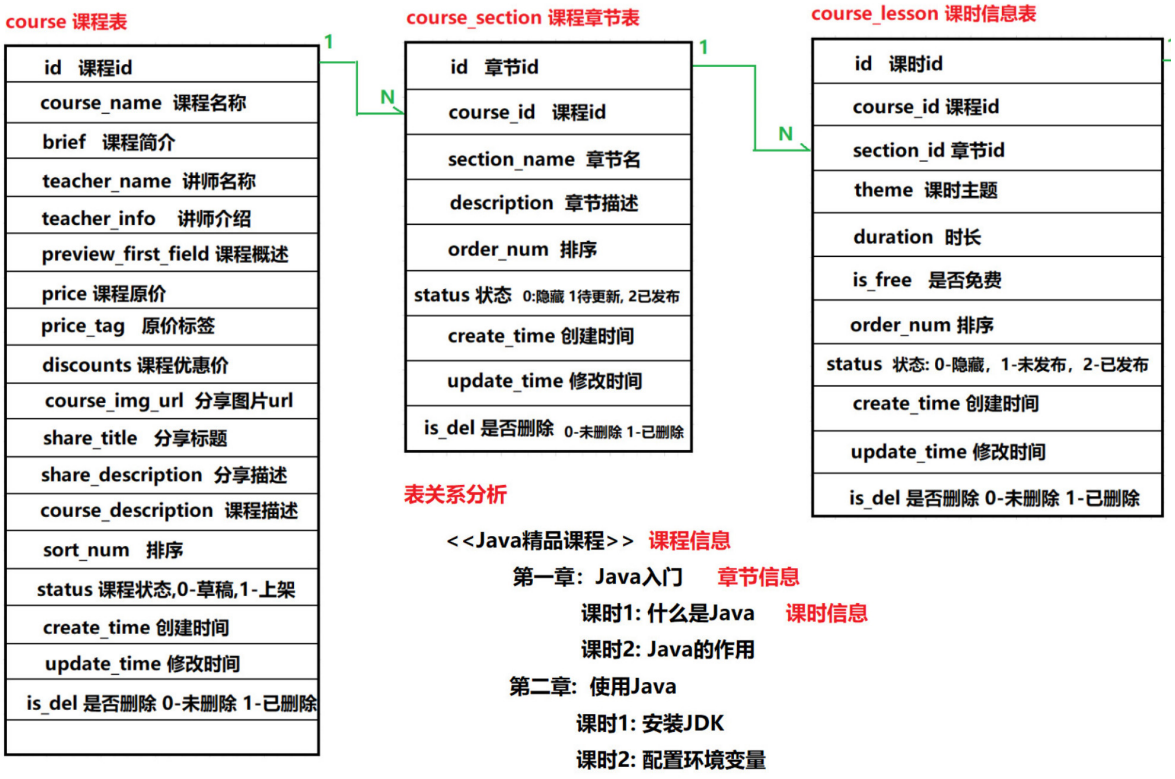
任务三 课程管理模块开发_02

1.开发流程

1.1 需求分析

我们接下来开发的是，配置课时(课程内容管理)模块，主要是对课程内容进行管理

1.2 数据库表分析



1.3 实体类设计

1) `Course`类 与 `Course_Section`类 是一对多关系

在`Course`类中定义一个List集合,并指定List的泛型是 `Course_Section`类型,表示 **一个课程中可以包含多个章节.**

```
Course类

    //添加list集合 泛型是 Course_Section
    List<Course_Section> sectionList = new ArrayList<>();
```

在 `Course_Section`类中,定义一个**Course**类型的属性，用来保存章节所对应的具体的课程信息

```
Course_Section 类

    //添加一个Course类型的属性
    private Course course;
```

2) `Course_Section`类 与 `Course_Lesson`类是一对多关系

在Course_Section类中定义一个List集合,并指定List的泛型是 Course_Lesson类型,这样就可以表示 **一个章节中包含多个课时.**

```
Course_Section类
    //添加一个list集合 泛型是 Course_lesson
    List<Course_Lesson> lessonList = new ArrayList<>();

Course_Lesson类

  //添加一个Course_Section类型的属性
  private Course_Section course_section;
```

