# Listener概述

Listener是Servlet规范的另一个高级特性。Listener用于监听JavaWeb程序中的事件，例如创建、修改、删除Session、request、Context等,并触发相应事件。利用Listener能够用很少的代码实现很绚丽的效果。

Listener也是Servlet规范2.3中加入的，对应设计模式中的Listener模式，事件发生的时候会自动触发该事件对应的Listener。Listener主要用于对Session、request、Context等进行监控。目前的Servlet3.1规范中共有8中Listener，分别完成对不同事件的监听。

# Listener的分类

Servlet3.1规范中共有8种Listener，分别用于监听Session、Request、Context等的创建和销毁、属性变化等。另有一个Listener能够监听存放在Session中的对象。共有6中Event。下面我们一一来认识一下

## 用户创建和销毁的监听器

HttpSessionListener、ServletContextListener、ServletRequestListener分别用于监听Session、Context、request的创建与销毁。触发时机分别为：

HttpSessionListener：监听Session的创建与销毁。创建Session时执行sessionCreated（HttpSessionEvent se）方法。Session超时或者执行session.invalidate（）执行sessionDestroyed(HttpSessionEvent se)方法。该Listener可用于收集在线者信息。

ServletContextListener：监听Context的创建与销毁。Context代表当前的web应用程序。服务器启动或者热部署war包时执行contextInitialized(ServletContextEvent event)方法。服务器关闭时，或者只关闭该web时执行contextDestroyed(ServletContextEvent event)方法。该Listener可用于启动时获取web.xml里配置的初始化参数。

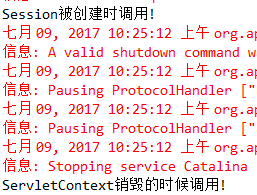
ServletRequestListener：监听request的创建和销毁。用户每次请求request都会执行requestInitialized(ServletRequesEvent event)方法。Request处理完毕自动销毁前执行requestDestroyed(ServletRequestEvent event)方法。注意如果一个HTML页面内包含有多个图片，则请求一次HTML页面可能会触发多次request事件。

### 练习：

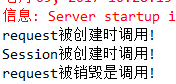
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"* id=*"WebApp\_ID"* version=*"3.0"*>  <display-name>listener</display-name>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  <listener>  <listener-class>com.practice.listener.AListener</listener-class>  </listener>  <listener>  <listener-class>com.practice.listener.SessionListener</listener-class>  </listener>  <listener>  <listener-class>com.practice.listener.RequestListener</listener-class>  </listener>  </web-app> |
| /\*  \* ServletContextListener实现类  \* contextDestroyed() -- 在ServletContext对象被销毁前调用  \* contextInitialized() -- -- 在ServletContext对象被创建后调用  \* ServletContextEvent -- 事件类对象  \* 该类有getServletContext()，用来获取ServletContext对象，即获取事件源对象  \*/  package com.practice.listener;  import javax.servlet.ServletContextEvent;  import javax.servlet.ServletContextListener;  /\*\*  \* 监听ServletContext的一生一死  \* @author Yorick  \*  \*/  public class AListener implements ServletContextListener {  @Override  public void contextDestroyed(ServletContextEvent arg0) {  System.out.println("ServletContext销毁的时候调用!");  }  @Override  public void contextInitialized(ServletContextEvent arg0) {  System.out.println("ServletContext创建的时候调用!");  }  } |



|  |
| --- |
| /\*  \* HttpSessionListener实现类  \* sessionCreated() -- 在HttpSession对象被创建后被调用  \* sessionDestroyed() -- -- 在HttpSession对象被销毁前调用  \* HttpSessionEvent -- 事件类对象  \* 该类有getSession()，用来获取当前HttpSession对象，即获取事件源对象  \*/  **package** com.practice.listener;  **import** javax.servlet.http.HttpSessionEvent;  **import** javax.servlet.http.HttpSessionListener;  **public** **class** SessionListener **implements** HttpSessionListener{  @Override  **public** **void** sessionCreated(HttpSessionEvent arg0) {  System.*out*.println("Session被创建时调用!");  }  @Override  **public** **void** sessionDestroyed(HttpSessionEvent arg0) {  System.*out*.println("Session被销毁是调用!");  }  } |



