

2017年6月6日 星期二

# Spring Cloud 肯德基全家桶~Netflix-Eureka

author: hushuang  
email : [hd1611756908@gmail.com](mailto:hd1611756908@gmail.com)  
adress: [www.dajiangdahe.com](http://www.dajiangdahe.com)

---

## 开发环境：

- 1.操作系统： MacPro
- 2.JDK1.8
- 3.Maven3.X
- 4.IDE: Eclipse luna
- 5.Spring Cloud 版本： Dalston SR1 （稳定版）

- 服务治理

- 注册中心

- 服务注册

- 服务发现与消费

一、什么是服务治理?

二、注册中心服务搭建

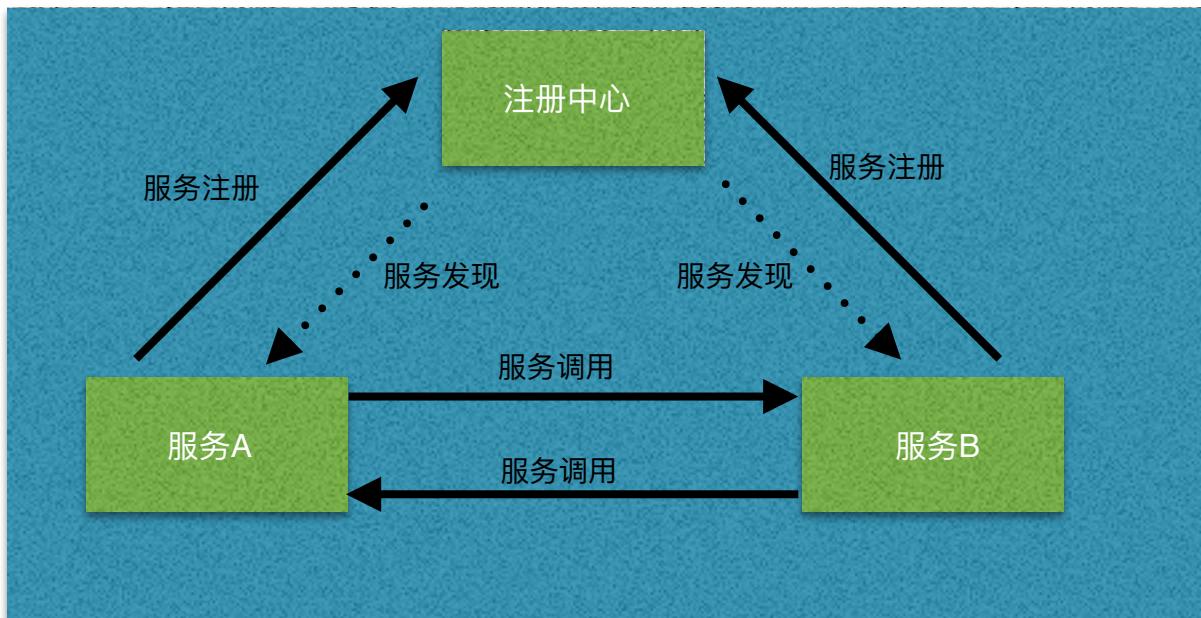
—单点注册中心的搭建

—高可用注册中心集群搭建

三、构建服务并注册到注册中心(服务注册)

四、构建服务并注册到注册中心并且可以调用到注册到注册中心的其他服务(服务发现)

图示



## 一、服务治理

**1.什么是服务治理：**字面意思就是对服务的管理。在分布式开发中如果A服务调用B服务才能完成一个业务，并且为了做到高可用B服务是一个集群，在A服务调用B服务的时候要采用负载均衡，失败等策论，都要维护一个B服务的清单(比如B服务在哪台机器什么端口号暴露的服务等)，如果服务架构简单的时候还可能手动维护，如果服务越来越复杂的时候手动维护清单列表显然是无能为力，这时出现了很多的服务治理框架(比如阿里巴巴的Dubbo)，但是Dobbo不只是服务治理那么简单，他是一个重量级的分布式框架内部集成了很多其他的功能。用一个例子说服务治理，比如有一个上传商品的服务，启动这个服务的时候就通知我有这么一个服务叫上传商品，我会记录下这个上传商品的服务的一些基本信息比如都有哪些机器，哪个端口暴露的这个服务(这个过程叫做服务注册)；如果有其他服务调用这个上传商品的服务，我就会告诉他都有哪些个机器节点有这个上传商品的服务，给他一个清单列表 如

