Building Large Programs

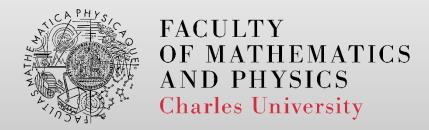
(Sestavování velkých aplikací)

http://d3s.mff.cuni.cz



Pavel Parízek

parizek@d3s.mff.cuni.cz



Outline



- Specific tools
 - Ant (Java)
 - Maven (Java)
 - MSBuild (C#/.NET)
 - Gradle (Java, Android)
 - CMake (C++)



Key topics



Avoiding cyclic dependencies

How to create modular build scripts



Ant



Ant

- Build tool mostly for Java projects
 - Wide support of tools and frameworks common in the Java world (JUnit, JSP/Servlets, EJB, ...)
- Web: http://ant.apache.org/
 - Docs: http://ant.apache.org/manual/index.html
- Highly extensible
 - Plug-ins written in Java
- Very portable scripting
- Scripts written in XML
 - Default file name: build.xml



Build file structure

```
oject name="MyProject" default="dist" basedir=".">
  cproperty name="src.dir" value="./src"/>
  cproperty name="build.dir" value="./build"/>
  <target name="init">
     <mkdir dir="${build.dir}"/>
  </target>
  <target name="compile" depends="init">
      <javac srcdir="${src.dir}" destdir="${build.dir}"/>
  </target>
  <target name="clean">
     <delete dir="${build.dir}"/>
  </target>
</project>
```

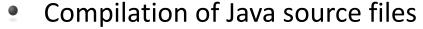


Terminology

- Task
 - Specific action to be performed during the build process
 - execute the Java compiler, create new directory
- Target
 - Goal required for building (compilation, packaging, running tests, generating documentation)
 - One phase of the whole process of building your project
 - Set of tasks that must be executed to fulfill the goal
 - May have dependencies on (multiple) other targets
- Project
 - Set of targets relevant for the application
- Property
 - name-value pair (strings)
 - usage: \${prop.name}



Basic tasks

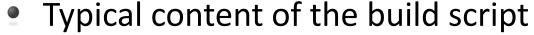


```
<javac srcdir="${src.dir}" destdir="./build"
    debug="on" deprecation="on"/>
```

Running an external Java program

Packaging class files in JAR archive

Usage



- Compilation
 - All source code files written in Java
- Packaging
 - Creating the JAR archive for distribution
- Execution of tests

Good practice

- Use properties where it makes sense, typical directory layout (./src, ./build), and standard targets (compile, build, init, clean, dist)
- Specify reasonable dependencies between targets
- Running Ant
 - Command-line: ant <target name>



Dependencies between targets

Build script

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="A,C"/>
<target name="E" depends="D,C,A"/>
```

Execution order

$$E \rightarrow D,C,A,E$$
 $D,C,A,E \rightarrow A,C,D,C,A,E$
 $A,C,D,C,A,E \rightarrow A,B,C,D,C,A,E$
 $A,B,C,D,C,A,E \rightarrow A,A,B,C,D,C,A,E$
 $A,A,B,C,D,C,A,E \rightarrow A,A,B,C,D,A,B,C,A,E$
 $A,A,B,C,D,A,B,C,A,E \rightarrow A,B,C,D,E$



Path-like structures

```
<path id="myapp.classpath">
  <pathelement path="${classpath}"/>
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
  <pathelement location="classes"/>
  <dirset dir="${build.dir}">
    <include name="apps/**/classes"/>
    <exclude name="apps/**/*Test*"/>
  </dirset>
  <pathelement location="third party/util.jar"/>
</path>
<iavac ...>
  <classpath refid="myapp.classpath"/>
</javac>
```

This is a modified version of an example from the Apache Ant documentation

Properties defined externally

• Create the file build.properties

• ... and include the file in build.xml

cproperty file="build.properties"/>

Dependencies between source files

- Recompile everything from scratch
 - We can probably recommend this approach

- Use task <depend>
 - Deletes all obsolete .class files (modified sources)
 - Re-use of some previously compiled class files
 - Limitation: cannot discover some dependencies
 - Example

