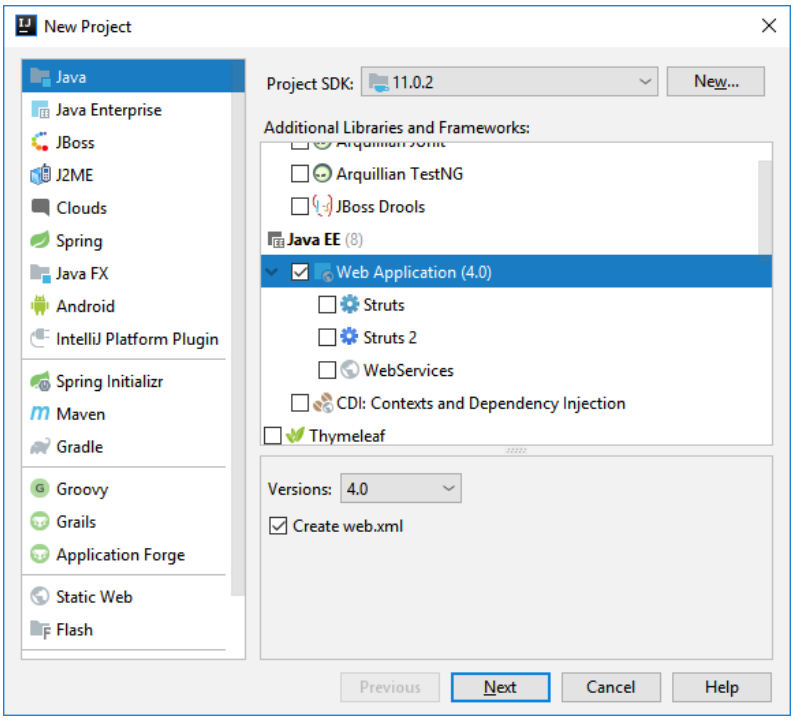
Дополнительный материал к Java10 Servlet

1. Базовые технологии сервлетов

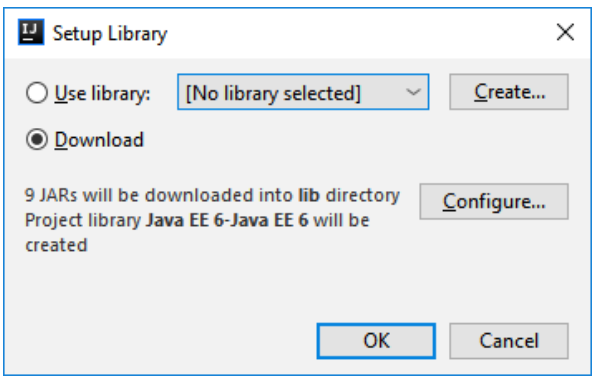
Создайте новое приложение Web application



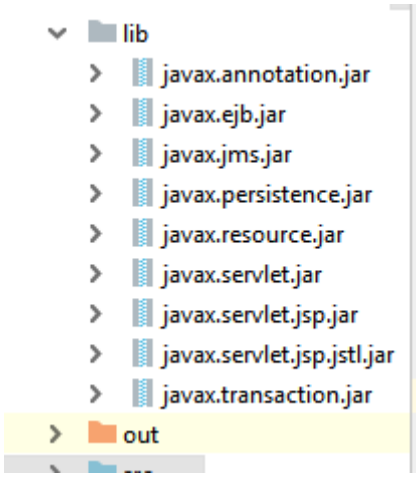
Создайте в папке **src** вашего проекта пакеты

Затем в ней создайте новый сервлет **New → Servlet.** Назовите его LoginServlet (\src\by\patsei\LoginServlet.java)

Чтобы создать сервет, нам нужен суперкласс HttpServlet, который определен в Java EE Web API. Добавьте зависимость. И для поддержки сервлетов необходим jar файл.
подключив или скачав пакет javax.servlet



