# 过滤器

## 什么是过滤器

**过滤器是Servlet的高级特性之一，也别把它想得那么高深，只不过是实现Filter接口的Java类罢了！**

当浏览器发送请求给服务器的时候，先执行过滤器，然后才访问Web的资源。服务器响应Response，从Web资源抵达浏览器之前，也会途径过滤器。

**常见的过滤器用途主要包括**：**对用户请求进行统一认证、对用户的访问请求进行记录和审核、对用户发送的数据进行过滤或替换、转换图象格式、对响应内容进行压缩以减少传输量、对请求或响应进行加解密处理、触发资源访问事件、对XML的输出应用XSLT等**。

**Filter简单应用**

* **1、可以在filter中根据条件决定是否调用chain.doFilter(request, response)方法，即是否让目标资源执行**
* **2、在让目标资源执行之前，可以对request\response作预处理，再让目标资源执行**
* **3、在目标资源执行之后，可以捕获目标资源的执行结果，从而实现一些特殊的功能**

**链式结构：所有的过滤器都放在FilterChain里边，如果符合条件，就执行下一个过滤器（如果没有过滤器了，就执行目标资源）**。

**FilterConfig接口**

　　用户在**配置filter时**，可以使用<init-param>为filter配置一些初始化参数，当web容器实例化Filter对象，调用其init方法时，会把封装了filter初始化参数的filterConfig对象传递进来。因此开发人员在编写filter时，通过filterConfig对象的方法，就可获得：  
　　String getFilterName()：得到filter的名称。  
　　String getInitParameter(String name)： 返回在部署描述中指定名称的初始化参数的值。如果不存在返回null.  
　　Enumeration getInitParameterNames()：返回过滤器的所有初始化参数的名字的枚举集合。  
　　public ServletContext getServletContext()：返回Servlet上下文对象的引用。

**<dispatcher> 子元素**可以设置的值及其意义：（**配置filter-mapping**）

1. REQUEST：当用户直接访问页面时，Web容器将会调用过滤器。如果目标资源是通过RequestDispatcher的include()或forward()方法访问时，那么该过滤器就不会被调用。
2. INCLUDE：如果目标资源是通过RequestDispatcher的include()方法访问时，那么该过滤器将被调用。除此之外，该过滤器不会被调用。
3. FORWARD：如果目标资源是通过RequestDispatcher的forward()方法访问时，那么该过滤器将被调用，除此之外，该过滤器不会被调用。
4. ERROR：如果目标资源是通过声明式异常处理机制调用时，那么该过滤器将被调用。除此之外，过滤器不会被调用。

**注解：**@WebFilter(filterName = "FilterDemo1",urlPatterns = "/\*")

## 为什么需要用到过滤器

### 解决中文乱码问题

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException, IOException {  
  
    //**将request和response强转成http协议的**  
    HttpServletRequest httpServletRequest = (HttpServletRequest) req;  
    HttpServletResponse httpServletResponse = (HttpServletResponse) resp;  
  
    httpServletRequest.setCharacterEncoding("UTF-8");  
    httpServletResponse.setCharacterEncoding("UTF-8");  
    httpServletResponse.setContentType("text/html;charset=UTF-8");  
  
    chain.doFilter(httpServletRequest, httpServletResponse);  
}

**分析：**

上面的过滤器是不完善的，因为**浏览器用get方式提交给服务器的中文数据，单单靠上面的过滤器是无法完成的！**

那么我们需要怎么做呢？？我们之前解决get方式的乱码问题是这样的：**使用request获取传递过来的数据，经过ISO 8859-1反编码获取得到不是乱码的数据（传到Servlet上的数据已经被ISO 8859-1编码过了，反编码就可以获取原来的数据），再用UTF-8编码，得到中文数据！**

**在Servlet获取浏览器以GET方式提交过来的中文是乱码的根本原因是：getParameter()方法是以ISO 8859-1的编码来获取浏览器传递过来的数据的，得到的是乱码**

既然知道了根本原因，那也好办了：**过滤器传递的request对象，使用getParameter()方法的时候，获取得到的是正常的中文数据**

**增强request对象，我们要使用包装设计模式！**

包装设计模式的五个步骤：

* \*\*1、实现与被增强对象相同的接口 \*\*
* **2、定义一个变量记住被增强对象**
* **3、定义一个构造器，接收被增强对象**
* **4、覆盖需要增强的方法**
* **5、对于不想增强的方法，直接调用被增强对象（目标对象）的方法**

split 的实现直接调用的 matcher 类的 split 的方法。“ **.** ”在正则表达式中有特殊的含义，因此我们使用的时候必须进行转义[\\.](file:///\\.)

