# 项目简介

本项目的名称为**SongGuo\_Learning**（以下内容使用“项目”代指本项目）， 取了我名字的一个字，给项目留下一点属于我个人的印记。

项目的基本类型是一个**在线学习平台**，它旨在开发一个在线学习平台，为学生和教师提供一个便捷的教学与学习环境。学生可以通过平台选择和参与教师创建的选修课程，进行课程学习和答题，教师可以创建课程、管理课程内容和监控学生学习情况。

项目源自广东工业大学QG训练营2024年招新考核中的后台组最终考核题目。本人选择此项目来作为最终考核的选题。

项目要求使用最基本的JavaWeb技术栈开发，对处在大一阶段的学生来说是一个巨大的挑战，对于工作室，其综合考察开发者的业务能力以及代码风格还有高级语言使用基础，是检验想加入QG工作室的大一学生的极佳方案。而对于参加考核的大一新生们， 此项目是对他们近期学习的检验工具，也是接触实际开发大型项目的流程的引路项目。

毋庸置疑，这是一个非常具有意义的项目。

# 设计思路

## 组成部分以及关系分析

首先，分析这个项目的各个组成部分以及它们之间的关系。

**教师、学生**--->使用平台的用户

**课程、章节、题目、评论**--->平台功能的承担者

在一个**教师**名下，会有Ta创建的**课程**；而**课程**下有不同的**章节**；**章节**下有安排好的**题目**。依据这个基本关系，有如下的结构图。

#### 拥有



拥有

课程

拥有

章节

评论区



题目

拥有

教师

而学生，主动参加课程，参与章节学习并且完成题目，这些行为会产生学生与上述**课程，章节，题目**的关系数据。

学生参加课程



学生



参与课程

挑选课程



生成一条学生参与课程的数据

学生进入课程某一章节学习



选择章节

进入课程



生成一条章节学习记录

完成题目提交

学生完成某一道题



生成一条题目提交记录



完成题目提交

进入章节

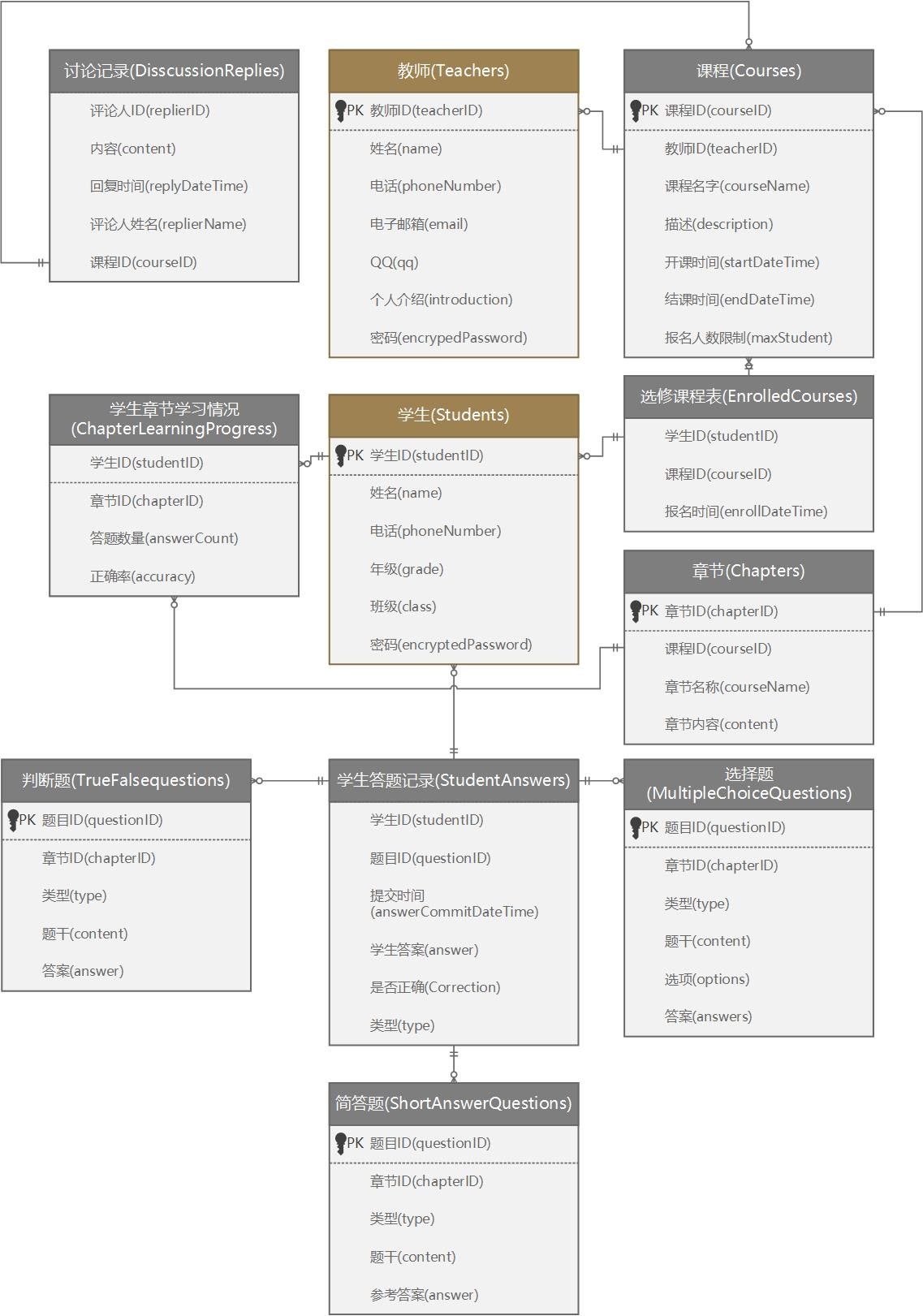
根据上述关系，得到三个存储关系的组成部分：**参与课程数据、章节学习记录数据、题目提交数据**。

