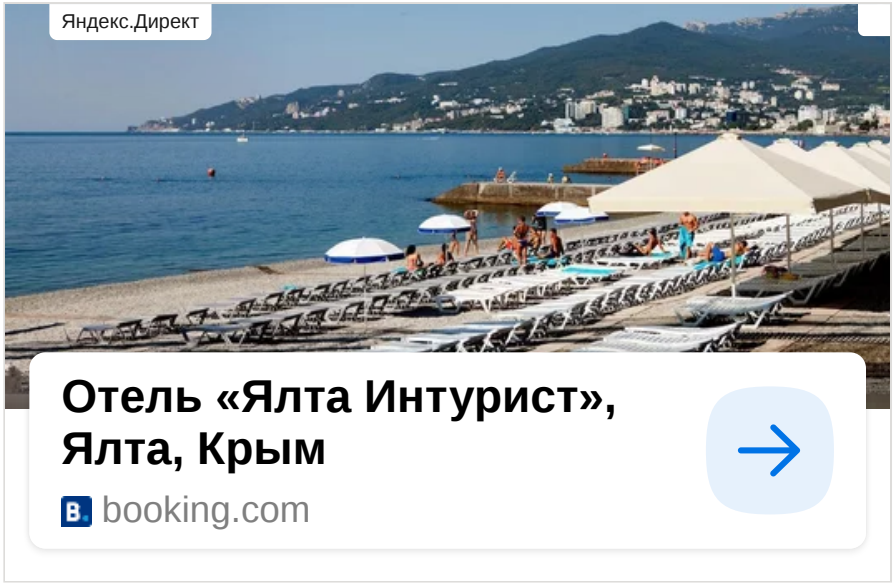
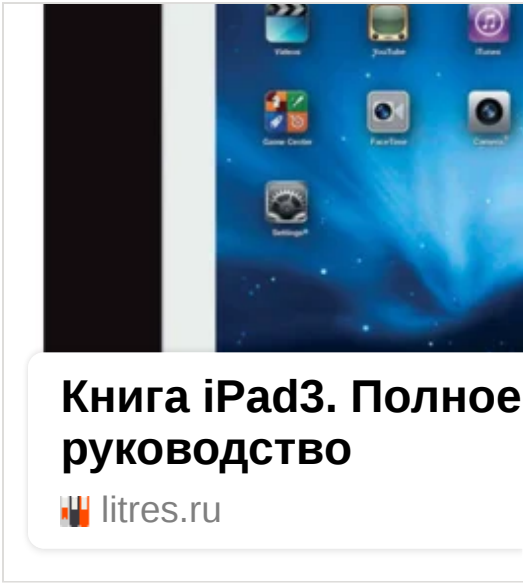




1 2 Следующая >

Java Server Page. Введение

Сервлеты позволяют получать запросы от клиента, совершать некоторую работу и выводить результаты на экран. Сервлет прекрасно работает до того момента, пока речь идет об обработке информации, т.е. до вывода информации на экран. В сервлет можно вставить достаточно сложную логику, сделать вызовы к базе данных и многое-многоедругое, что необходимо для приложения. Но вот осуществлять вывод на экран внутри самого сервлета - очень неудобно. А как быть при разработке сложных дизайнерских идей и последующее внесение изменений в пользовательский интерфейс?

Технология проектирования Java Server Pages (JSP) - это одна из технологий J2EE, которая представляет собой расширение технологии сервлетов для упрощения работы cWeb-содержимым.Страницы JSP позволяет легко разделитьWeb-содержимоена статическую и динамическую часть, допускающую многократное использование ранее определенных компонентов. Разработчики Java Server Pages могут использовать компоненты JavaBeans и создавать собственные библиотеки нестандартных тегов, которые инкапсулируют сложные динамические функциональные средства.

Спецификация Java Server Pages (http://java.sun.com/products/jsp) наследует и расширяет спецификацию сервлетов(http://java.sun.com/products/servlets). Как и сервлеты, компоненты JSP относятся к компонентам Web и располагаются в Webконтейнере. Страницы JSP не зависят от конкретной реализацииWeb-контейнера,что обеспечивает возможность их повторного использования.

В дополнение к классам и интерфейсам для программирования сервлетов (пакеты *javax.servlet ujavax.servlet/http*), в пакетах *javax.servlet.jsp ujavax.servlet.jsp.target* содержатся классы и интерфейсы, относящиеся к программированию Java Server Pages. Полное описание технологии Java Server Pages можно найти в спецификации по адресу(java.sun.com/products/jsp/download.htm)

Pages. Полное описание технологии Java Server Pages можно найти в спецификации по адресу (java.sun.com/products/jsp/download.html)

Обзор технологии Java Server Pages

Технология Java Server Pages содержит четыре ключевых компонента:

Директивы *(directive)* представляют собой сообщения для контейнера JSP, дающим возможность определить параметры страницы, подключения других ресурсов, использовать собственные нестандартные библиотеки тегов.

Действия *(actions)* инкапсулируют функциональные возможности в предопределенных тегах, которые можно встраивать в JSPстраницу. Действия часто выполняются на основе информации, посылаемой на сервер в составе запроса от определенного клиента. Действия также могут создавать объекты Java для использования их в скриптлетах JSP.

Скриптлеты *(scriptlets)* позволяют вставлять код Java в страницы JSP, который взаимодействует с объектами страницы при обработке запроса.

Библиотеки тегов *(tag library)* являются составной частью механизма расширения тегов, допускающего разработку и использование собственных тегов.

Наличие данных с неизменяемой структурой определяют выбор программиста в принятии решения, какую технологию следует использовать: сервлеты или страницы JSP. Программисты предпочитают использовать страницы JSP, если основная часть посылаемого клиенту содержимого представляет собой данные с неизменяемой структурой, и лишь небольшая часть содержимого генерируется динамически с помощью кода Java. Сервлеты предпочтительнее использовать, если только небольшая часть содержимого, посылаемого клиенту, представляет собой данные с неизменяемой структурой. На самом деле отдельные сервлеты могут вообще не генерировать содержимого для клиента, выполняя определенную задачу в интересах клиента, а затем вызывают другие сервлеты или JSP-страницы,чтобы отправить ответ.

