Глава 21
СЕССИИ, СОБЫТИЯ И ФИЛЬТРЫ

Сеанс (сессия)

При посещении клиентом Web-ресурса и выполнении вариантов запросов, контекстная информация о клиенте не хранится. В протоколе HTTP нет возможностей для сохранения и изменения информации о предыдущих посещениях клиента. При этом возникают проблемы в распределенных системах с различными уровнями доступа для разных пользователей. Действия, которые может делать администратор системы, не может выполнять гость. В данном случае необходима проверка прав пользователя при переходе с одной страницы на другую. В иных случаях необходима информация о предыдущих запросах клиента. Существует несколько способов хранения текущей информации о клиенте или о нескольких соединениях клиента с сервером.

Сеанс (сессия) – соединение между клиентом и сервером, устанавливаемое на определенное время, за которое клиент может отправить на сервер сколько угодно запросов. Сеанс устанавливается непосредственно между клиентом и Web-сервером. Каждый клиент устанавливает с сервером свой собственный сеанс.

Сеансы используются для обеспечения хранения данных во время нескольких запросов Web-страницы или на обработку информации, введенной в пользовательскую форму в результате нескольких HTTP-соединений (например, клиент совершает несколько покупок в интернет-магазине; студент отвечает на несколько тестов в системе дистанционного обучения). Как правило, при работе с сессией возникают следующие проблемы:

- поддержка распределенной сессии (синхронизация/репликация данных, уникальность идентификаторов и т.д.);
- обеспечение безопасности;
- проблема инвалидации сессии (expiration), предупреждение пользователя об уничтожении сессии и возможность ее продления (watchdog).

Чтобы открыть новый сеанс, используется метод **`getSession()`** интерфейса **`HttpServletRequest`**. Метод извлекает из переданного в сервлет запроса объект сессии класса **`HttpSession`**, соответствующий данному пользователю. Сессия содержит информацию о дате и времени создания последнего обращения к сессии, которая может быть извлечена с помощью методов **`getCreationTime()`** и **`getLastAccessedTime()`**.

Если для метода **`getSession(boolean param)`** входной параметр равен **`true`**, то сервлет-контейнер проверяет наличие активного сеанса, установленного с данным клиентом. В случае успеха метод возвращает дескриптор этого сеанса. В противном случае метод устанавливает новый сеанс:

```
HttpSession se = request.getSession(true);
```

после чего начинается сбор информации о клиенте.

Чтобы сохранить значения переменной в текущем сеансе, используется метод **setAttribute()** класса **HttpSession**, прочесть – **getAttribute()**, удалить – **removeAttribute()**. Список имен всех переменных, сохраненных в текущем сеансе, можно получить, используя метод **Enumeration getAttributeNames()**, работающий так же, как и соответствующий метод интерфейса **HttpServletRequest**.

Метод **String getId()** возвращает уникальный идентификатор, который получает каждый сеанс при создании. Метод **isNew()** возвращает **false** для уже существующего сеанса и **true** – для только что созданного.

Если требуется сохранить для использования одну из переменных сеанса, представляющего собой целое число, то:

```
se.setAttribute("teacherId", new Integer(71));
```

После этого любой подключившийся к текущему сеансу сервлет сможет прочесть значение переменной **teacherId** следующим образом:

```
Integer testId = (Integer)se.getAttribute("teacherID");
```

Завершить сеанс можно методом **invalidate()**. Сеанс уничтожает все связи с объектами, и данные, сохраненные в старом сеансе, будут потеряны для всех приложений.

