

# 监听器与过滤器

本章内容：共2小节，10个知识点

- 第1节：监听器
- 第2节：过滤器



# 本章目标



- 理解监听器和过滤器的作用
- 能够编写监听器和过滤器并进行配置使用；

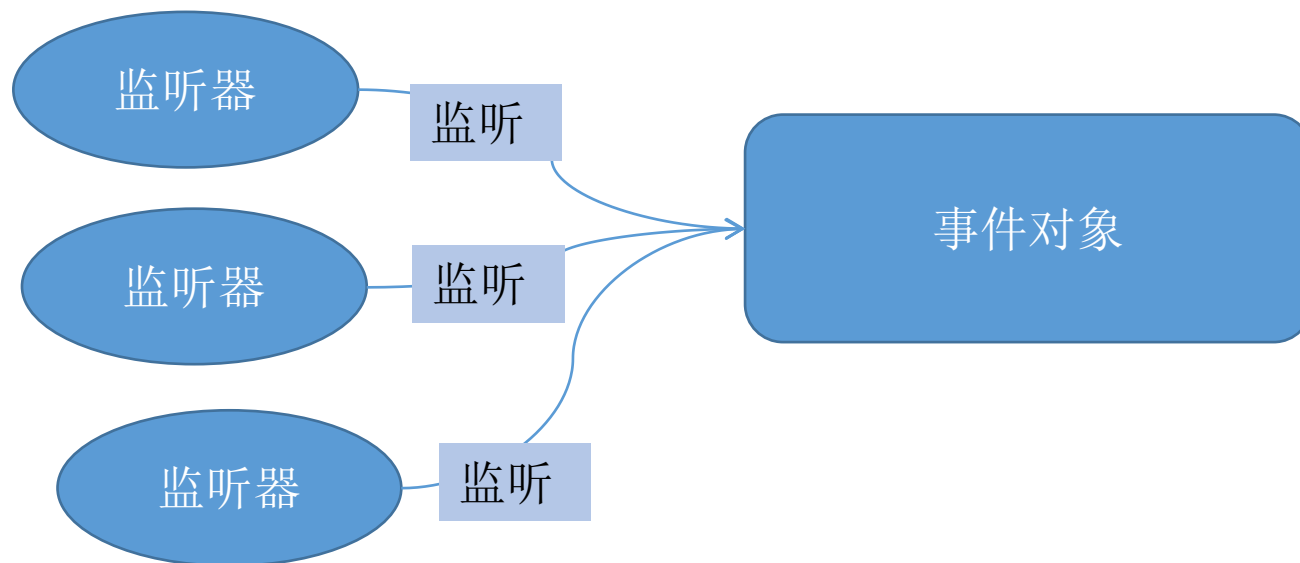
# 第1节 【监听器】



- 知识点1：监听器的作用
- 知识点2：监听器相关API
- 知识点3：监听器的开发与配置
- 知识点4：上下文相关监听器
- 知识点5：会话相关监听器

# 知识点1: 监听器的作用

- 事件发生的时间往往是不确定的，当事件发生的时候需要进行一些处理时，就可以使用监听器；
- 例如，上下文对象被创建或者被销毁就是一个事件，但是何时被创建或销毁是不确定的，如果需要只要上下文对象被创建或销毁就进行相应处理，就可以使用监听器；



## 知识点2：监听器相关的API-1



- 监听器相关的API包括**事件类**以及**监听器接口**；
- 事件类定义了事件类型，监听器接口定义了监听事件的方法；
- Servlet API中定义了6种事件类型
  - 上下文相关的事件
    - ServletContextEvent：该类表示上下文事件，当应用上下文对象发生改变，如创建或销毁上下文对象时，将触发上下文事件。
    - ServletContextAttributeEvent：该类表示上下文属性事件，当应用上下文的属性改变，如增加、删除、覆盖上下文中的属性时，将触发上下文属性事件。
  - 请求相关的事件
    - ServletRequestEvent：该类表示请求事件，当请求对象发生改变，如创建或销毁请求对象时，触发请求事件。
    - ServletRequestAttributeEvent：该类表示请求属性事件，当请求中的属性改变，如增加、删除、覆盖请求中的属性时，触发请求属性事件。
  - 会话相关的事件
    - HttpSessionEvent：该类表示会话事件，当会话对象发生改变，如创建或销毁会话对象，活化或钝化会话对象时，将触发会话事件。
    - HttpSessionBindingEvent：该类表示会话绑定事件，当会话中的属性发生变化时，如增加、删除、覆盖会话中的属性时，将触发会话绑定事件。

## 知识点2：监听器相关的API-2



- Servlet API中定义了8种监听器接口，用来监听不同的事件类型
- 上下文相关的监听器
  - ServletContextListener：上下文监听器，监听ServletContextEvent事件。
  - ServletContextAttributeListener：上下文属性监听器，用来监听ServletContextAttribute事件。
- 请求相关的监听器
  - ServletRequestListener：请求监听器，监听ServletRequestEvent事件。
  - ServletRequestAttributeListener：请求属性监听器，用来监听ServletRequestAttributeEvent事件。
- 会话相关的监听器
  - HttpSessionListener：会话监听器，监听HttpSessionEvent。
  - HttpSessionActivationListener：会话活化监听器，监听HttpSessionEvent事件。
  - HttpSessionAttributeListener：会话属性监听器，监听HttpSessionAttributeEvent事件。
  - HttpSessionBindingListener：会话绑定监听器，监听HttpSessionAttributeEvent事件。

