

## 今日目标

---

- 了解项目功能与工程结构
- 理解自定义MVC框架原理及基本使用
- 掌握项目工程初始化及子模块创建
- 掌握后台系统的登录与退出
- 掌握学科管理增删改查功能

## 1. 项目需求

---

### 1.1 项目背景

黑马面是一款面向程序员的面试刷题小程序。针对目前大量学员在培训完之后直接去面试企业的通过率低的问题，公司研发了黑马面小程序，学员在空闲时间可以通过查看企业真实面试题，不仅可以查看企业真题，也可以通过刷题寻找自己的短板进行补充。

### 1.2 需求概述

根据项目要求，前台展示是微信小程序，后台需要一个Web端运营管理后台完成对数据统一录入、筛选、审核和其他初始化的操作，故整个系统会分为Web端管理后台和微信小程序。

#### 1.2.1 运营管理后台

运营管理后台是一个Web应用，运用前期学过的Web应用开发知识，采用MVC设计模式来完成整体开发。管理后台的网页部分，也采用前后端分离的方式，全部通过ajax调用后端API接口来完成。

#### 1.2.2 微信小程序

微信小程序有自己独立的开发环境，其开发工具是微信官方提供，通过该开发工具完成小程序页面的开发及与微信生态环境的交互。小程序端的业务数据获取和业务提交只能通过发送ajax请求与后台接口通信来完成所有的业务操作，所以后台服务器必须为小程序所有业务操作提供相应的API接口（应用程序编程接口），数据交换可采用常规的JSON格式。

这种开发模式，是典型的前后端分离模式，前后端分离核心思想就是前端HTML页面通过AJAX调用后端API接口来完成，并使用JSON数据进行交互。

### 1.3 功能概述

#### 1.3.1 管理后台功能列表

序号	模块	子模块	描述
1	用户管理	角色管理	通过添加角色并对不同角色赋予不同的权限从而完成对系统中试题录入、审核等角色的配置，实现不同用户操作不同资源。
		用户管理	通过用户管理，可以派生不同角色和权限的新用户。 创建的新用户需要指定角色及权限。
2	企业管理	企业管理	面试真题来源企业，通过企业管理可以完成对面试题所属企业的管理。 新增题目可以关联所属企业。
3	方向管理	方向管理	方向为行业方向，比如电子商务、互联网金融、O2O、医疗服务等。 新增企业，必须选择1个或多个行业方向。 方向管理可以实现对行业方向的管理操作。
4	学科管理	学科管理	题目必须隶属某一个学科，通过学科管理可以管理面试题目的大学科，比如Java、Python、PHP等。
		学科目录管理	每个学科下面可以管理学科二级目录，比如Java学科，可以设置Java基础、Java Web、Spring框架等二级目录，目前仅创建学科二级目录。
		学科标签管理	新增题目时，可以为题目设置多个技术标签，标签是创建在学科下面的。 通过学科标签管理，实现对学科标签的操作管理。
5	题库管理	基础题库	用户根据自己的权限，可以录入基础题库。 基础题库模块可实现题目的新增、更新、预览、加入精选、复杂查询等操作。
		精选题库	用户根据自己的权限，可以录入精选题库。 基础题库模块可实现题目的新增、更新、预览、题目审核、复杂查询等操作。
		新增题目	题目必须隶属某一学科下的某一二级目录，可以为试题指定多个技术标签。 题目可以选择来源企业，选择来源企业可以选定其所在城市及行业方向 题目分为单选、多选及简答三种类型。 单选、多选选项可以选择图片上传 题目可以选择学科下的多个标签
		试题预览	试题列表中的题目都可以通过预览方式查看显示效果。
		试题审核	只要精选题目列表中的题目，可以进行试题审核。 审核通过自动发布。

1.3.2 前台小程序功能列表

序号	模块	子模块	描述
1	用户登录	用户登录	当前系统必须授权登录方可访问
2	设置城市及学科方向	设置城市及学科方向	后续看到的题目数据全部根据当前用户所选的城市及学科方向来提取数据
3	题库分类列表	题库分类列表	分类有三种方式（按技术、按企业、按方向） 按技术实际是按学科目录，后台接口根据当前学科的学科目录来提前学科目录列表 按企业，后台接口根据当前城市提前企业所属城市列表 按方向，后台接口根据所选城市和学科选取行业方向列表 列表中包含所有分类数据及用户记录数据（已完成题目记录）
4	题库分类题目列表	题库分类题目列表	根据所选分类，提前对应的题目列表，包含题目详情信息
5	题目操作	收藏	针对某一题目，用户可以收藏这个题目
	答案提交	答题	答题是在客户端完成 单选题目，只要选中某一选项，自动判断对错 多项选择，需要选择选项后，单独提交答案，完成判断对错 简单题，需要用户根据自己对题目的分析判断，通过查看解析后，完成理想与不理想操作提交
		提交答案	无论单选、多选还是简单，最终需要把当前题目信息提交到后端，后端保存用户做题记录。
6	个人中心	个人中心	获取用户信息数据，展示在个人中心
		继续答题	跳转到最后一次完成答题的位置，继续答题

2. 项目设计

2.1 数据库设计

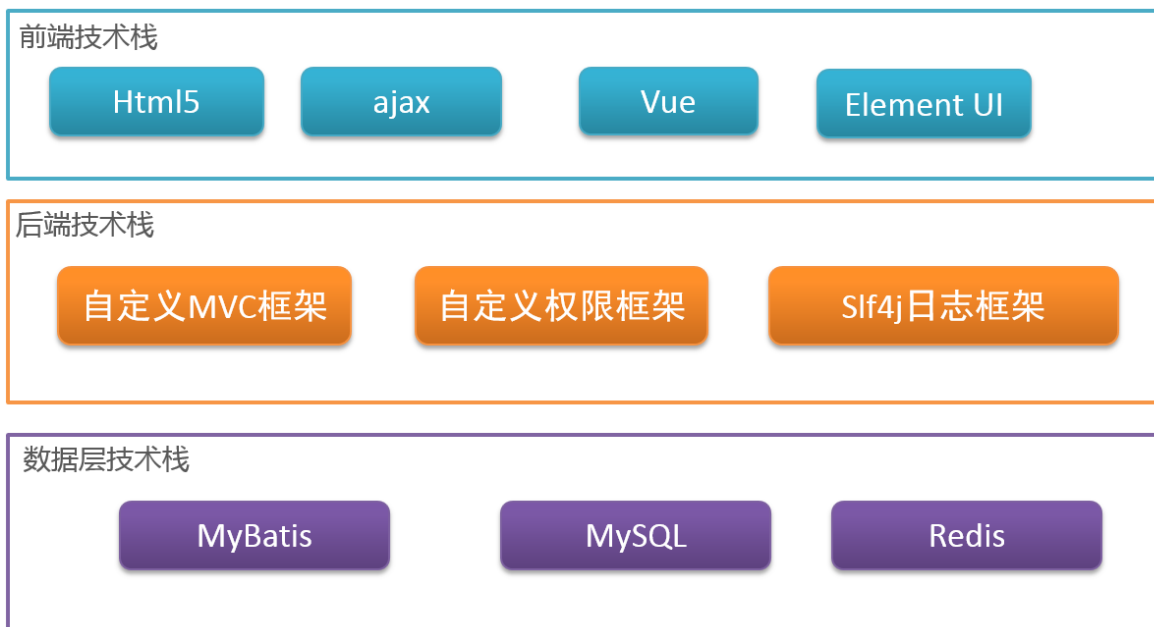
本项目一共有18张表，其中13张主表，5张关系表。

序号	中文名	表名	备注
1	t_user	用户名表	管理后台用户表
2	t_role	角色表	
3	t_permission	权限表	
4	tr_user_role	用户角色关系表	关系表
5	tr_role_permission	角色权限关系表	关系表
6	t_dict	数据字典表	存储项目中的常规数据信息，比如省市数据、邮政编码、职业类型等等。
7	t_company	公司表	题目来源公司表
8	t_industry	行业方向表	城市所属行业信息表
9	tr_company_industry	公司行业方向关系表	关系表
10	t_course	学科表	
11	t_catalog	学科目录表	学科二级目录
12	t_tag	学科标签表	学科所属标签
13	t_question	t题目表	存储题目信息
14	t_question_item	t题目选项表	存储题目选项信息（单选、多选选项）
15	tr_question_tag	题目标签关系表	关系表
16	t_review_log	题目审核表	存储审核记录
17	t_wx_member	会员表	小程序登录用户信息表
18	tr_member_question	会员做题记录表	关系表，c存储会员所有做题记录

