# day20 jdbc分页-监听器

## JavaWeb监听器

三大组件：

* Servlet
* Listener
* Filter

Listener：监听器

1. 初次相见：AWT
2. 二次相见：SAX

监听器：

* 它是一个接口，内容由我们来实现；
* 它需要注册，例如注册在按钮上！
* 监听器中的方法，会在特殊事件发生时被调用！

观察者：

* 事件源；
* 小偷
* 事件；
* 偷东西
* 监听器；
* 警察
* 监听器中的方法：抓捕

JavaWeb中的监听器

* 事件源：三大域！
* ServletContext
* 生命周期监听：ServletContextListener，它有两个方法，一个在出生时调用，一个在死亡时调用；
* void **contextInitialized**([ServletContextEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/ServletContextEvent.html) sce)：创建SErvletcontext时
* void **contextDestroyed**([ServletContextEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/ServletContextEvent.html) sce)：销毁Servletcontext时
* 属性监听：ServletContextAttributeListener，它有三个方法，一个在添加属性时调用，一个在替换属性时调用，最后一个是在移除属性时调用。
* void **attributeAdded**([ServletContextAttributeEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/ServletContextAttributeEvent.html) event)：添加属性时；
* void **attributeReplaced**([ServletContextAttributeEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/ServletContextAttributeEvent.html) event)：替换属性时；
* void **attributeRemoved**([ServletContextAttributeEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/ServletContextAttributeEvent.html) event)：移除属性时；
* HttpSession
* 生命周期监听：HttpSessionListener，它有两个方法，一个在出生时调用，一个在死亡时调用；
* void **sessionCreated**([HttpSessionEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/http/HttpSessionEvent.html) se)：创建session时
* void **sessionDestroyed**([HttpSessionEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/http/HttpSessionEvent.html) se)：销毁session时
* 属性监听：HttpSessioniAttributeListener，它有三个方法，一个在添加属性时调用，一个在替换属性时调用，最后一个是在移除属性时调用。
* void **attributeAdded**([HttpSessionBindingEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/http/HttpSessionBindingEvent.html) event)：添加属性时；
* void **attributeReplaced**([HttpSessionBindingEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/http/HttpSessionBindingEvent.html) event)：替换属性时
* void **attributeRemoved**([HttpSessionBindingEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/http/HttpSessionBindingEvent.html) event)：移除属性时
* ServletRequest
* 生命周期监听：ServletRequestListener，它有两个方法，一个在出生时调用，一个在死亡时调用；
* void **requestInitialized**([ServletRequestEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/ServletRequestEvent.html) sre)：创建request时
* void **requestDestroyed**([ServletRequestEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/ServletRequestEvent.html) sre)：销毁request时
* 属性监听：ServletRequestAttributeListener，它有三个方法，一个在添加属性时调用，一个在替换属性时调用，最后一个是在移除属性时调用。
* void **attributeAdded**([ServletRequestAttributeEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/ServletRequestAttributeEvent.html) srae)：添加属性时
* void **attributeReplaced**([ServletRequestAttributeEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/ServletRequestAttributeEvent.html) srae)：替换属性时
* void **attributeRemoved**([ServletRequestAttributeEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/ServletRequestAttributeEvent.html) srae)：移除属性时
* javaWeb中完成编写监听器：
* 写一个监听器类：要求必须去实现某个监听器接口；
* 注册，是在web.xml中配置来完成注册！
* 事件对象：
* ServletContextEvent：ServletContext getServletContext()
* HttpSessionEvent：HttpSession getSession()
* ServletRequest：
* ServletContext getServletContext()；
* ServletReques getServletRequest()；
* ServletContextAttributeEvent：
* ServletContext getServletContext()；
* String getName()：获取属性名
* Object getValue()：获取属性值
* [HttpSessionBindingEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/http/HttpSessionBindingEvent.html)：略
* [ServletRequestAttributeEvent](mk:@MSITStore:F:\F\帮助\j2EE6.0.chm::/javax/servlet/ServletRequestAttributeEvent.html) ：略