(192.168.1.100:21880,192.168.1.101:21880,192.168.1.102:21880...)(这个叫做服务发现)，这个想调用上传商品服务的客户端会采用一些算法来选择其中一台机器进行调用。这中间我扮演的就是服务治理的功能。服务治理的核心就是服务注册和发现。

## 二、注册中心搭建

### 1.单点注册中心的搭建

- ①创建项目并添加依赖为了项目的可读性所以我这边首先创建一个父工程(netflix-eureka)，以下所有的Eureka项目都放在父工程内单点eureka服务工程名称为eureka-service。
- ②添加依赖，创建Eureka注册中心服务只需要添加spring-cloud-starter-eureka-server依赖即可，但是在父工程中要添加基础依赖

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.2.RELEASE</version>
</parent>
```

```
<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>
```

当然要实现java -jar xxx.jar功能还要添加一个插件

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

父项目中依赖添加完成之后再eureka-service项目中添加spring-cloud-starter-eureka-server就可以编写代码了

### ③编写启动程序入口类EurekaServiceApplication

```
package com.im.sc;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@EnableEurekaServer //添加@EnableEurekaServer注解开启Eureka服务
@SpringBootApplication
public class EurekaServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServiceApplication.class, args);
    }
}
```

④配置文件application.properties此项目全部采用properties格式文件因为eclipse的YAML配置不提示配置信息。

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

这样单点的注册中心服务已经构建完成

github地址: <https://github.com/hd1611756908/netfilx-eureka> 下面的eureka-service项目

启动main函数如果没有报错的话打开浏览器输入<http://localhost:8761> 就可以进入注册中心面板

### 三、服务注册

- 1.构建一个服务并注册到上面构建的单点的注册中心上项目名称eureka-client
- 2.添加依赖 **spring-cloud-starter-eureka**
- 3.编写启动入口类**EurekaClientApplication**

```
package com.im.sc;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@EnableDiscoveryClient //启动Eureka客户端
@RestController
public class EurekaClientApplication {

    @Autowired
    private DiscoveryClient discoveryClient;

    @RequestMapping(value = "/service-instances/{applicationName}")
    public List<ServiceInstance> serviceInstancesByApplicationName(@PathVariable String applicationName)
    {
        return this.discoveryClient.getInstances(applicationName);
    }

    public static void main(String[] args) {
        SpringApplication.run(EurekaClientApplication.class, args);
    }
}
```

#### 4.配置文件**application.properties**

```
spring.application.name=eureka_client
```

Tips:服务注册的配置文件只配置了一个应用名称其余的什么都没有配置，连最起码的注册中心地址都没有配置，但是启动之后会发现不会报错，并且在注册中心中会发现此服务已经注册到注册中心上去了？原因是首先单点的注册中心配置的端口好为8761，其次就是eureka.client.service-url属性给默认赋了初值http://localhost:8761/eureka/ 主要添加注解@EnableDiscoveryClient启动服务如果没有配置注册中心地址客户端就会默认的去找http://localhost:8761/eureka/ 地址。所以上面的项目才可以正常运行。两个条件缺一不可，如果单点的注册中心修改了端口号不是8761那么此服务就会注册失败。

5.启动此项目之前首先启动上面的单点注册中心，然后在启动此项目。

6. 在浏览器输入 [http://127.0.0.1:8080/service-instances/eureka\\_client](http://127.0.0.1:8080/service-instances/eureka_client) 查看浏览器返回的此应用的基本信息

以上就是服务注册eureka-client已经注册到注册中心eureka-service上。

github地址：<https://github.com/hd1611756908/netfilx-eureka> 下面的eureka-client项目

接下来就是服务发现服务。

## 四、服务发现与消费

1.服务发现：服务发现属于一种机制，主要服务于服务的注册，目录和查找，解决了繁重的配置问题，不在需要通过修改配置来解决相互之间的调用问题，只需服务的应用名称就可以使用该服务。

2.服务消费：服务消费是比较复杂的，相比于服务端，客户端的设计要复杂的多，要各个方面进行考虑，比如一个客户端调用服务的时候面对着这个功能可能有几十几百个节点，我们需要选择调用哪个？如果调用失败了我们应该怎么处理等，这样的问题一般都是在客户端进行处理。

3.Ribbon组件-客户端的负载均衡：Ribbon客户端组件提供了一系列完善的配置，比如超时，重试负载均衡等。

4.Ribbon+Eureka Eureka客户端负责完成服务发现，Ribbon负责服务消费的任务

5.创建项目eureka-ribbon并添加依赖

