学成在线 第8天 讲义-课程图片管理 分布式文件系统

1 FastDFS研究

参考 “分布式文件系统 fastDFS研究.md”

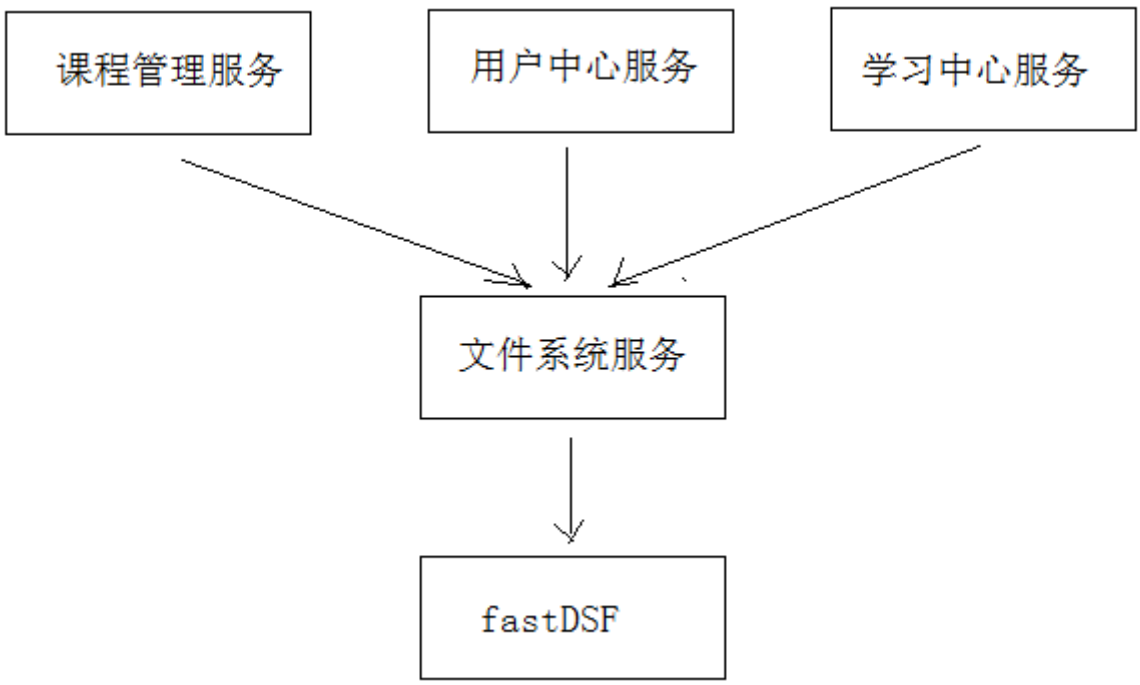
2 上传图片开发

1.1.1 需求分析

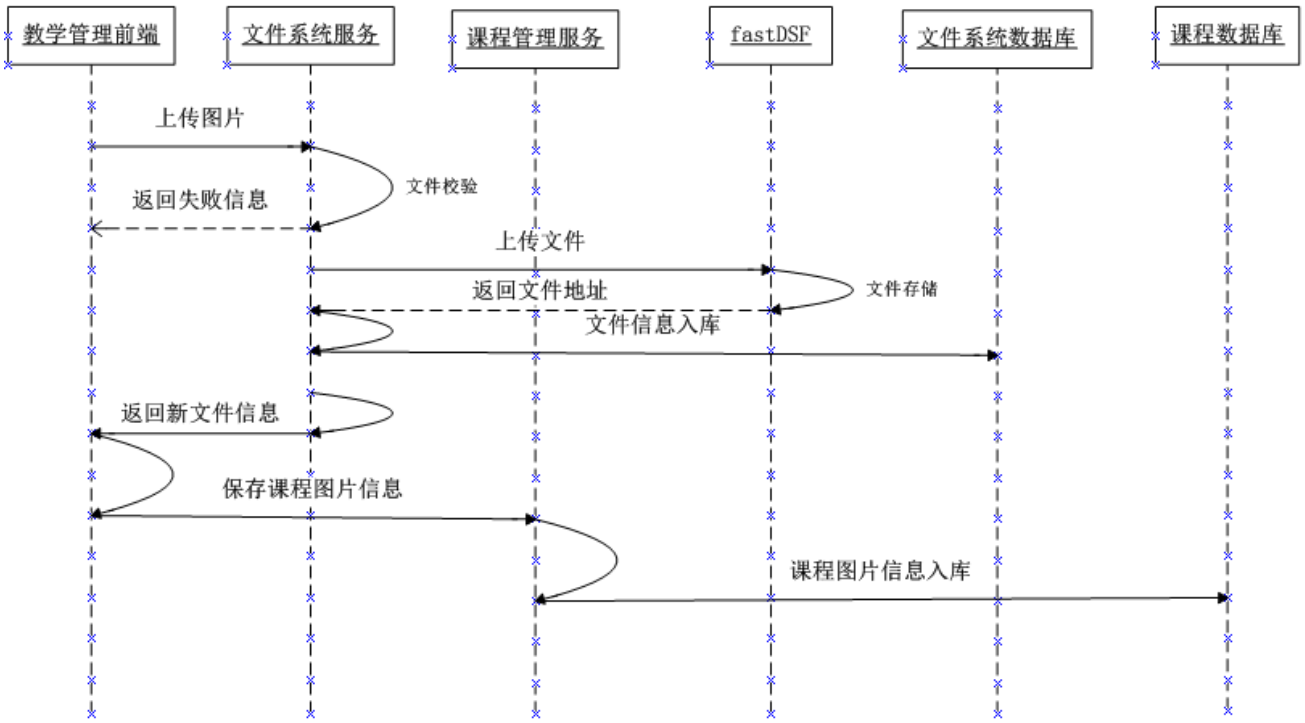
在很多系统都有上传图片/上传文件的需求，比如：上传课程图片、上传课程资料、上传用户头像等，为了提供系统的可重用性专门设立文件系统服务承担图片/文件的管理，文件系统服务实现对文件的上传、删除、查询等功能进行管理。

