



## 第二版：SpringCloud 20 道

### 目录

第二版：SpringCloud 20 道	1
1.什么是微服务	2
2. 微服务之间是如何独立通讯的	2
3. springcloud 和 dubbo 有哪些区别	3
4. springboot 和 springcloud 认识	4
5. 什么是服务熔断，什么是服务降级	4
服务熔断：	4
服务降级：	5
6. 微服务的优缺点	5
优点：	5
缺点：	6
7. 使用中碰到的坑	6
8. 列举微服务技术栈	6
9. eureka 和 zookeeper 都可以提供服务的注册与发现功能，他们的区别	7
Zookeeper 保证 CP	7
Eureka 保证 AP	7
10. eureka 服务注册与发现原理	8
11. dubbo 服务注册与发现原理	9
12. 限流	10
1、http 限流：我们使用nginx 的 limitzone 来完成：	10
2、dubbo 限流：dubbo 提供了多个和请求相关的 filter：ActiveLimitFilter	
ExecuteLimitFilter TPSLimiterFilter	10
1、 ActiveLimitFilter：	10
2、 ExecuteLimitFilter：	11
3、 TPSLimiterFilter：	11
3、springcloud 限流：	11
4、redis 计数器限流：	13
13. springcloud 核心组件及其作用，以及 springcloud工作原理：	13



14. eureka 的缺点: .....	14
15. eureka 缓存机制: .....	14
16. 熔断的原理, 以及如何恢复? .....	15
17. 服务雪崩? .....	16
18. 什么是 Hystrix 断路器? 我们需要它吗? .....	18
19. 多个消费者调用同一接口, eureka 默认的分配方式是什么? .....	19
20. 接口限流方法? .....	19

我们的网站: <https://tech.souyunku.com>

关注我们的公众号: **搜云库技术团队**, **回复以下关键字**

回复: **进群** 邀请您进「**技术架构分享群**」

回复: **内推** 即可进: 北京, 上海, 广州, 深圳, 杭州, 成都, 武汉, 南京, 郑州, 西安, 长沙「**程序员工作内推群**」

回复 **1024** 送 4000G 最新架构师视频

回复 **PPT** 即可无套路获取, 以下最新整理调优 PPT!

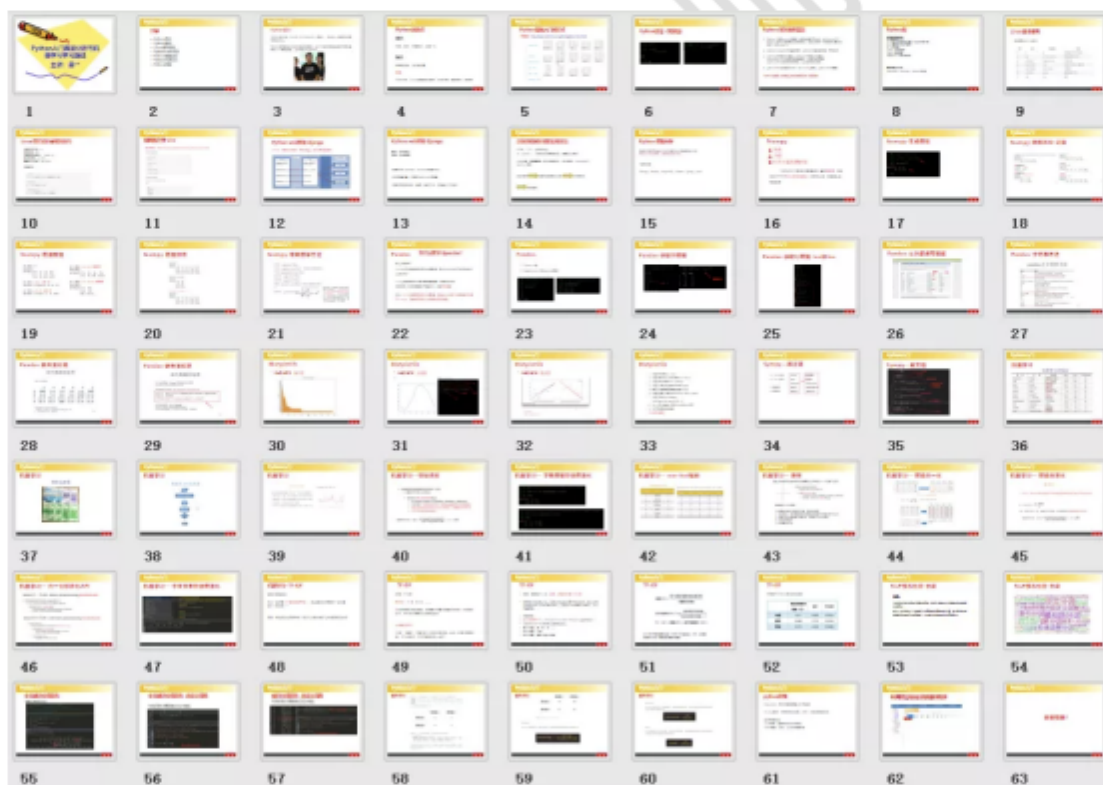
## 46 页《JVM 深度调优, 演讲 PPT》



## 53 页《Elasticsearch 调优演讲 PPT》



## 63 页《Python 数据分析入门 PPT》





微信扫一扫

<https://tech.souyunku.com>

技术、架构、资料、工作、内推  
专注于分享最有价值的互联网技术干货文章

## 1. 什么是微服务

- 1、微服务是一种架构风格，也是一种服务；
- 2、微服务的颗粒比较小，一个大型复杂软件应用由多个微服务组成，比如 Netflix 目前由 500 多的微服务组成；
- 3、它采用 UNIX 设计的哲学，每种服务只做一件事，是一种松耦合的能够被独立开发和部署的无状态化服务（独立扩展、升级和可替换）。

