Лекция 17

СЕРВЛЕТЫ

Сервлеты – компоненты приложений Java 2 Platform Enterprise Edition, выполняющиеся на стороне сервера, способные обрабатывать клиентские запросы и динамически генерировать ответы на них.

Наибольшее распространение получили сервлеты, обрабатывающие клиентские запросы по протоколу HTTP.

В настоящее время сервлеты поддерживаются большинством Web-серверов и являются частью платформы J2EE.

Все сервлеты реализуют общий интерфейс **Servlet**.

Для обработки HTTP-запросов можно воспользоваться в качестве базового класса абстрактным классом **HttpServlet.**

Классы относящиеся к технологии сервлетов находятся в пакете **javax.servlet**.

Жизненный цикл сервлета начинается с его загрузки в память контейнером сервлетов при старте либо в ответ на первый запрос.

Далее происходят инициализация, обслуживание запросов и завершение существования.

Прежде всего вызывается конструктор сервлета(если он есть), а затем вызывается метод init().

Он дает сервлету возможность инициализировать данные и подготовиться для обработки запросов.

Данные задаются в файле web.xml в виде

```
<init-param>
  <param-name> имя_параметра</param-name>
  <param-value>значение_парам</param-value>
</init-param>
```

После этого сервлет можно считать запущенным, он находится в ожидании запросов от клиентов.

Появившийся запрос обслуживается методом **service()** сервлета, а все параметры запроса упаковываются в объект **ServletRequest**, который передается в качестве первого параметра методу **service()**.

Второй параметр метода – объект **ServletResponse**.

В этот объект упаковываются выходные данные в процессе формирования ответа клиенту.

Каждый новый запрос приводит к новому вызову метода **service()**.

Метод **service()** должен уметь обрабатывать сразу несколько запросов, т.е. быть синхронизирован для выполнения в многопоточных средах.

Если же нужно избежать множественных запросов, сервлет должен реализовать интерфейс **SingleThreadModel**, который не содержит ни одного метода и только указывает серверу об однопоточной природе сервлета.

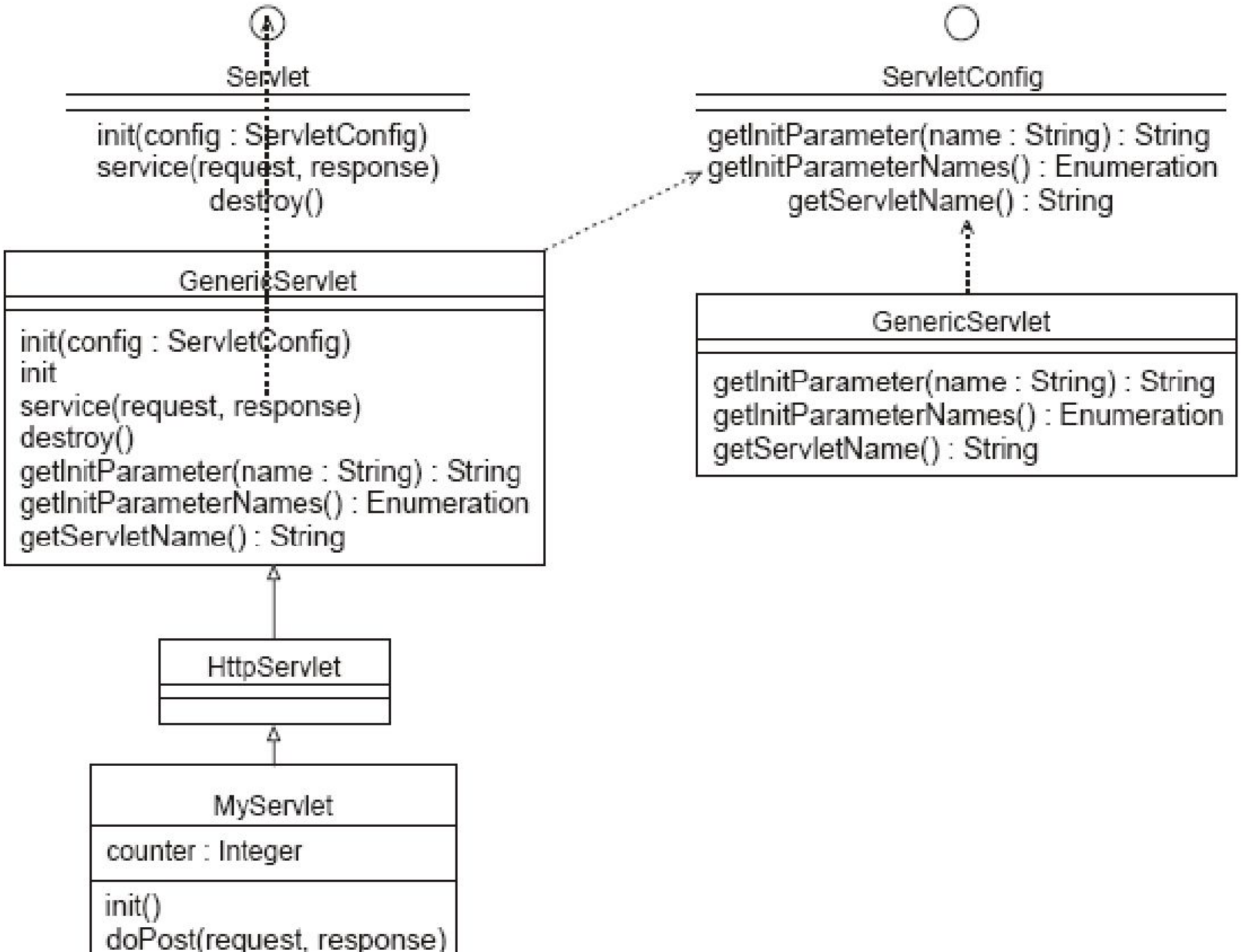
После завершения выполнения сервлета контейнер сервлетов вызывает метод **destroy()**, в теле которого следует помещать код освобождения занятых сервлетом ресурсов.

