Seminář Java

XII

Obsah

- ClassLoaders
- Servlets, Java Server Pages

Zavádění tříd

JVM zavádí třídy dynamicky

Class loader

- objekt schopný zavádět třídy
- abstraktní třída `java.lang.ClassLoader`
- každá třída (`java.lang.Class`) obsahuje referenci na svůj class loader

Implementace class loaderu

- zajištění specifického chování při zavádění tříd
- více verzí stejné třídy

Zavádění tříd

Spuštění aplikace:

```
java Test
```

Kroky při spouštění:

- JVM zjistí, že třída `Test` není zavedena
- JVM použije zavaděč (class loader) pro zavedení třídy `Test`

Proces zavádění třídy:

- načtení byte-code (loading)
- linkování (linking)
- inicializace (initialization)

Zavádění tříd

Loading

- vyhledání a načtení byte-code třídy
- různé zavaděče mají různou politiku (souborový systém, http, ...)

Linking

- operace nutné pro to, aby byla třída použitelná
  - verification
    - verifikace korektnosti binární reprezentace třídy
  - preparation
    - vytváří statické členy třídy, implicitní inicializace
  - resolution (optional)
    - závislosti na jiných třídách

Zavádění tříd

Initialization

- explicitní inicializace statických členů třídy
- před inicializací se musí inicializovat nadřazená třída (pokud ještě není)
- k inicializaci třídy/rozhraní T dochází před akcí:
  - T je třída a instance T je vytvářena
  - T je třída a její statická metoda je volána
  - statická proměnna deklarovaná v T je přiřazena
  - statická proměnna deklarovaná v T je použita

Zavádění tříd

```
class Super {
    static { System.out.print("Super "); }
}
class One {
    static { System.out.print("One "); }
}
class Two extends Super {
    static { System.out.print("Two "); }
}
class Test {
    public static void main(String[] args) {
        One o = null;
        Two t = new Two();
        System.out.println((Object)o == (Object)t);
    }
}
```

Zavádění tříd

⇒ Super Two false

```
class Super {
    static { System.out.print("Super "); }
}
class One {       // nebude nikdy linkována
    static { System.out.print("One "); }
}
class Two extends Super {
    static { System.out.print("Two "); }
}
class Test {
    public static void main(String[] args) {
        One o = null;
        Two t = new Two();
        System.out.println((Object)o == (Object)t);
    }
}
```

Zavádění tříd

```
class Super { static int taxi = 1729; }

class Sub extends Super {
    static { System.out.print("Sub "); }
}

class Test {
    public static void main(String[] args) {
        System.out.println(Sub.taxi);
    }
}
```

Zavádění tříd

```
class Super { static int taxi = 1729; }

class Sub extends Super {
    static { System.out.print("Sub "); }
}

class Test {
    public static void main(String[] args) {
/*1*/   Sub s = new Sub();
        System.out.println(Sub.taxi);
    }
}
```

```
1729

/*1*/ Sub 1729
```

