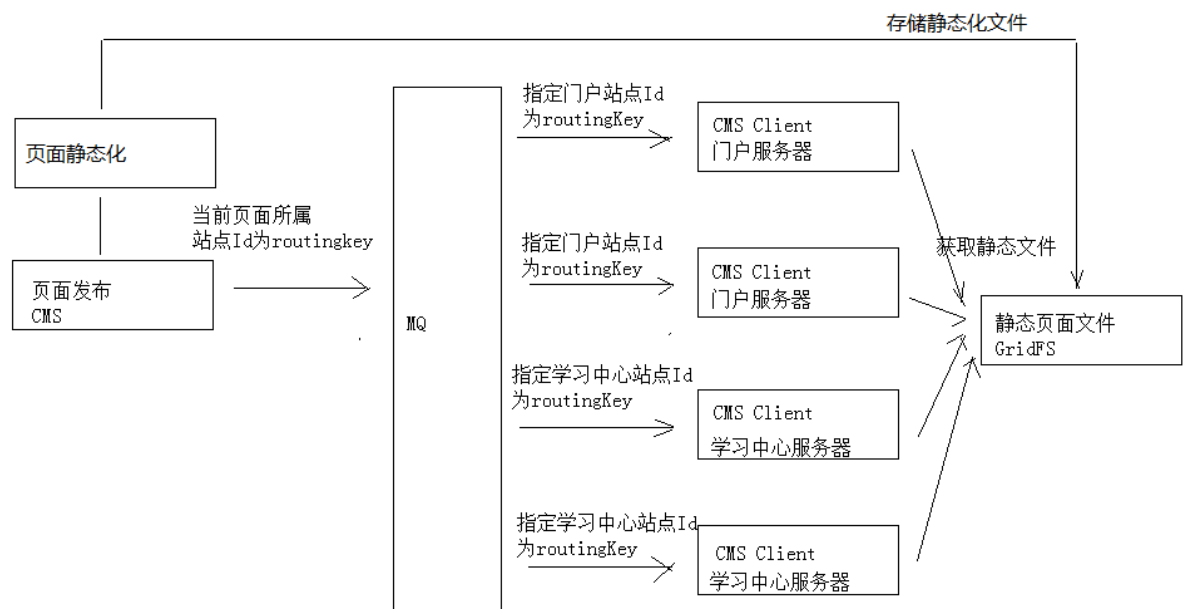


学成在线 第6天 讲义-页面发布 课程管理

1 页面发布

1.1 技术方案

本项目使用MQ实现页面发布的技术方案如下：



技术方案说明：

- 1、平台包括多个站点，页面归属不同的站点。
- 2、发布一个页面应将该页面发布到所属站点的服务器上。
- 3、每个站点服务部署cms client程序，并与交换机绑定，绑定时指定站点Id为routingKey。

指定站点id为routingKey就可以实现cms client只能接收到所属站点的页面发布消息。

- 4、页面发布程序向MQ发布消息时指定页面所属站点Id为routingKey，将该页面发布到它所在服务器上的cms client。

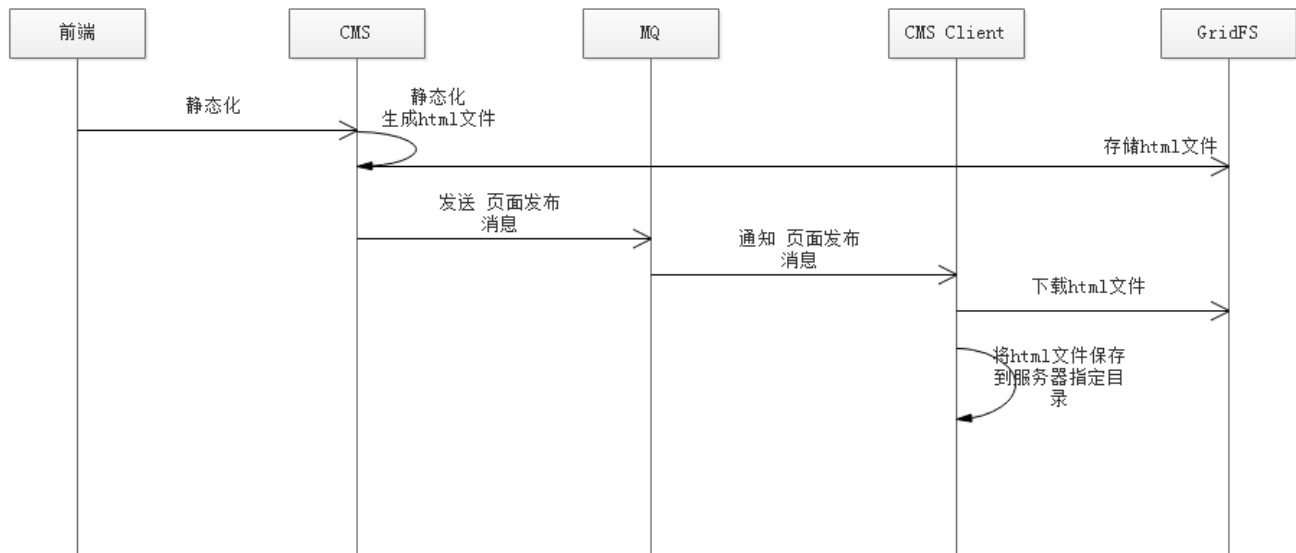
路由模式分析如下：

发布一个页面，需发布到该页面所属的每个站点服务器，其它站点服务器不发布。

比如：发布一个门户的页面，需要发布到每个门户服务器上，而用户中心服务器则不需要发布。

所以本项目采用routing模式，用站点id作为routingKey，这样就可以匹配页面只发布到所属的站点服务器上。

页面发布流程图如下：



- 1、前端请求cms执行页面发布。
- 2、cms执行静态化程序生成html文件。
- 3、cms将html文件存储到GridFS中。
- 4、cms向MQ发送页面发布消息
- 5、MQ将页面发布消息通知给Cms Client
- 6、Cms Client从GridFS中下载html文件
- 7、Cms Client将html保存到所在服务器指定目录

1.2 页面发布消费方

1.2.1需求分析

功能分析：

创建Cms Client工程作为页面发布消费方，将Cms Client部署在多个服务器上，它负责接收到页面发布 的消息后从GridFS中下载文件在本地保存。

需求如下：

- 1、将cms Client部署在服务器，配置队列名称和站点ID。
- 2、cms Client连接RabbitMQ并监听各自的“页面发布队列”
- 3、cms Client接收页面发布队列的消息
- 4、根据消息中的页面id从mongodb数据库下载页面到本地

调用dao查询页面信息，获取到页面的物理路径，调用dao查询站点信息，得到站点的物理路径

页面物理路径=站点物理路径+页面物理路径+页面名称。

从GridFS查询静态文件内容，将静态文件内容保存到页面物理路径下。

1.2.2创建Cms Client工程

1、创建maven工程

pom.xml

```
<?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">
    <parent>
        <artifactId>xc-framework-parent</artifactId>
        <groupId>com.xuecheng</groupId>
        <version>1.0-SNAPSHOT</version>
        <relativePath>../xc-framework-parent/pom.xml</relativePath>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>xc-service-manage-cms-client</artifactId>

    <dependencies>
        <dependency>
            <groupId>com.xuecheng</groupId>
            <artifactId>xc-framework-model</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-amqp</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-io</artifactId>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
```

```
        <artifactId>fastjson</artifactId>
    </dependency>
</dependencies>

</project>
```

