# Доклад 2 вар. Лазарев С. О. 6307

# Современные технологии разработки Java интернет-приложений

# Сервлеты Servlets

**Servlet** - это java-программа, которая выполняется на серверной стороне Web-приложения. Точно так же, как апплеты динамически расширяют функциональные возможности Web-браузера, сервлеты динамически расширяют функциональные возможности Web-сервера.

Работу **servlet**'а можно описать следующим образом: при приходе запроса от клиента Web-сервер с помощью специального конфигурационного файла может определить, какой сервлет необходимо выполнить. После этого Web-сервер запускает JVM, которая в свою очередь выполняет сервлет. Servlet обрабатывает запрос и передает содержимое Web-серверу (возможно в виде HTML-страницы). Web-сервер отправляет клиенту ответ (сформированную сервлетом HTML-страницу).

WEB-сервер по сути является неким контейнером, который загружает **servlet**'ы, выполняет их, и, получив от них результат, отправляет его клиенту.

## Servlet в архитектуре Web-приложения

Из-за своей мощности и гибкости, **servlet**'ы могут играть значительную роль в архитектуре системы. Они могут выполнять прикладные задачи, предназначенные для промежуточного уровня, работать как прокси-сервер для клиента и даже улучшать функциональность промежуточного уровня, добавляя поддержку новых протоколов и других функций. Промежуточный уровень выполняет функции сервера приложений в так называемой трехуровневой системе клиент-сервер и расположен между "легковесным" клиентом, таким как Web-браузер, и источником данных.

## Servlet как прокси-сервер

Для поддержки апплетов сервлеты могут выполнять функции их прокси-серверов. Это может быть важно, поскольку система безопасности Java позволяет апплетам соединяться только с сервером, с которого они были загружены. Если апплет нуждается в соединении с сервером баз данных, расположенном на другой машине, **servlet** может создать это соединение для апплета.

## Временные и постоянные servlet'ы

Сервлеты могут запускаться и останавливаться для каждого клиентского запроса. Также они могут запускаться при старте Web-сервера и существовать до его остановки. Временные **servlet**'ы загружаются по требованию и предлагают хороший способ сохранения ресурсов сервера для редко используемых функций. Постоянные сервлеты загружаются при старте Web-сервера и существуют до его остановки. Сервлеты устанавливаются как постоянные расширения для сервера в том случае, если затраты по их запуску очень велики (например, установка соединения с базой данных), если они предлагают постоянную функциональность на стороне сервера (например, служба RMI), или в случаях, когда они должны отвечать на запросы клиента как можно быстрее. Не существует специального кода для назначения **servlet**'а постоянным или временным; это функция настройки Web-сервера.

## Жизненный цикл сервлета, javax.servlet.Servlet

Сервлеты выполняются на платформе Web-сервера как часть того же процесса, что и сам Web-сервер. Web-сервер отвечает за инициализацию, вызов и уничтожение каждого экземпляра сервлета. Web-сервер взаимодействует с сервлетом через простой интерфейс: **javax.servlet.Servlet**.

Интерфейс javax.servlet.Servlet включает три главных методов:

* init()
* service()
* destroy()

и два вспомогательных методов:

* getServletConfig()
* getServletInfo()

Сходство между интерфейсами **servlet**'а и апплета Java очевидны. Именно так и было спроектировано! Java сервлеты являются для Web-серверов тем же самым, чем являются апплеты для Web-браузеров. Апплет выполняется в Web-браузере, выполняя действия по его запросу через специальный интерфейс. Сервлет делает то же самое, работая на Web-сервере.

## Инициализация сервлета, метод init()

При первой загрузке сервлета вызывается метод init(). Это дает возможность сервлету выполнить любую работу по установке, например, открытие файлов или установку соединений с их серверами. Если сервлет установлен на сервере постоянно, он загружается при запуске сервера. В противном случае сервер активизирует сервлет при получении первого запроса от клиента на выполнение услуги, обеспечиваемой этим сервлетом.

Гарантируется, что метод **init()** закончится перед любым другим обращением к сервлету – таким как, например, вызов метода **service()**. Обратите внимание, что *init()* будет вызван только один раз; он не будет вызываться до тех пор, пока сервлет не будет выгружен и затем загружен сервером снова.