更详细信息，参考资料中的数据库设计文档。

## 2.2 架构设计

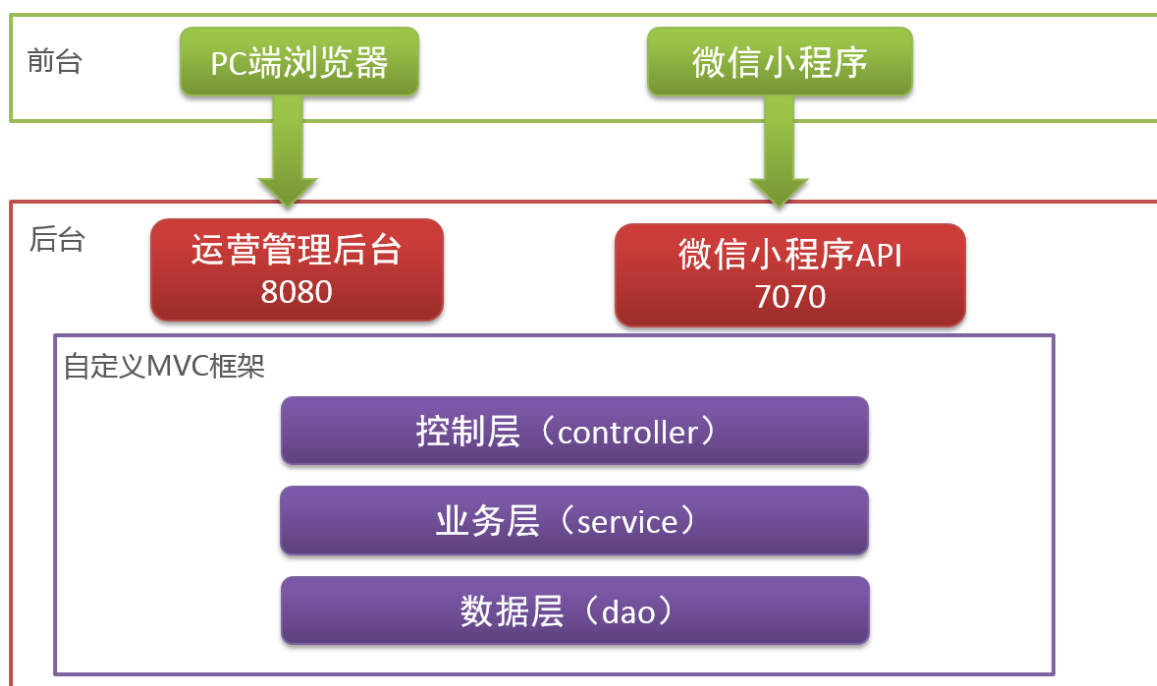
### 2.2.1 技术架构



### 2.2.2 系统架构

运营管理后台主要面向公司内部运营人员使用，访问人员主要来自公司内部，未来从安全性和访问量考虑分析，可以和小程序端API接口应用隔离安装部署，所以也需要单独构建一个Web应用。

微信小程序面向前端用户，未来从业务增长速度来讲，可能访问的用户越来越多，故从安全性、可维护升级和可扩展性等角度分析，微信小程序API接口需要独立安装部署，所以需要单独构建一个Web应用；



### 2.3 工程设计

通过系统概述分析，微信小程序端使用微信小程序开发工具来完成小程序端页面的开发，微信小程序API接口、运营管理后台API接口及运营管理后台网页中的ajax通信全部通过IDEA开发工具来开发。

通过以上分析，至少需要构建2个独立的Web应用，虽然独立但它们还是需要访问同一个数据库中的数据表，所以也会使用一些相同的数据模型和工具，比如POJO类、实体类、常量类、工具类等。所以最终采用1个父工程，多个子模块的方式来组织该项目。

本项目最终会有1个父工程，4个子模块构成。

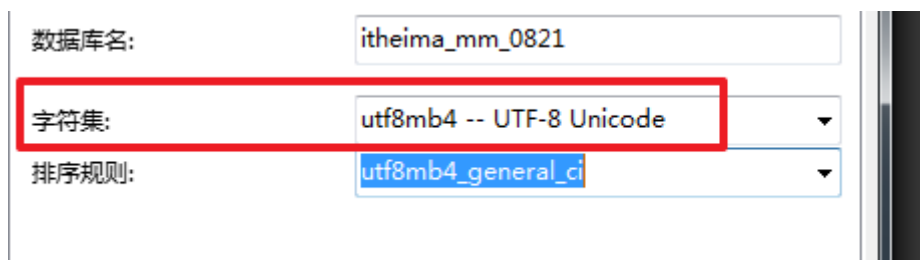
编号	中文名称	英文名称	备注
1	父工程	itheima_mm	库依赖管理 把需要的依赖全部放到pom文件中
2	自定义MVC框架	mvc_framework	自定义MVC框架 打包方式jar包
3	公共子模块	mm_common	pojo、entity、常量类、工具类 依赖mvc_framework 打包方式jar包
4	运营管理后台	mm_oms_backend	web工程（8080端口） 依赖mm_common 打包方式war包
5	微信小程序API	mm_wx_api	web工程（7070端口） 依赖mm_common 打包方式war包

## 3. 项目初始化

### 3.1 初始化数据库

#### 3.1.1 创建数据库

数据库的字符集必须选utfmb4，成员表是微信授权用户，为支持类似emoji表情等数据格式。



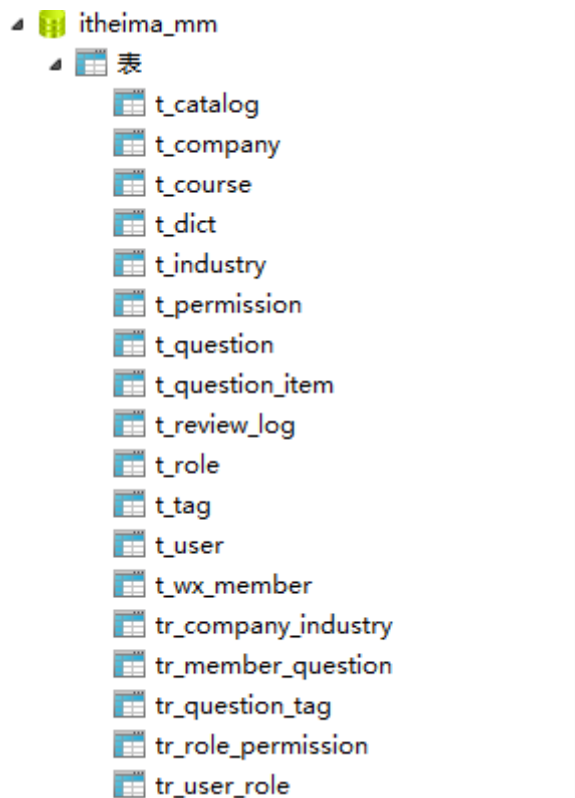
The screenshot shows a database configuration form with three fields: '数据库名:' (Database Name) with the value 'itheima\_mm\_0821', '字符集:' (Character Set) with the value 'utf8mb4 -- UTF-8 Unicode' (highlighted with a red box), and '排序规则:' (Collation) with the value 'utf8mb4\_general\_ci'.

```
drop database if exists `itheima_mm`;
create database `itheima_mm`
DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_general_ci;
```

#### 3.1.2 导入数据脚本

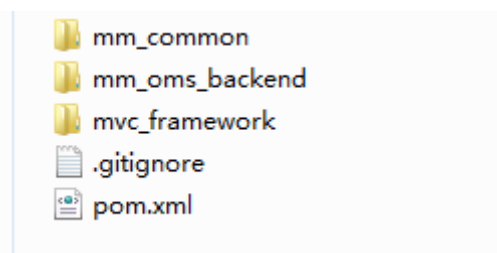
选中创建的数据库，运行itheima\_mm\_last.sql文件，该文件存放在资料-数据库设计中。

