过滤器Filter

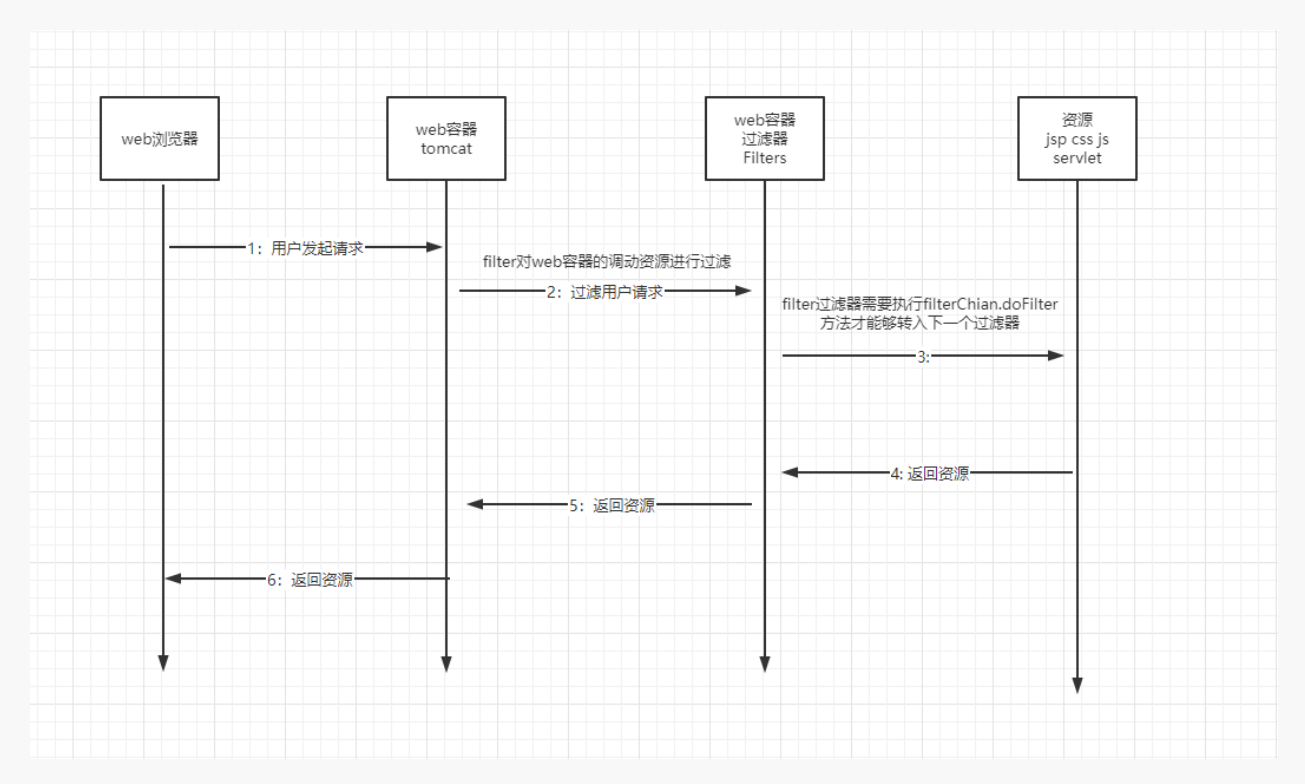
概述

过滤器是Web开发中很实用的意向技术，程序员可以通过过滤器对web服务资源，

- 静态资源 比如：静态HTML、静态图片，js或者css等，
- 动态资源比如：JSP，Servlet等进行拦截器

从而实现一些特殊的需求，比如：URL的访问权限，过滤的敏感词汇，压缩响应信息等，

完整的执行过程



Springboot如何定义Filter呢(默认机制)

