**HTTP** (*HyperText Transfer Protocol* — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов в формате HTML, в настоящий момент используется для передачи произвольных данных).

Именно этот протокол мы используем, когда просматриваем странички в браузере.

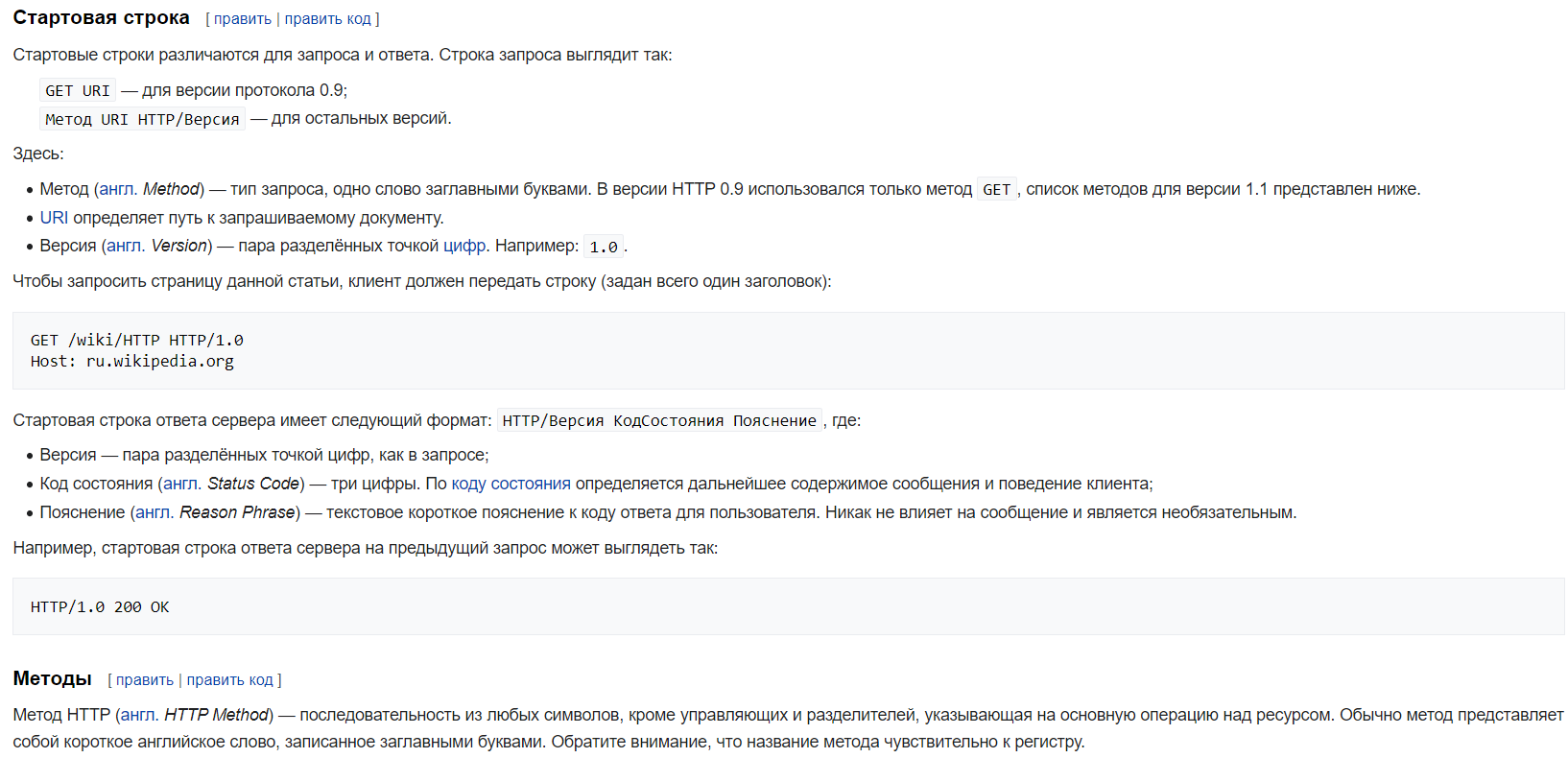
Но только два из них пользуются особой популярностью, а именно **GET** и **POST**.  Метод GET вызвать легко, достаточно ввести в браузере ссылку и перейти по ней, а вот POST не так просто выполнить, один из простых способов используя **form** тег указать ему **action**.

