# 单体项目的问题

## 什么是单体项目

传统的项目系统，都是单体项目。

互联网领域里单体项目表示一个所有功能集中的项目。

例如一个系统集中实现了订单功能、用户功能，如果做功能扩展，像购物车，商品，秒杀等等功能都要集中在一个运行程序中。

简言之：功能集中到一起的项目--->>单体项目

## 单体项目的问题

### 功能强耦合

开发过程中，企业、架构师，一般会把不同的功能交给不同的人员开发，每个开发者只需要了解自己领域里的业务知识。而单体项目难以满足这样需求。

例如:订单支付逻辑，完成积分新增逻辑，造成一个开发者，团队接触过多的业务，导致系统，企业的管理复杂。

### 并发集中

所有功能最终都运行在同一个系统中，其中某个功能在高并发情况下导致系统的资源占满，其他功能都不能使用。

需要从框架的角度去解决这个问题---微服务框架

# 项目拆分(单体项目的初步解决)

## 横向拆分

比如利用maven这种管理工具，将一个大型项目分模块交给不同人员开发，最终还是运行一个应 用的过程。所以横向拆分不能解决单体项目问题。

|  |
| --- |
|  |

## 纵向拆分

是按照系统功能进行拆分，每个拆分出来的内容，都可以独立运行、独立接受前端请求。

|  |
| --- |
|  |

### 解决了功能强耦合，如何解决系统之间的调用问题？

需要增加接口，功能之间相互调用

例如：订单支付完成，调用积分接口增加积分

|  |
| --- |
| public void payOrder*(*String orderId*) {  //利用orderId将数据库中数据查询 order对象* Order order = orderMapper.selectOrderById*(*orderId*)*;  *//积分更新功能，从订单发送一个请求访问/user-update* String url = "http://localhost:9001/update-point;  RestTemplate template = new RestTemplate*()*;  Integer success = template.getForObject*(*url，Integer.class*)*;  if *(*success != 1*) {  //更新积分失败*  *} }* |

### 负载均衡调用

需微服务架构解决

# 分布式事务

# 微服务思想

## 纵向拆分结构的潜在问题

### Nginx静态文件配置过于复杂

纵向拆分通常使用Ngnix做网关实现负载均衡。

随着业务功能变得越来越复杂，拆分的系统越来越多，对应Nginx Upstream静态配置越来越多.不仅多，维护麻烦。

### 故障调用没有处理机制

在负载均衡过程中，在拆分的系统中，存在系统之间的内部调用，随着业务功能越来越复杂，系 统之间调用关系越多，一旦出现某一个环节由于节点故障网络波动导致无法访问、超时，长此以往， 一定会导致整个集群的瘫痪。

目前拆分结构的问题不止这二个问题，还有其他的各种各样的潜在的问题，所以最终解决单体项目的问题不仅要通过项目的纵向拆分，还要引入架构技术管理这些拆分之后的系统-🡪微服务架构.

## 微服务概念

### 微

微不是描述的功能简单，系统单一，而是经过纵向拆分之后，系统相对于原有项目变的微小了。

### 服务

拆分出来的独立运行的系统，每一个具体的功能都需要被别人调用，叫做服务的调用，所以每一个具体的功能都是一个服务。

## 微服务解决方案

### SpringBoot + dubbo

* springboot构建项目
* dubbo形成微服务管理框架

### SpringBoot+SpringCloud

* springboot构建项目
* springcloud形成微服务管理框架

### Springboot +Spring Cloud Alibaba

* springcloud整合alibaba的微服务框架+springboot构建

## SpringCloud微服务框架

由Spring团队整合市面成熟技术形成的一套解决微服务问题的框架技术集合。

包括：

* Eureka：服务治理
* Ribbon：负载均衡
* Zuul：网关
* springcloud config：配置组件
* feign：Feign时提供负载均衡的Http客户端
* hystrix：熔断器组件

# Eureka服务治理组件

## 简介

Netflix公司的服务注册发现机制的技术，被SpringCloud拿来实现服务治理，SpringCloud中唯一可以使用的一个服务治理组件。SpringCloud核心组件，没有这个组件，无法实现其他的微服务功能。

## 功能

* 客户端：Eureka-Client，就是配合拆分的项目，实现注册与发现。
* 服务端：Eureka -Server，管理所有客户端的注册信息。

## Eureka治理组件的结构

|  |
| --- |
|  |

# 搭建注册中心Eureka-Server

## 创建一个父级工程

### 继承springboot-parent

|  |
| --- |
| <!--继承spring boot--> <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.9.RELEASE</version> </parent> |

### 定义为父工程类型

|  |
| --- |
| <!--定义为父工程--> <packaging>pom</packaging> |

### 定义子工程spring cloud组件版本

|  |
| --- |
| <!--定义所有子工程的spring cloud组件版本--> <!--子工程可不写version 默认使用该版本--> <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Edgware.RELEASE</version>  <!--导入父级工程-->  <scope>import</scope>  <!--父级工程类型-->  <type>pom</type>  </dependency>  </dependencies> </dependencyManagement> |