2、配置文件

在resources下配置application.yml和logback-spring.xml。

application.yml的内容如下：

```
server:
  port: 31000
spring:
  application:
    name: xc-service-manage-cms-client
  data:
    mongodb:
      uri: mongodb://root:123@localhost:27017
      database: xc_cms
    rabbitmq:
      host: 127.0.0.1
      port: 5672
      username: guest
      password: guest
      virtualHost: /
  xuecheng:
    mq:
      #cms客户端监控的队列名称（不同的客户端监控的队列不能重复）
      queue: queue_cms_postpage_01
      routingKey: 5a751fab6abb5044e0d19ea1    #此routingKey为门户网站ID
```

说明：在配置文件中配置队列的名称，每个 cms client在部署时注意队列名称不要重复

3、启动类

```
@SpringBootApplication
@EntityScan("com.xuecheng.framework.domain.cms")//扫描实体类
@ComponentScan(basePackages={"com.xuecheng.framework"})//扫描common下的所有类
@ComponentScan(basePackages={"com.xuecheng.manage_cms_client"})
public class ManageCmsClientApplication {
    public static void main(String[] args) {
        SpringApplication.run(ManageCmsClientApplication.class, args);
    }
}
```

1.2.3 RabbitmqConfig配置类

消息队列设置如下：

- 1、创建“ex_cms_postpage”交换机
- 2、每个Cms Client创建一个队列与交换机绑定
- 3、每个Cms Client程序配置队列名称和routingKey，将站点ID作为routingKey。

```
package com.xuecheng.manage_cms_client.config;

import org.springframework.amqp.core.*;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * @author Administrator
 * @version 1.0
 */
@Configuration
public class RabbitmqConfig {
    //队列bean的名称
    public static final String QUEUE_CMS_POSTPAGE = "queue_cms_postpage";
    //交换机的名称
    public static final String EX_ROUTING_CMS_POSTPAGE="ex_routing_cms_postpage";
    //队列的名称
    @Value("${xuecheng.mq.queue}")
    public String queue_cms_postpage_name;
    //routingKey 即站点Id
    @Value("${xuecheng.mq.routingKey}")
    public String routingKey;

    /**
     * 交换机配置使用direct类型
     * @return the exchange
     */
    @Bean(EX_ROUTING_CMS_POSTPAGE)
    public Exchange EXCHANGE_TOPICS_INFORM() {
        return ExchangeBuilder.directExchange(EX_ROUTING_CMS_POSTPAGE).durable(true).build();
    }
    //声明队列
    @Bean(QUEUE_CMS_POSTPAGE)
    public Queue QUEUE_CMS_POSTPAGE() {
        Queue queue = new Queue(queue_cms_postpage_name);
        return queue;
    }

    /**
     * 绑定队列到交换机
     *
     * @param queue the queue
     * @param exchange the exchange
     */
}
```

```
* @return the binding
*/
@Bean
public Binding BINDING_QUEUE_INFORM_SMS(@Qualifier(Queue_CMS_POSTPAGE) Queue queue,
@Qualifier(EX_ROUTING_CMS_POSTPAGE) Exchange exchange) {
    return BindingBuilder.bind(queue).to(exchange).with(routingKey).noargs();
}
}
```

1.2.4 定义消息格式

消息内容采用json格式存储数据，如下：

页面id：发布页面的id

```
{
    "pageId":""
}
```

1.2.5 PageDao

1、使用CmsPageRepository 查询页面信息

```
public interface CmsPageRepository extends MongoRepository<CmsPage,String> {
}
```

2、使用CmsSiteRepository查询站点信息，主要获取站点物理路径

```
public interface CmsSiteRepository extends MongoRepository<CmsSite,String> {
}
```

1.2.6 PageService

在Service中定义保存页面静态文件到服务器物理路径方法：

```
package com.xuecheng.manage_cms_client.service;

import com.mongodb.client.gridfs.GridFSBucket;
import com.mongodb.client.gridfs.GridFSDownloadStream;
import com.mongodb.client.gridfs.model.GridFSFile;
import com.xuecheng.framework.domain.cms.CmsPage;
import com.xuecheng.framework.domain.cms.CmsSite;
import com.xuecheng.framework.domain.cms.response.CmsCode;
import com.xuecheng.framework.exception.ExceptionCast;
import com.xuecheng.manage_cms_client.dao.CmsPageRepository;
import com.xuecheng.manage_cms_client.dao.CmsSiteRepository;
```



```
import org.apache.commons.io.IOUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.gridfs.GridFsResource;
import org.springframework.data.mongodb.gridfs.GridFsTemplate;
import org.springframework.stereotype.Service;

import java.io.*;
import java.util.Optional;

/**
 * @author Administrator
 * @version 1.0
 */
@Service
public class PageService {

    @Autowired
    CmsPageRepository cmsPageRepository;
    @Autowired
    CmsSiteRepository cmsSiteRepository;

    @Autowired
    GridFsTemplate gridFsTemplate;
    @Autowired
    GridFSBucket gridFSBucket;

    //将页面html保存到页面物理路径
    public void savePageToServerPath(String pageId){
        Optional<CmsPage> optional = cmsPageRepository.findById(pageId);
        if(!optional.isPresent()){
            ExceptionCast.cast(CmsCode.CMS_PAGE_NOTEXISTS);
        }
        //取出页面物理路径
        CmsPage cmsPage = optional.get();
        //页面所属站点
        CmsSite cmsSite = this.getCmsSiteById(cmsPage.getSiteId());
        //页面物理路径
        String pagePath = cmsSite.getSitePhysicalPath() + cmsPage.getPagePhysicalPath() +
        cmsPage.getPageName();
        //查询页面静态文件
        String htmlFileId = cmsPage.getHtmlFileId();
        InputStream inputStream = this.getFileById(htmlFileId);
        if(inputStream == null){
            ExceptionCast.cast(CmsCode.CMS_GENERATEHTML_HTMLISNULL);
        }
        FileOutputStream fileOutputStream = null;
        try {
            fileOutputStream = new FileOutputStream(new File(pagePath));
            //将文件内容保存到服务物理路径
            IOUtils.copy(inputStream, fileOutputStream);
        } catch (Exception e) {
```

```
        e.printStackTrace();
    }finally {
        try {
            inputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            fileOutputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

//根据文件id获取文件内容
public InputStream getFileById(String fileId){
    try {
        GridFSFile gridFSFile =
gridFsTemplate.findOne(Query.query(Criteria.where("_id").is(fileId)));
        GridFSDownloadStream gridFSDownloadStream =
gridFSBucket.openDownloadStream(gridFSFile.getObjectId());
        GridFsResource gridFsResource = new GridFsResource(gridFSFile,gridFSDownloadStream);
        return gridFsResource.getInputStream();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}

//根据站点id得到站点
public CmsSite getCmsSiteById(String siteId){
    Optional<CmsSite> optional = cmsSiteRepository.findById(siteId);
    if(optional.isPresent()){
        CmsSite cmsSite = optional.get();
        return cmsSite;
    }
    return null;
}
}
```

