# SpringCloud相关概念介绍



## 一、什么是微服务

### 1、微服务的由来

微服务最早由Martin Fowler与James Lewis于2014年共同提出，微服务架构风格是一种使用一套小服务来开发单个应用的方式途径，每个服务运行在自己的进程中，并使用轻量级机制通信，通常是HTTP API，这些服务基于业务能力构建，并能够通过自动化部署机制来独立部署，这些服务使用不同的编程语言实现，以及不同数据存储技术，并保持最低限度的集中式管理。

### 2、为什么需要微服务

在传统的IT行业软件大多都是各种独立系统的堆砌，这些系统的问题总结来说就是扩展性差，可靠性不高，维护成本高。到后面引入了SOA服务化，但是，由于 SOA 早期均使用了总线模式，这种总线模式是与某种技术栈强绑定的，比如：J2EE。这导致很多企业的遗留系统很难对接，切换时间太长，成本太高，新系统稳定性的收敛也需要一些时间。

### 3、微服务与单体架构区别

（1）单体架构所有的模块全都耦合在一块，代码量大，维护困难。

         微服务每个模块就相当于一个单独的项目，代码量明显减少，遇到问题也相对来说比较好解决。

（2）单体架构所有的模块都共用一个数据库，存储方式比较单一。

         微服务每个模块都可以使用不同的存储方式（比如有的用redis，有的用mysql等），数据库也是单个模块对应自己的数据库。

（3）单体架构所有的模块开发所使用的技术一样。

         微服务每个模块都可以使用不同的开发技术，开发模式更灵活。

### 4、微服务本质

（1）微服务，关键其实不仅仅是微服务本身，而是系统要提供一套基础的架构，这种架构使得微服务可以独立的部署、运行、升级，不仅如此，这个系统架构还让微服务与微服务之间在结构上“松耦合”，而在功能上则表现为一个统一的整体。这种所谓的“统一的整体”表现出来的是统一风格的界面，统一的权限管理，统一的安全策略，统一的上线过程，统一的日志和审计方法，统一的调度方式，统一的访问入口等等。  
（2）微服务的目的是有效的拆分应用，实现敏捷开发和部署 。  
（3）微服务提倡的理念团队间应该是 inter-operate, not integrate 。inter-operate是定义好系统的边界和接口，在一个团队内全栈，让团队自治，原因就是因为如果团队按照这样的方式组建，将沟通的成本维持在系统内部，每个子系统就会更加内聚，彼此的依赖耦合能变弱，跨系统的沟通成本也就能降低。

### 5、什么样的项目适合微服务

微服务可以按照业务功能本身的独立性来划分，如果系统提供的业务是非常底层的，如：操作系统内核、存储系统、网络系统、数据库系统等等，这类系统都偏底层，功能和功能之间有着紧密的配合关系，如果强制拆分为较小的服务单元，会让集成工作量急剧上升，并且这种人为的切割无法带来业务上的真正的隔离，所以无法做到独立部署和运行，也就不适合做成微服务了。

### 6、微服务开发框架

目前微服务的开发框架，最常用的有以下四个：

Spring Cloud：http://projects.spring.io/spring-cloud（现在非常流行的微服务架构）

Dubbo：http：//dubbo.io

Dropwizard：http://www.dropwizard.io （关注单个微服务的开发）

Consul、etcd&etc.（微服务的模块）

### 7、什么是Spring Cloud

Spring Cloud是一系列框架的集合。它利用Spring Boot的开发便利性简化了分布式系统基础设施的开发，如服务发现、服务注册、配置中心、消息总线、负载均衡、 熔断器、数据监控等，都可以用Spring Boot的开发风格做到一键启动和部署。Spring并没有重复制造轮子，它只是将目前各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过SpringBoot风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包

### 8、Spring Cloud和Spring Boot是什么关系

Spring Boot 是 Spring 的一套快速配置脚手架，可以基于Spring Boot 快速开发单个微服务，Spring Cloud是一个基于Spring Boot实现的开发工具；Spring Boot专注于快速、方便集成的单个微服务个体，Spring Cloud关注全局的服务治理框架； Spring Boot使用了默认大于配置的理念，很多集成方案已经帮你选择好了，能不配置就不配置，Spring Cloud很大的一部分是基于Spring Boot来实现，必须基于Spring Boot开发。可以单独使用Spring Boot开发项目，但是Spring Cloud离不开 Spring Boot。