Необходимо заметить, что во многих случаях сервлеты и JSP-страницыявляются взаимозамеяемыми. Подобно сервлетам, JSPстраницы обычно выполняются на стороне Web-сервера,который называют контейнером JSP.

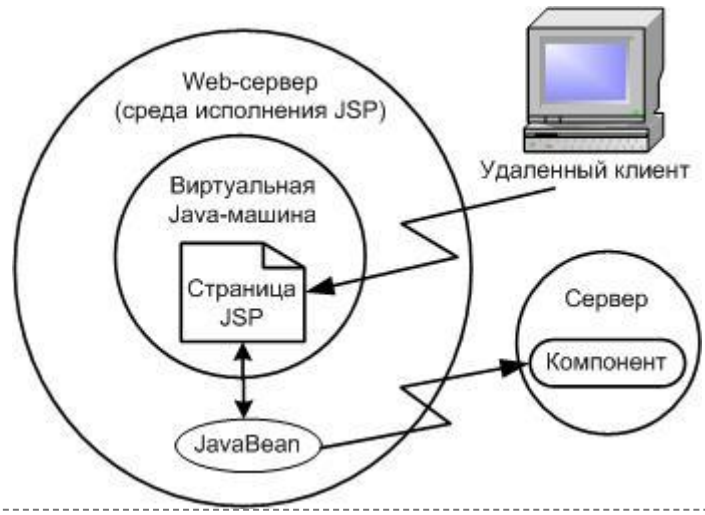
Когда Web-сервер,поддерживающий технологию JSP, принимает первый запрос наJSP-страницу,контейнер JSP транслирует эту JSPстраницу в сервлет Java, который обслуживает текущий запрос и все последующие запросы к этой странице. Если при компиляции нового сервлета возникают ошибки, эти ошибки приводят к*ошибкам на этапе компиляции.* Контейнер JSP на этапе трансляции помещает операторы Java, которые реализует ответJSP-страницы,в метод *_jspService.* Если сервлет компилируется без ошибок, контейнер JSP вызывает метод *_jspService* для обработки запроса.

JSP-страницаможет обработать запрос непосредственно, либо вызвать другие компоненты Web-приложения,чтобы содействовать обработке запроса. Любые ошибки, которые возникают в процессе обработки, вызывают*исключительную ситуацию в Web-сервере на этапе запроса*.

Весь статический текст HTML, называемый в документации JSP *шаблоном HTML* (template HTML), сразу направляется в выходной поток. Выходной поток страницы буферизуется. Буферизацию обеспечивает класс*JspWriter*, расширяющий класс*Writer*. Размер буфера по умолчанию ограничен до 8 Кбайт, но его можно изменить атрибутом*buffer* тега<%@ *page*>. Наличие буфера позволяет заносить заголовки ответа в выходной поток совместно с выводимым текстом. В буфере заголовки будут размещены перед текстом. Таким образом, достаточно написать страницу JSP, сохранить ее в файле с расширением*jsp* и установить файл в контейнер, так же, как и страницу HTML, не заботясь о компиляции. При установке можно задать начальные параметры страницы JSP так же, как и начальные параметры сервлета.

Java Server Page. Архитектура JSP-страницы

Базовая архитектура Java Server Pages в самом общем виде представлена на рисунке. Платформа J2EE обеспечивает базу, на которой функционирует все приложение в целом и страницы JSP, в частности, в то время, как сеть Интернет предоставляет механизм транспортировки данных. Страница JSP располагается на Web-серверев среде виртуальнойJava-машины.Доступ к страниц JSP, как и в случае сервлета, осуществляется через Web с использованием протокола HTTP.



Страница JSP функционирует под управлением JSP Engine (среды исполнения JSP). Страница JSP может взаимодействовать с программным окружением с помощью компонентов JavaBeans, получая и устанавливая его параметры, используя теги: *<jsp:useBean>, <jsp:getProperty>, <jsp:setProperty>*. Компонент JavaBean сам может

участвовать в других процессах, предоставляя результаты в виде своих параметров, доступных страницам JSP, участвующим в сеасе, а через них - всем пользователям, запрашивающим эти страницы JSP.
Использование межплатформенных компонентов JavaBeans и библиотек тегов значительно расширяет возможности JSP. Программный Java-кодв странице JSP, в идеале, должен использоваться только для управления представлением информации.

Основные модели архитектуры JSP

Возможны различные подходы к использованию технологии JSP. Два основных архитектурных подхода, нашедшие применение при реализации приложений уровня предприятия, имеют специальные названия:

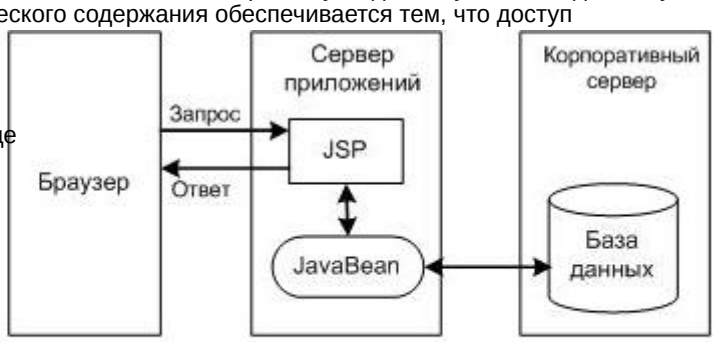
JSP Model 1 (Первая модель архитектуры JSP);

JSP Model 2 (Вторая модель архитектуры JSP);

JSP Model 2 (Вторая модель архитектуры JSP),

Первая модель архитектуры JSP

JSP Model 1 (Первая модель архитектуры JSP) практически реализует базовую архитектуру JSP. В архитектурном решении JSP Model 1 полностью отвечает за получение запроса от клиента, его обработку, подготовку ответа и доставку ответа пользователю. Разделение представления и динамического содержания обеспечивается тем, что доступ к данным осуществляется через компоненты JavaBeans. В сценарии *JSP Model 1* предполагается следующая последовательность действий:



Запрос пользователя посылается через Web-браузерстранице JSP.

