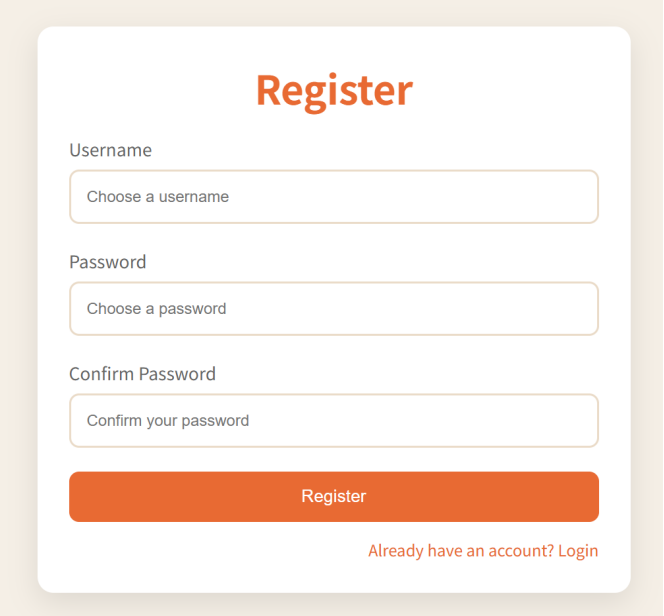
步骤5：对添加用户的mapper进行测试

同样地，在test.java.com.tfzhang.quiz.QuizApplicationTests中添加测试代码：

```
package com.tfzhang.quiz;

import com.tfzhang.quiz.mapper.UserMapper;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import com.tfzhang.quiz.model.User;

import java.util.Date;
import java.util.List;

@SpringBootTest
class QuizApplicationTests {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void testSaveUser(){
        User user = new User();
        user.setUserName("testuser");
        user.setUserPassword("password123");

        user.setUserRole(0);
        user.setIsDelete(0);

        Date now = new Date();
        user.setCreateTime(now);
        user.setUpdateTime(now);

        // 执行插入操作,插入后返回插入成功的行数，这里默认就应该是1；
        int result = -1;
        result = userMapper.saveUser(user);
        System.out.println(result);
        //生成的id会默认回填到user对象中；
        System.out.println(user.getId());
        return result;
    }
}
```

可以根据result的返回值，或者user.getId()的值来判断是否插入数据成功。

步骤6：编写添加用户的Controller



根据上图Quiz用户前端的注册新用户页面，添加新用户时，浏览器端会发送3个参数：

- username
- password
- confirmpassword

在controller目录下新建：UserController.java

```
@RestController
public class UserController {

    @RequestMapping("/register")
    public Result addUser(String username, String password, String checkpassword)
{
        //代码逻辑步骤；
        //1、用户输入的账户和密码不能为空；
        //2、校验用户的账户、密码是否符合要求：
        //   - 账户字符不能少于4个；
        //   - 密码不能小于8位；
        //   - 密码和确认密码要一致；
        //   - 账户不能与已有的重复；
        //   - 账户不能包含特殊字符；
        //   - 密码和校验密码相同；
        //3、对密码进行加密；保证后端工作人员不能看到用户密码；
        //4、向数据库插入数据；
    }
}
```

上述的很多检查可以在前端页面进行检查，但你不能保证所有的用户都是规矩的用户，所以有些检查，我们还是要在后端实现：

```
        //代码逻辑步骤：
        //1、用户输入的账户和密码不能为空；
        //2、校验用户的账户、密码是否符合要求：
           //    - 账户不能包含特殊字符；
           //    - 密码和确认密码要一致；
           //    - 账户不能与已有的重复；
        //3、对密码进行加密；保证后端工作人员不能看到用户密码；密码不要用明文；
        //4、向数据库插入数据；
```

1、用户输入的账户和密码不能为空；

```
  if(StringUtils.isAnyBlank(username, password, checkpassword)){
            return Result.error("用户名或密码为空");
  }
```

此处的 StringUtils库来自common lang库；访问maven官网，添加该库的配置文件到pom.xml

```
https://mvnrepository.com/
```

2、判断password和checkpassword两者是否一致；

```
    if (!password.equals(checkpassword)) {
        return Result.error("两次输入的密码不一致");
    }
```

