JAVA SERVER PAGES

Ненко Табаков
Пламен Танов
Технологическо училище “Електронни системи”
Технически университет – София

30 ноември 2008

ЛИТЕРАТУРА
НЕОБХОДИМИ ПРОГРАМИ

- The Java EE 5 Tutorial - http://java.sun.com/javaee/5/docs/tutorial/doc/JavaEETutorial.pdf
- Java API документация - http://java.sun.com/javase/6/docs/api/
- Eclipse - www.eclipse.org
- Apache Tomcat - http://tomcat.apache.org/
- Step-by-step tutorial: http://www.java-tips.org/java-tutorials/tutorials/introduction-to-java-se
- Java Beans Tutorial: http://java.sun.com/docs/books/tutorial/javabeans/index.html

JAVA BEANS

- ***Java Beans*** компоненти са ***Java*** класове, които лесно могат да се преизползват и вграждат(композират) в приложения
- Всеки ***Java*** клас, който следва определени конвенции е ***Java Bean*** компонент
- ***JSP*** технологията поддържа работа с ***Java Beans*** компоненти чрез стандартни елементи на технологията – не трябва да се инсталират допълнителни библиотеки

JAVA BEANS

- Един ***Java Bean*** компонент се състои от свойства
- Свойство се нарича атрибут на един ***Java Bean*** компонент, който може да определя неговото поведение или представяне
- Свойствата могат да бъдат:
  - Прости (***simple***) – свойство с една стойност
  - Индексирани (***indexed***)– свойство с няколко стойности
  - Свойства, които известяват за промяната си (***bound***)
  - Свойства, които при промяна се валидират от друг ***Java Bean*** компонент (***constrained***)

JAVA BEANS

Друга класификация на ***Java Bean*** компоненти може е:

- За четене/писане
- Само за четене
- Само за писане

Едно свойство не е задължително да бъде имплементирано като поле. Достатъчно е да може да бъде достигано чрез публични методи и да спазва ***Java Bean*** конвецията

ПРОСТИ СВОЙСТВА
SIMPLE PROPERTIES

За добавяне на такива свойства в един ***Java Bean*** компонент трябва да са спазени следните изисквания:

- За всяко свойство, което може да бъде четено трябва да има ***getXXX*** метод
- За всяко свойство, в което може да бъде писано трябва да има ***setXXX*** метод
- Трябва да има дефиниран конструктор, който е без параметри
- Ако свойството е от тип ***boolean,*** то тогава вместо ***getXXX*** метод трябва да има ***isXXX*** метод

ПРИМЕР *SIMPLE PROPERTIES*

```
public  class ForumPost {
    private int id;


    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
```

ИНДЕКСИРАНИ СВОЙСТВА *INDEXED PROPERTIES*

Индексирано свойство е масив от свойства или обекти, които поддържат множество от стойности.
Такива свойства трябва да спазват следните изисквания:

- Методите за достигане на една стойност трябва да имат име започващо с ***get*** и един параметър, който е ***int*** – индексът на стойността, която искаме да достигнем
- Методите за записване на една стойност трябва да започват с ***set*** и да имат два параметъра – индексът, където да запишем стойността и съответната стойност
- За работа със самото свойство се използват същите правила като при прости свойства

ПРИМЕР *INDEXED PROPERTIES*

```
public  class ForumDataBean {
   private ForumPost[] posts;

   public ForumPost[] getPosts () {
      return posts;
   }
   public void setPosts (ForumPost newPosts[]) {
      posts = newPosts;
   }
   public ForumPost getPosts (int index) {
      return posts[index];
   }
   public void setPosts (int index, ForumPost post) {
      posts[index] = post;
   }
}
```

EXPRESSION LANGUAGE (EL)

