7 More Maven interview Questions & Answers

Posted on September 8, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Q1. How will you go about creating master builds or releases using Maven?

A1. **Step 1:** Log into the server that has maven set up to perform release builds.

Step 2: You need to checkout the fresh copy of the artifacts that you are going to build. For example, if using subversion (i.e. SVN) as your code repository, you can checkout

```
1 svn checkout http://svnserver/sdlc/my_project/tru
```

where http://svnserver/sdlc/my_project/trunk is the url to the artifact and “mine” is the folder into which you are going to checkout the artfiacts.

Step 3: Prepare release (for non prod grade builds to save time for below cmd you can add -Dargument “-DskipTests”)

```
1 mvn clean release:prepare -Dusername=? -Dpassword
```

Follow the prompts provided by the release plugin ensuring that the final build version is not a snapshot. The defaults should be fine in most scenarios.

The **release:prepare** process will:

1) Validate that snapshot dependencies are not being referenced.
2) Update the version of the component to that of specified when prompted.
3) Run the build for the code.
4) Create a tag in the format rel–

Step 4 (Optional): If anything goes wrong with release, please do rollback

```
1 mvn release:rollback
```

Step 5: Perform the release if prepare goes well.

```
1 mvn release:perform
```

This will:

1) Checkout the tagged code
2) Run a build
3) Deploy the release artifact(s) to the nexus

Q2. What is Nexus?
A2. Nexus is a central repository manager that stores and organizes binary software components for use in development, deployment, and provisioning. Nexus becomes the deployment destination for the components that are created by your organization using Maven.

1) You can publish your own libraries or projects in JARs, WARs, EARs, zip, tar,etc. for both sanapshot and release

versions using Nexus Pro.
2) You can host your own repository to support your custom components and share those components with other users or computers on your network.
3) Once your components are placed in the repository – they can be made available to all developers securely using access controls and SSL.
You can search for artifacts.

Q3. How will you go about configuring maven to publish artifacts to Nexus?
A3. To configure a Maven project to publish artifacts to Nexus, you'll need to add a **DistributionManagement** element to your project's pom.xml.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.
3       xsi:schemaLocation="http://maven.apache.org
4      <modelVersion>4.0.0</modelVersion>
5
6      <groupId>com.mycompany</groupId>
7      <artifactId>mycompany-parent</artifactId>
8      <version>1.0.0-SNAPSHOT</version>
9      <packaging>pom</packaging>
10
11     <distributionManagement>
12         <snapshotRepository>
13             <id>mycompany-snapshots</id>
14             <name>mycompany Snapshots</name>
15             <url>http://myhost:8080/nexus/conten
16             <uniqueVersion>true</uniqueVersion>
17         </snapshotRepository>
18         <repository>
19             <id>mycompany-releases</id>
20             <name>mycompany Releases</name>
21             <url>http://myhost:8080/nexus/conten
22         </repository>
23     </distributionManagement>
24 </project>
25
```

If your Nexus server requires authentication, you will also need to place your credentials in ~/.m2/settings.xml in a servers element.

```
1 <servers>
2   <server>
3     <id>deployment</id>
4     <username>deployment</username>
5     <password>deployment123</password>
6   </server>
```

```
7 </servers>
8
```

Once you've configured your project's pom.xml and your Maven Settings, you can deploy your project with

```
1 mvn deploy
```

which will execute the build up to the deploy phase. When you run a deploy with a project that has a snapshot version, Maven will deploy to the repository defined in snapshotRepository, and when you run a build with a project that has a release version it will deploy to the repository defined in the repository element.

Q5. What do you understand by Maven's build life cycle?
A5. The build life cycle has a number phases like **validate**, **compile**, **test**, **package**, **integration-test**, **verify**, **install**, and **deploy**. Each phase can have **0 or more goals** (i.e. tasks). These build phases are executed sequentially to complete the default life cycle. Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar), run integration tests against that package, verify the package, install the verified package to the local repository, then deploy the installed package in a specified environment.

If you want to do all the phases, you only need to call the last build phase to be executed, which is deploy: This is because if you call a build phase, it will execute not only that build phase, but also every build phase prior to the called build phase.

A project specific customization of a build cycle involves binding specific goals to specific phases beyond the default settings in the configuration file (i.e. pom.xml). The code snippet below demonstrates how the maven-check-style plugin can be configured in the pom.xml file to execute the "checkstyle" task during the verify phase to evaluate code quality.

