


Logo

- [首页](#)
- [文章](#)
- [关注](#)
- [订阅专栏](#)

- [网站导航](#)
[学院](#) [博客](#) [下载](#) [家园](#) [论坛](#) [CTO训练营](#) [CTO俱乐部](#) [WOT](#) [51CTO](#)
- [注册](#)
- [登录](#)
- [手机阅读](#)


扫一扫体验手机阅读

- [搜索](#)
- [写文章](#)

原创

基于SpringMVC的拦截器（Interceptor）和过滤器（Filter）的区别与联系



[pangfc](#) 关注 1人评论 [42159人阅读](#) 2016-12-07 11:45:28

一 简介

(1) 过滤器：

依赖于servlet容器。在实现上基于函数回调，可以对几乎所有请求进行过滤，但是缺点是一个过滤器实例只能在容器初始化时调用一次。使用过滤器的目的是用来做一些过滤操作，获取我们想要获取的数据，比如：在过滤器中修改字符编码；在过滤器中修改HttpServletRequest的一些参数，包括：过滤低俗文字、危险字符等

关于过滤器的一些用法可以参考我写过的这些文章：

- [继承HttpServletRequestWrapper以实现在Filter中修改HttpServletRequest的参数](https://www.zifangsky.cn/677.html)： <https://www.zifangsky.cn/677.html>
- [在SpringMVC中使用过滤器（Filter）过滤容易引发XSS的危险字符](https://www.zifangsky.cn/683.html)： <https://www.zifangsky.cn/683.html>

(2) 拦截器：

依赖于web框架，在SpringMVC中就是依赖于SpringMVC框架。在实现上基于Java的反射机制，属于面向切面编程（AOP）的一种运用。由于拦截器是基于web框架的调用，因此可以使用Spring的依赖注入（DI）进行一些业务操作，同时一个拦截器实例在一个controller生命周期之内可以多次调用。但是缺点是只能对controller请求进行拦截，对其他的一些比如直接访问静态资源的请求则没办法进行拦截处理

关于过滤器的一些用法可以参考我写过的这些文章：

- [在SpringMVC中使用拦截器（interceptor）拦截CSRF***（修）](https://www.zifangsky.cn/671.html)： <https://www.zifangsky.cn/671.html>

在线
客服

- SpringMVC中使用Interceptor+Cookie实现在一定天数之内自动登录: <https://www.zifangsky.cn/700.html>

二 多个过滤器与拦截器的代码执行顺序

如果在一个项目中仅仅只有一个拦截器或者过滤器，那么我相信相对来说理解起来是比较容易的。但是我们是否思考过：如果一个项目中有多个拦截器或者过滤器，那么它们的执行顺序应该是什么样的？或者再复杂点，一个项目中既有多个拦截器，又有多个过滤器，这时它们的执行顺序又是什么样的呢？

下面我将用简单的代码来测试说明：

(1) 先定义两个过滤器：

i) 过滤器1：

```
package cn.zifangsky.filter;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.filter.OncePerRequestFilter;

public class TestFilter1 extends OncePerRequestFilter {

    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {
        //在DispatcherServlet之前执行
        System.out.println("#####TestFilter1 doFilterInternal executed#####");
        filterChain.doFilter(request, response);
        //在视图页面返回给客户端之前执行，但是执行顺序在Interceptor之后
        System.out.println("#####TestFilter1 doFilter after#####");
        //
        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

}
```

ii) 过滤器2：

```
package cn.zifangsky.filter;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.filter.OncePerRequestFilter;

public class TestFilter2 extends OncePerRequestFilter {

    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {
        System.out.println("#####TestFilter2 doFilterInternal executed#####");
        filterChain.doFilter(request, response);
        System.out.println("#####TestFilter2 doFilter after#####");
    }

}
```

iii) 在web.xml中注册这两个过滤器：

```
<!-- 自定义过滤器: testFilter1 -->
<filter>
    <filter-name>testFilter1</filter-name>
    <filter-class>cn.zifangsky.filter.TestFilter1</filter-class>
</filter>
<filter-mapping>
    <filter-name>testFilter1</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!-- 自定义过滤器: testFilter2 -->
<filter>
```

在线
客服

```

<filter-name>testFilter2</filter-name>
<filter-class>cn.zifangsky.filter.TestFilter2</filter-class>
</filter>
<filter-mapping>
    <filter-name>testFilter2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

(2) 再定义两个拦截器:

i) 拦截器1, 基本拦截器:

```

package cn.zifangsky.interceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

public class BaseInterceptor implements HandlerInterceptor{

    /**
     * 在DispatcherServlet之前执行
     */
    public boolean preHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2) throws Exception {
        System.out.println("*****BaseInterceptor preHandle executed*****");
        return true;
    }

    /**
     * 在controller执行之后的DispatcherServlet之后执行
     */
    public void postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView arg3)
        throws Exception {
        System.out.println("*****BaseInterceptor postHandle executed*****");
    }

    /**
     * 在页面渲染完成返回给客户端之前执行
     */
    public void afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)
        throws Exception {
        System.out.println("*****BaseInterceptor afterCompletion executed*****");
        Thread.sleep(10000);
    }
}

```

ii) 指定controller请求的拦截器:

```

package cn.zifangsky.interceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

public class TestInterceptor implements HandlerInterceptor {

    public boolean preHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2) throws Exception {
        System.out.println("*****TestInterceptor preHandle executed*****");
        return true;
    }

    public void postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView arg3)
        throws Exception {
        System.out.println("*****TestInterceptor postHandle executed*****");
    }

    public void afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)
        throws Exception {
        System.out.println("*****TestInterceptor afterCompletion executed*****");
    }
}

```

iii) 在SpringMVC的配置文件中注册这两个拦截器:

```

<!-- 拦截器 -->
<mvc:interceptors>
    <!-- 对所有请求都拦截, 公共拦截器可以有多个 -->
    <bean name="baseInterceptor" class="cn.zifangsky.interceptor.BaseInterceptor" />

```

在线
客服

```

<!-- <bean name="testInterceptor" class="cn.zifangsky.interceptor.TestInterceptor" /> -->
<mvc:interceptor>
    <!-- 对/test.html进行拦截 -->
    <mvc:mapping path="/test.html" />
    <!-- 特定请求的拦截器只能有一个 -->
    <bean class="cn.zifangsky.interceptor.TestInterceptor" />
</mvc:interceptor>
</mvc:interceptors>

```

(3) 定义一个测试使用的controller:

```

package cn.zifangsky.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class TestController {