1.4 Dao接口及实现类编写

```
/**
 * 课程内容管理 DAO层接口
 * */
public interface CourseContentDao {
}


/**
 * 课程内容管理 DAO层实现类
 * */
public class CourseContentDaoImpl implements CourseContentDao {

}
```

1.5 Service接口及实现类编写

```java
/**
 * 课程内容管理 Service层接口
 * */
public interface CourseContentService {
}


/**
 * 课程内容管理 Service层实现类
 * */
public class CourseContentServiceImpl implements CourseContentService {
}
```

1.6 CourseContentServlet 编写

CourseContentServlet 继承 BaseServlet

```java
@WebServlet("/courseContent")
public class CourseContentServlet extends BaseServlet {

}
```

2. 功能一: 展示课程内容

2.1 需求分析

分析: 要展示的内容是对应课程下的 章节与课时信息

1) 我们先写一条查询语句: 查询ID为1 的课程的章节与课时信息

```sql
SELECT
    cs.id '章节id',
    cs.section_name '章节名称',
    cl.id '课时id',
    cl.theme '课时描述'
FROM  course_section cs INNER JOIN course_lesson cl
ON cs.id = cl.section_id WHERE cs.course_id = ?
```

2) 我们在程序中尽量避免使用连接查询,我们可以将上面的SQL进行拆分,每一条SQL对应一个功能

```sql
-- 根据课程ID查询章节相关的内容
SELECT
id,
course_id,
section_name,
```

```
description,
order_num
FROM course_section cs WHERE course_id = ? ;

-- 根据章节ID查询课时相关的内容
SELECT
id,
course_id,
section_id,
theme,
duration,
is_free,
order_nu
FROM course_lesson WHERE section_id = ?;
```

2.2 DAO层编写

编写两个方法:

接口

```
    //根据课程ID查询课程相关信息
    public List<Course_Section> findSectionAndLessonByCourseId(int courseId);

    //根据章节ID 查询章节相关的课时信息
    public List<Course_Lesson> findLessonBySectionId(int sectionId);
```

实现类

```
     //根据课程ID查询课程相关信息
    @Override
    public List<Course_Section> findSectionAndLessonByCourseId(int courseId) {

        try {
            //1.创建QueryRunner
            QueryRunner qr = new QueryRunner(DruidUtils.getDataSource());

            //2.编写SQL
            String sql = "SELECT \n" +
                    "id,\n" +
                    "course_id,\n" +
                    "section_name,\n" +
                    "description,\n" +
                    "order_num,\n" +
                    "STATUS\n" +
                    "FROM course_section WHERE course_id = ?";

            //3.执行查询
            List<Course_Section> sectionList = qr.query(sql, new
BeanListHandler<Course_Section>(Course_Section.class), courseId);

            //4.根据章节ID查询课时信息
            for (Course_Section section : sectionList) {

                //调用方法 获取章节对应的课时
                List<Course_Lesson> lessonList =
findLessonBySectionId(section.getId());
```

```
                //将课时数据封装到 章节对象中
                section.setLessonList(lessonList);
            }

            return sectionList;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }


    //根据章节ID查询课时信息
    @Override
    public List<Course_Lesson> findLessonBySectionId(int sectionId) {

        try {
            QueryRunner qr = new QueryRunner(DruidUtils.getDataSource());

            String sql = "SELECT \n" +
                    "id,\n" +
                    "course_id,\n" +
                    "section_id,\n" +
                    "theme,\n" +
                    "duration,\n" +
                    "is_free,\n" +
                    "order_num,\n" +
                    "STATUS\n" +
                    "FROM course_lesson WHERE section_id = ?";

            List<Course_Lesson> lessonList = qr.query(sql, new
BeanListHandler<Course_Lesson>(Course_Lesson.class), sectionId);

            return lessonList;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }
```

