Software Build Automation & Ant

Software Build Automation (1/3)

- It is NOT continuous integration
- Command-line based tool
- Standard features of software build automation:
  - Dependencies management
  - Source code compilation (if applicable)
  - Software packaging
  - Automatic execution of unit tests
- Additional features:
  - Documentation generation
  - What else?

Software Build Automation (2/3)

- Benefits:
  - Simplifies the build and unit testing stages
  - Dramatically reduces the number of human operations required...
  - … and the corresponding failures
  - Saves time
- Drawbacks:
  - Initial setup may require some time

Software Build Automation (3/3)

- Some well-known build automation tools:
  - Make (GNU make, nmake, etc.)
  - Apache Ant
  - Apache Maven (?)

Ant – In a nutshell (1/2)

- Ant stands for Another Neat Tool
- Open Source (Apache Public License)
- http://ant.apache.org/
- Platform independent/highly portable:
  - Java-based runtime
  - XML-based build files

Ant – In a nutshell (2/2)

- Out-of-the box integration with tons of tools/utilities:
  - Archiving utilities
  - JUnit
  - SQL
  - etc.
- Highly extensible through the notion of *tasks*
- Does stuff based on a *build file*

Ant – Bases (1/6)

- Projects
  - Contain a set of *properties* and *targets* sharing a common goal (e.g. building a Java EE Web app)
  - There can be one and only one project per build file
  - A project can *call* other projects
- Properties
  - They are NOT variables
  - A property is set once (it is immutable)
  - They have to be used to make build files as generic as possible

Ant – Bases (2/6)

- Targets
  - They group together units of work (*tasks*)
  - One target usually achieve a single goal (e.g. compiling the code)
  - A target can be dependent on several other ones (e.g unit testing depends on compiling the code)
- Tasks
  - They are individual units of work
  - They are controlled using attributes and/or nested elements
  - Examples: copy, delete, javac, sql, zip, etc.

Ant – Bases (3/6)

```
<project
  name="project-builder"
  default="compile"
  basedir=".">
  …
</project>
```

Ant – Bases (4/6)

```
<project …>
  …
  <property environment="env"/>
  <property
    name="bin.path"
    value="${basedir}/bin"/>
  …
</project>
```

Ant – Bases (5/6)

```
<project …>
  …
  <target name="init">
    …
  </target>
  <target name="compile" depends="init">
    …
  </target>
  …
</project>
```

Ant – Bases (6/6)

```
<project …>
  …
  <target name="init">
    <delete includeEmptyDir="true">
      <fileset dir="${bin.path}"/>
    </delete>
    <mkdir dir="${bin.path}"/>
  </target>
  …
</project>
```

Ant – Some Concepts (1/6)

- Filesets
  - Used to specify a set of files
  - Are usually nested elements (e.g. inside a task)
  - Defined using the `<fileset>` tag
- Filelists
  - Used to specify an explicit and static list of files
  - Defined with `<filelist>`
- Dirsets
  - They are like filesets, but dedicated to folders
  - Defined with `<dirset>`

Ant – Some Concepts (2/6)

- References
  - Filesets, filelists, dirsets, paths, classpaths, etc., can be defined once and reused several times by using an `id`
  - They are reused using a `refid`
- Paths/Classpaths
  - Are used to define path and classpath structures, with the right path separator for the underlying OS (';' on Windows, ':' on Linux)
  - Are defined using `<path>` and `<classpath>`

Ant – Some Concepts (3/6)

```
…
<copy todir="dest">
  <fileset
    dir="src"
    includes="**/*"
    excludes="**/.copyarea.db"/>
</copy>
…
```

Ant – Some Concepts (4/6)

```
…
<copy todir="dest">
  <fileset dir="src">
    <include name="**/*"/>
    <exclude name="**/.copyarea.db"/>
  </fileset>
</copy>
…
```

Ant – Some Concepts (5/6)

```
…
<classpath>
  <pathelement location="extra/tools.jar"/>
  <fileset
    dir="lib"
    includes="**/*.jar"/>
</classpath>
…
```

Ant – Some Concepts (6/6)

```
…
<fileset
  id="libraries"
  dir="lib"
  includes="**/*.jar"/>
<classpath>
  <pathelement location="extra/tools.jar"/>
  <fileset refid="libraries"/>
</classpath>
…
```

Ant – External Tasks (1/6)

- Ant features a plugin mechanism to add external tasks

- External tasks are packaged as JAR files that have to be:
  - Either referenced using Ant's `-lib` option
  - Or put in `<ANT_HOME>/lib`
  - Or referenced using Ant's `<classpath>` instruction
  - Or put in `~/.ant/lib` (highly discouraged – *why?*)

Ant – External Tasks (2/6)

- External tasks have then to be declared using the `<taskdef>` instruction:
  - By declaring each task one by one

```
<taskdef
  name="foreach"
  classname=
    "net.sf.antcontrib.logic.ForEach"/>
```

  - Or by using a tasks bundle descriptor (preferred way)
    - Properties file
    - XML file