    @RequestMapping("/test.html")
    public ModelAndView handleRequest(){
        System.out.println("-----TestController executed-----");
        return new ModelAndView("test");
    }
}

```

(4) 视图页面test.jsp:

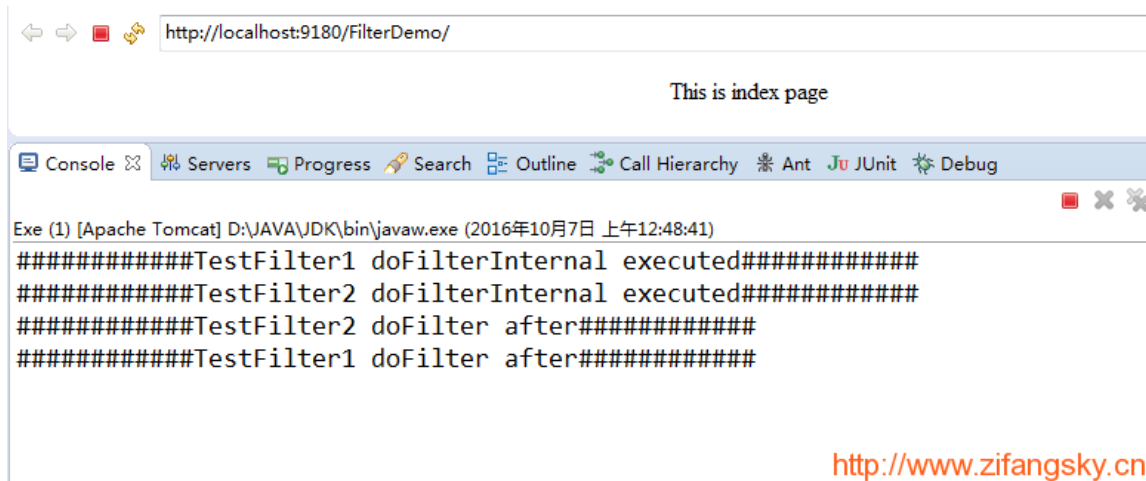
```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+ "://" + request.getServerName() + ":" + request.getServerPort() + path + "/";
%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<base href="<%=basePath%>">
<title>FilterDemo</title>
</head>
<body>
    <%
        System.out.println("test.jsp is loading");
    %>
    <div align="center">
        This is test page
    </div>
</body>
</html>

```

(5) 测试效果:

启动此测试项目，可以看到控制台中输出如下:



在线
客服

这就说明了过滤器的运行是依赖于servlet容器的，跟springmvc等框架并没有关系。并且，多个过滤器的执行顺序跟xml文件中定义的先后关系有关

接着清空控制台中的输出内容并访问: <http://localhost:9180/FilterDemo/test.html>

可以看到，此时的控制台输出结果如下：

↶ ↷ 🚫 🚧

http://localhost:9180/FilterDemo/test.html

This is test page

Console Servers Progress Search Outline Call Hierarchy Ant JUnit Debug

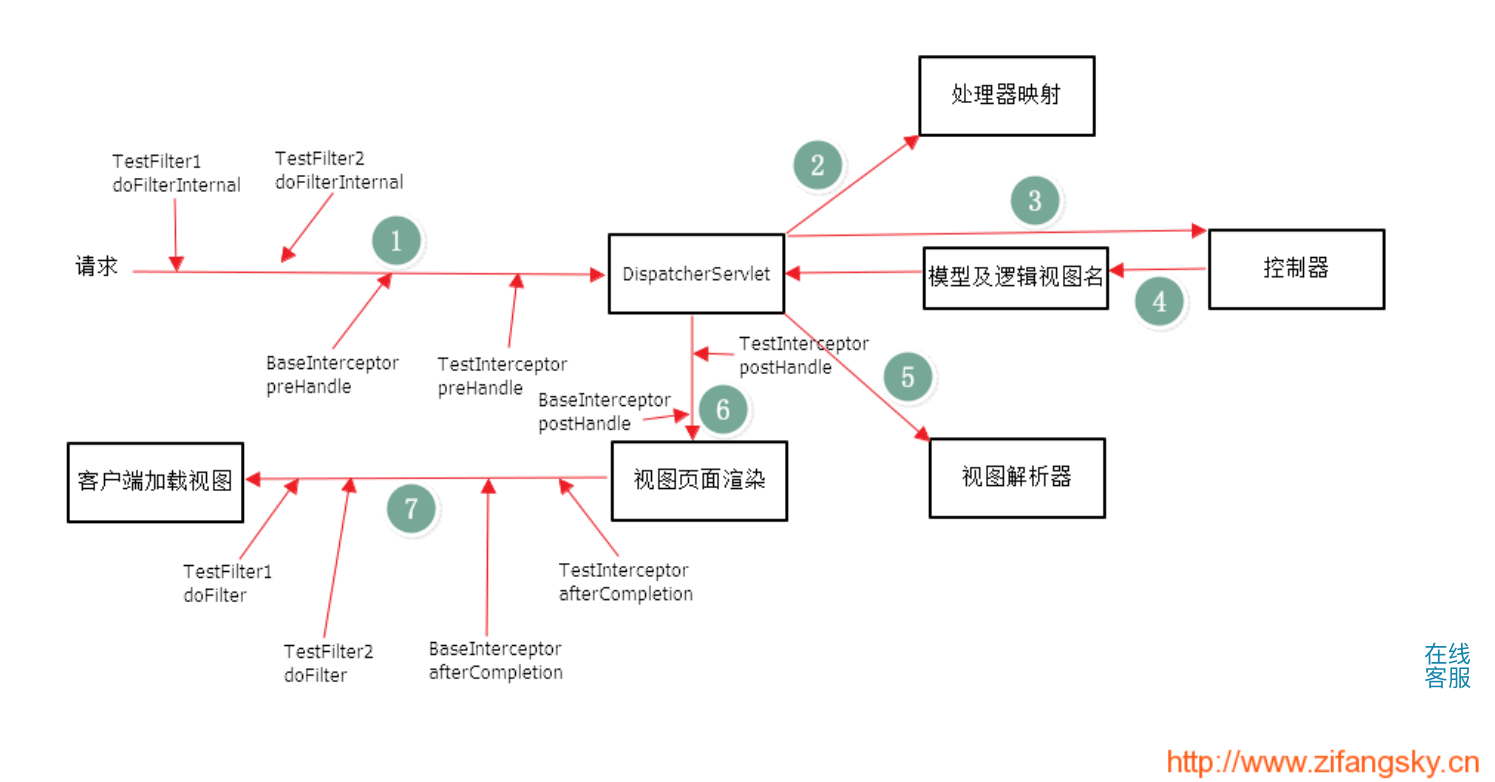
Exe (1) [Apache Tomcat] D:\JAVA\JDK\bin\javaw.exe (2016年10月7日 上午12:48:41)

#####TestFilter1 doFilterInternal executed#####
#####TestFilter2 doFilterInternal executed#####
*****BaseInterceptor preHandle executed*****
*****TestInterceptor preHandle executed*****
-----TestController executed-----
*****TestInterceptor postHandle executed*****
*****BaseInterceptor postHandle executed*****
test.jsp is loading
*****TestInterceptor afterCompletion executed*****
*****BaseInterceptor afterCompletion executed*****
#####TestFilter2 doFilter after#####
#####TestFilter1 doFilter after#####

http://www.zifangsky.cn

