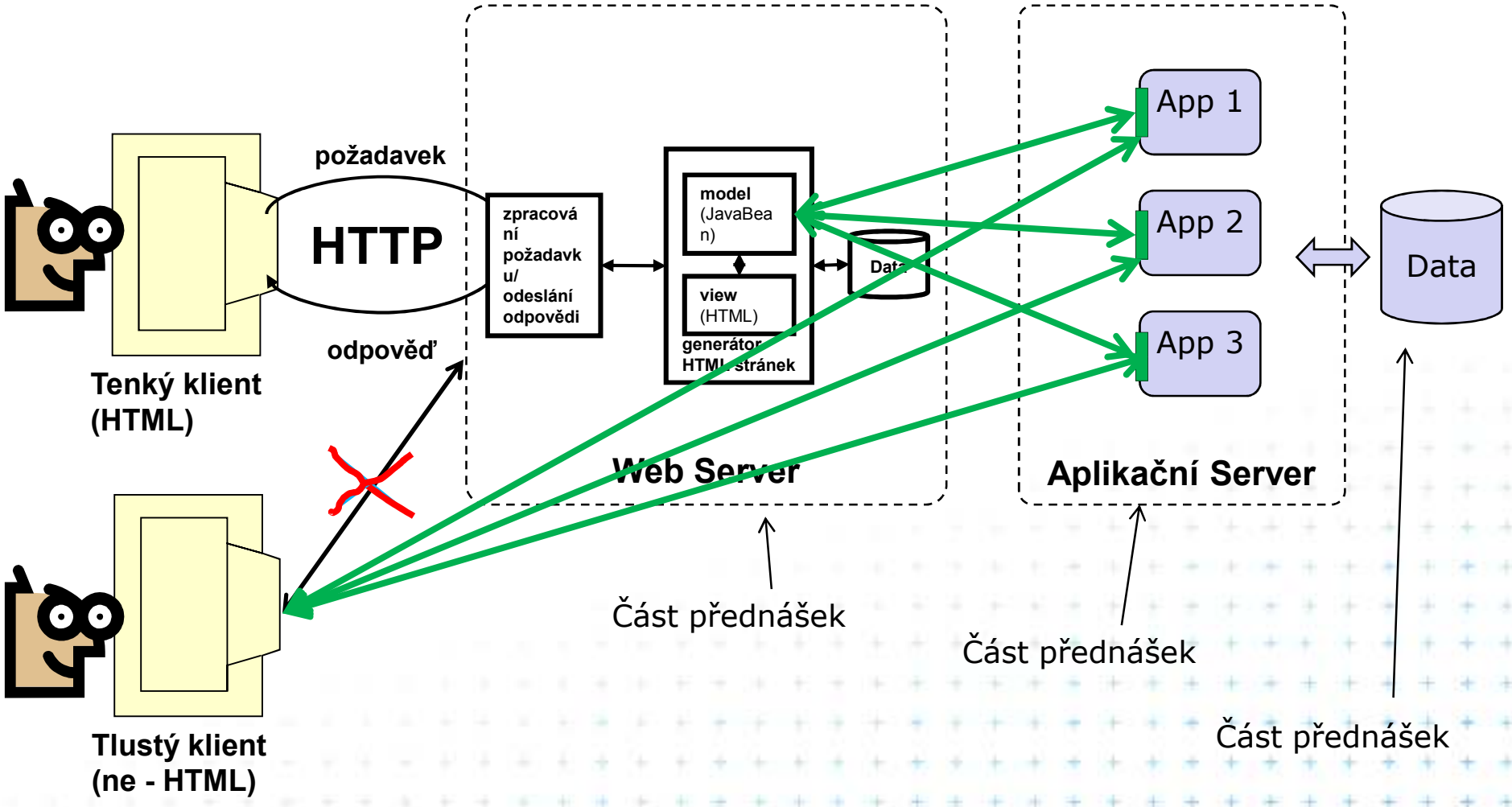
Tvorba Webu 2

Enterprise Beans

Martin Klíma

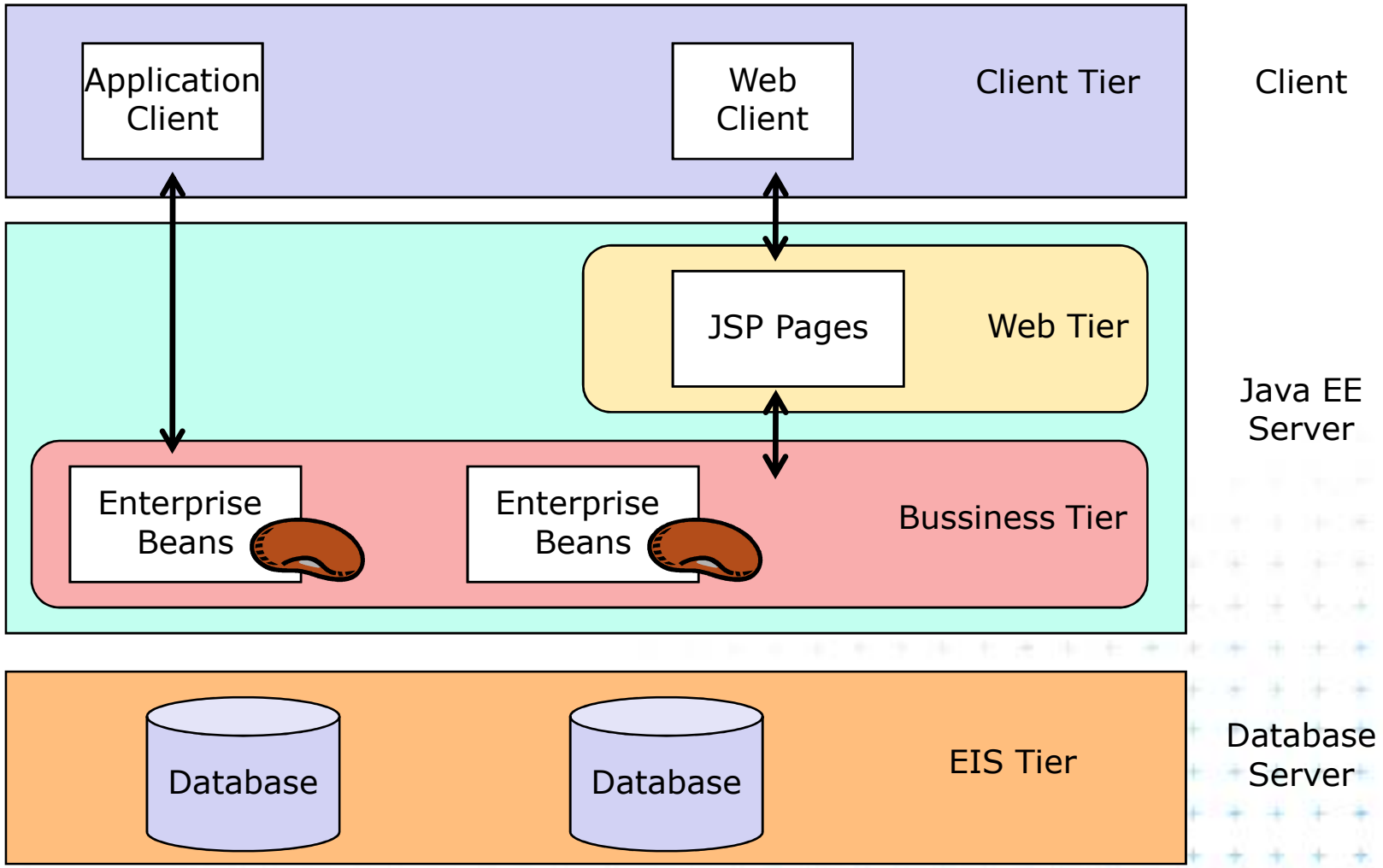
Architektura webové aplikace – aplikační server



Java

- Java ME = Java Micro Edition
  - mobilní aplikace, omezený rozsah funkcí
- Java SE = Java Standard Edition
  - desktopové aplikace
- Java EE = Java Enterprise Edition
  - webové aplikace, aplikační servery

Java EE


Application Client
Web Client
Client Tier
Client
JSP Pages
Web Tier
Java EE Server
Enterprise Beans
Enterprise Beans
Bussiness Tier
Database
Database
EIS Tier
Database Server

