12 Maven interview Questions & Answers

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

Q1. What is the difference between snapshot versions and release versions?

A1. The term "**SNAPSHOT**" means the build is a snapshot of your code at a given time, which means downloading 1.0-SNAPSHOT today might give a different file than downloading it tomorrow or day after. When you are ready to release your project, you will change **1.0-SNAPSHOT** to **1.0** in the pom.xml file. The 1.0 is the release version. After release, any new development work will stat using 1.1-SNAPSHOT and so on.

"SNAPSHOT" also means unstable version. Unlike regular versions, Maven checks for a new SNAPSHOT version in a remote repository at least once a day by default or if you use **-U** flag to every time you build. A team working on a cash-service module will release SNAPSHOT of its updated code every day to repository – let's say cash-service:1.0-SNAPSHOT replacing an older cash-service:1.0-SNAPSHOT jar. In case of released versions, if Maven once downloaded the version say cash-service:1.0, it will never try to download a newer 1.0 available in repository. In order to download the updated code, the cash-service version needs to be upgraded to 1.1.

When you release artifacts to nexus, you can have a snapshotRepository and a repository for the releases.

```
<distributionManagement>
    <snapshotRepository>
        <id>mycompany-snapshots</id>
        <name>mycompany Snapshots</name>
        <url>http://myhost:8080/nexus/content/re
        <uniqueVersion>true</uniqueVersion>
    </snapshotRepository>
    <repository>
        <id>mycompany-releases</id>
        <name>mycompany Releases</name>
        <url>http://myhost:8080/nexus/content/re
    </repository>
</distributionManagement>

```

Q2. Why do you need a SNAPSHOT version?
A2. If a team working on a cash-service jar module keeps uploading a new version every other day, the other teams that depend on this cash-service module

1. Need to be regularly notified so that they can update their pom.xml file to use the newer version.
2. In large projects, it is not easy to keep communicating these version changes. Using the older versions can cause unforeseen build issues

If you have a "SNAPSHOT" version that regularly gets overridden by the team that is making the regular changes and the team that depends on the cash-service module will also get their local repositories updated and use the latest code. This will alleviate the above 2 problems.

Q3. Do you have control over the snapshot versions?
A3. Yes. For example, a cash-service-1.0.jar library is considered as a stable version, and if Maven finds it in the local repository, it will use this one for the current build.

16 Technical Key Areas

open all | close all
- ⊞ Best Practice (6)
- ⊞ Coding (26)
- ⊞ Concurrency (6)
- ⊞ Design Concepts (7)
- ⊞ Design Patterns (11)
- ⊞ Exception Handling (3
- ⊞ Java Debugging (21)
- ⊞ Judging Experience I
- ⊞ Low Latency (7)
- ⊞ Memory Managemen
- ⊞ Performance (13)
- ⊞ QoS (8)
- ⊞ Scalability (4)
- ⊞ SDLC (6)
- ⊞ Security (13)
- ⊞ Transaction Managem

80+ step by step Java Tutorials

open all | close all
- ⊞ Setting up Tutorial (6)
- ⊞ Tutorial - Diagnosis (2
- ⊞ Akka Tutorial (9)
- ⊞ Core Java Tutorials (2
- ⊞ Hadoop & Spark Tuto
- ⊞ JEE Tutorials (19)
- ⊞ Scala Tutorials (1)
- ⊞ Spring & HIbernate Tu
- ⊞ Tools Tutorials (19)
- ⊞ Other Tutorials (45)

Now, if you need a cash-service-1.0-SNAPSHOT.jar library, Maven will know that this version is not stable and is subject to changes since it a SNAPSHOT. So, Maven will try to find a newer version in the remote repositories, even if a version of this library is found on the local repository. However, this check is made only once per day by default, but you can modify this update policy.

```
<repository>
    <id>central</id>
    <url>...</url>
    <snapshots>
        <enabled>true</enabled>
        <updatePolicy>always</updatePolicy>
    </snapshots>
</repository>

```

The other possible updatePolicy values are daily (i.e. default), interval:30 (every 30 minutes), and never.

– **always**: Maven will check for a newer SNAPSHOT version on every build.
– **daily**: Once a day (this is the default unless you use -U flag with mvn clean package -U)
– **interval:XXX**: an interval in minutes (XXX)
– **never**: Maven will never try to retrieve another version from the remote repository. It will do that only if it doesn't exist locally. The SNAPSHOT version will be handled as the stable version.

You can also turn off this by setting enabled to false. Even though Maven automatically fetches the latest SNAPSHOT on a daily basis by default, you can force maven to download latest snapshot build using -U switch to any maven command.

```
mvn clean package -U
```

Q4. What is the difference between a reactor pom and a parent pom?
A4. The mechanism in Maven that handles multi-module

