**Лабораторная работа № 15. Пользовательские теги в JSP-страницах**

**Цель работы:** Разобрать примеры на тему: Пользовательские теги в JSP-страницах.

**Задание:**

**1 Изучите теоретический материал**

**2.Выполните задание: разберите и выполните примеры (выделенные желтым цветом), приведенные ниже.**

**Дополнительные источники:**

[Apache Taglibs - Apache Standard Taglib: JSP[tm] Standard Tag Library (JSTL) implementations](https://tomcat.apache.org/taglibs/standard/)

[Manning | JSTL in Action](https://www.manning.com/books/jstl-in-action)

https://o7planning.org/ru/10429/java-jsp-standard-tag-library-jstl-tutorial

[[JSTL | Стандартная библиотека тегов JSP | Портал информатики для гиков (espressocode.top)](http://espressocode.top/jstl-jsp-standard-tag-library/) Создание JSP тэга и своей библиотеки тэгов (Java Servlet) – YouTube](https://www.youtube.com/watch?v=9aQWlWGXKcw)

[JSTL | Стандартная библиотека тегов JSP | Портал информатики для гиков (espressocode.top)](http://espressocode.top/jstl-jsp-standard-tag-library/)

[Java EE 40: WebXml 13: Создание тега - YouTube](https://www.youtube.com/watch?v=EKsDH5cbC3Q)

**1 Изучите теоретический материал:**

[Java EE | JSTL (metanit.com)](https://metanit.com/java/javaee/3.11.php)

## Установить библиотеку JSTL

## [Apache Taglibs - Apache Standard Taglib: JSP[tm] Standard Tag Library (JSTL) implementations](https://tomcat.apache.org/taglibs/standard/)

## Стандартный Талиб

**Реализации библиотеки стандартных тегов JSP (tm)**

Apache размещает Apache Standard Taglib, реализацию спецификации [JSP Standard Tag Library (JSTL).](http://www.oracle.com/technetwork/java/jstl-137486.html) Доступны различные версии.

|  |  |  |  |
| --- | --- | --- | --- |
| **Версия** | **Версия JSTL** | **Требования** | **Получение Taglib** |
| Стандарт 1.2.3 | JSTL 1.2 | Сервлет 2.5, JavaServer Страницы 2.1 | [скачать](http://tomcat.apache.org/download-taglibs.cgi) ([javadoc](https://tomcat.apache.org/taglibs/standard/apidocs/)) |
| Стандарт 1.1 | JSTL 1.1 | Сервлет 2.4, JavaServer Страницы 2.0 | [Скачать](http://archive.apache.org/dist/jakarta/taglibs/standard/) |
| Стандарт 1.0 | JSTL 1.0 | Сервлет 2.3, JavaServer Страницы 1.2 | [Скачать](http://archive.apache.org/dist/jakarta/taglibs/standard/) |

Чтобы начать работать с JSP-тегами, вам необходимо сначала установить библиотеку JSTL. Если вы используете контейнер Apache Tomcat, выполните следующие два шага:

**Шаг 1** — Загрузите бинарный дистрибутив из [Apache Standard Taglib](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://tomcat.apache.org/taglibs/index.html&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhg89C1ysQbdOHHOYJOggH9vgOnFXg) и распакуйте сжатый файл.

**Шаг 2.** Чтобы использовать Standard Taglib из своего **дистрибутива Jakarta Taglibs** , просто скопируйте JAR-файлы из каталога lib в дистрибутиве в каталог **webapps \ ROOT \ WEB-INF \ lib** вашего приложения.

Чтобы использовать любую из библиотек, вы должны включить директиву <taglib> в верхней части каждой JSP, которая использует библиотеку.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of the Tag Library** | **Function** | **URI** | **Prefix** |
| Core tag library | variable support Flow Control Iterator URL management Miscellaneous Core | [**http://java.sun.com/jsp/jstl/core**](http://java.sun.com/jsp/jstl/core) | c |
| xml tag library | Flow Control Transformation Locale | [**http://java.sun.com/jsp/jstl/xml**](http://java.sun.com/jsp/jstl/xml) | x |
| Internationalization tag library | Message Formatting Number and date formatting | [**http://java.sun.com/jsp/jstl/fmt**](http://java.sun.com/jsp/jstl/fmt) | fmt |
| SQL tag library | Database manipulation | [**http://java.sun.com/jsp/jstl/sql**](http://java.sun.com/jsp/jstl/sql) | sql |
| Functions tag library | Collection length String manipulation | [**http://java.sun.com/jsp/jstl/functions**](http://java.sun.com/jsp/jstl/functions) | fn |

|  |  |
| --- | --- |
| **N** | **Тег и описание** |
| 1 | [<с: из>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_out_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhjNT6b94BdU5mkrNXep6Y431B54yg)  Как <% = …>, но для выражений. |
| 2 | [<c: set>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_set_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhgHquuKF8ICR1CN5kEAA33r_hpshg)  Устанавливает результат вычисления выражения в **область видимости.** |
| 3 | [<c: удалить>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_remove_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhjVQkbRDev66aEdeV8hSZpj3IKl5w)  Удаляет **переменную области действия** (из определенной области, если она указана). |
| 4 | [<с: улов>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_catch_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhioBCekLC7uI8DKPYWj-R8_2uuRdw)  Ловит любой **Throwable,** который происходит в его теле и, возможно, подвергает его. |
| 5 | [<с: если>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_if_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhh-wj005QKKKRizNwliF_1R4r6a-w)  Простой условный тег, который вычисляет свое тело, если предоставленное условие истинно. |
| 6 | [<с: выбрать>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_choose_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhjqyk4t6OFdJy4AMfNi02s6LXxD-A)  Простой условный тег, который устанавливает контекст для взаимоисключающих условных операций, помеченных **<когда>** и **<в противном случае>** . |
| 7 | [<с: когда>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_choose_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhjqyk4t6OFdJy4AMfNi02s6LXxD-A)  Подтег **тега <выбрать>,** который включает его тело, если его условие принимает значение **«истина»** . |
| 8 | [<c: иначе>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_choose_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhjqyk4t6OFdJy4AMfNi02s6LXxD-A)  Подтег **тега <выбрать>,** который следует за тегами **<когда>** и выполняется, только если все предыдущие условия оцениваются как **«ложные»** . |
| 9 | [<с: импорт>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_import_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhidamXmk1dZylVQ3oY69Z_ohYSGBA)  Извлекает абсолютный или относительный URL-адрес и предоставляет его содержимое либо странице, либо String в **«var»** , либо Reader в **«varReader»** . |
| 10 | [<c: forEach>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_foreach_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhgjFRRcCc4h1c_7qK5Q-gq1U0irow)  Базовый тег итерации, принимающий множество различных типов коллекций и поддерживающий поднаборы и другие функции. |
| 11 | [<с: forTokens>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_foreach_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhgjFRRcCc4h1c_7qK5Q-gq1U0irow)  Перебирает токены, разделенные предоставленными разделителями. |
| 12 | [<с: PARAM>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_param_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhgFb4h5xEVGk1L7IQfmE4tKvcndLQ)  Добавляет параметр в URL-адрес содержащего тега **import** . |
| 13 | [<c: redirect>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_redirect_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhjmJ0TTWUgEFlpLYcJ5Dcu2ovAF6Q)  Перенаправляет на новый URL. |
| 14 | [<с: URL>](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://www.tutorialspoint.com/jsp/jstl_core_url_tag.htm&xid=17259,15700022,15700186,15700190,15700256,15700259,15700262&usg=ALkJrhjIkAxSsaukOz8kRBVptD-pqaNY5w)  Создает URL с необязательными параметрами запроса |

Пользовательские теги в JSP-страницах

Пользовательские теги в JSP-страницах

Стандартные JSP-теги, предназначенные для вызова компонентов JavaBeans и распределения запросов, упрощают разработку и обслуживание JSP-страницы. Технология JSP предоставляет также механизм для инкапсуляции других типов динамической функциональности в пользовательских тегах, которые являются расширениями языка JSP. Пользовательские теги обычно распространяются в виде библиотеки тегов, в которой определяется набор связанных пользовательских тегов и содержатся объекты, реализующие эти теги.