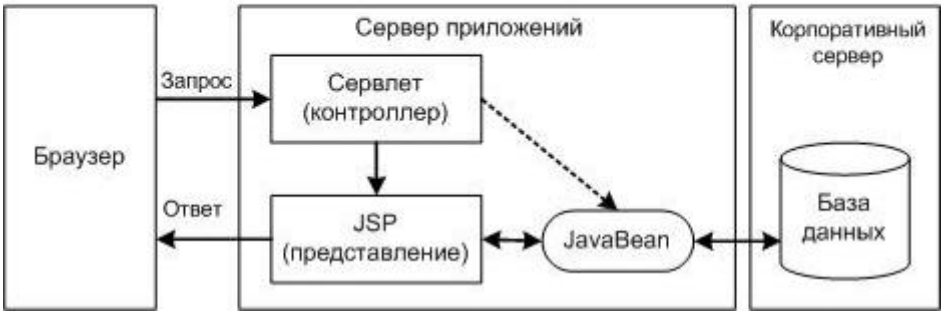
Страница JSP компилируется в сервлет (*при первом обращении*). Скомпилированный сервлет обращается к некоторому компоненту JavaBean, запрашивая у него информацию. Компонент JavaBean, в свою очередь, осуществляет доступ к информационным ресурсам (непосредственно или через компонент Enterprise JavaBeans).

Полученная информация отображается в свойствах компонента JavaBeans, доступных странице JSP. Формируется ответ в виде страницы HTML с комбинированным содержанием (статическое, динамическое).

Архитектура *JSP Model 1* может с успехом применяться для небольших приложений. Однако использование данной модели для более сложных задач вызывает определенные трудности и не является технологичнымиз-забольшого объема встроенных в страницу программных фрагментов. Для сложных корпоративных приложений рекомендуется применение второй модели архитектуры JSP.

Вторая модель архитектуры JSP

JSP Model 2 (Вторая модель архитектуры JSP) реализует гибридный подход к обслуживанию динамического содержания Webстраницы, при котором совместно используется сервлет и страница JSP. Эта модель позволяет эффективно использовать преимущества обеих технологий: сервлет поддерживает задачи, связанные с обработкой запроса и созданием объектов JavaBeans, используемых JSP, а страница JSP отвечает за визуальное представление информации. Сервлет используется как управляющее устройство (контроллер). Схематично вторая модель представлена на рисунке. Сценарии *JSP Model 2*, как правило реализует следующую типовую последовательность действий:



Запрос пользователя посылается через Web-браузерсервлету.

Сервлет обрабатывает запрос, создает и инициализирует объект JavaBean или другие объекты, используемые страницей JSP, и запрашивает динамическое содержание у компонента JavaBean.

Компонент JavaBean осуществляет доступ к информации непосредственно или через компонент Enterprise JavaBeans. Сервлет, направляющий запрос, вызывает сервлет, скомпилированный из страницы JSP.

Сервлет, скомпилированный из страницы JSP, встраивает динамическое содержание в статический контекст HTML-страницыи отправляет ответ пользователю.

Необходимо отметить, что в рамках этой модели страница JSP сама не реализует никакую логику, это входит в функции сервлета-контроллера.Страница JSP отвечает только за получение информации от компонента JavaBean, который был предварительно создан сервлетом, и за визуальное представление этой информации в удобном для клиента виде.

Архитектуры *JSP Model 2* в большей степени, чем архитектура*JSP Model 1*, соответствует идее отделения представления от содержания. Эта модель позволяет четко выделить отдельные части приложения и связанные с ними роли и обязанности персонала, занятого в разработке:

Дизайнер - разработка дизайна Web-страницы;Разработчик - реализация функций управления и обработки.

Чем сложнее разрабатываемая система, тем заметнее становятся преимущества архитектуры *JSP Model 2*.

Функционирование JSP

Работа со страницей JSP становится возможной только после ее преобразования в сервлет. В процессе трансляции как статическая, так и динамическая части JSP преобразуются в Java-кодсервлета, который передает преобразованное содержимое браузеру через выходной потокWeb-сервера.

Технология JSP является технологией серверной стороны, поэтому все процессы обработки JSP протекают на стороне сервера.

Страница JSP - текстовый документ, который в соответствии со спецификацией JSP, проходит две фазы:

фазу трансляции;

фазу выполнения.

При трансляции, которая выполняется один раз для каждой страницы JSP, создается или локализуется класс типа *Servlet*, реализующий JSP. Трансляция JSP может производиться как отдельно, до ее использования, так и в процессе размещения JSP наWeb-сервереили сервере приложений.

Во второй фазе осуществляется обработка запросов и подготовка ответов.

Java Server Page. Жизненный цикл страницы JSP

Страница JSP обслуживает запросы, как сервлет. Следовательно, жизненный цикл и многие возможности страниц JSP (в частности, динамические аспекты) определяются технологией Java Servlet и многие обсуждения в этой главе ссылаются на функции, описанные в разделе сервлетов. Когда запрос отображается на страницу JSP, он обрабатывается специальным сервлетом, который сначала проверяет, не старше ли сервлет страницы JSP, чем сама страница JSP. Если это так, он переводит страницу JSP в класс сервлета и компилирует класс. При разработкеWeb-приложенияодним из преимуществ страниц JSP перед сервлетами является то, что процесс построения (*компиляции страницы JSP в сервлет*) выполняется автоматически.

