------课程列表

# 01-课程列表的显示

## 一、后端实现

### 1、定义搜索对象

CourseQuery

@ApiModel*(*value = "Course查询对象", description = "课程查询对象封装"*)*

@Data  
public class CourseQuery implements Serializable *{* private static final long *serialVersionUID* = 1L;  
 @ApiModelProperty*(*value = "课程名称"*)* private String title;  
 @ApiModelProperty*(*value = "讲师id"*)* private String teacherId;  
 @ApiModelProperty*(*value = "一级类别id"*)* private String subjectParentId;  
 @ApiModelProperty*(*value = "二级类别id"*)* private String subjectId;  
*}*

### 2、定义service方法

接口

void pageQuery(Page<Course> pageParam, CourseQuery courseQuery);

实现

@Override

public void pageQuery*(*Page*<*Course*>* pageParam, CourseQuery courseQuery*) {*

QueryWrapper*<*Course*>* queryWrapper = new QueryWrapper*<*>*()*;  
 queryWrapper.orderByDesc*(*"gmt\_create"*)*;  
 if *(*courseQuery == null*) {* baseMapper.selectPage*(*pageParam, queryWrapper);  
 return;  
 *}* String title = courseQuery.getTitle*()*;  
 String teacherId = courseQuery.getTeacherId*()*;  
 String subjectParentId = courseQuery.getSubjectParentId*()*;  
 String subjectId = courseQuery.getSubjectId*()*;  
 if *(*!StringUtils.*isEmpty(*title*)) {* queryWrapper.like*(*"title", title*)*;  
 *}* if *(*!StringUtils.*isEmpty(*teacherId*)) {* queryWrapper.eq*(*"teacher\_id", teacherId*)*;  
 *}* if *(*!StringUtils.*isEmpty(*subjectParentId*)) {* queryWrapper.ge*(*"subject\_parent\_id", subjectParentId*)*;  
 *}* if *(*!StringUtils.*isEmpty(*subjectId*)) {* queryWrapper.ge*(*"subject\_id", subjectId*)*;  
 *}* baseMapper.selectPage*(*pageParam, queryWrapper);  
*}*

### 3、定义web层方法

@ApiOperation*(*value = "分页课程列表"*)*

@GetMapping*(*"{page}/{limit}"*)*public R pageQuery*(* @ApiParam*(*name = "page", value = "当前页码", required = true*)* @PathVariable Long page,  
 @ApiParam*(*name = "limit", value = "每页记录数", required = true*)* @PathVariable Long limit,  
 @ApiParam*(*name = "courseQuery", value = "查询对象", required = false*)* CourseQuery courseQuery*) {* Page*<*Course*>* pageParam = new Page*<*>*(*page, limit*)*;  
 courseService.pageQuery*(*pageParam, courseQuery*)*;  
 List*<*Course*>* records = pageParam.getRecords*()*;  
 long total = pageParam.getTotal*()*;  
 return R.*ok()*.data*(*"total", total*)*.data*(*"rows", records*)*;  
*}*

## 二、前端分页查询列表

### 1、定义api

course.js

getPageList(page, limit, searchObj) {

return request({

url: `${api\_name}/${page}/${limit}`,

method: 'get',

params: searchObj

})

},

### 2、组件中的js

src/views/edu/list.vue

<script>

    import course from '@/api/edu/course'

    import teacher from '@/api/edu/teacher'

    import subject from '@/api/edu/subject'

    export default {

      data() {

        return {

          listLoading: true, // 是否显示loading信息

          list: null, // 数据列表

          total: 0, // 总记录数

          page: 1, // 页码

          limit: 10, // 每页记录数

          searchObj: {

            subjectParentId: '',

            subjectId: '',

            title: '',

            teacherId: ''

          }, // 查询条件

          teacherList: [], // 讲师列表

          subjectNestedList: [], // 一级分类列表

          subSubjectList: [] // 二级分类列表,

        }

      },

      created() { // 当页面加载时获取数据

        this.fetchData()

        // 初始化分类列表

        this.initSubjectList()

        // 获取讲师列表

        this.initTeacherList()

      },

      methods: {

        fetchData(*page* = 1) { // 调用api层获取数据库中的数据

          console.log('加载列表')

          // 当点击分页组件的切换按钮的时候，会传输一个当前页码的参数page

          // 解决分页无效问题

          this.page = *page*

          this.listLoading = true

          course.getPageList(this.page, this.limit, this.searchObj).then(*response* *=>* {

            // debugger 设置断点调试

            if (*response*.success === true) {

              this.list = *response*.data.rows

              this.total = *response*.data.total

            }

            this.listLoading = false

          })

        },

        initTeacherList() {

          teacher.getList().then(*response* *=>* {

            this.teacherList = *response*.data.items

          })

        },

        initSubjectList() {

          subject.getNestedTreeList().then(*response* *=>* {

            this.subjectNestedList = *response*.data.items

          })

        },

        subjectLevelOneChanged(*value*) {

          for (*let* i = 0; i < this.subjectNestedList.length; i++) {

            if (this.subjectNestedList[i].id === *value*) {

              this.subSubjectList = this.subjectNestedList[i].children

              this.searchObj.subjectId = ''

            }

          }

        },

        resetData() {

          this.searchObj = {}

          this.subSubjectList = [] // 二级分类列表

          this.fetchData()

        }

      }

    }

</script>

### 3、组件模板

查询表单

<!--查询表单-->

<el-form :inline="true" class="demo-form-inline">

    <!-- 所属分类：级联下拉列表 -->

    <!-- 一级分类 -->

    <el-form-item label="课程类别">

      <el-select

        v-model="searchObj.subjectParentId"

        placeholder="请选择"

        @change="subjectLevelOneChanged">

        <el-option

          v-for="subject in subjectNestedList"

          :key="subject.id"

          :label="subject.title"

          :value="subject.id"/>

      </el-select>

      <!-- 二级分类 -->

      <el-select v-model="searchObj.subjectId" placeholder="请选择">

        <el-option

          v-for="subject in subSubjectList"

          :key="subject.id"

          :label="subject.title"

          :value="subject.id"/>

      </el-select>

    </el-form-item>

    <!-- 标题 -->

    <el-form-item>

      <el-input v-model="searchObj.title" placeholder="课程标题"/>

    </el-form-item>

    <!-- 讲师 -->

    <el-form-item>

      <el-select

        v-model="searchObj.teacherId"

        placeholder="请选择讲师">

        <el-option

          v-for="teacher in teacherList"

          :key="teacher.id"

          :label="teacher.name"

          :value="teacher.id"/>

      </el-select>

    </el-form-item>

    <el-button type="primary" icon="el-icon-search" @click="fetchData()">查询</el-button>

