Федеральное государственное бюджетное образовательное учреждение высшего образования
Казанский государственный энергетический университет
Кафедра информационных технологий и интеллектуальных систем

Задачи на Session

Методические указания для выполнения лабораторных работ

Казань, 2024

Лабораторная работа. Задачи на Session

Создание веб-приложения с использованием сессий, событий и фильтров

1. Цель работы

Получить навыки реализации JSP-страниц с использованием сессий, событий и фильтров.

Теоретический материалСеанс (сессия)

При посещении клиентом веб-ресурса и выполнении вариантов запросов контекстная информация о клиенте не хранится. В протоколе HTTP нет возможностей для сохранения и изменения информации о предыдущих действиях 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 есть сессия между клиентом и сервером, устанавливаемая на определенное время, за которое клиент может отправить на сервер сколько угодно запросов. Сеанс устанавливается непосредственно между клиентом и веб-сервером в момент получения первого запроса к веб-приложению. Каждый клиент устанавливает с сервером свой собственный сеанс, который сохраняется до окончания работы с приложением.

Сеанс используется для обеспечения хранения данных при последовательном выполнении нескольких запросов различных веб-страниц 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 явный доступ к экземпляру сеанса/сессии пользователя веб-приложения, используются методы getSession(boolean create) или getSession() интерфейса HttpServletRequest. Экземпляр запроса возвращает ссылку на объект сессии. Метод не создает сессию, а только дает доступ с помощью запроса к экземпляру сессии HttpSession, соответствующему данному пользователю.

Если для метода getSession(boolean create) 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/создает новый сеанс:

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

после чего начинается сбор информации о клиенте. Если метод getSession(boolean param) вызывать с параметром false, то ссылка на активный сеанс будет возвращена, если он существует, если же сеанс уничтожен или не был активирован, то метод возвратит null. Метод getSession() без параметров работает так же, как и getSession(true).

Сессия содержит информацию о дате и времени создания последнего обращения к сессии, которая может быть извлечена с помощью методов getCreationTime() и getLastAccessedTime().

Метод String getId() возвращает уникальный идентификатор, который получает каждый сеанс при создании. Метод isNew() возвращает false для уже существующего сеанса и true – для нового сеанса. Задать время (в секундах) бездействия сессии, по истечении которого экземпляр сессии будет уничтожен, можно методом setMaxInactiveInterval(long sec).

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

Принудительно завершить сеанс можно методом invalidate(). В результате для сеанса будут уничтожены все связи с используемыми объектами, и данные, сохраненные в старом сеансе, будут потеряны для клиента и всех приложений.

```
/* # 1 # добавление информации в сессию # SessionControlServlet.java */
package by.bsac.control.listener;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;

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

import java.io.IOException;

import
javax.servlet.annotation.WebServlet;
@WebServlet("/sessionservlet")

public class SessionControlServlet extends HttpServlet {

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

HttpSession session = request.getSession(true);
if (session.getAttribute("role") == null) {
session.setAttribute("role", "moderator");

}

/* количество запросов, которые были сделаны к данному сервлету текущим
пользователем в рамках текущей пользовательской сессии */

Integer counter = (Integer) session.getAttribute("counter");
if (counter == null) {

session.setAttribute("counter", 1);

} else {

/* увеличивает счетчик обращений к текущему сервлету и кладет его в
сессию */counter++;

session.setAttribute("counter", counter);

}

request.setAttribute("lifecycle", "CONTROL request LIFECYCLE");
request.getRequestDispatcher("/jsp/sessionattr.jsp").forward(request, response);
```

}

}

Задача страницы sessionattr.jsp – отразить информацию о роли пользователя и счетчике обращений.

 # 2 # информация о сессии # /jsp/sessionattr.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<html><head><title>Session Attribute</title></head>

<body> Role:
${role}

*
<hr />*
Counter: ${counter}

*
<hr />*

*MaxInactiveInterval: ${pageContext.session.maxInactiveInterval}
*
*ID session: ${pageContext.session.id}
*

*Lifecycle: ${lifecycle}
*

Back to index.jsp

</body></html>

В качестве данных сеанса выступают: счетчик обращений – объект типа Integer и роль пользователя. В ответ на пользовательский запрос сервлет SessionServlet возвращает страницу sessionattr.jsp, на которой отображаются все атрибуты сессии, а также идентификационный номер сессии и время инвалидации сессии. Время жизни сессии при отсутствии активности пользователя задано 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 результате в браузер будет выведено (см. рисунок 1):

