# 一 使用版本

Spring cloud : 最新的H版

Spring-boot: 使用2.2.X版本

# 二 组件

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 项目 |  |  |  |  |
| 服务注册中心 | Eureka(停更) | Zookeeper | Consul | Nacos(recommend) |
| 服务调用 | Ribbon（停更，但比较优秀还大量在用） | LoadBalancer（后面会用这个替换Ribbon） |  |  |
| 服务调用2 | Feign（停更） | OpenFeign（社区维护，推荐） |  |  |
| 服务降级 | Hystrix（停更，但国内还在使用，理念值得学习，老外不在用） | Resilience4j（老外搞的，国内用的少，国内推荐用下一个） | Sentinel （Spring cloud alibaba） |  |
| 服务网关 | Zuul（停更） | Zuul(这个家伙估计出不来了 。。) | Gateway（来自spring cloud ，值得推荐） |  |
| 服务配置 | Config（停更，也可以用） | Nacos（推荐，后者居上） |  |  |
| 服务总线 | Bus（停更） | Nacos（推荐，后者居上） |  |  |
|  |  |  |  |  |

# 三 服务注册

1. Eureka :

采用CS架构，server和client

默认开启自我保护模式，开启后如果出现由于网络等原因心跳无法检测到，不会马上踢掉服务，是一种高可用的实现方式。

2. Zookeeper

1.zookeeper包括两种节点：临时节点 持久节点

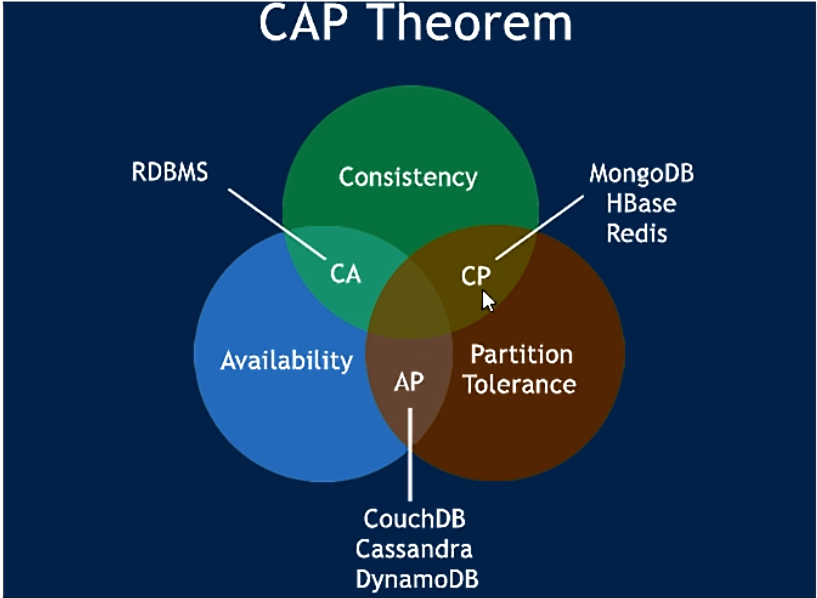
那么我们注册的是那种节点？ 答案是：临时节点

服务关闭后，不会马上踢掉服务，等一段时间会清除服务节点，重新注册到Zookeeper后生成新的节点。

3.Consul

使用go语言开发；有控制界面；来自

比较三个注册中心：

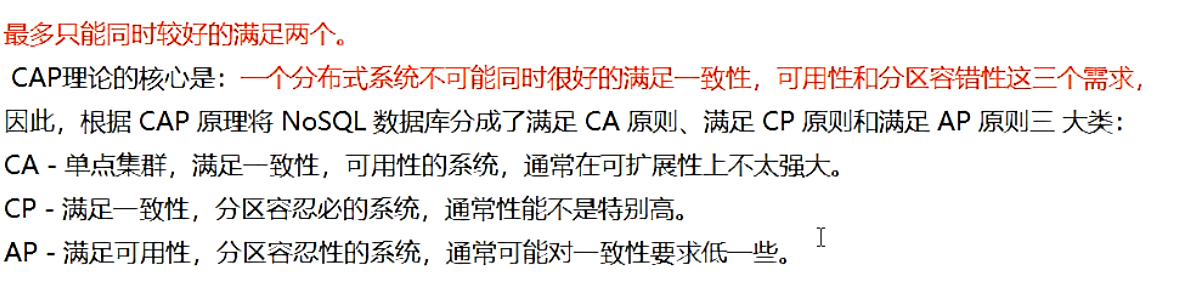


C：Consistency（强一致性）

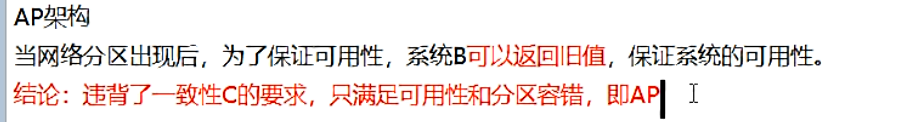
A：Availability（可用性）

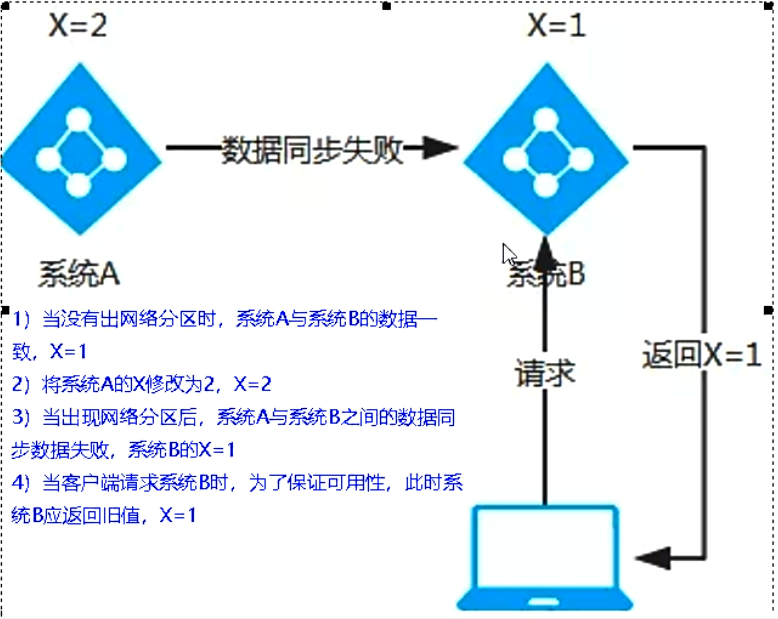
P：Partition Tolerance （分区容错性）

CAP理论关注粒度是数据，而不是整体系统设计的策略

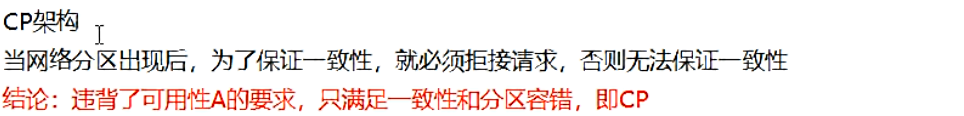


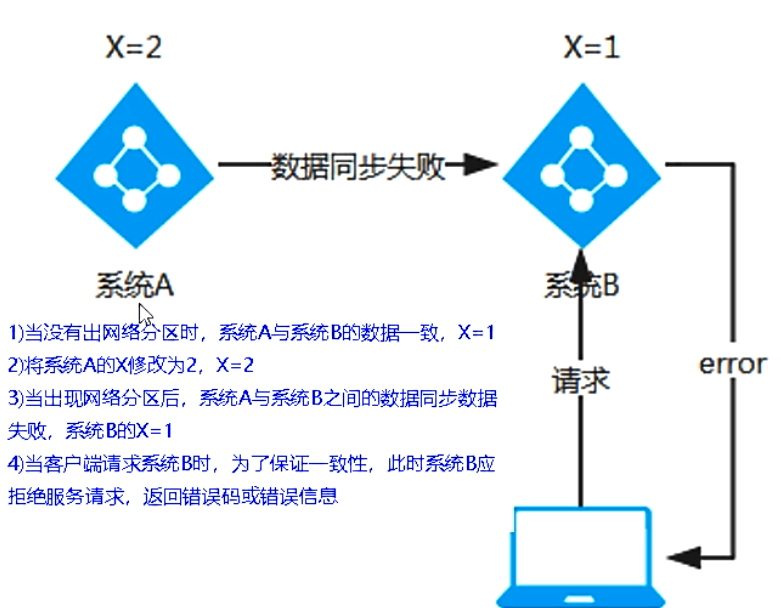
AP:





CP:





总结：

Eureka：(AP)主要保证A ，高可用。比如它的自我保护机制，保证高可用，好死不如赖活着的思想；偶尔宕机掉线了，一时半会收不到Eureka不会马上删除掉；

Consul：(CP)主要保证C，数据一致性

Zookeeper：(CP)主要保证C，数据一致性（同Consul），就比如Zookeeper创建的是临时节点而不是永久节点，心跳检测失败后马上剔除。

# 四. 服务调用Ribbon

1.Ribbon

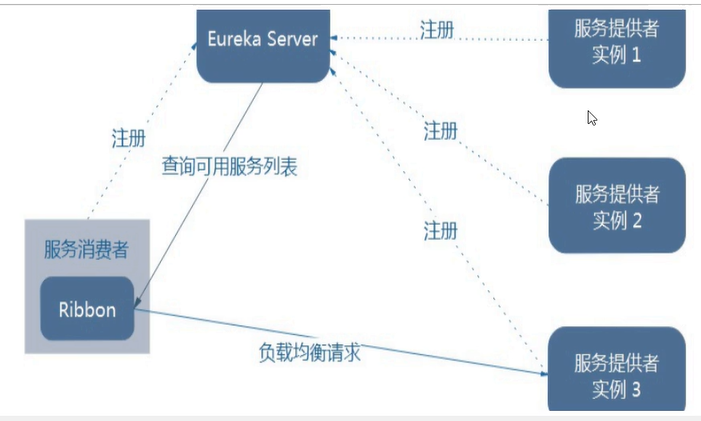
1.负载均衡（LB）

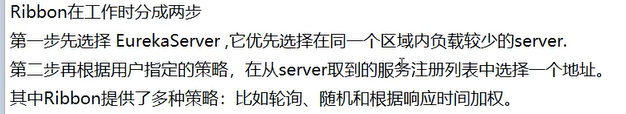
包括集中式LB：将用户请求平摊到多台服务器上，常见有Nginx、LVS、F5等

进程内LB（inter-process）：将LB逻辑集成到消费方，消费方从服务注册中心获知哪些地址可用，然后自己再从中选中一个合适的服务器。

Ribbon就属于进程内LB，它只有一个类库，集成于消费方进程，消费方通过它获取到服务提供方的地址。

架构：

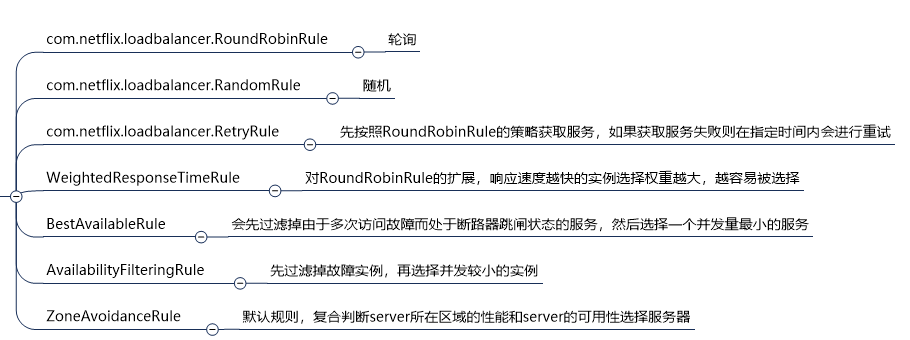




一句话：客户端负载均衡 + RestTemplate调用

2.Ribbon关键组件：Irule接口——根据特定算法从服务列表中选取一个要访问的服务；

负载均衡的方式有几种？



轮询是默认的，可以自己更改路由算法，如何改？

1. 配置细节



1. @Configuration  
   *public class* MySelfRule {  
     
    @Bean  
    *public IRule* myRule(){  
    *return new* RandomRule();*//定义为随机* }  
   }
2. 主启动类增加注解：

@EnableEurekaClient  
@SpringBootApplication  
@RibbonClient(name = "CLOUD-PAYMENT-SERVICE",configuration = MySelfRule.*class*)  
*public class* OrderMain80 {  
 *public static void* main(String[] args) {  
 SpringApplication.*run*(OrderMain80.*class*,args);  
 }  
  