各各子系统不再开发上传文件的请求，各各子系统通过文件系统服务进行文件的上传、删除等操作。文件系统服务最终会将文件存储到fastDSF文件系统中。

下图是各各子系统与文件系统服务之间的关系：



下图是课程管理中上传图片处理流程：



执行流程如下：

1、管理员进入教学管理前端，点击上传图片

2、图片上传至文件系统服务，文件系统请求fastDFS上传文件

3、文件系统将文件入库，存储到文件系统服务数据库中。

4、文件系统服务向前端返回文件上传结果，如果成功则包括文件的Url路径。

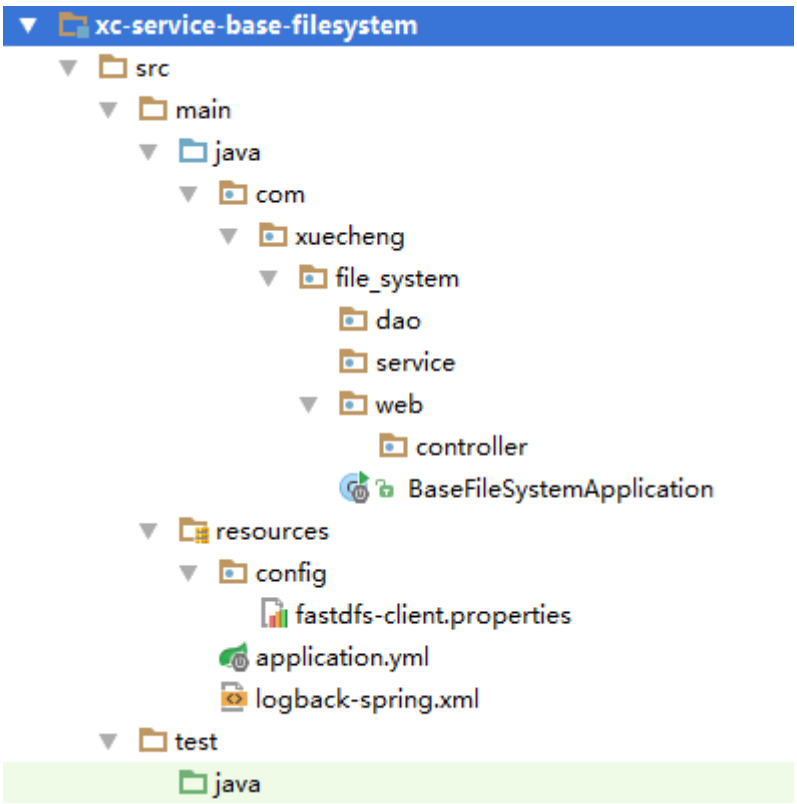
5、课程管理前端请求课程管理进行保存课程图片信息到课程数据库。

6、课程管理服务将课程图片保存在课程数据库。

1.1.2 创建文件系统服务工程

导入xc-service-base-filesystem.zip工程。

1）工程目录结构



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-base-filesystem</artifactId>
    <dependencies>
        <dependency>
            <groupId>com.xuecheng</groupId>
            <artifactId>xc-service-api</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>com.xuecheng</groupId>
            <artifactId>xc-framework-model</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>com.xuecheng</groupId>
```