3、校验用户的账户、密码是否符合要求：

```
        String regex = "^[a-zA-Z0-9]+$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(username);
        if(!matcher.matches()){
            return Result.error("用户名包含特殊字符");
        }
```

4、查询数据库，查看用户名username是否已经存在：

```
//涉及到数据库查询；暂时不实现；
//to do...
```

5、对密码进行加密；保证后端开发人员不能看到用户密码；

```
final String SALT = "com.quiz";
String encrptedPassword =
DigestUtils.md5DigestAsHex((SALT+userpassword).getBytes());
```

6、创建新用户，并且向数据库插入新数据；

```
    public Result addUser(String username, String password, String checkpassword)
{
        //此处的逻辑代码；
        if (StringUtils.isAnyBlank(username, password, checkpassword)) {
```

```
            return Result.error("用户名或密码为空");
        }

        if (!password.equals(checkpassword)) {
            return Result.error("两次输入的密码不一致");
        }

        String regex = "^[a-zA-Z0-9]+$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(username);
        if (!matcher.matches()) {
            return Result.error("用户名包含特殊字符");
        }

        //查询数据库，确定是否已经存在用户名;
        //to add...

        //对密码进行加密;
        final String SALT = "com.quiz";
        String encrptedPassword = DigestUtils.md5DigestAsHex((SALT +
password).getBytes());

        User user = new User();
        user.setUserName(username);
        user.setUserPassword(encrptedPassword);
        /**
         * 注册默认是普通用户，所以userRole设置为0;
         */
        user.setUserRole(0);
        user.setIsDelete(0);

        Date now = new Date();
        user.setCreateTime(now);
        user.setUpdateTime(now);

        //4.插入到数据库;
        int result = userMapper.saveUser(user);

        if (result > 0)
            return Result.success("新增用户成功");
        else
            return Result.error("注册用户失败");

    }
```

3、三层架构的改造

参考资料：Day05-09. 分层解耦-三层架构*哔哩哔哩*bilibili

根据三层架构，当前的controller的问题：

- 业务逻辑混杂，没有独立出来；

步骤1：创建Service目录及文件

创建Service目录，集中放置业务逻辑代码；Service采用接口与实现分离的方式来实现；

```java
//UserService.java:
public interface UserService{

    //保存新用户；
    public Result addUser(String username, String password, String
checkpassword);
}
```

```java
//UserServiceImpl.java

@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserMapper userMapper;

    public Result addUser(String username, String password, String checkpassword)
{
        //此处的逻辑代码；
        if (StringUtils.isAnyBlank(username, password, checkpassword)) {
            return Result.error("用户名或密码为空");
        }

        if (!password.equals(checkpassword)) {
            return Result.error("两次输入的密码不一致");
        }

        String regex = "^[a-zA-Z0-9]+$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(username);
        if (!matcher.matches()) {
            return Result.error("用户名包含特殊字符");
        }

        //查询数据库，确定是否已经存在用户名；
        //to add...

        //对密码进行加密；
        final String SALT = "com.quiz";
        String encrptedPassword = DigestUtils.md5DigestAsHex((SALT +
password).getBytes());

        User user = new User();
        user.setUserName(username);
        user.setUserPassword(encrptedPassword);
        /**
         * 注册默认是普通用户，所以userRole设置为0；
         */
        user.setUserRole(0);
        user.setIsDelete(0);

        Date now = new Date();
        user.setCreateTime(now);
        user.setUpdateTime(now);
```

```
        //4.插入到数据库;
        int result = userMapper.saveUser(user);

        if (result > 0)
            return Result.success("新增用户成功");
        else
            return Result.error("注册用户失败");
    }
}
```

步骤2：改造其他代码：

比如删减Controller组件中的代码；

4、作业：

完成Service中如下的代码：

```
        String regex = "^[a-zA-Z0-9]+$";
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(username);
        if (!matcher.matches()) {
            return Result.error("用户名包含特殊字符");
        }

        //查询数据库，确定是否已经存在用户名;
        //to add...

        //对密码进行加密;
        final String SALT = "com.quiz";
        String encrptedPassword = DigestUtils.md5DigestAsHex((SALT +
password).getBytes());
```

