Веб-сервер

Apache Tomcat 6

Содержание лекции

1. Ключевые вопросы организации веб-сервера
    1. Понятие веб-сервера и его функции
    2. Протокол HTTP
    3. Аутентификация и авторизация пользователей
    4. Управление сессиями и Cookies
    5. Постоянное HTTP-соединение
2. Веб-сервер Apache Tomcat 6
    1. Структура каталогов сервера
    2. Структура J2EE веб-приложения
    3. Введение в сервлеты
    4. Основные классы Servlet API
    5. Использование Comet Servlet
    6. JSP – Java Server Pages
3. Конфигурация веб-приложения для Tomcat
    1. Дескриптор контекста веб-приложения
    2. Дескриптор развёртывания веб-приложения

1.1 Понятие веб-сервера и его функции

Веб-сервер – это программное обеспечение, обеспечивающее доставку контента конечному пользователю по сети.

Веб-сервер реализует серверную часть протокола HTTP.

Функции и особенности веб-серверов:

- передача контента пользователю;
- получение контента от пользователей;
- поддержка динамически генерируемых страниц;
- аутентификация и авторизация пользователей;
- ведение журнала обращений пользователей к ресурсам;
- поддержка HTTPS для защищённых соединений с клиентами.

Реализации веб-серверов:

- исторически первым считается веб-сервер CERN httpd (1991 год)

java:

- Apache Tomcat
- Jetty

другие:

- Apache
- IIS
- Nginx
- Lighttpd

1.2 Протокол HTTP

- HyperText Transfer Protocol - текстовый протокол передачи данных прикладного уровня
- Текущая версия – 1.1. Принята в 1999 году. В этой версии добавлен режим «постоянного соединения» с сервером.

HTTP-пакет состоит из:

- Стартовая строка

  GET /page.php HTTP/1.1 – запрос

  HTTP/1.1 404 Not Found – ответ

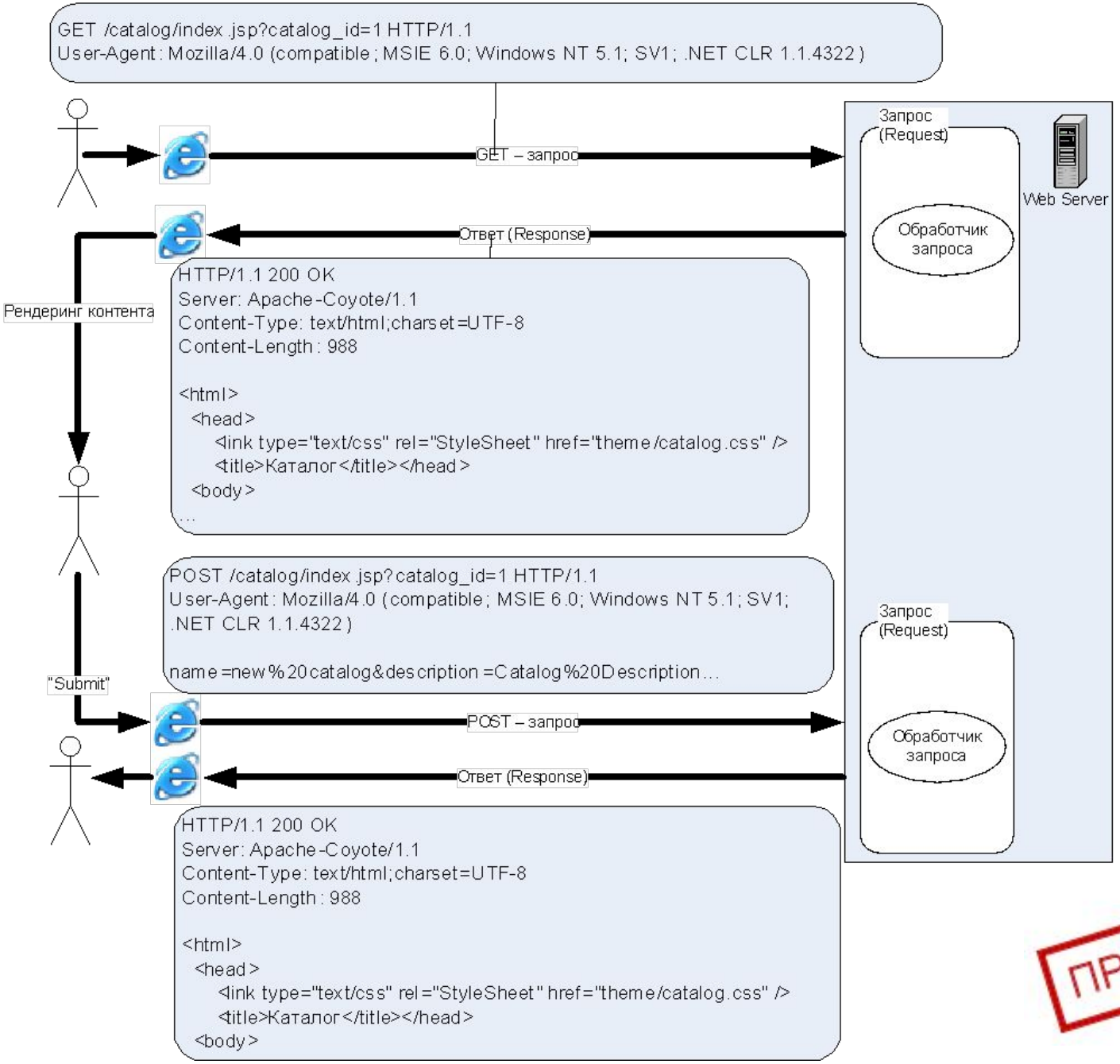
- Заголовки - разделённые двоеточием пары параметр-значение