```
            <artifactId>xc-framework-common</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>net.oschina.zcx7878</groupId>
            <artifactId>fastdfs-client-java</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-io</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
    </dependencies>

</project>
```

2）配置文件

原测试程序中fastdfs-client.properties的配置信息统一放在application.yml

application.yml

```
server:
  port: 22100
spring:
  application:
    name: xc-service-base-filesystem
#mongo配置
  data:
    mongodb:
      database: xc_fs
      uri: mongodb://root:123@127.0.0.1:27017
#SpringMVC上传文件配置
  servlet:
    multipart:
      #默认支持文件上传.
      enabled: true
      #支持文件写入磁盘.
```

```
      file-size-threshold: 0
      # 上传文件的临时目录
      location:
      # 最大支持文件大小
      max-file-size: 1MB
      # 最大支持请求大小
      max-request-size: 30MB
xuecheng:
  fastdfs:
    connect_timeout_in_seconds: 5
    network_timeout_in_seconds: 30
    charset: UTF-8
    tracker_servers: 192.168.101.64:22122
```

1.1.3 API接口

1.1.3.1模型类

系统的文件信息（图片、文档等小文件的信息）在mongodb中存储，下边是文件信息的模型类。

1) 模型如下：

```
@Data
@ToString
@Document(collection = "filesystem")
public class FileSystem {

    @Id
    private String fileId;
    //文件请求路径
    private String filePath;
    //文件大小
    private long fileSize;
    //文件名称
    private String fileName;
    //文件类型
    private String fileType;
    //图片宽度
    private int fileWidth;
    //图片高度
    private int fileHeight;
    //用户id，用于授权暂时不用
    private String userId;
    //业务key
    private String businesskey;
    //业务标签
    private String filetag;
    //文件元信息
    private Map metadata;


}
```

说明：

fileId：fastDFS返回的文件ID。

filePath：请求fastDFS浏览文件URL。

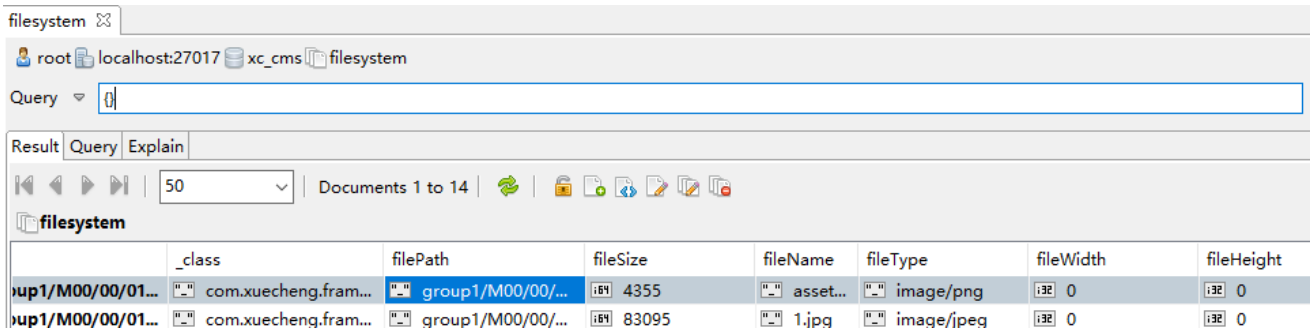
filetag：文件标签，由于文件系统服务是公共服务，文件系统服务会为使用文件系统服务的子系统分配文件标签，用于标识此文件来自哪个系统。

businesskey：文件系统服务为其它子系统提供的一个业务标识字段，各子系统根据自己的需求去使用，比如：课程管理会在此字段中存储课程id用于标识该图片属于哪个课程。

metadata：文件相关的元信息。

2) collection

在mongodb创建数据库xc_fs（文件系统数据库），并创建集合 filesystem。



1.1.3.2 Api接口

在api工程下创建com.xuecheng.api.filesystem包，

```
public interface FileSystemControllerApi {

    /**
     * 上传文件
     * @param multipartFile 文件
     * @param filetag 文件标签
     * @param businesskey 业务key
     * @param metedata 元信息,json格式
     * @return
     */
    public UploadFileResult upload(MultipartFile multipartFile,
                   String filetag,
                   String businesskey,
                   String metadata);
    }
```

1.1.2.3 Dao

将文件信息存入数据库，主要存储文件系统中的文件路径。

