
JAVA FOR STUDENTS
ФИЛИАЛ КАФЕДРЫ ПОИТ БГУИР В EPAM SYSTEMS
КУРС: ВЕБ-ТЕХНОЛОГИИ (JAVA)
SERVLETS FUNDAMENTALS
Olga Smolyakova , PhD
Oracle Certified Java 6 Programmer
Olga_Smolyakova@epam.com

Содержание

1. **Интернационализация**
2. **ResourceBundle**
3. **JSTL: fmt tags**
4. **Пользовательские теги**

ИНТЕРНАЦИОНАЛИЗАЦИЯ

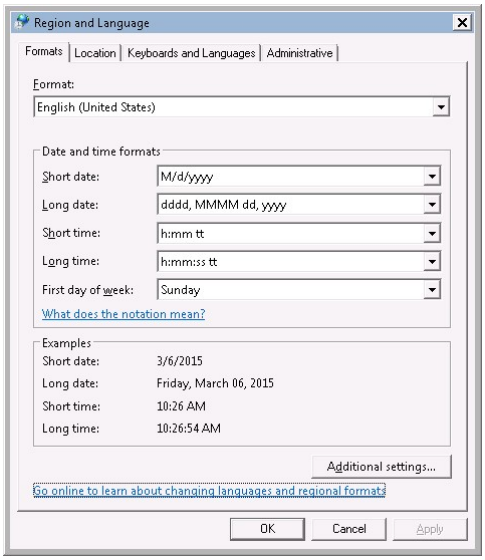
Определение

Интернационализация программы (i18n) –

Написание программы, работающей в различных языковых окружениях.

Локализация программы (l10n) –

Адаптация интернационализированной программы к конкретным языковым окружениям.



Локализация

Пакеты

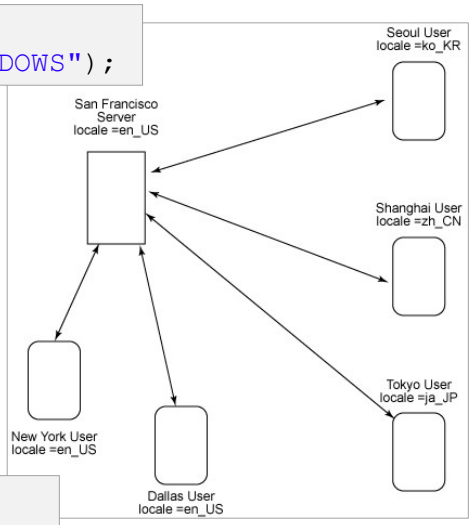
- java.util
- java.text

Пример	
en_UK_windows	**en_UK_unix**
choose the folder containing colour information	choose the directory containing colour information
en_US	**ru_RU_unix**
choose the folder containing color information	Выберите каталог, содержащий цветовую информацию

Класс Locale

```
Locale l =
new Locale("en","US","WINDOWS");
```

Класс **Locale**, (пакет java. util) идентифицирует используемое языковое окружение.




```
Locale l =
      new Locale("ru","RU");
```

Класс Locale

Локаль определяется:

1) константами: **Locale.US, Locale.FRANCE**

```
Locale locale1 = Locale.CANADA;
Locale locale2 = Locale.CANADA_FRENCH;
```

2) конструкторами класса **Locale**

- Locale(language) – по языку
- Locale(language, country) – по языку и стране
- Locale(language, country, variant) – по языку стране и варианту

```
Locale l = new Locale("ru","RU");
Locale l = new Locale("en","US","WINDOWS");
```

<epam>

Методы класса Locale

getDefault() – возвращает текущую локаль, сконструированную на основе настроек операционной системы

getLanguage() – код языка региона

getDisplayLanguage() – название языка

getCountry() – код региона

getDisplayCountry() – название региона

getAvailableLocales() – список доступных локалей

1

```
Locale defaultLocale = Locale.getDefault();
Locale frLocale = new Locale("fr", "FR");

S….o….p…(defaultLocale.getDisplayCountry());
S….o….p…(defaultLocale.getDisplayCountry(Locale.FRENCH));
S….o….p…(frLocale.getDisplayCountry(defaultLocale));
```

