Spring Cloud 学习总结

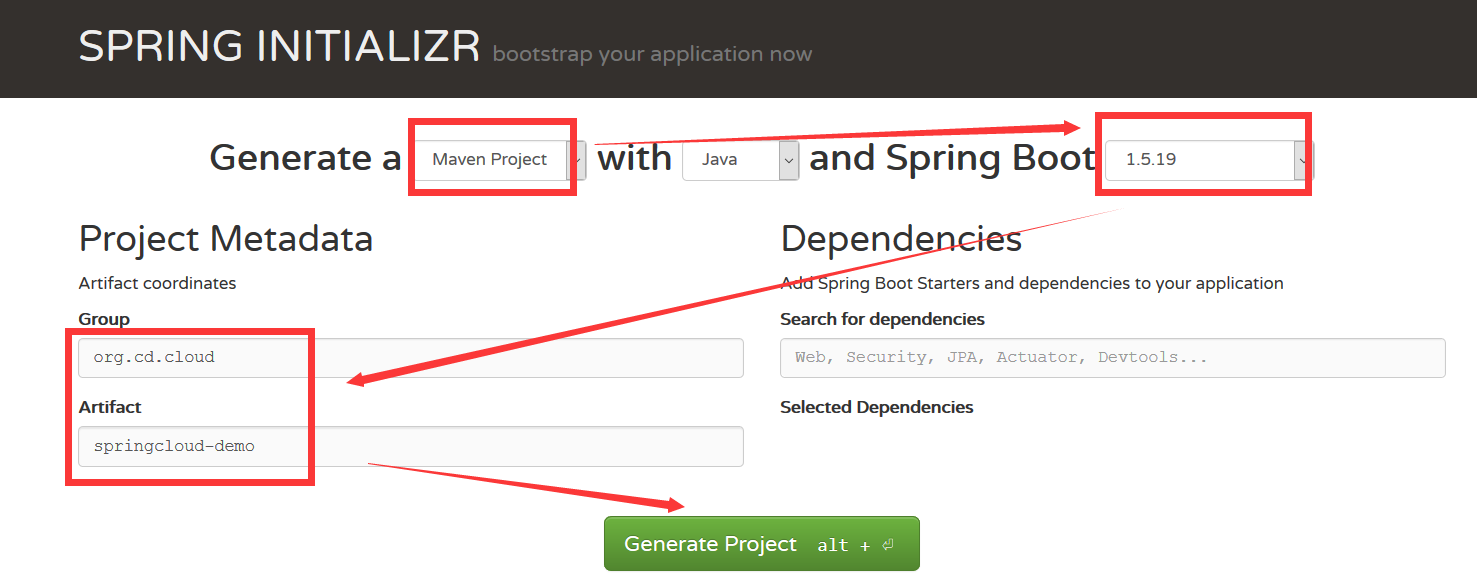
# Spring Cloud项目搭建

1. 预先需要准备的工具/环境：

Intellij IDEA, Maven(安装请自行百度)

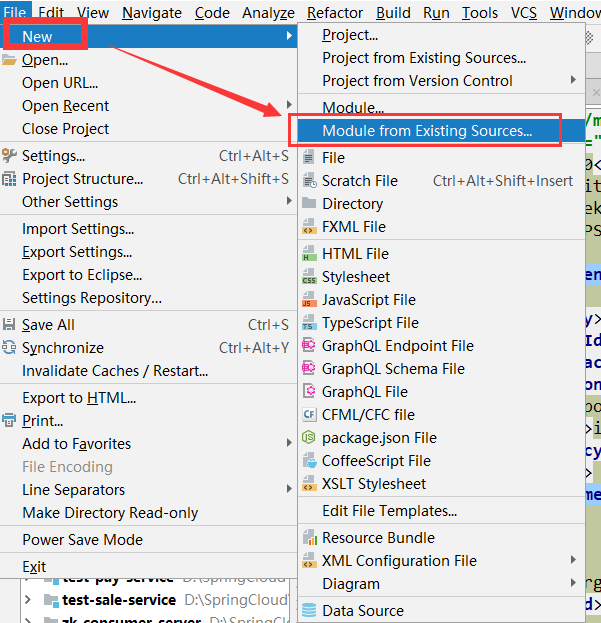
1. 到Spring的官网生成Spring Boot项目（Spring Cloud是基于Spring Boot框架）：

<https://start.spring.io/>



注意：这个截图只是示例，你可以选择生成一个Maven或Gradle项目，也可以选择Spring Boot的版本，剩下的就是填一下group 和 artifact信息，group的写法一般为com/org+公司名+项目名 （比如：org.cd.cloud）,artifact 填模块名，比如你这是要创建一个eureka的模块，就可以填myeureka-server。

生成完后，下载zip包，解压包，然后用IntelliJ IDEA或Eclipse导入。



导入之后，我们需要修改下pom.xml.

