

Java Packaging for Developers

(Get someone to package your Java application)

Red Hat

Author: *Stanislav Ochoťnický*
sochoťnický@redhat.com

Date: *5th February 2011*

Abstract

Basic guidelines for developers to make it easier for their applications to get into Linux distributions. Handling source releases, build systems and dependencies and making packagers happy in the process.

Main audience: Java developers with little packaging experience.

Linux distributions

- Take source code
- Build binary packages
- Test
- Fix
- Release packages

Thanks to Thierry Carrez <thierry (at) openstack (dot) com>

for providing next three introductory slides and several ideas from:

<http://fnords.wordpress.com/2010/09/24/the-real-problem-with-java-in-linux-distros/>

Java upstreams

- Write source code
- Build binary bundle with deps
- Test
- Fix
- Release binary bundle

Solutions

- Package binary bundle
- Package all versions of all libs
- Force software to use our versions of libs

Solutions

- Package binary bundle
- Package all versions of all libs
- Force software to use our versions of libs

Solutions

- Package binary bundle
- Package all versions of all libs
- Force software to use our versions of libs

Source releases

- Make complete source releases with build scripts

Results of no source releases

```
# latest release doesn't generate javadocs and there is no source
# tarball with pom.xml or ant build file
#
# svn export -r86 http://atinject.googlecode.com/svn/trunk atinject-1
# tar caf atinject-1.tar.xz atinject-1
```

Dependencies

- Try to pick dependencies from major projects
- Don't add another xml parser dep just because...
- Do not patch your dependencies.
- Know where your deps are coming from

Z, oh where are thou?

What are micro revisions?

- Z in X.Y.Z version string
- Micro revisions are smallest released changes
- Usually contain only bugfixes
- ALWAYS backward compatible
- No new dependencies

Java ecosystem understanding of Z in X.Y.Z

- Anything goes (for a LOT of projects)
- Binary compatibility does not matter in Java

Z, oh where are thou? (cont.)

Example Maven update from 3.0 to 3.0.1

- Required Aether update from 1.7 to 1.8
- Aether had new dependency on async-http-client
- Async-http-client added netty, jetty 7.x, etc
- Fortunately only netty was runtime dependency

How to do it?

- Z update = only backward-compatible bugfixes
- No changes to dependencies
- Check required deps for new dependencies

How to use Ant properly?

Short answer?

DON'T

Long answer

- Try apache-ivy instead of bundling dependencies
- Use properties to reuse definitions
- Place all dependencies into one directory
- Use name-version.jar for dependencies
- Don't be overly smart when writing build.xml

Advantages of Maven over Ant

For developers

- Declarative instead of descriptive
- Project metadata information in one place
- Good integration with other tools
- Support for running Ant for parts of build

For packagers

- Declarative instead of descriptive
- Clear dependencies including their versions
- No bundling of dependencies
- **Problems are the same in all projects**

Not easy to make a mess with Maven, but...

- Sometimes too easy to add new dependency
- Careful with `maven-dependency-plugin:copy-dependencies`
- Use `maven-bundle-plugin` carefully
 - Useful for metadata manipulations
 - `Export-Package` can include dependencies in resulting jar
 - This is practically static linking
- Please don't use `maven-shade-plugin`
 - Relocates package classes into different package
 - Hidden static linking
 - Impossible to reveal just from contents of jar

OSGI side of things

Make your jars OSGI-enabled:

```
...  
<packaging>bundle</packaging>  
...  
<build>  
  <plugins>  
    <plugin>  
      <groupId>org.apache.felix</groupId>  
      <artifactId>maven-bundle-plugin</artifactId>  
      <extensions>>true</extensions>  
    </plugin>  
  </plugins>  
</build>  
...
```

Using maven-assembly-plugin to release source

```
<build>
<plugins>
  ...
  <plugin>
    <artifactId>maven-assembly-plugin</artifactId>
    <configuration>
      <descriptorRefs>
        <descriptorRef>project</descriptorRef>
      </descriptorRefs>
    </configuration>
    <executions>
      <execution>
        <id>make-assembly</id>
        <phase>package</phase>
        <goals> <goal>single</goal> </goals>
      </execution>
    </executions>
  </plugin>
  ...
</plugins>
</build>
```

maven-shade-plugin in action

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <configuration>
    <relocations>
      <relocation>
        <pattern>org.objectweb.asm</pattern>
        <shadedPattern>org.packager</shadedPattern>
      </relocation>
    </relocations>
  </configuration>
</plugin>
```


maven-shade-plugin in action (cont.)

Contents of resulting jar

```
META-INF/  
META-INF/MANIFEST.MF  
META-INF/maven/  
META-INF/maven/org.packager/  
META-INF/maven/org.packager/Pack/  
META-INF/maven/org.packager/Pack/pom.properties  
META-INF/maven/org.packager/Pack/pom.xml  
org/  
org/packager/  
org/packager/signature/  
org/packager/signature/SignatureReader.class  
org/packager/signature/SignatureVisitor.class  
org/packager/signature/SignatureWriter.class  
org/packager/Pack.class
```

Can you tell where is signature sub-package coming from?

maven-bundle-plugin in action (cont.)

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <extensions>>true</extensions>
  <configuration>
    <instructions>
      <Export-Package>org.objectweb.asm.signature</Export-Package>
    </instructions>
  </configuration>
</plugin>
```

maven-bundle-plugin in action (cont.)

Contents of resulting jar

```
META-INF/MANIFEST.MF
META-INF/
META-INF/maven/
META-INF/maven/org.packager/
META-INF/maven/org.packager/Pack/
META-INF/maven/org.packager/Pack/pom.properties
META-INF/maven/org.packager/Pack/pom.xml
org/
org/objectweb/
org/objectweb/asm/
org/objectweb/asm/signature/
org/objectweb/asm/signature/SignatureReader.class
org/objectweb/asm/signature/SignatureVisitor.class
org/objectweb/asm/signature/SignatureWriter.class
org/packager/
org/packager/Pack.class
```

Summary

Please

- Provide source releases with build scripts
- Do not use niche libraries for dependencies
- Stable, bugfix-only, no-new-deps micro releases
- Try apache-ivy
- Ship pom.xml even if you build with ant
- Don't be too smart with build.xml
- Careful with certain maven plugins

The background is a dark blue, textured surface. It features several wavy, light blue lines that sweep across the frame. In the lower right quadrant, there is a cluster of small, light-colored birds in flight, scattered across the blue background.

The end.

Thanks for listening.