Трансляция и компиляция

На фазе трансляции каждый тип данных в странице JSP интерпретируется отдельно. Шаблонные данные трансформируются в код, который будет помещать данные в поток, возвращающий данные клиенту. Элементы JSP трактуются следующим образом:
директивы, используемые для управления тем, как Web-контейнерпереводит и выполняет страницу JSP;
скриптовые элементы вставляются в класс сервлета страницы JSP;
элементы в форме <jsp:XXX ... /> конвертируются в вызов метода для компонентов JavaBeans или вызовы API Java Servlet.
И фаза трансляции, и фаза компиляции могут порождать ошибки, которые будут выведены только, когда страница будет в первый раз запрошена. Если ошибка возникает при трансляции страницы (например, транслятор находит элемент JSP с неправильным форматом), сервер возвращает *ParseException*, и исходный файл класса сервлета будет пустым или незаконченным. Последняя незаконченная строка дает указатель на неправильный элемент JSP. Если ошибка возникает при компиляции страницы (например, синтаксическая ошибка в скриптлете), сервер возвращает*JasperException* и сообщение, которое включает в себя имя сервлета страницы JSP и строку, в которой произошла ошибка.

Когда страница оттранслирована и откомпилирована, сервлет страницы JSP в основном следует жизненному циклу сервлета, описанному в разделе Жизненный цикл сервлета :
Если экземпляр сервлета страницы JSP не существует,
контейнер: загружает класс сервлета страницы JSP;
создает экземпляр класса сервлета;
инициализирует экземпляр сервлета вызовом метода jspInit. Вызывает
метод _jspService, передавая ему объекты запроса и отклика.
Если контейнеру нужно удалить сервлет страницы JSP, он вызывает метод jspDestroy.

Выполнение страницы JSP

Различными параметрами выполнения страницы JSP можно управлять при помощи директив *page*. Когда страница JSP выполняется, вывод, записываемый в объект отклика, автоматически буферизируется. Размер буфера можно установить директивой*page*. Больший буфер позволяет записать больше содержимого, прежде чемчто-либодействительно будет передано клиенту, это предоставляет странице JSP больше времени на установку соответствующих кодов состояния и заголовков или на обращение к другомуWeb-ресурсу.Меньший буфер уменьшает загрузку памяти сервера и позволяет клиенту быстрее начать получение данных.

При выполнении страницы JSP может возникать любое число исключений. Чтобы определить, что Web-контейнердолжен передавать управление странице ошибки, если происходит исключение, необходимо определить в странице JSP следующую директиву page:

```
<%@ page errorPage="file_name" %>
```

Java Server Page. Синтаксис JSP-страницы

Страницы JSP имеют комбинированный синтаксис: объединение стандартного синтаксиса, соответствующего спецификации HTML, и синтаксиса JSP, определенного спецификацией Java Server Pages. Синтаксис JSP определяет правила записи страниц JSP, состоящих из стандартных тегов HTML и тегов JSP.
Страницы JSP, кроме HTML-тегов,содержат теги JSP следующих категорий:
директивы (directives);
page;
taglib;
include;
объявления (declarations);
скриптлеты (scriptlets);
выражения (expressions);
комментарии (comments);

Директивы JSP

Директивы обеспечивают глобальную информацию, касающихся конкретных запросов, направляемых в JSP, и предоставляют информацию, необходимую на стадии трансляции.
Директивы всегда помещаются в началеJSP-страницыдо всех остальных тегов, чтобы*parser* (анализатор) JSP при разборе текста в самом начале выделил глобальные инструкции. Таким, образом, JSP Engine (среда исполнения JSP), анализируя код, создает из JSP сервлет.*Директивы* представляют собой сообщения контейнеру JSP.
Синтаксис *директив* JSP выглядит следующим образом:

```
<%@ директива имяАтрибута="значение" %>
```

Синтаксис задания *директив* на XML:

```
<jsp:directive.директива имяАтрибута="значение" />
```

Директива может иметь несколько атрибутов. В этом случае*директива* может быть повторена для каждого из атрибутов. В то же время пары*"имяАтрибута=значение"* могут располагаться под одной директивой с пробелом в качестве разделителя.
Существует три типа директив:
page (страница)
taglib (библиотека тегов)
include (включить)

Директива page

Директива *page* определяет свойства страницы JSP, которые воздействуют на транслятор. Порядок следования атрибутов в директиве*page* не имеет значения. Нарушение синтаксиса или наличие нераспознанных атрибутов приводит к ошибке трансляции. Примером директивы*page* может служить следующий код:

```
<%@ page buffer="none" isThreadSafe="yes" errorPage="/error.jsp" %>
```

<%@ page buffer="none" isThreadSafe="yes" errorPage="/error.jsp" %>

Эта директива объявляет, что данная страница JSP не использует буферизацию, что возможно одновременное обращение к данной странице JSP многих пользователей, и что поддерживается страница ошибок с именем *error.jsp*. Директива*page* может содержать информацию о странице:

```
<%@ page info = "JSP Sample 1" %>
```

Список возможных атрибутов директивы *page* представлен в таблице.