## 2. 微服务之间是如何独立通讯的

- 1、Dubbo 使用的是 RPC 通信，二进制传输，占用带宽小；
- 2、Spring Cloud 使用的是 HTTP RESTful 方式。



线程数	Dubbo	Spring Cloud
10线程	2.75	6.52
20线程	4.18	10.03
50线程	10.3	28.14
100线程	20.13	55.23
200线程	42	110.21

### 3. springcloud 和 dubbo 有哪些区别

1、Dubbo 具有调度、发现、监控、治理等功能，支持相当丰富的服务治理能力。Dubbo 架构下，注册中心对等集群，并会缓存服务列表已被数据库失效时继续提供发现功能，本身的服务发现结构有很强的可用性与健壮性，足够支持高访问量的网站。

2、虽然 Dubbo 支持短连接大数据量的服务提供模式，但绝大多数情况下都是使用长连接小数据量的模式提供服务使用的。所以，对于类似于电商等同步调用场景多并且能支撑搭建 Dubbo 这套比较复杂环境的成本的产品而言，Dubbo 确实是一个可以考虑的选择。但如果产品业务中由于后台业务逻辑复杂、时间长而导致异步逻辑比较多的话，可能 Dubbo 并不合适。同时，对于人手不足的初创产品而言，这么重的架构维护起来也不是很方便。

3、Spring Cloud 由众多子项目组成，如 Spring Cloud Config、Spring Cloud Netflix、Spring Cloud Consul 等，提供了搭建分布式系统及微服务常用的工具，如配置管理、服务发现、断路器、智能路由、微代理、控制总线、一次性 token、全局锁、选主、分布式会话和集群状态等，满足了构建微服务所需的所有解决方案。比如使用 Spring Cloud Config 可以实现统一配置中心，对配置进行统一管理；使用 Spring Cloud Netflix 可以实现 Netflix 组件的功能 - 服务发现（Eureka）、智能路由（Zuul）、客户端负载均衡（Ribbon）。





4、Dubbo 提供了各种 Filter，对于上述中“无”的要素，可以通过扩展 Filter 来完善。

5、dubbo 的开发难度较大，原因是 dubbo 的 jar 包依赖问题很多大型工程无法解决。

核心要素	Dubbo	Spring Cloud
服务注册中心	Zookeeper、Redis	Spring Cloud Netflix Eureka
服务调用方式	RPC	REST API
服务网关	无	Spring Cloud Netflix Zuul
断路器	不完善	Spring Cloud Netflix Hystrix
分布式配置	无	Spring Cloud Config
分布式追踪系统	无	Spring Cloud Sleuth
消息总线	无	Spring Cloud Bus
数据流	无	Spring Cloud Stream 基于Redis,Rabbit,Kafka实现的消息微服务
批量任务	无	Spring Cloud Task

架构师专栏

## 4. springboot 和 springcloud 认识

- Spring Boot 是 Spring 的一套快速配置脚手架，可以基于 Spring Boot 快速开发单个微服务，Spring Cloud 是一个基于 Spring Boot 实现的云应用开发工具；
- Spring Boot 专注于快速、方便集成的单个微服务个体，Spring Cloud 关注全局的服务治理框架；
- Spring Boot 使用了默认大于配置的理念，很多集成方案已经帮你选择好了，能不配置就不配置；
- Spring Cloud 很大的一部分是基于 Spring Boot 来实现，可以不基于 Spring Boot 吗？不可以。



## 5. 什么是服务熔断，什么是服务降级

### 服务熔断：

如果检查出来频繁超时，就把 consumer 调用 provider 的请求，直接短路掉，不实际调用，而是直接返回一个 mock 的值。

### 服务降级：

**consumer 端：**consumer 如果发现某个 provider 出现异常情况，比如，经常超时(可能是熔断引起的降级)，数据错误，这时，consumer 可以采取一定的策略，降级 provider 的逻辑，基本的有直接返回固定的数据。

**provider 端：**当 provider 发现流量激增的时候，为了保护自身的稳定性，也可能考虑降级服务。

1、直接给 consumer 返回固定数据

2、需要实时写入数据库的，先缓存到队列里，异步写入数据库。

## 6. 微服务的优缺点

### 优点：

- 单一职责：每个微服务仅负责自己业务领域的功能；



- 自治：一个微服务就是一个独立的实体，它可以独立部署、升级，服务与服务之间通过 REST 等形式标准接口进行通信，并且一个微服务实例可以被替换成另一种实现，而对其它的微服务不产生影响。
- 逻辑清晰：微服务单一职责特性使微服务看起来逻辑清晰，易于维护。
- 简化部署：单系统中修改一处需要部署整个系统，而微服务中修改一处可单独部署一个服务
- 可扩展：应对系统业务增长的方法通常采用横向 (Scale out) 或纵向 (Scale up) 的方向进行扩展。分布式系统中通常要采用 Scale out 的方式进行扩展。
- 灵活组合：
- 技术异构：不同的服务之间，可以根据自己的业务特点选择不通的技术架构，如数据库等。