### pom.xml完整配置

|  |
| --- |
| <?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">  <modelVersion>4.0.0</modelVersion>  <groupId>cn.shu</groupId>  <artifactId>eureka-demo-parent</artifactId>  <version>1.0-SNAPSHOT</version>  <!--定义为父工程-->  <packaging>pom</packaging>  <!--继承spring boot-->  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.9.RELEASE</version>  </parent>  <!--定义所有子工程的spring cloud组件版本-->  <!--子工程可不写version 默认使用该版本-->  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Edgware.RELEASE</version>  <!--导入父级工程-->  <scope>import</scope>  <!--父级工程类型-->  <type>pom</type>  </dependency>  </dependencies>  </dependencyManagement> </project> |

## 创建子工程(Eureka注册中心)

### 继承父工程(刚创建的)

|  |
| --- |
| *<*parent*>  <*artifactId*>*eureka-demo-parent*</*artifactId*>  <*groupId*>*cn.shu*</*groupId*>  <*version*>*1.0-SNAPSHOT*</*version*> </*parent*>* |

### 添加Eureka-Server依赖

|  |
| --- |
| *<*dependencies*>  <*dependency*>  <*groupId*>*org.springframework.cloud*</*groupId*>  <*artifactId*>*spring-cloud-starter-eureka-server*</*artifactId*>  </*dependency*> </*dependencies*>* |

### pom.xml完整配置

|  |
| --- |
| <?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">  <!--继承父工程-->  <parent>  <artifactId>eureka-demo-parent</artifactId>  <groupId>cn.shu</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>eureka-demo-server01</artifactId>  <!--导入 eureka server依赖-->  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka-server</artifactId>  </dependency>  </dependencies> </project> |

### 创建启动类

需使用@EnableEurekaServer注解标识为“注册中心”

|  |
| --- |
|  |

## 创建配置文件(application.yml)

|  |
| --- |
| *#定义端口* server:  port: 8001 spring:  application:  *#定义服务名称* name: eureka-demo-server01 eureka:  instance:  *#开启ip优先，本地运行开不开都一样，但是一旦发布到服务器，必须开启外界访问的ip  #否则可能会使用默认的域名，比如localhost* prefer-ip-address: true  client:  *#eureka-server也会向自己发起注册申请 先关闭自我注册，并关闭抓取  #因为当server启动时，自己本身未启动成功，再自己注册到自己会抛异常* register-with-eureka: false  fetch-registry: false  *#当前注册中心作为客户端 可以访问的 其它注册中心地址  #这里写的自己 自己注册到自己？* service-url:  defaultZone: http://127.0.0.1:8001/eureka |

## 搭建SpringBoot的Eureka-Server的过程

一旦Maven依赖了Eureka-Server则 SpringBoot自动实现整个配置过程，启动时运行Eureka-Server进程。

搭建的Eureka-Server名字叫做注册中心。默认情况下把当前工程运行的进程看成服务端，也是客户端(默认将子自己的服务信息注册到注册中心)

* **服务名称**：高可用时，多个Eureka-Server共用的一个服务名称。
* **ip优先**：如果不设置ip优先，所有治理通信使用域名，本机localhost无法在服务器集群中使用，设为true则使用ip地址。
* **关闭抓取**：由于注册中心自己把自己看成eurekaclient，如果不关闭注册抓取，会在启动时报错，找不到服务端。
* **注册中心地址**：高可用结构中，多个注册中心使用配置的值相互注册。

# 搭建客户端Eureka-Client

客户端功能需要携带信息到“注册中心”注册数据，等待被别人(外界客户端，外界浏览器，内部其他服务)调用发现。所以Eureka-Client组件所在的工程都是微服务--服务提供者。

## 创建子工程

### 配置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">  <!--继承父工程-->  <parent>  <artifactId>eureka-demo-parent</artifactId>  <groupId>cn.shu</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>eureka-demo-client01</artifactId>  <!--导入 eureka client依赖-->  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka</artifactId>  </dependency>  </dependencies> </project> |

### 创建配置文件

|  |
| --- |
| *#定义端口* server:  port: 8101 spring:  application:  *#定义客户端名称* name: eureka-demo-client01 eureka:  instance:  *#开启ip优先，本地运行开不开都一样，但是一旦发布到服务器，必须开启外界访问的ip  #否则可能会使用默认的域名，比如localhost* prefer-ip-address: true  *#注册中心地址 客户端回去注册* service-url:  defaultZone: http://127.0.0.1:8001/eureka |

### 创建启动类

|  |
| --- |
| */\*\*  \* @作者 舒新胜  \* @项目 eureka-demo-parent  \* @创建时间 2020-5-25 22:29  \*/* @SpringBootApplication //客户端生效 @EnableEurekaClient public class StarterEurekaClient {  public static void main(String[] args) {  SpringApplication.*run*(StarterEurekaClient.class，args);  } } |

