Библиотека стандартных действий JSTL

Создание библиотеки стандартных действий JSTL преследовало цель упрощения разработки JSP-страниц для автора страниц – лица, ответственного за дизайн веб-приложения, создаваемый с помощью JSP-страниц. Многие авторы страниц не обладают достаточными познаниями в программировании, поэтому использование скриптового языка для динамической генерации результата в JSP-странице (по умолчанию – языка Java) представляет для них определенную сложность.

Библиотека JSTL предоставляет следующие возможности:

- Действия общего назначения – дополняют язык EL, позволяя выводить значение выражений на языке EL, устанавливать и удалять атрибуты в различных контекстах, а также обрабатывать исключения.
- Действия для управления потоком выполнения – условные и циклические действия, представляющие собой более удобный способ управления выполнением JSP-страницы, чем скриптлеты.
- Действия для доступа к ресурсам по URL.
- Действия для интернационализации интерфейса веб-приложения и форматирования текста.
- Действия для доступа к реляционным базам данных.
- Действия для обработки данных в формате XML.
- Функции манипулирования строками в EL-выражениях.

Так как библиотека JSTL включает большое количество действий, относящихся к различным группам функций, она логически разделена на несколько библиотек действий (табл. 9.1)..

Таблица 9.1

Библиотеки действий, входящие в JSTL

Функции	URI	Префикс
Базовые	`http://java.sun.com/jsp/jstl/core`	`c`
Обработка XML	`ttp://java.sun.com/jsp/jstl/xml`	`x`
Форматирование	`http://java.sun.com/jsp/jstl/fmt`	`fmt`
Доступ к РСУБД (SQL)	`http://java.sun.com/jsp/jstl/sql`	`sql`
Функции	`http://java.sun.com/jsp/jstl/functions`	`fn`

Все синтаксические ошибки в действиях JSTL должны выявляться на этапе трансляции JSP-страницы.

Преобразование строковых значений к допустимому типу значений атрибута выполняется в соответствии с правилами, определенными в спецификации JSP.

Так как автору страниц сложно работать с исключениями, библиотека JSTL пытается насколько это возможно избегать возникновения исключительных ситуаций. Например, если бы действие `<c:forEach>` выбрасывало исключение при передаче значения `null` для атрибуты `items`, то было бы затруднительно обрабатывать возможно отсутствующий массив строк, представляющий установленные в HTML-форме флажки и возвращаемый EL-выражением наподобие `${paramValues.selections}`. В данной ситуации лучшим действием является отсутствие какого-либо действия (и исключения).

Контекстные переменные

Действия обычно явно и/или неявно взаимодействуют со своим окружением.

Неявное взаимодействие часто выполняется через определенный интерфейс, который позволяет вложенным действиям прозрачно работать с объемлющим их

действием. Например, данный режим взаимодействия поддерживается циклическими действиями библиотеки JSTL.

Явное взаимодействие происходит, когда действие в явном виде предоставляет информацию своему окружению. Традиционно это выполнялось с помощью скриптовых переменных, значения которых брались из атрибутов, сохраненных обработчиком действия в том или ином контексте (страница, запрос, сессия, приложение). После появления языка EL потребность в скриптовых переменных существенно уменьшилась, поэтому все действия библиотеки JSTL предоставляют информацию только в виде атрибутов того или иного контекста, не определяя скриптовых переменных. Далее данные атрибуты называются контекстными переменными.

В библиотеке JSTL принято соглашение обозначать атрибут, определяющий имя контекстной переменной, именем `var`. Например, циклическое действие `<c:forEach>` предоставляет доступ к текущему элементу коллекции следующим образом:

```
<c:forEach var="customer" items="${customers}">
    Current customer is <c:out value="${customer}"/>
</c:forEach>
```

Контекстные переменные, экспортируемые действиями JSTL, разделяются на вложенные и доступные после окончания действия. Первые доступны только в теле действия и хранятся в контексте страницы. Последние доступны только после окончания действия и их жизненный цикл определяется контекстом, в котором они хранятся. По соглашению контекст хранения таких переменных определяется атрибутом действия `scope`.

Большинство атрибутов действий JSTL являются динамическими, то есть их значение может формироваться на этапе выполнения JSP-страницы. Наиболее важные исключения – атрибуты `var` и `scope`, определяющие имя и контекст экспортируемой действием переменной.

