

第2天 题库管理

今日目标

- 掌握基础题库列表展示
- 初始化学科列表
- 初始化行业方向
- 初始化公司列表
- 保存题目数据
- 保存题目选项
- 保存标签信息
- 保存公司及行业信息

1. 题库管理概述

1.1 需求分析

基础题库管理是普通录入人员操作的功能，新的题目先经过基础题库录入，加入精选后方可进入精选题库列表，此时题目的状态默认是待审核状态，有操作精选题库权限的用户，可以审核题目。

基础题库与精选题库，从数据库上都来源与同一张主表t_question,只是题目的类型有所不同。区分两种题库主要是通过is_classic字段来区分，is_classic为0是基础题目，为1是精选题目。题目状态是通过status区分，审核状态靠review_status来区分。具体题目的类型、状态及审核状态，总结如下：

题目类型（is_classic）：0 基础题目、1精选题目

题目状态（status）：0 待发布（待审核、已拒绝）、1 已发布（已审核）、2 已下架（已审核）

审核状态（review_status）：0 待审核、1 已审核、2 已拒绝

既然两种题库类型来源同一张表，在完成基础题库与精选题库列表的操作时，要考虑代码复用性。

1.2 基本思路

构建题目控制器，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。题目新增涉及表数据比较多，先展示题目列表，然后新增与编辑，然后是加入精选，最后完成精选题目列表展示。

题目业务涉及的表有t_user、t_question、t_question_item、t_company、t_industry、t_course、t_catalog、t_tag八张表。

2. 基础题库列表展示

基础题目列表中，需要展示创建者、使用次数及学科名称，故需要三表连接，列表中还需要显示使用次数，需要使用嵌套子查询统计题目被用户使用题的次数。基础题目列表需要分页，所有需要使用相同的条件查询总记录数和数据集。根据页面分析，查询需要组合的条件很多，可以使用QueryPageBean的queryParams来动态组合查询。试题编号规则采用从1000+主键ID

实现思路：

1. 实现前后端的联调，从前端发送请求到后端，即流程走通；
2. 实现无条件数据的查询与分页显示；
3. 实现带条件的查询；

2.1 新增QuestionDao接口及映射文件

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/12
 * @description : 题目Dao接口
 * @version: 1.0
 */
public interface QuestionDao {

    /**
     * 分页获取题目列表,根据是否是精选题目及条件
     * @param pageBean
     * @return
     */
    List<Question> selectIsClassicByPage(QueryPageBean pageBean);

    /**
     * 根据是否是精选及条件查询条件, 统计记录总数
     * @param pageBean
     * @return
     */
    Long selectCountIsClassicByPage(QueryPageBean pageBean);
}
```

```
<?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.QuestionDao">
    <sql id="select_where_by_page">
        <where>
            <if test="queryParams.isClassic != null ">
                AND tq.is_classic = #{queryParams.isClassic}
            </if>
            <if test="queryParams.courseId != null ">
                AND tq.course_id = #{queryParams.courseId}
            </if>
        </where>
    </sql>
    <select id="selectIsClassicByPage"
        resultType="com.itheima.mm.pojo.Question">
        SELECT tq.id,(1000+tq.id) as number,tq.name as
        courseName,type`,`subject`,tq.create_date as createDate,
        difficulty, (SELECT count(*) FROM tr_member_question tmq WHERE
        tmq.question_id = tq.id ) as usedQty, tu.username as creator,tq.`status`
        FROM t_question tq
        JOIN t_course tc ON tq.course_id = tc.id
        JOIN t_user tu ON tc.user_id = tu.id
        <include refid="select_where_by_page"/>
        order by id desc
        limit #{offset},#{pageSize}
    </select>
    <select id="selectCountIsClassicByPage" resultType="java.lang.Long">
        SELECT count(*)
        FROM t_question tq
        <include refid="select_where_by_page"/>
    </select>
</mapper>
```

```
</select>
</mapper>
```

2.2 新增QuestionService接口及实现类

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/12
 * @description : 题目业务类
 * @version: 1.0
 */
public interface QuestionService {
    /**
     * 分页获取题目列表
     * @param queryPageBean
     * @return
     */
    PageResult findByPage(QueryPageBean queryPageBean);
}
```