```
public interface FileSystemRepository extends MongoRepository<FileSystem,String> {

}
```

1.1.2.4 Service

```
@Service
public class FileSystemService {

    private static final Logger LOGGER = LoggerFactory.getLogger(FileSystemService.class);

    @Value("${xuecheng.fastdfs.tracker_servers}")
    String tracker_servers;
    @Value("${xuecheng.fastdfs.connect_timeout_in_seconds}")
    int connect_timeout_in_seconds;
    @Value("${xuecheng.fastdfs.network_timeout_in_seconds}")
    int network_timeout_in_seconds;
    @Value("${xuecheng.fastdfs.charset}")
    String charset;

    @Autowired
    FileSystemRepository fileSystemRepository;

    //加载fdfs的配置
    private void initFdfsConfig(){
        try {
            ClientGlobal.initByTrackers(tracker_servers);
            ClientGlobal.setG_connect_timeout(connect_timeout_in_seconds);
            ClientGlobal.setG_network_timeout(network_timeout_in_seconds);
            ClientGlobal.setG_charset(charset);
        } catch (Exception e) {
            e.printStackTrace();
            //初始化文件系统出错
            ExceptionCast.cast(FileSystemCode.FS_INITFDFSERROR);
        }
    }
    //上传文件
    public UploadFileResult upload(MultipartFile file,
                                   String filetag,
                                   String businesskey,
                                   String metadata){

        if(file == null){
            ExceptionCast.cast(FileSystemCode.FS_UPLOADFILE_FILEISNULL);
        }

        //上传文件到fdfs
```

```
        String fileId = fdfs_upload(file);
        //创建文件信息对象
        FileSystem fileSystem = new FileSystem();
        //文件id
        fileSystem.setFileId(fileId);
        //文件在文件系统中的路径
        fileSystem.setFilePath(fileId);
        //业务标识
        fileSystem.setBusinesskey(businesskey);
        //标签
        fileSystem.setFiletag(filetag);
        //元数据
        if(StringUtils.isNotEmpty(metadata)){
            try {
                Map map = JSON.parseObject(metadata, Map.class);
                fileSystem.setMetadata(map);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        //名称
        fileSystem.setFileName(file.getOriginalFilename());
        //大小
        fileSystem.setFileSize(file.getSize());
        //文件类型
        fileSystem.setFileType(file.getContentType());
        fileSystemRepository.save(fileSystem);
        return new UploadFileResult(CommonCode.SUCCESS,fileSystem);

    }

    //上传文件到fdfs，返回文件id
    public String fdfs_upload(MultipartFile file) {
        try {
            //加载fdfs的配置
            initFdfsConfig();
            //创建tracker client
            TrackerClient trackerClient = new TrackerClient();
            //获取trackerServer
            TrackerServer trackerServer = trackerClient.getConnection();
            //获取storage
            StorageServer storeStorage = trackerClient.getStoreStorage(trackerServer);
            //创建storage client
            StorageClient1 storageClient1 = new StorageClient1(trackerServer,storeStorage);
            //上传文件
            //文件字节
            byte[] bytes = file.getBytes();
            //文件原始名称
            String originalFilename = file.getOriginalFilename();
            //文件扩展名
            String extName = originalFilename.substring(originalFilename.lastIndexOf(".") + 1);
            //文件id

            String file1 = storageClient1.upload_file1(bytes, extName, null);
```

```
            return file1;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

}
```

1.1.2.5 Controller

```
@RestController
@RequestMapping("/filesystem")
public class FileSystemController implements FileSystemControllerApi {
    @Autowired
    FileSystemService fileSystemService;