```
<depend srcdir="./src" destdir="${build.dir}"/>
```



Ant – final remarks

- Examples of build.xml
 - https://github.com/apache/ant/
 - https://github.com/javapathfinder/jpfcore/tree/JPF-8.0



Maven

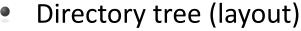


Maven

- Project management and building tool
 - mainly for Java
- Typical usage scenarios made simpler for users
- Encourages best-practices and conventions
 - Directory layout
 - Naming of tests
- Web: http://maven.apache.org/



Best-practice guidelines



```
my-app
       pom.xml
    -- src
                          -- mycompany
                  resources
                          -- mycompany
           -- classes
```

Test case names

```
**/*Test.java, **/Test*.java
```

Example taken from http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

Key concepts



- Single action to be executed
 - Construction of directory layout
 - Compilation of Java sources
- Similar to task in Ant

• Phase

- Step in the build lifecycle
 - generate-sources, compile, deploy
- Sequence of goals
- Similar to target in Ant
- Build lifecycle
 - Ordered sequence of phases
 - Similar to dependencies between targets in Ant



Typical build lifecycle

- 1. validate
- 2. compile
- 3. test
- 4. package
- 5. integration-test
- 6. verify
- 7. install ----- to local repository
- 8. deploy



Project Object Model (POM)

- Project's configuration (build script)
 - Stored in the pom.xml file

```
<modelVersion>4.0.0</modelVersion>
 <groupId>com.mycompany.app</groupId>
 <artifactId>my-app</artifactId>
 <packaging>jar</packaging>
 <version>1.0-SNAPSHOT</version>
 <name>Maven Quick Start Archetype</name>
 <url>http://maven.apache.org</url>
 <dependencies>
   <dependency>
     <groupId>junit
     <artifactId>junit</artifactId>
     <version>4.8.2
     <scope>test</scope>
   </dependency>
 </dependencies>
</project>
```

Example taken from http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

Usage

Project setup

```
mvn archetype:generate \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DgroupId=com.mycompany.app -DartifactId=my-app
```

- Build lifecycle: mvn < name of a phase >
 - Compilation: mvn compile
 - Packaging: mvn package
 - Web-site generation: mvn site
 - Rebuild into local repository: mvn clean install
- Default remote repository (central)
 - https://repo1.maven.org/maven2/



Advanced features

- Creating local repositories
- Creating packages with metadata
 - To be stored into repository
- Modifications of standard workflow
- Modules and project inheritance
- Extensibility via plugins
 - Each plugin implements a set of related goals
 - Core: http://maven.apache.org/plugins/index.html
 - Mojohaus: https://www.mojohaus.org/plugins.html



Selected plugins

- JAR
 - https://maven.apache.org/plugins/maven-jar-plugin/
- Clean
 - https://maven.apache.org/plugins/maven-clean-plugin/

- Exec
 - https://www.mojohaus.org/exec-maven-plugin/

Examples

- http://d3s.mff.cuni.cz/files/teaching/nswi154/ma ven-ex.tgz
 - DSI Utilities: original sources, build.xml, pom.xml
 - Project home page: http://dsiutils.di.unimi.it/
- Maven itself
 - https://github.com/apache/maven-parent
 - Shared global declarations
 - https://github.com/apache/maven (core)
 - Hierarchy of modules (pom.xml files): api, impl
 - https://maven.apache.org/scm.html
 - https://github.com/apache/maven-sources



Want to know more about Maven?

- Read the guide
 - http://maven.apache.org/guides/

- Try it yourself
 - Create new project
 - Add source files
 - Run compilation



Evaluation: Ant versus Maven

Ant

- Very flexible, gives you control over the build
- Better for small/student projects (less overhead)

Maven

- Quite heavy, enforces lot of best practices
- Good for large SW projects (enterprise-level)
- Support for modular build scripts is nice
 - Pitfalls: cyclic dependencies between modules



MSBuild



MSBuild

- XML syntax of build scripts ("Makefiles")
- Used internally by Visual Studio 20xx-22
- Syntax evolving (non-trivial differences)
- Familiar concepts: task, target, property

- Homepage
 - https://learn.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2022



NuGet

Package manager for .NET

Similar concepts to Maven

Integration to Visual Studio

Web: https://www.nuget.org/

Docs: https://learn.microsoft.com/en-us/nuget/

.NET Core Templates

- Support for project templates
 - Implementing best & recommended practices

- Additional information
 - https://learn.microsoft.com/en-us/dotnet/core/tools/custom-templates
 - https://learn.microsoft.com/en-us/dotnet/core/tools/dotnet-new
- Template engine
 - https://github.com/dotnet/templating/
- Available templates
 - https://github.com/dotnet/templating/wiki/Available-templates-for-dotnet-new



Gradle



Gradle

- Another popular general-purpose build tool
 - Java, Scala, C, C++, Android
- Encourages best practices (like Maven)
- Script language (DSL) based on Groovy

• Web: https://gradle.org/



Gradle – example build script

• Structure of the script file build.gradle