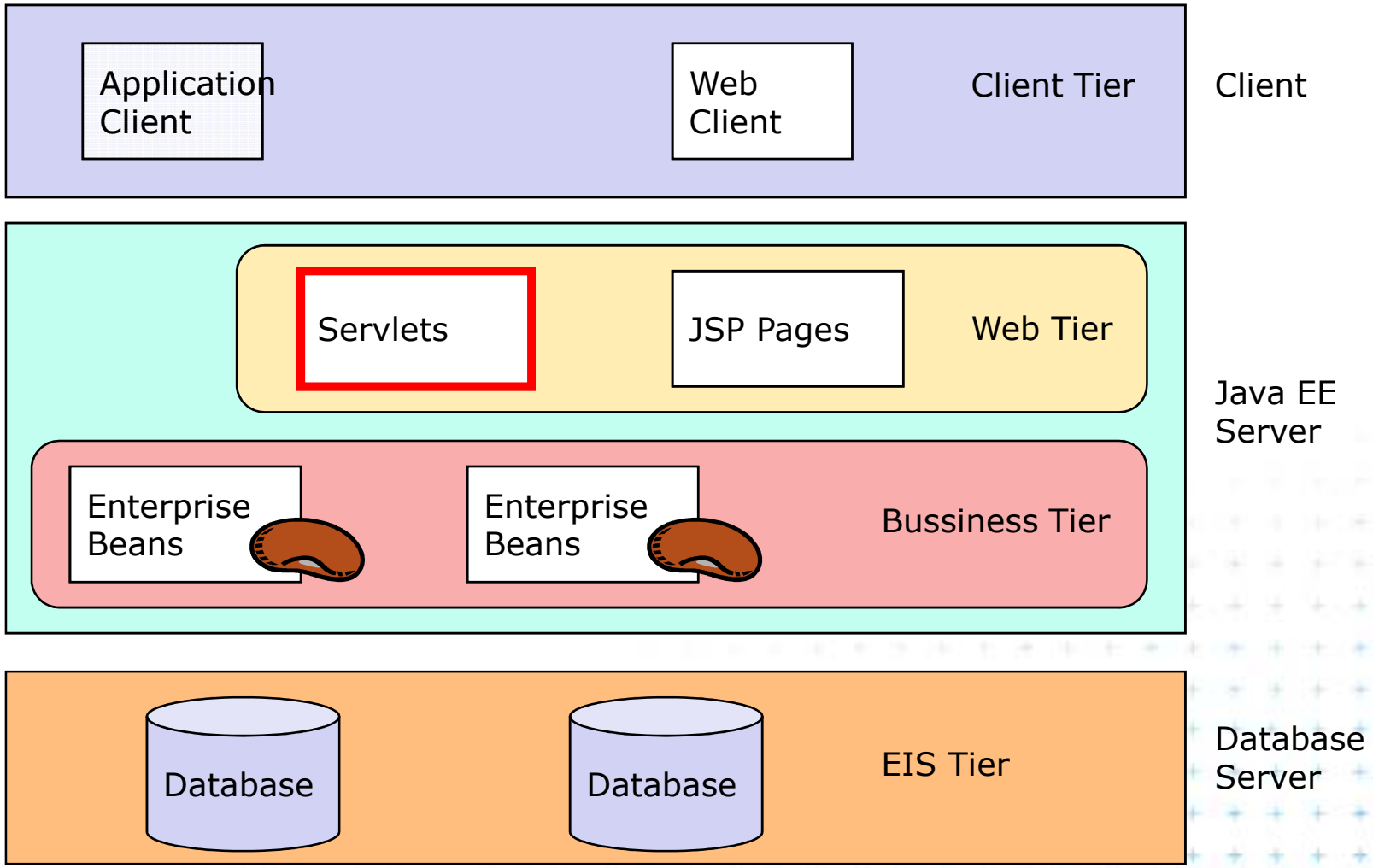
Sestavení a deployment

- Každá JEE aplikace je zabalena do standardního formátu

- Obsahuje
  - Funkční komponenty jako EJB, JSP, servelty, applety, statické stránky, …
  - Deskriptor

- Když je to zabalené, je možné to rozjet na kterémkoli serveru splňující specifikaci kontejneru.

Servlet


Application Client
Web Client
Client Tier
Client
Servlets
JSP Pages
Web Tier
Java EE Server
Enterprise Beans
Enterprise Beans
Bussiness Tier
Database
Database
EIS Tier
Database Server

Servlety

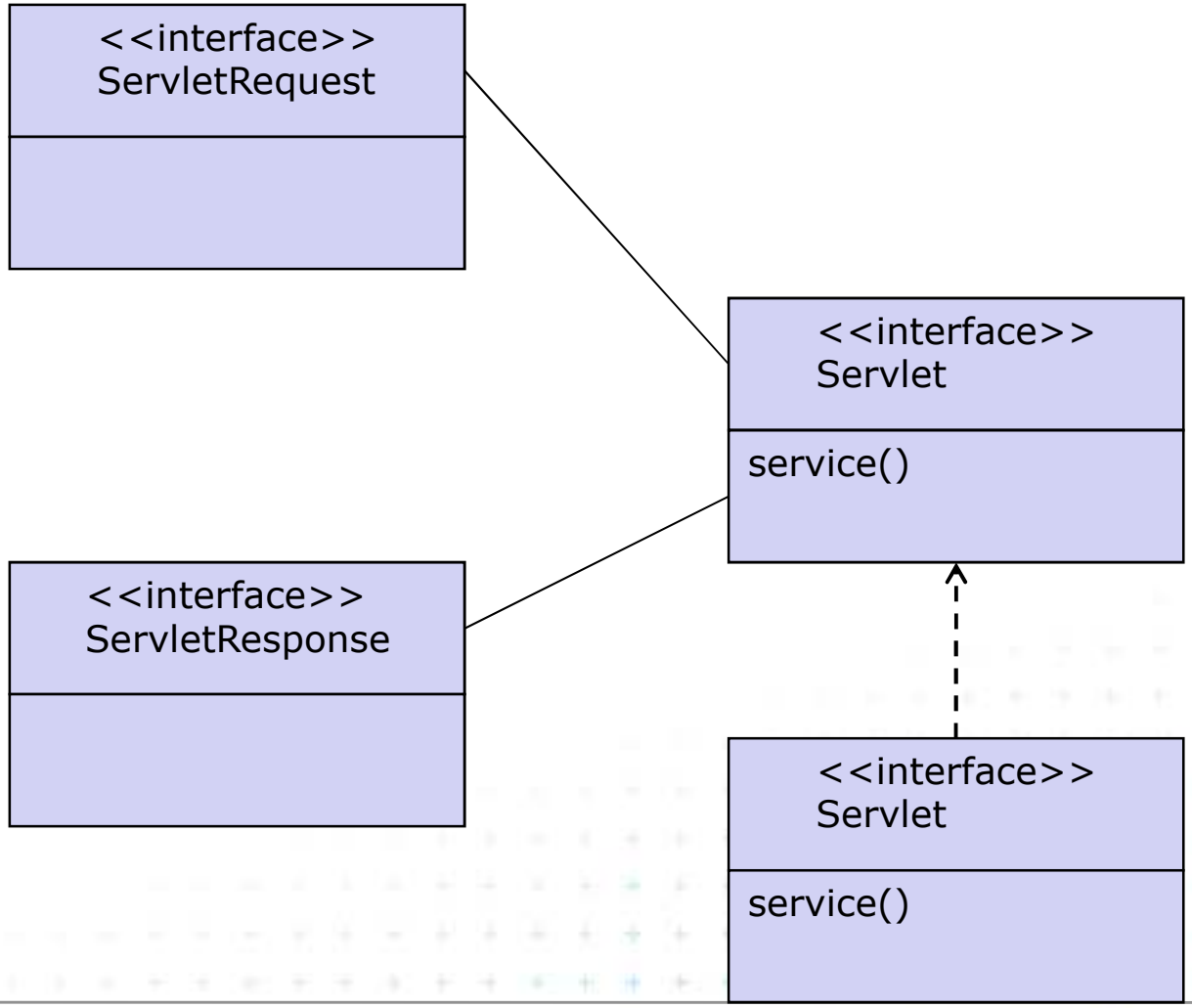
- Servlet je třída, která rozšiřuje schopnost serveru obsloužit http požadavek.
  - tím docilujeme možnosti generovat dynamické webové stránky

- Všechny servlety musí implementovat ***javax.servlet.Servlet*** interface, který definuje metody potřebné pro životní cyklus ve web kontejneru.

- Nejjběžnější je rozšířit třídu javax.servlet.http.HTTPServlet.

