Deployment aplikace, ekosystém Javy

RICHARD LIPKA

9.5.2022

Sestavení aplikace

Aplikace:

- Desítky balíků, stovky tříd při rozumné dekompozici
  - Pro GUI 1 okno = 1 třída, 1 třída pro každý nový editor buněk, třídy pro přenos dat mezi okny
- Moduly
  - Od Java 11 JavaFX oddělena jako „modul“ (viz https://www.oracle.com/corporate/features/understanding-java-9-modules.html)
- Zdroje dat
  - Pro GUI konfigurační soubory, texty, FXML, CSS, ikony a obrázky
  - Převody souborů (native2ASCII pro texty)
- Knihovny třetích stran ve správné verzi
  - Práce s databází (embedded databáze, Hibernate), nová primitiva (guava), speciální knihovny
- Automaticky spouštěné testy (JUnit, TestFX, …)

→ jak to dát rychle dohromady?

- S co nejmenším úsilím
- Neudělat chybu – nevynechat, nezkazit cesty

Java aplikace

Režimy spuštění

- *Standalone Java* – běžné spouštění přes JVM
- *Embedded into page* – stažení aplikace z webu a vložení jejího obsahu do stránky (**OpenWebStart –** z Javy odstraněno s verzí 9)
- *Standalone spuštění* – aplikace spuštěna ve vlastní lokální kopii Javy – jako nativní aplikace v daném OS (**jlink**, **Launch4J** nebo jiné nástroje)

  → nevyžaduje instalaci Javy v počítači

(v kontejneru – uvidíte později, celé předpřipravené prostředí pro virtualizaci)

Sandbox

- Omezené prostředí (pro kód stažený z webbu)
- Řízení přes `java.security.Policy`
- Nemohou přistupovat k FS, k dalším serverům, načítat lokální knihovny, měnit bezpečnostní pravidla, vytvářet *ClassLoadery*

Překlad a spuštění s moduly

Parametry pro java i javac stejné

- `--module-path "C:\Program Files\Java\javafx-sdk-11.0.2\lib" --add-modules=javafx.controls,javafx.fxml`

- **`--module-path`** – cesta kde jsou k dispozici stažené moduly
- **`--add-modules`** – seznam modulů které mají být použity

Modul v Javě

- Není totéž jako knihovní .jar soubor
- Obecně skupina balíků a zdrojů které potřebují k běhu, zabalené do .jar souboru
- Navíc **Module descriptor** – základní informace o modulu
  - Jméno, závislosti, zveřejněné části, poskytované služby, požadované služby, povolení pro reflexi

Moduly - příklady

MODULE DESCRIPTOR

```
module treeDemo {
  requires javafx.graphics;
  requires javafx.controls;

  exports uur.exercise08;
}
```

UŽITEČNÉ PŘÍKAZY

`java --list-modules`

- Zobrazí seznam dostupných modulů
- Funguje i s `--module-path` → pro kontrolu správného nastavení

`jdeps` *`jmeno_arefaktu`*

- Zobrazí na čem zadaný artefakt závisí (funguje pro `.jar`, `.class`)
- `jdeps -dotoutput dot -R -verbose:class uur.jar` zobrazí v podobě grafu pro graphviz, s detailním výpisem na úrovni tříd

Sestavení celé aplikace

Pokud je projekt modulární, lze použít `jlink` k vytvoření stand-alone aplikace (https://docs.oracle.com/en/java/javase/11/tools/jlink.html)

- Při překladu musí existovat module descriptor

```
javac --module-path %PATH_TO_FX% -d mods/hellofx classes
```

Cesta k modulům na kterých závisí aplikace

Cesta k výslednému .jar souboru

Jména souborů k překladu

- Moduly je možné slinkovat s celou Javou (respektive s potřebnými částmi – „custom JRE“)

```
jlink --module-path "%PATH_TO_FX_MODS%;mods"
      --add-modules hellofx --output hellofx
```

