微服务

在微服务架构风格中，一个大应用被拆分成为了多个小的服务系统提供出来，这些小的系统他们可以自成体系，也就是说这些小系统可以拥有自己的数据库，框架甚至语言等，这些小系统通常以提供 Rest Api 风格的接口来被 H5, Android, IOS 以及第三方应用程序调用。

SpringCloud

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、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并没有重复制造轮子，它只是将目前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。

微服务是可以独立部署、水平扩展、独立访问（或者有独立的数据库）的服务单元，springcloud就是这些微服务的大管家，采用了微服务这种架构之后，项目的数量会非常多，springcloud做为大管家需要管理好这些微服务，自然需要很多小弟来帮忙。

主要的小弟有：Spring Cloud Config、Spring Cloud Netflix（Eureka、Hystrix、Zuul、Archaius...）、Spring Cloud Bus、Spring Cloud for Cloud Foundry、Spring Cloud Cluster、Spring Cloud Consul、Spring Cloud Security、Spring Cloud Sleuth、Spring Cloud Data Flow、Spring Cloud Stream、Spring Cloud Task、Spring Cloud Zookeeper、Spring Cloud Connectors、Spring Cloud Starters、Spring Cloud CLI，

与springboot的关系

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、方便集成的单个个体，Spring Cloud是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。

Spring Boot可以离开Spring Cloud独立使用开发项目，但是Spring Cloud离不开Spring Boot，属于依赖的关系。

简介：

http://www.ityouknow.com/springcloud/2017/05/01/simple-springcloud.html

分布式与集群

分布式是指将不同的业务分布在不同的地方；而集群指的是将几台服务器集中在一起，实现同一业务。 分布式中的每一个节点，都可以做集群。 而集群并不一定就是分布式的。

需要选择一个来用

基于 RestTemplate 的服务调用

Spring框架提供的RestTemplate类可用于在应用中调用rest服务，它简化了与http服务的通信方式，统一了RESTful的标准，封装了http链接，我们只需要传入url及返回值类型即可。相较于之前常用的HttpClient，RestTemplate是一种更优雅的调用RESTful服务的方式。

原文链接：https://blog.csdn.net/weixin_53287520/article/details/120719034

`RestTemplate` 就是 Spring 封装的处理同步 HTTP 请求的类。

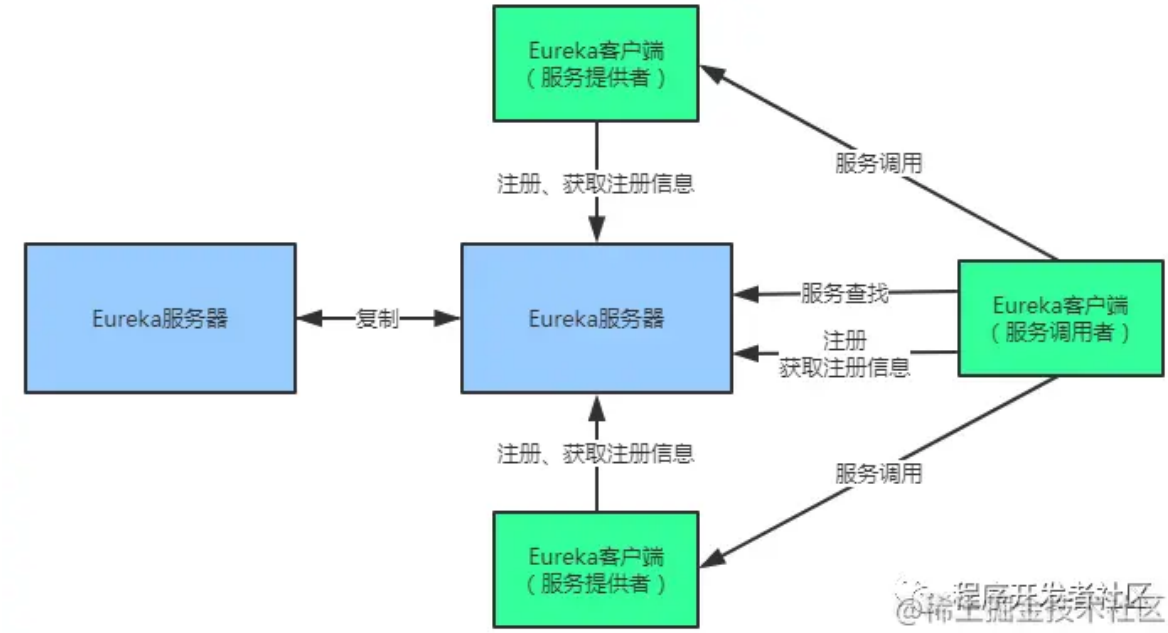
https://juejin.cn/post/6844903842065154061

