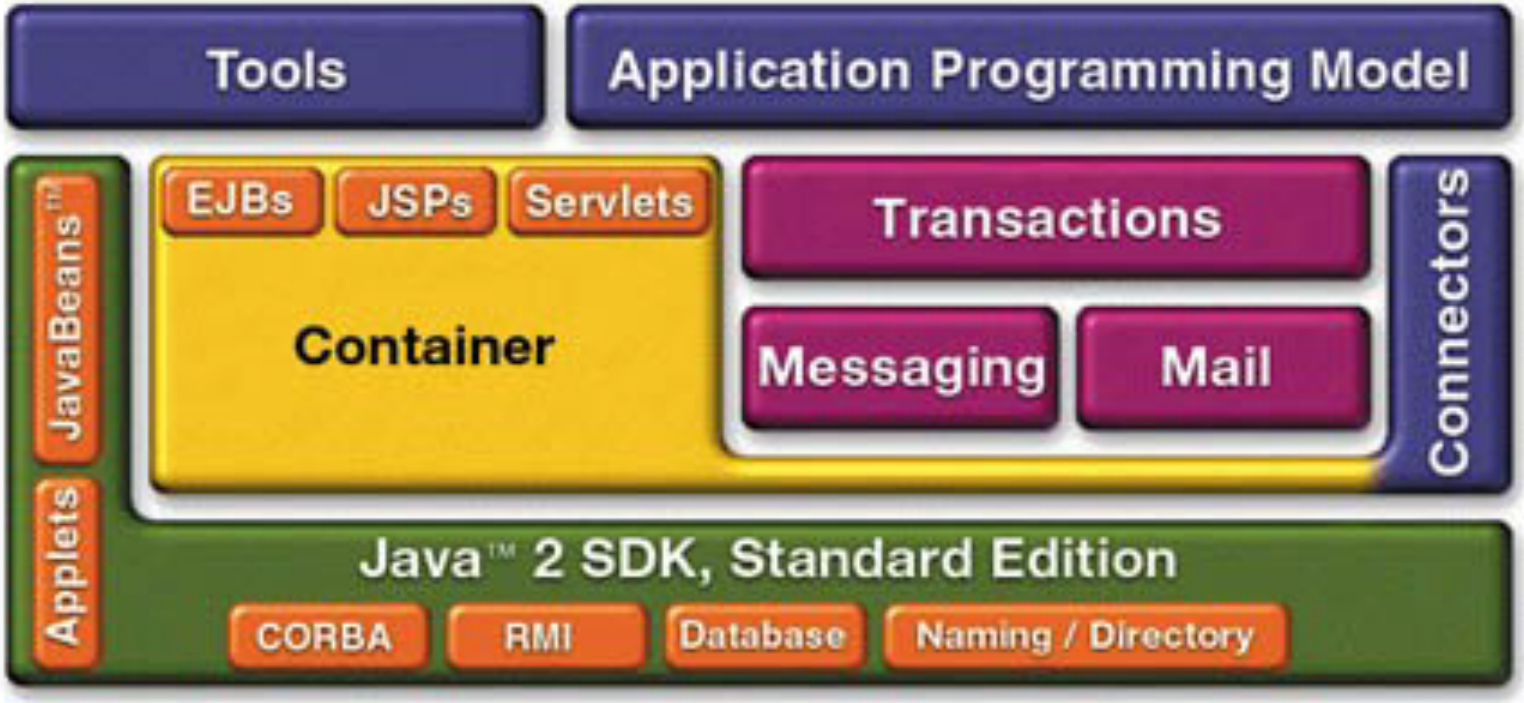
