Седьмая лекция
java for web
Cookie

Понятие Cookie

Cookie – это способ сервера (или сервлета, как части сервера) посылать клиенту на хранение часть информации, чтобы потом получать эту информацию от клиента.

Сервлеты посылают куки клиенту, добавляя код в ответ в HTTP заголовки.

Клиенты автоматически возвращают куки, добавляя код в запросы в HTTP заголовках.

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.

Этот механизм позволяет на протяжении нескольких **HTTP** запросов сохранять для браузера на клиенте ту или иную информацию, полученную от сервера.

Схема обмена Cookie

Сервер может отправлять одну или более куки для клиента.
Клиент (браузер) имеет следующие ограничения для cookies:

- всего может храниться до 300 значений cookies
- каждый cookie не может превышать 4Кбайт
- с одного сервера или домена может храниться до 20 значений cookie

Если ограничение 300 или 20 превышается, то удаляется первая по времени запись. При превышении лимита объема в 4Кбайт корректность значения cookie страдает - отрезается кусок записи (с начала этой записи) равный превышению объема.

Обычно файл где хранятся coockies называется 'cookies.txt' и лежит в рабочей директории установленного на компьютер браузера.

Response Header:

Set-Cookie: cname=cvalue;Expires=14-Feb-2006 23:13:26 GMT;Path=/

Request Header:

Cookie: cname=cvalue

Internet Explorer:

...\Documents and Settings\...\Cookies

Описание поля Set-Cookie HTTP заголовка:

Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure

Минимальное описание поля Set-Cookie HTTP заголовка:

Set-Cookie: NAME=VALUE;

NAME=VALUE - строка символов, исключая перевод строки, запятые и пробелы.

NAME-имя cookie, **VALUE** - значение. Не допускается использование двоеточия, запятой и пробела.

expires=DATE - время хранения cookie, т.е. вместо DATE должна стоять дата в формате "expires=Monday, DD-Mon-YYYY HH:MM:SS GMT", после которой истекает время хранения cookie. Если этот атрибут не указан, то cookie хранится в течение одного сеанса, до закрытия броузера.

domain=DOMAIN_NAME - домен, для которого значение cookie действительно. Например, "domain=cit-forum.com". В этом случае значение cookie будет действительно и для домена cit-forum.com, и для www.cit-forum.com. Если этот атрибут опущен, то по умолчанию используется доменное имя сервера, на котором было задано значение cookie.

path=PATH - этот атрибут устанавливает подмножество документов, для которых действительно значение cookie. Например, указание "path=/win" приведет к тому, что значение cookie будет действительно для множества документов в директории /win/, в директории /wings/ и файлов в текущей директории с именами типа wind.html и windows.shtml. Для того, чтобы cookie отсылались при каждом запросе к серверу, необходимо указать корневой каталог сервера, например, "path=/".

Если этот атрибут не указан, то значение cookie распространяется только на документы в той же директории, что и документ, в котором было установлено значение cookie.

secure - если стоит этот маркер, то информация cookie пересылается только через HTTPS (HTTP с использованием SSL - Secure Socket Level), в защищенном режиме. Если этот маркер не указан, то информация пересылается обычным способом.

Синтаксис HTTP заголовка для поля Cookie

Когда запрашивается документ с HTTP сервера, браузер проверяет свои cookie на предмет соответствия домену сервера и прочей информации. В случае, если найдены удовлетворяющие всем условиям значения cookie, броузер посылает их в серверу в виде пары имя/значение:

Cookie: NAME1=OPAQUE_STRING1; NAME2=OPAQUE_STRING2 ...

Класс javax.servlet.http.Cookie

Конструктор класса javax.servlet.http.Cookie создает куки с начальным именем и значением. Вы можете изменить значение куки позже, вызвав метод setValue.

Cookie(String name, String value);

get/setName(String)

get/setValue(String)

get/setAge(int)

get/setPath(String)

is/setSecure(boolean)

get/setVersion(int)