|  |
| --- |
| /\*  \* ServletRequestListener实现类  \* requestDestroyed() -- 在ServletRequest对象被销毁前调用  \* requestInitialized() -- 在ServletRequest对象被创建后调用  \* ServletRequestEvent -- 事件类对象  \* 该类有getServletContext()，用来获取ServletContext对象  \* 该类有getServletRequest()，用来获取当前ServletRequest对象，即事件源对象  \*/  **package** com.practice.listener;  **import** javax.servlet.ServletRequestEvent;  **import** javax.servlet.ServletRequestListener;  **public** **class** RequestListener **implements** ServletRequestListener{  @Override  **public** **void** requestDestroyed(ServletRequestEvent arg0) {  System.*out*.println("request被销毁是调用!");  }  @Override  **public** **void** requestInitialized(ServletRequestEvent arg0) {  System.*out*.println("request被创建时调用!");  }  } |



## 操作域属性的监听器

除了用于监听创建和销毁的监听器外,还有用于监听属性变化的监听器,下面我们就来认识一下:

当对域属性进行增、删、改时，执行的监听器一共有三个：

* ServletContextAttributeListener：在ServletContext域进行增、删、改属性时调用下面方法。
* public void attributeAdded(ServletContextAttributeEvent evt)
* public void attributeRemoved(ServletContextAttributeEvent evt)
* public void attributeReplaced(ServletContextAttributeEvent evt)
* HttpSessionAttributeListener：在HttpSession域进行增、删、改属性时调用下面方法
* public void attributeAdded(HttpSessionBindingEvent evt)
* public void attributeRemoved (HttpSessionBindingEvent evt)
* public void attributeReplaced (HttpSessionBindingEvent evt)
* ServletRequestAttributeListener：在ServletRequest域进行增、删、改属性时调用下面方法
* public void attributeAdded(ServletRequestAttributeEvent evt)
* public void attributeRemoved (ServletRequestAttributeEvent evt)
* public void attributeReplaced (ServletRequestAttributeEvent evt)

下面对这三个监听器的事件对象功能进行介绍：

* ServletContextAttributeEvent
* String getName()：获取当前操作的属性名；
* Object getValue()：获取当前操作的属性值；
* ServletContext getServletContext()：获取ServletContext对象。
* HttpSessionBindingEvent
* String getName()：获取当前操作的属性名；
* Object getValue()：获取当前操作的属性值；
* HttpSession getSession()：获取当前操作的session对象。
* ServletRequestAttributeEvent
* String getName()：获取当前操作的属性名；
* Object getValue()：获取当前操作的属性值；
* ServletContext getServletContext()：获取ServletContext对象；
* ServletRequest getServletRequest()：获取当前操作的ServletRequest对象。

|  |
| --- |
| **public** **class** MyListener **implements** ServletContextAttributeListener,  ServletRequestAttributeListener, HttpSessionAttributeListener {  **public** **void** attributeAdded(HttpSessionBindingEvent evt) {  System.*out*.println("向session中添加属性：" + evt.getName() + "=" + evt.getValue());  }  **public** **void** attributeRemoved(HttpSessionBindingEvent evt) {  System.*out*.println("从session中移除属性：" + evt.getName() + "=" + evt.getValue());  }  **public** **void** attributeReplaced(HttpSessionBindingEvent evt) {  System.*out*.println("修改session中的属性：" + evt.getName() + "=" + evt.getValue());  }  **public** **void** attributeAdded(ServletRequestAttributeEvent evt) {  System.*out*.println("向request中添加属性：" + evt.getName() + "=" + evt.getValue());  }  **public** **void** attributeRemoved(ServletRequestAttributeEvent evt) {  System.*out*.println("从request中移除属性：" + evt.getName() + "=" + evt.getValue());  }  **public** **void** attributeReplaced(ServletRequestAttributeEvent evt) {  System.*out*.println("修改request中的属性：" + evt.getName() + "=" + evt.getValue());  }  **public** **void** attributeAdded(ServletContextAttributeEvent evt) {  System.*out*.println("向context中添加属性：" + evt.getName() + "=" + evt.getValue());  }  **public** **void** attributeRemoved(ServletContextAttributeEvent evt) {  System.*out*.println("从context中移除属性：" + evt.getName() + "=" + evt.getValue());  }  **public** **void** attributeReplaced(ServletContextAttributeEvent evt) {  System.*out*.println("修改context中的属性：" + evt.getName() + "=" + evt.getValue());  }  } |
| **public** **class** ListenerServlet **extends** BaseServlet {  **public** String contextOperation(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  ServletContext context = **this**.getServletContext();  context.setAttribute("a", "a");  context.setAttribute("a", "A");  context.removeAttribute("a");  **return** "/index.jsp";  }    ///////////////////////////////    **public** String sessionOperation(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  HttpSession session = request.getSession();  session.setAttribute("a", "a");  session.setAttribute("a", "A");  session.removeAttribute("a");  **return** "/index.jsp";  }  ///////////////////////////////    **public** String requestOperation(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  request.setAttribute("a", "a");  request.setAttribute("a", "A");  request.removeAttribute("a");  **return** "/index.jsp";  }  } |
| <body>  <a href=*"*<c:url value=*'/ListenerServlet?method=contextOperation'*/>*"*>SevletContext操作属性</a>  <br/>  <a href=*"*<c:url value=*'/ListenerServlet?method=sessionOperation'*/>*"*>HttpSession操作属性</a>  <br/>  <a href=*"*<c:url value=*'/ListenerServlet?method=requestOperation'*/>*"*>ServletRequest操作属性</a> |  </body> |

