Java программирование интернет-приложений

Конспект лекций

Аннотация

Цель дисциплины “**Java программирование интернет-приложений**” - систематическое изучение средств J2EE, относящихся к разработке web-интерфейсов (web-tier в терминологии J2EE), включая такие программные интерфейсы как Java servlets, Java servlets filters, JSP, пользовательские библиотеки тегов и соответствующие шаблоны программирования, применяемые в данной области.

Обязательный курс для студентов I курса, читается во втором семестре.

1. HTTP протокол

Идеология построения протокола HTTP**.** Общая структура сообщений, методы доступа. Заголовок и данные HTTP запросов. Стандартные коды ответов.

HTTP (Hypertext Transfer Protocol) - основной протокол, используемый в Web-приложениях. Изначально был разработан для передачи гипертекста, реально может использоваться для передачи произвольных данных.

Важно понимание принципов его построения, поскольку реально программирование любого сервиса в сети Интернет (с точки зрения сервера, в первую очередь) есть, независимо от языка программирования, разбор HTTP запросов и выдача ответов в формате того же протокола.

Клиент в сети (например, браузер) посылает серверу (веб-серверу) текст, имеющий следующий формат:

Команда URI версия_протокола

Здесь:

Команда – одна из команд протокола

URI – конструкция, описывающая адрес некоторого документа (имя файла)

Версия_протокола: строка вида HTTP/1.0 или HTTP/1.1

Например:

GET http://www.yahoo.com HTTP/1.0

Далее следует набор заголовков. Например:

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*

Accept-Language: ru

User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)

Каждая строка соответствует одному параметру. В начале строки идет имя параметра, затем двоеточие и значение параметра.

Набор параметров запроса не фиксирован.

Поля заголовка запроса позволяют клиенту передавать серверу дополнительную информацию о запросе и о самом клиенте. Например, следующие поля описывают заголовки:

Accept

Accept-Charset

Accept-Encoding

Accept-Language

Authorization

From

Host

If-Modified-Since

If-Match

If-None-Match

If-Range

If-Unmodified-Since

Max-Forwards

Proxy-Authorization

Range

Referer

User-Agent

URI (Universal Resource Identifiers) определяет путь к некоторому ресурсу (файлу) на сервере. Другое используемое в этой связи обозначение - URL (Uniform Resource Locator) – частный случай URI. URI в HTTP может быть представлен в абсолютной или относительной форме по отношению к некоторому известному базовому URI, в зависимости от контекста его использования. Эти две формы отличаются тем, что абсолютный URI всегда начинается с имени схемы (протокола), за которым следует двоеточие (например HTTP: или FTP:).

Детальный синтаксис URL описывается в RFC-1738 и RFC-1808 .

Протокол HTTP не устанавливает каких-либо ограничений на длину URI.

HTTP URL (адресация сетевых ресурсов с помощью протокола HTTP) имеет следующую форму:

"http:" "//" host [":" port] [abs_path]

Если номер порта не указан, предполагается порт 80. Хост задается в виде IP адреса или имени DNS.

Если путь не указан, то он равен “/”

К числу возможных методов (команд HTTP) относятся:

OPTIONS

GET

HEAD

POST

PUT

DELETE

TRACE

Таким образом, клиент посылает запрос серверу в форме, определяющей метод, URI и версию протокола. В конце запроса следуют данные в MIME кодировке.

В HTTP/1.0 приложения используют новое соединение для каждого обмена запрос/отклик. В HTTP/1.1, одно соединение может быть использовано для нескольких обменов запрос/ответ.

Сервер откликается, посылая статусную строку, которая включает в себя версию протокола, код результата (статус), заголовки и данные в MIME кодировке.

Строка статуса имеет вид:

Версия_HTTP код причина

Код есть 3-х значный цифровой результат, причина – объясняющий комментарий. Первая цифрв кода ответа определяет класс отклика:

1xx: - запрос получен, процесс продолжается

2xx: Успех (Success) - запрос успешно обработан (e.g. 200)

3xx: Переадресация (Redirection) – запрос будет переадресован

4xx: Ошибка клиента (Client Error) - запрос содержит синтаксическую ошибку или не может быть выполнен (e.g. 404 – ресурс не найден)

5xx: Ошибка сервера (Server Error) - сервер не смог выполнить корректный запрос

Индивидуальные значения числовых статусных кодов (и комментарии) взяты непосредственно из стандарта:

100 Continue

101 Switching Protocols

200 OK

201 Created

202 Accepted

203 Non-Authoritative Information

204 No Content

205 Reset Content

206 Partial Content

300 Multiple Choices

301 Moved Permanently

302 Moved Temporarily

303 See Other

304 Not Modified

305 Use Proxy

400 Bad Request

401 Unauthorized

402 Payment Required

403 Forbidden

404 Not Found

405 Method Not Allowed

406 Not Acceptable

407 Proxy Authentication Required

408 Request Time-out

409 Conflict

410 Gone

411 Length Required

412 Precondition Failed

413 Request Entity Too Large

414 Request-URI Too Large

415 Unsupported Media Type

500 Internal Server Error

501 Not Implemented

502 Bad Gateway

503 Service Unavailable

504 Gateway Time-out

505 HTTP Version not supported

Поля заголовка ответа позволяют серверу передавать дополнительную информацию об отклике, дают информацию о сервере и доступе к запрашиваемому ресурсу:

Allow

Content-Base

Content-Language

Content-Length

Content-Location

Content-MD5

Content-Range

Content-Type

Expires

Last-Modified

Механизм расширения заголовка позволяет определить дополнительные поля заголовка.