基于 Eureka 的服务注册与发现

Eureka是Netflix开发的服务注册与发现框架。Spring Cloud集成了Eureka作为其默认的也是推荐的服务注册中心组件，并提供了开箱即用的支持。Eureka包括两个组件：Eureka Client和Eureka Server。

https://juejin.cn/post/6953140937358835719

Eureka 高可用集群



https://juejin.cn/post/6844904015768059912

基于 Ribbon 的服务调用&负载均衡

负载均衡与服务调用组件

这个链接指向RIBBON，Gatway,Config,Eureka

http://c.biancheng.net/springcloud/ribbon.html

Ribbon 可以与 RestTemplate（Rest 模板）配合使用，以实现微服务之间的调用。

RestTemplate 是 Spring 家族中的一个用于消费第三方 REST 服务的请求框架。RestTemplate 实现了对 HTTP 请求的封装，提供了一套模板化的服务调用方法。通过它，Spring 应用可以很方便地对各种类型的 HTTP 请求进行访问。

RestTemplate 针对各种类型的 HTTP 请求都提供了相应的方法进行处理，例如 HEAD、GET、POST、PUT、DELETE 等类型的 HTTP 请求，分别对应 RestTemplate 中的 headForHeaders()、getForObject()、postForObject()、put() 以及 delete() 方法。

基于 Feign 的服务调用&负载均衡

Feign是一个http请求调用的轻量级框架，可以以java接口注解的方式调用http请求，而不用像java中通过封装HTTP请求报文的方式直接调用。Feign通过处理注解，将请求模板化，当实际调用的时候，传入参数，根据参数再应用到请求上，进而转化成真正的请求，这种请求相对而言比较直观。

Feign是Netflflix开发的声明式，模板化的HTTP客户端，其灵感来自Retrofifit,JAXRS-2.0以及WebSocket.

Feign可帮助我们更加便捷，优雅的调用HTTP API。

在SpringCloud中，使用Feign非常简单——创建一个接口，并在接口上添加一些注解，代码就完成了。

Feign支持多种注解，例如Feign自带的注解或者JAX-RS注解等。

SpringCloud对Feign进行了增强，使Feign支持了SpringMVC注解，并整合了Ribbon和Eureka，从而让Feign的使用更加方便。

https://www.cnblogs.com/yangjiaoshou/p/15065961.html

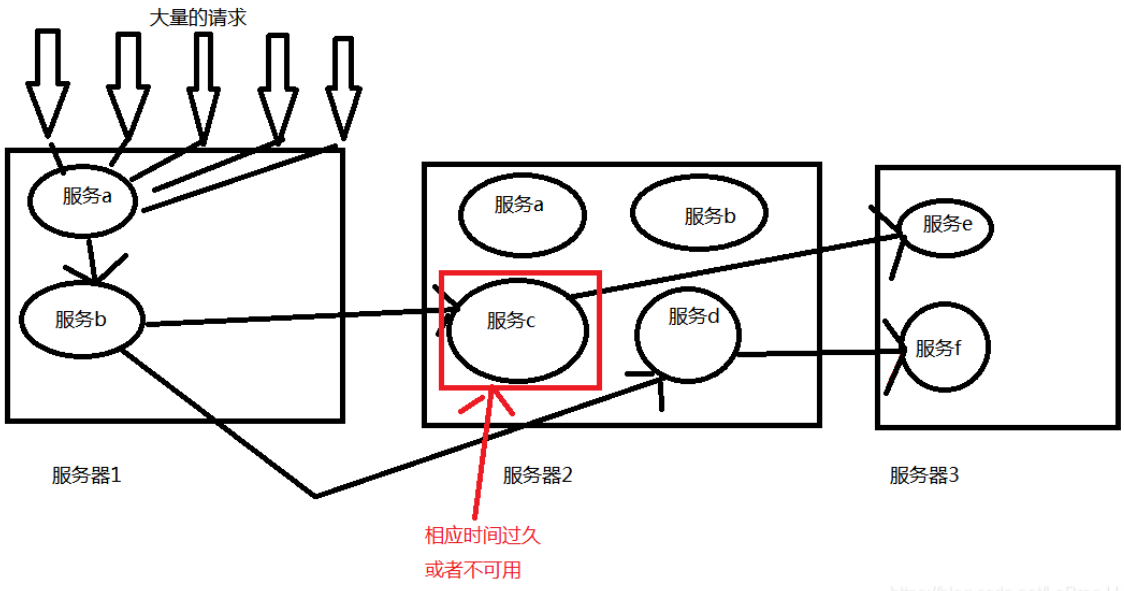
Ribbon是一个基于 HTTP 和 TCP 客户端 的负载均衡的工具。它可以 在客户端 配置 RibbonServerList（服务端列表），使用 HttpClient 或 RestTemplate 模拟http请求，步骤相当繁琐。

