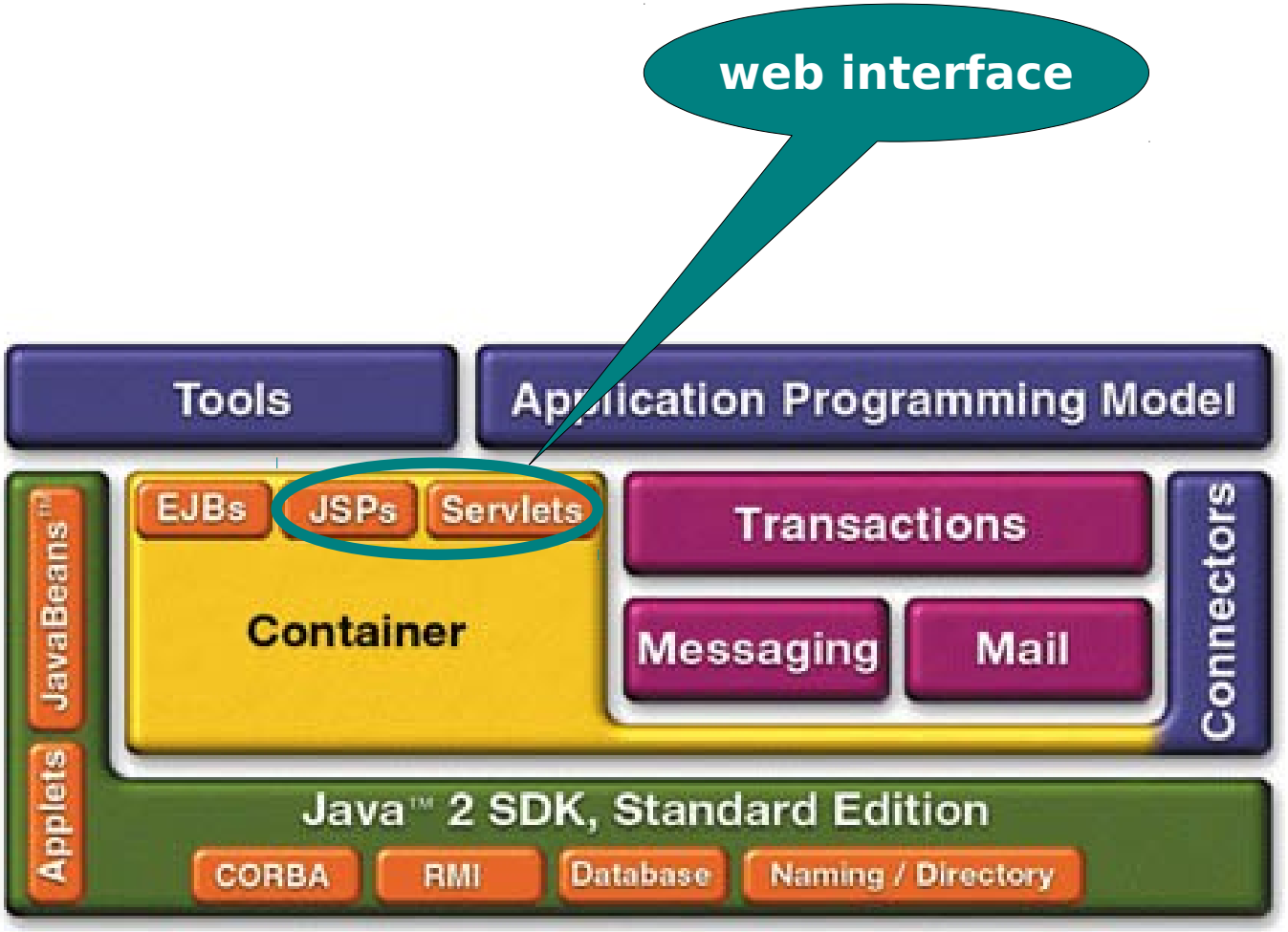
Servlety

Petr Aubrecht (CA)

while (!asleep()) $sheep++;