Cesty k modulům

Seznam používaných modulů

Adresář s cílovou aplikací

Samostatně spustitelná aplikace

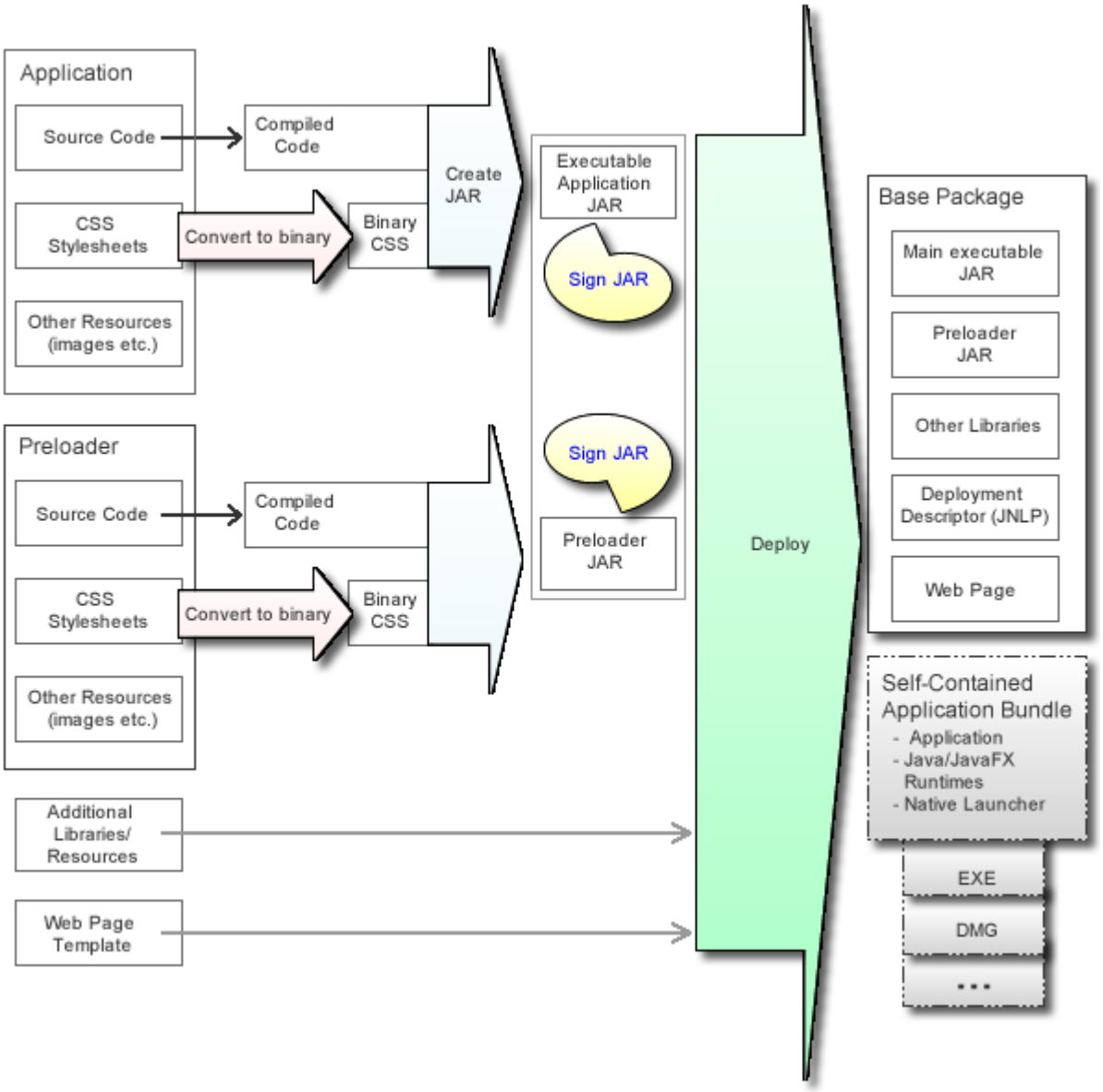
Výhody

- Aplikace se chová jako nativní, není třeba nic dalšího instalovat a nastavovat
- Odpadají problémy s kompatibilitou u různých verzí Javy
- Snazší příprava instalátoru – nepotřebuje žádná zvláštní oprávnění, instalace jen jedné věci a na jedno místo

Nevýhody

- Mnohem větší distribuce (táhne s sebou velkou část Javy a potřebné knihovny)
- Nepřenosná distribuce – pro každou platformu vlastní build (standardní .class soubory jsou bez problémů přenositelné)
- Lze vytvořit jen instalační balík, ne samotnou spustitelnou aplikaci

Tvorba přes Gradle, Maven, Ant nebo pluginy v IDE


Application
Source Code
Compiled Code
CSS Stylesheets
Convert to binary
Binary CSS
Other Resources (images etc.)
Create JAR
Preloader
Source Code
Compiled Code
CSS Stylesheets
Convert to binary
Binary CSS
Other Resources (images etc.)
Additional Libraries/ Resources
Web Page Template
Executable Application JAR
Sign JAR
Sign JAR
Preloader JAR
Deploy
Base Package
Main executable JAR
Preloader JAR
Other Libraries
Deployment Descriptor (JNLP)
Web Page
Self-Contained Application Bundle
- Application
- Java/JavaFX Runtimes
- Native Launcher
EXE
DMG
...

Nevynalézejte kolo!

Apache commons – rozsáhlá kolekce všeho možného (https://commons.apache.org/)
- Math, CLI, CSV, IO

Guava – Java knihovny od Google (https://github.com/google/guava)

google-gson – serializace Java tříd (https://github.com/google/gson), alternativne **FasterXML/Jackson** (https://github.com/FasterXML/jackson)

Hibernate-ORM – práce s DB (http://hibernate.org/orm/)

HyperSQL – embedded databáze pro Javu (http://hsqldb.org/)

Log4J – logování událostí za běhu aplikace (https://logging.apache.org/log4j/2.x/, http://www.slf4j.org/) ; nebo **SLF4J** (http://www.slf4j.org/)

Colt – knihovna pro statistiku (https://dst.lbl.gov/ACSSoftware/colt/)

JGraphT – knihovna pro práci s grafy, včetně algoritmů (https://jgrapht.org/)

Nevynalézejte kolo!

