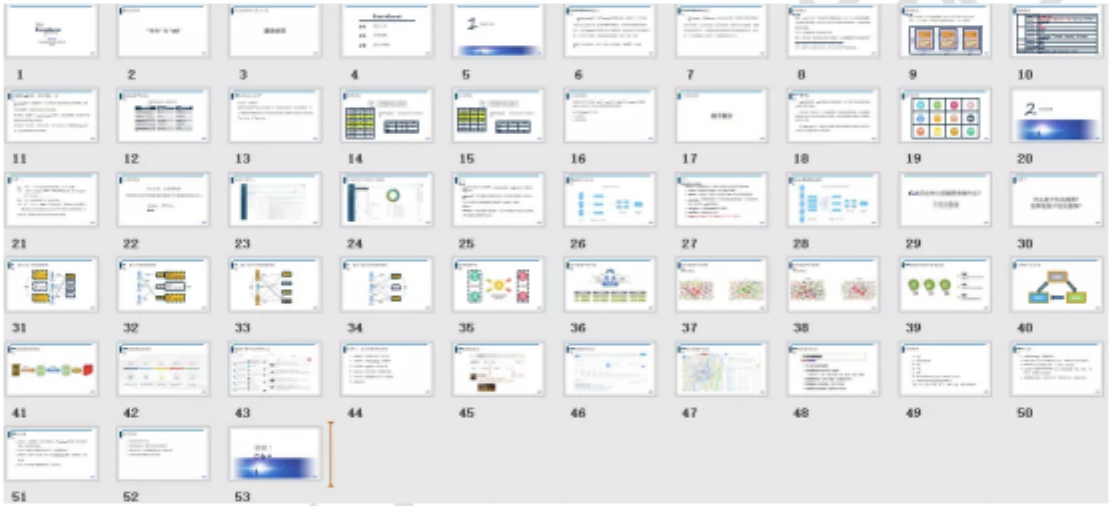
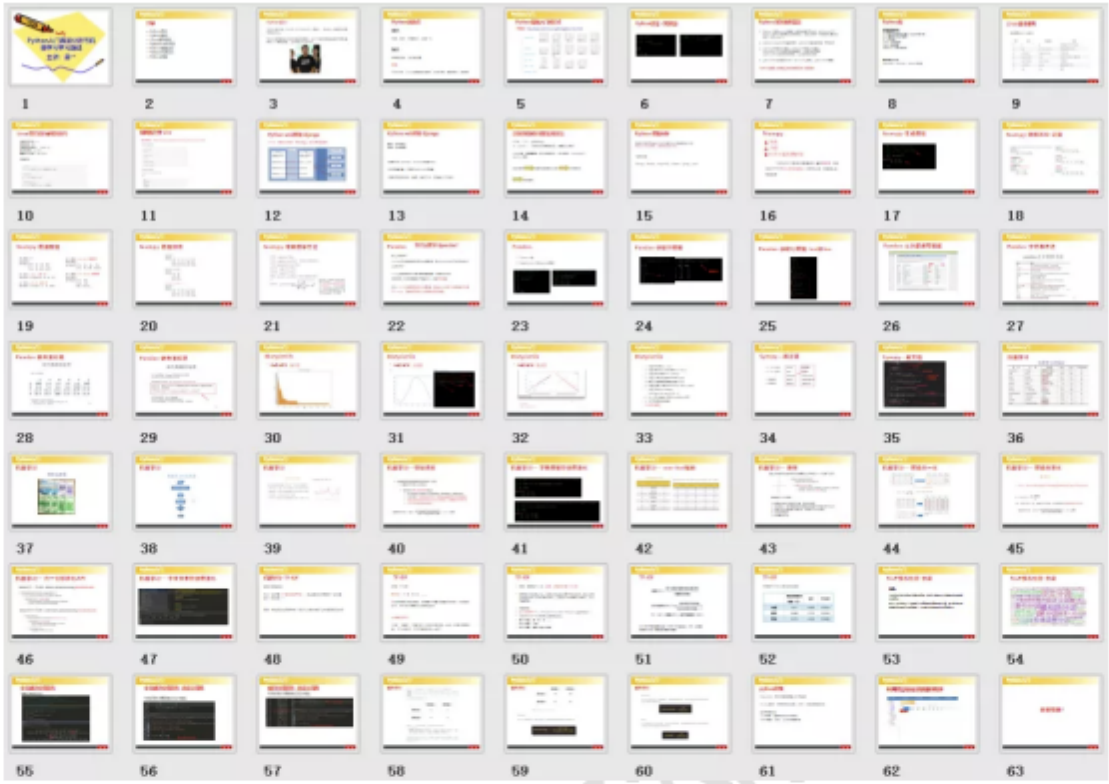
第二版：SpringCloud 70 道

