Java Platform, Enterprise Edition (Java EE)

Marek Rychlý

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů

Přednáška pro AIS a PDI
19. listopadu 2007

Obsah

Obsah

Úvod

Platforma Java od Sun Microsystems je rozdělena na tři části (edice):

Java ME (Micro Edition) – pro zařízení s omezenou kapacitou (např. mobilní zařízení),

Java SE (Standard Edition) – pro obecné použití na desktopech a serverech (základní knihovny),

Java EE (Enterprise Edition) – rozšířená Java SE o technologie pro vícevrstvé serverové aplikace.

- Do verze Java SE 5 (vývoj. verze 1.5) se edice platformy Java označovaly jako J2ME, J2SE a J2EE.
- Aktuální verze Java EE je Java Platform, Enterprise Edition 5 (Java EE 5).
- Aktuální verze Java SE je Java Platform, Standard Edition 6 (Java SE 6, vývoj. verze 1.6).

Proč a kdy zvolit Java EE?

- Potřebujeme vícevrstvou serverovou aplikaci.
- Nechceme řešit systémové (technologické) problémy, ale soustředit se na aplikační úlohy.

  (tzn. problémy jako je komunikace jednotlivých částí aplikace, složitá integrace „cizích“ knihoven, optimalizace pro HW platformu, apod.)
- Hledáme komplexní multiplatformní řešení běžící na různém SW a HW.

  (existují konkrétní certifikované Java EE aplikační servery, pro ty jsou certifikované různé operační systémy, které jsou spolehlivé na pro ně certifikovaném hardware – vše od různých výrobců, pro servery např. Sun, BEA Systems, SAP, TmaxSoft, atd.)
- Bohatá a standardizovaná dokumentace, školení, podpora u certifikovaných řešení, atd.
- Budoucnost vývoje?

  (open-source implementace OpenJDK, Google Dalvik, atd.)

Aplikační komponenty a kontejnery

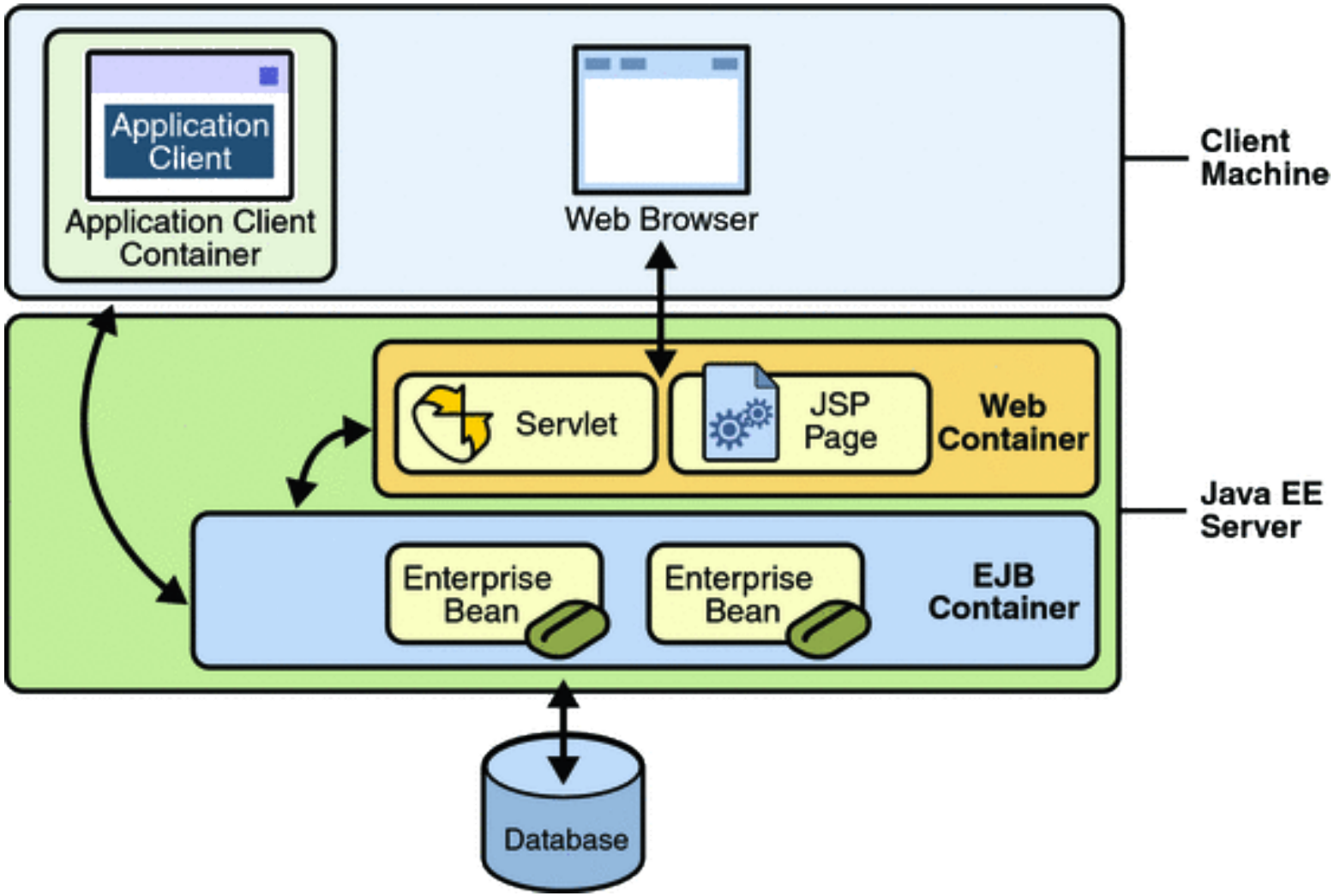
Aplikační komponenty a kontejnery:

komponenty: Části aplikace implementované vývojářem aplikace pomocí aplikačních technologií Java EE.
(klientské aplikace, applety, JSP a servlety, a komponenty EJB)

kontejnery: Části aplikačního serveru podle specifikace Java EE, které poskytují prostředí pro běh aplikačních komponent a zprostředkovávají obsluhu jejich rozhraní s okolním prostředím (spravují systémové zdroje).