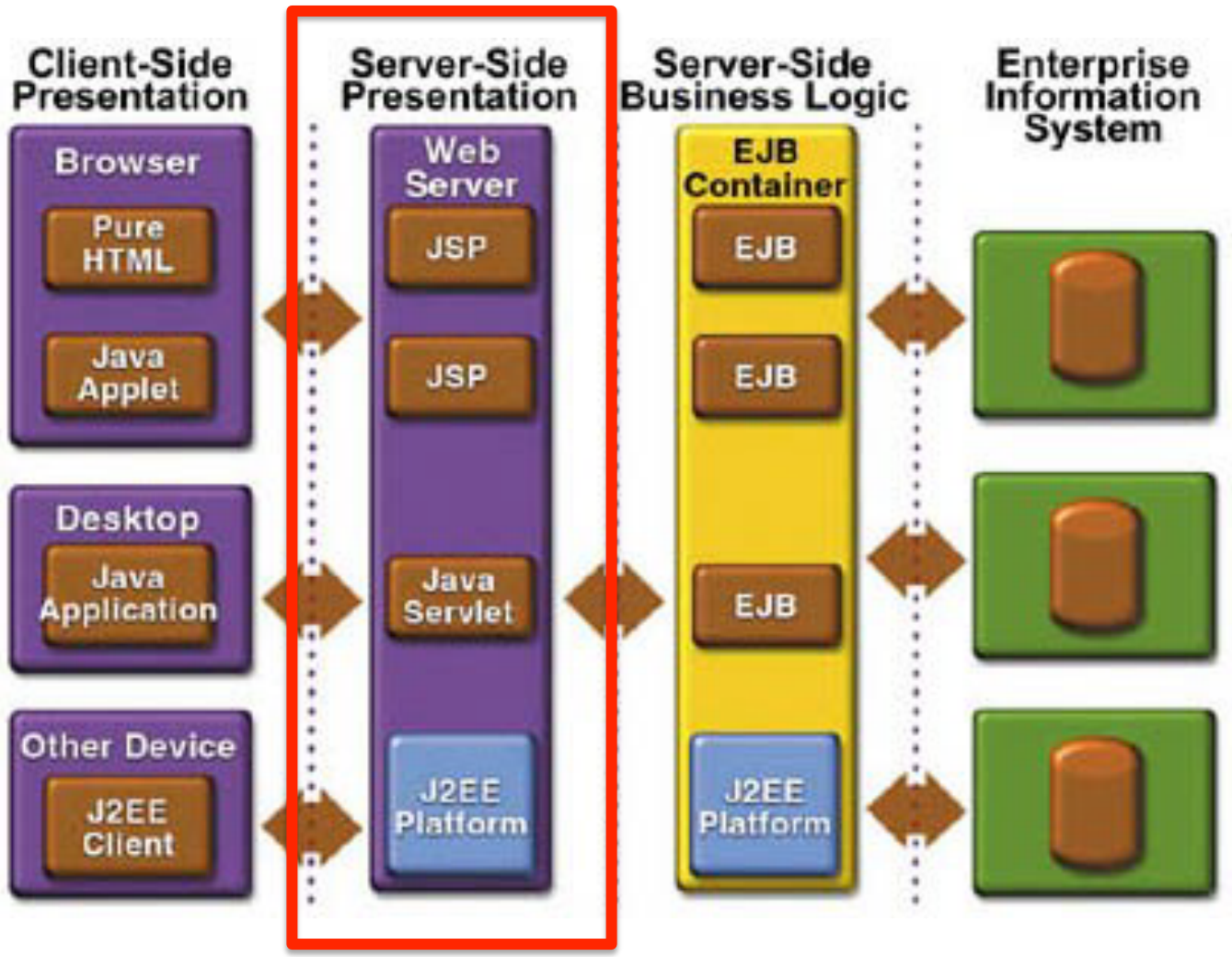
Servlety