Наименование атрибута	Значение	Описание
language	Строка	Определяет язык, используемый в скриптлетах файла JSP, выражениях или любых включаемых файлах, в том числе, в теле оттранслированного кода. По умолчанию принимается значение "java"
extends	Строка	Задает суперкласс для генерируемого сервлета. Этот атрибут следует использовать с большой осторожностью, поскольку возможно что сервер уже использует какой-нибудь суперкласс
import	Строка	Определение импортируемых пакетов., например: <%@ page import="java.util.* %>
Session	true или false	Значение *true* (принимается по умолчанию) свидетельствует о том, что заранее определенная переменная *session* (тип HttpSession) должна быть привязана к существующей сессии, если таковая имеется, в противном случае создается новая сессия, к которой осуществляется привязка. Значение *false* определяет что сессии не будут использоваться, и попытки обращения к переменной *session*приведут к возникновению ошибки при трансляции JSP страницы в сервлет
Buffer	none или размер буфера в кБ.	Задает размер буфера для JspWriter out. Значение принимаемое по умолчанию зависит от настроек сервера, и не должно превышать 8 кБ. Если значение равно *none* вывод происходит непосредственно в объект
autoFlush	true или false	Определяет, должен ли буфер освобождаться автоматически, когда он переполнен или произошла ошибка. По умолчанию значение *true*
isThreadSafe	true или false	Значение *true* (принимается по умолчанию) задает нормальный режим выполнения сервлета, когда множественные запросы обрабатываются одновременно с использованием одного экземпляра сервлета, исходя из соображения, что автор синхронизировал доступ к переменным этого экземпляра. Значение false ("ложь") сигнализирует о том, что сервлет должен наследовать *SingleThreadModel*(однопоточную модель), при которой последовательные или одновременные запросы обрабатываются отдельными экземплярами сервлета
info	Строка	Определяет строку информации о странице JSP, которая будет доступна с помощью метода *Servlet.getServletInfo ()*
errorPage	Строка	Значение атрибута представляет собой URL страницу, которая должна выводиться в случае возможных ошибок, вызывающих исключения
isErrorPage	true или false	Сигнализирует о том, может ли эта страница использоваться для обработки ошибок для других JSP страниц. По умолчанию принимается значение *false*
contentType	Строка	Определяет кодировку для страницы JSP и ответа, а также MIME-типответа JSP. Значение по умолчанию типа содержания - *text/html*, кодировки -*ISO-8859-1.* Например: contentType="text/html;charset=ISO-8859-1"
pageEncoding	Строка	Определяет кодировку символов страницы JSP. По умолчанию используется *charset* из атрибута *contentType*, если оно там определено. Если значение*charset* в атрибуте *contentType* не определено, значение*pageEncoding* устанавливается равным *ISO-8859-1*

Директива taglib

Директива *taglib* объявляет, что данная страница JSP использует библиотеку тегов, уникальным образом идентифицируя ее с помощью URI, и ставит в соответствие префикс тега, с помощью которого возможны действия в библиотеке. Если контейнер не может найти библиотеку тегов, возникает фатальная ошибка трансляции.

Директива *taglib* имеет следующий синтаксис:

```
<%@ taglib uri="URI включаемой библиотеки тегов" prefix="имяПрефикса" %>
```

Префикс "*имяПрефикса*" используется при обращении к библиотеке. Пример использования библиотеки тегов*mytags*:

```
<%@ taglib uri="http://www.taglib/mytags" prefix="customs" %>
. . .
<customs:myTag>
```

В данном примере библиотека тегов имеет URI-адрес*"http://www.taglib/mytags"*, в качестве префикса назначена строка*customs*, которая используется в странице JSP при обращении к элементам библиотеки тегов.

Директива include

Директива *include* позволяет вставлять текст или код в процессе трансляции страницы JSP в сервлет. Синтаксис директивы*include* имеет следующий вид:

```
<%@ include file="Относительный URI включаемой страницы" %>
```

Директива *include* имеет один атрибут *-file*. Она включает текст специфицированного ресурса в файл JSP. Эту директиву можно использовать для размещения стандартного заголовка об авторских правах на каждой странице JSP:

```
<%@ include file="copyright.html" %>
```

Контейнер JSP получает доступ к включаемому файлу. Если включаемый файл изменился, контейнер может перекомпилировать страницу JSP. Директива *include* рассматривает ресурс, например, страницу JSP, как статический объект.

Заданный URI обычно интерпретируется относительно JSP страницы, на которой расположена ссылка, но, как и при использовании любых других относительных URI, можно задать системе положение интересующего ресурса относительно домашнего каталога WEB-серверадобавлением в начало URI символа "/". Содержимое подключаемого файла обрабатывается как обычный текст JSP и поэтому может включать такие элементы, как статический HTML, элементы скриптов, директивы и действия.

Многие сайты используют небольшую панель навигации на каждой странице. В связи с проблемами использования фреймов HTML часто эта задача решается размещением небольшой таблицы сверху или в левой половине страницы, HTML код которой многократно повторяется для каждой страницы сайта. Директива *include* - это наиболее естественный способ решения данной задачи, избавляющий разработчика от кошмара рутины копирования HTML в каждый отдельный файл.

Поскольку директива *include* подключает файлы в ходе трансляции страницы, то после внесения изменений в панель навигации требуется повторная трансляция всех использующих ее JSP страниц. Если же подключенные файлы меняются довольно часто, можно использовать действие*jsp:include*, которое подключает файл в процессе обращения к JSP странице.

Объявления JSP

Declarations (Declarations) предназначены для определения переменных и методов на языке скриптов, которые в дальнейшем используются на странице JSP. Синтаксис*declarations* имеет следующий вид :

```
<%! код Java %>
```

Объявления располагаются в блоке объявлений, а вызываются в блоке выражений страницы JSP. Код в блоке объявлений обычно пишется на языке Java, однако серверы приложений могут использовать синтаксис и других скриптов. *Объявления* иногда используются для того, чтобы добавить дополнительную функциональность при работе с динамическими данными, получаемыми из свойств компонентов JavaBeans. Примеры*объявлений* представлены в таблице.

<%! int i ; %>	Объявление глобальной целочисленной переменной *i*.
<%! String s = "Hello, World!"; %>	Объявление и инициализация глобальной строковой переменной *s*.