Základem celé architektury je Java Platform, Standard Edition (Java SE), která poskytuje aplikačním komponentám a kontejnerům základní funkční API.

Architektura aplikace v Java EE podle [SUN, 2007]



Základní aplikační komponenty I

JSP (JavaServer Pages)

- pro generování dynamického obsahu textových dokumentů pro webový prohlížeč (tenký klient),
- do textového dokumentu (např. HTML či WML stránky) se vkládají úryvky kódu servletů (JSP elementy),
- JavaServer Pages Standard Tag Library (JSTL) poskytuje standardizované tagy podle Java EE specifikace (iterátory, podmínky, manipulace s XML, lokalizace, databáze, apod.).

Základní aplikační komponenty II

Servlety (Java Servlet)

- objekty generující HTTP odpovědi na základě HTTP dotazů (request-response programming model),
- obvykle implementují rychlé a optimalizované zpracování HTTP dotazů z webových stránek (např. obsluha formulářů),
- také JSP jsou za běhu aplikace přeloženy do podoby servletů (mechanicky, občas ne moc optimálně).

Základní aplikační komponenty III

EJB (Enterprise JavaBeans Technology)

- rámce pro implementaci aplikační logiky (business logic),
- základní stavební prvky aplikace (serverové komponenty),
- hierarchické členění aplikace mezi více procesů/vláken (uvnitř vrstev),
- dva typy[a]:
  - **session beans**: zpracovávají data v session (synchronní komunikace), rozlišujeme bezstavové a stavové session beans (stavové si uchovávají data mezi jednotlivými voláními),
  - **message-driven beans**: navíc přidávají asynchronní reakce na události (nevyžadují okamžitou odpověď), obecně přijímají zprávy pomocí Java Message Service (JMS).

[a] třetí typ **entity beans** pro persistentní objekty, byl v Java EE 5 nahrazen samostatnou technologií Java persistence API.

Další aplikační komponenty I

JDBC (Java Database Connectivity API)

- aplikační API pro volání SQL příkazů a dotazů,
- použití v enterprise bean pro přístup z session bean, nebo přímo ze servletu či JSP stránky,

JSF (JavaServer Faces)

- rámec pro implementaci uživatelských rozhraní webových aplikací (Java API a XML konfigurace),
- obsahuje komponenty GUI, model pro různou realizaci komponent v HTML a jiných značkovacích jazycích (standardně HTML/4.01),
- validace vstupních dat, zachytávání událostí, navigace mezi stránkami, a další.

Další aplikační komponenty II

JAX-WS (Java API for XML Web Services)

- podpora webových služeb pomocí JAXB API mapující XML data na Java objekty,
- pro implementaci poskytovatele i spotřebitele služby,
- při volání služby zachová prostředí a oprávnění dotazu spotřebitele při zpracování u poskytovatele služby.