感知监听（都与HttpSession相关）

* 它用来添加到JavaBean上，而不是添加到三大域上！
* 这两个监听器都不需要在web.xml中注册！

HttpSessionBindingListener：添加到javabean上，javabean就知道自己是否添加到session中了。

###### 1　JavaWeb监听器概述

在JavaWeb被监听的事件源为：ServletContext、HttpSession、ServletRequest，即三大域对象。

* 监听域对象“创建”与“销毁”的监听器；
* 监听域对象“操作域属性”的监听器；
* 监听HttpSession的监听器。

###### 2　创建与销毁监听器

创建与销毁监听器一共有三个：

* ServletContextListener：Tomcat启动和关闭时调用下面两个方法
* public void contextInitialized(ServletContextEvent evt)：ServletContext对象被创建后调用；
* public void contextDestroyed(ServletContextEvent evt)：ServletContext对象被销毁前调用；
* HttpSessionListener：开始会话和结束会话时调用下面两个方法
* public void sessionCreated(HttpSessionEvent evt)：HttpSession对象被创建后调用；
* public void sessionDestroyed(HttpSessionEvent evt)：HttpSession对象被销毁前调用；
* ServletRequestListener：开始请求和结束请求时调用下面两个方法
* public void requestInitiallized(ServletRequestEvent evt)：ServletRequest对象被创建后调用；
* public void requestDestroyed(ServletRequestEvent evt)：ServletRequest对象被销毁前调用。

##### 事件对象

* ServletContextEvent：ServletContext getServletContext()；
* HttpSeessionEvent：HttpSession getSession()；
* ServletRequestEvent：
* ServletRequest getServletRequest()
* ServletContext getServletContext()

编写测试例子：

* 编写MyServletContextListener类，实现ServletContextListener接口；
* 在web.xml文件中部署监听器；
* 为了看到session销毁的效果，在web.xml文件中设置session失效时间为1分钟；

|  |
| --- |
| /\*  \* 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  \*/  **public** **class** MyServletContextListener **implements** ServletContextListener {  **public** **void** contextDestroyed(ServletContextEvent evt) {  System.*out*.println("销毁ServletContext对象");  }  **public** **void** contextInitialized(ServletContextEvent evt) {  System.*out*.println("创建ServletContext对象");  }  } |
| /\*  \* 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  \*/  **public** **class** MyHttpSessionListener **implements** HttpSessionListener {  **public** **void** sessionCreated(HttpSessionEvent evt) {  System.*out*.println("创建session对象");  }  **public** **void** sessionDestroyed(HttpSessionEvent evt) {  System.*out*.println("销毁session对象");  }  } |
| /\*  \* 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  \*/  **public** **class** MyServletRequestListener **implements** ServletRequestListener {  **public** **void** requestDestroyed(ServletRequestEvent evt) {  System.*out*.println("销毁request对象");  }  **public** **void** requestInitialized(ServletRequestEvent evt) {  System.*out*.println("创建request对象");  }  } |
| <listener>  <listener-class>cn.itcast.listener.MyServletContextListener</listener-class>  </listener>  <listener>  <listener-class>cn.itcast.listener.MyHttpSessionListener</listener-class>  </listener>  <listener>  <listener-class>cn.itcast.listener.MyServletRequestListener</listener-class>  </listener>  <session-config>  <session-timeout>1</session-timeout>  </session-config> |

###### 3　操作域属性的监听器

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> |

###### 4　HttpSession的监听器

还有两个与HttpSession相关的特殊的监听器，这两个监听器的特点如下：

* 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) 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) 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。

示例步骤：

* 编写Person类，让其实现HttpSessionBindingListener监听器接口；
* 编写Servlet类，一个方法向session中添加Person对象，另一个从session中移除Person对象；
* 在index.jsp中给出两个超链接，分别访问Servlet中的两个方法。