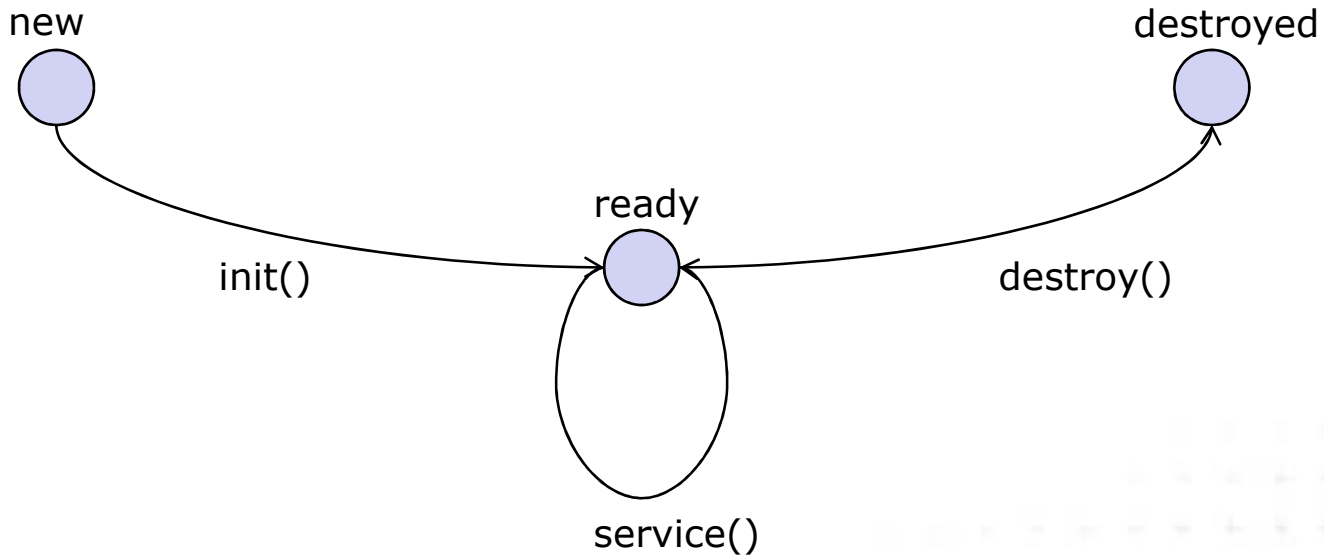
- Servlet je mapován na URL (jedno nebo více).
  - pomocí souboru web.xml

Servlets API



Životní cyklus servletu

- Cyklus řídí kontejner



Obsluhucící metoda - service

```
java.lang.Object
  |
  +--javax.servlet.GenericServlet          A
       |
       +--javax.servlet.http.HttpServlet   A
```

- GenericServlet má obsluhující metodu service
- HTTPServlet implementuje toto metodu a volá konkrétní obluhující metody podle metody HTTP requestu
  - doGet
  - doPost
  - doPut
  - doDelete
  - doOptions
  - doHead
  - doTrace

Parametry servisních metod

Získání informací z dotazu

- z objektu typu ServletRequest resp. HttpServletRequest
- viz http://java.sun.com/javaee/5/docs/api/javax/servlet/ServletRequest.html a http://java.sun.com/javaee/5/docs/api/javax/servlet/http/HttpServletRequest.html

HttpServletRequest – zajímavé metody

- Object getParameter(String name)
- void setParameter(String name, Object o)
- a mnoho dalších

Parametry servisních metod

Zapsání informací do odpovědi

- do objektu typu ServletResponse resp. HttpServletResponse
- viz http://java.sun.com/javaee/5/docs/api/javax/servlet/ServletResponse.html a http://java.sun.com/javaee/5/docs/api/javax/servlet/http/HttpServletResponse.html

HttpServletResponse – zajímavé metody

- addHeader(String name, String value)
- PrintWriter getWriter()
- void addCookie(Cookie cookie)
- a mnoho dalších

Parametry z formuláře


```
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
        try {
         out.println("<html>");
         out.println("<head>");
         out.println("<title>Servlet FirstServlet</title>");
         out.println("</head>");
         out.println("<body>");

         String jmeno = request.getParameter("nickname");
            if (null != jmeno) {
                out.println("Jméno");
            }

         out.println("</body>");
         out.println("</html>");
        } finally {
            out.close();
        }
```


Obvyklý postup - výroba odpovědi

- Získat output stream
  - pro výstup textu se hodí PrintWriter
  - pro binární data ServletOutputStream
- Nastavit typ obsahu, např. text/html, tzv. MIME type
  - http://www.iana.org/assignments/media-types/
  - metoda setContentType(String type)
- Nastavit bufferování
  - metoda setBufferSize(int size)

```
package cz.cvut.fel;
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
         out.println("<html>");
         out.println("<head>");
         out.println("<title>Servlet FirstServlet</title>");
         out.println("</head>");
         out.println("<body>");
         out.println("Ahoj");
         out.println("</body>");
         out.println("</html>");
        } finally {
            out.close();
        }
    }
}
```

Servlet reaguje na metodu GET

Zápis do výstupního streamu

WEB XML

Web.xml

- Řídí chování web kontejneru

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>FirstServlet</servlet-name>
        <servlet-class>cz.cvut.fel.FirstServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>FirstServlet</servlet-name>
        <url-pattern>/FirstServlet</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>FirstServlet</welcome-file>
        </welcome-file-list>
    </web-app>
```

Jméno servletu

Třída servletu

Web alias servletu

Servlet může mít více jmen

Co se má otevřít defaultně

Web.xml - parametry

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd">
    <context-param>
        <param-name>Author</param-name>
        <param-value>Martin Klíma</param-value>
    </context-param>
    <context-param>
        <param-name>Affiliation</param-name>
        <param-value>ČVUT</param-value>
    </context-param>
    <servlet>
        <servlet-name>FirstServlet</servlet-name>
        <servlet-class>cz.cvut.fel.FirstServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>FirstServlet</servlet-name>
        <url-pattern>/FirstServlet</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>FirstServlet</servlet-name>
        <url-pattern>/PrvniServlet</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>FirstServlet</welcome-file>
        </welcome-file-list>
    </web-app>
```

Komponenty v modulu sdílí init parametry v objektu Context

JAVASERVER PAGES - JSP

JSP

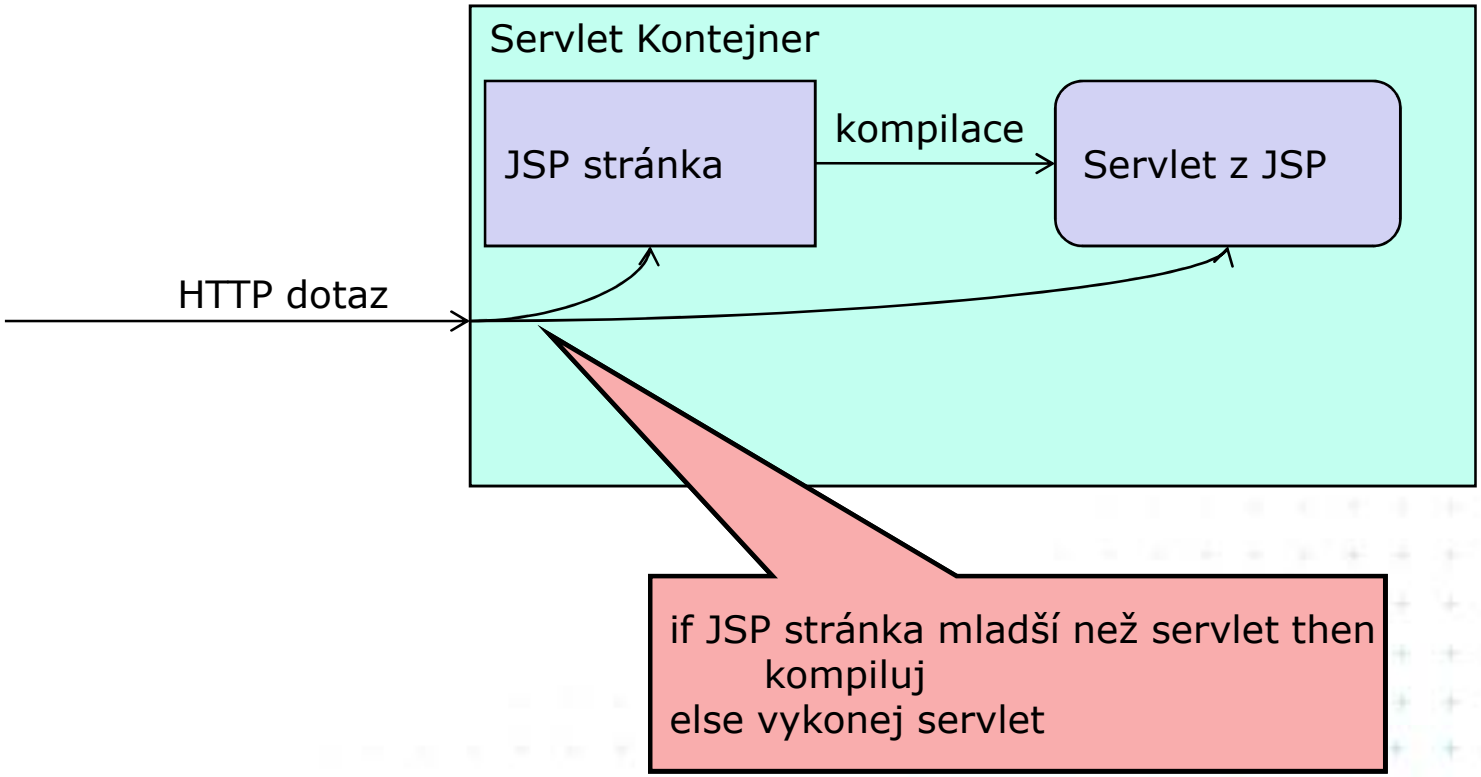
- JSP je textový dokument, který obsahuje
  - statická data, většinou (X)HTML
  - JSP elemety, které generují dynamický obsah

