**Java Server Pages** представляет технологию, которая позволяет создавать динамические веб-страницы.

По сути Java Server Page или JSP представляет собой html-код с вкраплениями кода Java. В то же время станицы jsp - это не стандартные html-страницы. Когда приходит запрос к определенной странице JSP, то сервер обрабатывает ее, генерирует из нее код html и отправляет его клиенту. В итоге пользователь после обращения к странице JSP видит в своем браузере обычную html-страницу.

***Главное отличие****между JSP и Servlet* является то, что **JSP - это язык сценариев веб-страниц, который может генерировать динамический веб-контент, в то время как сервлет - это Java-программа, которая уже скомпилирована и используется для создания динамического веб-контента.**

**Структура jsp**

Страница JSP – текстовый документ, обычно имеющий расширение jsp и содержащий данные двух типов: 1) статические данные, как  
правило, в формате HTML, XML, JavaScript; 2) динамические элементы – фрагменты java-кода. Может показаться странным, что JavaScript  
относится к статическим данным, но это потому, что динамика рассматривается относительно сервера. Web-сервер не интерпретирует  
код JavaScript, а лишь пересылает его клиенту

Любая jsp-страница, кроме обычных html-тегов, содержит специальные jsp-теги следующих категорий: директивы, объявления, скриплеты, выражения и комментарии

**Жизненный цикл JSP**

Жизненный цикл JSP состоит из нескольких фаз:

1. **Translation** – JSP контейнер проверяет код JSP страницы, парсит ее для создания кода сервлета. К примеру, в Tomcat вы можете найти классы сервлетов в директории **TOMCAT/work/Catalina/localhost/WEBAPP/org/apache/jsp**. Если страница JSP называется home.jsp, то созданный сервлет обычно будет иметь имя **home\_jsp** и имя файла — **home\_jsp.java**.
2. **Compilation** – JSP контейнер компилирует исходный код jsp класса и создает класс на этой фазе.
3. **Class Loading** – контейнер загружает классы в память на этой фазе.
4. **Instantiation** – внедрение конструкторов без параметров созданных классов для инициализации в памяти классов.
5. **Initialization** – в контейнере вызывается init метод объекта JSP класса и инициализируется конфигурация сервлета с init параметрами, которые указаны в дескрипторе развертывания (**web.xml**). После этой фазы JSP способен обрабатывать запросы клиентов.Обычно эти фазы происходят после первого запроса клиента (т.е. ленивая загрузка), но можно настроить загрузку и инициализацию JSP на старте приложения по аналогии с сервлетами.
6. **Request Processing** – длительный жизненный цикл  обработки запросов клиента JSP страницей. Обработка является многопоточной и аналогична сервлетам — для каждого запроса создается новая нить, создаются объекты ServletRequest и ServletResponse и происходит внедрение сервис методов JSP.
7. **Destroy** – последняя фаза жизненного цикла JSP на которой JSP класс удаляется из памяти. Обычно это происходит при выключении сервера или андеплое приложения.

Содержимое страницы JSP фактически делится на код html (а также css/javascript) и код на языке java. Для вставки кода Java на страницу JSP можно использовать пять основных элементов:

* Выражения JSP (JSP Expression)
* Скриплет JSP (JSP Scriplet)
* Объявления JSP (JSP Declaration)
* Директивы JSP (JSP Directive)
* Комментарии JSP

JSP Выражения представляет выражение, заключенное между тегами <%= и %>. При обращении к JSP вычисляется значение этого выражения.

<p>2 + 2 = <%= 2 + 2 %></p>

### JSP Scriplet

JSP Scriplet представляет одну или несколько строк на языке Java. Скриплет заключается внутри следующих тегов:

<%

            for(int i = 1; i < 5; i++){

                out.println("<br>Hello " + i);

            }

%>

### JSP Объявления

JSP Declaration позволяют определить метод, который мы затем можем вызывать в скриплетах или в JSP-выражениях. Определение метода помещается между тегами <%! и %>.