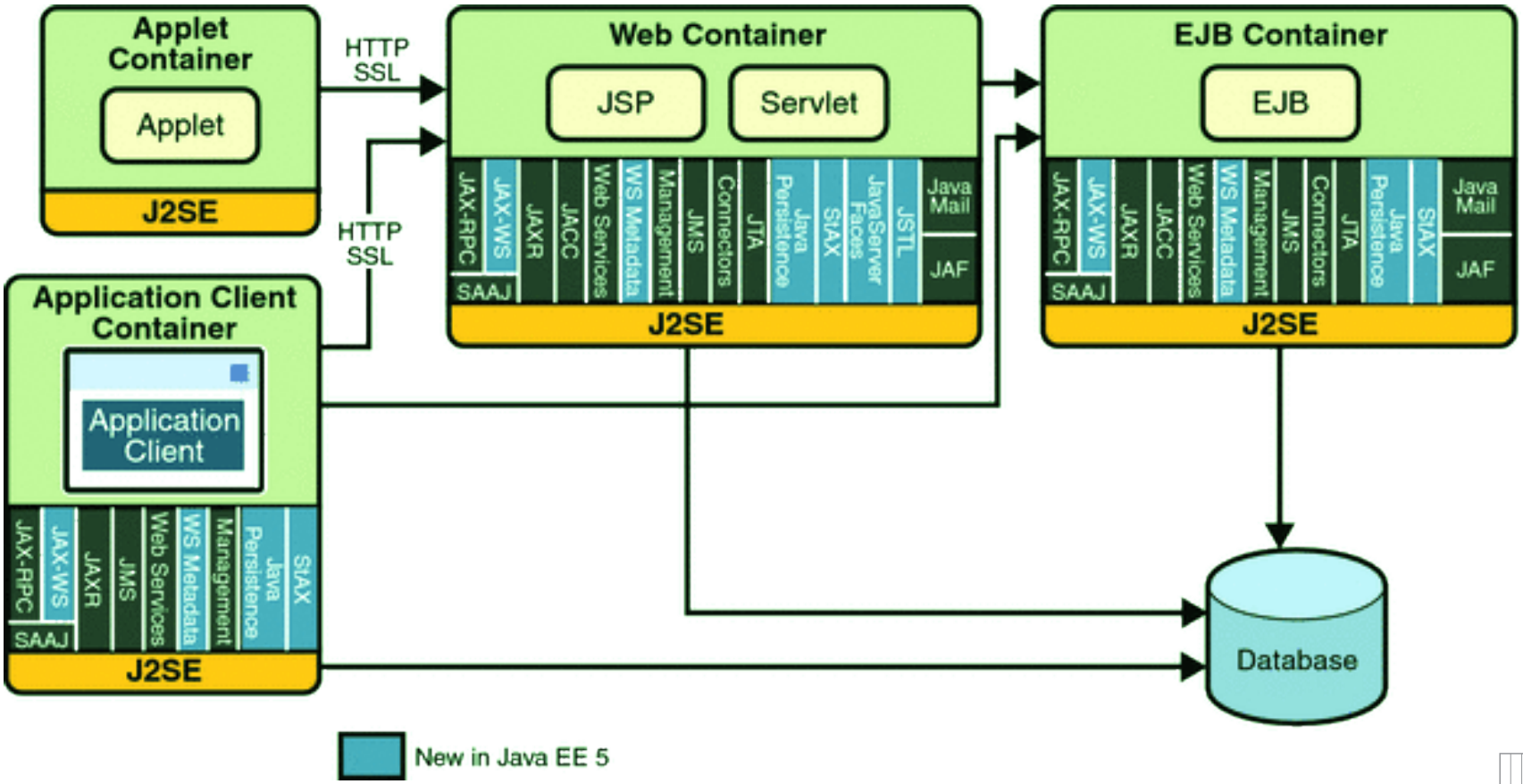
Java Persistence API

- provádí objektově-relační mapování Java objektů,
- poskytuje API pro persistenci objektů, dotazovací jazyk a metadata objektově-relačního mapování.

Další aplikační komponenty – stručně

- Java Message Service API
  (zpracování asynchronních volání)
- Java Transaction API
  (transakce, např. při současné změně dat ve více databázích)
- JavaMail API, JavaBeans Activation Framework
  (poštovní služby a práce se zprávami)
- Java API for XML Processing
- Java Architecture for XML Binding (JAXB), Java API for XML Registries
  (zpracování XML schémat a přístup k registrům XML schémat)
- SOAP with Attachments API for Java
- J2EE Connector Architecture, Simplified Systems Integration
  (přístup do systémů třetích stran a podpora trhů komponent a zdrojů)
- Java Naming and Directory Interface
  (abstrakce nad LDAP, NDS, DNS, NIS a dalšími)
- Java Authentication and Authorization Service

Technologie v Java EE podle [SUN, 2007]



tenký/tlustý klient – Java EE aplikační server – databáze

Architektura aplikace v Java EE

4 vrstvy: HTML client, JSP/Servlets, EJB, JDBC/Connector,

3 vrstvy:
a) HTML client, JSP/Servlets, JDBC,
b) EJB standalone applications, EJB, JDBC/Connector,

2 vrstvy: EJB standalone applications, JDBC/Connector.

B2B Enterprise aplikace:
propojením dvou a více aplikací v Java EE přes výměnu JMS nebo XML zpráv (např. pomocí webových služeb JAX-WS).

Srovnání Sun Java EE a Microsoft .NET

- .NET je platforma také pro jednovrstvé aplikace,
- .NET nabízí velké množství progr. jazyků i třetích stran,
- Microsoft Visual Studio .NET
  vs. IBM Eclipse, Sun Netbeans, Oracle jDeveloper, atd.
- .NET nabízí integraci COM komponent (Windows, Office, atd.),
- .NET WebForms a WindowsForms
  vs. Swing/AWT/SWT, JavaServer Faces a Struts,
- přístup k databázím via ADO.NET **vs.** JDBC,
- proprietární řešení Microsoftu a Mono projekt
  vs. Java standard Sunu a open-source implementace,
- komplexní řešení .NET od jednoho dodavatele
  vs. implementace Java EE a db. serveru od více dodavatelů.

podrobněji v [Šeda, 2003]

Obsah

Úvod

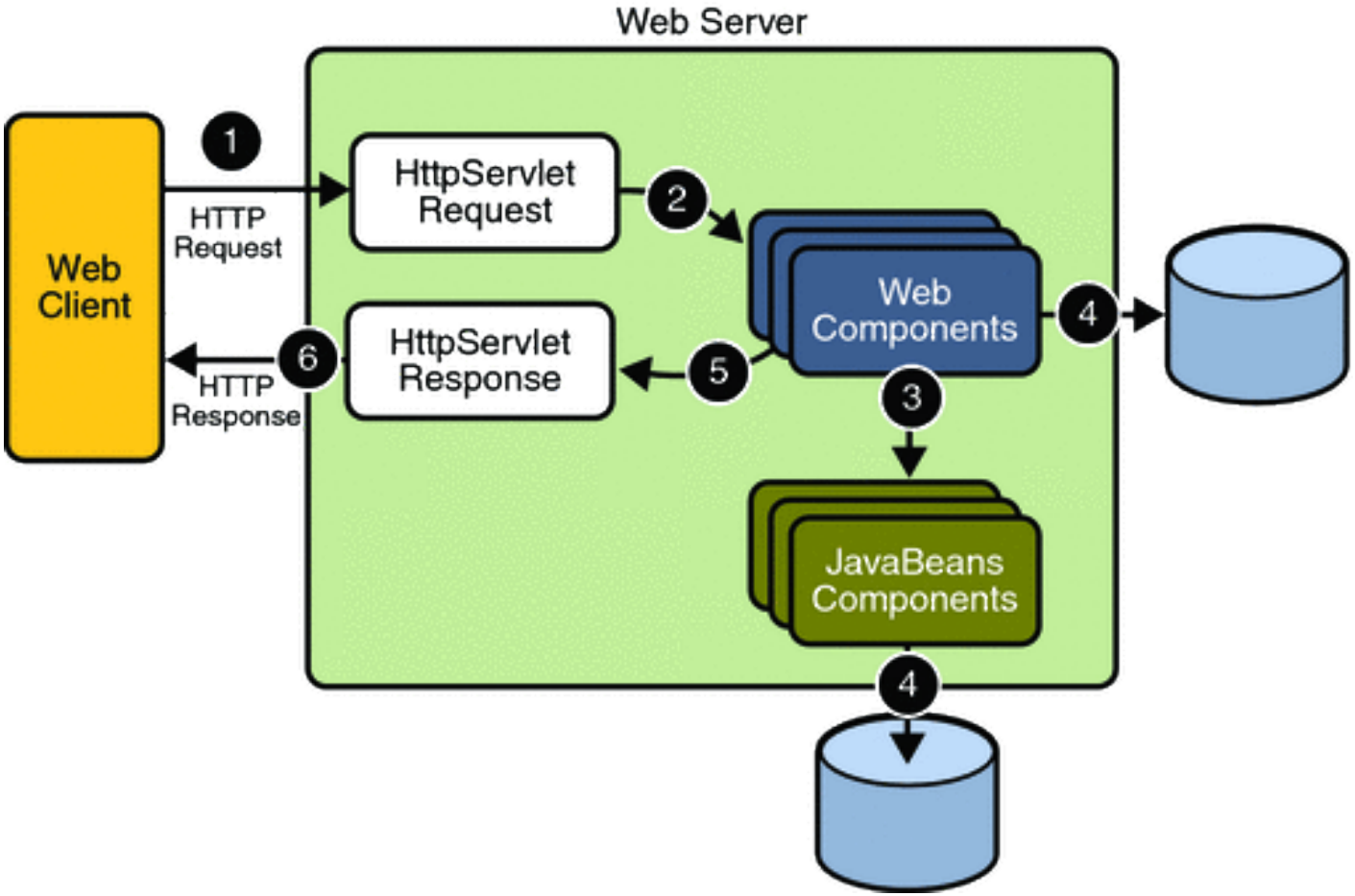
Aplikace v Java Platform, Enterprise Edition může být jeden Java EE modul, ale i skupina modulů zabalená do EAR archivu s popisem rozmístění (deployment) na více výpočetních uzlů, např.