## 知识点3：监听器的开发及配置



- 编写监听器非常简单，只需要：
  - 写一个类实现相应的XXXListener接口；
  - 重写接口中的方法，实现监听的功能；
- 要想监听器生效，需要在web.xml中进行配置，例如：

```
<listener>  
<listener-class>com.chinasofti.chapter06.section01.ListenerTest</listener-class>  
</listener>
```

接下来做具体的实例。



## 知识点4：上下文相关监听器-1



- 通过上面学习，发现和上下文有关的监听器有两个
  - 一个是上下文对象相关的【ServletContextListener】；
  - 一个是上下文属性相关的【ServletContextAttributeListener】；

## 知识点4：上下文相关监听器-2



- 要监听器生效，需要在web.xml中进行配置；

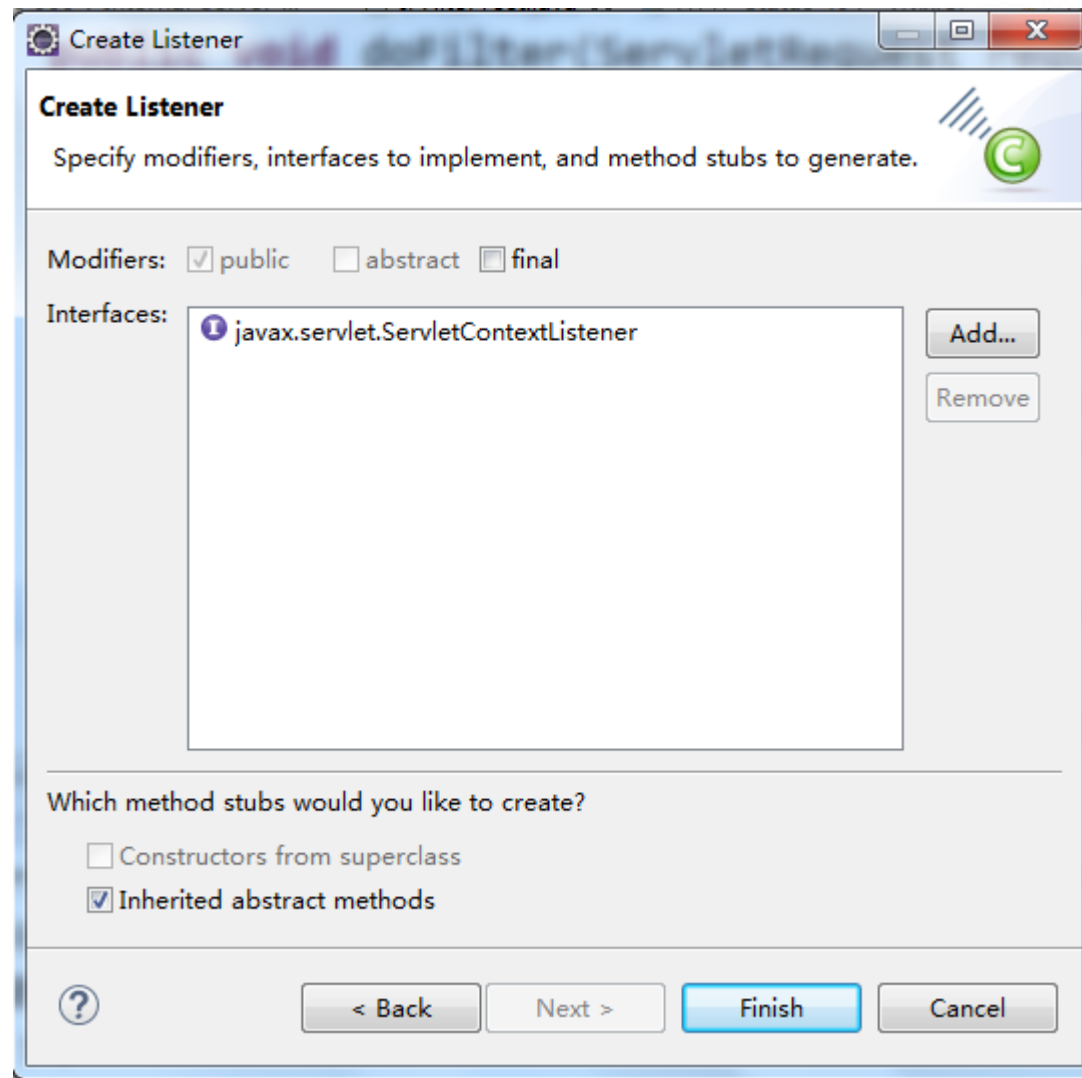
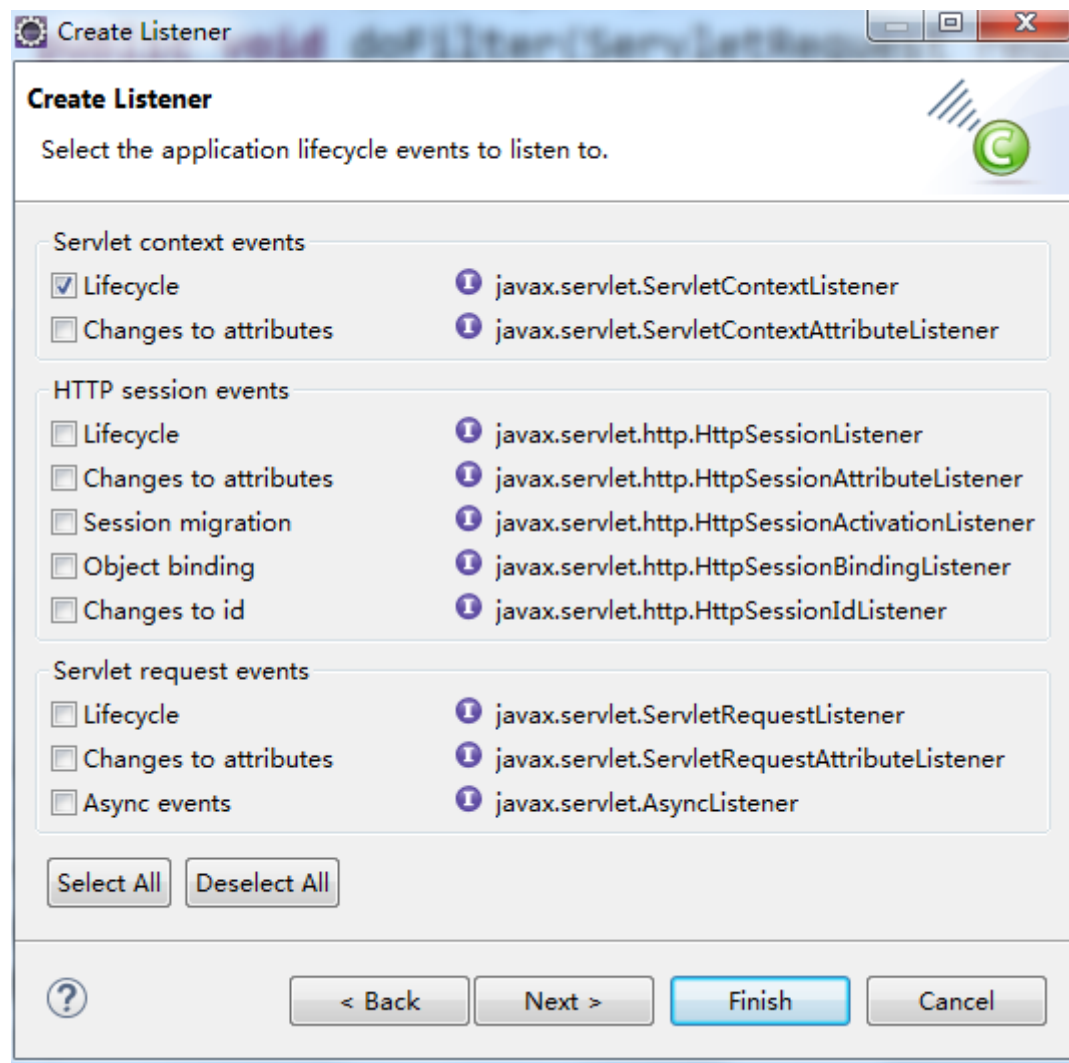
```
<listener>  
<listener-class>com.chinasofti.chapter06.section01.ListenerTest</listener-class>  
</listener>
```