<%!

    int square(int n){

        return n \* n;

    }

%>

### Комментарии

Комментарии JSP добавляются с помощью тега <%-- Текст\_комментария --%>:

### Директивы

Директивы предоставляют информацию контейнеру JSP, необходимую на стадии трансляции и имеют следующий синтаксис.  
**<% @ директива имя\_атрибута\_1 = “значение”  
имя\_атрибута\_2 = “значение” … %>**Существует три типа директив***:* page**, **taglib** и **include**

Директива page определяет свойства страницы JSP  
Значение атрибута language директивы page определяет язык (в  
наших лабах Java), используемый в скриплетах (фрагментах программного  
кода), в выражениях или других включаемых файлах. Значение атрибута contentType устанавливает MIME-тип ответа и кодировку страницы

**<%@ page language="java"  
contentType="text/html; charset=ISO-8859-1" %>**

<%@ page import="java.text.PavelRudenia" %>

Директива taglib указывает на то, что в странице JSP будут использоваться библиотеки тегов.

Директива include позволяет вставлять текст и код в процессе  
трансляции jsp-страницы.

<jsp:include page="night.jsp" />

Следует отметить существенное отличие между директивой  
**<%@ include %>** и стандартным действием **<jsp:include>**. Если  
директива внедряет в jsp-страницу текст до компиляции страницы,  
то стандартное действие внедряет результат исполнения вызываемого объекта (сервлет, html или jsp-страница) после компиляции на  
этапе исполнения.

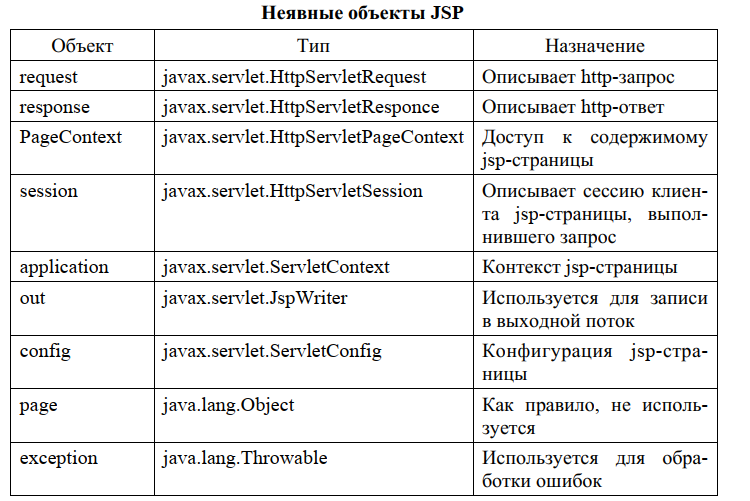
Действие **<jsp:forward>** используется для пересылки текущего запроса на другой ресурс, такой как HTML-страница, страница JSP или сервлет. Вот краткий пример:

<jsp:forward page="Another.jsp" /

При попытке перейти на страницу с jsp:forward нас сразу перекинет на страницу указанную в параметрах

***Неявные объекты*** (Implicit objects) – это объекты, автоматически  
доступные в скриплетах JSP без специального их объявления или импорта.

Значит явные обьекты –это обьекты которые мы через import добавляем в начале jsp страницы



Виды памяти которые используют сервлеты: контекст приложения, запрос, сессия

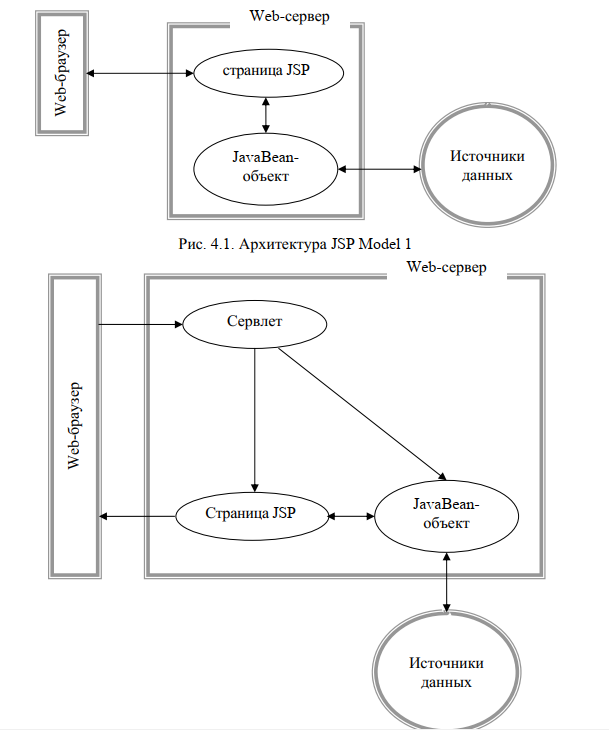
[**https://metanit.com/java/javaee/3.8.php**](https://metanit.com/java/javaee/3.8.php)

