

Servlety

# Maven dependency

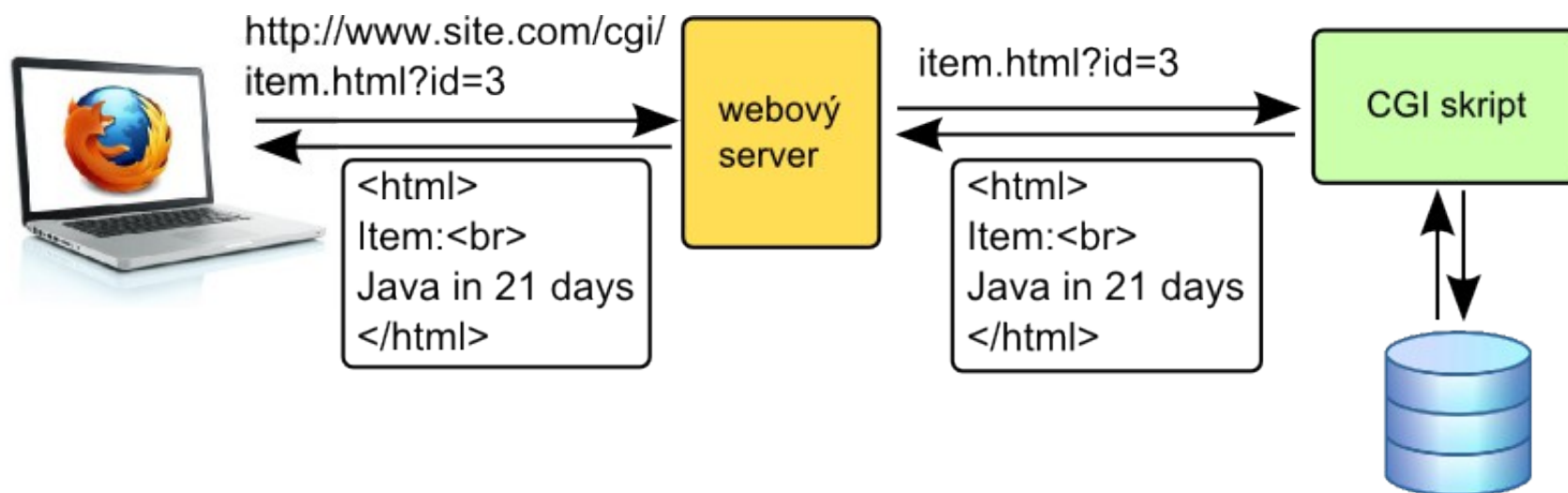
- Pokud používáte Maven, pak přidejte do pom.xml tyto dependency (pro Java EE 6):

