18.03.2020
Servlets API

Намиот Д.Е.
dnamiot@gmail.com

Пример сервлета

```
import java.io.*;
import javax.servlet.*;

public class TestServlet extends HttpServlet {

public void doGet(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {

      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      out.print("<html><body>");
      out.print("<h3>Hello Servlet</h3>");
      out.print("</body></html>");
   }
}
```

Пример сервлета

- Здесь в коде:
- Контейнер разбирает входящий HTTP запрос.
- Исходя из заданного отображения и URI в запросе определяется сервлет (Java класс), который должен обрабатывать запрос
- Выбирается загруженный экземпляр класса из пула или создается новый экземпляр
- При необходимости вызывается метод init(). В данном случае – он пустой
- В соответствии с командой HTTP запроса вызывается метод *doGet()* или *doPost()*

Пример сервлета

- *HttpServletRequest request* – инициализованный экземпляр класса *HttpServletRequest,* который содержит всю информацию, извлеченную из HTTP запроса
- *HttpServletResponse response* - инициализованный экземпляр класса *HttpServletResponse*, который используется для связи с контейнером (для выдачи результатов клиенту)
- *response.setContentType("text/html")* – устанавливаем заголовок HTTP отклика
- Статус по умолчанию – 200. Может быть задан явно: *response.setStatus (200)*
- *PrintWriter out = response.getWriter()* - *PrintWriter* есть стандартный дескриптор для текстового вывода в Java. В данном случает он получен “от контейнера”: *response.getWriter()* и соответствует выдаче данных в выходной поток (перенаправляет выдачу в выходной поток)

Пример сервлета

- В соответствии с установленным content-type сервлет формирует текстовую выдачу в выходной поток
- В данном случае мы выводим HTML код:
- out.print("<html><body>");
  out.print("<h3>Hello Servlet</h3>");
  out.print("</body></html>");
- В итоге клиент получит следующий отклик:

HTTP/1.1 200 Ok
Content-Type: text/html

<html><body>
<h3>Hello Servlet</h3>
</body></html>

Пример сервлета

- Далее уже клиентское приложение осуществляет обработку полученного отклика
- Если в качестве приложения выступает браузер – то он понимает, что отклик пришел в известном ему формате (HTML) и занимается “отрисовкой” HTML
- Если это какое-то пользовательское (вами написанное) приложение, то обработкой отклика занимается оно. Как это приложение будет трактовать (обрабатывать) HTML код определяется им самим.
- Как правило, в языке программирования будут какие-то библиотеки (пакеты), которые будут поддерживать отправку HTTP-запросов и получение откликов
- Отсюда следует главный вывод по модели работы сервлетов: изнутри Java-кода (в Java-коде) формируем контент (содержимое ответа на HTTP запрос).

HttpServletRequest

- Расширяет ServletRequest (класс из модели GenericServlet)
- String getMethod() – команда HTTP протокола
- String getLocalAddr() – локальный IP адрес
- String getRemoteAddr() – IP адрес клиента (или последнего proxy-приложения), от которого пришел запрос
- String getParameter(String name) – получение параметра по имени. Так, например, передаются данные из форм в HTML
- Map<String,String[]> getParameterMap() возвращает все параметры в агрегате Map
- Enumeration<String> getParameterNames() – перечисление с именами параметров
- String[] getParameterValues(String name) все значения параметра или null, если такого параметра не существует

HttpServletRequest

- Cookie[] getCookies() – массив заголовков Cookie
- long getDateHeader(String name) – заголовок Date
- String getHeader(String name) – заголовок по имени
- Enumeration<String> getHeaderNames() – все имена заголовков
- Enumeration<String> getHeaders(String name) – все значения указанного заголовка или null, если такого заголовка нет

- Различные методы выделения частей URI в HTTP запросе:
- String getPathInfo()
- String getQueryString()
- String getRequestURI()
- StringBuffer getRequestURL()
- String getServletPath()

HttpServletRequest

- Cookie[] getCookies() – массив заголовков Cookie
- long getDateHeader(String name) – заголовок Date
- String getHeader(String name) – заголовок по имени
- Enumeration<String> getHeaderNames() – все имена заголовков
- Enumeration<String> getHeaders(String name) – все значения указанного заголовка или null, если такого заголовка нет

- Различные методы выделения частей URI в HTTP запросе:
- String getPathInfo()
- String getQueryString()
- String getRequestURI()
- StringBuffer getRequestURL()
- String getServletPath()

URL

- http://www.example.com:80/my-application/path/to/servlet/path/info?a=1&b=2#boo

- Части строки запросов:

- scheme: http
- hostname: www.example.com
- port: 80
- context path: my-application
- servlet path: path/to/servlet
- path info: path/info
- query: a=1&b=2
- fragment: boo
- getRequestURL:http://www.example.com:80/my-application/path/to/servlet/path/info
- getRequestURI:/my-application/path/to/servlet/path/info
- getQueryString: a=1&b=2#boo

HttpServletResponse

- Расширяет ServletResponse
- ServletOutputStream getOutputStream() - ServletOutputStream дескриптор для записи двоичных данных
- PrintWriter getWriter() PrintWriter дескриптор для записи текстовых данных
- void setCharacterEncoding(java.lang.String charset)
  установка кодировки (e.g. UTF-8)
- void setContentLength(int len) – размер выдачи (заголовок Content-Length)
- void setContentType(java.lang.String type)
  установка заголовка Content-Type
- void setDateHeader(java.lang.String name, long date) – заголовок Date
- void setHeader(java.lang.String name, java.lang.String value)
  установка произвольного заголовка.
- void setIntHeader(java.lang.String name, int value)
  установка целочисленного заголовка.

