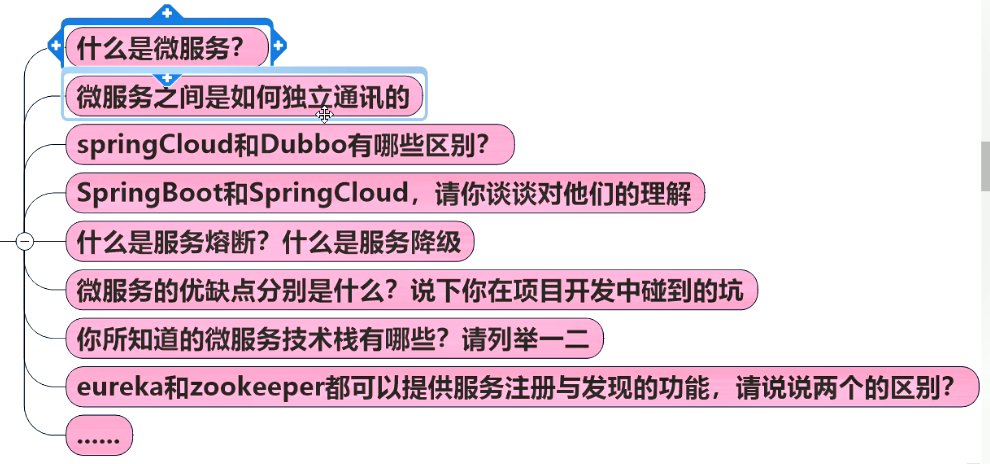
# 微服务概述与springcloud



## 微服务

强调的是服务的大小，它关注的是某一个点，是具体解决某一个问题/提供落地对应服务的一个服务应用,

狭意的看,可以看作Eclipse里面的一个个微服务工程/或者Module

微服务化的核心就是将传统的一站式应用，根据业务拆分成一个一个的服务，彻底地去耦合,每一个微服务提供单个业务功能的服务，一个服务做一件事

## 微服务架构

微服务架构是⼀种架构模式，它提倡将单⼀应⽤程序划分成⼀组⼩的服务，服务之间互相协调、互相配合，为⽤户提供最终价值。每个服务运⾏在其独⽴的进程中，服务与服务间采⽤轻量级的通信机制互相协作（通常是基于HTTP协议的RESTful API）。每个服务都围绕着具体业务进⾏构建，并且能够被独⽴的部署到⽣产环境、类⽣产环境等。另外，应当尽量避免统⼀的、集中式的服务管理机制，对具体的⼀个服务⽽⾔，应根据业务上下⽂，选择合适的语⾔、⼯具对其进⾏构建。

## springcloud的5大神兽

服务发现—NetFlix Eureka

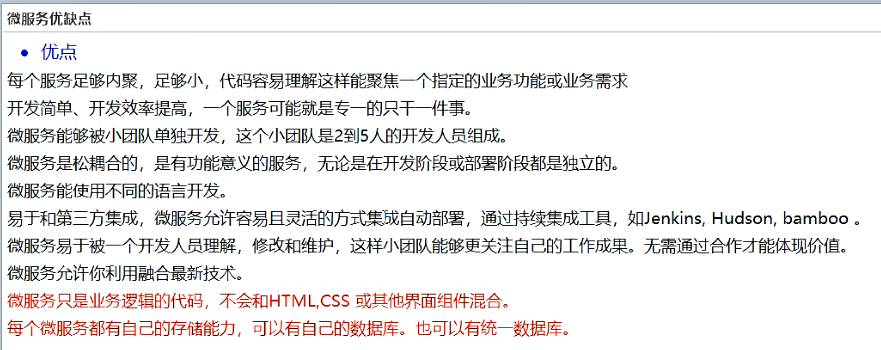
客户端负载均衡—Netflix Ribbon

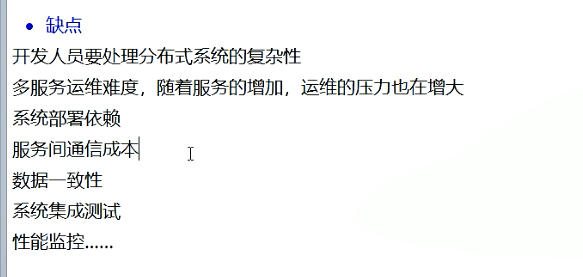
断路器—Netflix hystrix

服务网关—Netflix zuul

分布式配置—Spring cloud config

## 微服务的优缺点





## 微服务的技术栈



## springboot和springcloud是什么关系

springboot是微观的，（相当于医院的一个一个科室）

springcloud是宏观的（相当于对外的一个医院）

springcloud依赖于springboot.

SpringBoot专注于快速、方便的开发单个微服务个体，SpringCloud关注全局的服务治理框架。

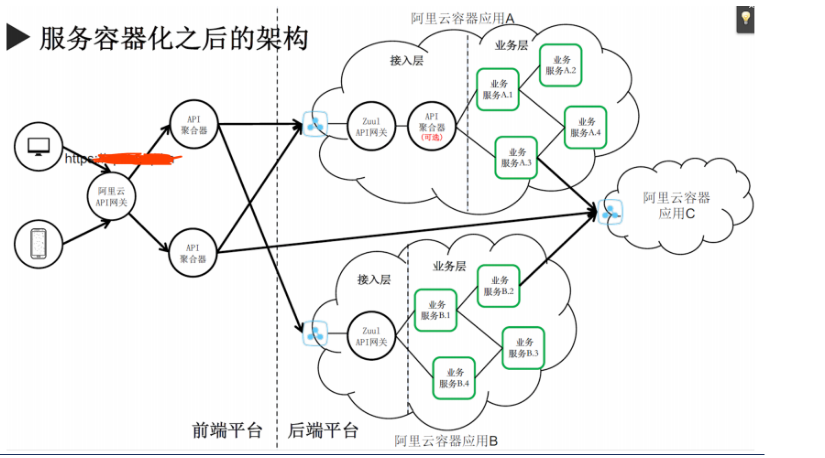
## dubbo和springcloud的区别



### 总结

Dubbo的定位始终是一款RPC框架，而spring cloud的目标是微服务架构下的一站式解决方案。再面临微服务基础框架选型时Dubbot与Springcloud只能二选一

