过滤器：

API(接口)

执行流程

应用、案例：

（在servlet中重复编写的代码，都可以放到过滤器中完成！）

过滤器实现编码统一处理

登陆权限验证

目标：

监听器

国际化

# 监听器

## 概述

监听器： 主要是用来监听特定对象的创建或销毁、属性的变化的！

是一个实现特定接口的普通java类！

对象：

自己创建自己用 (不用监听)

别人创建自己用 （需要监听）

Servlet中哪些对象需要监听？域对象

request / session / servletContext 问题：request/session/servletContext的对象是谁创建的，实现类代码是谁写的？体会面向接口编程的意义

分别对应的是request监听器、session相关监听器、servletContext监听器

监听器(listener)

监听器接口：

一、监听对象创建/销毁的监听器接口

Interface ServletRequestListener 监听request对象的创建或销毁

Interface HttpSessionListener 监听session对象的创建或销毁

Interface ServletContextListener 监听servletContext对象的创建或销毁

二、监听对象属性的变化

Interface ServletRequestAttributeListener 监听request对象属性变化: 添加、移除、修改

Interface HttpSessionAttributeListener 监听session对象属性变化: 添加、移除、修改

Interface ServletContextAttributeListener 监听servletContext对象属性变化

三、session相关监听器

Interface HttpSessionBindingListener 监听对象绑定到session上的事件

Interface HttpSessionActivationListener(了解) 监听session序列化及反序列化的事件

## 生命周期监听器

声明周期监听器： 监听对象的创建、销毁的过程！

监听器开发步骤：

1. 写一个普通java类，实现相关接口；
2. 配置(web.xml)

### ServletRequestListener

监听request对象的创建或销毁。

代码：

|  |
| --- |
| /\*\*  \* 监听request对象的创建或销毁  \* **@author** zhanghong  \*  \*/  **public** **class** MyRequestListener **implements** ServletRequestListener{  // 对象销毁  @Override  **public** **void** requestDestroyed(ServletRequestEvent sre) {  // 获取request中存放的数据  Object obj = sre.getServletRequest().getAttribute("cn");  System.*out*.println(obj);  System.*out*.println("MyRequestListener.requestDestroyed()");  }  // 对象创建  @Override  **public** **void** requestInitialized(ServletRequestEvent sre) {  System.*out*.println("MyRequestListener.requestInitialized()");  }  } |
| **Web.xml**  <!-- 监听request对象创建、销毁 -->  <listener>  <listener-class>cn.itcast.a\_life.MyRequestListener</listener-class>  </listener> |
|  |

### HttpSessionListener

监听session对象的创建或销毁。

默认情况下关闭浏览器，session并没有销毁，而是不能用了，session销毁方式一种调用sessions.invalidate()方法，另外一种等待过期时间过去（默认30分钟）

### ServletContextListener

监听servletContext对象的创建或销毁。

一般这个用的比较多，在服务器启动的时候，加载一些资源，在服务器停止的时候销毁一些资源

## 属性监听器

监听:request/session/servletContext对象属性的变化！

### ServletRequestAttributeListener

### HttpSessionAttributeListener

### ServletContextAttributeListener

总结：先写类，实现接口； 再配置

## 1.4 其他监听器： session相关监听器

### HttpSessionBindingListener

监听对象绑定/解除绑定到sesison上的事件!

步骤：

对象实现接口； 再把对象绑定/解除绑定到session上就会触发监听代码。

作用：

(实际案例：可以监控用户系统登录，用户系统退出)

|  |
| --- |
| /\*\*  \* 演示监听此对象绑定到session上的过程，需要实现session特定接口  \* **@author** zhanghong  \*  \*/  **public** **class** User **implements** HttpSessionBindingListener{    **private** **int** id;  **private** String userName;  **private** String password;  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **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;  }    /\*\*  \* 绑定到Session  \*/  @Override  **public** **void** valueBound(HttpSessionBindingEvent event) {  System.***out***.println("用户登录系统了");  }  /\*\*  \* 从Session解绑  \*/  @Override  **public** **void** valueUnbound(HttpSessionBindingEvent event) {  System.***out***.println("用户退出系统了");  }  } |

思考：

这个session监听器，和上面的声明周期、属性监听器区别？

--🡪 不用再web.xml配置

-🡪 因为监听的对象是自己创建的对象，不是服务器对象！

## 案例

需求：做一个在线列表提醒的功能!