  Content-Type: text/plain; charset=utf8

- Тело сообщения – любой текст, в том числе закодированный или сжатый

HTTP-методы:

- *HEAD, GET, OPTIONS, TRACE - безопасные*
- *POST, PUT, DELETE - небезопасные*


GET /catalog/index.jsp?catalog_id=1 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
GET – запрос
Запрос (Request)
Web Server
Обработчик запроса
Ответ (Response)
Рендеринг контента
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=UTF-8
Content-Length: 988
<html>
<head>
<link type="text/css" rel="StyleSheet" href="theme/catalog.css" />
<title>Каталог</title></head>
<body>
...
POST /catalog/index.jsp?catalog_id=1 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
name=new%20catalog&description=Catalog%20Description...
"Submit"
POST – запрос
Запрос (Request)
Обработчик запроса
Ответ (Response)
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=UTF-8
Content-Length: 988
<html>
<head>
<link type="text/css" rel="StyleSheet" href="theme/catalog.css" />
<title>Каталог</title></head>
<body>

ПРИМЕР

1. 3 Аутентификация и авторизация пользователей

- **Basic** access authentication - логин и пароль кодируются с помощью Base64, поддерживается всеми браузерами
- **Digest** access authentication - логин и пароль шифруются алгоритмом MD5

При доступе к защищённому контенту клиент получает ответ:

HTTP/1.1 401 Authorization Required

WWW-Authenticate: Basic realm="Secure Area"

либо

WWW-Authenticate: Digest realm="testrealm@host.com", nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093"

Авторизация не поддерживается протоколом HTTP и полностью возлагается на веб-сервер

1. 4 Управление сессиями и Cookies

Веб-приложению бывает необходимо контролировать передвижения клиентов по сайту. HTTP не отслеживает состояние своих клиентов (stateless протокол). Возможные решения данной проблемы:

1. Добавление параметра session_id в URL:
   - функции по контролю времени жизни сессии возлагаются на веб-сервер
   - сессия не сохраняется после перезапуска браузера
2. Хранение session_id в cookies браузера. Cookies – небольшой фрагмент данных, созданный веб-сервером и хранимый на компьютере пользователя в виде файла, который браузер может пересылать веб-серверу в HTTP-запросе:
   - функции по контролю времени жизни сессии возлагаются на браузер
   - сессия сохраняется после перезапуска браузера

1.5 Постоянное HTTP-соединение

Протокол HTTP поддерживает возможность отправки нескольких запросов веб-серверу сразу, не создавая при этом новых TCP-соединений.

Для этого в первом запросе в заголовке передаётся параметр Connection: Keep-Alive.

По умолчанию все соединения являются постоянными.

Преимущества постоянного соединения:

- меньшая нагрузка на память, сеть и процессор, т.к. создаётся меньше TCP-соединений
- возможность отправки сразу нескольких запросов, не дожидаясь ответов (http-pipeline)
- возможность ожидания ответа в течении длительного промежутка времени (server-push)

2. Apache Tomcat 6

- Apache Tomcat 6 является одним из наиболее популярных Web-серверов, реализующих спецификацию JEE 5 (Java Enterprise Edition). Он бесплатно распространяется для коммерческого и некоммерческого использования под лицензией Apache Software License с открытыми исходными кодами.
- Сам продукт, документацию и исходные коды можно найти на http://tomcat.apache.org.

Apache Tomcat состоит из следующих компонентов:

1. Web Connector ***Coyote***, реализующий протокол HTTP/1.1, с помощью которого пользователь, используя Интернет-браузер, может отправлять запросы к серверу и получать ответ.