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.0.1</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>javax.servlet.jsp</groupId>
  <artifactId>javax.servlet.jsp-api</artifactId>
  <version>2.2.1</version>
  <scope>provided</scope>
</dependency>
```

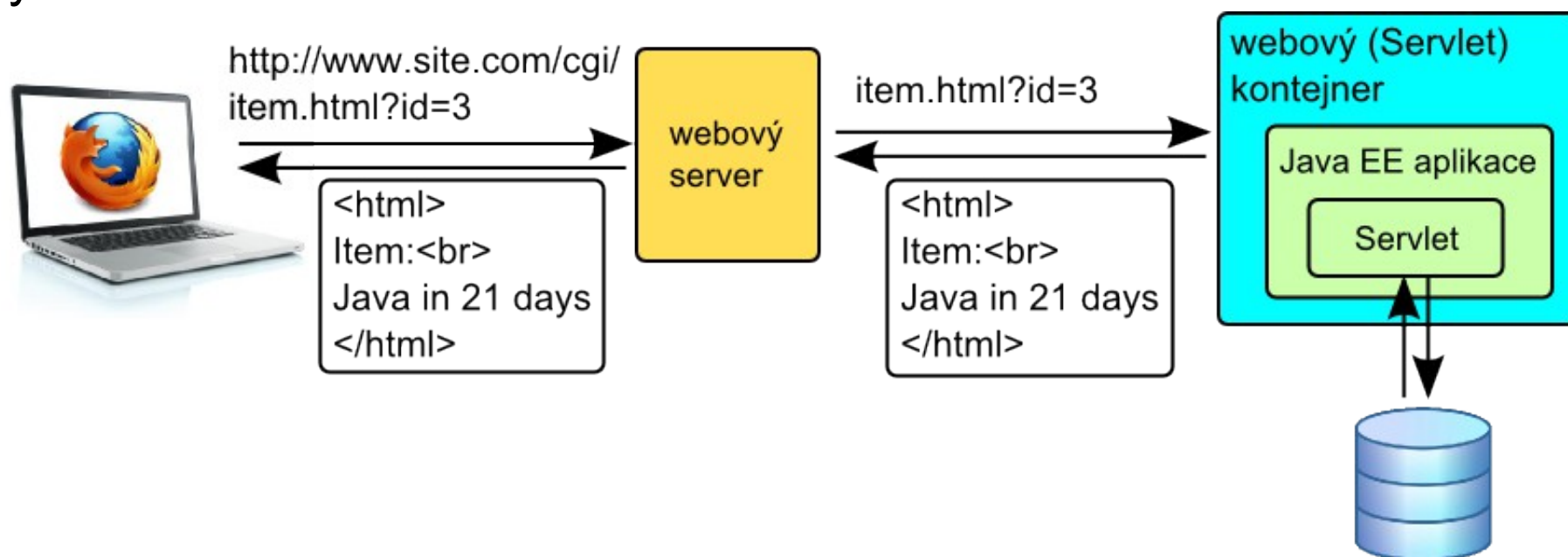
# Webový server, CGI skripty

- Apache HTTPD server je jenom webovým serverem, který má na starosti pouze „servírování“ statických stránek klientovi. Dnešní webové aplikace ale vyžadují dynamické prvky a to webový server neumí. Proto vznikly CGI skripty.
- CGI skripty byly aplikace (obvykle naprogramované v jazyku C), které webový server (například Apache HTTPD) zavolal s nějakými parametry a na standardní výstup poslaly HTML kód, který Apache HTTPD poslal klientovi.



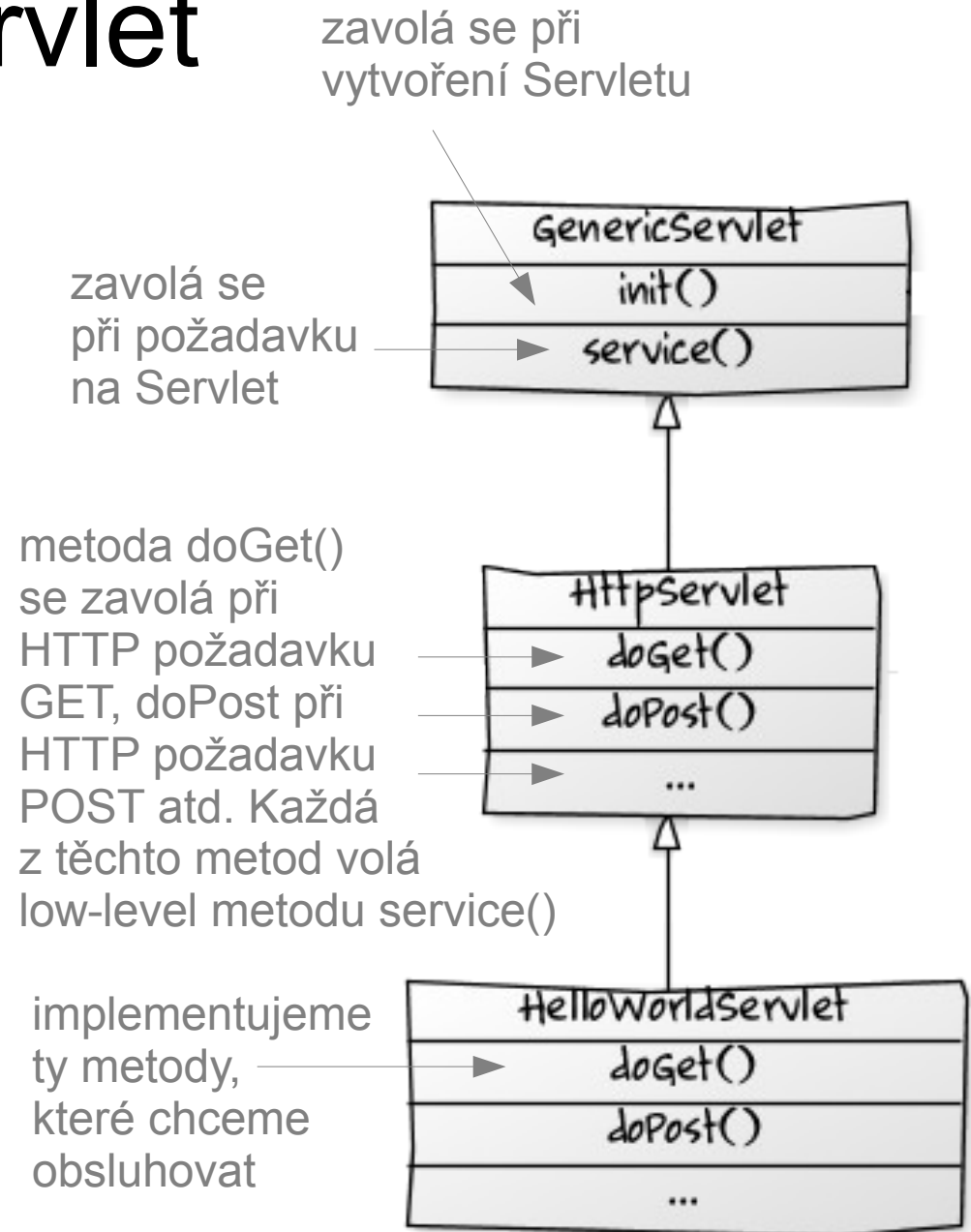
# Servlet kontejner, Servlety

- CGI skripty byly ale příliš „low level“ aplikace. Proto vznikly Servlety. Servlet je historicky alternativou vůči CGI skriptům. Je to základní komponenta v Java EE, která přijme HTTP request, zpracuje ho a pošle klientovi HTTP response.
- Servlet ale na rozdíl od CGI skriptů není aplikace, která by mohla běžet stand-alone. Život Servletu je svázán s životem Servlet kontejneru, který zabezpečuje komunikaci Servletů, řídí jejich životní cyklus atd.



# Servlet

- Servlet je třída, která je potomkem třídy `HttpServlet`, která je potomkem třídy `GenericServlet`. Je to tak z důvodu flexibility, ale v praxi se pravděpodobně nikdy nesetkáte s jiným potomkem třídy `GenericServlet`.
- V Servletu implementujeme metody `doGet()`, `doPost()` atd., které se zavolají při požadavku na Servlet HTTP GET požadavkem, HTTP POST požadavkem atd.



# Zapojení Servletu

- Sám o sobě je Servlet obyčejnou třídou. Co z ní dělá webovou komponentu reagující na HTTP požadavek klienta je konfigurace zapojení Servletu:
  - Buď v souboru `web.xml` (před Java EE 6)
  - Nebo pomocí anotace `@WebServlet` (nově od Java EE 6)