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/12
 * @description : 题目业务实现类
 * @version: 1.0
 */
@Component("questionService")
@Slf4j
public class QuestionServiceImpl extends BaseService implements QuestionService {
    @Override
    public PageResult findByPage(QueryPageBean queryPageBean) {
        log.info("QuestionService findByPage:{}", queryPageBean);
        SqlSession sqlSession = getSession();
        QuestionDao questionDao = getDao(sqlSession, QuestionDao.class);
        // 初始化查询参数, 为设置isClassic
        if(queryPageBean.getQueryParams() == null){
            queryPageBean.setQueryParams(new HashMap());
        }
        queryPageBean.getQueryParams().put("isClassic", 0);
        // 获取数据集
        List<Question> questionList =
            questionDao.selectIsClassicByPage(queryPageBean);
        // 获取数据记录格式
        Long totalCount = questionDao.selectCountIsClassicByPage(queryPageBean);
        closeSession(sqlSession);
        return new PageResult(totalCount, questionList);
    }
}
```

2.3 新增QuestionController及服务调用

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/12
```

```

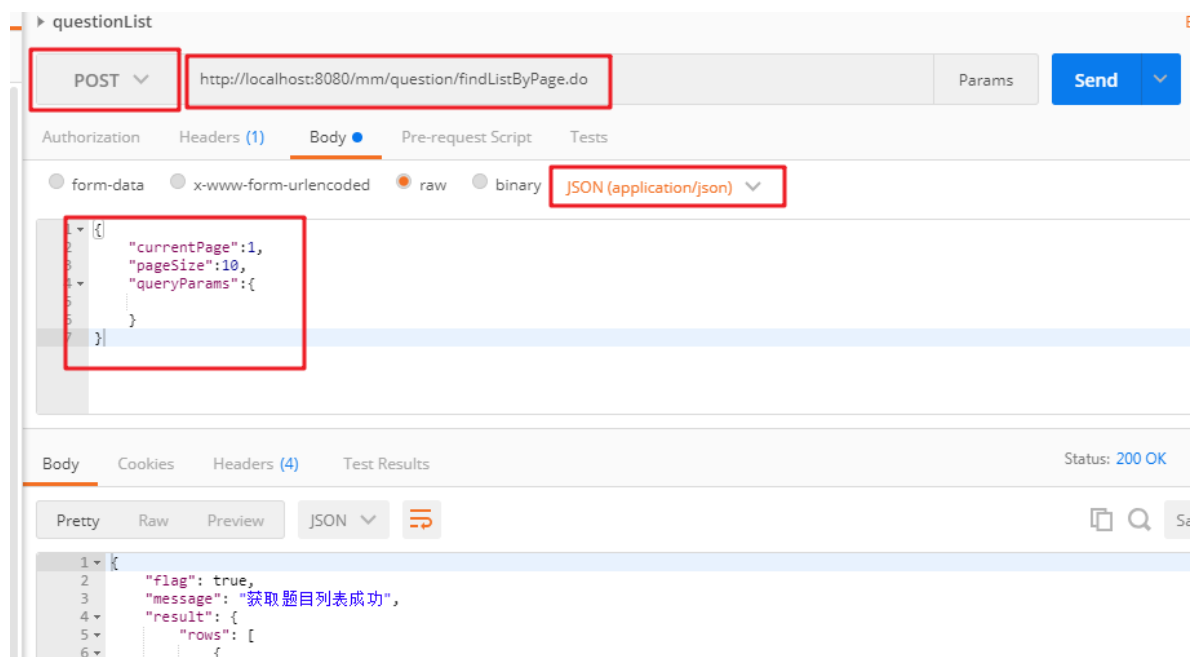
    * @description : 题目控制器
    * @version: 1.0
    */
    @HmComponent
    @Slf4j
    public class QuestionController extends BaseController {

        @HmSetter("questionService")
        private QuestionService questionService;

        /**
         * 获取题目列表
         * @param request
         * @param response
         * @throws ServletException
         * @throws IOException
         */
        @HmRequestMapping("/question/findListByPage")
        public void questionList (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);
            printResult(response, new Result(true, "获取题目列表成功", questionService.findByPage(pageBean)));
        }
    }

```

2.4 测试接口



2.5 编写前端代码

在`questionBasicList.html`中，找到`getList()`方法，去掉模拟代码，加入`axios`请求。

```
//t.pagination.total = data.result.total;
//t.items = data.result.rows;
//t.loading = true;
// 发送axios请求
axios.post(app_path+"/question/findListByPage.do",params).then((response)=>{
    if(response.data.flag){
        //this.$message.success(response.data.message);
        let result = response.data.result;
        t.pagination.total = result.total;
        t.items = result.rows;
        t.loading = true;
    }else{
        this.$message.error(response.data.message);
    }
})
```

3. 题目新增与编辑分析

3.1 功能分析

新增题目是本阶段比较复杂的业务，新增题目页面需要先初始化页面的数据（学科、学科目录及标签、公司、省市列表、行业方向），然后再提交数据进行保存，保存数据需要涉及大约九张表的更新操作。

省市级联数据，使用前端默认数据，暂时不从后端获取。

3.2 实现步骤

需要先初始化页面数据，方可进行新增操作。故先做数据的初始化，然后再处理提交表单的操作。

1. 初始化页面数据
 1. 初始化学科（包含学科目录、学科标签列表）
 2. 初始化公司
 3. 初始化行业方向
2. 保存题目数据
3. 题目新增与编辑

4. 初始化数据

4.3 初始化学科

页面选择学科时，学科目录及学科标签列表也随之发生改变。故需要在初始化学科数据时，与之对应的学科目录及标签数据同时获取。

参考前端页面原型，分析前端需要的数据返回。

4.3.1 修订相关接口及映射文件

CourseDao类：

```
/**
 * 获取全部学科、学科目录及学科标签列表
 * 为题库录入
 * @return
 */
List<Course> selectListAll();
```

```

<resultMap id="mapForAll" type="com.itheima.mm.pojo.Course">
    <id property="id" column="id"/>
    <collection property="catalogList"
        javaType="ArrayList"
        ofType="Catalog"
        column="id"

        select="com.itheima.mm.dao.CatalogDao.selectCatalogListByCourseId"/>
    <collection property="tagList"
        javaType="ArrayList"
        ofType="Tag"
        column="id"
        select="com.itheima.mm.dao.TagDao.selectTagListByCourseId"/>
</resultMap>
<select id="selectListAll" resultMap="mapForAll">
    SELECT id,name
    FROM t_course
</select>

```

CatalogDao.xml映射文件

```

<select id="selectCatalogListByCourseId" resultType="Catalog">
    SELECT id ,name FROM t_catalog
    WHERE course_id = #{id}
</select>

```

TagDao.xml映射文件

```

<select id="selectTagListByCourseId" resultType="com.itheima.mm.pojo.Tag">
    SELECT id,name
    FROM t_tag
    WHERE course_id = #{courseId}
</select>

```

4.3.2 修订CourseService接口及实现类

为学科业务增加获取全部学科列表的业务方法。

```

/**
 * 获取全部学科及目录列表
 * 为试题录入
 * @return
 */
List<Course> findListAll();