## springcloud在阿里云的应用



# Rest微服务构建工程

## 消费端的RestTemplate

RestTemplate提供了多种便捷访问远程Http服务的方法，

是一种简单便捷的访问restful服务模板类，是Spring提供的用于访问Rest服务的客户端模板工具集

## Eureka是什么

Eureka 采用了 C-S 的设计架构。Eureka Server（物业公司） 作为服务注册功能的服务器，它是服务注册中心。而系统中的其他微服务，使用 Eureka 的客户端连接到 Eureka Server并维持心跳连接（交物业费相当于维持心跳连接）。这样系统的维护人员就可以通过 Eureka Server 来监控系统中各个微服务是否正常运行。SpringCloud 的一些其他模块（比如Zuul）就可以通过 Eureka Server 来发现系统中的其他微服务，并执行相关的逻辑。

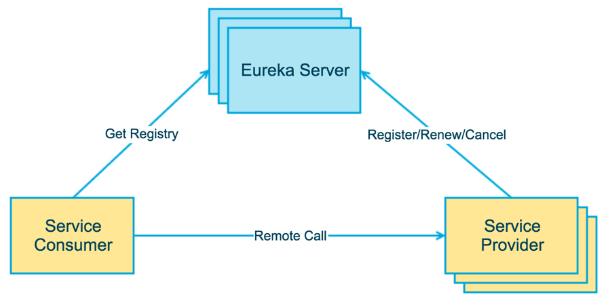
Eureka包含两个组件：Eureka Server和Eureka Client

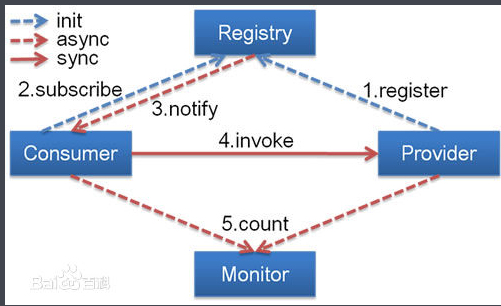
Eureka Server提供服务注册服务

各个节点启动后，会在EurekaServer中进行注册，这样EurekaServer中的服务注册表中将会存储所有可用服务节点的信息，服务节点的信息可以在界面中直观的看到

EurekaClient是一个Java客户端，用于简化Eureka Server的交互，客户端同时也具备一个内置的、使用轮询(round-robin)负载算法的负载均衡器。在应用启动后，将会向Eureka Server发送心跳(默认周期为30秒)。如果Eureka Server在多个心跳周期内没有接收到某个节点的心跳，EurekaServer将会从服务注册表中把这个服务节点移除（默认90秒）

## Eureka与dubbo的区别





## eureka的自我保护

一句话：某时刻某一个微服务不可用了，eureka不会立刻清理，依旧会对该微服务的信息进行保存

## eureka的服务发现

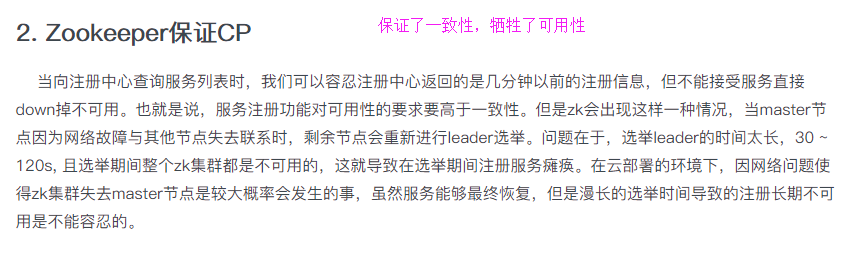
向外暴露你提供了那些微服务

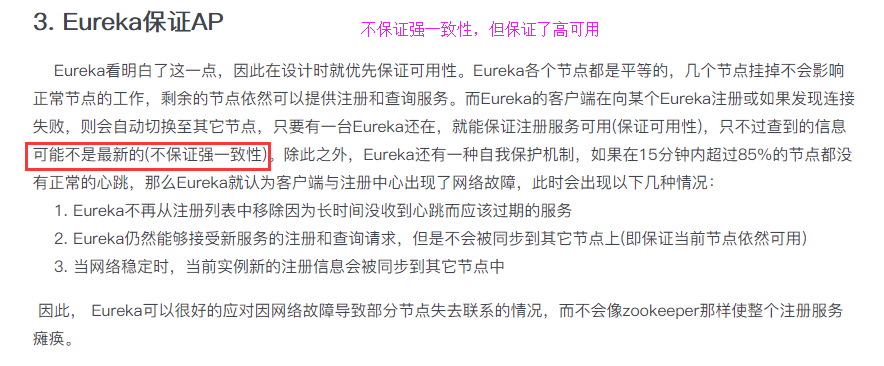
对于注册进Eureka里面的服务，可以通过服务发现来获得该服务的信息

## eureka与zookeeper的区别

Eureka:AP

zookeeper:CP



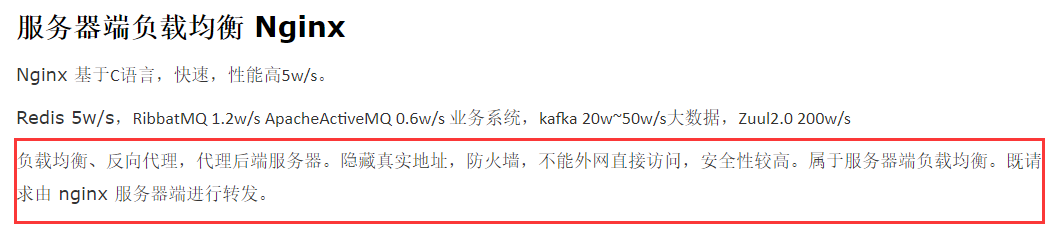


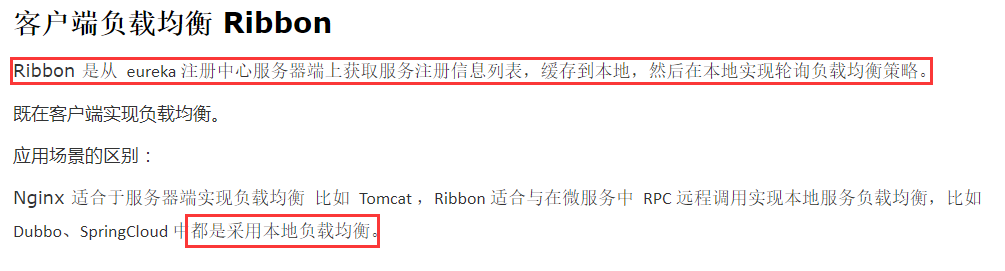
# Ribbon负载均衡

Spring Cloud Ribbon是基于Netflix Ribbon实现的一套*客户端*负载均衡的工具(比如说去麦当劳买汉堡，有3个窗口，1号窗口8个人，2号窗口6个人，3号窗口5个人，我肯定去3号窗口)。