    <el-button type="default" @click="resetData()">清空</el-button>

  </el-form>

表格和分页

表格添加了 row-class-name="myClassList" 样式定义

     <!-- 表格 -->

    <el-table v-loading="listLoading" :data="list" element-loading-text="数据加载中" border fit highlight-current-row

      row-class-name="myClassList">

      <el-table-column label="序号" width="70" align="center">

        <template slot-scope="scope">

          {{ (page - 1) \* limit + scope.$index + 1 }}

        </template>

      </el-table-column>

      <el-table-column label="课程信息" width="470" align="center">

        <template slot-scope="scope">

          <div class="info">

            <div class="pic">

              <img :src="scope.row.cover" alt="scope.row.title" width="150px">

            </div>

            <div class="title">

              <a href="">{{ scope.row.title }}</a>

              <p>{{ scope.row.lessonNum }}课时</p>

            </div>

          </div>

        </template>

      </el-table-column>

      <el-table-column label="创建时间" align="center">

        <template slot-scope="scope">

          {{ scope.row.gmtCreate.substr(0, 10) }}

        </template>

      </el-table-column>

      <el-table-column label="发布时间" align="center">

        <template slot-scope="scope">

          {{ scope.row.gmtModified.substr(0, 10) }}

        </template>

      </el-table-column>

      <el-table-column label="价格" width="100" align="center">

        <template slot-scope="scope">

          {{ Number(scope.row.price) === 0 ? '免费' :

      '¥' + scope.row.price.toFixed(2) }}

        </template>

      </el-table-column>

      <el-table-column prop="buyCount" label="付费学员" width="100" align="center">

        <template slot-scope="scope">

          {{ scope.row.buyCount }}人

        </template>

      </el-table-column>

      <el-table-column label="操作" width="150" align="center">

        <template slot-scope="scope">

          <router-link :to="'/edu/course/info/'+scope.row.id">

            <el-button type="text" size="mini" icon="el-icon-edit">编辑课程信息</el-button>

          </router-link>

          <router-link :to="'/edu/course/chapter/'+scope.row.id">

            <el-button type="text" size="mini" icon="el-icon-edit">编辑课程大纲</el-button>

          </router-link>

          <el-button type="text" size="mini" icon="el-icon-delete">删除</el-button>

        </template>

      </el-table-column>

    </el-table>

    <!-- 分页 -->

    <el-pagination :current-page="page" :page-size="limit" :total="total" style="padding: 30px 0; text-align: center;"

      layout="total, prev, pager, next, jumper" @current-change="fetchData" />

### 4、css的定义

<style scoped>

  .myClassList .info {

*width*: 450px;

*overflow*: hidden;

  }

  .myClassList .info .pic {

*width*: 150px;

*height*: 90px;

*overflow*: hidden;

*float*: left;

  }

  .myClassList .info .pic a {

*display*: block;

*width*: 100%;

*height*: 100%;

*margin*: 0;

*padding*: 0;

  }

  .myClassList .info .pic img {

*display*: block;

*width*: 100%;

  }

  .myClassList td .info .title {

*width*: 280px;

*float*: right;

*height*: 90px;

  }

  .myClassList td .info .title a {

*display*: block;

*height*: 48px;

*line-height*: 24px;

*overflow*: hidden;

*color*: #00baf2;

*margin-bottom*: 12px;

  }

  .myClassList td .info .title p {

*line-height*: 20px;

*margin-top*: 5px;

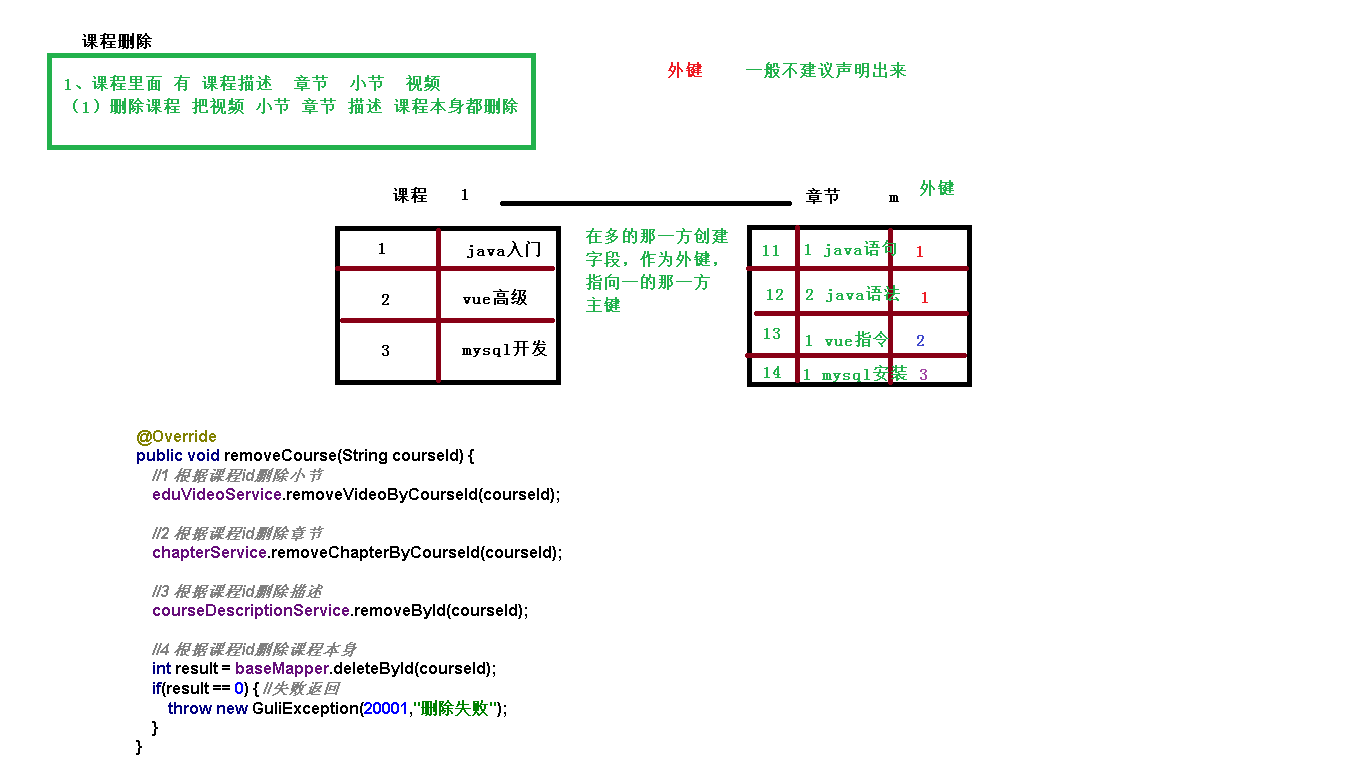
*color*: #818181;

  }

</style>

# 02-删除课程

## 后端实现



### 1、web层

定义删除api方法：CourseAdminController.java

@ApiOperation(value = "根据ID删除课程")

@DeleteMapping("{id}")

public R removeById(

@ApiParam(name = "id", value = "课程ID", required = true)

@PathVariable String id){

boolean result = courseService.removeCourseById(id);

if(result){

return R.ok();

}else{

return R.error().message("删除失败");

}

}

### 2、service层

如果用户确定删除，则首先删除video记录，然后删除chapter记录，最后删除Course记录

**2.1、在VideoService中定义根据courseId删除video业务方法**

接口

boolean removeByCourseId(String courseId);

实现

@Override

public boolean removeByCourseId(String courseId) {

QueryWrapper<Video> queryWrapper = new QueryWrapper<>();

queryWrapper.eq("course\_id", courseId);

Integer count = baseMapper.delete(queryWrapper);

return null != count && count > 0;

}

**2.2、在ChapterService中定义根据courseId删除chapter业务方法**

接口

boolean removeByCourseId(String courseId);

实现

@Override

public boolean removeByCourseId(String courseId) {

QueryWrapper<Chapter> queryWrapper = new QueryWrapper<>();

queryWrapper.eq("course\_id", courseId);