Нередко страница jsp обрабатывает запрос вместе сервлетом. В этом случае сервлет определяет логику, а jsp - визуальную часть. И при обработке запроса сервлет может перенаправить дальнейшую обработку странице jsp. Соответственно может возникнуть вопрос, как передать данные из сервлета в jsp?

Есть несколько способов передачи данных из сервлета в jsp, которые заключаются в использовании определенного контекста или scope. Есть несколько контекстов для передачи данных:

* **request** (контекст запроса): данные сохраняются в HttpServletRequest
* **session** (контекст сессии): данные сохраняются в HttpSession
* **application** (контекст приложения): данные сохраняются в ServletContext

Данные из контекста запроса доступны только в пределах текущего запроса. Данные из контекста сессии доступны только в пределах текущего сеанса. А данные из контекста приложения доступны постоянно, пока работает приложение.



# 5 Библиотека тегов

С точки зрения разработчика web-приложения ***библиотека тегов***(Tag Library) – это технология, позволяющая создавать собственные  
теги (будем их далее называть tdl-тегами), которые потом можно использовать в jsp-страницах.

Для того чтобы воспользоваться этой технологией, необходимо  
выполнить следующее:  
1) создать дескриптор библиотеки тегов (Tag library descriptor,  
TDL) и поместить его в директорий приложения;  
2) создать обработчики тегов (Tag handler) – java-классы, генерирующие html-текст, замещающий tdl-теги, в выходном потоке jspстраницы;  
3) поместить на jsp-странице директиву **taglib**, указывающую на  
месторасположение дескриптора библиотеки тегов и задающую префикс (пространство имен) для имен tdl-тегов в данной странице;  
4) добавить tdl-теги в jsp-страницу.

Тег **<taglib>** открывает описание библиотеки, которое располагается до закрывающего тега **</taglib>**. Описание библиотеки состоит из  
пролога и описаний tld-тегов библиотеки.  
Пролог содержит теги **<taglib-version>** для установки версии пользовательской библиотеки (в нашем случае установлена версия 1.0),  
**<jsp-version>** для указания применяемой спецификации JSP (в примере  
1.2), **<short-name>** для символического обозначения (наименования)  
библиотеки (в примере – **StaffTag**) и **<uri>**, содержащего идентификатор ресурса библиотеки тегов (в примере – **StaffTag.tld**).  
Описание каждого tld-тега библиотеки начинается с тега **<tag>** и  
заканчивается закрывающим тегом **</tag>**. В примере приводится  
описание двух различных тегов.

Обработчик тега – это java-класс, реализующий один из интерфейсов  
**Tag**, **IterationTag** или **BodyTag**. На практике обработчики тега представляют собой расширение одного из классов **TagSupport**, **BodyTagSupport**или **TagExtraInfo**, которые реализуют эти интерфейсы.  
На рис. 4.16 приводится пример обработчика tld-тега **surname**,  
описанного в дескрипторе библиотеки тегов **Stafftag** (рис. 4.15).  
Обработчик **Surname** расширяет класс **TagSupport** и реализует  
три метода: **doStartTag**, **setValue** и **getVlaue**.

Метод **doStartTag** предназначен для генерации текста, замещающего начальный тег.  
В данном примере tld-тег **<surname>** не имеет тела, поэтому начальный тег является единственным. Генерация текста осуществляется с помощью объекта **JspWriter**, ссылка на который может быть  
получена с помощью неявного объекта **pageContext**, доступного в  
классе-обработчике. Следует сразу отметить, что в обработчике могут использоваться все неявные объекты JSP.  
Метод **setValue** используется контейнером JSP, вызывающим его  
при разборе jsp-страницы для передачи значения атрибута **value**, указанного в теге **<surname>**. Имя **setValue** образовано как конкатенация  
префикса **set** и суффикса, совпадающего с именем атрибута **value**.  
Метод **getValue** используется в обработчике для доступа к значению атрибута в обработчике. Имя **getValue** образуется по тому же  
принципу, что и в предыдущем случае.  
Метод **doEndTag** предназначен для генерации текста, замещающего конечный тег. Как и прежде, генерация осуществляется с помощью объекта **JspWriter**.