简单的说，Ribbon是Netflix发布的开源项目，主要功能是提供客户端的软件负载均衡算法，将Netflix的中间层服务连接在一起。Ribbon客户端组件提供一系列完善的配置项如连接超时，重试等。简单的说，就是在配置文件中列出Load Balancer（简称LB）后面所有的机器，Ribbon会自动的帮助你基于某种规则（如简单轮询，随机连接等）去连接这些机器。我们也很容易使用Ribbon实现自定义的负载均衡算法。

## nginx,ribbon,feign的 区别







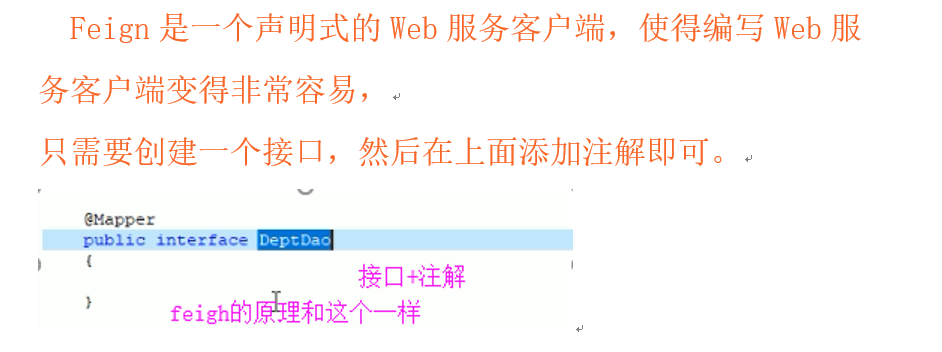
有时候有的项目会2个技术一起用在该项目中是因为

feign是远程调用的,ribbon是做负载均衡的,

## 负载均衡算法

**轮询，随机，优化的轮询，带权重的轮询，超时重试的轮询，并发最小的访问，判断选择区域内最好的。**

## Feign的原理







# Hystrix断路器

## 服务雪崩

多个微服务之间调用的时候，假设微服务A调用微服务B和微服务C，微服务B和微服务C又调用其它的微服务，这就是所谓的“扇出”。如果扇出的链路上某个微服务的调用响应时间过长或者不可用，对微服务A的调用就会占用越来越多的系统资源，进而引起系统崩溃，所谓的“雪崩效应”.

对于高流量的应用来说，单一的后端依赖可能会导致所有服务器上的所有资源都在几秒钟内饱和。比失败更糟糕的是，这些应用程序还可能导致服务之间的延迟增加，备份队列，线程和其他系统资源紧张，导致整个系统发生更多的级联故障。这些都表示需要对故障和延迟进行隔离和管理，以便单个依赖关系的失败，不能取消整个应用程序或系统。

## hystrix是什么

Hystrix是一个用于处理分布式系统的延迟和容错的开源库，在分布式系统里，许多依赖不可避免的会调用失败，比如超时、异常等，Hystrix能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故障，以提高分布式系统的弹性。

“断路器”本身是一种开关装置，当某个服务单元发生故障之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个符合预期的、可处理的备选响应（FallBack），而不是长时间的等待或者抛出调用方无法处理的异常，这样就保证了服务调用方的线程不会被长时间、不必要地占用，从而避免了故障在分布式系统中的蔓延，乃至雪崩。

## 服务熔断

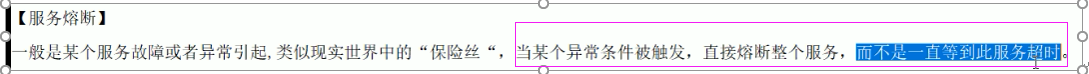
熔断机制是应对雪崩效应的一种微服务链路保护机制。

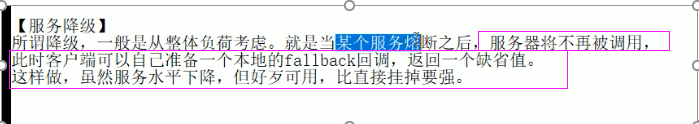
当扇出链路的某个微服务不可用或者响应时间太长时，会进行服务的降级，进而熔断该节点微服务的调用，快速返回"错误"的响应信息。当检测到该节点微服务调用响应正常后恢复调用链路。在SpringCloud框架里熔断机制通过Hystrix实现。Hystrix会监控微服务间调用的状况，当失败的调用到一定阈值，缺省是5秒内20次调用失败就会启动熔断机制。熔断机制的注解是@HystrixCommand。