```
<servlet>
<servlet-name>HelloWorldServlet</servlet-name>
<servlet-class>cz.java.skoleni.web.servlets.HelloWorldServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>HelloWorldServlet</servlet-name>
<url-pattern>/helloworld.html</url-pattern>
</servlet-mapping>
```

konfigurace pomocí `web.xml`

na jakém URL bude Servlet „naslouchat“. Je možné specifikovat více `url-pattern` nebo dokonce masku (\*)

```
@WebServlet("/helloworld.html")
public class HelloWorldServlet extends HttpServlet {
    ...
}
```


konfigurace pomocí anotace `@WebServlet`

# Životní cyklus Servletu, metoda doGet()

- Servlet má jednoduchý životní cyklus, buď neexistuje, nebo existuje a přijímá HTTP požadavky.
- Metody `doGet()`, `doPost()` apod. mají dva parametry následujících typů:
  - `HttpServletRequest`
  - `HttpServletResponse`
- Tyto třídy jsou potomky tříd `ServletRequest` a `ServletResponse` opět z důvodu vyšší flexibility, ale s jinými typy se v praxi pravděpodobně nesetkáte.

Vstupy od klienta.  
Zajímavé metody:  
`getParameter()`  
`getHeader()`  
`getSession()`

Výstup posílaný klientovi.  
Zajímavé metody:  
`getWriter()`  
`getOutputStream()`  
`sendRedirect()`  
`sendError()`



```
protected void doGet(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException { ... }
```

# HTTP parametry I.

- HTTP parametry získáte pomocí metody `request.getParameter()`:

index.jsp:

```
<form action="hello.html">
  <input type="text" name="jmeno" />
  <input type="submit" name="submit"
    value="Odeslat" />
</form>
```

Zavolá se následující URL, přičemž HTTP parametry jsou za otazníkem. Jednotlivé parametry jsou od sebe odděleny znakem &

**`http://localhost:8080/WebProject/hello.html?jmeno=Jirka&submit=Odeslat`**

HelloServlet.java:

```
String jmeno = request.getParameter("jmeno");
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("ahoj " + jmeno);
out.println("</html>");
```

V Servletu parametry zpřístupníme metodou `request.getParameter()`



# HTTP parametry II.

- Standardně se vše posílá přes HTTP metodu GET (když uživatel zadá do prohlížeče URL adresu, když klikne na odkaz, když pošle formulář).
- U formulářů je navíc možné poslat celý formulář metodou POST (`method="post"`). Formuláře se obvykle přes POST posílají. Proč?
  - Při poslání požadavku GET metodou jsou parametry vidět v URL.
  - URL má maximální velikost (4KB)
- Na druhou stranu ale není vhodné přes POST posílat úplně vše. Proč?
  - Zkuste dát F5 (refresh) na stránce, na kterou se přišlo přes POST.

# Výstupy ze Servletu I.

- Máme několik možností jak poslat data ze Servletu:
  - Znaky pomocí `response.getWriter()`
  - Byty pomocí `response.getOutputStream()` - vhodné pro zasílání souborů klientovi.
- Navíc máme další možnosti:
  - V případě, že nastala nějaká chybová situace může Servlet poslat klientovi chybový kód, populární jsou: 404 (File Not Found), 503 (Internal Server Error) apod:

```
response.sendError(HttpServletResponse.SC_NOT_IMPLEMENTED);
```

- Ve třídě `HttpServletResponse` jsou užitečné konstanty pro často používané chybové kódy.

# Výstupy ze Servletu II.

- V případě, že klientovi pošlete HTTP chybovou hlášku, zobrazí se mu obdobná chybová stránka:
- Taková chybová stránka není moc user-friendly.



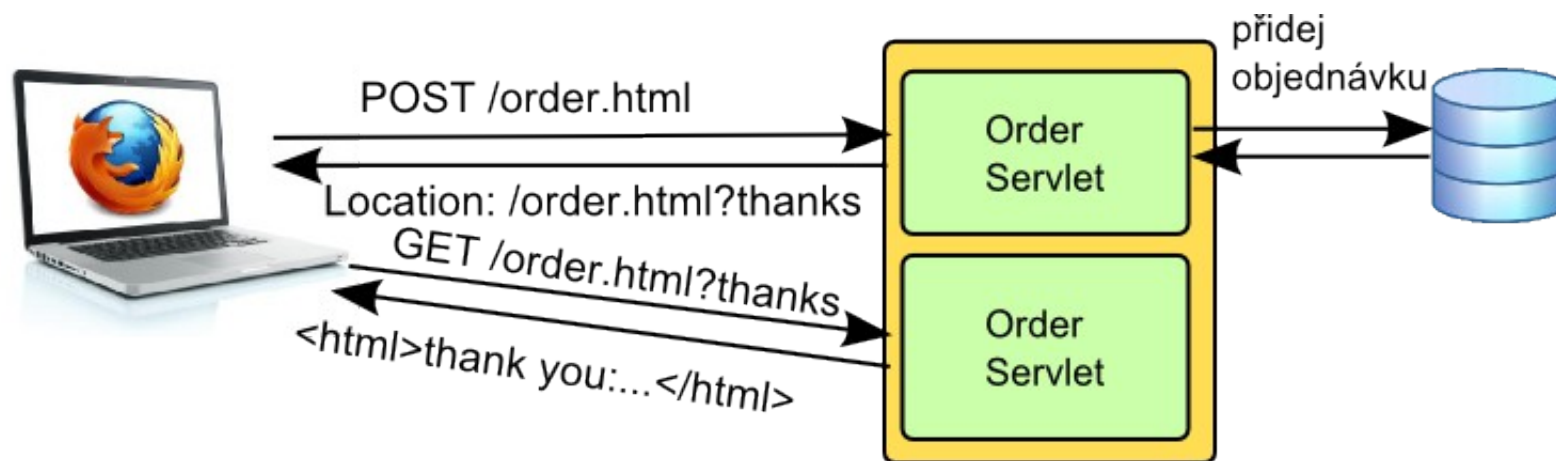
- Abyste ji změnili, vytvořte svůj soubor (například error.jsp) a ve web.xml ho zapojte. Mimochodem, je také možné nastavit obdobným způsobem chybovou hlášku také v případě vyhození určitého typu chyby.

