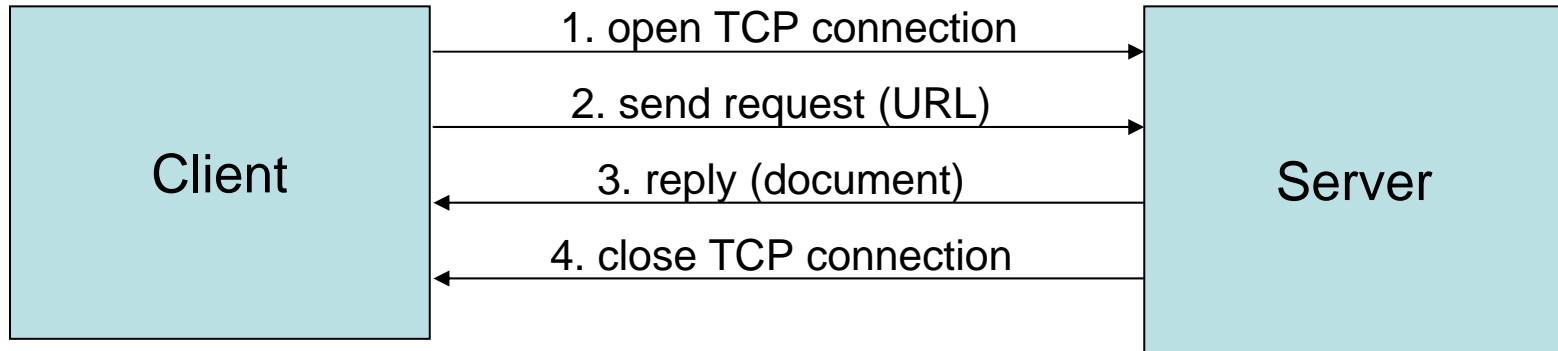


Hypertext Transfer Protocol (HTTP)

- **Simple data transfer protocol**
 - state-less
 - (partially) idempotent
- Client/Server, Request/Reply
- Application layer of TCP/IP protocol stack



HTTP: History

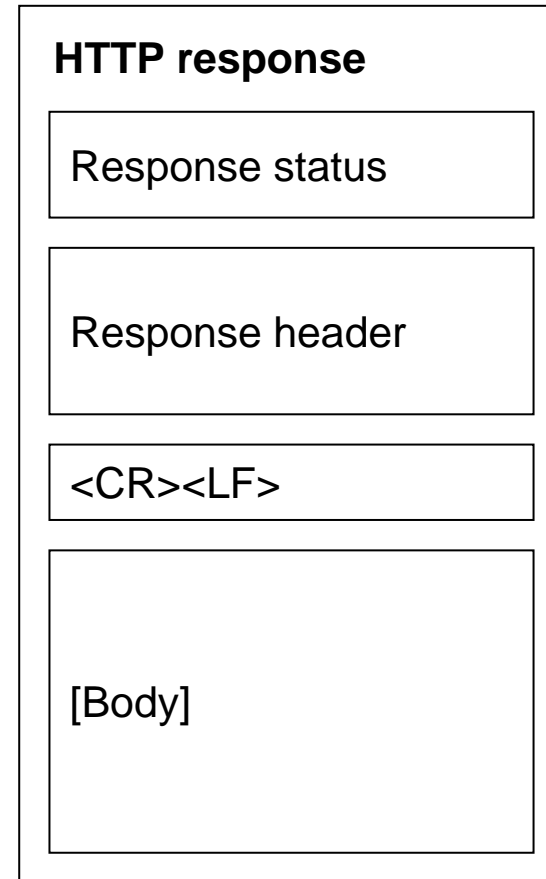
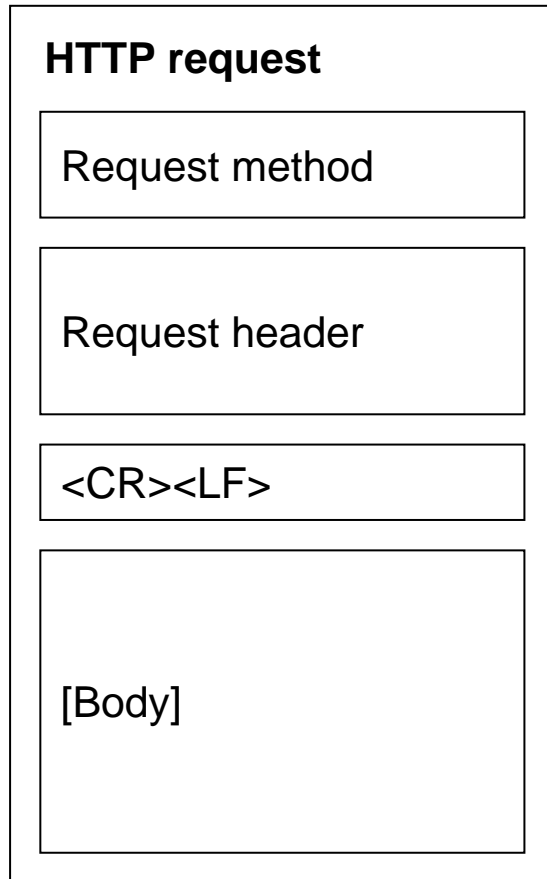
- **HTTP 0.9** (Tim Berners-Lee, 1991)
 - <http://www.w3.org/History/19921103-hypertext/hypertext/WWW/Protocols>
 - only GET method, no persistent connections
- **HTTP 1.0** (1996)
 - <http://tools.ietf.org/html/rfc1945>
 - structured message format, more request types, basic authentication
 - no persistent connections, no partial retransfer of documents, insecure authentication
- **HTTP 1.1** (1999)
 - <http://tools.ietf.org/html/rfc2616>
 - persistent connections, pipelining, cache control, byte range operations, cookies

HTTP: Request/Response

```

GET /index.html HTTP/1.1[<CR><LF>] ← request line
Host: www.apache.org[<CR><LF>] ← fields
[<CR><LF>]
HTTP/1.1 200 OK[<CR><LF>] ← status line
Date: Sun, 04 Mar 2007 17:25:02 GMT[<CR><LF>]
Server: Apache/2.3.0-dev (Unix)[<CR><LF>]
Last-Modified: Mon, 26 Feb 2007 18:14:27 GMT[<CR><LF>]
ETag: "a56ac-433a-1c9da6c0"[<CR><LF>]
Accept-Ranges: bytes[<CR><LF>]
Content-Length: 17210[<CR><LF>]
Cache-Control: max-age=86400[<CR><LF>]
Expires: Mon, 05 Mar 2007 17:25:02 GMT[<CR><LF>]
Vary: Accept-Encoding[<CR><LF>]
Content-Type: text/html[<CR><LF>]
[<CR><LF>]
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ← message
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> body
<!-- Copyright 1999-2006 The Apache Software Foundation [...]

