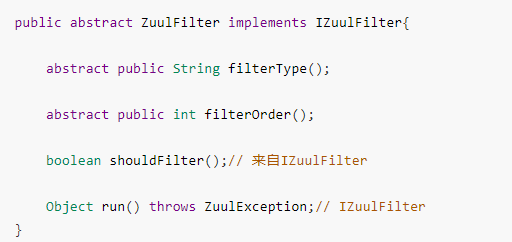
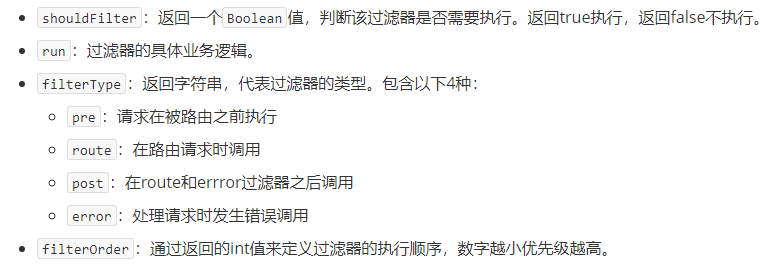
# 过滤器

Zuul作为网关的其中一个重要功能，就是实现请求的鉴权。而这个动作我们往往是通过Zuul提供的过滤器来实现的。

## ZuulFilter

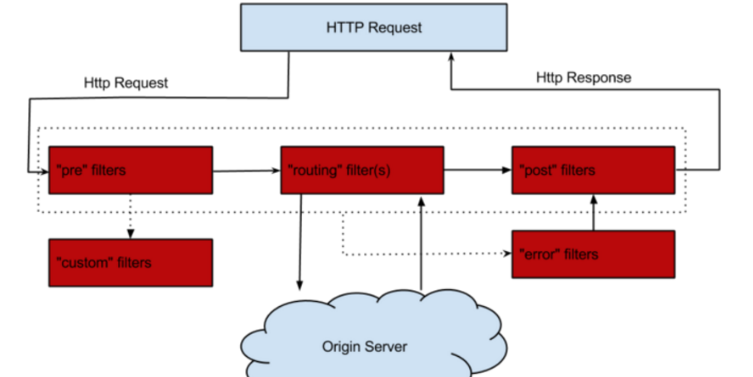
ZuulFilter是过滤器的顶级父类。在这里我们看一下其中定义的4个最重要的方法：



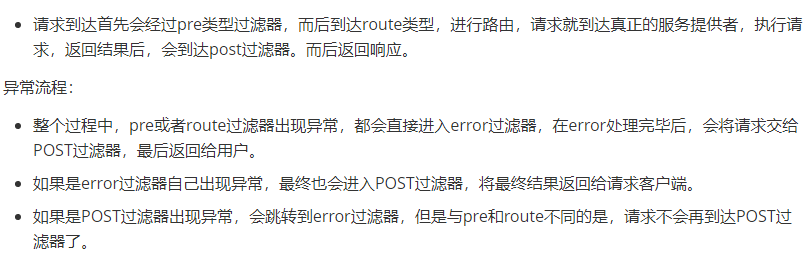


## 过滤器执行生命周期

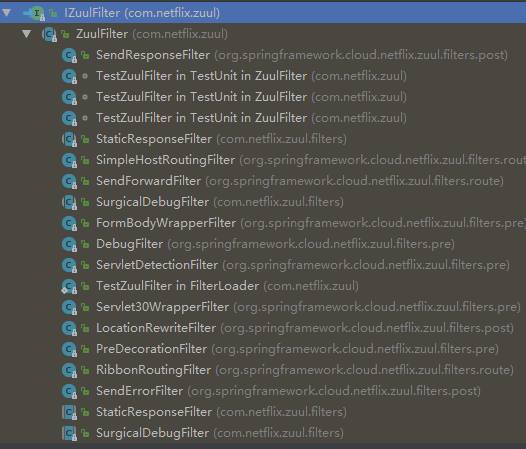
这张是Zuul官网提供的请求生命周期图，清晰的表现了一个请求在各个过滤器的执行顺序。



正常流程：



所有内置过滤器列表：



## 使用场景

场景非常多：

- 请求鉴权：一般放在pre类型，如果发现没有访问权限，直接就拦截了

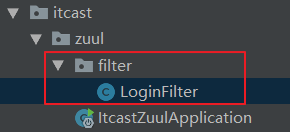
- 异常处理：一般会在error类型和post类型过滤器中结合来处理。

- 服务调用时长统计：pre和post结合使用。

# 自定义过滤器

接下来我们来自定义一个过滤器，模拟一个登录的校验。基本逻辑：如果请求中有access-token参数，则认为请求有效，放行。

## 定义过滤器类



内容：

@Component

public class LoginFilter extends ZuulFilter {

/\*\*

\* 过滤器类型，前置过滤器

\* @return

\*/

@Override

public String filterType() {

return "pre";

}

/\*\*

\* 过滤器的执行顺序

\* @return

\*/

@Override

public int filterOrder() {

return 1;

}

/\*\*

\* 该过滤器是否生效

\* @return

\*/

@Override

public boolean shouldFilter() {

return true;

}

/\*\*

\* 登陆校验逻辑

\* @return

\* @throws ZuulException

\*/

@Override

public Object run() throws ZuulException {

// 获取zuul提供的上下文对象

RequestContext context = RequestContext.getCurrentContext();

// 从上下文对象中获取请求对象

HttpServletRequest request = context.getRequest();

// 获取token信息

String token = request.getParameter("access-token");

// 判断

if (StringUtils.isBlank(token)) {

// 过滤该请求，不对其进行路由

context.setSendZuulResponse(false);

// 设置响应状态码，401

context.setResponseStatusCode(HttpStatus.SC\_UNAUTHORIZED);

// 设置响应信息

context.setResponseBody("{\"status\":\"401\", \"text\":\"request error!\"}");

}

// 校验通过，把登陆信息放入上下文信息，继续向后执行

context.set("token", token);

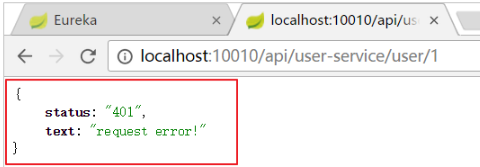
return null;

}

}

## 测试

没有token参数时，访问失败：



添加token参数后：