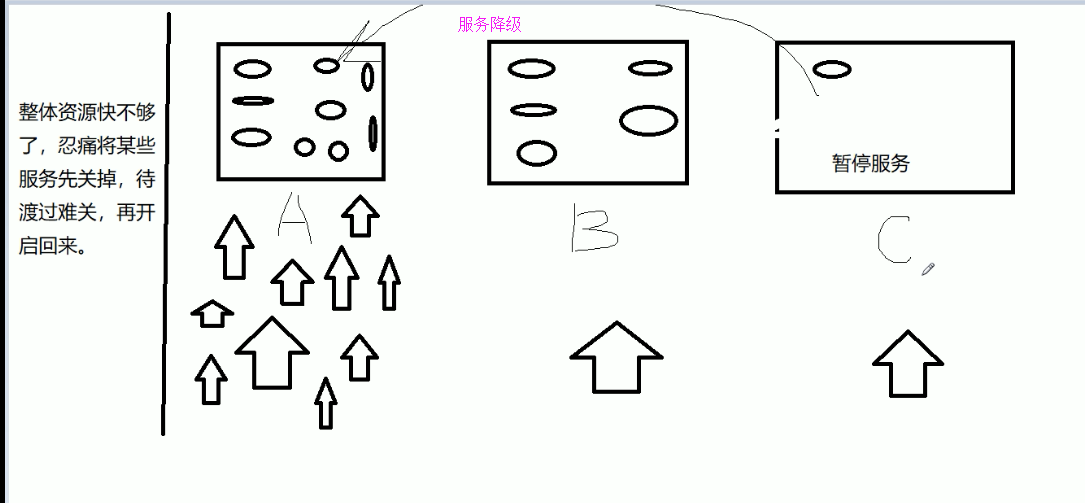
## 服务降级（服务降级处理是在客户端实现完成的，与服务端没有关系）

整体资源快不够了，忍痛将某些服务先关掉，待渡过难关，再开启回来。

服务降级处理是在客户端实现完成的，与服务端没有关系（比如说银行柜台，展厅服务，其他顾客会去找一个开着的柜台）







## 服务限流

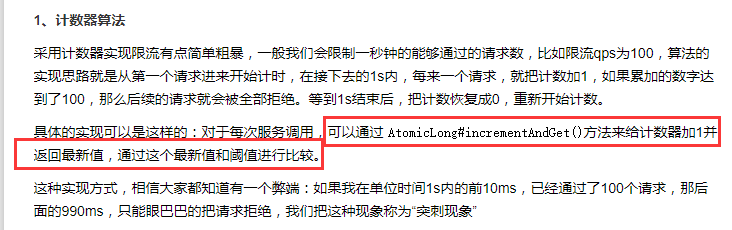
**在开发高并发系统时有三把利器用来保护系统：缓存、降级和限流。缓存的目的是提升系统访问速度和增大系统能处理的容量，可谓是抗高并发流量的银弹；而降级是当服务出问题或者影响到核心流程的性能则需要暂时屏蔽掉，待高峰或者问题解决后再打开；而有些场景并不能用缓存和降级来解决，比如稀缺资源（秒杀、抢购）、写服务（如评论、下单）、频繁的复杂查询（评论的最后几页），因此需有一种手段来限制这些场景的并发/请求量，即限流。**

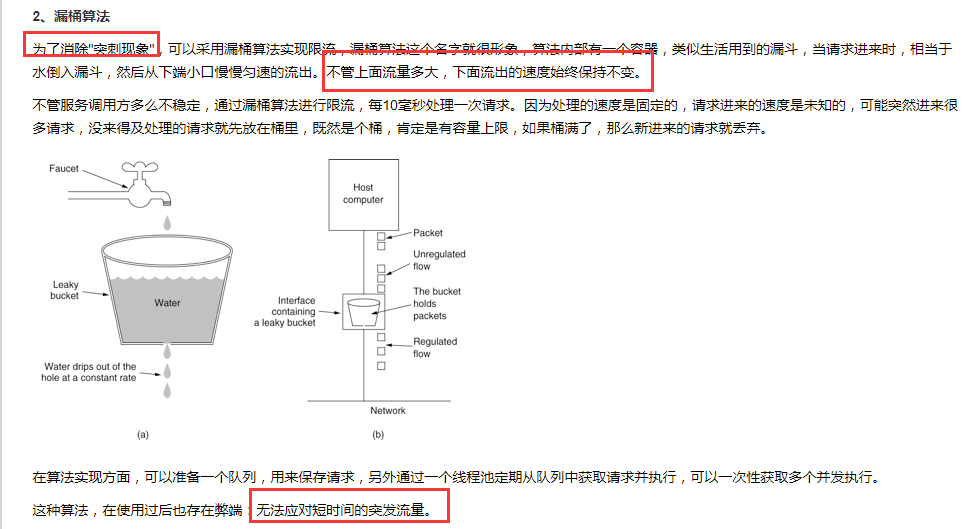
**限流的目的是通过对并发访问/请求进行限速或者一个时间窗口内的的请求进行限速来保护系统，一旦达到限制速率则可以拒绝服务（定向到错误页或告知资源没有了）、排队或等待（比如秒杀、评论、下单）、降级（返回兜底数据或默认数据，如商品详情页库存默认有货）。**

**一般开发高并发系统常见的限流有：限制总并发数（比如数据库连接池、线程池）、限制瞬时并发数（如nginx的limit\_conn模块，用来限制瞬时并发连接数）、限制时间窗口内的平均速率（如Guava的RateLimiter、nginx的limit\_req模块，限制每秒的平均速率）；其他还有如限制远程接口调用速率、限制MQ的消费速率。另外还可以根据网络连接数、网络流量、CPU或内存负载等来限流**

### 限流算法

令牌桶、漏桶，计数器。







## hystrix可以实时监控

除了隔离依赖服务的调用以外，Hystrix还提供了准实时的调用监控（Hystrix Dashboard），Hystrix会持续地记录所有通过Hystrix发起的请求的执行信息，并以统计报表和图形的形式展示给用户，包括每秒执行多少请求多少成功，多少失败等。

# 路由网关

Zuul包含了对请求的路由和过滤两个最主要的功能：

其中路由功能负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础而过滤器功能则负责对请求的处理过程进行干预，是实现请求校验、服务聚合等功能的基础.

Zuul和Eureka进行整合，将Zuul自身注册为Eureka服务治理下的应用，同时从Eureka中获得其他微服务的消息，也即以后的访问微服务都是通过Zuul跳转后获得。