### 9、Spring Cloud相关基础服务组件

服务发现——Netflix Eureka  （Nacos）

服务调用——Netflix Feign

熔断器——Netflix Hystrix

服务网关——Spring Cloud  GateWay

分布式配置——Spring Cloud Config  （Nacos）

消息总线 —— Spring Cloud Bus （Nacos）

### 10、Spring Cloud的版本

Spring Cloud并没有熟悉的数字版本号，而是对应一个开发代号。

| **Cloud代号** | **Boot版本(train)** | **Boot版本(tested)** | **lifecycle** |
| --- | --- | --- | --- |
| Angle | 1.2.x | incompatible with 1.3 | EOL in July 2017 |
| Brixton | 1.3.x | 1.4.x | 2017-07卒 |
| Camden | 1.4.x | 1.5.x | - |
| Dalston | 1.5.x | not expected 2.x | - |
| Edgware | 1.5.x | not expected 2.x | - |
| Finchley | 2.0.x | not expected 1.5.x | - |
| Greenwich | **2.1.x** |  |  |
| Hoxton | 2.2.x |  |  |

开发代号看似没有什么规律，但实际上首字母是有顺序的，比如：Dalston版本，我们可以简称 D 版本，对应的 Edgware 版本我们可以简称 E 版本。

#### 小版本

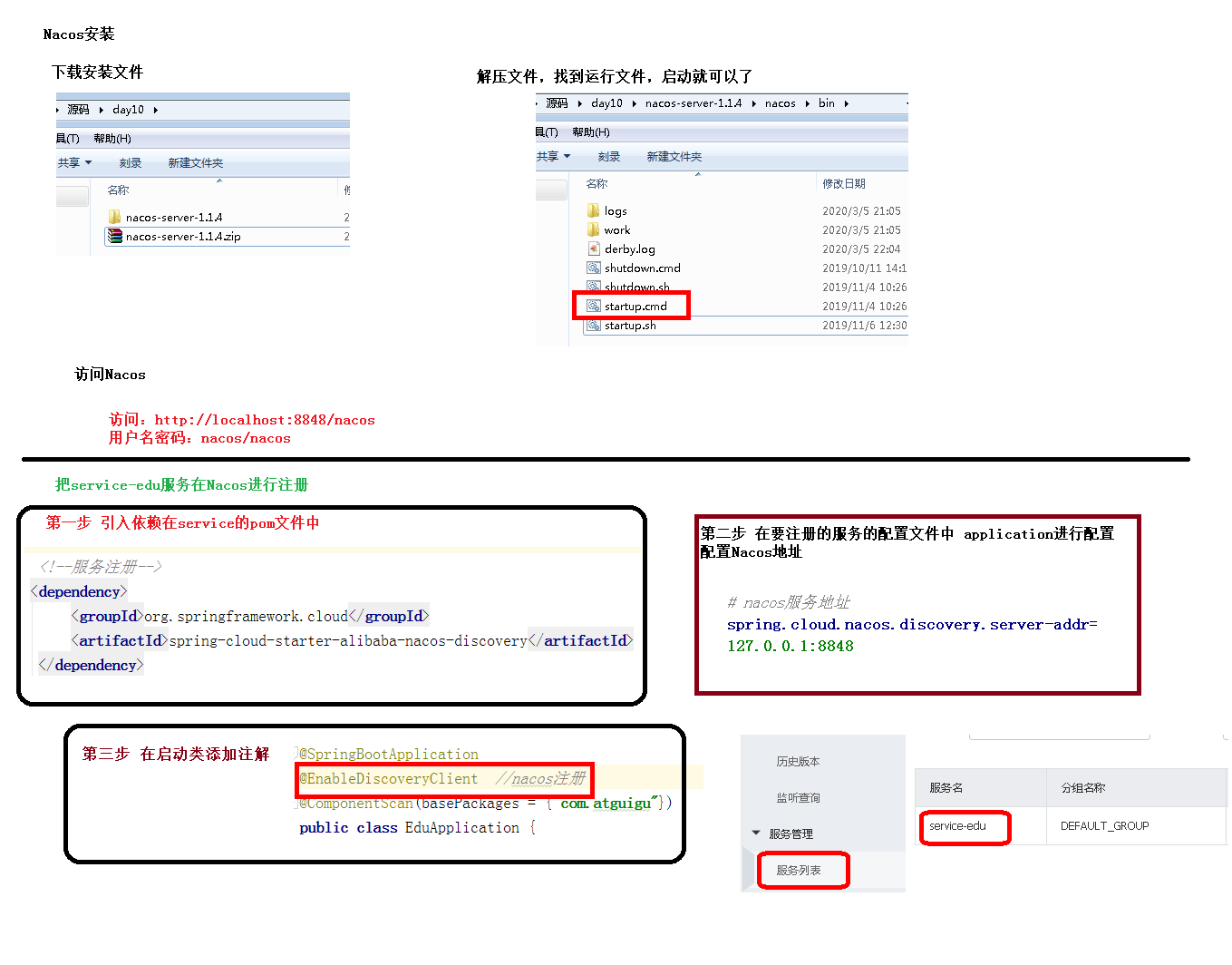
Spring Cloud 小版本分为:

SNAPSHOT： 快照版本，随时可能修改

M： MileStone，M1表示第1个里程碑版本，一般同时标注PRE，表示预览版版。

SR： Service Release，SR1表示第1个正式版本，一般同时标注GA：(GenerallyAvailable),表示稳定版本。

# 服务发现-搭建Nacos服务



## 一、Nacos

### 1、基本概念

**（1）**Nacos 是阿里巴巴推出来的一个新开源项目，是一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。Nacos 致力于帮助您发现、配置和管理微服务。Nacos 提供了一组简单易用的特性集，帮助您快速实现动态服务发现、服务配置、服务元数据及流量管理。Nacos 帮助您更敏捷和容易地构建、交付和管理微服务平台。 Nacos 是构建以“服务”为中心的现代应用架构 (例如微服务范式、云原生范式) 的服务基础设施。

**（2）**常见的注册中心：

1. Eureka（原生，2.0遇到性能瓶颈，停止维护）

2. Zookeeper（支持，专业的独立产品。例如：dubbo）

3. Consul（原生，GO语言开发）

4. Nacos

相对于 Spring Cloud Eureka 来说，Nacos 更强大。Nacos = Spring Cloud Eureka + Spring Cloud Config

 Nacos 可以与 Spring, Spring Boot, Spring Cloud 集成，并能代替 Spring Cloud Eureka, Spring Cloud Config

- 通过 Nacos Server 和 spring-cloud-starter-alibaba-nacos-discovery 实现服务的注册与发现。

**（3）**Nacos是以服务为主要服务对象的中间件，Nacos支持所有主流的服务发现、配置和管理。

Nacos主要提供以下四大功能：

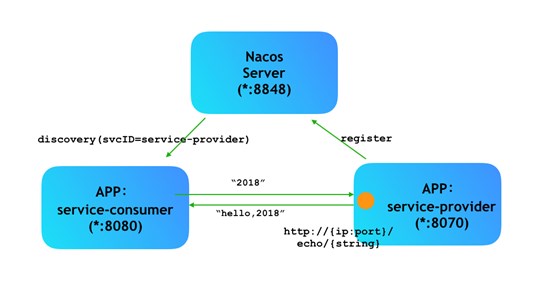
1. 服务发现和服务健康监测

2. 动态配置服务

3. 动态DNS服务

4. 服务及其元数据管理

**（4）**Nacos结构图



### 2、Nacos下载和安装

**（1）下载地址和版本**

下载地址：<https://github.com/alibaba/nacos/releases>

下载版本：nacos-server-1.1.4.tar.gz或nacos-server-1.1.4.zip，解压任意目录即可

### （2）启动nacos服务

- Linux/Unix/Mac

启动命令(standalone代表着单机模式运行，非集群模式)

