SpringCloud之六

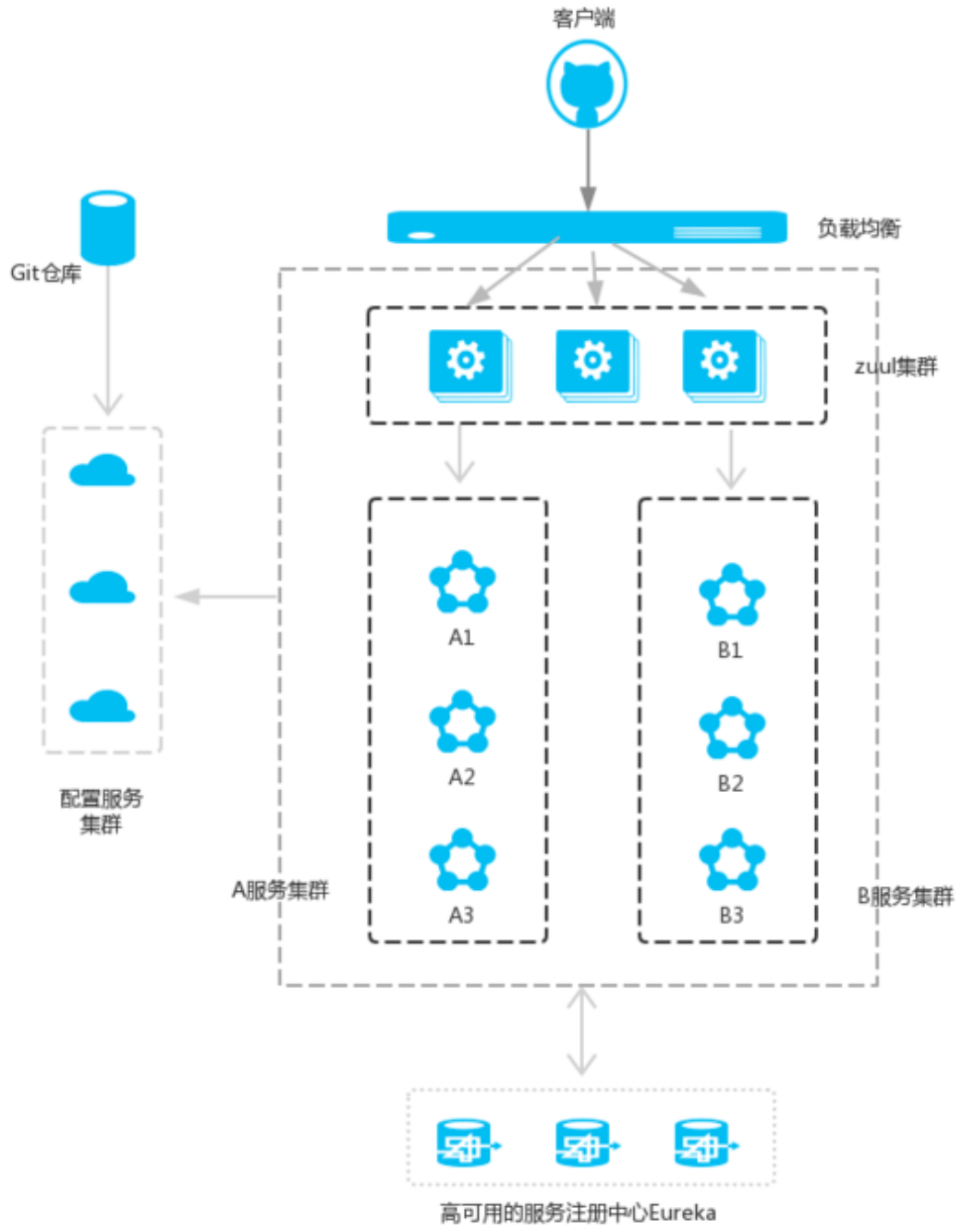
路由网关zuul

一、概述

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。

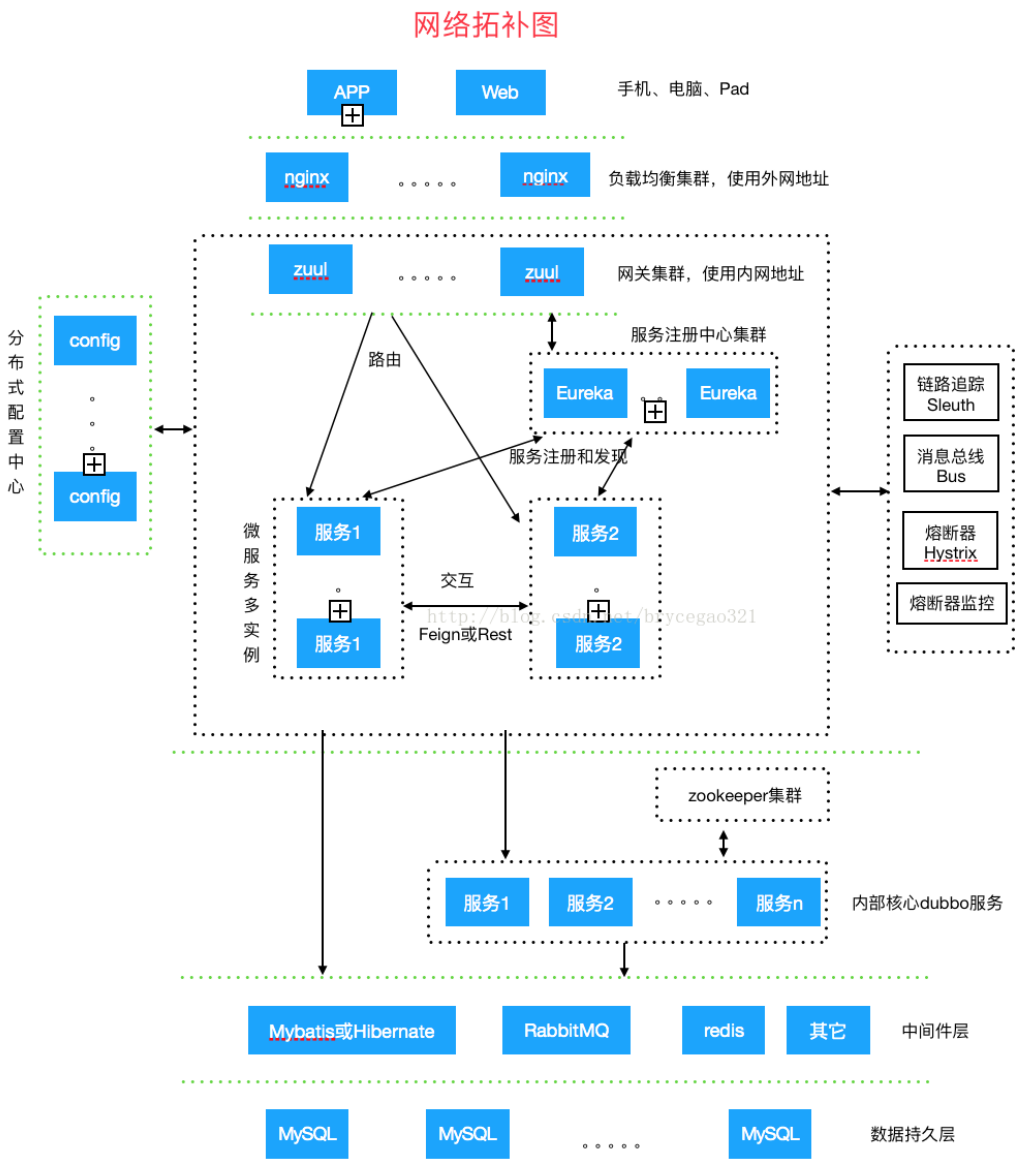
在微服务架构中，需要几个基础的服务治理组件，包括服务注册与发现、服务消费、负载均衡、断路器、智能路由、配置管理等，由这几个基础组件相互协作，共同组建了一个简单的微服务系统。

如图可以简单直观的看到zuul在微服务当中的角色：



（当然，A服务集群和B服务集群是可以互相通信的，少了箭头，见谅！）

在Spring Cloud微服务系统中，一种常见的负载均衡方式是，客户端的请求首先经过负载均衡（zuul、Ngnix），再到达服务网关（zuul集群），然后再到具体的服务。服务统一注册到高可用的服务注册中心集群，服务的所有的配置文件由配置服务管理（下一篇文章讲述），配置服务的配置文件放在git仓库，方便开发人员随时改配置。



如上图：Zuul的作用就是路由转发和过滤， 即将请求转发到微服务或拦截请求； Zuul默认集成了负载均衡功能。

在官网中的简介如图：

git上的描述如下：

Zuul

build passing

Zuul is an edge service that provides dynamic routing, monitoring, resiliency, security, and more. Please view the wiki for usage, information, HOWTO, etc https://github.com/Netflix/zuul/wiki

Here are some links to help you learn more about the Zuul Project. Feel free to PR to add any other info, presentations, etc.