```



HTTP: Requests

- **Format:**

Method Request URI HTTP version

- **Request methods**

- GET requests representation of specified resource
- HEAD asks for header of specified resource
- POST submits data to be processed (in body)
- PUT uploads representation of the specified resource
- DELETE deletes the specified resource
- OPTIONS server returns HTTP methods supported
- TRACE echoes back the received request
- CONNECT for proxy that can change to being an SSL tunnel

HTTP: Response Codes

- **Informational: 1xx**
 - request received, continuing process
 - e.g. “100 Continue”, “101 Switching Protocols”
- **Success: 2xx**
 - action successfully received, understood, and accepted
 - e.g. “200 OK”, “204 No Content”
- **Redirection: 3xx**
 - client must take additional action to complete the request
 - e.g. “301 Moved Permanently”, “304 Not Modified”
- **Client error: 4xx**
 - request contains bad syntax or cannot be fulfilled
 - e.g. “403 Forbidden”, “404 Not Found”, “405 Method not Allowed”
- **Server error: 5xx**
 - server failed to fulfill an apparently valid request
 - e.g. “500 Internal Server Error”, “505 HTTP Version not supported”

HTTP: Content Types

■ Media types

type / subtype

- **types**: text, image, video, audio, application, multipart, message, model
- **subtypes**: currently about 130 predefined
- previously called MIME types (Multi-purpose Internet Mail Extensions, RFC 2045-49)

■ Content type

Content-Type: *type / subtype (; parameter)**

- entity body resolved as *content-encoding(content-type(entity-body))*
- If not given, application/octet-stream assumed
- examples

```
Content-Type: text/html
```

```
Content-type: text/plain; charset=us-ascii (Plain text)
```

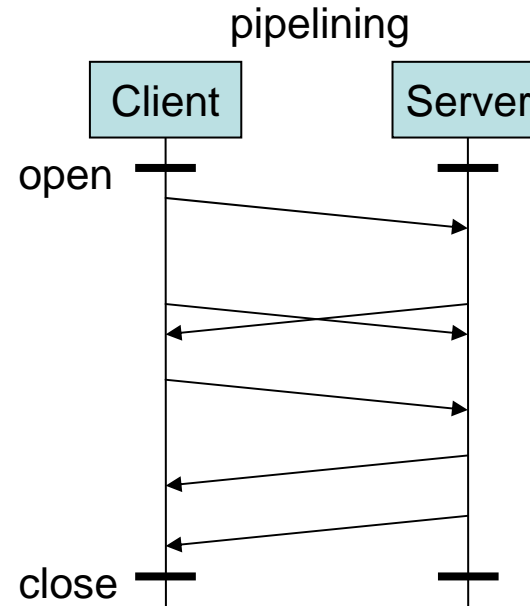
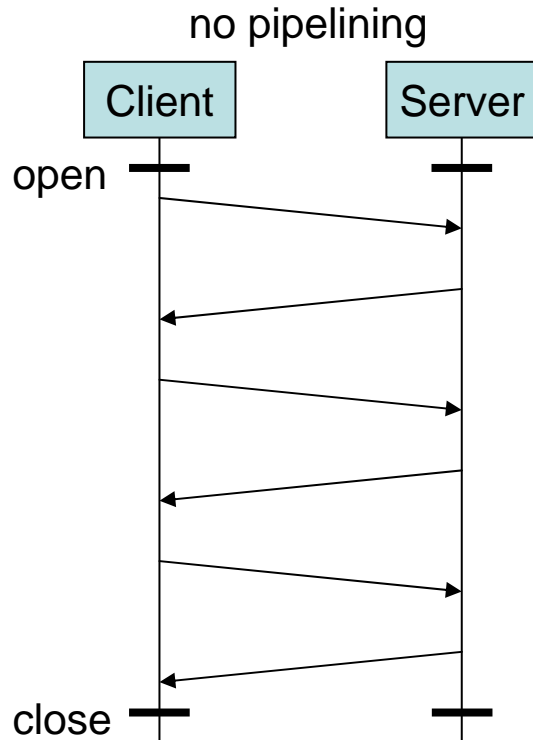
```
content-Type: image/jpeg; name="test.jpg"
```

```
content-type: application/vnd.mozilla.xul+xml
```

← vendor specific

HTTP: Persistent Connections

- Using less resources and TCP packets
- Enables pipelining
 - order has to be kept by the server
- Header field **Connection: keep-alive / close**