Результат:

```
Россия
Russie
Франция
```

2

```
S….o….p…(usLocale.getDisplayName());
S….o….p…(usLocale.getDisplayName(frLocale));
S….o….p…(rusLocale.getDisplayName(frLocale));
```

Результат:

```
английский (Соединенные Штаты)
anglais (Etats-Unis)
russe (Russie)
```

Интернационализация чисел и дат

Пакет
- java.text

Класс **NumberFormat**

- getNumberInstance(locale) – обычные числа;
- getIntegerInstance(locale) – целые числа (с округлением);
- getPercentInstance(locale) – проценты;
- getCurrencyInstance(locale) – валюта.

Класс **DateFormat**

- getDateInstance([dateStyle[, locale]]) – даты;
- getTimeIntance([timeStyle[, locale]]) – времени;
- getDateTimeIntance([dateStyle, timeStyle, [locale]]) – даты и времени.

Пакет java.text содержит и другие форматировщики данных.

Класс **NumberFormat**

Методы :

1. String format(long) – форматировать целое число;
2. String format(double) – форматировать число с плавающей точкой;
3. Number parse(String) – разобрать локализованное число. *ParseException*

```
NumberFormat nf = NumberFormat.getInstance(Locale.US);

System.out.println(100 + "\t" + nf.format(100));
System.out.println(1000 + "\t" + nf.format(1000));
System.out.println(10000 + "\t" + nf.format(10000));
System.out.println(1000000 + "\t" + nf.format(1000000));
```

Результат:

```
100             100
1000            1,000
10000           10,000
1000000         1,000,000
```

```
double number = 9876.598;
NumberFormat nfGer =
      NumberFormat.getNumberInstance(Locale.GERMANY);
NumberFormat nfJap =
      NumberFormat.getNumberInstance(Locale.JAPANESE);
NumberFormat nfDef =
      NumberFormat.getNumberInstance(Locale.FRANCE);

S….o….p…(": " + nfGer.format(number));
S….o….p…(": " + nfJap.format(number));
S….o….p…(": " + nfDef.format(number));
```

Результат:

```
: 9.876,598
: 9,876.598
: 9 876,598
```

```
String numGer = "9.876,598";
String curGer = "9.876,60 €";
NumberFormat nfGer =
       NumberFormat.getNumberInstance(Locale.GERMANY);
NumberFormat cfGer =
       NumberFormat.getCurrencyInstance(Locale.GERMANY);

double dGer = (Double) nfGer.parse(numGer);
double dcGer = (Double) cfGer.parse(curGer);

System.out.println(dGer + " " + dcGer);
```

Результат:

```
9876.598 9876.6
```

Класс **DateFormat**

Стили

- DEFAULT, FULL, LONG, MEDIUM, SHORT

Методы форматирования

- String format(date) – форматировать дату/время
- Date parse(String) – разобрать локализованную дату/время

ParseException

```
Date date = new Date();
DateFormat dfUSLong =
    DateFormat.getDateInstance(DateFormat.LONG, Locale.US);
DateFormat dfUSShort =
    DateFormat.getDateInstance(DateFormat.SHORT, Locale.US);

println(dfUSLong.format(date));
println(dfUSShort.format(date));
```

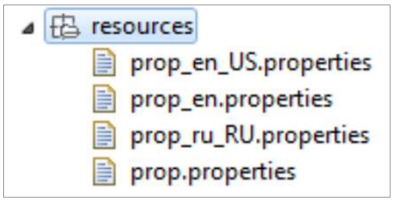
Результат:

```
March 1, 2014
3/1/14
```

<epam>

RESOURCEBUNDLE

Локализация



Управление набором ресурсов производится классом **ResourceBundle**, находящимся в пакете **java.util**.