2. Web Container ***Catalina*** реализует спецификацию Servlet API 2.5 из JEE 5. Servlet API является основой для всех остальных технологий Java касающихся Web и дает возможность динамически генерировать любой Web-контент, используя любые библиотеки, доступные для java.

3. ***Jasper Compiler*** – компилятор JSP-страниц (поддерживает спецификацию JSP 2.1). JSP страница является наиболее популярным (но не единственным) средством создания динамически-генерируемых HTML, XML и других документов, имеющих текстовое представление.

2.1 Структура каталогов сервера

/bin/ - скрипты запуска, остановки и пр.

/startup.bat (startup.sh) – запуск

/shutdown.bat (shutdown.sh) – остановка

/catalina.bat (catalina.sh) – скрипт запуска и настройки параметров (вызывается из startup.bat)

/conf/ - конфигурационные файлы

/server.xml – основной конфигурационный файл, тут задаются порты, коннекторы и пр.

/web.xml – включается во все web.xml файлы приложений по-умолчанию

/lib – библиотеки сервера, все библиотеки из этой папки доступны всем приложениям. Сюда необходимо положить JDBC-драйвер, если приложение использует БД

/log – логи (протоколы) сервера. Используются для статистики, диагностики и отладки

/temp – папка для временных файлов сервера и приложений

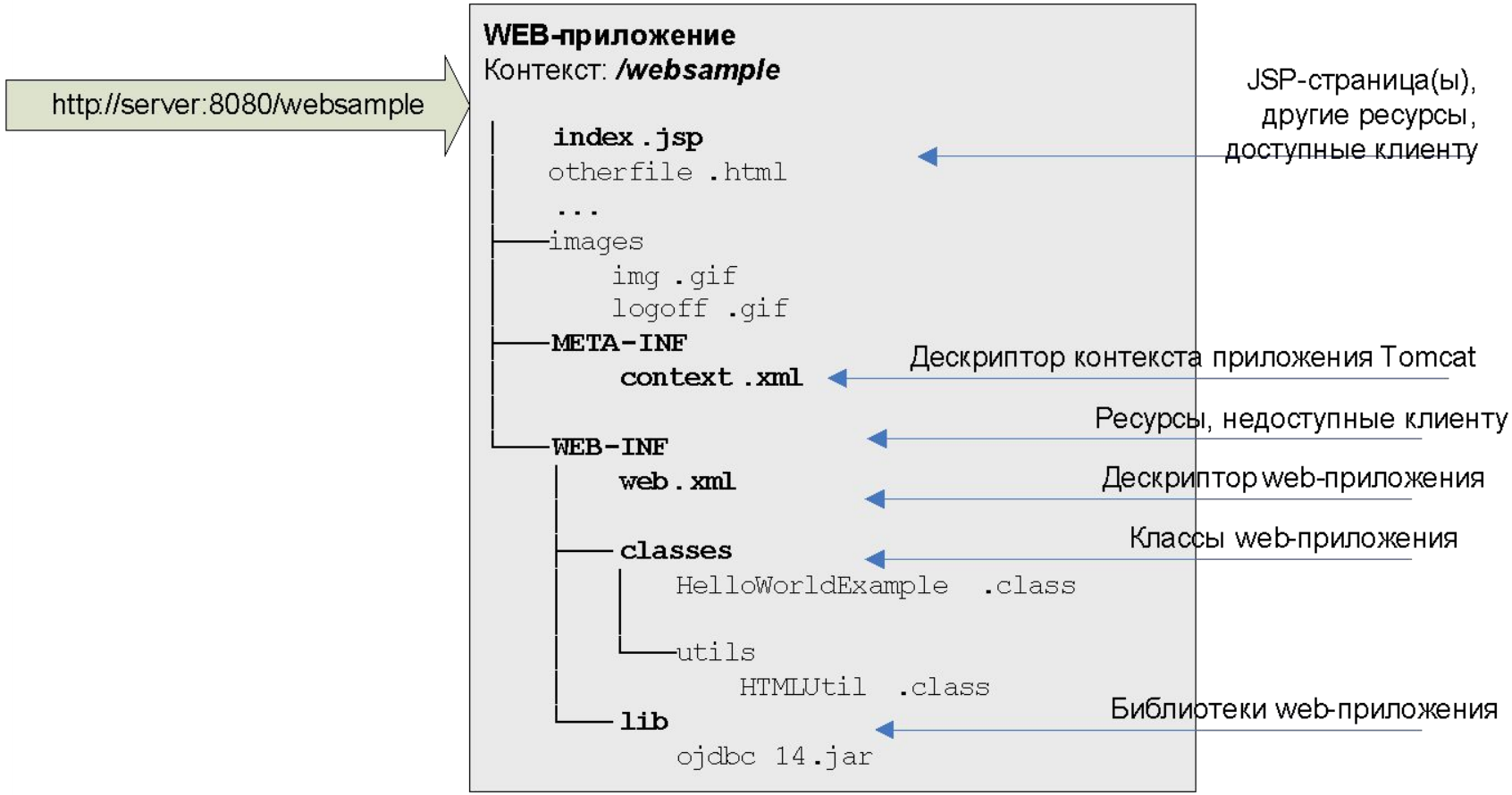
/webapps – папка для приложений. Все, что здесь лежит будет развернуто и запущено при старте сервера, а так же во время его работы (hot-deploy)