//tomcat容器默认采用了iso-8859-1的编码方法

//通过本为UTF-8编码却被tomcat用iso-8859-1解码的字进行恢复，

//其将解码后的字通过iso-8859-1反解码成二进制数组，再将该字节数组用UTF-8解码。

//最终被new String成字符串。

value = new String(value.getBytes("iso8859-1"),"UTF-8");

### 敏感词的过滤器

**增强request对象**

class MyDirtyRequest extends HttpServletRequestWrapper {  
    HttpServletRequest request;  
    //定义一堆敏感词汇  
    private List<String> list = Arrays.asList("傻b", "尼玛", "操蛋");  
    public MyDirtyRequest(HttpServletRequest request) {  
        super(request);  
        this.request = request;  
    }  
    @Override  
    public String getParameter(String name) {  
        String value = this.request.getParameter(name);  
        if (value == null) {  
            return null;  
        }  
        //遍历list集合，看看获取得到的数据有没有敏感词汇  
        for (String s : list) {  
            if (s.equals(value)) {  
                value = "\*\*\*\*\*";  
            }  
        }  
        return value ;  
    }  
}

### [截器和过滤器的区别](https://www.iteye.com/blog/2442198)

【Spring的拦截器】于【Servlet的Filter】有相似之处，比如二者都是AOP编程思想的体现，都能实现权限检查、日志记录等。

不同的是：

·使用范围不同：【Filter是Servlet规范固定的，只能用于Web程序中】，而【拦截器既可以用户Web程序，也可以用于Application、Swing程序中】。

·规范不同：Filter是在Servlet规范中定义的，是【Servlet容器支持】的，而拦截器是在Spring容器内的，是【Spring框架支持】的。

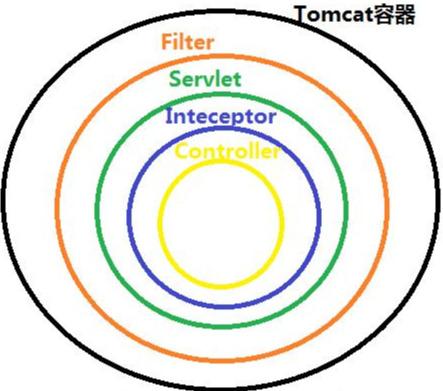
·使用的资源不同：同其他的代码块一样，拦截器也是一个Spring的组件，归Spring管理，配置在Spring文件中，因此能使用Spring里得任何资源、对象，例如Service对象、数据源、事务管理等，通过Ioc注入道拦截器即可。而Filter则不能。

·深度不同：【Filter在只在Servlet启动前后起作用。而拦截器能够深入到方法前后、异常抛出前后等】，因此拦截器的使用弹性更大一些，所以在使用Spring架构的程序中，要优先使用拦截器。

1、过滤器和拦截器触发时机不一样，

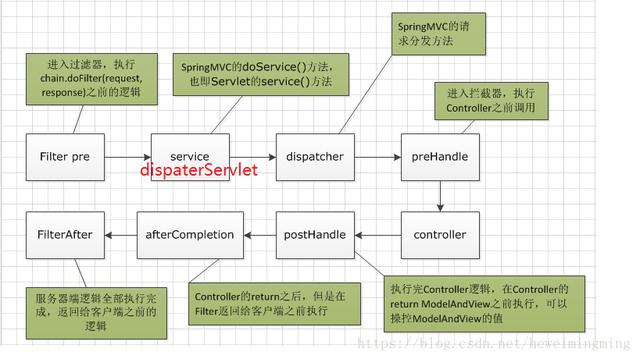
过滤器是在【请求进入容器后，但请求进入servlet之前】进行预处理的。请求结束返回也是，是【在servlet处理完后，返回给前端之前】。

如图：



2、拦截器可以获取IOC容器中的各个bean，而过滤器就不行，因为拦截器是spring提供并管理的，spring的功能可以被拦截器使用，在拦截器里注入一个service，可以调用业务逻辑。而过滤器是JavaEE标准，只需依赖servlet api ，不需要依赖spring。

过滤器拦截器运行先后步骤：



其中第2步，SpringMVC的机制是由DispaterServlet来分发请求给不同的Controller，其实这一步是在Servlet的service()方法中执行的.