启动命令：sh startup.sh -m standalone

- Windows

启动命令：cmd startup.cmd 或者双击startup.cmd运行文件。

访问：<http://localhost:8848/nacos>

用户名密码：nacos/nacos





## 二、服务注册（service\_edu为例）

把service-edu微服务注册到注册中心中，service-vod步骤相同

### 1、在service模块配置pom

配置Nacos客户端的pom依赖

<!--服务注册-->

*<*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-alibaba-nacos-discovery*</*artifactId*>  
</*dependency*>*

### 2、添加服务配置信息

配置application.properties，在客户端微服务中添加注册Nacos服务的配置信息

# nacos服务地址

spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848

### 3、添加Nacos客户端注解

在客户端微服务启动类中添加注解

@EnableDiscoveryClient

### 4、启动客户端微服务

启动注册中心

启动已注册的微服务，可以在Nacos服务列表中看到被注册的微服务



# 03-服务调用-Feign

## 一、Feign

### 1、基本概念

* Feign是Netflix开发的声明式、模板化的HTTP客户端， Feign可以帮助我们更快捷、优雅地调用HTTP API。
* Feign支持多种注解，例如Feign自带的注解或者JAX-RS注解等。
* Spring Cloud对Feign进行了增强，使Feign支持了Spring MVC注解，并整合了Ribbon和Eureka，从而让Feign的使用更加方便。
* Spring Cloud Feign是基于Netflix feign实现，整合了Spring Cloud Ribbon和Spring Cloud Hystrix，除了提供这两者的强大功能外，还提供了一种声明式的Web服务客户端定义的方式。
* Spring Cloud Feign帮助我们定义和实现依赖服务接口的定义。在Spring Cloud feign的实现下，只需要创建一个接口并用注解方式配置它，即可完成服务提供方的接口绑定，简化了在使用Spring Cloud Ribbon时自行封装服务调用客户端的开发量。

## 二、实现服务调用



### 1、需求

删除课时的同时删除云端视频

### 2、在service模块添加pom依赖

<!--服务调用-->

*<*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-openfeign*</*artifactId*>  
</*dependency*>*

### 3、在调用端的启动类添加注解

@EnableFeignClients

### 4、创建包和接口

创建client包

@FeignClient注解用于指定从哪个服务中调用功能 ，名称与被调用的服务名保持一致。

@GetMapping注解用于对被调用的微服务进行地址映射。

@PathVariable注解一定要指定参数名称，否则出错

@Component注解防止，在其他位置注入CodClient时idea报错

package com.guli.edu.client;

@FeignClient("service-vod")  
@Component  
public interface VodClient {  
 @DeleteMapping(value = "/eduvod/vod/video/{videoId}")  
 public R removeVideo(@PathVariable("videoId") String videoId);  
}

