Architektura webové aplikace, funkce jednotlivých vrstev, životní cyklus standardizovaných komponent Java EE, Servlety, JSP, frameworky, návrhové vzory

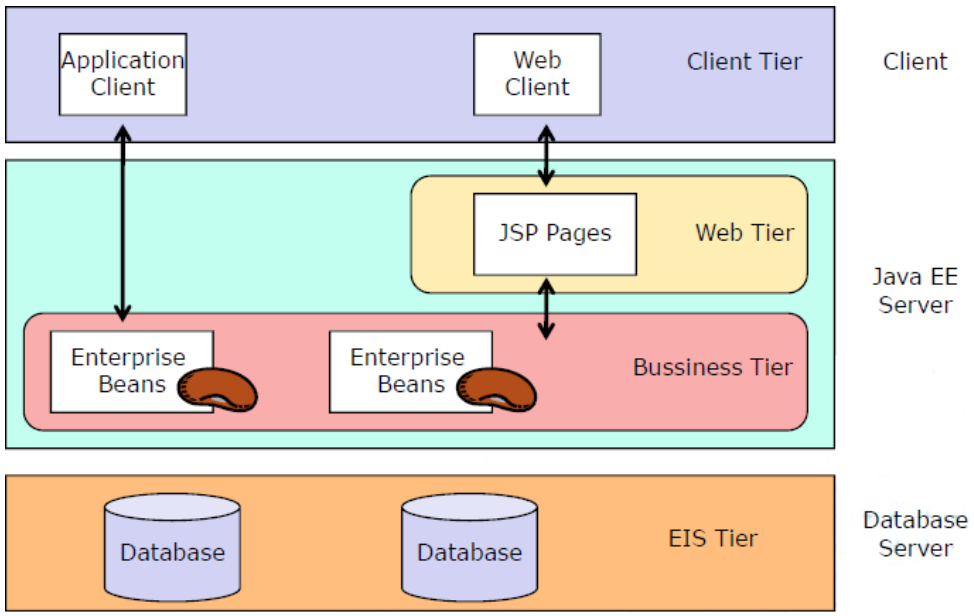
1. Distribuce Javy

Distribuce Javy se liší podle jejího zamýšleného použití:

- `JME` = Java Micro Edition pro mobilní aplikace, omezený rozsah funkcí
- `JSE` = Java Standard Edition pro desktopové aplikace (GUI např. pomocí `Swing`, `AWT`)
- `J2EE` = Java Enterprise Edition pro webové aplikace, obvykle je nasazena na aplikačním serveru (klient nemusí)

2. Vlastnosti J2EE aplikace

- komponentový přístup
- komponenty mohou být distribuované na různých strojích (klient, server, DB)
- aplikace rozdělena do vrstev (většinou 3)
- serverová část aplikace bývá nasazena na aplikačním serveru, který umožňuje zpracování více požadavků naráz (multi-threading)
- podpora JNDI, java beans, JAAS (autentizace), JMS (Java Messaging Service), servlety, transakce (JTA – Java Transactions API)



Obrázek 1 - Architektura J2EE aplikace

Obrázek 1 ukazuje dva různé přístupy. V levé části obrázku je aplikační **tlustý klient** přistupující rovnou k business vrstvě serverové části aplikace. V pravé části klient používá pro přístup webový prohlížeč (**tenký klient**), ve kterém se mu vykresluje GUI v podobě HTML stránek vygenerovaných z JSP stránek umístěných na serveru.

3. Fyzická architektura J2EE aplikace

`J2EE` aplikace je komponentová. Na aplikační server ji lze nasadit ve dvou podobách:

- jako EAR archiv
- jako WAR archiv

EAR archiv se používá v případě nasazení EJB (Enterprise Java Beans). Pak obsahuje jeden WAR archiv JSP stránkami, obrázky a styly a několik JAR archivů s EJB. JAR archivy obsahující EJB jsou rozdělené podle své funkčnosti a tvoří logické celky, aby je bylo možné znovu použít v jiném projektu.

Samotný **WAR archiv** lze použit, pokud nechceme nasazovat EJB. Tento způsob nasazení je běžný u light-weight frameworků jako JSF, Spring, Struts...