相信从这个打印输出，大家就可以很清晰地看到有多个拦截器和过滤器存在时的整个执行顺序了。当然，对于过个拦截器它们之间的执行顺序跟在SpringMVC的
配置文件中定义的先后顺序有关

注：对于整个SpringMVC的执行流程来说，如果加上上面的拦截器和过滤器，其最终的执行流程就如下图所示：



PS：这个图是我用画图软件画出来的，将就着看吧^_^

参考：

- 《Spring实战（第4版）》第5章
- <https://blog.51cto.com/983836259/1880286>

在线客服

http://www.zifangsky.cn

PS：上面图片中的水印是我个人博客的域名，因此还请管理员手下留情不要给我标为“转载文章”，谢谢！！

©著作权归作者所有：来自51CTO博客作者pangfc的原创作品，如需转载，请注明出处，否则将追究法律责任
[Spring interceptor Filter Spring](#)

2

分享



微博 QQ 微信

收藏

[上一篇：SpringMVC中使用Inte...](#) [下一篇：Java基础系列19：使用JXL...](#)



[pangfc](#)

286篇文章，188W+人气，7粉丝

个人博客：[zifangsky.cn](#)

关注



提问和评论都可以，用心的
回复会被更多人看到和认可

Ctrl+Enter 发布

发布

取消

1条评论

[按时间倒序](#) [按时间正序](#)

推荐专栏[更多](#)



[Web网站安全评估分析及防御](#)

企业级网安运维

共30章 | [simeon2005](#)

¥ 51.00 350人订阅

[订 阅](#)



JavaScript全栈工程师养成记

20年软件开发心法

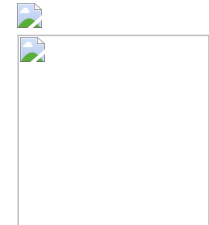
共17章 | [边城](#)

¥ 51.00 326人订阅

[订 阅](#)

猜你喜欢

[我的友情链接](#) [在Spring中使用@Configuration注解加载JavaConfig配置](#) [spring_boot + mybatis + layui + shiro后台权限管理系统](#) [Angular 6集成Spring_Boot 2, Spring_Security, JWT和CORS](#) [响应式Spring的道法术器（Spring_WebFlux 快速上手 + 全面介绍）](#) [深入理解spring注解之@ComponentScan注解](#) [（4）Reactor 3快速上手——响应式Spring的道法术器](#) [给你一份Spring_Boot核心知识清单](#) [Spring切入点表达式常用写法](#) [Spring_Boot 2.0\(四\)：使用 Docker 部署 Spring_Boot](#) [Spring_MVC Controller单例陷阱](#) [Spring_Boot 2.0\(五\)：感受 Docker 魅力，](#) [排解决多应用部署之疼](#) [jHipster开发中对配置文件.yo-ce.json分析](#) [AJAX入门学习-2：基于JS的AJAX实现\(以Django为例\)](#) [Quartz集群实战与原理分析](#) [ngrinder 压力测试实践（二）](#) [groovy 脚本实战](#) [javascript基础修炼（8）——指向FP世界的箭头函数](#) [一统江湖的大前端（5）](#) [editorconfig + eslint——你的代码里藏着你的优雅](#) [webpack4.0各个击破（9）—— karma篇](#) [【Recorder.js+百度语音识别】全栈方案技术细节](#)



扫一扫,领取大礼包

2

1

1

分享



关注
[pangfc](#)



在线客服