Kontext


Tools
Application Programming Model
JavaBeans™
Applets
EJBs
JSPs
Servlets
Container
Transactions
Messaging
Mail
Connectors
Java™ 2 SDK, Standard Edition
CORBA
RMI
Database
Naming / Directory

Kontext


Client-Side Presentation
Server-Side Presentation
Server-Side Business Logic
Enterprise Information System
Browser
Pure HTML
Java Applet
Desktop
Java Application
Other Device
J2EE Client
Web Server
JSP
JSP
Java Servlet
J2EE Platform
EJB Container
EJB
EJB
EJB
J2EE Platform

Http protokol, request response

```
$ telnet screwdriver 80
GET/HTTP/1.0
```

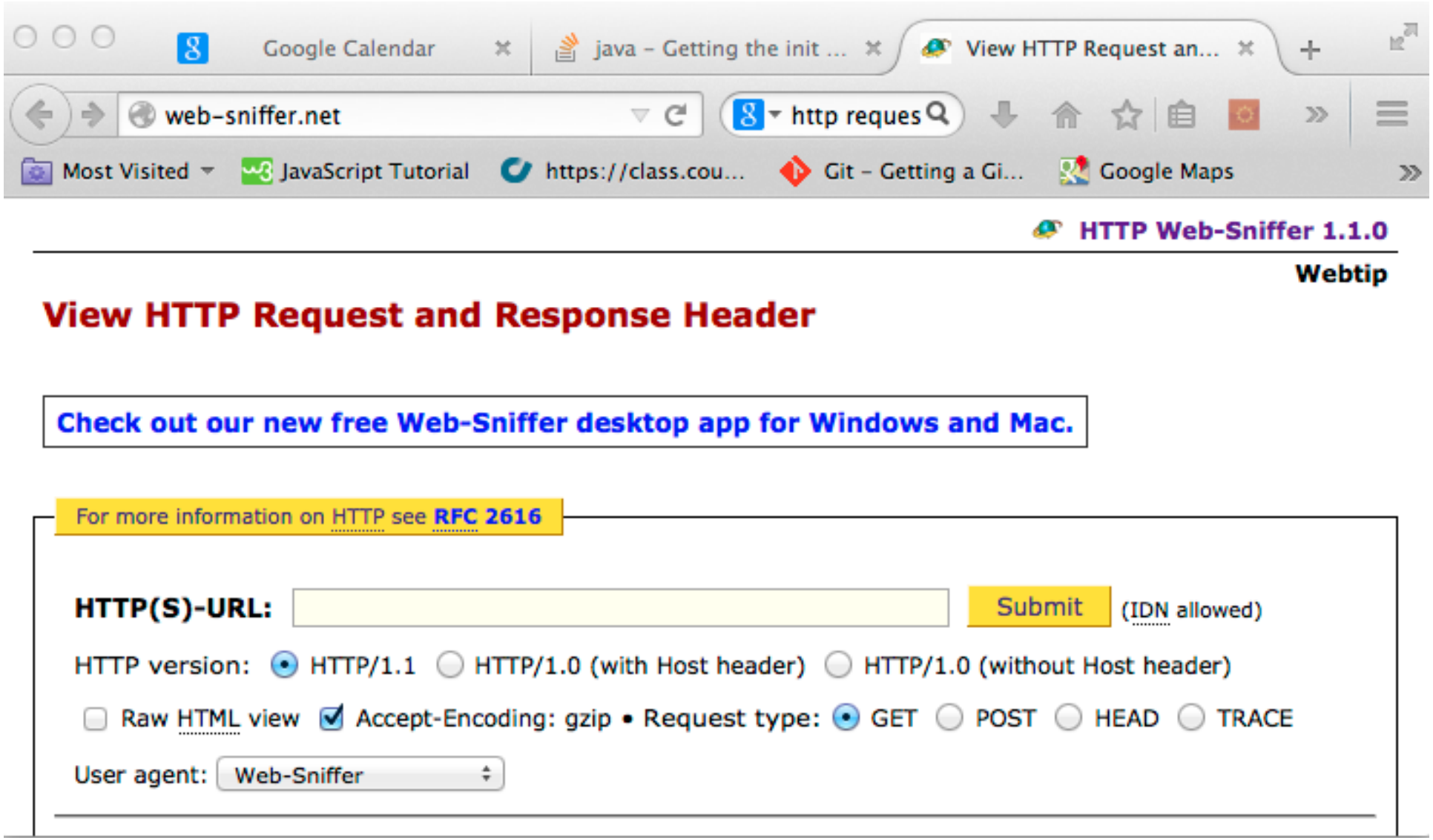
request

response

```
HTTP/1.1 200 OK
Date: Tue, 29 Sep 2009 21:09:43 GMT
Server: Apache/2.2.13 (Debian)
Content-Length: 863
Content-Type: text/html
<HTML>
<HEAD>
<TITLE>Short Index of
...
```

Snadné monitorování http protokolu

web-sniffer.net



www.google.com

HTTP Request Header

```
Connect to 173.194.113.50 on port 80 ... ok

GET / HTTP/1.1[CRLF] Host: www.google.com[CRLF]
Connection: close[CRLF]
User-Agent: Web-sniffer/1.1.0 (+http://web-
sniffer.net/)[CRLF]
Accept-Encoding: gzip[CRLF]
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7[CRLF]
Cache-Control: no-cache[CRLF]
Accept-Language: de,en;q=0.7,en-us;q=0.3[CRLF]
Referer: http://web-sniffer.net/[CRLF] [CRLF]
```

www.google.com

HTTP Response Header

Name	Value
Status: HTTP/1.1 302 Found	
Cache-Control:	private
Content-Type:	text/html; charset=UTF-8
Location:	http://www.google.de/?gfe_rd=cr&ei=ILsrVMquJauI8Qfm74C4Cw
Content-Length:	258
Date:	Wed, 01 Oct 2014 08:28:16 GMT
Server:	GFE/2.0
Alternate-Protocol:	80:quic,p=0.01
Connection:	close

www.google.com

Content (0.25 KiB)

```
<HTML>
   <HEAD>
      <meta http-equiv="content-type"
                          content="text/html;charset=utf-8">
      <TITLE>302 Moved</TITLE>
   </HEAD>
   <BODY>
      <H1>302 Moved</H1>
      The document has moved
      <A HREF="http://www.google.de/?
                gfe_rd=cr&amp;ei=ILsrVMquJauI8Qfm74C4Cw">
         here
      </A>.
   </BODY>
</HTML>
```