    注意：Zuul服务最终还是会注册进Eureka

提供=代理+路由+过滤三大功能

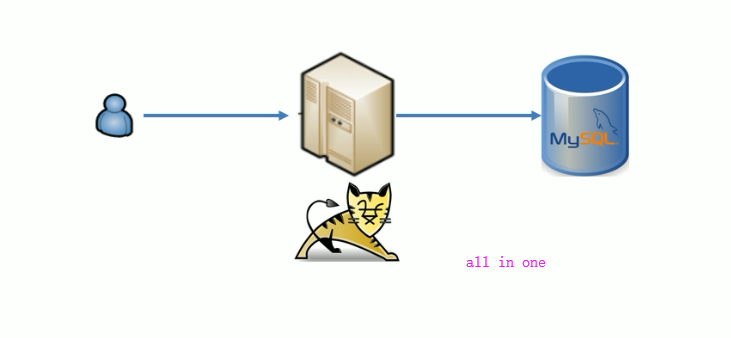
# springcloudconfig分布式配置中心



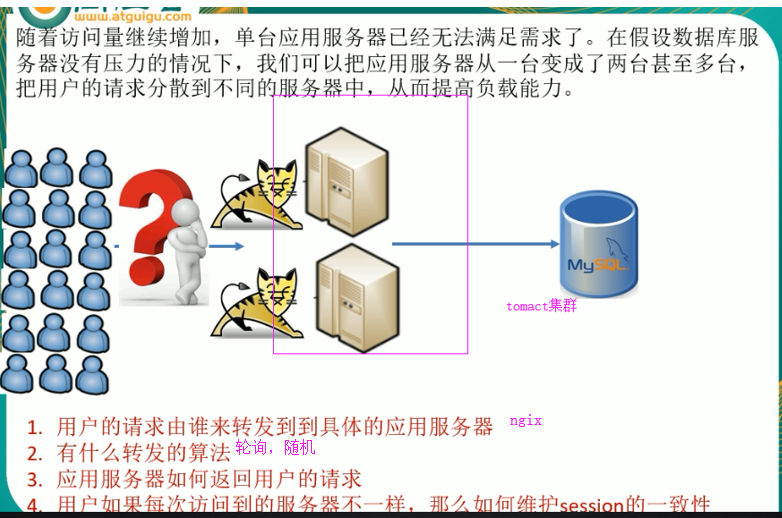
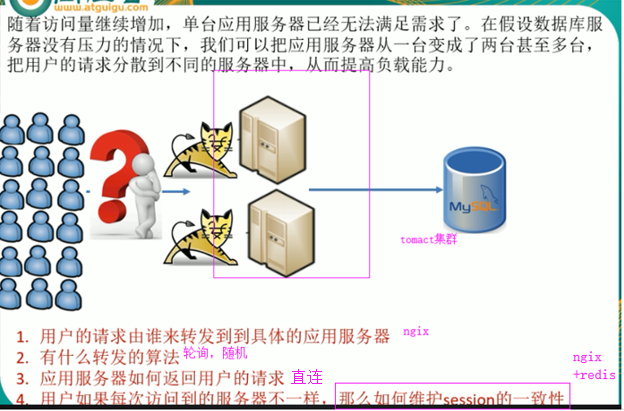
# 51SpringCloud重要必看，第一季架构技术总结和第二季展望