### 5、调用微服务

在调用端的VideoServiceImpl中调用client中的方法

@Override

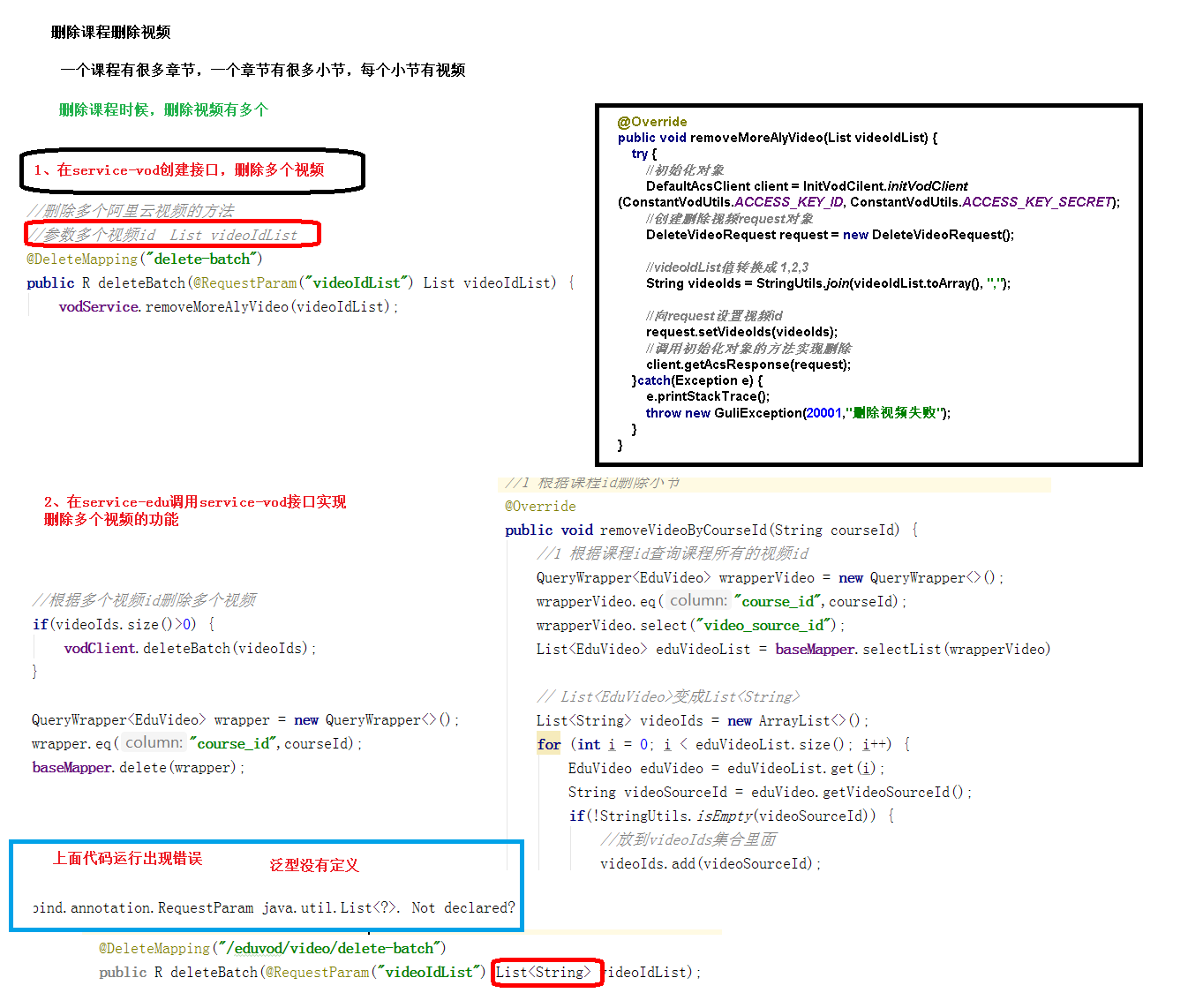
public boolean removeVideoById*(*String id*) {* //查询云端视频id  
 Video video = baseMapper.selectById*(*id*)*;  
 String videoSourceId = video.getVideoSourceId*()*;  
 //删除视频资源  
 if *(*!StringUtils.*isEmpty(*videoSourceId*)) {* vodClient.removeVideo*(*videoSourceId*)*;  
 *}* Integer result = baseMapper.deleteById*(*id*)*;  
 return null != result && result > 0;  
*}*

### 6、测试

启动相关微服务

测试删除课时的功能

# 完善删除课程业务



**需求**

删除课程的同时删除云端视频

## 一、vod服务

### 1、业务

业务接口：VideoService.java

void removeVideoList(List<String> videoIdList);

业务实现：VideoServiceImpl.java

@Override

public void removeVideoList*(*List*<*String*>* videoIdList*) {* try *{* //初始化  
 DefaultAcsClient client = AliyunVodSDKUtils.*initVodClient(* ConstantPropertiesUtil.*ACCESS\_KEY\_ID*,  
 ConstantPropertiesUtil.*ACCESS\_KEY\_SECRET)*;  
 //创建请求对象  
 //一次只能批量删20个  
 String str = org.apache.commons.lang.StringUtils.*join(*videoIdList.toArray*()*, ","*)*;  
 DeleteVideoRequest request = new DeleteVideoRequest*()*;  
 request.setVideoIds*(*str*)*;  
 //获取响应  
 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、web层接口

controller：VideoAdminController.java

/\*\*

\* 批量删除视频

\* @param videoIdList

\* @return

\*/

@DeleteMapping("delete-batch")

public R removeVideoList(

@ApiParam(name = "videoIdList", value = "云端视频id", required = true)

