# 什么是微服务？

微服务架构是一种架构模式或者说是一种架构风格，它提倡将单一应用程序划分为一组小的服务，每个服务运行在其独立的自己的进程中，服务之间相互协调、互相配合，为用户提供最终价值。服务之间采用轻量级的通信机制互相沟通（通常是基于HTTP的RESTful API）,每个服务都围绕着具体的业务进行构建，并且能够被独立的构建在生产环境、类生产环境等。另外，应避免统一的、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，可以有一个非常轻量级的集中式管理来协调这些服务，可以使用不同的语言来编写服务，也可以使用不同的数据存储。

# 什么是springboot

        Spring Boot 是 Spring 开源组织下的子项目，是 Spring 组件一站式解决方案，主要是简化了使用 Spring 的难度，简省了繁重的配置，提供了各种启动器，开发者能快速上手。

# Spring Boot有哪些优点？

配置简单了

1.springboot内嵌了servlet容器，降低了对环境的要求，机器有java运行环境，可以将项目打包成jar包，通过java命令 java -jar \*\*\*\*.jar 来执行。

2.快速整合第三方框架，无需配置文件

3.解决了Spring的弊端

4.代码少了、配置文件少了、不需要对第三方框架烦恼了、项目精简了，对整个团队的开发及维护来说，更大的节约了成本。

5.使用Java或Groovy开发基于Spring的应用程序非常容易。

6.它减少了大量的开发时间并提高了生产力。

7.它避免了编写大量的样板代码，注释和XML配置。

8.Spring Boot应用程序与其Spring生态系统(如Spring JDBC，Spring ORM，Spring Data，Spring Security等)集成非常容易。

它遵循“自用默认配置”方法，以减少开发工作量。

它提供嵌入式HTTP服务器，如Tomcat，Jetty等，以开发和测试Web应用程序非常容易。

它提供CLI(命令行界面)工具从命令提示符，非常容易和快速地开发和测试Spring Boot(Java或Groovy)应用程序。

它提供了许多插件来开发和测试Spring启动应用程序非常容易使用构建工具，如Maven和Gradle。

它提供了许多插件，以便与嵌入式和内存数据库工作非常容易。

# ****什么是Spring Cloud？****

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 Cloud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。

# ****SpringBoot和SpringCloud,请你谈谈对他们的理解？****

1）、SpringBoot专注于快速方便的开发单个个体微服务。

    2）、SpringCloud是关注全局的微服务协调、整理、治理的框架，它将SpringBoot开发的单体整合并管理起来。

    3）、SpringBoot可以离开SpringCloud独立使用开发项目，但是SpringCloud离不开SpringBoot，属于依赖关系。

# 微服务技术栈有哪些?

微服务条目 落地的技术

服务开发 SpringBoot、Spring、SpringMVC

服务配置管理 Netfilx公司的Archaius、阿里的Diamond等

服务注册与发现 Eureka、Consul、Zookeeper

服务调用 RPC、Rest、gRPC

服务熔断器 Hystrix、Envoy等

负载均衡 Nginx、Ribbon

服务接口调用（客户端调用服务的简化工具） Feign

消息队列 Kafka、RabbitMQ、ActiveMQ等

服务配置中心配置管理 SpringCloudConfig、Chef等

服务路由（API网关） Zuul

服务监控 Zabbix、Naggios、Metrics、Spectator等

全链路追踪 Zipkin、Brave、Dapper等

服务部署 Docker、OpenStack、Kubernetes等

数据流操作开发包 SpringCloud Stream

事件消息总线 Spring Cloud Bus

# Spring Cloud全家桶组件

在介绍Spring Cloud 全家桶之前，首先要介绍一下Netflix ，Netflix 是一个很伟大的公司，在Spring Cloud项目中占着重要的作用，Netflix 公司提供了包括Eureka、Hystrix、Zuul、Archaius等在内的很多组件，在微服务架构中至关重要，Spring在Netflix 的基础上，封装了一系列的组件，命名为：Spring Cloud Eureka、Spring Cloud Hystrix、Spring Cloud Zuul等，下边对各个组件进行分别得介绍：