<%! int n = 100; %>	Объявление и инициализация глобальной целочисленной переменной *n*.
<%! public int adding (int a, int b){return a + b}; %>	Объявление метода *adding* сложения двух целочисленных значений, глобального для всей страницы JSP

Объявление может содержать несколько строк, как например, в приведенном ниже коде вычисления значения функции*fact (int n)*, которая должна быть равна 1 при n меньше 2 и n! при положительном значении n;

```
<%!
public static int fact (int n)
{
  if (n <= 1)
    return 1;
  else
```

```
    return n * fact (n - 1);
}
%>
```

Объявления не производят никакого вывода в стандартный выходной поток*out*. Переменные и методы, декларированные в*объявлениях*, инициализируются и становятся доступными для скриптлетов и других*объявлений* в момент инициализации страницы JSP.

Скриптлеты JSP

Скриптлеты включают различные фрагменты кода, написанного на языке скрипта, определенного в директиве*language*. Фрагменты кода должны соответствовать синтаксическим конструкциям языка*скриптлетов*, т.е., как правило, синтаксису языка Java.*Скриптлеты* имеют следующий синтаксис:

```
<% текст скриптлета %>
```

Эквивалентом синтаксиса *скриптлета* для XML является:

```
<jsp:scriptlet> текст скриптлета </jsp:scriptlet>
```

Если в тексте *скриптлета* необходимо использовать последовательность символов %> именно как сочетание символов, а не как тег - признак окончания*скриптлета*, вместо последовательности %> следует использовать следующее сочетание символов %\>.
В спецификации JSP приводится простой и понятный пример *скриптлета*, обеспечивающего динамическое изменение содержимого страницы JSP в течение дня.

```
<% if (Calendar.getInstance ().get (Calendar.AM_PM) ==
  Calendar.AM) {%> Good Morning
<% } else { %>
  Good
  Afternoon
<% } %>
```

Необходимо заметить, что код внутри *скриплета* вставляется в том виде, как он записан, и весь статическийHTML-текст(текст шаблона) до или пос*ле скриплета* конвертируется при помощи оператора*print*. Это означает что скриплеты не обязательно должны содержать завершенные фрагменты на Java, и что оставленные открытыми блоки могут оказать влияние на статическийHTML-текствне*скриплета*.

Скриплеты имеют доступ к тем же автоматически определенным переменным, что и выражения. Поэтому, например, если есть необходимость вывестикакую-либоинформацию на страницу, необходимо воспользоваться переменной*out*.

```
<%
   String queryData = request.getQueryString ();
   out.println ("Дополнительные данные запроса: " + queryData);
%>
```

Выражения JSP

Выражение в странице JSP - это исполняемое выражение, написанное на языке скрипта, указанного в объявлении *language* (как правило Java). Результат*выражения* JSP, имеющий обязательный тип*String*, направляется в стандартный поток вывода*out* с помощью текущего объекта*JspWriter*. Если результат*выражения* не может быть приведен к типу*String*, возникает либо ошибка трансляции, если проблема была выявлена на этапе трансляции, либо возбуждается исключение*ClassCastException*, если несоответствие было выявлено в процессе выполнения запроса.*Выражение* имеет следующий синтаксис:

```
<%= текст выражения %>
```

альтернативный синтаксис для *выражений* JSP при использовании XML:

```
<jsp:expression> текст выражения </jsp:expression>
```

Порядок выполнения *выражений* в странице JSPслева-направо.Если*выражение* появляется более чем в одном атрибуте времени выполнения, то оно выполняетсяслева-направов данном теге.*Выражение* должно быть полным выражением на определенном скрипте (как правило Java).
Выражения выполняются во время работы протокола HTTP. Значение выражения преобразуется в строку и включается в соответствующую позицию файла JSP.
Выражения обычно используются для того, чтобы вычислить и вывести на экран строковое представление переменных и методов, определенных в блоке объявлений страницы JSP или полученных от компонентов JavaBeans, которые доступны из JSP. Следующий код*выражения* служит для отображения даты и времени запроса данной страницы:

```
Текущее время: <%= new java.util.Date () %>
```

Текущее время: <%= new java.util.Date() %>

Для того чтобы упростить *выражения* существует несколько заранее определенных переменных, которые можно использовать. Наиболее часто используемые переменные:
request, HttpServletRequest;
response, HttpServletResponse;
session, HttpSession - ассоциируется с запросом , если таковой имеется;
out, PrintWriter - буферизированный вариант типа JspWriter для отсылки данных клиенту.

Комментарии JSP

В страница JSP можно использовать два типа комментариев:
Выводимый комментарий (output comment).
Закомментированный блок (commented block).
Выводимый комментарий - это такой стиль комментирования, при котором комментарий выводится в выходной поток и отображается в браузере. Синтаксис комментариев данного типа следующий:

<!--Текст комментария-->

Можно также использовать комментарии с динамическим содержимым. Для этого внутрь закомментированного блока должен быть встроен тег скриптлета, например:

<!--Начало комментария <%= expression %> продолжение комментария-->

Закомментированный блок - это стиль комментирования, при котором тело закомментированного блока полностью игнорируется, не выводится в выходной поток и, следовательно, не отображается в браузере. Этот стиль обычно используется для поясняющих записей, касающихся фрагментов программного кода страницы JSP.Закомментированный блок имеет следующий синтаксис:

<%--Текст комментария--%>

Java Server Page. Действия JSP-страницы

Действия (*actions*) JSP могут воздействовать на стандартный поток вывода, использовать, модифицировать и создавать объекты. Действия используют конструкции с синтаксисом XML для управления работой движка сервлета, и позволяют, таким образом, динамически подключать файл, многократно использовать компоненты JavaBeans, направлять пользователя на другую страницу или генерировать HTML для Java plugin.
Согласно спецификации JSP, синтаксис действий базируется на XML. В таблице представлены соответствия утверждений JSP и XML.