DAO层测试

```
public class TestCourseContentDao {

    CourseContentDao contentDao = new CourseContentDaoImpl();

    //测试 查询对应课程下的章节与课时
    @Test
    public void testFindSectionAndLessonByCourseId(){

        List<Course_Section> list =
contentDao.findSectionAndLessonByCourseId(59);

        for (Course_Section courseSection : list) {
            System.out.println(courseSection.getId()+" =
"+courseSection.getSection_name());
```

```
            List<Course_Lesson> lessonList = courseSection.getLessonList();
            for (Course_Lesson lesson : lessonList) {
                System.out.println(lesson.getId()+" = "+lesson.getTheme()+" = "
+ lesson.getSection_id());
            }
        }
    }

}
```

2.3 Service层编写

接口

```
/**
 * 课程内容管理 Service层接口
 * */
public interface CourseContentService {

    //根据课程id查询课程内容
    public List<Course_Section> findSectionAndLessonByCourseId(int courseId);

}
```

实现类

```
/**
 * 课程内容管理 Service层实现类
 * */
public class CourseContentServiceImpl implements CourseContentService {

    CourseContentDao contentDao = new CourseContentDaoImpl();

    @Override
    public List<Course_Section> findSectionAndLessonByCourseId(int courseId) {
        List<Course_Section> sections =
contentDao.findSectionAndLessonByCourseId(courseId);
        return sections;
    }
}
```

2.4 Servlet编写

CourseContentServlet中添加 **findSectionAndLessonByCourseId** 方法

```
@WebServlet("/courseContent")
public class CourseContentServlet extends BaseServlet {

     /**
     * 展示对应课程的章节与课时信息
     * */
    public void findSectionAndLessonByCourseId(HttpServletRequest request ,
HttpServletResponse response){

        try {
            //1.获取参数
            String course_id = request.getParameter("course_id");
```

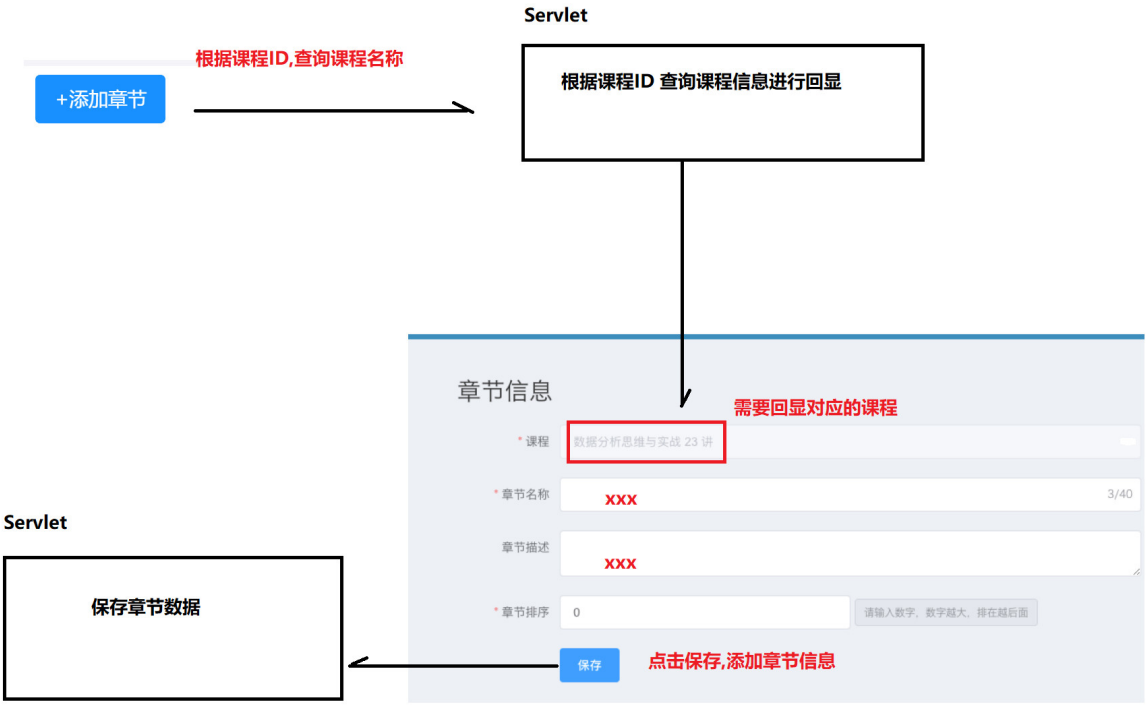
```
            //2.业务处理
            CourseContentService contentService = new
CourseContentServiceImpl();
            List<Course_Section> sectionList =
contentService.findSectionAndLessonByCourseId(Integer.parseInt(course_id));

            //3.返回结果
            String result = JSON.toJSONString(sectionList);
            response.getWriter().println(result);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2.5 接口测试

查看接口文档,进行测试

3.功能二: 新建章节信息

3.1 需求分析



3.2 DAO层编写

接口