Integer count = baseMapper.delete(queryWrapper);

return null != count && count > 0;

}

**2.3、删除当前course记录**

接口：CourseService.java

boolean removeCourseById(String id);

实现：CourseServiceImpl.java

@Override

public boolean removeCourseById(String id) {

//根据id删除所有视频

videoService.removeByCourseId(id);

//根据id删除所有章节

chapterService.removeByCourseId(id);

Integer result = baseMapper.deleteById(id);

return null != result && result > 0;

}

## 二、前端实现

### 1、定义api

course.js中添加删除方法

removeById(id) {

return request({

url: `${api\_name}/${id}`,

method: 'delete'

})

}

### 2、修改删除按钮

src/api/edu/course.js 删除按钮注册click事件

<el-button type="text" size="mini" icon="el-icon-delete" @click="removeDataById(scope.row.id)">删除</el-button>

### 3、编写删除方法

removeDataById(id) {

// debugger

this.$confirm('此操作将永久删除该课程，以及该课程下的章节和视频，是否继续?', '提示', {

confirmButtonText: '确定',

cancelButtonText: '取消',

type: 'warning'

}).then(() => {

return course.removeById(id)

}).then(() => {

this.fetchData()

this.$message({

type: 'success',

message: '删除成功!'

})

}).catch((response) => { // 失败

if (response === 'cancel') {

this.$message({

type: 'info',

message: '已取消删除'

})

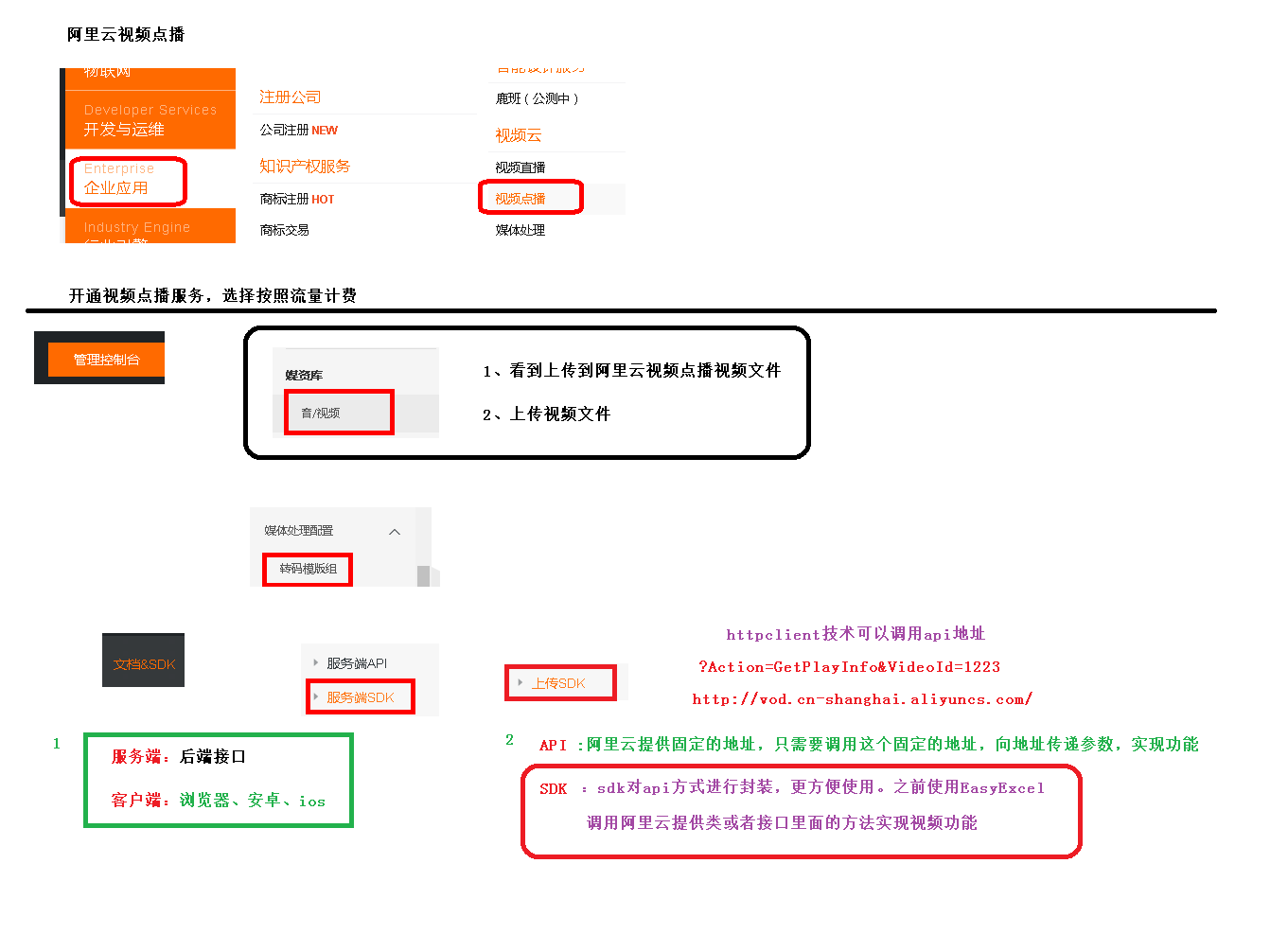
}

})

}

------视频点播

# 视频点播简介



## 一、阿里云视频点播技术能力盘点

**参考文章：**

<https://blog.csdn.net/qq_33857573/article/details/79564255>

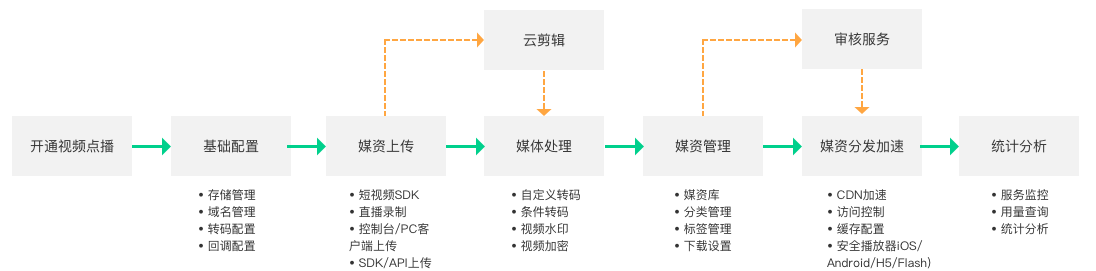
视频点播（ApsaraVideo for VoD）是集音视频采集、编辑、上传、自动化转码处理、媒体资源管理、分发加速于一体的一站式音视频点播解决方案。