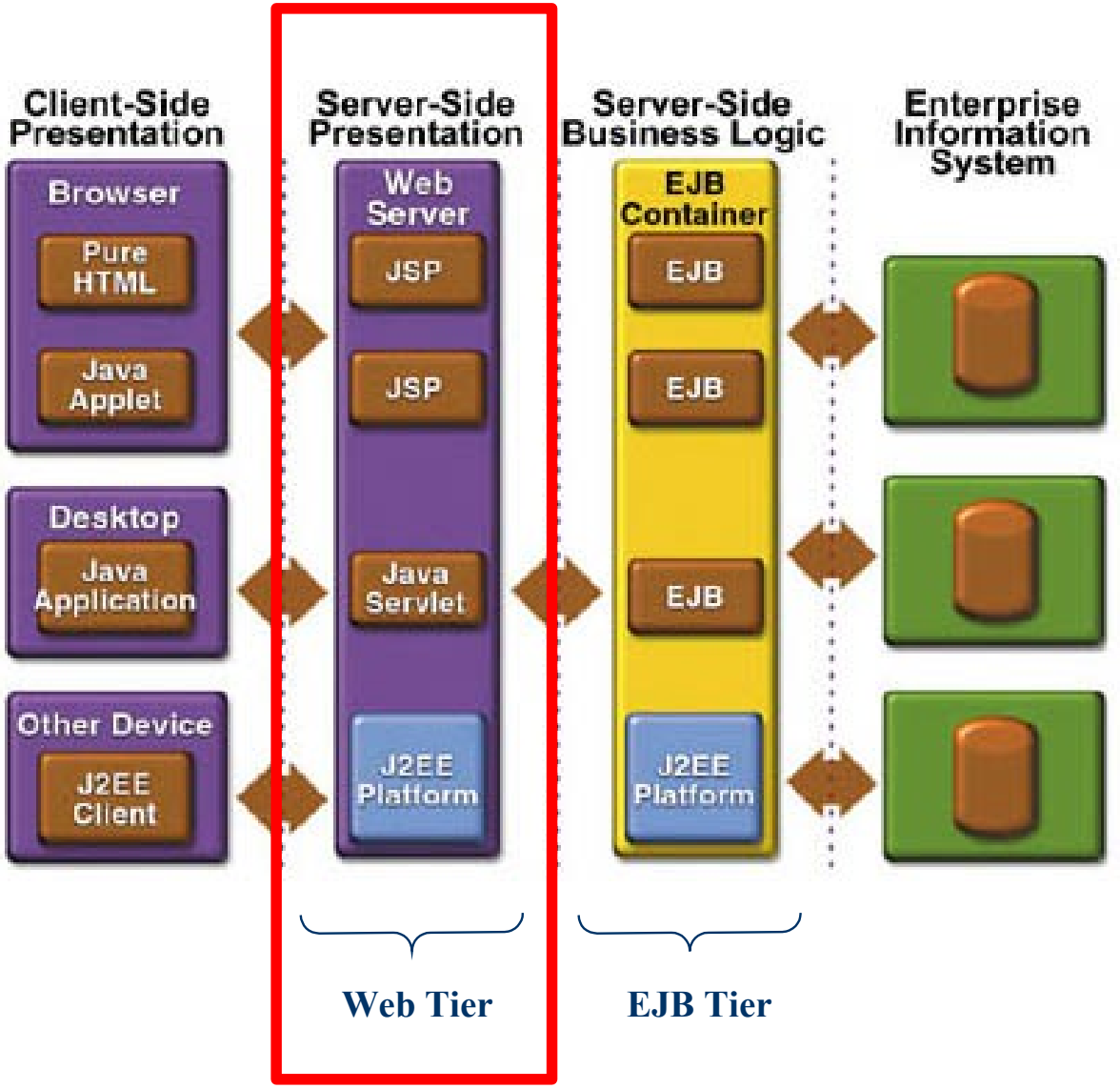
Servlets and JSP

- kontext:



Process Flow

- layers



Request, reponse reálně

$ telnet screwdriver 80

GET / HTTP/1.0

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

GET & POST

- GET posílá parametry v URL
  - vhodné pro získávání dat, v parametrech pouze ID
  - .../getarticle?id=123
- POST slouží k odesílání formulářových dat
  - lze odeslat libovolně mnoho dat
  - např. upload souboru

Znáte CGI?