Zuul是一项边缘服务，提供动态路由，监控，弹性，安全性等。请查看wiki的用法，信息，HOWTO等https://github.com/Netflix/zuul/wiki

然后就是一些其他的学习链接了。

Zuul的组件描述：

Zuul 2.x组件：

- zuul-core - Zuul 2.0的核心功能
- zuul-sample - Zuul 2.0的示例驱动程序应用程序

Zuul 1.x组件：

- zuul-core - 包含编译和执行Filters的核心功能的库
- zuul-simple-webapp - webapp，它显示了如何使用zuul-core构建应用程序的简单示例
- zuul-netflix - 用于向Zuul添加其他NetflixOSS组件的库 - 例如，使用Ribbon进行路由请求。

- zuul-netflix-webapp - 将zuul-core和zuul-netflix打包成一个易于使用的包的webapp

Zuul的一些功能描述：

Zuul使用一系列不同类型的过滤器，使我们能够快速灵活地将功能应用于我们的边缘服务。这些过滤器可帮助我们执行以下功能：

- 身份验证和安全性 - 确定每个资源的身份验证要求并拒绝不满足这些要求的请求。
- 洞察和监控 - 在边缘跟踪有意义的数据和统计数据，以便为我们提供准确的生产视图。
- 动态路由 - 根据需要动态地将请求路由到不同的后端群集。
- 压力测试 - 逐渐增加群集的流量以衡量性能。
- Load Shedding - 为每种类型的请求分配容量并删除超过限制的请求。
- 静态响应处理 - 直接在边缘构建一些响应，而不是将它们转发到内部集群
- 多区域弹性 - 跨AWS区域路由请求，以使我们的ELB使用多样化，并使我们的优势更接近我们的成员

英文描述大概是：如下

```
Authentication
Insights
Stress Testing
Canary Testing
Dynamic Routing
Service Migration
Load Shedding
Security
Static Response handling
 Active/Active traffic management
```

Zuul通过使用其他Netflix OSS组件，为我们提供了很多洞察力，灵活性和弹性：

- Hystrix用于包含对我们的起源的呼叫，这使我们能够在发生问题时清除流量并确定其优先级

- Ribbon是来自Zuul的所有出站请求的客户端，Zuul提供有关网络性能和错误的详细信息，以及处理均匀负载分配的软件负载平衡
- Turbine实时聚合精细的指标，以便我们能够快速观察并对问题做出反应
- Archaius处理配置并提供动态更改属性的功能