1.2.6ConsumerPostPage

在cms client工程的mq包下创建ConsumerPostPage类，ConsumerPostPage作为发布页面的消费客户端，监听页面发布队列的消息，收到消息后从mongodb下载文件，保存在本地。

```
package com.xuecheng.manage_cms_client.mq;

import com.alibaba.fastjson.JSON;
import com.xuecheng.framework.domain.cms.CmsPage;
```




```
import com.xuecheng.manage_cms_client.dao.CmsPageRepository;
import com.xuecheng.manage_cms_client.service.PageService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.Map;
import java.util.Optional;

/**
 * @author Administrator
 * @version 1.0
 */
@Component
public class ConsumerPostPage {
    private static final Logger LOGGER = LoggerFactory.getLogger(ConsumerPostPage.class);
    @Autowired
    CmsPageRepository cmsPageRepository;
    @Autowired
    PageService pageService;

    @RabbitListener(queues={"${xuecheng.mq.queue}"})
    public void postPage(String msg){
        //解析消息
        Map map = JSON.parseObject(msg, Map.class);
        LOGGER.info("receive cms post page:{",msg.toString());
        //取出页面id
        String pageId = (String) map.get("pageId");
        //查询页面信息
        Optional<CmsPage> optional = cmsPageRepository.findById(pageId);
        if(!optional.isPresent()){
            LOGGER.error("receive cms post page,cmsPage is null:{",msg.toString());
            return ;
        }
        //将页面保存到服务器物理路径
        pageService.savePageToServerPath(pageId);
    }
}
```

1.3 页面发布生产方

1.3.1 需求分析

管理员通过 cms系统发布“页面发布”的消费，cms系统作为页面发布的生产方。

需求如下：

- 1、管理员进入管理界面点击“页面发布”，前端请求cms页面发布接口。
- 2、cms页面发布接口执行页面静态化，并将静态化页面存储至GridFS中。
- 3、静态化成功后，向消息队列发送页面发布的消息。
 - 1) 获取页面的信息及页面所属站点ID。
 - 2) 设置消息内容为页面ID。（采用json格式，方便日后扩展）
 - 3) 发送消息给ex cms_postpage交换机，并将站点ID作为routingKey。

1.3.2 RabbitMQ配置

1、配置Rabbitmq的连接参数

在application.yml添加如下配置：

```
spring:
  rabbitmq:
    host: 127.0.0.1
    port: 5672
    username: guest
    password: guest
    virtualHost: /
```

2、在pom.xml添加依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

3、RabbitMQConfig配置

由于cms作为页面发布方要面对很多不同站点的服务器，面对很多页面发布队列，所以这里不再配置队列，只需要配置交换机即可。

在cms工程只配置交换机名称即可。

```
package com.xuecheng.manage_cms.config;

import org.springframework.amqp.core.*;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;
```

```
package com.xuecheng.managecmsclient.config;

import org.springframework.amqp.core.*;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitmqConfig {
    //交换机的名称
    public static final String EX_ROUTING_CMS_POSTPAGE = "ex_routing_cms_postpage";
    /**
     * 交换机配置使用direct类型
     * @return the exchange
     */
    @Bean(EX_ROUTING_CMS_POSTPAGE)
    public Exchange EXCHANGE_TOPICS_INFORM() {
        return ExchangeBuilder.directExchange(EX_ROUTING_CMS_POSTPAGE).durable(true).build();
    }
}
```

1.3.3 Api接口

在api工程定义页面发布接口：

```
@ApiOperation("发布页面")
public ResponseResult post(String pageId);
```

1.3.4 PageService

在PageService中定义页面发布方法，代码如下：

```
//页面发布
public ResponseResult postPage(String pageId){
    //执行静态化
    String pageHtml = this.getPageHtml(pageId);
    if(StringUtils.isEmpty(pageHtml)){
        ExceptionCast.cast(CmsCode.CMS_GENERATEHTML_HTMLISNULL);
    }
    //保存静态化文件
    CmsPage cmsPage = saveHtml(pageId, pageHtml);
    //发送消息
    sendPostPage(pageId);
    return new ResponseResult(CommonCode.SUCCESS);
}
//发送页面发布消息
private void sendPostPage(String pageId){
```



```
CmsPage cmsPage = this.getById(pageId);
if(cmsPage == null){
    ExceptionCast.cast(CmsCode.CMS_PAGE_NOTEXISTS);
}
Map<String,String> msgMap = new HashMap<>();
msgMap.put("pageId",pageId);
//消息内容
String msg = JSON.toJSONString(msgMap);
//获取站点id作为routingKey
String siteId = cmsPage.getSiteId();
//发布消息
this.rabbitTemplate.convertAndSend(RabbitmqConfig.EX_ROUTING_CMS_POSTPAGE,siteId, msg);
}
//保存静态页面内容
private CmsPage saveHtml(String pageId,String content){
    //查询页面
    Optional<CmsPage> optional = cmsPageRepository.findById(pageId);
    if(!optional.isPresent()){
        ExceptionCast.cast(CmsCode.CMS_PAGE_NOTEXISTS);
    }
    CmsPage cmsPage = optional.get();
    //存储之前先删除
    String htmlFileId = cmsPage.getHtmlFileId();
    if(StringUtils.isEmpty(htmlFileId)){
        gridFsTemplate.delete(Query.query(Criteria.where("_id").is(htmlFileId)));
    }
    //保存html文件到GridFS
    InputStream inputStream = IOUtils.toInputStream(content);
    ObjectId objectId = gridFsTemplate.store(inputStream, cmsPage.getPageName());
    //文件id
    String fileId = objectId.toString();
    //将文件id存储到cmspage中
    cmsPage.setHtmlFileId(fileId);
    cmsPageRepository.save(cmsPage);
    return cmsPage;
}
```

1.3.5 CmsPageController

编写Controller实现api接口，接收页面请求，调用service执行页面发布。

```
@Override
@PostMapping("/postPage/{pageId}")
public ResponseResult post(@PathVariable("pageId") String pageId) {
    return pageService.postPage(pageId);
}
```

1.4 页面发布前端

用户操作流程：

- 1、用户进入cms页面列表。
- 2、点击“发布”请求服务端接口，发布页面。
- 3、提示“发布成功”，或发布失败。

1.4.1 API方法

在 cms前端添加 api方法。

```
/*发布页面*/
export const page_postPage= id => {
  return http.requestPost(apiUrl+ '/cms/page/postPage/'+id)
}
```

1.4.2 页面

修改page_list.vue，添加发布按钮

```
<el-table-column label="发布" width="80">
  <template slot-scope="scope">
    <el-button
      size="small" type="primary" plain @click="postPage(scope.row.pageId)">发布
    </el-button>
  </template>
</el-table-column>
```

添加页面发布事件：

```
postPage (id) {
  this.$confirm('确认发布该页面吗?', '提示', {
  }).then(() => {
    cmsApi.page_postPage(id).then((res) => {
      if(res.success){
        console.log('发布页面id='+id);
        this.$message.success('发布成功，请稍后查看结果');
      }else{
        this.$message.error('发布失败');
      }
    });
  }).catch(() => {

  });
},
```

1.5 测试

这里测试轮播图页面修改、发布的流程：

1、修改轮播图页面模板或修改轮播图地址

注意：先修改页面原型，页面原型调试正常后再修改页面模板。

2、执行页面预览

3、执行页面发布，查看页面是否写到网站目录

4、刷新门户首页并观察轮播图是否变化。

1.6 思考

1、如果发布到服务器的页面内容不正确怎么办？

2、一个页面需要发布很多服务器，点击“发布”后如何知道详细的发布结果？

3、一个页面发布到多个服务器，其中有一个服务器发布失败时怎么办？

2 课程管理

2.1 需求分析

在线教育平台的课程信息相当于电商平台的商品。课程管理是后台管理功能中最重要的模块。本项目为教学机构提供课程管理功能，教学机构可以添加属于自己的课程，供学生在线学习。