Метод *init()* принимает один аргумент – ссылку на объект **ServletConfig**, который содержит аргументы для инициализации сервлета. Этот объект имеет метод *getServletContext()*, возвращающий объект **ServletContext**, который содержит информацию об окружении сервлета.

## Ядро сервлета, метод service()

Метод **service()** является сердцем сервлета. Каждый запрос от клиента приводит к одному вызову метода **service()**. Этот метод читает запрос и формирует ответное сообщение при помощи своих двух аргументов ServletRequest и ServletResponse :

* Объект [ServletRequest](http://java-online.ru/servlet.xhtml#ServletRequest) содержит данные от клиента. Данные состоят из пар имя/значение и InputStream. Существует несколько методов, возвращающих информацию о параметрах клиента. InputStream может быть получен при помощи метода getInputStream(). Этот метод возвращает объект ServletInputStream, который можно использовать для получения дополнительных данных от клиента. Если необходима выполнить обработку символьных данных, а не двоичных, то можно получить объект BufferedReader при помощи метода getReader().
* Объект [ServletResponse](http://java-online.ru/servlet.xhtml#ServletResponse) содержит ответ сервлета клиенту. Во время подготовки ответа прежде всего вызывается метод setContentType() для установки типа MIME ответа. Затем могут быть использованы методы getOutputStream() или getWriter() для получения объектов и ServletOutputStream или PrintWriter соответственно для передачи данных обратно клиенту.

Таким образом, существуют два способа передачи информации от клиента к сервлету. Первый – через передачу значений в параметрах запроса. Значения параметров могут быть вставлены в URL. Второй способ передачи информации от клиента к сервлету осуществляется через InputStream (или Reader).

Работа метода **service()** по существу проста – он создает ответ на каждый клиентский запрос, переданный ему с сервера. Однако необходимо помнить, что могут существовать несколько параллельных запросов, обрабатываемых в одно и то же время. Если метод **service()** требует каких-либо внешних ресурсов, таких как файлы, базы данных, то необходимо гарантировать, чтобы доступ к ресурсам являлся потокозащищенным.

## Выгрузка сервлета, метод destroy()

Метод **destroy()** вызывается для освобождения всех ресурсов (например, открытые файлы и соединения с базой данных) перед выгрузкой сервлета. Этот метод может быть пустым, если нет необходимости выполнения каких-либо завершающих операций. Перед вызовом метода *destroy()* сервер ждет либо завершения всех обслуживающих операций, либо истечения определенного времени. Это означает, что метод *destroy()* может быть вызван во время выполнения какого-либо продолжительного метода *service()*.

Важно оформить метод *destroy()* таким образом, чтобы избежать закрытия необходимых ресурсов до тех пор, пока все вызовы *service()* не завершатся.

## Конфигурация сервлета, метод getServletConfig()

Метод **getServletConfig()** возвращает ссылку на объект, который реализует интерфейс *ServletConfig*. Данный объект предоставляет доступ к информации о конфигурации сервлета, т.е. доступ к параметрам инициализации сервлета и объекту контектса сервлета *ServletContext*, который дает доступ к сервлету и его окружению.

## Информация о сервлете, метод getServletInfo()

Метод **getServletInfo()** определяется программистом, создающим сервлет, для возврата строки, содержащую информацию о сервлете, например: автор и версия сервлета.

## Интерфейс ServletRequest

ServletRequest предоставляет клиентскую информацию о параметрах HTTP запроса сервлету, т.е. обеспечивает данные включая название параметра и значения, атрибуты, и входной поток. Эта информация передается в метод *service()*.

Следующий **servlet пример** показывает, как получить информацию из параметра **request** метода *service()*:

BufferedReader reader;

String param1;

String param2;

public void service(ServletRequest request, ServletResponse response)

{

reader = request.getReader();

param1 = request.getParameter("First");

param2 = request.getParameter("Second");