```
spring-cloud-starter-eureka
spring-cloud-starter-ribbon
```

6.创建程序员启动入口类EurekaRibbonClient

```
package com.im.sc;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;
@SpringBootApplication
@EnableDiscoveryClient
public class EurekaRibbonClient {
    @Bean
    @LoadBalanced
    RestTemplate restTemplate(){
        return new RestTemplate();
    }
    public static void main(String[] args) {
        SpringApplication.run(EurekaRibbonClient.class, args);
    }
}
```

创建RestTemplate 实例通过@Bean注解加入到容器中通过@LoadBalanced实现负载均衡

## 7. 创建HelloController类通过上面创建的RestTemplate实现接口调用。

```
package com.im.sc;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class HelloController {
    @Autowired
    RestTemplate restTemplate;
    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String hello() {
        return restTemplate.getForEntity("http://CLIENT-REGISTRY/hello", String.class).getBody();
    }
}
```

看上面类里面的请求Url地址" `http://client-registry/hello` 这是一个非常重要的特性，请求地址有服务实例名称+请求映射的Url组成，而不是实际的地址名称。服务实例名称可以通过Eureka面板获取到。

## 8. 配置文件application.properties

```
spring.application.name=ribbon-client
server.port=8989
eureka.client.service-url.defaultZone=http://127.0.0.1:8761/eureka/
```

## 9. 需要在eureka-client项目中定义一个服务在EurekaClientApplication类中添加

```
@RequestMapping(value="/hello",method=RequestMethod.GET)
public String hello(){
    //获取本地服务实例，虽然已经废弃，只是用它做负载均衡的测试，看负载均衡是否实现
    ServiceInstance instance = discoveryClient.getLocalServiceInstance();
    System.out.println("host: "+instance.getHost()+":"+instance.getPort()
+"==serviceID: "+instance.getServiceId());
    return "Hello Eureka Client";
}
```

## 10. 启动项目

- >启动eureka-service
- >启动eureka-client
- >启动eureka-ribbon

## 11. 打开浏览器输入 `http://localhost:8989/hello`

如果想看负载均衡是否实现，可以多启动几个eureka-client实例，然后多调用几次 `http://localhost:8989/hello` 看几个eureka-client服务打印的日志信息。

github地址：<https://github.com/hd1611756908/netfilx-eureka> 下面的eureka-ribbon

## 五、服务注册集群的高可用

单点的注册中心肯定不能用于生产环境，为了实现服务的高可用，要将服务部署成一个集群。

1.搭建集群的步骤：

- >创建项目eureka-service-high
- >添加依赖spring-cloud-starter-eureka-server, spring-cloud-starter-eureka
- >添加插件spring-boot-maven-plugin
- >创建项目启动入口类EurekaServerApplication

```
package com.im.sc;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

—>创建配置文件

—>application-peer1.properties

```
spring.application.name=eureka_server_high
server.port=8888
eureka.instance.hostname=peer1
eureka.client.service-url.defaultZone=http://127.0.0.1:7777/eureka/,http://127.0.0.1:9999/eureka/
```

—>application-peer2.properties

```
spring.application.name=eureka_server_high
server.port=7777
eureka.instance.hostname=peer2
eureka.client.service-url.defaultZone=http://127.0.0.1:8888/eureka/,http://127.0.0.1:9999/eureka/
```

—>application-peer3.properties

```
spring.application.name=eureka_server_high
server.port=9999
eureka.instance.hostname=peer3
eureka.client.service-url.defaultZone=http://127.0.0.1:7777/eureka/,http://127.0.0.1:8888/eureka/
```

—>application.properties 可以不加只是设置了一个默认值

```
spring.profiles.active=peer1
```

—>启动项目

- >通过maven命令编译
- > java -jar xxx.jar —spring.profiles.active=peer1
- > java -jar xxx.jar —spring.profiles.active=peer2
- > java -jar xxx.jar —spring.profiles.active=peer3

启动单个节点

进入Eureka面板查看各个节点的消息

<http://localhost:7777>

<http://localhost:8888>

2017年6月6日 星期二

<http://localhost:9999>

如果要进行集群的测试请自行修改服务发现和注册的配置属性

```
eureka.client.service-url.defaultZone=http://127.0.0.1:7777/eureka/,http://127.0.0.1:8888/eureka/,http://127.0.0.1:9999/eureka/
```

多个地址用逗号隔开即可。

github地址：<https://github.com/hd1611756908/netfilx-eureka> 下的 eureka-service-high

测试高可用的项目：[eureka-client-high](#)