目录

53 页《Elasticsearch 调优演讲 PPT》



63 页《Python 数据分析入门 PPT》



什么是微服务架构

- 微服务架构就是将单体的应用程序分成多个应用程序，这多个应用程序就成为微服务，每个微服务运行在自己的进程中，并使用轻量级的机制通信。这些服务围绕业务能力来划分，并通过自动化部署机制来独立部署。这些服务可以使用不同的编程语言，不同数据库，以保证最低限度的集中式管理。

为什么需要学习 Spring Cloud

- 首先 springcloud 基于 spingboot 的优雅简洁，可还记得我们被无数 xml 支配的恐惧？可还记得 springmvc，mybatis 错综复杂的配置，有了 spingboot，这些东西都不需要了，spingboot 好处不再赘诉，springcloud 就基于 SpringBoot 把市场上优秀的服务框架组合起来，通过 Spring Boot 风格进行再封装屏蔽掉了复杂的配置和实现原理
- 什么叫做开箱即用？即使是当年的黄金搭档 dubbo+zookeeper 下载配置起来也是颇费心神的！而 springcloud 完成这些只需要一个 jar 的依赖就可以了！
- springcloud 大多数子模块都是直击痛点，像 zuul 解决的跨域，fegin 解决的负载均衡，hystrix 的熔断机制等等等等

Spring Cloud 是什么

- Spring Cloud 是一系列框架的有序集合。它利用 Spring Boot 的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、智能路由、消息总线、负载均衡、断路器、数据监控等，都可以用 Spring Boot 的开发风格做到一键启动和部署。
- Spring Cloud 并没有重复制造轮子，它只是将各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过 Spring Boot 风格进行再封装屏蔽掉了

复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包。

SpringCloud 的优缺点

优点：

1.耦合度比较低。不会影响其他模块的开发。

2.减轻团队的成本，可以并行开发，不用关注其他人怎么开发，先关注自己的开发。

3.配置比较简单，基本用注解就能实现，不用使用过多的配置文件。

4.微服务跨平台的，可以用任何一种语言开发。

5.每个微服务可以有自己的独立的数据库也有用公共的数据库。

6.直接写后端的代码，不用关注前端怎么开发，直接写自己的后端代码即可，然后暴露接口，通过组件进行服务通信。

缺点：

1.部署比较麻烦，给运维工程师带来一定的麻烦。

2.针对数据的管理比麻烦，因为微服务可以每个微服务使用一个数据库。

3.系统集成测试比较麻烦

4.性能的监控比较麻烦。【最好开发一个大屏监控系统】

- 总的来说优点大过于缺点，目前看来 Spring Cloud 是一套非常完善的分布式框架，目前很多企业开始用微服务、Spring Cloud 的优势是显而易见的。因此对于想研究微服务架构的同学来说，学习 Spring Cloud 是一个不错的选择。

SpringBoot 和 SpringCloud 的区别?

- SpringBoot 专注于快速方便的开发单个个体微服务。
- SpringCloud 是关注全局的微服务协调整理治理框架，它将 SpringBoot 开发的一个个单体微服务整合并管理起来，
- 为各个微服务之间提供，配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等等集成服务
- SpringBoot 可以离开 SpringCloud 独立使用开发项目， 但是 SpringCloud 离不开 SpringBoot ，属于依赖的关系
- 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。

Spring Cloud 和 SpringBoot 版本对应关系

Spring Cloud Version	SpringBoot Version
Hoxton	2.2.x
Greenwich	2.1.x
Finchley	2.0.x

Edgware	1.5.x
Dalston	1.5.x

SpringCloud 由什么组成

- 这就有很多了，我讲几个开发中最重要的
  - Spring Cloud Eureka：服务注册与发现
  - Spring Cloud Zuul：服务网关
  - Spring Cloud Ribbon：客户端负载均衡
  - Spring Cloud Feign：声明性的 Web 服务客户端
  - Spring Cloud Hystrix：断路器
  - Spring Cloud Config：分布式统一配置管理
  - 等 20 几个框架，开源一直在更新

使用 Spring Boot 开发分布式微服务时，我们面临什么问题

- （1）与分布式系统相关的复杂性-这种开销包括网络问题，延迟开销，带宽问题，安全问题。
- （2）服务发现-服务发现工具管理群集中的流程和服务如何查找和互相交谈。它涉及一个服务目录，在该目录中注册服务，然后能够查找并连接到该目录中的服务。
- （3）冗余-分布式系统中的冗余问题。

- （4）负载平衡 --负载平衡改善跨多个计算资源的工作负荷，诸如计算机，计算机集群，网络链路，中央处理单元，或磁盘驱动器的分布。
- （5）性能-问题 由于各种运营开销导致的性能问题。

Spring Cloud 和 dubbo 区别?