Сервер обычно имеет некоторое значение таймаута, за пределами которого он уже не поддерживает неактивное соединение.

Клиент, сервер или прокси могут закрыть транспортный канал в любое время.

Отклики HTTP сервера могут кэшироваться. И этим кэшем можно управлять.

2. CGI интерфейсы

Обработка динамических запросов. Соглашения о связях, передача параметров.

CGI - Common Gateway Interface – есть стандарт для связи прикладных программ с сервером, поддерживающим HTTP протокол. Идея абсолютно прозрачна. В базовой схеме HTTP клиент посылает запрос, который всегда содержит указатель на некоторый файл. Что если, это не запрашиваемый документ, а имя программы, которая выработает документ в результате исполнения? То есть запрос реально используется для запуска указанной программы на сервере. Или, более реально, URI в запросе обозначает лишь некоторый ключ, по котрому будет определено имя запускаемой программы.

Задачей CGI является описание того, как передавать параметры запроса таким запускаемым программам.

Для передачи данных об информационном запросе от сервера к приложению, сервер использует командную строку и переменные окружения. Эти переменные окружения устанавливаются в тот момент, когда сервер выполняет программу приложения.

Информация из HTTP запроса передается приложению в следующей форме

имя=значение&имя1=значение1&..,

где имя- имя переменной, а значение - ее реальное значение. В зависимости от метода, который используется для запроса, эта строка появляется или как часть URL (в случае метода GET), или как содержимое HTTP запроса (метод POST). В последнем случае, эта информация будет послана программе в стандартный поток ввода.

Количество байт, передаваемых программе, описывается параметром CONTENT_LENGTH. Так же сервер передает приложению значение параметра CONTENT_TYPE (тип передаваемых данных).

Часть информации передается через переменные окружения. Это сделано для того, чтобы исключить зависимость от операционной системы и языка программирования CGI приложения (так называемые CGI переменные окружения):

SERVER_SOFTWARE – имя и версия сервера

SERVER_NAME – имя хоста (сервера)

GATEWAY_INTERFACE – версия спецификации CGI

SERVER_PROTOCOL – имя и верси протокола

SERVER_PORT – номер порта, на который был послан запрос

REQUEST_METHOD – метод HTTP протокола (GET, POST и т.д.)

PATH_INFO – дополнительная информация о пути, переданная в запросе

PATH_TRANSLATED – физическое отображение значения PATH_INFO

SCRIPT_NAME - виртуальный путь к приложению, выполняемому для получения URL

QUERY_STRING – данные следующие за символом ? в URL запроса. По стандарту эта информация никаким образом не декодируется.

REMOTE_HOST – имя хоста, выполняющего запрос

REMOTE_ADDR - IP адрес хоста, выполняющего запрос.

AUTH_TYPE – метод идентификации пользователя

REMOTE_USER – имя пользователя (при авторизации)

CONTENT_TYPE – тип передаваемых данных

CONTENT_LENGTH – размер данных

Если запрос содержит дополнительные поля заголовка запроса, они помещаются в переменные окружения с префиксом HTTP_, за которым следует имя заголовка. Любые символы '-' в заголовке меняются на символы подчеркивания '_'. Например:

HTTP_ACCEPT – список MIME типов, которые клиент может обработатьп

HTTP_USER_AGENT – имя агента

Сервер также может исключить любые дополнительные поля заголовка в случае превышения пределов размера переменных окружения.

CGI приложение осуществляет вывод информации в стандартный поток вывода. Этот вывод может представлять собой непосредственно документ, сгенерированный приложением, или указание серверу, где получить необходимый документ.

Заголовок выходного потока содержит текстовые строки, в том же формате, как и в HTTP заголовке и завершается пустой строкой (содержащей только символ перевода строки или CR/LF).

Любые строки заголовка, не являющиеся директивами сервера, посылаются непосредственно клиенту. CGI спецификация определяет следующие директивы сервера:

Status – результат выполнения HTTP запроса

Content-type - MIME тип возвращаемого документа.

Location – используется в том случае, когда вместо документа возвращается ссылка на него. Если значением является URL, то сервер передаст клиенту указание на перенаправление запроса. Если аргумент представляет собой виртуальный путь, сервер вернет клиенту заданный этим путем документ, так, как если бы клиент запрашивал его непосредственно.