要求：

- 完成MyBatis端的以用户名查询数据库；
- 再完成UserServiceImpl中，to add处的代码；

Quiz后端-4、实现删除用户接口

日期：2025.07.19 作者：tfzhang

后端主要提供用户与题目的注册、查询、删除等操作。使用到的开发工具：

- IDEA2024; 关注公众号"青椒工具"，发送"IDEA"，获取windows下的IDEA安装包
- mysql 5.7；关注公众号"青椒工具"，发送"mysql"，获取windows下的mysql5.7安装包；

1、前情回顾：

- 1、Java访问数据库：MyBatis框架；
- 2、浏览器访问Java：Controller；
- 3、实现插入新用户接口；

2、上次视频遗留的查询用户；

步骤1：查询用户的UserMapper端代码

```
@Select("SELECT COUNT(*) FROM user WHERE useName = #{username} AND isDelete = 0")
public int existsByName(@Param("username") String username);
```

按照用户名查询，用户是否存在，如果用户存在，则返回1；否则返回0；

步骤2：测试上述的方法

步骤3：在UserService的实现类中，根据上述代码，进行完善；

3、实现根据用户Id或者用户名，删除用户：

步骤1：在usermapper中添加方法

在UserMapper中添加如下的代码：

```
    //根据id删除用户；
    @Update("UPDATE user SET isDelete = 1, updateTime = NOW() WHERE id = #{id}
AND isDelete = 0")
    public int deleteUserById(@Param("id") Long id);

    //根据username删除用户；
    @Update("UPDATE user SET isDelete = 1, updateTime = NOW() WHERE userName = #
{username} AND isDelete = 0")
    public int deleteByUsername(@Param("username") String username);
```

步骤2：测试usermapper方法

测试上述的UserMapper中的两个删除方法；

步骤3：在UserService中添加下面的接口和实现

```
    //接口：
```

```
    public boolean deleteUserById(Long id);
    public boolean deleteUserByName(String username);

    //实现：
    public boolean deleteUserById(Long id) {
        int result = userMapper.deleteUserById(id);
        return result > 0;
    }

    public boolean deleteByName(String username) {
        int result = userMapper.deleteUserByUsername(username);
        return result > 0;
    }
```

步骤4：controller中添加代码

```
        @GetMapping ("/deleteById")
        public Result deleteUserById(Long id) {
            boolean success = userService.deleteUserById(id);
            if (success) {
                return Result.success("用户已删除");
            }
            return Result.error("用户不存在或已被删除");
        }

        @GetMapping("/deleteByName")
        public Result deleteUser(String username) {
            boolean success = userService.deleteUserByName(username);
            if (success) {
                return Result.success("用户已删除");
            }
            return Result.error("用户不存在或已被删除");
        }
```

步骤5：apifox测试上述的delete接口

作业：书写hard delete的接口

Quiz后端-4、实现查询展示用户

日期：2025.07.19 作者：tfzhang

后端主要提供用户与题目的注册、查询、删除等操作。使用到的开发工具：

- IDEA2024; 关注公众号"青椒工具"，发送"IDEA"，获取windows下的IDEA安装包
- mysql 5.7；关注公众号"青椒工具"，发送"mysql"，获取windows下的mysql5.7安装包；

0、前情回顾：

- 1、Java访问数据库：MyBatis框架；
- 2、浏览器访问Java：Controller；
- 3、实现插入新用户接口；
- 4、实现删除用户；

1、查询展示用户的两个接口：

- 按照分页的方式展示用户；
- 按照用户名查询；

2、按照分页查询用户；

预备知识：mysql分页查询基础

a、在mysql中采用select后添加limit参数来实现分页查询：

```
select * from user limit 0, 5;
//limit后两个参数的含义0表示起始数据，5表示返回5条数据；
```

b、在mysql中要查询总的记录数：

```
select count(*) from user;
```

对比上述的前端查询页，在执行分页查询时，我们需要获取总的记录数和当前页的数据；

当前页的起始点计算：(当前页码-1)*每页条目数；

c、前端要提供给后端的数据：当前页码，每页展示数；

后端要提供给前端的数据：当前页码的数据list，以及总记录数；

参考资料：Day10-06. 案例-员工管理-分页查询-分析*哔哩哔哩*bilibili

步骤1：在model新建PageBean对象用于controller返回数据

```
@Data
public class PageBean{
    private int total;
    private List<User> rows;
}
```