/ пример # 1 : добавление информации в сессию : SessionServlet.java */*

```
package chapt21;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SessionServlet extends HttpServlet {
      protected void doGet(
            HttpServletRequest req,
            HttpServletResponse resp)
            throws ServletException {
            performTask(req, resp);
      }
      private void performTask(
            HttpServletRequest req,
            HttpServletResponse resp)
            throws ServletException {
            SessionLogic.printToBrowser(resp, req);
      }
}
```

```
package chapt21;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
```

```
public class SessionLogic {

      public static void printToBrowser(
      HttpServletResponse resp, HttpServletRequest req) {
            try {
/* возвращается ссылка на сессию для текущего пользователя (если сессия еще не
существует, то она при этом создается) */
            HttpSession session = req.getSession(true);

                  PrintWriter out = resp.getWriter();

                  StringBuffer url = req.getRequestURL();
                  session.setAttribute("URL", url);

                  out.write("My session counter: ");
/* количество запросов, которые были сделаны к данному сервлету текущим
пользователем в рамках текущей пользовательской сессии (следует приводить
значение к строковому виду для корректного отображения в результате) */
out.write(String.valueOf(prepareSessionCounter(session)));
                  out.write("<br> Creation Time : "
            + new Date(session.getCreationTime()));
                  out.write("<br> Time of last access : "
            + new Date(session.getLastAccessedTime()));
                  out.write("<br> session ID : "
            + session.getId());
                  out.write("<br> Your URL: " + url);
                  int timeLive = 60 * 30;
                  session.setMaxInactiveInterval(timeLive);
            out.write("<br>Set max inactive interval : "
            + timeLive + "sec");

                  out.flush();
                  out.close();
            } catch (IOException e) {
                  e.printStackTrace();
            throw new RuntimeException("Failed : " + e);
            }
      }
/* увеличивает счетчик обращений к текущему сервлету и кладет его в сессию */
      private static int prepareSessionCounter(
                        HttpSession session) {
            Integer counter =
            (Integer)session.getAttribute("counter");

            if (counter == null) {
               session.setAttribute("counter", 1);
                  return 1;
            } else {
                  counter++;
```

```
            session.setAttribute("counter", counter);
                  return counter;
            }
      }
}
```

В результате в браузер будет выведено:

My session counter: 3
Creation Time : Sun Jan 29 16:02:30 EET 2006
Time of last access : Sun Jan 29 16:02:38 EET 2006
session ID : 314A546CD9270A840E0BDA3286636B20
Your URL: http://localhost:8080/FirstProject/SessionServlet
Set max inactive interval : 1800sec

В качестве данных сеанса выступают: счетчик кликов – объект типа **Integer** и URL запроса, сохраненный в объекте **StringBuffer**. В ответ на пользовательский запрос сервлет **SessionServlet** возвращает страницу HTML, на которой отображаются все атрибуты сессии, время создания и последнего доступа, идентификационный номер сессии и время инвалидации (жизни) сессии. Это время можно задать с помощью тега **session-config** в **web.xml** в виде:

```
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```

где время ожидания задается в минутах.

В следующем примере рассмотрен процесс ликвидации сессии при отсутствии активности за определенный промежуток времени.

```
/* пример # 2 : инвалидация и ликвидация сессии : TimeSessionServlet.java */
package chapt21;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class TimeSessionServlet extends HttpServlet {
      boolean flag = true;

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

            HttpSession session = null;
            if (flag) {
            //создание сессии и установка времени инвалидации
                  session = req.getSession();
```

```
                int timeLive = 10; //десять секунд!
                session.setMaxInactiveInterval(timeLive);
                flag = false;
            } else {
            //если сессия не существует, то ссылка на нее не будет получена
                session = req.getSession(false);
                    }
            TimeSession.go(resp, req, session);
      }
}
package chapt21;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class TimeSession {

      public static void go(HttpServletResponse resp,
HttpServletRequest req, HttpSession session ) {
            PrintWriter out = null;
            try {
                out = resp.getWriter();
                out.write("<br> Creation Time : "
            + new Date(session.getCreationTime()));
                out.write("<br> Session alive! ");