（1）Spring Cloud Eureka

我们使用微服务，微服务的本质还是各种API接口的调用，那么我们怎么产生这些接口、产生了这些接口之后如何进行调用那？如何进行管理哪？

答案就是Spring Cloud Eureka，我们可以将自己定义的API 接口注册到Spring Cloud Eureka上，Eureka负责服务的注册于发现，如果学习过Zookeeper的话，就可以很好的理解，Eureka的角色和 Zookeeper的角色差不多，都是服务的注册和发现，构成Eureka体系的包括：服务注册中心、服务提供者、服务消费者。

（2）Spring Cloud Ribbon

在上Spring Cloud Eureka描述了服务如何进行注册，注册到哪里，服务消费者如何获取服务生产者的服务信息，但是Eureka只是维护了服务生产者、注册中心、服务消费者三者之间的关系，真正的服务消费者调用服务生产者提供的数据是通过Spring Cloud Ribbon来实现的。

在（1）中提到了服务消费者是将服务从注册中心获取服务生产者的服务列表并维护在本地的，这种客户端发现模式的方式是服务消费者选择合适的节点进行访问服务生产者提供的数据，这种选择合适节点的过程就是Spring Cloud Ribbon完成的。

Spring Cloud Ribbon客户端负载均衡器由此而来。

（3）Spring Cloud Feign

上述（1）、（2）中我们已经使用最简单的方式实现了服务的注册发现和服务的调用操作，如果具体的使用Ribbon调用服务的话，你就可以感受到使用Ribbon的方式还是有一些复杂，因此Spring Cloud Feign应运而生。

Spring Cloud 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，使用Feign 创建一个接口并对它进行注解，它具有可插拔的注解支持包括Feign注解与JAX-RS注解，Feign还支持可插拔的编码器与解码器，Spring Cloud 增加了对 Spring MVC的注解，Spring Web 默认使用了HttpMessageConverters, Spring Cloud 集成 Ribbon 和 Eureka 提供的负载均衡的HTTP客户端 Feign。

简单的可以理解为：Spring Cloud Feign 的出现使得Eureka和Ribbon的使用更为简单。

（4）Spring Cloud Hystrix

我们在（1）、（2）、（3）中知道了使用Eureka进行服务的注册和发现，使用Ribbon实现服务的负载均衡调用，还知道了使用Feign可以简化我们的编码。但是，这些还不足以实现一个高可用的微服务架构。

例如：当有一个服务出现了故障，而服务的调用方不知道服务出现故障，若此时调用放的请求不断的增加，最后就会等待出现故障的依赖方 相应形成任务的积压，最终导致自身服务的瘫痪。

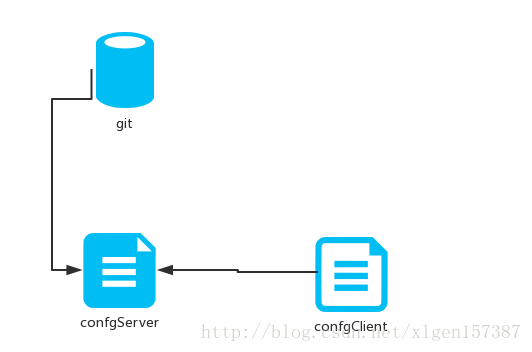
Spring Cloud Hystrix正是为了解决这种情况的，防止对某一故障服务持续进行访问。Hystrix的含义是：断路器，断路器本身是一种开关装置，用于我们家庭的电路保护，防止电流的过载，当线路中有电器发生短路的时候，断路器能够及时切换故障的电器，防止发生过载、发热甚至起火等严重后果。