- **Authorized access to resources**

- and logging of accessing users

- **In Apache**

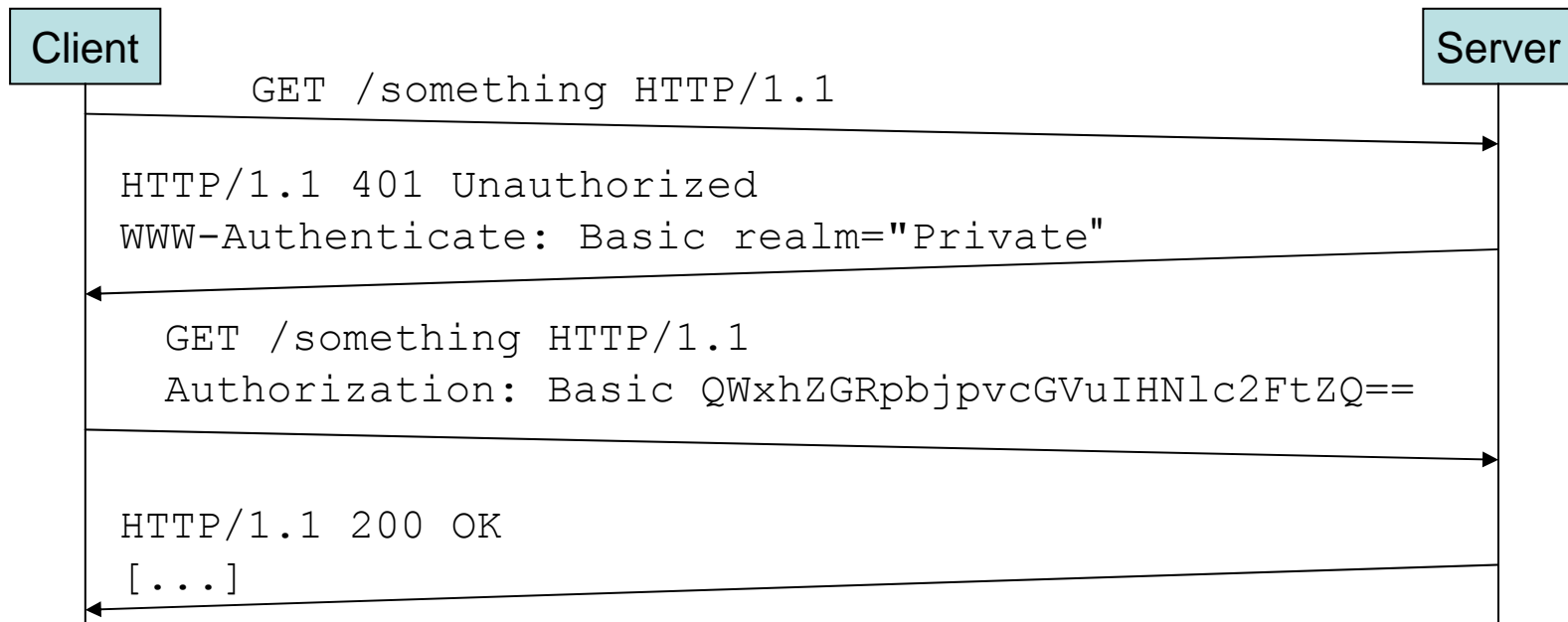
- **File .htaccess**

```
Satisfy all
AuthType Basic
AuthName "Private Area"
Require user nora
AuthUserFile /etc/httpd/.htpasswd
Order deny,allow
Allow from all
```

- **File .htpasswd**

```
# nora:test
nora:tPaLNTyn7sIJs
```

- **Basic authentication (RFC 2617)**
 - simple user and password schema (*username:password*, base64), no encryption



HTTP: Digest Access Authentication

- **Digest access authentication** (RFC 2617)
 - challenge/response procedure, password not sent in clear

- **Challenge**

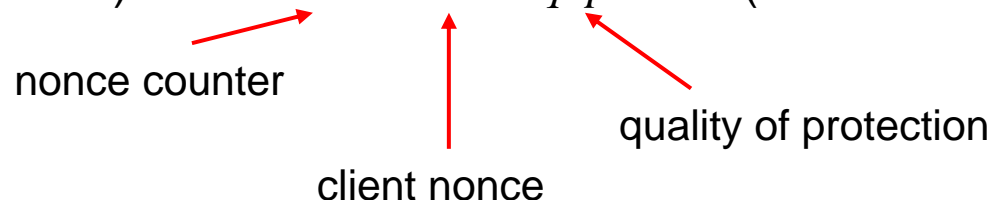
```
WWW-Authenticate: Digest realm="Private",
nonce="7aEjFtHKAwA=57e4252a2c68c0f9d30b0699f9b7de79d7895e1b",
algorithm=MD5, qop="auth"
```

- **Response**

```
Authorization: Digest username="nora", realm="Private",
nonce="7aEjFtHKAwA=57e4252a2c68c0f9d30b0699f9b7de79d7895e1b",
uri="/", algorithm=MD5,
response="e6f59508c24c4893b1acadf5551839b9", qop=auth,
nc=00000001, cnonce="e3719279e8bdaeel"
```

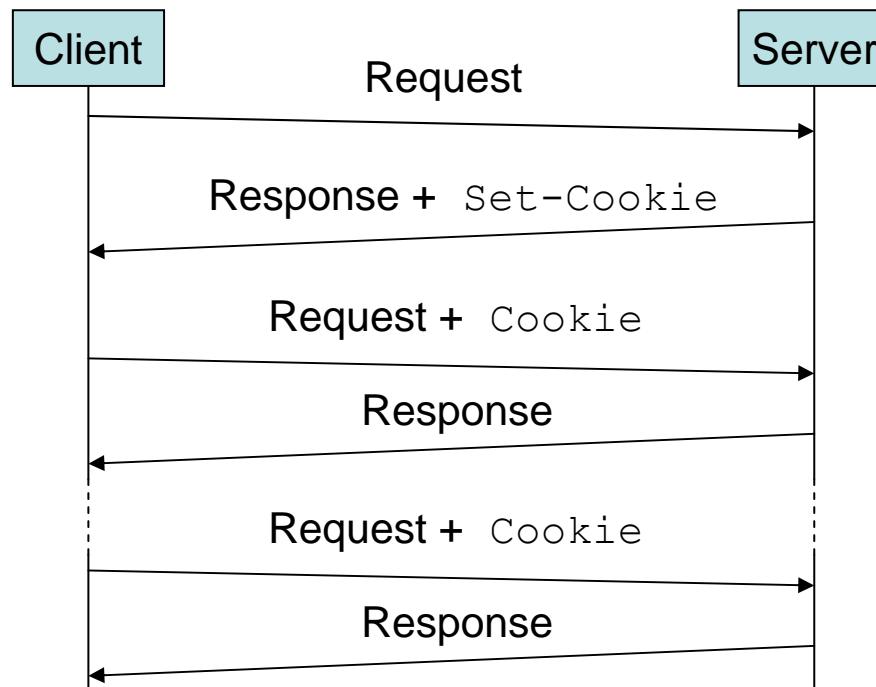
- **Structure of response**

$MD5(MD5(username : realm : password) : nonce : nc : cnonce : qop : MD5(method : URI))$



HTTP: Cookies (1)

