**Filter**

## 一、Filter的介绍

1 Filter技术是servlet 2.3新增加的功能。它能够对Servlet容器的请求和响应对象进行检查和修改。

2 Filter本身并不生成请求和响应对象，只是提供过滤功能。

3 Filter能够在Servlet被调用之前检查Request对象，并修改Request Header和Request内容；在Servlet被调用之后检查Response对象，修改Response Header和Response的内容。

4 Filter可以过滤的Web组件包括Servlet，JSP和HTML等文件。

Servlet中的过滤器Filter是实现了javax.servlet.Filter接口的服务器端程序，**主要的用途是过滤字符编码、做一些业务逻辑判断等。**其工作原理是，只要你在web.xml文件配置好要拦截的客户端请求，它都会帮你拦截到请求，此时你就可以对请求或响应(Request、Response)统一设置编码，简化操作；同时还可以进行逻辑判断，如用户是否已经登录、有没有权限访问该页面等等工作，它是随你的web应用启动而启动的，只初始化一次，以后就可以拦截相关的请求，只有当你的web应用停止或重新部署的时候才能销毁。

## 二、Filter的工作原理

1 当客户端发出Web资源的请求时，Web服务器根据应用程序配置文件设置的过滤规则进行检查，若客户请求满足过滤规则，则对客户请求／响应进行拦截，对请求头和请求数据进行检查或改动，并依次通过过滤器链，最后把请求／响应交给请求的Web资源处理。请求信息在过滤器链中可以被修改，也可以根据条件让请求不发往资源处理器，并直接向客户机发回一个响应。当资源处理器完成了对资源的处理后，响应信息将逐级逆向返回。同样在这个过程中，用户可以修改响应信息，从而完成一定的任务。

2 两个过滤器同时过滤一个请求时，就要用到过滤链FilterChain。Filter的FilterChain中，服务器会按照web.xml中过滤器定义的先后循序组装成一条链，然后一次执行其中的doFilter()方法。执行的顺序就如下图所示，执行第一个过滤器的chain.doFilter()之前的代码，第二个过滤器的chain.doFilter()之前的代码，请求的资源，第二个过滤器的chain.doFilter()之后的代码，第一个过滤器的chain.doFilter()之后的代码，最后返回响应。

3 Filter的执行流程就是：执行第一个过滤器的chain.doFilter()之前的代码——>第二个过滤器的chain.doFilter()之前的代码——>……——>第n个过滤器的chain.doFilter()之前的代码——>所请求servlet的service()方法中的代码——>所请求servlet的doGet()或doPost()方法中的代码——>第n个过滤器的chain.doFilter()之后的代码——>……——>第二个过滤器的chain.doFilter()之后的代码——>第一个过滤器的chain.doFilter()之后的代码。

## 三、Filter生命周期的四个阶段

**（1）实例化：**Web容器在部署Web应用程序时对所有过滤器进行实例化。Web容器回调它的无参构造方法。

**（2）初始化：**实例化完成之后，马上进行初始化工作。Web容器回调init()方法。

**（3）过滤：**请求路径匹配过滤器的URL映射时。Web容器回调doFilter()方法——主要的工作方法。

**（4）销毁：** Web容器在卸载Web应用程序前，Web容器回调destroy()方法。

## 四、Filter的API

所有的过滤器都必须实现Filter接口。该接口定义了init,doFilter0，destory()三个方法：

**1 public void init (FilterConfig filterConfig)**

当开始使用servlet过滤器服务时，Web容器调用此方法一次，为服务准备过滤器；然后在需要使用过滤器的时候调用doFilter()，传送给此方法的FilterConfig对象，包含servlet过滤器的初始化参数。

如获取Filter中配置的初始化参数：

<filter>

<filter-name>LoginFilter</filter-name>

<filter-class>com.itzhai.login.LoginFilter</filter-class>

<init-param>

<param-name>username</param-name>

<param-value>HelloWorld</param-value>

</init-param>

</filter>

在init方法中获取：

@Override

public void init(FilterConfig filterConfig) throws ServletException

{

//获取Filter初始化参数

String username = filterConfig.getInitParameter("username");