导入成功后，如图所示：



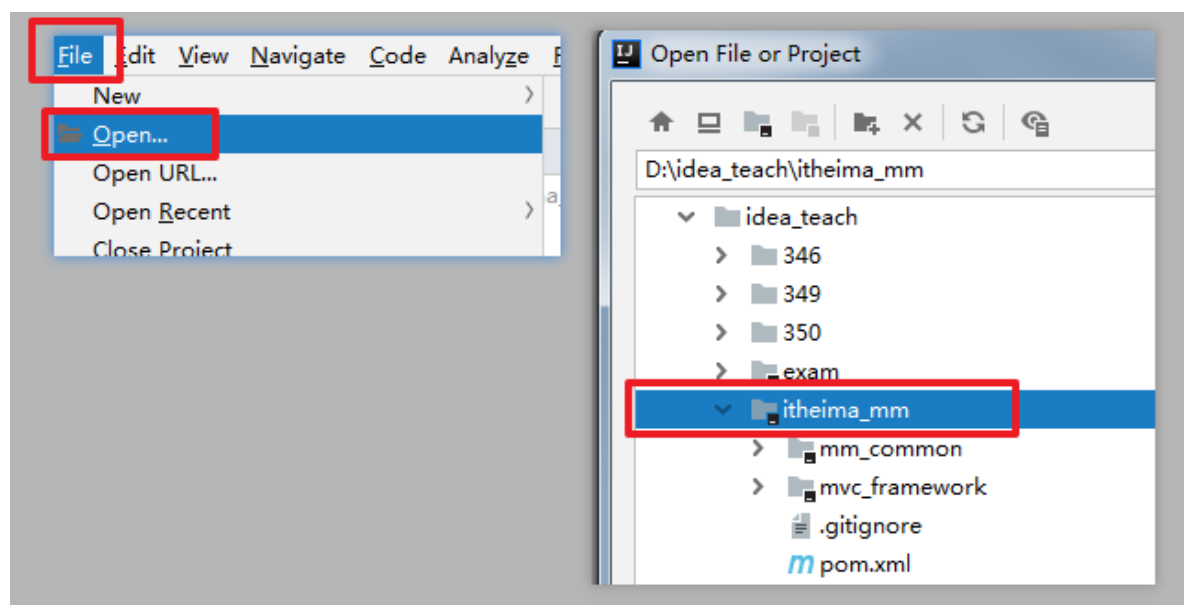
## 3.2 初始化工程

从资料中获取项目工程初始化代码，如图所示：

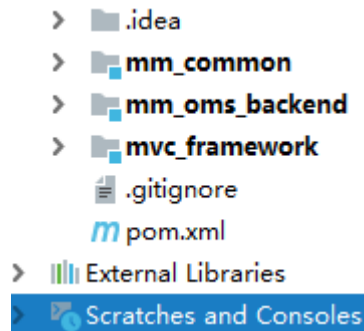


使用IDEA打开项目，而非导入项目。

如图操作：



打开完成后，如图所示：



注意项目初始化过程中，需要联网下载maven依赖库，需要网络可用。

### 3.3 关于父工程

父工程负责所有子模块需要的三方库的依赖管理，本工程涉及到Web开发、mysql数据库连接、mybatis开发、dom解析、HTTP网络编程、Redis开发、微信数据解析、文件上传及常规log4j、slf4j、lombok、fastjson、tomcat7插件等工具框架的使用。

如图是工程需要的所有库的版本信息：

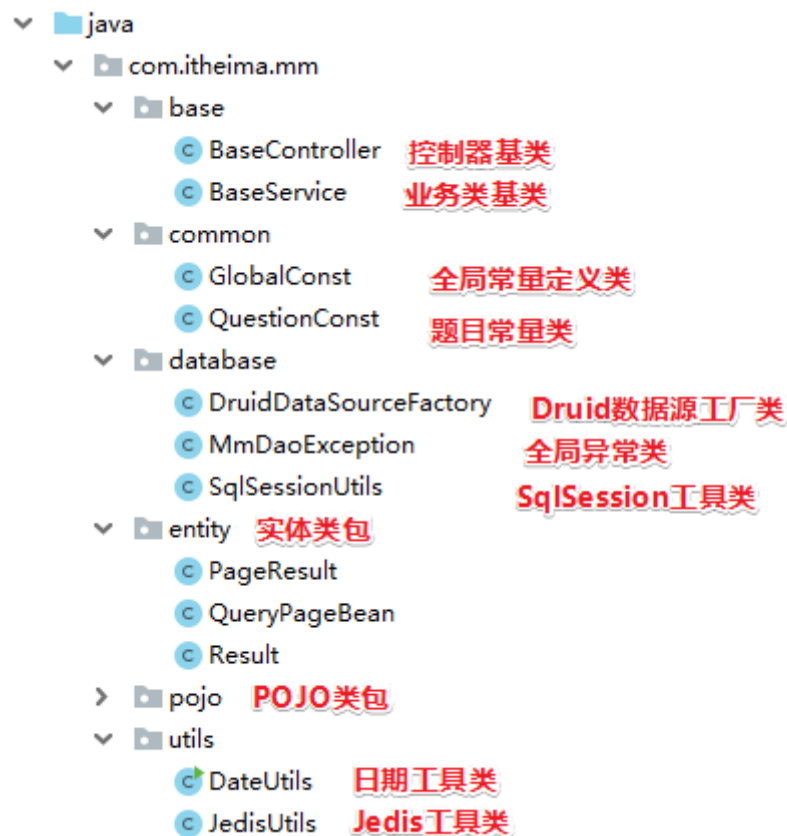
```
<properties>
  <junit.version>4.12</junit.version>
  <javax.servlet-api.version>3.1.0</javax.servlet-api.version>
  <jstl.version>1.2</jstl.version>
  <fastjson.version>1.2.47</fastjson.version>
  <mysql-connector-java.version>5.1.38</mysql-connector-java.version>
  <druid.version>1.1.10</druid.version>
  <mybatis.version>3.4.5</mybatis.version>
  <commons-fileupload.version>1.3.1</commons-fileupload.version>
  <commons-io.version>2.6</commons-io.version>
  <dom4j.version>1.6.1</dom4j.version>
  <jaxen.version>1.2.0</jaxen.version>
  <okhttp.version>3.1.0</okhttp.version>
  <okio.version>1.4.0</okio.version>
  <bcprov-jdk16.version>1.45</bcprov-jdk16.version>
  <jedis.version>2.9.0</jedis.version>
  <log4j.version>1.2.17</log4j.version>
  <slf4j-api.version>1.7.25</slf4j-api.version>
  <lombok.version>1.18.8</lombok.version>
  <tomcat7-maven-plugin.version>2.2</tomcat7-maven-plugin.version>
</properties>
```

### 3.4 关于公共模块

mm\_common模块是公共模块，其作用是其他子模块共同需要的类和依赖库进行统一管理，父模块已有的资源子模块无需再次导入或依赖。

本项目后续要管理后台子模块和创建微信小程序API子模块，目前两个模块的常用POJO类、实体类、常量类、工具类、基础父类都放在公共模块中，如图所示：





另外两个模块都是基于Web工程，都需要用到自定义MVC框架mvc\_framework子模块及用于Web开发的数据库、redis、日志及其他常规工具框架，最终mm\_common模块的pom.xml如下所示：

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <!--定义项目公共常量类、工具类-->
    <parent>
        <artifactId>itheima_mm</artifactId>
        <groupId>com.itheima.mm</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>mm_common</artifactId>

    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.7</maven.compiler.source>
        <maven.compiler.target>1.7</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>

```