### 两个特殊的监听Session内的对象的Listener

HttpSessionBindingListener：当对象被放到Session里是执行valueBound（HttpSessionBindingEvent event）方法。当对象被从Session里移除时执行valueUnbound（HttpSessionBindingEvent event）方法。对象必须实现该Listener接口。

HttpSessionActivationListener：服务器关闭时会将session里的内容保存到硬盘上，这个过程叫做钝化。服务器重新启动时，会将Session内容从硬盘重新加载。当session里的对象被钝化时会执行sessionWillPassivate(HttpSessionEvent event)方法，当对象被重新加载时执行sessionDidActivate(HttpSessionEvent event)。对象必须实现该Listener接口(对象若想保存到硬盘要实现序列化接口)。

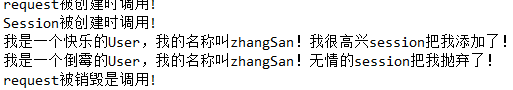
关闭序列化:在context.xml中配置命令

|  |
| --- |
| <Context>  <WatchedResource>WEB-INF/web.xml</WatchedResource>  <Manager pathname="" />  </Context> |
| 注意:   * 不用在web.xml文件中部署； * 这两个监听器不是给session添加，而是给Bean添加。即让Bean类实现监听器接口，然后再把Bean对象添加到session域中。   下面对这两个监听器介绍一下：   * HttpSessionBindingListener：当某个类实现了该接口后，可以感知本类对象添加到session中，以及感知从session中移除。例如让Person类实现HttpSessionBindingListener接口，那么当把Person对象添加到session中，或者把Person对象从session中移除时会调用下面两个方法： * public void valueBound([HttpSessionBindingEvent](mk:@MSITStore:F:\\帮助\\J2EE1.5.chm::/javax/servlet/http/HttpSessionBindingEvent.html" \o "class in javax.servlet.http) event)：当把监听器对象添加到session中会调用监听器对象的本方法； * public void valueUnbound([HttpSessionBindingEvent](mk:@MSITStore:F:\\帮助\\J2EE1.5.chm::/javax/servlet/http/HttpSessionBindingEvent.html" \o "class in javax.servlet.http) event)：当把监听器对象从session中移除时会调用监听器对象的本方法；   这里要注意，HttpSessionBindingListener监听器的使用与前面介绍的都不相同，当该监听器对象添加到session中，或把该监听器对象从session移除时会调用监听器中的方法。并且无需在web.xml文件中部署这个监听器。 |
| * HttpSessionActivationListener：Tomcat会在session从时间不被使用时钝化session对象，所   谓钝化session，就是把session通过序列化的方式保存到硬盘文件中。当用户再使用session时，Tomcat还会把钝化的对象再活化session，所谓活化就是把硬盘文件中的session在反序列化回内存。当session被Tomcat钝化时，session中存储的对象也被纯化，当session被活化时，也会把session中存储的对象活化。如果某个类实现了HttpSessionActiveationListener接口后，当对象随着session被钝化和活化时，下面两个方法就会被调用：   * public void sessionWillPassivate([HttpSessionEvent](mk:@MSITStore:F:\\帮助\\J2EE1.5.chm::/javax/servlet/http/HttpSessionEvent.html" \o "class in javax.servlet.http) se)：当对象感知被活化时调用本方法； * public void sessionDidActivate([HttpSessionEvent](mk:@MSITStore:F:\\帮助\\J2EE1.5.chm::/javax/servlet/http/HttpSessionEvent.html" \o "class in javax.servlet.http) se)：当对象感知被钝化时调用本方法；   HttpSessionActivationListener监听器与HttpSessionBindingListener监听器相似，都是感知型的监听器，例如让Person类实现了HttpSessionActivationListener监听器接口，并把Person对象添加到了session中后，当Tomcat钝化session时，同时也会钝化session中的Person对象，这时Person对象就会感知到自己被钝化了，其实就是调用Person对象的sessionWillPassivate()方法。当用户再次使用session时，Tomcat会活化session，这时Person会感知到自己被活化，其实就是调用Person对象的sessionDidActivate()方法。  注意，因为钝化和活化session，其实就是使用序列化和反序列化技术把session从内存保存到硬盘，和把session从硬盘加载到内存。这说明如果Person类没有实现Serializable接口，那么当session钝化时就不会钝化Person，而是把Person从session中移除再钝化！这也说明session活化后，session中就不在有Person对象了。 |