1. 如果你发现POM中有Spring Boot的依赖，可以暂时去掉，我们如果只是跑跑Spring Cloud的hello word，不需要这些：

*<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter</artifactId>  
</dependency>*

*<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
</dependency>*

*<build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
</build>*

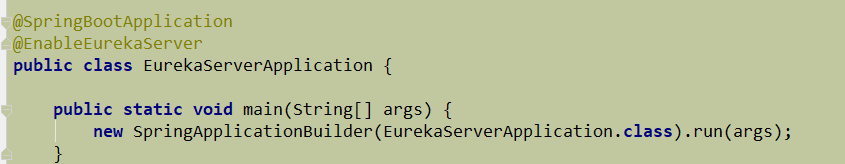
1. 手动为Spring Cloud项目添加Dependencies （我演示的Spring Cloud版本是Dalston.SR1）：

<**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-dependencies</**artifactId**>  
 <**version**>Dalston.SR1</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 </**dependencies**>  
</**dependencyManagement**>

1. 添加了Spring Cloud的DependencyManagement之后(这里面的依赖并不会生效)，我们还需要选择性的将需要的依赖引用进来，比如如果需要搭建一个Eureka Server，那么我们就要将该依赖添加到dependency：

<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-eureka-server</**artifactId**>  
 </**dependency**>  
</**dependencies**>

1. 进入java类文件目录： src/main/java -> org.cd.cloud, 会发现根目录有一个XXXApplication.java结尾的类，这是Spring Boot的启动类，我们在这个类做一点小改进，就是在类名上面加一个@EnableEurekaServer注解：



1. 进入资源文件目录： src/main/resource 你会发现有一个application.properties文件（或application.py），如果没有这个文件，则创建一个。这里面是Spring项目需要的一些配置信息，比如我们要指定端口和应用名称，则添加以下配置：

server.port=8761

spring.application.name=eureka-server

1. 运行XXXApplication.java 类，就可以启动Spring Cloud项目啦，启动之后会发现有点报错，是因为Eureka会进行自我注册，这个暂时可以忽略，下节我们会详细讲解。
2. 浏览器输入：<http://localhost:8761/> 便可以访问了。

是不是很简单？ 还不赶紧尝试一下！

# 微服务的发布与调用-Eureka

上节讲了Spring Cloud项目的搭建，这节一起来看看Spring Cloud微服务的发布与调用。本节后面还会演示Eureka的集群。

内容1：什么是Eureka？ （概念有的是摘抄《疯狂Spring Cloud 微服务架构与实战》一书）

Spring Cloud 集成了Netflix OSS的多个项目，不得不说Spring对第三方、优秀的项目真是无缝集成。 Netflix在Spring Cloud中的代号叫： spring-cloud-netflix. 这个里面包含多个子模块，这些子模块对Netflix下面的框架进行了封装，很方便我们使用。这节所演示的Eureka就是Netflix下面很重要的一个框架。

1. Eureka **提供基于REST的服务**，在集群中主要用于服务管理
2. Eureka **提供基于java语言的客户端组件**，**客户端组件实现了负载均衡**的功能
3. 可以将业务组件（service provider， service invoker…）注册到Eureka容器中，这些组件可进行集群部署，Eureka主要维护这些**服务的列表**，并自动**检查他们的状态**。

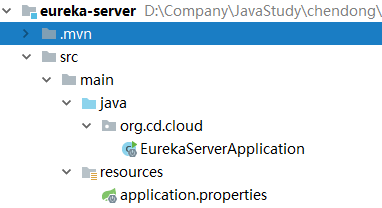
一旦微服务注册到Eureka中，我们就可以通过非常简单的方式调用了（http://{server-name}/xxxx）,并且可以通过Feign客户端来优雅地实现远程service调用。