- （1）服务调用方式：dubbo 是 RPC springcloud Rest Api
- （2）注册中心：dubbo 是 zookeeper springcloud 是 eureka，也可以是 zookeeper
- （3）服务网关，dubbo 本身没有实现，只能通过其他第三方技术整合，springcloud 有 Zuul 路由网关，作为路由服务器，进行消费者的请求分发,springcloud 支持断路器，与 git 完美集成配置文件支持版本控制，事物总线实现配置文件的更新与服务自动装配等等一系列的微服务架构要素。

Eureka

服务注册和发现是什么意思？Spring Cloud 如何实现?

- 当我们开始一个项目时，我们通常在属性文件中进行所有的配置。随着越来越多的服务开发和部署，添加和修改这些属性变得更加复杂。有些服务可能会下降，而某些位置可能会发生变化。手动更改属性可能会产生问题。 Eureka 服务注册和发现可以在这种情况下提供帮助。由于所有服务都在 Eureka 服务器上注

册并通过调用 Eureka 服务器完成查找，因此无需处理服务地点的任何更改和处理。

什么是 Eureka

- Eureka 作为 SpringCloud 的服务注册功能服务器，他是服务注册中心，系统中的其他服务使用 Eureka 的客户端将其连接到 Eureka Service 中，并且保持心跳，这样工作人员可以通过 Eureka Service 来监控各个微服务是否运行正常。

Eureka 怎么实现高可用

- 集群吧，注册多台 Eureka，然后把 SpringCloud 服务互相注册，客户端从 Eureka 获取信息时，按照 Eureka 的顺序来访问。

什么是 Eureka 的自我保护模式，

- 默认情况下，如果 Eureka Service 在一定时间内没有接收到某个微服务的心跳，Eureka Service 会进入自我保护模式，在该模式下 Eureka Service 会保护服务注册表中的信息，不在删除注册表中的数据，当网络故障恢复后，Eureka Servic 节点会自动退出自我保护模式

DiscoveryClient 的作用

- 可以从注册中心中根据服务别名获取注册的服务器信息。

Eureka 和 ZooKeeper 都可以提供服务注册与发现的功能,请说说两个的区别

1、 ZooKeeper 中的节点服务挂了就要选举 在选举期间注册服务瘫痪,虽然服务最终会恢复,但是选举期间不可用的， 选举就是改微服务做了集群，必须有一台主其他的都是从

2、 Eureka 各个节点是平等关系,服务器挂了没关系，只要有一台 Eureka 就可以保证服务可用，数据都是最新的。 如果查询到的数据并不是最新的，就是因为 Eureka 的自我保护模式导致的

3、 Eureka 本质上是一个工程,而 ZooKeeper 只是一个进程

4、 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 一样使得整个注册系统瘫痪

5、 ZooKeeper 保证的是 CP，Eureka 保证的是 AP

CAP： C：一致性>Consistency; 取舍：(强一致性、单调一致性、会话一致性、最终一致性、弱一致性) A：可用性>Availability; P：分区容错性>Partition tolerance;

Zuul

什么是网关?

- 网关相当于一个网络服务架构的入口，所有网络请求必须通过网关转发到具体的服务。

网关的作用是什么

- 统一管理微服务请求，权限控制、负载均衡、路由转发、监控、安全控制黑名单和白名单等

什么是 Spring Cloud Zuul（服务网关）

- Zuul 是对 SpringCloud 提供的成熟对的路由方案，他会根据请求的路径不同，网关会定位到指定的微服务，并代理请求到不同的微服务接口，他对外隐蔽了微服务的真正接口地址。 三个重要概念：动态路由表，路由定位，反向代理：
  - 动态路由表：Zuul 支持 Eureka 路由，手动配置路由，这俩种都支持自动更新
  - 路由定位：根据请求路径，Zuul 有自己的一套定位服务规则以及路由表达式匹配
  - 反向代理：客户端请求到路由网关，网关受理之后，在对目标发送请求，拿到响应之后在 给客户端
- 它可以和 Eureka,Ribbon,Hystrix 等组件配合使用，
- Zuul 的应用场景：
  - 对外暴露，权限校验，服务聚合，日志审计等

网关与过滤器有什么区别

- 网关是对所有服务的请求进行分析过滤，过滤器是对单个服务而言。

常用网关框架有那些?