## 缺点：

- **复杂度高**：服务调用要考虑被调用方故障、过载、消息丢失等各种异常情况，代码逻辑更加复杂；对于微服务间的事务性操作，因为不同的微服务采用了不同的数据库，将无法利用数据库本身的事务机制保证一致性，需要引入二阶段提交等技术。
- **运维复杂**：系统由多个独立运行的微服务构成，需要一个设计良好的监控系统对各个微服务的运行状态进行监控。运维人员需要对系统有细致的了解才够更好的运维系统。
- **通信延迟**：微服务之间调用会有时间损耗，造成通信延迟。

## 7. 使用中碰到的坑

- **超时**：确保 Hystrix 超时时间配置为长于配置的 Ribbon 超时时间





- **feign path:** feign 客户端在部署时若有 contextpath 应该设置 path="/\*\*" 来匹配你的服务名。
- **版本:** springboot 和 springcloud 版本要兼容。

## 8. 列举微服务技术栈

- 服务网关 Zuul
- 服务注册发现 Eureka+Ribbon
- 服务配置中心Apollo
- 认证授权中心Spring Security OAuth2
- 服务框架 Spring Boot
- 数据总线 Kafka
- 日志监控 ELK
- 调用链监控 CAT
- Metrics 监控 KairosDB
- 健康检查和告警 ZMon
- 限流熔断和流聚合 Hystrix/Turbine

## 9. eureka 和 zookeeper 都可以提供服务的注册与发现功能，他们的区别

### Zookeeper 保证 CP

当向注册中心查询服务列表时，我们可以容忍注册中心返回的是几分钟以前的注册信息，但不能接受服务直接 down 掉不可用。也就是说，服务注册功能对可用



性的要求要高于一致性。但是 zk 会出现这样一种情况，当 master 节点因为网络故障与其他节点失去联系时，剩余节点会重新进行 leader 选举。问题在于，选举 leader 的时间太长，30 ~ 120s，且选举期间整个 zk 集群都是不可用的，这就导致在选举期间注册服务瘫痪。在云部署的环境下，因网络问题使得 zk 集群失去 master 节点是较大概率会发生的事，虽然服务能够最终恢复，但是漫长的选举时间导致的注册长期不可用是不能容忍的。

## Eureka 保证 AP

Eureka 看明白了这一点，因此在设计时就优先保证可用性。Eureka 各个节点都是平等的，几个节点挂掉不会影响正常节点的工作，剩余的节点依然可以提供注册和查询服务。而 Eureka 的客户端在向某个 Eureka 注册或如果发现连接失败，则会自动切换至其它节点，只要有一台 Eureka 还在，就能保证注册服务可用(保证可用性)，只不过查到的信息可能不是最新的(不保证强一致性)。除此之外，Eureka 还有一种自我保护机制，如果在 15 分钟内超过 85% 的节点都没有正常的心跳，那么 Eureka 就认为客户端与注册中心出现了网络故障，此时会出现以下几种情况：

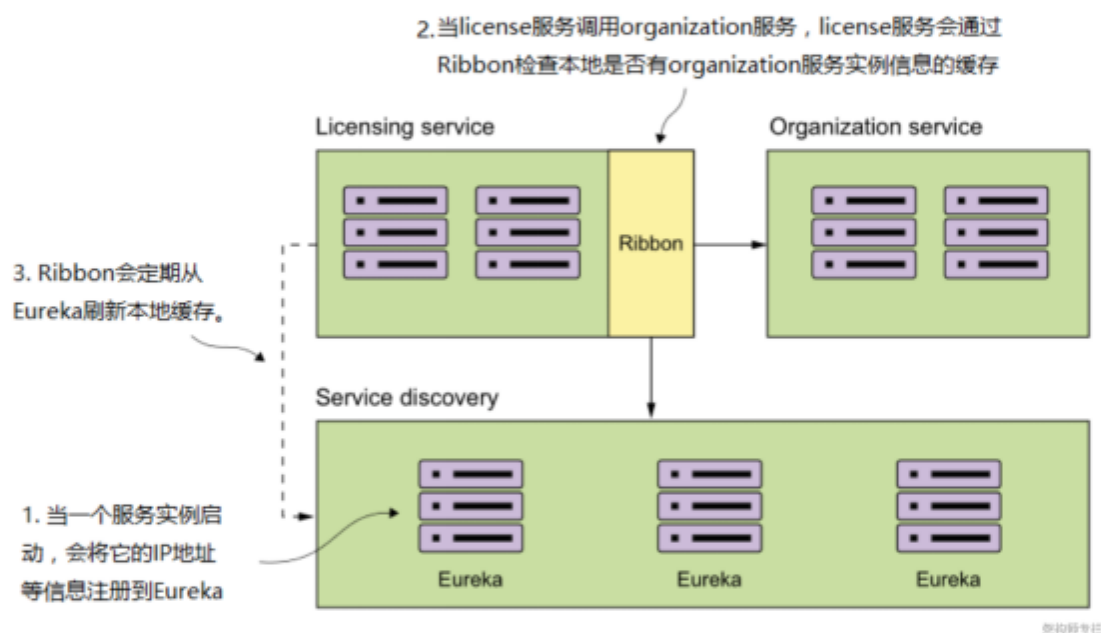
- 1、 Eureka 不再从注册列表中移除因为长时间没收到心跳而应该过期的服务
- 2、 Eureka 仍然能够接受新服务的注册和查询请求，但是不会被同步到其它节点上(即保证当前节点依然可用)
- 3、 当网络稳定时，当前实例新的注册信息会被同步到其它节点中