## 登录注册中心查看注册情况

|  |
| --- |
|  |

# Eureka治理组件的原理

## eureka客户端:

通过配置注册中心地址 <http://127.0.0.1:8001/eureka>， 对客户端提供了一个可以访问注册中心的接口地址，客户端总会通过http请求访问这个地址实现各种功能的调用。

|  |
| --- |
| client:  *#注册中心地址 客户端回去注册* service-url:  defaultZone: http://127.0.0.1:8001/eureka |

### 注册

客户端会在启动时，携带自身信息(服务名称，ip地址，端口号，等等)，在注册中心维护一份注册信息，注册的开启与关闭就是通过配置eureka.client.register-with-eureka实现的。

### 抓取

客户端会访问注册中心服务端，实现每隔30秒中抓取一次注册中心的注册信息。

抓取注册中心的双层map信息。通过http抓取吗？

手动抓取注册中心数据：设置请求头指定希望返回的数据类型

* + - 获取全部应用信息：<http://127.0.0.1:8888/eureka/apps>
    - 获取指定应用信息(autowechat)：http://127.0.0.1:8888/eureka/apps/autowechat

|  |
| --- |
|  |
|  |

### 心跳

客户端为了保证让服务端总是记录自己的信息而不做超时剔除，会每隔30秒钟发送一次心跳。

## eureka服务端

### 管理注册

根据Eureka客户端请求携带的信息，在内存中生成一份当前微服务集群里所有微服务服务信息 (双层map)，而客户端抓取的注册信息就是这个双层map对象。

双层map代码结构 ：map<String，map<String，Instance>>

|  |
| --- |
|  |

* + **第一层Map** key（**服务名称**）->value(第二层map)

整个集群中，同一个服务名称只能在一个注册中心中存在一个key

* + **第二层Map** key（**实例名称。默认结构 域名:服务名称:端口号**）->>value（**该实例的详细信息，包含ip 端口 时间戳等**）

### 超时监听

每隔60秒，判断一次注册信息中所有实例的最后时间戳（心跳一次刷新）是否超时（距离现在时间超过90秒），如果客户端没有发送心跳检测，则会超过时间点，注册中心将会把实例从map对象中key-value剔除。

|  |
| --- |
|  |

### 保护机制

为了防止由于网络波动导致的大面积的服务提供者未能有效的发送心跳续约，从而按照超时逻辑被剔除，而被剔除的服务并没有宕机故障。会造成微服务不可用，甚至瘫痪。所以出现了保护机制。当eureka服务端，一次性判断大量服务宕机故障（15%以上），会认为是网络波动导致没有接收到客户端的心跳信息，而不是真正宕机故障。这时开启保护机制，保存所有微服务不剔除。阈值（15%）测试环境中非常容易达到。

达到阈值后会有如下提示：

|  |
| --- |
| EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE. |

在注册中心中，可关闭自我保护机制

|  |
| --- |
| eureka.server.enable-self-preservation=false |

# eureka治理中专业名词

* 注册中心：是服务端，也是客户端。
* 服务提供者（服务实例）：实际就是客户端所在的工程。

# Eureka的高可用结构

## 客户端集群

一旦将多个客户端以同一个服务提供者身份启动并注册到注册中心，为同一个服务提供调用功能，这个集群就是高可用。

## 注册中心高可用

目前结构只有一个注册中心。没法形成高可用。

**原理**：将多个（这里举列2个）注册中心，相互作为客户端进行注册，相互抓取注册信息同步数据。 搭建2个注册中心的高可用结构 。

|  |
| --- |
|  |

Server2 启动时会到指定的地址(Server1)去注册

|  |
| --- |
| client:  service-url:  defaultZone: http://127.0.0.1:8001/eureka |

Server1启动时会到指定的地址(Server2)去注册

|  |
| --- |
| client:  service-url:  defaultZone: http://127.0.0.1:8002/eureka |

这样搭建的高可用注册中心，一定会在启动时报个错误（因为总有一个server先启动，另一个Server未启动，先启动会注册失败）。

|  |
| --- |
| Cannot excute request on any know server |

# 遇到的问题

* SpringBoot版本与SpringCloud版本有对应关系，如果不一致会出现很多问题。

查询版本对应关系：

<https://start.spring.io/actuator/info>

<https://spring.io/projects/spring-cloud>

* 不同版本的SpringCloud 中的组件导入方式不一样

例如：SpringCloud Hoxton.SR10版本

|  |
| --- |
| *<!--Eureka server--> <*dependency*>  <*groupId*>*org.springframework.cloud*</*groupId*>  <*artifactId*>*spring-cloud-starter-netflix-eureka-server*</*artifactId*> </*dependency*>* |

* [Eureka报错: Connect to localhost:8761 timed out](https://www.cnblogs.com/sansheng93/p/11286654.html)

启动完Eureka后, 启动各项服务, 进行发现与注册, 各项服务均能看到正常的注册, 然而Eureka还是会一直不停的尝试连接8761端口服务, 这是为什么呢?

查阅相关文件, 了解到 8761 是Eureka的默认路径, 当Eureka的启动配置里没有对它的url地址指定时, 会默认寻找这个地址

解决：不管是客户端还是服务端都需要配置，服务端也具有客户端功能

|  |
| --- |
|  |