课程管理包括如下功能需求：

1、分类管理

2、新增课程

3、修改课程

4、预览课程

5、发布课程

用户的操作流程如下：

1、进入我的课程

我的课程



课程名称

新增课程



Bootstrap开发框架

管理课程



spring cloud实战

管理课程



Javascript之VueJS

管理课程



Redis从入门到项目实战

管理课程

2、点击“添加课程”，进入添加课程界面

基本信息

课程图片

课程营销

课程计划

教师信息

发布课程

* 课程名称

适用人群

课程分类

请选择

* 课程等级

* 学习模式

课程介绍

提交

3、输入课程基本信息，点击提交

4、课程基本信息提交成功，自动进入“管理课程”界面，点击“管理课程”也可以进入“管理课程”界面

基本信息

课程名称：

课程首页 基本信息 课程图片 课程营销 课程计划 教师信息 发布课程

* 课程名称

适用人群

课程分类 请选择

* 课程等级 ☐ 低级 ☐ 中级 ☐ 高级

* 学习模式 ☐ 自由学习 ☐ 任务式学习

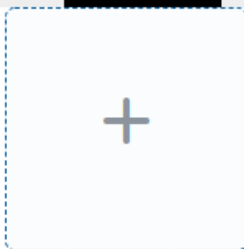
5、编辑图片

上传课程图片。

课程图片

课程名称：

课程首页 基本信息 课程图片 课程营销 课程计划 教师信息 发布课程



6、编辑课程营销信息

营销信息主要是设置课程的收费方式及价格。



课程首页

基本信息

课程图片

课程营销

课程计划

教师信息

发布课程

课程价格 ☐ 免费 ☐ 收费

金额 (元) :

课程有效期 ☐ 永久有效 ☐ 在指定日期下线

过期时间:

服务咨询QQ

提交

7、编辑课程计划

课程计划

课程首页

基本信息

课程图片

课程营销

课程计划

教师信息

发布课程

添加课程计划

▼ 项目概述

业务需求分析

订单业务分析

▼ 运单管理

ElasticSearch下载安装

ElasticSearch入门案例

选择视频 修改 删除

bos_v2.0_chapter08_01_第八章主要内容概述.avi 选择视频 修改 删除

bos_v2.0_chapter08_02_运单快速录入业务需求分析.avi 选择视频 修改 删除

选择视频 修改 删除

bos_v2.0_chapter08_15_ElasticSearch下载安装运行.avi 选择视频 修改 删除

bos_v2.0_chapter08_04_datagrid行编辑插入空行实现.avi 选择视频 修改 删除

添加课程计划：

学成在线

教学管理中心

课程管理

课程计划

课程首页

添加课程计划

项目概述

业务需求

订单业务

运单管理

ElasticSearch

ElasticSearch

添加课程计划

上级结点 不填表示根结点

* 章节/课时名称

课程类型 ☐ 视频 ☐ 文档

学习时长 (分钟) 请输入数字

排序字段

章节/课时介绍

* 状态 ☐ 未发布 ☐ 已发布

提交 重置

选择视频 修改 删除

apter08_01_第八章主要内容概述.avi 选择视频 修改 删除

8_02_运单快速录入业务需求分析.avi 选择视频 修改 删除

8_15_ElasticSearch下载安装运行.avi 选择视频 修改 删除

04_datagrid行编辑插入空行实现.avi 选择视频 修改 删除

2.2 教学方法

本模块对课程信息管理功能的教学方法采用实战教学方法，旨在通过实战提高接口编写的能力，具体教学方法如下：

1、前后端工程导入

教学管理前端工程采用与系统管理工程相同的技术，直接导入后在此基础上开发。

课程管理服务端工程采用Spring Boot技术构建，技术层技术使用Spring data Jpa（与Spring data MongoDB类似）、Mybatis，直接导入后在此基础上开发。

2、课程计划功能

课程计划功能采用全程教学。

3、我的课程、新增课程、修改课程、课程营销

我的课程、新增课程、修改课程、课程营销四个功能采用实战方式，课堂上会讲解每个功能的需求及技术点，讲解完成学生开始实战，由导师进行技术指导。

4、参考文档

实战结束提供每个功能的开发文档，学生参考文档并修正功能缺陷。

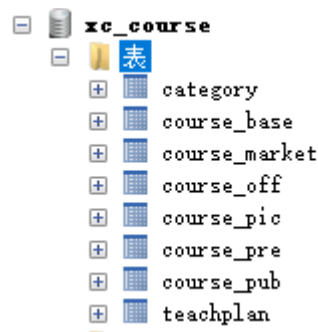
2.3 环境搭建

2.3.1 搭建数据库环境

1) 创建数据库

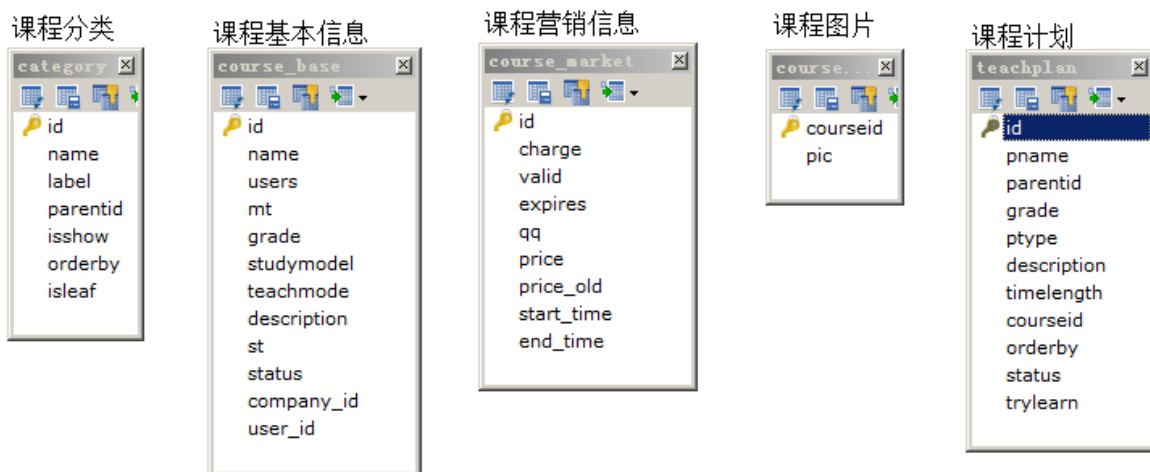
课程管理使用MySQL数据库，创建课程管理数据库：xc_course。

导入xc_course.sql脚本



2) 数据表介绍

课程信息内容繁多，将课程信息分类保存在如下表中：



数据表结构如下：



表: category

Columns (7)

计算适合的数据类

通过读取现有数据查找该表的最佳数据类型。 [详细了解](#)

	Field	Type	Comment
	id	varchar(32) NOT NULL	主键
	name	varchar(32) NOT NULL	分类名称
	label	varchar(32) NULL	分类标签默认和名称一样
	parentid	varchar(32) NULL	父结点id
	isshow	char(1) NULL	是否显示
	orderby	int(4) NULL	排序字段
	isleaf	char(1) NULL	是否叶子



表: course_base

Columns (12)

计算适合的数据类

通过读取现有数据查找该表的最佳数据类型。 [详细了解](#)


	Field	Type	Comment
	id	varchar(32) NOT NULL	
	name	varchar(32) NOT NULL	课程名称
	users	varchar(500) NULL	适用人群
	mt	varchar(32) NOT NULL	课程大分类
	grade	varchar(32) NOT NULL	课程等级
	studymodel	varchar(32) NOT NULL	学习模式
	teachmode	varchar(32) NULL	授课模式
	description	text NULL	课程介绍
	st	varchar(32) NOT NULL	课程小分类
	status	varchar(32) NULL	课程状态
	company_id	varchar(32) NOT NULL	教育机构
	user_id	varchar(32) NULL	创建用户