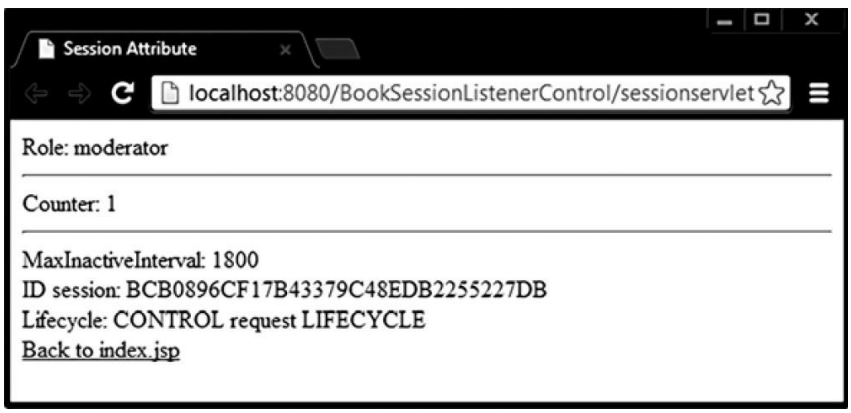


Рисунок 1 - Информация о сессии

 # 3 # стартовая страница # index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<html>

<head><title>Index Page</title></head>

<body>

Lifecycle: ${lifecycle}

<a href= "sessionservlet">Session Servlet</a>

</body></html>
```

В следующем примере произведено моделирование процесса ликвидации сессии при отсутствии активности за определенный промежуток времени. Реализация корректно работает только в случае, если приложением пользуется один клиент. Для нескольких клиентов алгоритм будем немного сложнее.

```
/* # 4 # инвалидация сессии по времени # TimeoutServlet.java
*/package by.bsac.timeoutsession;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;

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

import java.io.IOException;

public class TimeoutServlet extends HttpServlet {

protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

HttpSession session = null;
if (SessionLocator.flag) {

// "создание" сессии и установка времени инвалидации

session = req.getSession();

int timeLive = 10; // десять секунд!
session.setMaxInactiveInterval(timeLive);
SessionLocator.flag = false;

} else {

// если сессия не существует, то ссылка на нее не будет получена

session = req.getSession(false);
if (session == null) {
SessionLocator.flag = true;

}

}

req.setAttribute("messages", SessionLocator.addMessage(session));
req.getRequestDispatcher("/jsp/time.jsp").forward(req, resp);
```

```
}

}

/* # 5 # формирование сообщений для передачи в jsp # SessionLocator.java
*/package by.bsac.timeoutsession;

import java.util.ArrayList;
import java.util.Date;

import
javax.servlet.http.HttpSession;public
class SessionLocator {

private final static String BR = "<br/><hr/>";
public static boolean flag = true;

public static ArrayList<String> addMessage(HttpSession session) {
ArrayList<String> messages = new ArrayList<String>();

if (session != null) { // если сессия существует

messages.add("Creation Time : " + new Date(session.getCreationTime()) + BR);
messages.add("Session id : " + session.getId() + BR);

messages.add("Session alive!" + BR);

} else { // если сессии уже не существует

messages.add("Session disabled!" + BR);

}

return messages;