步骤2：usermapper中添加代码

```
@Select("SELECT COUNT(*) FROM user WHERE isDelete=0")
public int count();

@Select("SELECT * FROM user WHERE isDelete=0 limit #{start},#{pageSize}")
public List<User> page(Integer start, Integer pageSize);
```

步骤3：Controller中添加代码

```
@GetMapping("/users")
public Result getPage(@RequestParam(defaultValue="1")Integer page,
@RequestParam(defaultValue="5")Integer pageSize){
    PageBean pageBean=userService.page(page, pageSize);
    return Result.success(pageBean);
}
```

步骤4：Service层中添加代码

```
//接口：
PageBean page(Integer page, Integer pageSize);

//实现：
public PageBean page(Integer page, Integer pageSize){
    //获取总的记录数；
    Integer total=userMapper.count();

    //获取分页查询结果列表；
    Integer start = (page-1)*pageSize;
    List<User> userList=userMapper.page(start, pageSize);

    //封装PageBean对象；
    PageBean pageBean = new PageBean();
    pageBean.setTotal(total);
    pageBean.setRows(userList);

    return pageBean;
```

```
}
```

步骤5：分页查询测试；

如果出现错误，可以打开MyBatis的日志，输出错误信息到控制台：

Day09-03. Mybatis-基础操作-删除(预编译SQL)_哔哩哔哩_bilibili

3、按照用户名查询

数据库预备知识：

要在用户名中查询某个特定的字符串，可以使用如下 的sql模糊查询语句：

```
SELECT * FROM user WHERE username like '%abc%';
```

对应的MyBatis的mapper层的代码：

```
@Mapper
public interface UserMapper {
    @Select("SELECT * FROM user WHERE username LIKE CONCAT('%', #{keyword}, '%')
AND isDelete=0")
    List<User> findByName(String keyword);
}
```

Controller中的代码：

```
@GetMapping("/findUser")
public Result getUser(String keyword){
    List<User> users=userService.findByName(keyword);
    return Result.success(users);
}
```

Service层中的代码：

```
//接口；
public List<User> findByName(String keyword);

//实现：
public List<User> findByName(String keyword){
        List<User> userList=userMapper.findByName(keyword);
        for(User user:userList){
            user.setUserPassword("*******");
        }
        return userList;
}
```

Quiz后端-6、实现题目页的添加、删除与查询

日期：2025.07.24 作者：tfzhang

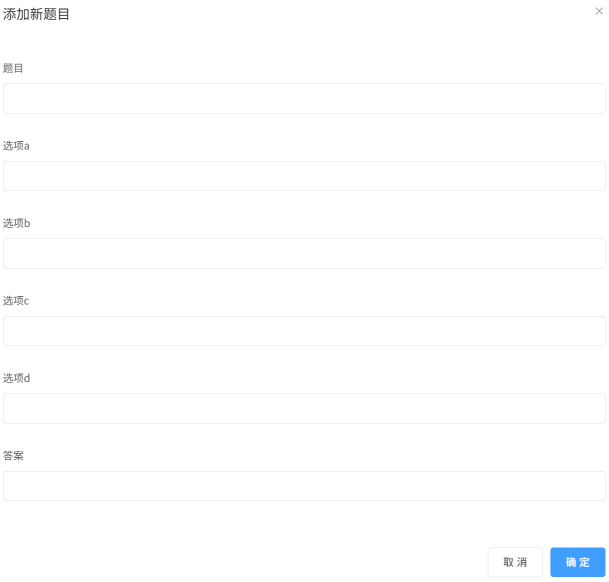
后端主要提供用户与题目的注册、查询、删除等操作。使用到的开发工具：

- IDEA2024; 关注公众号"青椒工具"，发送"IDEA"，获取windows下的IDEA安装包
- mysql 5.7；关注公众号"青椒工具"，发送"mysql"，获取windows下的mysql5.7安装包；

