1、过滤器（Filter）和拦截器（Interceptor）

（-1）认证模块：所有需要认证的服务，都需要依赖此模块

auth-security.zip 870 KB

（0）参考

- https://blog.csdn.net/weixin_42408447/article/details/118025142

（1）过滤器概述

- 前置知识：（web服务器 + servlet容器 = web容器）
  - http://c.biancheng.net/servlet2/container.html
- 过滤器的运行只依赖于servlet容器，不依赖于servlet程序。（意思就是不管有没有编写servlet程序，只要有servlet容器就可以），它的实现是基于函数回调（会回调过滤器对象的doFilter函数，函数参数是传入请求对象），它可以对所有的请求进行过滤，但是一个过滤器实例，只在servlet容器初始化的时候调用一次。

（2）使用过滤器的目的

- 对请求中的内容做一些过滤操作（过滤掉请求中的一些信息或者请求中提前设置一些参数），然后再将请求传入servlet（DispatcherServlet、业务Servlet）中进行业务操作。
- 案例：在过滤器中修改请求的字符编码、一些参数、权限校验（请求非登录接口时，过滤器判断是否携带token）、过滤低俗文字、危险字符等。

（3）拦截器概述

- 拦截器的运行只依赖于web框架（即b/s架构的s端、server端、也就是咱们的服务端程序、servlet程序、springmvc框架开发的程序，web程序），它的实现是基于java的反射机制（具体运用就是AOP的面向切面编程，又叫动态代理，在调用代理对象的方法时，内部会调用真实对象的业务方法，但是

笔记目录

在调用真实servlet对象的业务方法之前和之后都会走拦截器的方法。一个拦截器实例在一个servlet程序生命周期内（只要servlet对象不销毁，每次请求servlet，都会被拦截器拦截）是可以被多次调用，因为每来一次请求，就会先走拦截器，但是缺点只有是对servlet的请求才会走拦截器拦截，对一些静态资源的请求则没办法被拦截器拦截。

- 拦截器分类：客户端到服务端之间请求的拦截器，微服务与微服务之间请求的拦截器

（4）案例

（4-1）web.xml中配置编码过滤器类

复制代码

```java
//web.xml配置过编码过滤器类
<filter>
    <filter-name>encoding</filter-name>
    <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</fil
ter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>encoding</filter-name>
    <servlet-name>杠星</servlet-name>
</filter-mapping>
```

java

(4-2) springmvc.xml中配置拦截器类

复制代码

```java
//springmvc.xml中配置拦截器类
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/**" />
        <bean class="com.scorpios.atcrowdfunding.web.LoginInterceptor"></bean>
    </mvc:interceptor>
    <mvc:interceptor>
        <mvc:mapping path="/**" />
        <bean class="com.scorpios.atcrowdfunding.web.AuthInterceptor"></bean>
    </mvc:interceptor>
</mvc:interceptors>
```

java

(4-3) 两个过滤器类

复制代码

```java
//第一个过滤器类
public class TestFilter1 implements Filter {

        @Override
        protected void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
        //在DispatcherServlet之前执行
```

```java
        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###");
    }
}

//第二个过滤器
public class TestFilter2 implements Filter {

    @Override
    protected void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
        //在DispatcherServlet之前执行
        System.out.println("#TestFilter2 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("###TestFilter2 doFilter after###");
    }
}
```

java

（4-4）web.xml中注册这两个过滤器类

复制代码

```
    //在web.xml中注册这两个过滤器类
       <!—自定义过滤器：testFilter1—>
       <filter>
            <filter-name>testFilter1</filter-name>
```

```
            <filter-class>com.scorpios.filter.TestFilter1</filter-class>
        </filter>
        <filter-mapping>
            <filter-name>testFilter1</filter-name>
            <url-pattern>杠星</url-pattern>
        </filter-mapping>

        <!-- 自定义过滤器: testFilter2 -->
       <filter>
            <filter-name>testFilter2</filter-name>
            <filter-class>com.scorpios.filter.TestFilter2</filter-class>
        </filter>
        <filter-mapping>
            <filter-name>testFilter2</filter-name>
            <url-pattern>杠星</url-pattern>
        </filter-mapping>
```

java

(4-5) 两个拦截器类

复制代码

```
//第一个拦截器类
public class BaseInterceptor implements HandlerInterceptor{

    /**
     * 在DispatcherServlet之前执行
     * */
    public boolean preHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2) throws Exception {
```

```java
        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**");
    }
}


```

java

复制代码

```java
//第二个拦截器类
public class TestInterceptor implements HandlerInterceptor {
```

```java
    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**");
    }
}

```

java

(4-6) springmvc.xml中注册这两个拦截器类

复制代码

```
//springmvc.xml中注册这两个拦截器类
```

```
    <!-- 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="com.scorpios.interceptor.BaseInterceptor" />

        <mvc:interceptor>
            <!-- 对/xx/test进行拦截 -->
            <mvc:mapping path="/xx/test"/>
            <!-- 特定请求的拦截器只能有一个 -->
            <bean
class="com.scorpios.interceptor.TestInterceptor" />
        </mvc:interceptor>
    </mvc:interceptors>
```

java

（4-7）定义一个servlet程序

复制代码

```java
package com.scorpios.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

//定义一个servlet程序
@Controller
public class TestController {
    @RequestMapping("/test")
    public ModelAndView handleRequest(){
        System.out.println("----TestController executed----");
        return new ModelAndView("test");
    }
```

```
}