- 如果还需要对上下文进行其他监听，创建不同的监听器进行配置使用即可；

## 知识点5：会话相关监听器

- 通过前面学习，可见与会话相关的监听器有4个
  - HttpSessionListener：会话监听器，当会话对象被创建后或销毁前需要一些自定义处理时，可以用此监听器监听；
  - HttpSessionActivationListener：会话活化监听器，会话对象存在于服务器端，只要没有失效，服务器就得分配空间给其使用；为了能够提高使用效率，服务器有内在的活化钝化机制，可以将暂时不使用的会话对象钝化到外存，需要使用时再活化到内存。当活化后或钝化前需要一些自定义处理时，可以使用该监听器；
  - HttpSessionAttributeListener：会话属性监听器，当会话中的属性被添加、删除、替换时，要进行一些自定义处理时，可以使用该监听器，使用时可以用事件对象获取属性的名字等信息。
  - HttpSessionBindingListener：会话绑定监听器，当类实现了HttpSessionBindingListener接口后，该类对象绑定或解除绑定到会话时，就会被该监听器监听。绑定指的是调用setAttribute方法，解除绑定指的是调用removeAttribute方法，或者会话超时、会话失效等。

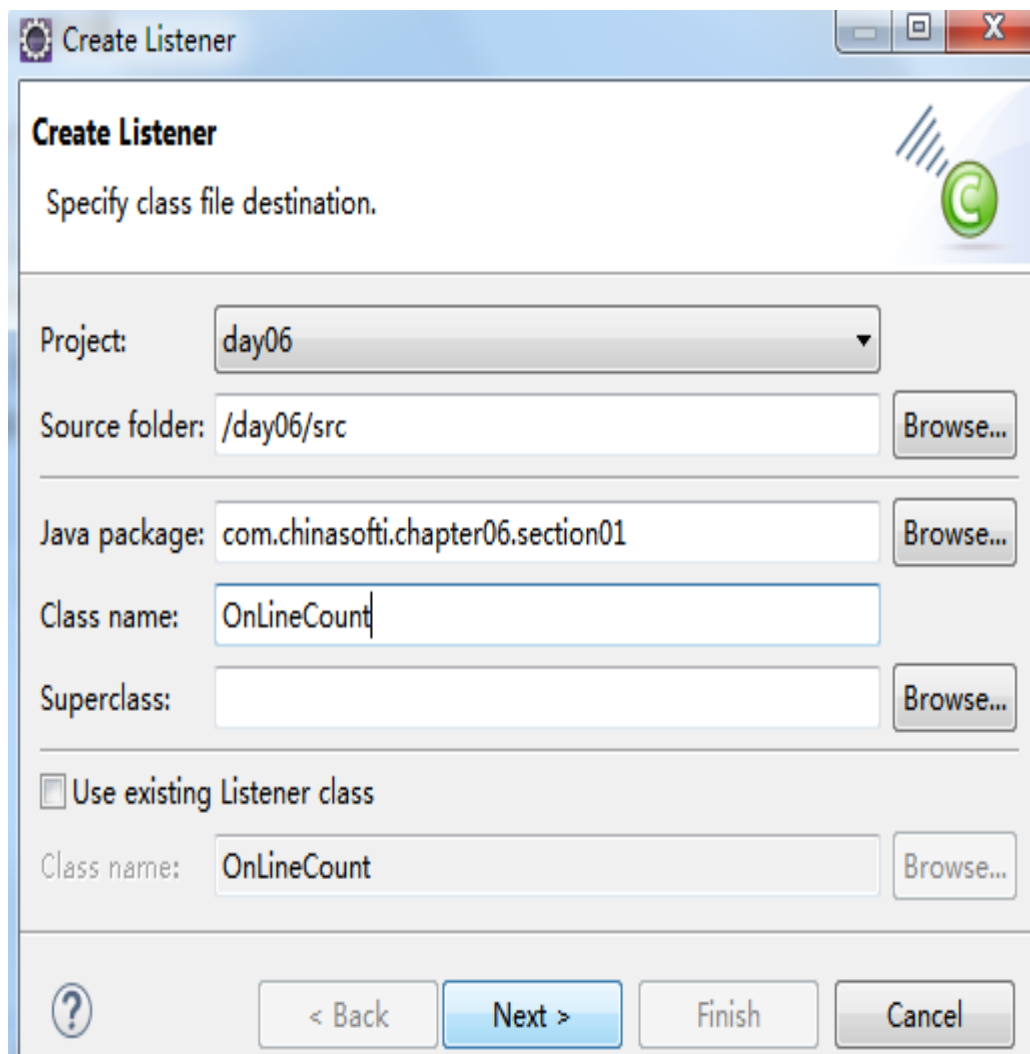
# 注解实现



```
@WebListener
public class ListenerTest implements ServletContextListener {
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("初始化！！！");
    }

    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("销毁！！！");
    }
}
```

# Eclipse中创建Listener



**Create Listener**

Specify class file destination.

Project:

Source folder:

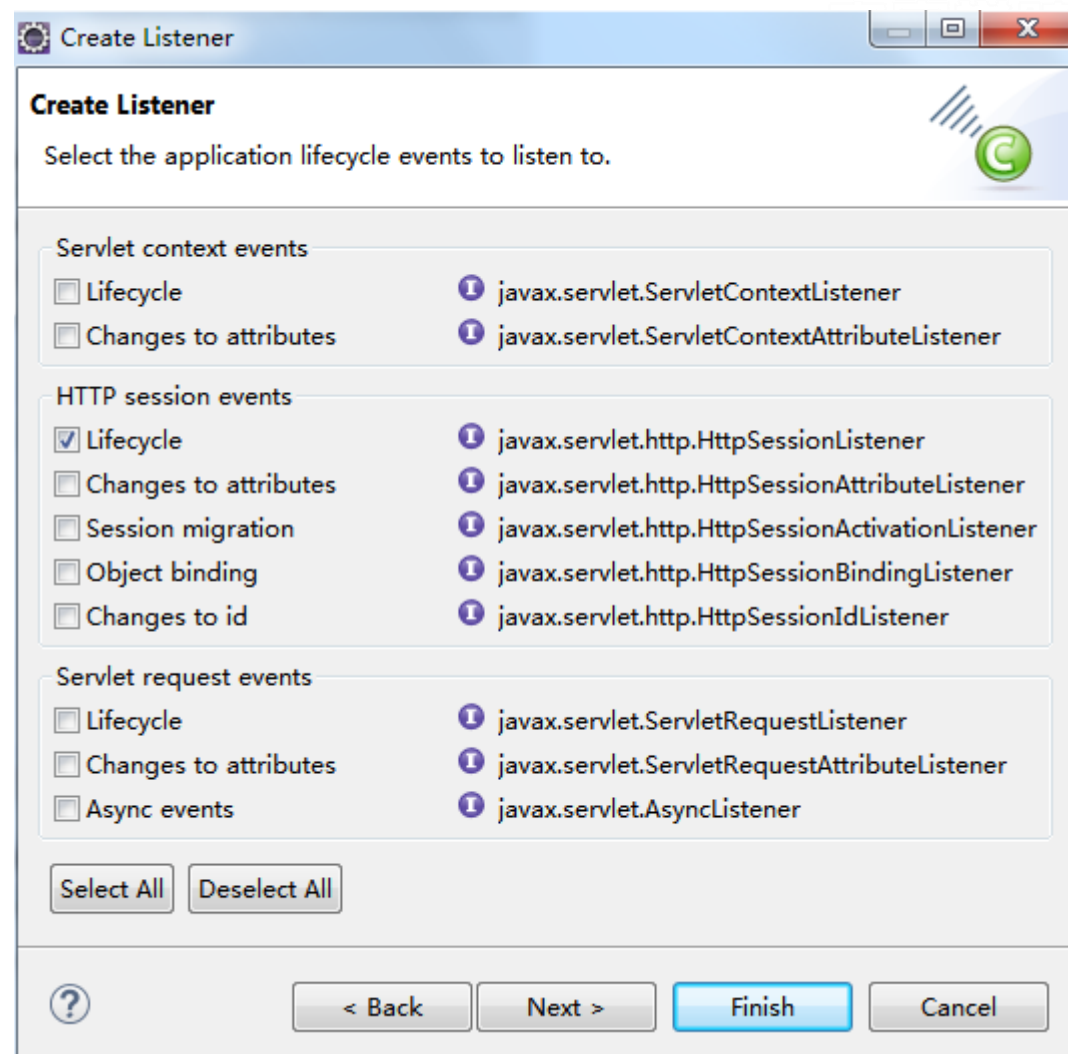
Java package:

Class name:

Superclass:

☐ Use existing Listener class

Class name:



**Create Listener**

Select the application lifecycle events to listen to.

**Servlet context events**

- ☐ Lifecycle  javax.servlet.ServletContextListener
- ☐ Changes to attributes  javax.servlet.ServletContextAttributeListener

**HTTP session events**

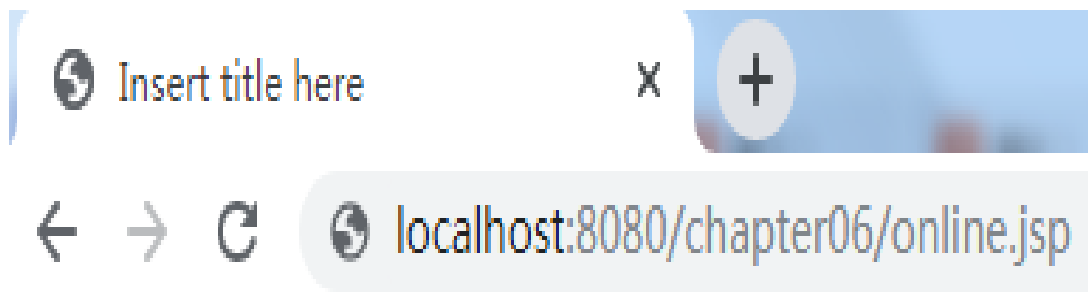
- ☒ Lifecycle  javax.servlet.http.HttpSessionListener
- ☐ Changes to attributes  javax.servlet.http.HttpSessionAttributeListener
- ☐ Session migration  javax.servlet.http.HttpSessionActivationListener
- ☐ Object binding  javax.servlet.http.HttpSessionBindingListener
- ☐ Changes to id  javax.servlet.http.HttpSessionIdListener

**Servlet request events**

- ☐ Lifecycle  javax.servlet.ServletRequestListener
- ☐ Changes to attributes  javax.servlet.ServletRequestAttributeListener
- ☐ Async events  javax.servlet.AsyncListener

```
@WebListener
public class OnLineCountListener implements HttpSessionListener {
    // 统计session数量
    private int count = 0 ;
    //监听session的创建
    public void sessionCreated(HttpSessionEvent se) {
        count++;
        se.getSession().setAttribute("Count", count);
    }
    //监听session的销毁
    public void sessionDestroyed(HttpSessionEvent se) {
        count--;
        se.getSession().setAttribute("Count", count); }
}
```

```
<body>  
    在线统计:  
    <%=session.getAttribute("Count") %>  
</body>
```



在线统计: 12



# servlet中的session监听和Attribute监听



servlet中对session的**监听**有很多接口，功能很灵活，最常用的是监听Session和Attribute。

这里要澄清一下概念，servlet中的session监听和Attribute监听含义有差别，session监听指的不是我们一般所理解的放置一个session或者销毁一个session，这是Attribute监听的功能，因为servlet中放置session的语法是session.setAttribute(“session名”,要放入的对象)。

而session监听，监听的是HTTP连接，只要有用户与server连接，就算连接的是一个空白的jsp页面，也会触发session事件，所以此处的session实际上指的是connection，用来统计当前在线用户数最合适了

# HttpSessionAttributeListener和HttpSessionBindingListener的区别-1



- 这两个监听器在文章中简称为BindingListener和AttributeListener.
- 1.BindingListener有两个方法
- valueBound(HttpSessionBindingEvent)
- 实现BindingListener接口的对象被**绑定**到session时触发valueBound事件,
- valueUnbound(HttpSessionBindingEvent)
- **解除绑定**时触发valueUnbound事件
- **注意**: 只有当该监听器(UserObject)保存到session中或从session移除时才会触发事件, 其他没有实现该listener对象保存到session时不会触发该事件。

## HttpSessionAttributeListener和HttpSessionBindingListener的区别-2



- 2.AttributeListener接口三个方法
- attributeAdded(HttpSessionBindingEvent),
- attributeRemoved(HttpSessionBindingEvent),
- attributeReplaced(HttpSeesionEvent)。
- 当在session中添加、移除或更改属性值时会触发相应的事件。