Feign 是在 Ribbon的基础上进行了一次改进，是一个使用起来更加方便的 HTTP 客户端。采用接口的方式， 只需要创建一个接口，然后在上面添加注解即可 ，将需要调用的其他服务的方法定义成抽象方法即可， 不需要自己构建http请求。然后就像是调用自身工程的方法调用，而感觉不到是调用远程方法，使得编写客户端变得非常容易

Feign中本身已经集成了Ribbon依赖和自动配置，因此我们不需要额外引入依赖，也不需要再注册 RestTemplate 对象。

基于 Feign、Hystrix 的服务熔断

Feign、Hystrix 的服务熔断



有大量的客户请求请求a服务,同时a服务需要调用b服务,b服务又同时调用了c和d服务, 这种情况叫做扇出, 那么现在, 假设c服务 相应时间过久或者根本不可用了, 大量的客户请求还继续去请求a服务, 导致大量的服务器资源都被这部分服务所抢占, 可能还会危其他服务的运行, 轻者可能是导致其他服务也出现问题, 重则可能导致整个系统面临崩盘,也就是服务雪崩, 几乎所有的服务都变得不可用了, 这是最惨的情况.

那么, 为了防止这种情况的发生, 我们介入Hystrix的服务熔断机制, 可以理解为保险丝, 一旦这个电路网络出错, 那么我们就将出错的那个点下线, 同时给其上级的调用者返回一个备用的处理, 从而让继续进来的请求不至于全部卡死, 导致服务器资源紧张.

服务降级 : 举个例子,比如说双十一了, 目前淘宝的主要三个服务分别是, 下单\物流\售后, 因为通常来说访问下单服务的人是肯定比售后和物流的人多得多的, 因此在双十一那样的高并发下, 我们为了让下单服务更加通畅, 可以暂时将物流和售后服务进行降级, 以便让更多系统资源被分配到下单的服务中去, 那么物流和售后服务就相当于遭受了服务降级.

因为服务降级是基于客户端的, 也就是消费者端, 因此我们要在消费者的部分去配置, 在消费者的Service层, 我们记住需要编写一个自定义FallBackFactory类,需要实现FallbackFactory接口

原文链接：https://blog.csdn.net/LeBronJJJ/article/details/115056372

基于 Gatway 的 API 网关

API 网关=Gatway

API网关是一个服务器，是系统的唯一入口。从面向对象设计的角度看，它与外观模式类似。API网关封装了系统内部架构，为每个客户端提供一个定制的API。它可能还具有其它职责，如身份验证、监控、负载均衡、缓存、请求分片与管理、静态响应处理。
API网关方式的核心要点是，所有的客户端和消费端都通过统一的网关接入微服务，在网关层处理所有的非业务功能。通常，网关也是提供REST/HTTP的访问API。服务端通过API-GW注册和管理服务。

SpringCloud Gateway 作为 Spring Cloud 生态系统中的网关，目标是替代 Zuul，在Spring Cloud 2.0以上版本中，没有对新版本的Zuul 2.0以上最新高性能版本进行集成，仍然还是使用的Zuul 2.0之前的非Reactor模式的老版本。而为了提升网关的性能，SpringCloud Gateway是基于WebFlux框架实现的，而WebFlux框架底层则使用了高性能的Reactor模式通信框架Netty。