Adresářová struktura WAR archivu

WAR archiv má předepsanou strukturu:

- v kořenovém adresáři jsou CSS styly, obrázky a JavaScript
- v adresáři META-INF je soubor MANIFEST.MF
- v adresáři WEB-INF obsahuje soubor web.xml (deskriptor)
- v adresáři WEB-INF/classes jsou zkompilované Java třídy
- v adresáři WEB-INF/jsp jsou Java Server Pages
- v adresáři WEB-INF/lib jsou knihovny ve formátu JAR

Deskriptor

Deskriptor je konfigurační soubor s názvem web.xml, který musí být v každé webové aplikaci. Obsahuje mapování servletů a filtrů na requesty, kódování JSP stránek, parametry servletů a stanovuje, která stránka se zobrazí jako první (welcome-file).

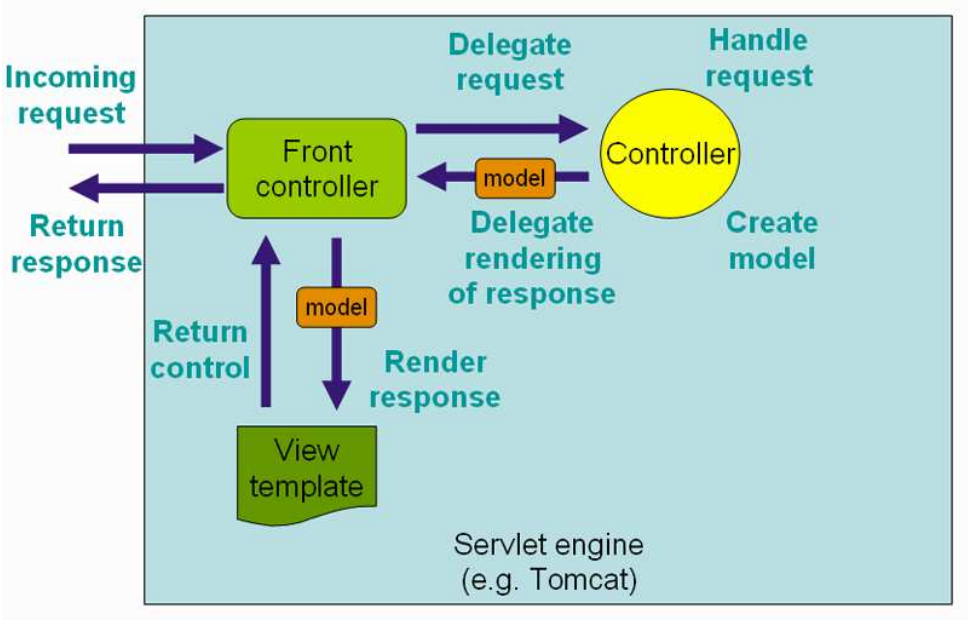
```
<servlet>
    <servlet-name>myTestServlet</servlet-name>
    <servlet-class>org.MyTestServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>myTestServlet</servlet-name>
    <url-pattern>*.html</url-pattern>
</servlet-mapping>

<jsp-config>
    <jsp-property-group>
        <url-pattern>*.jsp</url-pattern>
        <page-encoding>UTF-8</page-encoding>
    </jsp-property-group>
</jsp-config>
```

Obrázek 2 - Deskriptor

4. Logická architektura J2EE aplikace (vzor MVC)

Většinou je aplikace navržena podle voru **MVC** (model-view-controller). Jedná se o konkrétní aplikaci vzoru **oddělení zodpovědností** (separation of concerns), který předepisuje vysokou kohezi jednotlivých komponent. Každá **komponenta** by měla mít vysoce **soudržnou sadu zodpovědností** a všechny ostatní požadavky by měla delegovat komponentám, které jsou úzce specializované zase na jinou činnost.

MVC tedy odděluje zodpovědnosti za data (model), pohled na data (view) a manipulaci s pohledem na data (controller). Aplikace je rozdělena na tři vrstvy: datovou, prezentační a ovladače (controllers). Tyto tři **vrstvy jsou na sobě nezávislé** a umožňují snadné vyjmutí jedné vrstvy a nahrazení jinou implementací. Typicky je možné zaměnit vrstvu View implementovanou pomocí JSP za generování do PDF nebo do Excelu.