    @Override
    @PostMapping("/upload")                                                    对应下面的name
    public UploadFileResult upload(@RequestParam("file") MultipartFile file,
                                   @RequestParam(value = "filetag", required = true) String
filetag,
                                   @RequestParam(value = "businesskey", required = false) String
businesskey,
                                   @RequestParam(value = "metedata", required = false) String
metadata) {
        return fileSystemService.upload(file,filetag,businesskey,metadata);
    }
}
```

1.1.2.6 测试

使用swagger-ui或postman进行测试。

下图是使用swagger-ui进行测试的界面：

Parameters

Parameter	Value	Description	Parameter Type	Data Type
file	选择文件 logo.png	file	formData	file
filetag	course	filetag	query	string
businesskey	21343	businesskey	query	string
metedata	{"name":"测试文件"}	metedata	query	string

Response Messages

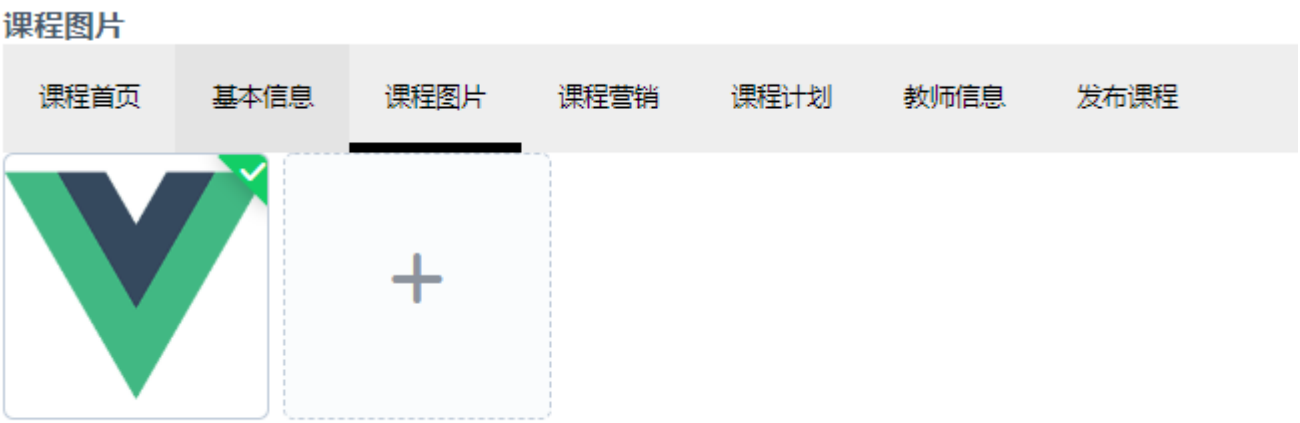
HTTP Status Code	Reason	Response Model	Headers
201	Created		
401	Unauthorized		
403	Forbidden		
404	Not Found		

Try it out! Hide Response

1.1.3 上传课程图片前端

1.1.3.1 需求

上传图片界面如下图：



点击“加号”上传图片，图片上传成功自动显示；点击“删除”将删除图片。

1.1.3.2 页面

使用Element-UI的Upload上传组件实现上边的效果。

1) template

```
<el-upload
  action="/filesystem/upload"   action="/api/filesystem/upload"   api是代理的部分
  list-type="picture-card"
  :before-upload="setbusinesskey"
  :on-success="handleSuccess"
  :file-list="fileList"
  :limit="picmax"
  :on-exceed="rejectupload"
  :data="uploadval">
  <i class="el-icon-plus"></i>
</el-upload>
```

这里要加一个name

el-upload参数说明：

action：必选参数，上传的地址

list-type：文件列表的类型（text/picture/picture-card）

before-upload：上传前执行钩子方法 ，function(file)

on-success：上传成功 执行的钩子方法 ，function(response, file, fileList)

on-error：上传失败的钩子方法，function(err, file, fileList)

on-remove：文件删除的钩子方法，function(file, fileList)

file-list：文件列表，此列表为上传成功 的文件

limit：最大允许上传个数

on-exceed：文件超出个数限制时的钩子，方法为：function(files, fileList)

data：提交上传的额外参数，需要封装为json对象，最终提交给服务端为key/value串

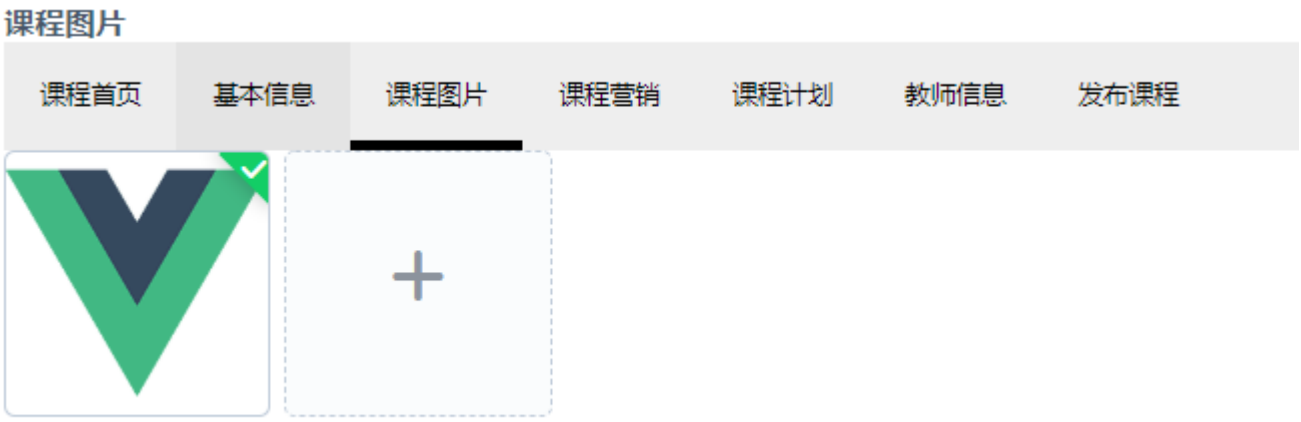
2)数据模型