prop.properties

```
1  prop.key1 = value 01 default
2  prop.key2 = value 02 default
```

prop_en_US.properties

```
1  prop.key1 = value 01 en US
2  prop.key2 = value 02 en US
```

prop_en.properties

```
1  prop.key1 = value 01 en
2  prop.key2 = value 02 en
```

prop_ru_RU.properties

```
1  prop.key1 = \u0437\u043D\u0430\u0447\u0435\u043D\u0438\u0439 01 ru RU
2  prop.key2 = \u0437\u043D\u0430\u0447\u0435\u043D\u0438\u0435 02 ru RU
```

Для корректного отображения нелатинских символов ознакомьтесь с работой утилиты native2ascii.

```
ResourceBundle bundle;
String key = "prop.key1";
1 bundle = ResourceBundle.getBundle("resources.prop",
        new Locale("ru", "RU")); S….o….p…(bundle.getString(key));

2 bundle = ResourceBundle.getBundle("resources.prop",
        new Locale("ru")); S….o….p…(bundle.getString(key));

3 bundle = ResourceBundle.getBundle("resources.prop",
        new Locale("ru", "BY")); S….o….p…(bundle.getString(key));

4 bundle = ResourceBundle.getBundle("resources.prop",
        new Locale("en", "UK")); S….o….p…bundle.getString(key));

5 bundle = ResourceBundle.getBundle("resources.prop",
        new Locale("fr", "FR")); S….o….p… (bundle.getString(key));
```

значений 01 ru RU

value 01 en US

value 01 en US

value 01 en

value 01 en US

ListResourceBundle

```
public class AppResources extends ListResourceBundle{
       public Object[][] getContents() {
           return new Object[][] {
                     { "prop.key1", "value01" },
                     { "prop.key2", "value02" },
           };
       }
}
```

```
ResourceBundle bundle;
String key = "prop.key1";
bundle = ResourceBundle.getBundle("AppResources");
System.out.println(bundle.getString(key));
```