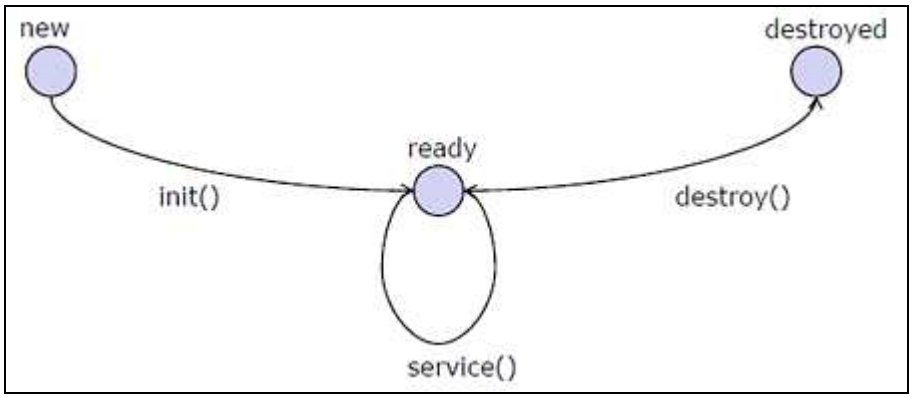


Obrázek 3 - Architektura MVC

Obrázek 3 ukazuje jednu z možných implementací MVC s FrontControllerem. Implementací může být více (např. knihovna Swing má pro desktopové aplikace svou implementaci MVC). Všechny requesty na zdroje aplikace jsou nejdříve přijaty komponentou **FrontController**. Podle požadované URL adresy najde odpovídající **controller** a předá mu request ke zpracování. Controller podle požadavku vytvoří takzvaný **model**, neboli data potřebná pro vytvoření správné odpovědi, která má být vrácena klientovi. Model je předán **View** vrstvě, která data vykreslí odpovídajícím způsobem. Vygenerovaná odpověď je zaslána zpět klientovi.

Rozvrstvením aplikace zvyšuje **přehlednost kódu** a oddělení nesouvisejících částí do logických celků, které spolu transparentně komunikují, aniž by se mezi nimi vyskytovala nějaká **těsná vazba**. Takto navržený systém je robustní vzhledem ke svému **rozšiřování** a úpravám.

5. Životní cyklus HTTP servletu



Obrázek 4 - Životní cyklus servletu

Aplikační server pro každý servlet vytvoří nový servlet a zavolá nejdříve metodu `init()`. Pak následuje obsluha požadavku metodou `service()` a nakonec je po zavolání `destroy()` servlet je zničen. Programátor může poskytnout vlastní implementaci metod `init()`, `service()` a `destroy()`.

Podle typu HTTP requestu metoda `service()` volá různé implementace: `doGet()`, `doPost()`, `doDelete()`, `...`

Důležité metody HTTP servletu

Parametrem metody doGet() je objekt třídy HttpServletRequest, který má tyto důležité metody:

`String`	`getParameter(String name)`	vrátí request parametr (např. hodnota formulářového pole)
`void`	`setParameter(String name)`	nastaví request parametr
`String`	`getRemoteAddr()`	vrací IP adresu klienta
`String`	`getScheme()`	vrací protokol requestu (http, https, ftp)
`String`	`getContextPath()`	vrací část aktuální URL adresy
`String`	`getHeader(String name)`	vrací obsah hlavičky
`HttpSession`	`getSession()`	vrací session objekt

Parametrem metody doGet() je objekt třídy HttpServletResponse, který má tyto důležité metody:

`void`	`addCookie(Cookie c)`	přidá do odpovědi cookie
`void`	`setHeader(String name, String value)`	nastaví hodnotu hlavičky
`ServletOutputStream`	`getOutputStream()`	poskytuje proud pro zápis na výstup
`void`	`setContentType(String type)`	nastaví typ výstupu (např. html, xml, pdf...)