}

**2 public void doFilter(ServletRequest request，ServletResponse response，FilterChain chain)**

每个过滤器都接受当前的请求和响应，且FilterChain过滤器链中的过滤器（应该都是符合条件的）都会被执行。doFilter方 法中，过滤器可以对请求和响应做它想做的一切，通过调用他们的方法收集数据，或者给对象添加新的行为。过滤器通过传送至 此方法的FilterChain参数，调用chain．doFilterO将控制权传送给下一个过滤器。当这个调用返回后，过滤器可以在它的 Filter方法的最后对响应做些其他的工作。如果过滤器想要终止请求的处理或得到对响应的完全控制，则可以不调用下一个过滤 器，而将其重定向至其它一些页面。当链中的最后一个过滤器调用chain．doFilterO方法时，将运行最初请求的Servlet。

**3 public void destroy()**

一旦doFilterO方法里的所有线程退出或已超时，容器调用此方法。**服务器调用destoryO以指出过滤器已结束服务**，用于释放过滤器占用的资源。

## 五、例子--过滤敏感词汇

package com.zhouyu.filter;

import java.io.IOException;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class FilterTest implements Filter

{

@Override

public void destroy()

{

// TODO Auto-generated method stub

}

@Override

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException

{

// 转换成实例的请求和响应对象

HttpServletRequest req = (HttpServletRequest) request;

HttpServletResponse resp = (HttpServletResponse) response;

// 获取评论并屏蔽关键字

String str = req.getParameter("str");

str = str.replace("你妹呀", "\*\*\*");

// 重新设置参数

req.setAttribute("str", str);

// 继续执行

chain.doFilter(request, response);