二、Zuul的应用

依照惯例，废话不多说，直接开干！！！

1、准备工作

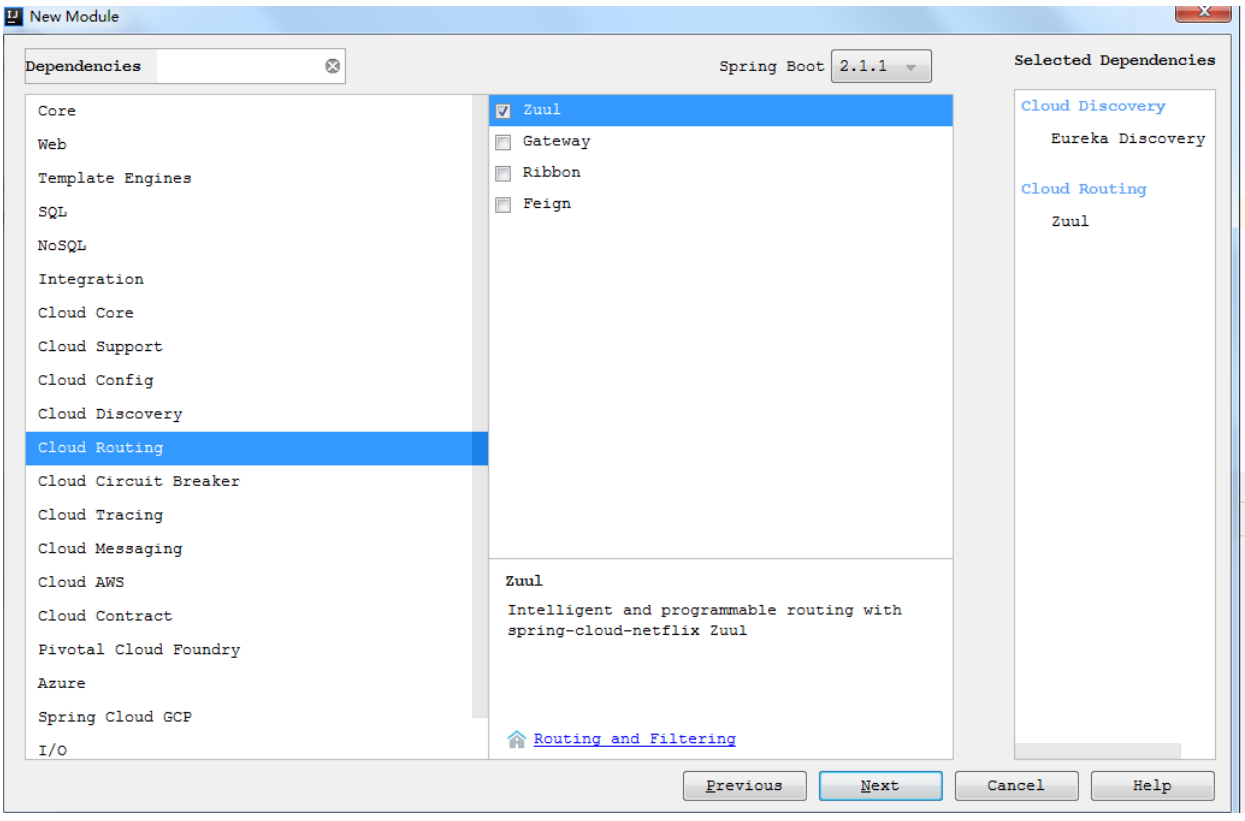
启动之前的eureka-server，login-service，ribbon-service，feign-service项目

2、创建Zuul服务项目

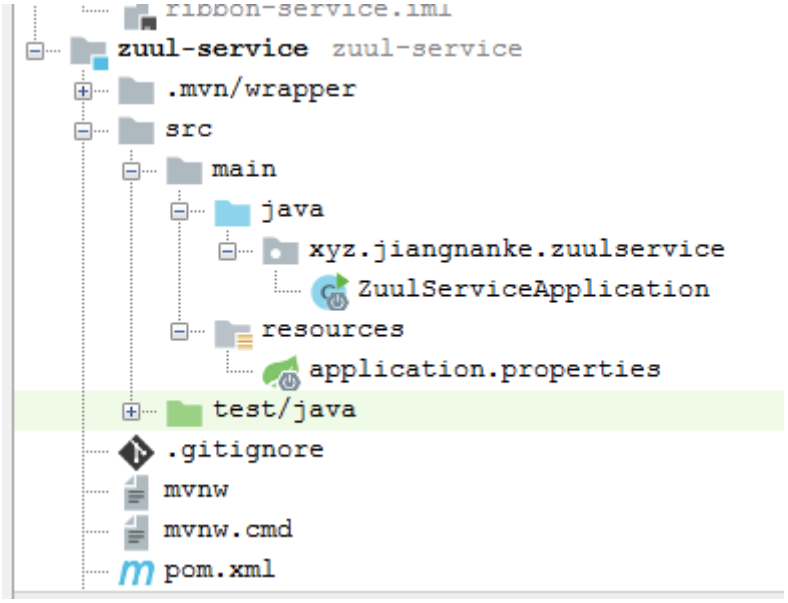
和之前一样创建一个spring-boot项目，我将之命名为zuul-service，同样关联到main下面。

2.1、关联依赖包

要包含eureka-client，以及zuul。当然由于是web项目也需要加上web依赖。



创建好后项目结构如图：



2.2、配置环境

首先是**配置pom.xml文件**，如下：

```
<?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>xyz.jiangnanke</groupId>
 <artifactId>zuul-service</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <name>zuul-service</name>
 <description>Demo project for Spring Boot</description>

  <parent>
 <groupId>xyz.jiangnanke</groupId>
 <artifactId>main</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 </parent>

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

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

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
</dependencies>

</project>
```