（5）Spring Cloud Config

对于微服务还不是很多的时候，各种服务的配置管理起来还相对简单，但是当成百上千的微服务节点起来的时候，服务配置的管理变得会复杂起来。

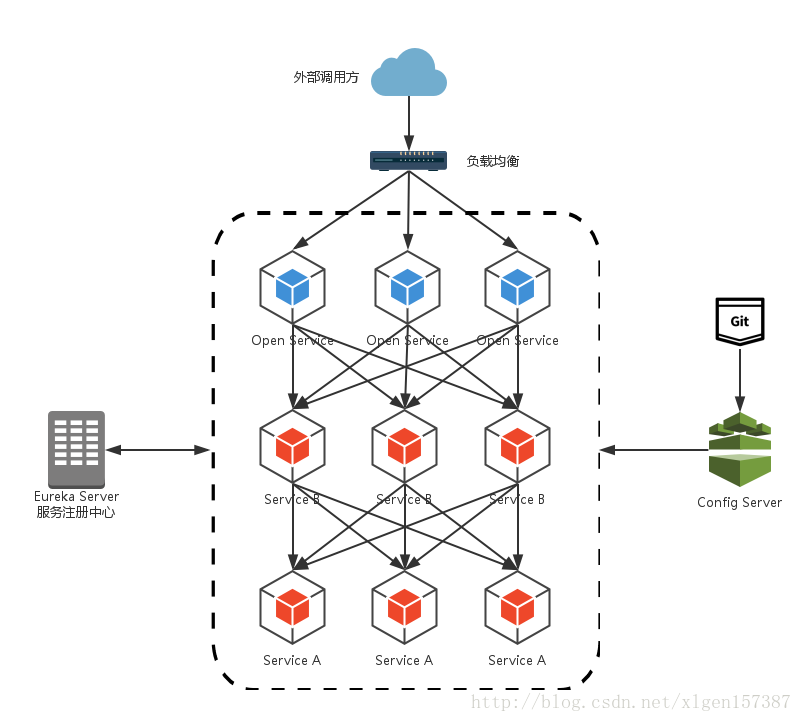
分布式系统中，由于服务数量巨多，为了方便服务配置文件统一管理，实时更新，所以需要分布式配置中心组件。在Spring Cloud中，有分布式配置中心组件Spring Cloud Config ，它支持配置服务放在配置服务的内存中（即本地），也支持放在远程Git仓库中。在Cpring Cloud Config 组件中，分两个角色，一是Config Server，二是Config Client。

Config Server用于配置属性的存储，存储的位置可以为Git仓库、SVN仓库、本地文件等，Config Client用于服务属性的读取。



（6）Spring Cloud Zuul

我们使用Spring Cloud Netflix中的Eureka实现了服务注册中心以及服务注册与发现；而服务间通过Ribbon或Feign实现服务的消费以及均衡负载；通过Spring Cloud Config实现了应用多环境的外部化配置以及版本管理。为了使得服务集群更为健壮，使用Hystrix的融断机制来避免在微服务架构中个别服务出现异常时引起的故障蔓延。



先来说说这样架构需要做的一些事儿以及存在的不足：

1、首先，破坏了服务无状态特点。为了保证对外服务的安全性，我们需要实现对服务访问的权限控制，而开放服务的权限控制机制将会贯穿并污染整个开放服务的业务逻辑，这会带来的最直接问题是，破坏了服务集群中REST API无状态的特点。从具体开发和测试的角度来说，在工作中除了要考虑实际的业务逻辑之外，还需要额外可续对接口访问的控制处理。

2、其次，无法直接复用既有接口。当我们需要对一个即有的集群内访问接口，实现外部服务访问时，我们不得不通过在原有接口上增加校验逻辑，或增加一个代理调用来实现权限控制，无法直接复用原有的接口。

面对类似上面的问题，我们要如何解决呢？下面进入本文的正题：服务网关！

为了解决上面这些问题，我们需要将权限控制这样的东西从我们的服务单元中抽离出去，而最适合这些逻辑的地方就是处于对外访问最前端的地方，我们需要一个更强大一些的均衡负载器，它就是本文将来介绍的：服务网关。

服务网关是微服务架构中一个不可或缺的部分。通过服务网关统一向外系统提供REST API的过程中，除了具备服务路由、均衡负载功能之外，它还具备了权限控制等功能。Spring Cloud Netflix中的Zuul就担任了这样的一个角色，为微服务架构提供了前门保护的作用，同时将权限控制这些较重的非业务逻辑内容迁移到服务路由层面，使得服务集群主体能够具备更高的可复用性和可测试性。