```

java

（4-8）请求执行结果

复制代码

```
http://localhost:/8080/demo
###TestFilter1 doFilterInternal executed###
###TestFilter2 doFilterInternal executed###
###TestFilter2 doFilter after###
###TestFilter1 doFilter after###

可以看到，请求一个不存在的servlet程序时，控制台打印了过滤器输出信息，就说明了过滤器的执行只依赖于servlet容器是否运行，跟servlet程序的是否运行并没有关系，并且多个过滤器的执行顺序跟web.xml文件中定义的先后顺序有关。
```

java

复制代码

```
http://localhost:/8080/test
###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**
```

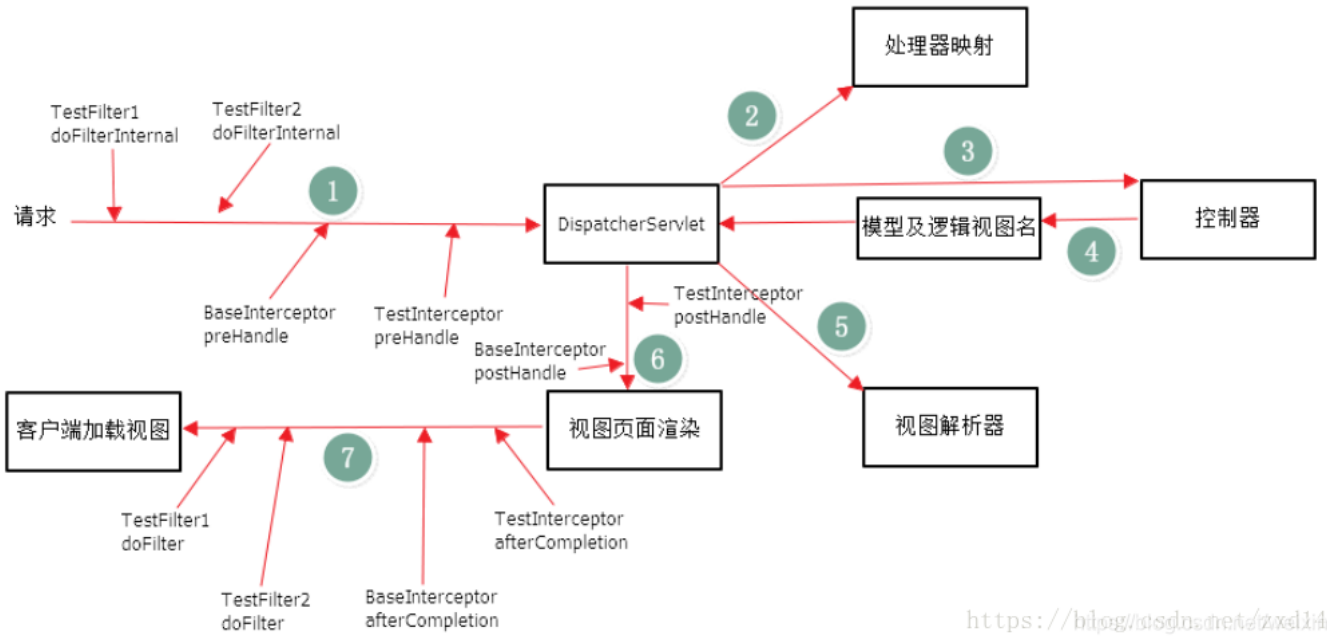
```
###TestFilter2 doFilter after###
###TestFilter1 doFilter after###

```

可以看到，请求存在的servlet程序时的控制台打印信息，说明了拦截器的执行依赖于servlet程序是否被调用，没有被调用，则就不会执行。很清晰的看到多个过滤器和多个拦截器的执行顺序了。当然多个拦截器的执行顺序跟springmvc.xml中定义的先后顺序有关。

java

（4-9）一次请求的整个执行流程



（5）本质区别

- 拦截器（Interceptor）是基于Java的反射机制，而过滤器（Filter）是基于函数回调。从灵活性上说拦截器功能更强大些，Filter能做的事情，都能做，而且可以在请求前，请求后执行，比较灵活。Filter

主要是针对URL地址做一个编码的事情、过滤掉没用的参数、安全校验（比较广泛，比如登录不登录之类），太细的话，还是建议用interceptor。不过还是根据不同情况选择合适的。

1、拦截器是基于java的反射机制的，而过滤器是基于函数回调
2、过滤器依赖于servlet容器，而拦截器不依赖于servlet容器
3、拦截器只能对action请求起作用，而过滤器则可以对几乎所有的请求起作用
4、拦截器可以访问action上下文、值栈里的对象，而过滤器不能
5、在action的生命周期中，拦截器可以多次被调用，而过滤器只在容器初始化时调用一次