### 1、应用场景

* 音视频网站：无论是初创视频服务企业，还是已拥有海量视频资源，可定制化的点播服务帮助客户快速搭建拥有极致观看体验、安全可靠的视频点播应用。
* 短视频：集音视频拍摄、特效编辑、本地转码、高速上传、自动化云端转码、媒体资源管理、分发加速、播放于一体的完整短视频解决方案。目前已帮助1000+APP快速实现手机短视频功能。
* 直播转点播：将直播流同步录制为点播视频，用于回看。并支持媒资管理、媒体处理（转码及内容审核/智能首图等AI处理）、内容制作（云剪辑）、CDN分发加速等一系列操作。
* 在线教育：为在线教育客户提供简单易用、安全可靠的视频点播服务。可通过控制台/API等多种方式上传教学视频，强大的转码能力保证视频可以快速发布，覆盖全网的加速节点保证学生观看的流畅度。防盗链、视频加密等版权保护方案保护教学内容不被窃取。
* 视频生产制作：提供在线可视化剪辑平台及丰富的OpenAPI，帮助客户高效处理、制作视频内容。除基础的剪切拼接、混音、遮标、特效、合成等一系列功能外，依托云剪辑及点播一体化服务还可实现标准化、智能化剪辑生产，大大降低视频制作的槛，缩短制作时间，提升内容生产效率。
* 内容审核：应用于短视频平台、传媒行业审核等场景，帮助客户从从语音、文字、视觉等多维度精准识别视频、封面、标题或评论的违禁内容进行AI智能审核与人工审核。

### 2、功能介绍

[](http://docs-aliyun.cn-hangzhou.oss.aliyun-inc.com/assets/pic/85506/cn_zh/1542030629145/%E4%BA%A7%E5%93%81%E5%8A%9F%E8%83%BD.png)

## 二、开通视频点播云平台

### 1、选择视频点播服务

产品->企业应用->视频云->视频点播

### 2、开通视频点播



### 3、选择按使用流量计费



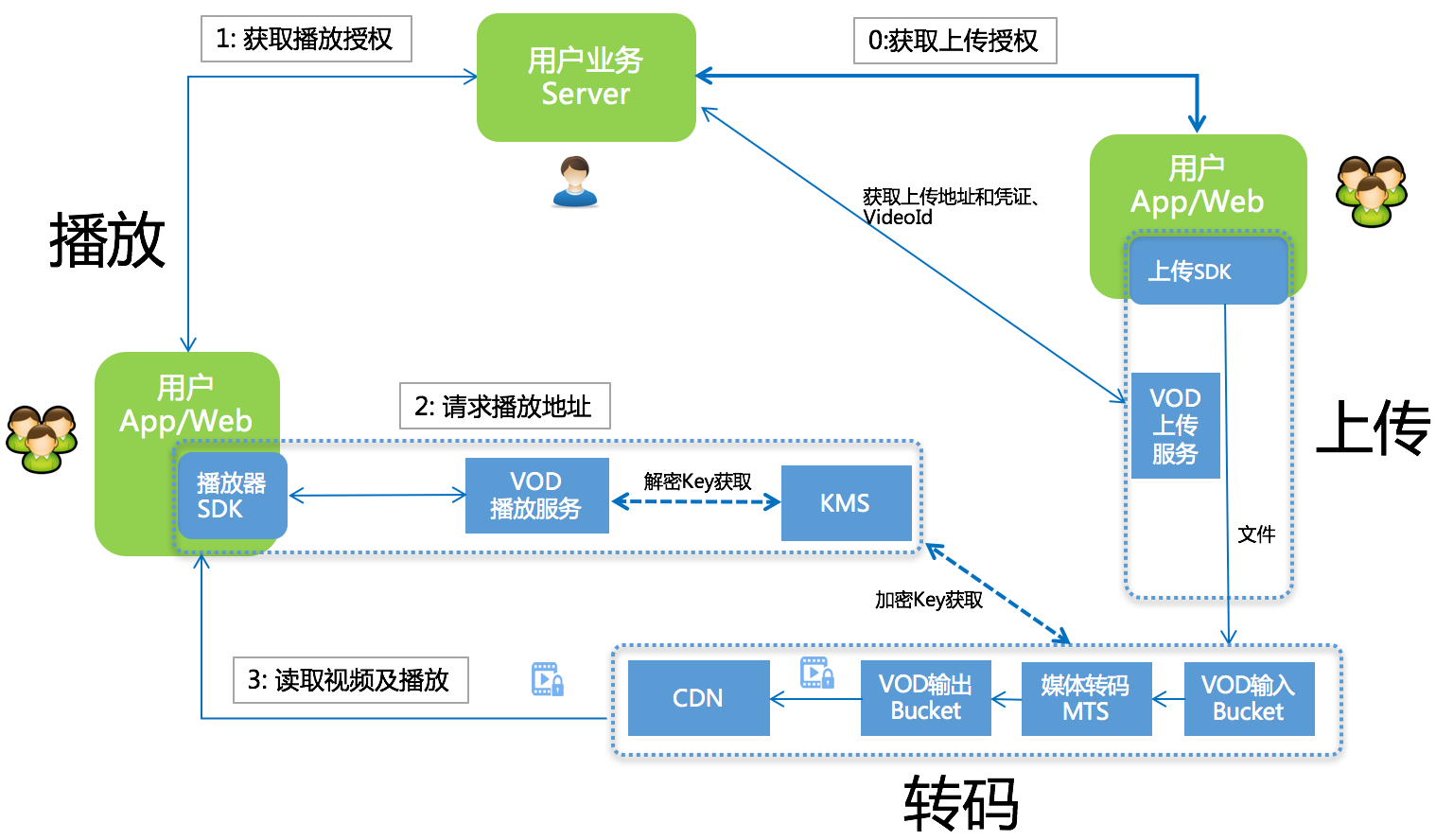
### 4、资费说明

<https://www.aliyun.com/price/product?spm=a2c4g.11186623.2.12.7fbd59b9vmXVN6#/vod/detail>

* 后付费
* 套餐包
* 欠费说明
* 计费案例：<https://help.aliyun.com/document_detail/64032.html?spm=a2c4g.11186623.4.3.363db1bcfdvxB5>

### 5、整体流程

使用视频点播实现音视频上传、存储、处理和播放的整体流程如下：



* 用户获取上传授权。
* VoD下发 上传地址和凭证 及 VideoId。
* 用户上传视频保存视频ID(VideoId)。
* 用户服务端获取播放凭证。
* VoD下发带时效的播放凭证。
* 用户服务端将播放凭证下发给客户端完成视频播放。

## 三、视频点播服务的基本使用

完整的参考文档

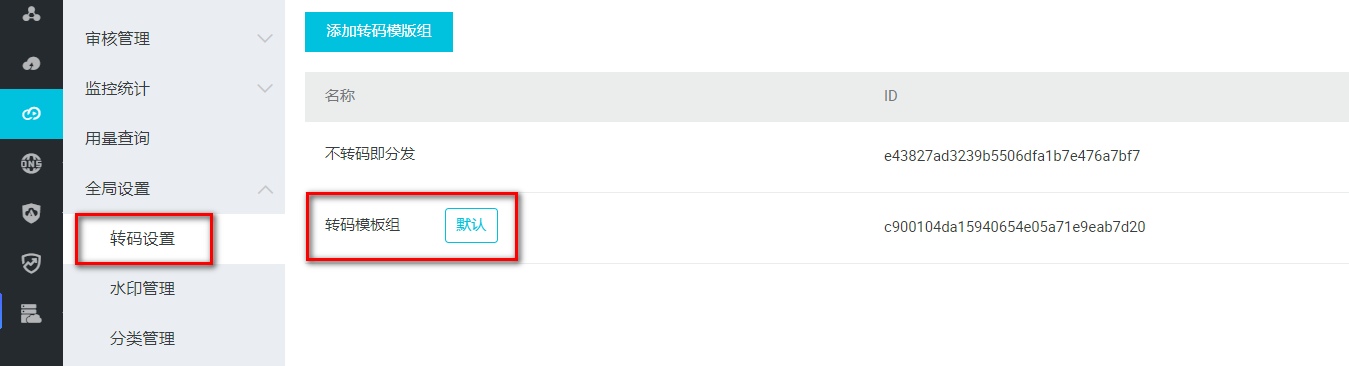
<https://help.aliyun.com/product/29932.html?spm=a2c4g.11186623.6.540.3c356a58OEmVZJ>