前情回顾：

- 用户的插入，删除，查询；

添加题目的前端页面：



1、题目的数据库表设计；

```sql
DROP TABLE IF EXISTS questions;

CREATE TABLE questions (
    id INT PRIMARY KEY AUTO_INCREMENT,
    questionText VARCHAR(255) NOT NULL,
    answer1Text VARCHAR(255) NOT NULL,
    answer1Correct BOOLEAN NOT NULL DEFAULT FALSE,
    answer2Text VARCHAR(255) NOT NULL,
    answer2Correct BOOLEAN NOT NULL DEFAULT FALSE,
    answer3Text VARCHAR(255) NOT NULL,
    answer3Correct BOOLEAN NOT NULL DEFAULT FALSE,
    answer4Text VARCHAR(255) NOT NULL,
    answer4Correct BOOLEAN NOT NULL DEFAULT FALSE,
    isDelete    tinyint  null,
    createTime   datetime default CURRENT_TIMESTAMP null,
    updateTime   datetime default CURRENT_TIMESTAMP null
);
```

此处我们的题目选项数固定为4，所以可以写死answer的数量。

如果题目的选项数不固定呢?

步骤1：

在quiz数据库中创建上述的questions数据库表格；

步骤2：

在model目录中，创建实体类：

```
public class Question {
    private Integer id;
    private String questionText;

    private String answer1Text;
    private Boolean answer1Correct;

    private String answer2Text;
    private Boolean answer2Correct;

    private String answer3Text;
    private Boolean answer3Correct;

    private String answer4Text;
    private Boolean answer4Correct;

        /* 是否删除 */
    private Integer isDelete;

    /* 创建时间 */
    private Date createTime;

    /*更新时间 */
    private Date updateTime;
    // Getters and Setters
}
```

2、题目的添加；

步骤1：创建QuestionMapper类

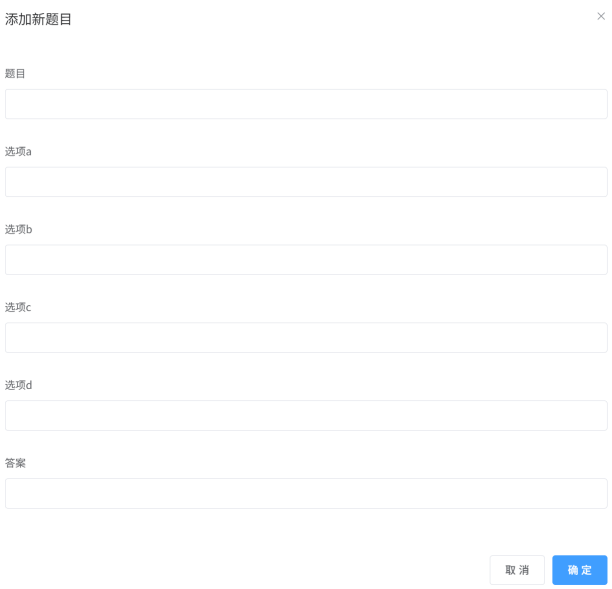
```
@Mapper
public interface QuestionMapper {
    @Insert("INSERT INTO questions (question_text,answer1_text,
answer1_correct,answer2_text, answer2_correct, answer3_text,
answer3_correct,answer4_text, answer4_correct, isDelete, createTime, updateTime)"
+
            "VALUES (#{questionText},#{answer1Text}, #{answer1Correct}, #
{answer2Text}, #{answer2Correct},#{answer3Text}, #{answer3Correct},#
{answer4Text}, #{answer4Correct}, #{isDelete},#{createTime},#{updateTime})")
    @Options(useGeneratedKeys = true, keyProperty = "id")
    int insertQuestion(Question question);
}
```

步骤2：测试QuestionMapper类

```
    @Autowired
    private QuestionMapper questionMapper;

    @Test
    public void testInsertQuestion(){
        Question question = new Question();
        question.setQuestionText("What is the capital of France?");

        question.setAnswer1Text("Paris");
        question.setAnswer1Correct(true);

        question.setAnswer2Text("London");
        question.setAnswer2Correct(false);

        question.setAnswer3Text("Berlin");
        question.setAnswer3Correct(false);

        question.setAnswer4Text("Madrid");
        question.setAnswer4Correct(false);

        question.setIsDelete(0);
        question.setCreateTime(new Date());
        question.setUpdateTime(new Date());

        // 执行插入操作
        int result = questionMapper.insertQuestion(question);
        System.out.println(result);
        System.out.println(question.getId());
    }
```