```

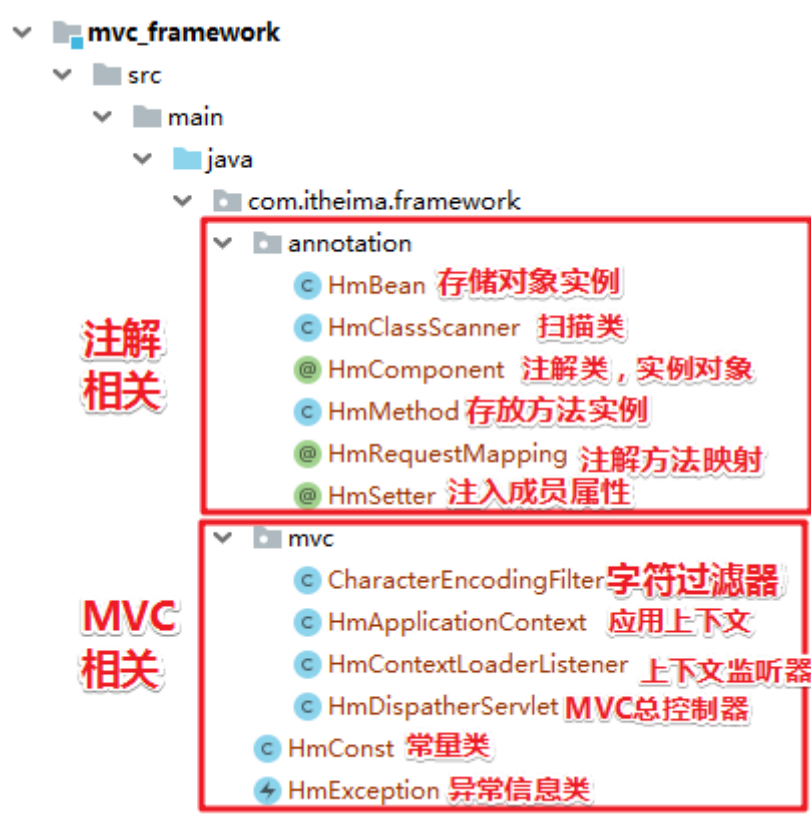
        <groupId>com.itheima.mm</groupId>
        <artifactId>mvc_framework</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
    </dependency>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>dom4j</groupId>
        <artifactId>dom4j</artifactId>
    </dependency>
    <dependency>
        <groupId>jaxen</groupId>
        <artifactId>jaxen</artifactId>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
    </dependency>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>fastjson</artifactId>
    </dependency>
    <dependency>
        <groupId>redis.clients</groupId>
        <artifactId>jedis</artifactId>
    </dependency>
</dependencies>

```

```
</project>
```

### 3.4 关于MVC框架模块

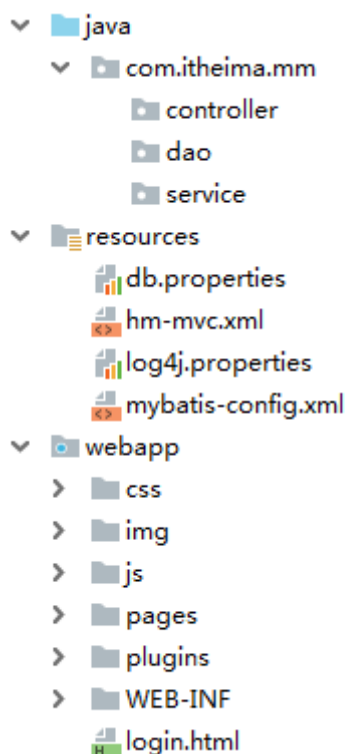
mvc\_framework模块是自定义MVC框架，该框架加入了类注解（用于实例化控制器、实例化业务类）、属性注解（把已实例化的对象注入到对象属性）、基于配置文件读取扫描包、字符编码过滤器、自定义异常、常量等元素，是一个可复用的独立MVC框架模块，如图所示：



本章节是为学习目的设计的自定义MVC框架，后续工作或商业建议使用已经成熟稳定的MVC框架产品。

### 3.5 关于管理后台模块

mm\_oms\_backend模块是基于Web骨架创建的模块，使用了自定义的MVC框架。已完成了初始化的资源及目录结构的初始化，如图所示：



- java目录  
已创建了分层包controller、dao及service，其他子包根据后续业务逐一创建。
- resources目录  
提供了mybatis配置，log4j配置及自定义框架基础配置hm-mvc.xml。
- webapp目录  
提供了后续常用的静态网页。

## 4. 开发思路与实现分析

任何功能的开发，先要通过产品原型详细此功能的业务需求，根据自己所学知识和经验构建其实现思路，梳理出实现的主次关系、数据库关系及技术要点，如果业务复杂，先把流程走通，然后再逐一实现其它功能。

功能需求和开发的基本思路明确后，其功能实现的方式有多形式，一般来讲一个功能的开发，可以从前向后写，即先去初始化前端代码，发送ajax请求，根据请求做处理，然后去写控制器Controller，之后去写业务Service,最后完成数据Dao,全部完成后一起与页面联调，这个实现过程也可以反过来完成，即从后往前写，也就是从Dao层开始写，最后完成页面请求发送与处理。



这个实现过程没有固定的模式与方式，根据不同业务特点、业务复杂及个人的编写习惯来完成即可。

以下是关于后续文档中的使用定义及框架的使用参考：

### 1. Dao接口及映射文件

为方便定义和使用，Dao接口和映射文件都放在Dao包中；

### 2. Service接口及实现类

在Service实现类中调用Dao,某些业务可能涉及多个Dao的调用，有些业务需要使用事务处理，根据实际情况来处理，通过框架提供的类注解，会提前实例化Service，并保存到容器；

**自定义框架提供了业务实现类的类注解，只需在实现类类定义时加入@HmComponent("service实例名")，容器加载时该实现类的会提前实例化，并放入HmApplicationContext容器中。**

### 3. Controller控制器

控制器中接收客户端请求，然后调用Service对象完成业务，业务完成后最终响应JSON数据给前端。

为提高自定义框架对自定义注解的扫描，控制器必须使用@HmComponent注解来声明类，通过自定义属性注解@HmSetter("service实例名")，把已经实例化的Service注入到控制器中。

当前项目Web端请求统一使用axios来完成。axios的post请求默认发送的application/json格式，而非form-data。故Controller接收数据的方式与传统方式不同。

### 4. 测试接口

Controller完成后，可使用postman请求API接口，如果业务比较简单，可直接在浏览器测试。

### 5. 编写前端代码

当前工程需要的前端页面已经完成，其中管理后台的前端页面采用vue+ElementUI实现，目前给到的原型已经完成了全部静态业务的模拟，后续找到当前业务需要的前端页面，去掉测试或逻辑代码，嵌入ajax请求即可。

## 5. 登录模块

### 5.1 需求分析

管理后台需要是注册或授权的用户方可进入，在登录页面输入相应的用户名及密码，信息正确登录到后台系统，信息错误，进行相应的提示（比如用户不正确、密码错误等信息）。进入系统后在主页右上角显示用户名，然后点击下方退出，方可退出系统。

如图所示：



### 5.2 实现思路

前端工程师提供的原型已完成页面链接，本项目是基于前后端分离，需要后台提供登录与退出接口，然后前端页面发送ajax请求，根据服务端返回的JSON数据，前端进行页面跳转。

当前业务属于用户业务，故后台提供用户控制器类、业务类及Dao实现，数据库方面仅需要t\_user表即可完成用户身份验证。

用户名显示在主页，可以在用户登录后，把用户信息存储到浏览器缓存，到主页再获取此信息。

在用户登录期间，为保存用户信息，需要登录成功之后，创建Session对象，并把用户信息存入上下文，用户退出时，再销毁此会话。

### 5.3 登录功能

在控制器中，调用Service获取用户对象，然后判断用户是否为空，如果不为空，判断密码是否与前端密码一致。如果一致登录成功，不一致登录失败。如果登录成功把用户信息存入Session对象。

#### 5.3.1 DAO接口及映射文件

以下文件都是存放在com.itheima.mm.dao包中

##### 1. 创建UserDao接口

需要具体判断出用户名不正确，密码错误，故Dao接口提供一个根据用户获取用户信息的接口。

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/11
 * @description : 用户Dao
 * @version: 1.0
 */
public interface UserDao {
```

```

/**
 * 根据用户获取用户信息
 * @param username
 * @return
 */
User findByUsername(String username);
}

```

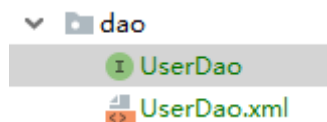
## 2. 创建UserDao.xml映射文件

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//com.itheima.mm.database.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.itheima.mm.dao.UserDao">
    <select id="findByUsername" resultType="com.itheima.mm.pojo.User">
        SELECT * FROM t_user WHERE username = #{username}
    </select>
</mapper>

```

实现后，工程目录结构如下：



## 5.3.2 Service接口及实现类

接口文件存放在com.itheima.mm.service包中，接口实现类存放在com.itheima.mm.service.impl包中。

### 1. UserService接口