Убедитесь, что jar подключен. Просмотрите папку lib



Как видите у вас уже есть два авто сгенерированных метода:

```
public class LoginServlet extends javax.servlet.http.HttpServlet {
    protected void doPost(javax.servlet.http.HttpServletRequest request,
                          javax.servlet.http.HttpServletResponse response)
                            throws javax.servlet.ServletException, IOException {

    }

    protected void doGet(javax.servlet.http.HttpServletRequest request,
                         javax.servlet.http.HttpServletResponse response)
                        throws javax.servlet.ServletException, IOException {

    }
}
```

Теория

Сервлеты. Жизненный цикл

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* состоит из трех главных методов:

- init()
- service()
- destroy()

и двух вспомогательных методов:

- getServletConfig()
- getServletInfo()

Сходство между интерфейсами сервлета и апплета Java очевидны. Именно так и было спроектировано! Сервлеты являются для Web-серверов тем же самым, чем являются апплеты для Web-броузеров. Апплет выполняется в Web-броузере, выполняя действия по его запросу через специальный интерфейс. Сервлет делает то же самое, работая на Web-сервере.

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.

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* 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* 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.

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.

Метод getServletConfig()

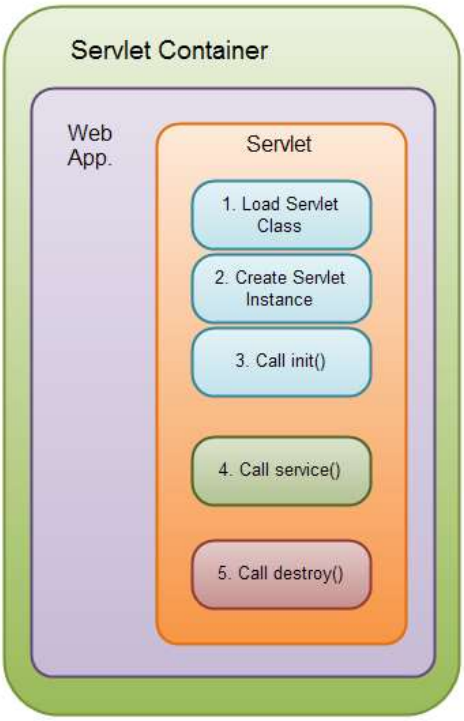
Метод *getServletConfig()* возвращает ссылку на объект, который реализует интерфейс *ServletConfig*. Данный объект предоставляет доступ к информации о конфигурации

сервлета, т.е. доступ к параметрам инициализации сервлета и объекту контектса сервлета *ServletContext*, который дает доступ к сервлету и его окружению.

Метод getServletInfo()