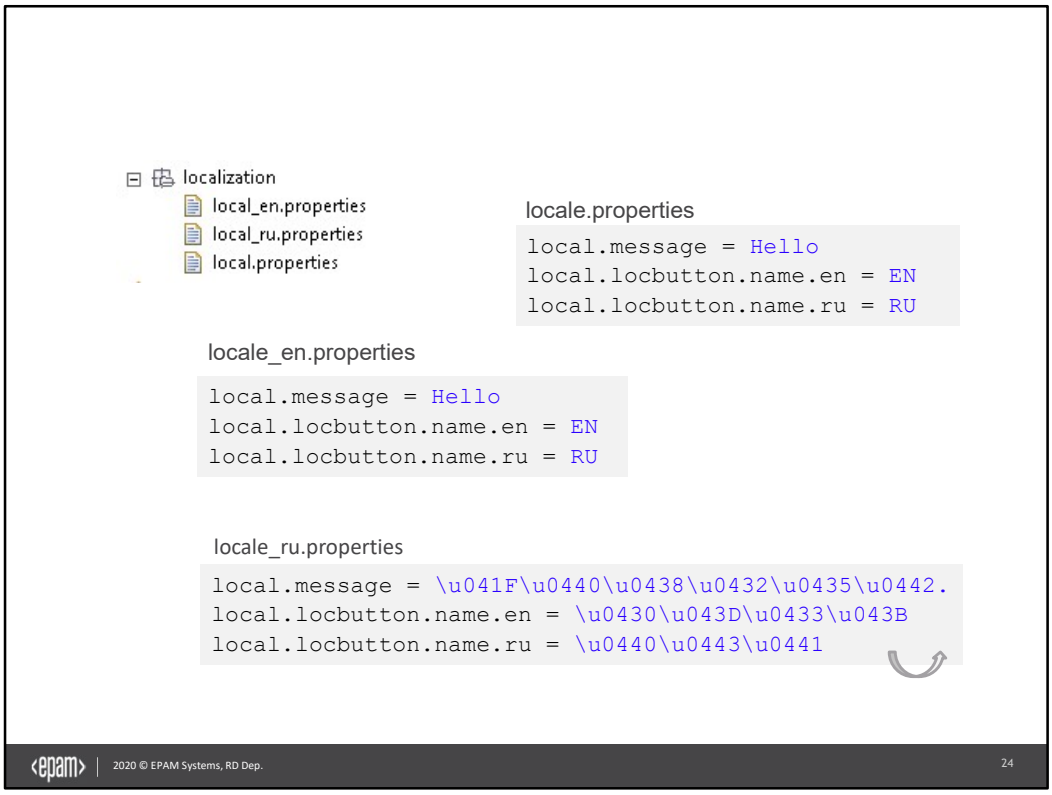
```
value01
```

<epam>

JSTL: FMT TAGS

Tag	Описание
<fmt:formatNumber>	Формирует целочисленное значение с определенной точностью или форматом
<fmt:parseNumber>	Разбирает строковое представление числа, валюты или процентов
<fmt:formatDate>	Форматирует дату и/или время, используя определенные стили и шаблоны
<fmt:parseDate>	Разбирает строковое представление даты/времени
<fmt:bundle>	Загружает ресурс-bundle, который будет использовать телом тега

Tag	Описание
<fmt:setLocale>	Сохраняет указанную локаль в конфигурационной переменной
<fmt:setBundle>	Загружает ресурс-bundle и хранит его в именованной переменной в контексте или в конфигурационной переменной
<fmt:timeZone>	Определяет часовой пояс для любого времени, форматируя и разбирая код в своем теле.
<fmt:setTimeZone>	Сохраняет часовой пояс в конфигурационной переменной
<fmt:message>	Отображает интернационализованное сообщение
<fmt:requestEncoding>	Устанавливает кодировку запроса.