题目也有不同的三种类型：**简答题，判断题，多选题**。学生和教师在课程页面可以发言，对应有**评论。**

##### 上述可得整个项目在数据结构上会包含以下的组成成分：

* + 1. 教师
    2. 学生
    3. 课程
    4. 章节
    5. 简答题
    6. 多选题
    7. 判断题
    8. 讨论记录
    9. 学生选修课程数据
    10. 章节学习情况数据
    11. 答题记录数据

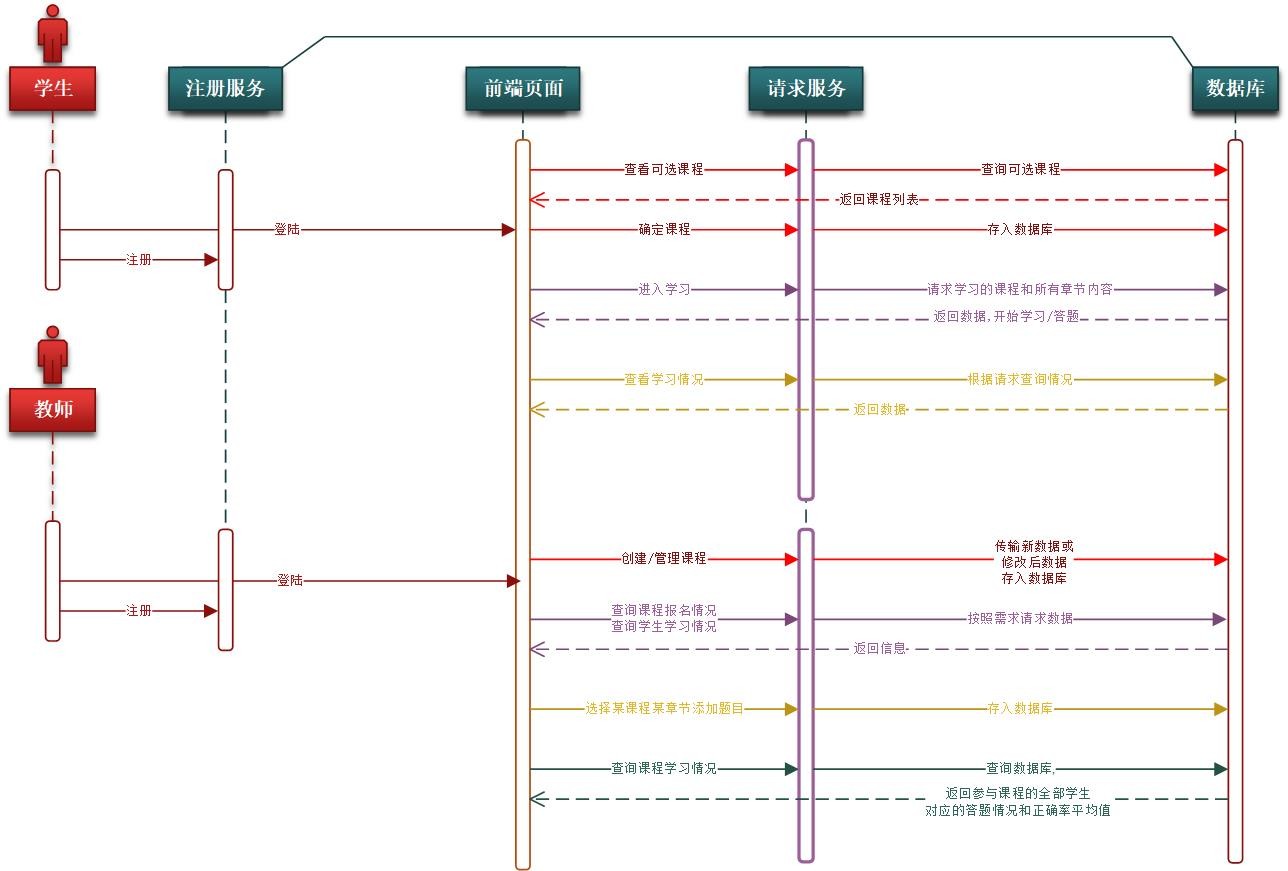
这十一个组分需要存储对应的数据供前端展示，则可以根据它们之间关系有如下表关系。



这**11张表**即为项目DAO层交互的数据库对象。

## 项目业务逻辑分析

作为一个在线学习平台，业务逻辑总体来看不是太复杂。



教师和学生的各项操作在整个业务逻辑的考量来说都是比较清晰明了的，大概分为两类。

* + 1. **查询**已有数据供前端显示
    2. **新建**操作、**提交**操作等等需要**传输数据后存入数据库**
    3. 比对数据，返回处理结果（如登陆校验以及题目答案比对）

## 分包结构以及pojo设计

题目要求使用**MVC分包以及BaseServlet还有统一结果集**。依据上述分析：项目分包（此处pojo包包含类的设计，其他仅代表包）

base\_servlet impl\_servlet service

##### pojo

impl

**question**

**MultipleChoiceQuestion.java Question.java ShortAnswerQuestion.java TrueFalseQUestion.java**

**Chapter.java ChapterLearningProgress.java