Intro to Maven/Gradle

Build Systems

Maven background

- Is a Java build tool
  - “project management and comprehension tool”
- An Apache Project
  - Mostly sponsored by Sonatype
- History
  - Maven 1 (2003)
    - Very Ugly
    - Used in Stack 1
- Maven 2 (2005)
  - Complete rewrite
  - Not backwards Compatible
  - Used in Stack 2.0,2.1,2.2,3.0

Maven features

- Dependency System
- Multi-module builds
- Consistent project structure
- Consistent build model

Maven mindset

- All build systems are essentially the same:
  - Compile Source code
  - Copy Resource
  - Compile and Run Tests
  - Package Project
  - Deploy Project
  - Cleanup

- Describe the project and configure the build
  - You don’t script a build
  - Maven has no concept of a condition

Maven POM

- Stands for Project Object Model
  - Describes a project
  - Name and Version
  - Artifact Type
  - Source Code Locations
  - Dependencies
  - Plugins
  - Profiles (Alternate build configurations)

Project naming (GAV)

- Maven uniquely identifies a project using:
  - groupID: Arbitrary project grouping identifier (no spaces or colons)
    - Usually loosely based on Java package
  - artfiactId: Arbitrary name of project (no spaces or colons)
  - version: Version of project
    - Format {Major}.{Minor}.{Maintanence}
    - Add '-SNAPSHOT ' to identifv in development
- (

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project>
3      <modelVersion>4.0.0</modelVersion>
4      <groupId>com.orasi</groupId>
5      <artifactId>maven-training</artifactId>
6      <version>1.0</version>
7  </project>
```

Packaging

- Build type identified using the “packaging” element
- Tells Maven how to build the project
- Example packaging types:

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.orasi</groupId>
    <artifactId>maven-training</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
</project>
```

tom

Project Inheritance

- Pom files can inherit configuration
  - groupId, version
  - Project Config
  - Dependencies
  - Plugin configuration

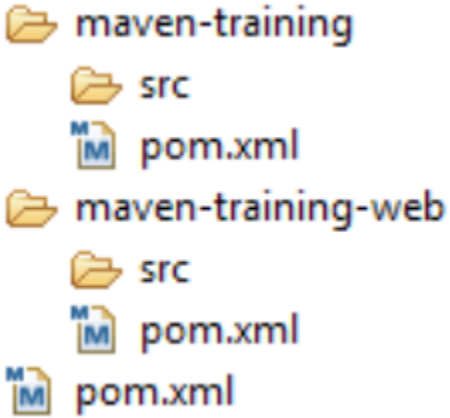
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <parent>
        <artifactId>maven-training-parent</artifactId>
        <groupId>com.orasi</groupId>
        <version>1.0</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>maven-training</artifactId>
    <packaging>jar</packaging>
</project>
```

Multi Module Projects

- Maven has 1st class multi-module support
- Each maven project creates 1 primary artifact

```
1  <project>
2      ...
3      <packaging>pom</packaging>
4      <modules>
5          <module>maven-training</module>
6          <module>maven-training-web</module>
7      </modules>
8  </project>
```

group modules



Maven Conventions

- Maven is opinionated about project structure
- target: Default work directory
- src: All project source files go in this directory
- src/main: All sources that go into primary artifact
- src/test: All sources contributing to testing project
- src/main/java: All java source files
- src/main/webapp: All web source files

Maven Build Lifecycle

- A Maven build follow a lifecycle
- Default lifecycle
  - generate-sources/generate-resources
  - compile
  - test
  - package
  - integration-test (pre and post)
  - install
  - deploy
- There is also a Clean lifecycle

Example Maven Goals

- To invoke a Maven build you set a lifecycle “goal”
- mvn install
  - Invokes generate* and compile, test, package, integration-test, install
- mvn clean
  - Invokes just clean
- mvn clean compile
  - Clean old builds and execute generate*, compile
- mvn compile install
  - Invokes generate*, compile, test, integration-test, package, install

Install JDK

- go to http://www.oracle.com/technetwork/java/javase/downloads/index.html
- download Java SE 7u51 JDK
- create JAVA_HOME variable

Install Maven

- go to: http://maven.apache.org/download.cgi
- grab apache-maven-3.1.1-bin.zip/tar.gz
- unzip into desired location: C:\apache-maven-3.1.1 or /root/apache-maven-3.1.1
- add to Path

Maven settings.xml

- Contains elements used to define values which configure Maven execution in various ways, like the pom.xml, but should not be bundled to any specific project, or distributed to an audience.

- These include values such as the local repository location, alternate remote repository servers, and authentication information.

settings.xml

- There are two locations where a settings.xml file may live:
  - The Maven install: $M2_HOME/conf/settings.xml
  - A user's install: ${user.home}/.m2/settings.xml
- The former settings.xml are also called global settings, the latter settings.xml are referred to as user settings.
- If both files exists, their contents gets merged, with the

Generate Maven Project

- create project directory
  - C:\Projects or /root/Projects
- run command
  - mvn archetype:generate \
    -DgroupId=com.orasi \
    -DartifactId=maven-training \
    -Dversion=1.0 \
    -DinteractiveMode=false

Inspect Project

- look at directory structure
- look at source file
- look at tests

Let’s build it

- run command
  - mvn clean install
- look at target directory contents

Let’s make another project

- Go to Projects directory
- run