```

```

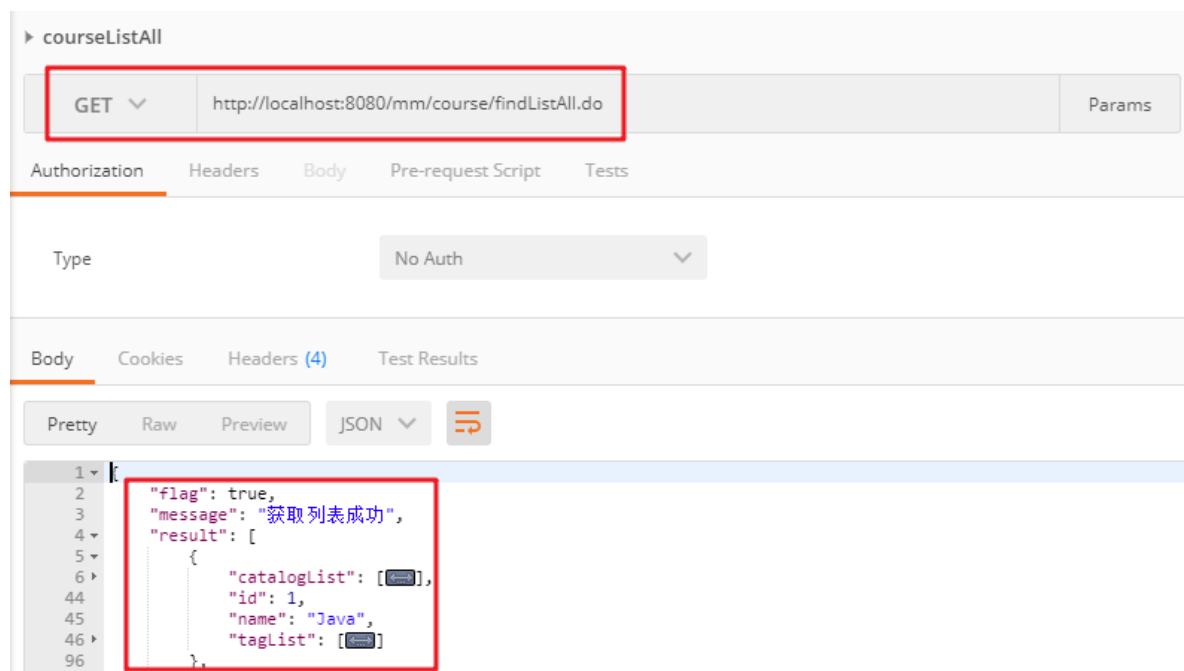
@Override
public List<Course> findListAll() {
    SqlSession sqlSession = getSession();
    CourseDao courseDao = getDao(sqlSession, CourseDao.class);
    List<Course> courseList = courseDao.selectListAll();
    closeSession(sqlSession);
    return courseList;
}

```

4.3.3 修订CourseController及服务调用

```
@RequestMapping("/course/findListAll")
public void findListForQuestion (HttpServletRequest request,HttpServletResponse
response) throws ServletException,IOException{
    try{
        List<Course> courseList = courseService.findListAll();
        printResult(response,new Result(true,"获取列表成功",courseList));
    }catch(RuntimeException e){
        log.error("findListForQuestion",e);
        printResult(response,new Result(false,"获取列表失败"));
    }
}
```

4.3.4 测试接口



4.3.5 编写前端代码

在questionEditor.html找到initCourses方法，加入axios请求。

```
// 学科下拉列表
initCourses() {
    let t = this;
    // t.courses = response.data.result;
    // 发送请求
    axios.get(app_path + "/course/findListAll.do").then((response) => {
        if (response.data.flag) {
            //this.$message.success(response.data.message);
            t.courses = response.data.result;
        } else {
            this.$message.error(response.data.message);
        }
    })
}
```

4.4 初始化行业方向

4.4.1 新增IndustryDao接口及映射文件

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/14
 * @description : 企业方向Dao接口
 * @version: 1.0
 */
public interface IndustryDao {

    /**
     * 获取全部行业列表
     * @return
     */
    List<Industry> selectListAll();

}
```

```
<?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.IndustryDao">
    <select id="selectListAll" resultType="Industry">
        SELECT id,name
        FROM t_industry
    </select>
</mapper>
```

4.4.2 新增IndustryService接口及实现类

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/15
 * @description : 企业行业方向业务接口
 * @version: 1.0
 */
public interface IndustryService {

    /**
     * 获取所有的行业方向数据
     * @return
     */
    List<Industry> findListAll();

}
```

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/15
 * @description : 企业行业方向业务实现类
 * @version: 1.0
 */
@Component("industryService")
public class IndustryServiceImpl extends BaseService implements IndustryService
{
    @Override
    public List<Industry> findListAll() {
```



```

        SqlSession sqlSession = getSession();
        IndustryDao industryDao = getDao(sqlSession, IndustryDao.class);
        List<Industry> industryList = industryDao.selectListAll();
        closeSession(sqlSession);
        return industryList;
    }
}

```

4.4.3 新增IndustryController及服务调用