Course.java DiscussionReply.java EnrolledCourseMap.java Result.java**

**Student.java**

**StudentAnswer.java Teacher.java**

dao

filter（过滤器） constnum（常量） config（配置类） utils（工具类）

其中，**Question类**以及**Result类**不在数据库中有对应的表，**Result类**为统一结果集类，作为前后端传输数据的载体，而**Question类**在后文详细设计模块会提及。

## 技术栈以及依赖引入

项目使用技术如下。

**jdk版本** 1.8

**Tomcat版本** 9.0.88

**maven版本** 3.9.6

**开发IDE** Idea2024.1

**前端开发** vue2+ElementUI+vue-cli

pom文件引入依赖如下。

**单元测试** Junit4 用于单元测试

**数据库连接驱动** MySQL8.0.29 用于连接数据库

**servlet** 3.1.0

**json序列化工具** fastjson 2.0.9.graal 用于解析json

**日志框架** slf4j 1.7.30 用于记录日志

**令牌** jjwt 0.9.1 用于生成token

除此之外，项目使用了手写的**mybatis**以及**数据库连接池**，在后文详细设计模块会提及。

# 详细设计

先详细介绍**手写mybatis、手写数据库连接池**的实现。

## 手写数据库连接池

### 读取配置信息

数据库连接池（下文简称“池”）需要从porperties文件 读取基本的配置信息来进行初始化和一系列的设置。故需要编写文件properties.properties 来配置池。