Apache POI – API a knihovny pro práci s MS formáty (https://poi.apache.org/), nebo **docx4j** (https://www.docx4java.org/trac/docx4j)

- Hlavně Excel – stejně na něm stojí všechen průmysl

iText – pro manipulaci s PDF soubory (https://itextpdf.com/en) – novější verze už není zdarma

- Apache FOP – pro generování většiny PDF postačí (https://xmlgraphics.apache.org/fop/)

jsoup – manipulace s HTML obsahem (https://jsoup.org/)

- parsování, prohledávání, změny

openNLP – nástroje pro práci s přirozeným jazykem (https://opennlp.apache.org/)

- tokenizace, detekce jmen, rozklad vět; může potřebovat natrénovat model

JavaCV – wrapper pro openCV – manipulaci s obrazem (https://github.com/bytedeco/javacv)

- Základní zpracování, detekce hran a objektů, filtry, stabilizace videa, …

webmagic – web crawler (https://github.com/code4craft/webmagic)

Nástroje pro automatický deployment

Ant

- Imperativní, procedurální „jazyk“ (XML)
- Skripty popisující co všechno má OS a překladač udělat
  → víc psaní, větší volnost práce

Maven

- Deklarativní „jazyk“ (XML)
- Stará se o celý projekt
  → pevně daná struktura souborů, obtížné udělat „něco jiného“
- Spolupráce s úložišti dat v síti (veřejný centrální repositář, lokální úložiště)
  → vyžaduje připojení k internetu

Gradle

- Skriptovací jazyk založený na Groovy
- „kombinace Ant a Maven“ – umí popisy úloh i řízení závislostí
- Umí zpracovávat Ant skripty

Maven

Původně vytvořen pro buildy *Jakarta Turbine* frameworku, Apache Licence 2.0

Cílem usnadnit buildování oproti práci s Antem

- První verze 2001, současná z března 2022 (3.8.5)

Funkcionalita v Maven pluginech, uživatel by ideálně měl jen dodržet konvence

- Lze vytvářet vlastní pluginy

Existují pluginy pro základní IDE, podpora v úložištích

V základu konzolová aplikace, ovládaná z příkazové řádky

Maven - získání

Na `maven.apache.org` ke stažení jako `.zip` soubor

- Po rozbalení spustitelný program
- Je třeba vytvořit a nastavit OS proměnnou `MAVEN_HOME`
- Do systémové `PATH` je třeba přidat cestu k binárkám Mavenu

Aby fungoval obvykle potřebuje přístup k internetu

- Hledá knihovny v hlavním Maven repositáři

Dokumentace na `maven.apache.org/guides/`

Základní použití

Není třeba říkat Mavenu co má dělat, to ví

Je třeba mít

- Správnou strukturu projektu
- pom.xml soubory s metadaty a popisy závislostí projektu

Maven umí předpřipravit strukturu adresářů

Maven respektuje životní cyklus projektu a automaticky umí projít několika jeho fázemi

- Trochu složitější pokud ho chcete nějak po svém

Struktura projektu

Nutné dodržet (lze upravit v konfiguračním souboru, ale nedělejte to!!!)

- Maven ji umí vytvořit

- `src/main/java` – zdrojové texty aplikace
- `src/main/resources` – knihovny, další zdroje
- `src/main/config` – konfigurační soubory
- `src/main/webapp` – webové zdrojové texty
- `src/test/java` – zdrojové soubory testů (junit)
- `src/test/resources` – další zdroje pro testy

Životní cyklus projektu - *Phases*

1. Validace – `validate` – ověří jestli je projekt korektní
2. Překlad – `compile` – překlad zdrojových textů
3. Testování – `test` – spuštění testů (např. junit)
4. Distribuce – `package` – výroba `.jar`, `.war` …
5. Integrace – `integration-test` – vložení .jar do produkčního prostředí
6. Verifikace – `verify` – testy v produkčním prostředí
7. Instalace – `install` – instalace .jar do lokálního repositáře Mavenu
8. Nasazení – `deploy` – instalace .jar do sdíleného repositáře

Spuštění Mavenu

Lze ovládat z konzole

Příkaz `mvn`

- `-help` – výpis nápovědy
- `-version` – výpis čísla aktuální verze

- `generate –` vytvoření struktury projektu
- `compile` – překlad projektu
- `test` – překlad a spuštění testů (pokud jsou)
- `deploy` – vše od překladu až pro vložení do repositáře
- `clean` – odstraní soubory z předchozích buildů