                out.flush();
                out.close();
            } catch (NullPointerException e) {
            //если сессия не существует, то генерируется исключение
                if (out != null)
                out.print("Session disabled!");
            } catch (IOException e) {
                e.printStackTrace();
            throw new RuntimeException("i/o failed: " + e);
            }
      }
   }
```

При первом запуске в браузер будет выведено:
Creation Time : Tue Aug 14 17:54:23 EEST 2007
Session alive!

Если повторить запрос к сервлету менее чем за 10 секунд, вывод будет повторен. Если же запрос повторить более через десять секунд, сессия будет автоматически уничтожена, и в браузер будет выведено следующее:
Session disabled!

Cookie

Для хранения информации на компьютере клиента используются возможности класса **Cookie**.

Cookie – это небольшие блоки текстовой информации, которые сервер посылает клиенту для сохранения в файлах cookies. Клиент может запретить браузеру прием файлов cookies. Браузер возвращает информацию обратно на сервер как часть заголовка HTTP, когда клиент повторно заходит на тот же Web-ресурс. Cookies могут быть ассоциированы не только с сервером, но и также с доменом – в этом случае браузер посылает их на все серверы указанного домена. Этот принцип лежит в основе одного из протоколов обеспечения единой идентификации пользователя (Single Signon), где серверы одного домена обмениваются идентификационными маркерами (token) с помощью общих cookies.

Cookie были созданы в компании Netscape как средства отладки, но теперь используются повсеместно. Файл cookie – это файл небольшого размера для хранения информации, который создается серверным приложением и размещается на компьютере пользователя. Браузеры накладывают ограничения на размер файла cookie и общее количество cookie, которые могут быть установлены на пользовательском компьютере приложениями одного Web-сервера.

Чтобы послать cookie клиенту, сервлет должен создать объект класса **Cookie**, указав конструктору имя и значение блока, и добавить их в объект-response. Конструктор использует имя блока в качестве первого параметра, а его значение – в качестве второго.

```
Cookie cookie = new Cookie("myid", "007");
response.addCookie(cookie);
```

Извлечь информацию cookie из запроса можно с помощью метода **getCookies()** объекта **HttpServletRequest**, который возвращает массив объектов, составляющих этот файл.

```
Cookie[] cookies = request.getCookies();
```

После этого для каждого объекта класса **Cookie** можно вызвать метод **getValue()**, который возвращает строку **String** с содержимым блока cookie. В данном случае этот метод вернет значение "007".

Объект **Cookie** имеет целый ряд параметров: путь, домен, номер версии, время жизни, комментарий. Одним из важнейших является срок жизни в секундах от момента первой отправки клиенту. Если параметр не указан, то cookie существует только до момента первого закрытия браузера. Для запуска следующего приложения можно использовать сервлет из примера # 1 этой главы, вставив в метод **performTask()** следующий код:

```
CookieWork.setCookie(resp); // добавление cookie
CookieWork.printToBrowser(resp, req); // извлечение cookie
```

Класс **CookieWork** имеет вид:

```
/* пример # 3 : создание и чтение cookie : CookieWork.java */
package chapt16;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.http.Cookie;
```

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

public class CookieWork {

public static void setCookie(HttpServletResponse resp) {
            String name = "Spiridonov";
            String role = "MegaAdmin";
            Cookie c = new Cookie(name, role);
            c.setMaxAge(3600);// время жизни файла
            resp.addCookie(c);
      }
public static void printToBrowser(
HttpServletResponse response, HttpServletRequest request) {
            try {
                  response.setContentType("text/html");
                  PrintWriter out = response.getWriter();

                  Cookie[] cookies = request.getCookies();
            if (cookies != null) {
                  out.print("Number cookies :"
                              + cookies.length + "<BR>");

            for (int i = 0; i < cookies.length; i++) {
            Cookie c = cookies[i];
            out.print("Secure :" + c.getSecure() + "<br>");
            out.print(c.getName() + " = " + c.getValue()
                                                + "<br>");
                                          }// end for
            }// end if
                  out.close();
            } catch (IOException e) {
                  e.printStackTrace();
                  throw new RuntimeException(e.toString());
            }
      }
}
```

В результате в файле cookie будет содержаться следующая информация:

Number cookies :1
Secure :false
Spiridonov = MegaAdmin

Файл cookie можно изменять. Для этого следует воспользоваться сервлетом из примера #1 и в метод **performTask()** добавить следующий код:

```
CookieCounter.printToBrowser(resp, req);
```