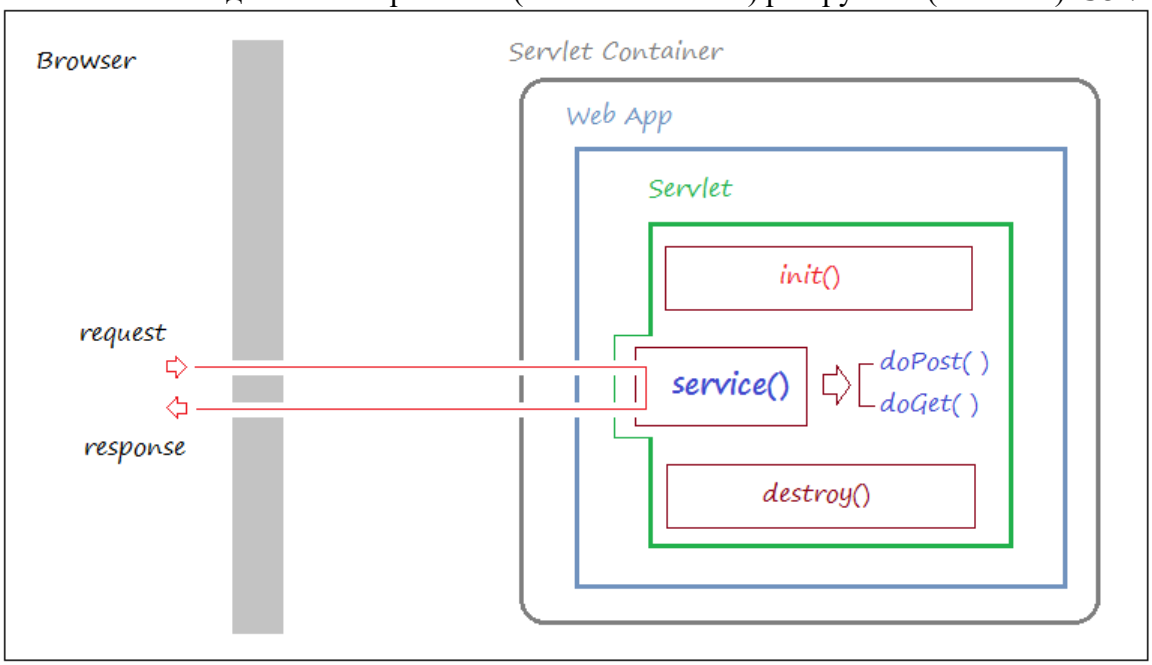
Метод *getServletInfo()* определяется программистом, создающим сервлет, для возврата строки, содержащую информацию о сервлете, например: автор и версия сервлета.

Итак, жизненный цикл **Servlet** с момента создания, обработки от пользователя, до момента разрушения можно представить так:



Шаги 1, 2 & 3 выполняются один единственный раз, когда Servlet загружен в первый раз. По умолчанию Servlet не загружаются (load) пока он не получит первый запрос от пользователя. Вы можете заставить Servlet Container (Servlet Контейнер) загрузить Servlet при запуске.

Шаг 4 выполняется много раз, каждый раз при запросе от пользователя к Servlet. Шаг 5 выполняется когда контейнер Servlet (Servlet Container) разгружает (unloaded) Servlet.