Пример использования.Cookie

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 c = new Cookie("cname", "cvalue");
response.addCookie(c);
```

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 "cvalue".

Преимущества Cookie

Отслеживание сеанса пользователя

Пользовательские настройки

Подстановка имени и пароля при повторном заходе на сайт

Направленная реклама

Использование cookie не представляет угрозы безопасности с точки зрения атак

Браузеры принимают только 20 cookies на сайт и 300 всего, каждое Cookie до 4 кбайт – отсутствует проблема засорения жесткого диска

Недостатки Cookies

Главной проблемой является изначальное недоверие пользователей к тому, что удаленные сервера без их (пользователей) ведома и согласия записывают на их собственные локальные диски какую либо информацию.

Cookie могут подделываться для идентификации пользователя в качестве другого

Cookie – открытые текстовые файлы. Поэтому в них нельзя хранить конфиденциальную информацию. Обычно хранится только идентификатор – данные в хеш-таблице или базе данных на сервере

Отслеживание сеанса

К сожалению, протокол **HTTP**, посредством которого осуществляется взаимодействие в Интернете, является протоколом без поддержки состояния.

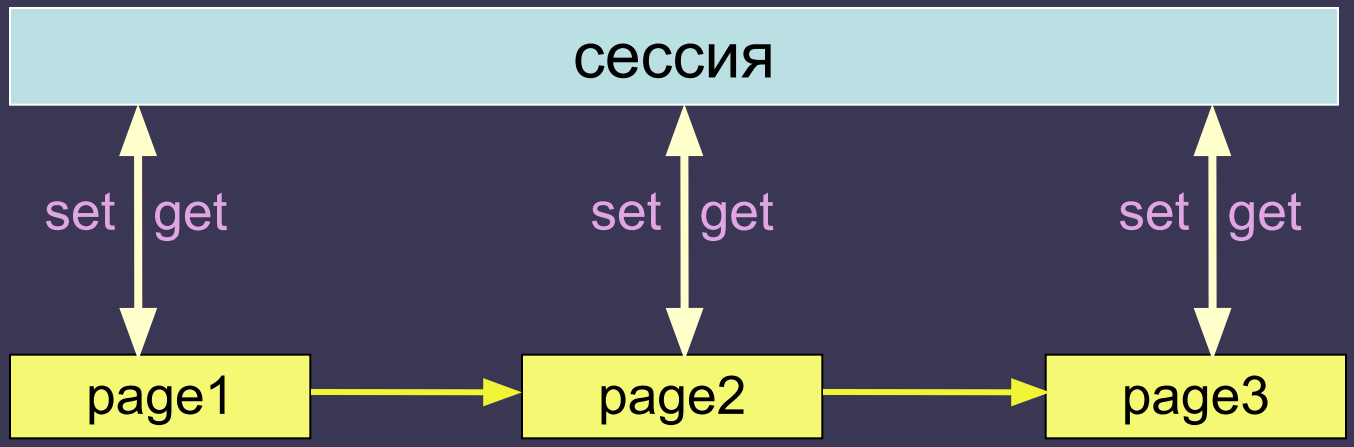
Каждый запрос, получаемый сервером, является независимым элементом данных, не связанным с ранее поступившими запросами.

Поэтому например при нажатии кнопки для добавления товара в корзину покупок приложение должно не только обеспечивать обновление корзины пользователя, но и предупреждать влияние данной корзины на корзины других пользователей, просматривающих сайт в это же время.

Чтобы правильно обработать описанный выше сценарий, необходимо реализовать функцию для создания и ведения сеанса на протяжении пребывания пользователя на сайте.

Технология сервлета, являющаяся основой для всех веб-приложений на базе Java, предоставляет для этих целей интерфейс **HttpSession**.

Сессии



Сессии

Существуют следующие три способа поддержания сеанса между веб-клиентом и веб-сервером:

Cookies

Веб-сервер может присвоить уникальный идентификатор сеанса, как куки для каждого веб-клиента и при последующих запросов от клиента они могут быть исползованы как индификатор.

response.addCookie(“JSESSIONID”, sessionId);

URL-rewriting

http://www.some.com/page.jsp?jsessionid=12345

http://www.some.com/page.jsp/12345

http://www.some.com/page.jsp;jsessionid=12345

Скрытые поля форм

<input type=“hidden” name=“JSESSIONID” value=“12345”>

В Java Web-контейнерами обычно используются по умолчанию Cookies, но если браузер их не поддерживает – автоматически осуществляется переход на URL-Rewriting

Сессии в Java

Сессия – соединение между клиентом и сервером, устанавливаемое на определенное время, за которое клиент может отправить на сервер сколько угодно запросов.

Сессия устанавливается непосредственно между клиентом и Web-сервером. Каждый клиент устанавливает с сервером свой собственный сеанс.

Сессия 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).

При работе с сессией необходимо.

- Создайть для пользователя сессию (объект HttpSession).
- Сохранять или читать данные из объекта HttpSession.
- Уничтожь сессию (необязательное).

•Класс HttpSession

Обьект класса предназначен для работы с сеансами.

Автоматически обеспечивает поддержку сеанса при помощи cookies или перезаписи URL.

Позволяет манипулировать данными о сессии, такими, как идентификатор, время создания, время жизни и т.п.

Позволяет сохранять данные, введенные клиентом в течение нескольких переходов по страницам

Получить ссылку на объект HttpSession в сервлете можно с помощью метода request.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().

Класс HttpSession

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);
```