В классе **CookieCounter** производится модификация файла cookie, хранимого на компьютере клиента.

```
/* пример # 4 : создание cookie и чтение количества вызовов сервлета из cookie :
CookieCounter.java */
package chapt16;
import java.io.IOException;
import java.io.Writer;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CookieCounter {
/* константа, которая будет использована для установки максимального
времени жизни cookie (здесь указано 30 дней) */
public static final int MAX_AGE_COOKIE = 3600 * 24 * 30;

      public static void printToBrowser(
HttpServletResponse response, HttpServletRequest request) {
            try {
                  Writer out = response.getWriter();
                  out.write("My Cookie counter: ");
/* устанавливает счетчик количества вызовов сервлета пользователем    */
out.write(String.valueOf(prepareCookieCounter(
                                    request,response)));
                  out.flush();
                  out.close();
            } catch (IOException e) {
                  e.printStackTrace();
            throw new RuntimeException("Failed: " + e);
            }
      }
      // обновляет в cookie счетчик обращений пользователя к сервлету
      private static int prepareCookieCounter(
HttpServletRequest request, HttpServletResponse response) {
            Cookie[] cookies = request.getCookies();
            Cookie counterCookie;
if (cookies != null) {
   for (int i = 0; i < cookies.length; i++) {
      if ("counter".equals(cookies[i].getName())) {
            String counterStr = cookies[i].getValue();
                  int counterValue;
                  try {
            counterValue = Integer.parseInt(counterStr);
                  } catch (NumberFormatException e) {
                        counterValue = 0;
                              }
      counterValue++;
      counterCookie = new Cookie("counter",
                  String.valueOf(counterValue));
      counterCookie.setMaxAge(MAX_AGE_COOKIE);
      response.addCookie(counterCookie);
```

```
                                return counterValue;
                            }//end if
                        }//end for
                    } //end if
                    counterCookie = new Cookie("counter", "1");
                    counterCookie.setMaxAge(MAX_AGE_COOKIE);
                    response.addCookie(counterCookie);
                    return 1;
            }
        }
```

В результате в файле cookie будет содержаться следующая информация:

```
counter
1
localhost/FirstProject/
1024
939371136
29793584
1067152496
29787549
*
```

В браузер будет выведено следующее сообщение:
My session counter: 1

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.

Интерфейсы и их методы
ServletContextListener **contextDestroyed(ServletContextEvent e)** **contextInitialized(ServletContextEvent e)**
HttpSessionListener **sessionCreated(HttpSessionEvent e)** **sessionDestroyed(HttpSessionEvent e)**
ServletContextAttributeListener **attributeAdded(ServletContextAttributeEvent e)** **attributeRemoved(ServletContextAttributeEvent e)** **attributeReplaced(ServletContextAttributeEvent e)**
HttpSessionAttributeListener **attributeAdded(HttpSessionBindingEvent e)** **attributeRemoved(HttpSessionBindingEvent e)** **attributeReplaced(HttpSessionBindingEvent e)**
HttpSessionBindingListener **valueBound(HttpSessionBindingEvent e)** **valueUnbound(HttpSessionBindingEvent e)**
HttpSessionActivationListener **sessionWillPassivate(HttpSessionEvent e)** **sessionDidActivate(HttpSessionEvent e)**
ServletRequestListener **requestDestroyed(ServletRequestEvent e)** **requestInitialized(ServletRequestEvent e)**
ServletRequestAttributeListener **attributeAdded(ServletRequestAttributeEvent e)** **attributeRemoved(ServletRequestAttributeEvent e)** **attributeReplaced(ServletRequestAttributeEvent e)**

Регистрация блока прослушивания производится в дескрипторном файле приложения.