Ant – External Tasks (3/6)

- Sample properties file-based descriptor:

```
if=net.sf.antcontrib.logic.IfTask
foreach=net.sf.antcontrib.logic.ForEach
throw=net.sf.antcontrib.logic.Throw
trycatch=net.sf.antcontrib.logic.TryCatchTask
switch=net.sf.antcontrib.logic.Switch
outofdate=net.sf.antcontrib.logic.OutOfDate
runtarget=net.sf.antcontrib.logic.RunTargetTask
…
```

Ant – External Tasks (4/6)

- Sample XML file-based descriptor:

```
<antlib>
  <taskdef
    resource="net/sf/antcontrib/antcontrib.properties"/>
  <typedef
    name="isgreaterthan"
    classname="net.sf.antcontrib.logic.condition.IsGreaterThan"/>
  <typedef
    name="islessthan"
    classname="net.sf.antcontrib.logic.condition.IsLessThan"/>
  …
</antlib>
```

Ant – External Tasks (5/6)

```
…
<taskdef
  resource=
    "net/sf/antcontrib/antcontrib.properties">
  <classpath>
    <pathelement
      location="blabla/ant-contrib-0.6.jar"/>
  </classpath>
</taskdef>
…
```

Ant – External Tasks (6/6)

```
…
<target name="do-for-all">
  <foreach
    list="a,b,c"
    target="do-for-one"
    param="message"/>
</target>
<target name="do-for-one">
  <echo message="${message}"/>
</target>
…
```

Ant – Writing Tasks (1/13)

- Ant provides two mechanisms to write custom classes
- A task can be a *bean* (that is, a plain old Java class) which provides the two following methods:
  - `public void execute()` → Mandatory
  - `public void setProject(Project project)` → Optional
  - To be used for very simple tasks only
  - *What are the two mechanisms behind this?*

Ant – Writing Tasks (2/13)

```
import org.apache.tools.ant.Project;
public class MyTask {
  private Project project;
  public void execute() {
    project.log("blablabla");
  }
  public void setProject(Project project) {
    this.project=project;
  }
}
```

Ant – Writing Tasks (3/13)

```
…
<taskdef
   name="sayBlabla"
   classname="MyTask"
   …/>
<target name="mytarget">
   <sayBlabla/>
</target>
…
```

Ant – Writing Tasks (4/13)

- A task can extend Ant's `Task` class:
- `Task` provides the following methods:
  - `void init()`
  - `void execute()`
  - `void log(…)`
  - `Target getOwningTarget()`
  - `Project getProject()`
  - etc.

Ant – Writing Tasks (5/13)

```
import org.apache.tools.ant.Task;
public class MyTask extends Task {
  public void execute() {
    log("blablabla");
  }
  public void init() {
    log("Initializing " + getClass().getName());
  }
}
```

Ant – Writing Tasks (6/13)

- To work with attributes, it's necessary to define the following method for each attribute to be supported:
  - `public void set<Attribute>(<Type> value)`
  - Such a method is standardized and is called a *setter* (as opposed to a *getter*)
  - The possible types are restricted:
    - Java primitives (and corresponding classes)
    - `String`, `File`, `Class`
    - `org.apache.tools.ant.types.Path`
    - etc.

Ant – Writing Tasks (7/13)

```
import org.apache.tools.ant.Task;
public class MyTask extends Task {
  protected String message;
  public void setMessage(String message) {
    this.message = message;
  }
  public void execute() {
    log(message);
  }
}
```

Ant – Writing Tasks (8/13)

```
…
public class MyTask extends Task {
  protected String message;
  public void setMessage(String message) {…}
  public void execute() {
    Echo echo = new Echo();
    echo.setProject(getProject());
    echo.setMessage(message);
    echo.execute();
  }
}
```

Ant – Writing Tasks (9/13)

```
…
<taskdef
   name="saySomething"
   classname="MyTask"
   …/>
<target name="mytarget">
   <saySomething message="blabla"/>
</target>
…
```

Ant – Writing Tasks (10/13)

- Nested elements:
  - A nested element is mapped to a single class using the same principles as tasks
  - A nested element is added to its parent task (or nested element) using one of the three methods:
    - `public NestedElement createNestedElement()`
    - `public void addNestedElement(NestedElement ne)`
    - `public void addConfiguredNestedElement(NestedElement ne)`

Ant – Writing Tasks (11/13)

```
…
<taskdef
   name="saySomething"
   classname="MyTask"
   …/>
<target name="mytarget">
   <saySomething>
      <message text="blabla"/>
   </saySomething>
</target>
…
```

Ant – Writing Tasks (12/13)

```
…
public class Message {
  protected String text;
  public String getText() {
    return text;
  }
  public void setText(String text) {
    this.text = text;
  }
}
```

Ant – Writing Tasks (13/13)

```
public class MyTask extends Task {
  protected Message message;
  public void addConfiguredMessage(Message m) {
    message = m;
  }
  public void execute() {
    …
    echo.setMessage(message.getText());
    echo.execute();
  }
}
```