Некоторыми примерами задач, которые могут быть выполнены при помощи пользовательских тегов, являются операции над скрытыми объектами, обработка форм, обращение к базам данных и к другим корпоративным службам, таким как электронная почта и службы каталогов, выполнение передачи управления. Библиотеки JSP-тегов создаются разработчиками, являющимися профессионалами в языке программирования Java и экспертами в области доступа к данным и другим службам. Эти библиотеки используются дизайнерами Web-приложений, которые могут сосредоточиться на вопросах презентации, а не заниматься доступом к корпоративным службам. В дополнение к обнадеживающему разделению труда между разработчиками библиотеки и пользователями библиотеки пользовательские теги увеличивают производительность посредством инкапсуляции повторяющихся задач, что дает возможность их повторного использования в более чем одном приложении.

Библиотекам тегов уделяется большое внимание в сообществе технологии JSP. Более подробная информация о библиотеках тегов и ссылки на некоторые свободно доступные библиотеки находятся [здесь](http://java.sun.com/products/jsp/taglibraries.html).

**Пользовательский тег** представляет собой определенный пользователем элемент JSP-языка. Когда JSP-страница, содержащая пользовательский тег, транслируется в сервлет, тег преобразовывается в операции над объектом, называемым обработчиком тега. Затем Web-контейнер вызывает эти операции во время выполнения сервлета JSP-страницы.

Пользовательские теги имеют богатый набор возможностей. Они могут:

* Быть настроенными при помощи атрибутов, переданных из вызывающей страницы.
* Обращаться ко всем объектам, доступным JSP-странице.
* Изменять ответ, генерируемый вызывающей страницей.
* Взаимодействовать между собой. Вы можете создать и инициализировать компонент JavaBeans, создать переменную, обращающуюся к этому компоненту в одном теге, и затем использовать этот компонент в другом теге.
* Быть вложенными, разрешая сложные взаимодействия в JSP-странице.

**JSP-страница использует библиотеку тегов, а также знакомит с различными типами тегов.**

**Объявление библиотек тегов**

Вы объявляете, что JSP-страница будет использовать теги, определенные в библиотеке тегов, при помощи помещения директивы taglib в страницу перед использованием какого-либо пользовательского тега:

<%@ taglib uri="/WEB-INF/tutorial-template.tld" prefix="tt" %>

Атрибут uri относится к URI, который уникально идентифицирует дескриптор библиотеки тегов (TLD), рассмотренный в разделе Дескрипторы библиотеки тегов. Этот URI может быть прямым или косвенным. Атрибут prefix определяет префикс, который отличает теги, определенные данной библиотекой тегов, от тегов, предоставляемых другими библиотеками.

Названия файлов дескрипторов библиотеки тегов должны иметь расширение .tld. TLD-файлы сохраняются в каталоге WEB-INF WAR или в подкаталоге каталога WEB-INF. К каталогу TLD можно ссылаться прямо или косвенно:

<%@ taglib uri="/WEB-INF/tutorial-template.tld" prefix="tt" %>

Следующая директива taglib использует короткое логическое имя для косвенной ссылки к TLD:

<%@ taglib uri="/tutorial-template" prefix="tt" %>

Логическое имя должно отображаться в абсолютный путь в дескрипторе размещения Web-приложения. Для отображения логического имени /tutorial-template в абсолютный путь /WEB-INF/tutorial-template.tld:

1. Выберите Bookstore3WAR.
2. Выберите закладку File Refs.
3. Нажмите кнопку Add в окне JSP Tag Libraries.
4. Введите относительный URI /tutorial-template в поле Coded Reference.
5. Введите абсолютный путь /WEB-INF/tutorial-template.tld в поле Tag Library.

**Типы тегов**

Пользовательские теги JSP пишутся с использованием синтаксиса XML. Они имеют начальный тег, конечный тег и, возможно, тело:

<tt:tag>

**body**

</tt:tag>

Пользовательский тег без тела:

<tt:tag />

**Простые теги**

Простой тег не содержит тела и атрибутов:

<tt:simple />

**Теги с атрибутами**

Пользовательский тег может иметь атрибуты. Атрибуты перечисляются в начальном теге и имеют синтаксис attr="value". Значения атрибута служат для настройки поведения пользовательского тега точно так же, как параметры служат для настройки поведения метода.

Типы атрибутов тега указываются в дескрипторе библиотеки тегов (см. раздел Дескрипторы библиотеки тегов).

Значение атрибута может устанавливаться из константы String или выражения времени исполнения. Процесс преобразования между константами, выражениями времени исполнения и типами атрибутов подчиняется правилам, определенным для свойств компонента JavaBeans и описанным в разделе Установка свойств компонента JavaBeans.

Атрибуты тега logic:present библиотеки Struts определяют, вычисляется ли тело тега. В следующем примере атрибут указывает параметр запроса с именем Clear:

<logic:present parameter=**"Clear"**>

Страница catalog.jsp приложения Duke's Bookstore использует выражение времени выполнения для установки значения атрибута, определяющего набор книг, которые перебираются тегом logic:iterate библиотеки Struts:

<logic:iterate collection=**"**<%=bookDB.getBooks()%>**"**

id=**"book"** type=**"database.BookDetails"**>

**Теги с телом**

Пользовательский тег между своими началом и концом может содержать пользовательский или базовый тег, элементы сценариев, HTML-текст и зависящее от тега тело.

В следующем примере страница showcart.jsp приложения Duke's Bookstore использует тег logic:present библиотеки Struts для очистки корзины покупок и вывода сообщения, если запрос имеет параметр с именем Clear:

<logic:present parameter=**"Clear"**>

<% cart.clear(); %>

<font color=**"#ff0000"** size=**"+2"**><strong>

**You just cleared your shopping cart!**

</strong><br>&nbsp;<br></font>

</logic:present>

**Выбор между передачей информации в виде атрибутов или тела**

Как показано в двух последних разделах, возможна передача определенной части данных как атрибута тега или как тела тега. Вообще говоря, любые данные, которые являются простой строкой или могут быть сгенерированы путем вычисления простого выражения, лучше передавать как атрибут.