- `[název pluginu]:[cíl]` – spuštění pluginů

Popis projektu – POM soubor

POM (Project Object Model) obsahuje popis projektu (artefaktu)

- Metadata – jméno projektu, verze, způsob distribuce
- Závislosti – potřebné knihovny
- Repositáře – kde hledat knihovny, kam nahrát projekty
- Pluginy – nastavení pluginů
- Resources – nastavení cest ke zdrojům (jiným než zdrojovým textům)

Funguje dědičnost – lze nastavit rodičovský artefakt (POM)

- Snazší organizace složitých projektů, lze mít „lokální“ a „globální“ pom soubory
- IDE většinou umí poskytnout náhled celého POM souborů který se uplatní

Minimální POM soubor

Nastavení základních metadat pro identifikaci artefaktu (projektu)

- Jméno: `<groupId>:<artefactId>:<version>`

```
<project>
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.mycompany.app</groupId>
	<artifactId>my-app</artifactId>
	<version>1</version>
</project>
```

Závislosti v POM

Nastavení potřebných knihoven

Maven je automaticky hledá v repositářích

- Centrální respositář + lokální repositáře
- Automaticky stahuje potřebné – požadované knihovny + vlastní pluginy (vzniká lokální repozitář - pozor na velikost)

```
<dependencies>
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>3.8.1</version>
                <scope>test</scope>
        </dependency>
</dependencies>
```

Centrální Maven repositář

Na adrese `search.maven.org`

Lze ručně procházet, obsahuje hotové .jar soubory (ne zdrojové texty)

Obsahuje např.

- `junit`
- `jmock`
- `log4j`
- `jgraph`
- `jfreechart`
- `joda-time`
- `jaxb`
- `textfx`

Pluginy v POM

Nastavení konfigurace pro jejich spuštění

Lze specifikovat konfiguraci pro jednotlivé cíle

```
<plugin>
    <groupId>my.group</groupId>
    <artifactId>my-plugin</artifactId>
    <configuration>
        <items>
            <item>parent-1</item>
        </items>
        <properties>
            <parentKey>parent</parentKey>
        </properties>
    </configuration>
</plugin>
```

Maven a JavaFX

Doplnit plugin pro JavaFX

Explicitně ho využít při spouštění:

- `mvn clean javafx:run`

(lze použít i pro ostatní cíle)

Umí i sestavení celé stand-alone aplikace

- `mvn clean javafx:jlink`

```
<plugins>
  <plugin>
    <groupId>
      org.openjfx
    </groupId>
    <artifactId>
      javafx-maven-plugin
    </artifactId>
    <version>
      0.0.4
    </version>
    <configuration>
      <mainClass>
        HelloFX
      </mainClass>
    </configuration>
  </plugin>
</plugins>
```

Gradle

Jazyk (DSL) pro skriptování automatických akcí
- Primárně „méně ukecaná“ náhrada za Maven
- Umí s Mavenem spolupracovat (lze konvertovat Maven projekt na Gradle)
- Umí využívat Maven repozitáře

První verze 2007, současná 7.4.2 z března 2022
- pozor na párování verzí s JDK
  - JDK 11, 12 – Gradle 5.0
  - JDK 13 – Gradle 6.0
  - JDK 14 – Gradle 6.3
  - JDK 15 – Gradle 6.7
  - JDK 16 – Gradle 7.0
  - JDK 17 – Gradle 7.3

Odděluje
- Projekty – sady dat k překladu / zpracování
- Tasky – co se má s projektem udělat

Syntaxe – Groovy → přehlednější než XML („víc imperativní“)

Pluginy pro všechna hlavní IDE, podpora mnoha jazyků (Java, C, C++, …)

Ukázka skriptu

```
plugins {
    id 'java,
    id 'maven-publish,
    id 'eclipse
}
repositories {
    jcenter()
    mavenLocal()
}
dependencies {
    implementation 'org.apache.logging.log4j:log4j-core:2.8,
}

group = 'cz.zcu.laps,
version = '1.0-SNAPSHOT,
description = 'LoadFlow method implementation,
sourceCompatibility = '1.8,

task sourcesJar(type: Jar) {
    from sourceSets.main.allJava
    archiveClassifier = 'sources,
}
publishing {
    publications {
        maven(MavenPublication) {
            artifactId = project.name
            from(components.java)
            artifact sourcesJar
        }
    }
}
```

Podpora pro Maven

Kde hledat závislosti

Vlastnosti pro MVN artefakt

Postup publikace

Gradle a JavaFX

```
plugins {
  id 'application,
  id 'org.openjfx.javafxplugin,
      version '0.0.8,
}
```

Plugin pro podporu JavaFX

```
javafx {
    version = "11.0.2„
    modules = ['javafx.controls']
}
```

Nasatvení pluginu – jaké moduly používat

(najde si je sám jako závislosti)