/ROOT – специальное имя для корневого контекста. Приложение в этой папке будет привязано к контексту "/", т.е. доступно через http://localhost:8080/

/docs, examples, host-manager, manager, ROOT – примеры приложений, документация и административная консоль, входящие в поставку сервера. Их можно без ущерба удалить.

/work – рабочая папка сервера. Сюда будут складываться скомпилированные на лету JSP файлы, сериализованные сессии пользователей (чтобы рестарт сервера не выкинул пользователей из системы) и пр. Содержимое можно (и, иногда, нужно) без ущерба удалять.

2.2 Структура J2EE веб-приложения



2.3 Введение в сервлеты

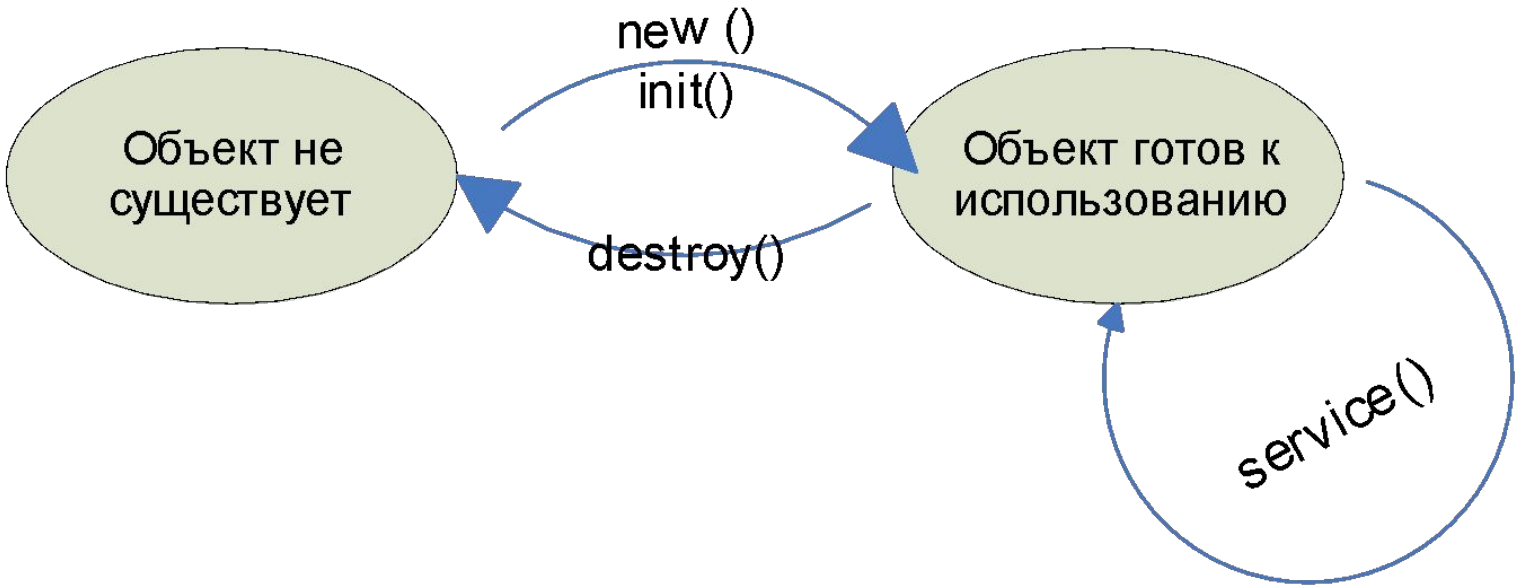
Сервлет – Java-класс, наследуемый от javax.servlet.http.HttpServlet:

- doGet(), обработка http-метода GET
- doPost(), обработка http-метода POST
- doPut(), обработка http-метода PUT
- doDelete(), обработка http-метода DELETE
- init() and destroy(), управление жизненным циклом сервлета
- getServletInfo(), возвращает описание сервлета