Spring Cloud Gateway 的目标，不仅提供统一的路由方式，并且基于 Filter 链的方式提供了网关基本的功能，例如：安全，监控/指标，和限流。

Config 配置中心

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。使用Config Server，您可以在所有环境中管理应用程序的外部属性。客户端和服务器上的概念映射与Spring `Environment`和`PropertySource`抽象相同，因此它们与Spring应用程序非常契合，但可以与任何以任何语言运行的应用程序一起使用。随着应用程序通过从开发人员到测试和生产的部署流程，您可以管理这些环境之间的配置，并确定应用程序具有迁移时需要运行的一切。服务器存储后端的默认实现使用git，因此它轻松支持标签版本的配置环境，以及可以访问用于管理内容的各种工具。可以轻松添加替代实现，并使用Spring配置将其插入。

Spring Boot 项目，除了引入相应的 maven 包之外，剩下的工作就是完善配置文件了，例如 mysql、redis 、security 相关的配置。除了项目运行的基础配置之外，还有一些配置是与我们业务有关系的，比如说七牛存储、短信相关、邮件相关，或者一些业务上的开关。

对于一些简单的项目来说，我们一般都是直接把相关配置放在单独的配置文件中，以 properties 或者 yml 的格式出现，更省事儿的方式是直接放到 application.properties 或 application.yml 中。但是这样的方式有个明显的问题，那就是，当修改了配置之后，必须重启服务，否则配置无法生效。