Zavádění tříd

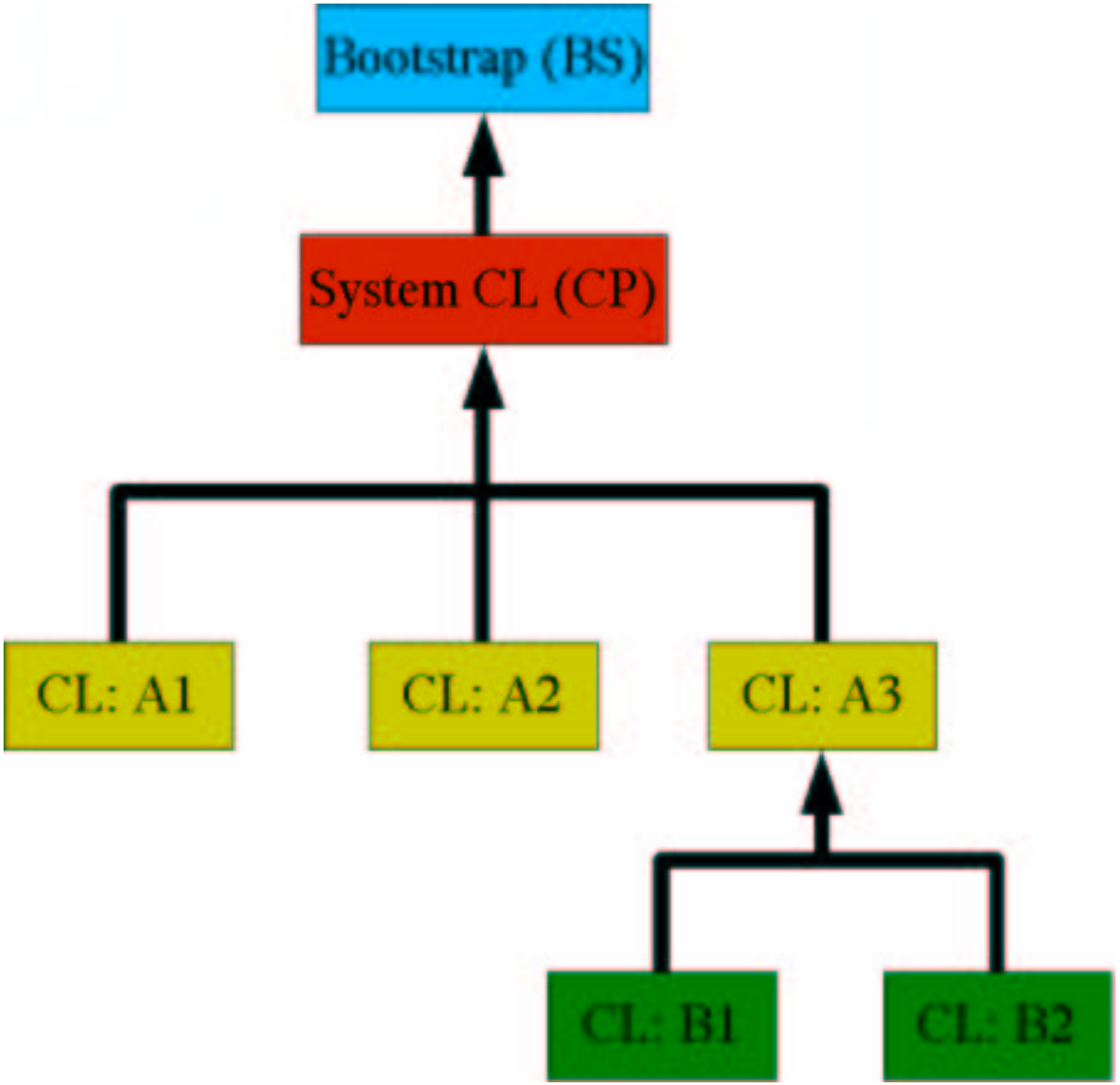
Zavaděče tříd (class loaders)

- vytvářejí hierarchickou strukturu
- každý zavaděč má svého předka (kromě vrcholového)
- zavaděč deleguje požadavky na svého předka
- každá třída je definována svým typem a svým zavaděčem
- každá třída je zavedena pouze jednou
- každá třída je uchována v zavaděči

Hierarchie zavaděčů

- `bootstrap loader`
  zavádí základní třídy JVM (`jr.jar`) + `extensions`
- `system loader`
  zavádí třídy dostupné přes `CLASSPATH` (nebo `-classpath`)

Hierarchie zavaděčů



Zavádění tříd

Metody třídy `java.lang.ClassLoader`

- `loadClass()`
  - test, zda už je třída zavedena (`findLoadedClass`)
  - deleguje zavádění (`loadClass`)
  - $fail \Rightarrow$ načte třídu (`findClass`)
- `findClass()`
  - čte byte-code reprezentující třídu
  - vytváří třídu (`defineClass`)
  - *odvozené zavaděče by měly předefinovat tuto metodu*
- `defineClass()`
  - jako parametr má pole bytů
  - konvertuje pole bytů na třídu
- `resolveClass()`
  - ověření symbolický referencí ze zaváděné třídy
  - zavedení závislých tříd (volitelné)

Zavádění tříd

```
protected synchronized Class<?> loadClass(String name,
        boolean resolve) throws ClassNotFoundException {

    Class c = findLoadedClass(name);
    if (c == null) {
        try {
            if (parent != null) {
                c = parent.loadClass(name, false);
            } else {
                c = findBootstrapClass0(name);
            }
        } catch (ClassNotFoundException e) {
            c = findClass(name);
        }
    }
    if (resolve) {
        resolveClass(c);
    }
    return c;
}
```