}

@Override

public void init(FilterConfig filterConfig) throws ServletException

{

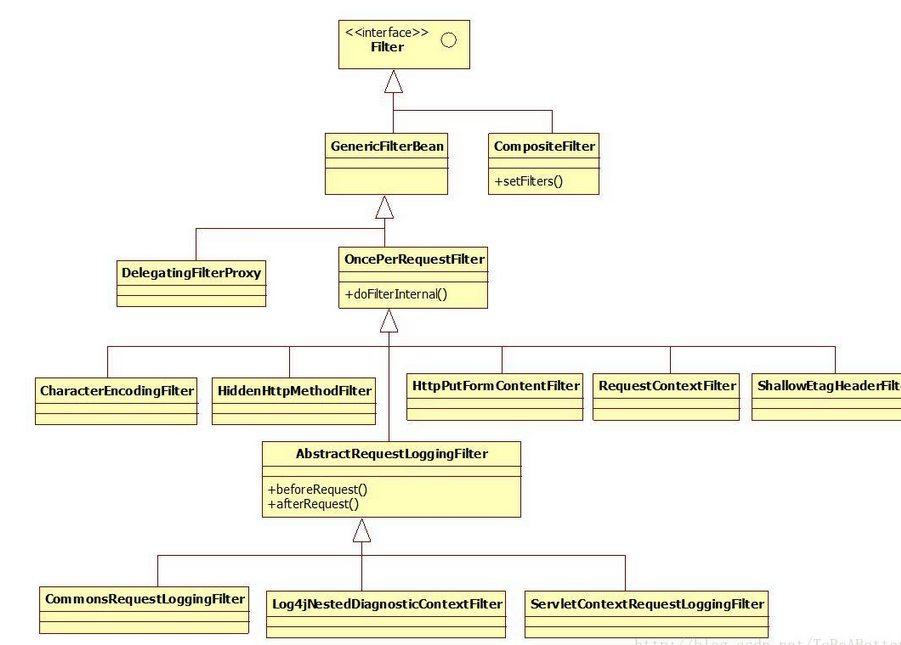
// TODO Auto-generated method stub

}

}

## Spring Filter

**过滤器是servlet的规范规定的，只能用于过滤请求，而interceptor是Spring里面基于切面编程的一种实现。**



**1 通常实现拦截器的方式**

实现javax.servlet.Filter接口，Filter是servlet包的定义的接口，现有的这种实现方法有CompositeFilter

继承抽象类GenericFilterBean 这个类实现了Filter接口。现有的实现有DelegatingFilterProxy

继承抽象类OncePerRequestFilter 这个类是GenericFilterBean 的子类。现有的实现类有：CharacterEncodingFilter 、HiddenHttpMethodFilter 、HttpPutFormContentFilter、RequestContextFilter和ShallowEtagHeaderFilter

继承抽象类AbstractRequestLoggingFilter，该类为OncePerRequestFilter的直接子类，这一类过滤器包括CommonsRequestLoggingFilter、Log4jNestedDiagnosticContextFilter和ServletContextRequestLoggingFilter。

**2 实现的接口有**

Filter：过滤器的实现

BeanNameAware：setBeanName方法，便于Bean管理器生成Bean

EnvironmentAware：用于指明Bean的运行环境

EnvironmentCapable：用于获取bean的运行环境

ServletContextAware：实现该接口的setServletContextAware方法，指明ServletContext

InitializingBean：实现该接口的afterPropertiesSet方法，在Bean初始化时设置属性

DisposableBean ：实现该接口的destroy方法，用于回收资源。

**Listener**

1. **什么是监听器**

监听器就是对整个web环境的,**当web环境发生改变时,立即调用想应的方法进行处理,Servlet的监听器Listener**，它是实现了javax.servlet.ServletContextListener接口的服务器端程序，它也是随web应用的启动而启动，只初始化一次，随web应用的停止而销毁。主要作用是：做一些初始化的内容添加工作、设置一些基本的内容、比如一些参数或者是一些固定的对象等等。

1. **应用:**

1)Java项目开发;

2)手机相关软件开发

**3.监听器相关名词解释:**

1).事件源: 就是指被监听的对象

2).监听器对象: 用来监听"事件源"对象的

3).册(绑定):将"监听器"注册给"事件源".当"事件源"发生某些行为时,监听对象将被执行.

4).事件:事件源行为的称呼

5).事件对象: 在"监听器"对象

**4.编写流程:**

1.编写监听器实现类,需要指定接口

2.在web.xml文件中配置监听器(有一些默认的监听器不需要配置)

<listener>

<listener-class>监听器类</listener-class>

</listener>

**5.监听器的分类**

**1)监听ServletRequest,HttpSession,ServletContext 三个域对象的创建和销毁**

1.1 javax.servlet.ServletRequestListener：监听Request对象创建或销毁

1.2 javax.servlet.http.HttpSessionListener ：监听Session对象创建或销毁

1.3 javax.servlet.ServletContextListener：监听ServletContext对象创建或销毁

**2)监听ServletRequest, HttpSession, ServletContext三个域对象属性变更**

2.1 javax.servlet.ServletRequestAttributeListener：监听request对象属性添加替换删除

2.2 javax.servlet.http.HttpSessionAttributeListener：监听session对象属性添加替换删除

2.3 javax.servlet.ServletContextAttributeListener：监听servletcontext对象属性添替换删除

**3)监听session作用域特殊Javabean，这两个监听器不需要再web.xml中配置**

3.1javax.servlet.http.HttpSessionBindingListener：监听实现指定接口javabean与sesseion作用域的绑定和解绑，即从session作用域存放或异常监听。javabean必须实现该接口

3.2javax.servlet.http.HttpSessionActivationListener：监听特殊JavaBean在session作用域钝化或活化， javabean必须实现该接口

**6.简单使用:**

（1）: 编写ServletContextListener接口实现类,用于监听ServletContext对象的创建和销毁

public class PathServletContextListener implement ServletContextListener{

//初始化方法

public void contextInitialized(ServletContextEvent sce){

String contextLocation = sce.getServletContext().getInitParameter("contextLocation");