Действия общего назначения

Действие `<c:out>` предоставляет возможности, аналогичные скриптовым выражениям и EL-выражениям в тексте шаблона, например:

```
You have <c:out value="${sessionScope.user.itemCount}"/> items.
```

По умолчанию действие `<c:out>` преобразует символы `<`, `>`, `'`, `"`, `&` в соответствующие т.н. сущности (например, `<` преобразуется в `&lt;`). Если не выполнять данное преобразование, то страница не сможет правильно отображаться в браузере, а также может открыть лазейку для межсайтовых скриптовых атак (cross-site scripting attacks)[1].

Действие `<c:out>` позволяет указать с помощью атрибута `default` значение по умолчанию для тех случаев, когда значение EL-выражения равно `null`. В следующем примере будет выведено значение «неизвестно», если свойство city недоступно:

```
<c:out value="${customer.address.city}" default="неизвестно"/>
```

Действие `<c:set>` устанавливает значение контекстной переменной:

```
<c:set var="foo" value="value"/>
```

Значение переменной можно также указать в теле действия `<c:set>`. Это позволяет присвоить переменной результат выполнения какого-либо действия. В следующем примере действие `<c:set>` присваивает контекстной переменной `att1` результат вывода действия `<acme:foo>`:

```
<c:set var="att1">
    <acme:foo>mumbojumbo</acme:foo>
</c:set>
<acme:atag att1="${att1}"/>
```

[1] Например, злоумышленник может использовать код на языке JavaScript для закрытия окна форума.

Действие <c:set> может также устанавливать свойство объекта JavaBeans или устанавливать конкретный элемент в хэш-таблице (объекте типа java.util.Map), например:

```
<!-- установить свойство в объекте JavaBeans -->
<c:set target="${cust.address}" property="city" value="${city}"/>
<!-- установить элемент хэш-таблицы -->
<c:set target="${preferences}" property="color"
value="${param.color}"/>
```

Действие <c:remove> естественным образом дополняет действие <c:set>, позволяя удалить контекстную переменную, например:

```
<c:remove var="cachedResult" scope="application"/>
```

Условные действия

Содержимое JSP-страницы часто формируется в зависимости от выполнения различных условий для данных, получаемых динамически. В таких случаях можно использовать простой скриптлет с условным оператором if, однако это заставляет автора страниц использовать скриптовый язык, синтаксис которого может стать причиной ошибок (например, можно забыть про операторные скобки {}). Условные действия из библиотеки JSTL упрощают условную обработку JSP-страницы.

Действие простого условного выполнения <c:if> вычисляет содержимое своего тела, только если истинно указанное условие. В следующем примере особое приветствие отображается пользователю только при первом посещении сайта:

```
<c:if test="${user.visitCount == 1}">
    Это Ваше первое посещение сайта. Добро пожаловать!
</c:if>
```

При обработке действия взаимоисключающего условного выполнения <c:choose> вычисляется тело первого из множества альтернативных действий <c:when>, для которого истинно указанное условие, либо тело действия <c:otherwise>, если все условия ложны. В следующем примере показано, как вывести различный текст в зависимости от категории пользователя:

```
<c:choose>
    <c:when test="${user.category == 'trial'}">
        ...
    </c:when>
    <c:when test="${user.category == 'member'}">
        ...
    </c:when>
    <c:when test="${user.category == 'vip'}">
        ...
    </c:when>
    <c:otherwise>
        ...
    </c:otherwise>
</c:choose>
```

С помощью действия <c:choose> можно достичь эффекта, аналогичного оператору if/then/else:

```
<c:choose>
    <c:when test="${count == 0}">
        Нет записей, отвечающих условиям запроса.
    </c:when>
    <c:otherwise>
        ${count} записей выбрано по запросу.
    </c:otherwise>
</c:choose>
```

Циклические действия

Проход в цикле по коллекции объектов часто встречается на JSP-страницах. Для этого можно использовать простой скриптлет, но это требует от автора страниц знание многих аспектов языка Java (способов итерации по коллекциям различных типов, приведение возвращаемого объекта к правильному типу, использование операторных скобок и т.п.) Циклические действия библиотеки JSTL упрощают работу с различными видами коллекций объектов.