- CGI je interface pro interaktivní aplikace
- parametry jsou předány v proměnných prostředí
- každý požadavek spustí nový proces
- implementace v C/C++, Perl, Python
- poměrně pracné
- problémové ve všech směrech: obtížné ladění, závislé na platformě, bezpečnost, ...
- výkon?

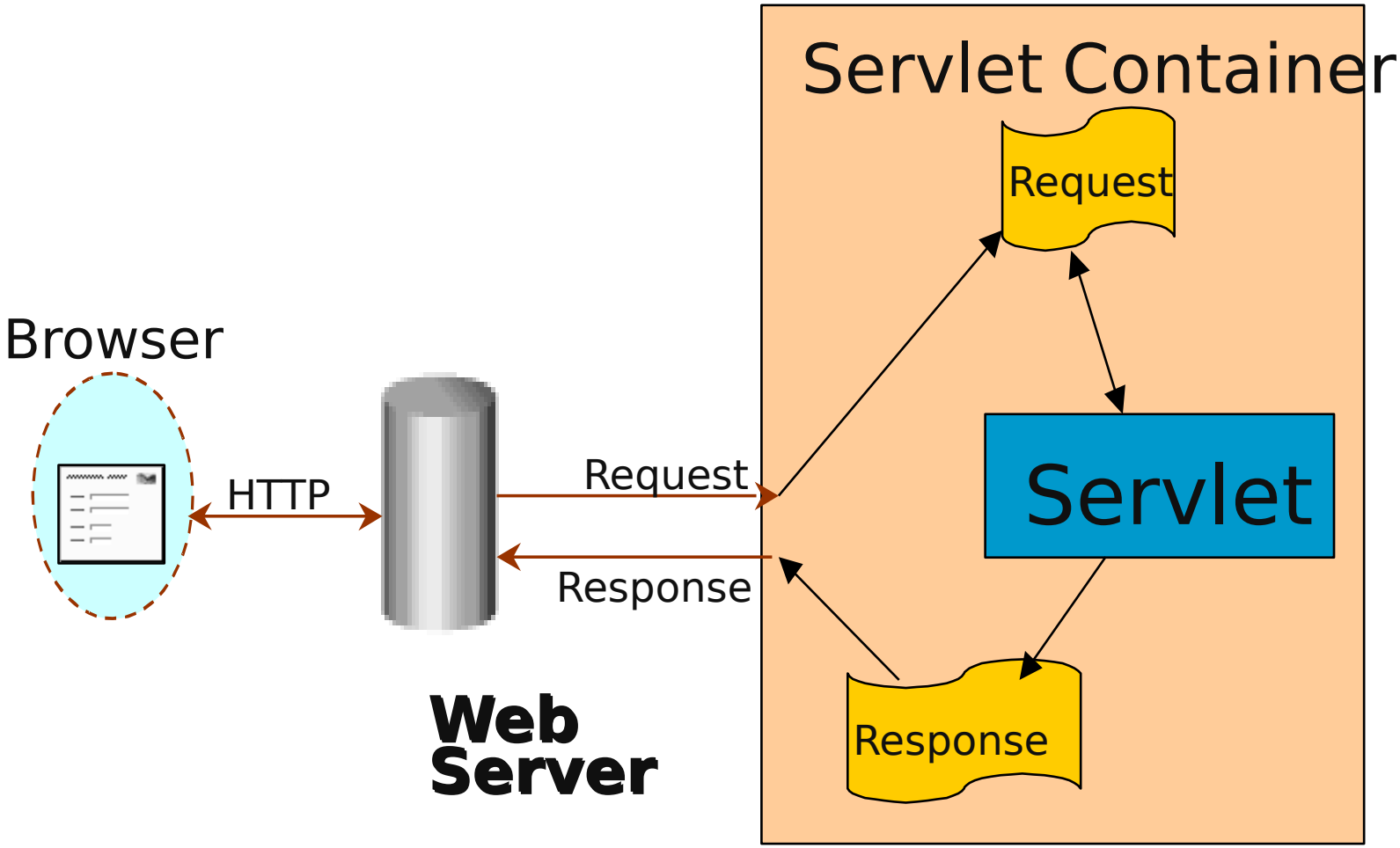
Servlet

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

Servlet kontejner

- 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.*)