编写好配置文件之后，在config 包下编写DataSourceConfig 类读取文件来获取配置信息。在类中选择使用静态代码块来进行properties文件的加载和解析。

使用ClassLoader加载类路径下的属性文件properties.properties 并将其加载到Properties对象中。获取ConnectionPool类的类加载器，然后使用该类加载器来获取属性文件的URL。如果资源不为 空，则加载属性文件到Properties对象中。类中根据解析的每一个属性都有对应的静态成员变量以及这些变量的getter和setter方法。

### 初始化空闲连接池

首先创建一个接口MyConnectionPool ，其中有两个方法。然后创建实现类ConnectionPool 实现接口方法。

其中， connectionCount 变量是记录当前连接数的变量，每创建一个新变量则会自增1。创建两个

Vector容器（线程安全）存放**空闲连接**以及**使用中连接（为包裹类，包含连接对象以及被调用的时**

**间）**。同时，使用synchronized 关键字保证线程安全，避免多个用户同时访问数据出现同时获取连接而发生线程冲突。而类ConnectionEntity 则是一个包裹类，其存放一个连接以及其对应的被获取的时间，为后面的健康检查提供依据。

创建连接以及释放连接的方法编写完毕之后，开始编写ConnectionPool 的构造函数，即池的初始化。初始化应该根据配置的信息有**初始连接数、是否开启健康检查**。

健康检查为检测连接被调用之后持续被调用状态的时间是否超时，若超时了则直接close这个连接。后文介绍获取连接的时候会进一步细讲健康检查的详细实现。

### 获取连接

池初始化完毕后，外部可以创建一个池的实例对象，每创建一个池对象则会创建若干连接放入此实例中供调用。（此处若采用单例模式会更好，不过在手写后自用的情况下保证自己不随意创建实例，满足基本需求即可，整个项目仅会实例化一次池）

当外部需要从池中调用连接，此时池会有以下情况：有空余连接

将空闲容器对应的连接移除，将其包裹后放入正在使用的连接容器中，随后return这个连接对象给外部调用。



有空余连接

从空闲容器中移除一个连接

包裹后放入正在使用的容器内

return此连接对象

没有空余连接，但是总连接数量没到达池最大容量

新创建一个连接，包裹后放入正在使用的连接容器中，随后return这个连接对象给外部调用。



没有空余连接但是连接总数未到达最大值

创建新连接对象

存入正在使用的容器内

return此连接对象

没有空余连接，总连接数量也已经到达最大容量

让线程等待，一定时间（配置文件设置）后进入递归，再次尝试获取连接。



递归调用再次尝试获取连接



线程进入等待

没有空余连接且连接数达到最大值

除此之外，如果开启了健康检查，池会创建一个定时任务，每隔一定时间则会检查使用中的连接容器，遍历所有包裹类，对比当前系统时间和包裹类含有的时间信息（即连接被调用的时间）的差值与配置中指定的超时时间，**如果差值大于超时时间，则认为这个连接出现异常，直接从容器中移除，全部连接总数自减1，然后直接close此连接对象。**

### 释放连接

外部释放调用的连接，从正在使用的连接容器中寻找连接对应的包裹类对象，将其从容器中

remove，然后把连接对象交还给空闲容器。

**同时，在remove之后，调用notifyAll()方法，唤醒正在等待的需要取得连接的线程（当然线程在指定等待时间后会自动唤醒，不一定需要被唤醒）。**

## 手写mybatis

首先明确手写mybatis之后能实现什么。

##### 定义一个mapper接口，接口内根据需求在方法上添加@Select、@Update等等注解，内部为sql语句，方法签名的参数表要根据sql填写传入的参数，同时在每一个参数旁加一个@Param注解，里面对应参数名字。service层获取一个mapper对象，调用这个mapper对象的方法查询数据库获取数据。