- Web moduly – JSP a servlety ve WAR archivu,
- EJB moduly – JavaBeans zabalené v JAR archivu,
- moduly konektorů a adaptéry zdrojů – J2EE Connector Architecture v RAR archivech,
  (implementují knihovny pro napojení na podnikové informační systémy)
- JAR archivy spustitelné, s využívanými třídami a komponentami aplikace, atd.

Balíky Java EE rozšiřující Java SE

javax.servlet.*	rozhraní pro servlety a zpracování HTTP požadavků,
javax.ejb.*	JavaBeans API pro persistenci, vzdálené volání, souběžnost, a kontrolu přístupu poskytované kontejnerem distribuovanému objektu EJB.
javax.naming	API pro Java Naming and Directory Interface (JNDI).
java(x).sql	API pro Java Database Connectivity (JDBC).
java.transaction.*	rozhraní Java Transaction API (JTA).
javax.xml.*	rozhraní pro parsování a manipulaci s XML (JAXP API).
javax.jms.*	API pro Java Message Service (JMS).
javax.jws.*	rozhraní pro webové služby (JAX-WS API).

Java Servlety podle [SUN, 2007]



servlety: objekty dynamicky zpracují požadavky a generují odpovědi,

JSP stránky: textové dokumenty spouštěné jako servlety.

Životní cyklus Java Servletu

1. pokud neexistuje instance servletu, webový kontejner nahraje třídu, vytvoří její instanci a zavolá metodu `init` servletu,
2. webový kontejner zavolá metodu `service` servletu a předá jí objekty požadavku a odpovědi,
3. servlet čte explicitní a implicitní data v požadavku klienta,
   (HTTP data a hlavičky požadavku)
4. servlet zpracovává data a generuje výsledek,
5. servlet odešle explicitní a implicitní data v odpovědi klientovi,
   (HTTP data, stavový kód a hlavičky odpovědi)
6. pokud webový kontejner potřebuje odstranit servlet, zavolá jeho metodu `destroy`.

Java Servlet a sdílení dat

Spolupracující webové komponenty sdílejí data prostřednictvím atributů objektů:

1. javax.servlet.ServletContext – společný pro webový kontejner, kde běží aplikace,
2. javax.servlet.http.HttpSession – společná pro obsluhu požadavků ve stejné „HTTP session“,
3. javax.servlet.ServletRequest – společný pro obsluhy jednoho požadavku (vícevláknové zpracování požadavku),
4. javax.servlet.jsp.JspContext – společný pro jednu JSP stránku (objekt vlastní JSP stránka).

Další možnost je použít JavaBeans objekty, předávání dat přes databázi nebo objekty se `synchronized` metodami.

Zpracování požadavku v Java Servletu

1. pomocí metody `service` rozhraní GenericServlet,
2. pomocí metod `doMethod`[1] rozhraní HttpServlet,
3. nebo specifickými metodami protokolu, který obsluhuje třída implementující rozhraní Servlet.

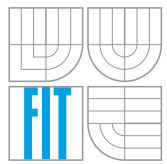
HTTP servlety obecně po přečtení parametrů požadavku získají „output stream“ pro odpověď, vyplní hlavičky odpovědi, a zapíší data odpovědi.

- parametry požadavku získány pomocí `getParameter` objektu s rozhraním HttpServletRequest (obecně ServletRequest),
- odpověď pomocí metod `setContentType`, `setBufferSize`, `getWriter` objektu s rozhraním HttpServletResponse (obecně ServletResponse)

[1] konkrétně: `doGet`, `doDelete`, `doOptions`, `doPost`, `doPut` a `doTrace`

Ukázka Java Servletu (a vložení zdroje)

```
public class MyServlet extends HttpServlet {
  public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // set headers before accessing the Writer
    response.setContentType("text/html");
    response.setBufferSize(8192);
    PrintWriter out = response.getWriter();
    // then write the response
    out.println("<html><head><title>MyServlet</title></head>");
    // Get the dispatcher; it gets something to the user
    RequestDispatcher d = getServletContext().getRequestDispatcher("/url");
    if (d != null) d.include(request, response);
    // Get the request parameter to display
    String myParam = request.getParameter("myParam");
    if (myParam != null) { // Print the information obtained
      out.println("MyParam is " + myParam);
      try { ... } catch (MyException e) {
        response.resetBuffer(); throw new ServletException(e);
    } }
    out.println("</body></html>"); out.close();
} }
```



Ladění Java Servletu

Detekce chyb a sledování běhu je u servletu složitější, protože běží jako komponenta ve webovém kontejneru a běží ve vícevláknovém prostředí.