```
<error-page>
    <error-code>404</error-code>
    <location>/error.jsp</location>
</error-page>
```

# Výstupy ze Servletu III.

- Ze Servletu můžete přesunout řízení na další webovou komponentu dvěma způsoby.
- Prvním z nich je **redirect**, pomocí kterého je možné provést přesměrování na jakoukoli webovou stránku na celém internetu. Není transparentní vůči klientovi, přesměrování fyzicky dělá sám klient.

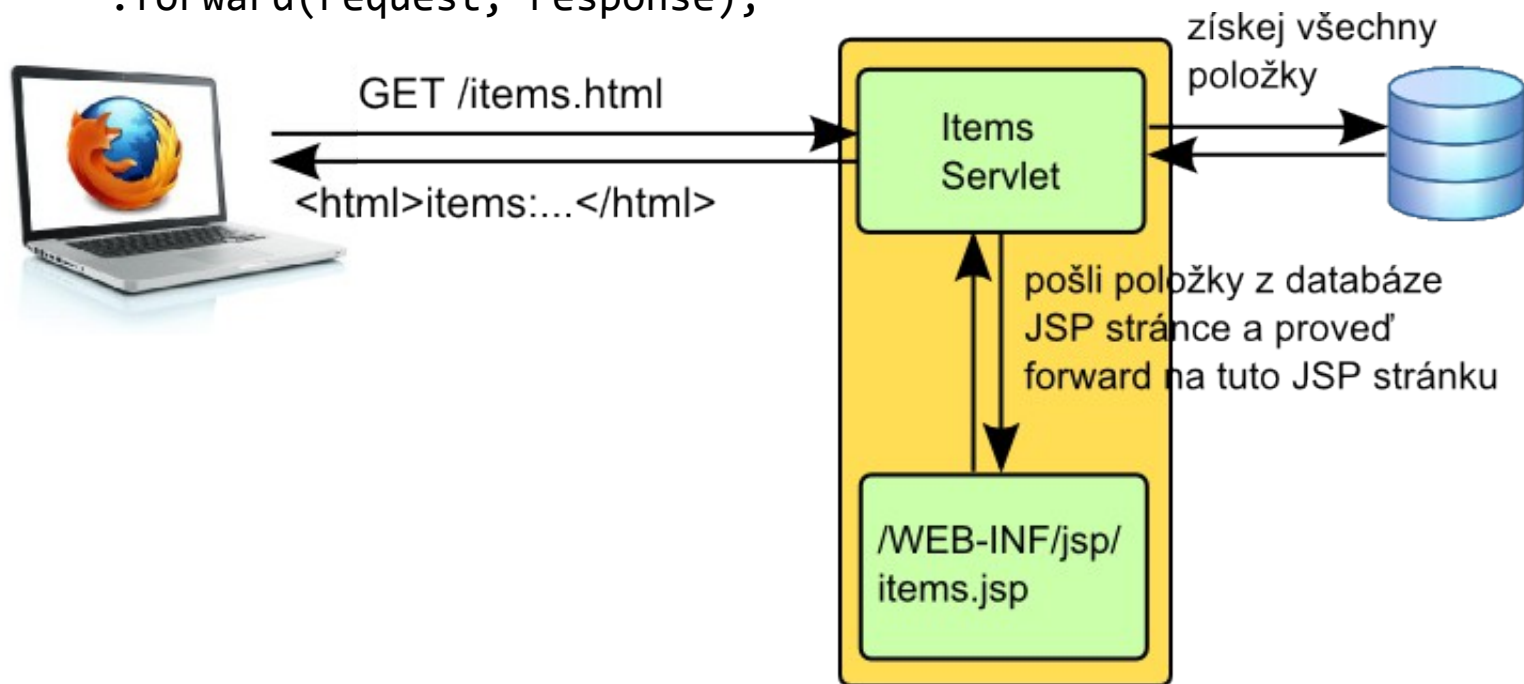
```
response.sendRedirect("order.html?thanks");
```



# Výstupy ze Servletu IV.

- Druhý způsob přesměrování řízení ze Servletu je **Request dispatcher**, pomocí kterého je možné provést přesměrování pouze na webovou komponentu, nacházející se ve stejné webové aplikaci. Je transparentní vůči klientovi, ten se o něm vůbec nedozví. Pomocí request dispatcheru je možné provádět **forward** nebo **include**.

```
request.getRequestDispatcher("/WEB-INF/jsp/items.jsp")  
    .forward(request, response);
```



# Atributy

- Webové komponenty na serveru nežijí osamoceny, občas musí vyměnit data s jinými komponentami a celkově spolu musí komunikovat. Jak na to? Pomocí atributů.
- Co je atribut? Je to objekt, který někdo připíchnul na vývěsní desku, aby k němu ostatní měli přístup. Fyzicky jsou atributy objekty s unikátním textovým klíčem.
- Základní otázka zní: Kdo má přístup k vývěsní desce a jak dlouho jsou na ní informace platné? Jinými slovy, jaký je rozsah platnosti (scope) atributu?



# Scope atributu

- Existují tři vývěsní desky (scope):
  - Context** (`getServletContext()`) – jedna vývěsní deska pro celou aplikaci.
  - Session** (`request.getSession()`) – jedna vývěsní deska pro jednoho uživatele.
  - Request** (`request`) – jedna vývěsní deska pro jeden request na serveru pomocí request dispatcheru.
- Práce s atributy:

```
// nastavení nebo prepsání atributu
```

```
request.setAttribute("items", itemList);
```

```
// získání atributu
```

```
List<Item> itemList = (List<Item>) request.getAttribute("items");
```

```
// odebrání atributu
```

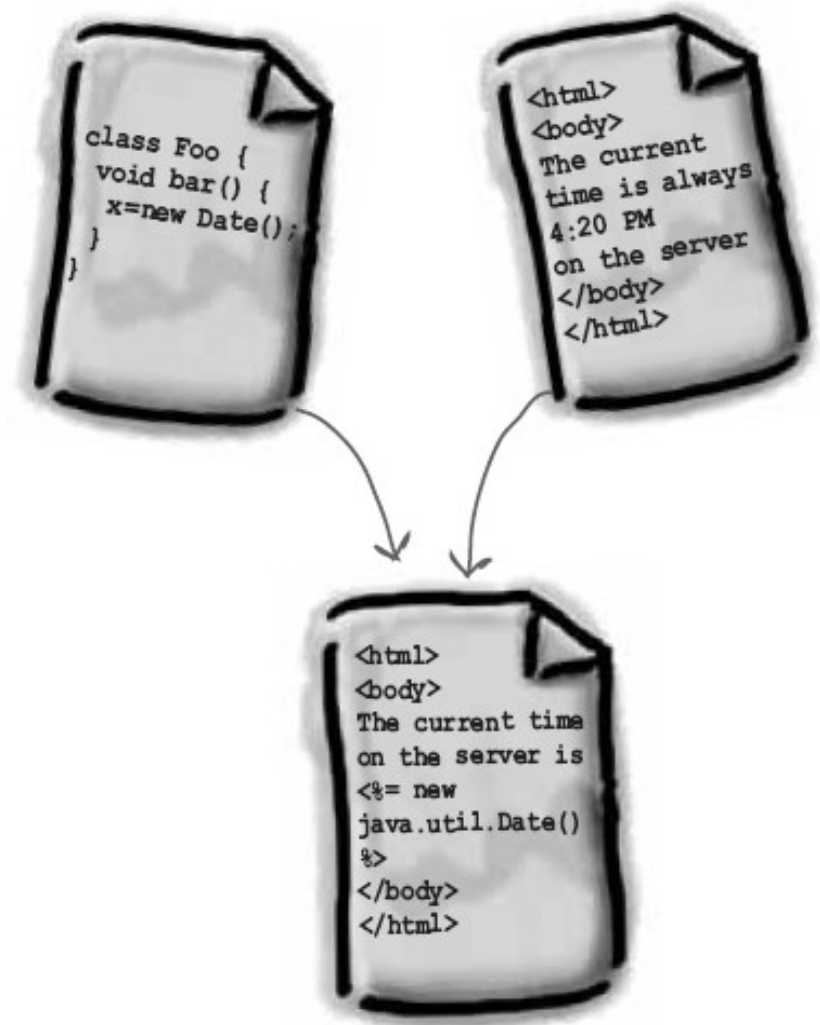
```
request.removeAttribute("items");
```



request.getAttribute() vrací typ  
Object, který je nutné přetypovat!

# JSP stránky

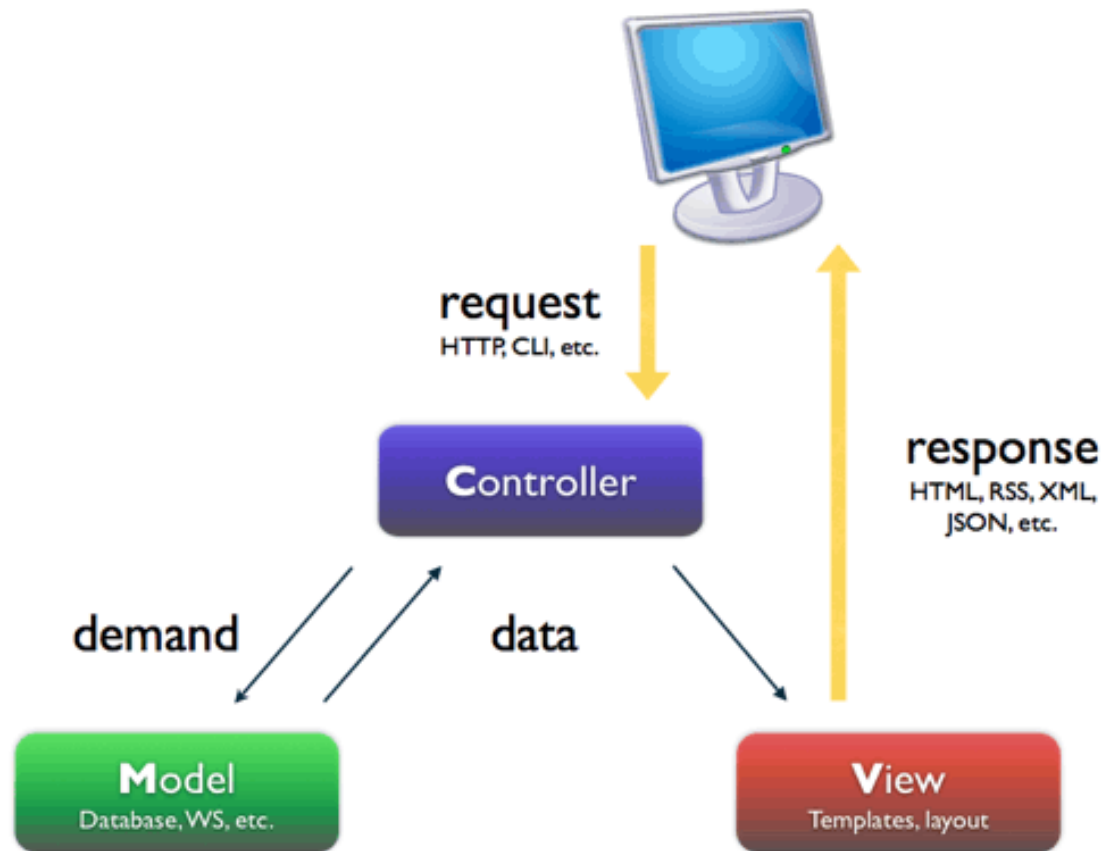
- Servlety jsou výborné webové komponenty pro dynamickou část webu (jsou to obyčejné Java třídy), ale nejsou vůbec vhodné pro zobrazení HTML ... v podání Servletu to je obyčejný String, to znamená že uvozovky ve výsledném HTML textu je nutné escapovat atd.
- JSP stránky na druhou stranu slouží pro efektivní zobrazení HTML kódu, do kterého je možné přidat dynamické prvky naprogramované v jazyce Java.





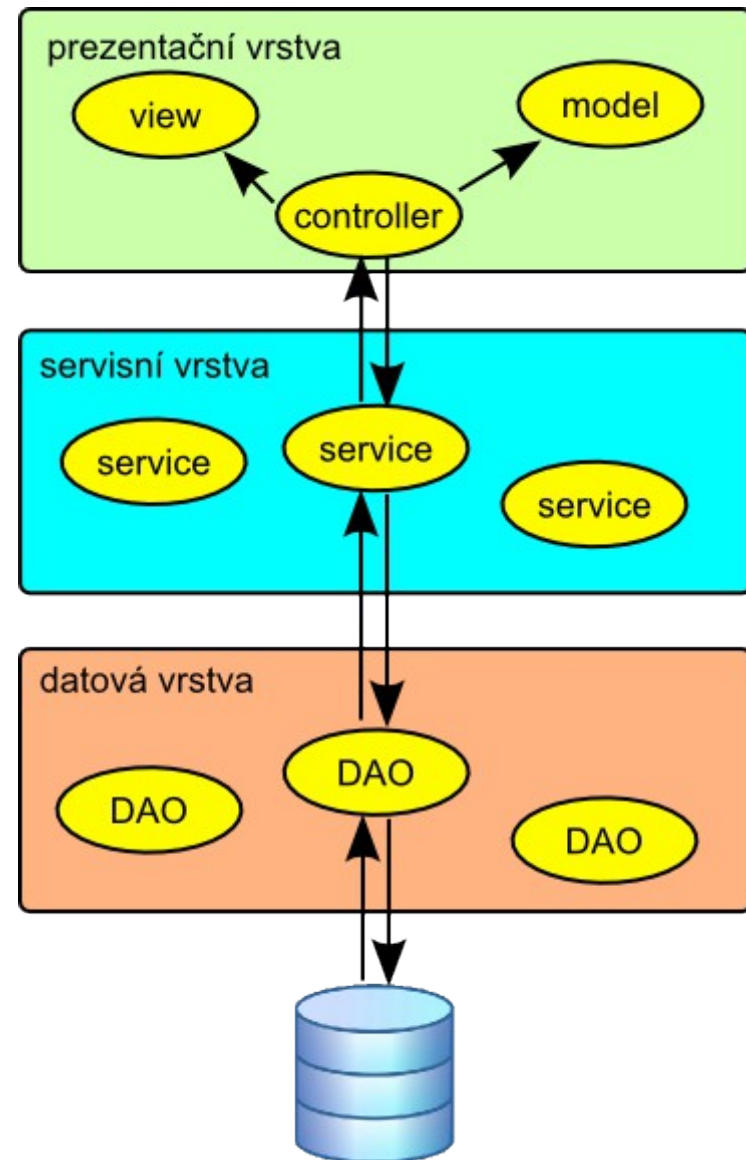
# MVC (Model View Controller)

- **MVC** je obvykle používaná architektura webových aplikací. Nejedná se o „Javovou věc“. Když programujete Vaši aplikaci s MVC architekturou, tak ji rozdělujete do tří částí:
  - **Model** – obsahuje stav aplikace a případně i logiku pro práci s databází.
  - **View** – má na starosti zobrazení dat klientovi.