- Nginx、Zuul、Gateway

Zuul 与 Nginx 有什么区别?

- Zuul 是 java 语言实现的，主要为 java 服务提供网关服务，尤其在微服务架构中可以更加灵活的对网关进行操作。Nginx 是使用 C 语言实现，性能高于 Zuul，但是实现自定义操作需要熟悉 lua 语言，对程序员要求较高，可以使用 Nginx 做 Zuul 集群。

既然 Nginx 可以实现网关？为什么还需要使用 Zuul 框架

- Zuul 是 SpringCloud 集成的网关，使用 Java 语言编写，可以对 SpringCloud 架构提供更灵活的服务。

如何设计一套 API 接口

- 考虑到 API 接口的分类可以将 API 接口分为开发 API 接口和内网 API 接口，内网 API 接口用于局域网，为内部服务器提供服务。开放 API 接口用于对外部合作单位提供接口调用，需要遵循 Oauth2.0 权限认证协议。同时还需要考虑安全性、幂等性等问题。

ZuulFilter 常用有那些方法

- Run()：过滤器的具体业务逻辑
- shouldFilter()：判断过滤器是否有效
- filterOrder()：过滤器执行顺序
- filterType()：过滤器拦截位置

如何实现动态 Zuul 网关路由转发

- 通过 path 配置拦截请求，通过 ServiceId 到配置中心获取转发的服务列表，Zuul 内部使用 Ribbon 实现本地负载均衡和转发。

Zuul 网关如何搭建集群

- 使用 Nginx 的 upstream 设置 Zuul 服务集群，通过 location 拦截请求并转发到 upstream，默认使用轮询机制对 Zuul 集群发送请求。

Ribbon

负载平衡的意义什么?

- 简单来说： 先将集群，集群就是把一个的事情交给多个人去做，假如要做 1000 个产品给一个人做要 10 天，我叫 10 个人做就是一天，这就是集群，负载均衡的话就是用来控制集群，他把做的最多的人让他慢慢做休息会，把做的最少的人让他加量让他做多点。
- 在计算中，负载平衡可以改善跨计算机，计算机集群，网络链接，中央处理单元或磁盘驱动器等多种计算资源的工作负载分布。负载平衡旨在优化资源使用，最大化吞吐量，最小化响应时间并避免任何单一资源的过载。使用多个组件进行负载平衡而不是单个组件可能会通过冗余来提高可靠性和可用性。负载平衡通常涉及专用软件或硬件，例如多层交换机或域名系统服务器进程。

Ribbon 是什么?

- 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
- 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，随即连接等）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。（有点类似 Nginx）

Nginx 与 Ribbon 的区别

- Nginx 是反向代理同时可以实现负载均衡，nginx 拦截客户端请求采用负载均衡策略根据 upstream 配置进行转发，相当于请求通过 nginx 服务器进行转发。Ribbon 是客户端负载均衡，从注册中心读取目标服务器信息，然后客户端采用轮询策略对服务直接访问，全程在客户端操作。

Ribbon 底层实现原理

- Ribbon 使用 discoveryClient 从注册中心读取目标服务信息，对同一接口请求进行计数，使用%取余算法获取目标服务集群索引，返回获取到的目标服务信息。

@LoadBalanced 注解的作用

开启客户端负载均衡。

Hystrix

什么是断路器

- 当一个服务调用另一个服务由于网络原因或自身原因出现问题，调用者就会等待被调用者的响应 当更多的服务请求到这些资源导致更多的请求等待，发生连锁效应（雪崩效应）
- 断路器有三种状态
    - 打开状态：一段时间内 达到一定的次数无法调用 并且多次监测没有恢复的迹象 断路器完全打开 那么下次请求就不会请求到该服务

- 半开状态：短时间内 有恢复迹象 断路器会将部分请求发给该服务，正常调用时 断路器关闭
- 关闭状态：当服务一直处于正常状态 能正常调用

什么是 Hystrix?