步骤3：Controller类使用到的model

假设来自前端的插入题目的页面：



根据这个前端页面，在model中书写相应的实体类：

```
public class QSBean {
    private String question;
    private String optiona;
    private String optionb;
    private String optionc;
    private String optiond;
    private String answer; // "a", "b", "c", or "d"
}
```

步骤4：在Service层创建QuestionService，并增加接口和实现

```
//接口代码；
public int insertQuestion(QuestionSubmitBean qsBean);

//实现类；
public int insertQuestion(QuestionSubmitBean qsBean){

    String ans = qsBean.getAnswer();
    if (!List.of("a", "b", "c", "d").contains(ans.toLowerCase())) {
        throw new IllegalArgumentException("Answer must be one of: a, b, c, or
d");
    }

    Question q = new Question();
    q.setQuestionText(qsBean.getQuestion());

    q.setAnswer1Text(qsBean.getOptiona());
    q.setAnswer1Correct("a".equalsIgnoreCase(ans));

    q.setAnswer2Text(qsBean.getOptionb());
    q.setAnswer2Correct("b".equalsIgnoreCase(ans));

    q.setAnswer3Text(qsBean.getOptionc());
    q.setAnswer3Correct("c".equalsIgnoreCase(ans));

    q.setAnswer4Text(qsBean.getOptiond());
```

```
    q.setAnswer4Correct("d".equalsIgnoreCase(ans));

    q.setIsDelete(0);
    q.setCreateTime(new Date());
    q.setUpdateTime(new Date());

    int result = questionMapper.insertQuestion(q);
    return result;
}
```

步骤5：创建QuestionControler，并添加代码：

```
    @Autowired
    private QuestionService questionService;

    @PostMapping("/addQuestion")
    public Result addQuestion(QSBean question) {
        int result = questionService.insertQuestion(question);
        if (result > 0){
            return Result.success("插入新问题成功");
        }else{
            return Result.error("插入失败");
        }
    }
```

步骤6：apifox测试

测试数据：

```
question: which of the following is not a programming language?
optiona: Java
optionb: Apple
optionc: Python
optiond: Javascript
answer: b
```

3、题目的删除；

- 根据id删除用户；
- 主要采用逻辑删除的方式；

参考删除用户的代码来实现。

4、题目的查询；

- 与用户的查询类似，提供：分页查询与按关键词查询两种方式；

参考用户查询的代码来实现。

Quiz后端-7、实现题目页的删除与查询

日期：2025.07.24 作者：tfzhang

后端主要提供用户与题目的注册、查询、删除等操作。使用到的开发工具：

- IDEA2024; 关注公众号"青椒工具"，发送"IDEA"，获取windows下的IDEA安装包
- mysql 5.7；关注公众号"青椒工具"，发送"mysql"，获取windows下的mysql5.7安装包；

1、题目的删除；

- 根据id删除题目；
- 主要采用逻辑删除的方式；

参考删除用户的代码来实现。

步骤1：questionmapper中添加删除方法

在QuestionMapper中添加如下的代码：

```
    //根据id删除题目；
    @Update("UPDATE questions SET isDelete = 1 WHERE id = #{id} AND isDelete =
0")
    public int delQuestionById(Integer id);
```

步骤2：测试QuestionMapper方法

测试上述的删除方法。

步骤3：在QuestionService中添加接口及方法

```
//接口；
public boolean delQuestion(Integer id);

//实现；
public boolean delQuestion(Integer id){
    int result = questionMapper.delQuestionById(id);
    return result>0;
}
```

步骤4：controller中添加代码

```
    @GetMapping ("/delQuestion")
    public Result deleteQuestion(Integer id) {
        boolean success = questionService.delQuestion(id);
        if (success) {
            return Result.success("用户成功删除");
        }
        return Result.error("用户不存在或已被删除");
    }
```

步骤5：在apifox中测试

2、题目的查询；

题目的查询来自用户端和管理端，为了作为区分：

用户端的查询称为：获取题目；

管理端的查询称为：查询题目；其中管理端的查询与用户查询类似，分为：

- 分页查询与按关键词查询；

用户端获取题目的实体类：

用户端需要的数据格式：

```
const quizQuestions = [
  {
    question: "What is the capital of France?",
    answers: [
      { text: "London", correct: false },
      { text: "Berlin", correct: false },
      { text: "Paris", correct: true },
      { text: "Madrid", correct: false },
    ],
  },
  {
    question: "Which planet is known as the Red Planet?",
    answers: [
      { text: "Venus", correct: false },
      { text: "Mars", correct: true },
      { text: "Jupiter", correct: false },
      { text: "Saturn", correct: false },
    ],
  }
]
```