Request, response



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();
  }
}
```

Servlet – co obsahuje

- v podstatě pouze doGet a doPost, které jsou co do zpracování identické (a často se spojují)
- HttpServletRequest request – informace o requestu
- HttpServletResponse response – nastavujeme odpověď
- výstup píšeme do response.getWriter (což je nečekaně Writer)

web.xml – servlety

```
<servlet>
    <servlet-name>FirstServlet</servlet-name>
    <servlet-
class>com.ca.eja.servlets.FirstServlet</servlet-class>
</servlet>

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

web.xml – parametry servletů

```
<servlet>
    <servlet-name>FirstServlet</servlet-name>
    <servlet-
class>com.ca.eja.servlets.FirstServlet</servlet-class>
        <init-param>
            <param-name>branding</param-name>
            <param-value>CA</param-value>
        </init-param>
</servlet>
```

Struktura webové aplikace

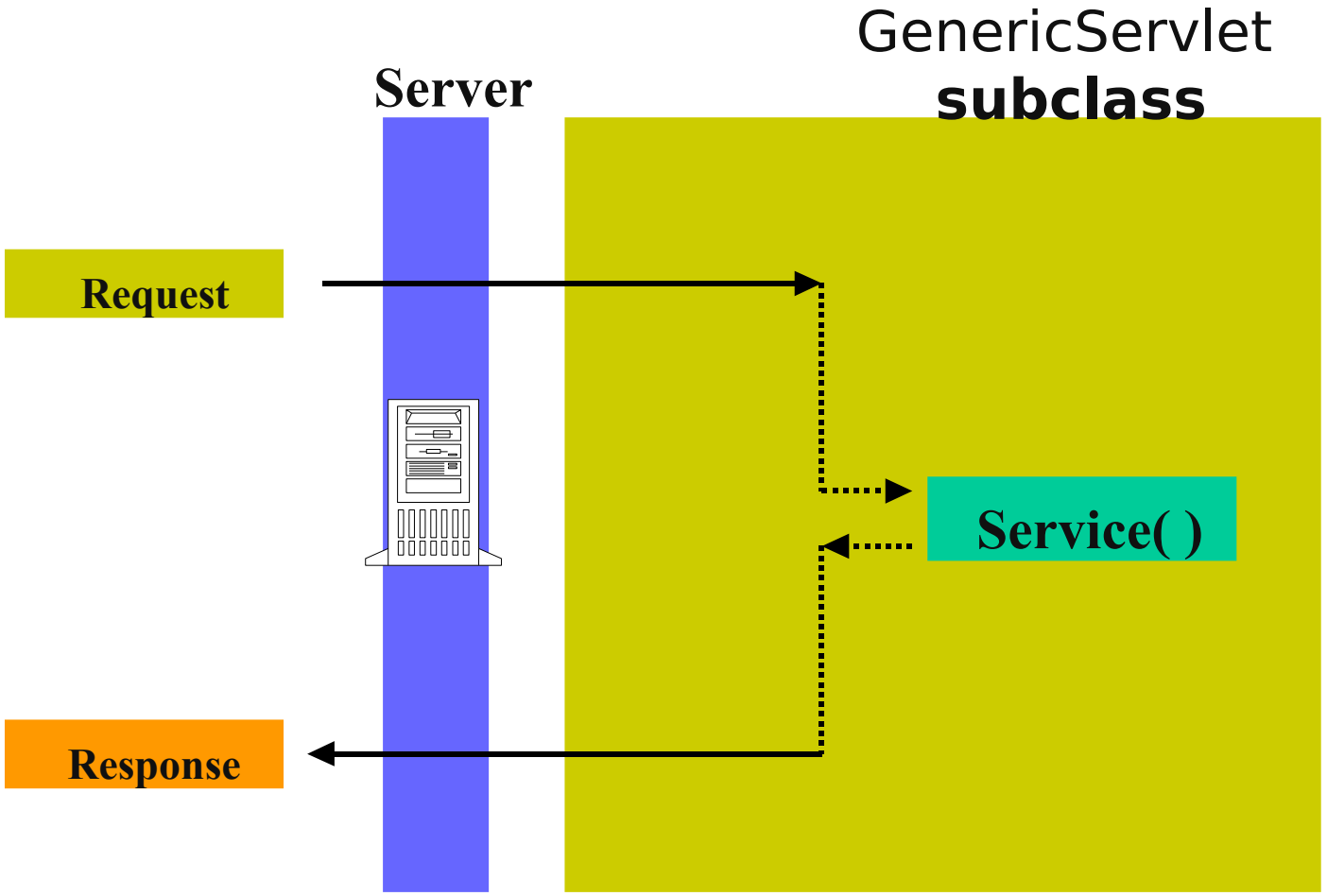
- WEB-INF
  - web.xml – konfigurace
  - libs – knihovny
  - classes – třídy (model, servlety, filtry, ...)
  - všechny soubory jsou nepřístupné přes HTTP
- další soubory jsou přístupné
- typicky se tato struktura zabalí do ZIPu a pojmenuje se WAR (Web ARchive)
- upload do kontejneru se jmenuje deployment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