**Теги, определяющие переменные сценариев**

Пользовательский тег может определить переменную, которая может использоваться в сценариях внутри страницы. В следующем примере показано, как определить и использовать переменную сценариев, которая содержит объект, возвращенный из JNDI-поиска. Примерами таких объектов могут быть корпоративные компоненты, транзакции, базы данных, переменные окружения и т.д.:

<tt:lookup id=**"tx"** type=**"UserTransaction"**

name=**"java:comp/UserTransaction"** />

<% tx.begin(); %>

В приложении Duke's Bookstore несколько страниц используют ориентированные на компоненты теги из библиотеки Struts для определения переменных сценариев. Например, bookdetails.jsp использует тег bean:parameter для создания переменной сценария bookId и установки ее в значение параметра запроса bookId. Оператор jsp:setProperty также устанавливает свойство bookId объекта bookDB в значение параметра запроса bookId. Тег bean:define извлекает значение свойства базы данных книжного магазина bookDetails и определяет результат в виде переменной сценария book:

<bean:parameter id=**"bookId"** name=**"bookId"** />

<jsp:setProperty name=**"bookDB"** property=**"bookId"**/>

<bean:define id=**"book"** name=**"bookDB"** property=**"bookDetails"**

type=**"database.BookDetails"**/>

<h2><jsp:getProperty name=**"book"** property=**"title"**></h2>

**Взаимодействующие теги**

Пользовательские теги могут взаимодействовать между собой через общие объекты. В следующем примере tag1 создает объект с именем obj1, который затем повторно используется тегом tag2.

<tt:tag1 attr1=**"obj1"** value1=**"value"** />

<tt:tag2 attr1=**"obj1"** />

В следующем примере объект, созданный внешним тегом группы вложенных тегов, доступен всем внутренним тегам. Поскольку объект не поименован, уменьшается вероятность конфликтов имен. В этом примере показывается, как набор взаимодействующих вложенных тегов может выглядеть в JSP-странице.

<tt:outerTag>

<tt:innerTag />

</tt:outerTag>

Страница template.jsp приложения Duke's Bookstore использует набор взаимодействующих тегов для определения экрана приложения. Эти теги рассмотрены в разделе Библиотека шаблонных тегов.

Для определения тега необходимо:

* Разработать обработчик тега и вспомогательный класс для тега
* Объявить тег в дескрипторе библиотеки тегов

В этом разделе рассмотрены свойства обработчиков тегов и TLD, а также объясняется, как создавать обработчики тегов и элементы дескриптора библиотеки для каждого типа тега, описанного в предыдущем разделе.

**Обработчики тегов**

Обработчик тега представляет собой объект, вызываемый Web-контейнером для вычисления пользовательского тега во время выполнения JSP-страницы, которая ссылается на тег. Обработчики тегов должны реализовывать интерфейсы либо Tag, либо BodyTag. Интерфейсы могут использоваться для существующего объекта Java, чтобы преобразовать его в обработчик тега. Для вновь созданных обработчиков можно использовать классы TagSupport и BodyTagSupport в качестве базовых классов. Эти классы и интерфейсы содержатся в пакете javax.servlet.jsp.tagext.

Методы обработчика тега, определенные интерфейсами Tag и BodyTag, вызываются сервлетом JSP-страницы на различных этапах вычисления тега. При встрече начального тега пользовательского тега сервлет JSP-страницы вызывает методы для инициализации соответствующего обработчика и затем вызывает метод обработчика doStartTag. При встрече конечного тега пользовательского тега вызывается метод обработчика doEndTag. Между этими моментами, если обработчик тега требует взаимодействия с телом тега, вызываются дополнительные методы. Дополнительная информация находится в разделе Как вызывается обработчик тега? Для того, чтобы реализовать обработчик тега, необходимо реализовать методы, перечисленные в таблице 1 и вызываемые на различных этапах обработки тега.

Обработчик тега имеет доступ к API, позволяющему взаимодействовать с JSP-страницей. Входной точкой в API является объект контекста страницы (javax.servlet.jsp.PageContext), через который обработчик тега может извлечь все другие внутренние объекты (запрос, сессию и приложение), доступные из JSP-страницы.

Внутренние объекты могут иметь связанные с ними поименованные атрибуты. Такие атрибуты доступны через методы [set|get]Attribute.

Если тег является вложенным, обработчик тега имеет также доступ к обработчику (называемому родитель), связанному с внешним тегом.

Таблица 1 Методы обработчика тега

|  |  |
| --- | --- |
| Тип обработчика тега | Методы |
| Простой | doStartTag, doEndTag, release |
| Атрибуты | doStartTag, doEndTag, set/getAttribute1...N, release |
| Тело, вычисление, без взаимодействия | doStartTag, doEndTag, release |
| Тело, повторяющиеся вычисления | doStartTag, doAfterBody, doEndTag, release |
| Тело, взаимодействие | doStartTag, doEndTag, release, doInitBody, doAfterBody, release |

Набор связанных классов обработчиков тегов (библиотека тегов) обычно пакетируется и размещается в виде JAR-архива.

**Дескрипторы библиотек тегов**

Дескриптор библиотеки тегов (TLD) представляет собой XML-документ, описывающий библиотеку тегов. TLD содержит общую информацию и информацию о каждом теге, содержащемся в библиотеке. TLD используется Web-контейнером для проверки тегов, а также средствами разработки JSP-страницы.

Названия TLD-файлов должны иметь расширение .tld. TLD-файлы хранятся в каталоге WEB-INF файла WAR или подкаталогах каталога WEB-INF. Когда вы добавляете TLD в WAR при помощи программы deploytool, она автоматически записывает его в WEB-INF.

TLD должен начинаться прологом XML-документа, указывающим версию XML и определение типа документа (DTD):

<?xml version=**"1.0"** encoding=**"ISO-8859-1"** ?>

<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"

"http://java.sun.com/dtd/web-jsptaglibrary\_1\_2.dtd">

J2EE SDK версии 1.3 может понимать DTD версий 1.1 и 1.2. Однако эта глава рассматривает версию 1.2, поскольку вы должны использовать более новую версию в любых библиотеках тегов, которые вы разрабатываете. TLD библиотеки шаблонов, tutorial-template.tld, согласован с версией 1.2. TLD библиотеки Struts согласован с версией 1.1 DTD, имеющей меньше элементов и использующей немного другие имена для некоторых элементов.

Корнем TLD является элемент taglib. Субэлементы taglib перечислены в таблице 2:

Таблица 2 Субэлементы taglib

| Элемент | Описание |
| --- | --- |
| tlib-version | Версия библиотеки тегов. |
| jsp-version | Версия спецификации JSP, требуемой для библиотеки тегов. |
| short-name | Необязательное имя, которое может использоваться авторскими средствами разработки JSP-страниц для создания имен с мнемоническим значением. |
| uri | URI, уникально идентифицирующий библиотеку тегов. |
| display-name | Необязательное имя, отображаемое инструментальными средствами. |
| small-icon | Необязательная маленькая иконка, которая может использоваться инструментальными средствами. |
| large-icon | Необязательная большая иконка, которая может использоваться инструментальными средствами. |
| description | Необязательная информация, относящаяся к тегу. |
| listener | См. раздел Элемент-перехватчик. |
| tag | См. раздел Элемент tag. |

**Элемент-перехватчик**

Библиотека тегов может определить некоторые классы, являющиеся перехватчиками событий (см. раздел Обработка событий цикла жизни сервлета). Перехватчики перечислены в TLD как элементы listener. Web-контейнер будет создавать экземпляры классов перехватчиков и регистрировать их способом, аналогичным способу, применяемому для перехватчиков, определенных на уровне WAR.

**Элемент tag**

Каждый тег в библиотеке описывается своим именем и классом своего обработчика тега, информацией о переменных сценариев, создаваемых тегом, и информацией об атрибутах тега. Информация о переменных сценариев может записываться прямо в TLD или при помощи класса дополнительной информации (см. раздел Теги, определяющие переменные сценариев). Каждое объявление атрибута содержит информацию о том, требуется ли атрибут, может ли его значение определяться в выражениях времени исполнения, а также тип атрибута (см. раздел Теги с атрибутами).