内容2：一个简单的通过Eureka发布与调用微服务

废话不多说，开干！

**创建Eureka Server：**

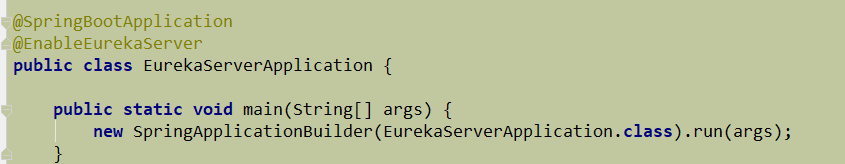
1. 创建一个Spring Cloud项目，叫eureka-server, 目录结构如图所示（我演示的项目是通过maven构建的，你也可以选择gradle）：



2）在pom.xml中加入以下依赖：

<**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-dependencies</**artifactId**>  
 <**version**>Dalston.SR1</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 </**dependencies**>  
</**dependencyManagement**>  
  
<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-eureka-server</**artifactId**>  
 </**dependency**>  
</**dependencies**>

3) 进入java类文件目录： src/main/java -> org.cd.cloud, 会发现根目录有一个XXXApplication.java结尾的类，这是Spring Boot的启动类，我们在这个类做一点小改进，就是在类名上面加一个@EnableEurekaServer注解：



4）进入资源文件目录： src/main/resource 你会发现有一个application.properties文件（或application.py），如果没有这个文件，则创建一个。这里面是Spring项目需要的一些配置信息，比如我们要指定端口和应用名称，则添加以下配置：

server.port=8761

spring.application.name=eureka-server

5）运行XXXApplication.java 类，就可以启动Spring Cloud项目啦，启动之后会发现有点报错（Connection refused: connect），是因为Eureka会进行自我注册，这个请看下面解说。浏览器输入：http://localhost:8761/ 便可以访问了。

6）Connection refused原因：在服务器启动时，Eureka Server会把自己当做一个客户端去注册Eureka，并且会去抓取信息，但是它本事只是一个服务器 并不是客户端，这样就冲突了，所以会refuse，需要在application.properties加入以下两个配置：

**eureka.client.register-with-eureka**=**false  
eureka.client.fetch-registry**=**false**

这样再启动的时候就不会报错啦！

**创建Eureka Client 1（Service Provider）**

1. 创建一个叫first-service-provider模块（直接拷贝eureka-server的代码都行，简单粗暴高效），pom.xml里面我们要做以下改变：

把上一步Eureka Server pom.xml中的eureka-server的dependency可以去掉，换成下面的：（这个dependency是Eureka 客户端）

<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-eureka</**artifactId**>  
 </**dependency**>  
</**dependencies**>

1. 在项目启动类里面加入下面代码：

类名上面加入@EnableEurekaClient注解，这个表示这是一个Eureka客户端

加入@RestController注解，因为我们要写一个接口来测试

加入一个message接口，以便一会测试。



1. 在application.properties里面加入以下配置：

**spring.application.name**=**first-service-provider  
server.port**=**8080  
eureka.instance.hostname**=**localhost**

**eureka.client.service-url.default-zone**=**http://localhost:8761/eureka/**

最后一行表示注册到Eureka Server

1. 好了，大胆启动这个first-service-provider吧，应该不会有exception，如果有，相信高智商的你处理不在话下。启动成功后，可以再次访问<http://localhost:8761/> 看看有啥不一样 😊, 不出意外的话应该可以看到有个新的服务注册进来啦。

**创建Eureka Client 2（Service Invoker）**

1. 创建一个叫first-service-invoker模块（直接拷贝first-service-provider的代码都行，简单粗暴高效），pom.xml里面记住把artifactId等标识信息都改成first-service-invoker哈，只是一个ctrlCV战士基本素养。
2. 在项目启动类里面加入下面代码：

类名上面加入@EnableEurekaClient注解，这个表示这是一个Eureka客户端

加入@RestController注解，因为我们一会要通过浏览器来访问这个/test接口

将RestTemplate加入到Spring的容器中，并加上@LoadBanlanced注解，这个应该很容易理解，就是为了在使用restTemplate的时候不需要每次都new, 并且它还可以负载均衡。

在/test 接口里面，我们通过http://{service-name}/xxx 的方式调用已经注册到Eureka的微服务，是不是so方便，不需要指定域名端口号啥的了，调用起来也是相当快速。这就是通过Eureka来实现服务之间的发布与调用。



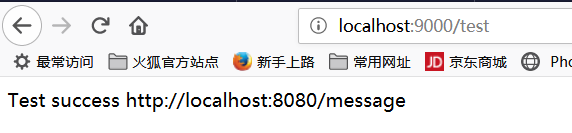
1. 在application.properties里面加入以下配置(改了port和applicationName)：