# 6. Применение параметров инициализации и атрибутов контекста

# Сервлеты и jsp-страницы, входящие в состав web-приложения, исполняются в специальной среде, называемой web-контейнером, и не могут (вернее, это запрещено спецификацией) взаимодействовать с внешним миром и между собой непосредственно. Более того, всем жизненным циклом сервлетов и jsp-страниц управляет контейнер, вызывая реализованные ими методы стандартных интерфейсов. Любое нарушение этих правил приводит к утрате свойства переносимости приложения. Разработчику web-приложения контейнер представляется в виде предопределенных объектов, которые передаются в виде параметров, ссылок или неявных объектов. На рис. 5.1 изображена объектная модель среды выполнения web-приложения.

# Рис. 5.1. Модель среды выполнения web-приложения

# Объект Request создается контейнером при получении httpзапроса к компоненту web-приложения и инкапсулирует всю необходимую информацию о запросе клиента. Этот объект существует и доступен только в рамках обработчика запроса (в нашем случае сервлета или jsp-страницы). Объект Response создается контейнером тоже при получении запроса. Методы этого объекта позволяют инкапсулировать всю информацию, необходимую контейнеру, для того, чтобы сформировать и передать ответ клиенту. Этот объект существует и доступен только в рамках обработчика запроса. Объект Context создается контейнером при его инициализации на основе дескриптора развертывания приложения. Помимо общего контекста создаются контексты для каждого сервлета и jsp-страницы

# Объект Session создается контейнером при получении первого запроса клиента. Объект существует до тех пор, пока интервал времени между последовательными запросами клиента не превысит установленное пороговое значение. Для обозначения каждой сессии используется специальный уникальный идентификатор, позволяющий разделить запросы разных клиентов

**параметры инициализации** (*статический* элемент контекста)

**атрибуты контекста** (*динамический* элемент контекста)

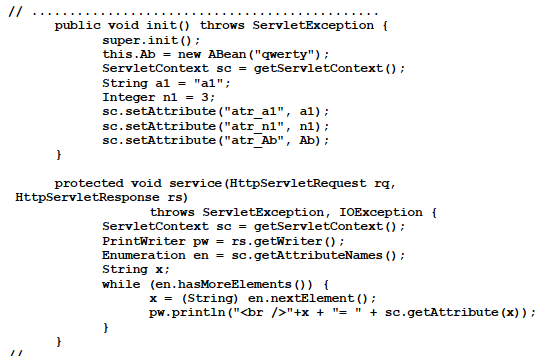
**параметры инициализации**

Параметры инициализации контекста web-приложения описываются в дескрипторе web-приложения и используются для его настройки.  
Например, значением этого параметра может быть URL базы данных  
или внешнего для web-приложения сервлета. Параметры инициализации следует использовать для хранения данных, которые нецелесообразно «зашивать» в программный код из-за того, что они могут измениться при установке и перенастройке web-приложения.

Описание параметров инициализации контекста в дескрипторе  
приложения должно предшествовать описанию сервлетов и JSP. Каждый параметр описывается в дескрипторе развертывания приложения  
с помощью тегов, заключенных между тегами **<contect-param>** и  
**</contect-param>**.  
Тег **<description>** не является обязательным и используется для  
комментариев. Теги **<param-name>** и **<parm-value>** используются для  
задания имени и значения параметра инициализации.Именно это имя используется в качестве параметра при вызове метода **getInitParameter** для получения значения параметра

**Атрибуты контекста**

Атрибуты являются динамическими элементами контекста и могут быть установлены, изменены или удалены в методах сервлетов  
или в скриплетах JSP



В методе **init** сервлета осуществляется установка значений атрибутов с помощью метода контекста **setAttribute**. Первый параметр  
этого метода задает имя атрибута, второй – его значение. Обратите  
внимание на атрибут с именем **atr\_Ab**, его значением является ссылка  
на объект java-класса с именем **ABean**.  
В методе **service** сервлета с помощью методов контекста  
**getAttributeNames** и **getAttribute** в окно браузера выводится имена  
всех атрибутов и их значения. Следует знать, что некоторые атрибуты  
могут создаваться и устанавливаться самими web-контейнером

Метод **setAttribute** предназначен как для создания атрибута, так и  
для его корректировки. При необходимости удалить атрибут используют метод **removeAttribute**.