1：定义一个过滤器WebFilter

```
package com.kuangstudy.config.filter;

import lombok.extern.slf4j.Slf4j;
import org.springframework.core.annotation.Order;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import javax.servlet.annotation.WebInitParam;
import java.io.IOException;

/**
 * @author 飞哥
 * @Title: 学相伴出品
 * @Description: 飞哥B站地址：https://space.bilibili.com/490711252
 * 记得关注和三连哦！
 * @Description: 我们有一个学习网站：https://www.kuangstudy.com
 * @date 2021/12/27 21:24
 */
@WebFilter(urlPatterns = "/api/*", filterName = "WebTestFilter", initParams = {
        @WebInitParam(name = "encoding", value = "UTF-8")
})
@Order(1)
@Slf4j
public class WebTestFilter implements Filter {


    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
```

```
29          // 初始化 获取拦截器配置的参数
30          String encoding =
    filterConfig.getInitParameter("encoding");
31          log.info("encoding:{}",encoding);
32      }
33
34      @Override
35      public void doFilter(ServletRequest servletRequest,
    ServletResponse servletResponse, FilterChain filterChain)
    throws IOException, ServletException {
36          log.info("1------>doFilter进来了....");
37          // 下面的代码很关键，如果没有就代表请求进入过滤器，就终止了，就
    不会返回资源给用户
38          filterChain.doFilter(servletRequest,
    servletResponse);// 1: 如果还有过滤器，会进入到下一个过滤器  2：如果
    没有过滤器。直接去访问servlet /jsp/ 静态资源
39      }
40
41      @Override
42      public void destroy() {
43          log.info("过滤执行完毕了.....");
44      }
45  }
46
```

2：开启springboot对filter和监听器的支持

增加@ServletComponentScan开关

```
1  package com.kuangstudy;
2
3  import org.springframework.boot.SpringApplication;
4  import
    org.springframework.boot.autoconfigure.SpringBootApplication;
5  import
    org.springframework.boot.web.servlet.ServletComponentScan;
6
7  @SpringBootApplication
```

```
@ServletComponentScan
public class SpringbootInterceptorApplication {

    public static void main(String[] args) {

 SpringApplication.run(SpringbootInterceptorApplication.class
, args);
    }

}

```

3：请求一个springmvc的路由资源

http://localhost:8987/api/index

查看结果得出结论如下：

```
2021-12-27 21:31:14.269  INFO 20460 --- [nio-8987-exec-1] c.k.config.filter.WebTestFilter          : 1------>doFilter进来了....
2021-12-27 21:31:27.664  INFO 20460 --- [nio-8987-exec-1] c.k.config.handler.LoginInterceptor      : 1---LoginInterceptor--preHandle----->
2021-12-27 21:31:27.664  INFO 20460 --- [nio-8987-exec-1] c.k.c.handler.PermissionInterceptor      : 1---PermissionInterceptor--preHandle----->
2021-12-27 21:31:27.679  INFO 20460 --- [nio-8987-exec-1] com.kuangstudy.web.HelloWorldController  : 2------>index
2021-12-27 21:31:27.687  INFO 20460 --- [nio-8987-exec-1] c.k.c.handler.PermissionInterceptor      : 3---PermissionInterceptor--postHandle----->
2021-12-27 21:31:27.687  INFO 20460 --- [nio-8987-exec-1] c.k.config.handler.LoginInterceptor      : 3--LoginInterceptor---postHandle----->
2021-12-27 21:31:27.741  INFO 20460 --- [nio-8987-exec-1] c.k.c.handler.PermissionInterceptor      : 4---PermissionInterceptor --afterCompletion----->
2021-12-27 21:31:27.741  INFO 20460 --- [nio-8987-exec-1] c.k.config.handler.LoginInterceptor      : 4---LoginInterceptor--afterCompletion----->
```