За заголовком следуют возвращаемые данные.

3. Java servlets API

Структура Java servlets API. Описание сервлетов и их применение. Модель жизненного цикла. Основные методы Java Servlets API. Примеры использования. Использование Java Servlets в JEE приложениях.

Сервлеты представляют собой принятую в Java (JEE) модель (шаблон) для разработки приложений в рамках подхода клинет-сервер. Клиент посылает запрос – сервер вырабатывает некоторый ответ.

При этом запросы могут быть адресованы серверу в соответствии с разными протоколами (HTTP, FTP etc.) Отсюда возникла модель: Generic Servlet (пакет javax.servlet)

Servlet API представляет собой набор интерфейсов, которые и реализуются прикладной программой. Эти интерфейсы используются контейнером сервлетов для их запуска, остановки и выработки реакций на запросы.

Пакет javax.servlet.http поддерживает модель HTTP сервлетов. Здесь запросы рассматриваются как HTTP запросы. Сервлеты замещают CGI скрипты в архитектуре Java приложений. Отличия состоят в том что:

- запускаемые приложения есть Java классы
- выделена специальная архитектурная компонента – контейнер, которая и управляет работой сервлетов (CGI скритпы работают на уровне ОС)

Жизненный цикл сервлетов определен в модели Generic Servlet и описывается тремя методами:

init()

service()

destroy()

Метод init() вызывается контейнером при первой загрузке сервлета (сервлет, как и все в Java есть некоторый класс). Сервлеты могут загружаться:

- при поступлении первого запроса
- предварительно, при старте системы
- по инициативе контейнера до обработки запросов для обеспечения наилучшей производительности

В любом случае, при запуске однократно должен быть выполнен метод init(). Этот метод будет гарантированно завершен до начала обработки запросов. Иными словами, контенер гарантирует, что ни один запрос не будет обработан, до тех пор пока не завершится метод init()

В качестве параметра метод init() имеет один аргумент типа ServletConfig. Это объект, служащий для связи с контейнером. Таким образом, сервлет может, например, получать начальные параметры. Другим важным методом в классе ServletConfig является функция getServletContext(), которая возвращает объект типа ServletContext - глобальная память для элементов контейнера.

Metod service() является основным методом сервлетов. Собственно вся модель и состоит в том, что выделяется один единственный метод, который и выполняет базовую функцию.

Метод имеет два параметра:

- объект типа ServletRequest содержит данные запроса. Это пары имя/значение для параметров и поток ввода (InputStream).
- объект типа ServletResponse представляет данные, посылаемые клиенту

Метод destroy() вызывается контейнером перед выгрузкой сервлета. Вместе с тем, он может вызываться и в процессе выполнения длинных операций

Пакет javax.servlet.http поддерживает так называемые HTTP сервлеты. Здесь запрос пользователя есть HTTP запрос. Соответственно этому к методу service() добавлены аналогичные по семантике методы, просто соответсвующие отдельным командам HTTP протокола. В соответствии с этим, HTTP сервлет будет поддерживать методы:

doGet(), doPost() и так.

Соответствующие параметры имеют тип HttpServletRequest и HttpServletResponse. Первый из них содержит праметры запроса, второй – используется для выдачи результатов.

4. Java servlets filters и системные события

Фильтры и обработка системных событий в JEE. Пре- и пост-процессинг запросов. Виды системных событий и примеры работы с ними.

Фильтры, согласно Java Servlet specification version 2.3, динамически прерывают ответы или запросы и позволяют вставить пользовательский код в стандартную цепочку обработки запроса.

Фильтры позволяют добавить специальную обработку к уже существующим приложениям. Важно при этом, что не требуется доступа к исходному коду приложения.

В качестве примеров такой обработки можно назвать, например, следующие приложения:

- аутентификация пользователей
- ведение журналов и аудит
- конвертация и масштабирование изображений
- компрессия данных
- локализация
- преобразование данных (например, XSLT преобразования для XML документов)
- кэширование результатов запросов
- блокировка запросов

Пакет javax.servlet содержит интерфейс Filter. Соответственно, любой пользовательский класс должен реализовывать этот интерфейс. Основной метод – doFilter().

Идея реализции фильтров состоит в выстраивании цепочки пользовательских обработчиков (если у нас есть более одного фильтра), пропускающих через себя HTTP запросы. Полная спецификация этого метода выглядит так:

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException

первые два параметра несут туже семантику, что и в сервлетах: данные HTTP запроса и объект для связи с внешней средой. Третий параметр есть ссылка на “следующий” элемент в цепочке фильтров

Методы init () и destroy() интерфейса Filter несут туже семантику, что и одноименные методы для сервлетов.