其次**配置application.yml文件：**

如下：

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

spring:
 application:
 name: zuul-service

server:
 port: 8769

zuul:
 routes:
 api-r:
 path: /api-r/**
 serviceId: ribbon-service
 api-f:
 path: /api-f/**
 serviceId: feign-service
```

指定服务注册中心的地址为http://localhost:8761/eureka/，服务的端口为8769，服务名为zuul-service；以/api-r/ 开头的请求都转发给ribbon-service服务；以/api-f/开头的请求都转发给feign-service服务；

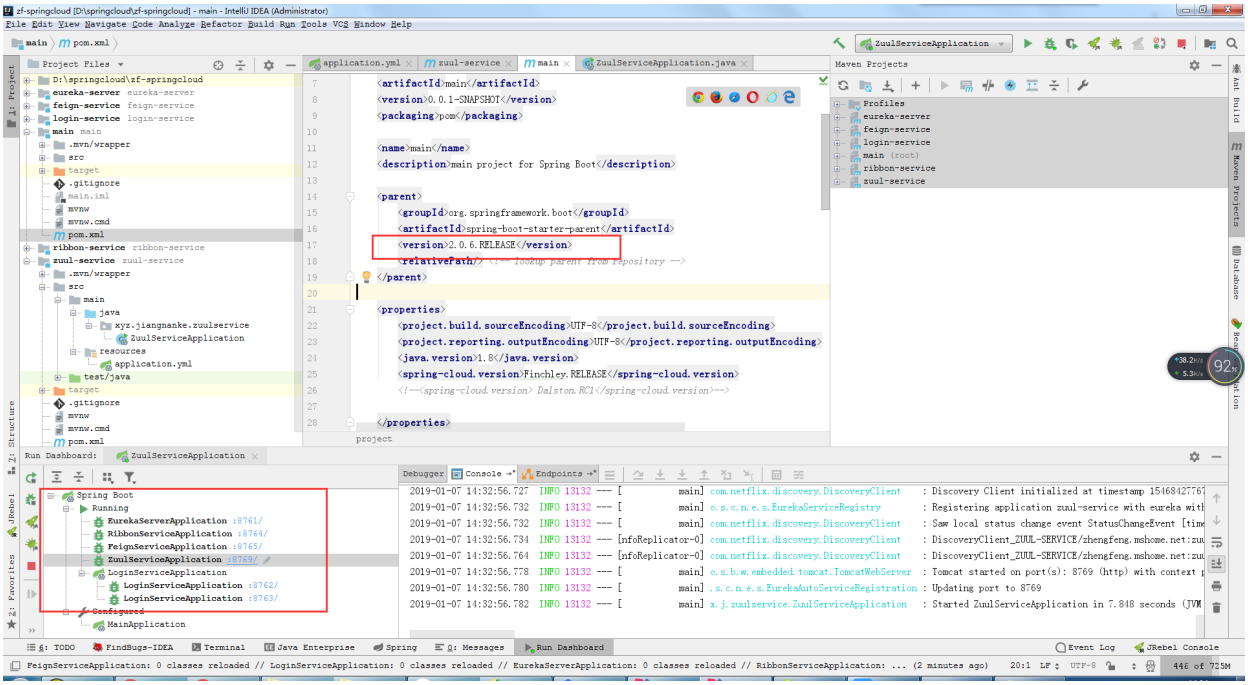
然后就是**配置启动类：**

ZuulServiceApplication.java类如下：

```java
package xyz.jiangnanke.zuulservice;

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
 @EnableEurekaClient
 @EnableDiscoveryClient
 public class ZuulServiceApplication {