}

Дополнительная информация о запросе доступна сервлету через методы, основные из которых приведены в следующей таблице:

|  |  |
| --- | --- |
| getAttribute () | Возвращает значение указанного атрибута этого запроса. |
| getContentLength () | Размер запроса, если известен. |
| getContentType () | Возвращает тип MIME тела запроса. |
| getInputStream () | Возвращает InputStream для чтения двоичных данных из тела запроса. |
| GetParameterNames () | Возвращает массив строк с именами всех параметров. |
| getParameterValues () | Возвращает массив значений для указанного параметра. |
| getProtocol () | Возвращает протокол и версию для запроса как строку вида <protocol>/<major version>.<minor version>. |
| getReader () | Возвращает BufferedReader для получения текста из тела запроса. |
| getRealPath () | Возвращает реальный путь для указанного виртуального пути. |
| getRemoteAddr () | IP-адрес клиента, пославшего данный запрос. |
| getRemoteHost () | Имя хоста клиентской машины, пославшего данный запрос. |
| getScheme () | Возвращает схему, используемую в URL этого запроса (например, https, http, ftp, и т.д.). |
| getServerName () | Имя хоста сервера, принявшего данный запрос. |
| getServerPort () | Возвращает номер порта, используемого для приема этого запроса. |

## Интерфейс ServletResponse

Интерфейс **ServletResponse** - это инструмент для отправки данных клиенту. Все методы данного инструмента служат именно для решения этой задачи :

public java.lang.String getCharacterEncoding()

public void setLocale(java.util.Locale loc)

public java.util.Locale getLocale()

Первый метод возвращает MIME тип кодировки (к примеру - UTF8), в которой будет выдаваться информация. Вторые два метода тоже работают с charset. Они указывают на язык используемый в документе (например - русский).

public ServletOutputStream getOutputStream() throws java.io.IOException

Метод getOutputStream возвращает поток вывода данных для сервлета. Этот поток используется, к примеру, для вывода бинарных файлов. Текстовые данные можно выводить с помощью java.io.Writer:

public java.io.PrintWriter getWriter() throws java.io.IOException

Метод getWriter() автоматически конвертирует строки в тот charset, что указан в методе getCharacterEncoding() и getLocale().

public void setContentLength(int len)

Методом setContentLength устанавливается значение поля HTTP заголовка "Content-Length"

public void setContentType(String type)

Метод setContentType используется для отправки MIME типа содержимого документа. Поле HTTP заголовка "Content-Type".

Поток вывода данных является буфферизованным. Это означает, что порция данных будет выдана клиенту только после заполнения буфера.

public void setBufferSize(int size)

public int getBufferSize()

public void flushBuffer() throws java.io.IOException

public void resetBuffer()

Приведенные выше 4 метода позволяют, соответственно, установить размер буффера отправки, получить его размер, инициализировать отправку содержимое буффера клиенту, не дожидаясь его заполнения, а так же очистить этот буффер от данных.

public boolean isCommitted()

Методом isCommitted можно получить флаг, начата ли уже отправка данных клиенту. Флаг будет положительным, если HTTP заголовок ответа был уже отправлен.

public void reset()

Если HTTP заголовок ещё не отправлен, то метод reset "сбрасывает" HTTP заголовок к значениям "по умолчанию".

# Java Server Pages JSP

**Java Server Pages (JSP)** - это одна из технологий J2EE, которая представляет собой расширение технологии сервлетов для упрощения работы с Web-содержимым. Страницы **JSP** позволяет легко разделить Web-содержимое на 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 наследует и расширяет спецификацию сервлетов. Как и сервлеты, компоненты JSP относятся к компонентам Web и располагаются в Web-контейнере. Страницы JSP не зависят от конкретной реализации Web-контейнера, что обеспечивает возможность их повторного использования.

В дополнение к классам и интерфейсам для программирования сервлетов (пакеты **javax.servlet** и **javax.servlet.http**), в пакетах **javax.servlet.jsp** и **javax.servlet.jsp.target** содержатся классы и интерфейсы, относящиеся к программированию Java Server Pages.

# Обзор технологии Java Server Pages

Технология Java Server Pages содержит четыре ключевых компонента:

1. **Директивы** *(directive)* представляют собой сообщения для контейнера JSP, дающим возможность определить параметры страницы, подключения других ресурсов, использовать собственные нестандартные библиотеки тегов.
2. [Действия actions](http://java-online.ru/jsp-actions.xhtml) инкапсулируют функциональные возможности в предопределенных тегах, которые можно встраивать в JSP-страницу. JSP actions 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.
3. [Скриптлеты scriptlets](http://java-online.ru/jsp-syntax.xhtml#scriptlets) позволяют вставлять код Java в страницы JSP, который взаимодействует с объектами страницы при обработке запроса.
4. [Библиотеки тегов](http://java-online.ru/jsp-taglib.xhtml) *(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-страницы являются взаимозамеяемыми. Подобно сервлетам, JSP-страницы обычно выполняются на стороне Web-сервера, который называют контейнером JSP.