Например, следующий фильтр устанавливает кодировку для запроса (код приведен в J2EE tutorial):

```
private FilterConfig filterConfig;
private String encoding;
```

```
public void doFilter(ServletRequest request,
  ServletResponse response, FilterChain chain) throws
  IOException, ServletException {
  if (encoding != null)
    request.setCharacterEncoding(encoding);

  // выполняем следующий шаг обработки
  chain.doFilter(request, response);
}
public void init(FilterConfig filterConfig) throws
  ServletException {
  this.filterConfig = filterConfig;
 // читаем параметр фильтра из web.xml:
  this.encoding = filterConfig.getInitParameter("encoding");
}
```

Идея обработки контента в фильтре (пост-процессинга) базируется на переопределении потока, куда элементы цепочки обработчиков будут помещать выходные данные. С точки зрения элементов цепочки ничего не меняется, они взаимодействуют с окружением также как и раньше (через свой объект ServletResponse), просто на первом шаге этот объект заменен другим объектом с тем же интерфейсом, но сохраняющим данные в доступном для чтения (обработки) виде. Этот подход соответствует шаблону Decorator в шаблонах объектно-ориентированного програмирования.

```
CharResponseWrapper wrapper = new CharResponseWrapper(
  (HttpServletResponse)response);
chain.doFilter(request, wrapper);

// здесь обрабатываем вывод
String content = wrapper.toString();
```

А наш декоратор выглядит так:

```
public class CharResponseWrapper extends
  HttpServletResponseWrapper {
  private CharArrayWriter output;
  public String toString() {
```

```
    return output.toString();
  }
  public CharResponseWrapper(HttpServletResponse response){
    super(response);
    output = new CharArrayWriter();
  }
  public PrintWriter getWriter(){
    return new PrintWriter(output);
  }
}
```

Как и сервлеты, фильтры должны быть описаны в файле web.xml (дескрипторе веб-приложения). Также как и для сервлетов в этот файл включаются два элемента – описание фильтра и описание условий его применения (filter-mapping по аналогии с servlet-mapping):

```
<filter>
  <filter-name>EncodingFilter</filter-name>
  <filter-class>com.myCompany.web.EncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>windows-1251</param-value>
  </init-param>
</filter>
```

и

```
  <filter-mapping>
    <filter-name>EncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

Спецификация Java Servlets 2.3 ввела понятие системных событий. Идея состоит во введении механизма callback для опеределенных моментов жизненного цикла приложения. Например:

- выполнить определенный пользователем код при создании новой сессии
- выполнить определенный пользователем код при изменении атрибута сессии

Разработчик веб-приложения готовит код (listener – в терминах JEE), который будет использоваться контейнером при наступлении соответствующего события.

События привязываются либо к глобальной области памяти (ServletContext), либо к сессиям.

Принципиально существует два типа событий:

- создание/уничтожение объекта
- изменение атрибутов объекта

Создание объекта типа ServletContext соответствует, очевидно, запуску приложения. Уничтожение такого объекта соответствует остановке приложения.

Создание объекта HttpSession связано, по определению, с заведением новой сессии. Если сессии создаются автоматически при обращении к JSP файлу, то это, по сути, означает поступление нового запроса.

Модификация атрибута, с практической точки зрения, есть, например, запоминание нового элемента в корзине покупателя в системе электронной коммерции. Данные, связанные с пользователем (включая его виртуальную корзину) реализуются с помощью сессий. И, соответственно, механизм системных событий озволяет отслеживать изменение параметров сессий.

Модификация атрибута на уровне глобальной памяти (ServletContext) может использоваться, например, когда в глобальном контексте сохраняются общие объекты. Например, соединение с базой данных.

Для собственного приложения, осуществляющего обработку системных событий, необходимо написать класс, реализующий интерфейс javax.Servlet.ServletContextListener и определить следующие методы:

void contextInitialized (ServletContextEvent sce)

void contextDestroyed (ServletContextEvent sce)

Названия говорят сами за себя – первый метод будет вызван при создании контекста, второй – при его уничтожении.

Переопределения для системных событий описываются в файле web.xml (тег listener):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>

  <!-- Define application events listeners -->
 <listener>
```

```
    <listener-class>
     com.listeners.MyContextListener
    </listener-class>
  </listener>

…
</web-app>
```

Аналогичные по семантике функции реализуются в интерфейсе javax.Servlet.ServletContextAttributesListener:

void attributeAdded (ServletContextAttributeEvent scab)

void attributeRemoved (ServletContextAttributeEvent scab)

void attributeReplaced (ServletContextAttributeEvent scab)

Для работы с сессиями поддерживается пара схожих интерфейсов. Интерфейс javax.Servlet.http.HttpSessionListener описывает методы, связанные с созданием и закрытием сессий:

void sessionCreated (HttpSessionEvent se)

void sessionDestroyed (HttpSessionEvent se)

Интерфейс javax.servlet.http.HttpSessionAttributesListener описывает работу с атрибутами сессий:

void attributeAdded (HttpSessionBindingEvent se)

void attributeRemoved (HttpSessionBindingEvent se)

void attributeReplaced (HttpSessionBindingEvent se)

5. JSP

Архитектура JSP. Синтаксис JSP: директивы, декларации, выражения, скриптлеты. Связь JSP и сервлетов

Если основной моделью использования Java сервлетов является использование языка разметки (например, HTML) в Java, то JSP возникли с обратной идеей: использовать Java внутри языка разметки.