用户--🡪 登陆

---🡪 显示登陆信息，列表展示。(list.jsp)

--🡪 显示在线用户列表 (list.jsp)

-🡪 列表点击进入“在线列表页面” onlineuser.jsp

实现：

1. **先增加退出功能； 再把session活跃时间1min;**
2. **写监听器，监听servletContext对象的创建： 初始化集合(onlineuserlist)**
3. **登陆功能： 用户登陆时候，把数据保存到servletContext中**
4. **List.jsp 增加超链接， 点击时候提交直接跳转到online.jsp**
5. **写监听器： 监听session销毁，把当前登陆用户从onlineuserlist移除！**

# 国际化

Javaweb增强：过滤器、监听器、**国际化**、文件上传下载、javaMail

* 国际化又简称为 i18n：internationalization 就是i和n中间有18个字母

国际化的人：

人，英语，汉语； 可以说这个人是国际化的人；

软件的国际化：

软件

中国： 显示中文，以及服务符合中国习惯的文本字符串！

1999-09-09

美国： 显示英文，以及服务符合他国习惯的文本字符串！

这种软件，就叫国际化的软件！

如何做到国际化的软件，要求：

1. 软件中存储特定的字符串
2. 知道浏览器当前使用哪种语言（Locale ）

## Locale 本地化

Java提供了一个本地化的对象！封装当前语言、国家（政治、经济等）、环境等特征！

|  |
| --- |
| **public** **class** Demo1 {      @Test  //1. 本地化对象:Locale  // 封装语言、国家信息的对象，有java.util提供  **public** **void** testLocale() **throws** Exception {  // 模拟中国语言等环境  //Locale locale = Locale.CHINA;  Locale locale = Locale.*getDefault*(); // 当前系统默认的语言环境  System.***out***.println(locale.getCountry()); // CN 国家的简称  System.***out***.println(locale.getDisplayCountry()); // 国家名称  System.***out***.println(locale.getLanguage()); // zh 语言简称    // 模拟美国国家  Locale l\_us = Locale.***US***;  System.***out***.println(l\_us.getCountry());  System.***out***.println(l\_us.getDisplayCountry());  }  } |

## 国际化

### 静态数据国际化

网站中显示的固定文本的国际化： “用户名”“密码“

国际化的软件：

1. 存储所有国家显示的文本的字符串
   1. 文件: properties
   2. 命名： 基础名\_语言简称\_国家简称.properties

例如：msg\_zh\_CN.properties 存储所有中文

Msg\_en\_US.properties 存储所有英文

1. 程序中获取

ResourceBundle类，可以读取国际化的资源文件!

### 动态文本国际化

中文：1987-09-19 ￥1000

英文： Sep/09 1987 $100

* **数值，货币，时间，日期等数据**由于可能在程序运行时动态产生，所以无法像文字一样简单地将它们从应用程序中分离出来，而是需要特殊处理。Java 中提供了解决这些问题的 **API 类**(位于 java.util 包和 java.text 包中)

