SC笔记

# 网站架构演变过程

## 传统架构(单点应用)

ssh或ssm项目，会将整个业务模块放在一个项目中，分为mvc架构、逻辑层、数据库访问层、控制层。

优点：适合单人或小团队开发。

缺点：耦合度太高，一个模块的失败，可能导致全项目出错，例如一个模块将数据库连接占完，则其它模块就不能访问了。

## 分布式架构

基于传统架构演变过来的。将多个模块分开为多个项目，比如登陆、积分模块分成两个项目，每个项目有自己的数据库。每个项目之间没有关系。

项目：包含业务逻辑层和视图层，即包括前台项目和后台项目。

服务：只包含义务逻辑层，没有视图层。

优点：耦合度降低、粒度更细。

## SOA架构

面向服务架构，俗称服务化，将共同的业务代码抽取出来，提供给其它接口调用，服务之间采用rpc远程调用。

服务：将共同的业务逻辑进行拆分，拆分成独立一个项目进行部署。比如前台服务调用后台服务。

特点：底层基于SOAP(简单对象协议http+xml混合物)或者ESB(消息总线)实现，底层使用HTTP或https协议 + 重量级XML数据交换格式进行通讯。在后面微服务中，以json格式替代xml。

缺点：

* 依赖中心化服务发现机制。
* SOAP的xml占用宽带，xml报文冗余大，高并发时不太好。
* 服务管理混乱，缺少服务管理和治理设施不完善。

rpc远程调用技术框架(核心底层socket技术或netty实现)：

* httpclient
* springcloud
* dubbo
* grpc

## 微服务架构

基于SOA架构演变过来，粒度比SOA更加精细，目的是为了提高效率。每个服务之间互不影响，且必须独立部署(独立数据库、独立redis等)，更加轻量级，采用restful风格提供API，也就是Http协议和Json格式，更加轻巧，适合敏捷开发。

SOA架构与微服务架构的区别：

* 微服务基于SOA，但是去除了SOA中的ESB消息总线，采用http+json(restful)进行传输。
* 微服务架构粒度更加精细，服务之间互不影响，每个微服务独立部署，更加轻巧。
* SOA架构中数据库存储会发生共享，微服务强调每个服务都是单独数据库。
* 微服务更适合敏捷开发，版本迭代快。

# SpringCloud简介

## 为什么使用SC

* 是目前一套比较完整的微服务解决方案框架。
* 解决了一套问题，而不是某个问题。

## 定义

sc是基于sb基础上开发的微服务框架，包括服务治理、注册中心、配置管理、断路器、智能路由、微代理、控制总线、全局锁、分布式会话。

# Eureka

## 相关概念

### 注册中心

存放服务地址相关信息(接口地址)

### 服务注册

将服务信息注册到注册中心上。

### 服务发现

从注册中心上获取服务信息。

### 服务提供者

提供服务接口方。

### 服务消费者

调用别人接口进行使用。一个服务可以既是消费者又是提供者。

## SC支持的注册中心

Eureka、Consule、Zookeeper

## 服务注册原理

* 启动注册中心。
* 启动A服务。
* A服务将服务地址和端口以别名方式注册到注册中心(serviced: Aservice value: 127.0.0.1:8080)，一般会有多个一模一样的A服务，用于分担压力。所以在注册中心会显示一个serviceId对应多个ip。
* B消费者调用接口时，使用服务别名，即serviceId去注册中心获取rpc远程调用地址，对于SC微服务来说，B会将serivceId对应的所有ip和port缓存到jvm(每30s刷新一次)，使用本地负载均衡决定调用哪个微服务。
* 决定后使用HttpClient实现调用。

## 集群配置查看代码

## Eureka自我保护机制

为了防止EurekaClient与Server网络不通的情况下，Server误将Client剔除的情况。

默认情况下client会定时向server发送心跳包。如果server在一段时间内未收到client发送的心跳包，便会直接从服务注册列表中剔除该服务(默认90s)。但在短时间内丢失了大量的服务实例心跳，这时候server会开启自我保护机制，不去剔除该服务。

建议在本地开发时禁止自我保护。生产环境开启自我保护(注意本地保存的30s时间)。具体配置查看服务端：sc2-eureka-server的application.yml，客户端：eureka-client-member的application.yml。