Gradle a jlink

```
plugins {
  id 'application,
  id 'org.openjfx.javafxplugin'
      version '0.0.7,
  id 'org.beryx.jlink'
     version '2.10.2,
}

javafx {
    version = "12.0.1„
    modules = ['javafx.controls']
}

jlink {
    launcher {
        name = 'hellofx,
    }
}
```

Plugin pro jlink

Nastavení pluginu – jméno modulu který vzniká (musí být v module descriptoru)

Ant – Another Neat Tool

Vytvořen pro Java aplikace (původně pro sestavování Tomcatu)

- Jako obdoba (a náhrada) **`make`**, ale přenositelný
- Existují i verze pro .NET a PHP

První verze z léta 2000, poslední červenec 2021 (1.10.11)

Široce rozšířen, často neviditelný (pluginy pro Eclipse, JBuilder, NetBeans, IntelliJ IDEA)

Vysoce konfigurovatelný, zvládne „cokoliv“

V základu konzolová aplikace + XML skripty

- Implementace ve standardní Javě
  → snadné rozšiřování, pluginy jsou .jar soubory
- „ukecané“ XML skripty – vyžadují poměrně hodně psaní

Ant - zdroje

Součást Apache projektů (licence Apache 2.0)

Na `ant.apache.org` ke stažení jako `.zip` soubor

- Obsahuje dokumentaci (`ant\docs\apendix_e.pdf` – popis všech funkcí – *task reference*)
- Po rozbalení rovnou spustitelný (pokud máte Javu)
  - Do PATH doplnit cestu k binárkám (`ant\bin`)
  - Nastavit systémovou proměnnou `ANT_HOME`

Dokumentace na `ant.apache.org/manual`

- Důležitá sekce *Ant Tasks*
- Přečíst `ant.apache.org/ant_in_anger`
  obsahuje řadu dobrých rad a doporučení jak Ant používat

Dokumentace na `http://ant.apache.org/external.html`

- Jak využít externí programy v Ant skriptech

Základní použití

XML soubor se skriptem – „sestavovací schéma“

- Implicitně `build.xml`, který zpracuje příkaz `ant`
- Pro každý projekt vlastní soubor

V jednom projektu lze určit několik cílů

- Cíle (`target`) se skládají z několika úkolů
  - Logické celky projektu (překlad, tvorba `.jar` souboru …)

→ Soubor může být poměrně rozsáhlý

- Úkoly jsou služby (`task`) antu

I tohle je zdrojový text

- Udržovat rozumnou strukturu, komentovat jednotlivé cíle
- Používat symbolické konstanty

Základní použití - ukázka

```
<?xml version="1.0„ encoding="UTF-8"?>
<project name="Adr„ default="vytvorAdr">
      <target  name="vytvorAdr">
            <mkdir  dir="muj-adresar"/>
            <echo  message="Hotovo"/>
      </target>
</project>
```

Základní použití – ukázka výstupu

```
D:\xml\ant>ant  -buildfile  adresar1.xml
Buildfile:  adresar1.xml

vytvorAdresar:
 mkdir]  Created  dir:  D:\xml\ant\muj-adresar
      echo]  Hotovo

BUILD  SUCCESSFUL
Total  time:  0  seconds
D:\xml\ant>
```

Výchozí cíl

Lze mít několik cílů

- Obvykle alespoň 4 (*překlad*, *vyčištění projektu*, *spuštění* a *generování* spustitelného .jar souboru)

Jeden lze označit jako defaultní – spouští se příkazem `ant`

Ostatní lze spustit přes jejich jméno v parametru, např.

```
D:\xml\ant>ant –buildfile adr.xml smaz
```

Vlastosti projektu `<project>`

Jméno projektu (`name="jmeno"`)

- Nepovinné, ale dobré pro přehlednost

Defaultní cíl (`default= "cil"`)

- Povinné, vybrat ten co se bude používat nejčastěji

Základní adresář (`basedir= "cesta"`)

- Nepovinný, základní adresář pro relativní cesty (typicky cesta ke kořeni projektu)
- Pokud není uveden použije se aktuální adresář
- Užitečné používat pro udržení pořádku v souborech

Použití konstant - <property>

Po nastavení nelze měnit

Užitečné pro udržení organizace zejména delších skriptů

- Zejména cesty a jména adresářů (čtených i vytvářených)
- V XML nejsou typy – všechno je řetězec

Určeny jménem a (textovou) hodnotou

```
<property name="jmeno" value="hodnota">
```

Po definování se na ně lze kdekoliv odkázat přes **$** - konstanta se nahradí hodnotou

```
${jmeno}
```

Použití konstant - `location`

Lze využít pro nastavování cest – místo `value` použít `location`

- Relativní cesty využívají `basedir`!