- přípona .jsp
- dvě verze
  - standardní
  - XML
- my budeme používat standardní syntaxi

Nejjednodušší stránka – standardní syntaxe

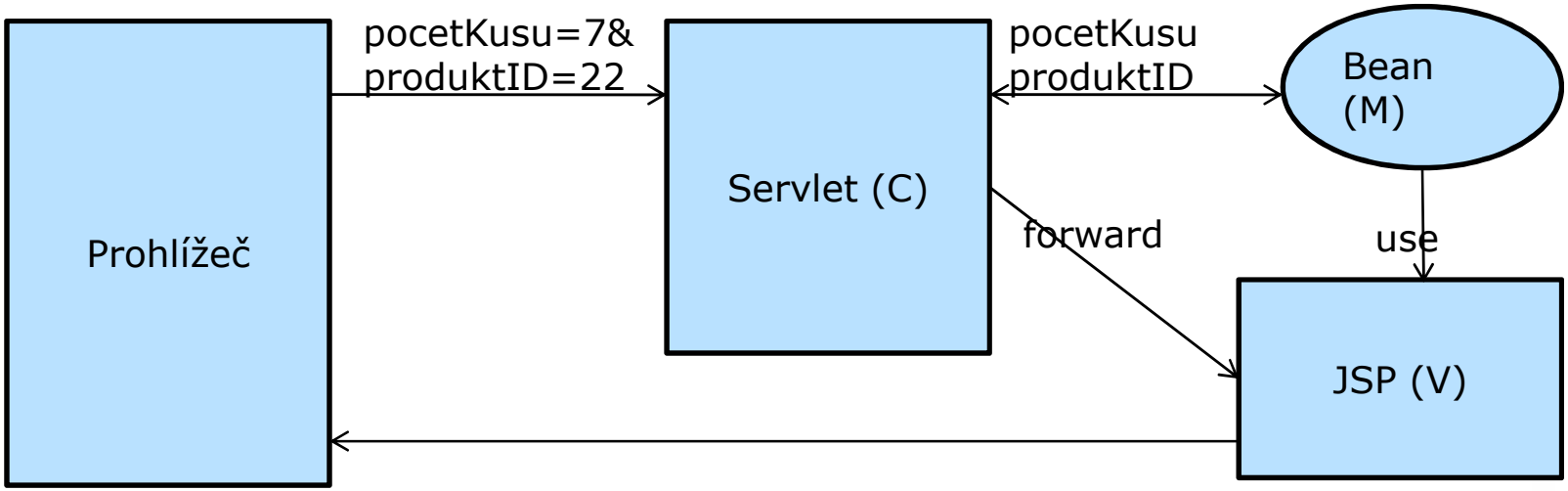
```
<%--
    Document   : stranka1
    Created on : 18.2.2008, 11:44:52
    Author     : xklima
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h2>Hello World!</h2>
    </body>
</html>
```

Zpracování JSP – životní cyklus



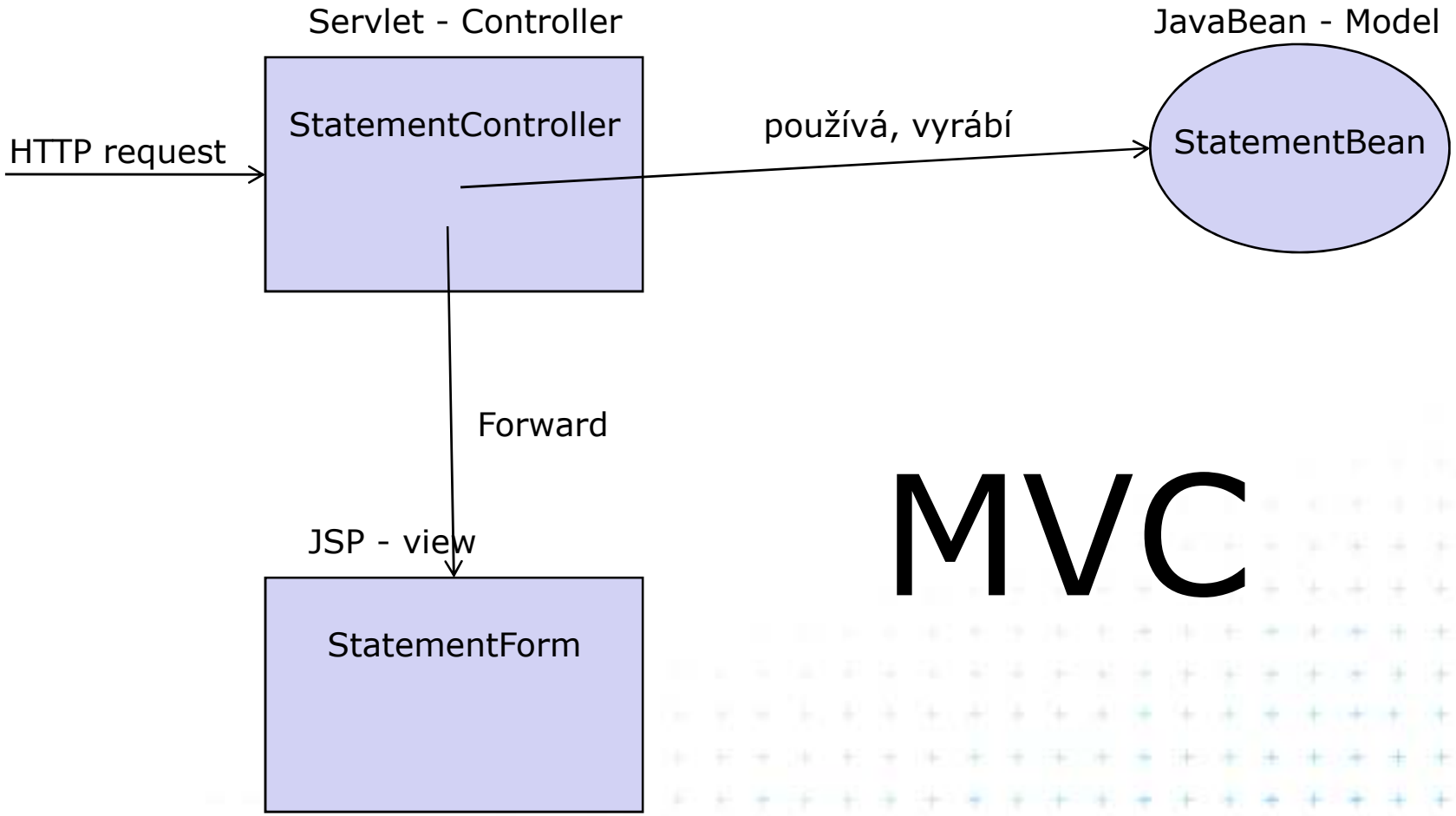
MVC – OPĚT NA SCÉNĚ

Logika (bean) je řízena hodnotami z HTTP dotazu



- Servlet (C) přečte http parametry a zavolá logiku (M).
- Parametry jsou obvykle svázané s formulářem
- Předá řízení JSP

Použiji beanu jako aplikační logiku, model



JavaBeans se dají použít k přenosu informací

```
public class StatementBean implements Serializable {
    protected ArrayList <String> statements;

    public StatementBean(){
        super();
        statements = new ArrayList <String>();
    }

    public synchronized String getStatement(int i) {
        return statements.get(i);
    }

    public synchronized List <String> getAllStatements(){
        return statements;
    }

    public synchronized void addStatement (String statement) {
        if (!statements.contains(statement)) {
            statements.add(statement);
        }
    }

    public synchronized int getStatementCount() {
        return statements.size();
    }
}
```