Жизненный цикл сервлета состоит из следующих шагов:

1. В случае отсутствия сервлета в контейнере.
    1. Класс сервлета загружается контейнером.
    2. Контейнер создает экземпляр класса сервлета.
    3. Контейнер вызывает метод init(). Этот метод инициализирует сервлет и вызывается в первую очередь, до того, как сервлет сможет обслуживать запросы. За весь жизненный цикл метод init() вызывается только однажды.
2. Обслуживание клиентского запроса. Каждый запрос обрабатывается в своем отдельном потоке. Контейнер вызывает метод service() для каждого запроса. Этот метод определяет тип пришедшего запроса и распределяет его в соответствующий этому типу метод для обработки запроса. Разработчик сервлета должен предоставить реализацию для этих методов. Если поступил запрос, метод для которого не реализован, вызывается метод родительского класса и обычно завершается возвращением ошибки инициатору запроса.
3. В случае если контейнеру необходимо удалить сервлет, он вызывает метод destroy(), который снимает сервлет из эксплуатации. Подобно методу init(), этот метод тоже вызывается единожды за весь цикл сервлета.

Жизненный цикл сервлета:



Пример сервлета:

```
public class SimpleServlet extends HttpServlet {
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
         out.println("<html><head><title>Заголовок</title></head><body><h1>Пример
сервлета</h1>");
         out.println("</body></html>");
      } finally {
         out.close();
      }
   }
   @Override
   public String getServletInfo() {
      return "Пример сервлета";
   }
}
```

2.4 Основные классы Servlet API

(javax.servlet.http.*)

- ***HttpServletRequest*** – класс, экземпляры кот. представляют запрос от браузера
  - ***String getContextPath()*** – возвращает путь к контексту приложения
  - ***String getServletPath()*** – URL вызванного сервлета (JSP)
  - ***HttpSession getSession()*** – Сессия пользователя
  - ***Object getAttribute() / void setAttribute(String name, Object value)*** – Хранение пользовательских атрибутов, связанных с запросом
  - ***String getParameter(String value)*** – Параметр запроса (и для GET и для POST)
  - ***void setCharacterEncoding(String enc)*** – Кодировка значений параметров запроса (windows-1251, UTF-8)

- ***HttpServletResponse*** – класс, экземпляры кот. представляют ответ браузеру
  - ***void setContentType(String contentType)*** – MIME-тип ответа браузеру
  - ***java.io.PrintWriter getWriter()*** – поток вывода для ответа браузеру
  - ***void sendRedirect(String location)*** – перенаправление на другую страницу

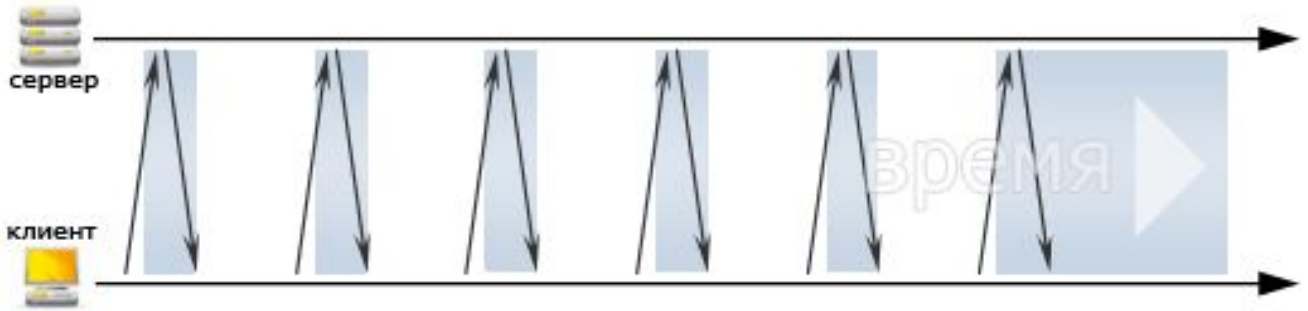
- ***HttpSession*** – класс, экземпляры кот. хранят состояние сессии клиента
  - ***Object getAttribute() / void setAttribute(String key, Object value)*** – Атрибуты сессии (сохраняются между запросами одного клиента)
- ***HttpServletContext*** – класс, экземпляры кот. представляют все web-приложение
  - ***Object getAttribute() / void setAttribute(String key, Object value)*** – Атрибуты контекста (общие для всех пользователей и запросов к web-приложению)

