Spring Cloud

(Finchley.RELEASE)

目录

[Spring Cloud是什么？ 3](#_Toc520908188)

[Spring Cloud有那些工具框架？ 3](#_Toc520908189)

# 1简介

## 1.1 Spring Cloud是什么？

1. Spring Cloud 是基于Spring Boot来实现的。
2. Spring Cloud 是一系列工具框架的集合体。
3. Spring Cloud 提供了一套完整的微服务解决方案。

## 1.2 Spring Cloud有那些工具框架？

1. Spring Cloud Config 配置中心，利用git集中管理程序的配置。
2. Spring Cloud Netflix 集成众多Netflix的开源软件。
3. Spring Cloud Bus 消息总线，利用分布式消息将服务和服务实例连接在一起，用于在一个集群中传播状态的变化。
4. Spring Cloud for Cloud Foundry 利用Pivotal Cloudfoundry集成你的应用程序。
5. Spring Cloud Cloud Foundry Service Broker 为建立管理云托管服务的服务代理提供一个起点。
6. Spring Cloud Cluster 基于Zookeeper，Redis，Hazelcast，Consul实现的领导选举和平明状态模式的抽象和实现。
7. Spring Cloud Consul 基于Hashicorp Consul实现的服务发现和配置管理。
8. Spring Cloud Security 在Zuul代理中为OAuth2 rest客户端和认证头转发提供负载均衡。
9. Spring Cloud Sleuth SpringCloud应用的分布式追踪系统，和Zipkin，HTrace，ELK兼容。
10. Spring Cloud Data Flow 一个云本地程序和操作模型，组成数据微服务在一个机构化的平台上。
11. Spring Cloud Stream 基于Redis，Rabbit，Kafka实现的消息微服务，简单声明模型用以在Spring Cloud应用中收发消息。
12. Spring Cloud Stream App Starters 基于Spring Boot为外部系统提供Spring的集成。
13. Spring Cloud Task短生命周期的微服务，为Spring Boot应用简单声明添加功能和非功能特性。
14. Spring Cloud Task App Starters
15. Spring Cloud Zookeeper 服务发现和配置管理基于Apache Zookeeper。
16. Spring Cloud for Amazon Web Service 快速和亚马逊网络服务集成。
17. Spring Cloud Connectors 便于Paas 应用在各种平台上连接到后端像数据库和消息经纪服务。
18. Spring Cloud Starters （项目已经终止在Angel.SR2后的版本和其他项目合并）
19. Spring Cloud CLI 插件用Groovy快速的创建Spring Cloud组件应用。

# 2 服务治理

## 2.1 简介

服务治理是微服务架构中最为核心和基础的模块，它主要用来实现各个微服务实例的自动化注册和发现。

## 2.2 Eureka服务治理

### 2.2.1 简介

Eureka是Spring Cloud Netflix 微服务套件中的一部分。Spring Cloud Eureka使用Netflix Eureka来实现服务注册与服务发现。Spring Cloud Eureka包含了服务端组件、客户端组件，并且服务端与客户端均采用java编写，所以Eureka主要适用于通过java实现分布式系统，或是与JVM兼容语言构建的系统。Eureka的服务端提供了比较完善的REST API，所以Eureka也支持将非java语言实现的服务纳入到Eureka服务治理体系中来，只需要其他语言平台实现自己的Eureka客户端。目前.Net平台的Steeltoe、Node.js的eureka-js-client等都已经实现了各自平台的Ereka客户端组件。

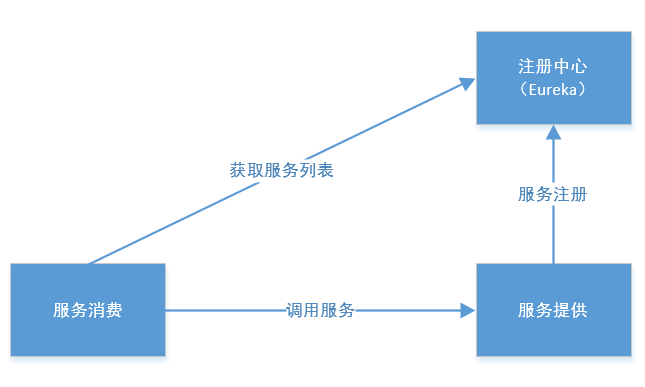
### 2.2.2 Eureka 服务端(注册中心)