```
    //添加章节时进行数据回显
    public Course findCourseByCourseId(int courseId);

    //保存章节信息
    public int saveSection(Course_Section section);
```

实现类

```
    /**
     * 添加章节时进行数据回显
     * */
    @Override
    public Course findCourseByCourseId(int courseId) {

        try {
            //1.创建QueryRunner
            QueryRunner qr = new QueryRunner(DruidUtils.getDataSource());

            //2.编写SQL
            String sql = "SELECT id,course_name FROM course WHERE id = ?";

            //3.执行查询
            Course course = qr.query(sql, new BeanHandler<Course>(Course.class),
courseId);

            //4.返回结果
            return course;

        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    @Override
    public int saveSection(Course_Section section) {
        try {
            //1.创建QueryRunner
            QueryRunner qr = new QueryRunner(DruidUtils.getDataSource());

            //2.编写SQL
            String sql = "INSERT INTO
course_section(course_id,section_name,description,order_num,STATUS,create_time,u
pdate_time)\n" +
                    "VALUES(?,?,?,?,?,?,?);";

            //3.准备参数
            Object[] param =
{section.getCourse_id(),section.getSection_name(),section.getDescription(),

 section.getOrder_num(),section.getStatus(),section.getCreate_time(),section.get
Update_time()};

            //4.执行插入
            int i = qr.update(sql, section.getCourse_id(), param);

            //4.返回结果
            return i;

        } catch (SQLException e) {
            e.printStackTrace();
            return 0;
        }
    }
```

3.3 Service层编写

```
接口
    public Course findCourseById(int courseId);

    public String saveSection(Course_Section section);

实现类
    @Override
    public Course findCourseById(int courseId) {
        Course course = contentDao.findCourseById(courseId);
        return course;
    }

    @Override
    public String saveSection(Course_Section section) {

        //1.补全章节信息
        section.setStatus(2); //状态，0:隐藏；1：待更新；2：已发布
        String date = DateUtils.getDateFormart();
        section.setCreate_time(date);
        section.setUpdate_time(date);


        //2.调用Dao进行插入
        int i = contentDao.saveSection(section);

        //3.根据插入是否成功,封装对应信息
        if(i > 0){
            //保存成功
            String result = StatusCode.SUCCESS.toString();
            return result;
        }else{
            //保存失败
            String result = StatusCode.FAIL.toString();
            return result;
        }
    }
```

3.4 Servlet编写

CourseContentServlet中添加 **findCourseById** 方法

3.4.1 课程信息回显接口

```
    //回显章节对应的课程信息
    public void findCourseById(HttpServletRequest request , HttpServletResponse
response){

        try {
            //1.获取参数
            String courseId = request.getParameter("course_id");

            //2.业务处理
            CourseContentService contentService = new
CourseContentServiceImpl();
```

```
            Course course =
contentService.findCourseById(Integer.parseInt(courseId));

            //3.返回数据,将对象转换为JSON,只转换需要的字段
            SimplePropertyPreFilter filter = new
SimplePropertyPreFilter(Course.class,"id","course_name");

            String result = JSON.toJSONString(course,filter);
            response.getWriter().println(result);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