Formulář pro přidávání statementForm.jsp

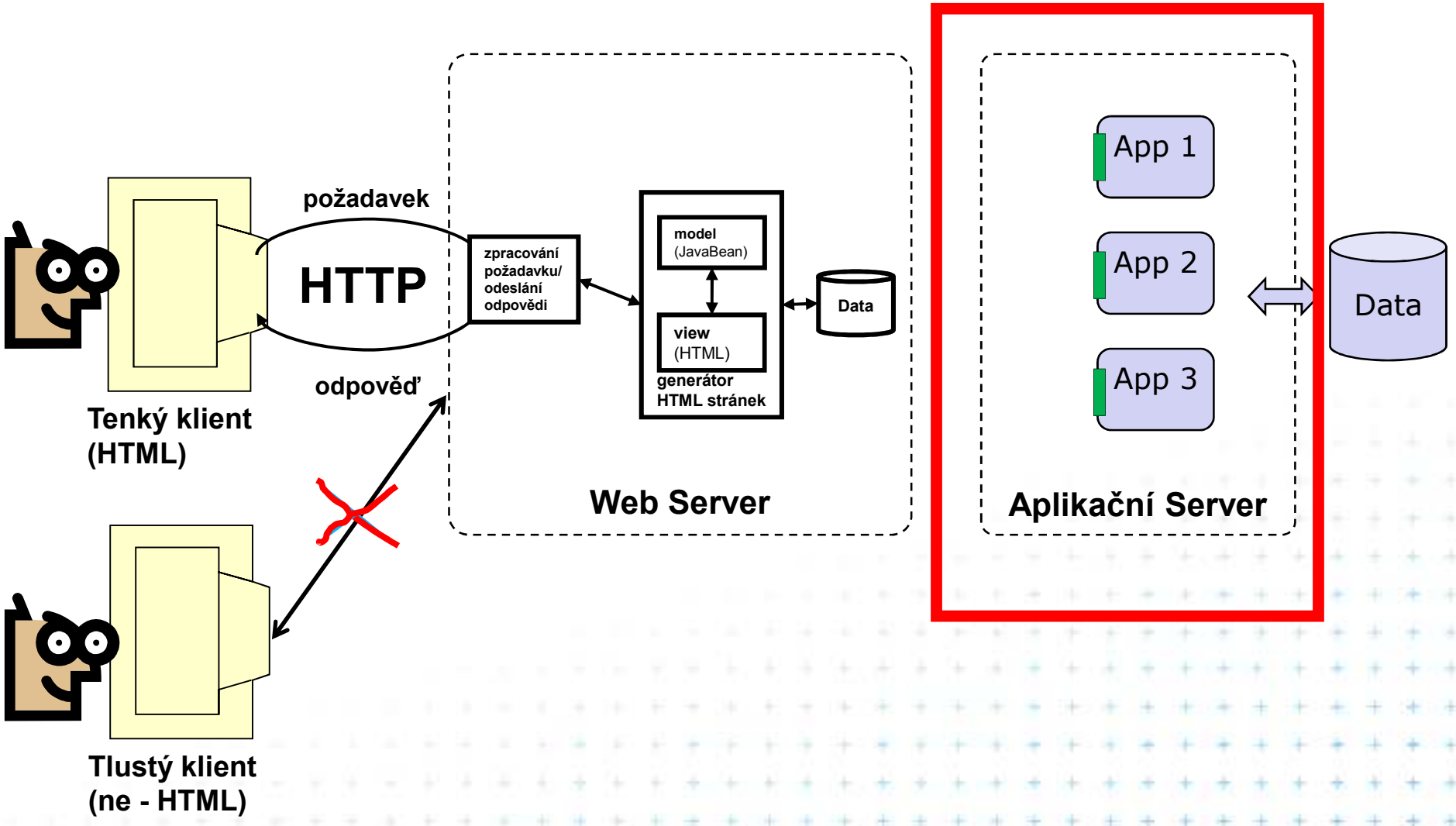
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Vyroky</title>
    </head>
    <body>
        <h2>Seznam</h2>
        <jsp:useBean id="statements" scope="session"
class="cz.cvut.fel.StatementBean" />

        <%
            for (String statement : statements.getAllStatements()) {
                out.println(statement + "<br/>");
            }

        %>

        <form action="StatementController" method="GET">
            <input type="text" name="statement"/>
            <input type="submit" value="Odeslat" name="submit"/>
        </form>
    </body>
</html>
```

Obsluhující servlet - controller

```
 protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            if (request.getParameter("statement") != null) {
                StatementBean sb = (StatementBean)
request.getSession().getAttribute("statements");
                if (sb == null) {
                    sb = new StatementBean();
                }
                sb.addStatement((String)request.getParameter("statement"));

                request.getSession().setAttribute("statements", sb);
            }
        } finally {
            RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/statementForm.jsp");
            dispatcher.forward(request, response);
        }
    }
```

APLIKAČNÍ SERVER

Architektura webové aplikace – aplikační server



Enterprise Beans

- Komponenta běžící v rámci aplikačního serveru
- Spravovaná EJB kontejnerem
- Přístup ze stand-alone aplikace
- Přístup z webové aplikace
- Automatizovaná, připravená funkcionalita (komponentní architektura)
  - session management
  - transaction management
  - db connection management
  - use authentication, authorization
  - asynchronous messaging

Enterprise beans

- Existují tyto typy EJB

- Session bean
  - Stateless
  - Singleton (od EJB 3.1)
  - Statefull
- ~~Entity bean~~ EJB 3.0 ->Entity
- Message driven bean

Anotace

- jsou to vlastně aditivní procesní instrukce
- píše je programátor do zdrojového kódu
- využívá je nějaký externí nástroj

```
@Resource(name = "customerDB")
  public void setDataSource(DataSource myDB) {
     this.ds = myDB;
  }
@EJB
  public ShoppingCart myShoppingCart;

@Local
  public interface RepeaterSessionBeanLocal {
  }

@Copyright("2002 Yoyodyne Propulsion Systems")
  public class OscillationOverthruster {
     ...
  }
```

Anotace jsou definovány pomocí konstruktu

```
public @interface RequestForEnhancement {
    int    id();
    String synopsis();
    String engineer() default "[unassigned]";
    String date()  default "[unimplemented]";
}
```

Použití

```
public class EnhancementTest {

    @RequestForEnhancement(id = 2868724,
    synopsis = "Enable time-travel",
    engineer = "Mr. Peabody",
    date = "4/1/3007")
    public static void travelThroughTime(Date destination) {
        // tady neco udelej
    }
}
```

Komplexní příklad

převzato z http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

Anotace anotace = metadata

Anotovaný program

```
public class Foo {
   @Test public static void m1() { }
   public static void m2() { }
   @Test public static void m3() {
      throw new RuntimeException("Boom");
   }
   public static void m4() { }
   @Test public static void m5() { }
   public static void m6() { }
   @Test public static void m7() {
      throw new RuntimeException("Crash");
   }
   public static void m8() { }
}
```

Využití anotace v kontrolním programu

```
public class RunTests {
  public static void main(String[] args) throws Exception
{
    int passed = 0, failed = 0;
    for (Method m : Class.forName(args[0]).getMethods()) {
      if (m.isAnnotationPresent(Test.class)) {
        try {
          m.invoke(null);   passed++;
        } catch (Throwable ex) {
          System.out.printf("Test %s failed: %s %n", m,
ex.getCause());   failed++;
        }
      }
    }
    System.out.printf("Passed: %d, Failed %d%n",
passed, failed);
  }
}
```

Využití reflexe

Drobnosti kolem anotací

- Lze definovat default hodnoty
- Některé anotace už v jazyce Java existují
  - @Retention
    - SOURCE (jen ve zdrojovém kódu), CLASS (v binární třídě), RUNTIME (za běhu)
  - @Target – výčet z ElementType
    - TYPE
    - FIELD
    - METHOD
    - PARAMETER
    - CONSTRUCTOR
    - LOCAL_VARIABLE
    - ANNOTATION_TYPE
    - PACKAGE
  - @Inherited
    - potomci anotované třídy jsou také anotováni

Anotace finále

- Anotace bez hodnoty

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

- Anotace s jedinou hodnotou

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Inherited
/**
 * Trida bude vracet chybovy stav uvedeny ve {@code value}.
 */
public @interface ErrorPage {
    int value();
}
```

- S více hodnotami a default

```
public @interface RequestForEnhancement {
    int    id();
    String synopsis();
    String engineer() default "[unassigned]";
    String date()  default "[unimplemented]";
}
```

Session bean

- Reprezentují aplikační logiku
- Mohou být svázány s danou session, tj. klientem (statefull)
  - pozor – nejedná se o http session!
- Mohou být obecné, nezávislé na klientovi (stateless)

Entity

- Objekty reprezentující perzistenci
- Jsou navázány na databázi (perzistence)
- Transakce, vyhledávání
- Objektový přístup k datům
- Primární klíč
- Relace na další entity

Message driven bean

- Aplikační logika vázaná na události
- Zpracování událostí generovaných jinými aplikacemi
- B2B
- Asynchronní vyvolání
- Krátká doba života
- Nereprezentují data v databázi
- Jsou bezestavové

EJB 3.x

- Bean je klasická POJO = Plain Old Java Object
- Anotace říkají kontejneru jak s ní zacházet
- Kontejner řídí její životní cyklus
- Lokální a vzdálené rozhraní
- Registrace v JNDI
  - vyhledávání podle jména
  - vzdálený přístup

Drobná ukázka

Stateless EJB
JNDI mapování

```
@Stateless (mappedName = "Repeater")
public class RepeaterSessionBean implements RepeaterSessionBeanRemote,
RepeaterSessionBeanLocal {

    public String repeatNormalLocal(final String text) {
        return text;
    }

    public String repeatNormalRemote(final String text) {
        return text;
    }

    public String repeatReverseRemote(final String text) {
        return getReverseString(text);
    }

    public String repeatReverseLocal(final String text) {
        return getReverseString(text);
    }

    private String getReverseString(String text) {
        StringBuffer b = new StringBuffer(text.length());
        for (int i = text.length(); i > 0; i--) {
            b.append(text.charAt(i - 1));
        }
        return b.toString();
    }
}
```

```
@Remote
public interface RepeaterSessionBeanRemote {
    String repeatNormalRemote(final String text);
    String repeatReverseRemote(final String text);
}
```

```
@Local
public interface RepeaterSessionBeanLocal {
    String repeatNormalLocal(final String text);
    String repeatReverseLocal(final String text);
}
```

Drobná ukázka pokr.

