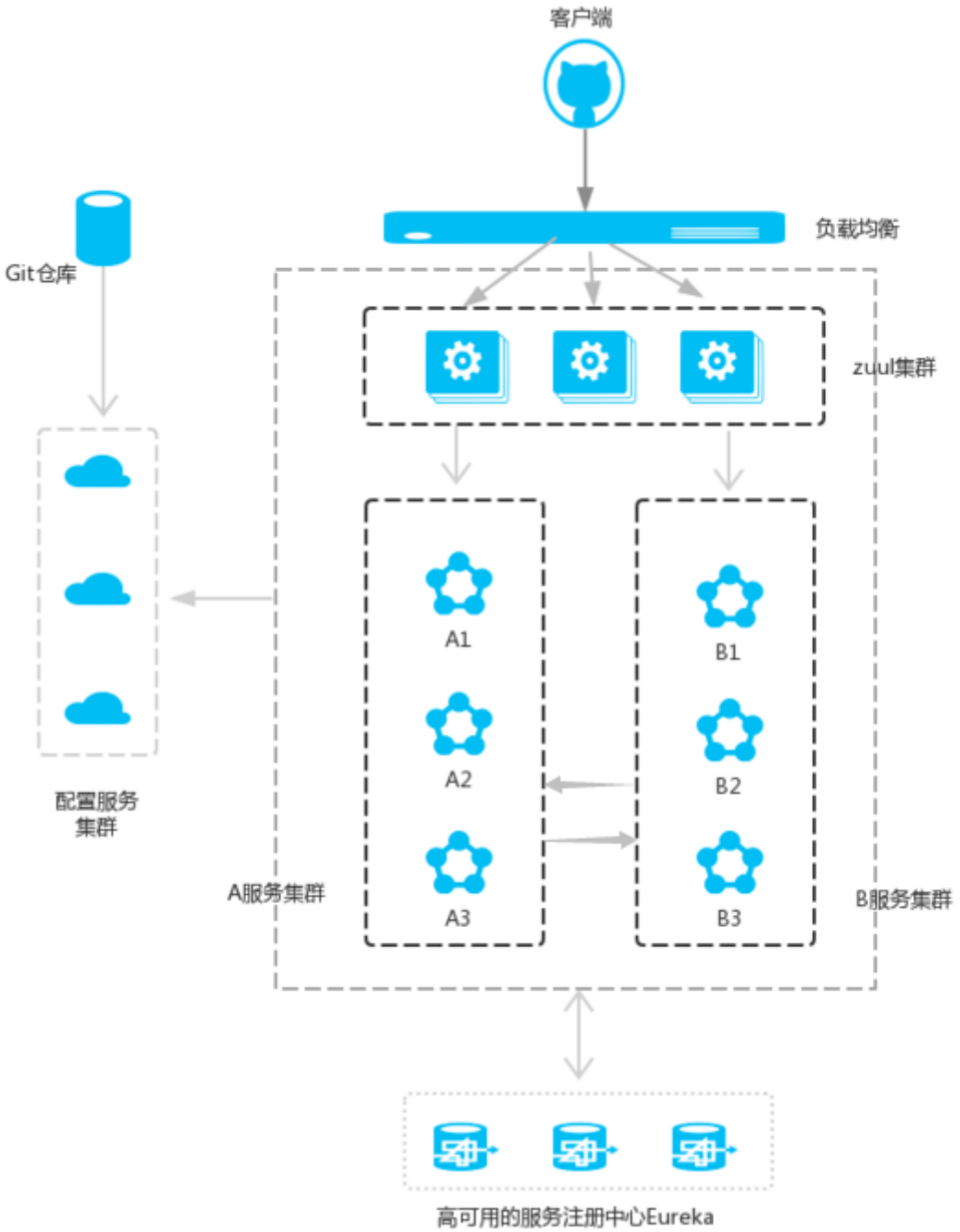
前言

在微服务架构中，具有几个基础的服务治理组件，包括服务注册与发现、服务消费、负载均衡、断路器、只能路由、配置管理等，这几个组件相互协调，共同组件一个简单的微服务系统，如下图所示：