@RequestParam("videoIdList") List videoIdList){

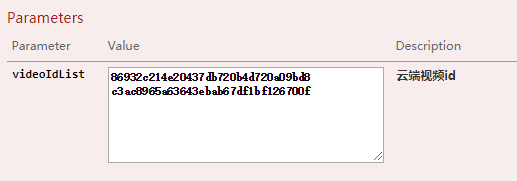
videoService.removeVideoList(videoIdList);

return R.ok().message("视频删除成功");

}

### 3、Swagger测试

输入多个id，每个一行



## 二、edu服务

### 1、client

VodClient.java

@DeleteMapping(value = "/admin/vod/video/delete-batch")

public R removeVideoList(@RequestParam("videoIdList") List<String> videoIdList);

### 2、业务

VideoServiceImpl.java

@Override

public boolean removeByCourseId*(*String courseId*) {* //根据课程id查询所有视频列表  
 QueryWrapper*<*Video*>* queryWrapper = new QueryWrapper*<*>*()*;  
 queryWrapper.eq*(*"course\_id", courseId*)*;  
 queryWrapper.select*(*"video\_source\_id"*)*;  
 List*<*Video*>* videoList = baseMapper.selectList*(*queryWrapper);  
 //得到所有视频列表的云端原始视频id  
 List*<*String*>* videoSourceIdList = new ArrayList*<>()*;  
 for *(*int i = 0; i < videoList.size*()*; i++*) {* Video video = videoList.get*(*i*)*;  
 String videoSourceId = video.getVideoSourceId*()*;  
 if *(*!StringUtils.*isEmpty(*videoSourceId*)) {* videoSourceIdList.add*(*videoSourceId*)*;  
 *}  
 }* //调用vod服务删除远程视频  
 if *(*videoSourceIdList.size*()* > 0*) {* vodClient.removeVideoList*(*videoSourceIdList*)*;  
 *}* //删除video表示的记录  
 QueryWrapper*<*Video*>* queryWrapper2 = new QueryWrapper*<*>*()*;  
 queryWrapper2.eq*(*"course\_id", courseId*)*;  
 Integer count = baseMapper.delete*(*queryWrapper2);  
 return null != count && count > 0;  
*}*

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*)*;  
 //根据id删除所有课程详情  
 courseDescriptionService.removeById*(*id*)*;  
 //删除封面 *TODO 独立完成* Integer result = baseMapper.deleteById*(*id*)*;  
 return null != result && result > 0;  
*}*