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

 }

```

2.3、启动

启动之后，得到的效果，如图：



注意：由于zuul-service启动的时候报错，所以我将spring-boot的版本降低到2.0.6了启动不报错（网上说添加# main: # allow-bean-definition-

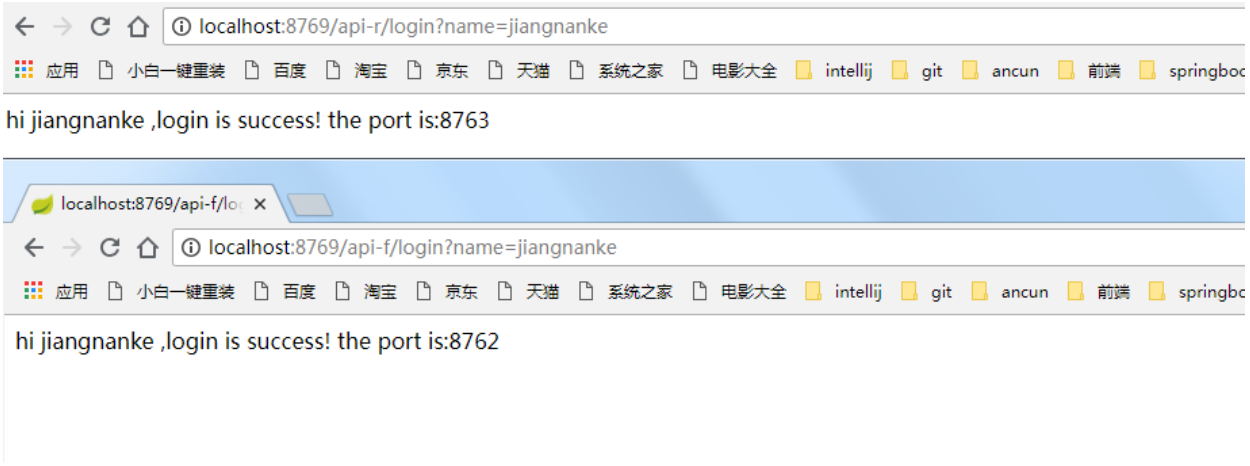
overriding: true 后来发现还是报错）

运行结束之后，依次访问：

http://localhost:8769/api-r/login?name=jiangnanke

http://localhost:8769/api-f/login?name=jiangnanke

得到的结果如图：



都有结果，说明zuul-service起到了路由的作用。

三、路由过滤

zuul经常在实际应用中不仅只是路由，而且还会做过滤作用，做一些安全验证之类的。接下来继续操作修改项目。

3.1、实现一个自定义的ZuulFilter

创建一个LoginPreFilter继承ZuulFilter，代码如下：

```
1  package xyz.jiangnanke.zuulservice.system.filter;
2
3  import com.netflix.zuul.ZuulFilter;
4  import com.netflix.zuul.context.RequestContext;
5  import org.slf4j.Logger;
6  import org.slf4j.LoggerFactory;
7  import org.springframework.stereotype.Component;
8
9  import javax.servlet.http.HttpServletRequest;
10
11 /**
12  * @Auther: zhengfeng
13  * @Date: 2019\1\7 0007 14:43
14  * @Description:
```

```
 */
@Component
public class LoginPreFilter extends ZuulFilter {

 private static Logger log = LoggerFactory.getLogger(LoginPreFilter.class);
 @Override
 public String filterType() {
 return "pre";
 }

 @Override
 public int filterOrder() {
 return 0;
 }

 @Override
 public boolean shouldFilter() {
 return true;
 }

 @Override
 public Object run() {
 RequestContext ctx = RequestContext.getCurrentContext();
 HttpServletRequest request = ctx.getRequest();
 log.info(String.format("%s >>> %s", request.getMethod(), request.getRequestURL().toString()));
 Object accessToken = request.getParameter("token");
 if(accessToken == null) {
 log.warn("token is empty");
 ctx.setSendZuulResponse(false);
 ctx.setResponseStatusCode(401);
 try {
 ctx.getResponse().getWriter().write("token is empty");
 }catch (Exception e){}