Когда Web-сервер, поддерживающий технологию JSP, принимает первый запрос на JSP-страницу, контейнер JSP транслирует эту JSP-страницу 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 ответ JSP-страницы, в метод \_jspService. Если сервлет компилируется без ошибок, контейнер JSP вызывает метод \_jspService для обработки запроса.

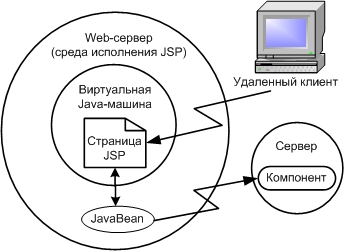
JSP-страница может обработать запрос непосредственно, либо вызвать другие компоненты Web-приложения, чтобы содействовать обработке запроса. Любые ошибки, которые возникают в процессе обработки, вызывают исключительную ситуацию в Web-сервере на этапе запроса.

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 также, как и начальные параметры сервлета.

# Архитектура JSP

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-сервере в среде виртуальной Java-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 сеансе, а через них - всем пользователям, запрашивающим эти страницы JSP.

Использование межплатформенных компонентов JavaBeans и библиотек тегов значительно расширяет возможности JSP. Программный Java-код в странице JSP, в идеале, должен использоваться только для управления представлением информации.

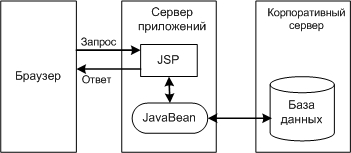
### Основные модели архитектуры JSP

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 1

Первая модель **JSP Model 1** 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.



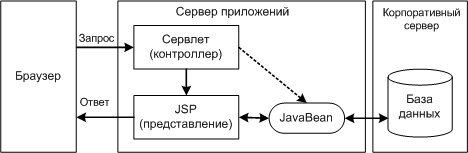
В сценарии *JSP Model 1* предполагается следующая последовательность действий:

1. Запрос пользователя посылается через Web-браузер странице JSP.
2. Страница JSP компилируется в сервлет (*при первом обращении*).
3. Скомпилированный сервлет обращается к некоторому компоненту JavaBean, запрашивая у него информацию.
4. Компонент JavaBean, в свою очередь, осуществляет доступ к информационным ресурсам (непосредственно или через компонент Enterprise JavaBeans).
5. Полученная информация отображается в свойствах компонента JavaBeans, доступных странице JSP.
6. 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 технологичным из-за большого объема встроенных в страницу программных фрагментов. Для сложных корпоративных приложений рекомендуется применение второй модели архитектуры JSP.

### JSP Model 2

Вторая модель **JSP Model 2** реализует гибридный подход к обслуживанию динамического содержания Web-страницы, 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:

1. Запрос пользователя посылается через Web-браузер сервлету.
2. Сервлет обрабатывает запрос, создает и инициализирует объект JavaBean или другие объекты, используемые страницей JSP, и запрашивает динамическое содержание у компонента JavaBean.
3. Компонент JavaBean осуществляет доступ к информации непосредственно или через компонент Enterprise JavaBeans.
4. Сервлет, направляющий запрос, вызывает сервлет, скомпилированный из страницы JSP.
5. Сервлет, скомпилированный из страницы JSP, встраивает динамическое содержание в статический контекст HTML-страницы и 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-код сервлета, который передает преобразованное содержимое браузеру через выходной поток Web-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 на Web-сервере или сервере приложений.

Во второй фазе осуществляется обработка запросов и подготовка ответов.

# Жизненный цикл JSP

Страница JSP обслуживает запросы, как сервлет. Следовательно, жизненный цикл и многие возможности страниц JSP (в частности, динамические аспекты) определяются технологией **Servlet** и многие обсуждения в этой главе ссылаются на функции, описанные на странице [cервлетов](http://java-online.ru/servlet.xhtml).