HTTP protocol methods

- OPTIONS
- **GET**
- HEAD
- **POST**
- PUT
- DELETE
- TRACE
- CONNECT

```
$ telnet screwdriver 80
GET/HTTP/1.0
```

GET

- parameters embedded in the URL
- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests should be used only to retrieve data

/test/demo_form.asp**?name1=value1&name2=value2**

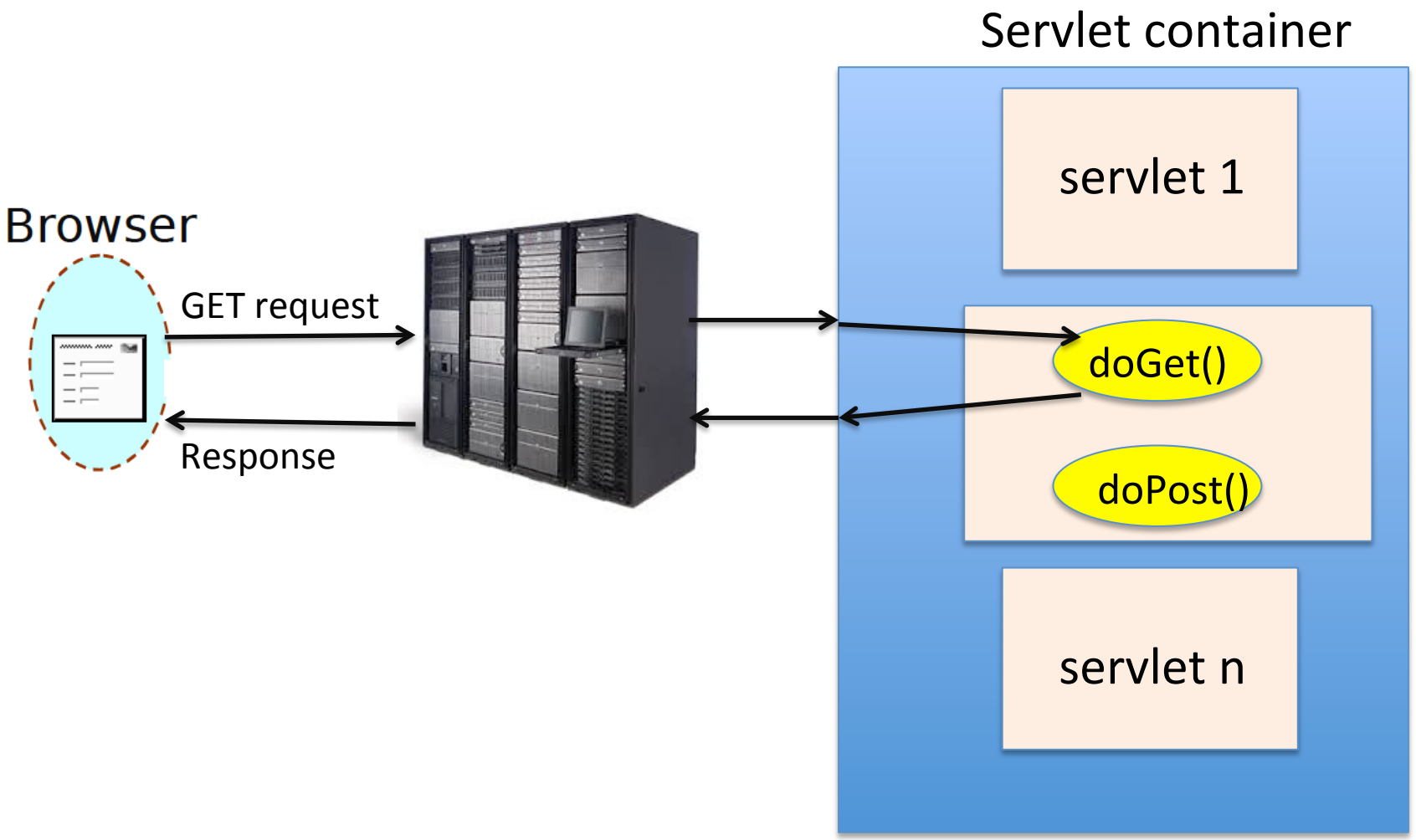
POST

Query strings (name/value pairs) sent in the message body:

```
POST /test/demo_form.asp HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

Servlet


Servlet container
servlet 1
Browser
GET request
Response
doGet()
doPost()
servlet n

Servlet

- proces Javy je stále spuštěný, takže se ušetří spouštění
- thread pool pro zpracování požadavků
- sdílení prostředků (paměť)
- automatické sledování session pomocí jsessionid v cookies
- přístup ke všem knihovnám Javy
- debugování (buď běží server přímo v debug režimu nebo vzdáleně)
- -POZOR: servlet je sdílen mezi více požadavky!

Servlet container

- Co pro nás dělá kontejner?
  - spojení TCP/IP
  - zpracování HTTP protokolu
  - zpracování parametrů (url)
  - správa zdrojů (thread pooly)
- Knihovny
  - obecný servlet – javax.servlet.*
  - HTTP servlet – javax.servlet.http.*

FirstServlet

```
public class FirstServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body></html>");
        out.close();
    }
}
```

Jak se liší `doGet()` a `doPost()` ?

Jak se liší `doGet()` a `doPost()` ?

- Často:

```
public class AServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
  { processRequest(request, response); }


  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
  { processRequest(request, response); }

  public void processRequest(HttpServletRequest request,
                             HttpServletResponse response)
  { processRequest(request, response); }

}
```

web.xml – servlet, servlet-mapping

```
<servlet>
   <servlet-name>FirstServlet</servlet-name>
      <servletclass>cz.cvut.fel.wpa.FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>FirstServlet</servlet-name>
   <url-pattern>/FirstServlet</url-pattern>
</servlet-mapping>
```

web.xml – servlet parameters I

```
<servlet>
   <servlet-name>FirstServlet</servlet-name>