# Lab 7Сеанс связи (сессия) Протокол HTTP является stat less протоколом, т. е. протоколом, не сохраняющим информацию о своем состоянии. Это означает, что каждый запрос и ответ имеют свой жизненный цикл, никак не связанный с предшествующими им запросами и ответами. Поэтому управление сеансом связи с пользователем является важной и нетривиальной задачей. Объект Session (сеанс связи, или сессия) реализует интерфейс HttpSession и служит для представления пользователя, работающего с клиентской частью web-приложения. Объект сессии создается, как правило, web-контейнером и становится доступным в сервлете или JSP с помощью метода getSession объекта Request. Время существование сессии зависит от параметра Session timeout, который может быть установлен в дескрипторе развертывания или в конфигурационных файлах сервера и может быть изменен динамически, например в сервлете. Этот параметр устанавливает максимальное время между запросами одного клиента. Превышение этого значения приводит к удалению объекта Session. В принципе, объект сессии может разрушить и изнутри, выполнив метод invalidate интерфейса HttpSession. Во время сеанса связи любой объект, связанный с сеансом связи, доступен любому сервлету или JSP, находящемуся в этом же контексте

# и недоступен для сервлетов и JSP другого контекста. Состояние сеанса позволяет отследить два специальных механизма: cookies и URL rewriting. В первом случае информация о сессии записывается в специальном файле на компьютере клиента, во втором случае такая же информация записывается непосредственно в URL каждого вызова. Идентификатор сессии С каждой сессией связан специальный уникальный идентификатор, который может быть получен с помощью метода getId интерфейса HttpSession. Идентификатор представляет собой строку, состоящую из 32 символов. Все приемы программирования, направленные на сохранение информации о состоянии сессии, в итоге опираются на использование этого идентификатора. Атрибуты сессии Как и с контекстом приложения и запросом, с сессией (с объектом, реализующим интерфейс HttpSession) можно динамически связать атрибуты. Создание, корректировка и удаление атрибутов осуществляется точно таким же способом, как и в предыдущих случаях. Единственным отличием является то, что методы, выполняющие эти операции, являются собственностью интерфейса HttpSession. На рис. 5.20 представлен фрагмент сервлета, создающего атрибут сессии, имеющего имя, которое совпадает с идентификатором сессии.

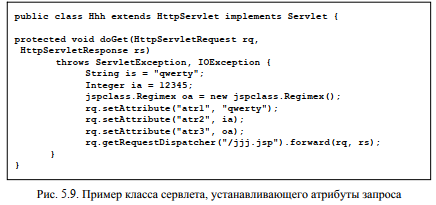
# 

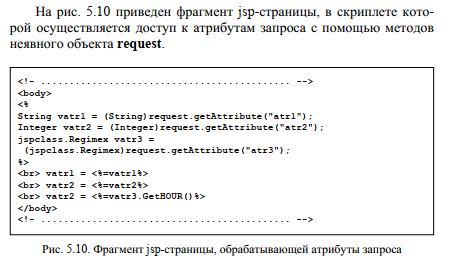
# Перед созданием осуществляется проверка на существование атрибута с таким именем, а в качестве значения атрибута используется ссылка на java-объект. На рис. 5.21 представлена страница JSP, на которую выполняется переопределение запроса из сервлета, рассмотренного в предыдущем примере. Объект сессии в jsp-странице становится доступным с помощью метода getSession неявного объекта request