```
mvn archetype:generate \
        -DgroupId=com.orasi \
        -DartifactId=maven-training-web \
        -Dversion=1.0 \
        -DarchetypeArtifactId=maven-archetype-webapp \
        -DinteractiveMode=false
```

Make a multi-module project

- copy pom.xml from one of projects and move to C:\Projects
- change artifactId to maven-training-parent
- change packaging to pom
- add modules block
- add parent block to individual project pom’s

Maven and Dependencies

- Maven revolutionized Java dependency management
  - No more checking libraries into version control
- Introduced the Maven Repository concept
  - Established Maven Central
- Created a module metadata file (POM)
- Introduced concept of transitive dependency

Adding a Dependency

- Dependencies consist of:
  - GAV
  - Scope: compile, test, provided (default=compile)
  - zip (default=jar)

```
<project>
    ...
    <dependencies>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

Maven Repositories

- Dependencies are downloaded from repositories
  - Via http
- Downloaded dependencies are cached in a local repository
  - Usually found in ${user.home}/.m2/repository
- Repository follows a simple directory structure
  - {groupId}/{artifactId}/{version}/{artifactId}-{version}.jar
  - groupId '.' is replaced with '/'
- Maven Central is primary community repo
  - http://repo1.maven.org/maven2

Proxy Repositories

- Proxy Repositories are useful:
  - Organizationally cache artifacts
  - Allow organization some control over dependencies
  - Combines repositories
- IMH uses the Nexus repository manager
- All artifacts in Nexus go through approval process
  - License verified

Defining a repository

- Repositories are defined in the pom
- Repositories can be inherited from parent

```
<project>
    ...
    <repositories>
        <repository>
            <id>ihc-nexus</id>
            <name>IHC Nexus Mirror</name>
            <url>http://artifacts.co.ihc.com/repomgr/content/groups/combined</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
</project>
```

Transitive Dependencies

- Transitive Dependency Definition:
  - A dependency that should be included when declaring project itself is a dependency
- ProjectA depends on ProjectB
- If ProjectC depends on ProjectA then ProjectB is automatically included
- Only compile and runtime scopes are transitive

Dependency Exclusions

- Exclusions exclude transitive dependencies

```
<project>
    ...
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>3.0.5.RELEASE</version>
            <exclusions>
                <exclusion>
                    <groupId>commons-logging</groupId>
                    <artifactId>commons-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
</project>
```

Optional Dependencies

- Don’t propagate dependency transitively
- Save space, memory, etc.

```
<project>
    ...
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>3.0.5.RELEASE</version>
            <optional>true</optional>
        </dependency>
    </dependencies>
</project>
```

Dependency Management

- What do you do when versions collide?
  - Allow Maven to manage it?
    - maven-enforcer-plugin with DependencyConververgence rule
    - run “mvn dependency:tree”
    - use before mentioned exclusion
    - DependencyManagement tag (next slide)
- Take control yourself

Using Dependency Management

```xml
<project>
    ...
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-core</artifactId>
                <version>3.0.5.RELEASE</version>
            </dependency>
        </dependencies>
    </dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </dependency>  <!-- No version -->
    </dependencies>
</project>
```

Conclusion

- Maven is a software project management and comprehension tool.
- Based on the concept of a project object model (POM), Maven can manage a project’s build, reporting, and documentation from a central piece of information.

So what’s Gradle?

What is Gradle?

- Gradle is a general purpose build system
- It comes with a rich build description language (DSL) based on Groovy
- Supports Build-by-Convention principle, but is very flexible and extensible
- Has built in plug-ins for Java, Groovy, Maven, etc
- Integrates well with Ant and Maven

Install Gradle

- http://www.gradle.org/downloads
- create GRADLE_HOME variable
- add to path

Let’s jump right in

- Create directory C:\Projects\gradle-training
- create text file: build.gradle
- add text:
  - task hello << {
  - println ‘Hello, World!’
  - }

Build a java project

- copy src folder from maven-training
- delete test subfolder

pom.xml vs build.gradle

```xml
<!-- The smallest possible Maven POM.xml -->
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.orasi</groupId>
  <artifactId>gradle-training</artifactId>
  <version>1.0</version>
</project>
```

```
//equivalent build.gradle
apply plugin: 'java'
```

build our project

- edit build.gradle
- add
  - apply plugin: ‘java’
- run ‘gradle build’ at prompt
- cd build\libs

Convert Maven project to Gradle

- go to C:\Projects\maven-training
- run
  - gradle setupBuild
  - gradle build
- go to C:\Projects\maven-training\build\reports\tests

End

http://www.onetimebox.org/box/C6bqG98XPhdCvHRrX