Eureka服务端，即为注册中心，用于提供服务的注册与发现。

Eureka与其他服务注册中心一样，支持高可用配置，当集群中有分片出现故障时，Eureka自动进入保护模式。它允许在分片发生故障的时候继续提供服务的注册与发现，当故障恢复时，集群中的其他分片会把他们的状态再次同步回来。集群中的不同服务注册中心通过异步模式互相复制各自的状态，这也就意味着在给定的时间每个实例关于所有服务的状态可能存在不一致现象。

在服务治理框架中，通常都会构建一个注册中心，每个服务单元向注册中心登记自己提供的服务，包括服务的主机和端口号、服务版本号、通讯协议等一些附加信息。注册中心按照服务名分类组织服务清单，同时还需要以心跳检测的方式去监测服务清单中的服务是否可用，若不可用需要从服务清单中剔除，已达到排除故障服务的效果。

#### 2.2.2.1 单点注册中心



##### 2.2.2.1.1 导入相关依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.1.RELEASE</version> </parent>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Finchley.RELEASE</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies> </dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency> </dependencies> |

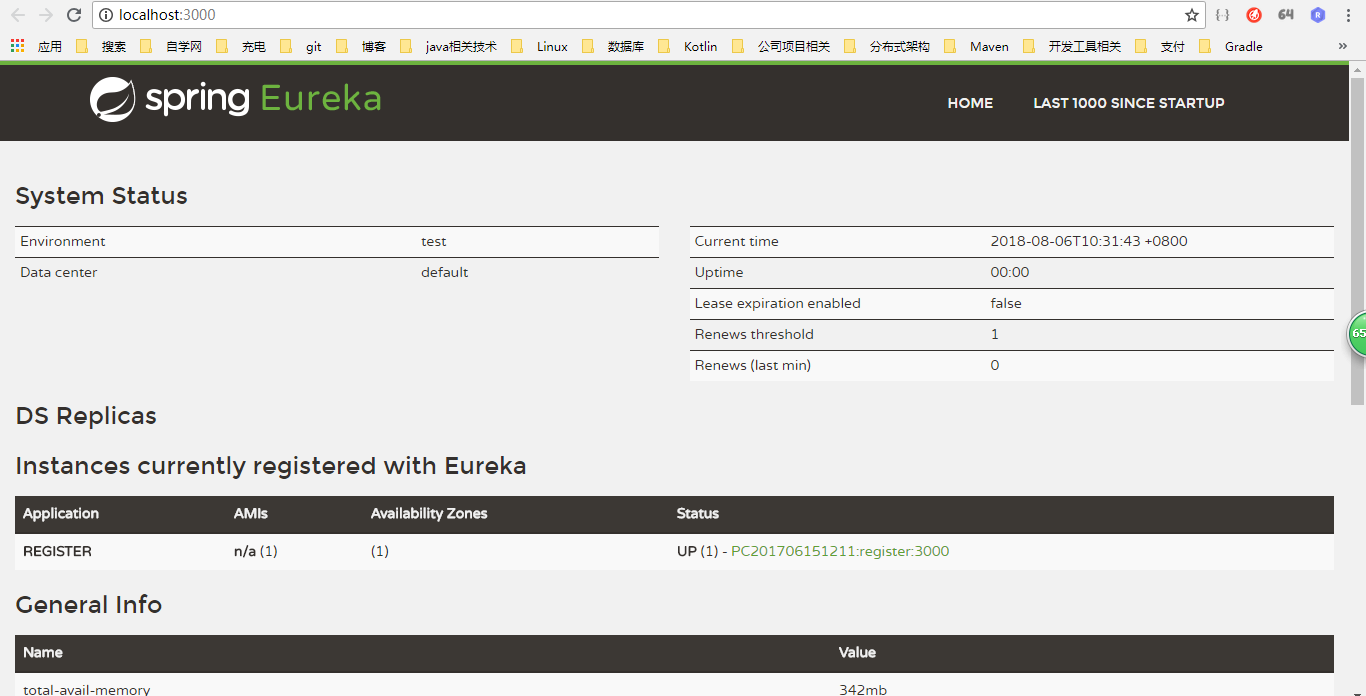
##### 2.2.2.1.2 相关配置

|  |
| --- |
| # 注册中心服务名 spring.application.name=register # 注册中心端口 server.port=3000 # 注册中心实例的主机名 eureka.instance.hostname=localhost # 是否向服务注册中心注册自己 eureka.client.register-with-eureka=true # 是否检索服务 # true 启动 抛出异常 com.netflix.discovery.shared.transport.TransportException: Cannot execute request on any known server eureka.client.fetch-registry=false # 指定注册中心位置 eureka.client.service-url.defaultZone=http://${eureka.instance.hostname}:${server.port}/eureka/ |

##### 2.2.2.1.3 启动注册中心

|  |
| --- |