Двоичные данные

- Пример установки заголовков для отправки клиенту .zip файла

```
String myFile = “file.zip”
int fileSize = 32000;

response.setContentType("application/octet-stream");

 // Content-Disposition  - имя файла при загрузке
response.setHeader("Content-Disposition",
"attachment; filename="" + myFile + "";");

response.setHeader("Cache-Control", "no-cache");
response.setContentLength(fileSize);
```

Двоичные данные

- Собственно чтение и отправка файла

```
ServletOutputStream sos =
response.getOutputStream();
 BufferedInputStream bis = new
BufferedInputStream(new
FileInputStream(“/mydir/myFile.zip”));
int data;
  while((data = bis.read()) != -1) {
       sos.write(data);
   }
  bis.close();
  sos.close();
```

Двоичные данные

- Основное заключение: это обычный ввод/вывод в Java
- Все, что добавляет здесь Servlet API – это дескриптор для вывода, который позволяет отправлять данные клиенту (в поток)
- За поддержку этого вывода отвечает контейнер сервлетов
- Это стандартный объект для вывода
- Он буферизуется. Фактической отправки клиенту не происходит до сброса буферов
- Можно отменять вывод (в части не отправленной еще информации)

Переадресация

- Еще одна форма выдачи результата в сервлете
- Метод, который обрабатывает запрос переадресует его на какой-то другой адрес. Возможно, на другой сервлет, который уже и будет заниматься выдачей результатов
- Технически – это переадресация запросов (3XX коды)
- HttpServletResponse.sendRedirect()
- Это – установка заголовка.

Переадресация

```
//temporary 302 HTTP status code:
// это поведение по умолчанию
response.sendRedirect("http://new.site.com");

// прямая установка заголовка:

response.setStatus(301);
response.setHeader("Location", "http://new.site.com");

// HttpServletResponse содержит константы с мнемоничными
именами, которые соответствуют HTTP статусам:

response.setStatus(HttpServletResponse.SC_MOVED_PERMANENTLY);
response.setHeader("Location", " http://new.site.com ");
```

HTTP сессии

- Объект (структура данных) на сервере, поддерживаемая контейнером
- Объект привязан к уникальному идентификатору, который контейнер присваивает каждой новой HTTP сессии (каждой новой последовательности запросов)
- Этот идентификатор размещается в заголовках Cookie или добавляется к URL (URL rewriting)
- *HttpSession* – класс для хранения данных сессии. Сессия характеризуется идентификатором сессии. Для каждого идентификатора создается экземпляр класса *HttpSession*
- Основное назначение – хранение данных, которые необходимо привязать (ассоциировать) с конкретной сессией
- Хеш-таблица (база ключ-значение)
- Ключ – некоторый идентификатор для поиска данных, значение – произвольный объект

Аннотирование сервлетов

- Наряду с описанием сервлетов в файле web.xml, начиная с версии Servlets API 3.0 есть возможность аннотировать непосредственно Java код.
- Контейнер сервлетов будет обрабатывать такие аннотации непосредственно извлекая их из файлов классов.
- Это замена web.xml, призванная упростить эксплуатацию (просто сохранить .class файл на сервере, не обновляя дескриптор приложения)

Аннотации в Java

- **Java**-аннотация — в языке **Java** специальная форма синтаксических метаданных, которая может быть добавлена непосредственно в исходный **код**.
- Аннотации используются для анализа **кода**, компиляции или выполнения.
- Аннотируемы пакеты, классы, методы, переменные и параметры
- символ @ в начале имени. Например: @Override

Примеры аннотирования /1

- @Override – стандартная аннотация Java, которая предупреждает, что ниже мы что-то переопределим:

- class SomeClass {

```
 void someMethod() {
        System.out.println("Работает метод родительского класса.");
   }
}
```

Примеры аннотирования / 2

- class ChildClass extends SomeClass { // наследуем методы SomeClass в новом классе

```
@Override
void someMethod() { // переопределяем метод
System.out.println("Работает метод класса-потомка.");
   }
}
```

- Аннотация *Override* не дает совершить ошибку в имени метода *someMethod*. Это обязательно должен быть метод из родительского класса

Что позволяют аннотации

- Автоматически создавать конфигурационные XML-файлы
- Автоматически создавать дополнительный Java-код на основе исходного аннотированного кода
- Документировать приложения параллельно с их разработкой
- Быстрее обрабатывать зависимости в программных компонентах;
- Выявлять семантические ошибки, незаметные компилятору

Аннотации сервлетов

- Показана часть из существующих:
- @WebServlet – декларация сервлета
- @WebInitParam – задание начальных параметров сервлета

Аннотации сервлетов

```
@WebServlet(value = "/Simple", initParams = {
   @WebInitParam(name = "foo", value = "Hello "),
   @WebInitParam(name = "bar", value = " World!")
})
public class Simple extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {
     …
  }
}
```

Аннотации сервлетов

Указанная конструкция семантически эквивалентна заданию отображения (mapping) для сервлета

/Simple

И двух начальных параметров:

foo со значением “Hello”

и

bar со значением “World!"