Глава 18
СЕРВЛЕТЫ

Сервлет применяется для создания серверного приложения, получающего от клиента запрос, анализирующего его, делающего выборку данных из базы, а затем пересылающего клиенту страницу HTML, сгенерированную с помощью JSP на основе полученных данных.

Преимуществом сервлетов перед CGI или ASP является быстродействие, переносимость на различные платформы, использование объектно-ориентированного языка высокого уровня Java, который расширяется большим числом классов и программных интерфейсов.

Сервлеты поддерживаются большинством Web-серверов и являются частью платформы J2EE. Сервлеты реализуют интерфейс **`Servlet`**, в котором, кроме рассмотренных выше методов **`service()`**, **`init()`**, **`destroy()`**, предусмотрена реализация еще двух методов:

`ServletConfig getServletConfig()` – возвращает объект, содержащий параметры конфигурации сервлета;

`String getServletInfo()` – определение информации о назначении сервлета.

Интерфейс ServletContext

Интерфейс **`ServletContext`** объявляет методы, которые сервлет применяет для связи с контейнером сервлетов и позволяет получать информацию о среде выполнения, а также использовать ресурсы совместно с другими объектами приложения. Каждому сервлету ставится в соответствие единственный объект, реализующий **`ServletContext`**. Контекст выполнения сервлета дает средства для общения с сервером. В частности, можно получить информацию о MIME-типе файла, добавить/удалить атрибуты контекста или записать информацию в log-файл. Получить ссылку на объект **`ServletContext`** можно вызовом метода **`getServletContext()`**.

Следующие методы позволяют получить из контекста сервлета базовую информацию:

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

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

`String getServerInfo()` – предоставляет информацию о самом сервере.

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

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

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

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

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

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

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

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

Интерфейс ServletConfig

Ранее уже упоминался метод **`getServletConfig()`**, но не было сказано об интерфейсе **`ServletConfig`**, с помощью которого контейнер сервлетов передает информацию сервлету в процессе его инициализации.

Некоторые методы класса:

`String getServletName()` – определение имени сервлета;

`Enumeration getInitParameterNames()` – определение имен параметров инициализации сервлета из дескрипторного файла **`web.xml`**;

`String getInitParameter(String name)` – определение значения конкретного параметра по его имени.

Чтобы задать параметры инициализации сервлета **`MyServlet`**, необходимо в тег <**`servlet`**> его описания вложить тег <**`init-param`**> с описанием имени и значения параметра в виде:

```
<servlet>
  <servlet-name>MyServletname</servlet-name>
  <servlet-class>chapt18.MyServlet</servlet-class>
            <init-param>
                <param-name>mail.smtphost</param-name>
                <param-value>mail.bsu</param-value>
            </init-param>
            <init-param>
                <param-name>mail.smtpport</param-name>
                <param-value>25</param-value>
            </init-param>
</servlet>
```

Тогда для доступа к параметрам инициализации сервлета и их дальнейшего использования можно применить следующую реализацию метода **`init()`** сервлета:

```
public void init() throws ServletException {
    ServletConfig sc = getServletConfig();

      // определение набора имен параметров инициализации
         Enumeration e = sc.getInitParameterNames();
```

```
        while(e.hasMoreElements()) {
    // определение имени параметра инициализации
            String name = (String)e.nextElement();
    // определение значения параметра инициализации
            String value = sc.getInitParameter(name);
            //...
        }
}
```

Такими же возможностями обладает и объект **ServletContext**, который содержит практически всю информацию о среде, в которой запущен и выполняется сервлет, например:

```
getServletContext().getInitParameter("mail.smtpport");
```

Интерфейсы ServletRequest и HttpServletRequest

Информация от компьютера клиента отправляется серверу в виде объекта запроса типа **HttpServletRequest**. Данный интерфейс является производным от интерфейса **ServletRequest**. Используя методы интерфейса **ServletRequest**, можно получить много дополнительной информации, в том числе и о сервлете и деталях протокола HTTP, закодированной и упакованной в запрос:

String getCharacterEncoding() – определение символьной кодировки запроса;

String getContentType() – определение MIME-типа (Multipurpose Internet Mail Extension) пришедшего запроса;

String getProtocol() – определение названия и версии протокола;

String getServerName(), **getServerPort()** – определение имени сервера, принявшего запрос, и порта, на котором запрос был принят сервером соответственно;

String getRemoteAddr(), **getRemoteHost()** – определение IP-адреса клиента, от имени которого пришел запрос, и его имени соответственно;

String getRemoteUser() – определение имени пользователя, выполнившего запрос;

ServletInputStream getInputStream(), **BufferedReader getReader()** – получение ссылки на поток, ассоциированный с содержимым полученного запроса. Первый метод возвращает ссылку на байтовый поток **ServletInputStream**, а второй – на объект **BufferedReader**. В результате можно прочитать любой байт из полученного объекта-запроса. Если метод **getReader()** был вызван после вызова **getInputStream()** для этого запроса, то генерируется исключение **IllegalStateException** и наоборот.