这是直观的dao层和service层通过mybatis交互的逻辑。

但是接口不能用来直接创建对象，实际获取的mapper对象应该是一个**代理对象**。

### 创建代理对象工厂

明确这个工厂的目的是生产什么。

##### 传入一个mapper的字节码文件，返回它的一个代理对象，代理对象可以像mybatis一样读取注解值，解析占位符并填入传入的参数值，执行sql语句并且按照需求返回数据。

故先设置一个静态成员变量**connectionPool**用来获取连接，定义一个静态方法**getMapper**，方法里创建一个新的**代理对象**，接受getMapper方法接受的mapper字节码文件。

### 分析注解

根据需求可以大体分成两种

* + - 1. **@Select** 查询数据，可能需要返回实体类
      2. **@Update、@Insert、@Delete** 更新数据，可选择返回受影响条数

（注：项目中注解名字添加了后缀，表明项目的mybatis为手写，文档依然采用正常注解名）而在处理sql语句执行后的resultSet之前的逻辑基本没有差别，故先展示从读取注解到构造

**preparedStatement**之间的实现逻辑。

假设现在代理对象被调用的方法注解为@Select。

##### 获取连接，创建存放返回结果的list

从工厂类的**connectionPool**中获取即可。

##### 获取@Select注解值

通过反射获取当前方法的@Select注解值，赋值给字符串**sql**



##### 获取传入参数名对应参数值

通过反射获取到的方法参数组成的数组，参数名称会**被覆盖为索引args[n]**。我们传入参数，是为了**参数名称和占位符内名称对应来赋值**，显然如果用args[n]是不能做到很好的代码可读性的。所以我们需要做一个map，**存放我们写代码时设定的参数名字以及实际调用方法时传入的对应的值，例如"name"**

**= "LiHua"，“age” = 1这种可读性强的对应关系**，而不是通过反射获取的args[0] = "Lihua"，args[1] = 1

这样不知所云的对应关系。而之前所说**通过@Param注解对应参数名字**就是拿来替换args[xxx]的。



获取方法的所有参数

得到Parameter数组

遍历数组取得 每个对象的 @Param注解值

map内放置新的键值对键为上一步取得的注解值值为代理对象接受到的 对应参数

这个map后文将叫做**ParameterMappingMap**。

### 解析sql语句

对于sql语句的处理，有两个要点。占位符

想要做到动态执行sql，占位符是必不可少的。**#{xxx}**内的xxx对应方法签名的相同名称参数，如何将**#{xxx}**替换为实际执行方法时传入的参数？

转化

为了防止sql注入，选择使用**preparedStatement**进行执行语句的构建，而要将注解中带有占位符的sql语句（现在还只是字符串）转化成一个**preparedStatement**则需要将占位符替换为符号**？**。

在**分析注解**一步，我们已经获取到了参数名以及调用方法时传入参数值的对应关系。根据上述两个要点，可以得出如下的执行逻辑。



取得xxx，存入一个容器parameterMappings

按顺序获取sql语句出现的#{xxx}形式的部分



遍历parameterMappings，用遍历到的xxx作为键从上一步获取的 ParameterMappingMap中获得实际传入参数值

将#{xxx}替换为？

到这里，意味着**已经取得#{xxx}中xxx的对应传入参数值，并且只需要按顺序把？替换为对应传入参数值即可。**但是问题又出现了：**如何替换？**

**preparedStatement**提供了set方法来用传入的参数替换sql语句的？符号，但是根据值的类型不同，set方法也不同，有setInt，setString，setBoolean等等。而我们传入参数也是动态的，应该如何去调用参数类型对应的setXXX方法来替换sql语句的问号呢？

##### 答案是采用TypeHandler接口，创建XXXTypeHandler实现类供选择，在赋值阶段根据参数类型获取对应的TypeHandler来进行赋值操作。

给工厂增加一个静态成员hashmap **typeHandlerMap**，将各个类例如**Integer、String、Float**的字节码文件作为键，**对应的TypeHandler**作为值。