| import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  */\*\*  \* 注册中心  \*  \** ***@author*** *roger  \** ***@email*** *190642964@qq.com  \** ***@create*** *2018-06-06 15:48  \*\*/* @SpringBootApplication @EnableEurekaServer public class RegistryApplication {  public static void main(String[] args){  SpringApplication.*run*(RegistryApplication.class,args);  } } |

##### 2.2.2.1.4 Eureka UI



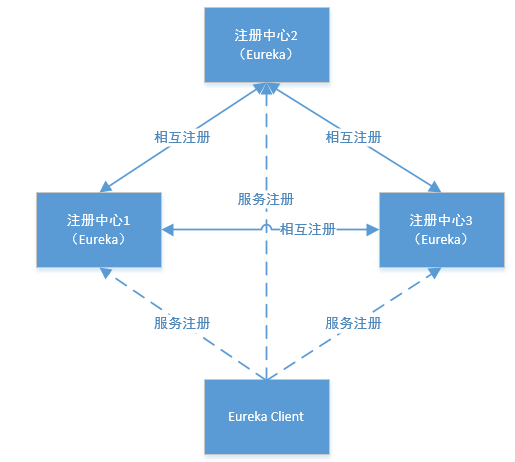
#### 2.2.2.2 高可用注册中心

##### 2.2.2.2.1 简介

考虑到发生故障的情况，服务注册红星发生故障必将会造成整个系统的瘫痪，因此需要保证服务注册中心的高可用。

Eureka Server在设计的时候就考虑了高可用设计，在Eureka服务治理设计中，所有节点既是服务的提供方，也是服务的消费方，服务注册中心也不例外。

Eureka Server 的高可用实际上就是将自己作为服务向其他服务注册中心注册自己，这样就可以形成一组互相注册的服务注册中心，以实现服务清单的互相同步，达到高可用的效果。



##### 2.2.2.2.2 集群相关配置

application-peer1.properties

|  |
| --- |
| # 注册中心服务名 spring.application.name=register # 注册中心端口 server.port=3000 # 注册中心实例的主机名 eureka.instance.hostname=localhost # 是否向服务注册中心注册自己 eureka.client.register-with-eureka=true # 是否检索服务 # true 启动 抛出异常 com.netflix.discovery.shared.transport.TransportException: Cannot execute request on any known server eureka.client.fetch-registry=false # 指定注册中心位置 eureka.client.service-url.defaultZone=http://${eureka.instance.hostname}:3001/eureka/,http://${eureka.instance.hostname}:3002/eureka/ |