```

/**
 * @author : seanyang
 * @date : Created in 2019/8/11
 * @description : 用户业务接口
 * @version: 1.0
 */
public interface UserService {
    /**
     * 根据用户名，获取用户信息
     * @param username
     * @return
     */
    User findByUsername(String username);
}

```

### 2. UserServiceImpl实现类

通过类注解HmComponent("userService")，让框架提前加载该类实例，实例标识userService。

Service调用Dao，需要使用mybatis框架的SqlSession,为便于操作，提供了BaseService基类，定义了SqlSession的常规操作，故后续的业务实现类都继承这个父类。

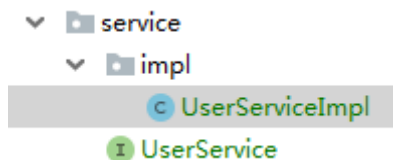
**注意：SqlSession一定及时关闭，否则造成数据库连接不能及时关闭，以免造成内存溢出。**

```

/**
 * @author : seanyang
 * @date : Created in 2019/8/11
 * @description : 用户业务实现类
 * @version: 1.0
 */
@HmComponent("userService")
@Slf4j
public class UserServiceImpl extends BaseService implements UserService {
    @Override
    public User findByUsername(String username) {
        log.info("UserServiceImpl findByUsername:{}",username);
        SqlSession sqlSession = getSession();
        UserDao userDao = getDao(sqlSession,UserDao.class);
        User user = userDao.findByUsername(username);
        closeSession(sqlSession);
        return user;
    }
}

```

实现后，工程结构如图所示：



### 5.3.3 控制器类及Service调用

使用类注解，提前加载子控制器到容器，使用属性注解把service实例注入到控制器，注意service名称一定与之前的定义一样，否则注入失败。使用方法注解，为当前login方法设置唯一访问标识（可参考接口文档中的路径名称）。

登录成功后以全局常量SESSION\_KEY\_USER为键，User对象为值，存入Session。

客户端发送的请求数据，使用了父类工具方法parseJSON2Object把其数据转换为POJO类对象。

完成业务后，返回客户端JSON数据，统一使用Result类，由父类printResult负责完成转换。

```

/**
 * @author : seanyang
 * @date : Created in 2019/8/9
 * @description : 用户控制器
 * @version: 1.0
 */
@HmComponent
@Slf4j
public class UserController extends BaseController {

    @HmSetter("userService")
    private UserService userService;

    /**
     * 登录接口
     * 根据用户名获取用户对象
     * 判断用户信息是否正确

```

```

    * 登录成功, 把用户信息存入session对象
    * 返回JSON结果给前端
    * @param request
    * @param response
    * @throws ServletException
    * @throws IOException
    */
    @RequestMapping("/user/login")
    public void login(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        log.info("UserController login...");
        // 把表单数据封装到实体对象中
        User loginForm = parseJSON2Object(request, User.class);
        log.debug("loginForm:{}", loginForm);
        // 根据用户名获取用户对象
        User user = userService.findByUsername(loginForm.getUsername());
        // 判断用户对象是否为空
        if(user == null){
            printResult(response, new Result(false, "用户名不正确"));
            return;
        }
        // 判断用户密码是否一致
        if (user.getPassword().equals(loginForm.getPassword())){
            // 把用户对象放入session中
            request.getSession(true).setAttribute(SESSION_KEY_USER, user);
            // 响应JSON
            printResult(response, new Result(true, "登录成功"));
        }else{
            printResult(response, new Result(false, "密码错误"));
        }
    }
}

```

- Result类

```

@Data
@AllArgsConstructor
public class Result implements java.io.Serializable {
    private boolean flag; //执行结果, true为执行成功 false为执行失败
    private String message; //返回结果信息
    private Object result; //返回数据
    public Result(boolean flag, String message){
        this.flag = flag;
        this.message = message;
    }
}

```

如果result为null, JSON格式将不显示。

- BaseController中的printResult

```

public class BaseController {
    public void printResult(HttpServletResponse response, Object obj) throws IOException {
        response.setContentType("application/json; charset=utf-8");
        JSON.writeJSONString(response.getWriter(), obj);
    }
}

```

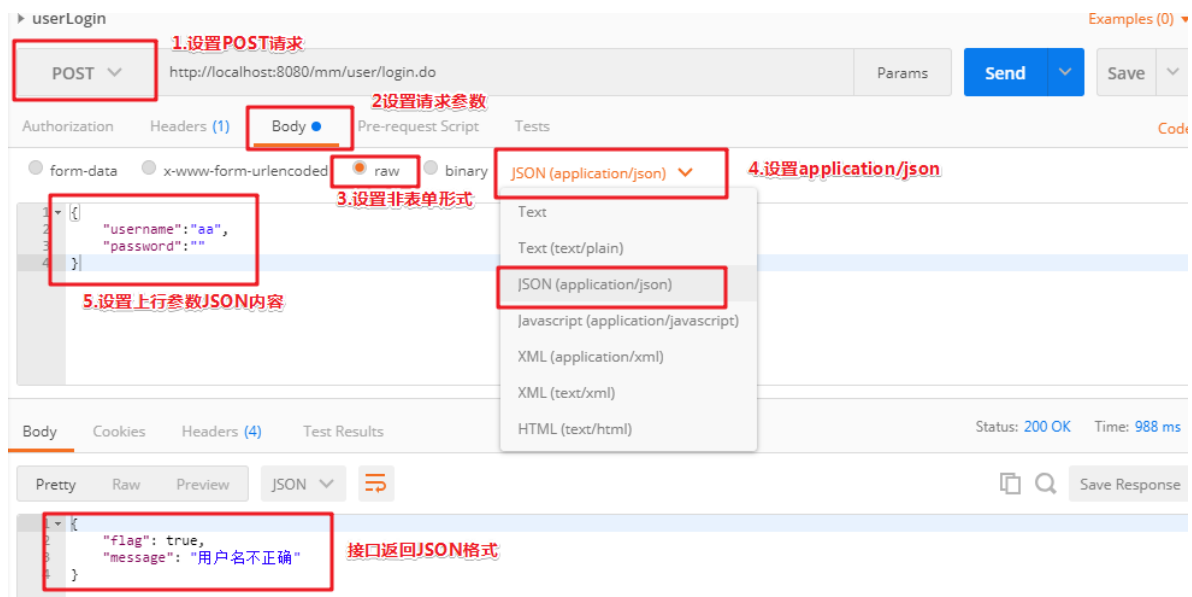


最终是使用fastjson框架的JSON类的工具方法来自动转换，需要提供输出流及要转换的对象。

完成以上代码后，可以整体编译与运行。

### 5.3.4 测试接口

测试登录接口是否正常使用，使用postman测试，由于客户端请求是application/json，故后续客户端post请求，统一按如图所示设置：



### 5.3.4 编写前端代码

在login.htm页面中已经完成模拟登陆，找到提交按钮的事件处理onSubmit方法，去掉模拟实现，嵌入axios请求，根据请求做相应的处理。

```
// 封装请求参数
var formData = {
  username: this.form.userName,
  password: this.form.pwd
};
// 去掉模拟跳转
//window.location.href = "pages/index.html";
// 发送axios请求
axios.post( app_path+"/user/login.do", formData).then((response)=>{
  console.log(response);
  if(response.data.flag){
    // 把用户名存储到当前浏览器缓存中
    sessionStorage.setItem("userName", this.form.userName);
    // 调整到主页
    window.location.href = "pages/index.html";
  }else{
    // 登录失败提示，直接显示后台返回的错误信息
    this.$message.error(response.data.message);
  }
});
```

编写完成后，无需重启服务，直接在浏览器中测试。

## 5.4 退出功能

退出功能不需要开发Dao和服务，只需要修订Controller，加入退出接口，退出需要销毁会话即可。

### 5.4.1 修订用户控制器

```
/**
 * 退出登录
 * 判断会话是否有效
 * 会话有效销毁回话
 * 响应JSON
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
@RequestMapping("/user/logout")
public void logout (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException{
    if( request.getSession(false) != null){
        request.getSession(false).invalidate();
    }
    printResult(response, new Result(true, "退出成功"));
}
```

### 5.4.2 编写前端代码

在pages/index.html中，找到logout方法，去掉模拟实现，加入axios请求，实现如下：

