1.4  [Spring Boot 日志记录 SLF4J](http://blog.csdn.net/catoop/article/details/50501714)

这里使用了SLF4J的logback来输出日志，其比log4j 要好，因为他效率更高。

1.4.1 [spring](http://lib.csdn.net/base/javaee) Boot 提供了一套日志系统，logback是最优先的选择。

配置了logback.xml可以利用Spring Boot提供的默认日志配置：

<?xml version="1.0" encoding="UTF-8"?>

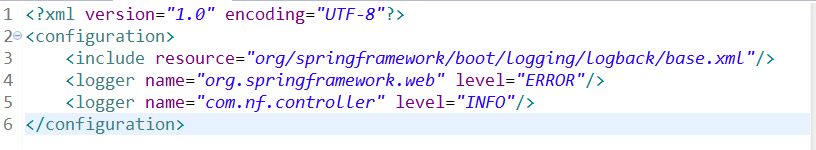
<configuration>

<include resource="org/springframework/boot/logging/logback/base.xml"/>

<logger name="org.springframework.web" level="DEBUG"/>

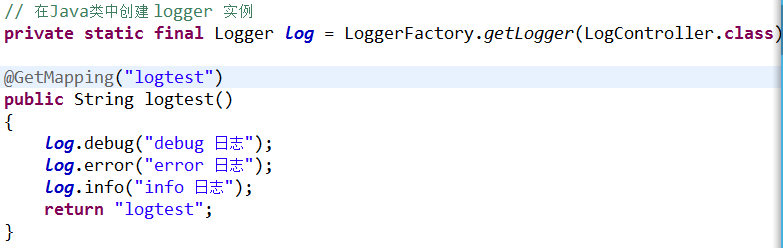
</configuration>

Spring Boot提供的默认日志配置，所以我们只需要在src/main/resources目录下建立一个logback.xml，并将我们需要记录日志的包路径和日志级别配置好，我们就可以进行日志的输出了。如下图所示：



在logback.xml中配置了包路径为com.nf.controller日志级别为INFO.

1.4.2 在Java代码中创建实例，并在需要输出日志的地方使用。



1.4.3 将日志记录到日志文件中

一般我们记录日志都是将日志记录到日志文件中，要记录到日志文件中，这需要我们在application.properties文件中加上logging.file= log/logtest.log的配置。这样我们就可以将控制台输出的日志记录到log文件夹里的logtest.log中了。

# 1.5 Spring Boot中使用JdbcTemplate访问数据库

通常我们做App也好，做Web应用也好，都需要内容，而内容通常存储于各种类型的数据库，服务端在接收到访问请求之后需要访问数据库获取并处理成展现给用户使用的数据形式。接下来我们介绍一下，spring boot是如何访问数据库的。

1.5.1 数据源的配置

（1）首先，为了连接数据库需要引入jdbc支持，这里我使用的是mysql数据库，在pom.xml中引入如下配置：

<dependency>

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

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

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.21</version>

</dependency>

（2）在src/main/resources/application.properties中配置数据库连接信息，配置如下：

spring.datasource.url=jdbc:mysql://localhost:3306/test

spring.datasource.username=root

spring.datasource.password=123456

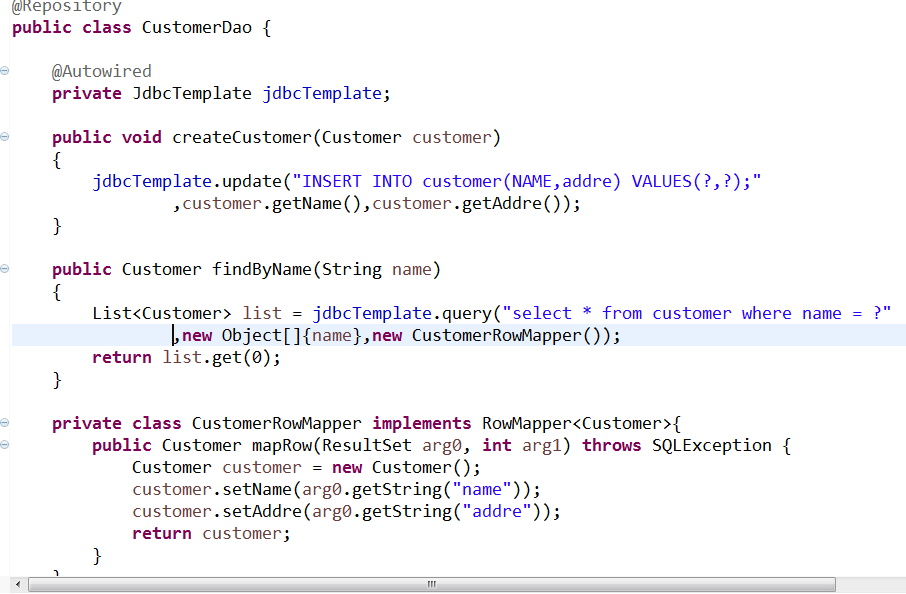
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

1.5.2 使用JdbcTemplate操作数据库

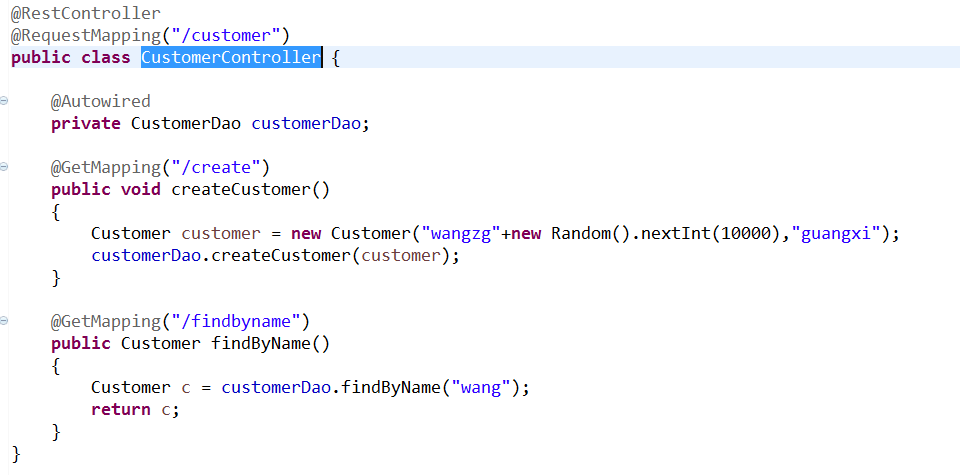
Spring的JdbcTemplate是自动配置的，你可以直接使用@Autowired来注入到你自己的bean中来使用。

举例：我们在创建customer表，包含属性name、addre，下面来编写数据访问对象和接口进行测试。

* 通过JdbcTemplate实现customerdao 插入和查询的数据访问操作



1.5.3创建CustomerController进行接口测试，通过调用创建和查询的接口来验证数据库操作的正确性。



# 2.Spring Cloud

## 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。

# 2.1分布式配置中心

Spring Cloud Config为服务端和客户端提供了分布式系统的外部化配置支持。配置服务器为各应用的所有环境提供了一个中心化的外部配置。配置服务器默认采用git来存储配置信息，这样就有助于对环境配置进行版本管理，并且可以通过git客户端工具来方便的管理和访问配置内容，当然他也提供本地化文件系统的存储方式。

## 2.1.1构建Config Server

（1）pom.xml中引入spring-cloud-config-server依赖，完整依赖配置如下：

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.nf.config</groupId>

<artifactId>configservice</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>configservice</name>

<parent>

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

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

<version>1.5.2.RELEASE</version>

<relativePath/>

</parent>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

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

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

</dependency>

<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-config-server</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<configuration>

<warName>springboot</warName>

</configuration>

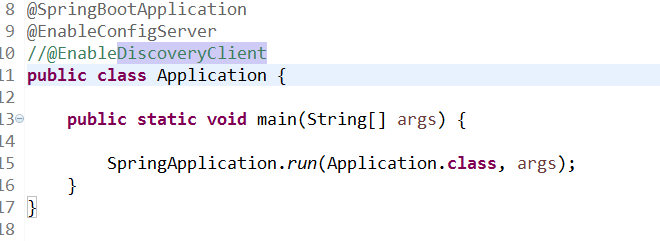
</plugin>

</plugins>

</build>

</project>

* （2）创建Spring Boot的程序主类，并添加@EnableConfigServer注解，开启Config Server



（3）application.properties中配置服务信息以及git信息，例如：

server.port=8889

spring.application.name=config-server

spring.cloud.config.server.git.uri=https://github.com/wangzigui/hello-wolrd

spring.cloud.config.server.git.username=username

spring.cloud.config.server.git.password=password

* spring.cloud.config.server.git.uri：配置git仓库位置
* spring.cloud.config.server.git.username：访问git仓库的用户名
* spring.cloud.config.server.git.password：访问git仓库的用户密码

到这里，使用一个通过Spring Cloud Config实现，并使用git管理内容的配置中心已经完成了，启动该应用，成功后开始下面的内容。

Spring Cloud Config也提供本地存储配置的方式。我们只需要设置属性spring.profiles.active=native，Config Server会默认从应用的src/main/resource目录下检索配置文件。也可以通过spring.cloud.config.server.native.searchLocations=file:F:/properties/属性来指定配置文件的位置。虽然Spring Cloud Config提供了这样的功能，但是为了支持更好的管理内容和版本控制的功能，还是推荐使用git的方式

## 2.1.2微服务端映射配置

在完成并验证了配置服务中心之后，下面看看我们如何在微服务应用中获取配置信息。

（1）创建一个Spring Boot应用，在pom.xml中引入spring-cloud-starter-config依赖，完整依赖关系如下：

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.nf.configclient</groupId>

<artifactId>configclient</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>configclient</name>

<parent>

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

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

<version>1.5.2.RELEASE</version>

<relativePath/>

</parent>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

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

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

</dependency>

<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-config</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<configuration>

<warName>springboot</warName>

</configuration>

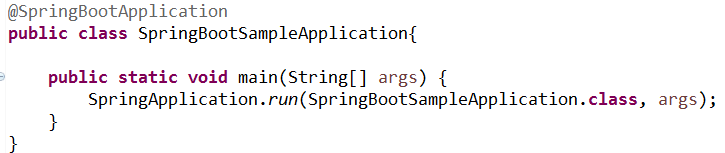
</plugin>

</plugins>

</build>

</project>

（2）创建最基本的Spring Boot启动主类



（3）创建bootstrap.properties配置，来指定config server，例如：

spring.application.name=client1

server.port=8006

spring.cloud.config.uri=http://localhost:8889

spring.cloud.config.name=sample-cloud-client

spring.cloud.config.profile=${config.profile:dev}

这里需要格外注意：上面这些属性必须配置在bootstrap.properties中，config部分内容才能被正确加载。因为config的相关配置会先于application.properties，而bootstrap.properties的加载也是先于application.properties

（4）创建application.properties并配置：

app.message=local message

user.username=my name is shanhy.(local)

user.userage=28

（5）在git仓库**<https://github.com/wangzigui/hello-wolrd>创建一个sample-cloud-client-dev.properties**的文件，文件里设置

app.message=Hi~wangzigui. message version 1.4 (dev)

（6）创建一个Rest Api来返回配置中心的from属性，具体如下：



启动该应用，并访问：http://localhost:8006/test，我们就可以根据配置内容输出对应环境的app.message内容了

## 2.2 服务注册与发现

## 2.2.1创建“服务注册中心”

（1）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>1.5.2.RELEASE</version>

<relativePath/>

</parent>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

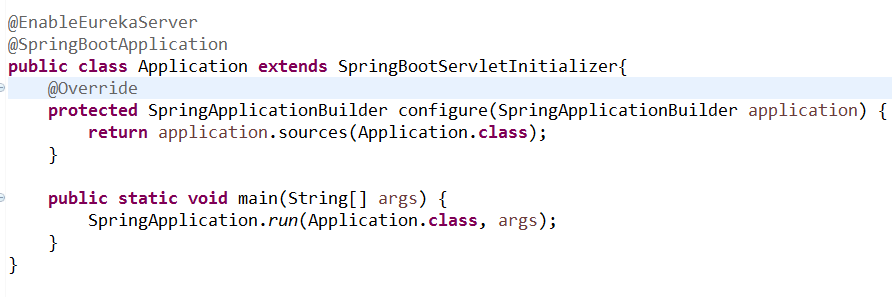
<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka-server</artifactId>

</dependency>

</dependencies>

（2）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：



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=8001

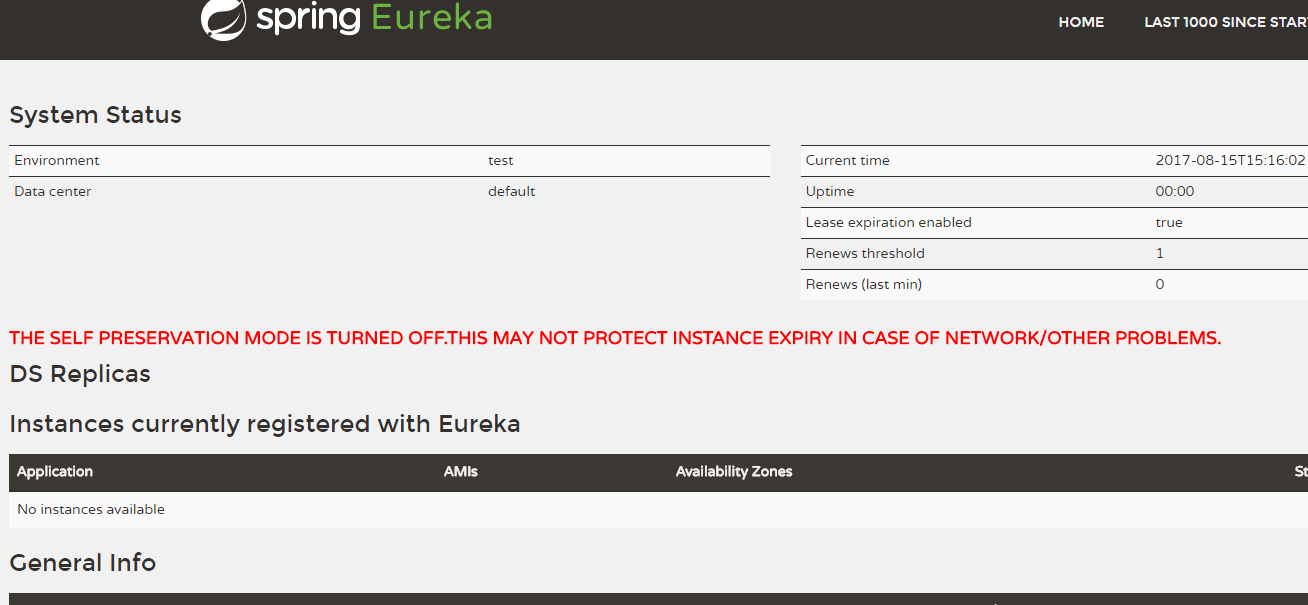
eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

eureka.client.serviceUrl.defaultZone=http://localhost:${server.port}/eureka

启动工程后，访问：<http://localhost:8001/>

可以看到下面的页面，其中还没有发现任何服务



#### 2.2.2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。

1. 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>1.5.2.RELEASE</version>

<relativePath/>

</parent>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

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

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

</dependency>

<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-config</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

1. 创建application.properties做一些配置工作，具体如下：

spring.application.name=client1

server.port=8006

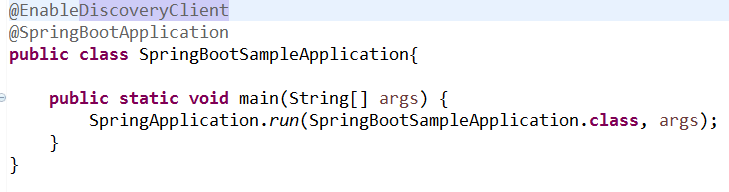
#配置服务注册中心

eureka.client.service-url.defaultZone=http://localhost:8001/eureka

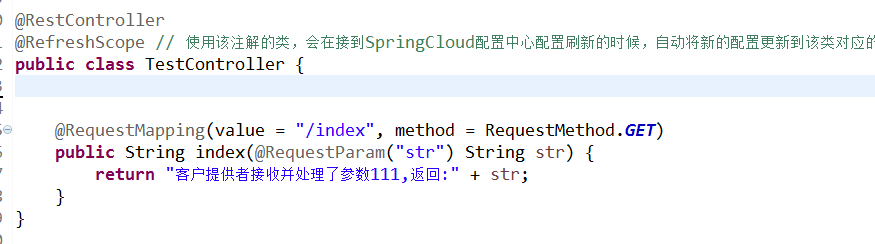
通过spring.application.name属性，我们可以指定微服务的名称后续在调用的时候只需要使用该名称就可以进行服务的访问。

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。

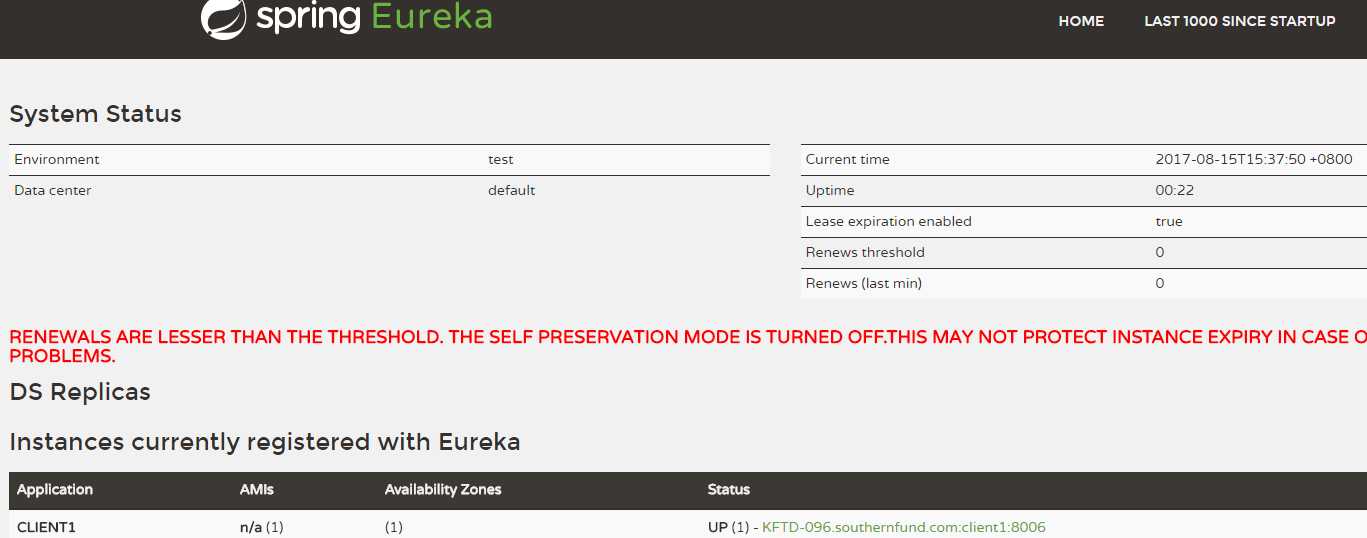
1. 在主类中通过加上@EnableDiscoveryClient注解，该注解能激活Eureka中的DiscoveryClient实现



1. 最后，实现/index请求处理接口，通过DiscoveryClient对象，在日志中打印出相关内容。



启动该工程后，再次访问：<http://localhost:8001/>，可以看到我们刚刚启动的微服务已经注册到注册中心



2.2.3 构建服务消费者

上面我们已经成功创建了“服务注册中心”，实现并注册了一个“服务提供者：client1。那么我们要如何去消费服务提供者的接口内容呢？

Feign是一个声明式的Web Service客户端，它使得编写Web Serivce客户端变得更加简单。我们只需要使用Feign来创建一个接口并用注解来配置它既可完成。它具备可插拔的注解支持，包括Feign注解和JAX-RS注解。Feign也支持可插拔的编码器和解码器。Spring Cloud为Feign增加了对Spring MVC注解的支持，还整合了Ribbon和Eureka来提供均衡负载的HTTP客户端实现

下面我们通过一个例子来展现Feign如何方便的声明对上述client1服务的定义和调用。

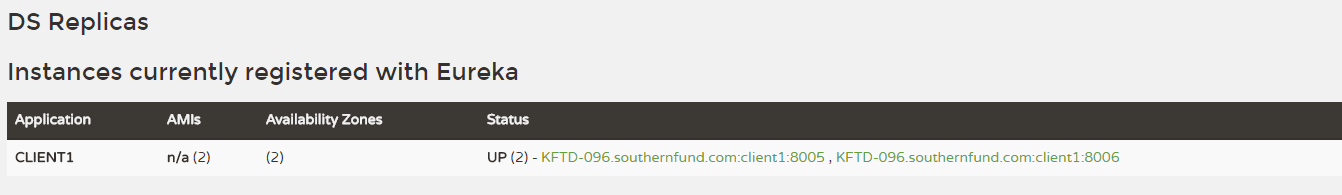
准备工作：

启动上面构建中的服务注册中心：eureka-server

启动上面构建中的服务提供方：client1

修改client1中的server-port为8005，再启动一个服务提供方：client1

此时访问：<http://localhost:8001/>



1. 再创建一个Spring Boot工程，配置pom.xml，具体如下：

<parent>

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

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

<version>1.5.2.RELEASE</version>

<relativePath/>

</parent>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Dalston.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-feign</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

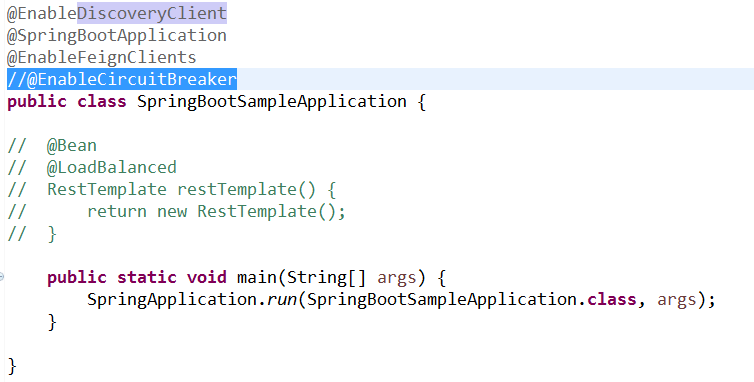
<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

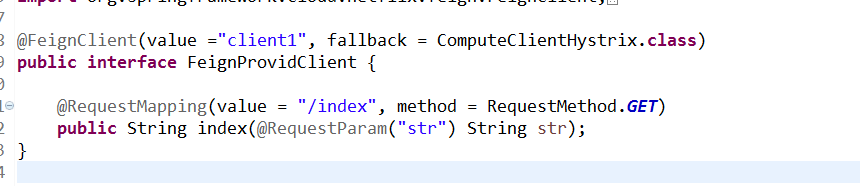
</dependency>

</dependencies>

（2）在应用主类中通过@EnableFeignClients注解开启Feign功能，具体如下：



（3）定义client1服务的接口，具体如下：



使用@FeignClient("client1")注解来绑定该接口对应compute-service服务

通过Spring MVC的注解来配置client1服务下的具体实现。

（4）在web层中调用上面定义的FeignProvidClient，具体如下



1. application.properties指定eureka服务注册中心即可，如：

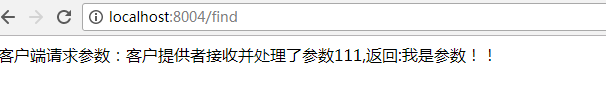
spring.application.name=client

server.port=8004

#配置服务注册中心

eureka.client.service-url.defaultZone=http://localhost:8001/eureka/

启动该应用，访问几次：<http://localhost:8004/find>



可以看到client调用了client1的接口成功，并返回了参数。