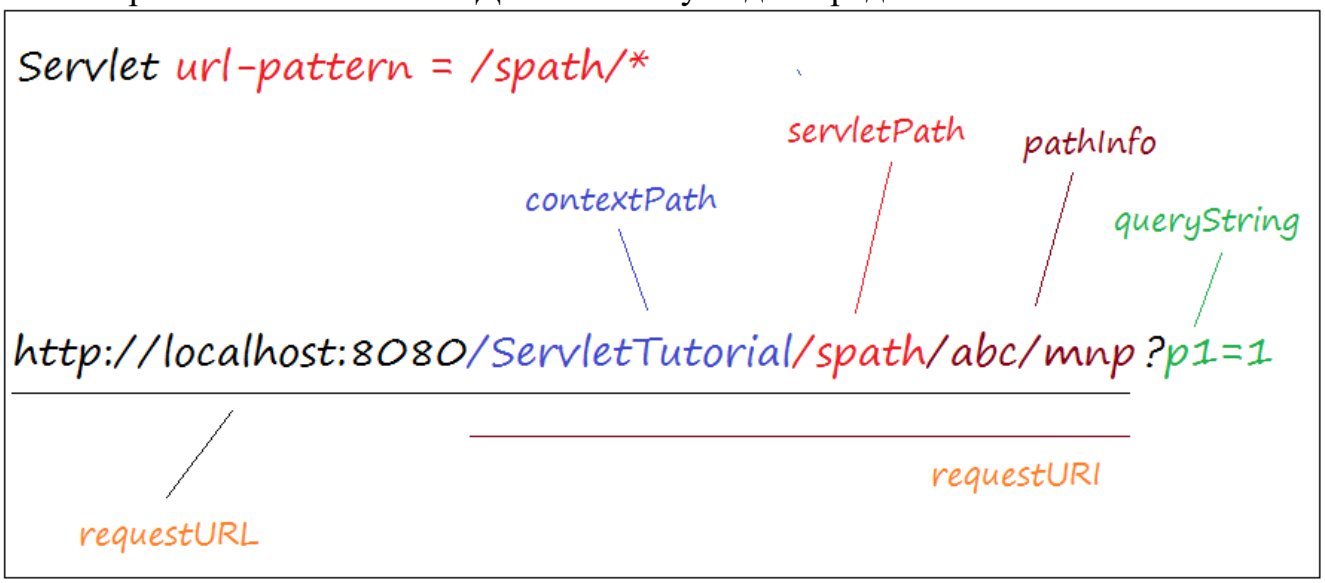


При отправлении запроса (request) пользователя к Servlet, Servlet вызывает метод service() чтобы обработать запрос пользователя, service() вызывает один из двух методов doGet() или doPost(). В вашем Servlet, вам нужно переопределить и обработать эти методы.

Так, когда пользователь отправляет запрос Servlet, Servlet создается в момент получения первого запроса, одновременно вызывается метод init() в servlet чтобы инициализировать его, init() вызывается один единственный раз. Метод destroy()используется для разрушения servlet, вызывается единственный раз когда вы отменяете развертывание (undeloy) веб приложения или отключаете (shutdown) Web Server (Веб Сервер).

Поскольку для обработки всех запросов создается один экземпляр сервлета, и все обращения к нему идут в отдельных потоках, то не рекомендуется в классе сервлета объявлять и использовать глобальные переменные, так как они не будут потокобезопасными.

Когда пользователь вводит ссылку в браузере, то она будет отправлена в **WebContainer. WebContainer** должен решить, какой **Servlet** будет обслуживать этот запрос от пользователя. Для этого ему надо определить urlPatterns



Servlet с **url-pattern = /** -
Это servlet по умолчанию, servlet будет использоваться для обработки запроса (request), который имеет ссылку несовпадающую ни с каким **url-pattern** другого **Servlet** объявленный в вашем приложении.

Начиная от версии 3.0 можно конфигурировать **Servlet** используя Annotation.

Напишите следующий код

```
@WebServlet(urlPatterns = "/login")
public class LoginServlet extends javax.servlet.http.HttpServlet {
    protected void doPost(javax.servlet.http.HttpServletRequest request,
                          javax.servlet.http.HttpServletResponse response)
                            throws javax.servlet.ServletException, IOException {
    }

    protected void doGet(javax.servlet.http.HttpServletRequest request,
```

```
                        javax.servlet.http.HttpServletResponse response)
                       throws javax.servlet.ServletException, IOException {

        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Good morning </title>");
        out.println("</head>");
        out.println("<body>");
        out.println(" First Servlet");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Перед определением класса указана аннотация WebServlet, которая указывает, с какой конечной точкой будет сопоставляться данный сервлет. То есть данный сервлет будет обрабатывать запросы по адресу "/login".

Для обработки GET-запросов (например, при обращении к сервлету из адресной строки браузера) сервлет должен переопределить метод doGet. В данном случае get-запрос по адресу /login будет обрабатываться методом doGet.

Этот метод принимает два параметра. Параметр типа HttpServletRequest инкапсулирует всю информацию о запросе. А параметр типа HttpServletResponse позволяет управлять ответом. А с помощью метода getWriter()объекта HttpServletResponse мы можем получить объект PrintWriter, через который можно отправить какой-то определенный ответ пользователю. В данном случае через метод println() пользователю отправляет простейший html-код. По завершению использования объекта HttpServletResponse его необходимо закрыть с помощью метода close().

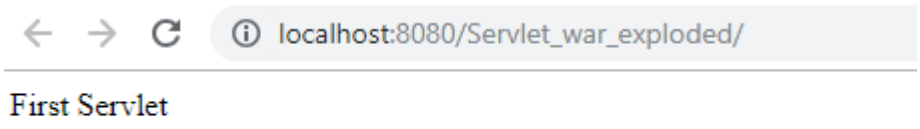
Зайдите в файл WEB-INF\web.xml и внесите изменения. Так как мы используем аннотации, то настройку в дескрипторе развертывания можно закомментировать.

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
    <!--<servlet>-->
        <!--<servlet-name>LoginServlet</servlet-name>-->
        <!--<servlet-class>by.iba.patsei.LoginServlet</servlet-class>-->
    <!--</servlet>-->
    <welcome-file-list>
        <welcome-file>login</welcome-file>
    </welcome-file-list>

</web-app>
```

Мы настроили welcome-файл логина.