Контенейр серлетов (JSP контейнер), получив по HTTP запрос на выдачу JSP файла, автоматически транслирует его в Java сервлет. Таким образом, каждому JSP файлу соответствует некоторый .class файл, который в действительности и обрабатывает пользовательский запрос. При поступлении запроса контейнер сравнивает даты модификации JSP файла (текста) и .class файла (байт-кода). Если исходный файл был модифицирован позднее, чем создан байт-код, то автоматически запускается компиляция. Иначе – просто запускается соответствующий сервлет.

JSP страница – это текстовая страница, где к базовому языку разметки (HTML, XML) могут быть добавлены некоторые динамические элементы. Динамические элементы оформляются либо как теги (дополняющие теги языка разметки), либо как фрагменты кода на языке Java.

Схема компиляции JSP страницы в соответствующий сервлет, на принципиальном уровне очень проста. Опуская детали, связанные с оптимизацией, мы можем представить это следующим образом:

В методе service( ) нашего сервлета содержатся операторы вывода данных для всех тегов языка разметки. Динамические теги JSP заменяются процедурными вызовами, фрагменты кода Java переносятся из JSP страницы в сервлет как есть.

Более точно, JSP добавляет к языку разметки следующие элементы:

JSP Expression

Формат: <%= выражение %>

В формате XML это имеет вид:

<jsp:expression>

expression

</jsp:expression>.

JSP Scriptlet

<%

Java код

%>

или в формате XML:

<jsp:scriptlet>

code

</jsp:scriptlet>.

JSP Declaration

<%! code %>

редко используемый элемент, тем не менее:

<%! String s; %> описывает, например, переменную s, которая будет декларирована на уровне класса, реализующего сервлет для JSP файла. В этом вся разница – подобного рода декларация описывает на уровне класса, а не на уровне метода service( )

Отличие от описания <% String s1; %> состоит в том, что переменная s будет сохранять свое значение между обращениями к JSP файлу (как переменная класса). А s1 в таком описании есть локальная переменная метода.

XML версия:

<jsp:declaration>

code

</jsp:declaration>.

Директива JSP page

<%@ page имя_атрибута="значение" %>

Эта директива задает атрибуты страницы. Примеры возможных атрибутов:

import="package.class"

contentType="MIME-Type"

isThreadSafe="true | false"

session="true | false"

buffer="sizekb | none"

autoflush="true | false"

extends="package.class"

info="message"

errorPage="url"

isErrorPage="true | false"

language="java"

Директива JSP include

<%@ include file="url" %>

позволяет включить (добавить) локальный файл в данную страницу на этапе компиляции. XML эквивалент:

<jsp:directive.include file="url"\>.

JSP комментарий

<%-- комментарии --%>

В отличие от директив, JSP теги описывают действия периода исполнения (не периода компиляции).

Тег jsp:include

<jsp:include page="relative URL" flush="true"/>

Тег позволяет включить страницу в результирующий документ во время выполнения. При этом значение атрибута, описывающего URL для включаемой страницы, вычисляется непосредственно перед выполнением. Дополнительный атрибут *flush* позволяет сбросить данные в поток после выполнения операции.

Тег jsp:useBean

Создает или находит в соответствующем контексте необходимый объект

<jsp:useBean список атрибутов/>

возможные атрибуты:

id – идентификатор (тип) объекта

scope – область видимости (page | request | session | application)

class – класс объекта

type – тип объекта

Тег jsp:setProperty

Устанавливает значение для атрибута (атрибутов) объекта

<jsp:setProperty список_атрибутов />

name – имя объекта

property – имя свойства

param – имя параметра HTTP запроса

value - значение

Тег jsp:getProperty

Читает (печатает) свойства объекта

<jsp:getProperty name="имя_объекта" property="имя_свойства" />

Тег jsp:forward

Передает управление на указанную страницу

<jsp:forward page="относительный адрес новой страницы"/>

Значения атрибутов в тегах JSP могут задаваться динамически. Можно использовать JSP выражения в качестве значения атрибутов. Например,

<%

String nextPage=”next.jsp”;

%>

<jsp:forward page="<%=nextPage%>"/>

При трансляции JSP файлов в сервлет, фрагменты Java кода переносятся в результирующий код. При этом:

- выражения <%= expression %> вычисляются, и в результирующий код вставляется результат

- код скриптлетов <% Java код %> переносится в метод sefvice() результирующего сервлета “как есть”

- описания <%! code %> переносятся в результирующий код “как есть” и включаются туда на уровне определения класса (не в методе service())

Отметим, что компиляция Java кода для результирующего сервлета происходит, естественно, уже после полной обработки всего JSP файла. Отсюда следует, в частности, что отдельный скриптлет не обязан содержать синтаксически правильный (законченный) фрагмент кода. Главное, чтобы правильный код был построен в результате. Второе, что из этого следут – это возможность ввести предопределенные переменные. Технически это означает, что при генерации кода сервлета мы можем изначально ввести там описания для каких-то переменных. Тогда, естественно, эти описания будут доступны для использования и во всех фрагментах кода, переносимых из JSP.