Действие `<c:forEach>` повторяет вычисление своего тела для каждого элемента коллекции объектов, указанной атрибутом `items`, экспортируя вложенную контекстную переменную, содержащую текущий элемент коллекции. Данное действие поддерживает следующие типы коллекций: все реализации интерфейсов `java.util.Collection` и `java.util.Map`, массивы объектов, массивы примитивных типов (текущий элемент массива преобразуется в объект класса-обертки). Также поддерживаются итераторы и перечисления (реализации интерфейсов `java.util.Iterator` и `java.util.Enumeration`, соответственно). Тип `java.sql.ResultSet` не поддерживается.

Следующий пример создает таблицу с одним столбцом, содержащим отображаемое значение по умолчанию для каждого элемента коллекции:

```
<table>
    <c:forEach var="customer" items="${customers}">
        <tr><td>${customer}</td></tr>
    </c:forEach>
</table>
```

Действие `<c:forEach>` поддерживает атрибуты ограничения диапазона итерации `begin`, `end` и `step`, определяющие начальный и конечный индексы и шаг итерации. Если значение атрибута `items` не указано, то значением текущего элемента является целочисленное значение текущего индекса. В следующем примере переменная `i` принимает значения от 100 до 110, включительно:

```
<c:forEach var="i" begin="100" end="110">
    ${i}
</c:forEach>
```

Действие `<c:forEach>` предоставляет интерфейс и базовый класс реализации, которые можно использовать при разработке как вложенных действий, так и собственных циклических действий.

Помимо действия `<c:forEach>` в библиотеку JSTL также входят циклические действия `<x:forEach>`, выполняющее цикл по результатам выполнения XPath-выражения, и `<c:forToken>`, поочередно выделяющее подстроки из заданной строки согласно установленному разделителю.

Действия для работы с URL

В JSP-страницах часто необходимо указывать ссылки и перенаправлять запросы на ресурсы по их URL, а также импортировать их содержимое. В библиотеку JSTL входят действия, позволяющие упростить эти действия.

Для определения гипертекстовой ссылки в языке HTML предназначен элемент `<A>`. Если ссылка относится к локальному ресурсу веб-приложения, то для поддержки сессий может потребоваться переписать URL. Кроме того, передаваемые в составе URL параметры запроса должны быть соответствующим образом закодированы[2]. Например, пробел в части `query` URL кодируется как '+':

```
http://acme.com/app/choose?country=Russian+Federation
```

В следующем примере показано, как комбинация действий `<c:url>` и `<c:param>` используется для переписывания и кодирования URL: действие `<c:url>` при

[2] Правила кодирования (т.н. URL encoding) приведены в RFC 2396.

необходимости переписывает URL, а действия <c:param> кодируют параметры строки запроса (и имена, и значения параметров). Результирующий URL сохраняется в контекстной переменной myUrl и выводится в составе элемента <A>.

```
<c:url value="http://acme.com/exec/register" var="myUrl">
    <c:param name="name" value="${param.name}"/>
    <c:param name="country" value="${param.country}"/>
</c:url>
<a href='<c:out value="${myUrl}"/>'>Register</a>
```

Действие <c:url> прозрачно добавляет корень веб-приложения к относительным URL. Например, если корень веб-приложения – /foo, то следующее действие возвращает адрес /foo/ads/logo.html:

```
<a href='<c:url value="/ads/logo.html"/>'>Logo</a>
```

По умолчанию полученный в действии <c:url> URL не сохраняется в контекстной переменной, а выводится в результат обработки JSP-страницы, что показано в предыдущем примере.

Стандартное действие <jsp:include> предназначено для включения статических и динамических ресурсов, располагающихся только в том же самом веб-приложении, что и текущая страница. Поэтому в библиотеку JSTL входит действие <c:import>, позволяющее включить в страницу произвольный ресурс по его URL.

В атрибуте url указывается абсолютный или относительный (в текущем или другом веб-приложении) URL включаемого ресурса:

```
<%-- импорт ресурса с абсолютным URL --%>
<c:import url="http://acme.com/exec/customers?country=Japan"/>
<%-- импорт ресурса с относительным URL в текущем веб-приложении --%>
<c:import url="/copyright.html"/>
<%-- импорт ресурса с относительным URL в другом веб-приложении --%>
<c:import url="/logo.html" context="/master"/>
```

Действие <c:redirect> выполняет перенаправление запроса на указанный URL, например:

```
<c:redirect url="http://acme.com/register"/>
```

Функции

Язык EL содержит механизм функций для расширения возможностей языка. В библиотеке JSTL определен стандартный набор EL-функций.