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:

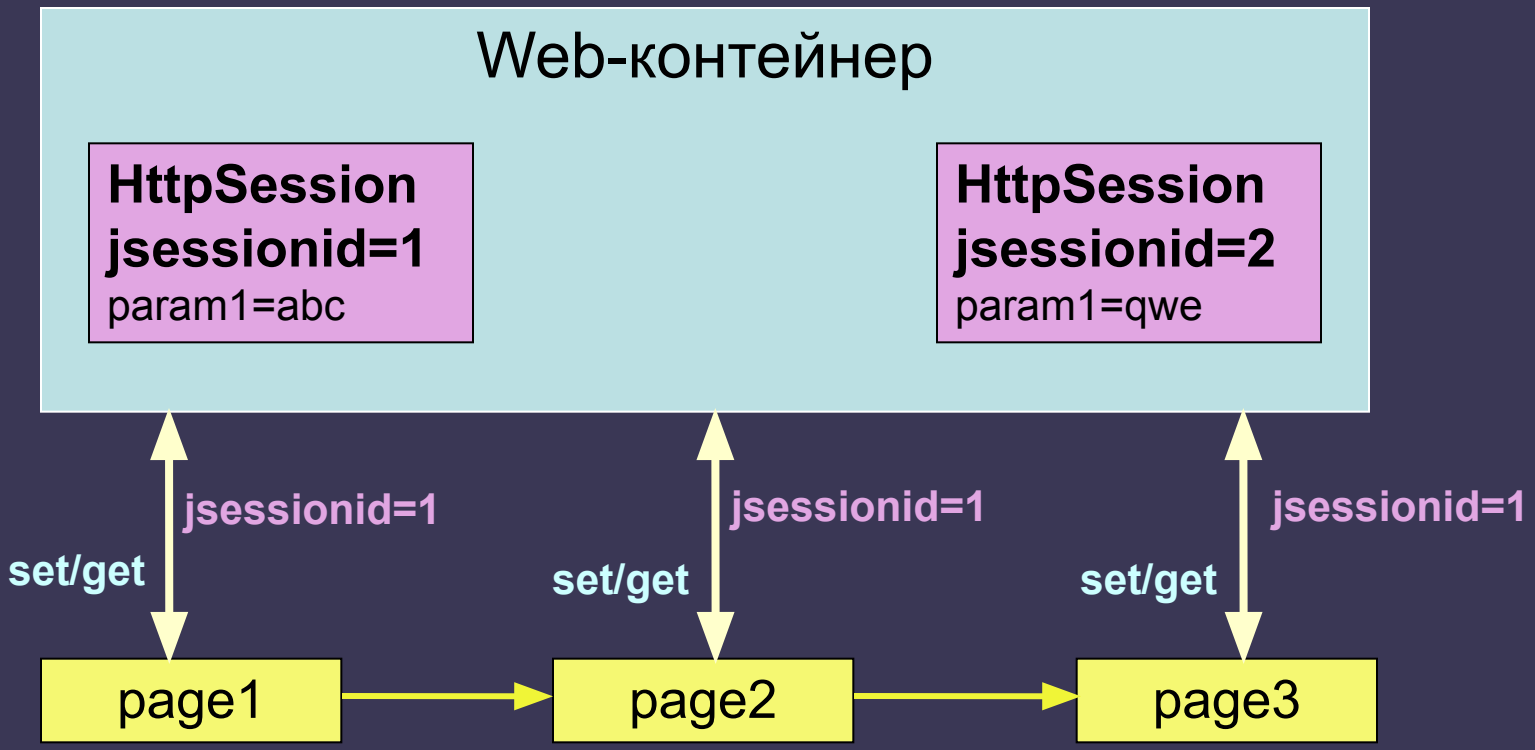
```
se.setAttribute("teacherId", 71);
```

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

```
int testId = 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.

Сессии в Java



Пример работы с сессией

Фрагмент сервлета, проверяющего правильность ввода имени и пароля