Интерфейсом **Servlet** предусмотрена реализация еще двух методов: **getServletConfig()** и **getServletInfo()**.

Первый возвращает объект типа **ServletConfig**, содержащий параметры конфигурации сервлета,

а второй – строку, описывающую назначение сервлета.

При разработке сервлетов в качестве базового класса в большинстве случаев используют не интерфейс **Servlet**, а класс **HttpServlet**, отвечающий за обработку запросов HTTP.


Servlet
init(config : ServletConfig)
service(request, response)
destroy()
ServletConfig
getInitParameter(name : String) : String
getInitParameterNames() : Enumeration
getServletName() : String
GenericServlet
init(config : ServletConfig)
init
service(request, response)
destroy()
getInitParameter(name : String) : String
getInitParameterNames() : Enumeration
getServletName() : String
GenericServlet
getInitParameter(name : String) : String
getInitParameterNames() : Enumeration
getServletName() : String
HttpServlet
MyServlet
counter : Integer
init()
doPost(request, response)

Класс **HttpServlet** имеет реализованный метод **service()**, служащий диспетчером для других методов, каждый из которых обрабатывает методы доступа к ресурсам.

В спецификации HTTP определены следующие методы: **GET**, **HEAD**, **POST**, **PUT**, **DELETE**, **OPTIONS** и **TRACE**.

Наиболее часто употребляются методы **GET** и **POST**, с помощью которых на сервер передаются запросы, а также параметры для их выполнения.

При использовании метода **GET** (по умолчанию) параметры передаются как часть URL, значения могут выбираться из полей формы или передаваться непосредственно через URL.

При этом запросы кэшируются и имеют ограничения на размер.

При использовании метода **POST** (**method=POST**) параметры (поля формы) передаются в содержимом HTTP-запроса и упакованы согласно полю заголовка **Content-Type**.

По умолчанию в формате:
<имя>=<значение>&<имя>=<значение>&...

В задачу метода **service()** класса **HttpServlet** входит анализ полученного через запрос метода доступа к ресурсам и вызов метода, имя которого сходно с названием метода доступа к ресурсам, но перед именем добавляется префикс **do**: **doGet()** или **doPost()**.

Кроме этих методов могут использоваться методы: **doHead()**, **doPut()**, **doDelete()**, **doOptions()** и **doTrace()**.

Очевидно, разработчик должен переопределить нужный метод, разместив в нем функциональную логику.

Рассмотрим пример:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class MyServlet extends HttpServlet {
public MyServlet() {
  super();
.....................................................
}
public void init() throws ServletException {
.....................................................................
}
```

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                    throws ServletException, IOException {
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   out.print("This is ");
   out.print(this.getClass().getName());
   out.print(", using the GET method");}

public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
                     throws ServletException, IOException {
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   out.print("This is ");
   out.print(this.getClass().getName());
   out.print(", using the POST method");}
```

```
public void destroy() {
   super.destroy();
.....}}
```

Интерфейсы ServletRequest и HttpServletRequest

Поток данных поступает от клиента в виде закодированного и упакованного запроса.

Вызывая методы интерфейса **ServletRequest**, можно получать определенный набор данных, посланных клиентом.

Метод **getCharacterEncoding()** определяет символьную кодировку запроса.

Методы **getContentType()** и **getProtocol()** – MIME-тип пришедшего запроса, а также название и версию протокола соответственно.

Информацию об имени сервера, принявшего запрос, и порт, на котором запрос был “услышан” сервером, выдают методы **getServerName()** и **getServerPort()**.

Можно также узнать данные и о клиенте, от имени которого пришел запрос.

Его IP-адрес возвращается методом **getRemoteAddr()**, а его имя – методом **getRemoteHost()**.

Если необходим прямой доступ к содержимому полученного запроса,то можно вызвать метод **getInputStream()** или **getReader()**.

Первый возвращает ссылку на объект класса **ServletInputStream**, а второй – на **BufferedReader**.

После этого можно читать любой байт из полученного запроса, используя технику работы с потоками Java.

Если, обращаясь к серверу, клиент помимо универсального адреса задал параметры сервлету то их можно узнать посредством следующих методов.

getParameter(String name) возвращает значение параметра по его имени или **null**, если параметра с таким именем нет.

getParameterValues(String name) возвращает массив строк, а именно все значения параметра по его имени, причем параметр может иметь несколько значений.

getParameterNames() возвращает объект типа **Enumeration**, позволяющий узнать имена всех присланных параметров.

Интерфейс **HttpServletRequest** является производным от интерфейса **ServletRequest** и используется для получения информации в HTTP-сервлетах.

В интерфейсе **HttpServletRequest** имеются дополнительные методы:

getCookies() возвращает массив cookies.

getQueryString() возвращает строку запроса HTTP

getRemoteUser() используется для получения имени пользователя, выполнившего запрос(если пользователь авторизировался или null) при использовании container-based authentication(устанавливается в настройках Tomcat в server.xml)

Рассмотрим пример:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class RequestClass extends HttpServlet {
public void doGet(HttpServletRequest req,
                                    HttpServletResponse resp)
                    throws IOException, ServletException{
  resp.setContentType("text/html");
  PrintWriter out = resp.getWriter();
```

```
out.println("<HTML><HEAD>");
 out.println("<TITLE>Request Information </TITLE>");
 out.println("</HEAD><BODY>");
 out.println("<H3>Request Information </H3>");
 out.println("<BR>Method: " + req.getMethod());
 out.println("<BR>Request URI: " +
                                        req.getRequestURI());
 out.println("<BR>Protocol: " + req.getProtocol());
 out.println("<BR>PathInfo: " + req.getPathInfo());
 out.println("<BR>Remote Address: " +
                                      req.getRemoteAddr());
out.println("</BODY></HTML>");
out.close();}}
```

На экране получим:

Request Information

Method: GET

Request URI: /FirstProject/RequestClass

Protocol: HTTP/1.1

PathInfo: null

Remote Address: 127.0.0.1

Стоит отметить метод **req.getPathInfo().**

Рассмотрим его работу. Пример:

```
String pathInfo = request.getPathInfo();
String pathTrans = request.getPathTranslated();
String uri = request.getRequestURI();
System.out.println("pathInfo: " + pathInfo);
System.out.println("pathTrans: " + pathTrans);
System.out.println("uri: " + uri);
```

Тогда при запросе

http://localhost:8080/blog/tag/java+mac+foo

Получим на экране

pathInfo: /java+mac+foo

pathTrans: /Users/al/tomcat-4.1.31/webapps/blog/tag/java+mac+foo

uri: /blog/tag/java+mac+foo

Интерфейсы ServletResponse и HttpServletResponse

Генерируемые сервлетами данные передаются серверу-контейнеру с помощью объектов, реализующих интерфейс **ServletResponse**, а сервер, в свою очередь, пересылает ответ клиенту, инициировавшему запрос.

В интерфейсе **ServletResponse** объявлены следующие методы:

setContentType() задает MIME-тип генерируемых документов

getOutputStream() возвращает ссылку на поток **ServletOutputStream.**

getWriter() вернет ссылку на поток типа **PrintWriter**.

В интерфейсе **HttpServletResponse**, наследующем интерфейс **ServletResponse**, объявлены следующие методы:

addCookie() пересылает cookie на клиентскую станцию.

sendError(int sc, String msg) сообщает о возникших ошибках в качестве параметра передается код ошибки и при необходимости текстовое сообщение.

setDateHeader() добавляет в заголовок сообщения параметры.

setAttribute(String name, Object ob) метод устанавливает значения атрибутов компонентов, являющиеся внутренними параметрами для передачи информации между компонентами приложения, например от сервлета к странице JSP или другому сервлету.

Рассмотрим пример:

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class RequestHeader extends HttpServlet {
  public void doPost(HttpServletRequest request,
                                HttpServletResponse response)
                        throws IOException, ServletException{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    Enumeration e = request.getHeaderNames();
     while (e.hasMoreElements()){
        String name = (String)e.nextElement();
        String value = request.getHeader(name);
        out.println(name + " = " + value);
     }
  }
}
```

Html файл будет иметь вид:

```
<HTML>
 <HEAD><TITLE>index.html</TITLE></HEAD>
<H2>ЗАПУСК СЕРВЛЕТА</H2> <BODY>
<FORM action="RequestHeader"  method="POST">
<INPUT type="submit" value="Execute">
</FORM> </BODY> </HTML>
```

В браузере получим

accept = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwave-flash, */*
referer = http://localhost:8080/FirstProject/index.html
accept-language = en-us
content-type = application/x-www-form-urlencoded
accept-encoding = gzip, deflate
user-agent = Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
host = localhost:8080
content-length = 0
connection = Keep-Alive
cache-control = no-cache

Интерфейс ServletConfig

Интерфейс **ServletConfig** имеет следующие методы:

getServletName() позволяет получить имя сервлета.

getInitParameterNames() метод возвращает имена параметров инициализации в виде **Enumeration**.

getInitParameter(String n) позволяет получить значение конкретного параметра.

getServletContext() возвращает ссылку на **ServletContext**, используя которую можно узнать практически всю информацию о среде, в которой запущен и выполняется сервлет.

Пример простого сервлета.

```
import java.io.*;
import java.util.Locale;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet(
  urlPatterns={"/FirstServlettest"}
  initParams={
   @WebInitParam(name="name1" value="value1")
   @WebInitParam(name="name1" value="value2")
  } ) //Можно вместо аннотаций использовать XML файл wb.xml. Если присуствует и
      web.xml и аннотации, то xml имеет приоритет
public class FirstServlet extends HttpServlet {
private volatile int count;
public void init() throws ServletException {
  super.init();
  count = 0;
  ServletConfig sc=this.getServletConfig();
  Enumeration e=sc.getParameterNames()
  while(e.hasMoreElements()){
    String name=(String)e.nextElement();
    String value=sc.getInitParameter(name);
} }
```

```
public void doGet( HttpServletRequest req, HttpServletResponse res) throws
    ServletException, IOException {
  performTask(req, res);
}

public void doPost( HttpServletRequest req, HttpServletResponse res)
                    throws ServletException, IOException{
   performTask(req, res);
}

public void performTask(HttpServletRequest req,
        HttpServletResponse res) throws ServletException,  IOException {
  String title = "First Servlet";
  PrintWriter out = res.getWriter();
   res.setContentType("text/html; charset=Cp1251");
   out.println("<HTML><HEAD><TITLE>"+ title + "</TITLE>"
               + "</HEAD><BODY><H2>This page is generated "
                + "by FirstServlet<H2><H3>This is its "+ ++count+ "
                                                execution</H3></BODY></HTML>");
   out.close();
 }
}
```

Рассмотрим развертывание сервлета на сервере Tomcat.

В каталоге, где установлен Web сервер будут следующие подкаталоги:

/bin – содержит **startup**, **shutdown** и другие исполняемые файлы;

/conf – содержит конфигурационные файлы, в частности конфигурационный файл контейнера сервлетов **server.xml**;

/server – помещаются классы;

/logs – помещаются log-файлы;

/webapps – в этот каталог помещаются папки, содержащие сервлеты и другие компоненты приложения

В каталог **/webapps** необходимо поместить папку **/FirstProject.**

Папка **/FirstProject** должна содержать каталог **/WEB-INF**, в котором помещаются подкаталоги:

/classes – содержит класс сервлета **FirstServlet.class**;

/lib – содержит библиотеки классов (если они есть), упакованные в JAR-файлы (архивы java);

/src – содержит исходный файл сервлета **FirstServlet.java**;

а также **web.xml** – конфигурационный файл приложения.

