SpringCloud是由多个独立的项目集合而成的，每个项目都是独立的，各自进行自己的迭代和版本发布。

# 服务的注册和发现Netflix Eureka

作用：实现服务治理(服务的注册和发现)

简介：Spring Cloud Eureka是Spring Cloud Netflix项目下的服务治理模块。

由两个模块组成，Eureka服务端和Eureka客户端

支持集群部署

在应用启动时，Eureka客户端向服务端注册自己的服务信息，同时将服务端的服务信息缓存到本地。客户端会和服务端周期性的进行心跳交互，以更新服务租约和服务信息。



Zookeeper当master节点挂掉时，就会从其他节点进行leader选举，问题就是选举的时间过长，且在选举期间zookeeper集群时不可用的。这就导致了在选举期间注册服务瘫痪。能够保持一致性。

Eureka处理方式是首先保证可用性，每个节点都是平等的，其中一个挂掉，其他节点可以继续提供服务，只要有一个节点是正常的就行，只不过查到的数据可能不是最新的，所以不能保证一致性。

## 服务中心Eureka

注册中心主要有zookeeper和eureka。Dubbo+zookeeper构建服务，spring cloud推荐使用eureka作为注册中心构建服务。但也可以使用其他的注册中心。

## 注册服务Provider

## 发现服务Consumer

## 服务中心高可用Eureka

# 客服端负载均衡Netflix Ribbon

Spring Cloud Ribbon是一个基于HTTP 和 TCP的客户端负载均衡工具，基于Netflix实现。通过SpringCloud的封装，可以轻松的面向服务的REST模板请求自动转换成客户端负载均衡的服务调用。

Spring Cloud Ribbon虽然只是一个工具类框架，他不像服务注册中心，配置中心。API网关那样需要独立部署，但他几乎存在于每一个Spring Cloud构建的微服务和基础设施中。因为微服务之间的调用、API网关的请求转发等内容，实际上都是通过Ribbon实现的。Feign，也是基于Ribbon实现的工具。

# 断路器Hystrix

# Feign

Feign整合了Ribbon和Hystrix，以后就可以不再显示的使用Hystrix和Ribbon，直接使用Feign就行了。

Feign是一个声明式的Web Service客户端，他的目的就是让Web Service调用更加简单，Feign提供了HTTP请求模板，通过编写简单的接口和插入注解，就可以定义好HTTP请求的参数、格式、地址等信息。