# 

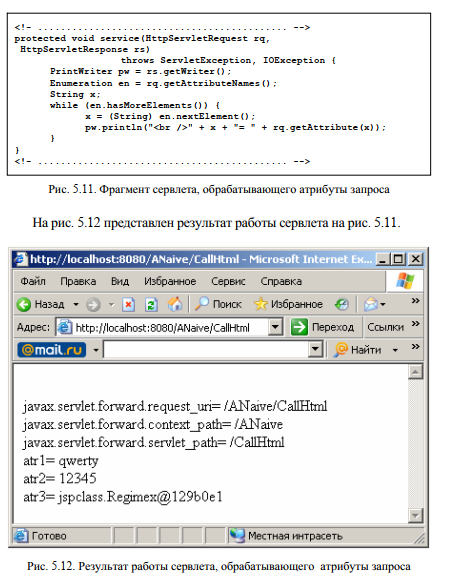
# Файлы cookies *Cookies* – это небольшие фрагменты текстовой информации, которым обменивается web-сервер и браузер в процессе установки соединения. Получая информацию в виде cookies от браузера, сервер может «узнать» клиента и предоставить ему заранее обусловленный перечень услуг. Объем cookie ограничен 4 Кб. Для отправки cookie от сервлета браузеру используется метод addCookie интерфейса HTTPServletResponse. Извлечь cookie, отправленный браузером, можно с помощью метода getCookie интерфейса HTTPServletRequest. Работа с cookie поддерживается классом Cookie.

# Lab 8 Атрибуты запроса

Атрибутами запроса могут быть любые объекты, ассоциируемые  
с запросом. Для доступа к атрибутам запроса используются следующие методы интерфейса **HttpServletRequest**: **getAttribute** (получить  
значение атрибута), **getAttribute** (установить значение атрибута),  
**getAttributeNames** (получить список имен атрибутов), **removeAtribute** (удалить атрибут).  
На рис. 5.9 приведен фрагмент сервлета, устанавливающего значение  
трех атрибутов – **atr1**, **atr2** и **atr3** перед выполнением переопределения  
запроса на jsp-страницу. При вызове таким способом страницы JSP ей передается объект запроса, с помощью методов которого в скриплете jspстраницы доступны ассоциированные с запросом атрибуты. 

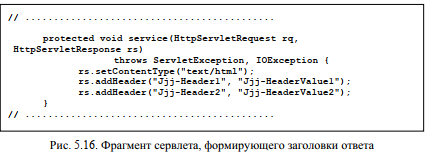


Другой важный момент – это применение в качестве значения атрибута ссылки на объект. В этом примере атрибут с именем **atr3** является ссылкой на объект типа **Regimex**. После приведения результата  
выполнения метода **getAttribute** к типу **Regimex** можно использовать  
все методы этого класса.  
На рис. 5.11 представлен фрагмент сервлета, выводящего в стандартный поток все существующие атрибуты запроса и их значения. Список  
всех атрибутов получен с помощью метода **getAttributeNames**, который  
возвращает объект типа **Enumeration**, позволяющий организовать цикл  
перебора всех имен атрибутов.



При задании имен атрибутов соблюдается соглашение о наименовании пакетов в языке Java. Кроме того, имена атрибутов, начинающиеся с префиксов «java.» или «javax.», зарезервированы для определения атрибутов спецификации сервлетов, а имена с префиксом «sun.» и «com.sun» зарезервированы компанией Sun Microsystems для собственных атрибутов.  
Обратите внимание на то, что помимо установленных в сервлете  
(рис. 5.9) значений атрибутов, в окно браузера выведены имена и значения еще трех атрибутов запросов, созданных контейнером.

**Заголовки ответа**Как и http-запрос, http-ответ всегда можно дополнить собственными заголовками, которые потом могут быть получены и обработаны на стороне клиента. На рис. 5.16 изображен фрагмент сервлета,  
формирующего заголовки ответа.  
В общем случае заголовку с одним именем может соответствовать  
несколько значений. Заголовки могут быть сформированы с помощью  
метода **addHeader**, как это сделано в примере на рис. 5.16, или метода  
**setHeader**. В первом случае заголовок создается (если заголовка с таким именем нет) или добавляется следующее значение (если такой заголовок уже есть). Метод **setHeader** тоже создает новый заголовок  
(если заголовка с таким именем нет) или заменяет значение.



Доступ к заголовкам ответа осуществляется точно по такому же  
принципу, что и к заголовкам запроса. На рис. 5.17 приводится вывод  
заголовков ответа, полученных с помощью метода **getHeaderNames** и  
**getHeader**.

Перед использованием собственных заголовком ответа, как и в  
случае заголовков запроса, следует ознакомиться с существующими  
ограничениями на их имена, накладываемые протоколом HTTP и webсервером.

**Lab 9**