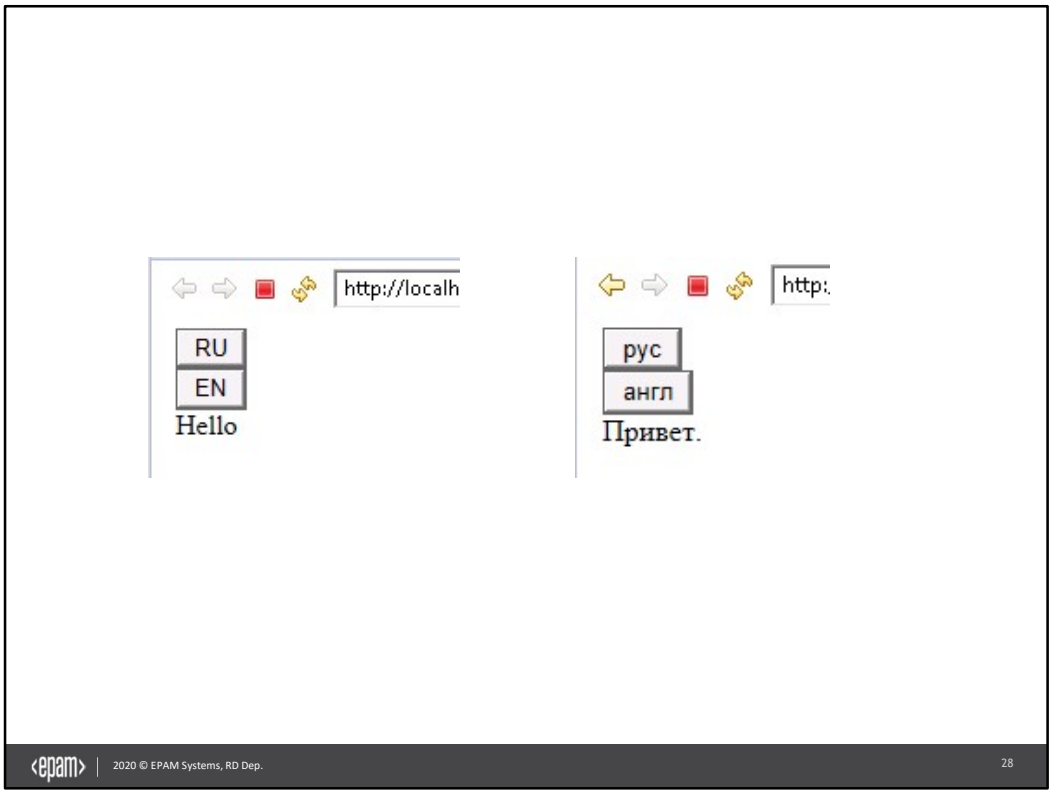

localization
local_en.properties
local_ru.properties
local.properties
locale.properties
local.message = Hello
local.locbutton.name.en = EN
local.locbutton.name.ru = RU
locale_en.properties
local.message = Hello
local.locbutton.name.en = EN
local.locbutton.name.ru = RU
locale_ru.properties
local.message = \u041F\u0440\u0438\u0432\u0435\u0442.
local.locbutton.name.en = \u0430\u043D\u0433\u043B
local.locbutton.name.ru = \u0440\u0443\u0441
2020 © EPAM Systems, RD Dep.
24

```
public class Controller extends HttpServlet {
        private static final long serialVersionUID = 1L;

        protected void doPost(HttpServletRequest request,
                HttpServletResponse response)
                        throws ServletException, IOException {
           request.getSession(true).setAttribute("local",
                        request.getParameter("local"));
           request.getRequestDispatcher("index.jsp").
                        forward(request, response);
        }

}
```

index.jsp

```
…
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt"
prefix="fmt"%>
…
<fmt:setLocale value="${sessionScope.local}" />
<fmt:setBundle basename="localization.local" var="loc" />
<fmt:message bundle="${loc}" key="local.message"
var="message" />
<fmt:message bundle="${loc}" key="local.locbutton.name.ru"
        var="ru_button" />
<fmt:message bundle="${loc}" key="local.locbutton.name.en"
        var="en_button" />
…
```

```
…
<body>
        <form action="Controller" method="post">
                <input type="hidden" name="local" value="ru" />
                <input type="submit" value="${ru_button}" /><br />
        </form>

        <form action="Controller" method="post">
                <input type="hidden" name="local" value="en" />
                <input type="submit" value="${en_button}" /><br />
        </form>

        <c:out value="${message}" />
</body>
…
```


http://localh
RU
EN
Hello
http:
рус
англ
Привет.
2020 © EPAM Systems, RD Dep.
28

ПОЛЬЗОВАТЕЛЬСКИЕ ТЕГИ

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Insert title here</title>
</head>
<body>

        <form action="Controller" method="post">
                <input type="submit" value="tap me" /><br />
        </form>

</body>
</html>
```

```
package _java._ee._02._jspbean;

import …;

public class JSPSetBean  {
         private Iterator it;
         private Set set;

         public JSPSetBean(Set set){
                  this.set = set;
         }

         public String getSize(){
                  it = set.iterator();
                  return Integer.toString(set.size());
         }

         public String getElement(){
                  return it.next().toString();
         }
}
```

```
package _java._ee._02._servlet;
import …;

public class Controller extends HttpServlet {
        protected void doGet(HttpServletRequest request, …) …{
                doProcess(request, response);      }

        protected void doPost(HttpServletRequest request, …) …{
                doProcess(request, response);      }

        private void doProcess(HttpServletRequest request, …) …{
                Set<String> set = new HashSet<String>();
                set.add("one");  set.add("two");  set.add("three");
                JSPSetBean jsp = new JSPSetBean(set);

                request.setAttribute("userbean", jsp);

                request.getRequestDispatcher("main.jsp").
                        forward(request, response);
        }
}
```

```
<?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_2_5.xsd" id="WebApp_ID"
version="2.5">
…

  <jsp-config>
        <taglib>
                <taglib-uri>
                        /WEB-INF/tld/taglib.tld
                </taglib-uri>