}

3．Ribbon负载均衡算法

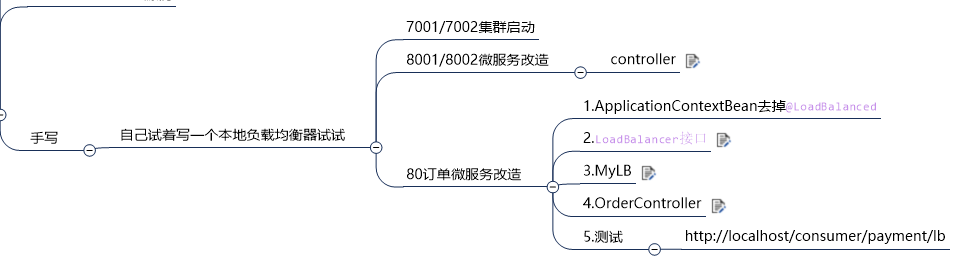
1.原理



2.查看源码：RoundRobinRule轮询

就是采用取模的方式（自旋锁？）获取服务列表的下标，再返回一个服务。

4.手写一个负载均衡算法



代码参见：cloud-consumer-order80

# 五．服务调用OpenFeign

1.OpenFeign 是什么？

Feign是一个声明式WebService客户端；使用Feign能让编写Web Service客户端更简单；

使用方法：定义一个接口 然后在接口上添加一个注解；

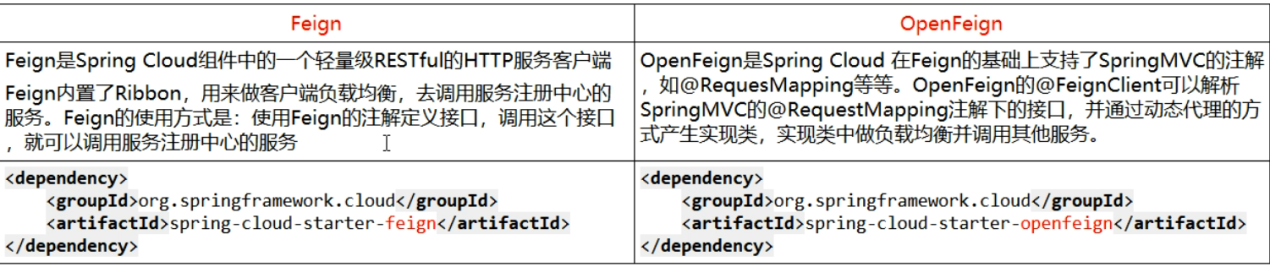
Feign 包括了Ribbon，所以就是对Ribbon的封装，不用去写resttimeplate请求提供方，而是直接面向接口编程；

2.OpenFeign能干什么？

Feign旨在使编写Java http客户端变得更容易；

ribbon+RestTemplate能解决远程方法调用，但是这种方式不够优雅，不够工业化，而Feign封装了ribbon+Resttemplate的实现细节，直接声明一个接口，实现了真正的面向接口编程，代码优雅且易使用；

3.feign和openfeign区别：



4.如何使用：

1. 微服务调用接口+@FeignClient

2. 主启动类：@EnableFeignClients

3. 详细参见：cloud-cosumerfeign-order80

5.penFeign超时控制

什么时候需要配置超时设置？

当客服端调用一个耗时长的服务时，或者调用远程服务包连接超时需要配置；（feign默认只等待一秒）

因为feign内置了ribbon，所以超时控制可以直接设置ribbon的超时配置：

ribbon:  
 ReadTimeout: 5000  
 ConnectTimeout: 5000

或者直接配置feign也可以，官方文档找到的：

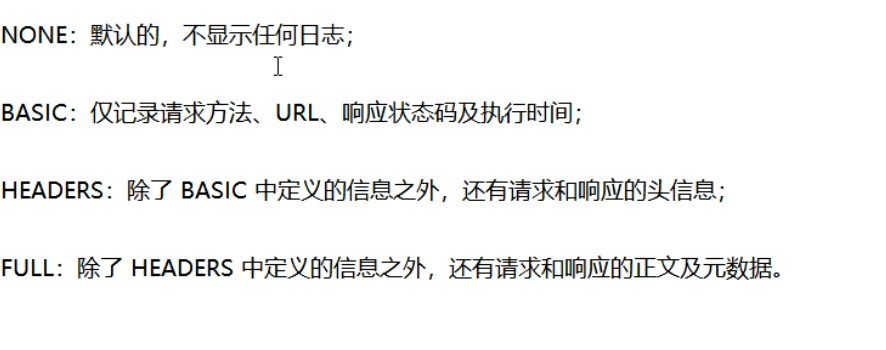
#feign:  
# client:  
# config:  
# default:  
# connectTimeout: 5000  
# readTimeout: 5000

6. OpenFeign日志打印功能：

Feign提供了日志打印功能，我们可调整日志级别，从而了解Feign中http请求的细节，说白了就是对Feign接口的调用情况进行监控和输出；

如何使用：

1. 日志级别：



1. 配置日志bean

@Configuration  
*public class* FeignConfig {  
  
 @Bean  
 Logger.Level feignLoggerLevel(){  
 *return* Logger.Level.*FULL*;  
 }  
}

1. YML文件里需要开启日志的Feign客户端

logging:  
 level:  
 com.atguigu.springcloud.service.PaymentFeignService: debug