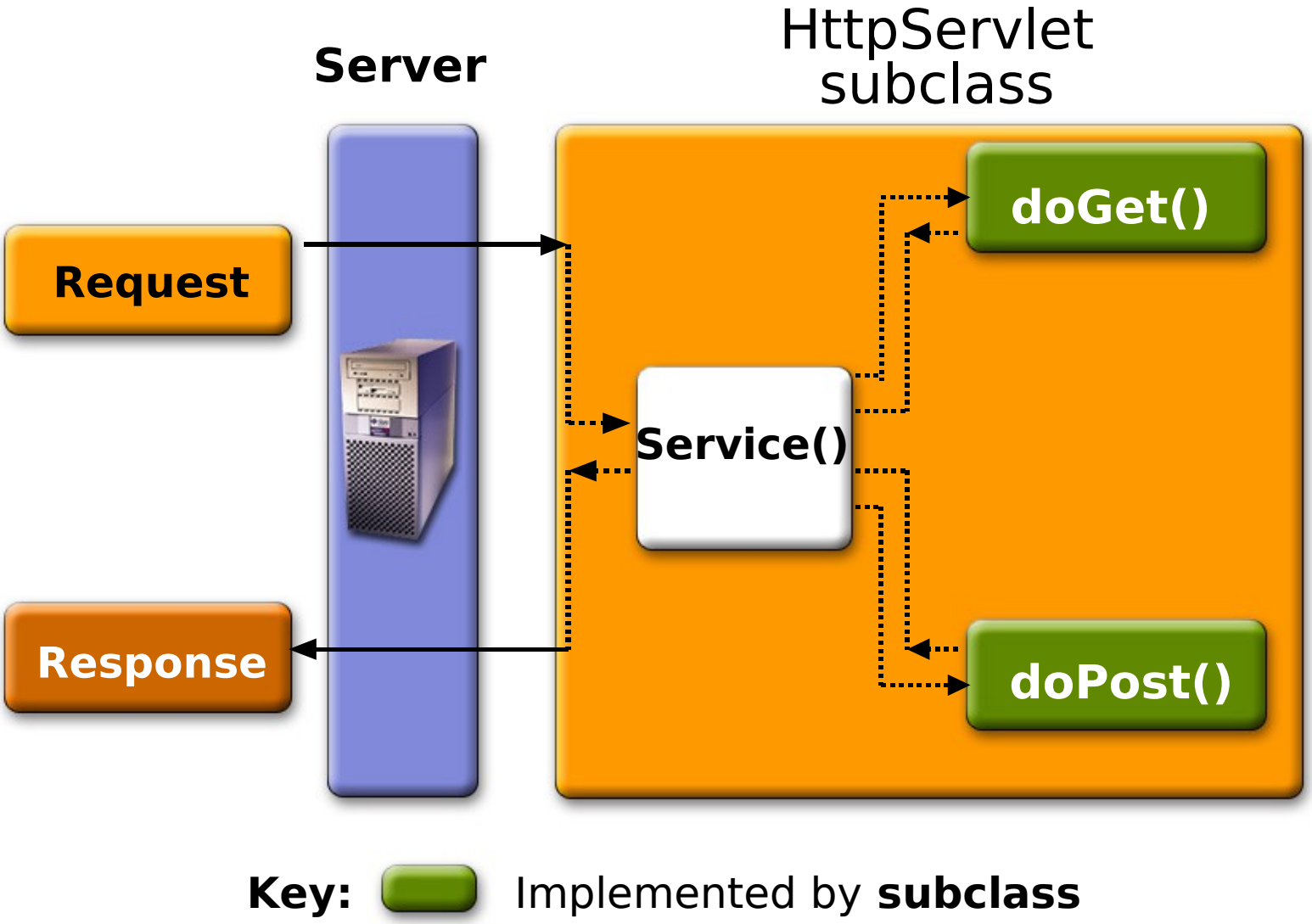
- tyto metody nemají být volány uživatelem
- „one instance per servlet definition“