Конструкция JSP	Эквивалентная конструкция в XML
<% page %>	<%jsp:directive.page />
<% include %>	<%jsp:directive.include />
<%! ... %>	<%jsp:declaration> . . . <%/jsp:declaration>
<% ... %>	<%/jsp:scriptlet> . . . <%/jsp:scriptlet>
<%= ... %>	<%/jsp:expression>
	. . . <%/jsp:expression>

Существует набор стандартных действий, которые должны быть в обязательном порядке реализованы любым контейнером JSP, удовлетворяющим спецификации. Кроме этого, возможно создание новых действий с помощью директивы библиотеки тегов*taglib*. Список стандартных действий представлен в следующей таблице.
Список стандартных действий

Тип действия	Назначение действия
<jsp:useBean>	Объявление объекта JavaBean, который будет использоваться на странице JSP
<jsp:setProperty>	Установление значения свойства объекта JavaBean
<jsp:getProperty>	Чтение значения свойства объекта JavaBean
<jsp:include>	Включение в страницу JSP дополнительных статических и динамических ресурсов
<jsp:forward>	Перенаправление обработки на другой статический ресурс, например сервлет

<jsp:plugin>	Подключение дополнительных программных модулей (компонент JavaBean или апплет)
<jsp:param>	Определение значения параметра

Атрибуты тега действия

Для интерпретации действий определяются значения атрибутов, слева-направо,причем, при этом могут совершаться преобразования типов, предусмотренные спецификацией JSP.

Значения атрибутов могут быть следующими:

translation-time,attribute values - фиксированное значение времени транслирования;

request-time,attribute values - значения, вычисляемые в процессе запроса. В этом случае значение атрибута описывается как "<%= выражение %>" или '<%= выражение %>'. Кавычки используются также, как и при определении других атрибутов. Значение атрибута присваивается как результат вызываемого выражения.

Значения атрибутов, вычисляемые в процессе запросов, используются только в действиях, т.е. они, например, не могут использоваться в директивах. Тип элемента действия определяет, зависит ли данный атрибут от запроса или нет.

По умолчанию все атрибуты имеют семантику *page* - фиксированное значение времени транслирования. Большинство атрибутов стандартных действий относятся к типу значений времени трансляции. К значениям, вычисляемым в процессе запроса, относятся атрибуты, представленные в таблице.

Наименование атрибута	Наименование действия	Примечание
beanName	jsp:useBean	Поиск или создание нового экземпляра JavaBean
height	jsp:plugin	Определение высоты объекта, размещаемого на странице JSP
width	jsp:plugin	Определение ширины объекта, размещаемого на странице JSP
bean | applet	jsp:plugin	Генерирование кода (в зависимости от типа используемого броузера), который создает тэг OBJECT или EMBED для Java plugin
page	jsp:include	Подключение файла в момент запроса страницы
page	jsp:forward	Перенаправление запроса на другую страницу
value	jsp:setProperty	Установка свойств компонента JavaBean
value	jsp:getProperty	Чтение свойств компонента JavaBean в выходной поток
value	jsp:param	Определение значения параметры для компонента, размещаемого на странице JSP

Java Server Page. Действие jsp:useBean

Тег jsp:useBean позволяет ассоциировать экземплярJava-класса,определенный в данной области видимости*scope*, с заданным внутренним идентификатором этого класса*id* в данной странице JSP. Прежде, чем использовать свойства компонента JavaBean (*setProperty* и*getProperty*), необходимо объявить тегjsp:useBean. При выполнении тегаjsp:useBean сервер приложений обеспечивает поиск (lookup) экземпляра данногоJava-класса,пользуясь значениями, определенными в атрибутах:

id - идентификатор экземпляра класса внутри страницы JSP; scope -

область видимости (page, request, session, application).

Если объект с данными атрибутами *id, scope* не найден, предпринимается попытка создать объект, используя значения, определенные в его атрибутах. Синтаксис действияjsp:useBean может включать тело :

Синтаксис действия jsp:useBean

```
<jsp:useBean id="идентификатор"
   scope = "page | request | session | application"
   class = "имяКласса" type = "имяТипа" |
   type = "имяТипа" | class = "имяКласса" |
   beanName = "имяКомпонентаJavaBean" | type =
   "имяТипа" | type = "имяТипа" | beanName =
   "имяКомпонентаJavaBean" | type = "имяТипа">
   <!--Тело-->
   </jsp:useBean>
```

При наличии в объявление *тела*, оно будет вызвано на выполнение, если компонент JavaBean, к которому обращено действие, уже существует. Содержимое тела действия строго не ограничено, однако, как правило, тело действия содержит скриптлеты или теги jsp:setProperty, модифицирующие свойства созданного объекта.

Действие jsp:useBean очень гибкое. Правила, по которым оно выполняется, зависят от текущих значений его атрибутов. С помощью действияjsp:useBean можно:

найти существующий компонент JavaBean. Если объект найден, то он доступен, например, из объекта *ServletRequest*, определенного для страницы с помощью метода*getAttribute(name);*

создать новый экземпляр класса JavaBean - создается новая переменная на языке скрипта с именем, определяемым параметром *id*, и в области видимости, заданной параметром*scope*;

используя атрибут *type*, задать локальное имя объекту, определенном в другой странице JSP или сервлете (атрибуты*class* или*beanName* при этом не используются).

Тег jsp:useBean имеет параметры, представленные в таблице.