- tisk ladících zpráv na konzoli,
- Apache Log4J,
- integrovaný ladící nástroj v IDE,
- ladící zprávy do HTML kódu,
- chybové kódy v odpovědi klientovi,
- ladící zprávy do log souboru,
  (`log("message")` nebo `log("message", Throwable)`)
- a další.

Servlet vs. JSP stránka

Vlastnosti JSP stránky:

- \+ oddělení vzhledu rozhraní od aplikační logiky,
- \+ lze použít klasické HTML nástroje pro design,
- \+ rozdělení rolí v týmu na programátory a designery,
- – servlet je rychlejší a méně zatěžuje server,
- – složitější kód implementuje servlet přehledněji,
- – servlet se lépe testuje (samostatně, „blackbox“),
- – servlet se lépe ladí (krokování, snazší sledování proměnných).

→ ve webové aplikaci kombinovat JSP a servlety.

Syntax JSP stránky

HTML text `<h1>blah</h1> <!-- comment -->`

JSP komentář `<%-- comment --%>`

JSP výraz `<%= expression %>`

skriptovací tag `<% java code; %>` nebo
`<jsp:scriptlet>java code;</jsp:scriptlet>`

deklarační tag `<%! type var = val; %>` nebo
`<jsp:declaration>type var = val;</jsp:declaration>`

JSP akce `<jsp:action params/>`

JSP akce jsou XML tagy pro komunikaci s webovým kontejnerem.

jsp:param přidá parametr mezi parametry požadavku,

jsp:include dočasně předá řízení servletu daného URI,

jsp:forward trvale předá řízení servletu daného URI,

jsp:useBean zpřístupní JavaBean pro tuto JSP stránku,

jsp:getProperty získá parametr daného JavaBean,

jsp:setProperty nastaví parametr daného JavaBean.

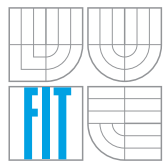
Životní cyklus JSP stránky

1. webový kontejner zkontroluje zastaralost servletu v porovnání s JSP stránkou, pokud je servlet starší, znovu převede JSP stránku na třídu servletu a tu zkompiluje,
2. spustí se servlet vzniklý překladem JSP stránky.

   (`init` a `destroy` využívány servletem, JSP str. smí použít `jspInit` a `jspDestroy`)

Statická data jsou přeložena na kód produkující tato data.

- fragmenty java kódu vloženy do metody `_jspService` servletu, „directives“ mohou řídit překlad JSP na servlet a jeho spuštění,
- JSP výrazy předány v parametrech volání příslušného procesoru,
- `jsp:[set|get]Property` jsou převedeny na volání metod JavaBeans komponent,
- `jsp:[include|forward]` převedeny na volání Java Servlet API,
- `jsp:plugin` jsou převedeny na aktivaci appletu,
- uživatelské tagy jsou předány obsluze v implementaci tagů.



Dynamický kód JSP stránky

JSP stránka může používat:

1. **implicitní objekty** vytvořené a spravované webovým kontejnerem ve vztahu k požadavku, stránce, session a aplikaci,
   (pageContext, servletContext, session, request, response, param, paramValues, header, headerValues, cookie, initParam, pageScope, requestScope, sessionScope a applicationScope)
2. **objekty aplikace** zapouzdřující aplikační logiku, např. v JavaBenas objektech,
   (objekty jsou vytvářeny a používány pomocí JSP tagů, stejně tak jsou nastavovány parametry (atributy) objektů)
3. **sdílené objekty** poskytující přístup ke sdíleným zdrojům.
   (pozor na souběžný přístup k objektům, paralelismus v rámci provádění JSP stránky lze redukovat pomocí `<%@ page isThreadSafe="true|false" %>`)

Ukázka JSP stránky (a použití JavaBeans) I

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="/functions" prefix="f" %>

<html><head><title>Localized Dates</title></head><body bgcolor="white">
<jsp:useBean id="locales" scope="application" class="mypkg.MyLocales"/>

<form name="localeForm" action="index.jsp" method="post">
<c:set var="selectedLocaleString" value="${param.locale}" />
<c:set var="selectedFlag" value="${!empty selectedLocaleString}" />
<b>Locale:</b>
<select name=locale>
<c:forEach var="localeString" items="${locales.localeNames}" >
  <c:choose>
    <c:when test="${selectedFlag}"><c:choose>
      <c:when test="${f:equals(selectedLocaleString, localeString)}" >
        <option selected>${localeString}</option>
      </c:when>
      <c:otherwise><option>${localeString}</option></c:otherwise>
```

Ukázka JSP stránky (a použití JavaBeans) II

```
    </c:choose></c:when>
    <c:otherwise><option>${localeString}</option></c:otherwise>
  </c:choose>
</c:forEach>
</select>
<input type="submit" name="Submit" value="Get Date">
</form>

<c:if test="${selectedFlag}" >
  <jsp:setProperty name="locales" property="selectedLocaleString"
    value="${selectedLocaleString}" />
  <jsp:useBean id="date" class="mypkg.MyDate"/>
  <jsp:setProperty name="date" property="locale"
    value="${locales.selectedLocale}"/>
  <b>Date: </b>${date.date}
</c:if>