```

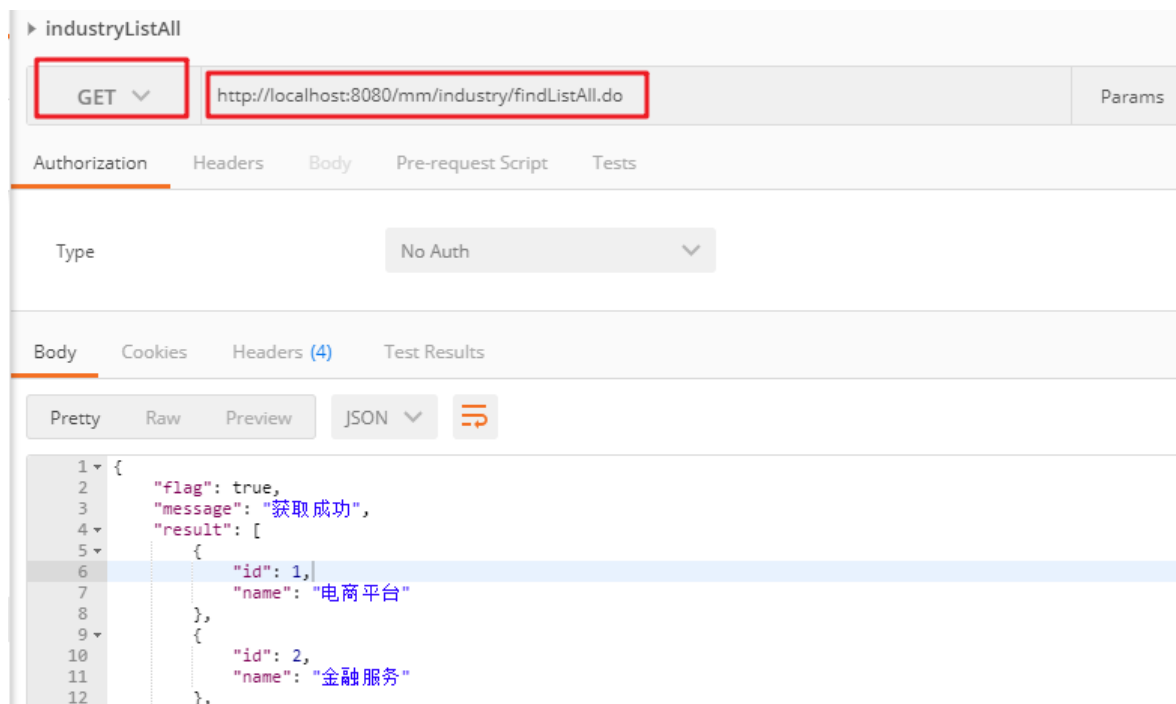
/**
 * @author : seanyang
 * @date : Created in 2019/8/15
 * @description : 企业行业方向控制器
 * @version: 1.0
 */
@Component
public class IndustryController extends BaseController {

    @HmSetter("industryService")
    private IndustryService industryService;

    @HmRequestMapping("/industry/findListAll")
    public void findListAll (HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        try{
            List<Industry> industryList = industryService.findListAll();
            printResult(response,new Result(true,"获取成功",industryList));
        }catch(RuntimeException e){
            log.error("IndustryController findListAll",e);
            printResult(response,new Result(false,"获取失败"));
        }
    }
}

```

4.4.4 测试接口



4.4.5 编写前端代码

在QuestionEditor.html页面中，找到initIndustrys方法，加入axios请求。

```
// 方向下拉列表
initIndustrys() {
  let t = this;
  //let response = {}
  // t.industrys = response.data.result;
  axios.get(app_path + "/industry/findListAll.do").then((response) => {
    if (response.data.flag) {
      //this.$message.success(response.data.message);
      t.industrys = response.data.result;
    } else {
      this.$message.error(response.data.message);
    }
  })
}
```

4.5 初始化公司

选择公司列表时，随之对应的省市数据、行业方向数据也要随之发生变化。所以提取公司数据时，需要级联提取所属城市、所属行业方向列表。

4.5.1 修订相关Dao接口及映射文件

- 新增CompanyDao接口

```
/**
 * @author : seanyang
 * @date : Created in 2019/8/14
 * @description : 公司Dao
 * @version: 1.0
 */
public interface CompanyDao {
```

```

/**
 * 获取全部公司及行业方向列表
 * 为试题输入提供源数据
 * @return
 */
List<Company> selectListAll();
}

```

- 新增CompanyDao.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.CompanyDao">
    <resultMap id="mapFormAll" type="com.itheima.mm.pojo.Company">
        <id column="id" property="id"/>
        <collection property="industryList"
            javaType="ArrayList"
            ofType="Industry"
            column="id"

select="com.itheima.mm.dao.IndustryDao.selectIndustryListByCompany"
        />
    </resultMap>
    <select id="selectListAll" resultMap="mapFormAll">
        SELECT id,short_name as shortName,city_id as cityId
        FROM t_company
    </select>
</mapper>

```

- 修订IndustryDao接口

```

/**
 * 根据公司ID，获取行业列表
 * @param id
 * @return
 */
List<Industry> selectIndustryListByCompany(Integer id);

```

- 修订IndustryDao.xml映射文件

```

<select id="selectIndustryListByCompany" resultType="Industry">
    SELECT id,name
    from tr_company_industry ci ,t_industry i
    where ci.company_id = #{id} and ci.industry_id = i.id
</select>

```

4.5.2 新增CompanyService接口及实现类

```

/**
 * @author : seanyang
 * @date : Created in 2019/8/15
 * @description : 公司业务接口
 * @version: 1.0
 */

```

```

public interface CompanyService {
    /**
     * 获取全部公司列表
     * 包含城市、方向列表
     * @return
     */
    List<Company> findListAll();
}

```

```

/**
 * @author : seanyang
 * @date : Created in 2019/8/15
 * @description : 公司业务实现类
 * @version: 1.0
 */
@HmComponent("companyService")
@Slf4j
public class CompanyServiceImpl extends BaseService implements CompanyService {
    @Override
    public List<Company> findListAll() {
        SqlSession sqlSession = getSession();
        CompanyDao companyDao = getDao(sqlSession, CompanyDao.class);
        List<Company> companyList = companyDao.selectListAll();
        closeSession(sqlSession);
        return companyList;
    }
}