Zavádění tříd

```
protected Class findClass(String pClassName)
                    throws ClassNotFoundException {
   try {
      File lClassFile = new File(mDirectory, pClassName +
                                                ".class" );
      InputStream lInput = new BufferedInputStream(
                       new FileInputStream(lClassFile));
      ByteArrayOutputStream lOutput =
                                  new ByteArrayOutputStream();
      int i = 0;
      while ((i = lInput.read()) >= 0) {
         lOutput.write( i );
      }
      byte[] lBs = lOutput.toByteArray();
      return defineClass(pClassName, lBs, 0, lBs.length);
   } catch (Exception e) {
      throw new ClassNotFoundException(...);
   }
}
```

Zavádění tříd

Metody třídy `java.lang.ClassLoader`

- statická metoda `getSystemClassLoader()`
  - vrací systémový zavaděč pro delegaci
- konstruktor `ClassLoader()`
  - inicializace, nastavení rodičovského zavaděče podle metody `getSystemClassLoader()`
- konstruktor `ClassLoader(ClassLoader parent)`
  - inicializace, nastavení rodičovského zavaděče

Zavádění tříd – ukázka

Ukázky nejsou úplné (třídy, výjimky)

```
// version_a
class M { .. }
public class Test {
    public void checkUpcast(Object pTestInstance)
                                throws ClassCastException
    {
        M lTest = (M) pTestInstance;
    }
}

// version_b
class M { ... }
public class Test {
    public void checkUpcast(Object pTestInstance)
                                throws ClassCastException
    {
        M lTest = (M) pTestInstance;
    }
}
```

Zavádění tříd – ukázka

Ukázky nejsou úplné (třídy, výjimky)

```
Object createInstance(ClassLoader pClLoader, String pClName)
{
   Class lClass = pClassLoader.loadClass( pClassName );
   return lClass.newInstance();
}

void checkUpcast(ClassLoader pTestCL, Object pInstance )
{
    Object lTestInstance = createInstance( pTestCL, "Test" );
    Method lCheckUpcastMethod = lTestInstance.getClass().
     getMethod("checkUpcast", new Class[] { Object.class } );

    lCheckUpcastMethod.invoke(lTestInstance,
                              new Object[] { pInstance } );
}
```

Zavádění tříd – ukázka

```
// Create two class loaders one for each version
ClassLoader lClLoader_A = new MyClLoader( "./version_a" );
ClassLoader lClLoader_B = new MyClLoader( "./version_b" );

// Load Class M from first CL and create instance
Object lInstance_M_A = createInstance( lClLoader_A, "M" );
// Load Class M from second CL and create instance
Object lInstance_M_B = createInstance( lClLoader_B, "M" );

checkUpcast( lClLoader_A, lInstance_M_A );
checkUpcast( lClLoader_A, lInstance_M_B );
checkUpcast( lClLoader_B, lInstance_M_A );
checkUpcast( lClLoader_B, lInstance_M_B );
```

Uvolnění objektu

Uvolnění objektu (*garbage collecting*)

- pokud již neexistuje reference na objekt z živého vlákna
- finalizace objektu

Finalizace objektu

- metoda `finalize()` třídy `Object` (bez efektu)
- odvozené třídy mohou předefinovat

```
protected void finalize() throws Throwable {
    super.finalize();
}
```

- před tím, než je uvolněno místo zabírané objektem, volá *garbage collector* metodu `finalize()`
- uvolnění zdrojů, ... před uplným zničením
- není zajištěno pořadí volání této metody
- *pokud není metoda `finalize()` překryta, nevolá se!*

Uvolnění třídy

Třída může být uvolněna:

- pokud její zavaděč (class loader) podlehne `garbage collectoru`
- třídy zavedené *bootstrap* zavaděčem nemohou být nikdy uvolněny
- třída *nemůže* být uvolněna, pokud je její zavaděč potenciálně dostupný

Shrnutí

- Třídy jsou zaváděny dynamicky
- Každá třída má svůj zavaděč
- Existují implicitní zavaděče
- Každá třída "dědí" zavaděč třídy, ze které bylo vyvoláno zavedení
- Je možné definovat vlastní zavaděče

Servlety, Java Server Pages (JSP)

Servlety (Servlets)

- Java přístup k CGI (Common Gateway Interface)
- programy běžící na webovém serveru
- *middle layer* mezi požadavky přicházejícími z WWW klienta a aplikací (databází, ...)