</body></html>
```

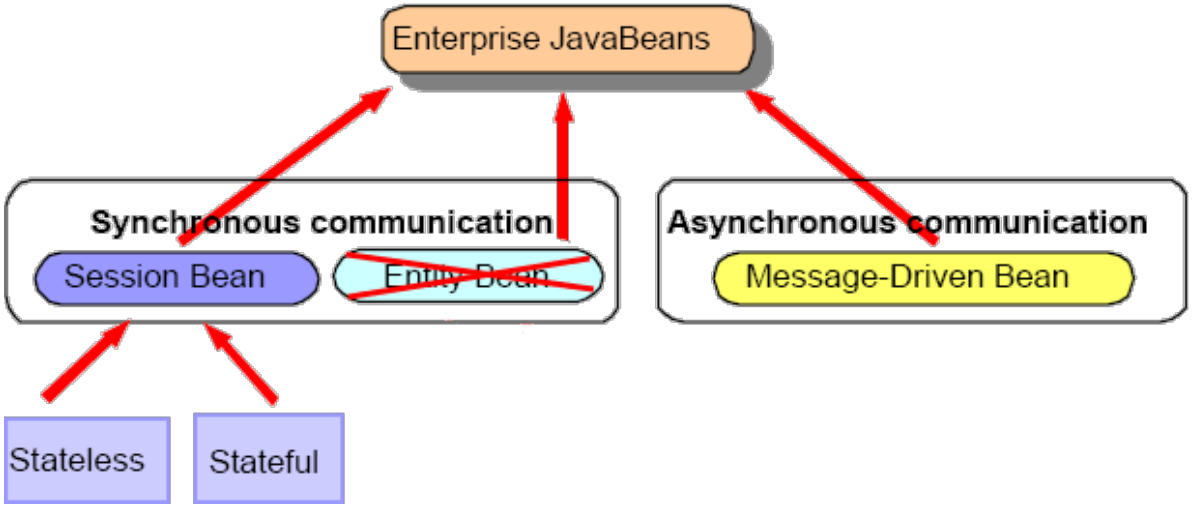
Enterprise JavaBeans (EJB)

- Serverové SW komponenty implementované v jazyce Java.
- Podporují znovupoužitelnost a hierarchickou (de)kompozici.
- Podporují konfiguraci v „deployment-time“.
  (rozmístění na různé výpočetní uzly, operační systémy, JavaEE servery, atd.)
- Musí vyhovovat specifikaci JavaBeans objektů.
  (dané pojmenování, konstrukce, chování, atd.)
  - třída musí být serializovatelná (schopná persistence),
  - třída musí mít bezparametrický konstruktor,
  - atributy musí být přístupné pomocí `get...` a `set...` metod,
  - přístup ke sdíleným zdrojům musí podporovat souběžnost.
    (tzv. „thread-safety“, toto je vhodné pro podporu atomických transakcí)

Session Beans a Message-Driven Beans

V Java EE 5 jsou dva typy JavaBeans:

- **session beans**: zpracovávají data v session (synchronní),
- **message-driven beans**: přidává navíc asynchronní reakce na události (odpověď nenásleduje ihned po dotazu).



Session Beans

- komponenta uvnitř aplikačního serveru poskytující službu externím klientům a ostatním komponentám serveru,
- vždy zpracovává jen jedenu sérii požadavků pro jednoho uživatele (session) a není persistentní.

Rozlišujeme stavové a bezstavové beans:

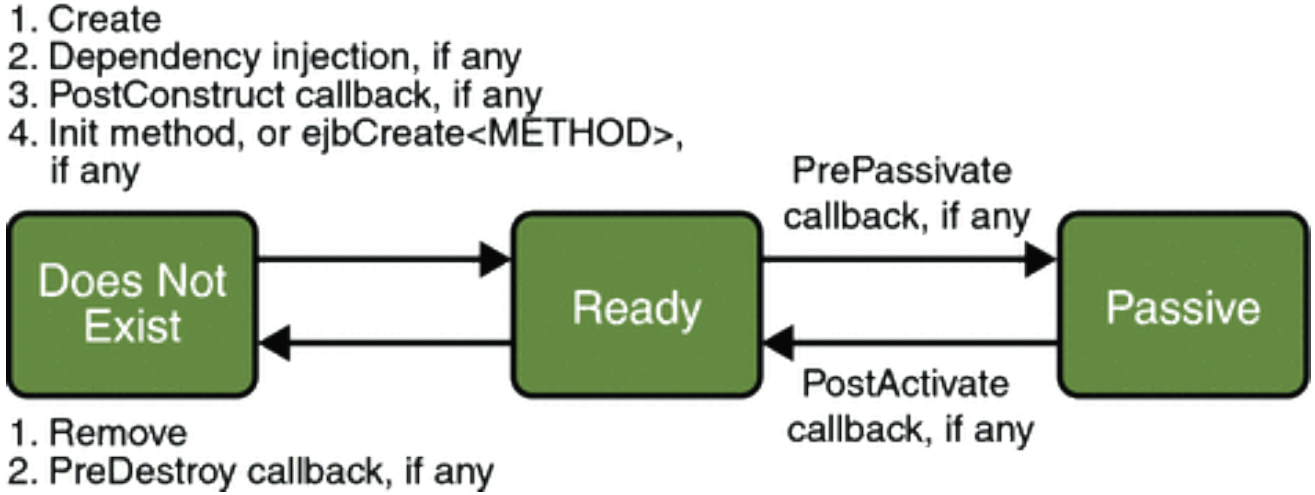
stateful: stav je zachován během celé session (konverzace),

stateless: stav je udržován pouze pro jeden dotaz (volání metody), mizí po skončení metody.
(používá se omezený počet instancí beans (tzv. „pool“) a stav přežije do dalšího volání stejné instance; nicméně server může další požadavek přiřadit jakékoliv instanci)

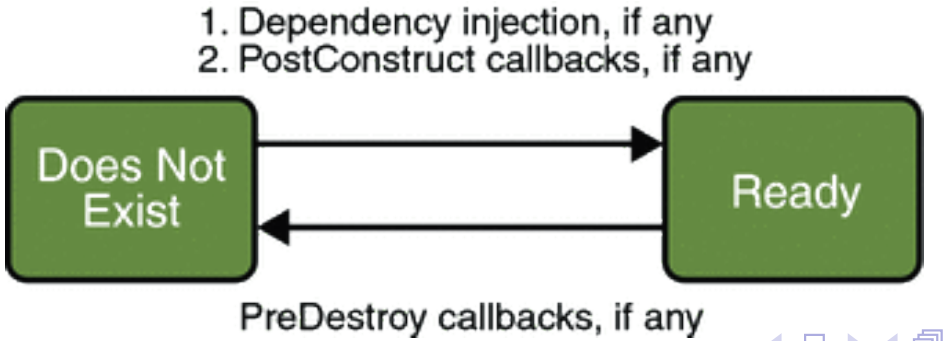
- bezstavové beans jsou vhodnější pro větší zátěže,
(lze uchovávat menší počet instancí (pool) než v případě stavových beans)
- stavové beans nemohou implementovat webové služby.

Životní cyklus Session Beans podle [SUN, 2007]

Stateful Session Bean



Stateless Session Bean



Ukázka Session Bean (a deklarace rozhraní)