}

}

# 6 # информация о состоянии сессии # /jsp/time.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<html>

<head><title>Session Control</title></head>

<body>

${messages}

<br/>

<a href ="timeoutservlet">Back to Servlet</a>

</body></html>
```

При первом запуске в браузер будет выведено (см. рисунок 2).

Рисунок 2 - Жизненный цикл сессии. Сессия «alive»

Если повторить запрос к сервлету менее чем через 10 секунд, вывод будет идентично повторен. Если же запрос повторить более чем через десять секунд, сессия будет автоматически уничтожена, и в браузер будет выведено следующее сообщение (см. рисунок 3):

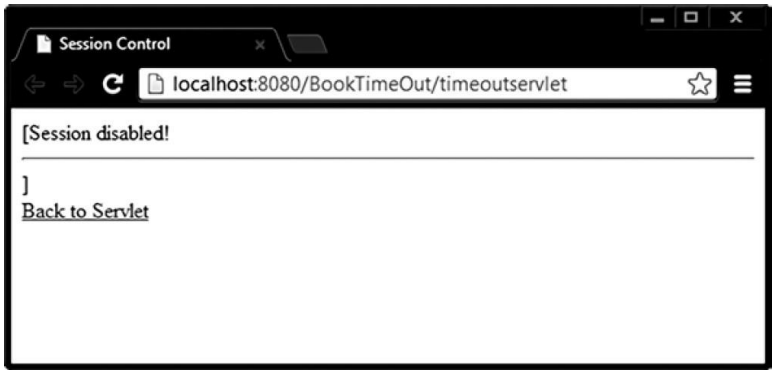


Рисунок 3 - Жизненный цикл сессии. Сессия «not alive»

Следующее обращение к сервлету приведет к созданию новой сессии (см. рисунок 4).

Рисунок 4 - Жизненный цикл сессии. Сессия вновь «alive»

Файлы Cookie

Для хранения информации на компьютере клиента используются возможности класса javax.servlet.http.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 веб-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 веб-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("model", "Canon D7000");
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 «Canon D7000».

Экземпляр Cookie имеет целый ряд параметров: путь, домен, номер версии, время жизни, комментарий. Одним из ключевых является срок жизни в секундах от момента первой отправки клиенту, определить который следует методом setMaxAge(long sec). Если параметр не указан, то cookie существует только до момента прерывания контакта клиента с приложением. В предложенном примере cookie добавляются в сервлете, в сервлете же инициируется процесс их извлечения и передачи информации, содержащейся в файле страницы maincookie.jsp.

```
# 7 # информация из cookie # /jsp/maincookie.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<html><head><title>from Cookie </title></head>

<body>

${messages}

<a href= "cookiecontroller">Messages from Cookie</a>

</body></html>
```

При старте приложения ${messages} ничего не выведет, так как атрибут messages не существует. Сервлет CookieController добавляет cookie к объекту-ответу и пытается извлечь cookie из объекта-запроса. При первом запуске извлекать нечего, так как запрос пришел без cookie.

```
/* # 8 # создание и чтение cookie # CookieController.java */
package by.bsac.servlet.cookie;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;
import
javax.servlet.http.HttpServletResponse;import
by.bsac.servlet.action;
```

```
public class CookieController extends HttpServlet {

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

CookieAction.setCookie(response); // добавление cookie

// извлечение cookie и добавление информации к request
request.setAttribute("messages", CookieAction.addToRequest(request));
request.getRequestDispatcher("/jsp/maincookie.jsp").forward(request, response);

}

}

/* # 9 # формирование и извлечение cookie # CookieAction.java
*/package by.bsac.servlet.action;

import java.util.ArrayList;
import java.util.Date;

import
javax.servlet.http.HttpSession;public
class CookieAction {

private static int number = 1;

public static void setCookie(HttpServletResponse resp) {
String name = "JamesBond";

String role = "00" + number++;
Cookie c = new Cookie(name, role);

c.setMaxAge(60 * 60); // время жизни файла cookie
resp.addCookie(c); // добавление cookie к объекту-
ответуvalue = resp.getLocale().toString();

Cookie loc = new Cookie("locale", value);
resp.addCookie(loc);

}

public static ArrayList<String> addToRequest(HttpServletRequest request) {
ArrayList<String> messages = new ArrayList<>();

Cookie[ ] cookies = request.getCookies();if
(cookies != null) {

messages.add("Number cookies : " + cookies.length);
for (int i = 0; i < cookies.length; i++) {

Cookie c = cookies[i];

messages.add(c.getName() + " = " + c.getValue());

} // end for

} // end if

return messages;