Тег указывается в TLD в элементе tag. Субэлементы tag перечислены в таблице 3:

Таблица 3 Субэлементы tag

|  |  |
| --- | --- |
| Элемент | Описание |
| name | Уникальное имя тега. |
| tag-class | Полное имя класса перехватчика тега. |
| tei-class | Необязательный подкласс javax.servlet.jsp.tagext.TagExtraInfo. См. раздел Класс TagExtraInfo. |
| body-content | Тип содержимого тела. См. разделы Элемент body-content и Элемент body-content. |
| display-name | Необязательное имя, отображаемое инструментальными средствами |
| small-icon | Необязательная маленькая иконка, которая может использоваться инструментальными средствами. |
| large-icon | Необязательная большая иконка, которая может использоваться инструментальными средствами. |
| description | Необязательная информация, относящаяся к тегу. |
| variable | Необязательная информация о переменной сценария. См. раздел Элемент variable. |
| attribute | Информация об атрибуте тега. См. раздел Элемент attribute. |

В следующем разделе рассмотрены методы и элементы TLD, которые вы должны разработать для каждого типа тега, описанного в разделе Использование тегов.

**Простые теги**

**Обработчики тегов**

**Обработчик** для простого тега должен реализовывать методы doStartTag и doEndTag интерфейса Tag. Метод doStartTag вызывается при обработке начального тега. Этот метод возвращает SKIP\_BODY, поскольку простой тег не имеет тела. Метод doEndTag вызывается при обработке конечного тега. Метод doEndTag должен возвращать EVAL\_PAGE, если оставшаяся часть страницы должна быть вычислена; в противном случае, он должен возвращать SKIP\_PAGE.

Простой тег, рассмотренный в первом разделе,

<tt:simple />

может быть реализован следующим обработчиком тега:

**public**SimpleTag **extends**TagSupport {  
   **public int**doStartTag() **throws**JspException {  
      **try**{  
         pageContext.getOut().print("Hello.");  
      } **catch**(Exception ex) {  
         **throw new**JspTagException("SimpleTag: " +  
            ex.getMessage());  
      }  
      **return**SKIP\_BODY;  
   }  
   **public int**doEndTag() {  
      **return**EVAL\_PAGE;  
   }  
}

**Элемент body-content**

Тег, не имеющий тела, должен объявить, что содержимое его тела пусто, при помощи элемента body-content:

<body-content>**empty**</body-content>

**Теги с атрибутами**

**Определение атрибутов в обработчике тега**

Для каждого атрибута тега необходимо в обработчике тега определить свойство и методы get и set, которые согласованы с требованиями архитектуры JavaBeans. Например, обработчик тега для тега logic:present библиотеки Struts,

<logic:present parameter="Clear">

содержит следующие объявления и методы:

**protected**String parameter = **null**;  
  
**public**String getParameter() {  
   **return**(**this**.parameter);  
}  
**public void**setParameter(String parameter) {  
   **this**.parameter = parameter;  
}

Обратите внимание, что если атрибут имеет имя id и обработчик тега наследует класс TagSupport, необязательно определять свойство и методы get и set, поскольку они уже определены в TagSupport.

Атрибут тега, имеющий значение String, может называть атрибут одного из внутренних объектов, доступных обработчикам тегов. Атрибут внутреннего объекта может быть доступен при передаче значения атрибута тега в метод внутреннего объекта [set|get]Attribute. Это хороший способ передачи имен переменных сценариев в обработчик тега, в котором они связываются с объектами, хранящимися в контексте страницы (см. раздел Теги, определяющие переменные сценариев).

**Элемент attribute**

Для каждого атрибута тега необходимо указать информацию, требуется ли атрибут, может ли его значение определяться выражениями, и, необязательно, тип атрибута в элементе attribute. Для статических значений тип всегда равен java.lang.String. Если элемент rtexprvalue установлен в true или yes, элемент type определяет тип, ожидаемый от любого выражения, указанного в качестве значения атрибута.

<attribute>

<name>**attr1**</name>

<required>**true|false|yes|no**</required>

<rtexprvalue>**true|false|yes|no**</rtexprvalue>

<type>**fully\_qualified\_type**</type>

</attribute>

Если атрибут тега не требуется, обработчик тега должен обеспечить значение по умолчанию.

Элемент tag для тега logic:present объявляет, что атрибут parameter не требуется (поскольку тег может также проверить наличие других сущностей, таких как свойства компонента) и что его значение может быть установлено выражением времени исполнения.

<tag>

<name>**present**</name>

<tag-class>**org.apache.struts.taglib.**

**logic.PresentTag**</tag-class>

<body-content>**JSP**</body-content>

**...**

<attribute>

<name>**parameter**</name>

<required>**false**</required>

<rtexprvalue>**true**</rtexprvalue>

</attribute>

**...**

</tag>

**Проверка атрибута**

Документация для библиотеки тегов должна описывать правильные значения для атрибутов тегов. При трансляции JSP-страницы Web-контейнер применит любые ограничения, содержащиеся в элементе TLD для каждого атрибута.

Переданные в тег атрибуты могут также быть протестированы во время трансляции при помощи метода isValid класса, наследованного от TagExtraInfo. Этот класс также используется для предоставления информации о переменных сценариев, определенных тегом (см. раздел Теги, определяющие переменные сценариев).

Методу isValid передается информация об атрибуте в объекте TagData, который содержит пары атрибут-значение для каждого из атрибутов тега. Поскольку проверка осуществляется во время трансляции, значение атрибута, вычисляемое во время запроса, будет установлено в TagData.REQUEST\_TIME\_VALUE.

Тег <tt:twa attr1="value1"/> имеет следующий элемент TLD attribute:

<attribute>

<name>**attr1**</name>

<required>**true**</required>

<rtexprvalue>**true**</a>

</attribute>

Это объявление указывает, что значение attr1 может определяться во время исполнения.

Следующий метод isValid проверяет, что значение attr1 является корректным значением Boolean. Обратите внимание, что, поскольку значение attr1 может быть вычислено во время исполнения, isValid должен проверять, был ли выбран пользователь тега для предоставления значения времени исполнения.

**public class**TwaTEI **extends**TagExtraInfo {  
  **public boolean**isValid(Tagdata data) {  
    Object o = data.getAttribute("attr1");  
    **if**(o != **null**&& o != TagData.REQUEST\_TIME\_VALUE) {  
      **if**(o.toLowerCase().equals("true")  
          || o.toLowerCase().equals("false"))  
        **return true**;  
      **else  
        return false**;  
    } **else  
      return true**;  
  }  
}

**Теги с телом**

**Обработчики тегов**

Обработчик тега для тега с телом реализуется по-разному, в зависимости от того, должен ли обработчик тега взаимодействовать с телом или нет. Под словом взаимодействовать понимается, может ли обработчик тега изменять содержимое тела.

**Обработчик тега не взаимодействует с телом**

Если обработчику тега не нужно взаимодействовать с телом, он должен реализовывать интерфейс Tag (или порождаться от TagSupport). Если тело тега нужно вычислять, метод doStartTag должен возвращать EVAL\_BODY\_INCLUDE; в противном случае, он должен возвращать SKIP\_BODY.

Если обработчик тега должен последовательно вычислять тело, он должен реализовывать интерфейс IterationTag или порождаться из TagSupport. Он должен возвращать EVAL\_BODY\_AGAIN из методов doStartTag и doAfterBody, если определит, что тело должно вычслиться повторно.