表: course_market

Columns (9)

计算适合的数据类

通过读取现有数据查找该表的最佳数据类型。 [详细了解](#)


	Field	Type	Comment
	id	varchar(32) NOT NULL	课程id
	charge	varchar(32) NOT NULL	收费规则，对应数据字典
	valid	varchar(32) NOT NULL	有效性，对应数据字典
	expires	timestamp NOT NULL	过期时间
	qq	varchar(32) NULL	咨询qq
	price	float(10,2) NULL	价格
	price_old	float(10,2) NULL	原价
	start_time	datetime NULL	课程有效期-开始时间
	end_time	datetime NULL	课程有效期-结束时间



表: course_pic

Columns (2)

计算适合的数据类

通过读取现有数据查找该表的最佳数据类型。 [详细了解](#)

	Field	Type	Comment
🔑	courseid	varchar(32) NOT NULL	课程id
	pic	varchar(256) NOT NULL	图片id



表: teachplan

Columns (11)

计算适合的数据类

通过读取现有数据查找该表的最佳数据类型。 [详细了解](#)

	Field	Type	Comment
🔑	id	varchar(32) NOT NULL	
	pname	varchar(64) NOT NULL	
	parentid	varchar(32) NULL	
	grade	char(1) NULL	层级，分为1、2、3级
	ptype	char(1) NULL	课程类型:1视频、2文档
	description	varchar(500) NULL	章节及课程时介绍
	timelength	double(5,2) NULL	时长，单位分钟
	courseid	varchar(32) NULL	课程id
	orderby	varchar(32) NULL	排序字段
	status	char(1) NULL	状态：未发布、已发布
	trylearn	char(1) NULL	是否试学

2.3.2导入课程管理服务工程

1) 持久层技术介绍：

课程管理服务使用MySQL数据库存储课程信息，持久层技术如下：

1、spring data jpa：用于表的基本CRUD。

2、mybatis：用于复杂的查询操作。

3、druid：使用阿里巴巴提供的spring boot 整合druid包druid-spring-boot-starter管理连接池。

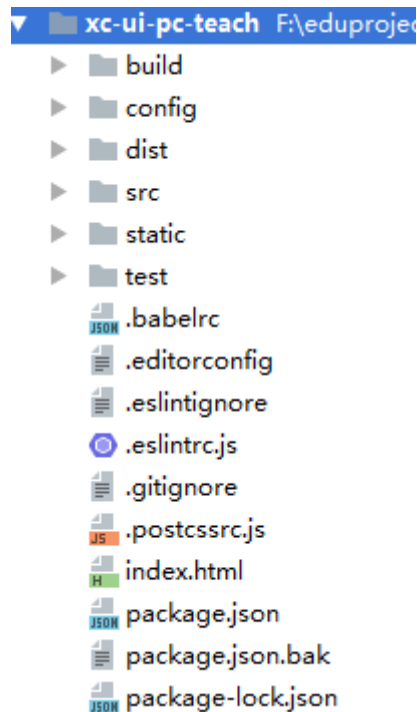
druid-spring-boot-starter地址：<https://github.com/alibaba/druid/tree/master/druid-spring-boot-starter>

2) 导入工程

导入资料下的“xc-service-manage-course.zip”。

2.3.3 导入课程管理前端工程

课程管理属于教学管理子系统的功能，使用用户为教学机构的管理人员和老师，为保证系统的可维护性，单独创建一个教学管理前端工程。教学管理前端工程与系统管理前端的工程结构一样，也采用vue.js框架来实现。



从课程资料目录拷贝xc-ui-pc-teach.zip到工程，使用webstorm打开，启动工程：

效果图如下：



3 课程计划

3.1 需求分析

什么是课程计划？

课程计划定义了课程的章节内容，学生通过课程计划进行在线学习，下图中右侧显示的就是课程计划。

课程计划包括两级，第一级是课程的大章节、第二级是大章节下属的小章节，每个小章节通常是一段视频，学生点击小章节在线学习。

教学管理人员对课程计划如何管理？

功能包括：添加课程计划、删除课程计划、修改课程计划等。



3.2 课程计划查询

3.2.1 需求分析

课程计划查询是将某个课程的课程计划内容完整的显示出来，如下图所示：



左侧显示的就是课程计划，课程计划是一个树型结构，方便扩展课程计划的级别。

在上边页面中，点击“添加课程计划”即可对课程计划进行添加操作。

点击修改可对某个章节内容进行修改。

点击删除可删除某个章节。

3.2.2 页面原型

3.2.2.1 tree组件介绍

本功能使用element-ui的tree组件来完成

使用 render-content

▼ ☐ 一级 1

Append Delete

▼ ☐ 二级 1-1

Append Delete

☐ 三级 1-1-1

Append Delete

☐ 三级 1-1-2

Append Delete

☐ testtest

Append Delete

▼ ☐ 一级 2

Append Delete

☐ 二级 2-1

Append Delete

☐ 二级 2-2

Append Delete

▼ ☐ 一级 3

Append Delete

☐ 二级 3-1

Append Delete

☐ 二级 3-2

Append Delete

使用 scoped slot

▼ ☐ 一级 1

Append Delete

▼ ☐ 二级 1-1

Append Delete

☐ 三级 1-1-1

Append Delete

☐ 三级 1-1-2

Append Delete

▼ ☐ 一级 2

Append Delete

☐ 二级 2-1

Append Delete

☐ 二级 2-2

Append Delete

▼ ☐ 一级 3

Append Delete

☐ 二级 3-1

Append Delete

☐ 二级 3-2

Append Delete

在course_plan.vue文件中添加tree组件的代码，进行测试：

1、组件标签

```
<el-tree
  :data="data"
  show-checkbox
  node-key="id"
  default-expand-all
  :expand-on-click-node="false"
  :render-content="renderContent">
</el-tree>
```

2、数据对象

```
let id = 1000;

export default {
  data() {
    return {
      data : [{
        id: 1,
        label: '一级 1',
        children: [{
          id: 4,
          label: '二级 1-1',
          children: [{
            id: 9,
            label: '三级 1-1-1'
          ]
        }
      ]
    }
  ]
}
```