}
```

}

При первом запуске приложения cookie, естественно, переданы не будут, так как они еще не созданы, и браузер еще не получал ни одного ответа. При повторном обращении к сервлету cookie уже будет передан с запросом и в браузер будет выведено (см. рисунок 5):

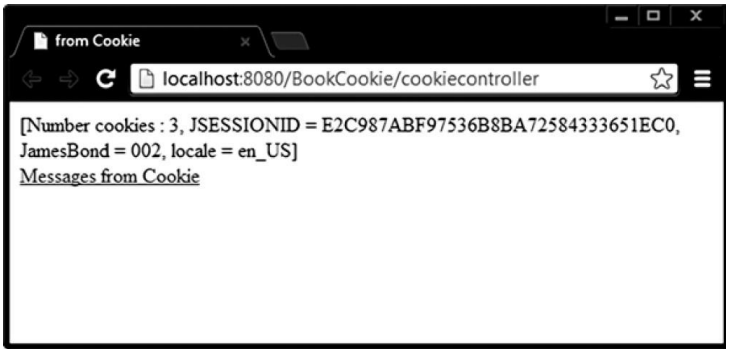


Рисунок 5 - Информация из файла cookie

Файл cookie, как это видно из указанного примера, может быть перезаписан при генерации каждого нового объекта-ответа.

Обработка событий

Изменение состояния некоторых объектов приложения представляют интерес и в чисто информационных целях, и в целях реализации реакции на происходящее. Например, можно отслеживать время входа и выхода из системы пользователей, действия, выполняемые пользователем. Осуществлять инициализацию ресурсов в момент запуска системы, создания пользовательской сессии, и просто служить источником дополнительной информации.

Интерфейсы и их методы
ServletContextListener **contextInitialized(ServletContextEvent e)** **contextDestroyed(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)**

Рисунок 6 - Интерфейсы событий и сигнатуры их методов

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

- javax.servlet.ServletContextListener – обрабатывает события создания/удаления контекста сервлета;
- javax.servlet.ServletContextAttributeListener – обрабатывает события создания/удаления/модификации атрибутов контекста сервлета;
- javax.servlet.http.HttpSessionListener – обрабатывает события создания/удаления HTTP-сессии;
- 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/деактивацией HTTP-сессии;
- javax.servlet.ServletRequestListener – обрабатывает события создания/удаления запроса;
- javax.servlet.ServletRequestAttributeListener – обрабатывает события создания/удаления/модификации атрибутов запроса сервлета. Регистрация блока прослушивания производится в дескрипторном файле приложения или аннотированием класса, реализующего интерфейс Listener.

Демонстрация обработки событий изменения состояния атрибутов сессии рассматривается на примере #1 данной главы. Обрабатываться будут события добавления атрибута role, а также добавления и изменения атрибута для счетчика обращений к сервлету counter. Для этого необходимо создать класс, реализующий интерфейс HttpSessionAttributeListener и реализовать необходимые для выполнения поставленной задачи методы.

/ # 10 # обработка событий добавления и изменения атрибута сессии # SessionListener.java */*

```
package by.bsac.control.listener;

import javax.servlet.annotation.WebListener;

import
javax.servlet.http.HttpSessionAttributeListener;import
javax.servlet.http.HttpSessionBindingEvent;
@WebListener

public class SessionListenerImpl implements HttpSessionAttributeListener {
public void attributeRemoved(HttpSessionBindingEvent ev) {

}

public void attributeAdded(HttpSessionBindingEvent ev) {

// запись в log-файл или иные действия

System.out.println("add: " + ev.getClass().getSimpleName() + " : "+ ev.getName()

+ " : " + ev.getValue());

}

public void attributeReplaced(HttpSessionBindingEvent ev) {

// запись в log-файл или иные действия
```

```
System.out.println("replace: " + ev.getClass().getSimpleName() + " : " + ev.getName()
+ " : " + ev.getValue());
}
}
```

Экземпляр класса события HttpSessionBindingEvent дает доступ к самой сессии методом getSession().

Зарегистрировать обработку событий можно также в элементе <listener> дескрипторного файла web.xml:

```
<listener>
<listener-class>by.bsac.control.listener.SessionListenerImpl</listener-class>
</listener>
```

В результате запуска сервлета SessionControlServlet и двух обращений к нему в консоль будет выведено:

```
add: HttpSessionBindingEvent : role : moderator
add: HttpSessionBindingEvent : counter : 1
replace: HttpSessionBindingEvent : counter : 1
replace: HttpSessionBindingEvent : counter : 2
```

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 уничтожения.

```
/* # 11 # обработка событий создания и уничтожения запроса к сервлету #
SimpleRequestListener.java */
package by.bsac.control.listener;
import javax.servlet.ServletContext;
import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
import
javax.servlet.annotation.WebListener;import
javax.servlet.http.HttpServletRequest;
@WebListener
public class SimpleRequestListener implements ServletRequestListener {
public void requestInitialized(ServletRequestEvent ev) {
// будет использован для получения информации о запросе
HttpServletRequest req = (HttpServletRequest) ev.getServletRequest();
String uri = "Request Initialized for " + req.getRequestURI();
String id = "Request Initialized with ID="+ req.getRequestedSessionId();
System.out.println(uri + "\n" + id);
ServletContext context = ev.getServletContext();
// счетчик числа созданных запросов
Integer reqCount = (Integer)req.getSession().getAttribute("counter");
if(reqCount == null) {
reqCount = 0;
```

```
}

context.log(uri + "\n" +id + ", Request Counter =" + reqCount);

}

public void requestDestroyed(ServletRequestEvent ev) {
System.out.println("Request Destroyed: "

+ ev.getServletRequest().getAttribute("lifecycle"));