`Location` se nehodí pro skládání jmen adresářů

- Jména podadresářů by byla chápána jako relativní cesta
- Příklad možných problémů:

```
<property name="adresar" location="muj-adresar"/>
<property name="podadresarLoc" location="vnoreny"/>

<echo  message="${adresar}/${podadresarLoc}"/>

[echo]  D:\xml\ant\muj-adresar/D:\xml\ant\vnoreny
```

Použití konstant - `file`

Odkaz na externí soubor, ze kterého se konstanty mají načíst → oddělím cesty od popisu buildu

- Stejný typ souboru jako při internacionalizaci

```
<property file="adresare.properties" prefix="adr"/>
<echo message="adresar:${adr.adr}/${adr.podadr}"/>
```

V souboru `adresare.properties`:

```
adr=muj-adresar
podadr=vnoreny
```

Pozor na uvozovky – byly by součástí konstanty

Použití konstant - `environment`

Pro získání hodnot systémových proměnných

Jména systémových proměnných jsou case sensitive

Pozor – pokud uděláte chybu a spletete jméno, použije se uvedené jméno, ne hodnota proměnné!

Příklad - výpis systémové proměnné Path:

```
<property  environment="e"/>
<echo>${e.Path}</echo>
```

Nastavení cest - <path>

Pro vytvoření seznamu cest (oddělených jmen adresářů, jako PATH v OS)

- Každá cesta je jeden <pathelement>

Cesta identifikována zadaným id, lze se na ni odkazovat přes refid při definici konstanty

```
<path id="mujClasspath">
        <pathelement path="C:\Program Files\java\"/>
        <pathelement location="prelozene/knihovna.jar"/>
 </path>

<property name="cesta" refid="mujClasspath"  />
```

Výpis:

```
 [echo]
    C:\Program Files\java;D:\xml\ant\prelozene\knihovna.jar
```

Nastavení jmen souborů - <fileset>

Vytvoření seznamu více souborů (<dirset> pro adresáře)

Lze používat masky - *.java, **/*.java

Přidání souborů – atribut include

Zakázání souborů – atribut exclude

Podrobné informace o výběru souborů v dokumentaci

- O vzorech:
  https://ant.apache.org/manual/dirtasks.html#patterns
- O selektorech:
  https://ant.apache.org/manual/Types/selectors.html

```
<fileset id="mojeSoubory" dir="." excludes="a*.xml,
                                    p*.xml  z*.xml">
                <include  name="**/*.java" />
                <exclude  name="build.xml"  />
 </fileset>

<property  name="soubory" refid="mojeSoubory"  />
```

Nastavení cílů - `<target>`

V projektu jich může být několik, identifikovány jménem (atribut `name`)

Každý cíl je „program“ – posloupnost funkcí které má ant provést

Parametr `-projecthelp` vypíše seznam cílů

Cíle mohou být zřetězené – závislé na sobě (atribut `depends`)

- Nesmí dojít k zacyklení (→ jen strom závislostí)
- Snažit se o přímočaré závislosti
- Ant zajistí že se nejprve provedou závislosti a až pak cíl

```
<target name="prvni" depends="druhy">
```

Podmíněné cíle

Existencí `property` lze ovlinit který cíl se provede (jednoduché větvení)

- Pozor – testuje se existence, ne hodnota property
- Pozor při kombinaci podmíněného cíle a závislosti!
- Konstrukce `if` – `unless`

```
<property name="povoleni" value="nezalezi"/>

<target name="prvni" if="povoleni">
  <echo message="prvni"/>
</target>

<target name="druhy" unless="povoleni">
  <echo message="druhy"/>
</target>
```

Úkoly - `<task>`

Příkazy prováděné v rámci cíle

1. Vestavěné
   - V dokumentace „*core tasks*“, fungují vždy
   - Např. `<javac>`
2. Volitelné (optional)
   - Obvykle potřebují nějakou knihovnu
   - Popsány v *Library Dependencies*
   - Např. `<XmlValidate>`
3. Vlastní
   - Vlastní třídy v Javě které lze z Antu volat
   - (`ant\docs\manual\tutorial-writing-tasks.html`)

Přehled úkolů - <copy>

Kopírování souborů, sad souborů (<fileset>) a adresářů

- Do souboru
- Do adresáře

```
<copy file="stary.txt" tofile="novy.txt"/>
<copy file="stary.txt" todir="D:\zzz"/>

<copy todir="D:\zzz">
  <fileset dir="."/>
</copy>
```

Přehled úkolů - <delete>

Mazání, funguje podobně jako <copy>

- Při mazání

```
<delete file="/lib/ant.jar"/>
<delete dir="lib"/>

<delete includeemptydirs="true„>
  <fileset dir="src" includes="**/.svn"
  defaultexcludes="false"/>
</delete>
```

Přehled úkolů - <mkdir>

Vytvoří adresář, případně celou zadanou cestu

Lze použít oddělovače / i \

Často se používají `property`, pozor ale na `location`!

```
<mkdir  dir="${pocatecni}/vnor1\vnor2/${koncovy}"/>
```