- ***EL*** позволява да се използват прости изрази за динамично четене на данни от ***JavaBean*** компоненти
- С ***EL*** може динамично да се достигат данни от ***JavaBean*** компоненти, различни структури от данни и обекти съдържащи служебна информация
- С ***EL*** може динамично да се записват данни в ***JavaBean*** компоненти
- С ***EL*** може да се извикват статични и публични методи
- С ***EL*** може да се извършват аритметични операции, чийто резултат се изчислява динамично
- ***EL*** може да се разширява, така че да поддържа изрази, които не са част от езика

EL – НЕПОСРЕДСТВЕНО ИЗЧИСЛЯВАНЕ НА ИЗРАЗИ

- Непосредствено изчисляване (***immediate evaluation***) означава, че ***JSP*** процесорът изчислява израза и връща стойността му веднага след като страницата е показана
- Такива изрази използват ***${}*** синтаксис
- Такива изрази могат да се използват само в текстови шаблони или като стойност в ***JSP*** таг, който може да приема изрази по време на изпълнение

```
<fmt:formatNumber value="${sessionScope.cart.total}"/>
```

EL – ПРЕДИЗВИКАНО ИЗЧИСЛЯВАНЕ НА ИЗРАЗИ

- Предизвикано изчисляване (***deffered evaluation***) означава, че технологията използваща ***EL*** може да използва свой собствен механизъм за изчисляване на израза по някое време по – късно в жизнения цикъл на страницата
- Такива изрази използват *#{}* синтаксис

```
<h:inputText id="name" value="#{customer.name}" />
```

EL ИЗРАЗИ

- ***EL*** дефинира два вида изрази – изрази за стойности (***value*** expression) и изрази за методи (***method expression***)
- Изразите за стойности могат да прочетат/запишат стойност
- Изразите за методи съдържат препратки към методи, които могат да бъдат извикани и от своя страна да върнат стойност
- При дефинирането на таг се указва какъв вид израз се очаква за атрибутите му

VALUE EXPRESSIONS

Изразите за стойности могат да се обръщат към следните типове обекти и към техните свойства:

- ***JavaBeans*** компоненти
- Колекции
- Изброими типове (***enumeration***)
- Обекти съдържащи служебна информация

```
${customer}
```

VALUE EXPRESSIONS

- Когато се направи обръщение към обект чрез израз за стойност, WEB контейнерът започва търсенето на обекта
- Търсенето става на база на неговото име
- Търсенето се извършва в рамките на страницата, запитването, сесията и приложението
- В случая WEB контейнерът ще търси името ***customer***
- Ако нищо не е намерено се връща ***null***

```
${customer}
```

VALUE EXPRESSIONS

- За обръщение към свойство на някой обект се използва „.“ или „[]“
- Ако искаме да прочетем свойството ***name*** на ***JavaBean*** компонента ***customer*** може да използваме следните изрази, като и

```
${customer.name}
${customer[“name”]}
```

- Двете нотации може се смесват

```
${customer.name[“address”]}
```

КЪДЕ МОЖЕ ДА СЕ ИЗПОЛЗВАТ *VALUE EXPRESSIONS*

- В статичен текст:

```
<someTag>
   Some text ${expr} is inserted
</someTag>
```

- Във всеки атрибут на таг, който може да използва израз:

```
<jsp:setProperty name="data" property="len" value="${expr}" />
<jsp:setProperty name="data" property="len" value="#{expr}" />
```

КЪДЕ МОЖЕ ДА СЕ ИЗПОЛЗВАТ *VALUE EXPRESSIONS*

- Няколко израза могат да се напишат един след друг:

```
<jsp:setProperty name="data" property="t" value="${e}${e2}" />
<jsp:setProperty name="data" property="t" value="#{e}#{e2}" />
```

- Такива изрази се наричат ***composite expressions***
- Те се изчисляват от ляво на дясно
- Не може да се смесва $ и # в едно поле:

```
<jsp:setProperty name="data" property="l" value="${e}#{e2}" />
```

METHOD EXPRESSIONS

- Такива изрази се използват за извикване на публичен метод на ***Java Bean*** компонент
- Един метод може да се извика както с оператор „.“, така и с „[]“
- Ползват се единствено с предизвикано изчисляване на изрази (#{ })

```
<h:inputText value="#{team.name}"
                              validator="#{team.validateName}"/>
<h:inputText value="#{team.name}"
                            validator="#{team[“validateName”]}"/>
```