Запустите проект



Мы получили следующее: urlPattern для LoginServlet - «/login ». Мы определили метод doGet (запрос HttpServletRequest, ответ HttpServletResponse) в LoginServlet. Метод doGet обрабатывает запросы GET к шаблону URL «/login».

Переопределите еще 3 метода жизненного цикла init, destroy, service.

Теория

Итак, для обработки запроса в HttpServlet определен ряд методов, которые мы можем переопределить в классе сервлета:

doGet: обрабатывает запросы GET (получение данных)
doPost: обрабатывает запросы POST (отправка данных)
doPut: обрабатывает запросы PUT (отправка данных для изменения)
doDelete: обрабатывает запросы DELETE (удаление данных)
doHead: обрабатывает запросы HEAD

Каждый метод обрабатывает определенный тип запросов HTTP, и мы можем определить все эти методы, но, зачастую, работа идет в основном с методами doGet и doPost.
Когда вы вводите URL-адрес браузера, браузер создает HTTP-запрос GET. Запрос HTTP GET получен веб-приложением, развернутым на сервере Tomcat.

Здесь можно почитать про запросы
https://www.w3schools.com/tags/ref_httpmethods.asp

Сервлеты. Контекст

Сервлет живет и умирает в пределах процесса сервера. Сервлет может получать информацию о своем окружении в различное время. Во время запуска сервлета доступна информация об инициализации; информация о сервере доступна в любое время, кроме этого, любой запрос может содержать дополнительную специфическую информацию.

Информация об инициализации сервера

Информация об инициализации передается сервлету при помощи параметра *ServletConfig* метода *init()*. Каждый Web-сервер обеспечивает свой способ передачи информации об инициализации в сервлет. Если, например, класс сервера *DatePrintServlet* принимает аргумент инициализации *timezone*, то необходимо определить следующие свойства в файле *servlets.properties*:

- servlet.dateprinter.code=DatePrinterServlet
- servlet.dateprinter.timezone=PST

Эта информация также может быть предоставлена административным средством GUI. В следующем коде сервлет получает доступ к информации *timezone* на этапе инициализации.

```
public void init(ServletConfig config)
{
```

```
    timeZone = config.getInitParameter("timezone");
}
```

Объект *Enumeration* со всеми параметрами инициализации доступен сервлету через метод *getInitParameterNames()*.

Информация о контексте сервера

Информация о контексте сервера доступна в любое время через объект *ServletContext*. Сервлет может получить этот объект, вызывая метод *getServletContext()* объекта *ServletConfig*. Необходимо помнить, что этот объект передается сервлету во время инициализации. Грамотно написанный метод *init()* сохраняет ссылку в переменной, имеющей тип доступа *private*. Интерфейс *ServletContext* определяет несколько методов, представленные в таблице:

getAttribute ()	Гибкий способ получения информации о сервере через пары атрибутов имя/значение. Зависит от сервера.
GetMimeType ()	Возвращает тип MIME данного файла.
getRealPath ()	Этот метод преобразует относительный или виртуальный путь в новый путь относительно месторасположения корня HTML-документов сервера.
getServerInfo ()	Возвращает имя и версию сетевой службы, в которой исполняется сервлет.
getServlet ()	Возвращает объект Servlet указанного имени. Полезен при доступе к службам других сервлетов.
getServletNames ()	Возвращает список имен сервлетов, доступных в текущем пространстве имен.
log ()	Записывает информацию в файл регистрации сервлета. Имя файла регистрации и его формат зависят от сервера.