- **HTTP stateless**
 - keep information between requests by using cookies or URL rewriting
- **Cookies (RFC 2965)**
 - response header: `Set-Cookie`
 - request header: `Cookie`, depending on domain and path



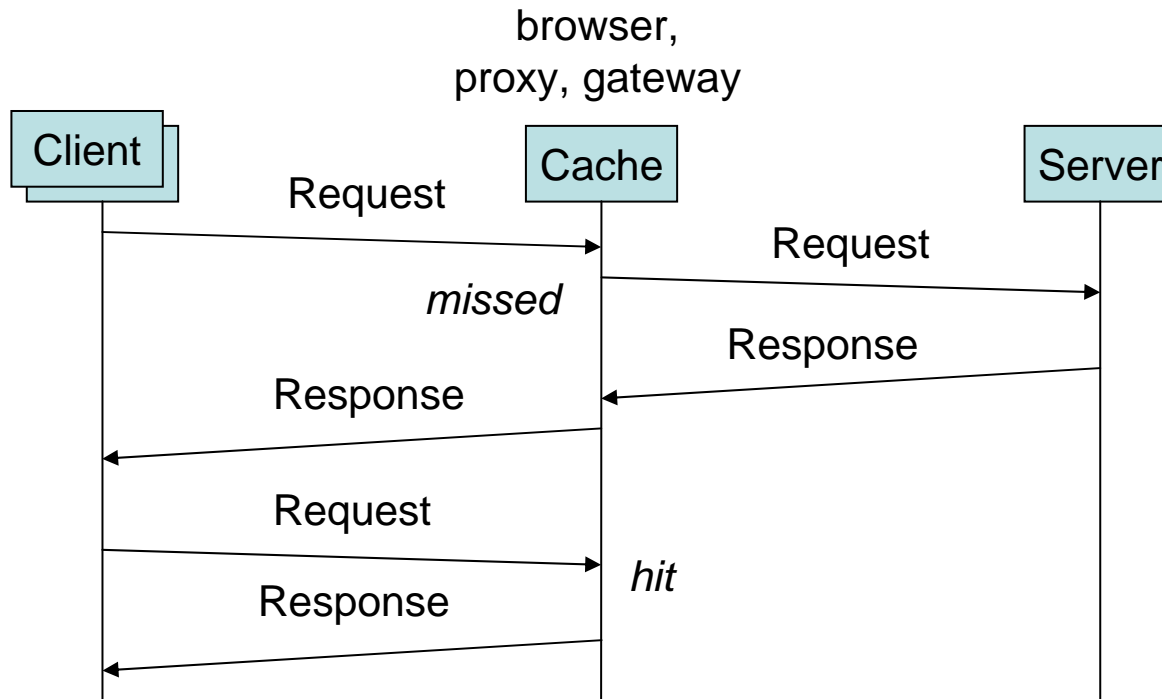
HTTP: Cookies (2)

```
GET / HTTP/1.1
Host: www.amazon.com
[<CR><LF>]
HTTP/1.1 200 OK
Date: Mon, 02 Apr 2007 08:05:45 GMT
Server: Server
Set-Cookie: skin=noskin; path=/; domain=.amazon.com; expires=Mon, 02-
  Apr-2007 08:05:45 GMT
Set-cookie: session-id-time=11761020001; path=/; domain=.amazon.com;
  expires=Mon Apr 09 07:00:00 2007 GMT
Set-cookie: session-id=002-5595215-2499224; path=/;
  domain=.amazon.com; expires=Mon Apr 09 07:00:00 2007 GMT
[...]
[<CR><LF>]
```

```
GET / HTTP/1.1
Host: www.amazon.com
Cookie: session-id-time=11761020001; session-id=002-5595215-2499224
[<CR><LF>]
```

■ Cache

A program's local store of response messages and the subsystem that controls its message storage, retrieval, and deletion. **A cache stores cacheable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests.** Any client or server may include a cache, though a cache cannot be used by a server that is acting as a tunnel. (RFC 2616)



HTTP: Caching (2)

■ Freshness

- cached representation can be served without checking with origin server

■ Validators

- used to check whether representation is fresh without retrieving the representation
- Last-modified (last modification time) – cache may ensure freshness of representation by If-modified-since
- ETag (resource identifier) – cache may ensure freshness of representation by If-none-match

■ Expires: *date* – representation fresh up to *date*

■ Cache-Control: no-store, no-cache, must-revalidate, max-age=*sec*

- no-store – do not cache representation
- no-cache – revalidate each time
- must-revalidate – strictly obey caching rules
- max-age=*sec* – representation fresh for *sec* seconds

- **URI scheme: https**
 - standard port: 443
 - involves trusted **certificate authority** (CA)
- **Asymmetric session initiation**
 1. Web browser sends connection request `https://...` on port 443 to Web server
 2. Web server sends its certificate to Web browser
 3. Web browser verifies certificate with CA
 4. Web browser extracts Web server public key from certificate
 5. Web browser generates nonce and sends it to Web server encrypted by extracted public key
 6. Web server extracts nonce by its private key and sends own nonce to Web browser
 7. Web browser and Web server construct session key from nonces using hashing
- **Symmetric SSL tunnel**
 8. Web browser sends session key encrypted requests
 9. Web server sends session key encrypted responses