因此，Eureka 可以很好的应对因网络故障导致部分节点失去联系的情况，而不会像 zookeeper 那样使整个注册服务瘫痪。

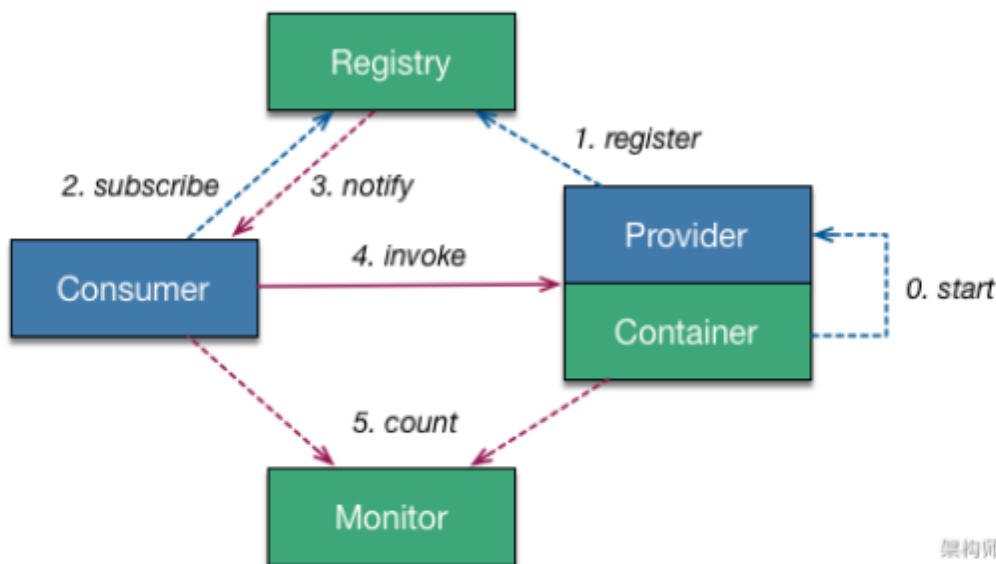
## 10. eureka 服务注册与发现原理



- 每 30s 发送心跳检测重新进行租约, 如果客户端不能多次更新租约, 它将在 90s 内从服务器注册中心移除。
- 注册信息和更新会被复制到其他 Eureka 节点, 来自任何区域的客户端可以查找注册中心信息, 每 30s 发生一次复制来定位他们的服务, 并进行远程调用。
- 客户端还可以缓存一些服务实例信息, 所以即使 Eureka 全挂掉, 客户端也是可以定位到服务地址的。



## 11. dubbo 服务注册与发现原理



架构师专栏

	角色说明
Provider	暴露服务的服务提供方
Consumer	调用远程服务的服务消费方
Registry	服务注册与发现的注册中心
Monitor	统计服务的调用次数和调用时间的监控中心
Container	服务运行容器

架构师专栏

#### 调用关系说明：

- 1、 服务容器负责启动,加载,运行服务提供者。
- 2、 服务提供者在启动时,向注册中心注册自己提供的服务。
- 3、 服务消费者在启动时,向注册中心订阅自己所需的服务。



- 4、注册中心返回服务提供者地址列表给消费者,如果有变更,注册中心将基于长连接推送变更数据给消费者。
- 5、服务消费者,从提供者地址列表中,基于软负载均衡算法,选一台提供者进行调用,如果调用失败,再选另一台调用。
- 6、服务消费者和提供者,在内存中累计调用次数和调用时间,定时每分钟发送一次统计数据到监控中心。

## 12. 限流

### 1、http 限流：我们使用nginx 的 limitzone 来完成：

```
//这个表示使用ip 进行限流 zone 名称为 req_one 分配了 10m 空间使用漏桶
算法 每秒钟允许 1 个请求
limit_req_zone $binary_remote_addr zone=req_one:10m rate=1r/s; //这
边 burst 表示可以瞬间超过 20 个请求 由于没有 noDelay 参数因此需要排队 如
果超过这 20 个那么直接返回 503
limit_req zone=req_one burst=20;
```

### 2、dubbo 限流：dubbo 提供了多个和请求相关的 filter：

**ActiveLimitFilter ExecuteLimitFilter TPSLimiterFilter**

#### 1、ActiveLimitFilter：

```
@Activate(group = Constants.CONSUMER, value =
Constants.ACTIVES_KEY)
```

作用于客户端，主要作用是控制客户端方法的并发度；





当超过了指定的 active 值之后该请求将等待前面的请求完成【何时结束呢？依赖于该方法的 timeout 如果没有设置 timeout 的话可能就是多个请求一直被阻塞然后等待随机唤醒。