Pseron.java

|  |
| --- |
| **public** **class** Person **implements** HttpSessionBindingListener {  **private** String name;  **private** **int** age;  **private** String sex;    **public** Person(String name, **int** age, String sex) {  **super**();  **this**.name = name;  **this**.age = age;  **this**.sex = sex;  }  **public** Person() {  **super**();  }  **public** String toString() {  **return** "Person [name=" + name + ", age=" + age + ", sex=" + sex + "]";  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }  **public** String getSex() {  **return** sex;  }  **public** **void** setSex(String sex) {  **this**.sex = sex;  }  **public** **void** valueBound(HttpSessionBindingEvent evt) {  System.*out*.println("把Person对象存放到session中：" + evt.getValue());  }  **public** **void** valueUnbound(HttpSessionBindingEvent evt) {  System.*out*.println("从session中移除Pseron对象：" + evt.getValue());  }  } |

ListenerServlet.java

|  |
| --- |
| **public** **class** ListenerServlet **extends** BaseServlet {  **public** String addPerson(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  Person p = **new** Person("zhangSan", 23, "male");  request.getSession().setAttribute("person", p);  **return** "/index.jsp";  }    **public** String removePerson(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  request.getSession().removeAttribute("person");  **return** "/index.jsp";  } |

index.jsp

|  |
| --- |
| <body>  <a href=*"*<c:url value=*'/ListenerServlet?method=addPerson'*/>*"*>addPerson</a>  <br/>  <a href=*"*<c:url value=*'/ListenerServlet?method=removePerson'*/>*"*>removePerson</a>  <br/>  </body> |

* 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) 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) 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。

示例步骤：

* 先不管HttpSessionActivationListener监听器接口，先来配置Tomcat钝化session的参数，把下面配置文件放到tomcat\conf\catalina\localhost目录下！文件名称为项目名称。

|  |
| --- |
| <Context>  <Manager className=*"org.apache.catalina.session.PersistentManager"* maxIdleSwap=*"1"*>  <Store className=*"org.apache.catalina.session.FileStore"* directory=*"mysession"*/>  </Manager>  </Context> |

　　　　访问项目的index.jsp页面，这会使Tomcat创建Session对象，然后等待一分钟后，查看Tomcat\work\Catalina\localhost\listener\mysession目录下是否会产生文件，如果产生了，说明钝化session的配置成功了，可以开始下一步了。

* 创建Person类，让Person类实现HttpSessionActivationListener和Serializable接口：

Person.java

|  |
| --- |
| **public** **class** Person **implements** HttpSessionActivationListener, Serializable {  **private** String name;  **private** **int** age;  **private** String sex;    **public** Person(String name, **int** age, String sex) {  **super**();  **this**.name = name;  **this**.age = age;  **this**.sex = sex;  }  **public** Person() {  **super**();  }  **public** String toString() {  **return** "Person [name=" + name + ", age=" + age + ", sex=" + sex + "]";  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }  **public** String getSex() {  **return** sex;  }  **public** **void** setSex(String sex) {  **this**.sex = sex;  }  **public** **void** sessionDidActivate(HttpSessionEvent evt) {  System.*out*.println("session已经活化");  }  **public** **void** sessionWillPassivate(HttpSessionEvent evt) {  System.*out*.println("session被钝化了！");  }  } |

* 与上例一样，编写Servlet，提供两个方法：一个向session中添加Person对象，另一个从session中移除Person对象：

Person.java

|  |
| --- |
| **public** **class** ListenerServlet **extends** BaseServlet {  **public** String addPerson(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  Person p = **new** Person("zhangSan", 23, "male");  request.getSession().setAttribute("person", p);  **return** "/index.jsp";  }    **public** String removePerson(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  request.getSession().removeAttribute("person");  **return** "/index.jsp";  }  } |