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

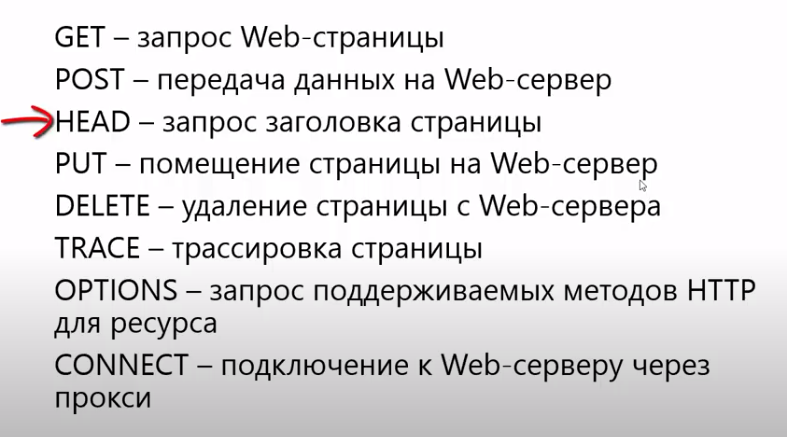
1. Стартовая строка ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Starting line*) — определяет тип сообщения;
2. Заголовки ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Headers*) — характеризуют тело сообщения, параметры передачи и прочие сведения;
3. Тело сообщения ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Message Body*) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

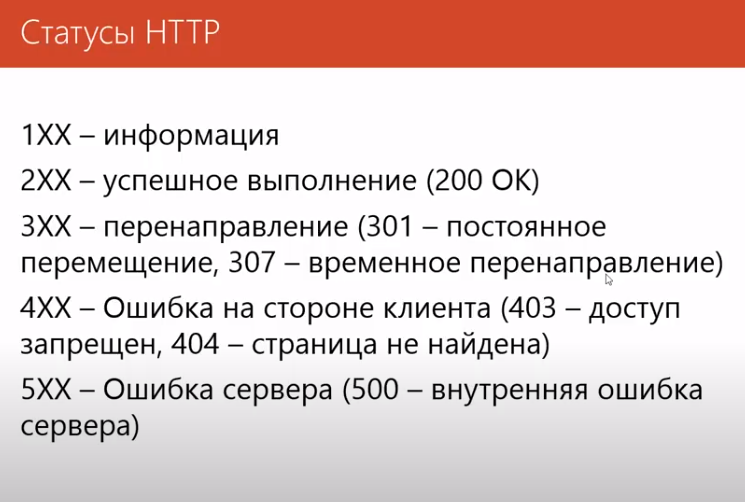
Тело сообщения может отсутствовать, но стартовая строка и заголовок являются обязательными элементами. Исключением является версия 0.9 протокола, у которой сообщение запроса содержит только стартовую строку, а сообщения ответа — только тело сообщения.