但是如果client真的宕机了，则调用端应该有重试机制、保证接口网络延迟幂等性、服务降级功能。

## zookeeper代替Eureka作为注册中心

zookeeper节点类型是临时节点。

停止服务端，在zk服务器端有缓存，会过段时间节点才消失。

查看eureka-client-order调用eureka-client-member

## consul代替Eureka作为注册中心

在consul.exe的文件夹cmd，输入consul agent -dev -ui -node=cy。

默认界面：<http://localhost:8500/>。

consul服务与发现查看模块

## DiscoveryClient类的使用

获取注册中心上的一些信息。具体查看sc-zookeeper-client-order模块。

## 本地负载均衡与服务端负载均衡区别

本地负载均衡ribbon(A服务调用B服务)：A去注册中心查找所有B服务(根据服务名，可以使用DicoveryClient类)，缓存到A服务的本地jvm，使用取模方式，轮询本地远程调用B服务。适合在微服务rpc远程调用中，dubbo、sc等。

服务端负载均衡nginx：客户端将所有请求交给nginx，让nginx实现转发请求。适合于针对于服务器端的，如tomcat、jetty。

# 远程调用

## RestTemplate

参考spring-cloud-xyb：eureka-client-order：com.xyb.api.controller.OrderApiController。

## Feign

是一个声明式http客户端调用工具，采用接口+注解实现。查看eureka-client-order。feign默认开启本地负载均衡。

# SC整合Hystrix

## 服务雪崩效应

默认情况下tomcat只有一个线程池去处理请求，比如50个线程，在高并发情况下，如果客户端所有请求堆积到同一个服务接口上，tomcat所有线程回去处理，导致其它接口“无法被访问” (其实还是会被访问，不过在卡着)。

## 连环雪崩效应

A服务会调用B服务，当高并发时，A服务出现雪崩效应，导致B服务的某接口也无法访问，连环雪崩。

## Hystrix解决服务雪崩原理

### 断路器

### 服务降级

在高并发情况下，防止用户一直等待。在tomcat中没有线程进行处理客户端请求，不应该一直让用户一直在转圈等待。返回给用户一个友好提示，不会去处理请求，调用fallback本地方法，提高用户体验。

如果调用其它接口超时的时候(默认是1s)，如果1s内没有响应，默认情况下业务逻辑是可以执行的，但是直接执行服务降级方法。所以建议将hystrix的超时时间设置稍微大点，默认1s。

### 服务熔断

目的是为了保护服务，在高并发情况下，如果请求达到了一定的极限(可以自定义阈值)，如果流量超出了阈值，自动开启保护服务功能，使用服务降级方式返回友好提示。服务熔断和服务降级一起使用。

### 服务隔离机制

有线程池隔离和信号量隔离两种方式。

#### 线程池隔离原理

在tomcat中开启多个线程池，每个线程池处理一个访问接口，则即使一个接口被一直占用，不会影响其它接口。但是会使CPU占用率非常高。

# Spring Cloud Config分布式配置中心

## 什么是分布式配置中心

在微服务中使用同一个服务器管理所有服务配置文件信息，能够实现后台可管理，当服务器正在运行时，如果配置文件需要发生改变，可以不需要重启服务器实时更改配置文件信息。

## 为什么使用分布式配置中心

产生背景：在微服务如果使用传统的方式管理配置文件，配置文件管理起来非常复杂。

如果生产环境配置文件改变，需要重新打包，发布重启，将配置文件读取到jvm。

## 分布式配置中心框架

* 阿波罗：携程写分布式配置中心有图形界面可管理配置文件信息，配置文件信息存放数据库里面。
* SC-config：没有后台可管理分布式配置中心，配置文件放在版本控制器里面(git、svn等)。
* zookeeper：使用zk实现分布式配置中心，持久节点+事件通知。

## sc-config分布式配置中心原理

### 需要的组件

* web管理系统--后台可以使用图像界面管理配置文件，sc-config没有。
* 存放分布式配置文件服务器(持久储存服务器)--可使用git。配置文件名称：服务名-环境(dev、prd).yml(或properties)。
* ConfigServer缓存配置文件服务器(临时缓存存放)，从git或svn等读取配置文件，并缓存到此服务器。
* ConfigClient读取ConfigServer配置文件信息，并保存到本地。