КЪДЕ МОЖЕ ДА СЕ ИЗПОЛЗВАТ *METHOD EXPRESSIONS*

- Могат да се използват само в атрибути на тагове
- Могат да бъдат само един израз

```
<someTag value="#{object.method}"/>
<someTag value="#{object["method"]}"/>
```

ОПЕРАТОРИ

Освен „.“ и „[]“ ***EL*** поддържа и други оператори:

- Аритметични: +, -, *, /, ***div***, %, ***mod***
- Логически: ***and,*** &&, ***or,*** ||, ***not***, !
- Релационни: ==, ***eq***, !=, ***ne***, <, ***lt***, >, ***gt***, <=, ***ge***, >=, ***le***
- ***empty*** – проверява даден израз е ***null*** или не съдържа стойност (празен е)
- Условни: A ? B : C

ЗАПАЗЕНИ ДУМИ

and, or, not, eq, ne, lt, gt, le, ge, true, false, null, instanceof, empty, div, mod

ПРИМЕРИ

```
${1 > (4/2)}
${4.0 > 3}
${100 == 100.0}
${(10*10) ne 100}
${'a' < 'b'}
${'hit' gt 'hip'}
${3 div 4}
${10 mod 4}
${!empty param.Add}
```

ПРИМЕРИ

```
${1 > (4/2)}                    false
${4.0 > 3}                      true
${100 == 100.0}                 true
${(10*10) ne 100}               false
${'a' < 'b'}                    true
${'hit' gt 'hip'}               true
${3 div 4}                      0,75
${10 mod 4}                     2
${!empty param.Add}             зависи от параметъра Add
```

ФУНКЦИИ

В ***EL*** могат да се дефинират функции, като характерно за тях е:

- Функциите са като статични методи
- Функциите се идентифицират статично по време на транслация
- Параметрите и извикванията се правят като част от ***EL*** израз
- Функциите могат да се извикват както в статичен текст, така и в атрибутите на тагове

ФУНКЦИИ

- За използване на функция в ***JSP*** страница се използва директивата ***taglib***
- С нея се указва, коя библиотека, съдържаща функциите да се включи

```
<%@ taglib prefix="f" uri="/WEB-INF/functions.tld" %>
// или:
<%@ taglib prefix="f" uri="/FunctionLibrary" %>
```

- С горния ред са включени функциите дефинирани в ***functions***
- Те могат да се използват чрез префикса дефиниран в директивата и името на функцията

```
${f:equals(selectedLocaleString, localeString)}
```

ДЕФИНИРАНЕ НА ФУНКЦИИ

- За дефиниране на функция е нужно да се направи публичен статичен метод в публичен клас
- После трябва да се свърже името на функцията с ***EL*** израза
- Такова свързване става чрез ***XML*** декларации във файл с разширение ***tld (Tag Library Definition)***

ДЕФИНИРАНЕ НА ФУНКЦИИ

```
package  jsp.examples;

public  class MyLocales {

   public static boolean equals( String l1, String l2 ) {
       return l1.equals(l2);
   }

}
```

ДЕФИНИРАНЕ НА ФУНКЦИИ

```
//functions.tld
<?xml version="1.0" encoding="UTF-8" ?>

<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee web-
                        jsptaglibrary_2_0.xsd" version="2.0">
    <description>
         A tag library that defines a function.
    </description>
    <tlib-version>1.0</tlib-version>
    <short-name>FunctionTagLibrary</short-name>
    <uri>/FunctionLibrary</uri>
  <function>
    <name>equals</name>
    <function-class>mypkg.MyLocales</function-class>
    <function-signature>boolean equals( java.lang.String,
                        java.lang.String )</function-signature>
  </function>
</taglib>
```