<https://blog.csdn.net/qqyb2000/article/details/80419592（web>服务端架构演变）

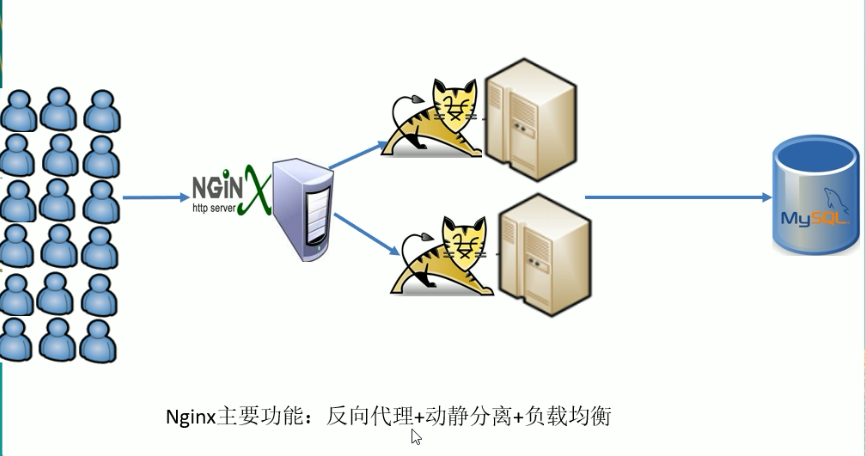
## 阶段一：单机集中构建网站



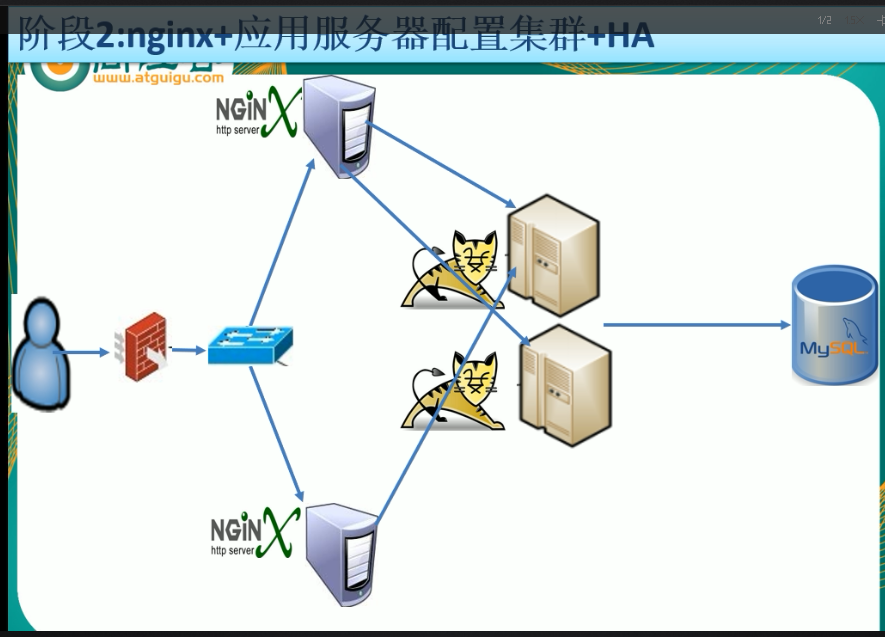
## 阶段二：应用服务配置集群

## 阶段二：ngix+应用服务器配置集群



## 阶段二：nginx+应用服务器配置集群+HA（避免单点故障）

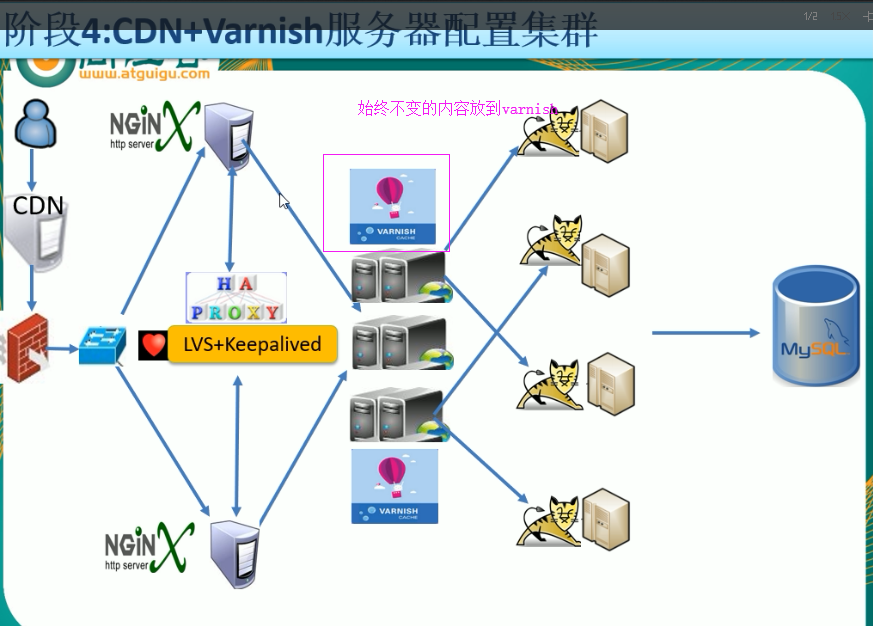


## 阶段三：负载均衡服务器配置集群



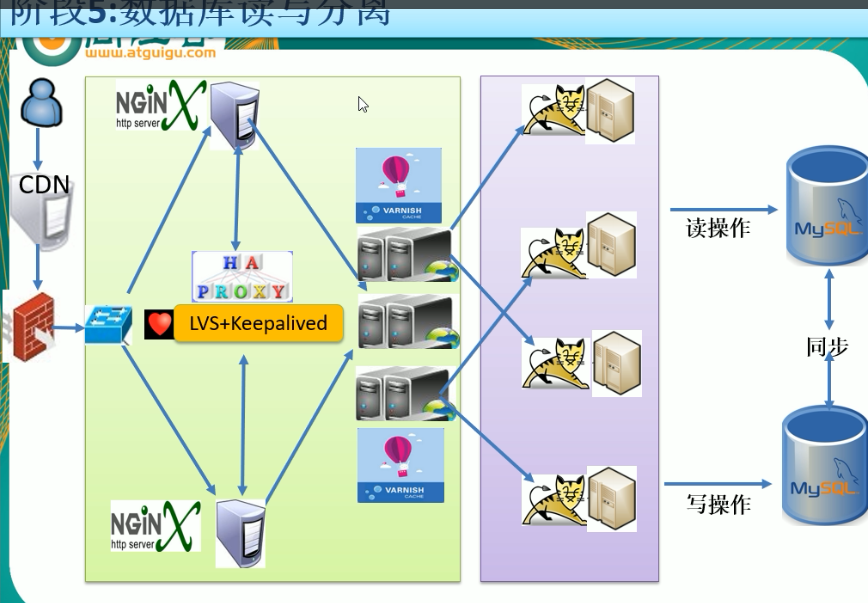
LVS（Linux Virtual Server）即Linux虚拟服务器，是由章文嵩博士主导的开源负载均衡项目，目前LVS已经被集成到Linux内核模块中。