6. Java Beans

Bean implementují aplikační logiku nebo vystupují v roli entit cílové domény (entita zákazník, výpůjčka...). Bean může mít **lokální nebo vzdálené rozhraní** podle toho, jestli ji se nachází na stejném stroji, nebo je na jiném stroji. **Bean je několik druhů**:

- statefull bean
- stateless bean
- message driven bean

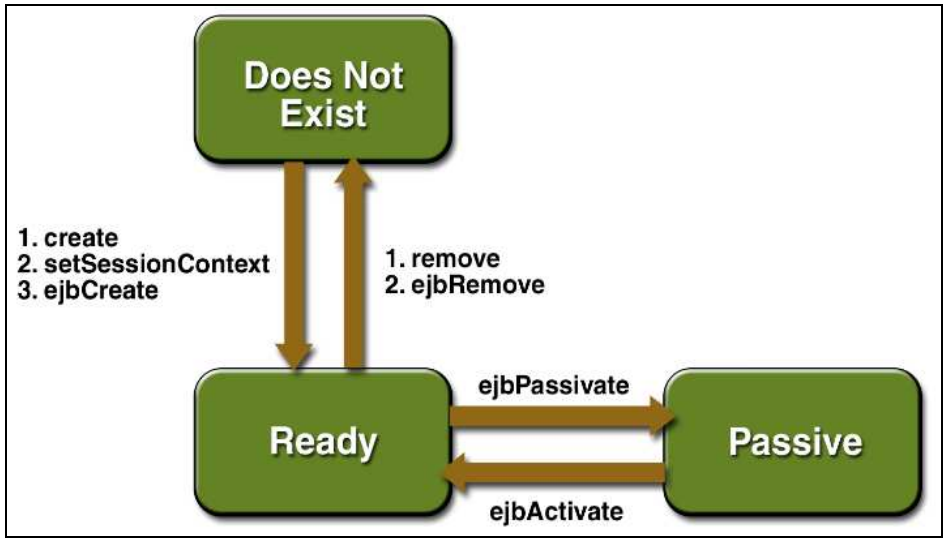
Statefull bean

Statefull bean si **uchovává kontext** pro každého klienta zvlášť. Je náročnější a pomalejší, proto bychom měli používat stateless, kdekoliv je to jen možné.

Životní cyklus:

1. klient zahájí životní cyklus beany voláním metody `create()`
2. EJB kontejner vytvoří instanci beany a zavolá `setSessionContext()` a `ejbCreate()`
3. bean je připravena na volání business metod
4. bean může být deaktivována voláním `ejbPassivate()`, čímž je bean přesunuta z primární do sekundární paměti
5. pokud je na deaktivovanou bean volána nějaká její business metoda, je bean přesunuta zpět do primární paměti
6. pokud bean není potřeba, je pomocí `remove()` zničena a `garbage collector` ji smaže z paměti

Programátor má kontrolu jen nad metodami `create()` a `remove()`. Ostatní si řídí kontejner sám.

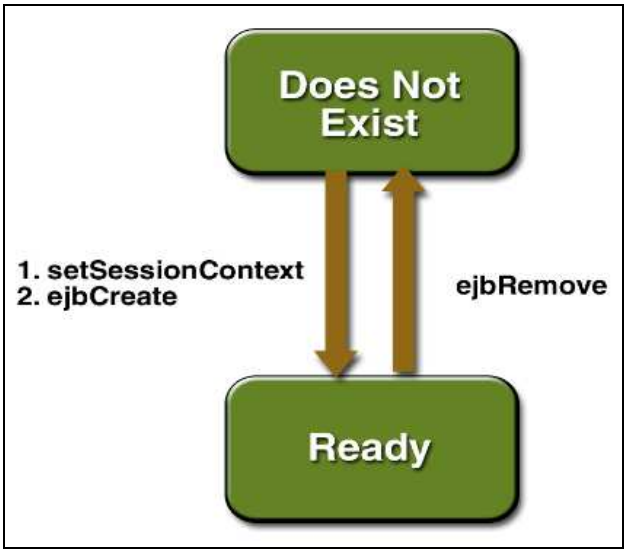


Obrázek 5 - Statefull bean

Stateless bean

Stateless bean je jednodušší. **Nepamatuje si kontext** volání klienta.