                <taglib-location>
                        /WEB-INF/tld/taglib.tld
                </taglib-location>
        </taglib>
</jsp-config>

</web-app>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<taglib xmlns="http://java.sun.com/JSP/TagLibraryDescriptor"
        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"><!--дескриптор библиотеки тегов -->
        <tlib-version> 1.0 </tlib-version>
        <short-name> mytag </short-name>
        <uri> /WEB-INF/tld/taglib.tld </uri>
        <tag>
          <name> jspset </name>
          <tag-class> _java._ee._02._jsptag.SpecialJSPTag </tag-class>
          <body-content> empty </body-content>
          <attribute>
                <name> set </name>
                <required> false </required>
                <rtexprvalue> true </rtexprvalue>
          </attribute>

        </tag>
```

```
        <tag>
          <name> bodyjspset </name>
          <tag-class> _java._ee._02._jsptag.JSPTagWithBody </tag-class>
          <body-content> JSP </body-content>
          <attribute>
                <name> num </name>
                <required> false </required>
                <rtexprvalue> true </rtexprvalue>
          </attribute>
        </tag>

</taglib>
```

```
<%@ page language="java" contentType="text/html; charset=utf-8"
        pageEncoding="utf-8"%>
<%@ taglib uri="/WEB-INF/tld/taglib.tld" prefix="mytag"%>

…
<body>

        <jsp:useBean id="userbean"
                        class="_java._ee._02._jspbean.JSPSetBean"
                        scope="request" />

        <mytag:jspset set="${userbean}" />

<br/>

        <mytag:bodyjspset num="${userbean.size}">
                ${userbean.element}
        </mytag:bodyjspset>

</body>
</html>
```

```java
package _java._ee._02._jsptag;

import …;

public class SpecialJSPTag extends TagSupport  {
        private JSPSetBean set;

        public JSPSetBean getSet() {
                return set;
        }

        public void setSet(JSPSetBean set) {
                this.set = set;
        }
```

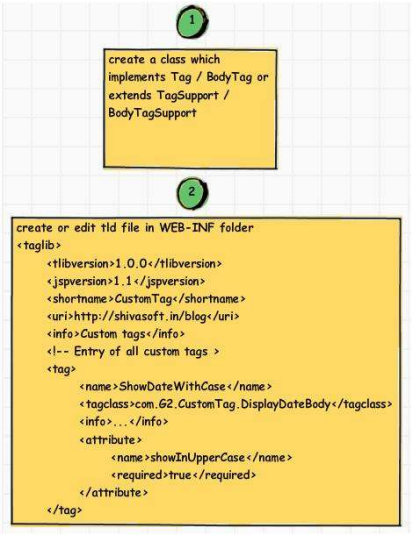
```
        @Override
        public int doStartTag() throws JspException {
                int size = new Integer(set.getSize());
                String str = "Size = <b>(" + size + ")</b>";
                try{
                        JspWriter out = pageContext.getOut();
                        out.write(str);

                        out.write("<table border=\"1\">");
                        for(int i=0; i<size; i++){
                                out.write("<tr><td>");
                                out.write(set.getElement());
                                out.write("</td></tr>");
                        }
                        out.write("</table>");

                }catch(IOException e){
                        throw new JspException(e.getMessage());
                }
                return SKIP_BODY;
        }
}
```

```
package _java._ee._02._jsptag;
import …;

public class JSPTagWithBody extends BodyTagSupport {
        private int num;

        public void setNum(String num) {
                this.num = new Integer(num);
        }

        public int doStartTag() throws JspTagException {
                try {
                        pageContext.getOut().
                                write("<TABLE BORDER=\"3\"
                                        WIDTH=\"100%\">");
                        pageContext.getOut().write("<TR><TD>");
                } catch (IOException e) {
                        throw new JspTagException(e.getMessage());
                }
                return EVAL_BODY_INCLUDE;
        }
```

