**Spring Cloud构建微服务架构（一）服务注册与发现**

**Spring Cloud简介**

Spring Cloud是一个基于Spring Boot实现的云应用开发工具，它为基于JVM的云应用开发中的配置管理、服务发现、断路器、智能路由、微代理、控制总线、全局锁、决策竞选、分布式会话和集群状态管理等操作提供了一种简单的开发方式。

Spring Cloud包含了多个子项目（针对分布式系统中涉及的多个不同开源产品），比如：Spring Cloud Config、Spring Cloud Netflix、Spring Cloud CloudFoundry、Spring Cloud AWS、Spring Cloud Security、Spring Cloud Commons、Spring Cloud Zookeeper、Spring Cloud CLI等项目。

**微服务架构**

“微服务架构”在这几年非常的火热，以至于关于微服务架构相关的产品社区也变得越来越活跃（比如：netflix、dubbo），Spring Cloud也因Spring社区的强大知名度和影响力也被广大架构师与开发者备受关注。

那么什么是“微服务架构”呢？简单的说，微服务架构就是将一个完整的应用从数据存储开始垂直拆分成多个不同的服务，每个服务都能独立部署、独立维护、独立扩展，服务与服务间通过诸如RESTful API的方式互相调用。

对于“微服务架构”，大家在互联网可以搜索到很多相关的介绍和研究文章来进行学习和了解。也可以阅读始祖Martin Fowler的[《Microservices》](http://martinfowler.com/articles/microservices.html" \t "_blank)，本文不做更多的介绍和描述。

**服务注册与发现**

在简单介绍了Spring Cloud和微服务架构之后，下面回归本文的主旨内容，如何使用Spring Cloud搭建服务注册与发现模块。

这里我们会用到[Spring Cloud Netflix](https://cloud.spring.io/spring-cloud-netflix/" \t "_blank)，该项目是Spring Cloud的子项目之一，主要内容是对Netflix公司一系列开源产品的包装，它为Spring Boot应用提供了自配置的Netflix OSS整合。通过一些简单的注解，开发者就可以快速的在应用中配置一下常用模块并构建庞大的分布式系统。它主要提供的模块包括：服务发现（Eureka），断路器（Hystrix），智能路有（Zuul），客户端负载均衡（Ribbon）等。

所以，我们这里的核心内容就是服务发现模块：Eureka。下面我们动手来做一些尝试。

**创建“服务注册中心”**

创建一个基础的Spring Boot工程，并在pom.xml中引入需要的依赖内容：

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId> <version>2.0.0.M3</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka-server</artifactId>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Brixton.RELEASE</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement> |

通过@EnableEurekaServer注解启动一个服务注册中心提供给其他应用进行对话。这一步非常的简单，只需要在一个普通的Spring Boot应用中添加这个注解就能开启此功能，比如下面的例子：

|  |
| --- |
| @EnableEurekaServer  @SpringBootApplication  **public** **class Application** {  **public** **static** **void** **main**(String[] args) {  SpringApplication.run(Application.**class**, args);  }  } |

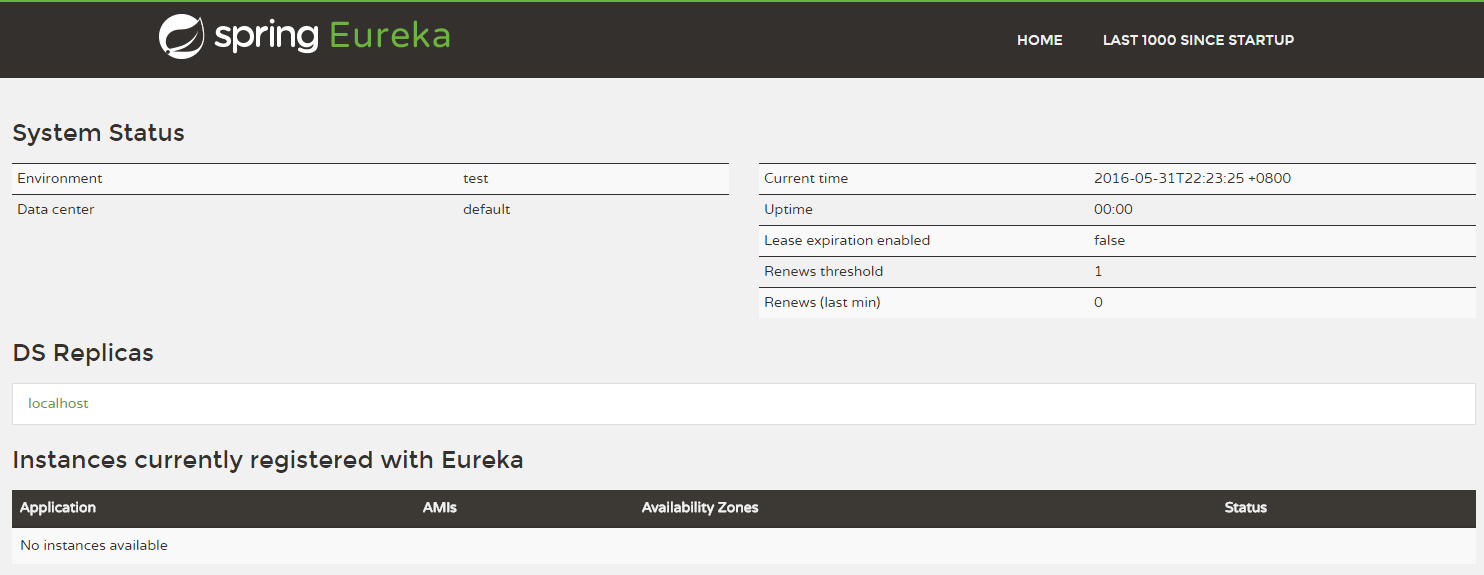
在默认设置下，该服务注册中心也会将自己作为客户端来尝试注册它自己，所以我们需要禁用它的客户端注册行为，只需要在application.properties中问增加如下配置：

|  |
| --- |
| server.port=1111  eureka.client.register-with-eureka=**false**  eureka.client.fetch-registry=**false**  eureka.client.serviceUrl.defaultZone=http://localhost:${server.port}/eureka/ |

为了与后续要进行注册的服务区分，这里将服务注册中心的端口通过server.port属性设置为1111。

启动工程后，访问：<http://localhost:1111/>

可以看到下面的页面，其中还没有发现任何服务

[](http://blog.didispace.com/content/images/2016/05/QQ--20160531222344.png)

**创建“服务提供方”**

下面我们创建提供服务的客户端，并向服务注册中心注册自己。

假设我们有一个提供计算功能的微服务模块，我们实现一个RESTful API，通过传入两个参数a和b，最后返回a + b的结果。

首先，创建一个基本的Spring Boot应用，在pom.xml中，加入如下配置：

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.0.M3</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka</artifactId>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Brixton.RELEASE</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement> |

其次，实现/add请求处理接口，通过DiscoveryClient对象，在日志中打印出服务实例的相关内容。

|  |
| --- |
| @RestController  **public** **class ComputeController** {  **private** **final** Logger logger = Logger.getLogger(getClass());  @Autowired  **private** DiscoveryClient client;  @RequestMapping(value = "/add" ,method = RequestMethod.GET)  **public** Integer **add**(@RequestParam Integer a, @RequestParam Integer b) {  ServiceInstance instance = client.getLocalServiceInstance();  Integer r = a + b;  logger.info("/add, host:" + instance.getHost() + ", service\_id:" + instance.getServiceId() + ", result:" + r);  **return** r;  }  } |

最后在主类中通过加上@EnableDiscoveryClient注解，该注解能激活Eureka中的DiscoveryClient实现，才能实现Controller中对服务信息的输出。

|  |  |
| --- | --- |
|  | @EnableDiscoveryClient  @SpringBootApplication  **public** **class ComputeServiceApplication** {  **public** **static** **void** **main**(String[] args) {  SpringApplication.run(ComputeServiceApplication.**class**, args);  }  } |

我们在完成了服务内容的实现之后，再继续对application.properties做一些配置工作，具体如下：

|  |
| --- |
| spring.application.name=compute-service  server.port=2222  eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/ |

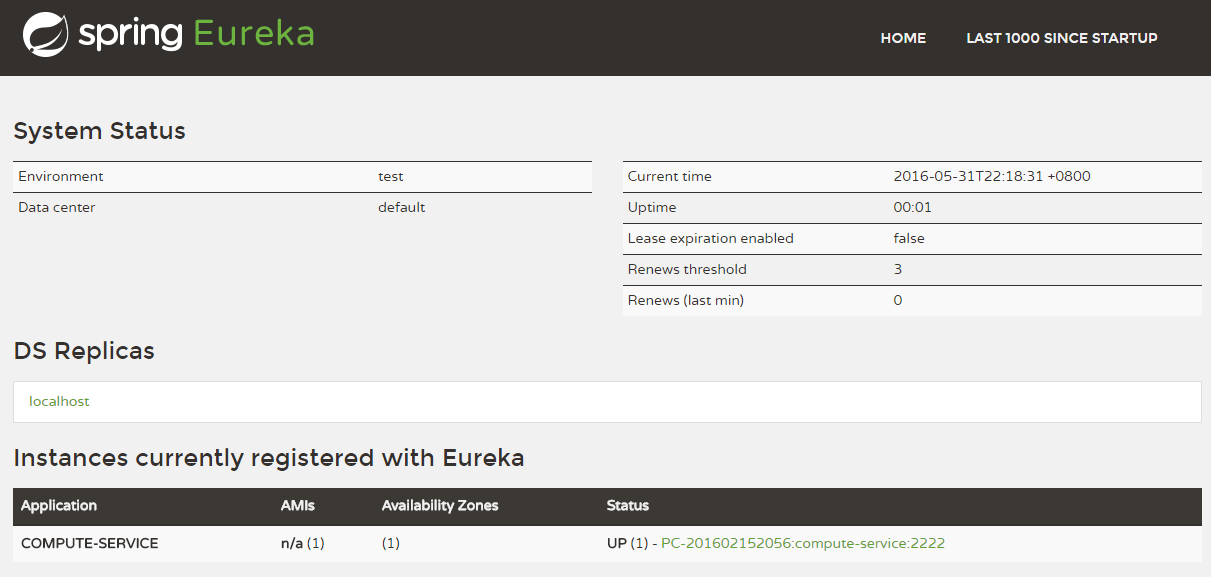
通过spring.application.name属性，我们可以指定微服务的名称后续在调用的时候只需要使用该名称就可以进行服务的访问。

eureka.client.serviceUrl.defaultZone属性对应服务注册中心的配置内容，指定服务注册中心的位置。

为了在本机上测试区分服务提供方和服务注册中心，使用server.port属性设置不同的端口。

启动该工程后，再次访问：<http://localhost:1111/>

可以看到，我们定义的服务被注册了。

[](http://blog.didispace.com/content/images/2016/05/QQ--20160531221940.png)