```
…
// Берем параметры из запроса
String login = request.getParameter("login");
String password = request.getParameter("password");
// создаем объект User
User user = new User(login, password):
if (userDatabase.contains(user)){       // Если такой есть – помещаем в сессию
    request.getSession().setAttribute("user", user);
}
…
```

Фрагмент сервлета, выводящего имя зарегистрированного пользователя

```
…
// извлекаем объект User из сессии
User user = (User)request.getSession().getAttribute("user");
// Печатаем имя пользователя
response.getWriter().println("User name: " + user.getName());
…
```

Данные, общие для всего приложения

Объект ServetContext существует в единственном экземпляре для одного WEB-приложения.

В нем можно хранить глобальные настройки приложения, с помощью методов get/setAttribute().

Другие методы ServletContext:

getRealPath(String path) – возвращает реальный путь к ресурсу файловой системы, находящемся по заданному виртуальному пути

getResourceAsStream(String path) – возвращает поток байт реального ресурса файловой системы

Получить ссылку на ServletContext из сервлета можно, например, так:

getServletContext()

или по объекту request

request.getSession().getServletContext()

Установка атрибутов

В качестве атрибутов выступают объекты. Атрибуты можно устанавливать на уровне.

запроса

```
request.setAttribute(“myattr”, new Integer(1));
Integer attr = (Integer)request.getAttribute(“myattr”);
```

сессии