Přehled úkolů - <move>

Funguje stejně jako <copy>, jen soubory přesouvá

```
<move file="file.orig" tofile="file.moved"/>
<move file="file.orig" todir="dir/to/move/to"/>
```

```
<move todir="some/new/dir">
      <fileset dir="my/src/dir">
            <include name="**/*.jar"/>
            <exclude name="**/ant.jar"/>
      </fileset>
</move>
```

Přehled úkolů - `<javac>`

Spustí překlad `.java` souborů

Zdrojový adresář prohledává automaticky rekurzivně
→ není problém s dělením aplikace do balíků a podbalíků

Překládá jen

- Chybějící `.class` soubory
- `.class` soubory starší než zdrojové `.java` soubory
  → překlad netrvá zbytečně dlouho, při deploymentu vyčistit projekt

Lze nastavit

- Parametry překladače `-<compilararg>`
- Přidání debug informace – `debug=on`
- Classpath (nejlépe přes `<path>`) `-<classpath>`

Přehled úkolů - <javac>

```
<javac srcdir="."/>

<javac srcdir="." destdir="./class"
      includes="Zakladni.java  src/balik/*.java">
      <compilerarg  value="-Xlint"  />
</javac>

<javac debug="on">
      <src path="."  />
      <classpath refid="classpathProJAXB" />
</javac>
```

Přehled úkolů - <java>

Spuštění libovolného Java programu

Lze spustit v nové instanci JVM – `fork=true`

- Nutné pro .jar soubory nebo pokud chceme další parametry JVM

```
<java classname="Zakladni"  />

<java classname="Zakladni"
 classpath="./class"  />

<java classname="balik.Hlavni"
      classpath="./class" fork="true">
      <jvmarg  value="-Xmx300M"/>
 </java>
```

Přehled úkolů - <javadoc>

Generování standardní dokumentace

Řada parametrů ovlivňující co se generuje (viz dokumentace)

```
<javadoc sourcepath="./src"
         destdir="./doc"
          windowtitle="MujBalik">

 <fileset  dir="./src" includes="**/*.java"  />

</javadoc>
```

Přehled úkolů - <jar>

Vytvoří .jar soubor ze zadané sady souborů
- Potřebuje přeložené soubory, nezajišťuje překlad!

Vytvoří příslušný manifest v adresáři META-INF
- Je třeba specifikovat hlavní třídu

```
<jar destfile="aplikace1.jar">
    <fileset  dir="./class"  />
    <fileset  dir="./src"  />
</jar>

<jar destfile="hlavni.jar" basedir="./class">
    <manifest>
        <attribute name="Main-Class„
                   value="balik.Hlavni"/>
    </manifest>
</jar>
```

Časová známka - `<tstamp>`

Pokud má být v názvu souborů aktuální datum / čas

Automaticky nastaví 3 `property`

- `DSTAMP` – rok, měsíc, den
- `TSTAMP` – hodina, minuta
- `TODAY` – měsíc (slovem), den, rok

Lze si nastavit vlastní formát

- Formátovací řetězec jako v Java formatterech (pro **`SimpleDateFormat`**)

```
<tstamp>
   <format property="MujDatum"
           pattern="yyyy-MM-dd"/>
   <format property="MujCas"
           pattern="HH:mm:ss"/>
</tstamp>
```

Přehled úkolů - `<native2ascii>`

Hlavně pro internacionalizační soubory, při problémech s akcenty

- Atribut `ext` – přípona nově vzniklých souborů

```
<native2ascii encoding="CP1250"
              src="./texts"
               dest="./convertedTexts"
              includes="**/*.java"
              ext=".java"/>
```

Ladicí výpisy

Lze získat detailní informace o tom co Ant dělá

- Parametr `-verbose`
- Trasování chodu Antu

Lze přepnout do debug módu

- Parametr `-debug`
- V podstatě jen podrobnější výstup

Lze vypnout všechny výpisy

- Paramter `-quiet`

Ukázka – Ant ze cvičení

Nastavení projektu:

```
<property name="app.name" value="Cviceni 9" />

<property name="src.dir" value="./src" />
<property name="res.dir" value="./resources" />
<property name="build.dir" value="./bin" />
<property name="doc.dir" value="./doc" />
<property name="jar.dir" value="./jar" />

<property name="bin.jar.file"
          value="InterFXMLDemo.jar" />

<property name="main-class"
          value="exercise10.InterFXMLDemo" />

<path id="classpath">
  <pathelement location="${build.dir}" />
</path>
```

Ukázka – Ant ze cvičení

Překlad projektu:

```
<target name="compile" >
  <mkdir dir="${build.dir}" />
  <javac srcdir="${src.dir}"
         destdir="${build.dir}"
         compiler="modern">
    <classpath refid="classpath" />
  </javac>
  <copy todir="${build.dir}">
    <fileset dir="${res.dir}"/>
  </copy>
</target>
```