Демонстрация обработки событий изменения состояния атрибутов сессии будет рассмотрена на примере №1 данной главы. Обрабатываться будут события добавления атрибута **URL**, а также добавления и изменения атрибута счетчика **counter**. Для этого необходимо создать класс, реализующий интерфейс **HttpSessionAttributeListener** и реализовать необходимые для выполнения поставленной задачи методы.

```
/* пример # 5 : обработка событий добавления и изменения атрибута сессии :
MyAttributeListener.java */
package chapt21;
```

```
import javax.servlet.http.HttpSessionAttributeListener;
import javax.servlet.http.HttpSessionBindingEvent;

public class MyAttributeListener
            implements HttpSessionAttributeListener {
     private String counterAttr = "counter";

public void attributeAdded(HttpSessionBindingEvent ev) {
     String currentAttributeName = ev.getName();
     String urlAttr = "URL";

     if (currentAttributeName.equals(counterAttr)) {

        Integer currentValueInt = (Integer) ev.getValue();
        System.out.println("в Session добавлен счетчик="
                           + currentValueInt);
     } else if (currentAttributeName.equals(urlAttr)) {

StringBuffer currentValueStr = (StringBuffer)ev.getValue();
        System.out.println("в Session добавлен URL="
                           + currentValueStr);
      } else System.out.println("добавлен новый атрибут");
}
public void attributeRemoved(HttpSessionBindingEvent ev) {
}
public void attributeReplaced(HttpSessionBindingEvent ev) {
     String currentAttributeName = ev.getName();
     // в случае  изменений, произведенных со счетчиком,
     // выводит соответствующее сообщение
     if (currentAttributeName.equals(counterAttr)) {
        Integer currentValueInt = (Integer) ev.getValue();
        System.out.println("В Session заменен счетчик="
                           + currentValueInt);
     }
  }
}
```

Чтобы события обрабатывались, необходимо включить упоминание об обработчике событий в элемент **<web-app>** дескрипторного файла **web.xml**.

```
<listener>
            <listener-class>chapt21.MyAttributeListener
            </listener-class>
</listener>
```

Тогда в результате запуска сервлета **SessionServlet** и нескольких обращений к нему в консоль будет выведено:

```
в Session добавлен URL=
http://localhost:8080/FirstProject/SessionServlet
в Session добавлен счетчик=1
в Session заменен счетчик=1
в Session заменен счетчик=2
в Session заменен счетчик=3
```

Интерфейс **ServletRequestListener** добавлен в Servlet API начиная с версии 2.4. С его помощью можно отследить события создания запроса при обращении к сервлету и его уничтожении.

```
/* пример # 6 : обработка событий создания и уничтожения запроса к сервлету :
MyRequestListener.java */
package chapt21;
import javax.servlet.ServletContext;
import javax.servlet.ServletRequest;
import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
import javax.servlet.http.HttpServletRequest;

public class MyRequestListener
            implements ServletRequestListener {
      // счетчик числа обращений к сервлету
      private static int reqCount;

    public void requestInitialized(ServletRequestEvent e) {
            //будет использован для доступа к log-файлу
            ServletContext context = e.getServletContext();
            //будет использован для получения информации о запросе
            ServletRequest req = e.getServletRequest();

            synchronized (context) {
                  String name = "";
      name = ((HttpServletRequest) req).getRequestURI();
                  // сохранение значения счетчика в log-файл
            context.log("Request for " + name
                              + "; Count=" + ++reqCount);
            }
      }
      public void requestDestroyed(ServletRequestEvent e) {
            // вызывается при уничтожении запроса
            System.out.println("Request уничтожен");
      }

}
```

В результате запуска сервлета **SessionServlet**, в log-файл, расположенный в каталоге **/logs** контейнера сервлетов, будет выведено:

```
28.02.2006 23:59:53 org.apache.catalina.core.ApplicationContext log
INFO: Request for /FirstProject/SessionServlet; Count=1
Request уничтожен
```

При этом класс «обработчик событий» должен быть зарегистрирован в файле **web.xml** следующим образом:

```
<listener>
      <listener-name>MyRequestListener</listener-name>
<listener-class>chapt21.MyRequestListener</listener-class>
 </listener>
```

Фильтры

Реализация интерфейса **Filter** позволяет создать объект, который может трансформировать заголовок и содержимое запроса клиента или ответа сервера. Фильтры не создают запрос или ответ, а только модифицируют его. Фильтр выполняет предварительную обработку запроса, прежде чем тот попадает в сервлет, с последующей (если необходимо) обработкой ответа, исходящего из сервлета. Фильтр может взаимодействовать с разными типами ресурсов, в частности и с сервлетами, и с JSP-страницами.