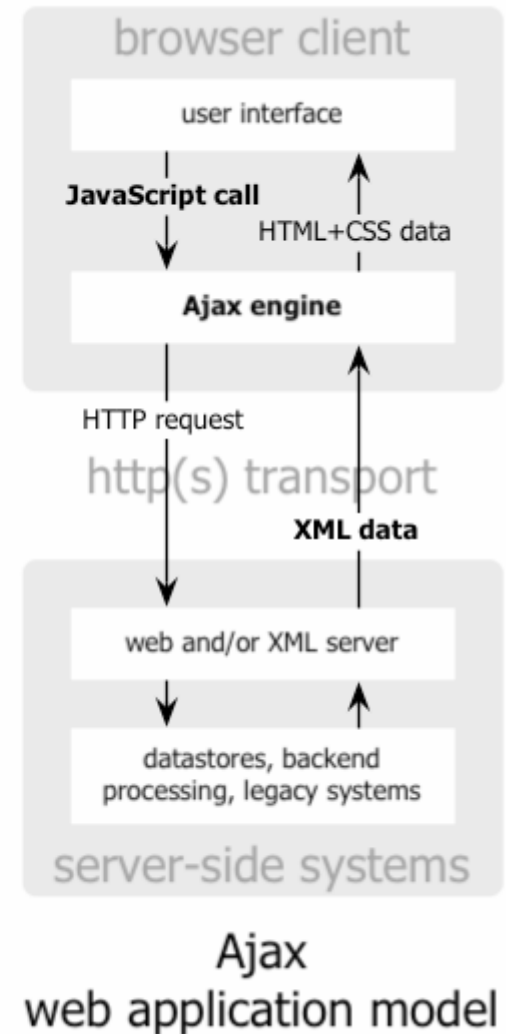
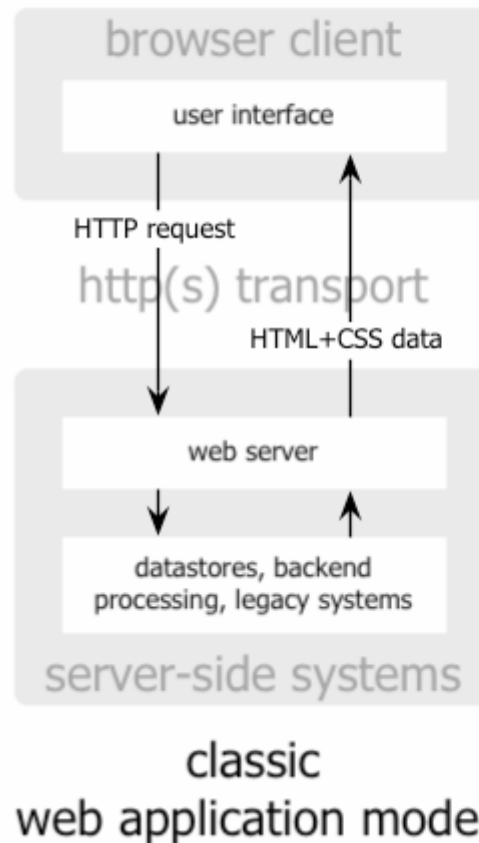
- **Extension of HTTP** (RFC 2518, 1999; RFC 2518bis, 2007)
 - collaborative editing and managing of files on Web servers
 - IETF WebDAV working group focused on distributed authoring

- **Additional methods**
 - PROPFIND retrieve properties (persisted as XML) from resource;
 retrieve collection structure (directory hierarchy)
 - PROPPATCH change and delete properties on resource
 - MKCOL create collection (directory)
 - COPY copy resource
 - MOVE move resource
 - LOCK put a lock on resource
 - UNLOCK remove a lock from resource

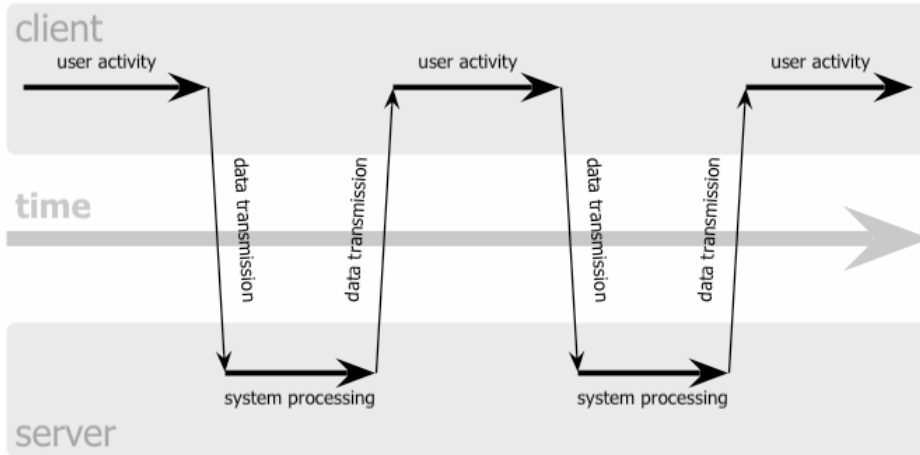
Ajax – Overview (1)

- **Asynchronous JavaScript and XML**
 - technology for interactive Web pages
 - asynchronous client/server communication using XMLHttpRequest

- Term “Ajax” coined by Jesse James Garrett in 2005, but many previous uses of similar approaches.



Ajax – Overview (2)



Classic model

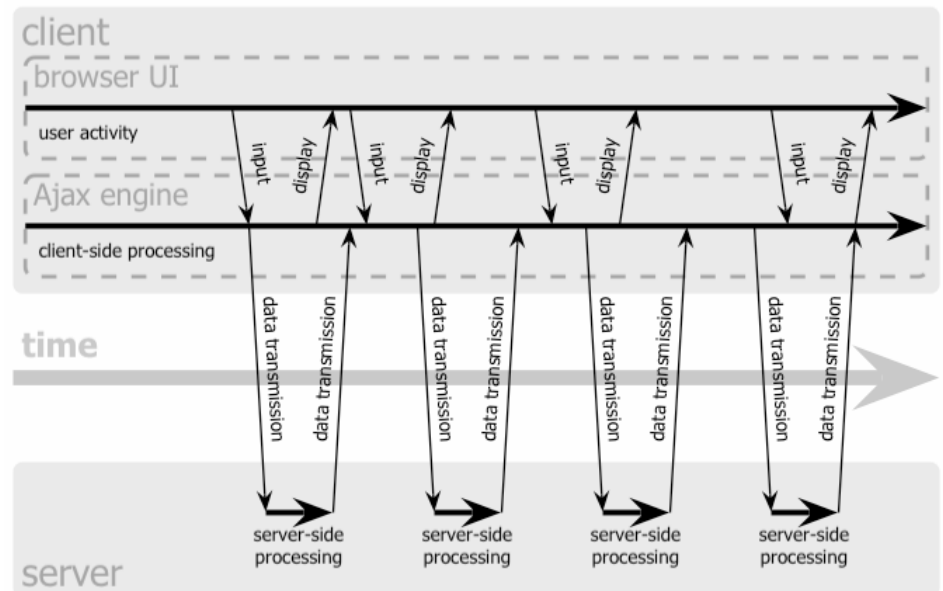
client/server communication interrupts user interaction with browser

synchronous communication

Ajax model

seemingly continuous user interaction with browser

concurrent asynchronous communication using callbacks



Ajax – Example (1)

```
<html>
<head>
<script>
  function submitForm() {
    ...
  }
</script>
</head>

<body>
  <form method="POST" name="ajax" action="">
    <input type="button" value="Submit" onclick="submitForm()" />
    <input type="text" name="dyn" value="" />
  </form>
</body>
</html>
```