2.5 Использование Comet Servlet

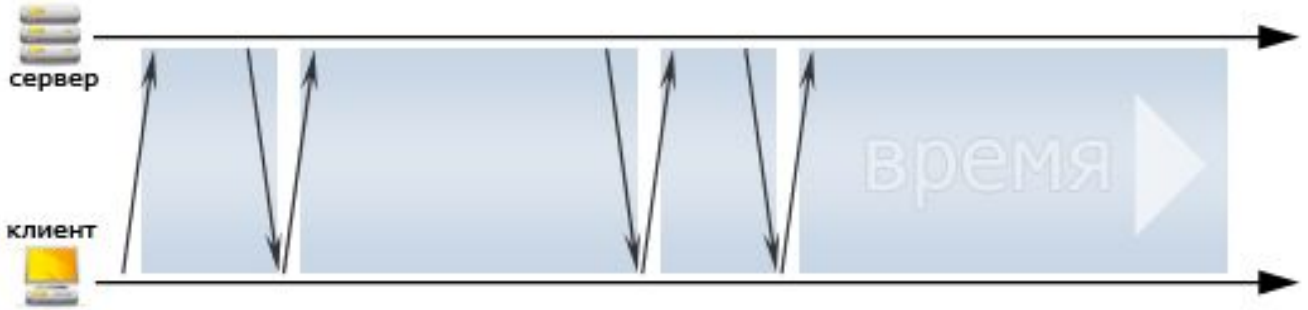
Comet - модель работы веб-приложения, при которой постоянное HTTP-соединение позволяет веб-серверу отправлять (push) данные браузеру, без дополнительного запроса со стороны браузера.

Примеры: различные чаты, vkontate.ru, GWT-декларант

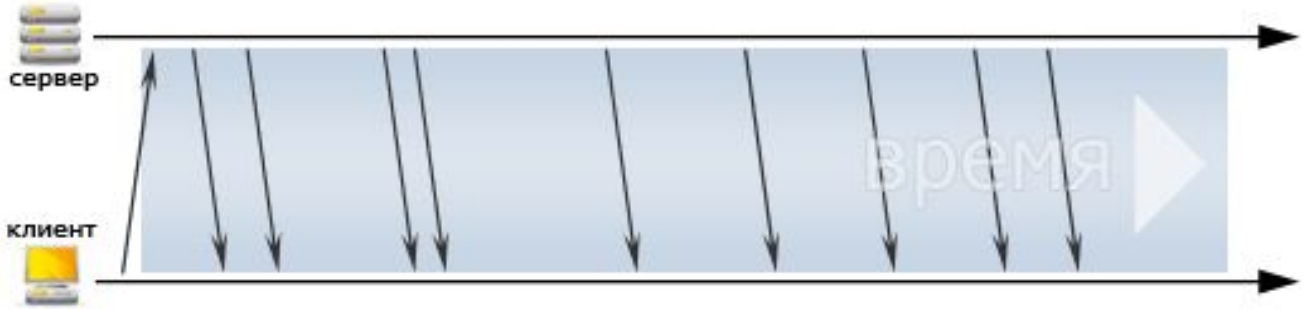
Сравнение классического подхода к построению веб-приложений с Comet



Классический подход



Comet подход (он же long polling)



Streaming (он же server-push)

Для использования Comet сервлет должен реализовать интерфейс org.apache.catalina.CometProcessor

с единственным методом **event**(CometEvent event), вызываемым при поступлении следующих событий:

- **EventType.BEGIN** – при создании соединения с веб-сервером
- **EventType.READ** – при получении данных для сервлета
- **EventType.END** – при завершении соединения с веб-сервером
- **EventType.ERROR** – при возникновении ошибки

Клиентский код для соединения с Comet сервлетом:

```
<SCRIPT TYPE="text/javascript">
        function go(){
            var url = "http://localhost:8080/CometServlet"
            var request =  new XMLHttpRequest();
            request.open("GET", url, true);
            request.setRequestHeader("Content-Type","application/x-javascript;");
            request.onreadystatechange = function() {
                if (request.readyState == 4) {
                    if (request.status == 200){
                        if (request.responseText) {
                            document.getElementById("cometresult").innerHTML = request.responseText;
                        }
                    }
                    go();
                }
            };
            request.send(null);
        }
</SCRIPT>
```

Клиентский код для соединения с Streaming сервлетом:

1) На страницу вставляется скрытый iframe, загружающий данные с Streaming сервлета
2) Данные от сервера должны поступать в виде кусков javascript:

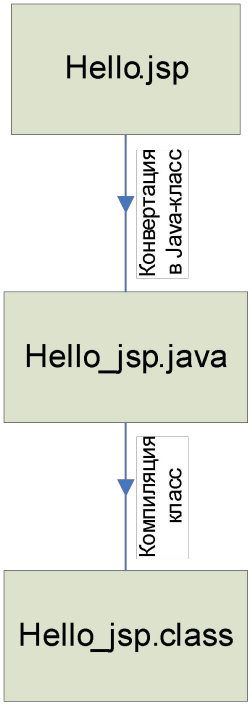
```
<SCRIPT TYPE="text/javascript">
        function go(){
    doSomething();
        }
</SCRIPT>
```

3) Соединение не будет закрываться до тех пор, пока не закроется браузер.