### 1、设置转码格式

选择全局设置 > 转码设置，单击添加转码模板组。

在视频转码模板组页面，根据业务需求选择封装格式和清晰度。

或直接将已有的模板设置为默认即可



### 2、分类管理

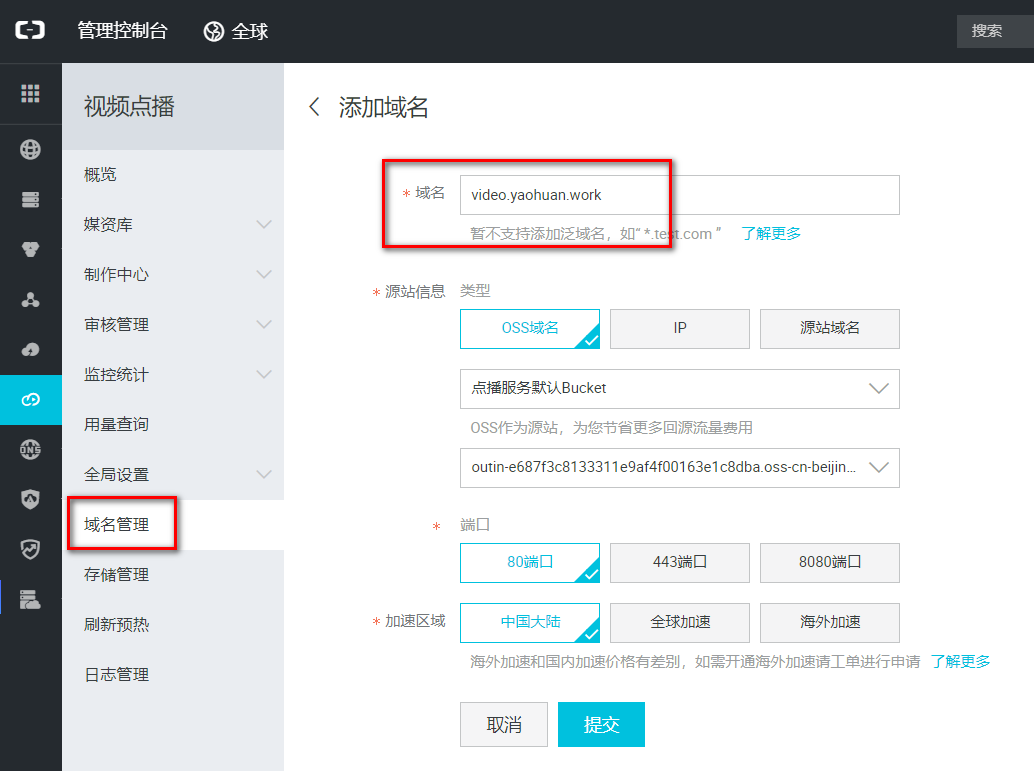
选择全局设置 > 分类管理

### 3、上传视频文件

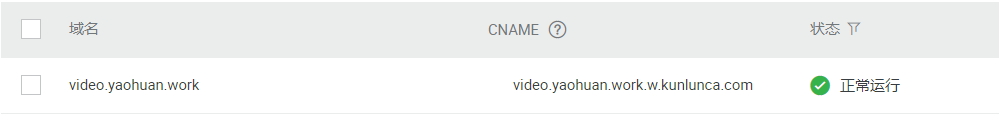
选择媒资库 > 音视频，单击上传音视频

### 4、配置域名

音视频上传完成后，必须配一个已备案的域名，并完成CNAME绑定



得到CNAME



在购买域名的服务商处的管理控制台配置域名解析

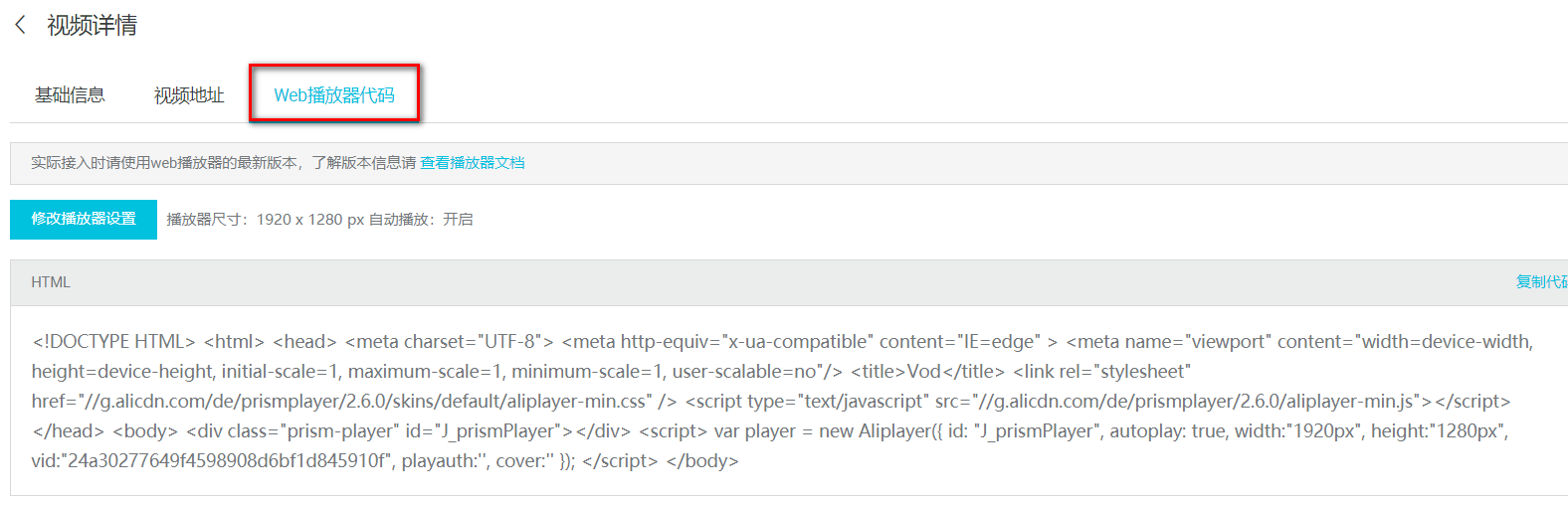


**5、在控制台查看视频**

此时视频可以在阿里云控制台播放

### 6、获取web播放器代码





# 02-使用服务端SDK

## 一、服务端SDK

### 1、简介

sdk的方式将api进行了进一步的封装，不用自己创建工具类。

我们可以基于服务端SDK编写代码来调用点播API，实现对点播产品和服务的快速操作。

### 2、功能介绍

* SDK封装了对API的调用请求和响应，避免自行计算较为繁琐的 API签名。
* 支持所有点播服务的API，并提供了相应的示例代码。
* 支持7种开发语言，包括：Java、Python、PHP、.NET、Node.js、Go、C/C++。
* 通常在发布新的API后，我们会及时同步更新SDK，所以即便您没有找到对应API的示例代码，也可以参考旧的示例自行实现调用。