Следующий пример показывает, как сервлет использует Web-сервер для записи сообщения в свой log-файл во время инициализации:

```
public HelloWorld implements Servlet
{
    private ServletConfig config;
    public void init (ServletConfig config) throws ServletException
    {
        this.config = config;
        ServletContext sc = config.getServletContext();
        sc.log( "Started OK!" );
    }
}
```

Контекст сервлета во время запроса на обслуживание

Каждый запрос на обслуживание может содержать информацию в форме пар параметров имя/значение, как *ServletInputStream*, или *BufferedReader*. Эта информация доступна при помощи объекта *ServletRequest*, который передается в метод *service()*. Следующий код показывает, как получить информацию во время работы 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");
}
```

Существует также дополнительная информация, доступная сервлету через объект *ServletRequest*. Она приведена 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 /..
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 HttpServletRequest

При каждом вызове методы *doGet* и *doPost* класса *HttpServlet* принимают в качестве параметра объект, который реализует интерфейс *HttpServletRequest*. Web-сервер, который исполняет сервлет, создает объект *HttpServletRequest* и передает его методу *service* сервлета (который в свою очередь передает его методу *doGet* или *doPost*). Данный объект содержит запрос, поступивший от клиента.

Имеется множество методов, дающих возможность сервлету обрабатывать клиентский запрос. Некоторые из этих методов принадлежат интерфейсу *ServletRequest* - интерфейсу, который расширяется интерфейсом *HttpServletRequest*.
Ряд ключевых методов, использованных в примерах, представлены в таблице. Полный список методов интерфейса *HttpServletRequest* можно найти в документации компании Sun.

Метод	**Описание**
String getParameter(String name)	Получение из запроса значения параметра. Наименование параметра определено значением *name.*
Enumeration getParameterNames()	Получение из запроса имен всех параметров.
String[ ] getParameterValues(String name)	Для параметра с несколькими значениями данный метод возвращает строковый массив.
Cookie[ ] getCookies ()	Получение массива объектов Cookie, сохраненных на компьютере клиента. Cookie могут быть использованы для уникальной идентификации клиента сервером.
HttpSession getSession(boolean create)	Возвращает объект *HttpSession* текущего сеанса

	клиента. Если параметр *create* равен *true* и объект *HttpSession* не существует, то создается новый объект *HttpSession.*

Интерфейс HttpServletResponse

При каждом обращении к сервлету методы *doGet* и *doPost* класса *HttpServlet* принимают объект, который реализует интерфейс *HttpServletResponse*. Web-сервер, который исполняет сервлет, создает объект *HttpServletResponse* и передает его методу *service* сервлета (который в свою очередь передает его методу *doGet* или *doPost*). Объект *HttpServletResponse* описывает ответ клиенту.

Имеется множество методов, дающих возможность сервлету сформировать ответ клиенту. Некоторые из этих методов принадлежат интерфейсу *ServletResponse* - интерфейсу, который расширяется интерфейсом *HttpServletResponse*. Ряд ключевых методов, использованных в примерах, представлены в таблице. Полный список методов интерфейса *HttpServletResponse* можно найти в документации компании Sun.

Метод	**Описание**
void addCookie (Cookie cookie)	Метод используется для добавления *Cookie* в заголовок ответа клинту. Установленный максимальный возвраст *Cookie*, а также разрешение клиентом хранения Cookie определяют, будут ли *Cookies* сохранены на клиенте и время их хранения.
ServletOutputStream getOutputStream()	Получение бинарного потока вывода для отправления бинарных данных клиенту.
PrintWriter getWriter	Получение символьного потока вывода для отправления текстовых данных клиенту.
void setContentType(String type)	Определение MIME-типа ответа браузеру. MIME-тип помогает браузеру определить, как отображать данные. Например, MIME-тип "text/html" указывает, что ответ является HTML-документом, поэтому браузер отображает HTML-страницу.