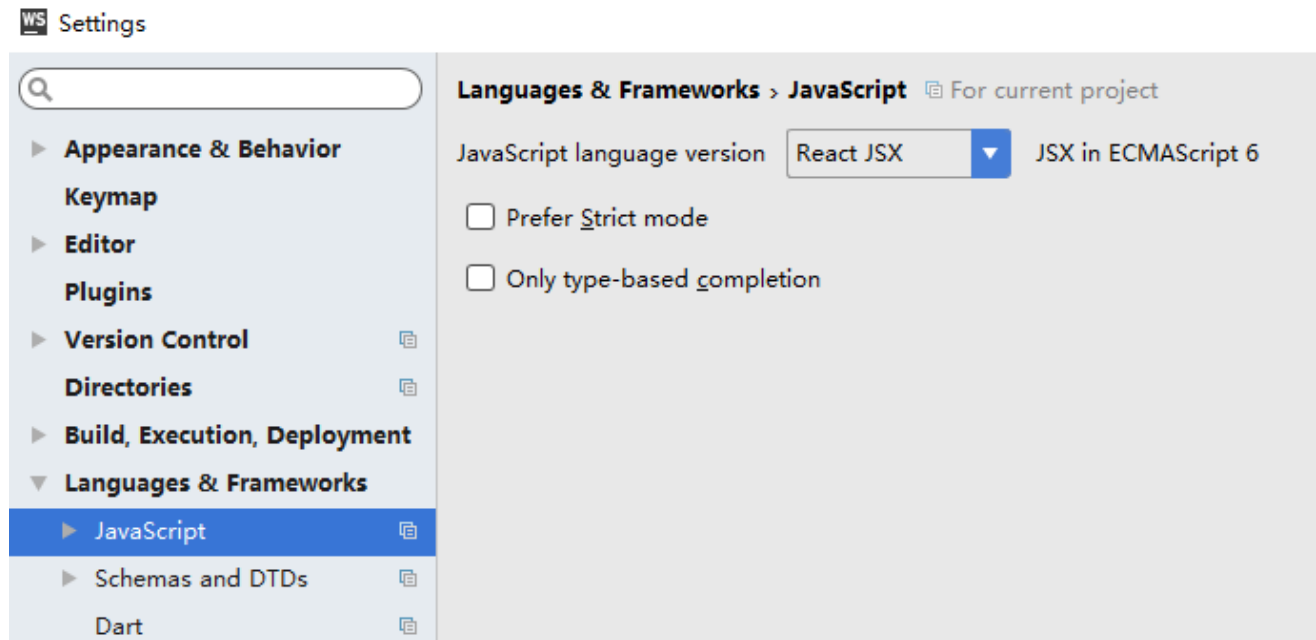

3.2.2.2 webstorm配置JSX

JSX 是 Javascript 和 XML 结合的一种格式，它是 React 的核心组成部分，JSX 和 XML 语法类似，可以定义属性以及子元素。唯一特殊的是可以用大括号来加入 JavaScript 表达式。遇到 HTML 标签（以 < 开头），就用 HTML 规则解析；遇到代码块（以 { 开头），就用 JavaScript 规则解析。

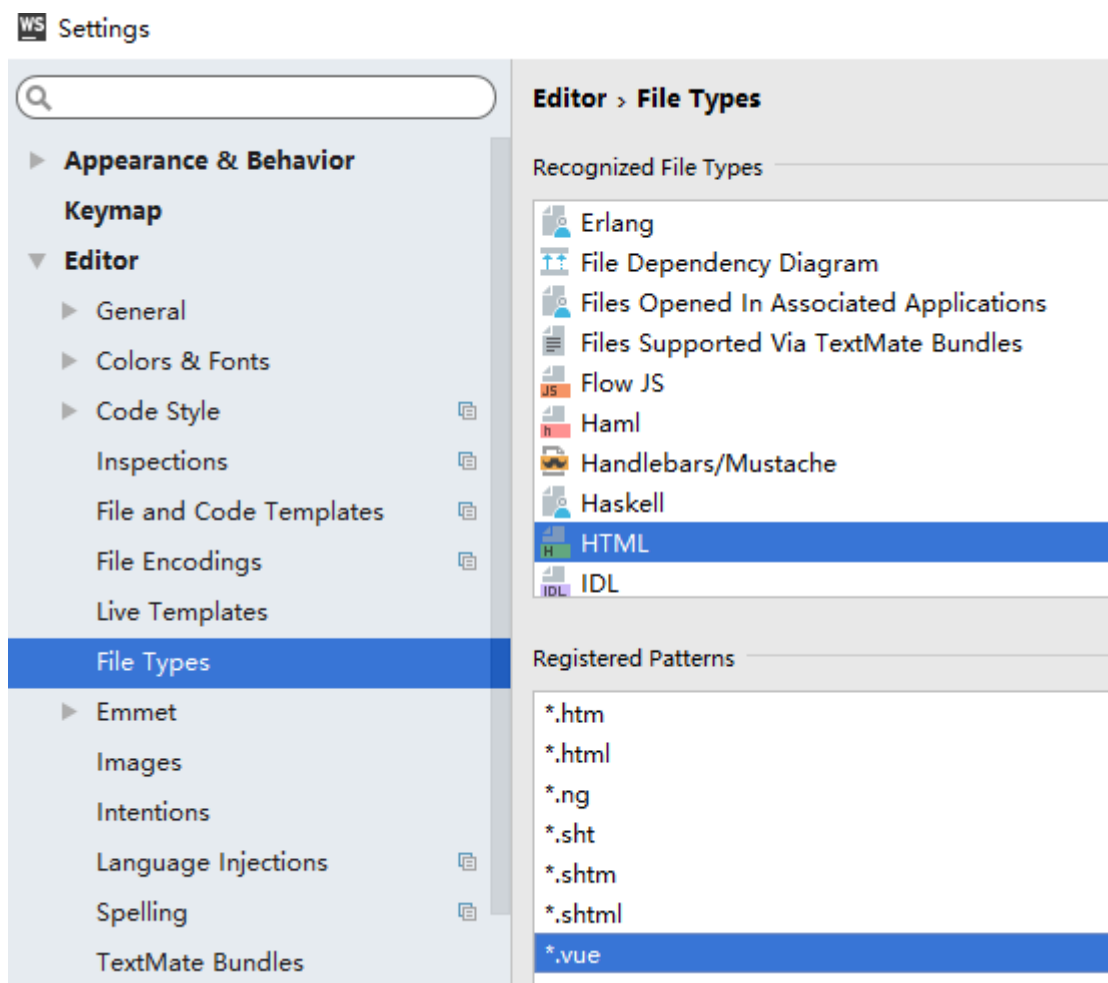
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello React!</title>
    <script src="vendor-js/react.js"></script>
    <script src="vendor-js/react-dom.js"></script>
    <script src="vendor-js/babel-core/browser.min.js"></script>
  </head>
  <body>
    <div id="example"></div>
    <script type="text/babel">
      ReactDOM.render(
        <h1>Hello, world!</h1>,
        document.getElementById('example')
      );
    </script>
  </body>
</html>
```

北京市昌平区建材城西路金燕龙办公楼一层 电话: 400-618-9090

1、Javascript version 选择 React JSX (如果没有就选择JSX Harmony)

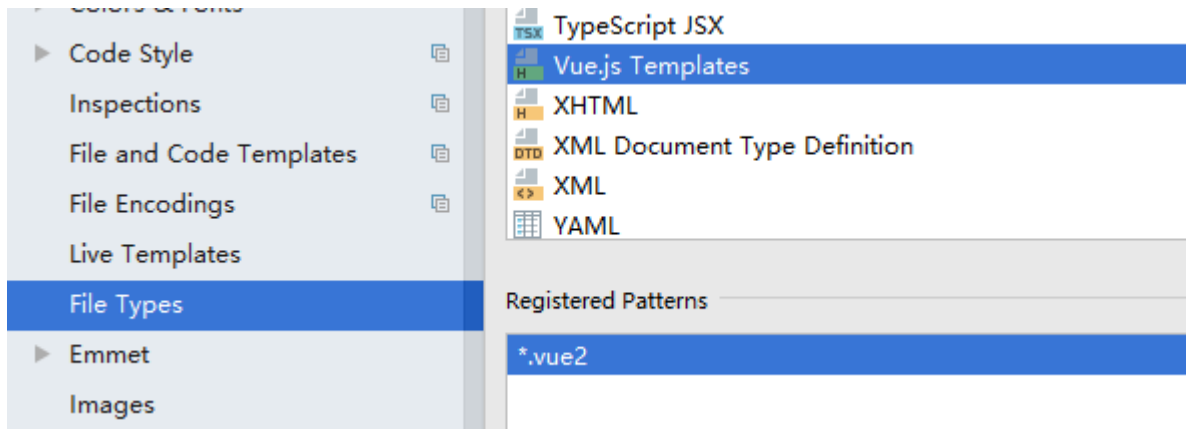


2、HTML 类型文件中增加vue



preferences -> Editor -> File Types 中找到上边框中HTML 在下边加一个 *.vue


如果已经在vue template 中已存在.vue 则把它改为.vue2(因为要在Html中添加.vue)



3.2.3 API接口

3.2.3.1 数据模型


1、表结构

 表: teachplan

Columns (11)

计算适合的数据类

通过读取现有数据查找该表的最佳数据类型。 [详细了解](#)

	Field	Type	Comment
	id	varchar(32) NOT NULL	
	pname	varchar(64) NOT NULL	
	parentid	varchar(32) NOT NULL	
	grade	char(1) NOT NULL	层级，分为1、2、3级
	ptype	char(1) NULL	课程类型:1视频、2文档
	description	varchar(500) NULL	章节及课程时介绍
	timelength	double(5,2) NULL	时长，单位分钟
	courseid	varchar(32) NULL	课程id
	orderby	varchar(32) NULL	排序字段
	status	char(1) NOT NULL	状态：未发布、已发布
	trylearn	char(1) NULL	是否试学

2、模型类

课程计划为树型结构，由树根（课程）和树枝（章节）组成，为了保证系统的可扩展性，在系统设计时将课程计划设置为树型结构。

@Data

@ToString