```
logout(){
    //window.location.href = "../login.html";
    axios.get(app_path+"/user/logout.do").then((response)=>{
        // 跳转到登录页面
        window.location.href = "../login.html";
        // 清除缓存
        sessionStorage.clear();
    });
}
```

## 6. 学科管理模块

### 6.1 需求分析

- 学科管理模块，需要完成学科列表展示、新增、更新、删除四个功能；
- 学科列表需要展示学科创建者，故创建的每个学科，需要关联当前用户ID；
- 学科列表需要展示管理的题目数量、标签数量、二级目录数量，这些查询需要嵌入子查询，开始可以先写固定数值，等调试成功后，再细化数值。
- 列表展示需要分页显示，每页显示10条记录；
- 删除学科，如果学科下已有数据，不能删除该学科；
- 新增、更新学科后刷新当前列表。

### 6.2 实现思路

构建学科控制器，为每个业务创建一个接口方法，根据接口文档，设置访问路径。根据功能分析，先实现新增学科，然后显示列表，在列表的基础上进行更新与删除操作。

学科业务相关的表有t\_course、t\_catalog、t\_tag，学科表与学科目录、学科标签表都是1对多的关系。

## 6.3 新增学科

### 6.3.1 Dao接口及映射文件

- CourseDao接口  
返回值为0，插入失败。

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/11
 * @description : 学科Dao接口
 * @version: 1.0
 */
public interface CourseDao {
    /**
     * 添加学科
     * @param course
     */
    Integer addCourse(Course course);
}
```

- CourseDao.xml映射文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.itheima.mm.dao.CourseDao">
    <insert id="addCourse">
        INSERT INTO t_course (name, create_date, is_show, user_id)
        VALUES (#{name},#{createDate},#{isShow},#{userId})
    </insert>
</mapper>
```

### 6.3.2 Service接口及实现类

- CourseService接口

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/11
 * @description : 学科标签业务接口
 * @version: 1.0
 */
public interface CourseService {
    /**
     * 添加学科
     * @param course
     */
    void addCourse(Course course);
}
```

- CourseServiceImpl实现类

```

/**
 * @author : seanyang
 * @date : Created in 2019/8/11
 * @description : 学科业务实现类
 * @version: 1.0
 */
@Slf4j
@Component("courseService")
public class CourseServiceImpl extends BaseService implements CourseService {
    @Override
    public void addCourse(Course course) {
        log.info("addCourse... course:{}",course);
        SqlSession sqlSession = getSession();
        try{
            CourseDao courseDao = getDao(sqlSession,CourseDao.class);
            Integer result = courseDao.addCourse(course);
            log.info("result:{}",result);
            if (result == 0){
                throw new MmDaoException("添加学科失败");
            }
            commitAndCloseSession(sqlSession);
        }catch(MmDaoException e){
            closeSession(sqlSession);
            log.error("addCourse",e);
            throw new MmDaoException(e.getMessage());
        }
    }
}

```

### 6.3.3 控制器类及Service调用

```

/**
 * @author : seanyang
 * @date : Created in 2019/8/11
 * @description : 学科控制器
 * @version: 1.0
 */
@Component
@Slf4j
public class CourseController extends BaseController {

    @HmSetter("courseService")
    private CourseService courseService;

    /**
     * 添加学科
     * 获取表单数据封装到POJO对象 (Course)
     * 初始化表单数据
     * 设置创建日期
     * 设置当前用户ID (从Session中获取对象)
     * 调用Service完成业务 (courseService.addCourse)
     * 返回JSON到前端
     * @param request
     * @param response
     * @throws ServletException
     * @throws IOException
     */
    @RequestMapping("/course/add")

```

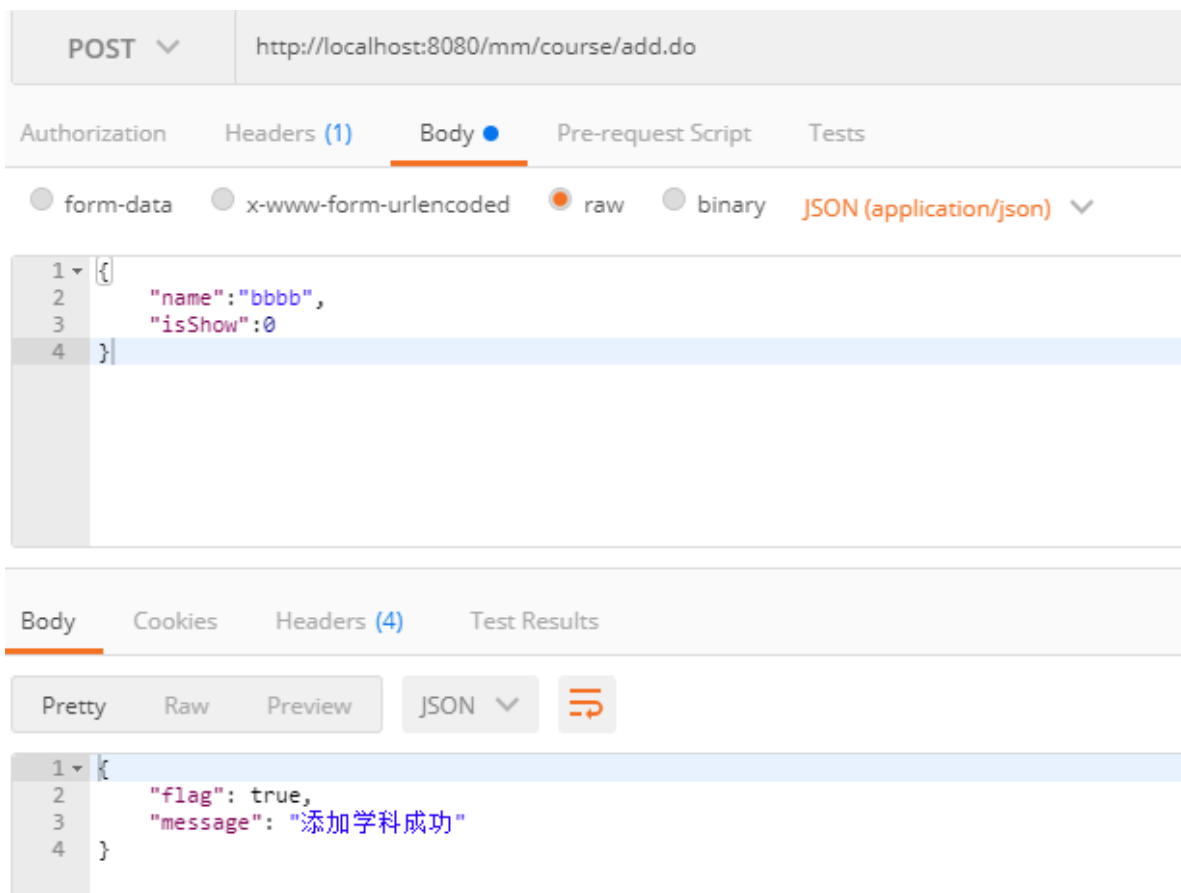
```

public void addCourse (HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException{
    try{
        Course course = parseJSON2Object(request, Course.class);
        log.info("addCourse,{}", course);
        // 设置创建日期
        course.setCreateDate(DateUtils.parseDate2String(new Date()));
        // 获取当前用户的用户信息
        User user = getSessionUser(request, GlobalConst.SESSION_KEY_USER);
        if (user != null) {
            course.setUserId(user.getId());
        } else {
            // 调试时，默认是管理员
            course.setUserId(1);
        }
        courseService.addCourse(course);
        printResult(response, new Result(true, "添加学科成功"));
    } catch (RuntimeException e) {
        log.error("addCourse", e);
        printResult(response, new Result(false, "添加学科失败,"+e.getMessage()));
    }
}
}

```

### 6.3.4 测试接口

在postman中，如图所示进行测试：



### 6.3.5 编写前端代码

在pages/courseList.html页面中，找到handleCreateConfirm()函数，去掉模拟代码，加入axios请求。

```

handleCreateConfirm() {
  this.$refs['form'].validate((valid) => {
    if (valid) {
      let params = this.form;
      console.log("学科添加请求参数: ");
      console.log(params);
      // 发送请求
      //this.$message.success("添加成功");
      //this.dialogFormVisible = false;
      //this.getList();
      // 发送请求
      axios.post(app_path+"/course/add.do",params).then((response)=>{
        if(response.data.flag){
          this.$message.success(response.data.message);
          this.dialogFormVisible = false;
        }else{
          this.$message.error(response.data.message);
        }
      }).finally(()=>{
        this.getList();
      });
    }
  });
}