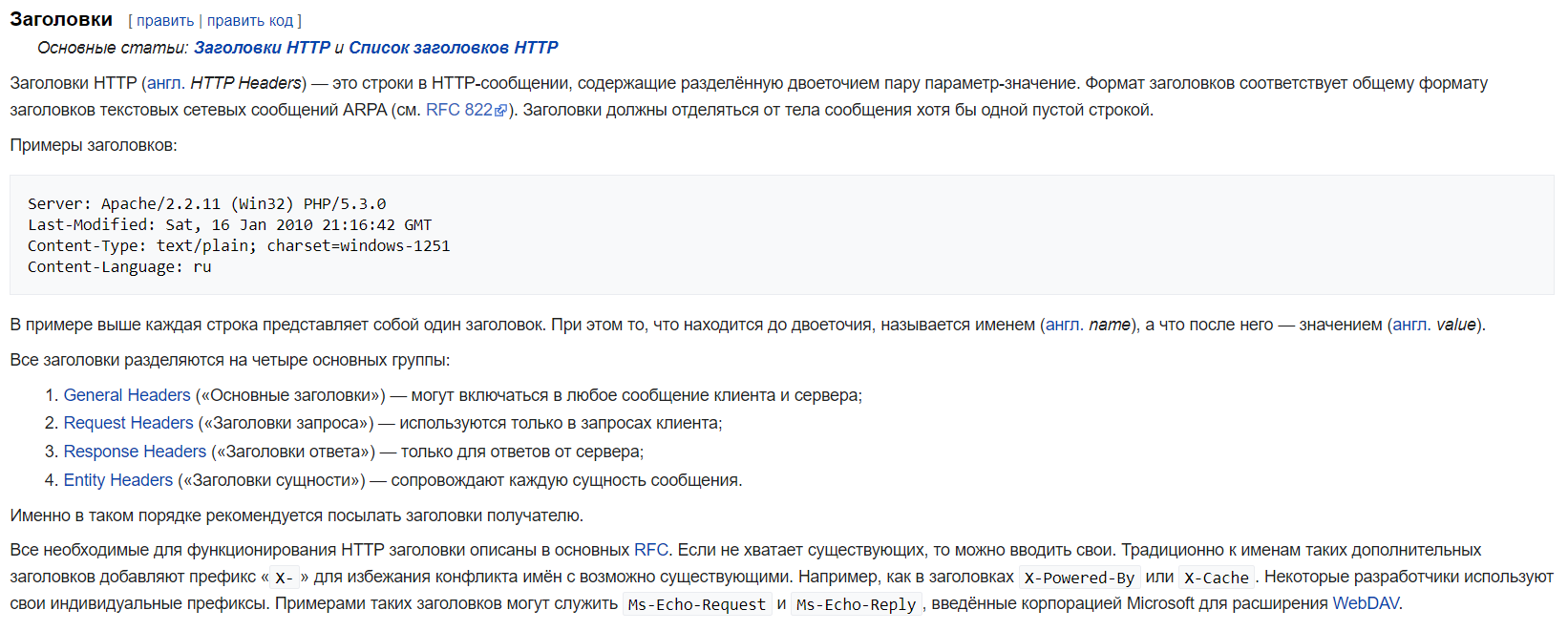
Для версии протокола 1.1 сообщение запроса обязательно должно содержать заголовок *Host*.