При обращении к серверу, как правило, передаются параметры и их значения. Для разбора параметров и извлечения их значений применяются методы:

String getParameter(String name) – определение значения параметра по его имени или **null**, если параметр с таким именем не задан;

String[] getParameterValues(String name) – определение всех значений параметра по его имени;

`Enumeration getParameterNames()` – определение ссылки на список имен всех параметров через объект типа **`Enumeration`**.

Непосредственно в интерфейсе **`HttpServletRequest`** объявлен ряд методов, позволяющих манипулировать содержимым запросов:

`void setAttribute(String name, Object ob)` – установка значения атрибута компонента, являющегося внутренним параметром для передачи информации между компонентами приложения, например от сервлета к странице JSP или другому сервлету;

`Enumeration getAttributeNames()` – извлечение перечисления имен атрибутов;

`Object getAttribute(String name)` – извлечение значения переданного атрибута по имени;

`Cookie[] getCookies()` – извлечение массива cookie, полученного с запросом. Файл cookie – маленький файл, сохраняемый приложением на стороне клиента;

`String getMethod()` – определение имени метода доступа к ресурсам, на основе которого построен запрос;

`String getQueryString()` – извлечение строки HTTP-запроса.

В следующем примере рассматривается применение некоторых методов интерфейса **`HttpServletRequest`** для получения данных о запросе, посылаемом методом **`GET`** и генерации ответа клиенту.

```
/* пример # 1 : извлечение информации из запроса и счетчик посещений : Request-
Servlet.java : RequestInfo.java : ClickOutput.java */
package chapt18;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

public class RequestServlet extends HttpServlet {
      // счётчик подключений к сервлету
      private int count = 0;

      public void doGet(HttpServletRequest req,
            HttpServletResponse resp)
            throws ServletException {
                  performTask(req, resp);
      }
      public void doPost(HttpServletRequest req,
            HttpServletResponse resp)
            throws ServletException {
                  performTask(req, resp);
      }
      private void performTask(HttpServletRequest req,
            HttpServletResponse resp)
            throws ServletException {
```

```
      try {
            // установка MIME-типа содержания ответа
resp.setContentType("text/html; charset=Windows-1251");

            // поток для данных ответа
            PrintWriter out = resp.getWriter();
            count = ClickOutput.printClick(out, count);
            //обращение к классу бизнес-логики
            if(count == 1)
                  RequestInfo.printToBrowser(out, req);
            // закрытие потока
            out.close();
      } catch (UnsupportedEncodingException e) {
            System.err.print("UnsupportedEncoding");
      } catch (IOException e) {
            System.err.print("IOErroR");
      }
      }
}
package chapt18;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.http.HttpServletRequest;

public class RequestInfo {
      static String br = "<br>";

      public static void printToBrowser(
PrintWriter out, HttpServletRequest req) {
out.println("Method: " + req.getMethod());
out.print(br + "Request URI: " + req.getRequestURI());
out.print(br + "Protocol: " + req.getProtocol());
out.print(br + "PathInfo: " + req.getPathInfo());
out.print(br + "Remote Address: " + req.getRemoteAddr());
                  // извлечение имен заголовочной информации запроса
Enumeration e = req.getHeaderNames();
out.print(br + "Header INFO: ");
            while (e.hasMoreElements()) {
                  String name = (String) e.nextElement();
                  String value = req.getHeader(name);
                  out.print(br + name + " = " + value);
                  }
      }
}
package chapt18;
import java.io.PrintWriter;

public class ClickOutput {
```

```
      public static int printClick(
                  PrintWriter out, int count) {

      out.print(++count + " -е обращение." + "<br>");
      return count;
            }
}
```

Приведенный выше сервлет вызывается нажатием кнопки "Выполнить" формы из документа **index.jsp** по адресу URL – **RequestServlet**.

```
<!-- пример # 2 :  стартовая страница вызова сервлета : index.jsp-->
<%@ page language="java" contentType="text/html; char-
set=ISO-8859-1"  pageEncoding="ISO-8859-5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional
//EN">
<html><head>
<meta http-equiv="Content-Type" content="text/html; char-
set=ISO-8859-5">
<title>Info about Request </title>
</head><body>
 <FORM action="RequestServlet" method="GET">
      <INPUT type="submit"  value="Выполнить">
 </FORM>
</body></html>
```

В результате выполнения в браузер будет выведено:

1-е обращение.
Method: GET
Request URI: /FirstProject/RequestServlet
Protocol: HTTP/1.1
PathInfo: null
Remote Address: 127.0.0.1
Header INFO:
accept = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
referer = http://localhost:8080/FirstProject/index.jsp
accept-language = ru
content-type = application/x-www-form-urlencoded
accept-encoding = gzip, deflate
user-agent = Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
host = localhost:8080
content-length = 0
connection = Keep-Alive
cache-control = no-cache
cookie = JSESSIONID=91014EB2B2208BCA18AE898424B71CEF

Сервлет **RequestServlet** следует попробовать вызывать с различных компьютеров локальной сети или вызвать несколько раз сервлет из нескольких браузеров, запущенных на одном компьютере.