```

4.5.3 修订CompanyController及服务调用

```

/**
 * @author : seanyang
 * @date : Created in 2019/8/15
 * @description : 企业控制器
 * @version: 1.0
 */
@HmComponent
@Slf4j
public class CompanyController extends BaseController {

    @HmSetter("companyService")
    private CompanyService companyService;

    /**
     * 获取所有企业数据
     * @param request
     * @param response
     * @throws ServletException
     * @throws IOException
     */
    @HmRequestMapping("/company/findListAll")
    public void findListAll (HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        try{
            log.info("findListAll");
            List<Company> companyList = companyService.findListAll();

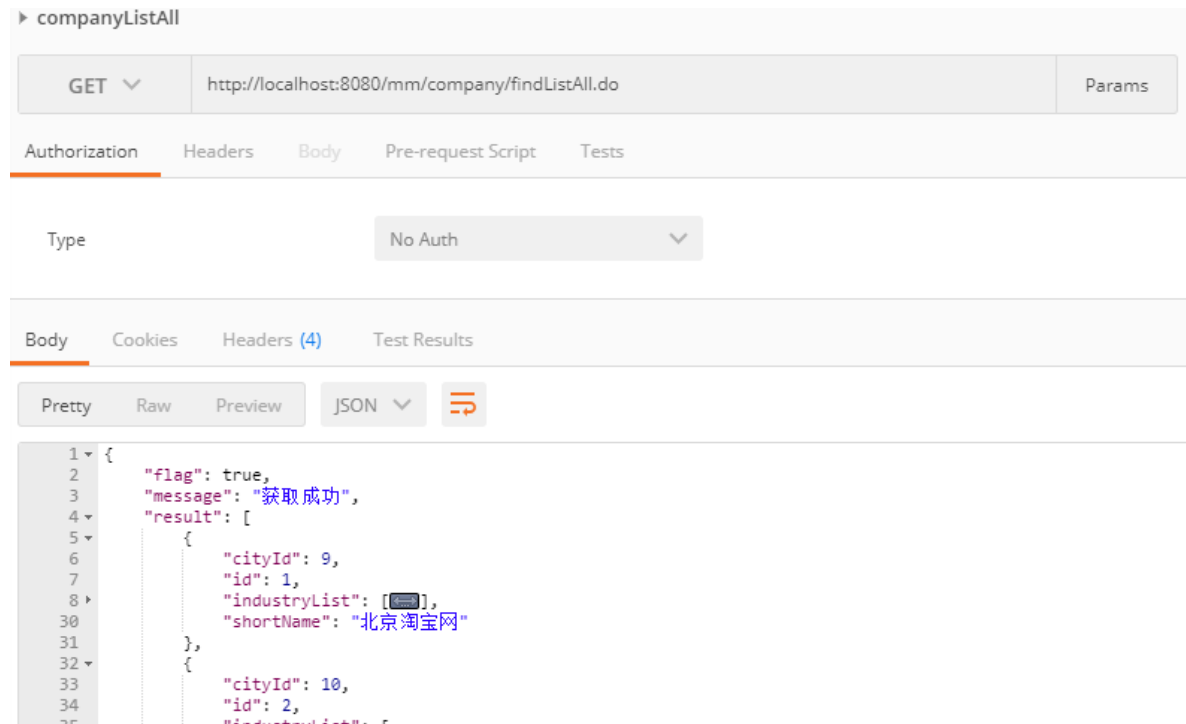
```

```

        printResult(response, new Result(true, "获取成功", companyList));
    } catch (Exception e) {
        log.error("findListAll", e);
        printResult(response, new Result(false, "获取失败"));
    }
}
}
}

```

4.5.4 测试接口



The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/mm/company/findListAll.do
- Params:** (empty)
- Authorization:** No Auth
- Body:** (selected tab)


```

{
  "flag": true,
  "message": "获取成功",
  "result": [
    {
      "cityId": 9,
      "id": 1,
      "industryList": [{"id": 1, "name": "北京淘宝网"}],
      "shortName": "北京淘宝网"
    },
    {
      "cityId": 10,
      "id": 2,
      "industryList": [{"id": 2, "name": "北京淘宝网"}]
    }
  ]
}
      
```

4.5.5 编写前端代码

在questionEditor.html页面中，找到initCompany方法，加入axios请求。

```

initCompany() {
    let t = this;
    let response = {}
    // t.companys = response.data.result;
    // 初始化试题(初始化公司时，需要在初始化公司之后再初始化)
    //this.initQuestionById()
    axios.get(app_path + "/company/findListAll.do").then((response) => {
        if (response.data.flag) {
            // this.$message.success(response.data.message);
            t.companys = response.data.result;
        } else {
            this.$message.error(response.data.message);
        }
        // 初始化试题(初始化公司时，需要在初始化公司之后再初始化)
        this.initQuestionById()
    })
}

```

5. 保存题目数据

5.1 功能分析

页面初始化已完成，用户根据页面进行数据输入，输入元素包含单选、复选、简单，其中单选、复选选项可以上传图片，如果不上传图片，页面中输入元素使用的是富文本编辑器输入文本。所谓富文本编辑器是指可以编写带有复杂样式的文本，数据保存到数据库中内容是带有html标签样式的。

