Глава 22

ПОЛЬЗОВАТЕЛЬСКИЕ ТЕГИ

Начиная с версии JSP 1.1 у разработчиков появилась возможность определения собственных тегов. Это значительно упростило жизнь Web-дизайнерам, которым привычнее использовать теги, а не код на языке Java. Если один и тот же скриптлет используется на разных страницах, то он явный кандидат для переноса кода в пользовательский тег. Фактически последний представляет собой перенос Java-кода из страницы JSP в Java-класс, что можно считать продолжением идеи о необходимости отделения логики от представления. JSP-страница должна содержать как можно меньше логики.

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.

Простой тег

Для создания тега без атрибутов или тела необходимо переопределить метод **doStartTag()**, определяющий код, который вызывается во время запроса, если обнаруживается начальный элемент тега.

В качестве примера можно привести следующий класс тега, с помощью которого клиенту отправляется информация о размере некоторой коллекции объектов.

```
/ пример # 1 : простейший тег без тела и атрибутов : GetInfoTag.java */
package test.mytag;
import javax.servlet.jsp.tagext.TagSupport;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import java.io.IOException;
// класс бизнес-логики (см. пример #7 этой главы)
import test.my.MySet;

public class GetInfoTag extends TagSupport {
      public int doStartTag() throws JspException {
//получение информации, передаваемой на страницу
```

```
        int size = new Integer(new MySet().getSize());
        String str = "Size =<B>( " + size + " )</B>";
              try {
                    JspWriter out = pageContext.getOut();
                    out.write(str);
              } catch (IOException e) {
              throw new JspException(e.getMessage());
              }
              return SKIP_BODY;
        }
}
```

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. Метод **getOut()** этого класса возвращает ссылку на поток **JspWriter**, с помощью которой осуществляется вывод. С помощью методов класса **PageContext** можно получить:

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

Кроме этого:

– с помощью метода **forward(String relativeUrlPath)** сделать перенаправление на другую страницу или action-класс;

– с помощью метода **include()** включить в поток выполнения текущие ресурсы **ServletRequest** или **ServletResponse**, определяемые относительным адресом.

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.

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. Для идентификации используется полное имя класса, например: **test.mytag.GetInfoTag**. Также должен присутствовать стандартный заголовок XML-файла с указанием версии и адреса ресурса для схемы XSD, который определяет допустимый формат тега **<taglib>**.

Перед списком тегов, сразу после открывающего тега **<taglib>**, указываются следующие параметры:

- **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**.

Вся эта информация помещается в файл **mytaglib.tld**, который для JSP версии 2.0 имеет вид:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<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/mytaglib.tld</uri>
      <tag>
            <name>getinfo</name>
            <!--класс обработки тега -->
            <tag-class>test.mytag.GetInfoTag</tag-class>
            <body-content>empty</body-content>
      </tag>
</taglib>
```

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 **mytag-lib.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/mytaglib.tld</taglib-uri>
      <taglib-location>/WEB-INF/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. Преимуществом данного способа является то, что так можно использовать библиотеку во многих приложениях под одним и тем же адресом URI. Естественно, в этом случае TLD-файл должен размещаться не в локальной папке проекта, а, например, в сети Интернет как независимый файл.

Непосредственное использование в странице JSP созданного и зарегистрированного простейшего тега выглядит следующим образом:

```
<!-- пример # 2 : вызов простого тега : demotag1.jsp -->
<HTML><HEAD>
<%@ taglib uri="/WEB-INF/mytaglib.tld"
             prefix="mytag" %>
      </HEAD>
            <BODY>
                  <mytag:getinfo/>
            </BODY>
</HTML>
```

В результате выполнения тега клиент в браузере получит следующую информацию:

```
Size = (3)
```

Тег с атрибутами

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**. 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).

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

В следующем примере рассматривается простейший тег с атрибутом **firstname**, который выводит пользователю сообщение:

// пример # 3 : тег с атрибутом : HelloTag.java

```
package test.mytag;
import javax.servlet.jsp.tagext.TagSupport;
import java.io.IOException;

public class HelloTag extends TagSupport {
    private String firstname;

     public void setFirstname(String firstname) {
        this.firstname = firstname;
    }
     public int doStartTag() {
        try {
        pageContext.getOut().write("Hello, " + firstname);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return SKIP_BODY;
    }
}
```

В файл **mytaglib.tld** должна быть помещена следующая информация о теге:

```
<tag>
  <name>hello</name>
  <tag-class>test.mytag.HelloTag</tag-class>
  <body-content>empty</body-content>
  <attribute>
      <name>firstname</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
  </attribute>
</tag>
```

Использовать созданный тег в файле **demotag2.jsp** можно следующим образом:

пример # 4 : вызов тега с передачей ему значения : demotag2.jsp

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

```
<%@ taglib uri="/WEB-INF/mytaglib.tld" prefix="mytag"%>
<%@ page
            language="java"
            contentType="text/html; charset=CP1251"
            pageEncoding="CP1251"
      %>
<HTML><HEAD>
      <TITLE>demotag2.jsp</TITLE>
      </HEAD>
            <BODY>
      <c:set var="login" value="Bender"/>
      <mytag:hello firstname="${login}" />
            </BODY>
</HTML>
```

При обращении по адресу:

http://localhost:8080/FirstProject/demotag2.jsp

в браузер будет выведено:

Hello, Бендер

Тег с телом

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.

Когда разрабатывается пользовательский тег с телом, то лучше наследовать класс тега от класса **BodyTagSupport**, реализующего в свою очередь интерфейс **BodyTag**. Кроме методов класса **TagSupport** (суперкласс для **BodyTagSupport**), он 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.

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.

В следующем примере рассматривается класс обработки тега, который получает значения атрибута **num** (в данном случае методом установки значения для

атрибута **num** будет метод **setNum(String num)**) и формирует таблицу с указанным количеством строк, куда заносятся значения из тела тега:

```
// пример # 5 : тег с телом : AttrTag.java
package test.mytag;
import java.io.IOException;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class AttrTag 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) {
                  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;
      }
}
```

В файл **.tld** следует вставить информацию о теге в виде:

```
<tag>
      <name>bodyattr</name>
      <tag-class>test.mytag.AttrTag</tag-class>
      <body-content>JSP</body-content>
      <attribute>
            <name>num</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
      </attribute>
</tag>
```

При использовании в файле JSP тег **bodyattr** может вызываться с параметрами и без них:

пример # 6 : тег с телом : demotag3.jsp

```
<%@ page language="java" contentType="text/html; char-
set=ISO-8859-5" pageEncoding="ISO-8859-5"%>
<%@ taglib uri="/WEB-INF/mytaglib.tld" prefix="mytag" %>
<HTML><HEAD>
      <TITLE>Example</TITLE>
</HEAD><BODY>
<jsp:useBean id="rw" scope="request" class=
                                          "test.my.MySet"/>
      <mytag:bodyattr num="${rw.size}">
            ${rw.element}
      </mytag:bodyattr>
      <mytag:bodyattr> Просто текст </mytag:bodyattr>
</BODY></HTML>
```

В результате запуска этой JSP клиенту будет возвращено:

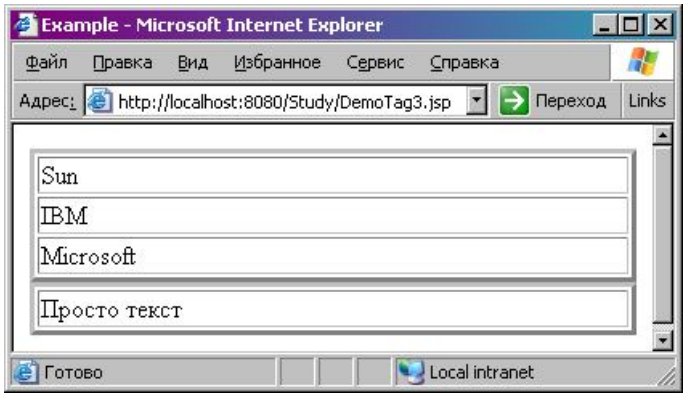

Рис. 22.1. Выполнение тега с телом

В примерах данной главы были использованы методы класса **Rows**, который приведен ниже:

/ пример # 7 : примитивный класс бизнес-логики : MySet.java */*

```
package test.my;
```

```
public class MySet extends java.util.HashSet {
      private java.util.Iterator it;

      public MySet() {
//переписать этот класс на чтение информации из БД
            this.add("Sun");
            this.add("Microsoft");
            this.add("IBM");
      }
      public String getSize() {
            it = this.iterator();
            return Integer.toString(this.size());
      }
      public String getElement() {
            return it.next().toString();
      }
}
```

Элементы action

Элемент **jsp:attribute** позволяет определить значение атрибута тега в теле XML-элемента, а не через значение атрибута стандартного или пользовательского тега:

```
<%@ taglib uri="/WEB-INF/mytaglib.tld" prefix="mytag" %>
<HTML>
      <mytag:hello>
            <jsp:attribute name="firstname">
                  Bender
            </jsp:attribute>
      </mytag:hello>
</HTML>
```

в браузер будет выведено:

Hello, Bender

Если в теле тега имеются элементы **jsp:attribute**, то тело тега нужно указать явно при помощи стандартного действия **jsp:body**:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c"%>
<%@ taglib uri="/WEB-INF/mytaglib.tld" prefix="mytag" %>
<HTML>
      <jsp:useBean id="rw" scope="request"
                           class="test.my.MySet"/>
      <mytag:bodyattr>
            <jsp:attribute name="num">
                  <c:out value="${requestScope.rw.size}"/>
            </jsp:attribute>
            <jsp:body>
                <c:out value="${requestScope.rw.element}"/>
            </jsp:body>
      </mytag:bodyattr>
</HTML>
```

В результате в браузер будет выведено:

IBM
Sun
Microsoft

Элемент **jsp:element** с обязательным атрибутом **name** используется для динамического определения элемента XML и дополнительно может содержать действия **jsp:attribute** и **jsp:body**:

```
<jsp:element name="H2" >
      <jsp:attribute name="Style">
            color:red
      </jsp:attribute>
      <jsp:body>
            Simple Text
      </jsp:body>
</jsp:element>
```

в результате должно быть сгенерировано:

<H2 Style="color:red">Simple Text</H2>

Стандартные действия **jsp:doBody** и **jsp:invoke** используются только в тег-файлах. Тег **jsp:doBody** вызывает тело тега, выводя результат в **JspWriter** или в атрибут области видимости. Действие **jsp:invoke** подобно действию **jsp:doBody** и используется для вызова атрибута-фрагмента. Например, поведение тега **bodyattr** можно воспроизвести так:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c"%>
<%@ taglib prefix="tags" tagdir="/WEB-INF/tags" %>
<tags:actiondemo>
      <jsp:attribute name="num">
            <c:out val-
ue="${sessionScope.mysetInstance.size}"/>
      </jsp:attribute>
      <jsp:body>
            <c:out val-
ue="${sessionScope.mysetInstance.element}"/>
      </jsp:body>
</tags:actiondemo>
```

Файл **actiondemo.tag** помещен в каталог **/WEB-INF/tags**:

```
<%@ tag import="test.my.MySet" %>
<%@ attribute name="num" fragment="true" %>
<%@ variable name-given="mysetInstance" %>
<%session.setAttribute("mysetInstance", new MySet());%>
<TABLE border=1>
<TR><TD>Rows number: <jsp:invoke fragment="num"</TD></TR>
      <TR><TD><jsp:doBody /></TD></TR>
      <TR><TD><jsp:doBody /></TD></TR>
      <TR><TD><jsp:doBody /></TD></TR>
</TABLE>
```

Здесь директива **tag** схожа с директивой **page** для страниц JSP. Директива **attribute** декларирует атрибут тега **actiondemo**, и если **fragment="true"**, то этот атрибут можно использовать совместно с **jsp:invoke**. Директива **variable** – для передачи переменной обратно в вызывающую JSP-страницу.
В браузер будет выведено:

Rows number: 3
IBM
Sun
Microsoft

Задания к главе 22

Вариант А

Создать классы пользовательских тегов, формирующих нужное количество элементов (строк, ячеек и др.) для размещения результатов выполнения запроса.

1. Элемент массива называют локальным максимумом, если у него нет соседа большего, чем он сам. Аналогично определяется локальный минимум. Определить количество локальных максимумов и локальных минимумов в заданном строкой массиве чисел. Массив задает клиент. Возвратить все максимумы и минимумы пользователю.
2. В неубывающей последовательности, заданной клиентом, найти количество различных элементов и количество элементов, меньших, чем заданное число, и вернуть ему результат.
3. Дана числовая последовательность $a_1, a_2, \ldots, a_n$. Вычислить суммы вида $S_i = a_i + a_{i+1} + \ldots + a_j$ для всех $1 \le i \le j \le N$ и среди этих сумм определить максимальную. Последовательность и число N задает клиент.
4. Точка A и некоторое конечное множество точек в пространстве заданы своими координатами и хранятся в базе данных. Найти N точек из множества, ближайших к точке A. Число N задает клиент.
5. В базе данных хранится список студентов и их оценок по предметам за сессию по 100-балльной системе. Выбрать без повторений все оценки и соответствующие им записи, встречающиеся более одного раза.
6. Получить упорядоченный по возрастанию массив C, состоящий из k элементов, путем слияния упорядоченных по возрастанию массивов A и B, содержащих n и m элементов соответственно, k = n + m. Элементы массивов хранятся в базе данных, а значения n и m задает клиент.
7. В матрице A найти сумму элементов, расположенных в строках с отрицательным элементом на главной диагонали, и произведение элементов, расположенных в строках с положительным элементом в первом столбце. Матрица размерности n хранится в базе данных. Клиент задает размерность m < n матрицы, для которой будет произведен расчет.
8. В программе, хранящейся в текстовом файле, удалить строки с № 1 до № 2, где № 1 и № 2 вводятся клиентом. Удаляемые строки возвратить клиенту. Предусмотреть случаи, когда, например, № 1 меньше номера первой строки, № 1 = № 2, № 2 больше номера последней строки, и другие исключительные ситуации.
9. После n-ой строки программы, которая хранится в файле, вставить m строк. Числа n, m и вставляемые строки вводятся пользователем. Новый набор данных сохранить на диске и вернуть клиенту.

10. В БД хранятся координаты множества m точек трехмерного пространства. Найти такую точку, чтобы шар заданного радиуса с центром в этой точке содержал максимальное число точек. Координаты найденных точек возвратить клиенту.
11. Из заданного множества точек на плоскости, координаты которых хранятся в базе данных, выбрать две различные точки, так чтобы окружности заданного пользователем радиуса с центрами в этих точках содержали внутри себя одинаковое количество заданных точек. Полученные множества возвратить клиенту.
12. В базе данных хранятся координаты конечного множества точек плоскости. Пользователем вводятся координаты центра и радиусы 5 концентрических окружностей. Между какими окружностями (1 и 2, 2 и 3, ..., 4 и 5) больше всего точек заданного множества? Полученное множество точек возвратить клиенту.
13. В базе данных хранятся координаты вершин выпуклых четырехугольников на плоскости. Сформировать ответ клиенту, содержащий координаты всех вершин трапеций, которые можно сформировать из данных точек.
14. В базе данных хранятся координаты вершин треугольников на плоскости. Для прямоугольных треугольников возвратить клиенту координаты вершин прямого угла, площадь и координаты вершин (одной или двух), ближайших к оси OX.
15. В базе данных хранятся координаты множества точек плоскости A и коэффициенты уравнений множества прямых в этой же плоскости. Передать клиенту набор из пар различных точек – таких, что проходящая через них прямая параллельна прямой из множества B.

Вариант B

Для заданий варианта B предыдущей главы применить пользовательские теги для визуализации работы приложения.

Тестовые задания к главе 22

Вопрос 22.1.

Какой элемент тега **<attribute>** определяет имя атрибута, которое должно быть передано обработчику тегов?

1) `<attribute-name>;`
2) `<name>;`
3) `<attributename>;`
4) `<param-name>`.

Вопрос 22.2.

Обработчик тега реализует интерфейс **BodyTag**. Сколько раз может быть в нем вызван метод **doAfterBody()**?

1) класс **BodyTag** не поддерживает метод **doAfterBody();**
2) 0;
3) 1;
4) 0 или 1;
5) сколько угодно раз.

Вопрос 22.3.

Какой метод обработчика тега будет вызван, если метод **doStartTag()** вернет значение **Tag.SKIP_BODY**?

1) doAfterBody();
2) doBody();
3) skipBody();
4) doEndTag();.
5) нет правильного.

Вопрос 22.4.

Какой из следующих элементов необходим для корректности тега **<taglib>** в файле **web.xml**?

1) <uri-tag>;
2) <tag-uri>;
3) <uri-name>;
4) <uri-location>;
5) <taglib-uri>.

Вопрос 22.5.

Какие элементы описывают характеристики пользовательского тега в файле **.tld**?

1) value;
2) name;
3) rtexprvalue;
4) class.

Вопрос 22.6.

Какие утверждения верны относительно метода **doInitBody()** класса **BodyTagSupport**?

1) используется контейнером и не может быть переопределен;
2) он может быть переопределен;
3) может возвращать или константы **SKIP_BODY**, или **EVAL_BODY_INCLUDE;**
4) его возвращаемое значение имеет тип **void**.

Вопрос 22.7.

Что нужно сделать в файле **.tld** для этого тега, чтобы в теле тега использовать скриптлеты?

1) в **body-content** должно быть выставлено значение **jsp;**
2) в **script-enabled** должно быть выставлено **true;**
3) ничего, так как скриптлеты используются по умолчанию.