Ajax – Example (2)

```
function submitForm() {
    var req = null;
    // browser-dependent req object: for Firefox, &c.
    if (window.XMLHttpRequest) req = new XMLHttpRequest();
    else if (window.ActiveXObject) req =
        new ActiveXObject(Microsoft.XMLHTTP); // ... for Microsoft Internet Explorer
    // callback function, called when status of req changes
    req.onreadystatechange = function() {
        if (req.readyState == 4) { // response ready
            if (req.status == 200) // status OK
                // change text of dyn field
                document.ajax.dyn.value = "Received:" + req.responseText;
            else
                document.ajax.dyn.value = "Error code " + req.status;
        }
    };
    // specify method, URL, and call type (asynchronous here) of req
    req.open("GET", "data.xml", true);
    req.send(null); // do request (optionally sending data, not used here)
}
```

Server Side Includes (SSI)

- **Small dynamic parts in mostly static Web pages**

- inclusion in HTML


- **SSI directives**

- `echo var` – include value of an environment variables
- `include file` – include file contents
- `exec cmd` – include output of a command
- `flastmod file, fsize file` – include date of last change/size of a file

- In Apache:

```
Options +Include
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
```

```
<html><head><title>SSI example</title></head>
<body>
  <!--#include virtual="/header.html"-->
  Welcome to <!--#echo var="SERVER_NAME"--><br/>
  You are connected from <!--#echo var="REMOTE_HOST"--> <br/>
</body></html>
```

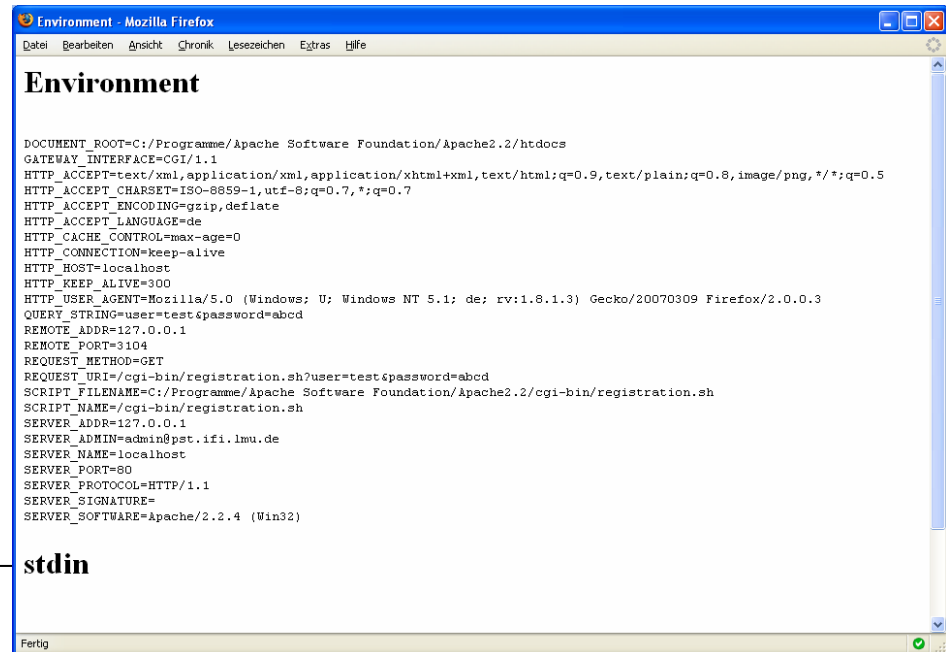
 **relative to server root**

Common Gateway Interface (CGI)

- **Interface between Web server and external processes**
 - parameter passing to process: [environment variables](#)
 - passing of Web resources from and to server ([stdin](#), [stdout](#))
- **History**
 - developed by NCSA 1993
 - CGI 1.2 specification ongoing work (since 1997)
 - <http://www.w3.org/CGI/>
- **Characteristics**
 - fine-grained security settings possible
 - new process started on every request
 - restricts interaction with Web server
 - remedied with FastCGI (keeps process, communication over sockets)

CGI: Example

```
#!/c:/cygwin/bin/bash
echo "Content-Type: text/html"
echo
echo "<html><head><title>Environment</title></head>"
echo "<body>"
echo "<h3>Environment</h3>"
echo "<pre>"
env | grep
    'HTTP_|DOCUMENT_|GATEWAY_|QUERY_|REMOTE_|REQUEST|SCRIPT|SERVER_'
echo "</pre>"
echo "<h3>stdin</h3>"
echo "<pre>"
cat -
echo "</pre>"
echo "<h3>Process</h3>"
echo "<pre>"
id
echo "</pre>"
echo "</body>"
```



```
Environment - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

Environment

DOCUMENT_ROOT=C:/Programme/Apache Software Foundation/Apache2.2/htdocs
GATEWAY_INTERFACE=CGI/1.1
HTTP_ACCEPT=text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
HTTP_ACCEPT_CHARSET=ISO-8859-1,utf-8;q=0.7,*;q=0.7
HTTP_ACCEPT_ENCODING=gzip,deflate
HTTP_ACCEPT_LANGUAGE=de
HTTP_CACHE_CONTROL=max-age=0
HTTP_CONNECTION=keep-alive
HTTP_HOST=localhost
HTTP_KEEP_ALIVE=300
HTTP_USER_AGENT=Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.8.1.3) Gecko/20070309 Firefox/2.0.0.3
QUERY_STRING=user=test&password=abcd
REMOTE_ADDR=127.0.0.1
REMOTE_PORT=3104
REQUEST_METHOD=GET
REQUEST_URI=/cgi-bin/registration.sh?user=test&password=abcd
SCRIPT_FILENAME=C:/Programme/Apache Software Foundation/Apache2.2/cgi-bin/registration.sh
SCRIPT_NAME=/cgi-bin/registration.sh
SERVER_ADDR=127.0.0.1
SERVER_ADMIN=admin@pst.ifi.lmu.de
SERVER_NAME=localhost
SERVER_PORT=80
SERVER_PROTOCOL=HTTP/1.1
SERVER_SIGNATURE=
SERVER_SOFTWARE=Apache/2.2.4 (Win32)

stdin
```