**Обработчик тега не взаимодействует с телом**

Если обработчику тега нужно взаимодействовать с телом, обработчик тега должен реализовывать BodyTag (или порождаться из BodyTagSupport). Такие обработчики обычно реализуют методы doInitBody и doAfterBody. Эти методы взаимодействуют с содержимым тела, переданного в обработчик тега сервлетом JSP-страницы.

Содержимое тела поддерживает несколько методов чтения и записи своего содержимого. Обработчик тега может использовать методы содержимого тела getString или getReader для извлечения информации из тела, и метод writeOut(out) для записи содержимого тела в выходной поток. Объект out, предоставленный методу writeOut, берется при помощи метода обработчика тега getPreviousOut. Этот метод используется для проверки того, что результаты обработчика тега доступны внешнему обработчику тега.

Если тело тега должно быть вычислено, метод doStartTag должен возвратить EVAL\_BODY\_BUFFERED; в противном случае, он должен возвратить SKIP\_BODY.

Метод doInitBody

Метод doInitBody вызывается после установки содержимого тела, но перед тем как оно будет вычисляться. Этот метод обычно используется для выполнения любой инициализации, зависящей от содержимого тела.

Метод doAfterBody

Метод doAfterBody вызывается после вычисления содержимого тела. Аналогично методу doStartTag метод doAfterBody должен возвратить признак, продолжать ли вычисление тела. То есть, если тело должно быть вычислено снова, например, в случае реализации итерационного тега, doAfterBody должен возвратить EVAL\_BODY\_BUFFERED; в противном случае, doAfterBody должен возвратить SKIP\_BODY.

Метод release

В методе release обработчик тега должен сбросить свое состояние и освободить все собственные ресурсы.

В следующем примере читается содержимое тела (содержащее SQL-запрос) и передается в объект, выполняющий запрос. Поскольку тело не должно быть вычислено повторно, doAfterBody возвращает SKIP\_BODY.

**public class**QueryTag **extends**BodyTagSupport {  
  **public int**doAfterBody() **throws**JspTagException {  
    BodyContent bc = getBodyContent();  
    // получить bc в виде строки  
    String query = bc.getString();  
    // очистка  
    bc.clearBody();  
    **try**{  
      Statement stmt = connection.createStatement();  
      result = stmt.executeQuery(query);  
    } **catch**(SQLException e) {  
      **throw new**JspTagException("QueryTag: " + e.getMessage());  
    }  
    **return**SKIP\_BODY;  
  }  
}

**Элемент body-content**

Для тегов с телом необходимо указывать тип содержимого тела при помощи элемента body-content:

<body-content>**JSP|tagdependent**</body-content>

Содержимое тела, содержащее пользовательские и базовые теги, элементы сценария и HTML-текст, отмечается как JSP. Это значение объявляется для тега logic:present библиотеки Struts. Все другие типы содержимого тела - например, SQL-операторы, переданные в тег запроса, - должны быть отмечены как tagdependent.

Обратите внимание, что значение элемента body-content не влияет на интерпретацию тела обработчиком тега; этот элемент намечалось использовать в авторских инструментальных средствах для отображения содержимого тела.

**Теги, определяющие переменные сценариев**

**Обработчики тега**

Обработчик тега ответственен за создание и установку объектов, на которые ссылается переменная сценариев в контексте, доступном из страницы. Это осуществляется при помощи методов pageContext.setAttribute(name, value, scope) или pageContext.setAttribute(name, value). Обычно атрибут, переданный в пользовательский тег, указывает имя объекта переменной сценария; это имя может быть извлечено при помощи вызова метода атрибута get, описанного в разделе Определение атрибутов в обработчике тега.

Если значение переменной сценария зависит от объекта, находящегося в контексте обработчика тега, оно может быть извлечено при помощи метода pageContext.getAtribute(name, scope).

Обычной процедурой является извлечение переменной сценария, выполнение некоторой обработки объекта и установка значения переменной сценария при помощи метода pageContext.setAttribute(name,object).

Области видимости, которые может иметь объект, перечислены в таблице 4. Область видимости ограничивает доступность и время жизни объекта.

Таблица 4 Область видимости объектов

| Имя | Доступен из | Время жизни |
| --- | --- | --- |
| page | Текущей страницы | Пока ответ не будет послан пользователю или запрос не будет передан на новую страницу |
| request | Текущей страницы и любых включенных или перенаправленных страниц | Пока ответ не будет послан пользователю |
| session | Текущего запроса и любого последующего запроса из того же самого броузера (субъект времени жизни сессии) | Время жизни сессии пользователя |
| application | Текущего и любого будущего запроса из того же Web-приложения | Время жизни приложения |

**Предоставление информации о переменной сценария**

Пример, описанный в разделе Теги, определяющие переменные сценариев, определяет переменную сценария book, используемую для доступа к информации о книге:

<bean:define id=**"book"** name=**"bookDB"** property=**"bookDetails"**

type=**"database.BookDetails"**/>

<font color=**"red"** size=**"+2"**>

<%=messages.getString("CartRemoved")%>

<strong><jsp:getProperty name=**"book"**

property=**"title"**/></strong>

<br>&nbsp;<br>

</font>

При трансляции JSP-страницы, содержащей этот тег, Web-контейнер генерирует код для синхронизации переменной сценария с объектом, на который ссылается эта переменная. Для генерации кода Web-контейнер требует определенной информации о переменной сценария:

* Имя переменной
* Класс переменной
* Обращается ли переменная к новому или к существующему объекту
* Доступность переменной

Существует два способа предоставить эту информацию: указывая субэлемент TLD variable, или определяя класс дополнительной информации и включая элемент tei-class в TLD. Использование элемента variable проще, но немного менее гибко.

**Элемент variable**

Элемент variable имеет следующие субэлементы:

* name-given: Имя переменной в виде константы.
* name-from-attribute: Имя атрибута, чье значение времени трансляции даст имя переменной.

Требуется один из элементов name-given или name-from-attribute. Следующие субэлементы необязательны:

* variable-class: Полностью указанное имя класса переменной. По умолчанию java.lang.Srtring.
* declare: Обращается ли переменная к новому объекту. По умолчанию - true.
* scope: Определенная область действия переменной сценария. По умолчанию - NESTED. В таблице 13-5 описывается доступность переменной сценария и методы, в которых значение переменной должно быть установлено или восстановлено.

Реализация тега bean:define библиотеки Struts соответствует спецификации JSP версии 1.1, которая требует определения класса дополнительной информации. Спецификация JSP версии 1.2 добавляет элемент variable. Вы могли бы определить следующий элемент variable для тега bean:define:

<tag>

<variable>

<name-from-attribute>**id**</name-from-attribute>

<variable-class>**database.BookDetails**</variable-class>

<declare>**true**</declare>

<scope>**AT\_BEGIN**</scope>

</variable>

</tag>

Таблица 5 Доступность переменных сценария

| Значение | Доступность | Методы |
| --- | --- | --- |
| NESTED | Между начальным тегом и конечным тегом | В doInitBody и doAfterBody для обработчика тега, реализующего BodyTag; в противном случае, в doStartTag |
| AT\_BEGIN | От начального тега до конца страницы | В doInitBody, doAfterBody и doEndTag для обработчика тега, реализующего BodyTag; в противном случае, в doStartTag и doEndTag |
| AT\_END | После конечного тега до конца страницы | В doEndTag |

**Класс TagExtraInfo**

Класс тега дополнительной информации определяется расширением класса javax.servlet.jsp.TagExtraInfo. Для возврата массива объектов VariabeInfo TagExtraInfo должен реализовывать метод getVariableInfo, содержащий следующую информацию:

* Имя переменной
* Класс переменной
* Обращается ли переменная к новому или к существующему объекту
* Доступность переменной

Web-контейнер передает параметр, называемый data, в метод getVariableInfo, содержащий пары атрибут-значение для каждого из атрибутов тега. Эти атрибуты могут быть использованы для обеспечения объекта VariableInfo именем переменной сценария и классом.