```

完成后，去数据库查看是否添加成功。

## 6.4 学科列表展示

学科列表分页展示，数据展示使用，分页插件使用。分页插件固定每页显示条数，向后端传递当前页码和每页记录数，后端接口返回总记录数和数据集，总记录数传递给分页插件，分页插件自动完成分页，数据集传递给表格组件。

### 6.4.1 修订Dao接口及映射文件

当前完成的是学科列表分页展示，分页查询需要使用相同的条件查询数据集同时要查询其条件下的总记录数，另外分页查询还有兼顾其他参数的条件查询，故使用分页查询的实体类QueryPageBean封装查询参数。

- QueryPageBean类

```

@Data
public class QueryPageBean implements Serializable{
    private Integer currentPage;    // 页码
    private Integer pageSize;      // 每页记录数
    private Map queryParams;       // 查询条件
    private Integer offset;        // 分页查询，开始记录下标

    /**
     * 获取分页起始记录位置
     * 根据分页页数，计算limit记录
     * @return
     */
    public Integer getOffset(){
        return (currentPage-1)*pageSize;
    }
}

```

```
}
```

- 为CourseDao增加的如下方法:

```
/**
 * 分页获取学科列表
 * @param queryPageBean
 * @return
 */
List<Course> selectListByPage(QueryPageBean queryPageBean);

/**
 * 基于条件获取学科记录总数
 * @param queryPageBean
 * @return
 */
Long selectTotalCount(QueryPageBean queryPageBean);
```

- 为CourseDao.xml映射文件，增加如下实现:

```
<sql id="select_where">
  <where>
    <if test="queryParams.name !=null and queryParams.name.length>0 ">
      and name like "%#{queryParams.name}%"
    </if>
    <if test="queryParams.status != null ">
      and is_show = #{queryParams.status}
    </if>
  </where>
</sql>
<select id="selectListByPage" resultType="com.itheima.mm.pojo.Course">
  SELECT tc.id,name,tc.create_date as createDate,tc.is_show as isShow,
  0 as catalogQty,
  0 as tagQty,
  0 as questionQty ,
  tu.username as creator
  FROM t_course tc join t_user tu on tc.user_id = tu.id
  <include refid="select_where"/>
  limit #{offset},#{pageSize}
</select>
<select id="selectTotalCount" resultType="java.lang.Long">
  SELECT count(*) FROM t_course
  <include refid="select_where"/>
</select>
```

### 6.4.2 修订Service接口及实现类

在CourseService接口中，加入分页业务，返回值使用PageResult实例类。

- PageResult类(已在mm\_common包中定义)：

```

/**
 * @author : seanyang
 * @date : Created in 2019/8/12
 * @description : 封装分页返回结果
 * @version: 1.0
 */
@Data
@AllArgsConstructor
public class PageResult implements Serializable{
    private Long total; //总记录数
    private List rows; //当前页结果
}

```

total代表总记录数，rows代表数据集。

- CourseService接口

```

/**
 * 分页获取学科列表
 * @param queryPageBean
 * @return
 */
PageResult findListByPage(QueryPageBean queryPageBean);

```

- CourseServiceImpl实现类

```

@Override
public PageResult findListByPage(QueryPageBean queryPageBean) {
    log.info("findListByPage:{}", queryPageBean);
    SqlSession sqlSession = getSession();
    CourseDao courseDao = getDao(sqlSession, CourseDao.class);
    // 获取分页数据集
    List<Course> courseList = courseDao.selectListByPage(queryPageBean);
    log.debug("findListByPage courseList:{}", courseList);
    // 获取记录总数
    Long totalCount = courseDao.selectTotalCount(queryPageBean);
    closeSession(sqlSession);
    return new PageResult(totalCount, courseList);
}

```

### 6.4.3 修订控制器类及Service调用

在CourseController类中，添加分页web方法，根据API接口文档定义访问路径

```

/**
 * 分页获取学科列表
 * 接收查询参数封装到QueryPageBean
 * 如果查询参数为null,初始化默认
 * 调用Service获取分页结果PageResult
 * 响应JSON数据
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */

```



```

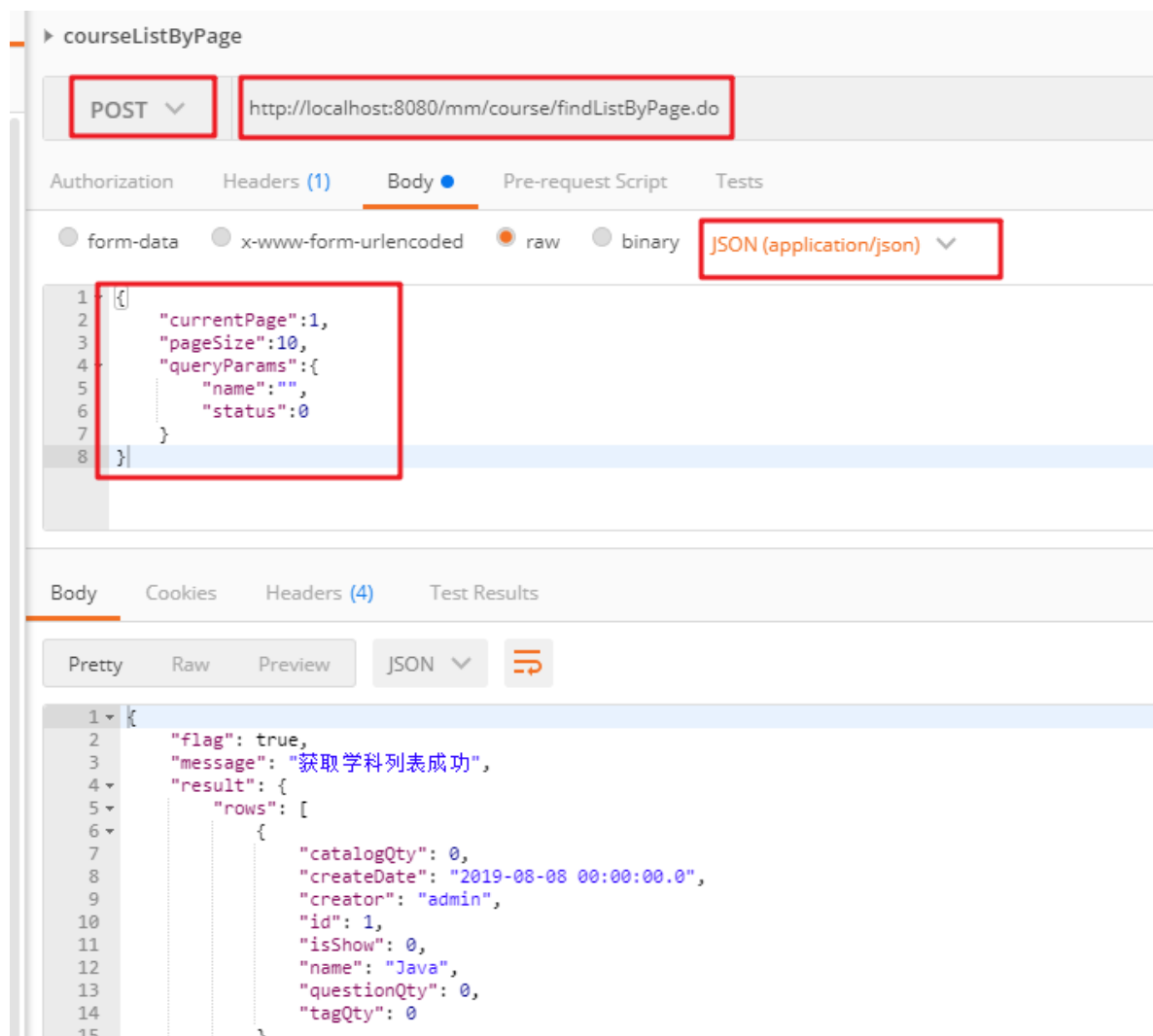
@HmRequestMapping("/course/findListByPage")
public void findListByPage (HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    QueryPageBean pageBean = parseJSON2Object(request,QueryPageBean.class);
    if (pageBean == null){
        pageBean = new QueryPageBean();
        pageBean.setCurrentPage(1);
        pageBean.setPageSize(10);
    }

    log.info("questionList pageBean:{",pageBean);
    PageResult pageResult = courseService.findListByPage(pageBean);
    printResult(response,new Result(true,"获取学科列表成功",pageResult));
}

