01.04.2020
Servlets API

Намиот Д.Е.
dnamiot@gmail.com

Фильтры: примеры

• Pre & Post – processing.

```
public void doFilter(ServletRequest request, ServletResponse
response,
        FilterChain chain)
        throws java.io.IOException, javax.servlet.ServletException
{
    long start = System.currentTimeMillis();
    System.out.println("Milliseconds in: " + start);

    chain.doFilter(request, response);

    long end = System.currentTimeMillis();
    System.out.println("Milliseconds out: " + end);
}
```

Фильтры: примеры

- Pre & Post – processing.

```
public void doFilter(ServletRequest request, ServletResponse
response,
      FilterChain chain)
      throws java.io.IOException, javax.servlet.ServletException
{
   long start = System.currentTimeMillis();
   System.out.println("Milliseconds in: " + start);

   request.setAttribute(“start_time”, new Long(start));
   chain.doFilter(request, response);

   long end = System.currentTimeMillis();
   System.out.println("Milliseconds out: " + end);
}
```

Фильтры: примеры

Переопределение базовых классов:

Wrapper: упаковка класса:

Для манипуляций с запросами расширяем
javax.servlet.http.HttpServletRequestWrapper class

Для манипуляций с откликом расширяем
javax.servlet.http.HttpServletResponseWrapper class

Если необходим специальный функционал (манипуляции)
с выводом, то переопределяем
javax.servlet.ServletOutputStream class

Фильтр будет использовать переопределенные классы

Фильтры: примеры

Переопределение базовых классов:

Wrapper: упаковка класса:

Для манипуляций с запросами расширяем
javax.servlet.http.HttpServletRequestWrapper class

Для манипуляций с откликом расширяем
javax.servlet.http.HttpServletResponseWrapper class

Если необходим специальный функционал (манипуляции)
с выводом, то переопределяем
javax.servlet.ServletOutputStream class

Фильтр будет использовать переопределенные классы

Фильтры: примеры

```
public class FilterServletOutputStream extends ServletOutputStream {

  private DataOutputStream stream;

  public FilterServletOutputStream(OutputStream output) {
    stream = new DataOutputStream(output);
  }

  public void write(int b) throws IOException  {
    stream.write(b);
  }

  public void write(byte[] b) throws IOException  {
    stream.write(b);
  }

  public void write(byte[] b, int off, int len) throws IOException  {
    stream.write(b,off,len);
  }

}
```

Фильтры: примеры

```
public class GenericResponseWrapper extends HttpServletResponseWrapper {
  private ByteArrayOutputStream output;
  private int contentLength;
  private String contentType;

  public GenericResponseWrapper(HttpServletResponse response) {
    super(response);     output=new ByteArrayOutputStream();
  }

  public byte[] getData() {     return output.toByteArray();    }

  public ServletOutputStream getOutputStream() {     return new FilterServletOutputStream(output);   }

  public PrintWriter getWriter() {     return new PrintWriter(getOutputStream(),true);   }

  public void setContentLength(int length) {
    this.contentLength = length;     super.setContentLength(length);
  }

  public int getContentLength() {      return contentLength;   }

  public void setContentType(String type) {
    this.contentType = type;     super.setContentType(type);
  }

  public String getContentType() {
    return contentType;   }
}
```

Фильтры: примеры

```
public void doFilter(final ServletRequest request,
                     final ServletResponse response,
                     FilterChain chain)
      throws IOException, ServletException {

  OutputStream out = response.getOutputStream();
  out.write(new String("<HR>PRE<HR>").getBytes());

  GenericResponseWrapper wrapper =
        new GenericResponseWrapper((HttpServletResponse)
response);
  chain.doFilter(request,wrapper);

  out.write(wrapper.getData());
  out.write(new String("<HR>POST<HR>").getBytes());
  out.close();
  }
```

Системные события

- События, которые происходят во время жизненного цикла.
- Общее название WebListener

- Класс, который реализует один или несколько интрефейсов из списка:
- ServletContextListener,
- ServletContextAttributeListener
- ServletRequestListener
- ServletRequestAttributeListener
- HttpSessionListener
- HttpSessionAttributeListener
- HttpSessionIdListener