Основные действия, которые может выполнить фильтр:

- перехват инициализации сервлета и определение содержания запроса, прежде чем сервлет будет инициализирован;
- блокировка дальнейшего прохождения пары request-response;
- изменение заголовка и данных запроса и ответа;
- взаимодействие с внешними ресурсами;
- построение цепочек фильтров;
- фильтрация более одного сервлета.

При программировании фильтров следует обратить внимание на интерфейсы **Filter**, **FilterChain** и **FilterConfig** из пакета **javax.servlet**. Сам фильтр определяется реализацией интерфейса **Filter**. Основным методом этого интерфейса является метод

void doFilter(ServletRequest req, ServletResponse res, FilterChain chain),

которому передаются объекты запроса, ответа и цепочки фильтров. Он вызывается каждый раз, когда запрос/ответ проходит через список фильтров **FilterChain**. В данный метод помещается реализация задач, обозначенных выше.

Кроме того, необходимо реализовать метод **void init(FilterConfig config)**, который принимает параметры инициализации и настраивает конфигурационный объект фильтра **FilterConfig**. Метод **destroy()** вызывается при завершении работы фильтра, в тело которого помещаются команды освобождения используемых ресурсов.

Жизненный цикл фильтра начинается с однократного вызова метода **init()**, затем контейнер вызывает метод **doFilter()** столько раз, сколько запросов будет сделано непосредственно к данному фильтру. При отключении фильтра вызывается метод **destroy()**.

С помощью метода **doFilter()** каждый фильтр получает текущий запрос и ответ, а также список фильтров **FilterChain**, предназначенных для обработки. Если в **FilterChain** не осталось необработанных фильтров, то продолжается передача запроса/ответа. Затем фильтр вызывает **chain.doFilter()** для передачи управления следующему фильтру.

В следующем примере рассматривается обращение к объекту **request** JSP-страницы **demofilter.jsp** и изменение значения атрибута запроса **testName**.

```
<!--пример # 7 : обращение к атрибуту : demofilter.jsp -->
<%@ page language="java" contentType="text/html;
charset=ISO-8859-5"  pageEncoding="ISO-8859-5"%>
```

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" pre-
fix="c"%>
<html><head><title>Demo Filter</title></head>
<body>
<c:out value="Info from filter: ${info}"/><br>
<P>Дублирование действий фильтра смотреть в консоли</P>
</body></html>
```

Если фильтр не подключать, то переменная **info** значения не получит.

Реализация интерфейса **Filter** для поставленной задачи выглялит следующим образом:

```
/* пример # 8 : простая фильтрация значения атрибута : MyFilter.java */
package chapt21;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyFilter implements Filter {
      private FilterConfig filterConfig;

      public void init(final FilterConfig filterConfig) {
            this.filterConfig = filterConfig;
      }
      public void doFilter(final ServletRequest request,
      final ServletResponse response, FilterChain chain)
                  throws java.io.IOException,
                        javax.servlet.ServletException {
            System.out.println("Вход в фильтр");
            String value = "Simple Filter";

            request.setAttribute("info", value);

            chain.doFilter(request, response);
            System.out.println("info = " + value);
            System.out.println("Окончание фильтра");
      }
      public void destroy() {
            System.out.println("Уничтожение фильтра");
      }
}
```

Чтобы к фильтру происходило обращение, необходимо включить упоминание о фильтре и обрабатываемом ресурсе в элемент **<web-app>** дескрипторного файла **web.xml** в виде:

```
<filter>
    <filter-name>simplefilter</filter-name>
    <filter-class>chapt21.MyFilter</filter-class>
 </filter>
```

```
<filter-mapping>
   <filter-name>simplefilter</filter-name>
   <url-pattern>/demofilter.jsp</url-pattern>
</filter-mapping>
```

Фильтр может модифицировать ответ сервера клиенту. Одним из распространенных приемов использования фильтра является модификация кодировки ответа. Когда сервлет посылает ответ клиенту через поток **PrintWriter**, используется установленная в сервлете кодировка. В следующем примере рассматривается фильтр, изменяющий кодировку ответа на кириллицу UTF-8.