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 разработке Web-приложения одним из преимуществ страниц JSP перед сервлетами является то, что процесс построения (*компиляции страницы JSP в сервлет*) выполняется автоматически.

### Трансляция и компиляция страницы JSP

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-контейнер переводит 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 на странице [сервлета](http://java-online.ru/servlet.xhtml#lifecycle):

1. Если экземпляр сервлета страницы JSP не существует, контейнер:
   * загружает класс сервлета страницы JSP;
   * создает экземпляр класса сервлета;
   * инициализирует экземпляр сервлета вызовом метода jspInit.
2. Вызывает метод \_jspService, передавая ему объекты запроса и отклика.

Если контейнеру нужно удалить сервлет страницы JSP, он вызывает метод jspDestroy.

# Portlet

Портлеты - представляют собой специальный тип сервлета, и они используют JSP-страницы для отображения пользовательского интерфейса.

Portlet API расширяет и является подклассом Servlet API, и это означает, что портлеты могут делать то же, что и сервлеты с некоторыми изменениями при исполнении и дополнительными возможностями.

Наиболее существенное изменение исполнения в том, как обслуживаются запросы: сервлеты обрабатывают запросы "doGet" и "doPost", которые преобразуются в http GET и POST запросы, **в то время как портлеты обрабатывают запросы "doView" и "doEdit", которые приходят не напрямую от Web-браузера, а от портального сервера.**

# Возможности портлетов:

* встроенная поддержка автоматического использования различных JSP- страниц для различных пользовательских устройств, таких как настольные компьютеры, Palm-компьютеры с ограниченными Web-браузерами, PDA и мобильные телефоны;
* назначать права пользователям групп на использование портлетов.

В случае отсутствия оных они даже не будут

видеть портлеты;

* создание сохраняемых между сессиями пользовательских настроек;
* публикация в виде Web-сервиса;
* разделение сложных приложений на задачи, когда группа тесно связанных задач равняется одному портлету;
* добавление новых функций к приложению;
* хорошая совместимость с брандмауэрами (firewalls), так как портлеты используют стандартные Web-протоколы для получения и отображения информации;
* одноразовая установка и настройка портлета для пользователей.

# Сходства и различия сервлетов и портлетов

Сходства между сервлетами и портлетами:

* относятся к J2EE Web-компонентам;
* управляются контейнерами;
* генерируют динамическое Web-содержимое при помощи запросов и ответов

Различия между сервлетами и портлетами:

* портлеты генерируют часть документа, в то время как сервлеты генерируют его полностью;
* за счёт того, что операции кодирования URL выполняются на стороне сервера, пользователь не может обратиться к нему напрямую, зная имя портлета: портлет - часть страницы, поэтому знания одного URL мало;
* портлеты имеют несколько иную схему управления запросами, которые делятся на запросы выполнения действий и запросы генерирования содержимого;
* портлеты придерживаются стандартного набора состояний, которые определяют их контекст работы и правила генерации содержимого.

# Жизненный цикл

* Как и у сервлетов, жизненный цикл портлетов управляется контейнером, и у него есть метод **init()**, который используется для инициализации всех данных, необходимых для корректной работы портлета (создание ресурсов, конфигурирование и т.д.).
* Метод **init()** в качестве параметра принимает объект, который реализует интерфейс **PortletConfig**, и этот объект предоставляет необходимые для инициализации параметры.
* Он может быть использован для получения ссылки на объект, реализующий интерфейс **PortletContext**.
* При создании портлета доступа к окружающему коду, например к контейнеру, нет, поэтому код внутри портлета не может оценить, насколько портлет доступен извне.
* Метод **destroy()** предоставляет возможность для произведения очистки ресурсов, которые были востребованы и инициализированы методом **init()**.
* Этот метод аналогичен методу **destroy()** в сервлетах и вызывается один раз: когда контейнер выгружает портлет.

# Состояния

Состояния портлетов – это часть портальной модели отображения.

Состояния позволяют портлету отображать различные «виды» в зависимости от ситуации.

Есть четыре состояния портлета:

* **View** – основное состояние портлета;
* **Help** – если портлет обеспечивает состояние помощи;
* **Edit** – редактирование параметров портлета с сохранением результатов для этого конкретного пользователя;
* **Configure** – конфигурирование портлета с охранением результатов для всех пользователей, права к состояниям никак не относятся.