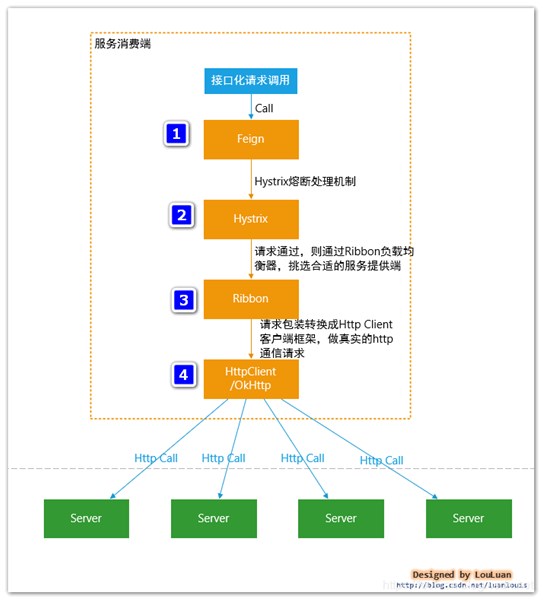
# 05-熔断器

## 一、Hystrix基本概念

### 1、Spring Cloud调用接口过程

Spring Cloud 在接口调用上，大致会经过如下几个组件配合：

**Feign ----->Hystrix —>Ribbon —>Http Client（apache http components 或者 Okhttp）** 具体交互流程上，如下图所示：



**（1）接口化请求调用**当调用被@FeignClient注解修饰的接口时，在框架内部，将请求转换成Feign的请求实例feign.Request，交由Feign框架处理。

**（）Feign** ：转化请求Feign是一个http请求调用的轻量级框架，可以以Java接口注解的方式调用Http请求，封装了Http调用流程。

**（）Hystrix**：熔断处理机制 Feign的调用关系，会被Hystrix代理拦截，对每一个Feign调用请求，Hystrix都会将其包装成HystrixCommand,参与Hystrix的流控和熔断规则。如果请求判断需要熔断，则Hystrix直接熔断，抛出异常或者使用FallbackFactory返回熔断Fallback结果；如果通过，则将调用请求传递给Ribbon组件。

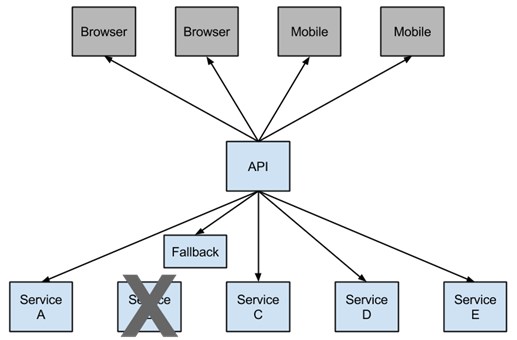
**（）Ribbon**：服务地址选择 当请求传递到Ribbon之后,Ribbon会根据自身维护的服务列表，根据服务的服务质量，如平均响应时间，Load等，结合特定的规则，从列表中挑选合适的服务实例，选择好机器之后，然后将机器实例的信息请求传递给Http Client客户端，HttpClient客户端来执行真正的Http接口调用；