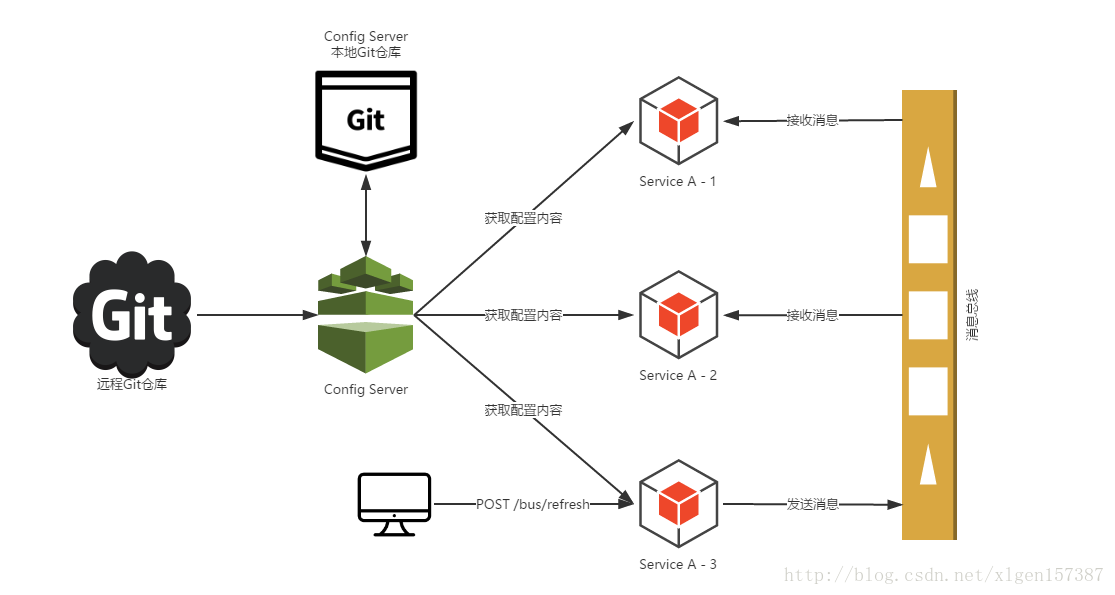
（7）Spring Cloud Bus

在（5）Spring Cloud Config中，我们知道的配置文件可以通过Config Server存储到Git等地方，通过Config Client进行读取，但是我们的配置文件不可能是一直不变的，当我们的配置文件放生变化的时候如何进行更新哪？

一种最简单的方式重新一下Config Client进行重新获取，但Spring Cloud绝对不会让你这样做的，Spring Cloud Bus正是提供一种操作使得我们在不关闭服务的情况下更新我们的配置。

Spring Cloud Bus官方意义：消息总线。

当然动态更新服务配置只是消息总线的一个用处，还有很多其他用处。



# 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？

Zookeeper保证了CP（C：一致性，P：分区容错性），Eureka保证了AP（A：高可用）

1.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信息，但不能容忍直接down掉不可用。也就是说，服务注册功能对高可用性要求比较高，但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，剩余节点会重新选leader。问题在于，选取leader时间过长，30 ~ 120s，且选取期间zk集群都不可用，这样就会导致选取期间注册服务瘫痪。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，虽然服务能够恢复，但是漫长的选取时间导致的注册长期不可用是不能容忍的。

2.Eureka保证了可用性，Eureka各个节点是平等的，几个节点挂掉不会影响正常节点的工作，剩余的节点仍然可以提供注册和查询服务。而Eureka的客户端向某个Eureka注册或发现时发生连接失败，则会自动切换到其他节点，只要有一台Eureka还在，就能保证注册服务可用，只是查到的信息可能不是最新的。除此之外，Eureka还有自我保护机制，如果在15分钟内超过85%的节点没有正常的心跳，那么Eureka就认为客户端与注册中心发生了网络故障，此时会出现以下几种情况：