```
import java.math.BigDecimal; import javax.ejb.*;

@Remote
public interface Converter {
  public BigDecimal dollarToYen(BigDecimal dollars);
  public BigDecimal yenToEuro(BigDecimal yen);
}

@Stateless
public class ConverterBean implements Converter {
  private BigDecimal yenRate = new BigDecimal("115.3100");
  private BigDecimal euroRate = new BigDecimal("0.0071");
  public BigDecimal dollarToYen(BigDecimal dollars) {
    return dollars.multiply(yenRate).setScale(2, BigDecimal.ROUND_UP);
  }
  public BigDecimal yenToEuro(BigDecimal yen) {
    return yen.multiply(euroRate).setScale(2, BigDecimal.ROUND_UP);
  }
}
```

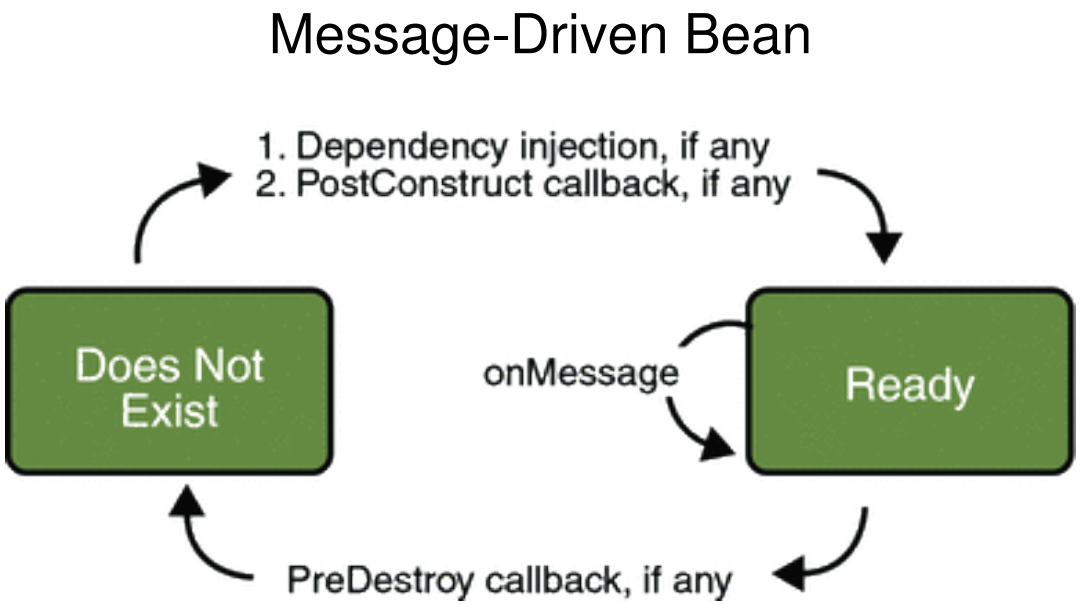
Message-Driven Beans

- podobné vlastnosti jako nestavové session beans, ale jsou volány asynchronně,
- naslouchají Java Message Service (JMS) zprávám[2],
  (podobně jako se realizuje naslouchání událostem, zde metodou `onMessage`)
- přijímají zprávy od ostatních Java EE komponent nebo od JMS aplikace, vč. systémů nepoužívajících Java EE platformu.

- použití při asynchronní komunikaci,
  (reakce na události, na které není potřeba přímá odpověď),
- mohou realizovat transakční (atomické) zpracování,
  (metoda `onMessage` je jedna transakce, která může být odvolána)
- JMS zprávy by neměly být zpracovávány synchronně.
  (hrozí blokování zdrojů serveru, asynchronní zpracování je rychlejší)

[2] může však zpracovávat i jiný typ zpráv, např. volání webových služeb

Životní cyklus Message-Driven Beans [SUN, 2007]



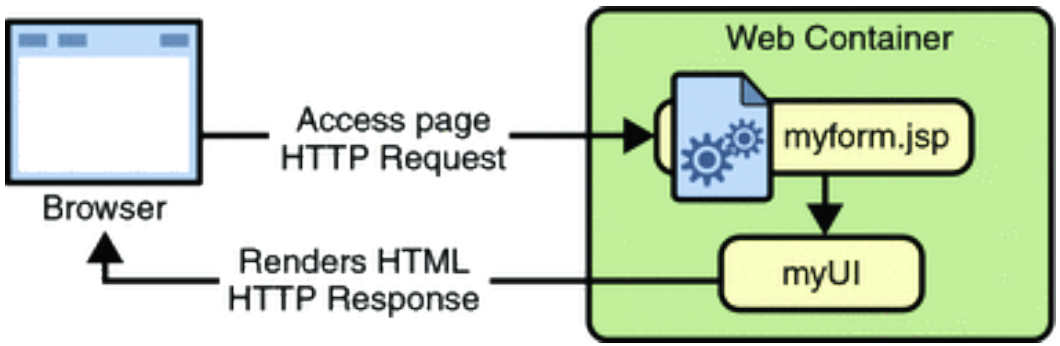
Ukázka Message-Driven Beans (a deklarace rozhraní)