- **Keep information over several requests and responses**

- **Hidden fields**

```
<form method="get" action="shop">...  
<input type="hidden" name="cart" value="item 1">  
<input type="hidden" name="cart" value="item 2">  
</form>
```

- problem: session breaks, when user changes to non-related page

- **URL rewriting**

```
http://www.myshop.com/conditions.html?cart=item+1
```

- problem: hard coded links in pages (each link should be computed from current session data)

- **Cookies**

- each cookie up to 4KB; client keeps 300 cookies in total, 20 per server

■ Server-side components

- running in context of Web server ([servlet container](#))
- based on Java
 - platform independent
 - using Java's security mechanisms

<http://java.sun.com/products/servlet/>

■ History

- initially conceived by James Gosling (1995)
- Netscape's "server-side applets" (1996)
- Sun's Servlet 1.0 Specification (1997)
- current version: Servlet 2.5 Specification (May 2006)

■ Servlet container

- Sun's reference implementation based on Apache Tomcat

<http://tomcat.apache.org/>

```
package simple;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class HelloWorld extends javax.servlet.http.HttpServlet {
    protected void doGet(HttpServletRequest request,
                          HttpServletResponse response)
        throws ServletException, IOException {
        String name = request.getParameter("name");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><head><title>HelloWorld</title></head><body>");
        out.println("Hello " + name);
        out.println("</body>");
        out.close();
    }
}
```

■ Declaration of servlets

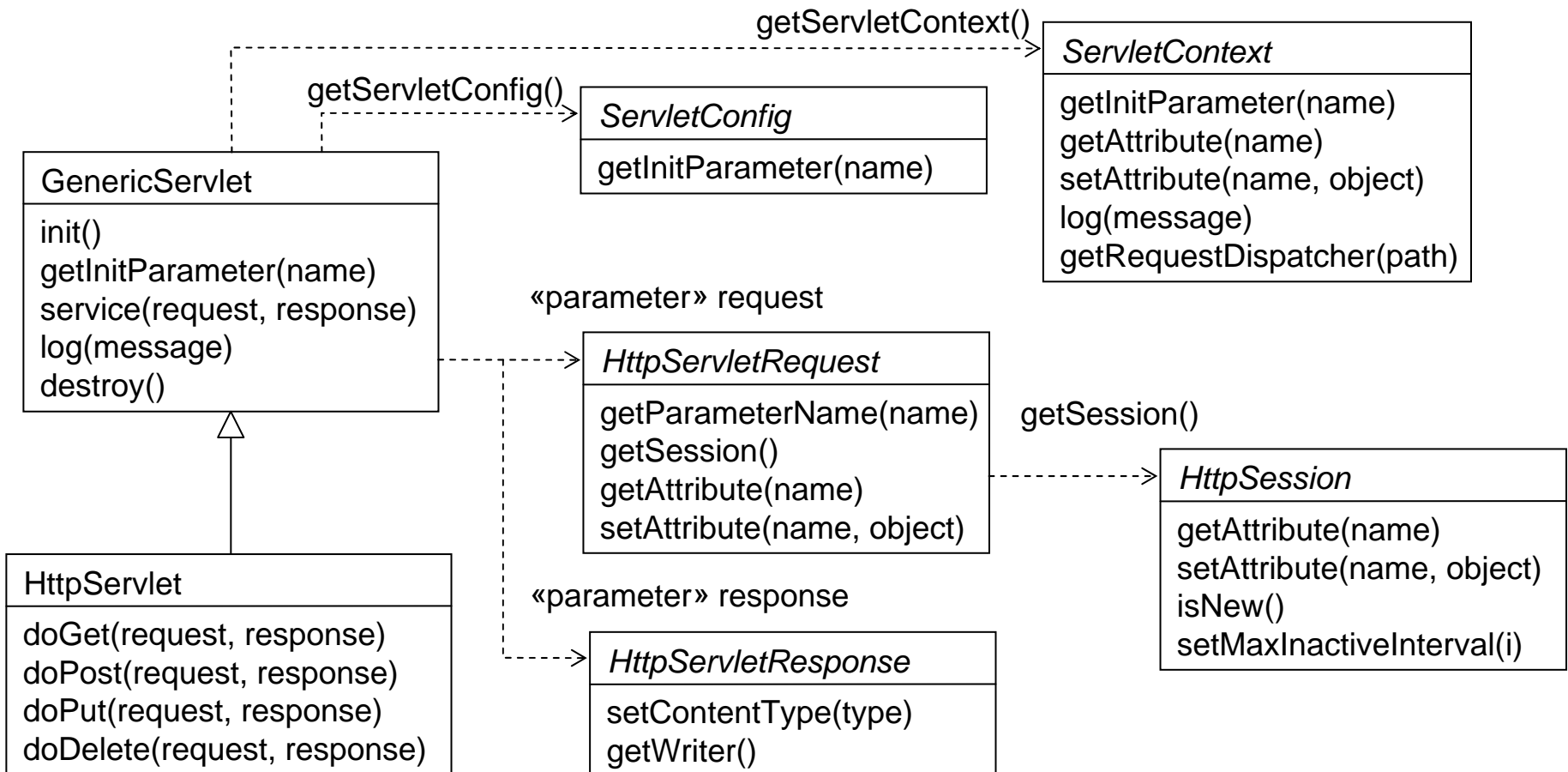
- access paths
- access rights
- initialization parameters
- ...

```
<web-app>
  <display-name>hello</display-name>
  <servlet>
    <description>HelloWorld</description>
    <display-name>HelloWorld</display-name>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>simple.HelloWorld</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

Servlets: File Structure

- **Document root** contains client-accessible data
 - HTML documents, images, ...
 - e.g., `http://localhost:8080/test/index.html`
- Sub-directory **WEB-INF** contains servlet data, not directly accessible by clients
 - `classes`: classes, in particular those extending `HttpServlet`
 - `lib`: jar archives, infrastructure libraries
 - `web.xml`: Web application deployment descriptor
- Particular **installation** depends on servlet container
 - Apache Tomcat
 - either default installation in sub-directory of `webapps` (directory name is base URL of Web application)
 - or special configuration (base URL, logs, ...) via `conf/server.xml`

- `javax.servlet.*`, `javax.servlet.http.*`



Servlets: Life Cycle

■ Servlet control by overriding

Hollywood principle: “Don’t call us, we will call you”

- `void init(ServletConfig config);`
- `void doDelete(HttpServletRequest request, HttpServletResponse response);`
- `void doGet(HttpServletRequest request, HttpServletResponse response);`
- `void doPost(HttpServletRequest request, HttpServletResponse response);`
- `void doPut(HttpServletRequest request, HttpServletResponse response);`
- `public void destroy();`

■ Life cycle

- call on `init()` on loading the servlet
- `doMethod()` on client requests
- call of `destroy()` on termination

Servlets: Synchronisation

- **Servlets in general called concurrently by several users**
 - per request a new thread is started
 - **servlets may share data** (static variables, `ServletContext` attributes, ...)
- **Synchronization in Java**
 - `synchronized` methods/statements
 - `wait/notify/notifyAll`
 - Java concurrency library `java.util.concurrent`

- **Each user of a server has at most one session object**
 - mechanism for session handling depends on server
 - method `getSession(boolean create)` of `HttpServletRequest` delivers an `HttpSession` object
 - either having been sent by request;
 - or creating a new one and sending it with response
- **Session specific data can be stored in session object**
 - `session.setAttribute(attrName, attrValue)`
 - `Object o = session.getAttribute(attrName)`
- **Sessions are assigned a limited validity**
 - as specified in deployment descriptor
 - explicit invalidation by `session.invalidate()`
 - for requests with invalid session, a new session object is created