①、Eureka不在从注册列表中移除因为长时间没有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他节点上（即保证当前节点仍然可用）

③、当网络稳定时，当前实例新的注册信息会被同步到其他节点。

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那样使整个微服务瘫痪。

# **Spring Boot 的核心配置文件有哪几个？它们的区别是什么？**

  Spring Boot 的核心配置文件是 application 和 bootstrap 配置文件。

application 配置文件这个容易理解，主要用于 Spring Boot 项目的自动化配置。

bootstrap 配置文件有以下几个应用场景。

 ●  使用 Spring Cloud Config 配置中心时，这时需要在 bootstrap 配置文件中添加连接到配置中心的配置属性来加载外部配置中心的配置信息；  
 ●  一些固定的不能被覆盖的属性；  
 ●  一些加密/解密的场景；

# **Spring Boot 的配置文件有哪几种格式？它们有什么区别？**

.properties 和 .yml，它们的区别主要是书写格式不同。

1).properties

app.user.name = javastack

2).yml

app:

  user:

    name: javastack

另外，.yml 格式不支持 @PropertySource 注解导入配置。

# **Spring Boot 的核心注解是哪个？它主要由哪几个注解组成的？**

启动类上面的注解是@SpringBootApplication，它也是 Spring Boot 的核心注解，主要组合包含了以下 3 个注解：

@SpringBootConfiguration：组合了 @Configuration 注解，实现配置文件的功能。

@EnableAutoConfiguration：打开自动配置的功能，也可以关闭某个自动配置的选项，如关闭数据源自动配置功能： @SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })。

@ComponentScan：Spring组件扫描。

# **Spring Boot 需要独立的容器运行吗？**

可以不需要，内置了 Tomcat/ Jetty 等容器。

# **SpringBoot 实现热部署有哪几种方式？**

主要有两种方式：

 ●  Spring Loaded  
 ●  Spring-boot-devtools

# **Spring Boot 可以兼容老 Spring 项目吗，如何做？**

可以兼容，使用 @ImportResource 注解导入老 Spring 项目配置文件。

# **保护 Spring Boot 应用有哪些方法？**

●  在生产中使用HTTPS  
 ●  使用Snyk检查你的依赖关系  
 ●  升级到最新版本  
 ●  启用CSRF保护  
 ●  使用内容安全策略防止XSS攻击

# springboot常用的starter有哪些

        spring-boot-starter-web 嵌入tomcat和web开发需要servlet与jsp支持

        spring-boot-starter-data-jpa 数据库支持

        spring-boot-starter-data-redis redis数据库支持

        spring-boot-starter-data-solr solr支持

        mybatis-spring-boot-starter 第三方的mybatis集成starter

# springboot自动配置的原理

        在spring程序main方法中 添加@SpringBootApplication或者@EnableAutoConfiguration

        会自动去maven中读取每个starter中的spring.factories文件  该文件里配置了所有需要被创建spring容器中的bean

# springboot集成mybatis的过程

        添加mybatis的starter maven依赖

                <dependency>

                        <groupId>org.mybatis.spring.boot</groupId>

                        <artifactId>mybatis-spring-boot-starter</artifactId>

                        <version>1.2.0</version>

                </dependency>

        在mybatis的接口中 添加@Mapper注解

        在application.yml配置数据源信息

# springboot如何添加【修改代码】自动重启功能

        添加开发者工具集=====spring-boot-devtools

# springcloud如何实现服务的注册和发现

        服务在发布时 指定对应的服务名（服务名包括了IP地址和端口） 将服务注册到注册中心（eureka或者zookeeper）

        这一过程是springcloud自动实现 只需要在main方法添加@EnableDisscoveryClient  同一个服务修改端口就可以启动多个实例

        调用方法：传递服务名称通过注册中心获取所有的可用实例 通过负载均衡策略调用（ribbon和feign）对应的服务

# 什么是服务熔断？什么是服务降级？

熔断机制是应对雪崩效应的一种微服务链路保护机制。当某个微服务不可用或者响应时间太长时，会进行服务降级，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秒内调用20次，如果失败，就会启动熔断机制。