接着上方的流程图，则有下一步操作。



取得参数的class，作为键取得typeHandler，对preparedStatement进行赋值

遍历parameterMappings，用遍历到的xxx作为键从上一步获取的 ParameterMappingMap中获得实际传入参数值

到此，**preparedStatement**构建完毕。下面继续讲解如何处理@Select需要返回实体类的业务

### 执行preparedStatement，获取resultSet

执行语句，获取结果集。

### 取得resultSet的数据，用实例封装数据

* + - 1. 先从方法签名的**返回类型**取得要返回的实体类。如果返回类型是一个普通的类，则直接取得这个类赋值给类型变量**resultType**，如果是一个List<?>，则取得？，然后将其赋值给 **resultType**。
      2. 再从resultType中获得查询的所有字段，将字段信息存入**columnList容器**中。

##### 通过反射和字符串拆解获取返回类型pojo类的属性名和对应setter，存入一个hashmap，名为setterMethodMap

* + - 1. 进入循环，每一轮循环创建一个**resultType**类型的实例对象，根据上述步骤中已经获得的 **columnList**里面存放的字段作为键从**setterMethodMap**取得setter方法，然后给这个实例对象设置属性值。

另外一提，设置属性值也需要**动态**地根据字段名从resultSet获得查询到的信息来调用setter方法给实例对象的属性赋值，类型是变化的，情况类似之前的**构造preparedStatement**。当然只需要通过 **TypeHandler处理即可**。

* + - 1. 每一轮构造完实例对象之后，将实例对象加入之前创建好的存放结果的容器中。

### 根据方法签名返回结果返回数据



不是

判断返回类型是不是List



是



是

返回null



不是

判断容器是否为空

直接返回存放结果的容器

判断返回类型是不是void

至此，@Select的业务逻辑说明完毕。



是

返回null



不是

返回容器的第一个元素

### 其他注解的处理方式

增删改只需要执行语句不需要返回值, 重点在获取注解值。所以直接获取方法上的所有注解，再获取第一个注解的类型，通过反射获取注解的**value**方法，执行方法获取注解值。

Annotation[] annotations = method.getAnnotations();

// 获取注解的类型

Class<? extends Annotation> annotationType = annotations[0].annotationType();

// 获取注解的值

// 通过反射获取 value 方法

Method valueMethod = annotationType.getDeclaredMethod("value");

// 如果注解有 value 方法，就获取值

String sql = (String) valueMethod.invoke(annotations[0]);

剩下的就和@Select注解没差别了，只需要返回语句执行后的受影响记录数就行。

### 连接与资源的释放

每次调用mapper之后将在业务执行完毕时通过连接池释放连接，同时关闭结果集。

## 登陆注册业务

### 注册

考虑到新用户不会有ID，使用电话号码作为注册时验证是否为新用户的凭证。

前端将会先传输电话号码**phoneNumber**还有身份类型**type**到服务器，服务器接受到这两个数据后，根据类型进入不同的方法验证是否为已注册用户。

Object received = null; if (accountType == 1) {

received = accountService.validateIsNewTeacherAccount(phoneNumber);

}

else if (accountType == 2) {

received = accountService.validateIsNewStudentAccount(phoneNumber);

}

在service层会与dao层交互，用电话号码作为条件查询数据库，返回查询到的数据到servlet。

如果数据为null，则在数据库中**根据提供的电话号码查询不到对应的信息**，显而易见这个电话号码未被注册过，通过新用户验证，向前端发送对应信息。

反之则易知此电话已经被注册，未通过新用户验证，发送相应信息。

if (Objects.isNull(received)) {

// 未找到传入的电话对应的用户

resp.getWriter().write(JSON.toJSONString(Result.success("VALIDATION\_PASSED")));

}else {

// 用户已存在

resp.getWriter().write(JSON.toJSONString(Result.error("VALIDATION\_FAILED")));

前端如果接受到新用户验证通过的信息，则会正式发送用户的注册信息到服务器。

此时，密码已在前端通过**sha256方式**加密随着用户数据传输到服务区，明文密码不会出现在传输过程，将被存储到数据库的是加密后密码。服务器接受到post请求，读取数据，将数据转化为一个 **JSONObject**（不使用fastjson的parseObject方法转化是因为不知道注册的是学生还是教师，其携带的数据不一样），将其传输service层的**createNewAccount**方法。