Ukázka – Ant ze cvičení

Spuštění:

```
<target name="run" depends="compile">
  <java classname="${main-class}"
        fork="true" dir="${build.dir}">
    <classpath refid="classpath" />
  </java>
</target>
```

Vyčištění:

```
<target name="clean">
  <delete dir="${build.dir}" />
  <delete dir="${bin.dir}" />
  <delete dir="${doc.dir}" />
  <delete dir="${jar.dir}" />
</target>
```

Ukázka – Ant ze cvičení

Sestavení .jar souboru:

```
<target name="distjar"
        depends="compile, javadoc">
  <mkdir dir="${jar.dir}" />
  <copy todir="${build.dir}/src">
    <fileset dir="${src.dir}"/>
  </copy>
  <jar destfile="${jar.dir}/${bin.jar.file}"
       basedir="${build.dir}">
    <manifest>
      <attribute name="Main-Class"
                 value="${main-class}" />
    </manifest>
  </jar>
</target>
```

Ant – tvorba spustitelné aplikace

Vyžaduje další nástroje pro tvorbu instalátorů (musí být v **`Path`**)

Instalátor	OS	Cesta instalace	Požadavky
EXE	Win	• **Uživatel:** %LOCALAPPDATA% • **Systém**: %ProgramFiles%	• Windows • Inno Setup 5 nebo vyšší
MSI	Win	• **Uživatel:** %LOCALAPPDATA% • **Systém**: %ProgramFiles%	• Widnows • WiX 3.0 nebo vyšší
DMG	Mac	• **Uživatel:** uživatelova plocha • **Systém**: /Applications	• Mac OS X
RPM	Linux	• **Uživatel:** --- • **Systém**: /opt	• Linux • RPM build
DEB	Linux	• **Uživatel:** --- • **Systém**: /opt	• Linux • Debian packaging tool

Spustitelná aplikace s *Inno Setup*

Nutné nainstalovat *Inno Setup* (`http://www.jrsoftware.org/isdl.php`)

- Vlastní GUI ve kterém lze sestavit instalátor
- Nastavit cestu do **`Path`** – z příkazové řádky musí jít spustit **`iscc.exe`**

Inno Setup připraví instalátor

- Zajistí zjištění cest k instalaci
- Zobrazí licenci (formát **`rtf`**)
- Může být podepsán certifikátem
- Nepoužívá *MSI installer* – neměl by vyžadovat administrátorská oprávnění (ale zařadí program mezi nainstalované aplikace)

Vytváření trvá několik minut

- Instalátor – cca 170 MB overhead
- Aplikace – cca 460 MB overhead

Ant a JavaFX

Sada nových úloh (*task*) které lze použít

- Není součástí standardního Antu → doplnit jejich načtení (z nainstalované Javy / adresáře kde jsou k dispozici)
- podpora vytvořena pro Javu 8 – Ant je výchozí build nástroj pro *Netbeans* a *Eclipse*

Do hlavičky projektu:

```
<project name = "MyProject" default ="compile„
         xmlns:fx="javafx:com.sun.javafx.tools.ant">
```

Na začátek projektu doplnit inicializaci modulů pro JavaFX Ant

Inicializace JavaFX Ant

```
<target name="init-fx-tasks">
  <path id="fxant">
    <filelist>
      <file name="D:\Program Files\Java\jdk1.8.0_40\lib\ant-
javafx.jar"/>
      <file name="D:\Program
Files\Java\jre1.8.0_40\lib\ext\jfxrt.jar"/>
    </filelist>
  </path>

  <taskdef resource="com/sun/javafx/tools/ant/antlib.xml„
  uri="javafx:com.sun.javafx.tools.ant"classpathref="fxant"/>
</target>
```

Vytvoření .jar souboru pro JavaFX

```
<target name = "fxjar" depends="init-fx-tasks,
                                compile">
  <fx:jar destfile="fxjar/application.jar">
    <fx:application name="Standalone demo„
                    MainClass="${main-class}"/>
    <fileset dir="${build.dir}"/>

    <manifest>
      <attribute name="Implementation-Vendor„
                 value="Richard Lipka"/>

      <attribute name="Implementation-Version„
                 value="1.0"/>

    </manifest>

  </fx:jar>

</target>
```

Vytvoření celé distribuce

```
<target name ="standalone" depends="fxjar">
  <fx:deploy nativeBundles="all" width="600"
             height="400" outdir="${basedir}/standalone"
             outfile="standaloneDemo">

    <fx:application name="Standalone demo"
                    mainClass="${main-class}"/>
    <fx:resources>
       …
    </fx:resources>

    <fx:platform>
      <fx:jvmarg value="-Xmx1024m"/>
      <fx:jvmarg value="-verbose:jni"/>
    </fx:platform>

    <fx:preferences install="false"/>
  </fx:deploy>
</target>
```

Děkuji za pozornost

OTÁZKY?

NÁZORY, PŘIPOMÍNKY … ?

PŘÍŠTĚ: ZKOUŠKA