В файле **web.xml** необходимо прописать имя и путь к сервлету. Например:

```
<web-app>
<servlet>
<servlet-name>FirstServletname</servlet-name>
<display-name>FirstServletdisplay</display-name>
<servlet-class>test.com.FirstServlet </servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>FirstServletname</servlet-name>
<url-pattern>/FirstServlettest</url-pattern>
</servlet-mapping>
</web-app>
```

Вызвать сервлет можно набрав в браузере

http://localhost:8080/FirstProject/FirstServlettest

Стоит отметить, что <display-name> - используется различными утилитами.

Извлечение информации из запроса

Пусть имеется следующая форма:

```
<HTML> <BODY>
<FORM action="testform" >
<H2> Name :
    <INPUT type="text" name="p0" value=" ">
    <BR>
    Credit: <INPUT type="text" name="p1" value="0"> <BR>
    Price: <INPUT type="text" name="p2" value="0"> <BR>
    Adress:<TEXTAREA name="Adress" rows=3 cols=20>
   </TEXTAREA>
   <BR>
  <INPUT type="submit" value="Submit"> <BR>
  </FORM></BODY></HTML>
```

Для обработки данных, полученных из полей формы, используется сервлет:

```
package test.com;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class FormRequest extends HttpServlet {
public void doGet(HttpServletRequest req,
                                HttpServletResponse resp)
          throws ServletException, IOException{
   performTask(req, resp);
}
```

```
public void performTask(HttpServletRequest req,
                                                HttpServletResponse resp) {
try {
 String val[] =  new String[3];
  int rest;
  for (int i = 0; i < val.length; i++)
  val[i] = req.getParameter("p" + i);
  int c = Integer.valueOf(val[1]).intValue();
  int p = Integer.valueOf(val[2]).intValue();
  rest = c - p;
  PrintWriter out = resp.getWriter();
  resp.setContentType("text/html; charset=Cp1251");
  out.println("<HTML><HEAD>");
  out.println("<TITLE>FormRequest</TITLE>");
  out.println("</HEAD><BODY><BR>");
  out.println("<TABLE BORDER=3><TR><TD>");
  out.println("Name</TD><TD>Credit</TD><TD>Price</TD><TD>
                                                          Rest ");
  out.println("</TD></TR><TR>");
```

```
 for (int i = 0; i < val.length; i++)
    out.println("<TD>" + val[i] + "</TD>");
    out.println("<TD>" + rest + "</TD></TR>");
    out.println("Adress:" +req.getParameter("Adress"));
    out.println("</TABLE></BODY></HTML>");
    out.close();}
 catch (Throwable e) {
    e.printStackTrace();
  }
 }
}
```

Cookie

Рассмотрим сервлет, который задает куки, а затем считывает их из браузера.

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.*;
public class FormRequest extends HttpServlet {
 public void doGet(HttpServletRequest req,
                                HttpServletResponse resp)
              throws ServletException, IOException{
   performTask(req, resp);
 }
```

```
public void performTask(HttpServletRequest req,
                                    HttpServletResponse resp) {
try {
   String name=req.getParameter("p0");
   String val="";
  //принимаем куки
   Cookie cookie[]=req.getCookies();
   PrintWriter out = resp.getWriter();
   resp.setContentType("text/html;charset=Cp1251");
   if((name==null)&&(cookie!=null))   {
     if(cookie[0].getName().equals("Hello")){
           val=cookie[0].getValue();
     }
```

```
 out.println("<HTML><HEAD>");
 out.println("<TITLE>FormRequest</TITLE>");
 out.println("</HEAD><BODY>"+"Hello "+val);
 out.println("</BODY></HTML>");    }
 if(name!=null)   {
  out.println("<HTML><HEAD>");
  out.println("<TITLE>FormRequest</TITLE>");
  out.println("</HEAD><BODY>"+"Hello "+name);
  out.println("</BODY></HTML>");
//Создаем куки и отсылаем их браузеру
 Cookie cook=new Cookie("Hello",name);
  resp.addCookie(cook);     }
```