* 在index.jsp页面中给出访问addPerson()和removePerson()的方法：

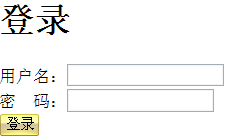
|  |
| --- |
| <body>  <a href=*"*<c:url value=*'/ListenerServlet?method=addPerson'*/>*"*>addPerson</a>  <br/>  <a href=*"*<c:url value=*'/ListenerServlet?method=removePerson'*/>*"*>removePerson</a>  <br/>  </body> |

* 打开index.jsp页面，这时Tomcat会创建session，必须在1分钟之前点击addPerson链接，这能保证在session被钝化之前把Person对象添加到session中；
* 等待一分钟，这时session会被钝化，也就会调用Person的sessionWillPassivate()；
* 刷新一下index.jsp页面，这会使session活化，会调用Person的sessionDidActivate()方法。

## 国际化

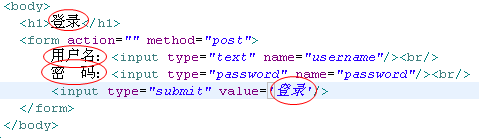
###### 1　什么是国际化

国际化就是可以把页面中的中文变成英文。例如在页面中的登录表单：

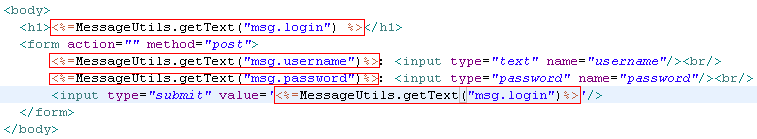


###### 2　理解国际化

想把页面中的文字修改，那么就不能再使用硬编码，例如下面的页面中都是硬编码：



上图中的中文想转换成英文，那么就需要把它们都变成活编码：



###### 3　Locale类

创建Locale类对象：

* new Locale(“zh”, “CN”)；
* new Locale(“en”, “US”)；

你一定对zh、CN或是en、US并不陌生，zh、en表示语言，而CN、US表示国家。一个Locale对象表示的就是语言和国家。

###### 4　ResourceBundle类

ReourceBundle类用来获取配置文件中的内容。

下面是两个配置文件内容：

res\_zh\_CN.properties



res\_en\_US.properties



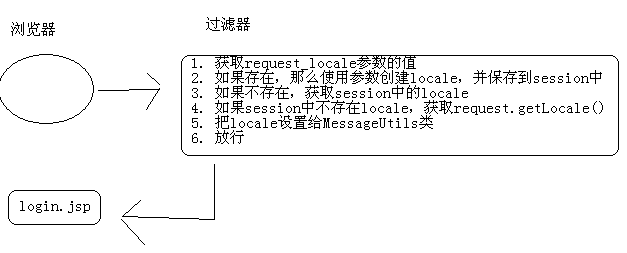
|  |
| --- |
| **public** **class** Demo1 {  @Test  **public** **void** fun1() {  ResourceBundle rb = ResourceBundle.*getBundle*("res", **new** Locale("zh", "CN" ));  String username = rb.getString("msg.username");  String password = rb.getString("msg.password");  System.*out*.println(username);  System.*out*.println(password);  }    @Test  **public** **void** fun2() {  ResourceBundle rb = ResourceBundle.*getBundle*("res", **new** Locale("en", "US" ));  String username = rb.getString("msg.username");  String password = rb.getString("msg.password");  System.*out*.println(username);  System.*out*.println(password);  }  } |

ResourceBundle的getBundle()方法需要两个参数：

* 第一个参数：配置文件的基本名称
* 第二个参数：Locale

getBundle()方法会通过两个参数来锁定配置文件！

###### 5　页面国际化



其实页面国际化也是同一道理，只需要通过切换Locale来切换配置文件。我们写一个MessageUtils类，内部需要ResourceBundle的实例。

我们再写一个过滤器MessageFilter，它会通过参数创建Locale对象，传递给MessageUtils，然后在页面中使用MessageUtils来获取文本信息。