- 在分布式系统，我们一定会依赖各种服务，那么这些个服务一定会出现失败的情况，就会导致雪崩，Hystrix 就是这样的一个工具，防雪崩利器，它具有服务降级，服务熔断，服务隔离，监控等一些防止雪崩的技术。
- Hystrix 有四种防雪崩方式:
  - 服务降级：接口调用失败就调用本地的方法返回一个空
  - 服务熔断：接口调用失败就会进入调用接口提前定义好的一个熔断的方法，返回错误信息
  - 服务隔离：隔离服务之间相互影响
  - 服务监控：在服务发生调用时,会将每秒请求数、成功请求数等运行指标记录下来。

谈谈服务雪崩效应

- 雪崩效应是在大型互联网项目中，当某个服务发生宕机时，调用这个服务的其他服务也会发生宕机，大型项目的微服务之间的调用是互通的，这样就会将服务的不可用逐步扩大到各个其他服务中，从而使整个项目的服务宕机崩溃.发生雪崩效应的原因有以下几点
- 单个服务的代码存在bug. 2 请求访问量激增导致服务发生崩溃(如大型商城的枪红包，秒杀功能). 3.服务器的硬件故障也会导致部分服务不可用.

在微服务中，如何保护服务?

- 一般使用使用 Hystrix 框架，实现服务隔离来避免出现服务的雪崩效应，从而达到保护服务的效果。当微服务中，高并发的数据库访问量导致服务线程阻塞，使单个服务宕机，服务的不可用会蔓延到其他服务，引起整体服务灾难性后果，使用服务降级能有效为不同的服务分配资源,一旦服务不可用则返回友好提示，不占用其他服务资源，从而避免单个服务崩溃引发整体服务的不可用.

服务雪崩效应产生的原因

- 因为 Tomcat 默认情况下只有一个线程池来维护客户端发送的所有的请求,这时候某一接口在某一时刻被大量访问就会占据 tomcat 线程池中的所有线程，其他请求处于等待状态，无法连接到服务接口。

谈谈服务降级、熔断、服务隔离

- 服务降级：当客户端请求服务器端的时候，防止客户端一直等待，不会处理业务逻辑代码，直接返回一个友好的提示给客户端。
- 服务熔断是在服务降级的基础上更直接的一种保护方式，当在一个统计时间范围内的请求失败数量达到设定值（requestVolumeThreshold）或当前的请求错误率达到设定的错误率阈值（errorThresholdPercentage）时开启断路，之后的请求直接走 fallback 方法，在设定时间（sleepWindowInMilliseconds）后尝试恢复。

- 服务隔离就是 Hystrix 为隔离的服务开启一个独立的线程池，这样在高并发的情况下不会影响其他服务。服务隔离有线程池和信号量两种实现方式，一般使用线程池方式。

服务降级底层是如何实现的?

- Hystrix 实现服务降级的功能是通过重写 HystrixCommand 中的 getFallback() 方法，当 Hystrix 的 run 方法或 construct 执行发生错误时转而执行 getFallback() 方法。

Feign

什么是 Feign?

- Feign 是一个声明 web 服务客户端，这使得编写 web 服务客户端更容易
- 他将我们需要调用的服务方法定义成抽象方法保存在本地就可以了，不需要自己构建 Http 请求了，直接调用接口就行了，不过要注意，调用方法要和本地抽象方法的签名完全一致。

SpringCloud 有几种调用接口方式

- Feign
- RestTemplate

Ribbon 和 Feign 调用服务的区别

- 调用方式同：Ribbon 需要我们自己构建 Http 请求，模拟 Http 请求然后通过 RestTemplate 发给其他服务，步骤相当繁琐
- 而 Feign 则是在 Ribbon 的基础上进行了一次改进，采用接口的形式，将我们需要调用的服务方法定义成抽象方法保存在本地就可以了，不需要自己构建 Http 请求了，直接调用接口就行了，不过要注意，调用方法要和本地抽象方法的签名完全一致。

Bus

什么是 Spring Cloud Bus?

- Spring Cloud Bus 就像一个分布式执行器，用于扩展的 Spring Boot 应用程序的配置文件，但也可以用作应用程序之间的通信通道。
- Spring Cloud Bus 不能单独完成通信，需要配合 MQ 支持
- Spring Cloud Bus 一般是配合 Spring Cloud Config 做配置中心的
- Springcloud config 实时刷新也必须采用 SpringCloud Bus 消息总线

Config

什么是 Spring Cloud Config?

- Spring Cloud Config 为分布式系统中的外部配置提供服务器和客户端支持，可以方便的对微服务各个环境下的配置进行集中式管理。Spring Cloud Config 分为 Config Server 和 Config Client 两部分。Config Server 负责读取配置文件，并且暴露 Http API 接口，Config Client 通过调用 Config Server 的接口来读取配置文件。

分布式配置中心有那些框架?