3.4.2 保存章节信息接口

1. POST请求方式介绍

POST 请求方法常用的三种数据提交格式

格式	说明
Content-Type : application/x-www-form-urlencoded	请求体中的数据会以普通表单形式（键值对）发送到后端。
Content-Type : application/json ; charset=utf-8	请求体中的数据会以json字符串的形式发送到后端。
Content-Type : multipart/form-data	多部件上传既可以上传键值对 也可以上传文件。

> 注意: 第二种JSON格式与第三种多部件上传,使用 getParameter() 方法都无法获取数据

2. 需求分析分析

根据接口文档描述: 前台传输的是JSON格式的数据, 使用**getParameter() 方法无法获取参数**

```
{
    "methodName":"saveOrUpdateSection",
    "course_id":19,
    "section_name:"微服务架构",
    "description":"跟着药水一起学习如何使用微服务",
    "order_num ":0
}
```

3. 修改BaseServlet

如果请求参数是JSON格式的数, 我们可以通过 **request.getReader()** 这个方法,获取一个流对象来进行读取.

1) 在**BaseServlet** 中创建一个方法,用来获取JSON格式的数据

```
    /**
     *  POST请求格式为 application/json;charset=utf-8
     *  在这个方法中我们使用流的方式,获取到POST请求的数据
     * */
    public String getPostJSON(HttpServletRequest request){
```

```
        try {
            //1.从request中获取 字符缓冲输入流对象
            BufferedReader reader = request.getReader();


            //2.创建 StringBuffer,用来保存读取出的数据
            StringBuffer sb = new StringBuffer();

            //3.循环读取
            String line = null;
            while((line = reader.readLine()) != null){
                //追加到 StringBuffer中
                sb.append(line);
            }

            //4.将读取到的内容转换为字符串,并返回
            return sb.toString();

        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }
```

2) 修改BaseServlet中的doGet方法

1.获取POST请求的 Content-Type类型

2.判断传递的数据是不是JSON格式

3.如果是 就调用上面编写的 getPostJSON方法,获取数据

4.将获取到的JSON格式的字符串转换为 Map

5.从Map中获取要调用的方法名

6.将Map保存到request域对象中 (流只能使用一次)

```
@Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

        //1.获取参数 要访问的方法名
        //String methodName = req.getParameter("methodName");
        String methodName = null;

        //2.获取POST请求的 Content-Type类型
        String contentType = req.getHeader("Content-Type");

        //3.判断传递的数据是不是JSON格式
        if("application/json;charset=utf-8".equals(contentType)){
            //是JOSN格式 调用getPostJSON
            String postJSON = getPostJSON(req);

            //将JSON格式的字符串转化为map
            Map<String,Object> map = JSON.parseObject(postJSON, Map.class);

            //从map集合中获取 methodName
```

```
            methodName =(String) map.get("methodName");

            //将获取到的数据,保存到request域对象中
            req.setAttribute("map",map);

        }else{
            methodName = req.getParameter("methodName");
        }

        //2.判断 执行对应的方法
        if(methodName != null){
            //通过反射优化代码 提升代码的可维护性

            try {
                //1.获取字节码文件对象
                Class c = this.getClass();

                //2.根据传入的方法名,获取对应的方法对象  findByName
                Method method = c.getMethod(methodName,
HttpServletRequest.class, HttpServletResponse.class);

                //3.调用method对象的 invoke方法,执行对应的功能
                method.invoke(this,req,resp);

            } catch (Exception e) {
                e.printStackTrace();
                System.out.println("请求的功能不存在!!");
            }
        }

    }
```