3、**过滤器的实现基于回调函数。**而【拦截器（代理模式）的实现基于反射】，代理分静态代理和动态代理，动态代理是拦截器的简单实现。

何时使用拦截器？何时使用过滤器？

如果是非spring项目，那么拦截器不能用，只能使用过滤器。

如果是处理controller前后，既可以使用拦截器也可以使用过滤器。

如果是处理dispaterServlet前后，只能使用过滤器。

**过滤器和拦截器的区别： 　　  
　　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拦截器不依赖与servlet容器，过滤器依赖与servlet容器。  
　　③拦截器只能对部分请求起作用，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spring容器上下文、值栈里的对象，而过滤器不能访问。  
　　⑤在controller的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次。  
　　⑥拦截器可以获取IOC容器中的各个bean，而过滤器就不行，这点很重要，在拦截器里注入一个service，可以调用业务逻辑。**

# 监听器

## 什么是监听器

监听器就是一个**实现特定接口的普通java程序**，这个程序专门用于**监听另一个java对象的方法调用或属性改变**，当被监听对象发生上述事件后，监听器某个方法将立即被执行。

监听器涉及三个组件：**事件源，事件对象，事件监听器**

**当事件源发生某个动作的时候，它会调用事件监听器的方法，并在调用事件监听器方法的时候把事件对象传递进去。**

**我们在监听器中就可以通过事件对象获取得到事件源，从而对事件源进行操作！**

## Servle监听器

和其它事件监听器略有不同的是，**servlet监听器的注册不是直接注册在事件源上，而是由WEB容器负责注册**，开发人员**只需在web.xml文件中使用<listener>标签配置好监听器**。

**监听对象的创建和销毁**

**HttpSessionListener、ServletContextListener、ServletRequestListener分别监控着Session、Context、Request对象的创建和销毁**

* **HttpSessionListener(可以用来收集在线者信息)**
* **ServletContextListener(可以获取web.xml里面的参数配置)**
* **ServletRequestListener**

**监听对象属性变化**

**ServletContextAttributeListener、HttpSessionAttributeListener、ServletRequestAttributeListener分别监听着Context、Session、Request对象属性的变化**

这三个接口中都定义了**三个方法来处理被监听对象中的属性的增加，删除和替换的事件**，同一个事件在这三个接口中对应的方法**名称完全相同，只是接受的参数类型不同**。

* **attributeAdded()**
* **attributeRemoved()**
* **attributeReplaced()**

**监听Session内的对象**

**除了上面的6种Listener，还有两种Linstener监听Session内的对象，分别是HttpSessionBindingListener和HttpSessionActivationListener，实现这两个接口并不需要在web.xml文件中注册**

* 实现HttpSessionBindingListener接口，\*\*JavaBean 对象可以感知自己被绑定到 Session 中和从 Session 中删除的事件【和HttpSessionAttributeListener的作用是差不多的】 \*\*
* 实现HttpSessionActivationListener接口，**JavaBean 对象可以感知自己被活化和钝化的事件（当服务器关闭时，会将Session的内容保存在硬盘上【钝化】，当服务器开启时，会将Session的内容在硬盘式重新加载【活化】）** 。。

**想要测试出Session的硬化和钝化，需要修改Tomcat的配置的**。在META-INF下配置的context.xml文件中添加下面的代码：

<Context>  
  <Manager className="org.apache.catalina.session.PersistentManager" maxIdleSwap="1">  
  <Store className="org.apache.catalina.session.FileStore" directory="zhongfucheng"/>  
  </Manager>  
</Context>