Когда клиент переходит по адресу URL, который обрабатывается сервлетом, контейнер сервлета перехватывает запрос и вызывает метод **doGet()** или **doPost()**. Эти методы вызываются после конфигурации объектов, наследующих интерфейсы **HttpServletRequest**, **HttpServletResponse**. Задача методов **doGet()** и **doPost()** – взаимодействие с HTTP-запросом клиента и создание HTTP-ответа, основанного на данных запроса. Метод **getWriter()** объекта-ответа возвращает поток **PrintWriter**, который используется для записи символьных данных ответа.

Интерфейсы ServletResponse и HttpServletResponse

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

Можно получить ссылки на потоки вывода одним из двух методов:

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

PrintWriter getWriter() – извлечение ссылки на поток типа **PrintWriter** для передачи символьной информации;

Если метод **getOutputStream()** уже был вызван для этого ответа, то генерируется **IllegalStateException**. Обратное также верно.

В интерфейсе **HttpServletResponse**, наследующем интерфейс **ServletResponse**, есть еще несколько полезных методов:

void setContentType(String type) – установка MIME-типа генерируемых документов;

void addCookie(Cookie c) – добавление cookie к объекту ответа для последующей пересылки на клиентский компьютер;

void sendError(int sc, String msg) – сообщение о возникших ошибках, где **sc** – код ошибки, **msg** – текстовое сообщение;

void setDateHeader(String name, long date) – добавление даты в заголовок ответа;

void setHeader(String name, String value) – добавление параметров в заголовок ответа. Если параметр с таким именем уже существует, то он будет заменен.

Обработка запроса

Распределенное приложение может быть эффективным только в случае, если оно способно принимать информацию от физически удаленных клиентов. В следующем примере сервлет извлекает данные пользовательской формы, переданные вместе с запросом по методу **GET**.

Приведенная на рисунке 18.1 форма является результатом отображения JSP-страницы **index.jsp**, находящейся в папке **/webapps/FirstProject3**.

В форме имеется текстовое поле с именем **name** и значением по умолчанию «**Название проекта**». Значение поля можно изменить непосредственно на странице.

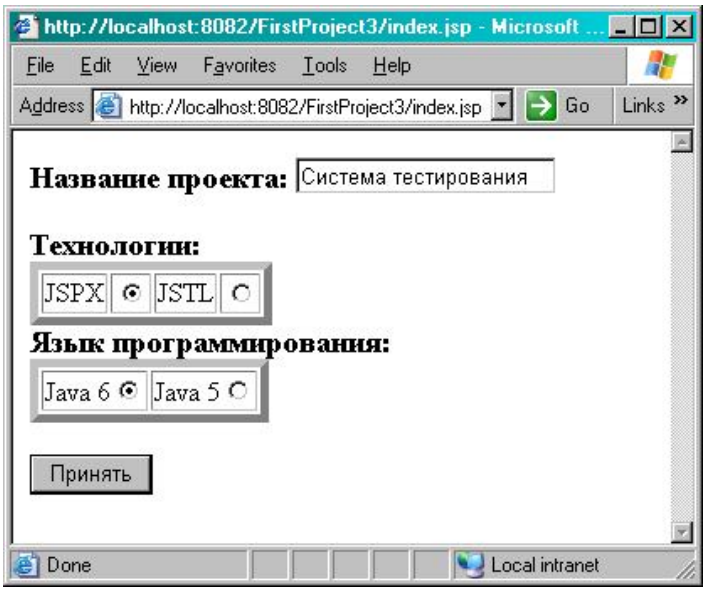


Рис. 18.1. JSP-форма

В форме заданы две группы по два элемента ввода типа **radio**, каждая из которых имеет свое имя. При наличии на странице нескольких полей, имеющих одно имя, можно выбрать только один из них. Им задаются соответствующие значения, и при выборе одной из кнопок значение, заданное соответствующей кнопке, заносится в значение своего элемента. По умолчанию для радиогрупп принято задавать одно из значений при помощи свойства **checked**.

В итоге пользователь может изменить значения текстового поля и радиогрупп. При нажатии кнопки типа происходит подтверждение формы и вызывается сервлет.