```
// пример # 9 : фильтр, устанавливающий кодировку запроса : SetCharFilter.java
package chapt21;
import java.io.IOException;
import javax.servlet.*;

public class SetCharFilter implements Filter {
      private FilterConfig filterConfig = null;

      public void init(FilterConfig config)
throws ServletException {
            this.filterConfig = config;

      }
public void doFilter(ServletRequest request,
ServletResponse response, FilterChain next)
throws IOException, ServletException {
      // чтение кодировки из запроса
      String encoding = request.getCharacterEncoding();
      System.out.println(encoding);
      // установка UTF-8, если не установлена
      if (!"UTF-8".equalsIgnorCase(encoding))
            response.setCharacterEncoding("UTF-8");
      next.doFilter(request, response);
      }
      public void destroy() {
      }
}
```

И его конфигурации в **web.xml**:

```
<filter>
      <filter-name>setCharFilter</filter-name>
      <filter-class>chapt21.SetCharFilter</filter-class>
</filter>
<filter-mapping>
      <filter-name>setCharFilter</filter-name>
      <url-pattern>/DemoCharServlet</url-pattern>
</filter-mapping>
```

Таким образом, ответ сервлета **DemoCharServlet** будет в необходимой кодировке.

```
/*пример # 10 : без установки кодировки ответ сервлета будет нечитаем :
DemoCharServlet.java*/
package chapt21;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DemoCharServlet extends HttpServlet {
      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("Кодировка установлена успешно!");
      }
      public void destroy() {
            super.destroy();
      }
}
```

В результате в браузер будет выведено:

Кодировка установлена успешно!

Задания к главе 21

Вариант А

Для всех заданий использовать авторизованный вход в приложение. Параметры авторизации, дату входа в приложение и время работы сохранять в сессии.

1. В тексте, хранящемся в файле, определить длину содержащейся в нем максимальной серии символов, отличных от букв. Все такие серии символов с найденной длиной сохранить в cookie.
2. В файле хранится текст. Для каждого из слов, которые вводятся в текстовые поля HTML-документа, вывести в файл cookie, сколько раз они встречаются в тексте.
3. В файле хранится несколько стихотворений, которые разделяются строкой, состоящей из одних звездочек. В каком из стихотворений больше всего восклицательных предложений? Результат сохранить в файле cookie.
4. Записать в файл cookie все вопросительные предложения текста, которые хранятся в текстовом файле.
5. Код программы хранится в файле. Подсчитать количество операторов этой программы и записать результаты поиска в файл cookie, перечислив при этом все найденные операторы.
6. Код программы хранится в файле. Сформировать файл cookie, записи которого дополнительно слева содержат уровень вложенности циклов. Ограничения на входные данные:

а) ключевые слова используются только для обозначения операторов;
б) операторы цикла записываются первыми в строке.

7. Подсчитать, сколько раз в исходном тексте программы, хранящейся на диске, встречается оператор, который вводится с терминала. Сохранить в файле cookie также номера строк, в которых этот оператор записан. Ограничения: ключевые слова используются только для обозначения операторов.
8. Сохранить в cookie информацию, введенную пользователем, и восстановить ее при следующем обращении к странице.
9. Выбрать из текстового файла все числа-полиндромы и их количество. Результат сохранить в файле cookie.
10. В файле хранится текст. Найти три предложения, содержащие наибольшее количество знаков препинания, и сохранить их в файле cookie.
11. Подсчитать количество различных слов в файле и сохранить информацию в файл cookie.
12. В файле хранится код программы. Удалить из текста все комментарии и записать измененный файл в файл cookie.