System.out.println("contextLocation" + contextLocation);

}

//销毁方法

public void contextDestoryed(ServletContextEvent sce){

}

}

**（2）:在web.xml上注册监听器**

<context-param>

<param-name>contextlocaltion</param-name>

<param-value>XX.xml</param-value>

</context-param>

**（3）:在web.xml上注册监听器**

<!--配置监听器 -->

<listener>

<listener-class>

你所配置的监听器的实现类的全路径

</listener-class>

</listener>

**7 监听器实现方法**

**（1）当范围对象的状态发生变化的时候，服务器自动调用监听器对象中的方法。**

例如：创建一个“人”类Class Person人拥有吃的方法public voideat(){},我们的目的就是，在这个人吃之前要提醒他洗手，所以我们就要监听这个人，不管他在什么时候吃，我们都要能够监听到并且提醒他洗手，通俗点讲这个就是监听器的作用(在吃之前提醒他洗手)。

**（2）Java的时间监听机制涉及到三个组件：事件源、事件监听器、事件对象。**

当事件源上发生操作时，它会调用事件监听器的一个方法，并且调用这个方法时，会传递事件对象过来事件监听器是由开发人员编写，开发人员在事件监听器中，通过事件对象可以拿到事件源，从而对事件源上的操作进行处理。

**（3）实现**

　　1）创建一个人类，人具有跑、吃的方法　　（创建一个事件源）

　　2）创建一个接口用来监视人的跑、吃　　　（事件监听器）

　　3）创建一个事件对象，里面用来封装人(Person)这个对象　　（事件对象）

　　4）创建一个测试类，用来测试Java监听机制

package com.hiya.ee.listener;

public class Event

{

private Person person;

public Event(Person person)

{

super();

this.person = person;

}

public Event()

{

super();

}

public Person getPerson()

{

return person;

}

public void setPerson(Person person)

{

this.person = person;

}

}

package com.hiya.ee.listener;

public interface IPersonListener

{

public void doRun(Event even);

public void doEat(Event even);

}

package com.hiya.ee.listener;

public class MyPersonListener implements IPersonListener

{

@Override

public void doRun(Event event)

{

Person person = event.getPerson();

System.out.println("人在跑之前执行的动作" + person);

}

@Override

public void doEat(Event even)

{

System.out.println("人在吃之前执行的动作");

}

}

package com.hiya.ee.listener;

public class Person

{

private IPersonListener listener;

public void registerListener(IPersonListener personListener)

{

this.listener = personListener;

}

public void run()

{

if (listener != null)

{

Event even = new Event(this);

this.listener.doRun(even);

}

System.out.println("人具有跑的方法");

}

public void eat()

{

if (listener != null)

{

Event even = new Event(this);

this.listener.doEat(even);

}

System.out.println("人具有吃的方法");

}

}

package com.hiya.ee.listener;

public class ListenerClient

{

public static void main(String[] args)

{

Person person=new Person();

person.registerListener(new MyPersonListener());

person.run();

person.eat();

}

}

**Interceptors**

**1、拦截器是在面向切面编程中应用的**，就是在你的service或者一个方法前调用一个方法，或者在方法后调用一个方法比如动态代理就是拦截器的简单实现，在你调用方法前打印出字符串（或者做其它业务逻辑的操作），也可以在你调用方法后打印出字符串，甚至在你抛出异常的时候做业务逻辑的操作。**拦截器不是在web.xml配置的，比如struts在struts.xml配置，在springMVC在spring与springMVC整合的配置文件中配置。**

**2、在springmvc中，定义拦截器要实现HandlerInterceptor接口，并实现该接口中提供的三个方法**

package com.cn.util;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.HandlerInterceptor;

import org.springframework.web.servlet.ModelAndView;

public class InterceptorUtil implements HandlerInterceptor{

//进入Handler方法之前执行

//可以用于身份认证、身份授权。如果认证没有通过表示用户没有登陆，需要此方法拦截不再往下执行，否则就放行

@Override

public boolean preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) throws Exception {