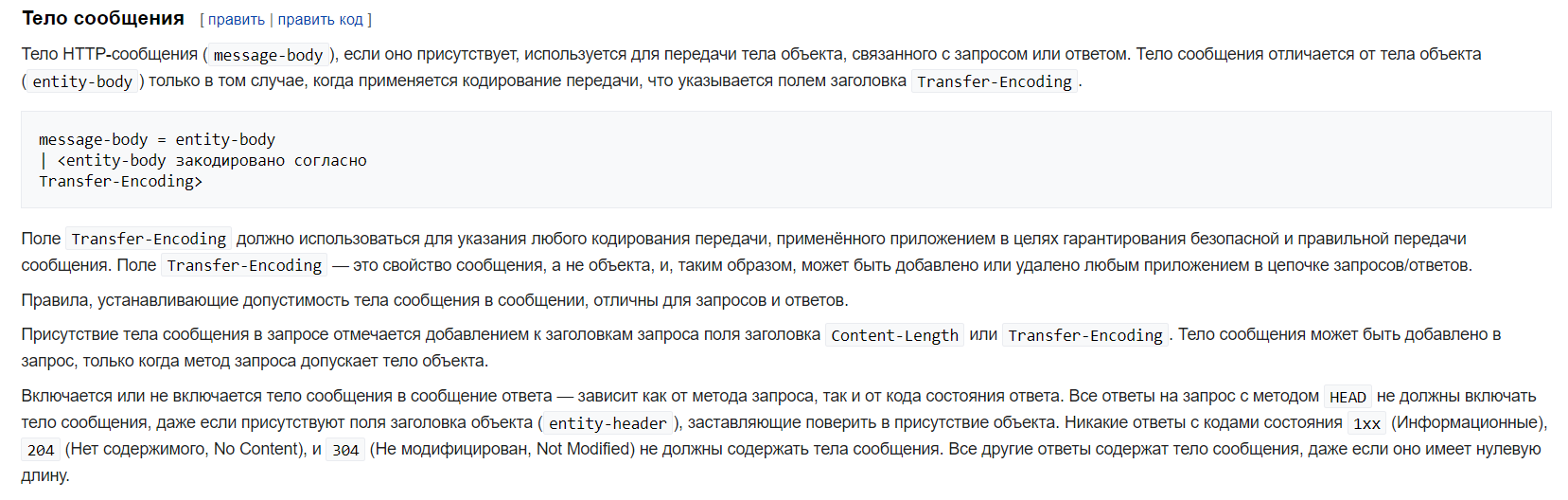


Известно, что HTTP протокол имеет 7 методов передачи данных:









1. **Что такое сервлет?**

**Сервлет** — это класс, который умеет получать запросы от клиента и возвращать ему ответы. Да, сервлеты в Java — именно те элементы, с помощью которых строится клиент-серверная архитектура.

**Servlet** - серверный web-компонент JavaEE приложения, предназначенный для обработки запроса, формирования и отправки ответа.

1. **Для чего нужен web-сервер?**

Веб-сервер — это сервер, который принимает HTTP-запросы от клиентов и выдающий им HTTP ответы (как правило, вместе с HTML страницей, изображением, файлом или другими данными). Запрашиваемые ресурсы обознаются URL-адресами.

Его главная задача – хранение информации. Страницы, файлы, изображения, текстовый контент.  
Задачи:

* Получать запросы.
* Запускать программы на специальных языках программирования (серверных).
* Формировать веб страницы.
* Отправлять их назад пользователю.
* Защищать информацию.
* Проводить идентификацию посетителей.
* Вести журнал обращений.
* Обслуживать запросы других типов: mailto, [FTP](https://semantica.in/blog/chto-takoe-ftp-server.html) и т.д.

1. **Расскажите о принципах построения и работы сервлета? Объясните общее взаимодействие клиентской и серверной сторон.**

Сервлет - это класс, который расширяет функциональность класса **HttpServlet** и запускается внутри контейнера сервлетов.

Сервлет размещается на сервере, однако чтобы сервер мог использовать сервлет для обработки запросов, сервер должен поддерживать движок или контейнер сервлетов (servlet container/engine). Например, Apache Tomcat по сути является контейнером сервлетов, поэтому он может использовать сервлеты для обслуживания запросов.

В основе работы приложения лежит так называемая модель взаимодействия клиент-сервер, которая позволяет разделять функционал и вычислительную нагрузку между клиентскими приложениями (заказчиками услуг) и серверными приложениями (поставщиками услуг).

Request- запрос; Response- отклик;

1. **Чем отличаются методы doGet и doPost друг от друга? В чем разница между методами GET и POST?**

* GET метод является неизменяемым, тогда как POST — изменяемый.
* С помощью метода GET можно посылать ограниченное кол-во данных, которые будут пересланы в заголовке URL. В случае POST метода мы можем пересылать большие объемы данных, т.к. они будут находится в теле метода.
* Данные GET метода передаются в открытом виде, что может использоваться в зловредных целях. POST данные передаются в теле запроса и скрыты от пользователя.
* GET метод является HTTP методом по умолчанию, а POST метод необходимо указывать явно, чтобы отправить запрос.
* GET метод используется гиперссылками на странице.

1. **Назовите методы жизненного цикла сервлета. В каком порядке они вызываются?**

Жизненный цикл любого сервлета состоит из 4 этапов:

Инициализация, которая происходит при вызове метода init()

Обработка клиентского запроса после вызова метода service()

Уничтожение сервлета с помощью вызова метода destroy()

Полное уничтожение сервлета сборщиком мусора Java (Garbage Collector)

1. **Как определить name и URL сервлета?**

Для получения актуального пути сервлета на сервере можно использовать эту конструкцию: getServletContext().getRealPath(request.getServletPath())

1. Как получить значение параметра формы в сервлете?