```
if((name==null)&&(cookie==null)){
  out.println("<HTML> <BODY>");
  out.println("<FORM action=\"FormRequest\" >");
  out.println("<INPUT type=\"text\" name=\"p0\"
                                                value=\" \"><br> ");
  out.println("<INPUT type=\"submit\"  value=\"Submit\"> ");
  out.println("</FORM></BODY></HTML>");
 }
  out.close();
}
  catch (Throwable e) {
     e.printStackTrace();
  }
 }
}
```

Методы класса Cookie:

String getComment() Возвращает строку, описывающую назначение cookie (null, если не было установлено каких-либо комментариев с помощью метода setCommerit).

String getDomain() Возвращает строку, содержащую имя домена cookie.

Тем самым определяется, какие серверы могут получить cookie.

По умолчанию cookie посылаются серверу, который отправил cookie клиенту.

int getMaxAge() Возвращает целое число, представляющее максимальный возраст cookie в секундах.

String getName() Возвращает строку, содержащую имя cookie, как оно было установлено конструктором.

String getPath() Возвращает строку, содержащую префикс URL для cookie.

Cookie могут быть «нацелены» на определенные URL, которые включают, каталоги Web-сервера.

По умолчанию cookie возвращается сервисам, работающим в том же каталоге, что и сервис, который отправил cookie, или в подкаталоге этого каталога.

boolean getSecure() Возвращает булево значение, указывающее, должен ли cookie передаваться с использованием протокола, обеспечивающего безопасность (true)

String getValue() Возвращает строку, содержащую значение cookie, как оно было установлено методом setValue или конструктором.

int getVersion() Возвращает целое число, содержащее номер версии протокола cookies, используемого для создания cookie.

Значение, равное 0 (по умолчанию), указывает на изначальный протокол cookies, определенный Netscape.

Значение, равное 1, указывает на текущую версию, основанную на документе RFC 2109.

void setComment(String purpose) Комментарий, описывающий назначение cookie, которое, предоставляется пользователю браузером. (Некоторые браузеры дают возможность пользователю принимать каждый cookie индивидуально)

void setDomain(String pattern) Определяет, какие серверы могут получать cookie.

По умолчанию cookie отправляются серверу, который послал cookie клиенту.

Домен задается в виде “aaa.com”, что указывает на возможность получения этого cookie всеми серверами, имена доменов которых оканчиваются на aaa.com

void setMaxAge(int expiry) устанавливает максимальный возраст cookie в секундах.

void setPath(String uri) Устанавливает префикс «целевого» URL, указывающий каталоги на сервере, которые соответствуют сервисам, способным принимать этот cookie.

void setSecure(boolean flag) значение **true** указывает, что cookie должен посылаться только с использованием протокола, обеспечивающего безопасность.

void setValue(String newValue) задает значение cookie.

void setVersion(int v) Задает протокол cookies для этого cookie.

Session

Когда пользователь делает запрос на сервер, сервер создает временную сессию для идентификации пользователя.

Когда тот же пользователь переходит на другую страницу сайта, сервер теперь может его идентифицировать.

Сессия - это временное уникальное соединение между сервером и пользователем и используется для идентификации пользователя среди других запросов или посещений на других страницах сайтов.

Для организации сессий используется класс HttpSession.

Объект HttpSession может быть создан с помощью метода **getSession(boolean par)**, класса HttpServletRequest.

Если par имеет значение **true**, то сервлет должен создать уникальный объект HttpSession для клиента.

Если параметр равен **false**, то метод getSession возвращает null, если объект HttpSession для клиента пока не существует.

Методы класса HttpSession:

Object getAttribute(String name) возвращает объект связанный с именем name в сессии.

Enumeration getAttributeNames() возвращает имена всех объектов в данной сессии.

void setAttribute(String name, Object value) привязывает объект value с именем name к сессии

void removeAttribute(String name) удаляет объект с именем name из сессии.

String getId(). Каждая сессия, созданная сервером, имеет уникальный id, ассоциированный с ней для идентификации сессии среди других сессий.

Метод возвращает такой id.

long getCreationTime(). Возвращает long значение, определяющее дату и время создания данной сессии.

long getLastTimeAccess(). Возвращает значение long, обозначающее последний визит на сайте.

boolean isNew(). Возвращает boolean, определяя новая ли сессия.

void invalidate(). Аннулирует сессию.

Этот метод можно использовать на странице 'logout', позволяя пользователю закончить сессию. Если после этого пользователь снова зайдет на сайт - создастся новая сессия под него.

Object getValue(String name) то же, что и **getAttribute**

void putValue(String name, Object value) то же, что и **setAttribute.**

Рассмотрим пример:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class BookStoreServlet extends HttpServlet {
 public void doGet (HttpServletRequest req,
                                    HttpServletResponse res)
                    throws ServletException, IOException{
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
```

```
 out.println("<HTML<");
 out.println("<BODY>");
 out.println("<CENTER>");
 out.println("<H1>Welcome to
                                  WholeLottaBooks.com!</H1>");
out.println("Would you like to begin shopping?");
out.println("<FORM ACTION='BookStoreServlet‘
                                                  METHOD='POST'>");
out.println("<INPUT TYPE='submit' VALUE='submit'>");
out.println("</FORM>");
out.println("</CENTER>");
out.println("</BODY>");
out.println("</HTML>");}
```

```
public void doPost (HttpServletRequest req,
                    HttpServletResponse res)
        throws ServletException, IOException{

  res.setContentType("text/html");
  PrintWriter out = res.getWriter();
  ShoppingCart sc;
  HttpSession session = req.getSession(true);
```

```
if (session.getValue("Cart") == null){
    sc= new ShoppingCart();
    session.putValue("Cart",sc);
    int acct = new Random().nextInt();
    if (acct < 0){acct = acct -1;}
       ((ShoppingCart)session.getValue("Cart"))
                                          .setAccountNum(acct);
 }
  out.println("Your account number is " +
            ((ShoppingCart)session.getValue("Cart"))
                                          .getAccountNum());
}
}
```

Класс ShoppingCart имеет вид:

```
import java.util.*;
import java.io.Serializable;
 public class ShoppingCart implements Serializable {
   private int accountNum;
   private Vector Items;
   public ShoppingCart(){ Items = new Vector();}
   public int getAccountNum(){
       return this.accountNum;
   }
   public void setAccountNum(int accountNum){
      this.accountNum = accountNum;
   }
```

```
public Iterator getItems(){return this.Items.iterator(); }
 public boolean addItem(int item){
      return Items.add(new Integer(item));\
  }
 public String toString(){
      return "ShoppingCart for " + accountNum;
 }
}
```

Фильтры

Сервлетный фильтр, в соответствии со спецификацией, это Java-код, пригодный для повторного использования и позволяющий преобразовать содержание HTTP-запросов, HTTP-ответов и информацию, содержащуюся в заголовках HTML.

Сервлетный фильтр занимается предварительной обработкой запроса, прежде чем тот попадает в сервлет, и/или последующей обработкой ответа, исходящего из сервлета.

Сервлетные фильтры могут:

- перехватывать инициацию сервлета прежде, чем сервлет будет инициирован;
- определить содержание запроса прежде, чем сервлет будет инициирован;
- модифицировать заголовки и данные запроса, в которые упаковывается поступающий запрос;
- перехватывать инициацию сервлета после обращения к сервлету.

Сервлетный фильтр может быть конфигурирован так, что он будет работать с одним сервлетом или группой сервлетов.

Основой для формирования фильтров служит интерфейс **javax.servlet.Filter**, который реализует три метода:

```
void init (FilterConfig config) throws
                                        ServletException;
void destroy ();
void doFilter (ServletRequest request,
                        ServletResponse response,
                                FilterChain chain)
                throws IOException, ServletException;
```

Метод **init** вызывается прежде, чем фильтр начинает работать, и настраивает конфигурационный объект 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**.