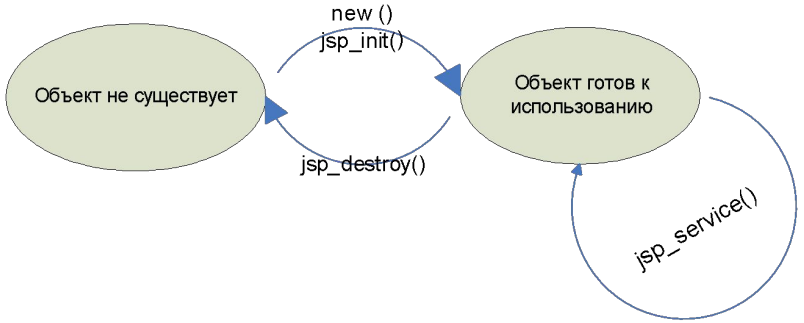
2.6 JSP – Java Server Pages

Жизненный цикл

1. Жизненный цикл класса страницы



2. Жизненный цикл объекта страницы



Элемент JSP	Представление в JSP-файле	Преобразуется в java-класс как
Импорт пакета	<%@ page import="java.util.*" %>	import java.util.*;
Скриптлет:	<% List items=new ArrayList(); **// любой java-код** %>	jsp_service() { … List items=new ArrayList(); //любой java-код … }
Вывод на страницу	<%= new Date() %>	out.write(new Date());

```
<%@ page language="java"
    contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<%@ page import="java.util.*" %>
<%!
// Объявляется поле в классе страницы
int my_integer_field = 777;
// Объявляется метод в классе страницы
    private String make_greeting(String name)
{
    return "Hello, "+name + "!";
    }
%>
<html>
<head>
    <title>Sample Hello world page</title>
</head>
<body>
<% // Скриптлет 1
   for (int i=0; i<10; i++) {
%>
<h1>
<%=
   /* Вывод в поток out */
   make_greeting("World "+i)
%>
</h1>
<%   // Скриптлет 2
   } // Конец цикла
%>
</body>
</html>
```

```
import java.util.*;
public final class hello_jsp extends
org.apache.jasper.runtime.HttpJspBase
{
// Объявляется поле в классе страницы
int my_integer_field = 777;
// Объявляется метод в классе страницы
private String make_greeting(String name) {
    return "Hello, "+name + "!";
}
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
      throws java.io.IOException, ServletException {
    PageContext pageContext = null;
    HttpSession session = null;
    JspWriter out = null;
    Object page = this;
      response.setContentType("text/html;
charset=UTF-8");
      session = pageContext.getSession();
      out = pageContext.getOut();
      out.write("<html>\n");
      out.write("<head>\n");
      out.write("\t<title>Sample Hello world
page</title>\n");
      out.write("</head>\n");
      out.write("<body>\r\n");
 // Скриптлет 1
   for (int i=0; i<10; i++) {

      out.write("\r\n");
      out.write("<h1> ");
      out.print( /* Вывод в поток out */
make_greeting("World "+i) );
      out.write("</h1>\r\n");
 // Скриптлет 2
   } // Конец цикла
      out.write("\n");
      out.write("</body>\n");
      out.write("</html>");
}
}
```

3. Конфигурация веб-приложения

Глобальные файлы конфигурации:

- Конфигурационный файл сервера **conf/server.xml**
- Дескриптор развёртывания **web-**приложения по-умолчанию **conf/web.xml**

Относящиеся к конкретному приложению:

- Дескриптор контекста **web-**приложения **META-INF/context.xml**
- Дескриптор развёртывания **web-**приложения **WEB-INF/web.xml**

3.1 Дескриптор контекста web-приложения context.xml

Контекст web-приложения – это совокупность следующих параметров URL:

- Виртуальное имя хоста (www.mysite.com)
- Относительный путь к web-приложению (/myapplication)

Таким образом, при запросе от клиента страницы по адресу http://www.mysite.com/myapplication/page.jsp веб-сервер отправит запрос на обработку нужному сервлету, описанному в файле web.xml приложения myapplication