Сервлеты автоматически обрабатывают данные, используя следующие методы в зависимости от ситуации:

getParameter () — Вы вызываете метод request.getParameter (), чтобы получить значение параметра формы

getParameterValues ​​() — Вызовите этот метод, если параметр появляется более одного раза и возвращает несколько значений, например, флажок

getParameterNames () — вызовите этот метод, если вы хотите получить полный список всех параметров в текущем запросе

1. **Как сервлет возвращает результат своей работы на сторону клиента?**

**HttpServletResponse** resp – это ответ который даст сервлет.

Просто вызов URL-адреса сервлета приведет вас к стороне клиента (браузеру). Вам не нужно делать ничего конкретного.

**9. Как происходит развертывание сервлета?**

Контейнер сервлетов обычно загружает сервлет при первом запросе клиента, но иногда необходимо загрузить сервлет прямо на старте приложения (например если сервлет объемный и будет долго грузиться). Для этого необходимо использовать элемент load-on-startup в дескрипторе (или аннотацию loadOnStartup), который укажет необходимость загрузки сервлета при запуске.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <servlet>      <servlet-name>foo</servlet-name>      <servlet-class>com.foo.servlets.Foo</servlet-class>      <load-on-startup>5</load-on-startup>  </servlet> |

Значение должно быть int. Если значение отрицательное, то сервлет будет загружен при запросе клиента, а если 0 и далее, то загрузится на старте приложения. Чем меньше число, тем раньше в очереди на загрузку будет сервлет.

1. **Для чего нужен интерфейс ServletContext?**

Интерфейс javax.servlet.ServletContext предоставляет доступ к параметрам веб приложения сервлету. Объект ServletContext является уникальным и доступен всем сервлетам веб приложения. Мы можем использовать объект ServletContext, когда нам необходимо предоставить доступ одному или нескольким сервлетам к инициализированным параметрам веб приложения. Для этого используется элемент <context-param> в web.xml. Объект ServletContext можно получить с помощью метода getServletContext() у интерфейса ServletConfig.

Контейнеры сервлетов так же могут предоставлять context объекты, уникальные для группы сервлетов. Каждая из групп будет связана со своим набором URL путей хоста.

ServletContext был расширен в спецификации Servlet 3 и предоставляет программное добавление слушателей и фильтров в приложение. Так же у этого интерфейса имеются множество полезных методов вроде getMimeType(), getResourceAsStream() и т.д..

**Интерфейс** javax.**servlet**.**ServletContext** определяет ряд методов, которые сервлет использует, чтобы связаться с его контейнером сервлета, например, получить тип MIME файла, диспетчеризировать запросы, или записать в файл журнала.

1. **Что такое атрибут контекста?**

серверный объект, предназначенный для хранения информации об одном web-приложении.

► формируется сразу при загрузке web-сервера, основные данные (параметры приложения) копируются из конфигурационного файла приложения, ► общий для всех сессий приложения, ► формат ключ/значение.

1. **Какую информацию можно получить из контекста сервлета?**

Следующие методы позволяют получить из контекста сервлета базовую информацию:

String getMimeType(String filename) – определение MIME-типа файла или документа. По умолчанию MIME-типом для сервлетов является text/plain, но используется обычно text/html;

String getRealPath(String filename) – определение истинного маршрута файла относительно каталога, в котором сервер хранит документы;

String getServerInfo() – предоставляет информацию о самом сервере.

Ряд методов предназначен для управления атрибутами, с помощью которых передается информация между различными компонентами приложения (JSP, сервлетами):

**Object getAttribute(String name)** – получает значение атрибута по имени;

**Enumeration getAttributeNames()** – получает список имен атрибутов;

**void setAttribute(String name, Object object)** – добавляет атрибут и его значение в контекст;

**void removeAttribute(String name)** – удаляет атрибут из контекста;

**ServletContext getContext(String uripath)** – позволяет получить доступ к контексту других ресурсов данного контейнера сервлетов;

**String getServletContextName()**– возвращает имя сервлета, которому принадлежит данный объект интерфейса **ServletContext**.