В форме задан метод **POST**, при помощи которого происходит передача данных формы в виде отдельных заголовков. Если не задавать этот метод, то по умолчанию будет использоваться метод **GET**, и данные формы будут передаваться через универсальный запрос (URL), в который к адресу будут добавлены значения соответствующих элементов.

```
<!-- пример # 3 :  стартовая страница : index.jsp-->
<%@ page language="java" contentType=
      "text/html; charset=utf-8" pageEncoding="utf-8"%>
<html><body>
<FORM action="testform" method=POST>
<H3>Название проекта:
<INPUT type="text" name="Имя проекта" value="-задать!-">
Технологии:
<TABLE BORDER=5> <tr>
<td>JSPX</td><td><INPUT type="radio"
            name="Технология"
            value="JSP в формате XML"></td>
<td>JSTL</td><td><INPUT type="radio"
```

```
            name="Технология"
            value="Библиотека тегов JSTL"></td>
            </tr></TABLE>
Язык программирования:
<TABLE BORDER=5> <tr>
 <td>Java 6<INPUT type="radio"
            name="Язык"
            value="Java SE 6"></td>
<td>Java 5<INPUT type="radio"
            name="Язык"
            value="Java 1.5.0" checked></td>
</tr></TABLE></H3>
      <INPUT type="submit" value="Принять"> <BR>
</FORM>
</body></html>
```

При подтверждении из формы вызывается сервлет **FormRequest**. Сервлет получает и извлекает значения всех переменных формы и отображает их вместе с именами переменных. Для обработки данных, полученных из полей формы, используется приведенный ниже сервлет.

```
/* пример # 4 : обработка запроса клиента : FormRequest.java :
ParameterOutput.java */
package chapt18;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FormRequest extends HttpServlet {
      protected void doPost(HttpServletRequest req,
                        HttpServletResponse resp)
                        throws ServletException {
            performTask(req, resp);
      }
      private void performTask(HttpServletRequest req,
                              HttpServletResponse resp) {
            RequestOutput.generate(resp, req);
      }
}
```

В методе **performTask()** происходит обращение к другому классу-обработчику запроса пользователя с передачей ему объектов **HttpServletRequest req** и **HttpServletResponse resp**.

```
/* пример # 5 : извлечение информации из запроса клиента : RequestOutput.java */
package chapt18;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class RequestOutput {
      public static void generate(HttpServletResponse resp,
                                  HttpServletRequest req) {
            try {
                  String name, value;
resp.setContentType("text/html; charset=utf-8");
PrintWriter out = resp.getWriter();

      out.print("<HTML><HEAD>");
      out.print("<TITLE>Результат</TITLE>");
      out.print("</HEAD><BODY>");
      out.print("<TABLE BORDER=3>");
      Enumeration names = req.getParameterNames();
            while (names.hasMoreElements()) {
                  name = (String) names.nextElement();
                  value = req.getParameterValues(name)[0];
      /*
name = new String(name.getBytes("ISO-8859-1"), "utf-8");
value = new String(value.getBytes("ISO-8859-1"), "utf-8");
       */
                  out.print("<TR>");
                  out.print("<TD>" + name + "</TD>");
                  out.print("<TD>" + value + "</TD>");
                  out.print("</TR>");
                  }
                  out.print("</TABLE></BODY></HTML>");
                  out.close();
            } catch (IOException e) {
                  e.printStackTrace();
            }
      }
}
```

В классе в объекте **resp** задается тип содержимого **text/html** и кодировка **UTF-8,** если нужно отобразить кириллицу. После этого объект **out** устанавливается в выходной поток **resp.getWriter()**, в который будут помещаться данные. Из запроса **HttpServletRequest req** извлекается объект типа **Enumeration** с текстовыми значениями имен переменных формы. Далее, итерируя по элементам этого объекта, последовательно извлекаются все параметры. Для каждого имени переменной можно при необходимости (если не указана кодовая страница) произвести перекодировку: вначале извлекается объект итерации в кодировке, в которой он передается, а именно **ISO-8859-1**, после создается новая строка с необходимой кодировкой, в данном случае **UTF-8**. Для каждой из переменных извлекаются из запроса соответствующие им значения при помощи метода **getParameterValues(name)**. Тем же способом их кодировка может быть изменена и добавлена в выходной поток.

Класс сервлета относится к пакету **chapt18**, поэтому файл **FormRequest.class** должен быть размещен в папке

/webapps/FirstProject3/WEB-INF/classes/chapt18

и обращение к этому классу, например из документа HTML, должно производиться как **chapt18.FormRequest**. В файле **web.xml** должны находиться строки:

```
<servlet>
    <servlet-name>MyForm</servlet-name>
    <servlet-class>chapt18.FormRequest</servlet-class>
</servlet>
  <servlet-mapping>
    <servlet-name>MyForm</servlet-name>
    <url-pattern>/testform</url-pattern>
  </servlet-mapping>
```

Обращение к сервлету производится по его URL-имени **testform**. Результат выполнения:

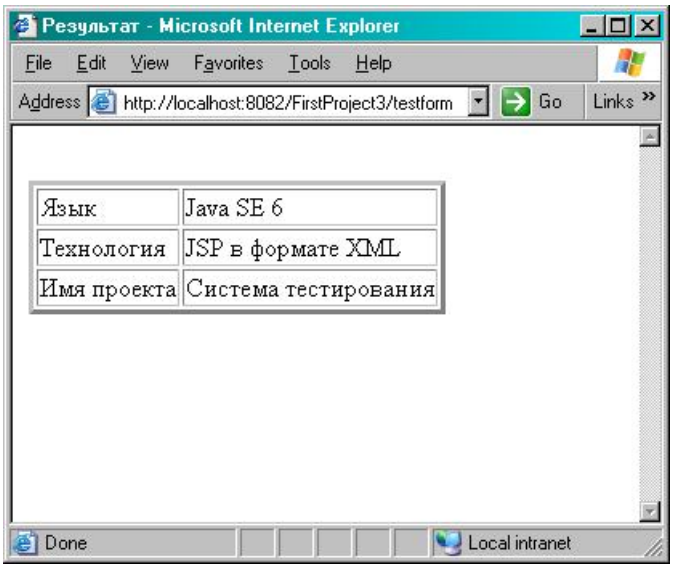


Рис. 18.2. Результат выполнения запроса

Метод **getParameterValues()** возвращает значения любой переменной формы по имени этой переменной. Массив возвращается потому, что некоторые переменные формы могут иметь несколько значений, например группа флажков или радиокнопок. Другой метод доступа не предполагает предварительного знания их имен. Метод **getParameterNames()** возвращает объект **Enumeration**, в котором содержатся все имена переменных, извлеченных из формы.

Многопоточность

Контейнер сервлетов будет иметь несколько потоков выполнения, распределяемых согласно запросам клиентов. Вероятна ситуация, когда два клиента одно-

временно вызовут методы **doGet()** или **doPost()**. Метод **service()** должен быть написан с учетом вопросов многопоточности. Любой доступ к разделяемым ресурсам, которыми могут быть файлы, объекты, необходимо защитить ключевым словом **synchronized**. Ниже приведен пример посимвольного вывода строки сервлетом с паузой между выводом символов в 500 миллисекунд, что позволяет другим клиентам, вызвавшим сервлет, успеть вклиниться в процесс вывода при отсутствии синхронизации.

```
/* пример # 6 : доступ к синхронизированным ресурсам :
ServletSynchronization.java */
package chapt18;
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;
public class ServletSynchronization extends HttpServlet {
    // синхронизируемый объект
  private StringBuffer locked = new StringBuffer();

  protected void doGet(HttpServletRequest req,
                       HttpServletResponse res)
                throws ServletException, IOException {
                performTask(req, res);
  }
  private void performTask(HttpServletRequest req,
                       HttpServletResponse res)
                throws ServletException, IOException {
    try {
      Writer out = res.getWriter();
      out.write(
        "<HTML><HEAD>"
        + "<TITLE>SynchronizationDemo</TITLE>"
        + "</HEAD><BODY>");
      out.write(createString());
      out.write("</BODY></HTML>");
      out.flush();
      out.close();
    } catch (IOException e) {
      throw new RuntimeException(
        "Failed to handle request: " + e);
    }
  }
  protected String createString() {
      // оригинал строки
      final String SYNCHRO = "SYNCHRONIZATION";

    synchronized (locked) {
      try {
        for (int i = 0; i < SYNCHRO.length(); i++) {
          locked.append(SYNCHRO.charAt(i));
          Thread.sleep(500);
        }
```

```
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
        String result = locked.toString();
        locked.delete(0, SYNCHRO.length() - 1);
        return result;
      }
    }
}
```

Результаты работы сервлета при наличии и отсутствии синхронизации представлены на рисунках.

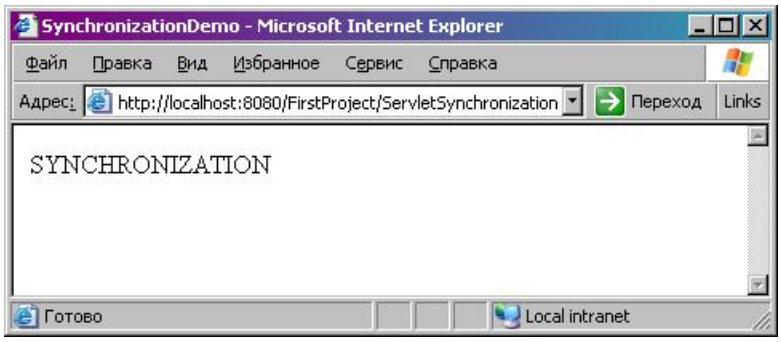
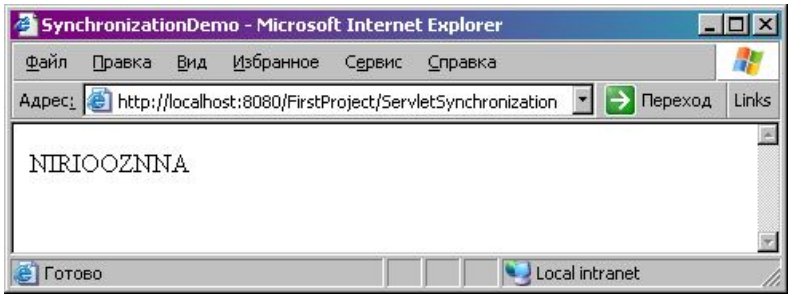