 return null;
 }
 log.info("ok");
 return null;
 }
```

```
54  }
55
```

注意：

1、filterType：返回一个字符串代表过滤器的类型，在zuul中定义了四种不同生命周期的过滤器类型，具体如下：

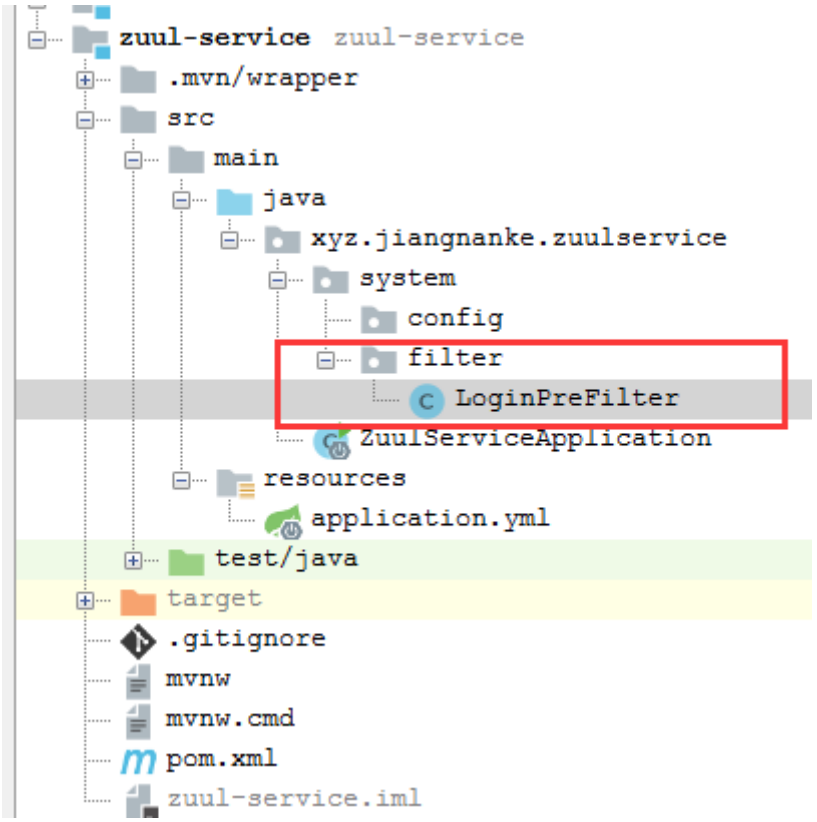
1. **pre：**路由之前
2. **routing：**路由之时
3. **post：** 路由之后
4. **error：**发送错误调用

2、filterOrder：过滤的顺序，数字越小，优先级越高

3、shouldFilter：这里可以写逻辑判断，是否要过滤，本文true,永远过滤。

4、run：过滤器的具体逻辑。可用很复杂，包括查sql，nosql去判断该请求到底有没有权限访问。

项目结构如图：



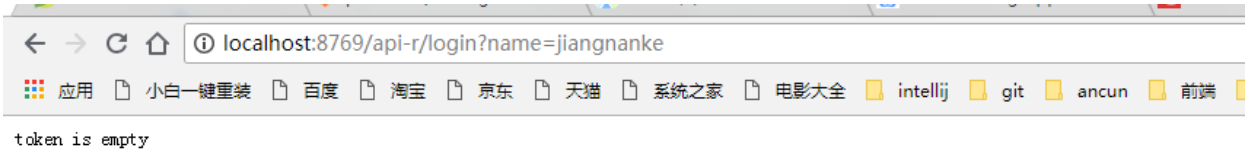
3.2、启动运行

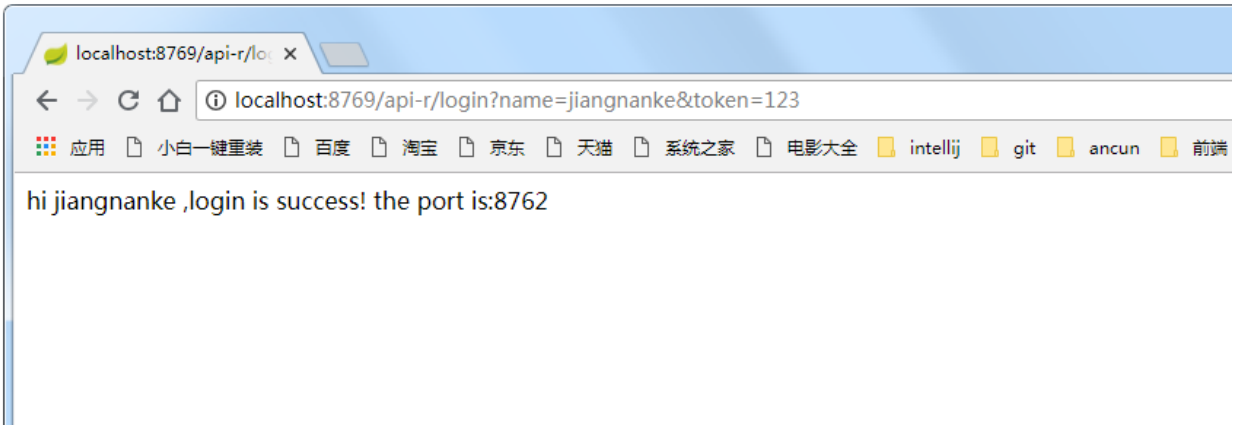
启动运行后依次访问：

http://localhost:8769/api-r/login?name=jiangnanke

http://localhost:8769/api-r/login?name=jiangnanke&token=123

做了一个小比较，如图：





从结果可以知道，LoginPreFilter已经起到了拦截验证的作用了。

至于后面的黑名单，白名单等功能的实现之类的，我们后面有时间再去处理。

附录：