```
<script>
  import * as sysConfig from '@/../config/sysConfig';
  import * as courseApi from '../../api/course';
  import utilApi from '../../../../common/utils';
  import * as systemApi from '../../../../base/api/system';
  export default {
    data() {
      return {
        picmax:1,
        courseid:'',
        dialogImageUrl: '',
        dialogVisible: false,
        fileList:[],
        uploadval:{filetag:"course"},
        imgUrl:sysConfig.imgUrl
      }

    },
```

```
    methods: {
      //超出文件上传个数提示信息
      rejectupload(){
        this.$message.error("最多上传"+this.picmax+"个图片");
      },
      //在上传前设置上传请求的数据
      setuploaddata(){

      },
      //删除图片
      handleRemove(file, fileList) {
        console.log(file)

        alert('删除')

      },
      //上传成功的钩子方法
      handleSuccess(response, file, fileList){
        console.log(response)
        alert('上传成功')

      },
      //上传失败执行的钩子方法
      handleError(err, file, fileList){
        this.$message.error('上传失败');
        //清空文件队列
        this.fileList = []
      }
    },
    mounted(){
      //课程id
      this.courseid = this.$route.params.courseid;

    }
  }
</script>
```

1.1.3.3 测试

1、点击“加号”测试上传图片。



3 保存课程图片

1.2.1 需求分析

图片上传到文件系统后，其它子系统如果想使用图片可以引用图片的地址，课程管理模块使用图片的方式是将图片地址保存到课程数据库中。

业务流程如下：

1、上传图片到文件系统服务

2、保存图片地址到课程管理服务

在课程管理服务创建保存课程与图片对应关系的表 course_pic。

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

3、在course_pic保存图片成功后方可查询课程图片信息。

通过查询course_pic表数据则查询到某课程的图片信息。

1.2.2 课程管理服务端开发

1.2.2.1 API

课程管理需要使用图片则在课程管理服务中要提供保存课程图片的api。

```
@ApiOperation("添加课程图片")
public ResponseResult addCoursePic(String courseId,String pic);
```

1.2.2.2 Dao

模型：

```
@Data
@ToString
@Entity
@Table(name="course_pic")
@GenericGenerator(name = "jpa-assigned", strategy = "assigned")
public class CoursePic implements Serializable {
    private static final long serialVersionUID = -916357110051689486L;

    @Id
    @GeneratedValue(generator = "jpa-assigned")
    private String courseid;
    private String pic;
}
```

API如下：

```
public interface CoursePicRepository extends JpaRepository<CoursePic, String> {

}
```

1.2.3.3 Service

```
//添加课程图片
    @Transactional
    public ResponseResult saveCoursePic(String courseId,String pic){
        //查询课程图片
        Optional<CoursePic> picOptional = coursePicRepository.findById(courseId);
        CoursePic coursePic = null;
        if(picOptional.isPresent()){
            coursePic = picOptional.get();
        }
        //没有课程图片则新建对象
        if(coursePic == null){
            coursePic = new CoursePic();
        }
        coursePic.setCourseid(courseId);
        coursePic.setPic(pic);

        //保存课程图片
```

```
        coursePicRepository.save(coursePic);
        return new ResponseResult(CommonCode.SUCCESS);
    }
```

1.2.3.4 Controller

```
    @Override
    @PostMapping("/coursepic/add")
    public ResponseResult addCoursePic(@RequestParam("courseId") String courseId,
@RequestParam("pic") String pic) {
        //保存课程图片
        return courseService.saveCoursePic(courseId,pic);
    }
```

1.2.4前端开发

前端需要在上传图片成功后保存课程图片信息。 文件系统不负责保存,保存由子系统负责

1.2.4.1 Api方法

```
//添加课程图片
export const addCoursePic= (courseId,pic) => {
  return http.requestPost(apiUrl+'/course/coursepic/add?courseId='+courseId+"&pic="+pic)
}
```

1.2.4.2 页面

1）添加上传成功的钩子 :on-success="handleSuccess"