https://www.cnblogs.com/fengzheng/p/11242128.html

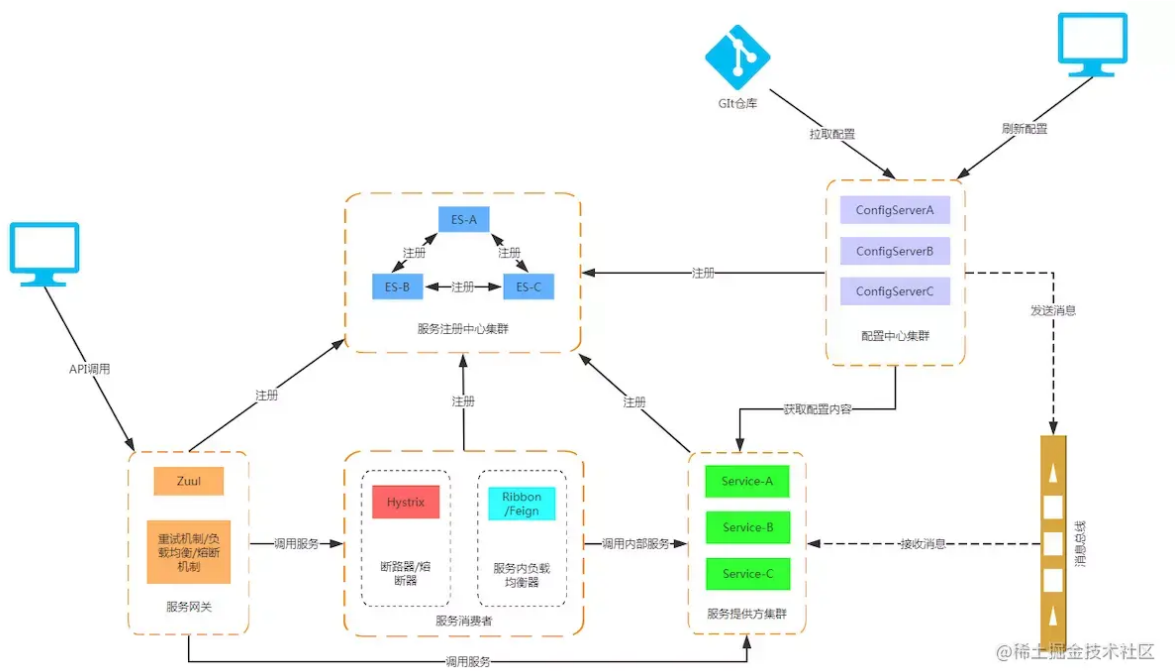
实现自动刷新

基于 Bus 的配置刷新

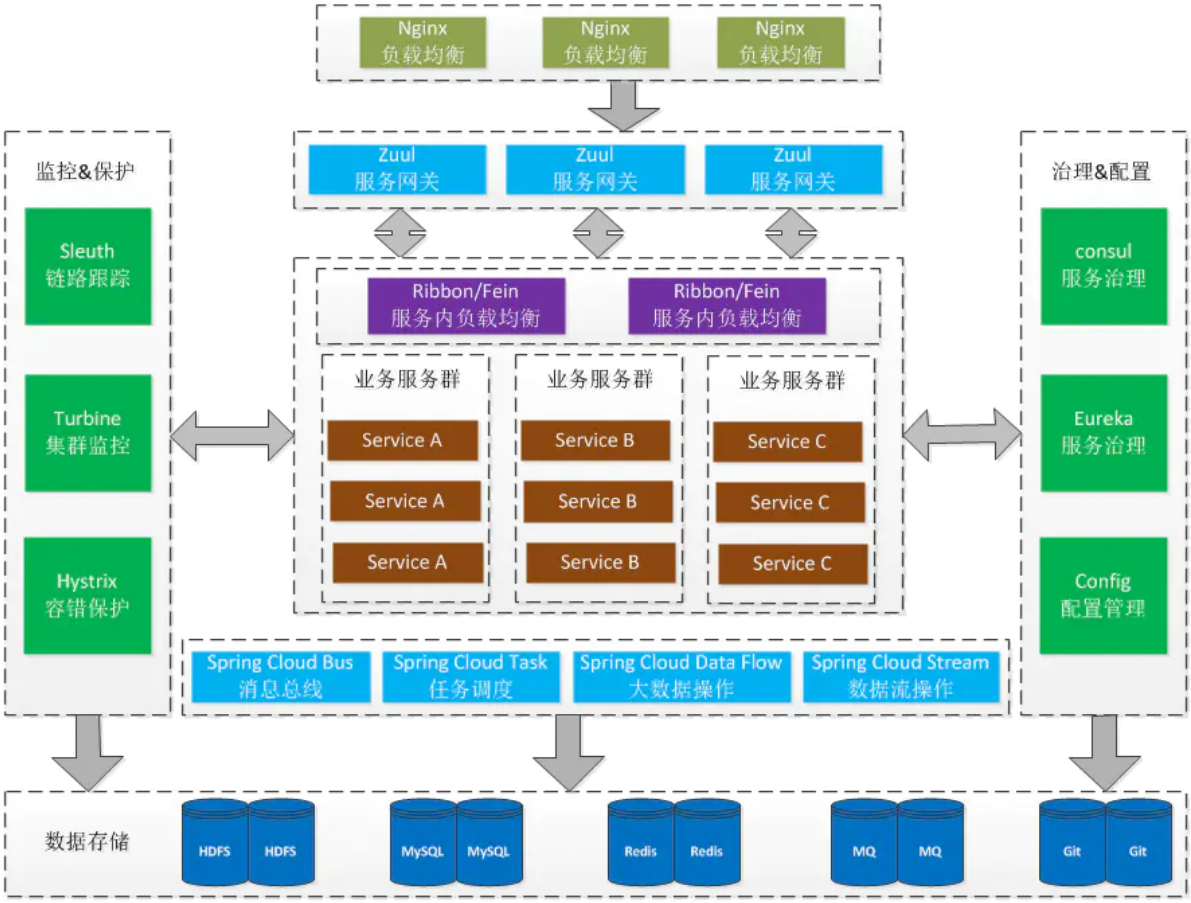
Spring Cloud Bus将分布式系统的节点与轻量级消息代理链接。这可以用于广播状态更改（例如配置更改）或其他管理指令。一个关键的想法是，Bus就像一个扩展的Spring Boot应用程序的分布式执行器，但也可以用作应用程序之间的通信渠道。当前唯一的实现是使用AMQP代理作为传输，但是相同的基本功能集（还有一些取决于传输）在其他传输的路线图上。

如果有几十个微服务,而每一个服务又是多实例,当更改配置时,需要重新启动多个微服务实例,会非常麻烦。 Spring Cloud Bus的一个功能就是让这个过程变得简单,当远程Git仓库的配置更改后,只需要向某一个微服务实例发送一个Post请求,通过消息组件通知其他微
服务实例重新拉取配置文件。

原文链接：https://blog.csdn.net/x7536987/article/details/106163392



https://juejin.cn/post/6844904103974273038



https://www.jianshu.com/p/cfbda198d15a