// TODO Auto-generated method stub

System.out.println("InterceptorUtil...........preHandle");

String user= (String) request.getSession().getAttribute("user");

if(user != null){

return true;

}

request.getRequestDispatcher("/WEB-INF/jsp/index.jsp").forward(request, response);

//true表示放行，false表示不放行

return false;

}

@Override

public void postHandle(HttpServletRequest request,

HttpServletResponse response, Object handler,

ModelAndView modelAndView) throws Exception {

// TODO Auto-generated method stub

System.out.println("InterceptorUtil...........postHandle");

}

@Override

public void afterCompletion(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex)

throws Exception {

// TODO Auto-generated method stub

System.out.println("InterceptorUtil...........afterCompletion");

}

}

**3 API概述**

**（1） preHandle方法：**进入Handler方法之前执行。可以用于身份认证、身份授权。比如如果认证没有通过表示用户没有登陆，需要此方法拦截不再往下执行（return false），否则就放行（return true）。

1. **postHandle方法：**进入Handler方法之后，返回ModelAndView之前执行。可以看到该方法中有个modelAndView的形参。应用场景：从modelAndView出发：将公用的模型数据（比如菜单导航之类的）在这里传到视图，也可以在这里同一指定视图。

**（3）afterCompletion方法：**执行Handler完成之后执行。应用场景：统一异常处理，统一日志处理等。在springmvc中，拦截器是针对具体的HandlerMapping进行配置的，也就是说如果在某个HandlerMapping中配置拦截，经过该 HandlerMapping映射成功的handler最终使用该拦截器。

**4 spring-mvc.xml配置文件中：**

<!-- 拦截器配置 -->

<mvc:interceptors>

<!--多个拦截器,顺序执行 -->

<!-- 登陆认证拦截器 -->

<mvc:interceptor>

<!-- /\*\* 表示拦截所有url包括子url路径，/\*只拦截根下的url -->

<mvc:mapping path="/\*\*"/>

<bean class="com.cn.util.InterceptorUtil"></bean>

</mvc:interceptor>

<!-- 其他拦截器 -->

</mvc:interceptors>

**6 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、过滤器依赖与servlet容器，而拦截器不依赖与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，而过滤器只能在容器初始化时被调用一次

**7 执行顺序：**

过滤前 - 拦截前 - Action处理 - 拦截后 -过滤后。个人认为过滤是一个横向的过程，首先把客户端提交的内容进行过滤(例如未登录用户不能访问内部页面的处理)；过滤通过后，拦截器将检查用户提交数据的验证，做一些前期的数据处理，接着把处理后的数据发给对应的Action；Action处理完成返回后，拦截器还可以做其他过程，再向上返回到过滤器的后续操作。

**8 三者不同点**

**（1）过滤器（Filter）：**当你有一堆东西的时候，你只希望选择符合你要求的某一些东西。定义这些要求的工具，就是过滤器。filter主要用途是过滤字符编码、做一些业务逻辑判断等。工作原理是，只要在web.xml文件配置好拦截的客户端请求，它都会帮你拦截到请求，此时你就可以对请求或响应统一设置编码，简化操作；同时还进行逻辑判断，如用户是否已经登陆、有没有权限访问该页面等工作。随web应用启动而启动，只初始化一次。

**（2）拦截器（Interceptor）**：在一个流程正在进行的时候，你希望干预它的进展，甚至终止它进行，这是拦截器做的事情。拦截器是在面向切面编程中应用的，就是在你的service或者一个方法前调用一个方法，或者在方法后调用一个方法。基于java的反射机制

**（3）监听器（Listener）：**当一个事件发生的时候，你希望在这个时候做一些额外的事情，而并不想干预这个事件本身的进程，这就要用到监听器。listener主要作用是：做一些初始化的内容添加工作、设置一些基本的内容、比如一些参数或者一些固定的对象。