ИЗПОЛЗВАНЕ НА *JAVA BEANS* КОМПОНЕНТИ

За декларация на ***Java Bean*** компонента, в една ***JSP*** страница се използва елементът ***<jsp:useBean>***

```
<jsp:useBean id="beanName"
    class="fully-qualified-classname" scope="scope"/>



<jsp:useBean id="beanName"
    class="fully-qualified-classname" scope="scope">
    <jsp:setProperty .../>
</jsp:useBean>
```

ИЗПОЛЗВАНЕ НА *JAVA BEANS* КОМПОНЕНТИ

id – името, което идентифицира компонента
class **–** пълно име на класа
scope – в рамките на кой обект този компонент е видим (живее)

```
<jsp:useBean id="locales" scope="application"
    class="mypkg.MyLocales"/>
```

ИЗПОЛЗВАНЕ НА *JAVA BEANS* КОМПОНЕНТИ

Стандартният начин за записване на стойност в ***Java Bean*** компонент е с елемента ***<jsp:setProperty>***

```
<jsp:setProperty name="beanName"
   property="propName" value="string-constant"/>

<jsp:setProperty name="beanName"
   property="propName" param="paramName"/>
```

JSTL

- ***JSTL – JavaServerPages Standard Tag Library***
- ***JSTL*** съдържа основна функционалност, която е обща са всички ***JSP*** приложения
- ***JSTL*** има тагове за итериране, условия, управление на ***XML*** документи, достъп до база данни чрез ***SQL*** и често използвани функции
- Макар, че ***JSTL*** има функционалност за достъп до данни – ***SQL, XML***, то тези функции са добавени само за изготвяне на прототип на приложение (определяне на функционалност и т.н.), а не за реално приложение
- За реално приложение по – добрият подход е да се раздели логиката от представянето – ***JavaBeans*** компоненти или други реализации

JSTL ТАГОВЕ

Име	*URI*	Префикс
Core	http://java.sun.com/jsp/jstl/core	***c***
XML	http://java.sun.com/jsp/jstl/xml	
Internationalization		
SQL		
Functions		

JSTL УПОТРЕБА

Тагове от ***JSTL*** се включват с ***taglib*** директивата

```
<%@ taglib prefix="tt" [tagdir=/WEB-INF/tags/dir | uri=URI ]
%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"
%>
```

JSTL УПОТРЕБА

Core Tag Library

Операции с променливи	***remove***
	set
Условия	***choose*** ***when*** ***otherwise***
	forEach
	forTokens
	if

JSTL УПОТРЕБА

Core Tag Library

Операции с ***URL***	***import*** ***param***
	redirect ***param***
	url ***param***
Други	***catch***

JSTL – ОПЕРАЦИИ С ПРОМЕНЛИВИ

- Тагът ***set*** задава стойност на дадена ***EL*** променлива
- Ако променливата не съществува, тя се създава

```
<c:set var="bookId" value=#{param.Remove}/>
```

- Тагът ***remove*** изтрива дадена ***EL*** променлива

```
<c:remove var="bookId" scope=”session”/>
```

JSTL – ТАГОВЕ ЗА УСЛОВИЯ

- Тагът ***if*** позволява изпълнение на неговото тяло, ако дадено условие е изпълнено

```
<c:if test="${!emptyparam.Remove}"/>
   …
</c:if>
```

```
<c:choose>
   <c:when test="${!emptyparam.Remove}">
      …
   </c:when>
   <c:when test="${emptyparam.Remove}">
      …
   </c:when>
   <c:otherwise>
   </c:otherwise>
</c:choose>
```

JSTL – ТАГОВЕ ЗА ИТЕРИРАНЕ

- Тагът ***forEach*** позволява да се итерира над дадена колекция
- Атрибутът ***items*** определя колекцията
- Атрибутът ***var*** дава достъп до текущия елемент от колекцията по време на итерирането

```
<c:forEach var="currentElement" items="${collection.items}">
   ...
      <tr><td>${currentElement.name}</td></tr>
   ...
</c:forEach>
```