根据上述用户端的数据格式，创建实体类QSBeanOut:

```
public class QSBeanOut{
    private String question;
    private List<AnsBean> answers;
    //setter.getter.
}
```

再创建AnsBean类：

```
public class AnsBean{
    private String text;
    private boolean correct;
    //类构造方法;
    public AnsBean(String text, boolean correct) {
        this.text = text;
        this.correct = correct;
    }
    //setter.getter.
}
```

获取题目的QuestionMapper:

用户每次获取题目的数量是固定的，就是5个题目;

对应的sql语句如下:

```
SELECT * FROM questions ORDER BY RAND() LIMIT 5;
```

对应的代码:

```
@Mapper
public interface QuestionMapper {
    @Select("SELECT * FROM questions where isDelete=0 ORDER BY RAND() LIMIT 5")
    public List<Question> getQuestions();
}
```

测试getQuestions方法

```
    @Test
    public void testQuestionList(){
        List<Question> qlist = questionMapper.getQuestions();
        for (Question q : qlist) {
            System.out.println(q);
        }
    }
```

Controller层创建方法:

```
    @GetMapping("/getQuestion")
    public Result getQuestion() {
        List <QSBeanOut> qsBeanOut = questionService.getQuestions();
        return Result.success(qsBeanOut);
    }
```

此处返回的对象是QSBeanOut的List，但是在mapper层返回的是Question类型的list，所以此处我们需要作一个转换，我们在创建utils目录，并在其中撰写转换的工具方法;

```
public static List<QSBeanOut> convertToQSBeanList(List<Question> questionList) {
    List<QSBeanOut> result = new ArrayList<>();

    for (Question question : questionList) {
        QSBeanOut qsBean = new QSBeanOut();
```

```
        qsBean.setQuestion(question.getQuestionText());

        List<AnsBean> answers = new ArrayList<>();
        answers.add(new AnsBean(question.getAnswer1Text(),
question.getAnswer1Correct()));
        answers.add(new AnsBean(question.getAnswer2Text(),
question.getAnswer2Correct()));
        answers.add(new AnsBean(question.getAnswer3Text(),
question.getAnswer3Correct()));
        answers.add(new AnsBean(question.getAnswer4Text(),
question.getAnswer4Correct()));

        qsBean.setAnswers(answers);
        result.add(qsBean);
    }
    return result;
}
```

Service层的接口与代码

```
//interface:
    //获取5个新问题:
    public List<QSBeanOut> getQuestions();

//impl
    public List<QSBeanOut> getQuestions(){
        //获取question list;
        List<Question> questionList = questionMapper.getQuestions();
        //将question list转化为QSBeanOut;
        List<QSBeanOut> qsBeanOutList = Tools.convertToQSBeanList(questionList);
        return qsBeanOutList;
    }
```

apifox端测试

3、管理端的查询

管理端的题目查询与用户的查询比较一致，因此可以参考用户的查询：

- 分页查询
- 按题目的关键词查询；

首先要观察管理端的前端页面，来确定实体类：

分页和按题目关键词查询的题目都包含三个部分：

- 题目；
- 选项；
- 答案；

对应的数据接口：

```
{
    questionText:"题目内容",
    options:{"a","b","c","d"},
    answer:"b"
}
```

在model目录中，创建相应的类：

```
public class QSBeanPage{
    private int total;
    private List<QSBeanOutManage> qsBeanList;
}

public class QSBeanOutManage {
    private String questionText;
    private List<String> options;
    private String answer;
}
```

步骤2：questionmapper中添加代码

```
@Select("SELECT COUNT(*) FROM questions WHERE isDelete=0")
public int count();

@Select("SELECT * FROM questions WHERE isDelete=0 limit #{start},#{pageSize}")
public List<Question> page(Integer start, Integer pageSize);
```

步骤3：QuestionController中添加代码

```
@GetMapping("/questions")
public Result getPage(@RequestParam(defaultValue="1")Integer page,
@RequestParam(defaultValue="5")Integer pageSize){
    QSBeanPage qsBeanPage=questionService.page(page, pageSize);
    return Result.success(qsBeanPage);
}
```