## 使用SDK



### 1、安装

参考文档：<https://help.aliyun.com/document_detail/57756.html>

添加maven仓库的配置和依赖到pom

*<*repositories*>*

*<*repository*>  
 <*id*>*sonatype-nexus-staging*</*id*>  
 <*name*>*Sonatype Nexus Staging*</*name*>  
 <*url*>*https://oss.sonatype.org/service/local/staging/deploy/maven2/*</*url*>  
 <*releases*>  
 <*enabled*>*true*</*enabled*>  
 </*releases*>  
 <*snapshots*>  
 <*enabled*>*true*</*enabled*>  
 </*snapshots*>  
 </*repository*>  
</*repositories*>*

*<*dependency*>*

*<*groupId*>*com.aliyun*</*groupId*>  
 <*artifactId*>*aliyun-java-sdk-core*</*artifactId*>  
 <*version*>*4.3.3*</*version*>  
</*dependency*>  
<*dependency*>  
 <*groupId*>*com.aliyun*</*groupId*>  
 <*artifactId*>*aliyun-java-sdk-vod*</*artifactId*>  
 <*version*>*2.15.5*</*version*>  
</*dependency*>  
<*dependency*>  
 <*groupId*>*com.google.code.gson*</*groupId*>  
 <*artifactId*>*gson*</*artifactId*>  
 <*version*>*2.8.2*</*version*>  
</*dependency*>*

### 2、初始化

参考文档：<https://help.aliyun.com/document_detail/61062.html>

根据文档示例创建 AliyunVODSDKUtils.java

package com.atguigu.aliyunvod.util;

public class AliyunVodSDKUtils *{* public static DefaultAcsClient initVodClient*(*String accessKeyId, String accessKeySecret*)* throws ClientException *{* String regionId = "cn-shanghai"; // 点播服务接入区域  
 DefaultProfile profile = DefaultProfile.getProfile*(*regionId, accessKeyId, accessKeySecret*)*;  
 DefaultAcsClient client = new DefaultAcsClient*(*profile*)*;  
 return client;  
 *}  
}*

### 3、创建测试类

创建 VodSdkTest.java

package com.atguigu.aliyunvod;

public class VodSdkTest {

String accessKeyId = "你的accessKeyId";

String accessKeySecret = "你的accessKeySecret";

}

## 三、创建测试用例

参考文档：<https://help.aliyun.com/document_detail/61064.html>

### 1、获取视频播放凭证

根据文档中的代码，修改如下

*/\*\**

*\* 获取视频播放凭证  
 \*  
 \** ***@throws*** *ClientException  
 \*/*@Test  
public void testGetVideoPlayAuth*()* throws ClientException *{* //初始化客户端、请求对象和相应对象  
 DefaultAcsClient client = AliyunVodSDKUtils.*initVodClient(*accessKeyId, accessKeySecret*)*;  
 GetVideoPlayAuthRequest request = new GetVideoPlayAuthRequest*()*;  
 GetVideoPlayAuthResponse response = new GetVideoPlayAuthResponse*()*;  
 try *{* //设置请求参数  
 request.setVideoId*(*"视频ID"*)*;  
 //获取请求响应  
 response = client.getAcsResponse*(*request*)*;  
 //输出请求结果  
 //播放凭证  
 System.*out*.print*(*"PlayAuth = " + response.getPlayAuth*()* + "\n"*)*;  
 //VideoMeta信息  
 System.*out*.print*(*"VideoMeta.Title = " + response.getVideoMeta*()*.getTitle*()* + "\n"*)*;  
 *}* catch *(*Exception e*) {* System.*out*.print*(*"ErrorMessage = " + e.getLocalizedMessage*())*;  
 *}* System.*out*.print*(*"RequestId = " + response.getRequestId*()* + "\n"*)*;  
*}*

### 2、获取视频播放地址

*/\*\**

*\* 获取视频播放地址  
 \*  
 \** ***@throws*** *ClientException  
 \*/*@Test  
public void testGetPlayInfo*()* throws ClientException *{* //初始化客户端、请求对象和相应对象  
 DefaultAcsClient client = AliyunVodSDKUtils.*initVodClient(*accessKeyId, accessKeySecret*)*;  
 GetPlayInfoRequest request = new GetPlayInfoRequest*()*;  
 GetPlayInfoResponse response = new GetPlayInfoResponse*()*;  
 try *{* //设置请求参数  
 //注意：这里只能获取非加密视频的播放地址  
 request.setVideoId*(*"视频ID"*)*;  
 //获取请求响应  
 response = client.getAcsResponse*(*request*)*;  
 //输出请求结果  
 List*<*GetPlayInfoResponse.PlayInfo*>* playInfoList = response.getPlayInfoList*()*;  
 //播放地址  
 for *(*GetPlayInfoResponse.PlayInfo playInfo : playInfoList*) {* System.*out*.print*(*"PlayInfo.PlayURL = " + playInfo.getPlayURL*()* + "\n"*)*;  
 *}* //Base信息  
 System.*out*.print*(*"VideoBase.Title = " + response.getVideoBase*()*.getTitle*()* + "\n"*)*;  
 *}* catch *(*Exception e*) {* System.*out*.print*(*"ErrorMessage = " + e.getLocalizedMessage*())*;  
 *}* System.*out*.print*(*"RequestId = " + response.getRequestId*()* + "\n"*)*;  
*}*

# 03-文件上传测试



参考文档：<https://help.aliyun.com/document_detail/53406.html>

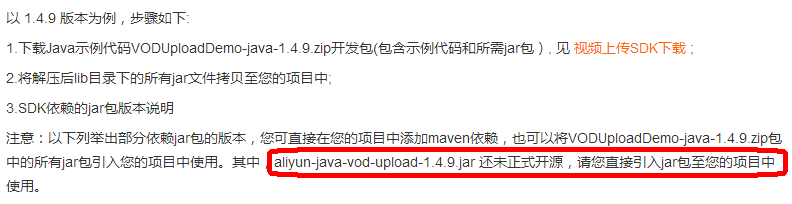
## 一、安装SDK

### 1、配置pom

*<*dependency*>*

*<*groupId*>*com.aliyun*</*groupId*>  
 <*artifactId*>*aliyun-java-sdk-core*</*artifactId*>  
 <*version*>*4.3.3*</*version*>  
</*dependency*>  
<*dependency*>  
 <*groupId*>*com.aliyun.oss*</*groupId*>  
 <*artifactId*>*aliyun-sdk-oss*</*artifactId*>  
 <*version*>*3.1.0*</*version*>  
</*dependency*>  
<*dependency*>  
 <*groupId*>*com.aliyun*</*groupId*>  
 <*artifactId*>*aliyun-java-sdk-vod*</*artifactId*>  
 <*version*>*2.15.2*</*version*>  
</*dependency*>  
<*dependency*>  
 <*groupId*>*com.alibaba*</*groupId*>  
 <*artifactId*>*fastjson*</*artifactId*>  
 <*version*>*1.2.28*</*version*>  
</*dependency*>  
<*dependency*>  
 <*groupId*>*org.json*</*groupId*>  
 <*artifactId*>*json*</*artifactId*>  
 <*version*>*20170516*</*version*>  
</*dependency*>  
<*dependency*>  
 <*groupId*>*com.google.code.gson*</*groupId*>  
 <*artifactId*>*gson*</*artifactId*>  
 <*version*>*2.8.2*</*version*>  
</*dependency*>*