**spring.application.name**=**first-service-invoker   
server.port**=**9000  
eureka.instance.hostname**=**localhost**

**eureka.client.service-url.default-zone**=**http://localhost:8761/eureka/**

最后一行表示注册到Eureka Server

1. 启动first-service-invoker，然后可以看到<http://localhost:8761/>又有服务注册进来了。在浏览器访问<http://localhost:9000/test> (这个是刚才first-service-invoker里面的接口，我们通过浏览器访问invoker的接口，然后Invoker通过Eureka，很轻松的访问到了service provider里面的数据。最终可以在浏览器看到service provider 里面API返回的信息)



好了，光荣宣布，本小节结束，休息一会喝口水 广告之后更加精彩！接下来我们要体验Eureka更高级一点的技能！

内容3：Eureka集群搭建

为了方便演示集群，需要多个hostname，所以请先进入 C:\Windows\System32\drivers\etc 这个目录，找到hosts文件，然后加入以下内容：

**127.0.0.1 localhost localhost1 localhost2**

我们把本机指定几个别名。

**改进Eureka Server：**

1）pom.xml保持不变，注释掉application.properties里面的所有内容（没错, 废弃它），创建一个application-server1.properties 和 application-server2.properties.

2) 在application-server1.properties里面加入以下配置：

**server.port**=**8761  
spring.application.name**=**eureka-server  
eureka.instance.hostname**=**localhost1  
eureka.client.service-url.default-zone**=**http://localhost2:8762/eureka/**

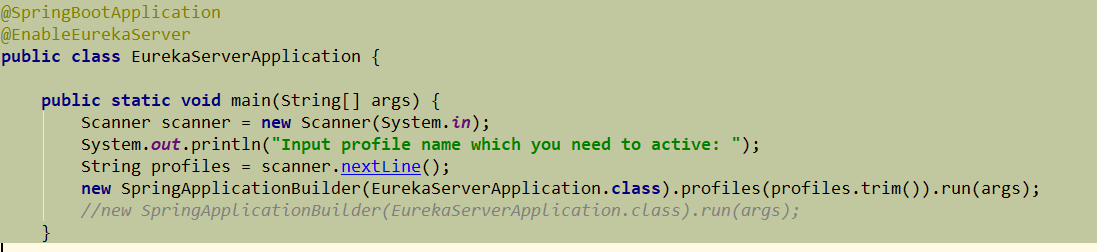
注意最后一行，我们一会会启动两个Eureka Server实例，这样的话就会有两个端口，localhost1:8761表示第一个Eureka Server，localhost2:8762表示第二个Eureka Server。

我们要将Eureka Server1 注册到Server2上，将Server2注册到Server1上，所以这里的service-url.default-zone是分别指向另一个Eureka Server。

在application-server2.properties里面加入以下配置：

**server.port**=**8762  
spring.application.name**=**eureka-server  
eureka.instance.hostname**=**localhost2  
eureka.client.service-url.default-zone**=**http://localhost1:8761/eureka/**

3）在启动类加入以下代码来手动确定激活哪个profile：



4）修改完成后，运行两次这个启动类，分别输入：server1 和 server2 来启动两个Eureka Server。 第一个在启动的时候会报错，这是正常现象，因为它要注册第二个Eureka Server，但此时第二个还未运行起来。

5）两次运行完成后，在浏览器访问<http://localhost1:8761/> 和 <http://localhost2:8762> 试试。

**改进Eureka Client 1（Service Provider）**

1. 修改first-service-provider -> application.properties里面的这个属性：

**eureka.client.service-url.default-zone**=**http://localhost:8761/eureka/, http://localhost:8762/eureka/**

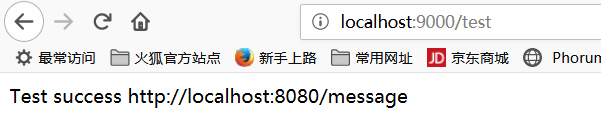
让这个微服务分别注册8761端口和8762端口的Eureka Server

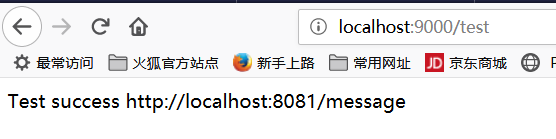
2）想办法启动这个微服务两次（端口号不能一样，否则会冲突）。聪明的你肯定会有各种办法，我在这里列举一个：可以再启动类里面加入Scanner，手动输入端口号，运行两次first-service-provider构建出来的jar文件， 比如第一次输入8080运行，第二次输入8081运行。为啥不能直接想上面那样在IDE里面运行两次启动类的main方法呢，原因貌似是标记过@EurekaClient的启动类只能运行一次，再次运行IDE会提示已经有相同的项目在运行 巴拉巴拉的（反正我的是这样）