Системные события

- Аннотация:

```
public @interface WebListener
```

- web.xml

```
<web-app>

  <listener>
    <listener-class>com.acme.MyConnectionManager</listenerclass>
  </listener>
  <listener>
    <listener-class>com.acme.MyLoggingModule</listener-class>
  </listener>

 <servlet>
    <display-name>RegistrationServlet</display-name>
    ...
  </servlet>
</web-app>
```

Системные события

• ServletContextListener
Создание и удаление контекста. Старт и остановка контейнера

• void contextInitialized(ServletContextEvent sce)
Создание контекста

• void contextDestroyed(ServletContextEvent sce)
Удаление контекста

• Использование

```
void contextInitialized(ServletContextEvent sce)
{
  sce.setAttribute(“key”, some_common_data);
}
```

Системные события

ServletContextAttributeListener

void attributeAdded(ServletContextAttributeEvent event)
Добавление атрибута к ServletContext.

void attributeRemoved(ServletContextAttributeEvent event)
Удаление атрибута из ServletContext.

void attributeReplaced(ServletContextAttributeEvent event)
Изменение атрибута

String name = event.getName()
Object value = event.getValue()

Системные события

ServletRequestListener

void requestInitialized(ServletRequestEvent sre)
Появление нового ServletRequest. Начало обработки нового запроса (фильтр или сервлет).

void requestDestroyed(ServletRequestEvent sre)
Завершение обработки запроса (фильтр или сервлет).

ServletContext context = sre.getServletContext()
ServletRequest request = sre. getServletRequest()

Системные события

ServletRequestAttributeListener

void attributeAdded(ServletRequestAttributeEvent srae)
Добавление атрибута к некоторому ServletRequest.

void attributeRemoved(ServletRequestAttributeEvent srae)
Удаление атрибута из ServletRequest.

void attributeReplaced(ServletRequestAttributeEvent srae)
Изменение атрибута в некотором ServletRequest.

String name = srae.getName()
Object value = srae.getValue()
ServletContext context = srae.getServletContext()
ServletRequest request = srae.getServletRequest()

Системные события

HttpSessionListener

void sessionCreated(HttpSessionEvent se)
Создание сессии

void sessionDestroyed(HttpSessionEvent se)
Удаление сессии

HttpSession sess = se.getSession()

Пример использования: счетчик сессий. Количество посетителей сайта в данный момент.

Системные события

HttpSessionAttributeListener

void attributeAdded(HttpSessionBindingEvent event)
Добавление атрибута к сессии.

void attributeRemoved(HttpSessionBindingEvent event)
Удаление атрибута сессии.

void attributeReplaced(HttpSessionBindingEvent event)
Изменение атрибута сессии.

Асинхронная обработка

JVM поддерживает исполнение параллельных задач (процессов). Это Java thread

Они могут, например, реально исполняться параллельно (native thread - e.g. разные ядра процессора), или параллелизм будет имитироваться (green thread). Современные JVM – native threads

По умолчанию, контейнер использует одну задачу (Java thread) на один запрос

Асинхронная обработка означает, что сервлет запускает новые задачи.

Это именно для нагруженных систем – принять запрос, быстро запустить его обработку и быть готовым к приему новых запросов

Асинхронная обработка

Два сценария, когда запрос будет ожидать чего-либо:

- Ожидаем получения какого-либо ресурса, прежде чем сможем сформировать отклик. Типичный пример: в коде сервлета какой-то сложный запрос к базе данных. Отклик (выдачу) можно сформировать только после выполнения запроса.

- Для формирования отклика нужно ждать какого-то события. Например, появление новых данных в очереди, записи в базе данных и т.п.

- Общее – в коде появляются блокирующие операции. doGet() или doPost() не завершится, пока не разблокируются эти операции. Соответственно, данная копия сервлета будет занята и не сможет принимать новые запросы. Нужно будет подгружать новые, что занимает время.

Асинхронная обработка

Общее решение:

Завершить doGet() или doPost() как можно скорее без выдачи результата. Копия сервлета освободится и будет способна к приему новых запросов. Не нужно будет увеличивать пул сервлетов.

Все блокирующие операции переместить в новую задачу. Она и сформирует отклик

Внимание: это никак не ускорит выдачу клиенту. Если нужно ждать завершения запроса из БД – его нужно ждать. Но это даст возможность обрабатывать много запросов.