## 步骤

### 搭建git

目的：持久化存储配置文件。git环境上文件夹以项目进行区分。即一个模块创建一个文件夹，不同模块的文件夹不要相同。

### 公司项目中环境的区分

* dev 开发
* sit 测试
* pre 预发布
* prd 准生产

### 在git上存放配置文件

### 搭建ConfigServer

查看sc-parent：sc-config-server

### 搭建ConfigClient

查看sc-parent：sc-config-client

### 如何实时获取git上更新后的文件actuator

#### 手动刷新(建议方式)

需要人工调用接口，读取最新配置文件(监控中心)。查看查看sc-parent：sc-config-client

#### 自动刷新(监听影响效率)

消息总线进行实时通知--springbus

# Zuul网关

## 产生背景

在面向微服务架构和微服务背景下产生，都是为了解耦，在rpc远程调用中产生的。

## 接口分类

### 开放接口

其它机构合作伙伴进行调用(必须在外网访问)，如蚂蚁开放平台、微信公众号开发等。需要通过appid+appsocet生成accessToken进行通讯。对接支付开发、微信开发。目的可以授权一些接口权限OAuth2.0协议方式第三方联合登陆。

### 内部接口

一般只能在局域网中进行访问，服务与服务调用之间的关系都在同一个微服务中，目的是为了保证安全问题。

## 网关

相当于客户端请求统一先请求到网管服务器上，在由网关服务器进行转发到实际服务器上。zuul整合了ribbon，可实现本地负载均衡。

### 作用

可以拦截客户端所有请求，对该请求进行权限控制、负载均衡、日志管理、接口调用监控、黑白名单、安全控制(统一异常处理、xxs、sql注入)等。

### 与过滤器的区别

过滤器适合单个Tomcat服务器进行拦截请求，网管是拦截整个微服务所有请求。

## Nginx和Zuul的异同

### 相同点

都可实现负载均衡、反向代理、过滤请求、实现网关效果。

### 不同点

* Nginx用C编写，Zuul用java编写。注意网关要用微服务一样的语言，否则要多学一门语言，没那闲时间。
* 负载均衡实现：Zuul采用ribbon+eureka实现，Nginx采用服务器端实现。
* nginx可以整合脚本语言。
* nginx适合服务器端负载均衡+也可以实现网关。Zuul适合微服务中实现网关。
* 最好使用nginx+zuul实现网关，nginx实现反向代理，zuul对微服务网关拦截。

## zuul过滤器

查看sc-parent：sc-zuul-gateway：com.xyb.filter.TokenFilter。

## zuul集群

Nginx+zuul：客户端发送请求到nginx，再使用niginx实现反响代理和负载均衡，采用轮询算法转发到zuul上。

|  |
| --- |
|  |

# swagger

## 传统api文档编写缺点

* api返回信息不明确

## swagger优点

* 功能丰富：支持多种注解，自动生成接口文档界面，支持再界面测试API接口功能。
* 及时更新：开发过程中花一点写注释的时间，就可以及时更新API文档。
* 整合简单：通过添加pom依赖和简单配置，内嵌于应用中就可以同时发布API接口文档界面，不需要部署独立服务。

## 模块

查看sc-parent：sc-swagger

用zuul管理swagger接口，查看sc-parent：sc-zuul-gateway的启动类。

# sc-bus消息总线

## 原理

|  |
| --- |
|  |

原本来说，当配置中心配置改变时，所有相关的微服务都要重新从配置中心下载，sc是向每个相关的微服务发消息让其去下载，当微服务很多的时候会很麻烦。

总线是当客户端通知一个微服务刷新配置文件时，此服务发送消息到队列服务器(发布订阅模式)，所有包含相关配置文件的微服务都会在此服务器上监听，队列服务器接收到通知后，会通知剩余相关微服务更改信息。

## 步骤

### 搭建分布式配置中心config

### 存放文件到git或svn

### 搭建分布式配置中心服务

### 客户端读取分布式中心服务

# sc-stream(消息驱动)

## 什么是sc-stream

可以简化开发人员对消息中间件的使用复杂度，让系统开发人员更多尽力专注与核心业务逻辑的开发。

## 流程

生产者：生产者将消息发送给通道，通道绑定mq。具体查看sc-parent：sc-stream-producer。