Рис. 18.3. Результат работы сервлета **Synchronization** с блоком синхронизации



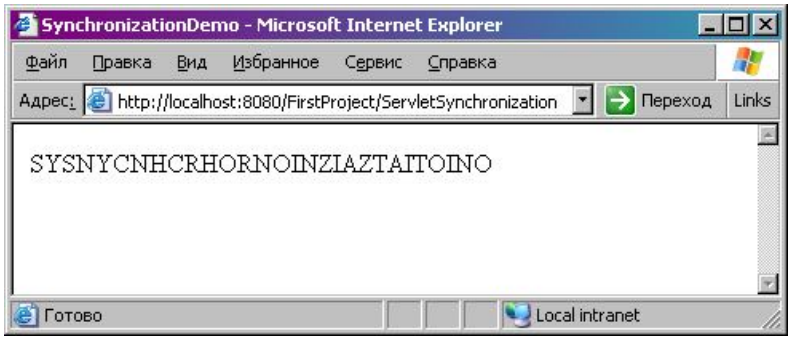


Рис. 18.4. Результат работы сервлета **Synchronization** без синхронизации

Можно синхронизировать и весь сервлет целиком, но причиной, почему это не делается, является возможность нахождения критической секции вне основного пути выполнения программы.

Электронная почта

Рассылка электронной почты, в том числе и автоматическая, является стандартным родом деятельности при использовании Web-приложений. Собственный почтовый сервер создать достаточно легко, только необходимо указать адрес почтового севера, который будет использован в качестве транспорта.

Следующий пример использует интерфейсы API JavaMail для работы с электронной почтой в сервлетах и JSP. API JavaMail содержит классы, с помощью которых моделируется система электронной почты. Класс **`javax.mail.Session`** представляет сеанс почтовой связи, класс **`javax.mail.Message`** – почтовое сообщение, класс **`javax.mail.internet.InternetAddress`** – адреса электронной почты.

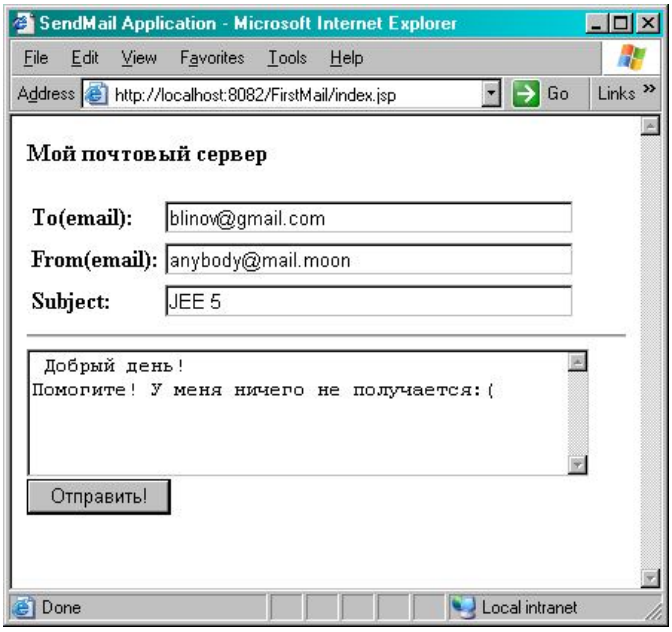


Рис. 18.5. Формирование запроса на отправку письма

Для работы с данной частью платформы J2EE необходимо скачать zip-файлы, расположенные по адресу **`http://java.sun.com/products/javamail/`**, содержащие архивы **`mail.jar`** и **`activation.jar`**. И добавить эти файлы в каталог jar-файлов серверам приложений (**`common/lib`** для Tomcat). Также необходимо запустить почтовую программу James, являющуюся также одним из проектов **`apache.org`**.