1、ZuulFilter

Filter 我们用的比较多，大部分是Servlet中的Filter,通常我们用来做一些拦截，权限验证之类的工作。

今天介绍的Filter是Zuul中提供的，跟我们之前使用的Servlet Filter不太一样。

Zuul中提供Filter的作用有哪些，我觉得分为如下几点：

- 网关是暴露在外面的，必须要进行权限控制
- 可以针对服务做控制，在路由的时候处理，比如服务降级
- 防止爬虫，利用Filter对请求进行过滤
- 流量控制，只允许最高的并发量，保护后端的服务
- 灰度发布，可以针对不用的用户进行路由来实现灰度

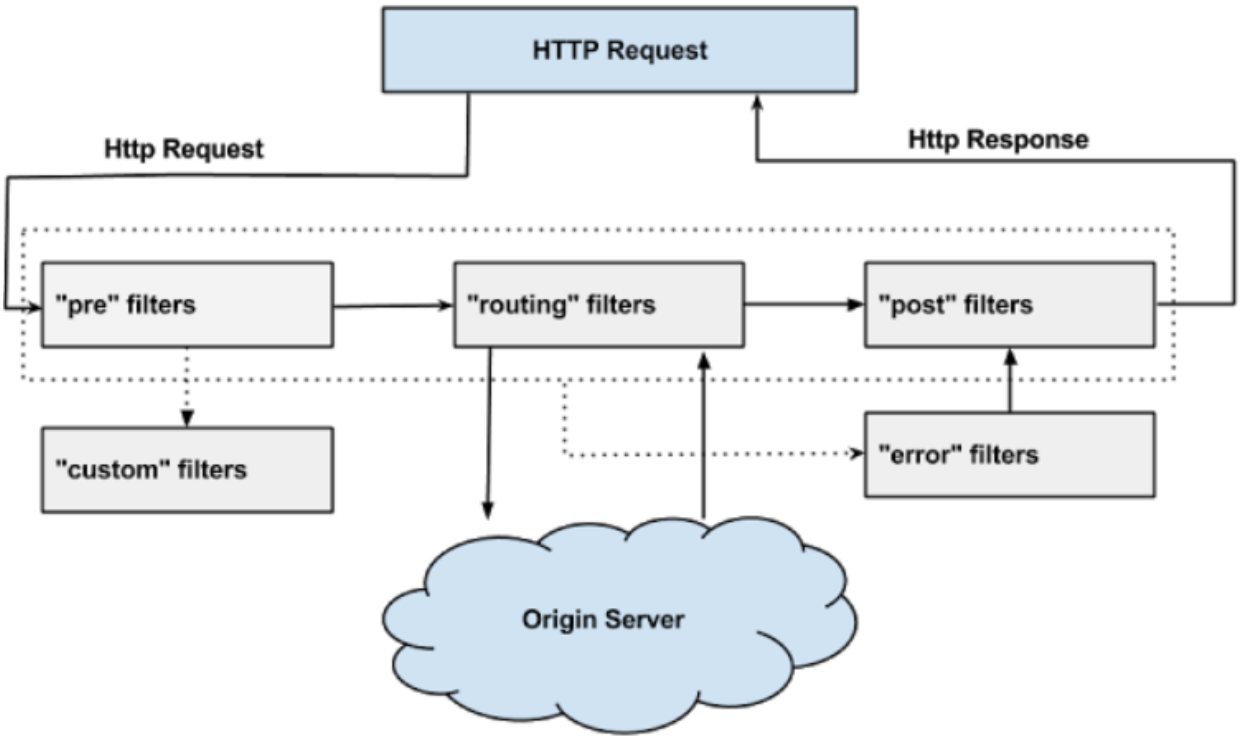
2、Filter的种类

- pre：可以在请求被路由之前调用
- route：在路由请求时候被调用

- post：在route和error过滤器之后被调用
- error：处理请求时发生错误时被调用

3、Filter的生命周期

关于zuul的Filter的生命周期，见下图



4、git上的链接

由于国家网关的显示，所以基本上git上面的学习链接都是用不了的，除非翻墙操作一下。

参考资料：

https://springcloud.cc/

https://github.com/Netflix/zuul

https://blog.csdn.net/f

http://cxytiandi.com/blog/detail/12632orezp/article/details/81041012

https://github.com/yinjihuan/smconf