Injection

```
public class FormBean1 {
   @EJB(name="Repeater")
   RepeaterSessionBeanLocal repeaterBean;
   private String text;
   private String reverse;

   public FormBean1() {   }

   public String getText() {
      return text;   }

   public void setText(String text) {
      this.text = text;   }

   public String translate() {
      setReverse(repeaterBean.repeatReverseLocal(text));
      return null;
   }

   public String getReverse() {
     return reverse;   }

   public void setReverse(String reverse) {
      this.reverse = reverse;
   }
}
```

Použití

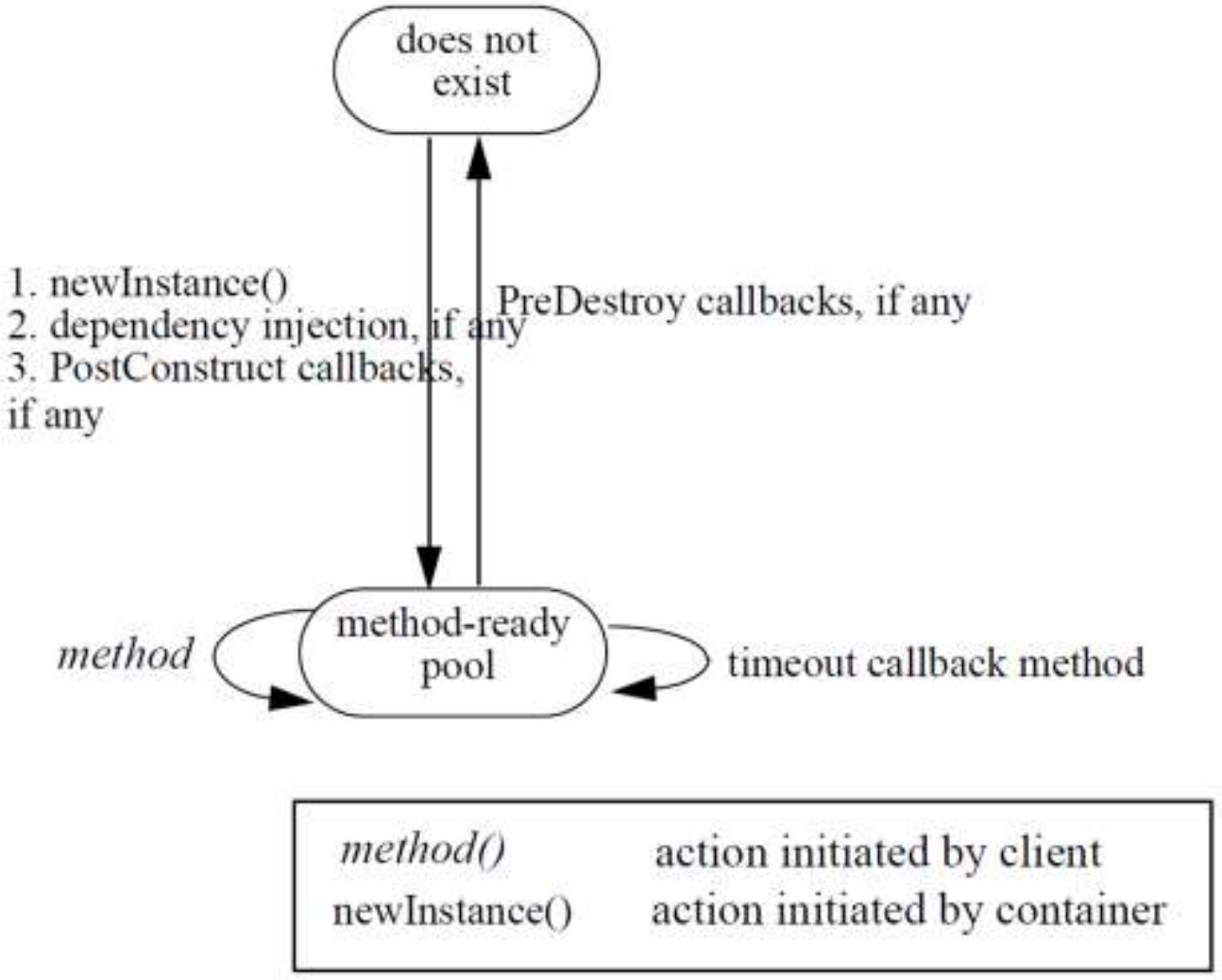
EJB anotace

```
@Target(TYPE) @Retention(RUNTIME)
public @interface Stateless {
        String name() default "";
        String mappedName() default "";
        String description() default "";
}
```

```
@Target(TYPE) @Retention(RUNTIME)
public @interface Stateful {
        String name() default "";
        String mappedName() default "";
        String description() default "";
}
```