Ниже приведена страница JSP, содержащая форму для заполнения основных полей: «**Кому**» – «**to**», «**От кого**» – «**from**», «**Тема сообщения**» – «**subj**»

```
<!-- пример # 7 : страница создания электронного письма : index.jsp-->
<%@ page language="java" contentType=
```

```
"text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html><head><title>SendMail Application</title></head>
      <b>Мой почтовый сервер</b>
      <form method="post" action="sendmail">
      <table>
            <tr><td><b>To(email):</b></td><td>
<input name="to" type="text" size=40></td></tr>
            <tr><td><b>From(email):</b></td><td>
<input name="from" type="text" size=40></td></tr>
            <tr><td><b>Subject:</b></td><td>
<input name="subj" type="text" size=40></td></tr>
      </table>
            <hr>
<textarea name="body" type="text" rows=5 cols=45>
Добрый день!</textarea>
            <br>
            <input type="submit" value="Отправить!">
      </form>
</body></html>
```

Параллельные процессы по отправке письма и предложению пользователю в это же самое время создать новое письмо организуются с применением потока в следующем сервлете.

```
/* пример # 8 : доступ к синхронизированным ресурсам :
SendMailServlet.java */
package chapt18;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.activation.*;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

public class SendMailServlet
      extends javax.servlet.http.HttpServlet
            implements javax.servlet.Servlet {
      //объект почтовой сессии
       private Session mailSession = null;

            public void init(ServletConfig config)
                  throws ServletException {
            //mailSession = Session.getDefaultInstance(System.getProperties());
```

```
            final String host = "mail.smtphost";
            final String port = "mail.smtpport";
      //запрос параметров почтового сервера из web.xml
      String hostvalue = config.getInitParameter(host);
      String portvalue = config.getInitParameter(port);
java.util.Properties props = new java.util.Properties();
      //загрузка параметров почтового сервера в объект свойств
        props.put(host, hostvalue);
        props.put(port, portvalue);
      //загрузка параметров почтового сервера в объект почтовой сессии
  mailSession = Session.getDefaultInstance(props, null);
      }
      protected void doPost(HttpServletRequest request,
                        HttpServletResponse response)
            throws ServletException, IOException {
      //извлечение параметров письма из запроса
      String from = request.getParameter("from");
      String to = request.getParameter("to");
      String subject = request.getParameter("subj");
      String content = request.getParameter("body");
            if((from == null) || (to == null)
      || (subject == null) || (content == null)) {
      /*при отсутствии одного из параметров предлагается повторить
                                                        ввод*/
                  response.sendRedirect("index.jsp");
                  return;
            }
//запуск процесса отправки почты в отдельном потоке
(new MailSender(to, from, subject, content)).start();
//формирование страницы с предложение о создании нового письма
response.setContentType("text/html; charset=CP1251");
PrintWriter out = response.getWriter();
out.println("<html><head><title>");
out.println("SendMail Application</title></head>");
out.println("Message to <b>" + to
            + "</b> sending in progress");
out.println("<a href = \"index.jsp\">New message</a>");
out.println("</body></html>");
      }
      private class MailSender extends Thread {
            private String mailTo;
            private String mailFrom;
            private String mailSubject;
            private String mailContent;
          MailSender(String mailTo, String mailFrom,
            String mailSubject, String mailContent) {
            setDaemon(true);
            this.mailTo = mailTo;
```

```
                this.mailFrom = mailFrom;
                this.mailSubject = mailSubject;
                this.mailContent = mailContent;
                }
                public void run() {
                      try {
          //создание объекта почтового сообщения
          Message message = new MimeMessage(mailSession);
          //загрузка параметров в объект почтового сообщения
          message.setFrom(new InternetAddress(mailFrom));
          message.setRecipient(Message.RecipientType.TO,
                      new InternetAddress(mailTo));
          message.setSubject(mailSubject);
          message.setContent(mailContent, "text/plain");
          //отправка почтового сообщения
                Transport.send(message);
                      } catch(AddressException e) {
                            e.printStackTrace();
                          System.err.print("Ошибка адреса");
                      } catch(MessagingException e) {
                            e.printStackTrace();
                          System.out.print("Ошибка сообщения");
                      }
                }
          }
    }
```

В результате в браузер будет выведено:

Message to blinov@gmail.com sending in progress New message

где **New message** представляет собой активную ссылку, перенаправляющую при запуске на **index.jsp** для создания еще одного письма. Процесс же отправки письма будет функционировать независимо от дальнейшей работы приложения.

Файл **web.xml** для данного приложения имеет вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
      <display-name>FirstMail</display-name>
      <servlet>
            <display-name>
            SendMailServlet</display-name>
            <servlet-name>SendMailServlet</servlet-name>
            <servlet-class>
            SendMailServlet</servlet-class>
            <init-param>
                  <param-name>mail.smtphost</param-name>
```