```
@MessageDriven(mappedName="jms/Queue")
public class SimpleMessageBean implements MessageListener {
  @Resource
  private MessageDrivenContext mdc;
  public void onMessage(Message inMessage) {
    TextMessage msg = null;
    try {
      if (inMessage instanceof TextMessage) {
        msg = (TextMessage) inMessage;
        logger.info("MESSAGE BEAN: Message received: "+msg.getText());
      } else {
        logger.warning("Message wrong type: "+inMessage.getClass().getName());
      }
    } catch (JMSException e) {
      e.printStackTrace();
      mdc.setRollbackOnly();
    } catch (Throwable te) {
      te.printStackTrace();
  } }
}
```

JavaServer Faces podle [SUN, 2007]

Zahrnuje dvě části:

1. API pro reprezentaci a změny stavu UI komponent,
   (obsluha událostí, validace a konverze dat na straně serveru, definice navigačních prvků, podpora lokalizace, atd.)
2. dvě knihovny JSP tagů pro vizualizaci prvků UI v JSP stránkách a jejich napojení na serverové objekty JavaServer Faces.
   (umožňuje napojit serverový kód na události UI u klienta, vhodně reprezentovat data ze serveru v UI u klienta, uplatnit principy znovupoužitelnosti a rozšiřitelnosti na části UI aplikace, uchovat stav UI a předávat ho mezi požadavky, apod.)



Použití JavaServer Faces

Aplikace používá JSF následovně:

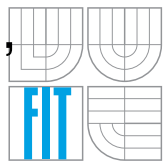
- uživatelské rozhraní tvoří JSP stránky s tagy z JSF knihoven,
- obsluhu uživatelských rozhraní.

  (realizují JSF komponenty mapované na tagy v JSP stránkách, obsluhy událostí, validátory a převodníky dat registrované na komponentách, a JavaBean komponenty zapouzdřující data a aplikačně specifickou funkčnost komponent)

Aplikace tedy obsahuje:

- sadu JSP stránek a sadu JavaBeans (backing bean) definující UI komponenty na stránkách,
- konfigurační soubory definující navigační pravidla a konfigurují JavaBeans,
- popis rozmístění (deployment descriptor),
- soubor uživatelských objektů, např. uživatelských komponent UI, validátorů a převaděčů dat, a obsluhy událostí.

  (vč. souboru uživ. tagů pro reprezentaci uživ. komponent v JSP stránkách)



Ukázka JSP s JavaServer Faces

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<f:view><h:form id="helloForm1">
  <h2>Hi. My name is Duke. I'm thinking of a number from
    <h:outputText lang="en_US" value="#{UserNumberBean.minimum}"/> to
    <h:outputText value="#{UserNumberBean.maximum}"/>.
    Can you guess it?</h2>
  <h:graphicImage id="waveImg" url="/wave.med.gif" />
  <h:inputText id="userNo" label="User Number"
    value="#{UserNumberBean.userNumber}">
    <f:validateLongRange minimum="#{UserNumberBean.minimum}"
      maximum="#{UserNumberBean.maximum}" />
  </h:inputText>
</h:form></f:view>
```

Ukázka JavaBeans pro JavaServer Faces

```
Integer userNumber = null;
Integer minimum = 0;
Integer maximum = 10;
...
public void setUserNumber(Integer user_number) {
  userNumber = user_number;
}
public Integer getUserNumber() {
  return userNumber;
}
public String getResponse() {
  if(userNumber != null && userNumber.compareTo(randomInt) == 0) {
    return "Yay! You got it!";
  } else {
    return "Sorry, "+userNumber+" is incorrect.";
  }
}
```

Java Persistence API

Poskytuje objektově-relační mapování Java objektů. Zahrnuje:

1. aplikační rozhraní Java Persistence API
2. dotazovací jazyk,
3. metadata popisující objektově-relační mapování.

Persistentní třída (entity class) musí splňovat:

- musí být označena jako `javax.persistence.Entity`,
- musí mít `pulic/protected` bezparametrický konstruktor,
- nesmí být deklar. jako `final`, ani její metody a persit. atributy,
- pokud budou instance třídy předávány vzdáleným objektům, musí implementovat rozhraní `Serializable`,
- persistentní atributy musí být deklarovány jako `private/protected/package-private` a přistupovány výhradně pomocí metod třídy (`get...` a `set...` metody),
- může rozšiřovat i non-entity třídu a může být rozšířena na non-entity třídu.

Datové typy pro Java Persistence

Persistentní mohou být typy:

- primitivní typy a jejich „wrappers“,
- `java.lang.String`, `java.math.BigInteger` a `BigDecimal`, `java.util.Date` a `Calendar`, `java.sql.Date`, `Time` a `TimeStamp`,
- uživatelské serializovatelné typy,
- pole `byte[]`, `Byte[]`, `char[]` a `Character[]`,
- výčtové typy, enitity a kolekce entit,
- vnořené třídy v souladu s Java Persistence API.

Pro persistentní atributy musí jejich mateřská entita implementovat přístupové metody[3].

[3] `isProperty` pro typ boolean, jinak obecně `Type getProperty()` a `void setProperty(Type type)`

Mapování v Java Persistence

Jsou podporovány různé anotace pro specifikaci vlastností:

@Entity	persistentní třída,
@Transient	nepersistentní atributy,
@Id	primární klíče (po relačním mapování atributu),
@OneToOne	typ vazby 1:1 (po relačním mapování vztahu),
@OneToMany	typ vazby 1:N (po relačním mapování vztahu),
@ManyToMany	typ vazby N:M (po relačním mapování vztahu),
@MappedSuperclass	označení nepersitentních nadtříd s definicí peristence,
@Inheritance	mapování dedičnosti (možnosti SINGLE_TABLE, JOINED a TABLE_PER_CLASS).

Ukázka Java Persistence

```
@Entity
public class Part {
  ...
  @Id
  public int getItemId() {
    return itemId;
  }
  @OneToMany(cascade=ALL, mappedBy="order")
    public Collection<LineItem> getLineItems() {
    return lineItems;
  }
  @ManyToOne
    public VendorPart getVendorPart() {
    return vendorPart;
  }
  @Temporal(TIMESTAMP)
  public Date getLastUpdate() {
    return lastUpdate;
  }
  ...
}
```

Literatura

SUN (2007).

The Java™ EE 5 Tutorial.

Sun Microsystems, 4150 Network Circle, Santa Clara, CA 95054, U.S.A.

Šeda, J. (2003).

J2EE, .NET a vývoj rozsáhlých systémů.

interval.cz.