Java Server Pages (JSP)

- spojení statických stránek (HTML) s dynamicky generovaným obsahem ze servletů
- Alternativa ASP, PHP, JavaScript

Systémy

Volně dostupné systémy implementující Servlets a JSP

- Apache Tomcat
  `http://jakarta.apache.org`
- JavaServer Web Development Kit
  `http://java.sun.com/products/servlet/download.html`
- New Atlanta's ServletExec
  `http://newatlanta.com`
  může být vložen do WWW serveru;
  některé vlastnosti dostupné po zakoupení licence
- ...

Apache Tomcat

- tutorial
  `http://www.coreservlets.com/Apache-Tomcat-Tutorial/`

Server

Apache Tomcat

- implicitní port 8080
- princip kontejnerů
- adresářová struktura
  - `install_dir/webapps/ROOT/WEB_INF/classes`
    standardní umístění tříd (servletů)
  - `install_dir/classes`
    alternativní umístění tříd
  - `install_dir/lib`
    standardní umístění archivů jar

Potřebné třídy pro kompilaci servletů

- servlet API (`install_dir/common/lib/servlet-api.jar`)
- JSP API (`install_dir/common/lib/jsp-api.jar`)
- nastavení `CLASSPATH`

Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyApp extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
              throws ServletException, IOException
  {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";

    out.println(docType + "<HTML>\n" +
                "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1>Hi</H1>\n" + "</BODY></HTML>");
  }
}
```

Servlet – základní struktura

Požadavek

- `http://localhost:8080/servlet/MyApp`

Třída `javax.servlet.http.HttpServlet`

- implementuje základní chování (http) servletu
- servlety (http) dědí tuto třídu
- servlety jsou singletony (mají právě jednu instanci)
- při požadavku se vyvolá příslušná metoda objektu ve zvláštním vláknu

Ošetření požadavků

- `doGet(HttpServletRequest, HttpServletResponse)`
- `doPost(HttpServletRequest, HttpServletResponse)`
- ...

Servlet – základní struktura

Životní cyklus servletu

- `init`
  volána právě jednou při vytvoření servletu
- `service(HttpServletRequest, HttpServletResponse)`
  při každém příchodu požadavku vytvoří server vlákno a z něj volá metodu `service`
  podle typu požadavku volá metodu `doXXX`
- `destroy`
  pokud se server rozhodne odstranit servlet, volá nejříve tuto metodu

Souběžné ošetření požadavku

- nad objektem může pracovat více vláken (nutnost synchronizace)
- pokud servlet implementuje rozhraní `SingleThreadModel`, systém garantuje výlučný přístup
- zpomaluje běh

Servlet – inicializace

Možnost parametrizované inicializace servletů

```
private String message;
public void init(ServletConfig config)
                                  throws ServletException
{
   super.init(config);
   message = config.getInitParameter("message");
   ...
}

public void doGet(HttpServletRequest request,
                     HttpServletResponse response) ...
{
   ...
   PrintWriter out = response.getWriter();
   ...
   out.println(message + "<br>");
   ...
}
```

Servlet – inicializace

Soubor .../WEB_INF/web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

    <servlet>
        <servlet-name>App</servlet-name>
        <servlet-class>MyApp</servlet-class>
        <init-param>
           <param-name>message</param-name>
           <param-value>blah blah</param-value>
        </init-param>
    </servlet>
</web-app>
```

Session

Session

- uchovává informace o jednom sezení
- "perzistentní" objekt na serveru (časově omezená životnost)
- `HttpSession s = request.getSession()`
  - vrátí existující / vytvoří novou
  - jednoznačná identifikace (`getId()`)

Typické využití

- internetový obchod
- každý klient má svůj nákupní košík

Session

```
public class MyApp extends HttpServlet {
  private myValue = 10;

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
              throws ServletException, IOException
  {
      HttpSession session = request.getSession();
      Integer storage = (Integer)
                session.getAttribute("storage");
      if (storage == null) {
         storage = myValue;
         session.setAttribute("storage", storage);
      }
      myValue++;
      ...
  }
}
```

Java Server Pages (JSP)