```
                <param-value>mail.bsu.bsu</param-value>
            </init-param>
            <init-param>
                <param-name>mail.smtpport</param-name>
                <param-value>25</param-value>
            </init-param>
        </servlet>
        <servlet-mapping>
         <servlet-name>chapt18.SendMailServlet</servlet-name>
         <url-pattern>/sendmail</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
            <welcome-file>index.jsp</welcome-file>
        </welcome-file-list>
</web-app>
```

В качестве значения параметра **`mail.smtphost`** можно попробовать использовать адрес почтового сервера **`mail.attbi.com`**.

Задания к главе 18

Вариант А

Создать сервлет и взаимодействующие с ним пакеты Java-классов и JSP-страницы, выполняющие следующие действия:

1. Генерация таблиц по переданным параметрам: заголовок, количество строк и столбцов, цвет фона.
2. Вычисление тригонометрических функций в градусах и радианах с указанной точностью. Выбор функций должен осуществляться через выпадающий список.
3. Поиск слова, введенного пользователем. Поиск и определение частоты встречаемости осуществляется в текстовом файле, расположенном на сервере.
4. Вычисление объемов тел (параллелепипед, куб, сфера, тетраэдр, тор, шар, эллипсоид и т.д.) с точностью и параметрами, указываемыми пользователем.
5. Поиск и (или) замена информации в коллекции по ключу (значению).
6. Выбор текстового файла из архива файлов по разделам (поэзия, проза, фантастика и т.д.) и его отображение.
7. Выбор изображения по тематике (природа, автомобили, дети и т.д.) и его отображение.
8. Информация о среднесуточной температуре воздуха за месяц задана в виде списка, хранящегося в файле. Определить:
   а) среднемесячную температуру воздуха; б) количество дней, когда температура была выше среднемесячной; в) количество дней, когда температура опускалась ниже 0ºС; г) три самых теплых дня.
9. Игра с сервером в “21”.
10. Реализация адаптивного теста из цепочки в 3–4 вопроса.

11. Определение значения полинома в заданной точке. Степень полинома и его коэффициенты вводятся пользователем.
12. Вывод фрагментов текстов шрифтами различного размера. Размер шрифта и количество строк задаются на стороне клиента.
13. Информация о точках на плоскости хранится в файле. Выбрать все точки, наиболее приближенные к заданной прямой. Параметры прямой и максимальное расстояние от точки до прямой вводятся на стороне клиента.
14. Осуществить сортировку введенного пользователем массива целых чисел. Числа вводятся через запятую.
15. Реализовать игру с сервером в крестики-нолики.
16. Осуществить форматирование выбранного пользователем текстового файла, так чтобы все абзацы имели отступ ровно 3 пробела, а длина каждой строки была ровно 80 символов и не имела начальными и конечными символами пробел.

Вариант В

Для заданий варианта В главы 4 на основе сервлетов разработать механизм аутентификации и авторизации пользователя. Сервлет должен сгенерировать приветствие с указанием имени, роли пользователя, а также указать текущую дату и IP-адрес компьютера пользователя.

Тестовые задания к главе 18

Вопрос 18.1.

Каким образом в методе **`init()`** сервлета получить параметр инициализации сервлета с именем “URL”? (выберите два)

1) `ServletConfig.getInitParameter("URL");`
2) `getServletConfig().getInitParameter("URL");`
3) `this.getInitParameter("URL");`
4) `HttpSerlet.getInitParameter("URL");`
5) `ServletContext.getInitParameter("URL").`

Вопрос 18.2.

Какой метод сервлета **`FirstServlet`** будет вызван при активизации ссылки следующего HTML-документа?

```
<html><body>
  <a href="/FirstProject/FirstServlettest">OK!</a>
</body><html>
```

Соответствующий сервлету тег **`<url-pattern>`** в файле **`web.xml`** имеет вид:

```
<url-pattern>/FirstServlettest</url-pattern>
```

1) `doGet();`
2) `doGET();`
3) `performTask();`
4) `doPost();`
5) `doPOST().`

Вопрос 18.3.

Контейнер вызывает метод **`init()`** экземпляра сервлета…

1) при каждом запросе к сервлету;
2) при каждом запросе к сервлету, при котором создается новая сессия;
3) при каждом запросе к сервлету, при котором создается новый поток;
4) только один раз за жизненный цикл экземпляра;
5) когда сервлет создается впервые;
6) если время жизни сессии пользователя, от которого пришел запрос, истекло.

Вопрос 18.4.

Каковы типы возвращаемых значений методов **`getResource()`** и **`getResourceAsStream()`** интерфейса **`ServletContext`**?

1) `ServletContext` не имеет таких методов;
2) `String` и `InputStream`;
3) `URL` и `InputStream`;
4) `URL` и `StreamReader`.

Вопрос 18.5.

Какие интерфейсы находятся в пакете **`javax.servlet`**?

1) `ServletRequest`;
2) `ServletOutputStream`;
3) `PageContext`;
4) `Servlet`;
5) `ServletContextEvent`;
6) ни один из перечисленных.

Вопрос 18.6.

Как можно получить всю информацию из запроса, посланного следующей формой? (выберите два варианта ответа)

```
<HTML><BODY>
  <FORM action="/com/MyServlet">
    <INPUT type="file" name="filename">
    <INPUT type="submit" value="Submit">
  </FORM></BODY></HTML>
```

1) `request.getParameterValues("filename");`
2) `request.getAttribute("filename");`
3) `request.getInputStream();`
4) `request.getReader();`
5) `request.getFileInputStream()`.