# springcloud

1. 技术架构演变
   1. 单一应用架构
      1. 全部功能都在一个项目中，打一个war包
   2. 垂直应用架构
      1. 按照业务进行分割，形成一个个小的单体项目，每个业务模块就是一个war
   3. 分布式服务架构
      1. 将核心业务抽取出来
      2. 缺少统一管理资源调度的框架，也就是说应用调用服务会把ip写死，如果这个服务挂了就不能用了
   4. 面向服务(SOA)架构
      1. **流动计算架构**
         1. 把dubbo作为调度的工具（RPC协议），ZK作为注册中心
      2. **微服务架构**
         1. 每个服务完全独立和解耦了，HTTP协议传输（Spring cloud）
2. RPC和HTTP
   1. RPC远程过程调用：基于Socket，速度快，Dubbo
   2. HTTP网络传输协议：基于TCP/IP，传输速度就比较慢，但是对技术没有限定
3. Eureka的原理
   1. 1. 首先它有两个角色：服务端和客户端，服务端就是一个注册中心。而客户端就是那些消费者和提供者都是微服务
   2. 2. 首先客户端会将自己的服务名+ip+port等信息注册到注册中心去
   3. 3. 而消费端会每隔30秒去注册中心拉取这些服务列表，然后存到本地缓存中
   4. 4. 从本地缓存中取出，进行一个动态的调用
   5. 期间客户端会每隔30s向注册中心发个消息，表示自己还存在，这就是心跳检测
   6. 注册中心会每隔60s就检测一次，客户端有没有发这个心跳过来，如果没有，那么倒计时90s，一旦超时就将这个服务挂掉（前提是关闭了自自我保护机制，默认是开启的）这就是失效剔除机制
   7. 自我保护机制开启的时候，当客户端十五分钟之内，发现这个续约失败率低于百分之85的话是不会被剔除的
4. Ribbon
   1. 解决在集群服务中，做负载均衡的问题
   2. 负载均衡算法
      1. 轮询（默认）
         1. 一人一下的来
      2. 随机
      3. 重试
         1. 先按照轮询的策略获取服务，如果获取服务失败就会在指定的时间内进行重试，获取可用的服务
      4. 加权
         1. 会根据平均响应时间去计算所有的权重，响应越快那个服务权重就越大，刚刚启动的时候统计信息不足以判断，则会实现轮询的策略，后期转换为自身规则
      5. 7种（有个IRule接口）
   3. 负载均衡的流程
      1. 他有个LoadBalancedIntercetor拦截器，会把请求拦截下来
      2. 调用一个intercept方法里面会把获取到端口号，服务名等信息，然后传过到一个getLoadBalanced（）方法，他会去注册中心获取服务列表，然后根据自己配置或者定义的负载均衡算法决定哪个服务
   4. 轮询算法
      1. 接口请求次数 % 集群机器数 = 实际调用的下标 （求模取余）
5. Hystrix熔断降级
   1. 是一个分布式容错框架，防止级联失效和雪崩效应
   2. 什么是雪崩效应？
      1. 一个请求可能需要多个微服务接口才能实现
      2. 如果其中某个服务出现异常或者阻塞了，线程又不能释放，于是请求堆积越来越多
      3. 单服务器支持的线程和并发数有限，如果一直阻塞，会导致服务器资源耗尽，导致其他有关联的服务都不可用，从而形成雪崩效应
      4. 解决雪崩：触发机制（熔断器）有三种模式
         1. 服务降级-fallback（备用模式）
         2. 线程隔离-壁仓模式
         3. 限流-信号量模式
   3. 熔断原理分析
      1. 关闭（所有请求都正常访问）
      2. 打开（所有请求都会降级），豪猪会对请求情况进行计数，当一定时间失败请求到达一个百分比，就会触发熔断，默认比例是50%，次数不少于20次
      3. 半开（打开状态不是永久的，打开一会后会进入休眠时间（5s），休眠时间一过就会进入半开状态）熔断器会判断下一次请求的返回情况，如果成功，熔断器就切会关闭状态，失败则继续回到打开状态
6. Feign
   1. 代替了RestTemplate发送Rest请求，以接口调用的方式实现远程调用，增强了远程调用的功能（负载均衡，熔断，日志启用等）
7. GetWay
   1. 提供一种统一有效的API路由管理方式
   2. 可以在网关中实现微服务鉴权，安全控制，请求监控