Servlet Processing



Key: Implemented by **subclass**

HttpServlet Processing



Kontext

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

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

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.

Příklad filtru

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

Konfigurace filtru

```
<filter>
    <filter-name>F1</filter-name>
    <filter-class>com.ca.eja.filters.F1</filter-class>
</filter>
<filter-mapping>
    <filter-name>F1</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Error page

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

Drobnosti

- HTTP 1.1
  - perzistentní connection, nepřerušuje se po každém requestu
- HTTP headers
  - if-modified-since – optimalizace přenosu
  - referer – odkazující stránka
  - user-agent – identifikace browseru
- Pokud si to studenti zaslouží, pak zjednodušený zápis servletu a asynchronní volání (násl. dva slidy).

Anotace v Servletech 3.0

- Netřeba zápis do web.xml

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

   public class CalculatorServlet extends HttpServlet{

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

Asynchronní volání v Servletech 3.0

- Jednoduché řešení dlouhotrvajících requestů bez blokování zpracujících threadů, případně pushing (server to client)

```
@WebServlet(name="CalculatorServlet", asyncSupported=true,
urlPatterns={"/calc", "/getVal"})

  public class CalculatorServlet extends HttpServlet{

    public void doGet(HttpServletRequest req, HttpServletResponse
res) {
      AsyncContext aCtx = req.startAsync(req, res);
      ...
    }
    ...
  }
```

Status kód

- 100-199 Informational
- 200-299 Successful
- 300-399 Redirection
- 400-499 Incomplete
- 500-599 Server Error
- Nejčastější
  - 200
  - 404
  - 5xx
- Jsou doprovázeny i slovním vysvětlením a občas i obsahem (často 404)
- sendError(int code [, String desc])

Shrnutí

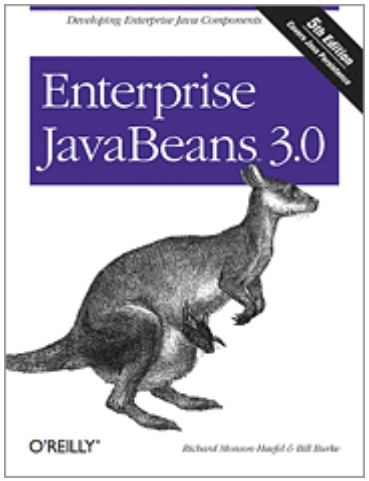
- Poměrně jednoduché rozhraní, výkonné
- Bude všechno v servletech?
- Dovedete si představit větší AJAX aplikaci postavenou nad servlety? Navigaci, komponenty, životní cyklus stránek...
- MVC???
- Servlety se používají na ne-HTML stránky, obrázky ap.

Materiály k předmětu

- java.sun.com - hromady materiálů, tutoriálů
- www.javapassion.com - výborný zdroj informací, slidy
- http://java.sun.com/products/servlet/

- Java servlety a stránky JSP

- Enterprise JavaBeans 3.0, O'Reilly, 2006



Co jsme probrali

- zopakovali jsme základy HTTP
- strukturu webové aplikace
- anatomii a konfiguraci servletu
- zjistili jsme, jaké informace lze získat
- filtry

Příště

- tohle MUSÍ JÍT JEDNODUŠEJI! V konkurenci PHP nemůže Java obstát se servlety. Naučíme se JSP.