Уведомлять клиента об ожидании можно при отправке запроса (Ajax).

Асинхронная обработка

Общее решение:

Завершить doGet() или doPost() как можно скорее без выдачи результата. Копия сервлета освободится и будет способна к приему новых запросов. Не нужно будет увеличивать пул сервлетов.

Все блокирующие операции переместить в новую задачу. Она и сформирует отклик

Внимание: это никак не ускорит выдачу клиенту. Если нужно ждать завершения запроса из БД – его нужно ждать. Но это даст возможность обрабатывать много запросов.

Уведомлять клиента об ожидании можно при отправке запроса (Ajax).

Асинхронная обработка

```
@WebServlet(urlPatterns={"/asyncservlet"}, asyncSupported=true)
public class MyServlet extends HttpServlet { ... }
```

The javax.servlet.AsyncContext class – здесь весь функционал для асинхронной обработки. Экземпляр класса “добывается” из HttpServletRequest.

```
public void doGet(HttpServletRequest req, HttpServletResponse resp) {
   ...
   AsyncContext acontext = req.startAsync();
   ...
}
```

Асинхронная обработка

void start(Runnable run) – запуск процесса

ServletRequest getRequest() – узнать “родительский” запрос

ServletResponse getResponse() – объект для связи с контейнером

void complete() – завершение работы

void dispatch(String path) – передать обработку другому сервлету

Асинхронная обработка

```
@WebServlet(urlPatterns={"/syncservlet"})
public class SyncServlet extends HttpServlet {
  private MyRemoteResource resource;
  @Override
  public void init(ServletConfig config) {
    resource = MyRemoteResource.create("config1=x,config2=y");
  }

  @Override
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response) {
    response.setContentType("text/html;charset=UTF-8");
    String param = request.getParameter("param");
    String result = resource.process(param);
    /* ... print to the response ... */
  }
}
```

Асинхронная обработка

```
@WebServlet(urlPatterns={"/asyncservlet"}, asyncSupported=true)
public class AsyncServlet extends HttpServlet {
  /* ... Same variables and init method as in SyncServlet ... */

  @Override
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response) {
    response.setContentType("text/html;charset=UTF-8");
    final AsyncContext acontext = request.startAsync();
    acontext.start(new Runnable() {
      public void run() {
        String param = acontext.getRequest().getParameter("param");
        String result = resource.process(param);
        HttpServletResponse response = acontext.getResponse();
        /* ... print to the response ... */
        acontext.complete();
  }
}
```

JSP

- HTML + Java
- Язык разметки + фрагменты кода на Java
- Транслируется на лету в Java servlet

```
<html>
  <head>
     <title>JSP Page</title>
  </head>
  <body>
     <p>Today is <%=new java.util.Date()%> </p>
     <p> <% out.print("Hello World");%>  </p>
  </body>
</html>
```

JSP - трансляция

• page.jsp -> page.jsp.java
Исходный файл преобразуется в Java сервлет

• page.jsp.java - > page.jsp.class
Полученный сервлет компилируется

• page.jsp.class запускается как обычный сервлет
И генерирует отклик

• Это выполняется автоматически при первом запросе (чаще всего) или принудительно, до начала выполнения запросов (реже)

• Если исходный файл не меняется, то не меняется и код сервлета

• Проверка на изменение: сравнить даты последней модификации page.jsp и page.jsp.class

JSP - трансляция

• page.jsp

```
<html>
  <p> Hello, world </p>
</html>
```

• Как это вывести в сервлете?

• Открыть выходной поток и просто напечатать туда этот код.

• Мы можем сгенерировать Java-код, который выводит этот текст

JSP - трансляция

```
public class page.jsp extends GenericServlet{

public void service(ServletRequest request,ServletResponse response)
throws IOException,ServletException{
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        out.print("<html>");
        out.print("<p>Hello, world</p>");
        out.print("</html>");
  }
 }
```

Цветом выделен текст, который меняется (зависит от исходного файла)

JSP - трансляция

page.jsp

```
<html>
  <p> Hello, world </p>
  <% out.println(“<p>It is Java</p>”); %>
</html>
```

Как это вывести в сервлете?