  - **Controller** – řídí běh aplikace. Vezme vstup od klienta, rozhodne co s ním, případně zavolá model a pošle data do view.



# Třívrstvá architektura

- MVC se obvykle používá spolu s třívrstvou architekturou. V této architektuře je aplikace rozdělena na tři části:
  - **Prezentační vrstva** – obsahuje MVC architekturu.
  - **Servisní vrstva** – obsahuje tzv. servisní třídy, ve kterých je obsažena logika aplikace.
  - **Datová vrstva** – má na starosti práci s databází. V případě použití ORM frameworku (například Hibernate) se může vynechat.



# Session I.

- Webové servery a protokol HTTP pojem session neznají. Jakmile klient udělá další request, data z předchozího requestu se zapomínají. Někdy nám to stačí, ale u moderních aplikací to je hodně málo.
- Servlet API tento problém řeší tím způsobem, že umožňuje udržování session klienta. Session si udržuje stav při více requestech od jednoho klienta.
- Jak server pozná, že se jedná o stejného klienta v dalším requestu? Pomocí **session ID**.
  - Při prvním requestu od klienta kontejner vygeneruje unikátní session ID a vrátí ho klientovi v response. Klient poté posílá serveru toto session ID s každým dalším requestem.
  - Jak si klient a server vymění informace o session ID? Pomocí cookies. Co je cookie? Jedná se o parametr v header.

# Session II.