## HttpSessionAttributeListener和HttpSessionBindingListener的区别-3



- a.只有实现了HttpSessionBindingListener的类在和session绑定、解除绑定时触发其事件。
- b.实现了HttpSessionAttributeListener后任何对象（**不论**其是否实现了AttributeListener)在变化时均触发对应的事件。

## 本节总结提问【监听器】



- Servlet规范中定义了多少种事件，多少种监听器？
- 编写监听器的步骤是什么？
- 如何配置监听器？

## 本节总结 【监听器】



- Web应用中定义了六种事件类型，每种事件类型至少有一个监听器。分为上下文相关、会话相关、请求相关；
- 监听器共有8种，其中会话相关的4种，其他的分别两种；
- 使用监听器非常简单，只要自定义类实现相应的监听器接口，重写监听器里的方法即可；
- 监听器要生效，需要在web.xml中使用<listener>标签配置,也可以通过注解配置监听器；

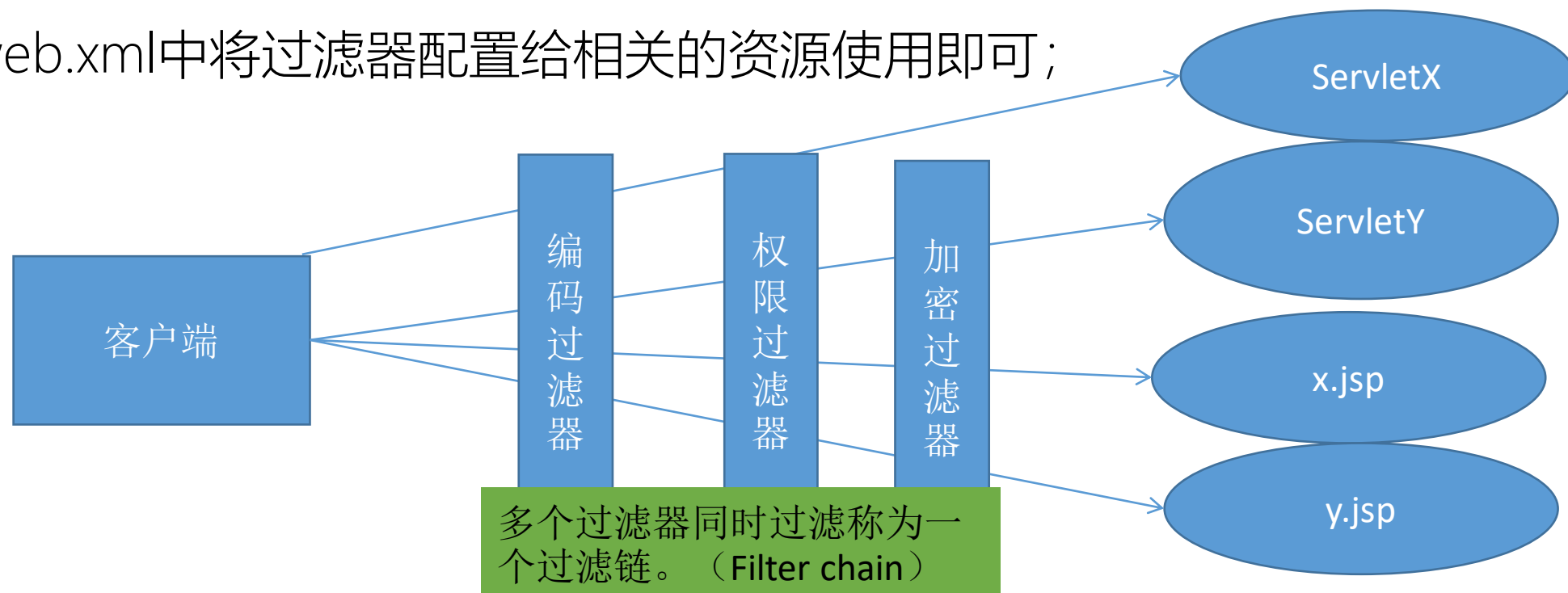
## 第2节 【过滤器】



- 知识点1：过滤器的作用
- 知识点2：过滤器的开发方法
- 知识点3：过滤器的配置
- 知识点4：利用过滤器实现访问控制
- 知识点5：防盗链等其他过滤器应用场景

## 知识点1：过滤器的作用

- 在Web应用中，如果对服务器端的多个资源（Servlet/JSP）有“通用”的处理，可以在每个资源中写相同的代码，而这样做显然过于冗余，修改时就需要逐一修改，效率低下；
- 过滤器可以解决这样的问题：把通用的、相同的处理代码用过滤器实现，然后在web.xml中将过滤器配置给相关的资源使用即可；





## 知识点2：过滤器的开发方法-1



- 过滤器的开发非常简单：
  - 自定义类实现Filter接口；
  - 实现接口中的方法，重点是doFilter方法：
- Filter接口中有三个方法，如下；

方法声明	方法描述
<code>void init(FilterConfig filterConfig)</code>	容器初始化过滤器对象后调用该方法，其中参数可以获取过滤器配置信息；
<code>void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)</code>	过滤器的服务方法，有三个参数，其中FilterChain中也定义了名字为doFilter方法，不过只有两个参数，可以把当前的请求和响应沿着过滤链进行传递；
<code>void destroy()</code>	容器销毁过滤器对象前进行调用；