## 2、ExecuteLimitFilter:

```
@Activate(group = Constants.PROVIDER, value =
Constants.EXECUTES_KEY)
```

作用于服务端，一旦超出指定的数目直接报错 其实是指在服务端的并行度【需要注意这些都是指的是在单台服务上而不是整个服务集群】

## 3、TPSLimiterFilter:

```
@Activate(group = Constants.PROVIDER, value =
Constants.TPS_LIMIT_RATE_KEY)
```

- 作用于服务端，控制一段时间内的请求数；
- 默认情况下取得 tps.interval 字段表示请求间隔 如果无法找到则使用60s 根据 tps 字段表示允许调用次数。
- 使用AtomicInteger表示允许调用的次数 每次调用减少 1 次当结果小于 0 之后返回不允许调用

## 3、springcloud 限流:

1、我们可以通过 semaphore.maxConcurrentRequests,coreSize,maxQueueSize 和



queueSizeRejectionThreshold 设置信号量模式下的最大并发量、线程池大小、缓冲区大小和缓冲区降级阈值。

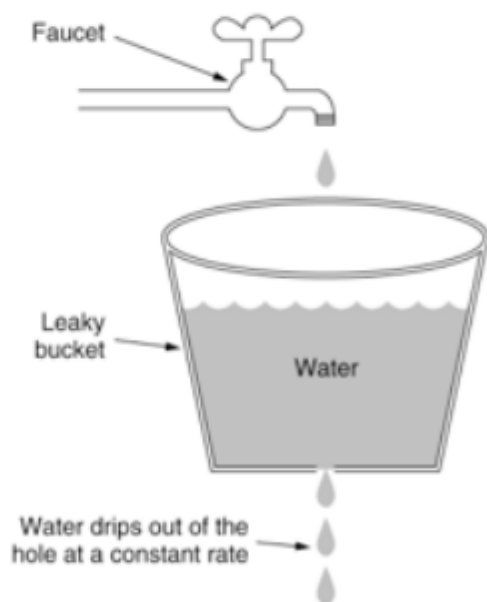
#不设置缓冲区，当请求数超过 coreSize 时直接降级

hystrix.threadpool.userThreadPool.maxQueueSize=-1 #超时时间大于我们的 timeout 接口返回时间

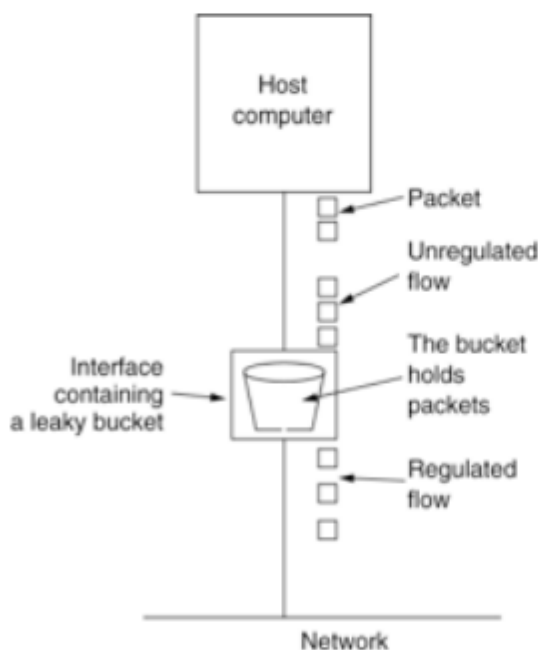
hystrix.command.userCommandKey.execution.isolation.thread.timeoutInMilliseconds=15000

这个时候我们连续多次请求/user/command/timeout 接口，在第一个请求还没有成功返回时,查看输出日志可以发现只有第一个请求正常的进入到 user-service 的接口中，其它请求会直接返回降级信息。这样我们就实现了对服务请求的限流。

**2、漏桶算法：**水（请求）先进入到漏桶里，漏桶以一定的速度出水，当水流入速度过大会直接溢出，可以看出漏桶算法能强行限制数据的传输速率。



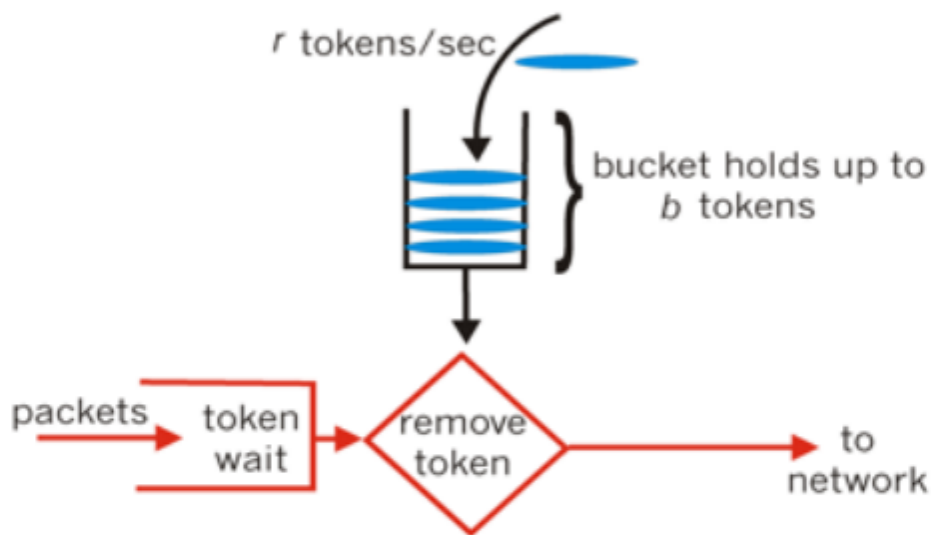
(a)



(b)