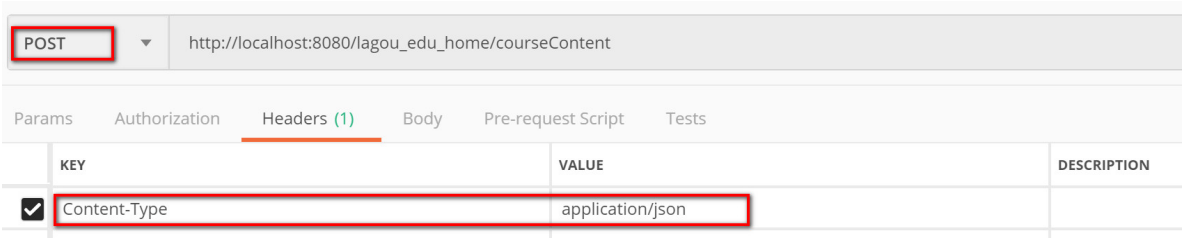
4. 编写接口代码

```
    /**
     * 保存&修改 章节信息
     * */
    public void saveOrUpdateSection(HttpServletRequest request
,HttpServletResponse response){

        try {
            //1.获取参数  从域对象中获取
            Map<String,Object> map = (Map)request.getAttribute("map");

            //2.创建Course_Section
            Course_Section section = new Course_Section();

            //3.使用BeanUtils工具类,将map中的数据封装到 section
            BeanUtils.populate(section,map);

            //4.业务处理
            CourseContentService contentService = new
CourseContentServiceImpl();
            String result = contentService.saveSection(section);

            //5.响应结果
            response.getWriter().print(result);
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
```

5. 测试接口

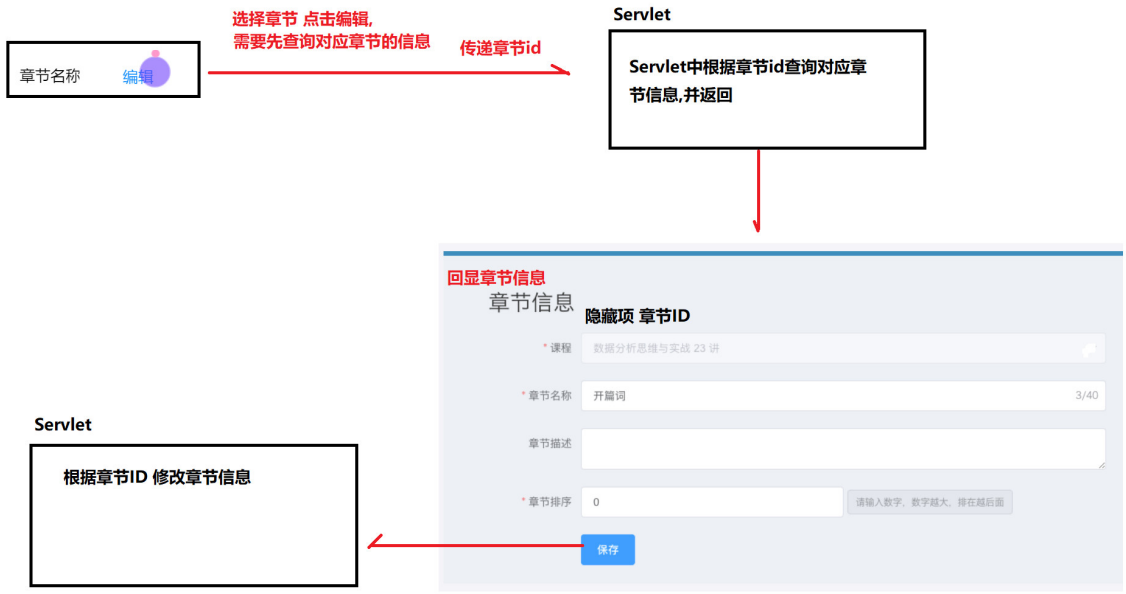
1) 选择POST请求方式,设置Content-Type = application/json



2) 选择raw 发送JSON格式数据

4.功能三: 章节信息修改

需求分析:



注意: 接口文档中并没有要求编写回显接口,说明回显操作由前端完成.

4.1 DAO层编写

接口

```
    //修改章节信息
    public int updateSection(Course_Section section);
```

接口实现类

```
     /**
     * 修改章节信息
     * */
    @Override
    public int updateSection(Course_Section section) {

        try {
            //1.创建QueryRunner
            QueryRunner qr = new QueryRunner(DruidUtils.getDataSource());

            //2.编写SQL
            String sql = "UPDATE course_section SET\n" +
                    "section_name = ?,\n" +
                    "description = ?,\n" +
                    "order_num = ?,\n" +
                    "update_time = ?  WHERE id = ?;";

            //3.准备参数
            Object[] param =
{section.getSection_name(),section.getDescription(),section.getOrder_num(),
            section.getUpdate_time(),section.getId()};

            //4.执行修改操作
            int row = qr.update(sql, param);

            return row;
        } catch (SQLException e) {
            e.printStackTrace();
            return 0;
        }
    }
```