Таким образом, фильтр имеет следующий вид:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
@WebFilter(  urlPatterns={“/*”}
   initParams={@WebInitParam(name=“active” value=“true”)}
   ) //Эти параметры также можно задавать и в файле web.xml
public class FilterConnect implements Filter{
    private FilterConfig config = null;
    private boolean active =false;
    public void init (FilterConfig config) throws
                                                    ServletException {
      this.config = config;
      String act = config.getInitParameter("active");
      if (act != null){
          active = (act.toUpperCase().equals("TRUE"));
      }
```

```
 public void doFilter (ServletRequest request,
                       ServletResponse response,
                       FilterChain chain)
          throws IOException, ServletException {
      if (active){
      ……..
      }
     chain.doFilter(request, response);
 }
public void destroy() {
     config = null; }
}
```

Интерфейс **FilterConfig** содержит метод для получения имени фильтра, его параметров инициализации и контекста активного в данный момент сервлета.

С помощью метода **doFilter** каждый фильтр получает текущий запрос **request** и ответ **response**, а также **FilterChain**, содержащий список фильтров, предназначенных для обработки.

В **doFilter** фильтр может делать с запросом и ответом всё, что необходимо.

Затем фильтр вызывает **chain.doFilter**, чтобы передать управление следующему фильтру.

После возвращения этого вызова фильтр может по окончании работы своего метода **doFilter** выполнить дополнительную работу над полученным ответом.

После того, как класс-фильтр откомпилирован, его необходимо установить в контейнер и "приписать"(map) к одному или нескольким сервлетам.

Например:

```
<filter>
  <filter-name>FilterName</filter-name>
  <filter-class>FilterConnect</filter-class>
  <init-param>
    <param-name>active</param-name>
    <param-value>true</param-value>
 </init-param>
</filter>
<-- Подключение фильтра к сервлету -->
<filter-mapping>
  <filter-name>FilterName</filter-name>
  <servlet-name>ServletName</servlet-name>
</filter-mapping>
```

В представленном коде дескриптора поставки **web.xml** объявлен класс-фильтр **FilterConnect** с именем **FilterName**.

Фильтр имеет параметр инициализации **active**, которому присваивается значение **true**.

Фильтр **FilterName** в разделе **<filter-mapping>** подключен к сервлету **ServletName**.

Порядок, в котором контейнер строит цепочку фильтров для запроса определяется следующими правилами:

цепочка, выстраивается в том порядке, в котором встречаются соответствующие описания фильтров в **web.xml**;

Для связи фильтра со страницами HTML или группой сервлетов необходимо использовать тег **<url-pattern>.**

Например:

```
<filter-mapping>
  <filter-name>FilterName</filter-name>
   <url-pattern>*.html</url-pattern>
</filter-mapping>
```

фильтр будет применен ко всем вызовам страниц HTML.

Рассмотрим пример использования фильтра.

Рассмотрим задачу нахождения минимального и максимального элемента массива.

Фильтр **MaxFilter** находит максимальный элемент массива:

```
public class MaxFilter implements Filter{
    private FilterConfig config = null;
    private boolean active = false;

  public void init (FilterConfig config) throws
                                                ServletException {
       this.config = config;
       String act = config.getInitParameter("active");
       if (act != null){
           active = (act.toUpperCase().equals("TRUE"));}
  }
```

```
public void doFilter (ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException{
        if (active){
            String param=request.getParameter("ArrayName");
             String[] mass=param.split(" ");
             int mas[]=new int[mass.length];
             int i=0;
             for(String s: mass){
                mas[i]=Integer.parseInt(s);
                i++;  }
             int max=mas[0];

             for(int j=1; j<mas.length;j++){
                if(max<mas[j]){
                    max=mas[j]; }
             }
            request.setAttribute("maxValue", new Integer(max));
      }
      chain.doFilter(request, response);
}
```

```
public void destroy(){
        config = null;
   }
}
```

Фильтр **MinFilter** находит минимальный элемент массива. Вызывается после фильтра **MaxFilter.**

```
public class MinFilter implements Filter{
    private FilterConfig config = null;
        private boolean active = false;
    ……
```

```
public void init (FilterConfig config) throws ServletException{
    ……}
public void destroy(){
        config = null;

    }
public void doFilter (ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        if (active){
//Данная строка нужна, чтобы убедиться, что первым был вызван
    фильтр MaxFilter
            Integer value=(Integer)request.getAttribute("maxValue");
            if(value!=null){
                String param=request.getParameter("ArrayName");
             String[] mass=param.split(" ");
             int mas[]=new int[mass.length];
             int i=0;
             for(String s: mass){
                mas[i]=Integer.parseInt(s);
                i++;
             }
```

```
 int min=mas[0];

                for(int j=1; j<mas.length;j++){
                    if(min>mas[j]){
                        min=mas[j]; }
                }
   request.setAttribute("minValue", new Integer(min));
 }
}
 chain.doFilter(request, response);
 }
}
```

Тогда сервлет обслуживающий запросы имеет вид:

```
public class ActionServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request,
                        HttpServletResponse response)
                    throws ServletException, IOException {
       processRequest(request, response);
   }
```

```
protected void processRequest(
                                HttpServletRequest request,
                                HttpServletResponse response)
                        throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");
Integer
        value=(Integer)request.getAttribute("maxValue");
Integer
       value1=(Integer)request.getAttribute("minValue");
int maxValue=value.intValue();
int minValue=value1.intValue();
PrintWriter out = response.getWriter();
```

```
try {
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet ActionServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>The max value is " + maxValue +"</h1>");
        out.println("<h1>The min value is " + minValue +"</h1>");
        out.println("</body>");
        out.println("</html>");

} finally {
        out.close();
    }
  }
}
```

Соответственно форма имеет вид:

```
<form method="post" action="MyServlet">
  <input type="text" name="ArrayName" size="20"
                                    maxlength="15" value="">
  <input type="submit" value="Подсчитать">
</form>
```

Файл web.xml в данном случае имеет вид:

```
<servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>ActionServlet</servlet-class>
</servlet>
```

```
<filter>
        <filter-name>MaxFilter</filter-name>
        <filter-class>MaxFilter</filter-class>
        <init-param>
            <param-name>active</param-name>
            <param-value>true</param-value>
        </init-param>
</filter>
<filter>
        <filter-name>MinFilter</filter-name>
        <filter-class>MinFilter</filter-class>
        <init-param>
            <param-name>active</param-name>
            <param-value>true</param-value>
        </init-param>
</filter>
```

```
<servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/MyServlet</url-pattern>
</servlet-mapping>
<filter-mapping>
    <filter-name>MaxFilter</filter-name>
        <servlet-name>MyServlet</servlet-name>
</filter-mapping>
<filter-mapping>
      <filter-name>MinFilter</filter-name>
        <servlet-name>MyServlet</servlet-name>
</filter-mapping>
```

Класс RequestDispatcher

В некоторых случаях необходимо обратиться к другому сервлету, странице JSP, документу HTML, XML или другому ресурсу.

Если требуемый ресурс находится в том же контексте, что и сервлет, который его вызывает, то для получения ресурса необходимо использовать метод

public RequestDispatcher
getRequestDispatcher(String path);

представленному в интерфейсе ServletRequest.

Здесь **path** - это путь к ресурсу относительно контекста.

Например, необходимо обратиться к сервлету Connect:

```
RequestDispatcher rd =
        request.getRequestDispatcher("Connect");
```

Если ресурс находится в другом контексте, то необходимо предварительно получить контекст методом

public ServletContext getContext(
String uripath);

интерфейса ServletContext, а потом использовать метод

public RequestDispatcher
getRequestDispatcher(String uripath);

интерфейса ServletContext.

Здесь путь uripath должен быть абсолютным, т.е. начинаться с наклонной черты /.

Например:

```
RequestDispatcher rd =
                        config.getServletContext()
.getContext("/prod").getRequestDispatcher(
                                "/prod/Customer");
```

Если требуемый ресурс - сервлет, помещенный в контекст под своим именем, то для его получения можно обратиться к методу

```
RequestDispatcher getNamedDispatcher (
                                String name);
```

интерфейса ServletContext.

Все три метода возвращают null, если ресурс недоступен или сервер не реализует интерфейс RequestDispatcher.

Как видно из описания методов, к другим ресурсам можно обратиться только через объект типа RequestDispatcher, который предлагает два метода обращения к ресурсу. Первый метод

public void forward (ServletRequest request, ServletResponse response);

просто передает управление другому ресурсу, предоставив ему свои аргументы request и response.

Вызывающий сервлет выполняет предварительную обработку объектов request и response и передает их вызванному сервлету или другому ресурсу, который окончательно формирует ответ response и отправляет его клиенту или, опять-таки, вызывает другой ресурс.

Например:

```
if (rd != null) {
        rd.forward (request, response);}
else {
        response.sendError(
            HttpServletResponse.SC_NO_CONTENT);
}
```

Вызывающий сервлет не должен выполнять какую-либо отправку клиенту до обращения к методу forward, иначе будет выброшено исключение класса **IllegalStateException**.

Если же вызывающий сервлет уже что-то отправлял клиенту, то следует обратиться ко второму методу

public void include (ServletRequest request, ServletResponse response);

Этот метод вызывает ресурс, который на основании объекта request может изменить тело объекта response.

Но вызванный ресурс не может изменить заголовки и код ответа объекта response.

Это естественное ограничение, поскольку вызывающий сервлет мог уже отправить заголовки клиенту.

Попытка вызванного ресурса изменить заголовок будет просто проигнорирована

Рассмотрим пример:

```
……….
request.getSession().setAttribute("model", model);
RequestDispatcher dispatcher = getServletContext()
                        .getRequestDispatcher("/form.jspx");
try {
   dispatcher.forward(request, response);
 } catch (ServletException e) {
   e.printStackTrace();
 } catch (IOException e) {
   e.printStackTrace();
 }
```

При этом form.jspx имеет вид:

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
          xmlns:c="http://java.sun.com/jsp/jstl/core" version="2.0">

<jsp:directive.page contentType="text/html"
                                                pageEncoding="UTF-8"/>

<jsp:element name="text">
    <jsp:attribute name="lang">EN</jsp:attribute>
     <jsp:body>
       <b>Значение:</b><c:out value="${model.result}"/><br/>
     </jsp:body>
  </jsp:element>
</jsp:root>
```

Обработка событий

Существует несколько интерфейсов, которые позволяют следить за событиями, связанными с сеансом, контекстом и запросом сервлета, генерируемыми во время жизненного цикла Web-приложения:

javax.servlet.ServletContextListener – обрабатывает события создания/удаления контекста сервлета;

javax.servlet.http.HttpSessionListener – обрабатывает события создания/удаления HTTP-сессии;

javax.servlet.ServletContextAttributeListener – обрабатывает события создания/удаления/модификации атрибутов контекста сервлета;

javax.servlet.http.HttpSessionAttributeListener – обрабатывает события создания/удаления/модификации атрибутов HTTP-сессии;

javax.servlet.http.HttpSessionBindingListener – обрабатывает события привязывания/разъединения объекта с атрибутом HTTP-сессии;

javax.servlet.http.HttpSessionActivationListener – обрабатывает события связанные с активацией/дезактивацией HTTP-сессии;

javax.servlet.ServletRequestListener – обрабатывает события создания/удаления запроса;

javax.servlet.ServletRequestAttributeListener – обрабатывает события создания/удаления/модификации атрибутов запроса сервлета.

Методы соответствующих интерфейсов.

ServletContextListener

Этот Listener позволяет разработчику "словить" момент когда ServletContext инициализируется либо уничтожается.