```
request.getSession().setAttribute(“myattr”, new Integer(1));
```

приложения

```
getServletContext().setAttribute(…)
```

Фильтры

Фильтр – это Java-код, пригодный для многократного использования и позволяющий осуществлять операции над содержимым HTTP-запросов, ответов и заголовков.

Фильтры не создают запрос или ответ, а только модифицируют его.

Другими словами 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;

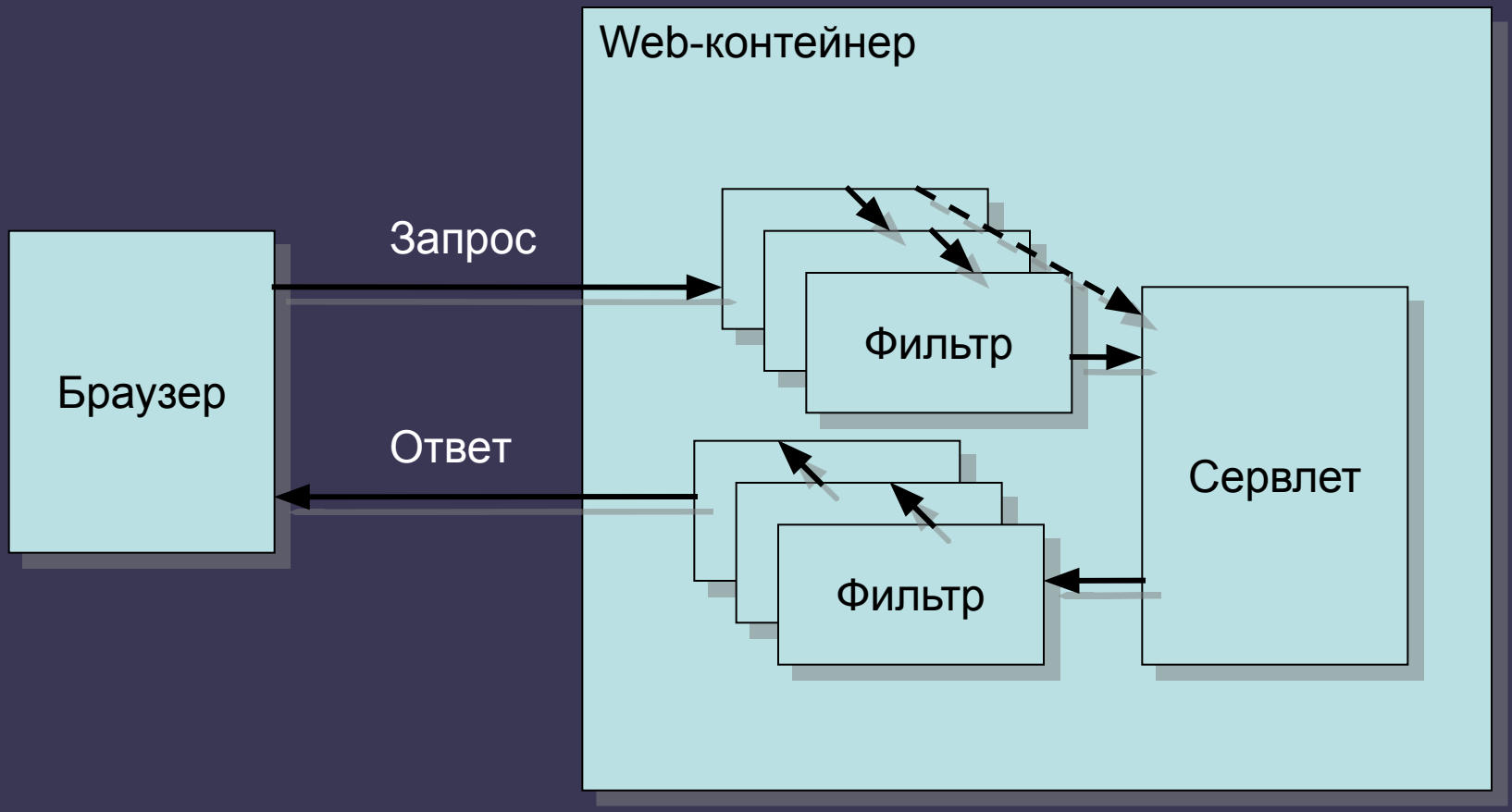
· изменение заголовка и данных запроса и ответа;

· взаимодействие с внешними ресурсами;

· построение цепочек фильтров;

· фильтрация более одного сервлета.

Роль фильтра в обработке запроса



Интерфейс javax.servlet.Filter

Основным методом этого интерфейса является метод

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

которому передаются объекты запроса, ответа и цепочки фильтров.

В данный метод помещается реализация задач, кроме того, необходимо реализовать метод