```
HttpSession session = request.getSession();
```

- Pokud ještě nebyla vytvořena session, tak se vytvoří. Pokud už vytvořena byla, tak se získá aktuální session
- Session je jednou z nástěnek (scope), kam je možné vložit atribut.
- Co když klient nepodporuje cookies? V dnešní době se to již obvykle neřeší, pokud si klient vypne cookies, tak mu nebude fungovat téměř žádná webová aplikace.
  - Pokud ale přesto tento problém chcete vyřešit, tak řešení se jmenuje **URL rewriting**. Při tomto postupu se session neukládá do header hlavičky, ale přidává se k URL. Problém je v tom, že musíte tzv. encodovat každé URL. Jakmile totiž klient klikne na URL, které Session ID neobsahuje, session se ztratí.

# Session III.

- Jsou dvě možnosti jak ukončit session:
  - Zavoláme `session.invalidate();`
  - Session se automaticky ukončí po určité době neaktivity klienta (obvykle 30 minut, nastavuje se ve `web.xml`):

```
<session-config>
```

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

```
</session-config>
```

# Session IV.

- Od Java EE 6 (Tomcat 7) můžete uvnitř `web.xml` uvést, co chcete používat k transportu session ID. Je best practice přidat do `web.xml`:

```
<session-config>  
    <tracking-mode>COOKIE</tracking-mode>  
</session-config>
```

# Cookies

- Cookies se nemusí používat pouze pro uchování Session ID. Cookie není nic jiného než pár Stringů: jméno / hodnota.
- Cookie při výchozím nastavení zmizí, jakmile klient zavře prohlížeč. Je ale možné nastavit delší dobu platnosti cookie (pomocí maxAge):

```
//ziskani cookies z requestu
```

```
Cookie[] cookies = request.getCookies();
```

```
//vytvoreni cookie a poslani klientovi
```

```
Cookie cookie = new Cookie("name", "value");
```

```
cookie.setMaxAge(30 * 60); // v sekundach
```

```
response.addCookie(cookie);
```