步骤4：Question Service层中添加代码

```
//接口：
public QSBeanPage page(Integer page, Integer pageSize);

//实现：
public QSBeanPage page(Integer page, Integer pageSize){
    //获取总的记录数；
    Integer total=questionMapper.count();

    //获取分页查询结果列表；
    Integer start = (page-1)*pageSize;
    List<Question> questionList=questionMapper.page(start, pageSize);

    //将questionList转化为QSBeanOutManage
    //to be done;
    QSBeanOutManage qsBeanOutManage =
Tools.convertToQSBeanManageList(questionList);

    //封装PageBean对象；
    QSBeanPage qsBeanPage = new QSBeanPage();
    qsBeanPage.setTotal(total);
    qsBeanPage.setQsBeanList(qsBeanOutManage);

    return qsBeanPage;
}
```

其中的Tools.convertToQSBeanManageList是定义在utils中的工具类：

```
    public static List<QSBeanOutManage> convertToQSBeanManageList(List<Question>
questionList) {
        List<QSBeanOutManage> result = new ArrayList<>();
        if (questionList == null) {
            return result;
        }

        for (Question question : questionList) {
            // 跳过已删除的问题
            if (question.getIsDelete() != null && question.getIsDelete() == 1) {
                continue;
            }

            QSBeanOutManage bean = new QSBeanOutManage();
            bean.setQuestionText(question.getQuestionText());

            // 构建选项列表（固定4个选项）
            List<String> options = new ArrayList<>(4);
```

```
            options.add(question.getAnswer1Text() != null ?
question.getAnswer1Text() : "");
            options.add(question.getAnswer2Text() != null ?
question.getAnswer2Text() : "");
            options.add(question.getAnswer3Text() != null ?
question.getAnswer3Text() : "");
            options.add(question.getAnswer4Text() != null ?
question.getAnswer4Text() : "");
            bean.setOptions(options);

            // 查找正确答案文本
            bean.setAnswer(determineCorrectAnswer(question));
            result.add(bean);
        }

        return result;
    }

    private static String determineCorrectAnswer(Question question) {
        if (question.getAnswer1Correct() != null && question.getAnswer1Correct())
{
            return question.getAnswer1Text() != null ? question.getAnswer1Text()
: "";
        }
        if (question.getAnswer2Correct() != null && question.getAnswer2Correct())
{
            return question.getAnswer2Text() != null ? question.getAnswer2Text()
: "";
        }
        if (question.getAnswer3Correct() != null && question.getAnswer3Correct())
{
            return question.getAnswer3Text() != null ? question.getAnswer3Text()
: "";
        }
        if (question.getAnswer4Correct() != null && question.getAnswer4Correct())
{
            return question.getAnswer4Text() != null ? question.getAnswer4Text()
: "";
        }
        return ""; // 没有正确答案时返回空字符串
    }
```

步骤5：分页查询测试；

如果出现错误，可以打开MyBatis的日志，输出错误信息到控制台：

Day09-03. Mybatis-基础操作-删除(预编译SQL)*哔哩哔哩*bilibili

3、按照关键词查询

数据库预备知识：

要在用户名中查询某个特定的字符串，可以使用如下 的sql模糊查询语句：

```
SELECT * FROM user WHERE username like '%abc%';
```

对应的MyBatis的mapper层的代码：

```
@Mapper
public interface Questionapper {
    @Select("SELECT * FROM questions WHERE questionText LIKE CONCAT('%', #
{keyword}, '%') AND isDelete=0")
    List<Question> findByName(String keyword);
}
```

Controller中的代码：

```
@GetMapping("/findQuestion")
public Result getUser(String keyword){
    List<qsBeanOutManage> qsBeanOut=questionService.findByName(keyword);
    return Result.success(qsBeanOut);
}
```

Service层中的代码：

```
//接口;
public  List<qsBeanOutManage> findByName(String keyword);

//实现:
public  List<qsBeanOutManage> findByName(String keyword){
        List<Question> userList=userMapper.findByName(keyword);

        List<QSBeanOutManage>qsBeanOutManage =
Tools.convertToQSBeanManageList(questionList);
        return qsBeanOutManage;
}
```