К числу таких преопределенных объектов относятся:

request – переменная, соответствующая объекту типа HttpServletRequest

response – переменная, описывающая объект типа HttpServletResponse

session – переменная типа HttpSession, описывающая текущую сессию

out – переменная типа PrintWriter. Дескриптор, используемый для вывода данных

application – глобальная область памяти (ServletContext)

config – дескриптор конфигурации (объект типа ServletConfig) для результирующего сервлета

pageContext – коллекция объектов, ассоциированных с данной страницей

6. Пользовательские теги JSP

Расширение набора тегов в JSP. Типы тегов и принципы их обработки. Описание использования. Примеры применения.

Пользовательские теги в JSP – это возможность расширения JSP тегами, огранизованными по тому же принципу, что и стандартные теги (типа <jsp:include). Идея постоения тегов очень проста. Любой тег может быть представлен стандартным образом:

Заголовок тега

Тело тега

На этом стандартном представлении мы отмечаем позиции, где компилятор JSP в процессе разбора JSP тега будет вызывать опередеренные нами функции, которые и детализируют, каким образом обрабатывать наш тег.

Например:

Заголовок тега (1)

(2) начало тела тега

конец тела тега (3)

закрытие тега (4)

Здесь:

(1) определяемая пользователем функция, которая вызывается после разбора заголовка

(2) определяемая пользователем функция, которая вызывается перед началом рабора тела тега

(3) определяемая пользователем функция, которая вызывается после окончания разбора тела тега

(4) определяемая пользователем функция, которая вызывается после завершения тега

Собственно тела тега представляет собой JSP код и разбирается JSP компилятором обычным образом. Заговок тега также выглядит “стандартно”: это префикс, имя тега и набор пар: имя атрибута = значение.

Соответственно, для JSP компилятора мы должны указать описание тега – какой префикс мы будем использовать, как называется тег, какие у него есть атрибуты.

Пользовательские теги могут быть следующих типов: без тела и содержащие тело. В качестве тела тега рассматривается произвольный JSP (HTML) код. Например:

<prefix:tagName attributeName="value" anotherAttributeName="anotherValue"/>

аналог в HTML:

```
<IMG SRC="/directory/file.gif">
```

при наличии тела тега:

```
<prefix:tagName attributeName="value" anotherAttributeName="anotherValue"/>
        ...tag body...
</prefix:tagName>
```

аналогии с HTML:

```
<H2>Custom Tags</H2>
```

С практической точки зрения в теле пользовательского тега можно запретить использовать скриптлеты, а также полностью запретить его интерпретацию, рассматривая тело как просто текст.

С точки зрения программирования, пользовательский тег есть Java класс, реализующий некоторый стандартный интерфейс. Поэтому с принципиальной точки зрения, теги не могут предложить больше функциональности, чем прямое использование скриптлета. Их идея состит в другом:

- теги сокращают использование Java кода в страницах
- теги упрощают разработку страниц
- теги обеспечивают (упрощают) переиспользование кода
- теги позволяют создать для разработчиков страниц (дизайнеров) некоторое проблемно-ориентированное расширение JSP

Стандартный интерфейс Tag выглядит следующим образом:

```
public interface Tag {
  public final static int SKIP_BODY = 0;
  public final static int EVAL_BODY_INCLUDE = 1;
  public final static int SKIP_PAGE = 5;
  public final static int EVAL_PAGE = 6;
```

```
  void setPageContext(PageContext pageContext);
  void setParent(Tag parent);
  Tag getParent();
  int doStartTag() throws JspException;
  int doEndTag() throws JspException;
  void release();
}
```

Здесь перечислены все методы, которые вызываются JSP компилятором при интерпретации тега

Каждому тегу, помимо Java кода, должно быть поставлено в соответствие некоторое XML описание (tag library descriptor). Это описание сообщает JSP компилятору, какой класс реализует тег, есть у тега тело или нет, какие атрибуты возможны и какие являются обязательными:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
      PUBLIC "-//Sun Microsystems, Inc.//
      DTD JSP Tag Library 1.1//EN"
      "http://java.sun.com/j2ee/dtds/
      web-jsptaglibrary_1_1.dtd">

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>Example</shortname>
  <uri>http://www.mycompany.com/example</uri>
  <info>A simple tab library</info>