Библиотека тегов Struts предоставляет информацию о переменной сценария, созданной тегом bean:define в теге DefineTei класса дополнительной информации. Поскольку имя (book) и класс (database.BookDetails) переменной сценария передается как атрибут тега, они могут быть извлечены методом data.getAttributeString и использованы для заполнения в конструкторе VariableInfo. Для разрешения использования переменной сценария book в оставшейся части страницы область видимости book установлена в AT\_BEGIN.

**public class**DefineTei **extends**TagExtraInfo {  
  **public**VariableInfo[] getVariableInfo(TagData data) {  
    String type = data.getAttributeString("type");  
    **if**(type == **null**)  
      type = "java.lang.Object";  
    **return new**VariableInfo[] { **new**VariableInfo(data  
        .getAttributeString("id"), type, true, VariableInfo.AT\_BEGIN) };  
  }  
}

Полностью указанное имя класса тега дополнительной информации, определенное для переменной сценария, должно быть объявлено в субэлементе TLD tei-class элемента tag. То есть, элемент tei-class для DefineTei может быть следующим:

<tei-class>**org.apache.struts.taglib.bean.DefineTagTei**

</tei-class>

**Взаимодействующие теги**

Теги взаимодействуют при помощи разделения объектов. Технология JSP поддерживает два стиля совместного доступа к объекту. Первый стиль требует, чтобы общий объект был поименован и сохранен в контексте страницы (один из внутренних объектов доступен и в JSP-страницах и в обработчиках тегов). Для обращения к объектам, созданным и поименованным в другом теге, обработчик тега использует метод pageContext.getAttribute(name, scope).

При втором стиле совместного доступа к объекту объект, созданный внешним обработчиком тега группы вложенных тегов, доступен всем внутренним обработчикам тегов. Эта форма совместного доступа к объекту имеет преимущество в том, что использует собственное пространство имен для объектов, уменьшая, таким образом, вероятность конфликта имен.

Для доступа к объекту, созданному во внешнем теге, обработчик тега должен сначала получить свой внешний тег при помощи статического метода TagSupport.findAncestorWithClass(from, class) или метода TagSupport.getParent. Первый метод должен использоваться, когда определенное вложение обработчиков тегов не может быть гарантировано. Как только был получен предок, обработчик тега может обратиться к любым статически или динамически созданным объектам. Статически созданные объекты являются членами родителя. Собственные объекты могут также быть созданы динамически. Такие объекты могут быть сохранены в обработчике тега при помощи метода setValue и извлечены при помощи метода getValue.

В следующем примере показан обработчик тега, поддерживающий оба подхода к совместному доступу к объектам: и именованный, и собственный объект. В примере обработчик тега запроса поверяет, установлен ли атрибут с именем connection в методе doStartTag. Если атрибут connection установлен, обработчик извлекает объект connection из контекста страницы. В противном случае обработчик сначала извлекает обработчик тега внешнего тега, а затем извлекает объект connection из этого обработчика.

**public class**QueryTag **extends**BodyTagSupport {  
  **private**String connectionId;  
  
  **public int**doStartTag() **throws**JspException {  
      String cid = getConnection();  
      **if**(cid != **null**) {  
      // существует id соединения, использовать его  
         connection =(Connection)pageContext.  
            getAttribute(cid);  
      } **else**{  
         ConnectionTag ancestorTag =  
            (ConnectionTag)findAncestorWithClass(this,  
               ConnectionTag.**class**);  
         **if**(ancestorTag == **null**) {  
            **throw new**JspTagException("A query without  
               a connection attribute must be nested  
               within a connection tag.");  
         }  
         connection = ancestorTag.getConnection();  
      }  
   }  
}

Тег запроса, реализованный этим обработчиком тега, может быть использован одним из следующих способов:

<tt:connection id=**"con01"** ....> **...** </tt:connection>

<tt:query id=**"balances"** connection=**"con01"**>

**SELECT account, balance FROM acct\_table**

**where customer\_number =** <%= request.getCustno()%>

</tt:query>

<tt:connection ...>

<x:query id=**"balances"**>

**SELECT account, balance FROM acct\_table**

**where customer\_number =** <%= request.getCustno()%>

</x:query>

</tt:connection>

TLD для обработчика тега должен указать, что атрибут connection является необязательным, при помощи следующего объявления:

<tag>

**...**

<attribute>

<name>**connection**</name>

<required>**false**</required>

</attribute>

</tag>

Описанные ниже пользовательские теги демонстрируют решение двух проблем рекуррентности в разработке JSP-приложений: минимизация количества программирования на Java в JSP-страницах и гарантирование цельности и понимания приложения. Для этого примеры демонстрируют многие из стилей тегов, рассмотренных в первой части главы.

**Итерационный тег**

Создание содержимого страницы, которое зависит от динамически сгенерированных данных, часто требует использования операторов сценария управления передачей. При перемещение логики управления передачей в обработчики тегов теги управления передачей уменьшают количество сценариев, требуемых для JSP-страниц.

Тег logic:iterate библиотеки Struts извлекает объекты из коллекции, хранимой в компоненте JavaBeans, и назначает их переменным сценария. Тело тега извлекает информацию из переменной сценария. Пока элемент коллекции остается в коллекции, тег iterate вызывает повторное вычисление тела.

**JSP-страница**

Две страницы приложения Duke's Bookstore catalog.jsp и showcart.jsp для перебора объектов коллекции используют тег logic:iterate. Выдержка из catalog.jsp приведена ниже. JSP-страница инициализирует тег iterate коллекцией (указанной в атрибуте property) компонентов bookDB. Тег iterate устанавливает переменную book на каждой итерации. Свойство bookId переменной book выражено как еще одна переменная сценария. Свойства обеих переменных используются для динамической генерации таблицы, содержащей ссылки к другим страницам и информации о каталоге книг.

<logic:iterate name=**"bookDB"** property=**"books"**

id=**"book"** type=**"database.BookDetails"**>

<bean:define id=**"bookId"** name=**"book"** property=**"bookId"**

type=**"java.lang.String"**/>

<tr>

<td bgcolor=**"#ffffaa"**>

<a href=**"**<%=request.getContextPath()%>

**/bookdetails?bookId=**<%=bookId%>**"**>

<strong><jsp:getProperty name=**"book"**

property=**"title"**/>&nbsp;</strong></a></td>

<td bgcolor=**"#ffffaa"** rowspan=2>

<jsp:setProperty name=**"currency"** property=**"amount"**

value=**"**<%=book.getPrice()%>**"**/>

<jsp:getProperty name=**"currency"** property=**"format"**/>

&nbsp;</td>

<td bgcolor=**"#ffffaa"** rowspan=2>

<a href=**"**<%=request.getContextPath()%>

**/catalog?Add=**<%=bookId%>**"**>

&nbsp;<%=messages.getString("CartAdd")%>

&nbsp;</a></td></tr>

<tr>

<td bgcolor=**"#ffffff"**>

&nbsp;&nbsp;<%=messages.getString("By")%><em>

<jsp:getProperty name=**"book"**

property=**"firstName"**/>&nbsp;

<jsp:getProperty name=**"book"**

property=**"surname"**/></em></td></tr>

</logic:iterate>

Обработчик тега

Реализация тега logic:iterate библиотеки Struts следует спецификации JSP версии 1.1, которая требует расширения класса BodyTagSupport. Спецификация JSP версии 1.2 добавляет возможности (описанные в разделе Обработчик тега, не взаимодействующий с телом), упрощающие программирование тегов, которые последовательно вычисляют свое тело. Нижеследующее обсуждение основано на реализации, использующей эти возможности.