## 阶段四：CDN+Varnish服务器配置集群（动静分离）

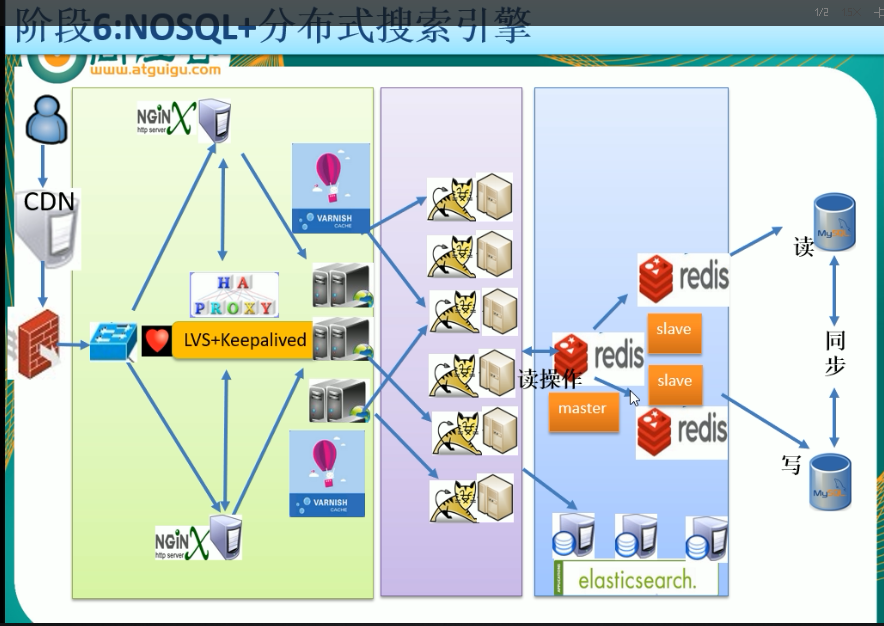


百年不变的内容放到varnish中（网站首页等） （varnish反向代理服务器的加速器） 有些内容不用去找tomact直接去找varnish获得

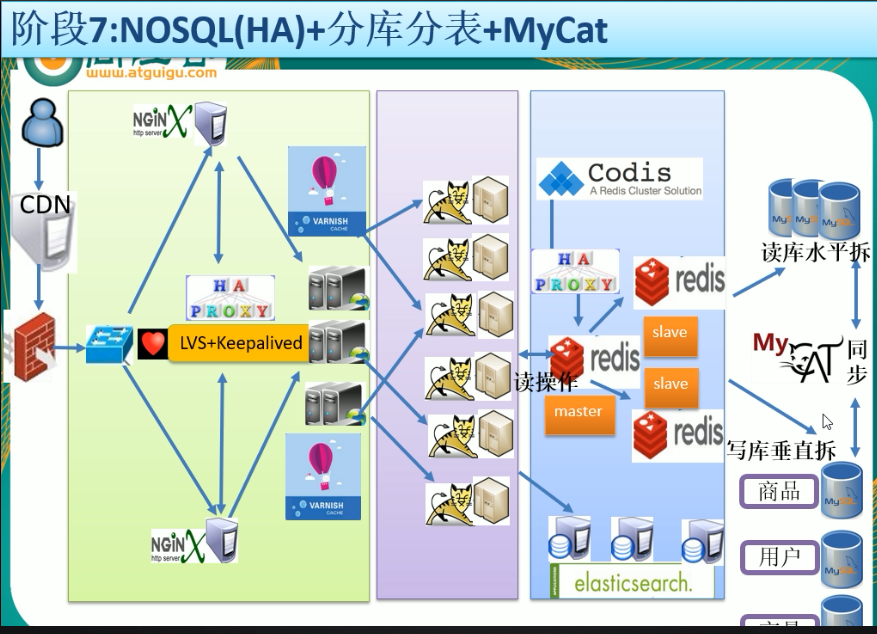
## 阶段五:数据库读写分离



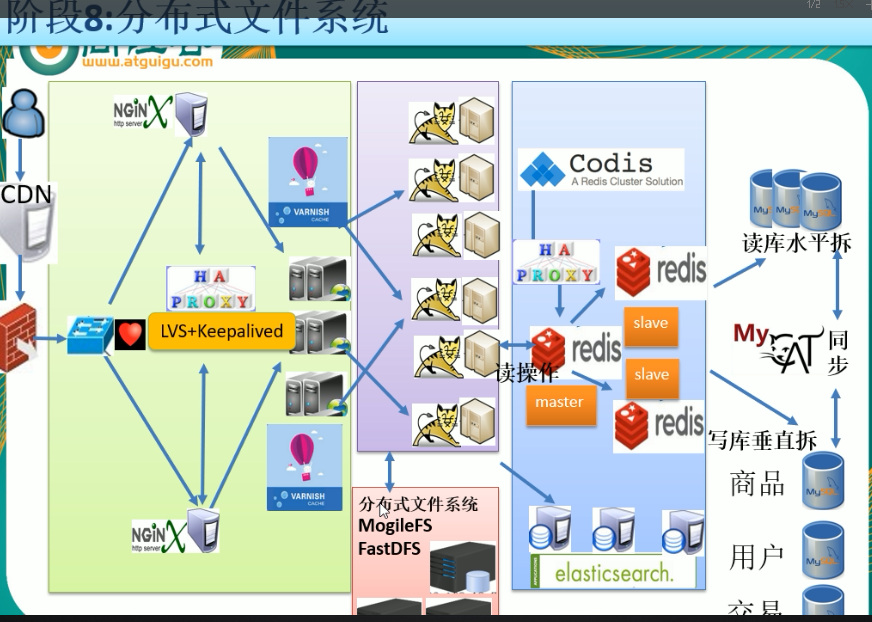
## 阶段6：NOSQL+分布式搜索引擎



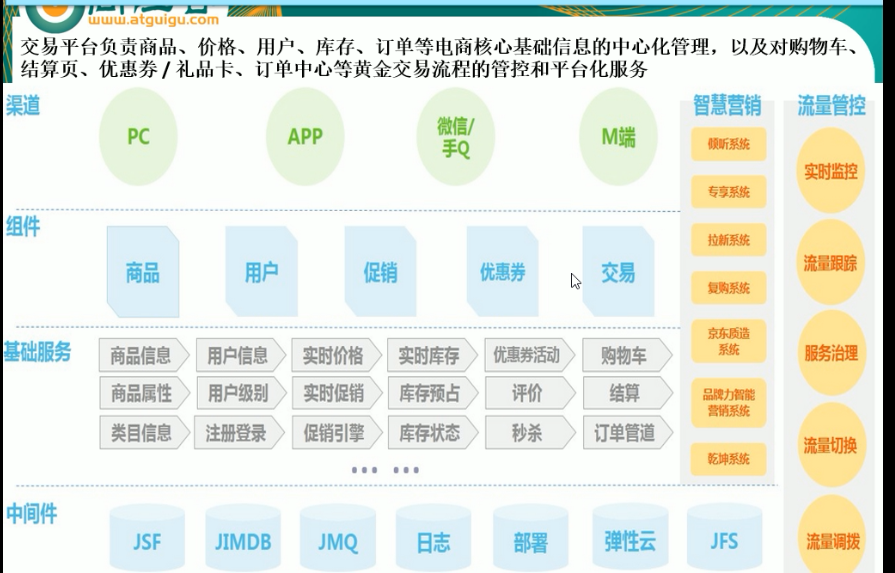