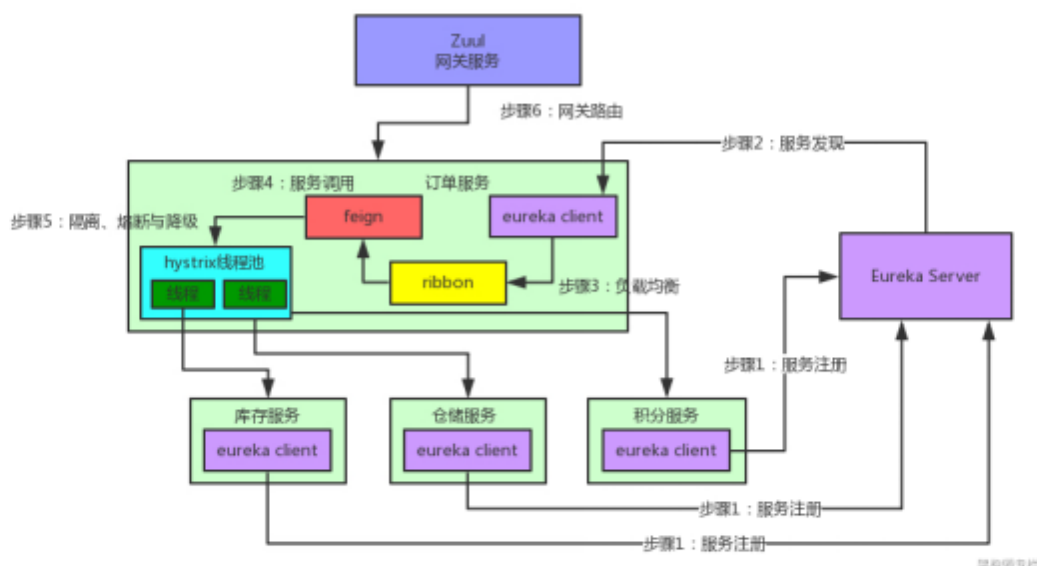


**3、令牌桶算法：**除了要求能够限制数据的平均传输速率外，还要求允许某种程度的突发传输。这时候漏桶算法可能就不合适了，令牌桶算法更为适合。如图所示，令牌桶算法的原理是系统会以一个恒定的速度往桶里放入令牌，而如果请求需要被处理，则需要先从桶里获取一个令牌，当桶里没有令牌可取时，则拒绝服务。



**4、redis 计数器限流；**

**13. springcloud 核心组件及其作用，以及 springcloud 工作原理：**



springcloud 由以下几个核心组件构成：

- **Eureka:** 各个服务启动时，Eureka Client 都会将服务注册到 Eureka Server，并且 Eureka Client 还可以反过来从 Eureka Server 拉取注册表，从而知道其他服务在哪里
- **Ribbon:** 服务间发起请求的时候，基于 Ribbon 做负载均衡，从一个服务的多台机器中选择一台
- **Feign:** 基于 Feign 的动态代理机制，根据注解和选择的机器，拼接请求 URL 地址，发起请求
- **Hystrix:** 发起请求是通过 Hystrix 的线程池来走的，不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务雪崩的问题
- **Zuul:** 如果前端、移动端要调用后端系统，统一从 Zuul网关进入，由 Zuul网关转发请求给对应的服务

## 14. eureka 的缺点：



某个服务不可用时，各个 Eureka Client 不能及时的知道，需要 1~3 个心跳周期才能感知，但是，由于基于 Netflix 的服务调用端都会使用 Hystrix 来容错和降级，当服务调用不可用时 Hystrix 也能及时感知到，通过熔断机制来降级服务调用，因此弥补了基于客户端服务发现的时效性的缺点。