}

}
```

С помощью экземпляра ServletRequestEvent, как видно из примера, можно получить доступ к экземпляру запроса HttpServletRequest, а через него и к сессии пользователя, и к контексту приложения.

После вызова страницы index.jsp будет создан новый request с нулевым значением атрибута lifecycle:

```
Request Initialized for /BookSessionListenerControl/index.jsp

Request Initialized with ID=944B8FBDB21C7DCDF9D37745C90C4EC3
Request Destroyed: null

При переходе к сервлету атрибут будет добавлен:

Request Initialized for /BookSessionListenerControl/sessionservlet
Request Initialized with ID=944B8FBDB21C7DCDF9D37745C90C4EC3
Request Destroyed: CONTROL request LIFECYCLE

После обратного перехода к index.jsp объект request, а вместе с ним
атри-бут lifecycle, будет уничтожен:

Request Initialized for /BookSessionListenerControl/index.jsp

Request Initialized with ID=944B8FBDB21C7DCDF9D37745C90C4EC3
Request Destroyed: null

/* # 12 # обработка событий инициализации контекста приложения #

SimpleContextListener.java */
package by.bsac.control.listener;
import javax.servlet.ServletContext;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import
javax.servlet.annotation.WebListener;
@WebListener

public class SimpleContextListener implements ServletContextListener {
public void contextInitialized(ServletContextEvent ev) {

ServletContext context = ev.getServletContext();

String mailFeedback = context.getInitParameter("feedback");
context.log("Context Initialized with parameter: " + mailFeedback);
System.out.println("Context Initialized with parameter: " + mailFeedback);

}
```

public void contextDestroyed(ServletContextEvent ev) {

}

}

Параметры инициализации представлены в web.xml в виде:
<context-param>

<param-name>feedback</param-name>

<param-value>bsac@gmail.com</param-value>

</context-param>

Так как событие инициализации контекста приложения производится только один раз за его жизненный цикл, то за все время работы приложения в log-файл только один раз будут выведены записи:

Dec 31, 2012 11:57:05 PM org.apache.catalina.core.ApplicationContext log
INFO: Context Initialized with parameter: bsac@gmail.com

Фильтры

Реализация интерфейса ***Filter*** позволяет создать объект, который перехватывает запрос, может трансформировать заголовок и содержимое запроса клиента. Фильтры не создают запрос или ответ, а только модифицируют их. 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 или jsp 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 и/или jsp.

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 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(ServletRequest request, ServletResponse response, FilterChain chain) 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.

Фильтр может модифицировать ответ сервера клиенту. Одним из распространенных приемов использования фильтра является модификация кодировки запроса и/или ответа.

Когда страница посылает в сервлет запрос клиента, используется установленная в запросе кодировка. В настоящее время некоторыми браузерами кодировка устанавливается по умолчанию и не зависит от кодировки страницы, обращающейся к серверу, и к тому же вообще не передается серверу приложений. Для корректной интерпретации передаваемых с запросом, например, символов кириллицы, необходимо перехватить запрос и заменить его кодировку.

 # 13 # обращение к кодировке запроса, переданной браузером # index.jsp

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<html><head><title>Encoding Filter</title></head>

<body>

Кодировка запроса: ${ pageContext.request.characterEncoding }

</body></html>

Если фильтр не подключать, то кодировка запроса не будет определена, потому что браузер всегда передает значение null для кодировки запроса. Реализация интерфейса Filter для поставленной задачи, изменяющей кодировку запроса и ответа на кодировку, заданную параметром фильтра.

```
// # 14 # фильтр, корректирующий кодировку запроса и ответа # EncodingFilter.java
package by.bsac.barrier.filter;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;

import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;

import
javax.servlet.annotation.WebInitParam;
@WebFilter(urlPatterns = { "/*" },

initParams = {

@WebInitParam(name = "encoding", value = "UTF-8", description = "Encoding Param")

})

public class EncodingFilter implements Filter {
private String code;

public void init(FilterConfig fConfig) throws ServletException {
code = fConfig.getInitParameter("encoding");

}

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

String codeRequest = request.getCharacterEncoding();

// установка кодировки из параметров фильтра, если не
установленаif (code != null && !code.equalsIgnoreCase(codeRequest)) {
```

```
request.setCharacterEncoding(code);
response.setCharacterEncoding(code);

}

chain.doFilter(request, response);

}

public void destroy() {
code = null;

}

}
```

Параметры инициализации задаются внутри объявления аннотации @WebFilter аннотацией @WebInitParam. Аналогичная конфигурация для предыдущей версии определяется в дескрипторе web.xml:

```
<filter>

<filter-name>encodingfilter</filter-name>

<filter-class>by.bsac.sample.filter.EncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingfilter</filter-name>

<url-pattern>/*</url-pattern>

</filter-mapping>
```

Для использования фильтра для всех сервлетов и JSP используется шаблон "/*", только для JSP – "*.jsp", для каталога – "/jsp/admin/*", для конкретной jsp – "/index.jsp", для сервлета – "/controller".

Фильтры применяются для обеспечения безопасности приложения, а именно, легко решают задачу запрещения несанкционированного прямого обращения к JSP. Следствием установки шаблона urlPatterns в значение "/jsp/*" будет вызов фильтра при любом прямом обращении из строки браузера. Фильтр в итоге перенаправит вызов на указанную страницу.

```
/* # 15 # фильтр, перенаправляющий все прямые обращения к страницам на стартовую

страницу # PageRedirectSecurityFilter.java */
package by.bsac.barrier.filter;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;

import javax.servlet.ServletException;
```

```
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;

import
javax.servlet.annotation.WebInitParam;import
javax.servlet.http.HttpServletRequest; import
javax.servlet.http.HttpServletResponse;
@WebFilter( urlPatterns = { "/jsp/*" },

initParams = { @WebInitParam(name = "INDEX_PATH", value = "/index.jsp") })
public class PageRedirectSecurityFilter implements Filter {

private String indexPath;

public void init(FilterConfig fConfig) throws ServletException {
indexPath = fConfig.getInitParameter("INDEX_PATH");

}

public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException {
HttpServletRequest httpRequest = (HttpServletRequest) request;
HttpServletResponse httpResponse = (HttpServletResponse) response;

// переход на заданную страницу
httpResponse.sendRedirect(httpRequest.getContextPath() +
indexPath);chain.doFilter(request, response);

}

public void destroy() {

}

}
```

Фильтр также успешно перехватывает обращения к сервлету. Например, обращение к сервлету из любого источника перехватывается следующим фильтром. Фильтр проверяет тип пользователя, извлекая его значение из сессии. Если тип не задан, то фильтр присваивает ему значение GUEST и перенаправляет клиента на страницу guest.jsp.

/ # 16 # фильтр, перенаправляющий всех новых пользователей на гостевую страницу*

```
# ServletSecurityFilter.java */
package by.bsac.barrier.filter;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
```

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

import by.bsac.barrier.client.ClientType;

@WebFilter(urlPatterns = { "/controller" }, servletNames = { "MainServlet" })
public class ServletSecurityFilter implements Filter {

public void destroy() {

}

public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException {
HttpServletRequest req = (HttpServletRequest) request;

HttpServletResponse resp = (HttpServletResponse) response;
HttpSession session = req.getSession();

ClientType type = (ClientType) session.getAttribute("userType");if
(type == null) {

type = ClientType.GUEST;
session.setAttribute("userType",
type);

RequestDispatcher dispatcher = request.getServletContext()

.getRequestDispatcher("/jsp/guest.jsp");
dispatcher.forward(req, resp);

return;

}

// pass the request along the filter
chainchain.doFilter(request, response);

}

public void init(FilterConfig fConfig) throws ServletException {

}

}

/* # 17 # перечисление с типами пользователей некоторой системы #
ClientType.java

*/

package by.bsac.barrier.client;
public enum ClientType {

GUEST, USER, ADMINISTRATOR

}
```

Фильтры в общем случае не перехватывают запросы, перенаправленные с помощью методов forward() и include() экземпляра RequestDispatcher. Элемент dispatcherTypes позволяет указать фильтру способ получить управление: непосредственно от сервлета или клиента значением DispatcherType.REQUEST, в результате перенаправления с

использованием метода forward() значением DispatcherType.FORWARD или метода include() значением DispatcherType.INCLUDE) и некоторых других:

```
@WebFilter(dispatcherTypes = {
DispatcherType.REQUEST,
DispatcherType.FORWARD,
DispatcherType.INCLUDE

}, urlPatterns = { "/jsp/*" })
```

Конфигурировать доступ можно любым необходимым набором значений.

Ход работы

Задание: Необходимо реализовать механизм блокировки пользователя, который неверно ввел пароль n-раз, а также отображения даты и времени блокировки (Задание выполнять на основе предыдущей лабораторной работы).