- Apollo、zookeeper、springcloud config。

分布式配置中心的作用?

- 动态变更项目配置信息而不必重新部署项目。

SpringCloud Config 可以实现实时刷新吗?

- springcloud config 实时刷新采用 SpringCloud Bus 消息总线。

Gateway

什么是 Spring Cloud Gateway?

- Spring Cloud Gateway 是 Spring Cloud 官方推出的第二代网关框架，取代 Zuul 网关。网关作为流量的，在微服务系统中有着非常作用，网关常见的功能有路由转发、权限校验、限流控制等作用。
- 使用了一个 RouteLocatorBuilder 的 bean 去创建路由，除了创建路由 RouteLocatorBuilder 可以让你添加各种 predicates 和 filters，predicates 断言的意思，顾名思义就是根据具体的请求的规则，由具体的 route 去处理，filters 是各种过滤器，用来对请求做各种判断和修改。

SpringCloud 主要项目

- Spring Cloud 的子项目，大致可分成两类，一类是对现有成熟框架"Spring Boot 化"的封装和抽象，也是数量最多的项目；第二类是开发了一部分分布式系统的基础设施的实现，如 Spring Cloud Stream 扮演的就是 kafka, ActiveMQ 这样的角色。

Spring Cloud Config

- Config 能够管理所有微服务的配置文件
- 集中配置管理工具，分布式系统中统一的外部配置管理，默认使用 Git 来存储配置，可以支持客户端配置的刷新及加密、解密操作。

Spring Cloud Netflix(重点，这些组件用的最多)

- Netflix OSS 开源组件集成，包括 Eureka、Hystrix、Ribbon、Feign、Zuul 等核心组件。

  - Eureka：服务治理组件，包括服务端的注册中心和客户端的服务发现机制；
  - Ribbon：负载均衡的服务调用组件，具有多种负载均衡调用策略；
  - Hystrix：服务容错组件，实现了断路器模式，为依赖服务的出错和延迟提供了容错能力；
  - Feign：基于 Ribbon 和 Hystrix 的声明式服务调用组件；
  - Zuul：API 网关组件，对请求提供路由及过滤功能。

我觉得 SpringCloud 的福音是 Netflix，他把人家的组件都搬来进行封装了，使开发者能快速简单安全的使用

Spring Cloud Bus

- 用于传播集群状态变化的消息总线，使用轻量级消息代理链接分布式系统中的节点，可以用来动态刷新集群中的服务配置信息。
- 简单来说就是修改了配置文件，发送一次请求，所有客户端便会重新读取配置文件。

  - 需要利用中间插件 MQ

Spring Cloud Consul

- Consul 是 HashiCorp 公司推出的开源工具，用于实现分布式系统的服务发现与配置。与其它分布式服务注册与发现的方案，Consul 的方案更“一站式”，内置了服务注册与发现框架、分布一致性协议实现、健康检查、Key/Value 存

储、多数据中心方案，不再需要依赖其它工具（比如 ZooKeeper 等）。使用起来也较为简单。Consul 使用 Go 语言编写，因此具有天然可移植性(支持 Linux、windows 和 Mac OS X)；安装包仅包含一个可执行文件，方便部署，与 Docker 等轻量级容器可无缝配合。