## 阶段七:NOSQL（HA）+分库分表+myCat



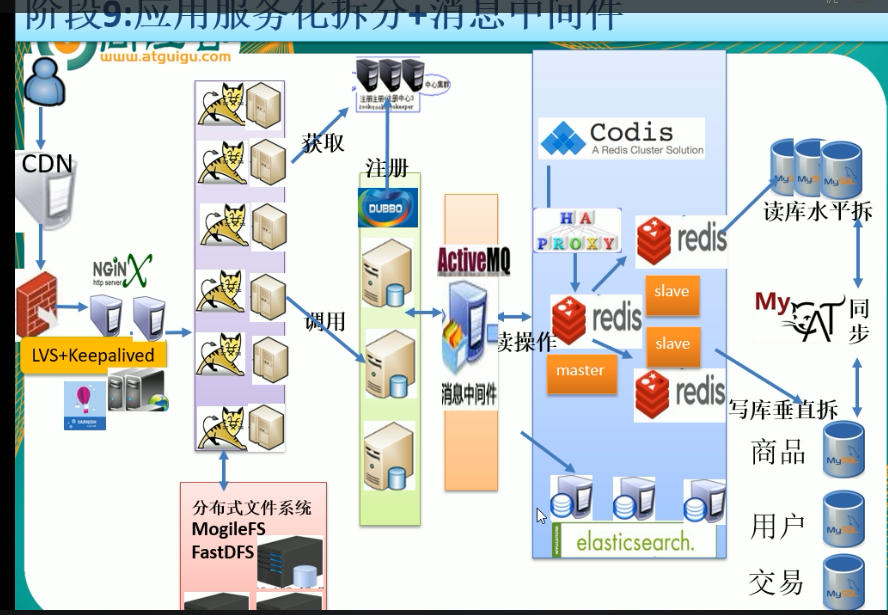
## 阶段八：分布式文件系统



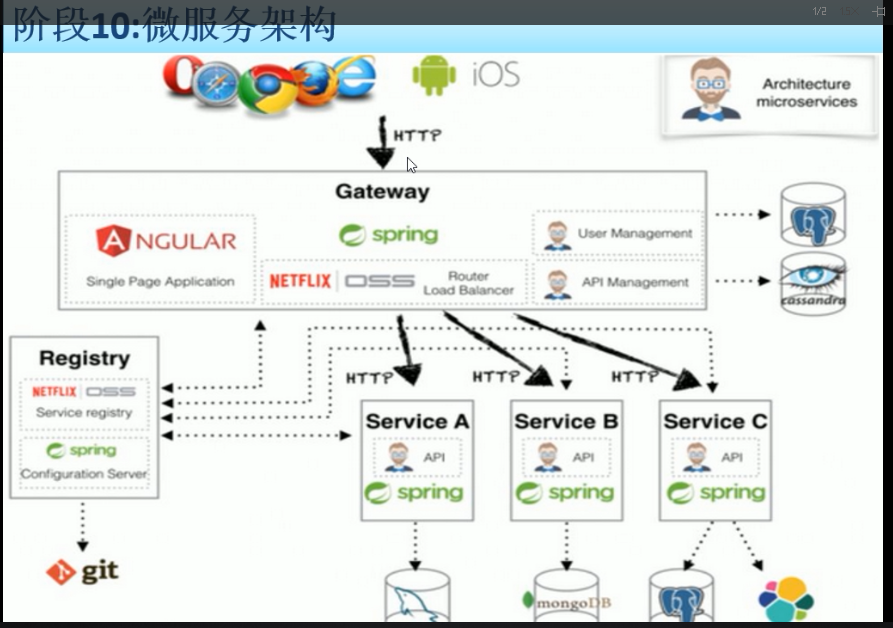
## 京东2017618交易平台目前的架构体系



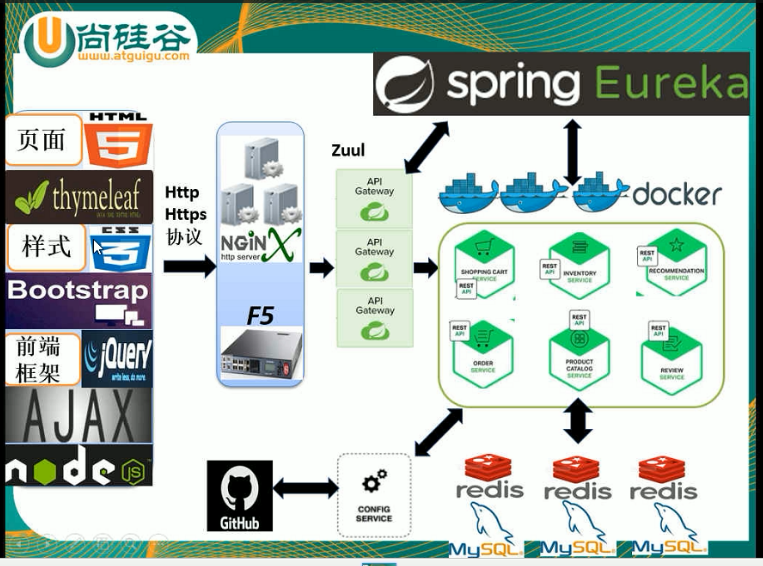
## 阶段九：应用服务化拆分+消息中间件

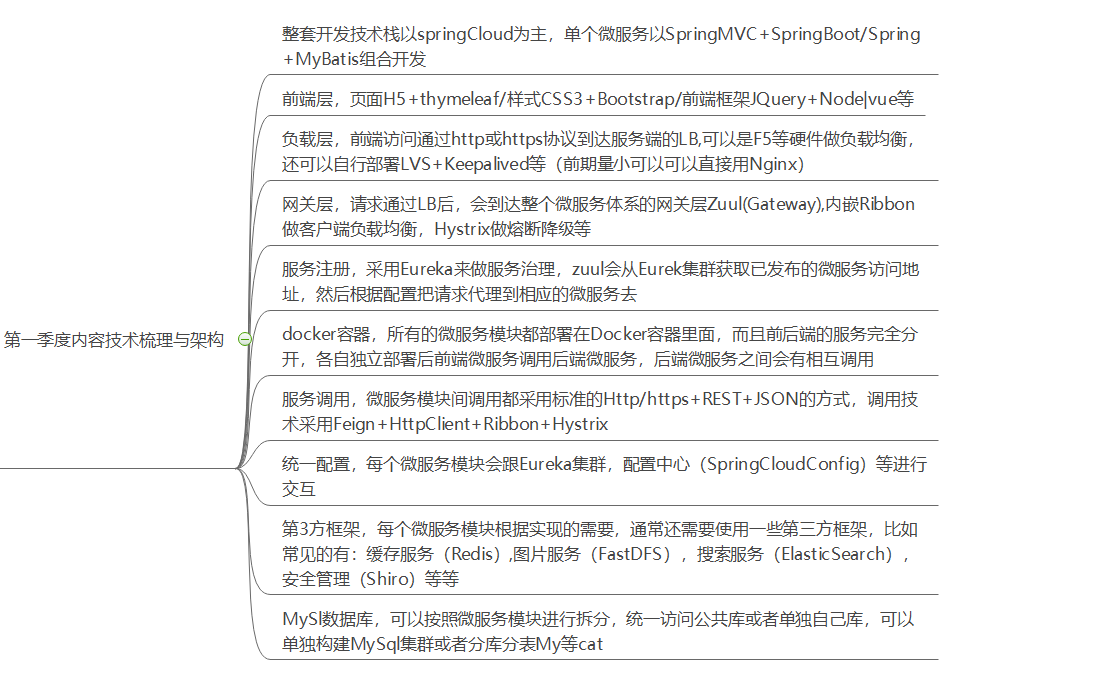


## 阶段十微服务架构



## 第一季度内容技术梳理与架构





## 第二季展望