1. Создайте новую JSP страницу blocked.jsp в той же директории, где хранятся другие jsp.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title>Login failed</title>
</head>
<body>
   <p>You are blocked since you tried to type incorrect login or password 3 times.</p>
   <p>Blocked date: ${blockedDate}</p>
   <p>Blocked time: ${blockedTime}</p>
</body>
</html>
```

2. Создайте класс RequestTools в пакете **by.bsac.request** для написания вспомогательных методов в Request.

RequestTools.java

```
package by.bsac.request;

import  java.io.IOException;
import  java.time.LocalDate;
import java.time.LocalTime;

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

public class RequestTools {
   public static void addLoginAttemptToSession(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
       int attempts = Integer.parseInt(req.getSession().getAttribute("failedLoginAttemptsCount").toString());
       req.getSession().setAttribute("failedLoginAttemptsCount", attempts + 1);
```

```
    }

    public static boolean isBlocked(HttpServletRequest req, HttpServletResponse resp) throws
IOException, ServletException {
        if (req.getSession().getAttribute("blockedDate") == null ||
req.getSession().getAttribute("blockedTime") == null) {
            req.getSession().setAttribute("blockedDate", LocalDate.now());
            req.getSession().setAttribute("blockedTime", LocalTime.now());
        }
        return
Integer.parseInt(req.getSession().getAttribute("failedLoginAttemptsCount").toString()) >= 3 ?
true : false;
    }

    public static void redirectToBlockedPage(HttpServletRequest req, HttpServletResponse resp)
throws IOException, ServletException {
        req.getRequestDispatcher("/blocked.jsp").forward(req,resp);
    }
}
```

3. Замените метод doFilter в классе SessionFilter на предложенный ниже.

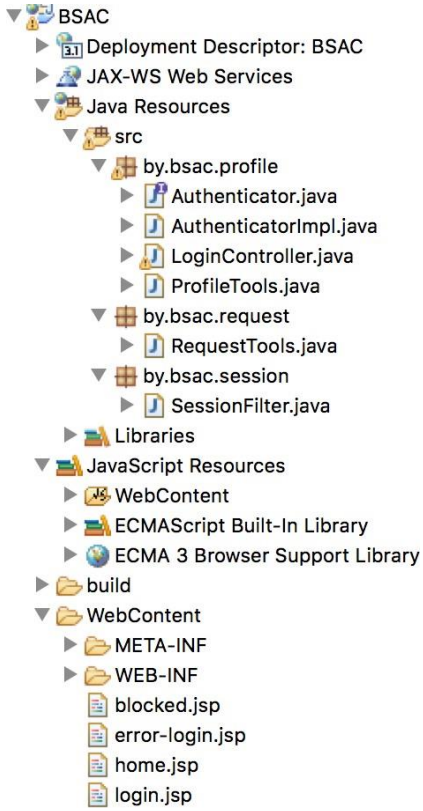
```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;

        if(req.getSession().getAttribute("failedLoginAttemptsCount") == null) {
            req.getSession().setAttribute("failedLoginAttemptsCount", 0);
        }

        // получить текущий uri
        String requestUri = req.getRequestURI();
        boolean shouldBeIgnored = isIgnoredUrl(requestUri);
        // проверяем "Это url, который нужно проигнорировать?"
        // Если url в списке не отработаных и пользователь не залогинен,
        // то перенаправляем на страницу login.jsp
        if(RequestTools.isBlocked(req, res)) {
            RequestTools.redirectToBlockedPage(req, res);
        } else if (!shouldBeIgnored && !ProfileTools.isLoggedIn(req)) {
            req.getRequestDispatcher("/login").forward(req,res);
        } else {
            // pass the request along the filter chain
            chain.doFilter(request, response); // выход из фильтра
        }
    }
```

4. Замените метод login в классе LoginController на нижеприведённый.

```
private void login(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String authTypeParam = request.getParameter("authType");
        // заменить
```