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/myapplication" cookies="true"  reloadable="true" >
    <Resource >…</Resource >
    <Valve> …</Valve>
    <Manager>…</Manager>
    <Realm>…</Realm>
    <Listener className="com.mycompany.mypackage.MyListener" />
</Context>
```

- **Resource** – подключение ресурсов, например, исчтоников данных (БД)
- **Valve** – подключение фильтров, например, блокирование запросов по IP-адресу
- **Manager** – использовать собственный менеджер сессий
- **Realm** – подключение источника данных, хранящий логины и пароли пользователей для их аутентификации в приложении
- **Listener** – отслеживание жизненного цикла контекста

Настройка источника данных (подключение к БД):

Файл /META-INF/context.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
  <!--  контекст web-приложения - префикс всех URL для данного приложения  -->
     <Context path="/websample">
           <!-- Объявление ресурса-источника данных  -->
           <Resource auth="Container" type="javax.sql.DataSource"
                      driverClassName="oracle.jdbc.driver.OracleDriver
                      maxActive="20" maxIdle="10" maxWait="-1"
                     name="jdbc/sample"
                      url="jdbc:oracle:thin:@:1521:spm" username="o50"
                     password="o50" />
     </Context>
```

Использование DataSource:

```
// Создаем начальный контекст JNDI (Java Naming Directory)
InitialContext ctx = new InitialContext();

// Достаем из контекста источник данных
DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/sample");

// Получаем соединение с БД из источника данных
return ds.getConnection();
```

3.2 Дескриптор развёртывания web-приложения web.xml

Веб-приложения Java используют файл дескриптора развертывания для определения способа сопоставления URL с сервлетами, URL, для которых необходима аутентификация, и других сведений. Этот файл называется web.xml и находится в каталоге WEB-INF/web.xml и является частью стандарта Servlet для веб-приложений.

Сервлеты и пути URL

```
<servlet>
    <servlet-name>SendEmailService</servlet-name>
    <servlet-class>mycompany. SendEmailServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>AbcServlet</servlet-name>
    <servlet-class>mycompany.AbcServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>SendEmailService</servlet-name>
    <url-pattern>/SendEmail</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>AbcServlet</servlet-name>
    <url-pattern>/*.abc</url-pattern>
</servlet-mapping>
```

Безопасность и аутентификация

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>SecurePages</web-resource-name>
        <url-pattern>/admin/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <description>Only authenticated users may pass</description>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>

<security-role>
    <description>Administrator role</description>
    <role-name>admin</role-name>
</security-role>

<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>MySite Admin Security Check</realm-name>
</login-config>
```

Список приветственных файлов

```
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
 </welcome-file-list>
```

Обработчики ошибок

```
<error-page>
    <error-code>500</error-code>
    <location>/errors/servererror.jsp</location>
 </error-page>
```

Кодировка и локаль

```
<locale-encoding-mapping-list>
    <locale-encoding-mapping>
      <locale>ru</locale>
      <encoding>UTF-8</encoding>
    </locale-encoding-mapping>
 </locale-encoding-mapping-list>
```

Время жизни сессии

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

Контрольные вопросы

1. Из каких частей состоит HTTP-пакет?
2. Что такое Cookies? Как реализовано управление сессией с помощью Cookies?
3. Из каких основных компонентов состоит Apache Tomcat?
4. Что такое сервлет? Что такое Comet-сервлет?
5. Перечислите файлы конфигурации веб-приложения. Какие параметры в них можно задавать?

Практика

Написать веб-приложение, которое по GET-запросу http://localhost:8082/app/sum?a=3&b=4 выводит на страницу сумму a и b.

1. Создать сервлет SumServlet.
2. В нём переопределить метод doGet(). Используя метод getParameter() объекта request получить значения a и b
3. Используя объект response вывести результат на страницу:

```
PrintWriter out = response.getWriter();
    try {
      out.println(...);
    } finally {
       out.close();
          }
```

4. Прописать созданный сервлет в дескрипторе развёртывания web.xml:

```
<servlet>
      <servlet-name>SumServlet</servlet-name>
     <servlet-class>mycompany.SumServlet</servlet-class>
</servlet>
<servlet-mapping>
     <servlet-name>SumService</servlet-name>
      <url-pattern>/sum</url-pattern>
</servlet-mapping>
```

Полезные ссылки

1. http://tomcat.apache.org
2. http://en.wikipedia.org/wiki/Http
3. http://www.ibm.com/developerworks/web/library/wa-cometjava/
4. http://java.sun.com/developer/onlineTraining/Servlets/Fundamentals/servlets.html