```
public int doAfterBody() throws JspTagException {
        if (num > 1) {
          num = num - 1;
          try {
            pageContext.getOut().
                write("</TD></TR><TR><TD>");
          } catch (IOException e) {
                throw new JspTagException(e.getMessage());
          }
          return EVAL_BODY_AGAIN;
        } else {
                return SKIP_BODY;
        }
}
```

```
public int doEndTag() throws JspTagException {
                try {
                        pageContext.getOut().write("</TD></TR>");
                        pageContext.getOut().write("</TABLE>");
                } catch (IOException e) {
                        throw new JspTagException(e.getMessage());
                }
                return SKIP_BODY;
        }
}
```

Для создания пользовательских тегов необходимо определить класс обработчика тега, определяющий его поведение, а также дескрипторный файл библиотеки тегов (файл **.tld), в которой описываются один или несколько тегов, устанавливающих** соответствия между именами XML-элементов и реализацией тегов.



При определении нового тега создается класс Java, который должен реализовывать интерфейс

javax.servlet.jsp.tagext.Tag.

Обычно создается класс, который наследует один из классов **TagSupport** или **BodyTagSupport** (для тегов без тела и с телом соответственно).

Указанные классы реализуют интерфейс **Tag** и содержат стандартные методы, необходимые для базовых тегов.

Класс для тега должен также импортировать классы из пакетов **javax.servlet.jsp** и, если необходима передача информации в поток вывода, то **java.io** или другие классы.

Если в теге отсутствует тело, метод **doStartTag()** должен возвратить константу **SKIP_BODY**, дающую указание системе игнорировать любое содержимое между начальными и конечными элементами создаваемого тега.

Чтобы сгенерировать вывод, следует использовать метод **write()** класса **JspWriter**, который выводит на страницу содержимое объекта **str**.

Объект **pageContext** класса **PageContext** – это атрибут класса, унаследованный от класса **TagSupport**, обладающий доступом ко всей области имен, ассоциированной со страницей JSP.

С помощью методов класса **PageContext** можно получить:

- **getRequest()** – объект запроса;
- **getResponse()** – объект ответа;
- **getOut()** – поток **JspWriter**
- **getServletContext()** – объект контекста сервлета;
- **getServletConfig()** – объект конфигурации сервлета;
- **getSession()** – объект сессии;
- **ErrorData getErrorData()** – информацию об ошибках.

Следующей задачей после создания класса обработчика тега является идентификация этого класса для сервера и связывание его с именем XML-тега. Эта задача выполняется в формате XML с помощью дескрипторного файла библиотеки тегов .tld.

Файл дескриптора **.tld** пользовательских тегов должен содержать корневой элемент **<taglib>**, содержащий список описаний тегов в элементах **<tag>**.

Каждый из элементов определяет имя тега, под которым к нему можно обращаться на странице JSP, и идентифицирует класс, который обрабатывает тег.

Параметры тега <taglib>

- **tlib-version** – версия пользовательской библиотеки тегов;
- **short-name** – краткое имя библиотеки тегов. В качестве него принято указывать рекомендуемое сокращение для использования в JSP-страницах;
- **uri** – уникальный идентификатор ресурса, определяющий данную библиотеку. Параметр необязательный, но если его не указать, то необходимо регистрировать библиотеку в каждом новом приложении через файл **web.xml**;
- **info** – указывается область применения данной библиотеки.

Основным в элементе **<taglib> является элемент <tag>. В элементе tag** между его начальным **<tag> и конечным </tag> тегами должны находиться** четыре составляющих элемента:

- **name** – тело этого элемента определяет имя базового тега, к которому будет присоединяться префикс директивы **taglib**;
- **tag-class** – полное имя класса-обработчика тега;
- **info** – краткое описание тега;
- **body-content** – имеет значение **empty**, если теги не имеют тела. Теги с телом, содержимое которого может интерпретироваться как обычный JSP-код, используют значение **jsp**, а редко используемые теги, тела которых полностью обрабатываются, используют значение **tagdependent**.

Для JSP версии 2.1 тег **taglib** записывается в виде:

```
<taglib version= "2.1"
  xmlns= "http://java.sun.com/xml/ns/javaee"
  xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation= "http://java.sun.com/xml/ns/javaee
web-jsptaglibrary_2_1.xsd">
```

Зарегистрировать адрес URI библиотеки пользовательских тегов *mytaglib.tld* для приложения можно двумя способами

1. Указать доступ к ней в файле **web.xml**, для чего следует указать после <welcome-file-list\>:

```
<jsp-config>
          <taglib>
                    <taglib-uri>
                              /WEB-INF/tld/mytaglib.tld
                    </taglib-uri>

                    <taglib-location>
                              /WEB-INF/tld/mytaglib.tld
                    </taglib-location>
          </taglib>
</jsp-config>
```

2. Прописать URI библиотеки в файле-описании (**.tld**) библиотеки и поместить этот файл в папку **/WEB-INF** проекта. В таком случае в файле **web.xml** ничего прописывать не требуется.

Тег может содержать параметры и передавать их значения для обработки в соответствующий ему класс. Для этого при описании тега в файле ***.tld** используются атрибуты, которые должны объявляться внутри элемента **tag** с помощью элемента **attribute**.

Соответственно для каждого из атрибутов тега класс, его реализующий, должен содержать метод **set**ИмяАтрибута**()**.

Внутри элемента **attribute** между тегами **<attribute>** и **</attribute>** могут находиться следующие элементы:

- **name** – имя атрибута (обязательный элемент);
- **required** – указывает на то, всегда ли должен присутствовать данный атрибут при использовании тега, который принимает значение **true** или **false** (обязательный элемент);
- **rtexprvalue** – показывает, может ли значение атрибута быть JSP-выражением вида **${expr}** или **<%=expr%>** (значение **true**) или оно должно задаваться строкой данных (значение **false**). По умолчанию устанавливается **false**, поэтому этот элемент обычно опускается, если не требуется задавать значения атрибутов во время запроса (необязательный элемент).

Когда разрабатывается пользовательский тег с телом, то лучше наследовать класс тега от класса **BodyTagSupport**, реализующего в свою очередь интерфейс **BodyTag**.

Для того чтобы тело было обработано, метод **doStartTag()** должен вернуть **EVAL_BODY_INCLUDE** или **EVAL_BODY_BUFFERED**; если будет возвращено **SKIP_BODY**, то метод **doInitBody()** не вызывается.

Как и в обычных тегах, между открывающим и закрывающим пользовательскими тегами может находиться тело тега, или **body**. Пользовательские теги могут использовать содержимое элемента **body-content**.

На данный момент поддерживаются следующие значения для **body-content**:

- **empty** – пустое тело;
- **jsp** – тело состоит из всего того, что может находиться в JSP-файле. Используется для расширения функциональности JSP-страницы;
- **tagdependent** – тело интерпретируется классом, реализующим данный тег. Используется в очень частных случаях.

Кроме методов класса **TagSupport** (суперкласс для **BodyTagSupport**), интерфейс **BodyTag** имеет методы, среди которых следует выделить:

- **void doInitBody()** – вызывается один раз перед первой обработкой тела, после вызова метода **doStartTag()** и перед вызовом **doAfterBody()**;
- **int doAfterBody()** – вызывается после каждой обработки тела. Если вернуть в нем константу **EVAL_BODY_AGAIN**, то **doAfterBody()** будет вызван еще раз. Если **SKIP_BODY**, то обработка тела будет завершена;
- **int doEndTag()** – вызывается один раз, когда отработаны все остальные методы.

СПАСИБО ЗА ВНИМАНИЕ!

ВОПРОСЫ?

Java for students

Филиал кафедры ПОИТ БГУИР в Epam Systems
курс: Веб-технологии (JAVA)

Author: **Olga Smolyakova , PhD**
Oracle Certified Java 6 Programmer
Olga_Smolyakova@epam.com