 <tag>
   <name>hello</name>
   <tagclass>com.company.tags.HelloTag</tagclass>
   <bodycontent>empty</bodycontent>
   <info>just a comment</info>

 <attribute>
    <name>name</name>
```

```
    <required>false</required>
    <rtexprvalue>false</rtexprvalue>
  </attribute>
  </tag>
</taglib>
```

В JSP странице должно присутствовать описание тега. Например:

```
<@ taglib uri="/WEB-INF/jsp/mytaglib.tld"  prefix="first" %>
```

Поддержка параметров сводится к определению set/get методов для класса.

Следующий шаг после стандартных интерфейсов – введение адаптеров. Классы TagSupport и BodyTagSupport содержат реакции по умолчанию для всех переопределяемых методов. Это сокращает время разработки тегов – нужно переопределить только реально используемые методы.

7. JSTL, JSF

Стандартная библиотека тегов, EL-выражения. Основные теги и примеры использования. Принципы построения Java Server Faces

JSTL (The JavaServer Pages Standard Tag Library), сообразно своему названию представляет собой стандартную библиотеку тегов для JSP. Слово “стандартный” здесь нужно поннимать в том смысле, что данная бибилиотека являетс частью стандарта JEE (описывается спецификацией JSR-52), и Вы можете ожидать ее наличия в каждом J2EE сервере (контейнере сервлетов).

JSTL представляет собой совокупность 4 отдельных бибилиотек:

- core action (базовые операции)
- XML processing (работа с XML документами)
- internationalization (создание многоязыковых приложений)
- database access (доступ к базам данных)

Также, спецификация JSTL добавила к JSP так называемый язык выражений (EL - expression language).

JSTL требует поддержки контейнером спецификаций Java Servlet 2.3 и JavaServer Pages 1.2 specifications.

Описание библиотеки в теле JSP страницы может проволиться обычным образом:

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

Вот типичный пример использования (теги JSTL core и выражения):

```
<c:set var="browser" value="${header['User-Agent']}"/>
<c:out value="${browser}"/>
```

Здесь в первой строке определяется переменная с именем browser, которая инициализируется значением заголовка User-Agent. Во второй строке на страницу (в поток) выводится значение этой переменной.

Естественно, эти действия могут быть заменены скриптлетом:

```
<%
 String browser = request.getHeader(“User-Agent”);
out.println(browser);
%>
```

Основная задача JSTL - упростить написание кода и автоматизировать часто встречающиеся последовательности действий. Так, c:out выполняет еще полезные операции по автоматическому построению escape-последовательностей.

Идеей EL является упрощение описания доступа к данным. Например:

Скриптлет: <%=someVariable%>

EL: ${someVariable}

EL позволяет атоматически определить подходящий объект в следующих примерах:

<someTags:aTag attribute="${aName}">

вместо

<someTags:aTag attribute="<%= pageContext.getAttribute("aName") %>">

или

${aCustomer.address.country}

вместо

<%= aCustomer.getAddress().getCountry() %>

Начиная с версии JSP 2.0, EL является стандартной возможностью.

EL поддерживает следующий набор операций:

+	сложение
-	вычитание
*	умножение
/ или div	деление
% или mod	вычисление остатка
= =	равенство
!=	неравенство
<	меньше
>	больше
<=	меньше или равно
>=	больше или равно
&&	логическое И
||	логическое ИЛИ

! или not отрицание

empty проверка пустого значения

Также в системе поддерживается набор встроенных объектов:

applicationScope - коллекция глобальных объектов

cookie - коллекция cookies из текущего запроса

header – коллекция HTTP заголовков текущего запроса

headerValues коллекция значений для HTTP заголовков

initParam коллекция начальных параметров

pageContext The javax.servlet.jspPageContext текущей страницы

pageScope коллекция всех объектов страницы

param коллекция параметров запроса

paramValues коллекция значений параметров запроса

requestScope коллекция всех объектов запроса

sessionScope коллекция всех объектов сессии

Например: ${param.qty} вместо <%request.getParameter(“qty”)%>

JSTL поддерживает следующий набор библиотек: Core, XML Processing, Internationalization и Database Access. Core включает набор базовых функций, назначение остальных библиотек понятно из их названия.

Библиотеки описываются обычным образом (как и любые другие библиотеки). Например,

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

Core Tag Library включает следующие теги: вывод данных, работа с переменными, условные операторы, циклы, операции с URL и обработка ошибок.

Например:

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

выводит значение customer.getCity() или unknown, если указанное значение есть null

Тег

<c:if test="${customer.city == 'Moscow'}">

  HTML (JSP) код

</c:if>

выполняет собственное тело, если customer.getCity() есть Moscow.

Одна из часто встречающихся задач – перебор данных в некоторой коллекции. Например, в своей JSP странице Вы формируете HTML список элементов. Тег forEach позволяет делать это следующим образом:

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

Тег <c:forEach> просто перебирает элементы коллекции

Операции с URL позволяют, например, импортировать данные с других сайтов. Стандартный тег <jsp:include> позволяет включать только локальные ресурсы (из того же самого контекста, если точно). JSTL тег import позволяет преодолеть это ограничение:

```
<c:import url="http://www.host.com/index.html"/>
```

XML теги JSTL облегчают работу с XML документами. Сюда входит, в первую очередь, разбор XML документов и обработка запросов по XPath.

Напрмер:

1) <x:parse xml="${xmltext}" var="output"/>

разбор XML текста, доступного через переменную xmltext.

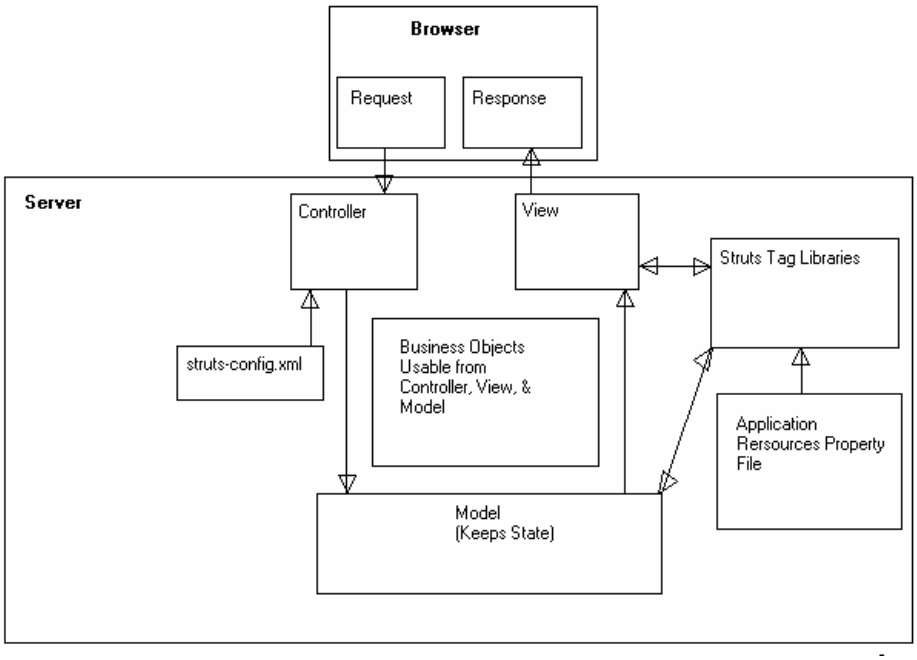
2) <x:out select="$output/books/book[1]/title"/>

в разобранном документе найти тег books, в нем – первый из тегов book и прочитать значение тега title.

8. Web-frameworks, Struts

Шаблон MVC (Model View Controller) и его использование. Пакет Struts. Основные возможности и примеры использования. Обзор популярных подходов в разработке web приложений: WebWork, Tapestry etc.

Общая схема Struts-приложений



Разделы, включаемые в рассмотрение:

Шаблон Model-View-Controller

Разработка уровня Model

Компоненты уровня View

Разработка уровня Controller

Библиотеки тегов Struts

Конфигурационные файлы Struts

Ресурсные файлы

Список литературы и веб-источников

1. Пол Дж. Перроун, Венката Венката С. Р. "Кришна" Р. Чаганти Создание корпоративных систем на основе Java2 Enterprise Edition. Руководство разработчика. ISBN: 5845901685, 0672317958, 2001 г. Издательство: “Вильямс”

2. Марти Холл, Лэрри Браун. Программирование для Web. Библиотека профессионала, 2001 г. Издательство: “Вильямс”

3. Х. М. Дейтел, П. Дж. Дейтел, С. И. Сантри
Технологии программирования на Java 2. Книга 3. Корпоративные системы, сервлеты, JSP, Web-сервисы Advanced Java 2 Platform. How to Program
Издательство: Бином-Пресс, 2003 г.

4. Кришнамурти, Дж. Рексфорд
Web-протоколы. Теория и практика. HTTP/1.1, взаимодействие протоколов, кэширование, измерение трафика
Издательство: Бином, 2002 г.

5. Сью Шпильман JSTL. Практическое руководство для JSP-программистов
Издательство: КУДИЦ-Образ, 2004 г.

6. Java портал Sun Microsystems: http://java.sun.com

Список вопросов, выносимый на экзамен

1. Язык программирования Java: структура и базовые принципы
2. Пакеты Java (JDK)
3. Протокол HTTP
4. Поддержка сессий (cookie, URL rewriting)
5. Общая организация/структура Java servlets
6. Огранизация контейнеров для поддержки Java servlets
7. Основные классы Java servlets development kit
8. Обработка динамической информации в Java servlets
9. Общая организация/структура JSP
10. Теги (команды) JSP
11. JSP taglib
12. Разработка JSP приложений. 3 tier model, MVC.