服务降级，一般是从整体负荷考虑。就是当某个服务熔断之后，服务器将不再被调用，此时客户端可以自己准备一个本地的fallback回调，返回一个缺省值。这样做，虽然水平下降，但好歹可用，比直接挂掉强。

# ribbon和feign区别

        Ribbon添加maven依赖 spring-starter-ribbon 使用@RibbonClient(value="服务名称") 使用RestTemplate调用远程服务对应的方法

        feign添加maven依赖 spring-starter-feign 服务提供方提供对外接口 调用方使用 在接口上使用@FeignClient("指定服务名")

Ribbon和Feign的区别：

        Ribbon和Feign都是用于调用其他服务的，不过方式不同。

        1.启动类使用的注解不同，Ribbon用的是@RibbonClient，Feign用的是@EnableFeignClients。

        2.服务的指定位置不同，Ribbon是在@RibbonClient注解上声明，Feign则是在定义抽象方法的接口中使用@FeignClient声明。

        3.调用方式不同，Ribbon需要自己构建http请求，模拟http请求然后使用RestTemplate发送给其他服务，步骤相当繁琐。

        Feign则是在Ribbon的基础上进行了一次改进，采用接口的方式，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。不过要注意的是抽象方法的注解、方法签名要和提供服务的方法完全一致。

# springcloud断路器的作用

        当一个服务调用另一个服务由于网络原因或者自身原因出现问题时 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时

                导致更多的请求等待 这样就会发生连锁效应（雪崩效应） 断路器就是解决这一问题

                断路器有完全打开状态

                        一定时间内 达到一定的次数无法调用 并且多次检测没有恢复的迹象 断路器完全打开，那么下次请求就不会请求到该服务

                半开

                        短时间内 有恢复迹象 断路器会将部分请求发给该服务 当能正常调用时 断路器关闭

                关闭

                        当服务一直处于正常状态 能正常调用 断路器关闭

# 如何重新加载Spring Boot上的更改，而无需重新启动服务器？

这可以使用DEV工具来实现。通过这种依赖关系，您可以节省任何更改，嵌入式tomcat将重新启动。Spring Boot有一个开发工具（DevTools）模块，它有助于提高开发人员的生产力。Java开发人员面临的一个主要挑战是将文件更改自动部署到服务器并自动重启服务器。开发人员可以重新加载Spring Boot上的更改，而无需重新启动服务器。这将消除每次手动部署更改的需要。Spring Boot在发布它的第一个版本时没有这个功能。这是开发人员最需要的功能。DevTools模块完全满足开发人员的需求。该模块将在生产环境中被禁用。它还提供H2数据库控制台以更好地测试应用程序。

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

# Spring Boot中的[监视器](https://www.baidu.com/s?wd=%E7%9B%91%E8%A7%86%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)是什么？

Spring boot actuator是spring启动框架中的重要功能之一。Spring boot监视器可帮助您访问生产环境中正在运行的应用程序的当前状态。有几个指标必须在生产环境中进行检查和监控。即使一些外部应用程序可能正在使用这些服务来向相关人员触发警报消息。监视器模块公开了一组可直接作为HTTP URL访问的REST端点来检查状态。

# 如何在自定义端口上运行Spring Boot应用程序？

为了在自定义端口上运行Spring Boot应用程序，您可以在application.properties中指定端口。

server.port = 8090

# ****Spring Cloud解决了哪些问题？****

在使用Spring Boot开发分布式微服务时，我们面临的问题很少由Spring Cloud解决。

* **与分布式系统相关的复杂性 –**包括网络问题，延迟开销，带宽问题，安全问题。
* **处理服务发现的能力 –**服务发现允许集群中的进程和服务找到彼此并进行通信。
* **解决冗余问题 –**冗余问题经常发生在分布式系统中。
* **负载平衡  –**改进跨多个计算资源（例如计算机集群，网络链接，中央处理单元）的工作负载分布。
* **减少性能问题 –**减少因各种操作开销导致的性能问题。