Тег logic:iterate поддерживает инициализацию коллекции несколькими путями: из коллекции, предоставленной в виде атрибута тега, или из коллекции, являющейся компонентом или свойством компонента. Наш пример использует последний метод. Большая часть кода в doStartTag связана с построением итератора над объектом коллекции. Метод сначала проверяет, установлено ли свойство обработчика collection и, если нет, продолжает работу, проверяя атрибуты bean и property. Если оба атрибута bean и property установлены, doStartTag вызывает служебный метод, использующий методы интроспекции JavaBeans для извлечения коллекции. Как только объект коллекции определяется, метод создает итератор.

Если итератор содержит еще какие-либо элементы, doStartTag устанавливает значение переменной сценария в следующий элемент и указывает, что тело должно быть вычислено повторно; в противном случае, он заканчивает итерацию, возвратив SKIP\_BODY.

После вычисления тела метод doAfterBody извлекает содержимое тела и записывает его в выходной поток. Объект содержимого тела затем очищается для подготовки к следующему вычислению тела. Если итератор содержит еще какие-либо элементы, doAfterBody опять устанавливает значение переменной сценария в следующий элемент и возвращает EVAL\_BODY\_AGAIN для указания необходимости повторного вычисления тела. Это вызывает повторное выполнение doAfterBody. Если элементов больше нет, doAfterBody завершает процесс, возвращая SKIP\_BODY.

**public class**IterateTag **extends**TagSupport {  
   **protected**Iterator iterator = **null**;  
   **protected**Object collection = **null**;  
   **protected**String id = **null**;  
   **protected**String name = **null**;  
   **protected**String property = **null**;  
   **protected**String type = **null**;  
   **public int**doStartTag() **throws**JspException {  
      Object collection = **this**.collection;  
      **if**(collection == **null**) {  
         **try**{  
            Object bean = pageContext.findAttribute(name);  
            **if**(bean == **null**) {  
               ... **throw**an exception  
            }  
            **if**(property == **null**)  
               collection = bean;  
            **else**collection =  
                  PropertyUtils.  
                     getProperty(bean, property);  
            **if**(collection == **null**) {  
               ... **throw**an exception  
            }  
         } **catch**... **catch**exceptions thrown  
               by PropertyUtils.getProperty  
         }  
      }  
      // Создать итератор для этой коллекции  
      **if**(collection **instanceof**Collection)  
         iterator = ((Collection) collection).iterator();  
      **else if**(collection **instanceof**Iterator)  
         iterator = (Iterator) collection;  
         ...  
      }  
      // Сохранить первое значение и вычислить,  
      // или пропустить тело, если нет элементов  
      **if**(iterator.hasNext()) {  
         Object element = iterator.next();  
         pageContext.setAttribute(id, element);  
         **return**(EVAL\_BODY\_AGAIN);  
      } **else  
         return**(SKIP\_BODY);  
}  
   **public int**doAfterBody() **throws**JspException {  
      **if**(bodyContent != **null**) {  
         **try**{  
            JspWriter out = getPreviousOut();  
            out.print(bodyContent.getString());  
            bodyContent.clearBody();  
         } **catch**(IOException e) {  
            ...  
         }  
      }  
      **if**(iterator.hasNext()) {  
         Object element = iterator.next();  
         pageContext.setAttribute(id, element);  
         **return**(EVAL\_BODY\_AGAIN);  
      } **else  
         return**(SKIP\_BODY);  
      }  
   }  
}

**Класс дополнительной информации тега**

Информация о переменной сценария предоставляется в классе дополнительной информации тега IterateTei. Имя и класс переменной сценария передается в виде атрибута тега и используется для заполнения в конструкторе VariableInfo.

**public class**IterateTei **extends**TagExtraInfo {  
  **public**VariableInfo[] getVariableInfo(TagData data) {  
    String type = data.getAttributeString("type");  
    **if**(type == **null**)  
      type = "java.lang.Object";  
  
    **return new**VariableInfo[] { **new**VariableInfo(data  
        .getAttributeString("id"), type, true, VariableInfo.AT\_BEGIN) };  
  }  
}

**Библиотека шаблонных тегов**

Шаблоны обеспечивают способ отделения общих элементов, являющихся частью каждого экрана, от элементов, меняющихся в каждом экране приложения. Расположение всех общих элементов вместе в одном файле облегчает поддержку и гарантирует цельность и понимание всех экранов. Это также упрощает разработку индивидуальных экранов, потому что дизайнер может сосредоточиться на части экрана, специфичной для этого экрана, в то время, когда шаблон позаботится об общей части.

Шаблон представляет собой JSP-страницу со знаками-заместителями вместо частей, которые меняются в каждом экране. Каждый из этих знаков-заместителей является параметром шаблона. Например, простой шаблон может содержать параметр заголовка для верхней части экрана и параметр тела для ссылки на JSP-страницу, определяющую индивидуальное содержимое экрана.

Шаблон использует набор вложенных тегов - definition, screen и parameter - для определения таблицы определения экрана для экрана приложения и использует тег insert для вставки параметров из определения экрана в экран приложения.

**JSP-страница**

Шаблон template.jsp примера Duke's Bookstore показан на следующей странице. Эта страница включает JSP-страницу, создающую определение экрана, и затем использующую тег insert для вставки параметров из определения в экран приложения.

<%@ taglib uri="/tutorial-template.tld" prefix="tt" %>

<%@ page errorPage="errorpage.jsp" %>

<%@ include file="screendefinitions.jsp" %>

<html>

<head>

<title>

<tt:insert definition=**"bookstore"**

parameter=**"title"**/>

</title>

</head>

<tt:insert definition=**"bookstore"**

parameter=**"banner"**/>

<tt:insert definition=**"bookstore"**

parameter=**"body"**/>

</body>

</html>

screendefinition.jsp создает определение экрана, основанное на атрибуте запроса selectedScreen:

<tt:definition name=**"bookstore"**

screen=**"**<%= (String)request.

getAttribute(\"selectedScreen\") %>**"**>

<tt:screen id=**"/enter"**>

<tt:parameter name=**"title"**

value=**"Duke's Bookstore"** direct=**"true"**/>

<tt:parameter name=**"banner"**

value=**"/banner.jsp"** direct=**"false"**/>

<tt:parameter name=**"body"**

value=**"/bookstore.jsp"** direct=**"false"**/>

</tt:screen>

<tt:screen id=**"/catalog"**>

<tt:parameter name=**"title"**

value=**"**<%=messages.getString("TitleBookCatalog")%>**"**

direct=**"true"**/>

**...**

</tt:definition>

Экземпляр шаблона создается сервлетом Dispatcher. Dispatcher сначала получает запрошенный экран и сохраняет его как атрибут запроса. Это необходимо, потому что когда запрос перенаправляется в template.jsp, URL запроса не содержит оригинальный запрос (например, /bookstore3/catalog), а вместо этого отображает путь к перенаправленной странице (/bookstore3/template.jsp). Наконец, сервлет направляет запрос в template.jsp:

**public class**Dispatcher **extends**HttpServlet {  
  **public void**doGet(HttpServletRequest request, HttpServletResponse response) {  
    request.setAttribute("selectedScreen", request.getServletPath());  
    RequestDispatcher dispatcher = request  
        .getRequestDispatcher("/template.jsp");  
    **if**(dispatcher != **null**)  
      dispatcher.forward(request, response);  
  }  
  
  **public void**doPost(HttpServletRequest request, HttpServletResponse response) {  
    request.setAttribute("selectedScreen", request.getServletPath());  
    RequestDispatcher dispatcher = request  
        .getRequestDispatcher("/template.jsp");  
    **if**(dispatcher != **null**)  
      dispatcher.forward(request, response);  
  }  
}

**Обработчики тегов**

Библиотека шаблонных тегов содержит четыре обработчика тегов - DefinitionTag, ScreenTag, ParameterTag и InsertTag - которые демонстрируют использование взаимодействующих тегов. DefinitionTag, ScreenTag, и ParameterTag содержат набор вложенных обработчиков тегов, которые совместно используют общие и частные объекты. DefinitionTag создает общий поименованный объект с названием definition, который используется InsertTag.

В doStartTag DefinitionTag создает общий поименованный объект с названием screens, содержащий хэш-таблицу определений экранов. Определение экрана состоит из идентификатора экрана и набора параметров, связанных с экраном.

**public int**doStartTag() {  
   HashMap screens = **null**;  
   screens = (HashMap) pageContext.getAttribute("screens",  
      pageContext.APPLICATION\_SCOPE);  
   **if**(screens == **null**)  
      pageContext.setAttribute("screens", **new**HashMap(),  
         pageContext.APPLICATION\_SCOPE);  
   **return**EVAL\_BODY\_INCLUDE;  
}

Таблица определений экранов заполняется тегами ScreenTag и ParameterTag из текста, предоставляемого этим тегам как атрибуты. В таблице 13-6 показано содержимое хэш-таблицы определений экранов для приложения Duke's bookstore

Таблица 13-6 Определения экранов

| ScreenId | Title | Banner | Body |
| --- | --- | --- | --- |
| /enter | Duke's Bookstore | /banner.jsp | /bookstore.jsp |
| /catalog | Каталог книг | /banner.jsp | /catalog.jsp |
| /bookdetails | Описание книги | /banner.jsp | /bookdetails.jsp |
| /showcart | Ваша корзина покупок | /banner.jsp | /showcart.jsp |
| /cashier | Кассир | /banner.jsp | /cashier.jsp |
| /receipt | Получение | /banner.jsp | /receipt.jsp |

В doEndTag DefinitionTag создает общий объект класса Definition, выбирает определение экрана из объекта screens, основанного на URL, переданном в запросе, и использует его для инициализации объекта Definition.

**public int**doEndTag()**throws**JspTagException {  
   **try**{  
      Definition definition = **new**Definition();  
      Hashtable screens = **null**;  
      ArrayList params = **null**;  
      TagSupport screen = **null**;  
      screens = (HashMap)  
         pageContext.getAttribute("screens",  
            pageContext.APPLICATION\_SCOPE);  
      **if**(screens != **null**)  
         params = (ArrayList) screens.get(screenId);  
      **else**...  
      **if**(params == **null**)  
         ...  
      Iterator ir = **null**;  
      **if**(params != **null**)  
         ir = params.iterator();  
      **while**((ir != **null**) && ir.hasNext())  
         definition.setParam((Parameter) ir.next());  
         // записать определение в контекст страницы  
      pageContext.setAttribute(  
         definitionName, definition);  
   } **catch**(Exception ex) {  
      ex.printStackTrace();  
   }  
   **return**EVAL\_PAGE;  
}

Если URL, переданный в запросе, равен /enter, Definition содержит элементы из первой строки таблицы 13-6:

| Title | Banner | Body |
| --- | --- | --- |
| Duke's Bookstore | /banner.jsp | /bookstore.jsp |

Определение для URL /enter показано в таблице 7. Определение указывает, что значение параметра Title, Duke's Bookstore, должно быть вставлено непосредственно в выходной поток, а значения Banner и Body должны подключаться динамически.

Таблица 7 Определение экрана для URL /enter

| Имя параметра | Значение параметра | isDirect |
| --- | --- | --- |
| title | Duke's Bookstore | TRUE |
| banner | /banner.jsp | FALSE |
| body | /bookstore.jsp | FALSE |

InsertTag использует Definition для вставки параметров определения экрана в ответ. В методе doStartTag он извлекает объект определения из контекста страницы.

**public int**doStartTag() {  
   // получить определение из контекста страницы  
   definition = (Definition) pageContext.  
      getAttribute(definitionName);  
   // получить параметр  
   **if**(parameterName != **null**&& definition != **null**)  
      parameter = (Parameter)definition.  
         getParam(parameterName);  
   **if**(parameter != **null**)  
      directInclude = parameter.isDirect();  
   **return**SKIP\_BODY;  
}

Метод doEndTag вставляет значение параметра. Если параметр прямой, он непосредственно вставляется в ответ; в противном случае запрос посылается параметру, а ответ динамически подключается к общему ответу.

**public int**doEndTag()**throws**JspTagException {  
   **try**{  
      **if**(directInclude && parameter != **null**)  
         pageContext.getOut().print(parameter.getValue());  
      **else**{  
         **if**((parameter != **null**) &&  
            (parameter.getValue() != **null**))  
            pageContext.include(parameter.getValue());  
      }  
   } **catch**(Exception ex) {  
      **throw new**JspTagException(ex.getMessage());  
   }  
   **return**EVAL\_PAGE;  
}

Интерфейс Tag определяет базовый протокол между обработчиком тега и сервлетом JSP-страницы. Он определяет цикл жизни и методы, которые надо вызвать при обнаружении начального и конечного тегов.

Сервлет JSP-страницы вызывает setPageContext, setParent и методы установки атрибутов перед вызовом doStartTag. Сервлет JSP-страницы также гарантирует, что метод release обработчика тега будет вызван перед концом страницы.

Вот типичная последовательность вызовов методов обработчика тега:

ATag t = **new**ATag();  
t.setPageContext(...);  
t.setParent(...);  
t.setAttribute1(value1);  
t.setAttribute2(value2);  
t.doStartTag();  
t.doEndTag();  
t.release();

Интерфейс BodyTag расширяет Tag, определяя дополнительные методы, которые позволяют обработчику тега обращаться к своему телу. Интерфейс предоставляет три новых метода:

* setBodyContent: Создает содержимое тела и добавляет к обработчику тега.
* doInitBody: Вызывается перед вычислением тела тега.
* doAfterBody: Вызывается после вычисления тела тега.

Типичная последовательность вызовов такова:

t.doStartTag();  
out = pageContext.pushBody();  
t.setBodyContent(out);  
// выполнить любую инициализацию, требуемую после установки содержимого тела  
t.doInitBody();  
t.doAfterBody();  
// пока doAfterBody возвращает EVAL\_BODY\_BUFFERED мы  
// повторяем вычисление тела  
...  
t.doAfterBody();  
t.doEndTag();  
t.pageContext.popBody();  
t.release();

Теги: [jsp](http://javatutor.net/tag/jsp) [taglib](http://javatutor.net/tag/taglib) [tags](http://javatutor.net/tag/tags)