选择学科及学科目录，选择不同的学科，与之对应的标签列表也会不同；

选择公司后，与之对应的省市、行业方向是自动选择，同时可以再次编辑后与题目一起提交；

可以为当前题目设置多个标签，标签列表是根据当前选择的学科变化而变化的。

5.2 实现思路

根据以上分析，保存题目数据的同时，需要更新题目数据、题目选项数据、标签数据、公司数据，故实现本小结功能，

我们先把流程业务走通，把数据从前端页面传递控制器，从控制器传递到Service，Service先保证传递到后端的数据是正确的，然后再分步完成数据的保存。

5.3 数据提交

5.3.1 修订QuestionService接口及实现类

兼顾保存与更新，方法名为addOrUpdate。

```
/**
 * 新增题目
 * @param question
 */
void addOrUpdate(Question question);
```

```
/**
 * 更新题目信息
 * 更新选项信息（更新旧选项，添加信息选项）
 * 更新标签信息 （添加新标签，删除旧关系，插入新关系）
 * 更新行业方向 （添加新行业，删除旧关系，插入新关系）
 * 更新公司信息 （更新公司信息）
 * @param question
 */
@Override
public void addOrUpdate(Question question) {
    try{
        log.info("QuestionService question:{})",question);
        // 1 添加题目信息
        // 2.更新题目选项信息
        // 3.更新标签信息
        // 4.更新公司信息(行业信息)
        // 提交保存

    }catch(MmDaoException e){
        log.error("addOrUpdate",e);
        throw new MmDaoException(e.getMessage());
    }
}
```

5.3.2 修订QuestionController及服务调用

```

/**
 * 添加基础题目
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
@RequestMapping("/question/addOrUpdate")
public void addOrUpdate(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException{
    // 获取表单数据
    try{
        Question question = parseJSON2Object(request, Question.class);
        User user = getSessionUser(request, GlobalConst.SESSION_KEY_USER);
        // 从上下文获取用户ID, 调试默认为1
        question.setUserId(user!=null?user.getId():1);
        // 保存数据
        questionService.addOrUpdate(question);
        printResult(response, new Result(true, "更新成功"));
    }catch(RuntimeException e){
        log.error("addOrUpdate", e);
        printResult(response, new Result(false, e.getMessage()));
    }
}

```

5.3.3 测试接口

questionAddOrUpdate

POST http://localhost:8080/mm/question/addOrUpdate.do

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw **JSON (application/json)** binary

```

1 {
2   "id":0,
3   "is_classic":0,
4   "courseId":2,
5   "catalogId":3,
6   "createDate":"2019-08-16 09:08:52",
7   "company": {
8     "id":2,
9     "cityId":2,
10    "industryList":[
11      {
12        "id": 1,
13        "name": "电商平台"
14      },
15      {
16        "id": 2,
17        "name": "金融服务"
18      },
19      {
20        "id": 0,
21        "name": "新增方向7"

```

请求参数数据如下：

```

{
  "id":0,
  "is_classic":0,
  "courseId":2,

```

```
"catalogId":3,
"createDate":"2019-08-16 09:08:52",
"company": {
  "id":2,
  "cityId":2,
  "industryList":[
    {
      "id": 1,
      "name": "电商平台"
    },
    {
      "id": 2,
      "name": "金融服务"
    },
    {
      "id": 0,
      "name": "新增方向7"
    }
  ]
},
"type":2,
"difficulty":3,
"subject":"test33333",
"questionItemList": [
  {
    "id":0,
    "content": "aaa",
    "isRight": 1
  },
  {
    "id":0,
    "content": "bbb",
    "isRight": 1
  },
  {
    "id":0,
    "content": "ccc",
    "isRight": 1
  },
  {
    "id":0,
    "content": "ddd",
    "isRight": 1
  }
],
"analysis": "111",
"analysisvideo": "1111",
"remark": "",
"tagList": [
  {
    "id": 1,
    "name": "String字符串"
  },
  {
    "id": 2,
    "name": "Java类与对象"
  },
  {
```



```

        "id":0,
        "name":"新增标签0"
    }
]
}

```

5.3.4 编写前端代码

在QuestionEditor.html页面中，找到createItem方法，删除模拟调用，加入axios请求。

```

/*
this.$message.success("操作成功");
// 返回到上一级
if (!this.formData.is_classic) {
    setTimeout(function () {
        window.location.href = "questionBasicList.html";
    }, 1000);
} else {
    setTimeout(function () {
        window.location.href = "questionClassicList.html";
    }, 1000);
}*/
axios.post(app_path + "/question/addOrUpdate.do", formData).then((response) => {
    if (response.data.flag) {
        this.$message.success(response.data.message);
        // 返回到上一级
        if (!this.formData.is_classic) {
            setTimeout(function () {
                // 暂时可以先注释掉，调试结束后完关闭注释
                //window.location.href = "questionBasicList.html";
            }, 1000);
        } else {
            setTimeout(function () {
                window.location.href = "questionClassicList.html";
            }, 1000);
        }
    } else {
        this.$message.error(response.data.message);
    }
})

```

5.4 数据保存

表单题目数据已传递到QuestionServiceImpl实现类，保存题目及相关的数据都需要与题目数据进行关联，所以先保存题目数据，然后再保存其他相关数据，另外这个保存操作涉及多个表单更新，需要用事务来控制数据的一致性。

5.4.1 保存题目

当前的保存兼顾新增与编辑，通过客户端是否传递id来判断是新增还是编辑。

- 修订QuestionDao接口及映射文件

```

/**
 * 添加题目信息
 * @param question
 * @return
 */
Integer add(Question question);

/**
 * 更新题目信息
 * @param question
 * @return
 */
Integer update(Question question);

```

```

<insert id="add">
    INSERT INTO t_question (subject, type, difficulty, analysis,
    analysis_video, remark, is_classic, review_status, status, create_date,
    user_id, company_id, catalog_id, course_id)
    VALUES ({subject},{type},{difficulty},{analysis}, {analysisVideo},
    #{remark}, #{isClassic}, #{reviewStatus},{status}, #{createDate}, #
    {userId}, #{companyId}, #{catalogId},{courseId})
    <selectKey order="AFTER" resultType="integer" keyProperty="id" >
        SELECT LAST_INSERT_ID()
    </selectKey>
</insert>
<update id="update">
    UPDATE t_question
        set number=#{number},subject=#{subject},type=#{type},difficulty=#{
    difficulty},analysis=#{analysis},analysis_video=#{analysisVideo},remark=#{
    remark},company_id=#{companyId},catalog_id=#{catalogId}
    WHERE id = #{id}
</update>