此时，学生不需要凭证，可以自由注册；而教师需要携带**教师身份凭证**，只有在前端输入了正确的凭证密钥，教师注册才可以通过。

在service层依据类型决定ID的首位，随后数字随机生成拼接成九位的数字作为ID，取出

**JSONObject**的各个属性值，根据类型传入dao层对学生或教师的添加方法，创建新用户。

servlet接受新增操作受影响的记录数i，i大于零则成功注册，向前端返回相应信息。

### 登陆校验

用成功注册之后，系统会返回ID，在登录时可以使用注册时使用的手机号或者ID登录。当用户准备登录时，填写完信息之后，点击确定登录，前端将会发送一次请求，**只携带登录的凭证（ID或者手机 号）还有账户类型（学生还是教师？）**，在服务器将会对这个凭证进行校验 **loginValidation**，查找是否有相应的账户。前端已对这个凭证**使用正则表达式规范输入**，只允许9位的id或者11位的手机号。

在service层，将根据凭证的长度转化成Integer代表ID或者String代表手机号。与dao层交互后返回一个object，servlet将判断这个object是否为null，**如果是**则登陆账户不存在，返回相应信息给前端。而**如果不是**null，则凭证有对应的账户存在，返回相应信息给前端。



根据凭证长度确定是ID登陆还是手机号登陆

接收登陆凭证

#### 凭证长度为9 凭证长度为11



ID登陆，根据类型查询用户表

手机号登陆，根据类型查询用户表

返回查询结果 返回查询结果



是null

账户不存在

判断查询结果是否为null

前端如果接受到不存在的返回结果则弹出对应提示，不再进行操作。如果接受到账户存在的返回结果，则会**将登陆凭证和加密后密码（前端使用sha256算法加密）再次传输到服务器进行账密校验 loginConfirmation**。与上方验证是否存在账户的业务逻辑相似，也是根据信息查询表，这次多了一个加密后的密码作为条件。



不是null

账户存在

如果查询结果不为null，则说明在前端，登陆者输入了正确的账号密码。此时将查询结果作为载荷，生成Token，发送给前端的统一结果集中，data为token，而message为查询到的用户ID。

if (o instanceof Teacher || o instanceof Student) { Map<String, Object> claims = new HashMap<>(); claims.put("userData", o);

String Jwt = Jwts.builder()

.signWith(SignatureAlgorithm.HS256, ConstString.JwtKey)// 签名算法 密钥

.setClaims(claims)// 载荷

.setExpiration(new Date(System.currentTimeMillis() + 3600 \* 1000 \* 3))// 三小时有效期

.compact();

int id;

if (o instanceof Teacher) {

id = ((Teacher) o).getteacherID();

} else {

id = ((Student) o).getStudentID();

}

String rs = JSON.toJSONString(Result.success(String.valueOf(id), Jwt));

## 账户个人信息修改以及数据回显

此部分对于学生和教师的操作将会交给各自对应的servlet处理，不需要携带类型信息。

用户修改个人信息时，前端将会将修改后的个人信息传输到服务器，同时携带用户ID信息。servlet接受到请求，获取json数据后**使用序列化工具将json数据转化为相应的实例对象**。

随后在service层与dao层交互，更新ID对应的数据信息。

前端每次加载或者刷新个人信息主页时，都会向服务器发送请求，其携带着ID信息，每一次查询都会返回对应的用户信息。

## 教师添加课程、章节以及数据回显

教师填写新课程表单，提交到服务器接收，将json数据提取，使用序列化工具将json数据转化为

**Course**实例，传输到service层与dao层交互，进行新课程的添加。

章节同理，服务器转化json数据，提取属性值进行新章节的添加。新章节附属与课程，前端会在发送数据时带上它附属的课程ID一起存入数据库。

教师每一次刷新课程管理页面，前端都会发送请求到服务器请求课程信息。其**携带教师ID**，服务器接收ID去查询数据库，返回教师下的课程列表信息给前端。而进入课程管理页面后，每次刷新也会向服务器请求数据，携带**课程ID信息**，返回课程下的章节列表信息给前端。