Životní cyklus:



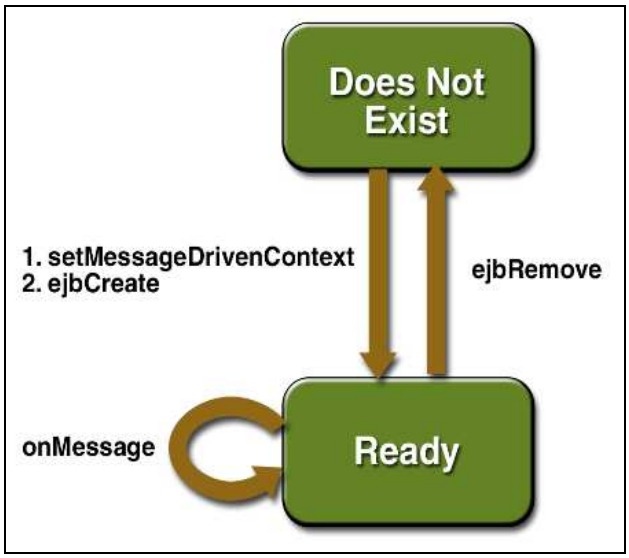
Obrázek 6 - Stateless bean

Message driven bean

Pomocí `JMS` můžeme posílat zprávy jiným klientům, kteří na ně reagují (implementují metodu `MessageListener.onMessage()`).

Jsou dva modely komunikace:

- Point to point pomocí fronty (queues, FIFO), příjemce je jen jeden
- Publish / subscribe pomocí tzv. topics, příjemců je více

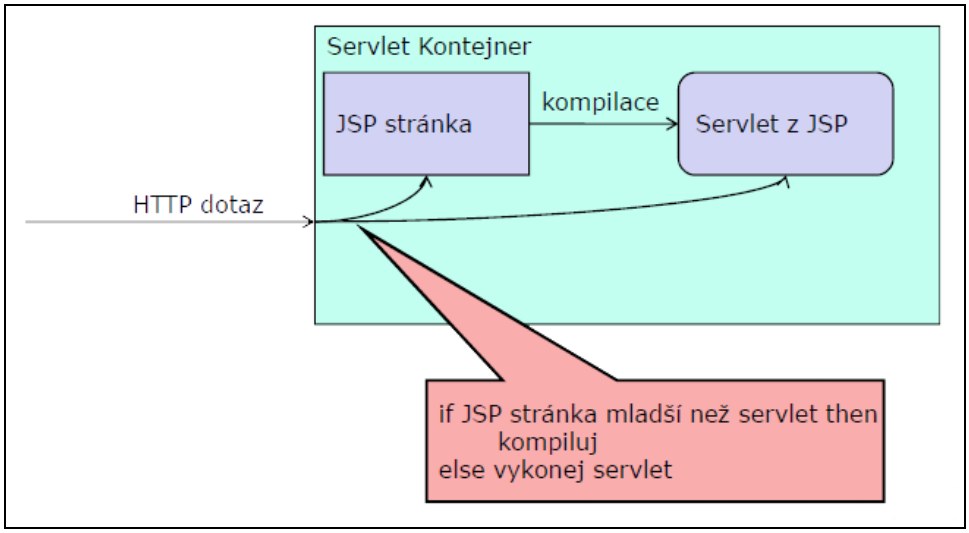


Obrázek 7 - Message Driven bean

7. JSP

- JSP je textový dokument obsahující statické (X)HTML tagy a JSP tagy
- JSP tagy generují dynamický obsah
- JSP umožňuje používat kusy Java kódu v HTML stránce (scriptlet) na způsob ošklivého PHP kódu, ale tak bychom to dělat neměli! Lepší je vytvořit data pomocí controlleru a předat

je v podobě modelu do view vrstvy, kde se zobrazí za pomoci JSP tagů (rozšíření se jmenuje JSTL)
- JSP se při prvním použití kompiluje a vytvoří se Servlet, který dělá to, co uměla původní JSP stránka (Obrázek 8)



Obrázek 8 - Kompilace JSP na Servlet

8. Frameworky

Je jich spousta [4]. Jmenujme Struts, Tapestry, Spring, JSF, GWT, Vaadin.

Zdroje

[1] Dokumentace J2EE.
`http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Overview.html`

[3] Dokumentace EJB.
`http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts9.html`

[2] Mrázek, Pavel: Diplomová práce.
`https://dip.felk.cvut.cz/browse/pdfcache/mrazepa1_2011dipl.pdf`

[3] Klíma, Martin: Přednáška z WA2.
`http://edux.feld.cvut.cz/courses/A4M39WA2/_media/lectures/03/javaee.pdf`

[4] Přehled J2EE frameworků.
`http://en.wikipedia.org/wiki/Comparison_of_Web_application_frameworks#Java_2`