Используя объект **ServletContext**, можно регистрировать события сервлета, сессии и запроса.

1. **Опишите интерфейс ServletConfig?**

Экземпляр ServletConfig - содержит параметры инициализации сервлета

ServletConfig sc = this.getServletConfig();

Методы интерфейса:

Enumeration getInitParameterNames() //спискок имен параметров инициализации сервлета;

String getInitParameter(String name) // определение значения параметра по его имени;

String getServletName() // определение имени сервлета

String getServletContext() // получение экземпляра контекста

1. **Какую информацию содержит интерфейс HttpServletRequest?**

Объект запроса – содержит информацию от клиента веб-приложения:

► информация о сервлете

► доступ к сессии и параметрам запроса

► управление атрибутами и кодировками

► детали протокола HTTP

1. **Как можно использовать интерфейс HttpServletResponse?**

Расширяет [ServletResponse](https://translated.turbopages.org/proxy_u/en-ru.ru.7980a2e8-62a9dea0-d9b717f9-74722d776562/https/docs.oracle.com/cd/E17802_01/products/products/servlet/2.2/javadoc/javax/servlet/ServletResponse.html) интерфейс для предоставления специфичных для HTTP функций при отправке ответа. Например, он имеет методы для доступа к HTTP-заголовкам и файлам cookie.

void setContentType(String type) //установка MIME-типа генерируемых документов;   
void addCookie(Cookie cookies) // Метод используется для добавления Cookie в заголовок ответа клинту.  
void sendError(int sc, String msg) // сообщение о возникших ошибках   
void sendRedirect(String location) // переход по указанному адресу, //без передачи экземпляра request.

ServletOutputStream getOutputStream() // извлечение ссылки на поток // для передачи бинарной информации;

PrintWriter getWriter() // извлечение ссылки на поток // для передачи символьной информации;

►Позволяет буферизацию

►Устанавливают + добавляют заголовки

1. **Как выполняется переадресация запросов и как перенаправление откликов? В чем принципиальная разница?**

**Процесс пересылки:**

Сначала клиент отправляет запрос на сервер, который сопоставляет сервлет и указывает выполнение. Когда сервлет выполняется, он вызывает метод getRequestDispacther () и перенаправляет запрос в указанный тестовый сервлет. Весь процесс завершается на стороне сервера и выполняется в одном запросе, поэтому сервлет и тестовый сервлет совместно используют один и тот же запрос. Запрос, все что положено в ForwardServlet, можно вынести в testservlet. Поэтому testservlet может получить результат getAttribute (). После выполнения getAttribute () результат возвращается клиенту. Весь процесс представляет собой запрос и ответ.

**Процесс перенаправления:**

Клиент отправляет запрос на сервер, и сервер сопоставляет сервлет, который совпадает с переадресацией запроса. После обработки сервлета вызывается метод sendRedirect (), который является методом ответа. Итак, после обработки этого сервлета, посмотрите метод response.sendRedirect () и немедленно верните ответ клиенту. Строка ответа сообщает клиенту, что вы должны отправить еще один запрос для доступа к тестовому сервлету, и тогда клиент получит это Сразу после запроса отправляется новый запрос для запроса тестового сервлета. Два запроса не мешают друг другу и не зависят друг от друга. Ничего из setAttribute () в предыдущем запросе не может быть получено в следующем запросе. Поэтому в sendRedirect () есть два запроса и два ответа.

Пересылка - это переход со стороны сервера, то есть клиент отправляет запрос на сервер, а сервер напрямую передает информацию о параметрах, связанных с запросом, в другой jsp или сервлет сервера для обработки. SendRedirect () - это переход клиента, сервер вернет заголовок ответа и новый URL-адрес клиента, исходная информация о параметрах не существует, если сервер не имеет специальной обработки, браузер получит доступ к сервлету или новый URL-адрес, указанный JSP, это может быть не веб-сервис на исходном сервере.