```
@Entity
@Table(name="teachplan")
@GenericGenerator(name = "jpa-uuid", strategy = "uuid")
public class Teachplan implements Serializable {
    private static final long serialVersionUID = -916357110051689485L;
    @Id
    @GeneratedValue(generator = "jpa-uuid")
    @Column(length = 32)
    private String id;
    private String pname;
    private String parentid;
    private String grade;
    private String ptype;
    private String description;
    private String courseid;
    private String status;
    private Integer orderby;
    private Double timelength;
    private String trylearn;
}
```

3.2.3.2 自定义模型类

前端页面需要树型结构的数据来展示Tree组件，如下：

```
[{
  id: 1,
  label: '一级 1',
  children: [{
    id: 4,
    label: '二级 1-1'
  }]
}]
```

自定义课程计划结点类如下：

```
@Data
@ToString
public class TeachplanNode extends Teachplan {
    List<TeachplanNode> children;
}
```

3.2.3.3 接口定义

根据课程id查询课程的计划接口如下，在api工程创建course包，创建CourseControllerApi接口类并定义接口方法如下：

```
public interface CourseControllerApi {  
    @ApiOperation("课程计划查询")  
    public TeachplanNode findTeachplanList(String courseId);  
}
```

3.2.3 课程管理服务

3.2.3.1 Sql

课程计划是树型结构，采用表的自连接方式进行查询，sql语句如下：

```
SELECT  
    a.id one_id,  
    a.pname one_pname,  
    b.id two_id,  
    b.pname two_pname,  
    c.id three_id,  
    c.pname three_pname  
FROM  
    teachplan a  
    LEFT JOIN teachplan b  
        ON a.id = b.parentid  
    LEFT JOIN teachplan c  
        ON b.id = c.parentid  
WHERE a.parentid = '0'  
    AND a.courseid = '402885816243d2dd016243f24c030002'  
ORDER BY a.orderby,  
    b.orderby,  
    c.orderby
```

3.2.3.2 Dao

1) mapper接口

```
@Mapper  
public interface TeachplanMapper {  
    public TeachplanNode selectList(String courseId);  
}
```

2) mapper映射文件

```
<resultMap type="com.xuecheng.framework.domain.course.ext.TeachplanNode" id="teachplanMap" >  
    <id property="id" column="one_id"/>  
    <result property="pname" column="one_name"/>  
  
    <collection property="children"
```

```
ofType="com.xuecheng.framework.domain.course.ext.TeachplanNode">
    <id property="id" column="two_id"/>
    <result property="pname" column="two_name"/>
    <collection property="children"
ofType="com.xuecheng.framework.domain.course.ext.TeachplanNode">
    <id property="id" column="three_id"/>
    <result property="pname" column="three_name"/>
    </collection>
</collection>
</resultMap>

<select id="selectList" resultMap="teachplanMap" parameterType="java.lang.String" >
    SELECT
        a.id one_id,
        a.pname one_name,
        b.id two_id,
        b.pname two_name,
        c.id three_id,
        c.pname three_name

    FROM
        teachplan a LEFT JOIN teachplan b
            ON a.id = b.parentid
        LEFT JOIN teachplan c
            ON b.id = c.parentid
    WHERE a.parentid = '0'
    <if test="_parameter!=null and _parameter!=''">
        and a.courseid=#{courseId}
    </if>

    ORDER BY a.orderby,
        b.orderby,
        c.orderby
</select>
```

说明：针对输入参数为简单类型#{ }中可以是任意类型，判断参数是否为空要用 _parameter（它属于mybatis的内置参数）

3.4.3.3 Service

创建CourseService类，定义查询课程计划方法。

```
@Service
public class CourseService {
    @Autowired
    TeachplanMapper teachplanMapper;
    //查询课程计划
    public TeachplanNode findTeachplanList(String courseId){
        TeachplanNode teachplanNode = teachplanMapper.selectList(courseId);
        return teachplanNode;
    }
}
```

3.4.3.4 Controller

```
@RestController
@RequestMapping("/course")
public class CourseController implements CourseControllerApi {
    @Autowired
    CourseService courseService;

    //查询课程计划
    @Override
    @GetMapping("/teachplan/list/{courseId}")
    public TeachplanNode findTeachplanList(String courseId) {
        return courseService.findTeachplanList(courseId);
    }
}
```

3.4.3.5 测试

使用postman或swagger-ui测试查询接口。

Get 请求：<http://localhost:31200/course/teachplan/list/402885816243d2dd016243f24c030002>

3.2.4 前端页面

3.2.4.1 Api方法

定义课程计划查询的api方法：

```
/*查询课程计划*/
export const findTeachplanList = courseid => {
    return http.requestQuickGet(apiUrl+'/course/teachplan/list/'+courseid)
}
```

3.2.4.2 Api调用

1、在mounted钩子方法 中查询 课程计划

定义查询课程计划的方法，赋值给数据对象teachplanList

```
findTeachplan(){
  courseApi.findTeachplanList(this.courseid).then((res) => {
    this.teachplanList = []; //清空树
    if(res.children){
      this.teachplanList = res.children;
    }
  });
}
```

2) 在mounted钩子中查询课程计划

```
mounted(){
  //课程id
  this.courseid = this.$route.params.courseid;
  //课程计划
  this.findTeachplan();
}
```

3) 修改树结点的标签属性

课程计划信息中pname为结点的名称，需要修改树结点的标签属性方可正常显示课程计划名称，如下：

```
defaultProps: {
  children: 'children',
  label: 'pname'
}
```

3.2.4.3 测试

课程计划

课程首页	基本信息	课程图片	课程营销	课程计划	教师信息	发布课程	
▼ 程序入门							修改 删除
Hello World							修改 删除
▼ 编程基础							修改 删除
表达式							修改 删除
逻辑运算							修改 删除
▼ 面向对象							修改 删除
一切皆为对象							修改 删除

3.3 添加课程计划

3.3.1 需求分析

用户操作流程：

- 1、进入课程计划页面，点击“添加课程计划”
- 2、打开添加课程计划页面，输入课程计划信息



上级结点说明：

不选择上级结点表示当前课程计划为该课程的一级结点。

当添加该课程在课程计划中还没有节点时要自动添加课程的根结点。

- 3、点击提交。

3.3.1.1 页面原型说明

添加课程计划采用弹出窗口组件Dialog。

- 1、视图部分

在course_plan.vue页面添加添加课程计划的弹出窗口代码：