```

- 修订QuestionServiceImpl实现类

在此类中加入私有方法addOrUpdateQuestion，然后在保存更新主方法addOrUpdate来调用。

需要初始化题目的创建日期，发布状态为待发布，审核状态为待审核。题目数据插入成功，需要返回题目ID。

```

// 新增或更新题目
private Integer addOrUpdateQuestion(SqlSession sqlSession, Question question)
{
    log.debug("QuestionService addOrUpdateQuestion:{},question);
    QuestionDao questionDao = getDao(sqlSession, QuestionDao.class);
    Integer result = 0;
    // 设置公司ID
    question.setCompanyId(question.getCompany().getId());
    if(question.getId()==null || question.getId()==0 ){
        // 初始化题目相关信息
        question.setCreateDate(DateUtils.parseDate2String(new Date()));
        // 通过枚举下标，获取的值对应数据需要的值
        // 初始化题目状态及审核状态
        question.setStatus(QuestionConst.Status.PRE_PUBLISH.ordinal());

        question.setReviewStatus(QuestionConst.ReviewStatus.PRE_REVIEW.ordinal());
        result = questionDao.add(question);
    }
}

```

```

    }else{
        result = questionDao.update(question);
    }
    return result;
}

```

然后在主方法中调用此方法，如下：

```

/**
 * 更新题目信息
 * 更新选项信息（更新旧选项，添加信息选项）
 * 更新标签信息（添加新标签，删除旧关系，插入新关系）
 * 更新行业方向（添加新行业，删除旧关系，插入新关系）
 * 更新公司信息（更新公司信息）
 * @param question
 */
@Override
public void addOrUpdate(Question question) {
    SqlSession sqlSession = getSession();
    try{
        log.info("QuestionService question:{0}",question);
        // 1 添加题目信息
        if( addOrUpdateQuestion(sqlSession,question) == 0){
            throw new MmDaoException("更新数据失败");
        }
        log.debug("QuestionService question.id:{0}",question.getId());

        // 2.更新题目选项信息
        // 3.更新标签信息
        // 4.更新公司信息(行业信息)
        // 提交保存
        commitAndCloseSession(sqlSession);
        log.debug("dao question:{0}",question);

    }catch(MmDaoException e){
        log.error("addOrUpdate",e);
        rollbackAndCloseSession(sqlSession);
        throw new MmDaoException(e.getMessage());
    }
}
}

```

5.4.2 保存题目选项

题目选项数据保存到t_question_item表中，如果选项Id为0，即为保存，非0为更新。

- 新增QuestionItemDao接口及映射类

```

/**
 * @author : seanyang
 * @date : Created in 2019/8/15
 * @description : 选项Dao接口
 * @version: 1.0
 */
public interface QuestionItemDao {

```

```

/**
 * 添加选项
 * @param questionItem
 */
Integer addQuestionItem(QuestionItem questionItem);

/**
 * 更新选项
 * @param questionItem
 */
Integer updateQuestionItem(QuestionItem questionItem);
}

```

```

<?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.QuestionItemDao">
    <insert id="addQuestionItem">
        INSERT INTO t_question_item (content, img_url, is_right,
question_id)
        VALUES (#{content},#{imgUrl},#{isRight},#{questionId})
    </insert>
    <update id="updateQuestionItem">
        UPDATE t_question_item
        SET content = #{content},img_url=#{imgUrl},is_right=
#{isRight},question_id=#{questionId}
        WHERE id=#{id}
    </update>
</mapper>

```

- 修订QuestionServiceImpl实现类

增加一个内部实现方法，然后在主方法addOrUpdate调用即可。

```

// 更新题目选项信息
private void updateQuestionItemList(SqlSession sqlSession,Question question)
{
    log.info("QuestionService updateQuestionItemList:
{}",question.getQuestionItemList());
    QuestionItemDao questionItemDao =
getDao(sqlSession,QuestionItemDao.class);
    for (QuestionItem questionItem:question.getQuestionItemList()){
        // 设置应对的题目ID
        questionItem.setQuestionId(question.getId());
        // 如果选项内容或选项为空，不插入到数据库
        if(questionItem.getContent() == null ||
questionItem.getContent().length() ==0){
            if(questionItem.getImgUrl() == null ||
questionItem.getImgUrl().length() == 0)
                continue;
        }
        if(questionItem.getId() == null || questionItem.getId() ==0){
            // 新增
            questionItemDao.addQuestionItem(questionItem);
        }else{
            // 更新
            questionItemDao.updateQuestionItem(questionItem);
        }
    }
}

```

```

    }
}

```

调用更新选项方法，如图：

```

@Override
public void addOrUpdate(Question question) {
    SqlSession sqlSession = getSession();
    try{
        log.info("QuestionService question: {}", question);
        // 1 添加题目信息
        if( addOrUpdateQuestion(sqlSession, question) == 0){
            throw new MmDaoException("更新数据失败");
        }
        log.debug("QuestionService question.id: {}", question.getId());
        // 2.更新题目选项信息
        updateQuestionItemList(sqlSession, question);
    }
}

```

5.4.3 保存标签信息

题目与学科标签是多对多的关系，需要使用关系表tr_question_tag来保存题目与标签的关系。当前业务需要考虑题目的新建与更新，对于的标签关系也是如此。

如果标签关系发生改变，需要同时删除旧关系，新增新关系的方式来实现。

- 修订TagDao接口及映射关系

```

/**
 * 删除某一题目标签关系
 * @param questionId
 */
void deleteTagByQuestionId(Integer questionId);

/**
 * 新增某一题目标签关系
 * @param one
 */
void addTagByQuestionId(Map one);