MessageUtils.java

|  |
| --- |
| **public** **class** MessageUtils {  **private** **static** String *baseName* = "res";  **private** **static** Locale *locale*;  **public** **static** String getText(String key) {  **return** ResourceBundle.*getBundle*(*baseName*, *locale*).getString(key);  }  **public** **static** Locale getLocale() {  **return** *locale*;  }  **public** **static** **void** setLocale(Locale locale) {  MessageUtils.*locale* = locale;  }  } |

MessageFilter.java

|  |
| --- |
| **public** **class** MessageFilter **implements** Filter {  **public** **void** destroy() {  }  **public** **void** doFilter(ServletRequest request, ServletResponse response,  FilterChain chain) **throws** IOException, ServletException {  HttpServletRequest req = (HttpServletRequest) request;  String l = req.getParameter("request\_locale");  Locale locale = **null**;  **if**(l != **null** && !l.isEmpty()) {  String[] strs = l.split("\_");  locale = **new** Locale(strs[0], strs[1]);  req.getSession().setAttribute("WW\_TRANS\_I18N\_LOCALE", locale);  } **else** {  locale = (Locale)req.getSession().getAttribute("WW\_TRANS\_I18N\_LOCALE");  }  **if**(locale == **null**) {  locale = req.getLocale();  }  MessageUtils.*setLocale*(locale);  chain.doFilter(request, response);  }  **public** **void** init(FilterConfig fConfig) **throws** ServletException {  }  } |

login.jsp

|  |
| --- |
| <body>  <h1><%=MessageUtils.getText("msg.login") %></h1>  <form action=*"*<c:url value=*'/index.jsp'*/>*"* method=*"post"*>  <%=MessageUtils.getText("msg.username")%>：<input type=*"text"* name=*"username"*/><br/>  <%=MessageUtils.getText("msg.password")%>：<input type=*"password"* name=*"password"*/><br/>  <input type=*"submit"* value='<%=MessageUtils.getText("msg.login")%>'/>  </form>  </body> |

index.jsp

|  |
| --- |
| <body>  <p><%=MessageUtils.getText("hello") + ":" +request.getParameter("username") %>：</p>  </body> |

###### 6　NumberFormat

　　NumberFormat类用来对数字进行格式化，我们只需要使用String format(double)一个方法即可。下面是获取NumberFormat实例的方法：

* NumberFormat format = NumberFormat.getNumberFormat()
* NumberFormat format = NumberFormat.getNumberFormat(Locale)
* NumberFormat format = NumberFormat.getCurrentcyFormat()
* NumberFormat format = NumberFormat.getCurrentcyFormat(Locale)
* NumberFormat format = NumberFormat.getPercentFormat()
* NumberFormat format = NumberFormat.getPercentFormat(Locale)

###### 7　DateFormat

DateFormat类用来对日期进行格式化，我们只需要使用String format(Date)一个方法即可。下面是获取DateFormat实例的方法：

* DateFormat format = DateFormat.getDateFormat();
* DateFormat format = DateFormat.getTimeFormat();
* DateFormat format = DateFormat.getDateTimeFormat();
* DateFormat format = DateFormat.getDateFormat(int style, Locale locale);
* DateFormat format = DateFormat.getTimeFormat(int style, Locale locale);
* DateFormat format = DateFormat.getDateTimeFormat(int style, Locale locale);

其中style是对日期的长、中、短，以及完整样式。

* SHORT；
* MEDIUM；
* LONG；
* FULL

###### 8　MessageFormat

MessageFormat可以把模式中的{N}使用参数来替换。我们把{N}称之为点位符。其中点位符中的N是从0开始的整数。

MessageFormat.format(String pattern, Object… params)，其中pattern中可以包含0~n个点位符，而params表示对点位符的替换文本。注意，点位符需要从0开始。

String p = “{0}或{1}错误”;

String text = MessageFormat.format(p, “用户名”, “密码”);

System.out.println(text);//用户名或密码错误