```
<el-dialog title="添加课程计划" :visible.sync="teachplayFormVisible" >
  <el-form ref="teachplayForm" :model="teachplanActive" label-width="140px"
    style="width:600px;" :rules="teachplanRules" >
    <el-form-item label="上级结点" >
      <el-select v-model="teachplanActive.parentid" placeholder="不填表示根结点">
        <el-option
          v-for="item in teachplanList"
          :key="item.id"
          :label="item.pname"
          :value="item.id">
        </el-option>
      </el-select>
    </el-form-item>
    <el-form-item label="章节/课时名称" prop="pname">
```

```
<el-input v-model="teachplanActive.pname" auto-complete="off"></el-input>
</el-form-item>
<el-form-item label="课程类型" >
  <el-radio-group v-model="teachplanActive.ptype">
    <el-radio class="radio" label='1'>视频</el-radio>
    <el-radio class="radio" label='2'>文档</el-radio>
  </el-radio-group>
</el-form-item>
<el-form-item label="学习时长（分钟） 请输入数字" >
  <el-input type="number" v-model="teachplanActive.timelength" auto-complete="off" ></el-
input>
</el-form-item>
<el-form-item label="排序字段" >
  <el-input v-model="teachplanActive.orderby" auto-complete="off" ></el-input>
</el-form-item>
<el-form-item label="章节/课时介绍" prop="description">
  <el-input type="textarea" v-model="teachplanActive.description" ></el-input>
</el-form-item>

<el-form-item label="状态" prop="status">
  <el-radio-group v-model="teachplanActive.status" >
    <el-radio class="radio" label="0" >未发布</el-radio>
    <el-radio class="radio" label='1'>已发布</el-radio>
  </el-radio-group>
</el-form-item>
<el-form-item >
  <el-button type="primary" v-on:click="addTeachplan">提交</el-button>
  <el-button type="primary" v-on:click="resetForm">重置</el-button>
</el-form-item>

</el-form>
</el-dialog>
```

2、数据模型

在数据模型中添加如下变量：

```
在数据对象中添加：
teachplayFormVisible:false,
teachplanRules: {
  pname: [
    {required: true, message: '请输入课程计划名称', trigger: 'blur'}
  ],
  status: [
    {required: true, message: '请选择状态', trigger: 'blur'}
  ]
},
teachplanActive:{},
```

3、添加按钮

通过变量teachplayFormVisible控制弹出窗口是否显示。

```
<el-button type="primary" @click="teachplayFormVisible = true">添加课程计划</el-button>
```

4、定义表单提交方法和重置方法

```
//提交课程计划
addTeachplan(){
    alert()
},
//重置表单
resetForm(){
    this.teachplanActive = {}
},
```

3.3.3 API接口

1) 添加课程计划

```
@ApiOperation("添加课程计划")
public ResponseResult addTeachplan(Teachplan teachplan);
```

3.3.4 课程管理服务

3.3.3.1 Dao

```
public interface TeachplanRepository extends JpaRepository<Teachplan, String> {
    //定义方法根据课程id和父结点id查询出结点列表，可以使用此方法实现查询根结点
    public List<Teachplan> findByCourseidAndParentid(String courseId,String parentId);
}
```

3.3.3.2 Service

```
//获取课程根结点，如果没有则添加根结点
public String getTeachplanRoot(String courseId){
    //校验课程id
    Optional<CourseBase> optional = courseBaseRepository.findById(courseId);
    if(!optional.isPresent()){
        return null;
    }
    CourseBase courseBase = optional.get();

    //取出课程计划根结点
```



```
List<Teachplan> teachplanList = teachplanRepository.findByCourseidAndParentid(courseId,
"0");
if(teachplanList == null || teachplanList.size()==0){
    //新增一个根结点
    Teachplan teachplanRoot = new Teachplan();
    teachplanRoot.setCourseid(courseId);
    teachplanRoot.setPname(courseBase.getName());
    teachplanRoot.setParentid("0");
    teachplanRoot.setGrade("1");//1级
    teachplanRoot.setStatus("0");//未发布
    teachplanRepository.save(teachplanRoot);
    return teachplanRoot.getId();
}
Teachplan teachplan = teachplanList.get(0);
return teachplan.getId();

}
//添加课程计划
@Transactional
public ResponseResult addTeachplan(Teachplan teachplan){
    //校验课程id和课程计划名称
    if(teachplan == null ||
        StringUtils.isEmpty(teachplan.getCourseid()) ||
        StringUtils.isEmpty(teachplan.getPname())){
        ExceptionCast.cast(CommonCode.INVALIDPARAM);
    }

    //取出课程id
    String courseid = teachplan.getCourseid();
    //取出父结点id
    String parentid = teachplan.getParentid();
    if(StringUtils.isEmpty(parentid)){
        //如果父结点为空则获取根结点
        parentid= getTeachplanRoot(courseid);
    }
    //取出父结点信息
    Optional<Teachplan> teachplanOptional = teachplanRepository.findById(parentid);
    if(!teachplanOptional.isPresent()){
        ExceptionCast.cast(CommonCode.INVALIDPARAM);
    }
    //父结点
    Teachplan teachplanParent = teachplanOptional.get();
    //父结点的级别
    String parentGrade = teachplanParent.getGrade();
    //设置父结点
    teachplan.setParentid(parentid);
    teachplan.setStatus("0");//未发布
    //子结点的级别，根据父结点来判断
    if(parentGrade.equals("1")){
        teachplan.setGrade("2");
    }else if(parentGrade.equals("2")){

        teachplan.setGrade("3");
    }
}
```

```
    }  
    //设置课程id  
    teachplan.setCourseid(teachplanParent.getCourseid());  
    teachplanRepository.save(teachplan);  
    return new ResponseResult(CommonCode.SUCCESS);  
}
```

3.3.3.3 controller

```
//添加课程计划  
@Override  
@PostMapping("/teachplan/add")  
public ResponseResult addTeachplan(@RequestBody Teachplan teachplan) {  
    return courseService.addTeachplan(teachplan);  
}
```

3.3.3.4 测试

复杂一些的业务逻辑建议写完服务端代码就进行单元测试。

使用swagger-ui或postman测试上边的课程计划添加接口。

3.3.5 前端

3.3.5.1 Api调用

1、定义 api方法

```
/*添加课程计划*/  
export const addTeachplan = teachplan => {  
    return http.requestPost(apiUrl+'course/teachplan/add',teachplan)  
}
```

2、调用 api

```
addTeachplan(){  
    this.$refs.teachplanForm.validate((valid) => {  
        if (valid) {  
            //添加课程计划时带上课程id  
            this.teachplanActive.courseid = this.courseid;  
            courseApi.addTeachplan(this.teachplanActive).then((res) => {  
                if(res.success){  
                    this.$message.success('提交成功');  
                    //清空表单  
                    this.teachplanActive = {}  
                    //刷新整个树  
  
                    this.findTeachplan();  
                }  
            })  
        }  
    })  
}
```

```
        }else{
            this.$message.error('提交失败');
        }
    });
}
})
},
```

3.3.5 测试

测试流程：

- 1、新建一个课程
- 2、向新建课程中添加课程计划

添加一级结点

添加二级结点