**Разница между перенаправлением и пересылкой**

1. Пересылка выполняется на стороне сервера, а перенаправление происходит на стороне клиента;

2. Переадресация - это один и тот же запрос, а перенаправление - два запроса;

3. Адресная строка пересылки не изменилась, а адресная строка переадресации изменилась;

4. Переадресация может осуществляться только в рамках одного веб-проекта, перенаправление может осуществляться в пределах одного веб-проекта или может быть перенаправлено в любое другое место;

1. **Как можно сохранить данные приложения между запросами?**

Куки + сессии. Все современные фреймворки умеют работать с различными хранилищами данных сессии: бд, файлы, мемкеш (имхо для сессий самое то). НО не надо сохранять всю инфу пользователя в сессию, она для этого не предназначена. Можно хранить в сессии ID пользователя и остальные ключи, что бы в нужный момент легко выдернуть нужные данные из бд. Если данных много и они не менятся каждый раз, при запросе от пользователя, то можно кешировать данные, например в мемкеш, при этом данные пользователей которые давно не посещали сервис должны вытесняться данными пользователей, посетившими сервис только что.

Сохранение данных приложения между запросами (scope)   
1) сохранение объекта в запросе HttpRequest.setAttribute() HttpRequest.getAttribute()   
2) присоединение объекта к объекту сеанса(сессии в HttpSession) – не будут видимы для других пользователей   
3) присоединение объекта к контексту сервлета (ServletContext) - будут видимы все пользователям

1. **Как получить объект HttpSession и что она содержит?**

Для получения объекта сессии в сервлете у объекта HttpServletRequest определен метод getSession(). Он возвращает объект HttpSession. ( HttpSession session = request.getSession(); )

Сессия позволяет сохранять некоторую информацию на время сеанса. Когда клиент обращается к сервлету или странице JSP, то движок сервлетов проверяет, определен ли в запросе параметр ID сессии. Если такой параметр неопределен (например, клиент первый раз обращается к приложению), тогда движок сервлетов создает уникальное значение ID и связанный с ним объект сессии. Объект сессии сохраняется на сервере, а ID оправляется в ответе клиенту и по умолчанию сохраняется на клиенте в куках. Затем когда приходит новый запрос от того же клиента, то движок сервлетов опять же может получить ID и сопоставить его с объектом сессии на веб-сервере.

Хотя по умолчанию ID сессии хранится в куках, но возможна ситуация, когда куки отключены на клиенте. Для решения этой проблемы есть ряд техник, в частности, добавление ID в адрес.

Для управления сессией объект HttpSession предоставляет ряд методов:

* setAttribute(String name, Object o): сохраняет в сессии под ключом name
* getAttribute(String name): возвращает из сессии объект с ключом name. Если ключа name в сессии неопределено, то возвращается null
* removeAttribute(String name): удаляет из сессии объект с ключом name

1. **Что такое файлы Cookie? Как его создать и поместить информацию информацию?**

**Cookies** («куки», в переводе с англ. «печенье») — это небольшие текстовые документы, которые с помощью браузера сохраняет на компьютере пользователя веб-сервер (сайт). В эти **файлы** можно записать практически любую **информацию** о посетителе сайта: во сколько и с какого устройства человек заходил на страницу, какими товарами интересовался и так далее.

1. **С какой целью используют интерфейс Filter?**

Реализация интерфейса Filter позволяет создать объект, который перехватывает запрос, может трансформировать заголовок и содержимое запроса клиента. Фильтры не создают запрос или ответ, а только модифицируют их. Фильтр выполняет предварительную обработку запроса, прежде чем тот попадает в сервлет, с последующей (если необходимо) обработкой ответа, исходящего из сервлета. Фильтр может взаимодействовать с разными типами ресурсов, в частности, и с сервлетами, и с JSP-страницами.

**21. Перечислите методы жизненного цикла фильтра?**

Метод init вызывается прежде, чем фильтр начинает работать, и определяет конфигурационные параметры фильтра. Метод **doFilter** выполняет непосредственно работу фильтра. Таким образом, сервер вызывает init один раз, чтобы запустить фильтр в работу, а затем вызывает **doFilter** столько раз, сколько запросов будет сделано непосредственно к данному фильтру. После того, как фильтр заканчивает свою работу, вызывается метод destroy.