В языке EL отсутствует возможность получения размера коллекции объектов, так как метод Collection.size() и атрибут массива length не являются свойствами, отвечающими требованиям JavaBeans. В библиотеке JSTL определена функция length, которая принимает аргумент любого типа из числа поддерживаемых действием <c:forEach> и возвращает размер коллекции (или длину строки). В следующем примере выводится длина коллекции объектов класса Athlete, хранящейся в контекстной переменной athletes:

```
There are ${fn:length(athletes)} athletes representing ${country}
```

В библиотеке JSTL определены следующие функции манипулирования строками:

- изменение регистра строки (toLowerCase, toUpperCase);
- получение подстроки (substring, substringAfter, substringBefore);
- удаление начальных и конечных пробельных символов из строки (trim);
- замена символов в строке (replace);
- проверка наличия подстроки в строке (indexOf, startsWith, endsWith, contains, containsIgnoreCase);
- разбивка строки на массив подстрок (split) и соединение массива подстрок в строку (join)
- преобразование служебных символов XML в сущности (escapeXml).

Следующие примеры демонстрируют использование данных функций.

```
<%-- получение первых 30 символов строки и перевод их в верхний регистр --%>
${fn:toUpperCase(fn:substring(name, 0, 30))}
<%-- значение переменной text до первого символа '*' --%>
${fn:substringBefore(text, '*')}
<%-- удаление лишних пробелов из контекстной переменной custId, чтобы URL не
содержал лишних символов '+' --%>
<c:url var="myUrl" value="${base}/cust">
    <c:param name="custId" value="${fn:trim(custId)}"/>
</c:url>
<%-- вывод текста, заключенного в скобках --%>
${fn:substring(text, fn:indexOf(text, '(')+1,
fn:indexOf(text, ')'))}
<%-- вывод переменной name, если она содержит искомую строку --%>
<c:if test="${fn:containsIgnoreCase(name, searchString)}">
    Found name: ${name}
</c:if>
<%-- вывод последних 10 символов переменной text --%>
${fn:substring(text, fn:length(text)-10)}
<%-- вывод значения переменной text с точками вместо '-' --%>
${fn:replace(text, '-', '&#149;')}
```

Хотя для преобразования служебных символов XML можно использовать действие `<c:out>`, функция `escapeXml` предоставляет для этого более понятный синтаксис, что видно на примере установки значения атрибута `name` действия `<my:tag>`:

```
<%-- Использование действия <c:out> совместно с <c:set> --%>
<c:set var="nameEscaped">
    <c:out value="${name}"/>
</c:set>
<my:tag name="${nameEscaped}"/>
<%-- Использование действия <c:out> совместно с <jsp:attribute>--%>
<my:tag>
    <jsp:attribute name="title">
        <c:out value="${name}"/>
    </jsp:attribute>
</my:tag>
<%-- Использование функции fn:escapeXml --%>
<my:tag title="${fn:escapeXml(name)}"/>
```

Пример JSP-страницы, использующей библиотеку JSTL

```
<%@ page contentType="text/html" pageEncoding="UTF-8" %>
<%@ page import=" jdbcsample.dao.*" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Client List</title>
    </head>
    <body>

<%@ include file="/header.jsp" %>
    <h2>All Clients by JSTL</h2>

    <table border="1">
        <thead>
            <tr>
                <th>Name</th>
                <th>Place</th>
                <th>Email</th>
            </tr>
        </thead>
        <tbody>
            <c:set var="clients" value="<%= new ClientDAO().findAllClients()
%>"/>
            <c:forEach var="client" items="${clients}">
            <tr>
                <td><a href='<c:url value="/infoClient.jsp"><c:param
name="id" value="${client.id}"/></c:url>'>
                    <c:out value="${client.lastName}
${client.firstName}"/></a></td>
                <td>${client.place}</td>
                <td<c:if test="${fn:endsWith(client.email,'mail.ru')}">
style="color: red"</c:if>>
                    <c:out value="${client.email}" default="(not specified)"
/>
                </td>
            </tr>
            </c:forEach>
        </tbody>
    </table>

    </body>
</html>
```