**（）HttpClient** ：Http客户端，真正执行Http调用根据上层Ribbon传递过来的请求，已经指定了服务地址，则HttpClient开始执行真正的Http请求

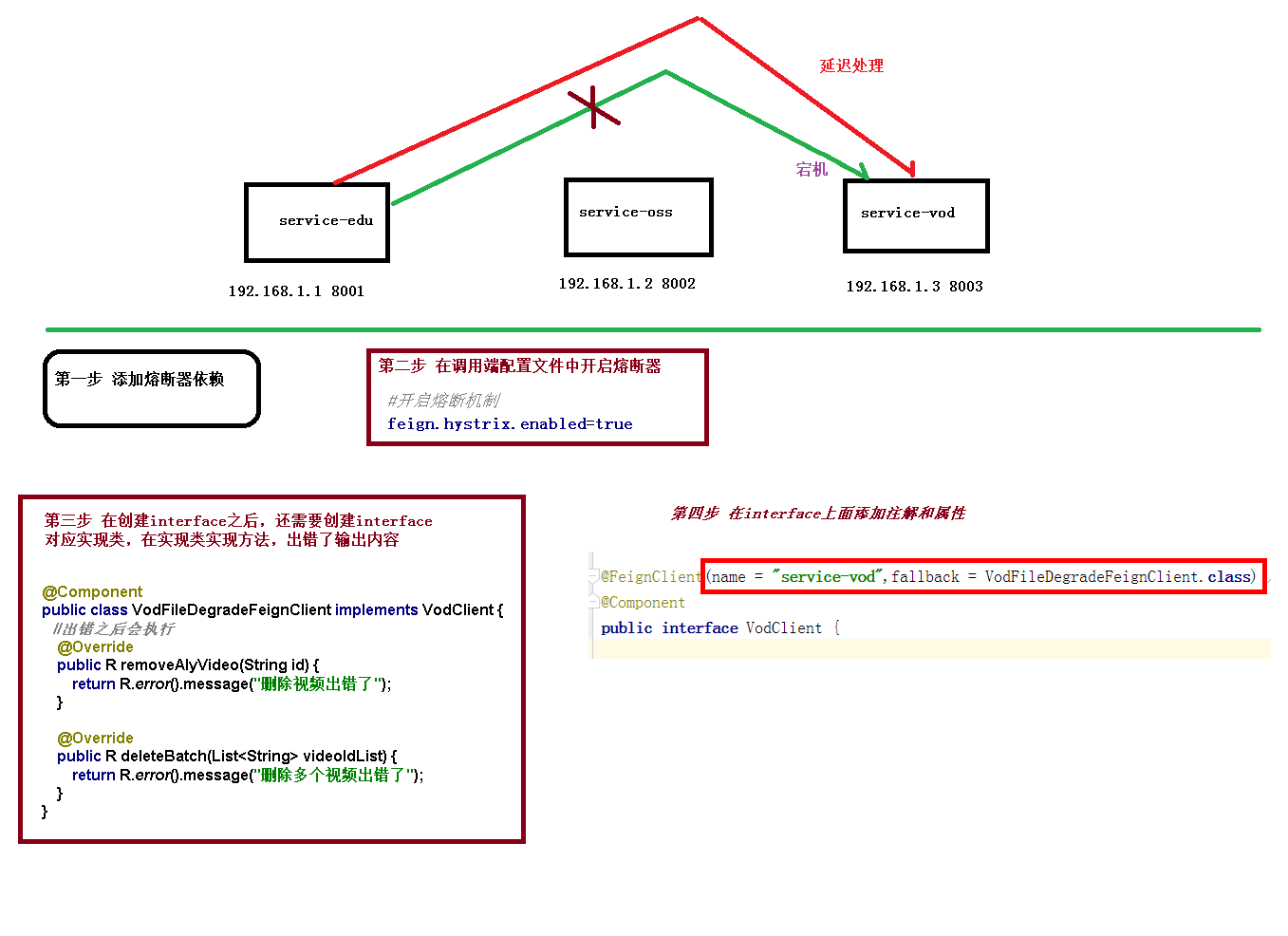
### 2、Hystrix概念

Hystrix 是一个供分布式系统使用，提供延迟和容错功能，保证复杂的分布系统在面临不可避免的失败时，仍能有其弹性。

比如系统中有很多服务，当某些服务不稳定的时候，使用这些服务的用户线程将会阻塞，如果没有隔离机制，系统随时就有可能会挂掉，从而带来很大的风险。SpringCloud使用Hystrix组件提供断路器、资源隔离与自我修复功能。下图表示服务B触发了断路器，阻止了级联失败

****

## 二、feign结合Hystrix使用



**改造service-edu模块**

### 1、在service的pom中添加依赖

*<*dependency*>*

*<*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-netflix-ribbon*</*artifactId*>  
</*dependency*>*<!--hystrix依赖，主要是用 @HystrixCommand -->  
*<*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-netflix-hystrix*</*artifactId*>  
</*dependency*>*<!--服务注册-->  
*<*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-alibaba-nacos-discovery*</*artifactId*>  
</*dependency*>*<!--服务调用-->  
*<*dependency*>  
 <*groupId*>*org.springframework.cloud*</*groupId*>  
 <*artifactId*>*spring-cloud-starter-openfeign*</*artifactId*>  
</*dependency*>*

### 2、在配置文件中添加hystrix配置

#开启熔断机制

feign.hystrix.enabled=true  
# 设置hystrix超时时间，默认1000ms  
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=6000

### 3、在service-edu的client包里面创建熔断器的实现类

@Component

public class VodFileDegradeFeignClient implements VodClient *{* @Override  
 public R removeVideo*(*String videoId*) {* return R.*error()*.message*(*"time out"*)*;  
 *}* @Override  
 public R removeVideoList*(*List videoIdList*) {* return R.*error()*.message*(*"time out"*)*;  
 *}  
}*

### 4、修改VodClient接口的注解

@FeignClient*(*name = "service-vod", fallback = VodFileDegradeFeignClient.class*)*

@Component  
public interface VodClient *{* @DeleteMapping*(*value = "/eduvod/vod/{videoId}"*)* public R removeVideo*(*@PathVariable*(*"videoId"*)* String videoId*)*;  
  
 @DeleteMapping*(*value = "/eduvod/vod/delete-batch"*)* public R removeVideoList*(*@RequestParam*(*"videoIdList"*)* List videoIdList*)*;  
*}*

### 5、测试熔断器效果