## 15. eureka 缓存机制：

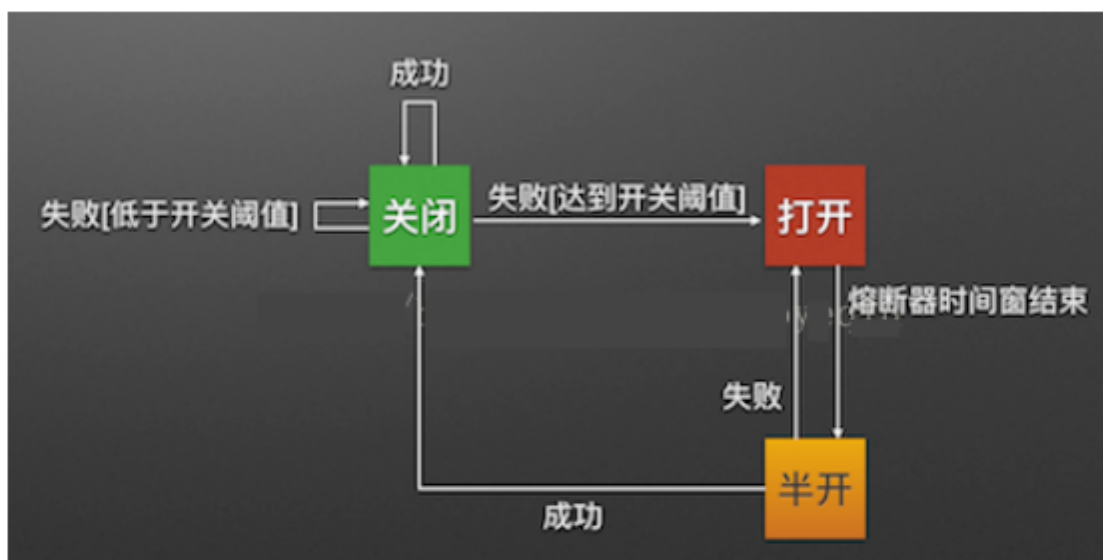


- **第一层缓存：**readOnlyCacheMap，本质上是 ConcurrentHashMap：这是一个 JVM 的 CurrentHashMap 只读缓存，这个主要是为了供客户端获取注册信息时使用，其缓存更新，依赖于定时器的更新，通过和 readWriteCacheMap 的值做对比，如果数据不一致，则以 readWriteCacheMap 的数据为准。  
readOnlyCacheMap 缓存更新的定时器时间间隔，默认为 30 秒
- **第二层缓存：**readWriteCacheMap，本质上是 Guava 缓存：此处存放的是最终的缓存，当服务下线，过期，注册，状态变更，都会来清除这个缓存里面的数据。然后通过 CacheLoader 进行缓存加载，在进行 readWriteCacheMap.get(key)的时候，首先看这个缓存里面有没有该数据，如果没有则通过 CacheLoader 的 load 方法去加载，加载成功之后将数据放入缓存，同时返回数据。readWriteCacheMap 缓存过期时间，默认为 180 秒。
- **缓存机制：**设置了一个每 30 秒执行一次的定时任务，定时去服务端获取注册信息。获取之后，存入本地内存。





## 16. 熔断的原理，以及如何恢复？



a. 服务的健康状况 = 请求失败数 / 请求总数。

熔断器开关由关闭到打开的状态转换是通过当前服务健康状况和设定阈值比较决定的。

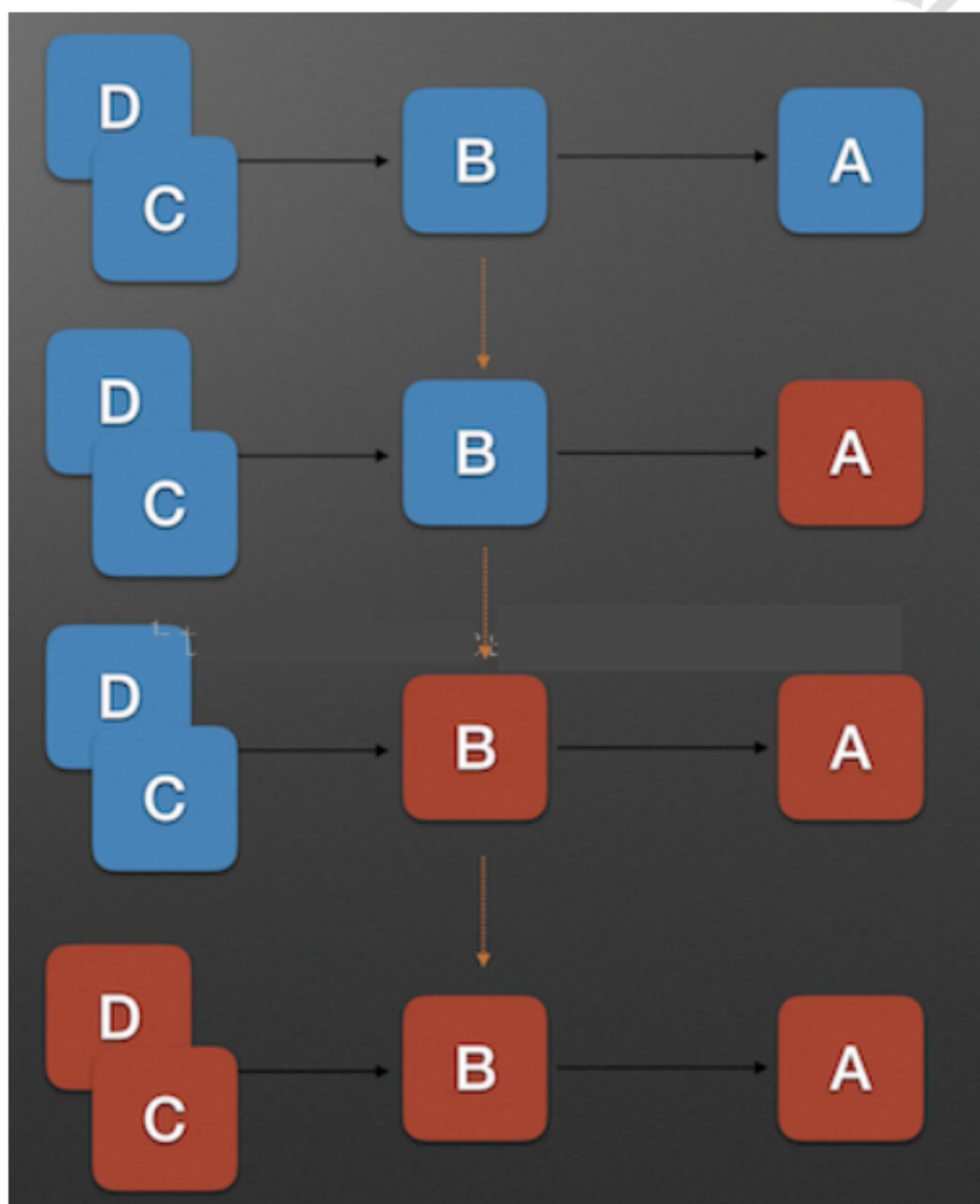
- 当熔断器开关关闭时，请求被允许通过熔断器。如果当前健康状况高于设定阈值，开关继续保持关闭。如果当前健康状况低于设定阈值，开关则切换为打开状态。
- 当熔断器开关打开时，请求被禁止通过。
- 当熔断器开关处于打开状态，经过一段时间后，熔断器会自动进入半开状态，这时熔断器只允许一个请求通过。当该请求调用成功时，熔断器恢复到关闭状态。若该请求失败，熔断器继续保持打开状态，接下来的请求被禁止通过。