```
<el-upload
  action="/api/filesystem/upload"
  list-type="picture-card"
  :on-success="handleSuccess">
  <i class="el-icon-plus"></i>
</el-upload>
```

2）在钩子方法 中保存课程图片信息

如果保存图片失败则上传失败，清除文件列表。 不清空不能再上传,达到了上传的文件数目限制

```
 //上传成功的钩子方法
      handleSuccess(response, file, fileList){
        console.log(response)
        if(response.success){
```

```
        //alert('上传成功')
        //图片上传成功将课程图片地址保存到课程数据库
        let pic = response.fileSystem.filePath
        courseApi.addCoursePic(this.courseid,pic).then((res) => {
          if(res.success){
            this.$message.success('上传成功');
          }else{
            this.handleError()
          }
        });
      }else{
        this.handleError()
      }
    },
    //上传失败执行的钩子方法
    handleError(err, file, fileList){
      this.$message.error('上传失败');
      //清空文件队列
      this.fileList = []
    }
```

4 图片查询

1.3.1 需求分析

课程图片上传成功，再次进入课程上传页面应该显示出来已上传的图片。

1.3.2 API

在课程管理服务定义查询方法

```
@ApiOperation("获取课程基础信息")
public CoursePic findCoursePic(String courseId);
```

1.3.2 课程管理服务开发

1.3.2.1Dao

使用CoursePicRepository即可，无需再开发。

1.3.2.2 Service

根据课程id查询课程图片

```
public CoursePic findCoursepic(String courseId) {
    return coursePicRepository.findOne(courseId);
}
```

1.3.2.3 Controller

```
    @Override
    @GetMapping("/coursepic/list/{courseId}")
    public CoursePic findCoursePic(@PathVariable("courseId") String courseId) {
        return courseService.findCoursepic(courseId);
    }
```

1.3.3 前端开发

1.3.3.1API方法

```
//查询课程图片
export const findCoursePicList = courseId => {
  return http.requestQuickGet(apiUrl+'/course/coursepic/list/'+courseId)
}
```

1.3.3.2页面

在课程图片页面的mounted钩子方法 中查询课程图片信息，并将图片地址赋值给数据对象

1、定义图片查询方法

```
      //查询图片
     list(){
       courseApi.findCoursePicList(this.courseid).then((res) => {
         console.log(res)
         if(res.pic){
           let name = '图片';
           let url = this.imgUrl+res.pic;
           let fileId = res.courseid;
           //先清空文件列表，再将图片放入文件列表
           this.fileList = []
           this.fileList.push({name:name,url:url,fileId:fileId});
         }
         console.log(this.fileList);
       });
```

格式要符合规定，它会自动请求查出来

```
    }
```

2）mounted钩子方法

在mounted钩子方法中调用服务端查询文件列表并绑定到数据对象。

```
mounted(){
   //课程id
   this.courseid = this.$route.params.courseid;
   //查询图片
   this.list()
}
```

1.3.3.3测试

测试流程：

1、上传图片成功

2、进入上传图片页面，观察图片是否显示

5 课程图片删除

1.4.1 需求分析

课程图片上传成功后，可以重新上传，方法是先删除现有图片再上传新图片。

注意：此删除只删除课程数据库的课程图片信息，不去删除文件数据库的文件信息及文件系统服务器上的文件，由于课程图片来源于该用户的文件库，所以此图片可能存在多个地方共用的情况，所以要删除文件系统中的文件需要到图片库由用户确认后再删除。

1.4.2API

在课程管理服务添加删除课程图片api：

```
@ApiOperation("删除课程图片")
public ResponseResult deleteCoursePic(String courseId);
```

1.4.3课程管理服务端开发

1.4.2.1 Dao

CoursePicRepository父类提供的delete方法没有返回值，无法知道是否删除成功，这里我们在CoursePicRepository下边自定义方法：

```
//删除成功返回1否则返回0      这个由返回值,表示影响记录的行数
long deleteByCourseid(String courseid);
```

1.4.2.2 Service

```
    //删除课程图片
    @Transactional
    public ResponseResult deleteCoursePic(String courseId) {
        //执行删除，返回1表示删除成功，返回0表示删除失败
        long result = coursePicRepository.deleteByCourseid(courseId);
        if(result>0){
            return new ResponseResult(CommonCode.SUCCESS);
        }
        return new ResponseResult(CommonCode.FAIL);
    }
```

1.4.2.3 Controller