## 教师的课程章节添加题目以及数据回显

**这是项目的一个难点。**

### 添加题目

前面分析需求时得到，题目有三种类型：判断题，选择题和简答题。添加题目的业务应该如何实现呢？

##### 我的方案为，前端限制每次为一个章节添加一道题，填写新题目表单后，向服务器发送题目数据时会携带题目类型参数，根据参数决定后台把接收到的json数据转化成什么类型的题目实例。

int type = Integer.parseInt(req.getParameter("questionType"));// 题目类型参数

String s = req.getReader().readLine();// json数据 Question question = null;

if (type == ConstNum.TrueFalseQuestion) {

// 判断题

question = JSON.parseObject(s, TrueFalseQuestion.class);

} else if (type == ConstNum.ShortAnswerQuestion) {

// 简答题

question = JSON.parseObject(s, ShortAnswerQuestion.class);

} else if (type == ConstNum.MultipleChoiceQuestion) {

// 选择题

question = JSON.parseObject(s, MultipleChoiceQuestion.class);

}

此处使用了自定义的常量类ConstNum ，其中包括一些表示类型的参数，如**1代表老师，2代表学**

**生，3代表课程等等。**

到这一步，题目会根据类型变成三种不同的实例类。类型不同，传入service层就需要三种对应的方法，这显然有点不合理。

此时，可以使用**多态。**创建一个超类Question ，其包含题目共有的一些特性例如**所属章节ID，题目ID，类型参数，题干**，让三个题目对应类继承Question ，成为子类，其包含题目特有的属性，例如选择题会含有**选项**。在service层的对应方法参数表可以这样编写：

// 添加一道新题目, 传入type

void addQuestionToChapter(Question question, int type);

三个子类对应实例都可以作为参数传入方法，只需要同时携带类型参数就可以在service层根据类型参数决定传入的题目将要存储到哪一张题目表。添加成功，服务器向前端发送对应信息。

### 数据回显

在课程管理页面，得到章节信息后，每一个章节的内容以及题目会用一个Vue组件来展示。组件加载时会携带章节ID信息向服务器请求章节已有题目数据。

此时，题目数据根据类型存放在不同的表中，如何查询附属在某一个章节下的题目数据呢？

##### 依然是多态。

依据章节ID在三张题目表中分别查询题目数据，得到List ，List，List，再创建一个List，将三个列表全部合并到这个超类实例组成的列表中，将这个列表返回给前端进行动态处理（每一道题都包含类型参数）。

## 学生挑选课程以及参与课程

### 挑选可选课程

学生打开挑选课程页面时，前端携带学生ID向服务器发送请求，查询学生未参加过并且在开课时间内的课程。



查询学生已经参加过的课程，得到一个enrolledCourseMap存放课程ID和学生ID对应关系

查询开课时间内的课程，得到一个courses列表



遍历courses, 剔除ID包括在ids里面的课程

遍历enrolledCourseMap，取得一个ids列表存放学生参加过的课程ID

将courses返回前端进行展示。



得到所有可参与的课程courses

### 参与课程

学生在前端点击**参与课程**按钮，前端将发送请求，携带课程ID以及学生ID，服务器接收请求之后存入**EnrolledCourses**表，这样就存入了一个学生参与课程的关系数据了。

## 学生进入课程学习与答题

### 进入已参与课程学习页面

进入课程学习，首先要展示学生已经参与了的课程。上文提到了**挑选可选课程要展示学生未参与的课程**，逻辑在参与与否上是对立的，那么我们只需要将挑选可选课程的service层代码稍作修改，将**剔除 ID包括在ids里面的课程**这一步修改为**剔除ID不包括在ids里面的课程**，其他的逻辑保持一致，则可以将已参与的课程信息传输到前端了。

学生点击**进入课程界面**按钮，则会跳转到课程学习页面。这个页面的数据回显与教师的课程编辑界面的数据回显基本一致，同样返回课程信息和课程下章节信息，但是在章节数据回显方面不大相同。

### 答题

##### 这是项目的又一大难点。

实际上这个难点在于前端如何编写。此文档不对前端做详细介绍，故只对前端提交题目的方案做一个说明：**以一个章节为单位，答题数据作为post请求的载荷，url携带章节ID作为参数。**

服务器接受到这个post请求，首先获取章节ID，再获取载荷携带的json数据，使用json序列化工具将json数据转化为**List**，将两个获取到的数据传入service层。