熔断器的开关能保证服务调用者在调用异常服务时，快速返回结果，避免大量的同步等待。并且熔断器能在一段时间后继续侦测请求执行结果，提供恢复服务调用的可能。



## 17. 服务雪崩?

**简介：**服务雪崩效应是一种因服务提供者的不可用导致服务调用者的不可用,并将不可用逐渐放大的过程。





形成原因：

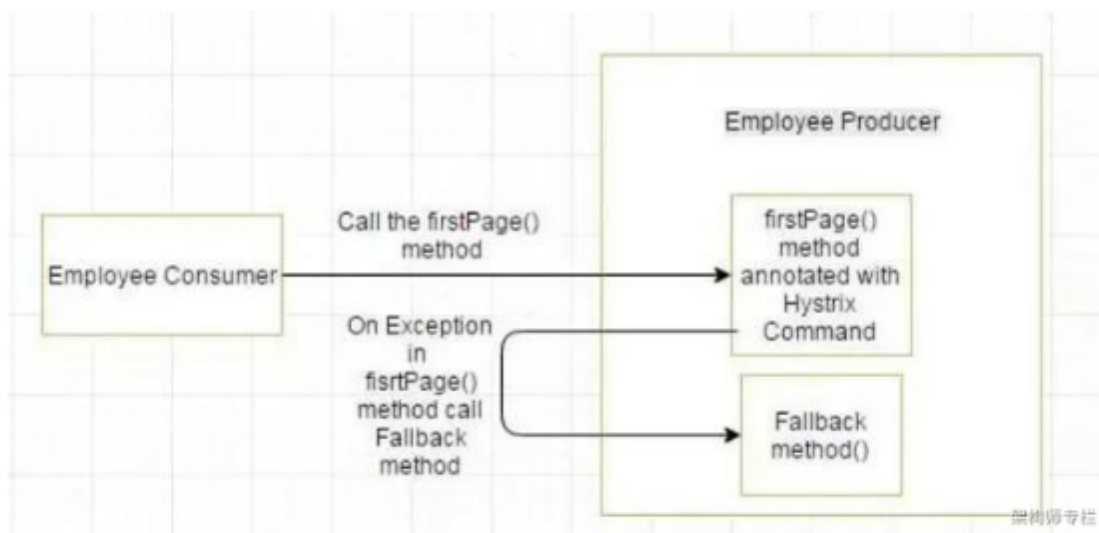
- 服务提供者不可
- 重试加大流量
- 服务调用者不可用

采用策略：

- 流量控制
- 改进缓存模式
- 服务自动扩容
- 服务调用者降级服务

## 18. 什么是 Hystrix 断路器？我们需要它吗？

由于某些原因，employee-consumer 公开服务会引发异常。在这种情况下使用 Hystrix 我们定义了一个回退方法。如果在公开服务中发生异常，则回退方法返回一些默认值



中断，并且员工使用者将一起跳过 firstPage 方法，并直接调用回退方法。断路器的目的是给第一页方法或第一页方法可能调用的其他方法留出时间，并导致异常恢复。可能发生的情况是，在负载较小的情况下，导致异常的问题有更好的恢复机会。



## 19. 多个消费者调用同一接口，eruka 默认的分配方式是什么？



- RoundRobinRule:轮询策略, Ribbon 以轮询的方式选择服务器,这个是默认值。所以示例中所启动的两个服务会被循环访问;
- RandomRule:随机选择, 也就是说 Ribbon 会随机从服务器列表中选择一个进行访问;
- BestAvailableRule:最大可用策略, 即先过滤出故障服务器后, 选择一个当前并发请求数最小的;
- WeightedResponseTimeRule:带有加权的轮询策略, 对各个服务器响应时间进行加权处理, 然后在采用轮询的方式来获取相应的服务器;
- AvailabilityFilteringRule:可用过滤策略, 先过滤出故障的或并发请求大于阈值一部分服务实例, 然后再以线性轮询的方式从过滤后的实例清单中选出一个;
- ZoneAvoidanceRule:区域感知策略, 先使用主过滤条件 (区域负载器, 选择最优区域) 对所有实例过滤并返回过滤后的实例清单, 依次使用次过滤条件列表中的过滤条件对主过滤条件的结果进行过滤, 判断最小过滤数 (默认 1) 和最小过滤百分比 (默认 0), 最后对满足条件的服务器则使用RoundRobinRule(轮询方式)选择一个服务器实例。

## 20. 接口限流方法?

- 限制 总并发数 (比如 数据库连接池、线程池)
  - 1、 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)
  - 2、 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)
  - 3、 限制 远程接口 调用速率
  - 4、 限制 MQ 的消费速率
  - 5、 可以根据网络连接数、网络流量、CPU 或内存负载等来限流