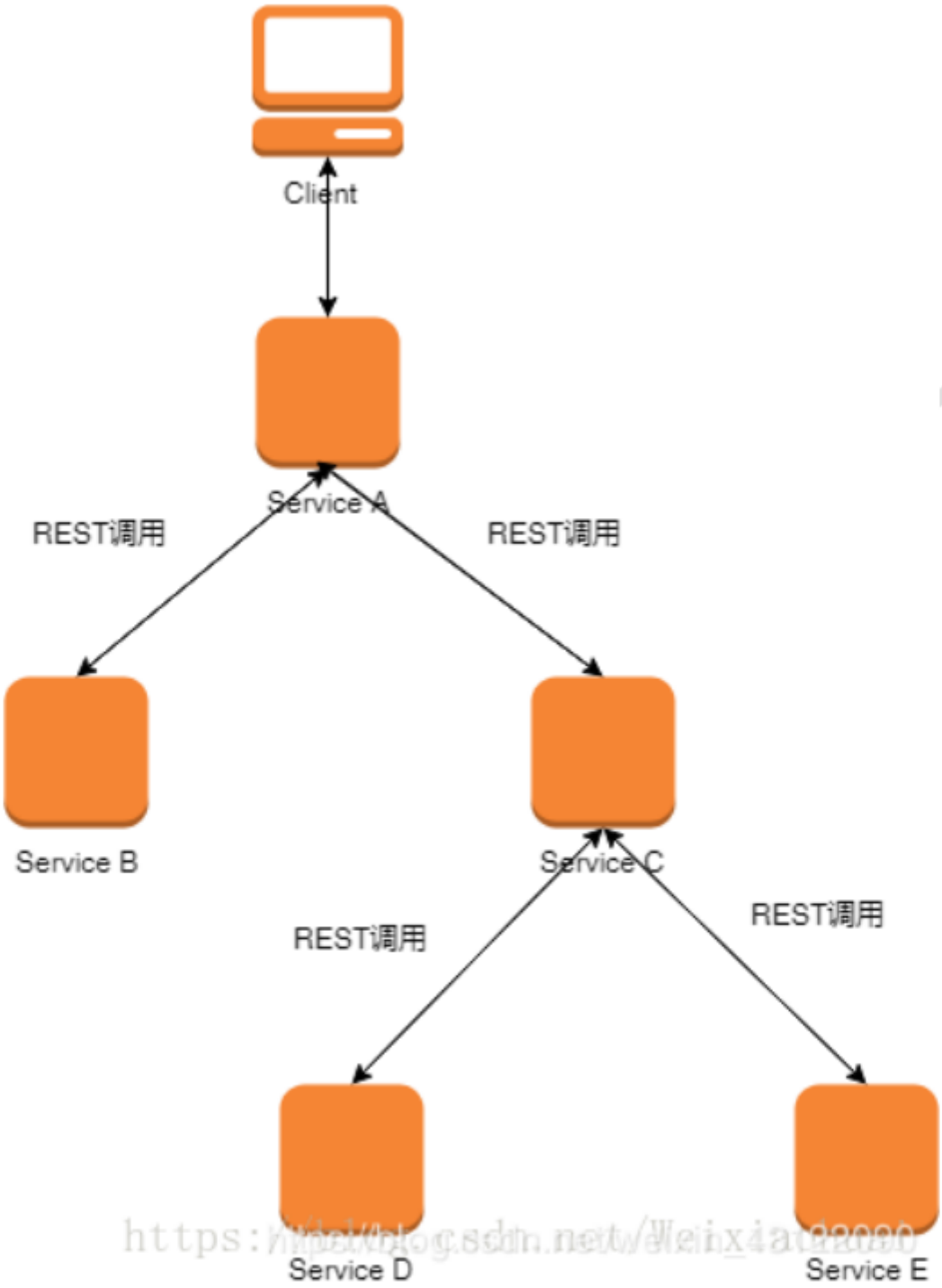
Spring Cloud Security

- 安全工具包，他可以对
  - 对 Zuul 代理中的负载均衡从前端到后端服务中获取 SSO 令牌
  - 资源服务器之间的中继令牌
  - 使 Feign 客户端表现得像 OAuth2RestTemplate（获取令牌等）的拦截器
  - 在 Zuul 代理中配置下游身份验证
- Spring Cloud Security 提供了一组原语，用于构建安全的应用程序和服务，而且操作简便。可以在外部（或集中）进行大量配置的声明性模型有助于实现大型协作的远程组件系统，通常具有中央身份管理服务。它也非常易于在 Cloud Foundry 等服务平台中使用。在 Spring Boot 和 Spring Security OAuth2 的基础上，可以快速创建实现常见模式的系统，如单点登录，令牌中继和令牌交换。

Spring Cloud Sleuth

- 在微服务中，通常根据业务模块分服务，项目中前端发起一个请求，后端可能跨几个服务调用才能完成这个请求（如下图）。如果系统越来越庞大，服务之间的调用与被调用关系就会变得很复杂，假如一个请求中需要跨几个服务调用，其中一个服务由于网络延迟等原因挂掉了，那么这时候我们需要分析具体哪一个服务出问题了就会显得很困难。Spring Cloud Sleuth 服务链路跟踪功能就可以帮助

我们快速的发现错误根源以及监控分析每条请求链路上的性能等等。



Spring Cloud Stream

- 轻量级事件驱动微服务框架，可以使用简单的声明式模型来发送及接收消息，主要实现为 Apache Kafka 及 RabbitMQ。

Spring Cloud Task

- Spring Cloud Task 的目标是为 Spring Boot 应用程序提供创建短运行期微服务的功能。在 Spring Cloud Task 中，我们可以灵活地动态运行任何任务，按需分配资源并在任务完成后检索结果。Tasks 是 Spring Cloud Data Flow 中的一个基础项目，允许用户将几乎任何 Spring Boot 应用程序作为一个短期任务执行。