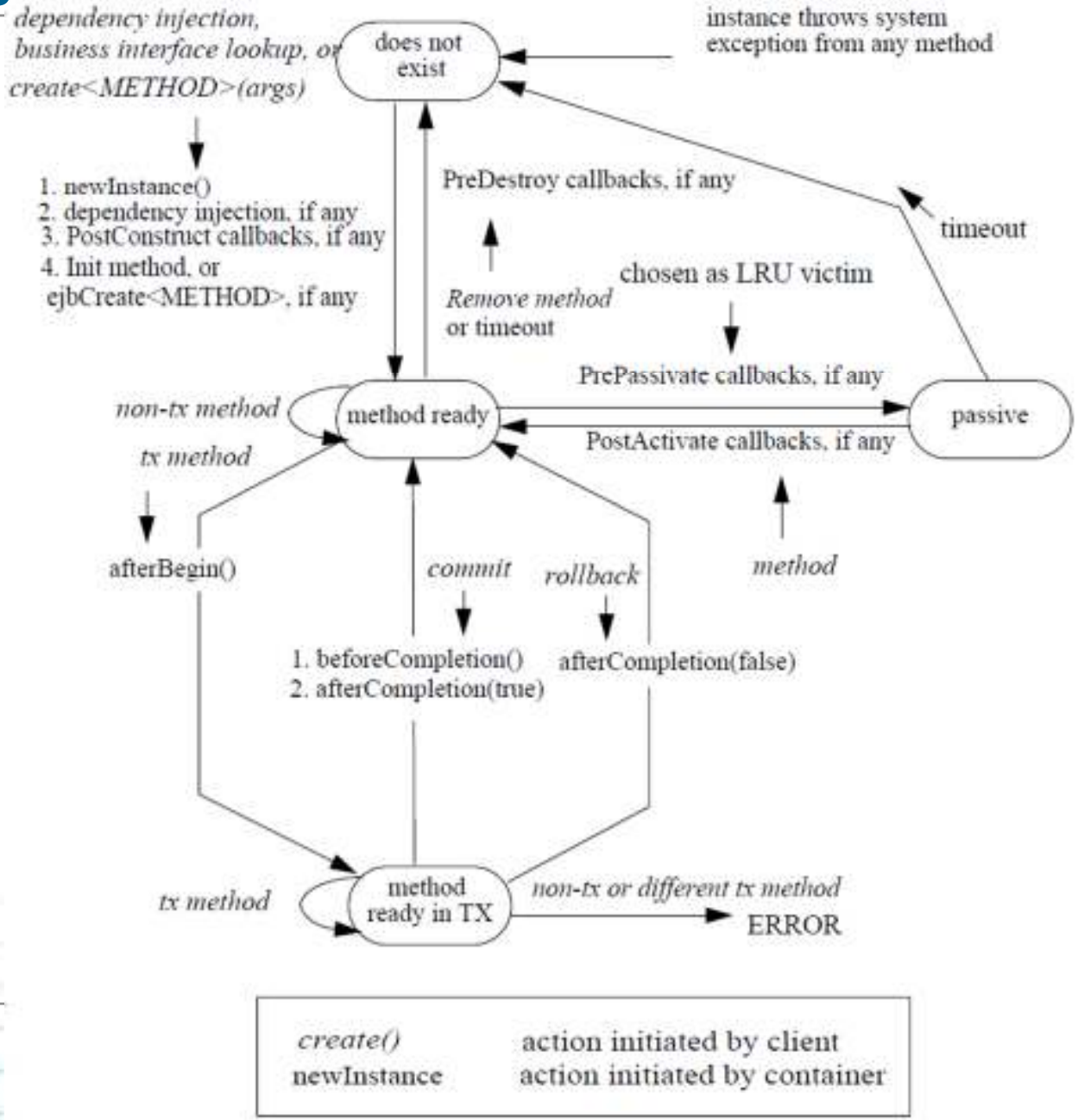
Životní cyklus Stateless EJB

nedrží kontextové informace

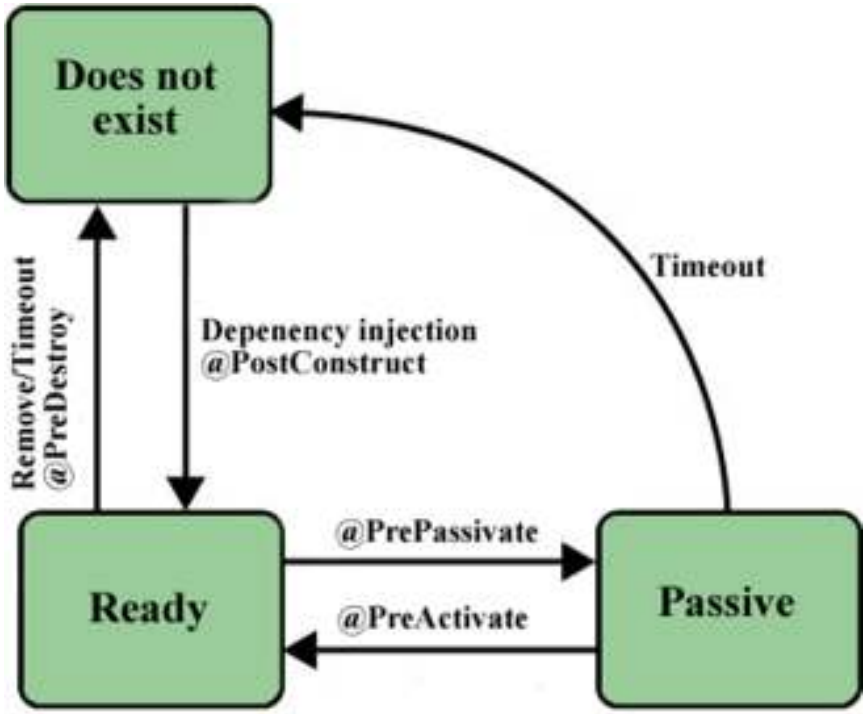


Životní cyklus Stateful EJB

Drží kontextové informace



To samé ale jednoduše (bez transakcí)



Nalezení a použití EJB

1. Injection
   - Kontejner zajistí instanci zdrojů
   - Pomůže mu v tom anotace

```
public class FormBean1 {
   @EJB(name="Repeater")
   RepeaterSessionBeanLocal repeaterBean;
…
}
```

2. JNDI lookup

```
static RepeaterSessionBeanRemote repeaterSessionBean;
Context context;
      try {
         context = new InitialContext();
         repeaterSessionBean = (RepeaterSessionBeanRemote) context.lookup("JNDI_NAME");
      } catch (NamingException e) {
         e.printStackTrace();
         throw new RuntimeException(e);
      }
```

Volání Stateful EJB z webové aplikace

- V zásadě stejné jako ze stand-alone aplikace
- Pokud chceme pracovat se stále stejnou instancí, musíme si jí zapamatovat v session.
- Pozor! Session Bean není HTTP Session

- Nedoporučuje se používat Stateful EJB z bezestavového kontextu
  - a to je právě HTTPServlet
- Používejte Stateless kdekoli to je možné

Local & Remote interface

- RMI je dosti náročná procedura
- V některých případech víme, že EJB se nachází na stejném stroji, lze jí tedy volat přes lokální API
- Proto byla zavedena alternativní varianta volání – Local

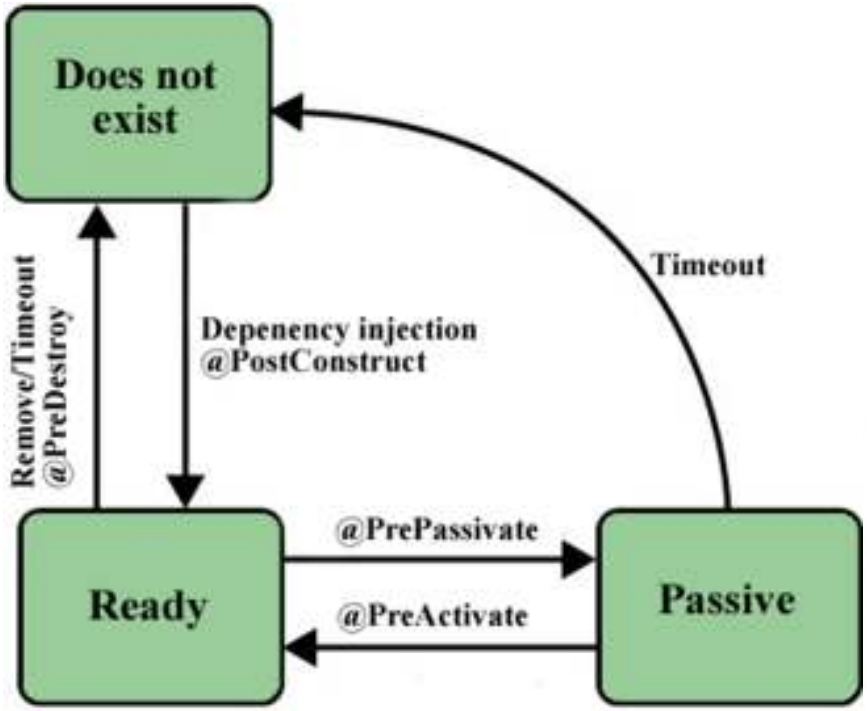
Stateless vs Statefull EJB

- Kde to půjde, používejte stateless
- Stateless se dobře škálují
- Stateless se nemusí ukládat

- Statefull se musí ukládat
- Zatěžují více server
- Problém s distribuovaností
- Vhodné na velké objemy dat, zajistí transakce
- Nemusím se starat o vícevláknovost aplikace

Interceptor

- Interceptor je vlastně handler události
- Podobně jako v Java Swing jsou události, v EJB jsou také
- Podle typu beany (a jejícho životního cyklu), paříklad



Dva způsoby psaní interceptorů

1. Přímo jako metoda v dané beaně

```
@AroundInvoke
   public Object profile(InvocationContext inv) throws Exception {
      long time = System.currentTimeMillis();
      try {
         return inv.proceed();
      } finally {
         long endTime = time - System.currentTimeMillis();
         System.out.println(inv.getMethod() + " took " + endTime + "

         milliseconds.");

      }
   }
```

Dva způsoby psaní interceptorů

EJB

```
@Interceptors(cz.cvut.fel.ejb.interceptors.RepeaterInterceptor.class)
@Stateless(mappedName = "Repeater")
public class RepeaterSessionBean implements RepeaterSessionBeanRemote,
RepeaterSessionBeanLocal {

    public String repeatNormalLocal(final String text) {
        return text;
    }

    public String repeatNormalRemote(final String text) {
        return text;
 }
}
```

Čárkou oddělený seznam interceptorů

Volaná na všechny metody EJB

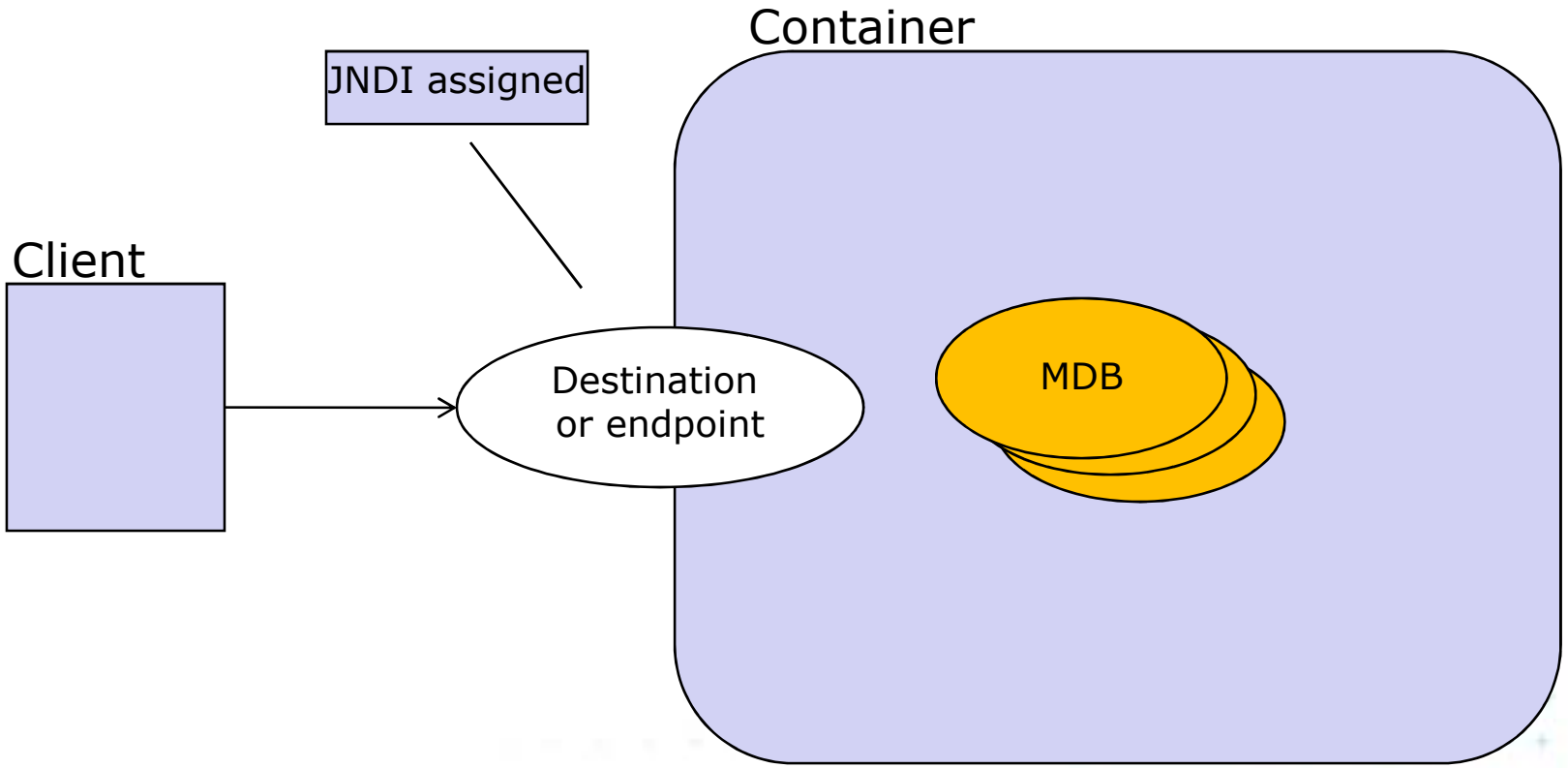
```
public class RepeaterInterceptor {

    @AroundInvoke
    public Object profile(InvocationContext ctx) throws Exception {
        try {
            System.out.print("Zavolana metoda " + ctx.getMethod().getName());
        } finally {
            return ctx.proceed();
        }
    }
}
```