```
<web-app> ...
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Administration</web-resource-name>
      <url-pattern>/hello</url-pattern>
    </web-resource-collection>
    <auth-constraint> ← required user role
      <role-name>admin</role-name>
    </auth-constraint>
    <user-data-constraint>
      <transport-guarantee>NONE</transport-guarantee> ← no encryption (also: INTEGRAL, CONFIDENTIAL)
    </user-data-constraint>
  </security-constraint>
  <login-config>
    <auth-method>FORM</auth-method> ← form based (also: BASIC, DIGEST)
    <realm-name>Administration</realm-name>
    <form-login-config>
      <form-login-page>/login.html</form-login-page>
      <form-error-page>/error.html</form-error-page>
    </form-login-config>
  </login-config>
</web-app>
```

```
<tomcat-users>
  <role rolename="admin"/>
  <user password="a7!2"
    roles="admin"
    username="root"/>
</tomcat-users>
```

Java Server Pages (JSPs)

■ Server-side scripting

- **embedding of scripts** (or other software components) into HTML
 - close to HTML (reuse of tools and knowledge)
- examples: Server-Side JavaScript, ASP, JSP, PHP, etc.

■ Java Server Pages (JSPs)

<http://java.sun.com/products/jsp/index.html> (current version: JSP 2.1)

- HTML with embedded Java scripts
- tag extensions by tag libraries
 - JSP Standard Tag Library (JSTL)
- **transparent translation into servlets**

```
<%@ page language="java" pageEncoding="ISO-8859-1"
      contentType="text/html; charset=ISO-8859-1"
      import="java.util.*" %>
<!-- Initialise local variable clock --%>
<% GregorianCalendar clock = new GregorianCalendar(); %>
<!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=ISO-8859-1">
<title>Time and date</title></head>
<body>
<%! private int accessCount = 0; %>
Host: <%= request.getRemoteHost() %><br>
<% String queryData = request.getQueryString();
      out.print("Attached GET data: " + queryData); %><br>
Current time: <%= clock.getTime() %><br>
Accesses: <%= accessCount++ %>
</body></html>
```

JSP directive

JSP comment

JSP declaration

JSP expression

JSP scriptlet

JSPs: Compilation

■ Generation of servlet

- service method generated from template text, expressions, and scriptlets
- provides “implicit” objects (out, request, response, ...)
- names of the form `[_]jsp[x]_*` reserved

```
<% if (a > b) %>  
a = <%= a%>  
<% else %>  
b = <%= b%>
```



```
if (a > b)  
out.print("a = ");  
out.print(a);  
else  
out.print("b = ");  
out.print(b);
```

error

```
<?xml version="1.0" encoding="utf-8"?>
<jsp:root xmlns="http://www.w3.org/1999/xhtml"
          xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0">
<jsp:output doctype-root-element="html"
            doctype-public="-//W3C//DTD XHTML 1.1//EN"
            doctype-system="http://www.w3c.org/TR/xhtml11/DTD/xhtml11.dtd"/>
<jsp:directive.page contentType="text/html; charset=utf-8"
language="java"/>
<html xmlns="http://www.w3.org/1999/xhtml">
<jsp:declaration>private int accessCount = 0;</jsp:declaration>
<jsp:text>Host: </jsp:text>
<jsp:expression>request.getRemoteHost()</jsp:expression><br/>
<jsp:scriptlet>
    String queryData = request.getQueryString();
    out.print("Attached GET data: " + queryData);
</jsp:scriptlet><br/>
<jsp:expression>accessCount++</jsp:expression>
</html>
</jsp:root>
```

References

- Jeffrey C. Jackson. *Web Technologies*. Pearson Prentice Hall, Upper Saddle River, 2007.
- Heiko Wöhr. *Web-Technologien*. dpunkt.verlag, Heidelberg, 2004.
- SELFHTML e.V. (HTML, CSS, JavaScript, DOM, XML, CGI, ...)
<http://de.selfhtml.org/>
- Sun's J2EE 1.4 tutorial (Servlets, JSP)
<http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html>
- James Gillies, Robert Cailliau. *How the Web was Born*. Oxford University Press, Oxford, 2000.