在Spring Cloud微服务系统中，一种常见的负载均衡方式是：

1. 客户端的请求首先经过负载均衡（zuul，Ngnix）
2. 到达服务网关（zuul集群）
3. 再到达具体的服务

所有服务统一注册到服务注册中心，所有配置文件由配置服务器管理，配置服务的配置文件放在git仓库，方便开发人员随时修改。

简介

Zuul的主要功能是路由转发和过滤器。路由功能是微服务的一部分，比如/api/user转发到到user服务，/api/shop转发到到shop服务。zuul默认和Ribbon结合实现了负载均衡的功能。

准备工作

1. 启动eureka-server，端口8761
2. 启动eureka-client1，端口8762
3. 启动service-feign和service-ribbon，端口分别是8764和8765
4. 新建一个spring boot模块

创建service-zuul工程

1.pom.xml文件

```
<?xml version="1.0" encoding="UTF-8"?>
<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.gewdata</groupId>
 <artifactId>service-zuul</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>jar</packaging>

  <parent>
  <groupId>com.gewdata</groupId>
  <artifactId>eureka-server</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  </parent>

  <name>service-zuul</name>
```

```
<description>Demo project for Spring Boot</description>

<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-web</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-zuul</artifactId>
</dependency>
</dependencies>
</project>
```

2.在启动类上加上注解@EnableZuulProxy，开启zuul的功能：

```
package com.gewdata;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@SpringBootApplication
@EnableZuulProxy // 开启zuul功能
@EnableEurekaClient
@EnableDiscoveryClient
public class ServiceZuulApplication {

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

3.application.yml加上配置：

```
# 指定服务注册中心地址
```

```
eureka:
 client:
 serviceUrl:
 defaultZone: http://localhost:8761/eureka/

server:
 port: 8769

spring:
  application:
  name: service-zuul

 # 配置路由路径
 # 以/api-a/ 开头的请求都转发给service-ribbon服务；以/api-b/开头的请求都转发给service-feign服务；
 zuul:
  routes:
  api-a:
  path: /api-a/**
  serviceId: service-ribbon
  api-b:
  path: /api-b/**
  serviceId: service-feign
```

4.打开浏览器访问：

http://localhost:8769/api-a/hi?name=gewdata

```
hi gewdata,i am from port:8762
```

http://localhost:8769/api-b/hi?name=gewdata

```
hi gewdata,i am from port:8762
```

服务过滤

zuul还可以作为过滤器，做一些安全验证

1.创建一个MyFilter类继承ZuulFilter：

```
package com.gewdata.config;

import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.context.RequestContext;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```java
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;

/**
 * zuul自定义过滤器
 * @author: JunYaoWang
 * @create: 2018-12-12 09:12
 **/
@Component
public class MyFilter extends ZuulFilter {

 private static Logger log = LoggerFactory.getLogger(MyFilter.class);

 /**
 * 过滤器类型
 * pre：路由之前
 * routing：路由之时
 * post：路由之后
 * error：发送错误调用
 * @return “pre”代表过滤器类型（路由之前）
 */
 @Override
 public String filterType() {
 return "pre";
 }

 /**
 * 过滤的顺序
 * @return
 */
 @Override
 public int filterOrder() {
 return 0;
 }

 /**
 * 可以写逻辑判断，判断是否要过滤
 * @return
 * true：永远过滤
 * false：不过滤
```

```
48   */
49   @Override
50   public boolean shouldFilter() {
51   return true;
52   }
53
54   /**
55   * 过滤器的具体逻辑，可以判断该请求有没有访问权限
56   * @return
57   */
58   @Override
59   public Object run() {
60   RequestContext ctx = RequestContext.getCurrentContext();
61   HttpServletRequest request = ctx.getRequest();
62   log.info(String.format("%s >>> %s", request.getMethod(), request.getReq
uestURL().toString()));
63   Object accessToken = request.getParameter("token");
64   // 判断是否存在token，不存在就打印"token is empty"
65   if(accessToken == null){
66   log.warn("token is empty");
67   ctx.setSendZuulResponse(false);
68   ctx.setResponseStatusCode(401);
69   try{
70   ctx.getResponse().getWriter().write("token is empty");
71   }catch (Exception e){
72
73   }
74   return null;
75   }
76   log.info("ok");
77   return null;
78   }
79  }
```

2.打开浏览器访问：

http://localhost:8769/api-a/hi?name=gewdata

```
1  token is empty
```

http://localhost:8769/api-a/hi?name=gewdata&token=11

```
1  hi gewdata,i am from port:8762
```