##### 练习：

A.jsp b.jsp

|  |
| --- |
| <body>  <%session.setAttribute("user", **new** com.practice.listener.User("zhangSan", "123")); %>  <%-- <%session.removeAttribute("user"); %> --%>  <%request.getRequestDispatcher("b.jsp").forward(request, response); %>  </body> |
| <body>  <%session.removeAttribute("user");%>  </body> |
| package com.practice.listener;  import javax.servlet.http.HttpSessionBindingEvent;  import javax.servlet.http.HttpSessionBindingListener;  public class User implements HttpSessionBindingListener{  private String username;  private String password;  public String getUsername() {  return username;  }  public void setUsername(String username) {  this.username = username;  }  public String getPassword() {  return password;  }  public void setPassword(String password) {  this.password = password;  }  public User(String username, String password) {  super();  this.username = username;  this.password = password;  }  public User() {  super();  // TODO Auto-generated constructor stub  }  @Override  public String toString() {  return "User [username=" + username + ", password=" + password + "]";  }  @Override  public void valueBound(HttpSessionBindingEvent evt) {  /\*  \* 当前对象被添加到session中时被调用  \*/  System.out.println("我是一个快乐的User，我的名称叫" + username + "！我很高兴session把我添加了！");  }  @Override  public void valueUnbound(HttpSessionBindingEvent arg0) {  /\*  \* 当前对象被session移除时被调用  \*/  System.out.println("我是一个倒霉的User，我的名称叫" + username + "！无情的session把我抛弃了！");  }  } |



|  |
| --- |
| <%  out.print(request.getLocale() + "<br/>");  session.setAttribute("msg", **new** com.practice.listener.Person("zhangSan"));  %> |
| <%  out.print(session.getAttribute("msg"));  %> |
| **package** com.practice.listener;  **import** java.io.Serializable;  **import** javax.servlet.http.HttpSessionActivationListener;  **import** javax.servlet.http.HttpSessionEvent;  /\*\*  \* 如果不实现序列化接口,那么既是实现了HttpSessionActivationListener也不会将对象保存到硬盘  \* 想要将对象保存到硬盘必须实现序列化接口  \* **@author** Yorick  \*/  **public** **class** Person **implements** Serializable,HttpSessionActivationListener{  **private** String username;  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** Person() {  **super**();  // **TODO** Auto-generated constructor stub  }  **public** Person(String username) {  **super**();  **this**.username = username;  }  @Override  **public** String toString() {  **return** "User [username=" + username + "]";  }  /\*\*  \* 活化时被调用  \*/  **public** **void** sessionDidActivate(HttpSessionEvent evt) {  System.*out*.println("哇哇哇！我和session从重地球了！");  }  /\*\*  \* 钝化时被调用  \*/  **public** **void** sessionWillPassivate(HttpSessionEvent evt) {  System.*out*.println("啊~我和session一起去火星了！");  }  } |





## 事件对象

* ServletContextEvent：ServletContext getServletContext()；
* HttpSessionBindingEvent
* ServletContextAttributeEvent
* ServletRequestAttributeEvent
* HttpSeessionEvent：HttpSession getSession()；
* ServletRequestEvent：
  + ServletRequest getServletRequest()
  + ServletContext getServletContext()