### 2、安装非开源jar包





在本地Maven仓库中安装jar包：

下载视频上传SDK，解压，命令行进入lib目录，执行以下代码

mvn install:install-file -DgroupId=com.aliyun -DartifactId=aliyun-sdk-vod-upload -Dversion=1.4.11 -Dpackaging=jar -Dfile=aliyun-java-vod-upload-1.4.11.jar

然后在pom中引入jar包

*<*dependency*>*

*<*groupId*>*com.aliyun*</*groupId*>  
 <*artifactId*>*aliyun-sdk-vod-upload*</*artifactId*>  
 <*version*>*1.4.11*</*version*>  
</*dependency*>*

## 二、测试

### 1、创建测试文件

package com.atguigu.aliyunvod;

public class UploadTest {

//账号AK信息请填写(必选)

private static final String accessKeyId = "你的accessKeyId";

//账号AK信息请填写(必选)

private static final String accessKeySecret = "你的accessKeySecret";

}

### 2、测试本地文件上传

*/\*\**

*\* 视频上传  
 \*/*@Test  
public void testUploadVideo*() {* //1.音视频上传-本地文件上传  
 //视频标题(必选)  
 String title = "3 - How Does Project Submission Work - upload by sdk";  
 //本地文件上传和文件流上传时，文件名称为上传文件绝对路径，如:/User/sample/文件名称.mp4 (必选)  
 //文件名必须包含扩展名  
 String fileName = "E:/共享/资源/课程视频/3 - How Does Project Submission Work.mp4";  
 //本地文件上传  
 UploadVideoRequest request = new UploadVideoRequest*(*accessKeyId, accessKeySecret, title, fileName*)*;  
 /\* 可指定分片上传时每个分片的大小，默认为1M字节 \*/  
 request.setPartSize*(*1 \* 1024 \* 1024L*)*;  
 /\* 可指定分片上传时的并发线程数，默认为1，(注：该配置会占用服务器CPU资源，需根据服务器情况指定）\*/  
 request.setTaskNum*(*1*)*;  
/\* 是否开启断点续传, 默认断点续传功能关闭。当网络不稳定或者程序崩溃时，再次发起相同上传请求，可以继续未完成的上传任务，适用于超时3000秒仍不能上传完成的大文件。  
 注意: 断点续传开启后，会在上传过程中将上传位置写入本地磁盘文件，影响文件上传速度，请您根据实际情况选择是否开启\*/  
 request.setEnableCheckpoint*(*false*)*;  
 UploadVideoImpl uploader = new UploadVideoImpl*()*;  
 UploadVideoResponse response = uploader.uploadVideo*(*request*)*;  
 System.*out*.print*(*"RequestId=" + response.getRequestId*()* + "\n"*)*; //请求视频点播服务的请求ID  
 if *(*response.isSuccess*()) {* System.*out*.print*(*"VideoId=" + response.getVideoId*()* + "\n"*)*;  
 *}* else *{* /\* 如果设置回调URL无效，不影响视频上传，可以返回VideoId同时会返回错误码。其他情况上传失败时，VideoId为空，此时需要根据返回错误码分析具体错误原因 \*/  
 System.*out*.print*(*"VideoId=" + response.getVideoId*()* + "\n"*)*;  
 System.*out*.print*(*"ErrorCode=" + response.getCode*()* + "\n"*)*;  
 System.*out*.print*(*"ErrorMessage=" + response.getMessage*()* + "\n"*)*;  
 *}  
}*

整合阿里云视频点播-------

# 视频点播微服务的创建



## 一、创建视频点播微服务

### 1、创建微服务模块

Artifact：service-vod

**2、pom**

**（1）service-vod中引入依赖**

*<*dependencies*>*

*<*dependency*>  
 <*groupId*>*com.aliyun*</*groupId*>  
 <*artifactId*>*aliyun-java-sdk-core*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.aliyun.oss*</*groupId*>  
 <*artifactId*>*aliyun-sdk-oss*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.aliyun*</*groupId*>  
 <*artifactId*>*aliyun-java-sdk-vod*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.aliyun*</*groupId*>  
 <*artifactId*>*aliyun-sdk-vod-upload*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.alibaba*</*groupId*>  
 <*artifactId*>*fastjson*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*org.json*</*groupId*>  
 <*artifactId*>*json*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*com.google.code.gson*</*groupId*>  
 <*artifactId*>*gson*</*artifactId*>  
 </*dependency*>  
 <*dependency*>  
 <*groupId*>*joda-time*</*groupId*>  
 <*artifactId*>*joda-time*</*artifactId*>  
 </*dependency*>  
</*dependencies*>*

### 3、application.properties

# 服务端口

server.port=8003  
# 服务名  
spring.application.name=service-vod  
# 环境设置：dev、test、prod  
spring.profiles.active=dev  
#阿里云 vod  
#不同的服务器，地址不同  
aliyun.vod.file.keyid=your accessKeyId  
aliyun.vod.file.keysecret=your accessKeySecret  
# 最大上传单个文件大小：默认1M  
spring.servlet.multipart.max-file-size=1024MB  
# 最大置总上传的数据大小 ：默认10M  
spring.servlet.multipart.max-request-size=1024MB

### 4、logback.xml

### 5、启动类

VodApplication.java

package com.guli.vod

import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;  
import org.springframework.context.annotation.ComponentScan;  
  
@SpringBootApplication*(*exclude = DataSourceAutoConfiguration.class*)*@ComponentScan*(*basePackages = *{*"com.atguigu"*})*public class VodApplication *{* public static void main*(*String*[]* args*) {* SpringApplication.*run(*VodApplication.class, args*)*;  
 *}  
}*

## 二、整合阿里云vod实现视频上传

### 1、创建常量类

ConstantPropertiesUtil.java

package com.guli.vod.util;

@Component

//@PropertySource("classpath:application.properties")  
public class ConstantPropertiesUtil implements InitializingBean *{* @Value*(*"${aliyun.vod.file.keyid}"*)* private String keyId;  
 @Value*(*"${aliyun.vod.file.keysecret}"*)* private String keySecret;  
 public static String *ACCESS\_KEY\_ID*;  
 public static String *ACCESS\_KEY\_SECRET*;  
  
 @Override  
 public void afterPropertiesSet*()* throws Exception *{  
 ACCESS\_KEY\_ID* = keyId;  
 *ACCESS\_KEY\_SECRET* = keySecret;  
 *}  
}*

### 2、复制工具类到项目中

AliyunVodSDKUtils.java

package com.guli.vod.util;

public class AliyunVodSDKUtils *{* public static DefaultAcsClient initVodClient*(*String accessKeyId, String accessKeySecret*)* throws ClientException *{* String regionId = "cn-shanghai"; // 点播服务接入区域  
 DefaultProfile profile = DefaultProfile.*getProfile(*regionId, accessKeyId, accessKeySecret*)*;  
 DefaultAcsClient client = new DefaultAcsClient*(*profile*)*;  
 return client;  
 *}  
}*

### 3、配置swagger

web和admin

### 4、创建service

接口：VideoService.java

package com.guli.vod.service;

public interface VideoService *{*

String uploadVideo*(*MultipartFile file*)*;  
*}*

实现：VideoServiceImpl.java

package com.guli.vod.service.impl;