```
    @Override
    @DeleteMapping("/coursepic/delete")
    public ResponseResult deleteCoursePic(@RequestParam("courseId") String courseId) {
        return courseService.deleteCoursePic(courseId);
    }
```

1.4.3 前端开发

1.4.3.1 API 调用

```
//删除课程图片
export const deleteCoursePic= courseId => {
  return http.requestDelete(apiUrl+'/course/coursepic/delete?courseId='+courseId)
}
```

1.4.3.2 页面测试

1) before-remove钩子方法

在upload组件的before-remove钩子方法 中实现删除动作。

```
<el-upload
  action="/filesystem/upload"
  list-type="picture-card"
  :before-remove="handleRemove">
  <i class="el-icon-plus"></i>
</el-upload>
```

before-remove说明：删除文件之前的钩子，参数为上传的文件和文件列表，若返回 false 或者返回 Promise 且被 reject，则停止删除。

返回其他或者不返回都会删除

定义handleRemove方法进行测试：

handleRemove 返回true则删除页面的图片，返回false则停止删除页面的图片。

```
//删除图片
handleRemove(file, fileList) {
        console.log(file)
        alert('删除成功')
        return true;
}
```

1.4.3.3 promise异步调用

嵌套promise的目的就是为了必须要返回值才可以进行下一步

在handleRemove方法调用删除图片的api方法，删除成功时return true，删除失败时return false;

这里是没有嵌套一层promise

```
      //删除图片
      handleRemove(file, fileList) {
        console.log(file)
//        alert('删除')
//        return true; 这里是同步,会等待返回值,这里现在是true
        //删除图片
          courseApi.deleteCoursePic('1').then((res) => {
            if(res.success){
              this.$message.success('删除成功');
              return true;
            }else{
              this.$message.error(res.message);
              return false;
            }
          });
      },
```

返回false却删除的原因在于,这里的底层是ajax,属于异步,不会等待返回值,相当于没有返回值

在上边代码中将提交的课程id故意写错，按照我们预期应该是删除失败，而测试结果却是图片在页面上删除成功。

问题原因：

通过查询deleteCoursePic方法的底层代码，deleteCoursePic最终返回一个promise对象。

但是需要把这个promise对象中的信息拿出来,所以需要在操作后再在外层包装一个promise对象

Promise是ES6提供的用于异步处理的对象，因为axios提交是异步提交，这里使用promise作为返回值。

Promise的使用方法如下：

Promise对象在处理过程中有三种状态：

pending：进行中

resolved：操作成功

rejected: 操作失败

Promise的构建方法如下：

```
const promise = new Promise(function(resolve,reject){
     //...TODO...
    if(操作成功){
        resolve(value);
    }else{
        reject(error);
    }
})
```

function方法有简写,看下面例子

里面是一个方法,不过需要把参数,resolve,reject传进去

上边的构造方法function(resolve,reject)执行流程如下：

1）方法执行一些业务逻辑。

也就是pending不操作

2）如果操作成功将Promise的状态由pending变为resolved，并将操作结果传出去

3）如果操作失败会将promise的状态由pending变为rejected，并将失败结果传出去。

上边说的操作成功将操作结果传给谁了呢？操作失败将失败结果传给谁了呢？

通过promise的then、catch来指定

```
promise.then(function (result) {
    console.log('操作成功：' + result);
});
promise.catch(function (reason) {
    console.log('操作失败：' + reason);
});
```

例子如下：

1、定义一个方法，返回promise对象

```
testpromise(i){
  return new Promise((resolve,reject)=>{
    if(i % 2==0){
      resolve('成功了')
    }else{
      reject('拒绝了')
    }
  })
}
```

立即调用then后面的方法

2、调用此方法

向方法传入偶数、奇数进行测试。

```
this.testpromise(3).then(res=>{//在then中对成功结果进行处理
    alert(res)
}).catch(res=>{//在catch中对操作失败结果进行处理
    alert(res)
})
```

then后面也是一个回调方法,操作成功后调用
res是回调的结果集参数

前面的很多方法都没有catch是因为服务端已经进行异常处理,因此都有返回值,有返回值就没有操作失败

3、最终将handleRemove方法修改如下

handleRemove方法返回promise对象，当删除成功则resolve，删除失败则reject。

```
    //删除图片
      handleRemove(file, fileList) {
        console.log(file)
        return new Promise((resolve,reject)=>{
        //删除图片
          courseApi.deleteCoursePic(this.courseid).then((res) => {
            if(res.success){
              this.$message.success('删除成功');
              resolve()//通过
            }else{
              this.$message.error(res.message);
              reject()//拒绝
            }
          });
        })

      }
```