Открыть выходной поток и просто напечатать туда код, который не относится к Java

Java-код – перенести “как есть”

Java-код – определяется по специальной разметке <% (выделена цветом)

JSP - трансляция

```
public class page.jsp extends GenericServlet{

public void service(ServletRequest request,ServletResponse response)
throws IOException,ServletException{
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        out.print("<html>");
        out.print("<p>Hello, world</p>");

        // это Java-код
        out.println(“<p>It is Java</p>”);

        out.print("</html>");
  }
 }
```

Цветом выделен текст, который меняется (зависит от исходного файла)

JSP – предопределенные объекты

В Java-коде на JSP странице для вывода информации мы использовали переменную out

В предположении, что эта переменная ссылается на дескриптор для вывода данных

Так и есть – в результирующем сервлете, до пользовательского кода (кода, перенесенного из JSP), определена переменная out:

```
out = response.getWriter();
```

Соответственно, код, который будет добавлен после этого будет успешно транслирован

Это приводит к тому, что в JSP коде мы можем использовать другие переменные, определенные в результирующем сервлете

JSP – предопределенные объекты

- public void service(ServletRequest ***request***,ServletResponse ***response***)
- Отсюда две переменных:
  request
  response

  Технически там приведение типа:
  ```
  public void service(ServletRequest req,ServletResponse res)
  {
     HttpServletRequest request = (HttpServletRequest) req;
     HttpServletResponse response = (HttpServletResponse) res;

     …
  }
  ```

- Определяются в сервлете:
  JspWiter out; <- отличается от PrintWriter !
  HttpSession session = request.getSession();

JSP – предопределенные объекты

• application
Это ссылка на ServletContext

• pageContext
Объект типа PageContext

Создается на этапе выполнения. Можно рассматривать как единую точку доступа ко всем свойствам (параметрам) страницы:

pageContext.getRequest()
pageContext.getSession()
pageContext.getServletContext()
И
pageContext.setAttribute(String name, Object value)
pageContext.setAttribute(String name, Object value, int scope)

pageContext.getAttribute(name)
pageContext.getAttribute(name, scope)

JSP – компиляция

```
<html>
<p>Заголовок страницы</p>
<%  int i=10; %>
<p>Текст страницы</p>
<p>Значение i:
<% out.println(i+1); %>
</html>
```

Транслируется в теле сервлета:

```
out.print(“<html>”);
out.print(“<p>Заголовок страницы</p>”);
int i=10;   // Java код транслируется “as is”
out.print(“<p>Текст страницы</p>”);
out.print(“<p>Значение i:”);
out.println(i+1);
out.print(“</html>”);
```

JSP – компиляция

```
<html>
<% if (request.getRemoteAddr().startsWith(“192.168”)) { %>
    <p>Это пользователь из локальной сети</p>
<% } else { %>
    <p>Это пользователь из глобальной сети </p>
<% } %>
</html>
```

Транслируется в теле сервлета:

```
out.print(“<html>”);
if (request.getRemoteAddr().startsWith(“192.168”)) {
   out.print(“<p> Это пользователь из локальной сети </p>”);
} else {
   out.print(“Это пользователь из глобальной сети</p>”);
}
out.print(“</html>”);
```

JSP – компиляция

```
<html>
<table>
<% for (int i=1; i<=5; i++) { %>
<tr><td>
   <% out.println(“строка “+i); %>
</td></tr>
<% } %>
</table>
</html>
```

Транслируется в теле сервлета:

```
out.print(“<html>”); out.print(“<table>”);
for (int i=1; i<=5; i++) {
 out.print(“<tr><td> ”);
 out.println(“строка “+i);
 out.print(“</td></tr> ”);
}
out.print(“</table>”); out.print(“</html>”);
```

JSP – компиляция

```
<html>
<p> Строка 1
<% return; %>
<p> Строка 2
</html>
```

Какой эффект:

Строка 1

Ошибка компиляции ?

JSP – компиляция

<html>

Строка 1

Компилируется в:

out.print(“ “);
out.print(“
Строка 1”);

Пустые строки – значимы !

JSP – скритлет

<% out.println(some_expression); %>

Эквивалентно

<%= some_expression %>

Например

<% out.println(new java.util.Date()); %>

И

<%= new java.util.Date()%>

Note: нет ; в конце выражения