Spring Cloud Zookeeper

- SpringCloud 支持三种注册方式 Eureka， Consul(go 语言编写)，zookeeper
- Spring Cloud Zookeeper 是基于 Apache Zookeeper 的服务治理组件。

Spring Cloud Gateway

- Spring cloud gateway 是 spring 官方基于 Spring 5.0、Spring Boot2.0 和 Project Reactor 等技术开发的网关，Spring Cloud Gateway 旨在为微服务架构提供简单、有效和统一的 API 路由管理方式，Spring Cloud Gateway 作为 Spring Cloud 生态系统中的网关，目标是替代 Netflix Zuul，其不仅提供统一的路由方式，并且还基于 Filer 链的方式提供了网关基本的功能，例如：安全、监控/埋点、限流等。

Spring Cloud OpenFeign

- Feign 是一个声明性的 Web 服务客户端。它使编写 Web 服务客户端变得更容易。要使用 Feign，我们可以将调用的服务方法定义成抽象方法保存在本地添加一点点注解就可以了，不需要自己构建 Http 请求了，直接调用接口就行了，不过要注意，调用方法要和本地抽象方法的签名完全一致。

Spring Cloud 的版本关系

- Spring Cloud 是一个由许多子项目组成的综合项目，各子项目有不同的发布节奏。 为了管理 Spring Cloud 与各子项目的版本依赖关系，发布了一个清单，其中包括了某个 Spring Cloud 版本对应的子项目版本。 为了避免 Spring Cloud 版本号与子项目版本号混淆，Spring Cloud 版本采用了名称而非版本号的命名，这些版本的名字采用了伦敦地铁站的名字，根据字母表的顺序来对应版本时间顺序，例如 Angel 是第一个版本，Brixton 是第二个版本。 当 Spring Cloud 的发布内容积累到临界点或者一个重大 BUG 被解决后，会发布一个 "service releases"版本，简称 SRX 版本，比如 Greenwich.SR2 就是 Spring Cloud 发布的 Greenwich 版本的第 2 个 SRX 版本。目前 Spring Cloud 的最新版本是 Hoxton。

Spring Cloud 和 SpringBoot 版本对应关系

Spring Cloud Version	SpringBoot Version

Hoxton	2.2.x
Greenwich	2.1.x
Finchley	2.0.x
Edgware	1.5.x
Dalston	1.5.x

Spring Cloud 和各子项目版本对应关系

- Edgware.SR6：我理解为最低版本号
- Greenwich.SR2 :我理解为最高版本号
- Greenwich.BUILD-SNAPSHOT（快照）：是一种特殊的版本，指定了某个当前的开发进度的副本。不同于常规的版本，几乎每天都要提交更新的版本，如果每次提交都申明一个版本号那不是版本号都不够用?

Component	Edgware.SR6	Greenwich.SR2	Greenwich.BUILD-SNAPSHOT
spring-cloud-aws	1.2.4.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
spring-cloud-bus	1.3.4.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
spring-cloud-cli	1.4.1.RELEASE	2.0.0.RELEASE	2.0.1.BUILD-SNAPSHOT
spring-cloud-commons	1.3.6.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT

spring-cloud-contract	1.2.7.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
spring-cloud-config	1.4.7.RELEASE	2.1.3.RELEASE	2.1.4.BUILD-SNAPSHOT
spring-cloud-netflix	1.4.7.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
spring-cloud-security	1.2.4.RELEASE	2.1.3.RELEASE	2.1.4.BUILD-SNAPSHOT
spring-cloud-cloudfoundry	1.1.3.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
spring-cloud-consul	1.3.6.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
spring-cloud-sleuth	1.3.6.RELEASE	2.1.1.RELEASE	2.1.2.BUILD-SNAPSHOT
spring-cloud-stream	Ditmars.SR5	Fishtown.SR3	Fishtown.BUILD-SNAPSHOT
spring-cloud-zookeeper	1.2.3.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
spring-boot	1.5.21.RELEASE	2.1.5.RELEASE	2.1.8.BUILD-SNAPSHOT
spring-cloud-task	1.2.4.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
spring-cloud-vault	1.1.3.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
spring-cloud-gateway	1.0.3.RELEASE	2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT

spring-cloud-openfeign		2.1.2.RELEASE	2.1.3.BUILD-SNAPSHOT
spring-cloud-function	1.0.2.RELEASE	2.0.2.RELEASE	2.0.3.BUILD-SNAPSHOT