|  |
| --- |
| **public** **class** App {  // 国际化 - 静态数据  @Test  **public** **void** testI18N() **throws** Exception {    // 中文语言环境  Locale locale = Locale.***CHINA***;    // 创建工具类对象ResourceBundle  ResourceBundle bundle = ResourceBundle.*getBundle*("com.train.day12.i18n.statics.msg", locale);  // 根据key获取配置文件中的值  System.***out***.println(bundle.getString("hello"));  System.***out***.println(bundle.getString("username"));  System.***out***.println(bundle.getString("pwd"));  }    /\*\*  \* 国际化货币：NumberFormat.getCurrencyInstance(); 有用  \* 国家化数字：NumberFormat.getNumberInstance(); 没什么用  \* 国际化百分比：NumberFormat.getPercentInstance(); 没什么用  \* 国际化日期：DateFormat.getDateTimeInstance(dateStyle, timeStyle, aLocale) 有用  \*/      /\*\*  \* 国际化 - 动态文本 - 1. 国际化货币  \* 案例：100块人民币的国际化显示  \* **@throws** Exception  \*/  @Test  **public** **void** testI18N3() **throws** Exception {  // 模拟语言环境  Locale locale = Locale.***CHINA***;  // 数据准备  **double** number = 100;  // 工具类  NumberFormat nf = NumberFormat.*getCurrencyInstance*(locale);  // 国际化货币  String m = nf.format(number);  // 测试  System.***out***.println(m);  }    /\*\*  \* 案例用国际化工具类计算 $100 \* 10  \* **@throws** Exception  \*/  @Test  **public** **void** eg() **throws** Exception {  String str = "$100";  **int** num = 10;    // 1. 分析str值是哪一国家的货币  Locale us = Locale.***US***;  // 2. 国际化工具类  NumberFormat nf = NumberFormat.*getCurrencyInstance*(us);  // 3. 解析str国币  Number n = nf.parse(str);    System.***out***.println(n.intValue() \* num);  }    /\*\*  \* 国际化 - 动态文本 - 2. 国际化数值  \* **@throws** Exception  \*/  @Test  **public** **void** testI18N4() **throws** Exception {  // 模拟语言环境  NumberFormat nf = NumberFormat.*getNumberInstance*(Locale.***US***);  String str = nf.format(1000000000);  System.***out***.println(str);  }    /\*\*  \* 问题：把"100,000,000,000"转换成数字  \*  \*/    /\*\*  \* 国际化 - 动态文本 - 3. 国际化日期  \*  \* 日期  \* FULL 2015年3月4日 星期三  \* LONG 2015年3月4日  \* MEDIUM 2015-3-4  \* SHORT 15-3-4  \*  \* 时间  \* FULL 下午04时31分59秒 CST  \* LONG 下午04时32分37秒  \* MEDIUM 16:33:00  \* SHORT 下午4:33  \* **@throws** Exception  \*/  @Test  **public** **void** testI18N5() **throws** Exception {    // 日期格式  **int** dateStyle = DateFormat.***SHORT***;  // 时间格式  **int** timeStyle = DateFormat.***SHORT***;    // 工具类  DateFormat df =  DateFormat.*getDateTimeInstance*(dateStyle, timeStyle, Locale.***CHINA***);  String date = df.format(**new** Date());    System.***out***.println(date);  }    /\*\*  \* 案例： 将时间值：09-11-28 上午10时25分39秒 CST，反向解析成一个date对象。  \* **@throws** Exception  \*/  @Test  **public** **void** eg2() **throws** Exception {  String str = "09-11-28 上午10时25分39秒 CST";  // 创建DateFormat工具类，国际化日期  DateFormat df = DateFormat.*getDateTimeInstance*(  DateFormat.***SHORT***, DateFormat.***FULL***, Locale.*getDefault*());  Date d = df.parse(str);    System.***out***.println(d);  }  } |

### Jsp页面国际化

|  |
| --- |
| <html>  <head>  <%  ResourceBundle bundle = ResourceBundle.getBundle("com.train.day12.i18n.statics.msg",request.getLocale());  %>  <title><%=bundle.getString("title") %></title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  </head>    <body>  <form name=*"frmLogin"* action=*"*${pageContext.request.contextPath }*/admin?method=login"* method=*"post"*>  <table align=*"center"* border=*"1"*>  <tr>  <td><%=bundle.getString("username") %></td>  <td>  <input type=*"text"* name=*"userName"*>  </td>  </tr>  <tr>  <td><%=bundle.getString("pwd") %></td>  <td>  <input type=*"password"* name=*"pwd"*>  </td>  </tr>  <tr>  <td>  <input type=*"submit"* value=*"*<%=bundle.getString("submit") %>*"*>  </td>  </tr>  </table>  </form>  </body>  </html> |

### Jsp页面国际化 – 使用jstl标签

JSP2.0规范：不允许在jsp中出现java代码

JSTL标签：

**核心标签库**

**国际化与格式化标签库**

数据库标签库(没用)

函数库

<fmt:setLocale value=""/> 设置本地化对象

<fmt:setBundle basename=""/> 设置工具类

<fmt:message></fmt:message> 显示国际化文本

格式化数值

<fmt:formatNumber pattern="#.##" value="100.99"></fmt:formatNumber>

格式化日期:

<fmt:formatDate pattern="yyyy-MM-dd" value="${date}"/>

|  |
| --- |
| <html>  <head>  <!-- 一、设置本地化对象 -->  <fmt:setLocale value=*"*${pageContext.request.locale}*"*/>  <!-- 二、设置工具类 -->  <fmt:setBundle basename=*"com.train.day12.i18n.statics.msg"* var=*"bundle"*/>  <title><fmt:message key=*"title"* bundle=*"*${bundle}*"*></fmt:message></title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  </head>    <body>  <form name=*"frmLogin"* action=*"*${pageContext.request.contextPath }*/admin?method=login"* method=*"post"*>  <table align=*"center"* border=*"1"*>  <tr>  <td><fmt:message key=*"username"* bundle=*"*${bundle}*"*></fmt:message></td>  <td>  <input type=*"text"* name=*"userName"*>  </td>  </tr>  <tr>  <td><fmt:message key=*"pwd"* bundle=*"*${bundle}*"*></fmt:message></td>  <td>  <input type=*"password"* name=*"pwd"*>  </td>  </tr>  <tr>  <td>  <input type=*"submit"* value=*"*<fmt:message key=*"submit"* bundle="${bundle}"/>*"*>  </td>  </tr>  </table>  </form>  </body>  </html> |