```
plugins {
  id 'application' // 'java-library', 'java'
java {
  toolchain {
    languageVersion = JavaLanguageVersion.of(11)
sourceSets { ... }
dependencies { ... }
// other custom tasks
```

Gradle – example build script

Fragments of the build script (configuration)

```
sourceSets {
 main {
    java {
      srcDirs = ['src']
dependencies {
  implementation files('lib/commons-logging-1.0.3.jar')
  implementation fileTree(dir: 'lib', include: '**/*.jar')
```

Gradle – example build script

Fragments of the build script (actions, custom tasks)

```
tasks.register('initDirs') {
 doLast {
    mkdir "build"
tasks.named('clean') {
 delete "build"
tasks.register('copyJar', Copy) {
 from layout.buildDirectory.dir("libs/junit.jar")
  into "dist"
task copyJarToBin(type: Copy) {
 from 'build/libs/GradleJarProject.jar'
  into "/usr/bin"
```



Gradle – usage

- Running
 - gradle clean build
 - gradle run
- Project template for Java
 - gradle init --type java-application
- Wrapper script: gradlew. {bat, sh}
 - Highly recommended to provide for customers
- Additional information
 - https://docs.gradle.org/current/userguide/tutorial using tasks.html
 - https://docs.gradle.org/current/userguide/building_java_projects.html



CMake



CMake

- Cross-platform free and open-source build management application
- Very popular (usage) for programs in C++
- Compiler-independent tool
 - Supports various native build systems (make, Xcode, MS Visual Studio)
- Web: https://www.cmake.org/
- Two phases of the build process
 - Generate native build scripts from platform-independent configuration (CMakeLists.txt)
 - Run target platform's native tool for the actual build



CMake – build script

```
cmake minimum required(VERSION 3.10)
project(myapp)
add_executable(myapp myapp.cpp myapp_gui.cpp)
target include directories(myapp include)
add library(mylib mylib core.cpp mylib utils.cpp)
add subdirectory(mylib)
target link libraries(myapp mylib)
find package(solver REQUIRED)
target link libraries(myapp ${Solver LIBS})
```



Other build tools

- Ivy
 - https://ant.apache.org/ivy/
- Scons
 - http://www.scons.org/
- Bazel
 - http://bazel.io/
- Cake
 - https://cakebuild.net/



Useful skills

- Experience with build scripts for some tools
 - Creating new scripts from scratch for your own small projects
 - Editing some parts of already existing scripts and configurations

- Recommendations
 - Learn some tools that are new for you ("broaden your horizons")
 - Ability to modify large scripts is really important



Homework

- Assignment
 - ReCodEx: group associated with this course
 - Web: http://d3s.mff.cuni.cz/files/teaching/nswi154/ukoly
- Deadline
 - **19.3.2025**