Základní idea

- použít HTML pro většinu stránky
- servletový kód uzavřít mezi speciální značky (*tags*)
- JSP se transformuje na servlet
- balíky v `javax.servlet.jsp`

Ukázka

- JSP
```
<h2>Ahoj <%=request.getParameter("name")%></h2>
```
- požadavek
```
http://localhost:8080/hello.jsp?name=Ferda
```
- výstup
```
Ahoj Ferda
```

JSP: dostupné reference

Dostupné reference (nemusí se deklarovat)

- `request`
- `response`
- `pageContext`
- `session`
- `application`
- `out`
- `config`
- `page`
- `exception`

JSP: typy značek (tags)

Directives

- instrukce zpracované při překladu

```
<%@ page
language="java"
import=". . ."
errorPage=". . ."
contentType=". . ."
%>
```

Hidden Comments

- `<%-- comment --%>`

Declarations

- deklarace proměnných, metod

```
<%! String strMyString = "hello";
private void doSomething() { ... }
%>
```

JSP: typy značek (tags)

Expressions

- výsledek operace je transformován do řetězce a vložen na dané místo ve výstupu

```
<%= strMyStringVariable %>
<%= getData() %>
```

Scriptlets

- fragment kódu

```
<% if(request.getParameter("user").equals("new")) { %>
<B>Please sign up!</B>
<% } else { %>
<B>Welcome back!</B>
<% } %>
```

Actions

- `<jsp:  ...  >`

JavaBeans

Java Beans

- komponenta
- vizuální nástroje (příp. jiné aplikace) mohou s komponentou manipulovat
- možnost zjistit interní informace
- $\Rightarrow$ standardizovaný formát třídy

Základní vlastnosti

- pouze bezparametrický konstruktor
- nemá veřejné položky
- perzistentní položky mají přístupové metody
  - property `name`
  - `getName()`
  - `setName(...)`

JavaBeans a JSP

- použití bean komponenty

```
<jsp:useBean id="beanName" class="bean class"/>
```

Ukázka použití bean komponenty

```
<jsp:useBean id="kosik" class="cz.dum.Kosik">
```

Použití komponenty jako normální třída

```
<% cz.dum.Kosik kosik = new cz.dum.Kosik(); %>
```

Použití bean komponenty

- jednodušší zadávání parametrů
- jednodušší sdílení komponenty mezi různými JSP

JavaBeans a JSP

- nastavení property

```
<jsp:useBean id="kosik" class="cz.dum.Kosik">
    <jsp:setProperty name="kosik"
                     property="zakaznik"
                     value="<%= zakaznik%>" />
</jsp:useBean>
```

- nastavení property z requestu

```
<jsp:setProperty name="kosik" property="zakaznik"
                              param="inputName" />
```

- získání hodnoty property

```
<jsp:getProperty name="kosik" property="zakaznik" />
```

- sdílení komponenty

```
<jsp:useBean id="kosik"
             class="cz.dum.Kosik"
             scope="session" />
```

JSP: uživatelské tagy

Co to je?

- lze definovat vlastní značky (tags)
- každý tag je svázán s obsluřnou třídou

Obslužná třída

- implementuje `javax.servlet.jsp.tagext.Tag`
- rozšiřuje `TagSupport`
- rozšiřuje `BodyTagSupport`

Popisovač

- XML soubor popisující tag

Použití tagu v JSP

- import tagu
- definice prefixu
- použití tagu

JSP: obslužná třída tagu

```
package cz.ija.tags;

import javax.servlet.jsp.*;
import javax.servlet.tagext.jsp.*;
import java.io.*;
import java.util.*;

public class MyDateTag extends TagSupport {
    public int doStartTag() {
        try {
            String date = new Date().toString();
            out.print(date);
        } catch (IOException ex) { ... }
        return SKIP_BODY;
    }
}
```

JSP: popisovač tagu

```
<?xml version="1.0"?>
<!DOCTYPE taglib ...>

<taglib>
    <tlibversion>1.0</tlibversion>
    <jspversion>1.1</jspversion>

    <tag>
        <name>datum</name>
        <tagclass>cz.ija.tags.MyDateTag</tagclass>
    </tag>
</taglib>
```

JSP: použití v JSP

```
<%@ taglib uri="ija-taglib.tld" prefix="ija" %>

...

<p>Dnešní datum je <ija:datum/> </p>
```

Tomcat: zavádění servletů a tříd