   <servletclass>cz.cvut.fel.wpa.FirstServlet</servlet-class>
   <init-param>
      <param-name>branding</param-name>
      <param-value>CVUT FEL</param-value>
   </init-param>
</servlet>
```

web.xml – servlet parameters II

java.lang.Object
    javax.servlet.**GenericServlet**
        javax.servlet.http.**HttpServlet**

HttpServlet implements

```
public java.lang.String
getInitParameter(java.lang.String name)
```

Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.

Struktura webové aplikace

- WEB-INF
  - web.xml – konfigurace
  - libs – knihovny
  - classes – třídy (model, servlety, filtry, ...)
  - všechny soubory jsou nepřístupné přes HTTP
- další soubory jsou přístupné
- typicky se tato struktura zabalí do ZIPu a pojmenuje se WAR (Web ARchive)
- upload do kontejneru se jmenuje deployment
  - pro servlety stačí upload a inicializace kontejnerem

Lifecycle

- na začátku se zavolá `init()`
- při každém požadavku se zavolá

  `service(req, res)`
- při úklidu se zavolá `destroy()`
- tyto metody nemají být volány uživatelem
- „one instance per servlet definition“

Kontext

- Možnost ukládat data mezi requesty:
  - application: getServletContext()
  - session: request.getSession()
  - request
  - page (JSP)
- Příklad

```
String id = request.getParameter(„id“);
User login
     = (User) request.getSession().getAttribute("user");
```

Client Session State

- Pokud je server zcela bezestavový (stateless), musí se stav sešny přenášet mezi klientem a serverem – může jít o velký objem dat
- Pro tenkého webového klienta – 3 možnosti
  - **Parametry v URL** – negativa: (i) omezená délka URL, (ii) parametry zobrazeny v URL stránky, (iii) problémy s bookmarkováním stránky
  - **Skrytá vstupní pole** (hidden fields) – skryta ve smyslu, že nejsou na stránce zobrazena
  - **Cookies** – problémy: (i) uživatel může cookies zakázat, (ii) cookies organizovány po doménách – co když více aplikací v doméně

HTTP cookies

```
GET/index.html HTTP/1.0
Host: www.example.org
```

request

HTTP cookies

```
GET/index.html HTTP/1.0
Host: www.example.org
```

request

response

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: name=value
Set-Cookie: name2=value2; Expires=Wed, 09 Jun 2021 10:18:14 GMT
...
```