Его можно использовать, например, для открытия соединения с базой данных в момент создания контекста и закрытия соединения в момент уничтожения контекста.

contextDestroyed(ServletContextEvent e)

contextInitialized(ServletContextEvent e)

Соответственно ServletContextEvent содержит метод

ServletContext getServletContext(), который возвращает измененнный **ServletContext.**

Пример1.

```
WebServlet( urlPatterns={"/Servlet1"} )
public class Sevlet1 extends HttpServlet {
 protected void processRequest(HttpServletRequest request,
   HttpServletResponse response) throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
         out.println("<html>");
          out.println("<head>");
          out.println("<title>Servlet Sevlet1</title>");
          out.println("</head>");
          out.println("<body>");
          out.println("<h1>Servlet Sevlet1 " +"</h1>");
          out.println("</body>");
          out.println("</html>");
     } finally {  out.close(); }
   }
```

Обработчик события:

```
@WebListener()
//Зарегестрировать обработку событий можно также и в web.xml
// <listener>  ServletListener1 </listener>
public class ServletListener1 implements ServletContextListener {
 public void contextInitialized(ServletContextEvent sce) {
       System.out.println("created");
   }
 public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("destroyed");
   }
}
```

Теперь в консоли сервера при загрузке сервлета будет выведено created, а при выгрузке destroyed.

HttpSessionListener

Этот Listener позволяет разроботчику "словить" момент создания и уничтожения сессии.

sessionCreated(HttpSessionEvent e)

sessionDestroyed(HttpSessionEvent e)

Класс HttpSessionEvent имеет метод

HttpSession getSession() – возвращает сессию, которая была изменена.

ServletContextAttributeListener

Этот Listener используется для “прослушивания" событий, происходящих с атрибутами в контексте сервлета (ServletContext).

attributeAdded(ServletContextAttributeEvent e)

attributeRemoved(ServletContextAttributeEvent e)

attributeReplaced(ServletContextAttributeEvent e)

Вызывается когда атрибут меняет значение

Класс ServletContextAttributeEvent имеет два метода

String getName() - возвращает имя измененного атрибута

Object getValue() –возвращает значение атрибута, которое было удалено или добавлено.

HttpSessionAttributeListener

Этот Listener используется для "слушания" событий, происходящих с атрибутами в сервлет контексте (ServletContext).

attributeAdded(HttpSessionBindingEvent e)

Вызывается когда атрибут добавляется в ServletContext

attributeRemoved(HttpSessionBindingEvent e)

Вызывается когда атрибут удаляется из ServletContext-а

attributeReplaced(HttpSessionBindingEvent e)

Вызывается когда атрибут меняет значение

Класс **HttpSessionBindingEvent** имеет два метода

String getName()- возвращает имя аттрибута под которым объект был привязан к сессии

HttpSession getSession() - возвращает имя сессии, к которой объект был привязан.

HttpSessionBindingListener

Этот Listener так-же используется для прослушивания событий происходящих с атрибутами в сессии. Разница между **HttpSessionAttributeListener** и **HttpSessionBindingListener**
HttpSessionAttributeListener*:* декларируется в web.xml, экземпляр класса создается автоматически (контейнером) в единственном числе и применяется ко всем сессиям
HttpSessionBindingListener: экземпляр этого класса должен быть создан и закреплён за определённой сессией программистом "вручную", количество экземпляров регулируется программистом.

valueBound(HttpSessionBindingEvent e)

Вызывается когда объект добавляется в сессию

valueUnbound(HttpSessionBindingEvent e)

Вызывается когда объект удаляется из сессии

Рассмотрим пример.

```
public class MyBindingListener implements
                                        HttpSessionBindingListener {
   public void valueBound(HttpSessionBindingEvent e){
       System.out.println("Bound“ +e.getName());
    }
    public void  valueUnbound(HttpSessionBindingEvent e){
       System.out.println("UnBound“+ e.getName());


    }
}
```

Сервлет имеет вид:

```
public class Servlet1 extends HttpServlet {
protected void processRequest(HttpServletRequest request,
   HttpServletResponse response) throws ServletException, IOException {
      try {
         MyBindingListener mb=new MyBindingListener();
          HttpSession hs=request.getSession();
          hs.setAttribute("name", mb); // в этом месте в консоль сервера
                                                  будет выведено Bound name
          hs.setAttribute("name", "bbb"); // в этом месте в консоль сервера
                                                  будет выведено UnBound name
       } finally {
       }
   }
 protected void doPost(HttpServletRequest request,
   HttpServletResponse response) throws ServletException, IOException {
         processRequest(request, response); }
}
```

HttpSessionActivationListener

Этот Listener используется атрибутами сессии в случае, если сессия будет "мигрировать" между различными JVM в распределённых приложениях.

Имеет следующие методы:

void sessionWillPassivate(HttpSessionEvent se)

Вызывается перед тем, как сессия станет пассивной перед миграцией

void sessionDidActivate(HttpSessionEvent se)

Вызывается после того, как сессия стала активной после миграции

ServletRequestListener

Этот Listener используется, соответственно, для того, чтоб "словить" момент создания и уничтожения запроса.

void requestDestroyed(ServletRequestEvent se)

Вызывается когда запрос уничтожается

void requestInitialized(ServletRequestEvent se)

Вызывается когда запрос инициализируется

ServletRequestEvent имеет два метода

ServletContex getServletContext() возвращает контекст сервлета для текущего приложения

ServletRequest getServletRequest() возвращает измененній запрос.

ServletRequestAttributeListener

Этот Listener используется при “прослушивании" событий происходящих с атрибутами запроса.

void attributeAdded(ServletRequestAttributeEvent e)

Вызывается когда атрибут добавляется в запрос

void attributeRemoved(ServletRequestAttributeEvent e)

Вызывается когда атрибут удаляется из запроса

void attributeReplaced(ServletRequestAttributeEvent e)

Вызывается когда атрибут меняет значение

Класс **ServletRequestAttributeEvent**

имеет два метода

String getName() Возвращает имя измененного атрибута

Object getValue() – Возвращает значение добавленного, удаленного или измененного атрибута.