13. В файле хранится HTML-документ. Проверить его на правильность и записать в файл cookie первую строку и позицию (если они есть), нарушающую правильность документа.
14. В файле хранится HTML-документ. Найти и вывести все незакрытые теги с указанием строки и позиции начала в файл cookie. При выполнении задания учесть возможность присутствия тегов, которые не требуется закрывать. Например:
.
15. В файле хранится HTML-документ с незакрытыми тегами. Закрыть все незакрытые теги так, чтобы документ HTML стал правильным, и записать измененный файл в файл cookie. При выполнении задания учесть возможность присутствия тегов, которые не требуется закрывать. Например:
.
16. В файле хранятся слова русского языка и их эквивалент на английском языке. Осуществить перевод введенного пользователем текста и записать его в файл cookie.
17. Выбрать из файла все адреса электронной почты и сохранить их в файле cookie.
18. Выбрать из файла имена зон (*.by, *.ru и т.д.), вводимые пользователем, и сохранить их в файле cookie.
19. Выбрать из файла все заголовки разделов и подразделов (оглавление) и записать их в файл cookie.
20. При работе приложения сохранять в сессии имена всех файлов, к которым обращался пользователь.

Вариант В

Для заданий варианта В главы 4 каждому пользователю должен быть поставлен в соответствие объект сессии. В файл cookie должна быть занесена информация о времени и дате последнего сеанса пользователя и информация о количестве посещений ресурса и роли пользователя.

Тестовые задания к главе 21

Вопрос 21.1.

Каким образом можно явно удалить объект сессии?

1) нельзя, так как сессия может быть удалена только после истечения времени жизни;
2) вызовом метода **`invalidate()`** объекта сессии;
3) вызовом метода **`remove()`** объекта сессии;
4) вызовом метода **`delete()`** объекта сессии;
5) вызовом метода **`finalize()`** объекта сессии.

Вопрос 21.2.

Какие методы могут быть использованы объектом сессии?

1) `setAttribute(String name, Object value);`
2) `removeAttribute();`
3) `deleteAttribute();`
4) `setValue(String name, Object value);`
5) `getAttributeNames();`
6) `getInactiveTime()`.

Вопрос 21.3.

Каким образом можно получить объект-сеанс из ассоциированного с ним объекта-запроса **`HttpServletRequest req`** ?

1) `HttpSession session = req.getSession();`
2) `HttpSession session = req.createHttpSession();`
3) `Session session = req.createSession();`
4) `Session session = req.getSession();`
5) `HttpSession session = req.getHttpSession();`
6) `HttpSession session = req.createSession();`
7) `HttpSession session = req.getSession(true)`.

Вопрос 21.4.

Какие из следующих утверждений относительно объекта **`Cookie`** являются верными?

1) имя файла передается конструктору в качестве параметра при создании объекта **`Cookie`** и далее не может быть изменено;
2) имя файла может быть изменено с помощью вызова метода **`Cookie.setName(String name);`**
3) значение объекта может быть изменено с помощью вызова метода **`setValue(String value);`**
4) браузер ограничивает размер одного файла cookie до 4096 байт;
5) браузер не ограничивает общее число файлов cookie;
6) максимальное время существования файла cookie в днях устанавливается вызовом метода **`Cookie.setMaxAge(int day)`.**

Вопрос 21.5.

Какие из следующих объявлений объекта класса **`Cookie`** верны?

1) `Cookie c1 = new Cookie ();`
2) `Cookie c2 = new Cookie ("cookie2");`
3) `Cookie c3 = new Cookie ("$cookie3", "value3");`
4) `Cookie c4 = new Cookie ("cookie  4", "value4");`
5) `Cookie c5 = new Cookie ("cookie5", "value5");`
6) `Cookie c6 = new Cookie ("6cookie", "value6");`
7) `Cookie c7 = new Cookie ("c7,8", "value7").`

Вопрос 21.6.

Каким образом файлы cookie присоединяются к объекту-ответу **`HttpServletResponse resp`**?

1) `resp.setCookie(Cookie cookie);`
2) `resp.addCookie(Cookie cookie);`
3) `resp.createCookie(Cookie cookie);`
4) `resp.putCookie(Cookie cookie);`
5) `resp.setCookies(Cookie cookie).`