MESSAGE DRIVEN BEANS
MDB

Co to jsou MDB?

- Komponenty, které zpracovávají zprávy
- Jsou podobné Stateless EJB
  - nejsou stavové
  - jsou zaměnitelné
- …ale mají jiný účel
- Zasílání zpráv je běžná součást distribuovaných aplikací
- Pro klienta je MDB prostě konzumentem zpráv

MDB

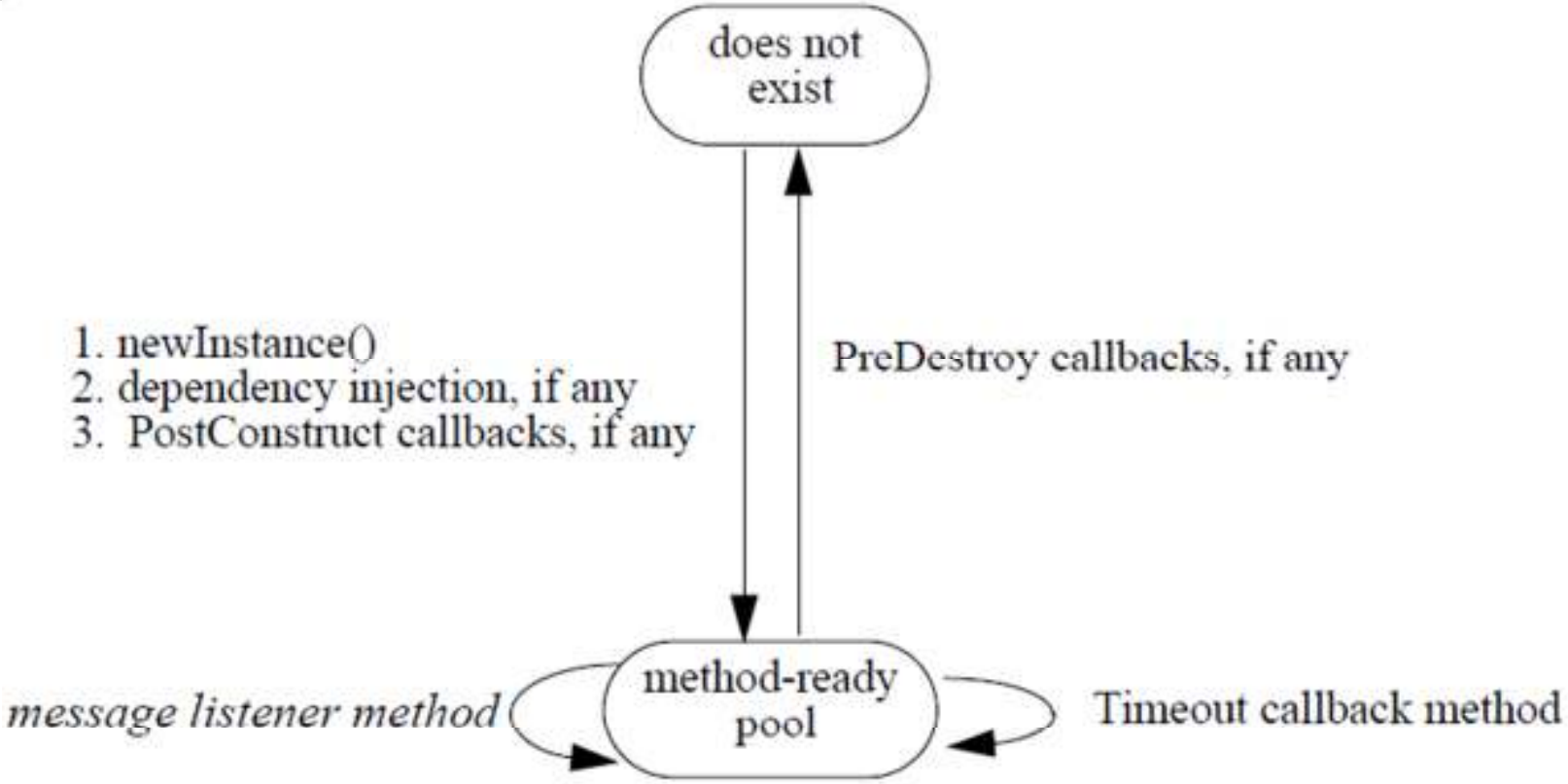

Container
JNDI assigned
Client
Destination
or endpoint
MDB

MDB

- Dva modely komunikace
- – Point to point: fronty (queues, FIFO), jeden příjemce
- – Publish / subscribe: topics, distribuce více příjemcům

- Implementuje rozhraní javax.jms.MessageListener
  - – metoda onMessage

MDB – Životní cyklus



JMS – Java Message Service

- **JMS zpráva**
  - několik typů zpráv
- Zpráva má hlavičku, properties a tělo
  - Hlavička: delivery mode, msg ID, timestamp, priorita, ReplyTo, msg type
  - Properties: jméno, hodnota
  - Tělo: Stream, Map, Text, Object, Bytes

Ukázka MDB

```
@MessageDriven(mappedName = "jms/school", activationConfig = {
   @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-
acknowledge"),
   @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue")
})
public class SchoolMessageBean implements MessageListener {

   public SchoolMessageBean() {
   }

   public void onMessage(Message message) {
      TextMessage tm = (TextMessage) message;
      String teacher;
      try {
         teacher = tm.getText();
         System.out.println("Prijata informace o pridani ucitele: " + teacher);
      } catch (JMSException ex) {
         Logger.getLogger(SchoolMessageBean.class.getName()).log(Level.SEVERE, null, ex);
      }

   }
}
```

Ukázka klienta

```
@Stateless
public class SchoolSessionBean implements SchoolSessionBeanRemote, SchoolSessionBeanLocal {
    @Resource(name = "jms/school")
    private Queue school;
    @Resource(name = "jms/schoolFactory")
    private ConnectionFactory schoolFactory;

    public PartTimeTeacherEntity addPartTimeTeacher(final String firstName, final String lastName, final float partTime) {
        try {
            sendJMSMessageToSchool(firstName + ” teacher added”);
        } catch (JMSException ex) {        Logger.getLogger(SchoolSessionBean.class.getName()).log(Level.SEVERE, null, ex);        }
        return teacher;
    }

   private Message createJMSMessageForjmsSchool(Session session, Object messageData) throws JMSException {
        TextMessage tm = session.createTextMessage();
        tm.setText(messageData.toString());
        return tm;
    }
```

… pokračování na dalším slide

Ukázka klienta cont.

```
private void sendJMSMessageToSchool(Object messageData) throws JMSException {
        Connection connection = null;
        Session session = null;
        try {
            connection = schoolFactory.createConnection();
            session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
            MessageProducer messageProducer = session.createProducer(school);
            messageProducer.send(createJMSMessageForjmsSchool(session, messageData));
        } finally {
            if (session != null) {
                try {
                    session.close();
                } catch (JMSException e) {
                    Logger.getLogger(this.getClass().getName()).log(Level.WARNING, "Cannot close session", e);
                }
            }
            if (connection != null) {
                connection.close();
            }
        }
    }
}
```

Reference

- http://java.sun.com/developer/technicalArticles/ebeans/ejb_30/
- http://java.sun.com/products/ejb/docs.html