void init(FilterConfig config),

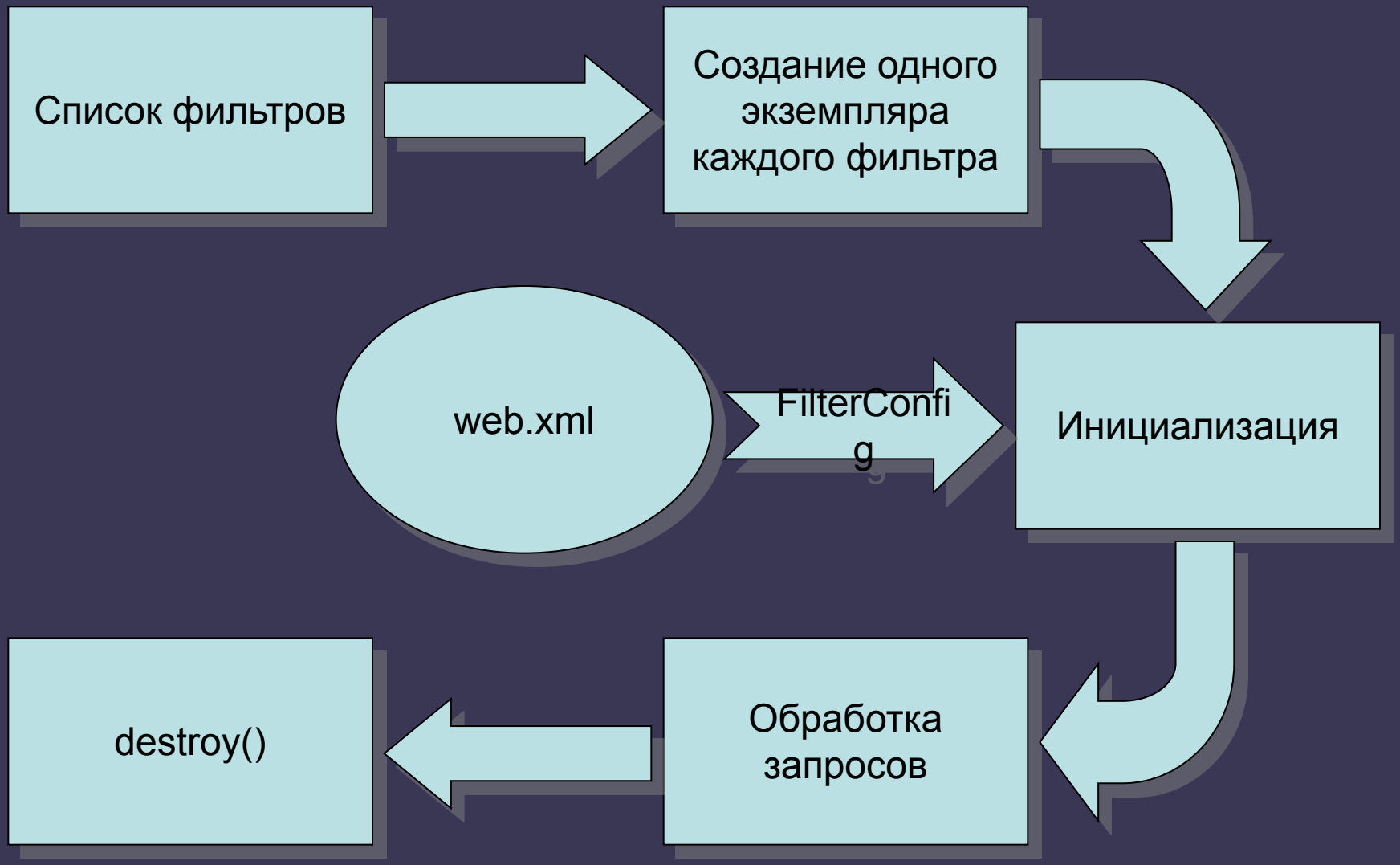
который принимает параметры инициализации и настраивает конфигурационный объект фильтра FilterConfig.

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().

Жизненный цикл фильтра



Интерфейс javax.servlet.FilterConfig

Служит для передачи информации о настройках фильтру при его инициализации. Интерфейс FilterConfig содержит метод для получения имени фильтра, его параметров инициации и контекста активного в данный момент сервлета.

String getFilterName()

String getInitParameter(String)

Enumeration getInitParameterNames()

ServletContext getServletContext()

•Описание фильтров в web.xml

Описание фильтров и их привязок описывается в web.xml-файле перед определением сервлетов.

Цепочка фильтров-обработчиков строится исходя из последовательности появления соответствующих привязок в web.xml.

Привязка фильтров бывает:

к сервлету