Атрибут	Описание атрибута
id	Параметр, идентифицирующий экземпляр объекта в пространстве имен, специфицированном в атрибуте *scope*. Это имя используется для ссылки на компонент JavaBean из страницы JSP.
scope	Атрибут, определяющий область видимости ссылки на экземпляр объекта JavaBean. Допустимыми значениями являются *page, request, session, application*. Данный атрибут фактически описывает пространство имен и цикл жизни ссылки на объект. По умолчанию значение объекта равно *page*. page (страница). Объект, определенный с областью видимости *page*, доступен до тех пор, пока не будет отправлен ответ клиенту или пока запрос к текущей странице JSP не будет перенаправлен куда-нибудьеще. Ссылки на объект возможны только в пределах страницы, в которой этот объект определен. Объекты, объявленные с атрибутом *page*, сохраняются в объекте*pageContext*. request (запрос). Объект, имеющий область видимости *request*, существует и доступен в течение текущего запроса, и остается видимым, даже если запрос перенаправляется другому ресурсу в том же самом цикле выполнения. Объекты, декларированные с атрибутом области видимости *request*, сохраняются в объекте *request*. session (сессия). Объект, имеющий область видимости *session* доступен в течение текущего сеанса, если страница JSP "знает" о сеансе. application (приложение). Объект, имеющий область видимости *application* доступен страницам, обрабатывающим запросы в одном и том же приложении Web, и существует до тех пор, пока сервер приложений поддерживает объект ServletContext. Объекты, объявленные с атрибутом области видимости*application*, сохраняются в объекте*application*.
class	Параметр, определяющий полное имя класса реализации объекта. Данный атрибут используется при инициализации экземпляра компонента JavaBean, если он еще не существует в данной области видимости *scope*.
beanName	Наименование класса реализации объекта. Данный параметр используется, если компонент JavaBean еще не существует. Параметр *beanName* должен удовлетворять правилам наименования переменных, предусмотренным спецификацией языка скриптлетов. Формат параметра : "строка1.строка2.строка3" - для классов и "строка1/строка2/строка3" - для ресурсов. Параметр*beanName* предполагает использование метода*instantiate ()* класса*java.beans.Beans*.
type	Атрибут *type*, определяющий тип специфицированного объекта, дает возможность определить тип переменных скрипта как класс, суперкласс или интерфейс, реализуемый классом. С помощью атрибута *type* можно избежать автоматической инициализации компонента JavaBean, если он еще не существует в данной области видимости. По умолчанию атрибут type имеет значение, определенное в атрибуте *class*. Если объект не соответствует даваемому атрибутом*type* типу, может быть возбуждено исключение*java.lang.ClassCastException*.

Если имя класса (*class*) и имя объекта (*beanName*) не определены, объект должен быть представлен в заданной области видимости. Значение идентификатора*id* должно быть уникально в текущем модуле трансляции(JSP-странице),иначе возникает ошибка трансляции.

Java Server Page. Действие jsp:setProperty

Тег jsp:setProperty позволяет присваивать значения свойствам компонента JavaBean, который должен быть предварительно создан действиемjsp:useBean и содержать соотетствующие свойства. Действиеjsp:setProperty имеет следующий синтаксис :

Синтаксис действия jsp:setProperty

```
<jsp:setProperty name="идентификатор"
  property = "*" |
  property = "имяСвойства" |
  property = "имяСвойства" | param = "имяПараметра" |
  property = "имяСвойства" | value = "значение" | property
  = "имяСвойства" | value = <%= выражение %>
/>
```

Атрибут	Описание атрибута
	Параметр, идентифицирующий экземпляр объекта JavaBean, предварительно определенный в name теге jsp:useBean, свойство которого устанавливаются текущим тегом jsp:setProperty
	Имя свойства, которому необходимо определить значение. Если используется символ "*", то предполагается автоматическая установка значений свойств. В последнем случае соответствующие элементы формы должны

property	иметь имена, совпадающие с именами устанавливаемых свойств компонента JavaBean. В этом случае по именам элементов формы осуществляется поиск (look up) соответствующих свойств компонента JavaBean с последующей установкой их значений
param	Имя параметра запроса, который передается свойству компонента JavaBean. Параметры запроса, как правило, ссылаются на соответствующие элементыHTML-страницы.Этот атрибут не может использоваться одновременно с атрибутом *value*
value	Новое значение устанавливаемого свойства

Значения свойств компонента JavaBean устанавливается с учетом соответствия типов значения и свойства.
Тег jsp:setProperty позволяет устанавливать значения как простых, так и индексированных ствойств. Свойства компонента JavaBean имеют определенный тип, а также методы*setter* и*getter*. При установке параметров обычно проверяются наличия свойств компонента JavaBean, их имена и типы, являются ли свойства простыми или индексированными и т.д.
Значения одного или нескольких свойств компонента JavaBean могут быть установлены несколькими способами:
с помощью параметров объектов типа *request* (запрос);
с использованием строковой константы;
с помощью выражения, вычисляемого во время запроса.
Пример использования тега jsp:setProperty представлен в следующем листинге:
Листинг примера использования тега jsp:setProperty

```
<jsp:useBean id="user" class="hall.users" />
<jsp:setProperty name="user" property="name" value="alex" />
<jsp:setProperty name="user" property="name" value="serg" />
```

Java Server Page. Действие jsp:getProperty

После объявления компонента JavaBean с помощью действия jsp:useBean его незащищенные свойств становятся доступными для действияjsp:getProperty. Терjsp:getProperty делает свойства компонента JavaBean видимыми. Данный тег включает значение типа *String* или объект типа, преобразованный к типу*String*, в выходной поток. Преобразование простых типов в тип*String* происходит автоматически. Для объектов необходим вызов метода*toString*.
Действие jsp:getProperty имеет следующий синтаксис:
Синтаксис действия jsp:getProperty

```
<jsp:getProperty name="идентификатор"
   property = "имяСвойства"
/>
```

Тег jsp:getProperty имеет ряд атрибутов, которые представлены в таблице. При записи тега допускается использовать не все возможные атрибуты.

Атрибут	Описание атрибута
name	Параметр, идентифицирующий экземпляр объекта JavaBean, предварительно определенный в теге jsp:useBean

Соседние файлы в папке лабораторная работа 6 (jsp)

code.txt	46	#	7.7 Кб	09.05.2014
Java Server Page.docx	55	#	183.72 Кб	09.05.2014
Java Server Page.pdf	56	#	785.64 Кб	09.05.2014
lab2.doc	50	#	156.16 Кб	09.05.2014
Введение в JSP.docx	50	#	166.4 Кб	09.05.2014
Введение в JSP.pdf	53	#	524.65 Кб	09.05.2014