## 知识点2：过滤器的开发方法-2



- 可见，除了Filter接口外，与过滤器有关的还有FilterConfig和FilterChain接口；
- 其中FilterConfig接口中定义了如下方法：

方法声明	方法描述
<code>java.lang.String getFilterName()</code>	返回web.xml中配置的Filter的名字信息；
<code>java.lang.String getInitParameter(java.lang.String name)</code>	返回web.xml中配置的Filter的初始化参数的值；与Servlet初始化参数类似，只能在当前的Filter中使用；
<code>java.util.Enumeration&lt;java.lang.String&gt; getInitParameterNames()</code>	返回web.xml中配置的Filter的所有初始化参数的名字；
<code>ServletContext getServletContext()</code>	返回当前的上下文对象；

## 知识点2：过滤器的开发方法-3



- 可见，除了Filter接口外，与过滤器有关的还有FilterConfig和FilterChain接口；
- 其中FilterChain接口中定义了如下方法：

方法声明	方法描述
doFilter(ServletRequest request, ServletResponse response)	如果当前的过滤器之后还有其他过滤器，则调用下一个过滤器；如果已经是最后一个过滤器，则调用目标资源；同时将请求和响应传到下一个资源；此方法是过滤器编程中最常用的方法；

# 注解创建

Create Filter

Specify class file destination.

Project: day06

Source folder: /day06/src Browse...

Java package: com.chinasofti.chapter06.section02 Browse...

Class name: IPFileterTest

Superclass: Browse...

☐ Use existing Filter class

Class name: IPFileterTest Browse...

< Back Next > Finish Cancel

Create Filter

Enter servlet filter deployment descriptor specific information.

Name: IPFilter

Description:

Initialization parameters:

Name	Value	Description
ip	0:0:0:0:0:0:1	

OK Cancel

Filter mappings:

URL Pattern / Servlet Name	Dispatchers
/IPFilter	

Add... Edit... Remove

☐ Asynchronous Support

< Back Next > Finish Cancel

```
@WebFilter(urlPatterns = { "/IPFilterTest" },  
           initParams = {  
               @WebInitParam(name = "ip", value = "0:0:0:0:0:0:0:1")  
           })
```

```
public class IPFilterTest implements Filter {  
    protected FilterConfig filterConfig;  
    protected String ip;
```

//初始化方法就是过滤器初始化的时候调用该方法，这个过滤器初始化的工作就是从配置文件中读取参数的内容

```
    public void init(FilterConfig fConfig) throws ServletException {  
        this.filterConfig = fConfig;  
        this.ip = this.filterConfig.getInitParameter("ip");  
        System.out.println(ip);  
    }
```

//该方法就是这个过滤器真正要执行的处理功能，本例主要是实现了对IP的限制

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

```
    String remoteIP = request.getRemoteAddr();
```

```
    System.out.println(remoteIP+" "+ip);
```

```
    if(remoteIP.equals(ip)) {
```

```
        response.setContentType("text/html;charset=utf-8");
```

```
        PrintWriter out = response.getWriter();
```

```
        out.println("<b>你的IP地址被禁用， <br>");
```

```
        System.out.println("不让你访问了！ ！ ！ ");
```

```
    }else{
```

```
        PrintWriter out = response.getWriter();out.println(remoteIP+" "+ip);
```

```
        chain.doFilter(request,response);
```

```
    }chain.doFilter(request, response);
```

```
}
```

## 知识点2：过滤器的开发方法-4

课堂案例：  
[CharacterFilter.java](#)

- 编写编码格式过滤器CharacterFilter;

```
public class CharacterFilter implements Filter {
    private String character;

    @Override
    public void destroy() {
    }

    @Override
    public void doFilter(ServletRequest arg0, ServletResponse arg1,
                        FilterChain arg2) throws IOException, ServletException {
        HttpServletRequest request=(HttpServletRequest)arg0;
        request.setCharacterEncoding(character);
        arg2.doFilter(arg0, arg1);
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        character=filterConfig.getInitParameter("character");
    }
}
```

## 知识点3：过滤器的配置-1



- 编写过滤器后，要使过滤器生效，必须在web.xml中进行配置，主要配置信息有：
  - 过滤器的名字和过滤器的类信息；
  - 过滤器对哪些URL生效；
  - 过滤器对这些URL生效时的访问方式；
  - 过滤器初始化参数；
- 一个过滤器可以配置给多个URL，一个URL也可以配置多个过滤器，按照配置顺序执行；多个过滤器组成一个过滤链；



## 知识点3：过滤器的配置



- 以前面的CharacterFilter为例，配置信息如下：

```
<filter>
  <filter-name>CharacterFilter</filter-name>
  <filter-class>com.chinasofti.chapter6.section02.filter.CharacterFilter</filter-class>
  <init-param>
    <param-name>character</param-name>
    <param-value>utf-8</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>CharacterFilter</filter-name>
  <url-pattern>*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>INCLUDE</dispatcher>
  <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

配置Filter的名字和类

配置Filter的初始化参数

配置过滤器能够过滤的资源URL，此处是对所有资源可以过滤

配置以何种方式访问url-pattern指定的资源才能被过滤：

- 1、REQUEST：直接URL访问、响应重定向、超级链接、表单提交、静态包含
- 2、FORWARD：请求转发
- 3、INCLUDE：动态包含【后续学习】
- 4、ERROR（错误页面跳转）

如果不配置，默认是REQUEST方式

## 知识点4：利用过滤器实现访问控制-1



- 实际应用中，有很多资源需要登录后才能访问，称为访问控制；
- 前面学习中我们实现过该功能；
- 然而，如果有多个资源都需要访问控制，则使用过滤器更为便捷；