**Eureka Client 1（Service Invoker）**

这个微服务不做改动

下面我们就来体验一下这个时候再访问http://localhost:9000/test接口会出现什么情况：





会发现/test接口里面请求service provide微服务里面的/message接口的时候，端口号不断的切换成8080和8081。 这是Eureka的负载均衡，会将请求分配到不同的微服务上，以缓解某个微服务的压力，体现集群的优势。

# 微服务的发布与调用-Eureka (下)

*NOTE：本博客演示的所有代码可以去github下载：*[*https://github.com/aharddreamer/chendong.git*](https://github.com/aharddreamer/chendong.git)

上节我们演示了简单的将微服务发布Eureka与相互之间的调用，还演示了Eureka的集群及负载均衡功能。这节我们要学习Eureka更高级一点的功能—微服务实例的健康自检。

在默认情况下，Eureka客户端（微服务提供者或调用者）会每隔30秒发送一次心跳给Eureka服务器，告诉服务器自己是否是存活状态，但是实际情况可能会这样：客户端出现了部分异常，服务已经不可用了，但是还在正常的向服务器发送心跳，因为很显然，即使某个接口不能用了整个微服务还并没有挂掉。比如数据库连接中断 或者依赖的上游系统挂了等等。

那么，我们就需要一个健康自检机制，更加智能的处理这种问题。刚好 Eureka提供了这么一种功能—健康检查控制器。

本节演示的项目结构 (没有建过项目的同学请看我上一节博客)：

eureka-server

first-service-provider

first-service-invoker

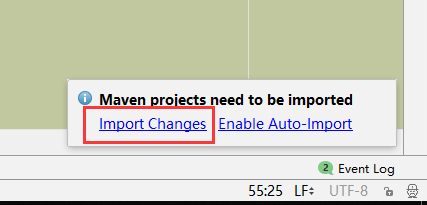
修改first-service-provider 项目的POM，我们要加入下面依赖：

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-actuator</**artifactId**>  
 <**version**>1.5.3.RELEASE</**version**>  
</**dependency**>

加好之后，启动first-service-provider试试。我在启动的时候报了一个jackson包未找到的错误，于是我又加入下面的依赖：

<**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-core</**artifactId**>  
 <**version**>2.9.8</**version**>  
</**dependency**>  
  
<**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-databind</**artifactId**>  
 <**version**>2.9.8</**version**>  
</**dependency**>

*PS: POM加完依赖之后，如果你用的IDE是Intellij IDEA，那么修改POM后右下角会出现一个弹出框，请点击Import Changes，这样依赖就会自动下载了。*

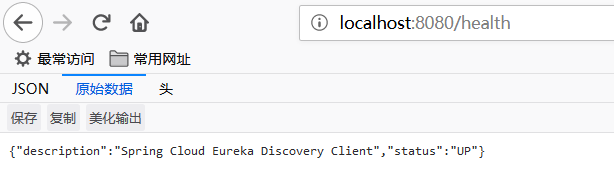


*如果是eclipse，修改完POM之后，请选中POM 右键->update maven project.*

依次启动eureka-server和first-service-provider.

启动完之后，在浏览器输入： <http://localhost:8080/health>

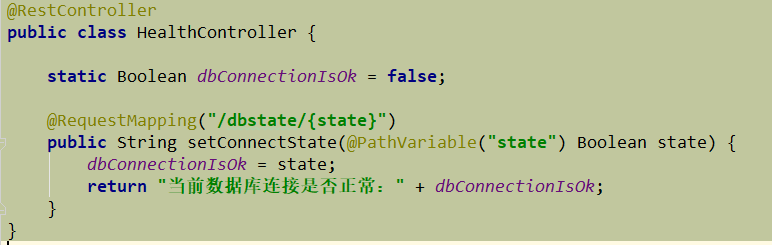
可以看到浏览器返回如下：



说明该REST服务向外展示当前应用的状态为UP （服务正常）。

接下来我们要试试actuator自带的健康自检功能。

在first-service-provider里面新建一个Controller，我们要在这里面写一个测试的接口：

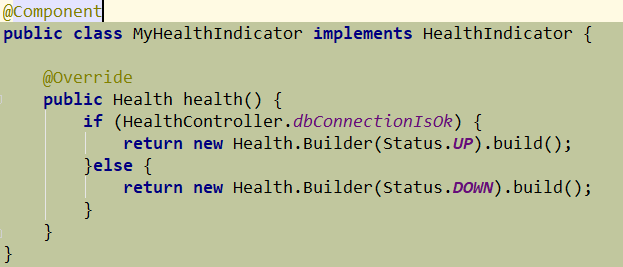


定义了一个变量，访问的时候通过控制这个变量来表示数据库的连接状态。

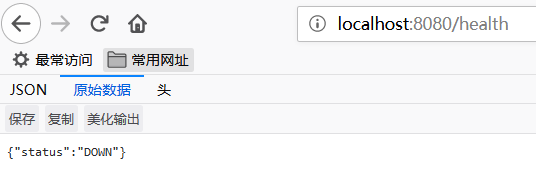
True表示当前数据库连接良好

False表示当前数据库连接中断

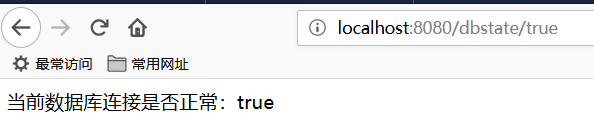
新建一个类，这个类实现了HealthIndicator接口，讲实现它的一个health方法，这个方法是用来构建当前微服务状态的。我们要做的就是当数据库连接中断（dbConnectionIsOk=false）的时候，来调整当前微服务的状态为DOWN。



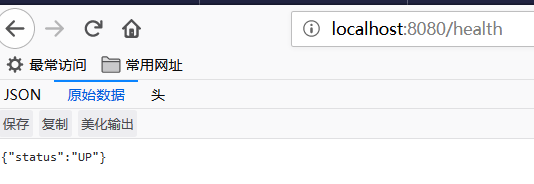
好了，重新启动first-service-provider试试，然后访问<http://localhost:8080/health> 接口看看。会发现开始当前微服务状态为DOWN：



然后通过测试接口，修改状态为true：



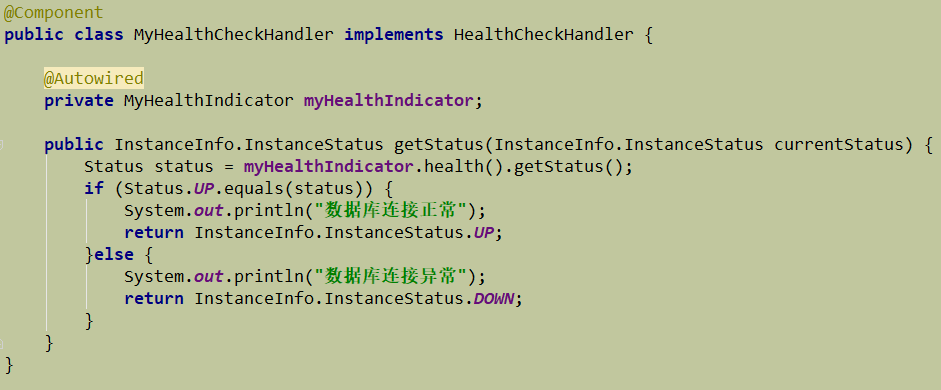
再访问health接口试试：



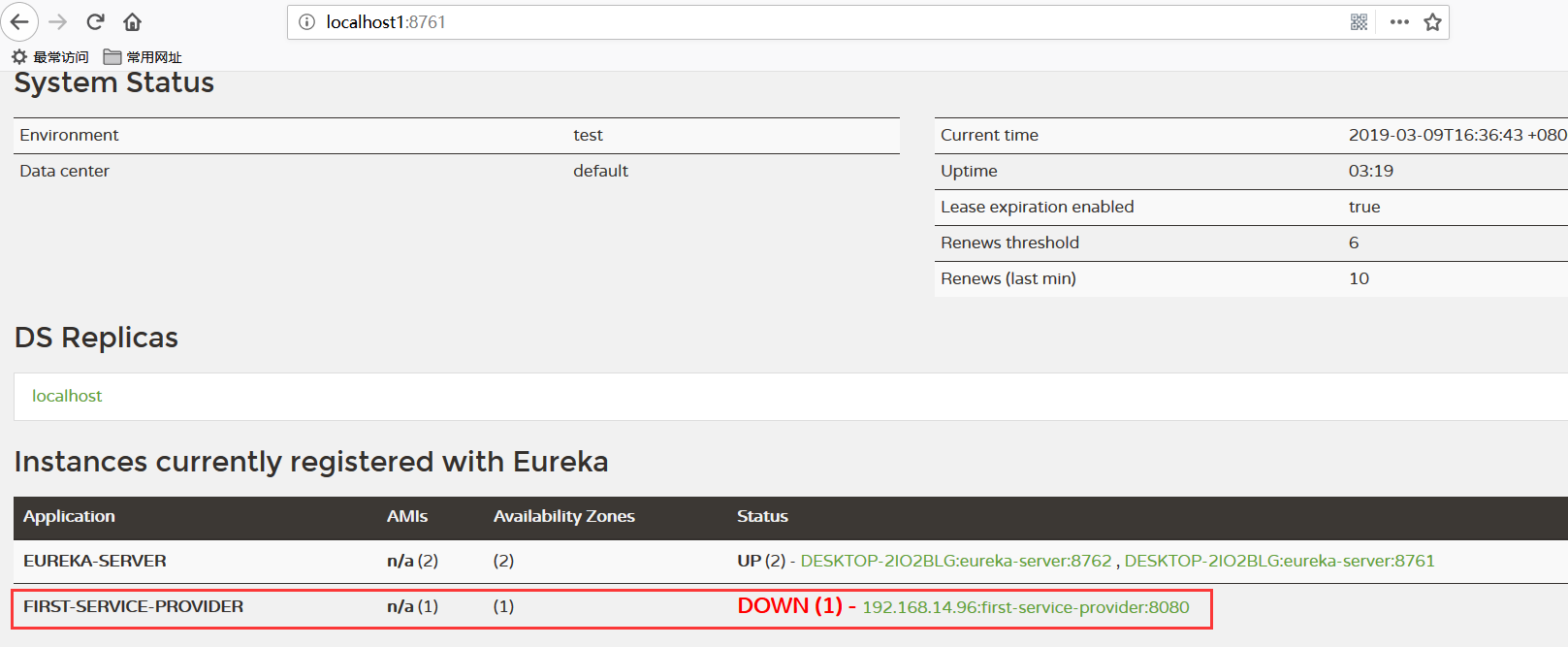
可以看到服务已经起来了。

光这样的话只有服务提供者自己知道当前健康状态，我们还需要将之告诉服务器，然后在Eureka服务器上注册的其他微服务将拿不到这些“生病”的微服务。

新建一个健康检查处理器，处理器会将健康状态保存在内存，一旦状态发生改变将报告给服务器，如果状态不正常，其他微服务也就拿不到它了。

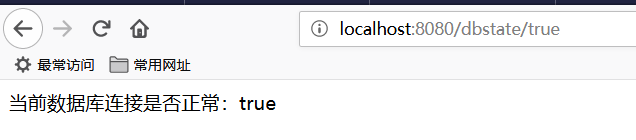
*NOTE: 注意都不要忘了加@Component注解哦，要加入到Spring的容器托管，不然不会起作用的。*

改好之后，再启动first-service-provider，启动的过程中可以看到控制台打印的“数据库连接异常”信息，这是因为dbConnectionIsOk变量的初始值是false。启动完了之后访问Eureka Server看看：