```
        // Authenticator authenticator = new AuthenticatorImpl();
        // на
        Authenticator authenticator = new SimpleAuthenticatorImpl();
        // заменить
        // boolean isAuthentificated = false;
        // на
        Person person;
        String password = request.getParameter("psw");
        String authValue = request.getParameter("loginValue");
        if (authTypeParam.equals("email")) {
            person = authenticator.authenticateByUserEmail(authValue, password);
        } else {
            person = authenticator.authenticateByUserName(authValue, password);
        }
        // заменить
        // if (isAuthentificated) {
        // на
        if (person != null) {
            HttpSession session = request.getSession();
            // добавить
            person.setLoginDate(ProfileTools.generateLoginDate());
            getPersonDAO().save(person);
            // заменить
            // session.setAttribute(ProfileTools.SESSION_LOGGEDIN_ATTRIBUTE_NAME,
            // authValue);
            // на
            session.setAttribute(ProfileTools.SESSION_LOGGEDIN_ATTRIBUTE_NAME,
person.getName());
            // добавить
            if (ProfileTools.isAdmin(person)) {
                // добавить
                session.setAttribute(ProfileTools.PERSON_IS_ADMIN, true);
                session.setAttribute(ProfileTools.ALL_PERSONS_ATTRIBUTE_NAME,
getPersonDAO().getAll());
            }
            request.getSession().setAttribute("failedLoginAttemptsCount", 0);
            request.getSession().removeAttribute("blockedDate");
            request.getSession().removeAttribute("blockedTime");
            response.sendRedirect("home.jsp");
        } else {
            RequestTools.addLoginAttemptToSession(request, response);
            if (RequestTools.isBlocked(request, response)) {
                RequestTools.redirectToBlockedPage(request, response);
            } else {
                response.sendRedirect("error-login.jsp");
            }
        }
    }
```

Структура файлов:



4. Порядок выполнения работы

- изучить теоретический материал;
- выполнить задания из теоретической части лабораторной работы;
- разработайте веб-приложение к индивидуальному заданию (вариант уточните у преподавателя).

5. Индивидуальные задания

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: ключевые слова используются только для обозначения операторов, 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.

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.

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, *.kz 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.

Для вариантов 21 - 30 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.

21. Цветочница. Определить иерархию цветов. Создать несколько объектов-цветов. Собрать букет (используя аксессуары) с определением его стоимости. Провести сортировку цветов в букете на основе уровня свежести. Найти цветок в букете, соответствующий заданному диапазону длин стеблей.

22. Новогодний подарок. Определить иерархию конфет и прочих сладостей. Создать несколько объектов-конфет. Собрать детский подарок с определением его веса. Провести сортировку конфет в подарке на основе одного из параметров. Найти конфету в подарке, соответствующую заданному диапазону содержания сахара.

23. Домашние электроприборы. Определить иерархию электроприборов. Включить некоторые в розетку. Подсчитать потребляемую мощность. Провести сортировку приборов в квартире на основе мощности. Найти прибор в квартире, соответствующий заданному диапазону параметров.

24. Шеф-повар. Определить иерархию овощей. Сделать салат. Подсчитать калорийность. Провести сортировку овощей для салата на основе одного из параметров. Найти овощи в салате, соответствующие заданному диапазону калорийности.

25. Звукозапись. Определить иерархию музыкальных композиций. Записать на диск сборку. Подсчитать продолжительность. Провести перестановку композиций диска на

основе принадлежности к стилю. Найти композицию, соответствующую заданному диапазону длины треков.

26. Камни. Определить иерархию драгоценных и полудрагоценных камней. Отобрать камни для ожерелья. Подсчитать общий вес (в каратах) и стоимость. Провести сортировку камней ожерелья на основе ценности. Найти камни в ожерелье, соответствующие заданному диапазону параметров прозрачности.

27. Мотоциклист. Определить иерархию амуниции. Экипировать мотоциклиста. Подсчитать стоимость. Провести сортировку амуниции на основе веса. Найти элементы амуниции, соответствующие заданному диапазону параметров цены.

28. Транспорт. Определить иерархию подвижного состава железнодорожного транспорта. Создать пассажирский поезд. Подсчитать общую численность пассажиров и багажа. Провести сортировку вагонов поезда на основе уровня комфортности. Найти в поезде вагоны, соответствующие заданному диапазону параметров числа пассажиров.

29. Авиакомпания. Определить иерархию самолетов. Создать авиакомпанию. Посчитать общую вместимость и грузоподъемность. Провести сортировку самолетов компании по дальности полета. Найти самолет в компании, соответствующий заданному диапазону параметров потребления горючего.

30. Таксопарк. Определить иерархию легковых автомобилей. Создать таксопарк. Подсчитать стоимость автопарка. Провести сортировку автомобилей парка по расходу топлива. Найти автомобиль в компании, соответствующий заданному диапазону параметров скорости.

6.Контрольные вопросы.

1. Что такое сессия?
2. Жизненный цикл сессии.
3. Что такое cookies?
4. Что такое Filter? Приведите примеры, где используются фильтры.
5. Перечислите основные действия, которые может выполнить фильтр?
6. Дайте определение понятиям «авторизация» и «аутентификация», в чем ихразличия?