```

```

<insert id="addTagByQuestionId">
    INSERT INTO tr_question_tag (question_id, tag_id)
    VALUES (#{questionId},#{tagId})
</insert>
<delete id="deleteTagByQuestionId">
    DELETE FROM tr_question_tag
    WHERE question_id = #{questionId}
</delete>

```

如果addTag方法在插入成功后没有返回ID，需要做如下修改：

```
<insert id="addTag">
    INSERT INTO t_tag (name, create_date, status, user_id, course_id)
    VALUES (#{name},#{createDate},#{status},#{userId},#{courseId})
    <selectKey keyProperty="id" order="AFTER" resultType="integer">
        SELECT LAST_INSERT_ID()
    </selectKey>
</insert>
```

- 修订QuestionServiceImpl实现类

增加一个内部实现方法，然后在主方法addOrUpdate调用即可。标签需要初始化启用状态、用户ID、创建日期、学科ID。

```
// 更新标签信息
private void updateTagList(SqlSession sqlSession, Question question){
    log.info("QuestionService updateTagList:{}", question.getTagList());
    TagDao tagDao = getDao(sqlSession, TagDao.class);
    // 删除之前的题目标签关系
    tagDao.deleteTagByQuestionId(question.getId());
    // 增加新关系
    for (Tag tag: question.getTagList()){
        Map<String, Object> mapQuestionTag = new HashMap<>();
        mapQuestionTag.put("questionId", question.getId());
        mapQuestionTag.put("tagId", tag.getId());
        tagDao.addTagByQuestionId(mapQuestionTag);
    }
}
```

调用更新标签方法，如图：

```
public void addOrUpdate(Question question) {
    SqlSession sqlSession = getSession();
    try{
        log.info("QuestionService question: {}", question);
        // 1 添加题目信息
        if( addOrUpdateQuestion(sqlSession, question) == 0){
            throw new MmDaoException("更新数据失败");
        }

        log.debug("QuestionService question.id: {}", question.getId());

        // 2.更新题目选项信息
        updateQuestionItemList(sqlSession, question);
        // 3.更新标签信息
        updateTagList(sqlSession, question);
        // 4.更新公司信息(行业信息)
```

5.4.4 保存公司及行业信息

公司信息发生改变需要更新公司信息，公司信息更新包含公司隶属城市ID及当前操作用户ID。

公司与公司行业关系是多对多关系，在更新二者关系时，需要与标签处理多对多使用相同的方式。

- 更新CompanyDao接口及映射文件

```
/**
 * 更新公司信息
 * @param company
 * @return
 */
Integer updateCompanyCity(Company company);

/**
 * 新增公司行业方向关系
 * @param one
 * @return
 */
Integer addCompanyIndustry(Map one);

/**
 * 基于公司ID，删除公司所属行业方向
 * @param companyId
 * @return
 */
Integer deleteCompanyIndustryByCompanyId(Integer companyId);
```

```
<update id="updateCompanyCity">
    UPDATE t_company
    SET city_id = #{cityId},user_id = #{userId}
    where id = #{id}
</update>
<insert id="addCompanyIndustry">
    INSERT INTO tr_company_industry (company_id, industry_id)
    VALUES (#{companyId},#{industryId})
</insert>
<delete id="deleteCompanyIndustryByCompanyId">
    DELETE FROM tr_company_industry
    WHERE company_id = #{companyId}
</delete>
```

- 更新修订QuestionServiceImpl实现类

增加一个内部实现方法，然后在主方法addOrUpdate调用即可。

```
// 更新公司信息
private void updateCompany(SqlSession sqlSession,Question question){
    log.info("QuestionService updateCompany:{}",question.getCompany());
    CompanyDao companyDao = getDao(sqlSession,CompanyDao.class);
    Company company = question.getCompany();
    // 更新公司城市ID及用户ID
    company.setUserId(question.getUserId());
    companyDao.updateCompanyCity(company);
    // 删除旧企业方向关系
    companyDao.deleteCompanyIndustryByCompanyId(company.getId());
    // 新增新的行业方向及关系
    List<Industry> industryList = company.getIndustryList();
    for (Industry industry:industryList){
```

```

        Map<String, Object> one = new HashMap<>();
        one.put("companyId", company.getId());
        one.put("industryId", industry.getId());
        companyDao.addCompanyIndustry(one);
    }
}

```

调用更新公司方法，如图：

```

public void addOrUpdate(Question question) {
    try{
        log.debug("QuestionService question: {}", question);
        openSession();
        // 1 添加题目信息
        if( addOrUpdateQuestion(question) == 0){
            throw new MmDaoException("更新数据失败");
        }
        log.debug("QuestionService question.id: {}", question.getId());
        // 2.更新题目选项信息
        updateQuestionItemList(question);
        // 3.更新标签信息
        updateTagList(question);
        // 4.更新公司信息(行业信息)
        updateCompany(question);
        // 提交保存
        commitAndCloseSession();
        log.debug("dao question: {}", question);
    }
}

```

全部完成后，测试题目数据保存的过程，可以使用如下查询，查询数据是否正确保存。

```

#查询题目
SELECT * from t_question order by id desc;
#查询题目选项
SELECT * from t_question_item order by question_id desc;
#查询标签
SELECT * from t_tag order by id desc;
#查询题目标签关系
SELECT * from tr_question_tag order by question_id desc;
#查询公司
SELECT * from t_company order by id ;
#查询行业方向
SELECT * from t_industry order by id desc;
#查询公司行业方向关系
SELECT * from tr_company_industry ORDER BY company_id ;

```