```

#### 6.4.4 测试接口

使用postman测试分页获取学科列表，上行参数可参考接口文档。



#### 6.4.5 编写前端代码

在学科分页列表页面`courseList.html`中找到`getList()`方法，去掉模拟代码，加入`axios`请求。

```

// 发送请求获取数据
//this.pagination.total = response.data.result.total;
//this.items = response.data.result.rows;
// 发送请求获取数据
axios.post(app_path+"/course/findListByPage.do",params).then((response)=>{
    console.log("学科分页列表返回数据: ");
    console.log(response);
    this.pagination.total = response.data.result.total;
    this.items = response.data.result.rows;
    this.loading = true;
});

```

在浏览器，测试分页列表展示效果，并输入查询条件测试有条件下的查询是否正常。

### 6.4.6 完善Dao查询条件

分页列表查询已经完成，但是关于二级目录、标签、题目数量目前显示都是默认固定数值，现在修订查询的SQL语句，完善查询业务。可以使用嵌套子查询，在查询一条语句时，可以针对某一数值，进行嵌套查询，修订CourseDao.xml映射文件中的selectListByPage，完成如下：

```

<select id="selectListByPage" resultType="com.itheima.mm.pojo.Course">
    SELECT tc.id,name,tc.create_date as createDate,tc.is_show as isShow,
    <!--根据学科ID，统计学科目录数量-->
    (select count(*) from t_catalog WHERE course_id = tc.id) as catalogQty,
    <!--根据学科ID，统计学科标签数量-->
    (select count(*) from t_tag WHERE course_id = tc.id) as tagQty,
    <!--
        根据学科ID，查询隶属学科目录，在题目表中统计隶属这些学科目录的题目数量
        题目表中也冗余设计了学科ID，可以直接用学科ID，统计题目数量
    -->
    (select count(*) from t_question WHERE catalog_id in (select id from
t_catalog where course_id = tc.id)) as questionQty ,
    tu.username as creator
    FROM t_course tc join t_user tu on tc.user_id = tu.id
    <include refid="select_where"/>
    limit #{offset},#{pageSize}
</select>

```

```

<select id="selectListByPage" resultType="com.itheima.mm.pojo.Course">
    SELECT tc.id,name,tc.create_date as createDate,tc.is_show as isShow,
    <!--根据学科ID，统计学科目录数量-->
    (select count(*) from t_catalog WHERE course_id = tc.id) as catalogQty,
    <!--根据学科ID，统计学科标签数量-->
    (select count(*) from t_tag WHERE course_id = tc.id) as tagQty,
    <!--
        根据学科ID，查询隶属学科目录，在题目表中统计隶属这些学科目录的题目数量
        题目表中也冗余设计了学科ID，可以直接用学科ID，统计题目数量
    -->
    (select count(*) from t_question WHERE catalog_id in (select id from t_catalog where course_id = tc.id)) as questionQty,
    tu.username as creator
    FROM t_course tc join t_user tu on tc.user_id = tu.id
    <include refid="select_where"/>
    limit #{offset},#{pageSize}
</select>

```

## 6.5 更新学科

更新学科业务不复杂，只要保证数据上线正确即可。

### 6.5.1 修订Dao接口及映射文件

- CourseDao接口，增加如下方法：

```
/**
 * 更新学科信息
 * @param course
 */
Integer updateCourse(Course course);
```

- CourseDao.xml映射文件，增加如下实现：

```
<update id="updateCourse">
    UPDATE t_course
    SET name = #{name},is_show = #{isShow}
    where id = #{id}
</update>
```

### 6.5.2 修订Service接口及实现类

- CourseService接口，增加如下方法：

```
/**
 * 更新学科
 * @param course
 */
void updateCourse(Course course);
```

- CourseServiceImpl实现类，增加如下实现：

```
@Override
public void updateCourse(Course course) {
    log.info("updateCourse... course:{}",course);
    SqlSession sqlSession = getSession();
    try{
        CourseDao courseDao = getDao(sqlSession,CourseDao.class);
        Integer result = courseDao.updateCourse(course);
        if (result == 0){
            throw new MmDaoException("更新学科失败，Id不存在");
        }
        commitAndCloseSession(sqlSession);
    }catch(MmDaoException e){
        closeSession(sqlSession);
        log.error("updateCourse",e);
        throw new MmDaoException(e.getMessage());
    }
}
```

### 6.5.3 修订控制器类及Service调用

在CourseController类中，添加分页web方法，根据API接口文档定义访问路径。

```
/**
 * 更新学科
 * 获取表单数据封装到POJO对象（Course）
```

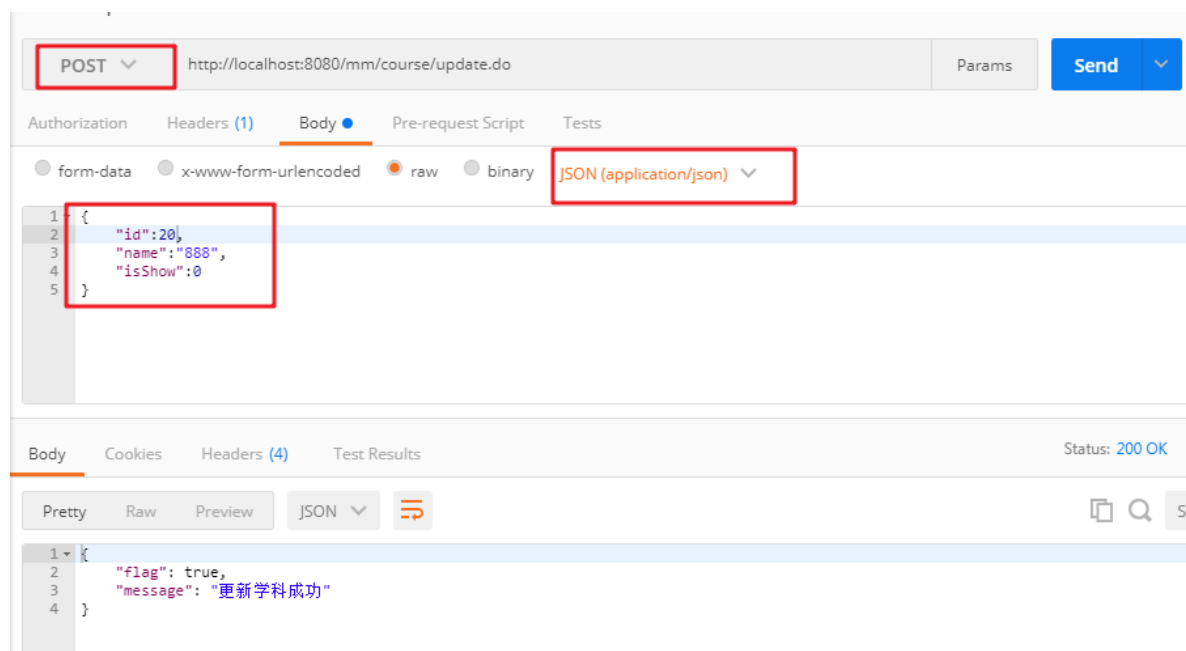
```

* 调用Service完成业务（courseService.updateCourse(course)）
* 返回JSON到前端
* @param request
* @param response
* @throws ServletException
* @throws IOException
*/
@RequestMapping("/course/update")
public void updateCourse(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException{
    try{
        Course course = parseJSON2Object(request, Course.class);
        courseService.updateCourse(course);
        printResult(response, new Result(true, "更新学科成功"));
    } catch (RuntimeException e){
        log.error("updateCourse", e);
        printResult(response, new Result(false, e.getMessage()));
    }
}

```

### 6.5.4 测试接口

使用postman测试更新业务接口，上行参数可参考接口文档。



### 6.5.5 编写前端代码

在courseList.html中，找到handleUpdateConfirm方法，加入axios请求。

```

handleUpdateConfirm() {
    this.$refs['form'].validate((valid) => {
        if (valid) {
            let params = this.form;
            console.log("学科更新请求参数: ");
            console.log(params);
            //this.$message.success("更新成功");
            //this.dialogFormVisible = false;
            axios.post(app_path+"/course/update.do", params).then((response)=>{
                if(response.data.flag){
                    this.$message.success(response.data.message);
                }
            });
        }
    });
}

```

```
        this.getList();  
    }else{  
        this.$message.error(response.data.message);  
    }  
    this.dialogFormVisible = false;  
  })  
  }  
});  
}
```

完成后，测试前端调用后台接口是否正常。