```
<filter-mapping>
  <filter-name>Filter1</filter-name>
  <servlet-name>Servlet1</servlet-name>
</filter-mapping
```

к ресурсу по маске

```
<filter-mapping>
  <filter-name>SystemAccessFilter</fliter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

Описание фильтров в web.xml

Порядок, в котором контейнер строит цепочку фильтров для запроса выстраивается в том порядке, в котором встречаются соответствующие описания фильтров в web.xml;

Фрагмент web.xml:

```
 <filter>
    <filter-name>WebtasksCharsetFilter</filter-name>
    <filter-class>filters.WebtasksCharsetFilter</filter-class>
    <init-param>
      <param-name>requestEncoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>WebtasksCharsetFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

...

Пример фильтра

```
package filters;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class WebtasksCharsetFilter extends AbstractFilter{
	private String encoding;
	public void init(FilterConfig config) throws ServletException
	{
	  encoding = config.getInitParameter("requestEncoding");
	  if( encoding==null ) encoding="UTF-8";
	}

	public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain next)
		throws IOException, ServletException
	{

	  if(request.getCharacterEncoding() == null)
		request.setCharacterEncoding(encoding);
	  if(response.getCharacterEncoding() == null)
	  response.setCharacterEncoding(encoding);

	  next.doFilter(request, response);
	}
}
```

Слушатели событий Listener

Слушатель Listener - это уведомляемый о некотором событии объект. Чтобы слушатель смог реагировать на определенное событие источника он должен быть им зарегистрирован, т.е. подключен к источнику. Listener должен реализовывать определенные методы для получения и обработки уведомлений о событии.

Listener находится в постоянном ожидании, пока в источнике, в котором он зарегистрирован, не наступит соответствующее событие, при возникновении которого слушатель получает управление.

Как и фильтры, слушатели создаются и инициализируются web-контейнером при загрузке web-сервера.

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

Интерфейсы listeners и их методы

javax.servlet.ServletContextListener – позволяет разработчику "уловить" момент когда ServletContext инициализируется либо уничтожается. Его можно использовать, например, для открытия соединения с базой данных в момент создания контекста и закрытия соединения в момент уничтожения контекста.

javax.servlet.http.HttpSessionListener – позволяет разроботчику "уловить" момент создания и уничтожения сессии. **javax.servlet.ServletContextAttributeListener** – используется для "слушания" событий, происходящих с атрибутами в сервлет контексте (ServletContext);

javax.servlet.http.HttpSessionAttributeListener – используется для "слушания" событий происходящих с атрибутами в сессии.

javax.servlet.http.HttpSessionBindingListener – так-же используется для прослушивания событий происходящих с атрибутами в сессии. Разница между HttpSessionAttributeListener и HttpSessionBindingListener:

HttpSessionAttributeListener: декларируется в web.xml, экземпляр класса создается автоматически (контейнером) в единственном числе и применяется ко всем сессиям
HttpSessionBindingListener: экземпляр этого класса должен быть создан и закреплён за определённой сессией программистом "вручную", количество экземпляров регулируется программистом.

javax.servlet.http.HttpSessionActivationListener – используется атрибутами сессии в случае, если сессия будет "мигрировать" между различными JVM в распределённых приложениях.;

javax.servlet.ServletRequestListener – используется, соответственно, для того, чтоб "уловить" момент создания и уничтожения запроса.

javax.servlet.ServletRequestAttributeListener – используется при "слушании" событий происходящих с атрибутами запроса.

Servlet Context Listener

ServletContextListener

contextDestroyed(ServletContextEvent e)

contextInitialized(ServletContextEvent e)

ServletContextAttributeListener

attributeAdded(ServletContextAttributeEvent e) - атрибут добавляется в ServletContext

attributeRemoved(ServletContextAttributeEvent e) - атрибут удаляется из ServletContext

attributeReplaced(ServletContextAttributeEvent e) - атрибут меняет значение

ServletRequestListener

requestDestroyed(ServletRequestEvent e) - вызывается когда запрос уничтожается

requestInitialized(ServletRequestEvent e) - вызывается когда запрос инициализируется

ServletRequestAttributeListener

attributeAdded(ServletRequestAttributeEvent e) - атрибут добавляется в запрос

attributeRemoved(ServletRequestAttributeEvent e) - атрибут удаляется из запроса

attributeReplaced(ServletRequestAttributeEvent e) - атрибут меняет значение

HTTP session listeners

javax.servlet.HttpSessionListener:

void sessionCreated(HttpSessionEvent se)

void sessionDestroyed(HttpSessionEvent se)

javax.servlet.HttpSessionAttributeListener:

void attributeAdded(HttpSessionBindingEvent e) - атрибут добавляется в сессию

void attributeRemoved(HttpSessionBindingEvent e) - атрибут удаляется из сессии

void attributeReplaced(HttpSessionBindingEvent e) - атрибут меняет значение

javax.servlet.HttpSessionBindingListener:

void valueBound(HttpSessionBindingEvent event)

void valueUnbound(HttpSessionBindingEvent event)

Описание в web.xml

В web.xml слушатели событий прописываются следующим образом:

```
<listener>
    <listener-class>Полное имя класса</listener-class>
</listener>
```

Слушатели событий описываются после привязок фильтров до определения сервлетов.

HttpSessionBindingListener в web.xml не прописывается, а реализуется классом, который должен отслеживать свою привязку и удаления из сесии

Пример Listener

```
@WebListener
public class SessionListener implements HttpSessionListener {
    private static final Logger LOGGER = Logger.getLogger(SessionListener.class);

    @Override
    public void sessionCreated(HttpSessionEvent se) {
        HttpSession session = se.getSession();
        if(LOGGER.isDebugEnabled()) {
            LOGGER.info("A new session with id='" + session.getId() + "' has been created");
        }
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        HttpSession session = se.getSession();
        if(LOGGER.isDebugEnabled()) {
            LOGGER.info("Session with id='"+session.getId()+"' has been destroyed");
        }
    }
}
```

web.xml:

```
<listener>
        <listener-class>listeners.SessionListener </listener-class>
</listener>
```

Отдельно стоит рассмотреть **HttpSessionBindingListener**, так как он подключается непосредственно в сессию в качестве атрибута. Этот интерфейс содержит два метода: valueBound и valueUnbound. Метод valueBound вызывается перед связыванием объекта с сеансом в качестве идентификатора. Метод valueUnbound вызывается перед отменой связывания объекта с сеансом. Тоесть:

- создать экземрляр класса реализующего этот интерфейс

- положить созданый экземпляр в сессию при помощи setAttribute(String, Object)

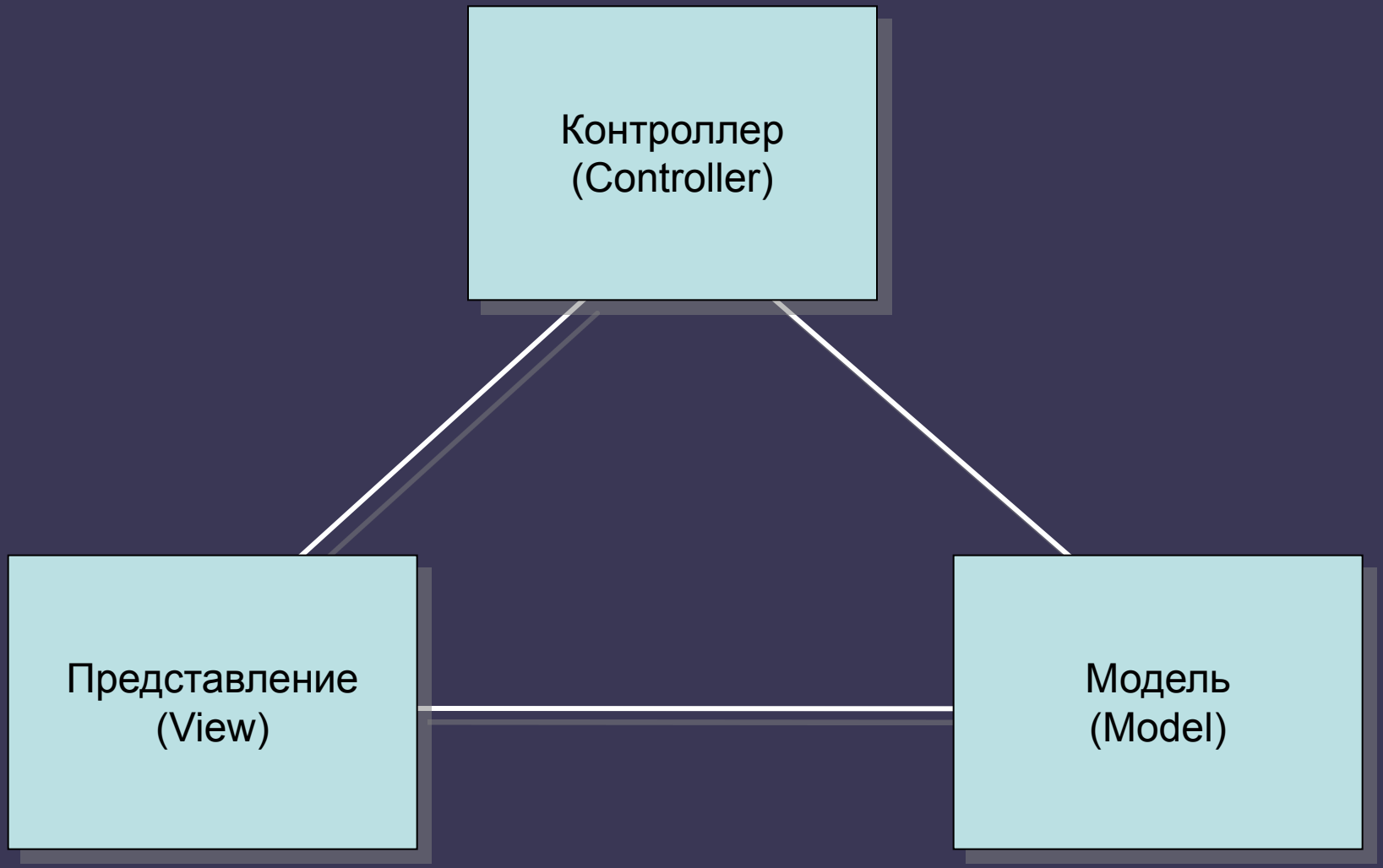
Пример:

Допустим есть класс наследующий HttpServlet с переопределённым методом doGet

```
public void doGet(HttpServletRequest req, HttpServletResponse resp) {
...
    MyInstanceOfHttpSessionBindingListener miohsbl = new
      MyInstanceOfHttpSessionBindingListener();
    HttpSession session = req.getSession();
    req.setAttribute("anyName", miohsbl);            <---- A
{
```

метод valueBound класса MyInstanceOfHttpSessionBindingListener будет вызван в момент A.

Треугольник MVC



Controler:

загружает переменные окружения (POST/GET переменные, параметры командной строки, URL параметры и т. д.);

выполняет первичную обработку переменных окружения (проверка типов переменных, их наличие, установка значений по умолчанию и т. д.);

реализует механизмы контроля за внештатными ситуациями;

реализует механизмы логгирования (не аутентификации, а ведение журналов).

Model:

выполняет конечную проверку входящих параметров (допустимость значений, диапазонов и т. д.);

реализует взаимодействие с системами хранения данных (базы данных, файлы, SOAP и т. д.);

реализует логику работы программы;

подготавливает данные для визуализации.

View:

организует механизмы визуализации результатов работы программы.