HTTP cookies

```
GET/index.html HTTP/1.0
Host: www.example.org
```

request

response

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: name=value
Set-Cookie: name2=value2; Expires=Wed, 09 Jun 2021 10:18:14 GMT
...
```

request

```
GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: name=value; name2=value2
Accept: */*
```

Client Session State

- Udržování stavu na klientovi může být výhodou při clusteringu, failover
- Obsahuje-li stav citlivá data, měl by být šifrován => režie
- Server by měl příchozí data revalidovat, aby nebyla poškozena jejich konzistence
- Stav na klientovi v minimální variantě obsahuje sessionId – odkazující na session, která drží stav na serveru
- Session steeling – uživatel modifikuje sessionID tak, aby získal session někoho jiného

Server Session State

- Stav “sešny” typicky reprezentován
  - Binárně (BLOB – Binary Large OBject) – problém s verzováním
  - Textově (typicky XML)
- Udržován
  - Lokálně
    - v paměti aplikačního serveru
      - HashMap, jejímž klíčem je sessionID
    - Ve filesystému aplikačního serveru
    - V lokální DB aplikačního serveru
    - Problém v případě clusteru aplikačních serverů, při failover/switchover
  - Ve sdílené databázi
    - Stav uložen nestrukturovaně (např. jako BLOB) – v opačném případě viz Database Session State
    - Umožňuje clusterování, failover/switchover
    - Musí se řešit zapomínání stavu ukončených a “vyprchaných” sessions

Database Session State

- Speciální případ předchozího “server session state” – stav uložen jako strukturovaná data
- Dva případy
  - V “ostrých” tabulkách
    - Přidat sloupec SessionID, if null => ostrá data, if not null => pracovní data rozpracované sešny
    - Rozpracovaná (pending) data nemusí být konzistentní => nemusí být na ně aplikovatelná integritní omezení
  - V “pending” tabulkách
- Mazat data nedokončených (abandoned) a přerušených (cancelled) sessions

Získání informací o spojení

- klient
  - request.getRemoteAddr()
  - request.getRemoteHost()
- server
  - request.getServerName()
  - request.getServerPort()
  - request.getContextPath()
  - PROČ to potřebujeme vědět???

- isSecure, isUserInRole, getAuthType, getCookies, getHeaderNames ...

Filtry

- Umožňují vstoupit mezi klienta a servlet a změnit data
  - Komprese, kódování obrázků, ...
  - Pozměňování výstupu servletu (kódování, logo, ...)
  - Bezpečnost (odmítnutí přístupu)
  - Integrace web aplikací (SSO)
  - ...
- Filtry se za sebe řetězí v tom pořadí, jak jsou definovány ve web.xml.

Filters

`public interface` **`Filter`**

Method Summary	
`void`	**`destroy`**`()`
`void`	**`doFilter`**`(ServletRequest request, ServletResponse response, FilterChain chain)`
`void`	**`init`**`(FilterConfig filterConfig)`

Filters perform filtering in the `doFilter` method. Every Filter has access to a `FilterConfig` object from which it can obtain its initialization parameters, and a reference to the `ServletContext` which it can use, for example, to load resources needed for filtering tasks.

Filters

Examples that have been identified for this design are:

1. Authentication Filters
2. Logging and Auditing Filters
3. Image conversion Filters
4. Data compression Filters
5. Encryption Filters
6. Tokenizing Filters
7. Filters that trigger resource access events
8. XSL/T filters
9. Mime-type chain Filter

Příklad filtru

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
              throws IOException,
                     ServletException {
   ...
   chain.doFilter(wrappedRequest,
                  wrappedResponse);
   ...
}
```

Konfigurace filtru ve web.xml

```
<filter>
   <filter-name>F1</filter-name>
   <filter-class>cz.cvut.fel.filters.F1</filter-class>
</filter>

<filter-mapping>
   <filter-name>F1</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

Error page configuration in web.xml

```
<error-page>
    <exception-type>
        exception.BookNotFoundException
    </exception-type>
    <location>/errorpage1.html</location>
</error-page>
```

Servlet 3.0 annotations

- konfigurace ne nutně pomocí web.xml

```
@WebServlet(name="CalculatorServlet", urlPatterns={"/calc","/getVal"})
public class CalculatorServlet extends HttpServlet{

   public void doGet(HttpServletRequest req, HttpServletResponse res) {
      ...
   }
...
}
```