4.2 Service层编写

接口

```
    public String updateSection(Course_Section section);
```

实现类

```
    @Override
    public  String updateSection(Course_Section section) {

        //1.补全章节信息
        String date = DateUtils.getDateFormart();
        section.setUpdate_time(date);

        //2.调用Dao进行插入
        int i = contentDao.updateSection(section);

        //3.根据修改是否成功,封装对应信息
        if(i > 0){
            //保存成功
            String result = StatusCode.SUCCESS.toString();
```

```
            return result;
        }else{
            //保存失败
            String result = StatusCode.FAIL.toString();
            return result;
        }

    }
```

4.3 Servlet编写

保存章节信息和修改章节信息,访问的是同一个接口,所以在**saveOrUpdateSection**方法中中,我们要进行一下判断

- 携带id 就是修改章节操作
- 未携带id就是新增章节操作

```
 /**
     * 保存或修改章节信息
     * */
    public void saveOrUpdateSection(HttpServletRequest request ,
HttpServletResponse response){

        try {
            //1.获取参数
            Map<String, Object> map =(Map) request.getAttribute("map");

            //2.创建 Course_Section
            Course_Section section = new Course_Section();

            //3.使用BeanUtils,将map中的数据封装到section对象里
            BeanUtils.copyProperties(section,map.get("section"));

            //4.业务处理
            CourseContentService contentService = new
CourseContentServiceImpl();

            if(section.getId() != 0){

                //修改操作
                String result = contentService.updateSection(section);

                //5.返回结果数据
                response.getWriter().println(result);

            }else{
                //添加操作
                String result  = contentService.saveSection(section);
                //5.返回结果数据
                response.getWriter().println(result);

            }


        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }
```

4.4 接口测试

查看接口文档,进行测试

5.功能四: 章节状态管理

6.1 需求分析

根据选择的状态信息,发送对应的状态编号 进行修改, status 状态，0:隐藏；1：待更新；2：已发布

6.2 DAO层编写

接口

```
    //修改章节的状态
    public int updateSectionStatus(int id,int status);
```

实现类

```
    /**
     * 修改章节状态
     * */
    @Override
    public int updateSectionStatus(int id,int status) {

        try {
            //1.创建QueryRunner
            QueryRunner qr = new QueryRunner(DruidUtils.getDataSource());

            //2.编写SQL
            String sql = "UPDATE course_section SET STATUS = ?,update_time = ?
WHERE id = ?;";

            //3.准备参数
            Object[] param = {status , DateUtils.getDateFormart(),id};

            //4.执行修改操作
            int row = qr.update(sql, param);

            return row;
        } catch (SQLException e) {
            e.printStackTrace();
            return 0;
```

```
        }
    }
```

6.3 Service层编写

接口

```
    public  String  updateSectionStatus(int id,int status);
```

实现类

```
    @Override
    public String updateSectionStatus(int id, int status) {

        //调用Dao 修改状态
        int i = contentDao.updateSectionStatus(id,status);

        //3.根据修改是否成功,封装对应信息
        if(i > 0){
            String result = StatusCode.SUCCESS.toString();
            return result;
        }else{

            String result = StatusCode.FAIL.toString();
            return result;
        }

    }
```

6.4 Servlet编写

```
    /**
     * 修改章节状态
     * */
    public void updateSectionStatus(HttpServletRequest request ,
HttpServletResponse response){

        try {
            //1.获取参数
            int id = Integer.parseInt(request.getParameter("id"));
            int status = Integer.parseInt(request.getParameter("status"));

            //4.业务处理
            CourseContentService contentService = new
CourseContentServiceImpl();
            String result = contentService.updateStatus(id, status);

            //5.返回结果数据findSectionById
            response.getWriter().println(result);

        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }
```

6.5 接口测试

按照接口文档进行测试