application-peer2.properties

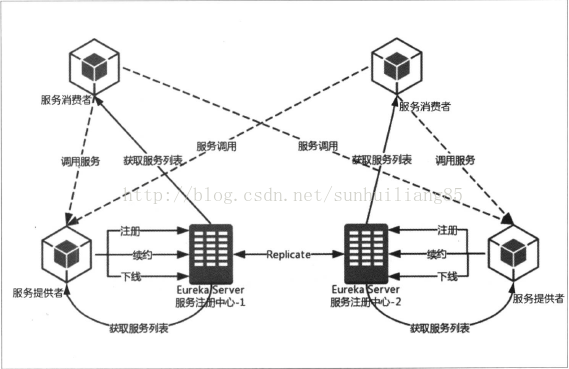
|  |
| --- |
| # 注册中心服务名 spring.application.name=register # 注册中心端口 server.port=3001 # 注册中心实例的主机名 eureka.instance.hostname=localhost # 是否向服务注册中心注册自己 eureka.client.register-with-eureka=true # 是否检索服务 # true 启动 抛出异常 com.netflix.discovery.shared.transport.TransportException: Cannot execute request on any known server eureka.client.fetch-registry=false # 指定注册中心位置 eureka.client.service-url.defaultZone=http://${eureka.instance.hostname}:3000/eureka/,http://${eureka.instance.hostname}:3002/eureka/ |

application-peer3.properties

|  |
| --- |
| # 注册中心服务名 spring.application.name=register # 注册中心端口 server.port=3002 # 注册中心实例的主机名 eureka.instance.hostname=localhost # 是否向服务注册中心注册自己 eureka.client.register-with-eureka=true # 是否检索服务 # true 启动 抛出异常 com.netflix.discovery.shared.transport.TransportException: Cannot execute request on any known server eureka.client.fetch-registry=false # 指定注册中心位置 eureka.client.service-url.defaultZone=http://${eureka.instance.hostname}:3000/eureka/,http://${eureka.instance.hostname}:3001/eureka/ |

### 2.2.3 Eureka客户端

Eureka客户端，主要处理服务的注册和发现。客户端通过注册和参数配置的方式，嵌入在客户端应用程序代码中。在应用程序程序启动时，Eureka客户端向服务注册中心注册自身提供的服务，并周期性的发送心跳来更新它的服务租约。同时，他也能从服务端查询当前注册的服务信息并把他们缓存到本地并周期的的刷新服务状态。



#### 2.2.3.1 导入相关依赖

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.0.1.RELEASE</version>  
</parent>

<dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>Finchley.RELEASE</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
</dependencyManagement>

<dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
</dependencies>

#### 2.2.3.2 相关配置

|  |
| --- |
| # 注册中心服务名 spring.application.name=client # 注册中心端口 server.port=4000 # 注册中心实例的主机名 eureka.instance.hostname=localhost # 是否向服务注册中心注册自己 eureka.client.register-with-eureka=true  # 是否检索服务 eureka.client.fetch-registry=true # 表示eureka客户端多久去拉取服务注册信息，默认为30秒 eureka.client.registry-fetch-interval-seconds=30 # 指定注册中心位置 eureka.client.service-url.defaultZone=http://${eureka.instance.hostname}:3000/eureka/ |

#### 2.2.3.3 启动客户端

|  |
| --- |
| import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  */\*\*  \* Eureka 客户端案例  \** ***@author*** *roger  \** ***@email*** *190642964@qq.com  \** ***@create*** *2018-07-26 18:39  \*\*/* @SpringBootApplication @EnableDiscoveryClient public class ClientApplication {  public static void main(String[] args) {  SpringApplication.*run*(ClientApplication.class,args);  } } |

### 2.2.5 服务时效剔

有些时候，我们的服务实例并不一定会正常下线，可能由于内存溢出、网络故障等原因使服务部能正常运行。而注册中心并未收到“服务下线”的请求，为了从服务列表中将这些无法提供服务的实例剔除，Eureka Server 在启动的时候会创建一个定时任务，默认每隔一段时间(默认为60秒)将当前清单中超时(默认为90秒)没有续约的服务剔除出去。

### 2.2.6 自我保护

服务注册到Eureka Server后，会维护一个心跳连接，告诉Eureka Server自己还活着。Eureka Server 在运行期间会统计心跳失败的比例在15分钟以内是否低于85%，如果出现低于的情况，Eureka Server 会将当前实例注册信息保护起来，让这些实例不会过期。这样做会使客户端很容易拿到实际已经不存在的服务实例，会出现调用失败的情况。因此客户端要有容错机制，比如请求重试、断路器。

|  |
| --- |
| # 确保注册中心将不可用的实例删除 eureka.server.enable-self-preservation=true |