@Service

public class VideoServiceImpl implements VideoService *{* @Override  
 public String uploadVideo*(*MultipartFile file*) {* try *{* InputStream inputStream = file.getInputStream*()*;  
 String originalFilename = file.getOriginalFilename*()*;  
 String title = originalFilename.substring*(*0, originalFilename.lastIndexOf*(*"."*))*;  
 UploadStreamRequest request = new UploadStreamRequest*(* ConstantPropertiesUtil.*ACCESS\_KEY\_ID*,  
 ConstantPropertiesUtil.*ACCESS\_KEY\_SECRET*,  
 title, originalFilename, inputStream*)*;  
 UploadVideoImpl uploader = new UploadVideoImpl*()*;  
 UploadStreamResponse response = uploader.uploadStream*(*request*)*;  
 //如果设置回调URL无效，不影响视频上传，可以返回VideoId同时会返回错误码。  
 // 其他情况上传失败时，VideoId为空，此时需要根据返回错误码分析具体错误原因  
 String videoId = response.getVideoId*()*;  
 if *(*!response.isSuccess*()) {* String errorMessage = "阿里云上传错误：" + "code：" + response.getCode*()* + ", message：" + response.getMessage*()*;  
 log.warn*(*errorMessage*)*;  
 if *(*StringUtils.isEmpty*(*videoId*)) {* throw new GuliException*(*20001, errorMessage*)*;  
 *}  
 }* return videoId;  
 *}* catch *(*IOException e*) {* throw new GuliException*(*20001, "guli vod 服务上传失败"*)*;  
 *}  
 }  
}*

### 5、创建controller

VideoAdminController.java

@Api*(*description = "阿里云视频点播微服务"*)*

@CrossOrigin //跨域  
@RestController  
@RequestMapping*(*"/admin/vod/video"*)*public class VideoAdminController *{* @Autowired  
 private VideoService videoService;  
  
 @PostMapping*(*"upload"*)* public R uploadVideo*(* @ApiParam*(*name = "file", value = "文件", required = true*)* @RequestParam*(*"file"*)* MultipartFile file*)* throws Exception *{* String videoId = videoService.uploadVideo*(*file*)*;  
 return R.*ok()*.message*(*"视频上传成功"*)*.data*(*"videoId", videoId*)*;  
 *}  
}*

### 6、启动后端vod微服务

### 7、swagger测试

# 02-前端整合视频上传



## 一、配置nginx反向代理

将接口地址加入nginx配置

location ~ /vod/ {

proxy\_pass http://localhost:8003;

}

配置nginx上传文件大小，否则上传时会有 413 (Request Entity Too Large) 异常

打开nginx主配置文件nginx.conf，找到http{}，添加

client\_max\_body\_size 1024m;

重启nginx

nginx -s reload

## 二、前端实现

### 1、数据定义

        fileList: [],//上传文件列表

        BASE\_API: process.env.BASE\_API // 接口API地址

### 2、整合上传组件

<el-form-item label="上传视频">

    <!-- TODO -->

    <el-upload :on-success="handleVodUploadSuccess" :on-remove="handleVodRemove" :before-remove="beforeVodRemove"

    :on-exceed="handleUploadExceed" :file-list="fileList" :action="BASE\_API+'/admin/vod/video/upload'"

    :limit="1" class="upload-demo">

    <el-button size="small" type="primary">上传视频</el-button>

    <el-tooltip placement="right-end">

        <div slot="content">最大支持1G，<br>

        支持3GP、ASF、AVI、DAT、DV、FLV、F4V、<br>

        GIF、M2T、M4V、MJ2、MJPEG、MKV、MOV、MP4、<br>

        MPE、MPG、MPEG、MTS、OGG、QT、RM、RMVB、<br>

        SWF、TS、VOB、WMV、WEBM 等视频格式上传</div>

        <i class="el-icon-question" />

    </el-tooltip>

    </el-upload>

</el-form-item>

### 3、方法定义

//成功回调

handleVodUploadSuccess(response, file, fileList) {

this.video.videoSourceId = response.data.videoId

},

//视图上传多于一个视频

handleUploadExceed(files, fileList) {

this.$message.warning('想要重新上传视频，请先删除已上传的视频')

},

# 03-删除云端视频



文档：服务端SDK->Java SDK->媒资管理

<https://help.aliyun.com/document_detail/61065.html?spm=a2c4g.11186623.6.831.654b3815cIxvma#h2--div-id-deletevideo-div-7>

## 一、后端

### 1、service

接口

void removeVideo(String videoId);

实现

@Override

public void removeVideo(String videoId) {

try{

DefaultAcsClient client = AliyunVodSDKUtils.initVodClient(

ConstantPropertiesUtil.ACCESS\_KEY\_ID,

ConstantPropertiesUtil.ACCESS\_KEY\_SECRET);

DeleteVideoRequest request = new DeleteVideoRequest();

request.setVideoIds(videoId);

DeleteVideoResponse response = client.getAcsResponse(request);

System.out.print("RequestId = " + response.getRequestId() + "\n");

}catch (ClientException e){

throw new GuliException(20001, "视频删除失败");

}

}

### 2、controller

@DeleteMapping("{videoId}")

public R removeVideo(@ApiParam(name = "videoId", value = "云端视频id", required = true)

@PathVariable String videoId){

videoService.removeVideo(videoId);

return R.ok().message("视频删除成功");

}

## 二、前端

### 1、定义api

api/edu/vod.js

import request from '@/utils/request'

const api\_name = '/admin/vod/video'

export default {

removeById(id) {

return request({

url: `${api\_name}/${id}`,

method: 'delete'

})

}

}

### 2、组件方法

views/edu/course/chapter.vue

import vod from '@/api/edu/vod'

beforeVodRemove(file, fileList) {

return this.$confirm(`确定移除 ${file.name}？`)

},

handleVodRemove(file, fileList) {

console.log(file)

vod.removeById(this.video.videoSourceId).then(response=>{

this.$message({

type: 'success',

message: response.message

})

//清空文件列表

fileList = [];

//删除后情况视频id

this.video.videoSourceId = "";

this.video.videoOriginalName = "";

})

},

# 04-视频文件回显.ziw

## 一、数据库优化冗余字段

### 1、video表中添加一列

video\_original\_name varchar 100 原始文件名称

### 2、pojo中定义新增字段

Video.java、VideoVo.java、VideoForm.java

@ApiModelProperty(value = "云服务器上存储的视频文件名称")

private String videoOriginalName;

## 二、前端修改

### 1、chapter.vue添加videoOriginalName的数据定义

video: {// 课时对象

title: '',

sort: 0,

free: false,

videoSourceId: '',

videoOriginalName: ''

},

### 2、上传成功回调

添加对videoOriginalName的赋值

handleVodUploadSuccess(response, file, fileList) {

this.video.videoSourceId = response.data.videoId

this.video.videoOriginalName = file.name;

},

### 3、修改回调Video

editVideo(videoId) {

this.dialogVideoFormVisible = true

video.getVideoInfoById(videoId).then(response => {

this.video = response.data.item

this.fileList = [{'name': this.video.videoOriginalName}]

})

},

### 4、删除云端video回调

handleVodRemove(file, fileList) {

console.log(file)

vod.removeById(this.video.videoSourceId).then(response => {

this.video.videoSourceId = ''

this.video.videoOriginalName = ''

this.fileList = []

this.$message({

type: 'success',

message: response.message

})

})

},