```
<plugin>
   <groupid>org.apache.maven.plugins</groupid>
   <artifactid>maven-checkstyle-plugin</artifact
   <version>2.2</version>
   <configuration>
      <failonviolation>true</failonviolation>
      <failsonerror>true</failsonerror>
      <consoleoutput>true</consoleoutput>
      <cachefile>true</cachefile>
      <configlocation>${project.parent.basedir}/
      <suppressionslocation>${project.parent.bas
    </configuration>
    <executions>
       <execution>
           <phase>verify</phase>
           <goals>
              <goal>checkstyle</goal>
           </goals>
        </execution>
     </executions>
</plugin>
```

Q6. In your experience, what features do you expect in a build tool?

A6.

– Compile Java code and build jar, war, and ear files for deployment and release.
– Versioning and dependency management of relevant artifacts and modules.
– Execute tests and report test results.
– Run code quality check tools like Sonar, Checkstyle, Findbugs, etc.
– Environment property substitution to pick the right properties files
– Generation of files (e.g. Java source code from WSDL, AspectJ, XSL transformation, etc)
– Support for cross-platform (UNIX, Windows, etc) and IDEs (e.g eclipse, NetBeans, IntelliJ, etc)
– Proper documentation and support.

Q7. What Java build tools are you experienced with? Which tool will you choose?

A7. **Ant + Ivy** and **Maven** are the ones have been around for a while now. **Gradle** is a less matured build tool that takes a hybrid approach of both Ant and Maven, but looks very promising. Buildr is a build system for Java-based

applications, including support for Scala, Groovy and a growing number of JVM languages and tools.

Ant is more like a **build language** offering great flexibility as to how you build. You can write elegant and modular builds with it, but the main problem with this flexibility is that many don't write good build programs. For any non-trivial projects, over time it can lead to duplication of configuration between builds if proper care is not taken. If you need great flexibility in builds or you are working with some legacy code base where you have no control over the convention that Maven expects, then go for Ant. Ant is used for defining and executing the build steps. If you need dependency management to be integrated with Ant, then use Ivy as the dependency manager.

Maven is a **build framework**. Maven takes the opposite approach to Ant and expects you to completely integrate with the Maven lifecycle. Maven's philosophy is "Ant With Convention Over Configuration". For example conventions like the "Standard Directory Layout" (e.g. src/main/java, src/main/resources, src/test/java, src/test/resources, etc) , build cycles, and phases. The Maven plugin mechanism allows for very powerful build configurations, and the inheritance model allows you to define a small set of parent POMs encapsulating your build configurations for the whole enterprise, and individual projects can inherit those configurations, leaving them lightweight. If you want to do anything that is "not the Maven way" you have to write a plugin or use the Ant integration. Maven also has great tool support with IDEs like eclipse, NetBeans, and IntelliJ. Maven is very handy working with multiple-module projects and in scenarios where your project consists of several teams working on dependent modules. These modules can be automatically versioned and composed in to the application regularly.

Gradle is more of a **build language** like Ant that incorporates some of the pluses of Maven. Gradle tries to provide a hybrid solution to meet in the middle between Ant and Maven. It uses Ivy's approach for dependency resolution. It allows for convention over configuration but also includes Ant tasks as

first class citizens for greater flexibility. It also allows you to use existing Maven/Ivy repositories.

Q7. In your experience, what are some of the challenges you faced with maven, and how do you over come them?
A7. When it works which is 98% of the time it's quite nice, and when it doesn't, it can be a challenge. Some of the challenges can be resolved by keeping some tips in mind. For example,

#1: It might not automatically download a particular artefact from a repository. In this case, you can manually install it to your local repository using the following command.

```
1  mvn install:install-file -DgroupId=javax.transac
2              -DartifactId=jta -Dpackaging=jar -Dve
3              -Dfile=c:\Temp\my-commonservices-1.5.
4
```

#2: You have no control over, and limited visibility into, the dependencies specified by your dependencies (i.e. transitive dependencies). Builds will break because different copies of Maven will download different artifacts at different times. Your local build can break again in the future when the dependencies of your dependencies accidentally release new, non-backwards compatible changes without remembering to bump their version number. You can use the following commands and tools to check the dependency tree.

```
1 mvn dependency:tree
2 mvn dependency:analyze
3
```

#3: Most issues arise due to a developer's lack of understanding as to how maven works. There are IDEs based tools and Maven commands to debug and fix issues.

The IDEs like eclipse do provide very useful graphical tools to display dependencies graphically. It can show transitive dependencies as to which jar prought in other dependent jars. You can also view the effective pom.

The following flags are handy for debugging any other maven issues:

```
1 mvn -X install | tee maven_out.txt
```

To show the logical pom.xml or settings.xml being used

```
1 mvn help:effective-pom
2 mvn help:effective-settings
3
```

There are other handy options to resume your build from where maven failed in a multi module project. For example, if your build failed at proj-commons, you can run the build from this module by typing:

```
1 mvn -rf proj-commons clean install   //Resume fro
2 mvn -pl proj-commons, proj-service clean install
3 mvn -nsu clean install            //Don't update
4
```

rf –> resume from, **pl** –> project list, nsu –> no snapshot update, **am** –> also make (If project list is specified, also build projects required by the list), **amd** –> also make dependents (If project list is specified, also build projects that depend on projects on the list) .

To debug maven plugins via your IDE like eclipse,set the MAVEN_OPTS environmental variable as shown below

```
1 MAVEN_OPTS="-Xdebug -Xnoagent -Xrunjdwp:transport
2 export MAVEN_OPTS
3
```

Popular Member Posts

♦ 11 Spring boot interview questions & answers

905 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

816 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

427 views

18 Java scenarios based interview Questions and Answers

408 views

♦ 7 Java debugging interview questions & answers

324 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

311 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

304 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

301 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

251 views

♦ Object equals Vs == and pass by reference Vs value

234 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription

based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ 12 Maven interview Questions & Answers JSF interview Q&A ›

Posted in Maven, member-paid

Tags: Java/JEE FAQs, Popular frameworks

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer