# Spring Cloud Bus

1. **Spring Cloud Bus是什么？**

Spring Cloud Bus隶属于Spring Cloud体系，使用轻量级的消息代理（例如RabbitMQ、Kafka等）连接分布式系统的节点，通过广播传播状态的更改（例如配置的更新）或者其他的管理指令。如果你熟悉Spring Boot Actuator，可以把Spring Cloud Bus想象成一个分布式的Spring Boot Actuator。

1. **Spring Cloud Bus意义**

Spring Cloud Bus使用轻量级的消息代理（例如RabbitMQ、Kafka等），因此具备了消息代理的特性。

1. **实时性高**

与REST服务的“请求-响应”模式不同，消息总线的实时性非常高。使用了消息总线，生产者一方只要把消息发送到队列后，就可以响应用户。无须等待处理结果，实现了异步处理；同时，对于消费者而言，消费者对于消息的到达感知也非常及时。消费者会对消息总线进行监听，只要有消息进入队列，就立即得到通知。

这种优势是REST服务所不能具备的。在REST服务中，要想及时获取到更新通知，就不得不进行轮询。这往往非常低效。

1. **生产者与消费者解耦**

在消息总线中，生产者负责将消息发送到队列中，而消费者把消息从队列中取出来。生产者无须等待消费者启动，消费者也无须关心生产者是否已经处于就绪状态。所以，这种模式能很好地实现生产者与消费者的解耦；对比REST服务，服务调用方必须等待服务的提供方准备就绪，否则调用失败。

1. **故障率低**

消息总线拥有对其他通信方式更高的成功率。一方面，生产者与消费者之间实现了解耦，所以，生产者与消费者之间不存在强关联关系，即便是生产者或消费者任意一方掉线，也不会影响消息的最终到达；另一方面，消息总线往往结合数据库来实现消息的持久化，并设置状态标识。只有消息消费成功，才会去修改状态标识。

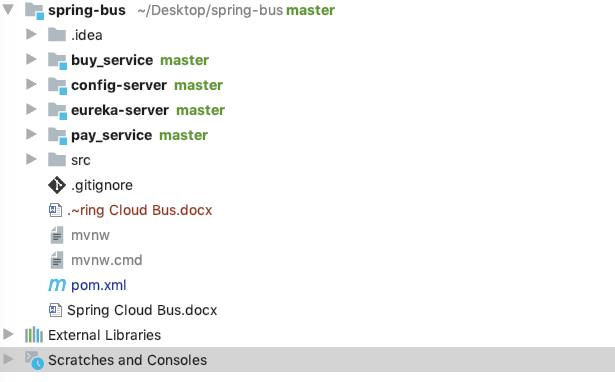
消息总线同时还承担着缓冲区的作用。大量业务消息首先会进入消息队列进行缓存，消息的消费者可以根据自己的处理能力来进行消费，所以不管消息的数据量有多少，都不会对消费者造成冲击。

1. **Spring Cloud Bus实现消息总线**
2. **说明**

下面举例说明如何实现消息总线。

本例使用RabbitMQ实现Spring Cloud Bus消息总线，更新各个微服务节点配置功能。

项目架构图如下



本例采用maven作为项目依赖管理工具，使用父pom的方式统一管理Spring Boot和Spring Cloud的版本，降低项目维护成本；同时，在未来，可以将父pom项目，抽象成一个maven的dependence配置，发布在自己或公司的maven仓库中，统一版本管理的同时，简化各个分布式系统的配置。

项目有4个module：

1. eureka-server:是服务的发现和注册中心；
2. config-server:是分布式系统中的配置服务中心，用来读取远程配置中心的配置，本例中使用Git作为远程配置中心；
3. buy\_service:分布式系统中需要读取配置的服务节点。
4. pay\_service:分布式系统中需要读取配置的服务节点。
5. **实现环境**

JDK : 1.8.0\_181;

Spring Boot : 2.0.3.RELEASE;

Spring Cloud :Finchley.RELEASE;

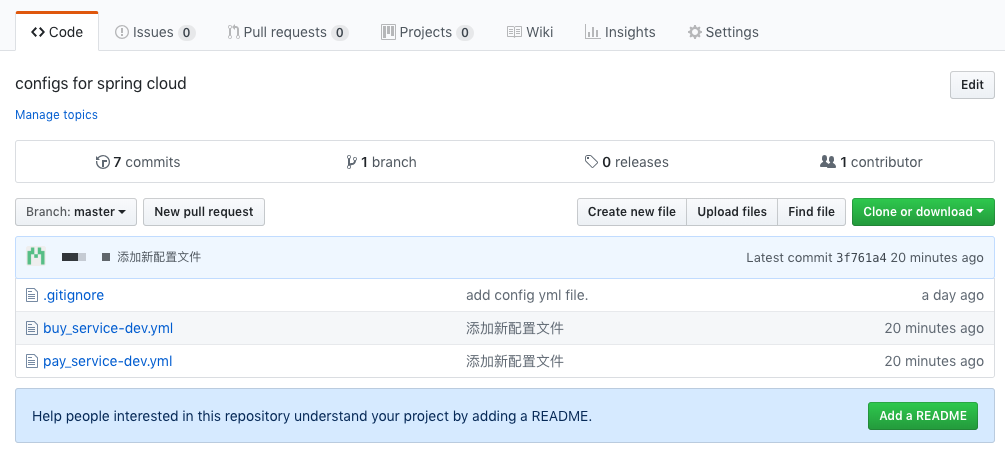
Rabbit MQ : 3.7.9

1. **实现步骤**

本例需要先安装Rabbit MQ，请到Rabbit MQ官网http://www.rabbitmq.com/下 载安装。

1. 在Github上创建配置信息。

在Github上创建名称为configs的repository，并在configs目录下新建两个配置文件，如下图：



如上图所示，两个配置文件代码如下：

buy\_service-dev.yml中的代码如下：

spring:

redis:

port: 9090

pay\_service-dev.yml中的代码如下：

spring:

redis:

port: 9091

创建maven项目，pom文件如下：

右键项目，添加模块eureka-server，pom文件如下：

在eureka-server项目启动类上添加@EnableEurekaServer注解

在eureka-server配置文件application.yml中添加如下代码：

server:

port: 8889

eureka:

instance:

hostname: localhost

client:

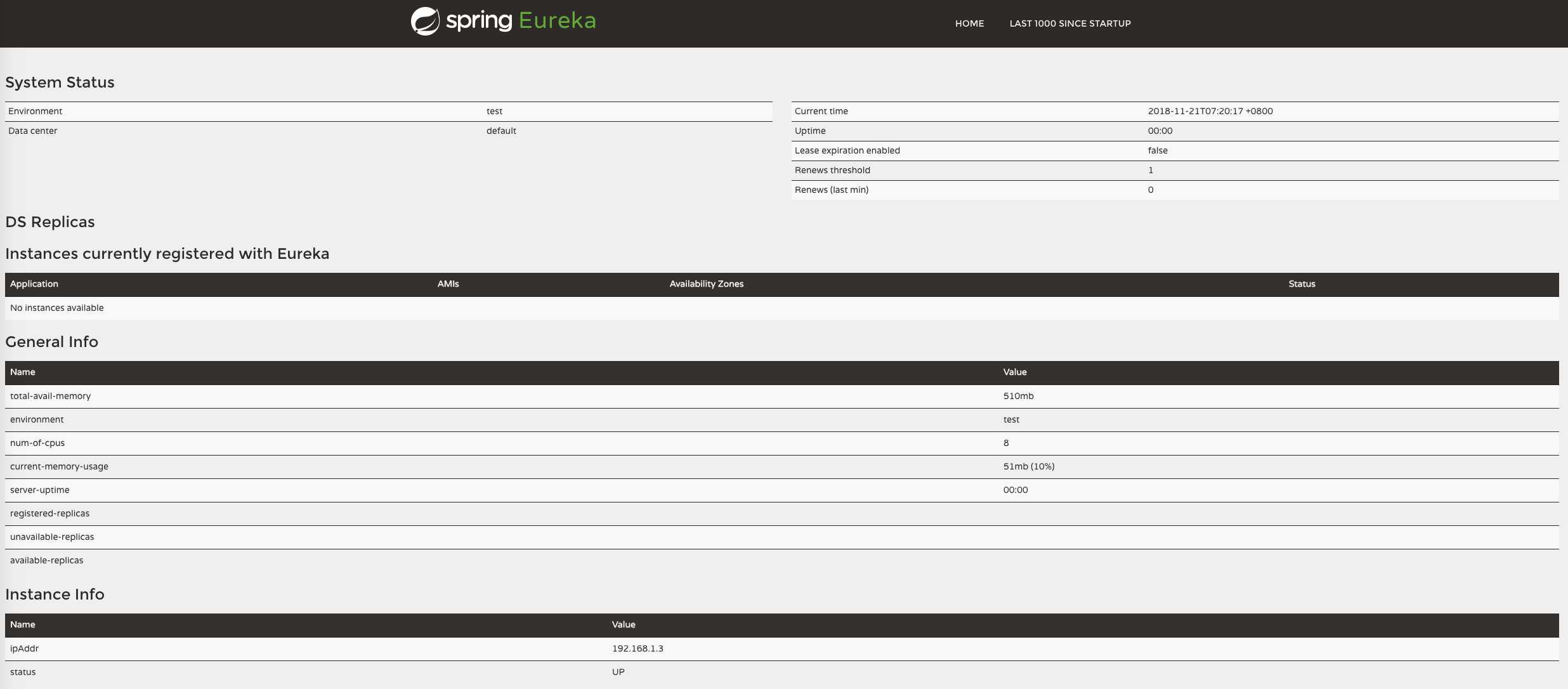
registerWithEureka: false

fetchRegistry: false

serviceUrl:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

启动项目，成功后访问：<http://127.0.0.1:8889/> 如下图



右键项目，添加模块config-server，pom文件如下：

在config-server项目启动类上添加@EnableConfigServer和

@EnableEurekaClient注解

在eureka-server配置文件application.properties中添加如下代码：

spring.application.name=config-server

server.port=8888

spring.cloud.config.server.git.uri=https://github.com/juxincoder/configs.git

spring.cloud.config.server.git.searchPaths=configs

spring.cloud.config.label=master

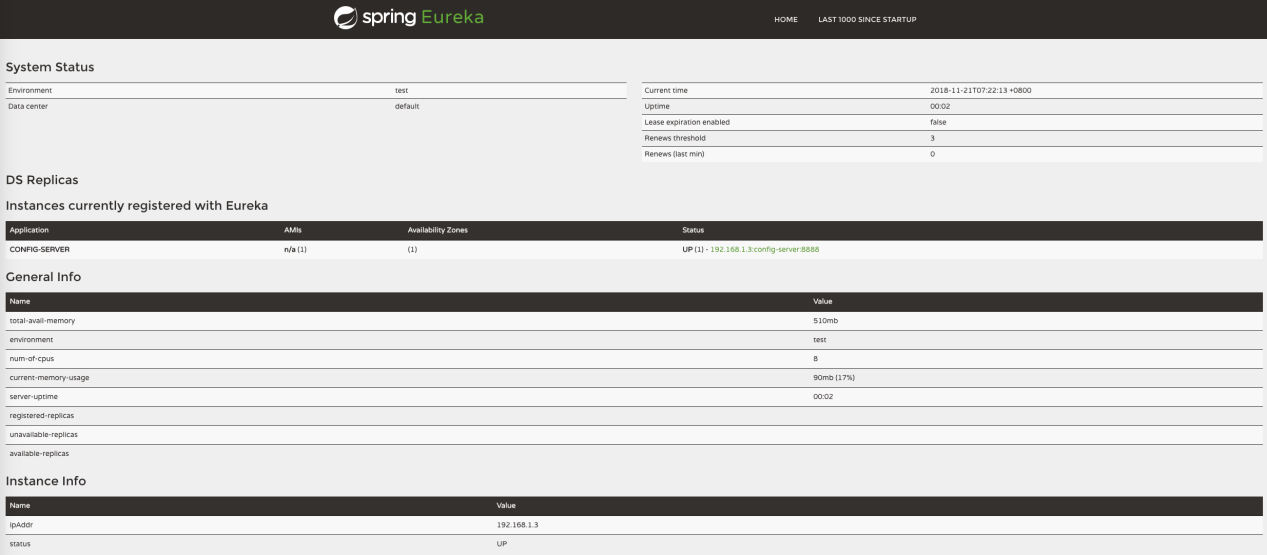
spring.cloud.config.server.git.username=

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

eureka.client.serviceUrl.defaultZone=http://localhost:8889/eureka/

其中：

启动项目，成功后访问：<http://127.0.0.1:8889/> 如下图



右键项目，添加模块buy\_service，pom文件如下：

在eureka-server项目启动类上添加@EnableEurekaServer注解

在eureka-server配置文件application.yml中添加如下代码：

server:

port: 8889

eureka:

instance:

hostname: localhost

client:

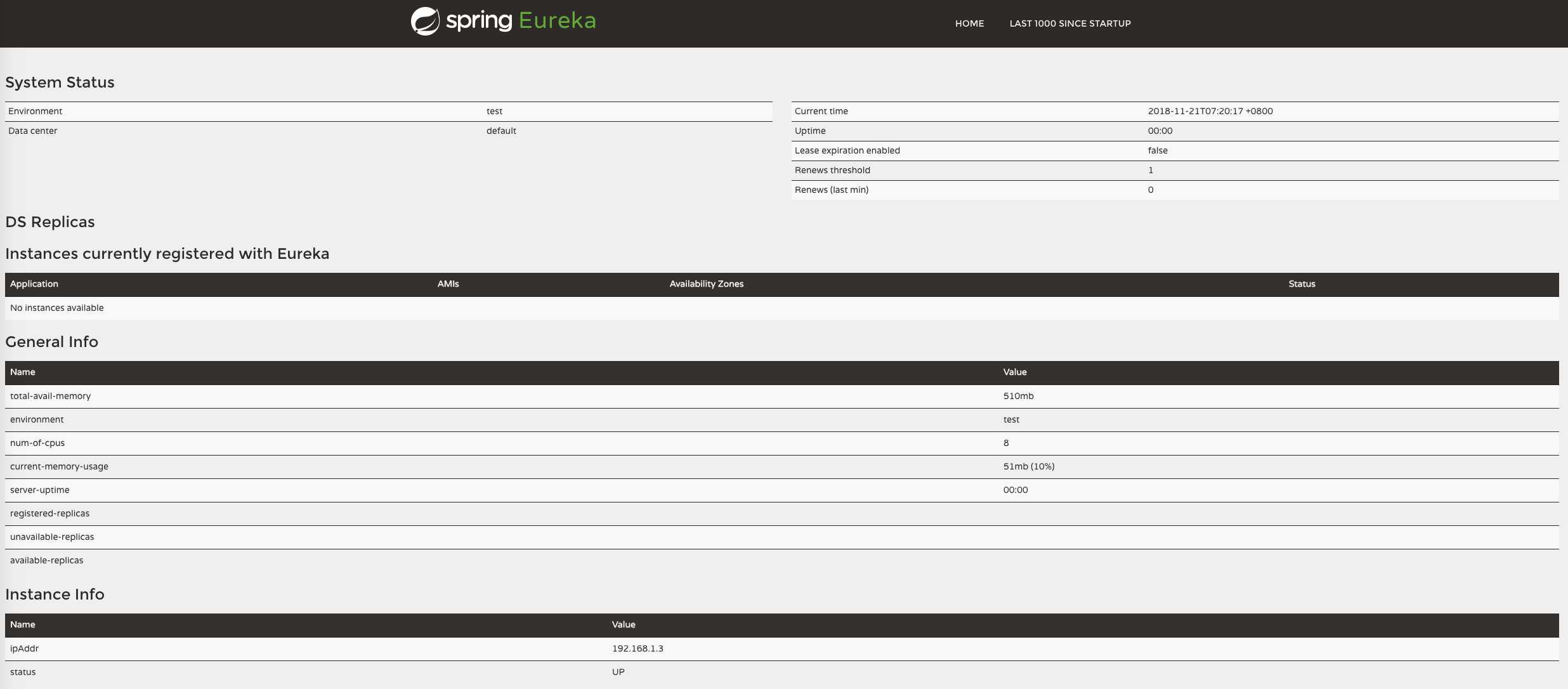
registerWithEureka: false

fetchRegistry: false

serviceUrl:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

启动项目，成功后访问：<http://127.0.0.1:8889/> 如下图



右键项目，添加模块eureka-server，pom文件如下：

在eureka-server项目启动类上添加@EnableEurekaServer注解

在eureka-server配置文件application.yml中添加如下代码：

server:

port: 8889

eureka:

instance:

hostname: localhost

client:

registerWithEureka: false

fetchRegistry: false

serviceUrl:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

启动项目，成功后访问：<http://127.0.0.1:8889/> 如下图