4：总结

过滤器Filter先于拦截器Intetercptor执行。

03、自定义配置类 + 注册过滤器

自定义配置类配置过滤器就是定义一个：过滤器实现Filter接口，然后此过滤器的以@Bean的形式注册到配置类中。以及它他规则全部在@Bean定义的过滤器的方法中进行完成。

```
package com.kuangstudy.config.filter;

import org.apache.catalina.filters.RequestFilter;
import
org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.servlet.Filter;

/**
 * @author 飞哥
 * @Title: 学相伴出品
 * @Description: 飞哥B站地址:
https://space.bilibili.com/490711252
 * 记得关注和三连哦!
 * @Description: 我们有一个学习网站：https://www.kuangstudy.com
 * @date 2021/12/27 21:40
 */
@Configuration
public class RequestTestFilterConfiguration {

    @Bean
    public FilterRegistrationBean filterRequeset() {
        // 1: 定义一个过滤器的注册bean
        FilterRegistrationBean<Filter>
filterFilterRegistrationBean = new FilterRegistrationBean<>
();
        // 2: 初始化自定义filter
        RequestTestFilter requestTestFilter = new
RequestTestFilter();
        // 注册过滤器到ioc容器中
```

```
28
 filterFilterRegistrationBean.setFilter(requestTestFilter);
29          // 设置过滤器规则
30
 filterFilterRegistrationBean.addUrlPatterns("/api/**");
31          // 设置过滤器名字
32
 filterFilterRegistrationBean.setName("RequestTestFilter");
33          // 设置参数
34
 filterFilterRegistrationBean.addInitParameter("encoding",
"GBK");
35          // 设置执行顺序
36          filterFilterRegistrationBean.setOrder(3);
37          // 返回
38          return filterFilterRegistrationBean;
39      }
40
41      @Bean
42      public FilterRegistrationBean filterRequeset2() {
43          // 1: 定义一个过滤器的注册bean
44          FilterRegistrationBean<Filter>
filterFilterRegistrationBean = new FilterRegistrationBean<>
();
45          // 2: 初始化自定义filter
46          RequestTestFilter2 requestTestFilter2 = new
RequestTestFilter2();
47          // 注册过滤器到ioc容器中
48
 filterFilterRegistrationBean.setFilter(requestTestFilter2);
49          // 设置过滤器规则
50
 filterFilterRegistrationBean.addUrlPatterns("/api/**");
51          // 设置过滤器名字
52
 filterFilterRegistrationBean.setName("RequestTestFilter2");
53          // 设置参数
```

```
54
    filterFilterRegistrationBean.addInitParameter("encoding",
"GBK");
55          // 设置执行顺序
56          filterFilterRegistrationBean.setOrder(2);
57          // 返回
58          return filterFilterRegistrationBean;
59      }
60
61  }
62
```

过滤器和拦截器的区别

- 拦截器不依赖与servlet容器，过滤器依赖与servlet容器。
- 过滤器(Filter)是容器的基于函数回调的执行。拦截器：是基于java的反射机制的，

  filter是由web容器来调用，拦截器是springmvc的ioc容器来调用。

  ```
  1  Filter的执行是一种基于函数回调执行，代表就是：所以过滤器的都由容器来统一执行。web/servlet容器。tomcat在启动的时候---初始化servlet，Filter 在初始化Filter时候就根据接口找到所有的自定义的filter，将其加载内存中，然后请求进入之间把加载号的filter一个个执行触发。
  ```

- 过滤器(Filter) 几乎可以顾虑所以的请求（不论静态（js/css/img/html）和动态(jsp/servlet)）。而拦截器只能拦截controller中定义的接口路由请求。拦截器不处理静态资源

  > 番外：那么*springmvc*处理静态资源机制是什么：过滤器 拦截器所以 */static/**

- 拦截器可以访问controller中方法和上下文，以及值栈信息，而过滤器不能获取。
- 在controller的生命周期中，

    - 过滤器只能在容器初始化的时候调用一次init方法，
    - 拦截器必须要在请求的时候才会执行和触发。
- 拦截器可以获取ioc各种的bean,根据需求进行业务处理，但是过滤不支持这种方式。