```
@WebFilter("/*")
public class LoginFilter implements Filter {
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException,
        ServletException {
        String name = request.getParameter("username");
        String pass = request.getParameter("password");
        if(name!=null && pass !=null) {
            if("bianxw".equals(name)&&"111111".equals(pass)) {
                request.getRequestDispatcher("/loginsuccess.jsp").forward(request, response);
            }else {
                request.getRequestDispatcher("/login.jsp").forward(request, response);
            }
        }
        chain.doFilter(request, response);
    }
}
```

---

<center>

```
<form name = "form1" method = "post">
```

用户名:

密    码： <input type = "password" name = "password"><br>

&lt;/form&gt;

</center>

---

```
<body bgcolor = "blue">  
<center>  
    欢迎您访问我们的网站！ ！ ！  
</center>  
</body>
```

# 登录界面



http://localhost:8080/chapter06/login.jsp

用户名:

密 码:

用户名密码不符合要求不予登录

http://localhost:8080/chapter06/login.jsp

欢迎您访问我们的网站!!!

用户名为bianxw，密码为111111  
跳转到成功页面 loginsuccess.jsp

# 知识点5：防盗链等其他过滤器应用场景

- 什么是防盗链？



理想很丰满，现实很骨感系列



## 知识点5：防盗链等其他过滤器应用场景

- 明明引用了一个正确的图片地址，但显示出来的却是一个红叉或写有“此图片仅限于\*\*\*网站用户交流沟通使用”之类的“假图片”（下图便是网易博客的防盗链效果）。用嗅探软件找到了多媒体资源的真实地址用下载软件仍然不能下载。下载一些资源时总是出错，如果确认地址没错的话，大多数情况都是遇上防盗链系统了。常见的防盗链系统，一般使用在图片、音视频、软件等相关的资源上
- **盗链**是指**在自己的页面上展示一些并不在自己服务器上的内容**。通常的做法是通过技术手段获得它人服务器上的资源地址，绕过别人的资源展示页面，直接在自己的页面上向最终用户提供此内容。比较常见的是一些小站盗用大站的资源（图片、音乐、视频、软件等），对于这些小站来说，通过盗链的方法可以减轻自己服务器的负担，因为真实的空间和流量均是来自别人的服务器。
- 防盗链系统就是防范盗链的系统，防止别人通过一些技术手段绕过本站的资源展示页面，盗用本站的资源，让绕开本站资源展示页面的资源链接失效。实施防盗链系统后，因为屏蔽了那些盗链的间接资源请求，从而可以大大减轻服务器及带宽的压力，也正如此，越来越多的站点都开始实施防盗链技术



## 知识点5：防盗链等其他过滤器应用场景

- 防盗链的一个基本方法就是在请求资源时判定请求资源和发起请求的来源是否是同一个站点
- 在HTTP协议中，有一个表头字段叫referer，采用URL的格式来表示从哪儿链接到当前的网页或文件。换句话说，通过referer，网站可以检测目标网页访问的来源网页，如果是资源文件，则可以跟踪到显示它的网页地址。有了referer跟踪来源就好办了，这时就可以通过技术手段来进行处理，一旦检测到来源不是本站即进行阻止或者返回指定的页面

## 知识点5：防盗链等其他过滤器应用场景



- 防盗链的基本方法如下：

```
String path = request.getContextPath();  
// 获取本站截至到context root的域名信息  
String basePath = request.getScheme() + "://" + request.getServerName()  
                  + ":" + request.getServerPort() + path + "/";  
  
// 获取上一个页面的地址  
String fromUrl = request.getHeader("referer");  
// 判定是否外站请求并返回结果  
return fromUrl != null && fromUrl.startsWith(basePath) ? true : false;
```

- 可以将该代码注册为过滤器，用于保护各类动态、静态资源
- 利用该方式还可以完成基本的钓鱼网站请求判定和请求参数纠错等

## 本节总结提问【过滤器】

- 过滤器有什么作用?
- 和过滤器有关的接口有哪三个?
- 过滤器主要配置哪些信息?



## 本节总结【过滤器】



- 过滤器可以对一些资源进行统一处理，例如访问控制、加密、压缩等；
- 和过滤器有关的接口有Filter、FilterConfig、FilterChain；
- 过滤器需要在web.xml中进行配置，包括过滤器的名字和类、过滤器初始化参数、过滤的URL模式、过滤资源的访问方式；

# 本章总结



- 本章主要学习了监听器和过滤器;
- 监听器用来监听事件, 对事件进行处理; 定义了六种事件类型和八种监听器接口类型;
- 过滤器用来对资源进行统一的处理; 定义了三个接口支持过滤器编程;
- 二者都需要在web.xml中配置或注解生成方能生效;

# 本章作业



- 作业1:
- 题目：使用监听器实现某网站的访问量
- 难度：中
- 作业2:
- 题目：使用过滤器实现权限验证，只有角色是admin的用户才能访问；其他角色的用户不能访问。
- 难度：中