Feign完全代理HTTP请求，只需要像调用方法那样调用他就可以完成服务请求及相关处理。

可插拔的注解支持，包括Feign注解和JAX-RS注解；

支持可插拔的HTTP编码器和解码器

支持Hystrix和他的Fallback

支持Ribbon的负载均衡

支持HTTP请求和相应的压缩



# 服务网关——Netflix Zuul

网站后端所有请求的前门，核心是一系列的filters，其作用可以类比Servlet框架的Filter，或者AOP。

Zuul是Spring Cloud全家桶中的微服务API网关。

各个Filter之间没有直接的联系，但是都通过RequestContext共享一些状态数据。

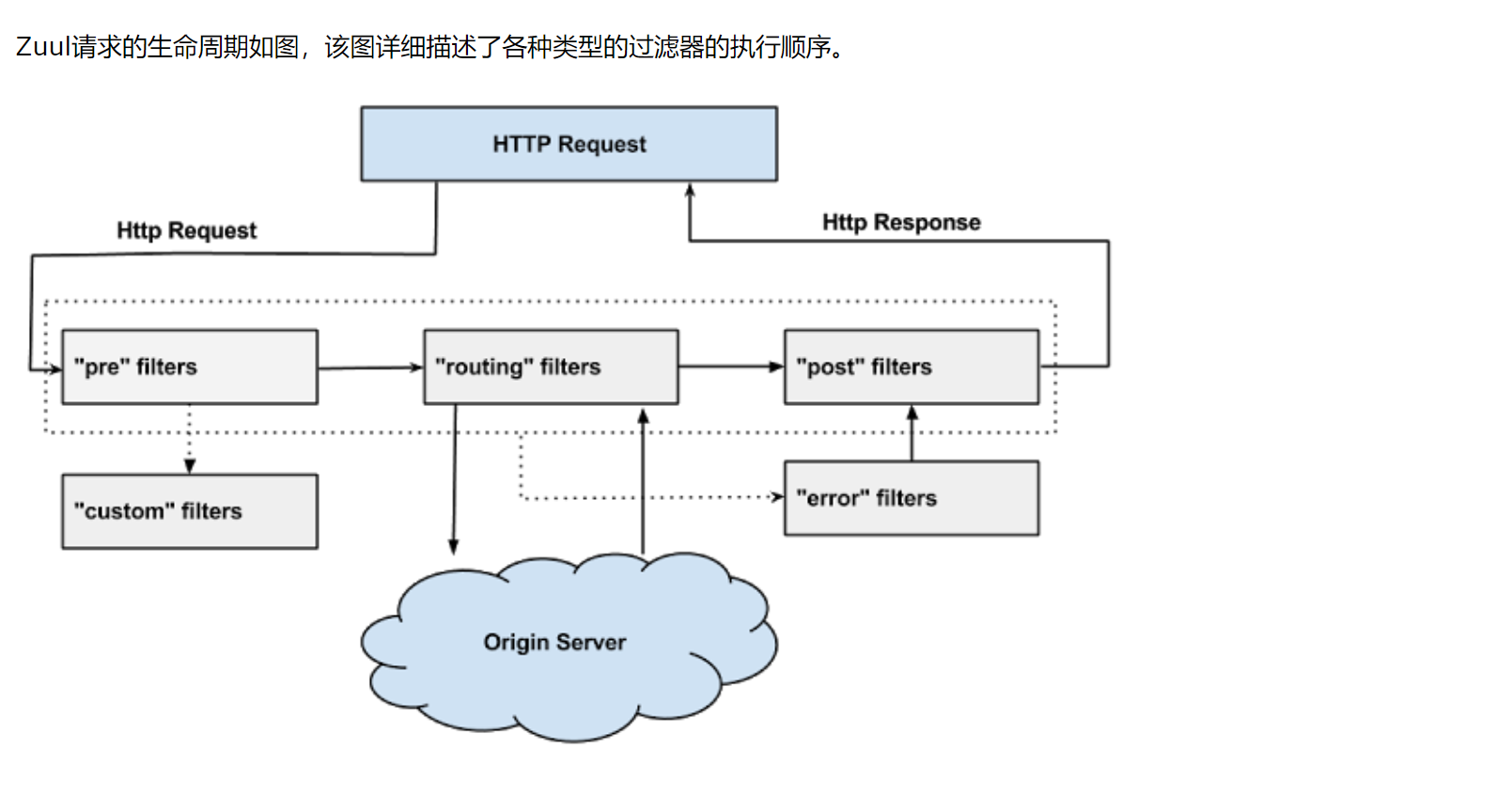
Zuul大部分功能是通过过滤器实现的，Zuul中定义了四种标准的过滤器类型。

1. PRE：这种过滤器在请求被路由之前调用，可以利用这种过滤器进行身份验证、在集群中选择请求的微服务、记录调试信息等。
2. Routing：请求路由到微服务，用于构建发送给微服务的请求，并使用Apache HttpClient或者Ribbon请求微服务。
3. POST：路由到微服务之后执行，这种过滤器可以用来为响应添加标准的HTTP Header、收集统计信息和指标、将响应从微服务发送到客户端。
4. ERROR：在其他阶段发生错误时执行该过滤器。