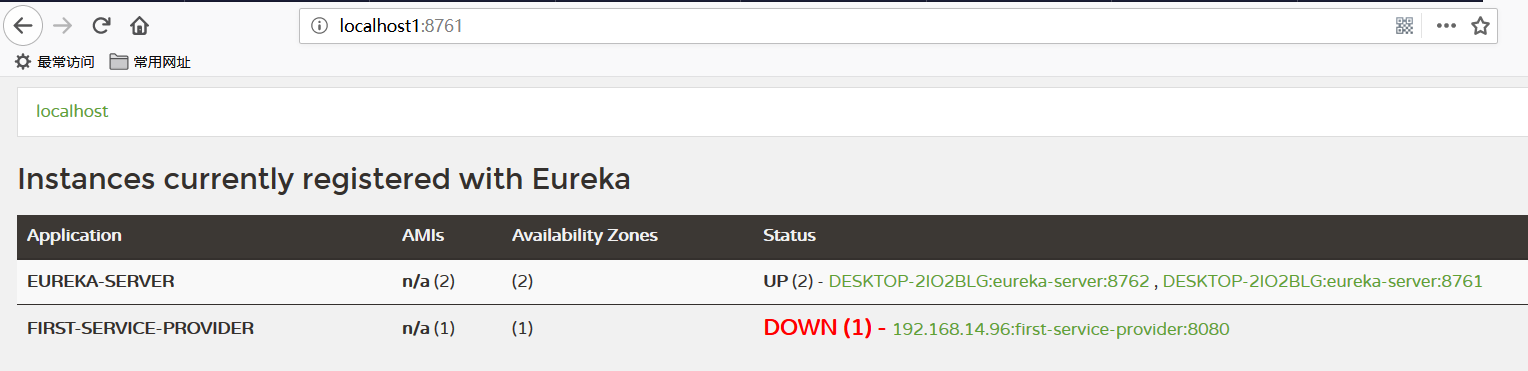


可以看到目前first-service-provider是DOWN的，被红色标记出来了。

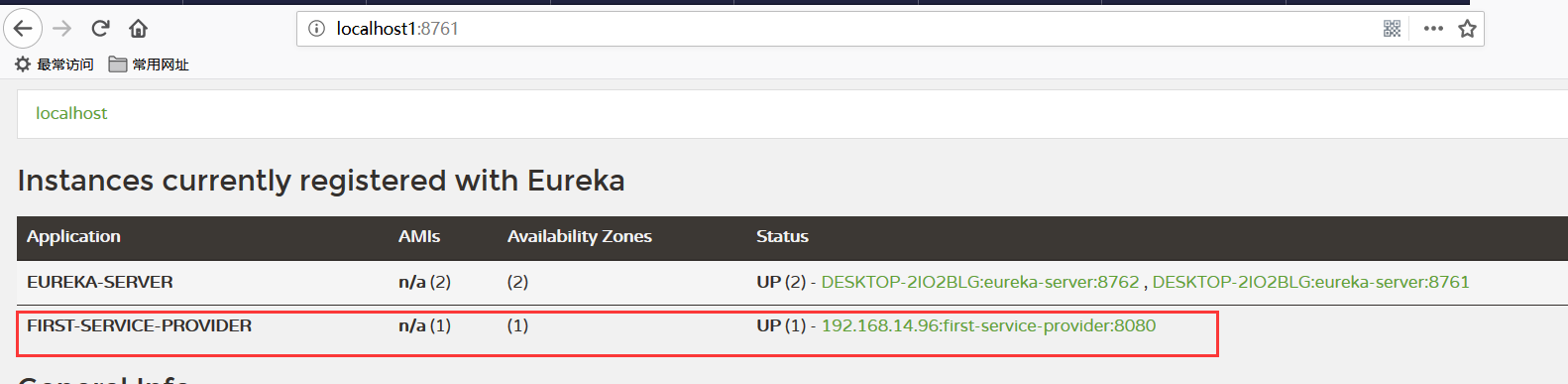
我们通过测试接口改下状态：



咦！为啥还是DOWN的？



嚯嚯，是因为传状态给服务器的动作是定时执行的，并不是实时执行，默认是30秒钟一次，所以等待30秒钟再看看：



可以发现，它已经UP了。

那么我们怎么修改这个默认时间呢？可以再application.properties里面加入这个属性：

**eureka.client.instance-info-replication-interval-seconds**=**10**

这样就能将时间限制为10秒了。如果是application.yml文件，那就是这样的：

**eureka.client.instanceInfoReplicationIntervalSeconds**: **10**

自己动手试试吧！

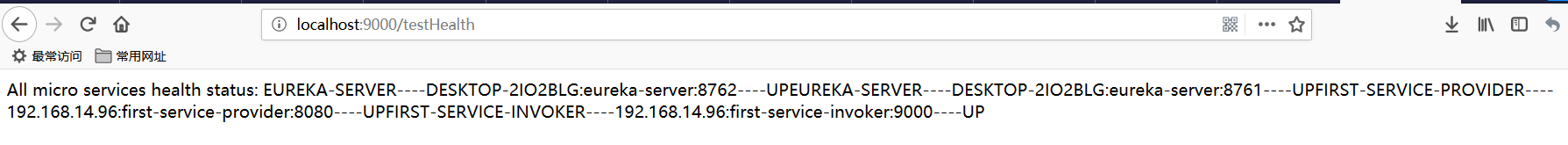
接下来，演示一下微服务查询。

这次我们要修改的是first-service-invoker(调用者) 的代码来看看健康自检的效果。

在first-service-invoker里面加一个测试接口，代码如下：



启动first-service-invoker, 在浏览器输入：<http://localhost:9000/testHealth> 就可以看到各个服务的状态了



我来美化一下：