内置特殊的过滤器：

自定义的过滤器：

可以创建自定义的过滤器，



ZuulServlet类似SpringMvc的DispatcherServlet，所有的Request都要经过ZuulServlet的处理

三个核心的方法preRoute(),route(), postRoute()，zuul对request处理逻辑都在这三个方法里

ZuulServlet交给ZuulRunner去执行。

由于ZuulServlet是单例，因此ZuulRunner也仅有一个实例。

ZuulRunner直接将执行逻辑交由FilterProcessor处理，FilterProcessor也是单例，其功能就是依据filterType执行filter的处理逻辑

FilterProcessor对filter的处理逻辑。

## Zuul的容错和回退

前面已经说过，Hystrix可以实现微服务的容错和回退，其实Zuul默认已经整合了Hystrix。

实现ZuulFallbackProvider接口，并实现getRoute和fallbackResponse方法。

## Zuul回退

|  |
| --- |
| package com.pupeiyuan.fallback;  import java.io.ByteArrayInputStream;  import java.io.IOException;  import java.io.InputStream;  import java.nio.charset.Charset;  import org.springframework.cloud.netflix.zuul.filters.route.ZuulFallbackProvider;  import org.springframework.http.HttpHeaders;  import org.springframework.http.HttpStatus;  import org.springframework.http.MediaType;  import org.springframework.http.client.ClientHttpResponse;  import org.springframework.stereotype.Component;  @Component  public class ConsumerFallbackProvider implements ZuulFallbackProvider {  @Override  public ClientHttpResponse fallbackResponse() {  return new ClientHttpResponse() {    @Override  public HttpHeaders getHeaders() {  // headers设定  HttpHeaders headers = new HttpHeaders();  MediaType mt = new MediaType("application","json", Charset.forName("UTF-8"));  headers.setContentType(mt);  return headers;  }    @Override  public InputStream getBody() throws IOException {  // 响应体  return new ByteArrayInputStream(("moveserver微服务不可用，请稍后再试。" + ConsumerFallbackProvider.this.getRoute()).getBytes());  }    @Override  public String getStatusText() throws IOException {  return HttpStatus.BAD\_REQUEST.getReasonPhrase();  }    @Override  public HttpStatus getStatusCode() throws IOException {  return HttpStatus.BAD\_REQUEST;  }    @Override  public int getRawStatusCode() throws IOException {  return HttpStatus.BAD\_REQUEST.value();  }    @Override  public void close() {  // TODO Auto-generated method stub    }  };  }  @Override  public String getRoute() {  // TODO Auto-generated method stub  return "MOVESERVER";  }  } |

## Zuul过滤器

|  |
| --- |
| package com.pupeiyuan.fallback;  import javax.servlet.http.HttpServletRequest;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.stereotype.Component;  import com.netflix.zuul.ZuulFilter;  import com.netflix.zuul.context.RequestContext;  @Component  public class PreZuulFilter extends ZuulFilter {  private static final Logger LOGGER = LoggerFactory.getLogger(PreZuulFilter.class);  @Override  public boolean shouldFilter() {  return true;  }  @Override  public Object run() {  HttpServletRequest request = RequestContext.getCurrentContext().getRequest();  String host = request.getRemoteHost();  PreZuulFilter.LOGGER.info("请求的host:{}", host);  return null;  }  @Override  public String filterType() {  return "pre";  }  @Override  public int filterOrder() {  return 1;  }  } |

## 高可用策略

如果Zuul也注册到了Eureka上，那么zuul就已经实现了高可用配置，这种情况和普通的服务消费者和服务提供者高可用是一致的。

## Zuul和Nginx

都可以实现负载均衡、反向代理、过滤请求、实现网关的效果。

Nginx是C语言开发的，Zuul是Java语言开发的。

Zuul采用Ribbon+Eureka实现本地负载均衡，适合微服务中实现网关。Nginx采用服务器实现负载均衡，适合于服务器负载均衡。

# 配置中心Spring Cloud Config

是一种用来动态获取git、svn、本地的配置文件的工具，