100+ Java pre-interview coding tests

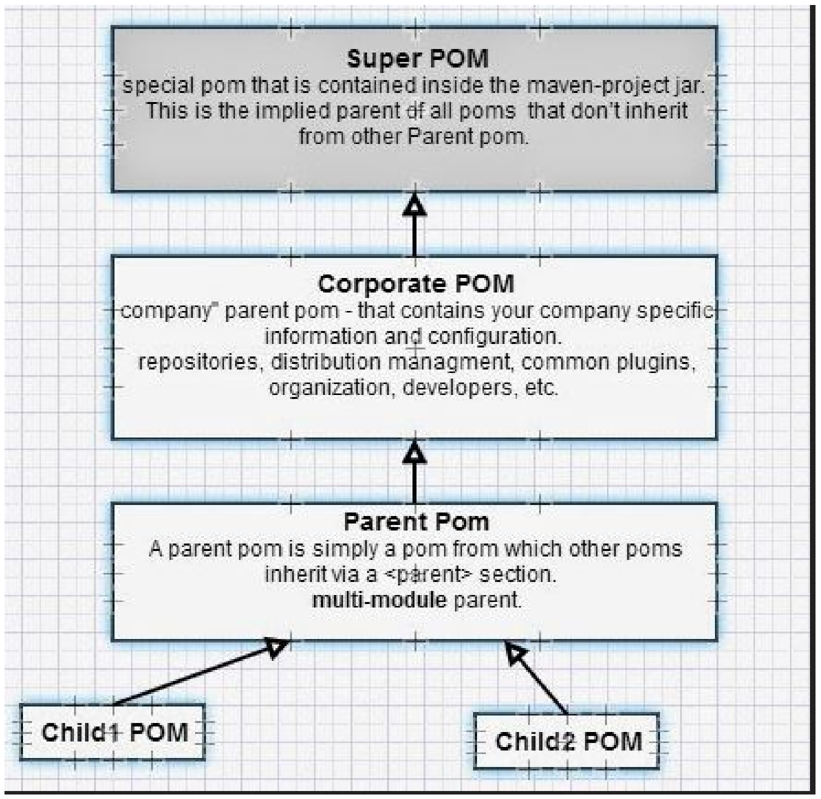
open all | close all
Can you write code? (
♦ Complete the given
Converting from A to I
Designing your classe
Java Data Structures
Passing the unit tests
What is wrong with th
Writing Code Home A
Written Test Core Jav
Written Test JEE (1)

How good are your .....?

open all | close all
Career Making Know-
Job Hunting & Resum

projects is referred to as the reactor. A reactor pom is a pom.xml file you define with , and Maven will read all of these modules and build then in the correct order based on dependencies.

A parent pom is a pom from which other poms inherit via a section. Poms that inherit from a parent are referred to as “child poms.” There are different types of parent poms.


Super POM
special pom that is contained inside the maven-project jar. This is the implied parent of all poms that don't inherit from other Parent pom.
Corporate POM
company" parent pom - that contains your company specific information and configuration. repositories, distribution managment, common plugins, organization, developers, etc.
Parent Pom
A parent pom is simply a pom from which other poms inherit via a <parent> section. multi-module parent.
Child1 POM
Child2 POM

Maven poms

Q5. How do you handle multi-module projects in Maven?
A5. A multi-module project is defined by a parent POM referencing one or more sub modules. You can have different structures like

Approach 1. parent pom is in the project root

```
myproject/
  myproject-client/
  myproject-model/
  myproject-services/
  pom.xml

```

Approach 2. separate project for the parent pom

```
myproject/
  mypoject-parent/
    pom.xml
  myproject-client/
  myproject-model/
  myproject-services/

```

The parent pom.xml will define the modules that needs to be aggregated.

```
<project xmlns="....">
  <description>MyApp</description>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.myapp</groupId>
  <artifactId>myproject</artifactId>
  <version>1.0.0-RC1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <name>MyProject</name>

  <scm>
    <url>http://sdlc/svn/myapp/myproject/trunk/<
    <connection>scm:svn:http://sdlc/svn/myapp/my
     <developerConnection>scm:svn:http://sdlc/sv
  </scm>

  <modules>
    <module>model</module>
    <module>client</module>
    <module>services</module>
  </modules>

  .....
</project>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="ns">
  <modelVersion>4.0.0</modelVersion>
  <artifactId>myproject-model</artifactId>
  <name>MyProject Model</name>
  <packaging>jar</packaging>
  <parent>
    <groupId>com.myapp</groupId>
    <artifactId>myproject</artifactId>
    <version>1.0.0-RC1.0-SNAPSHOT<</version>
    <relativePath>../pom.xml</relativePath>
 </parent>
    ...
</project>
```

Q. Which approach do you favor?

A. Approach 1 is the is the default maven convention for multi-module projects, and the intention is to be scalable to a large scale build. If parent pom has its own life cycle, and if it can be released separately of the other modules, then approach 2 may be an option.

Q6. What are the different aspects managed by Maven in the SDLC?
A6. SCMs (Source control), Dependencies (i.e transitive dependencies), Builds, Releases, Documentation, and Distribution.

Q7. What is the difference between "DependencyManagement" and "Dependencies" as shown below in poms?

Parent pom with version defined

```
<dependencyManagement>
  <dependencies>
    <dependency>
       <groupId>javax.servlet</groupId>
       <artifactId>javax.servlet-api</artifactId
       <version>3.0.1</version>
       <scope>provided</scope>
    </dependency>
    //..
  </dependencies>
</dependencyManagement>
```

and child pom with no version.

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

A7. Dependency Management allows to consolidate and centralize the management of dependency versions without adding dependencies which are inherited by all children. It will be a maintenance night-mare to change version numbers of all child poms in a large commercial project.

"DependencyManagement" allows you to define a standard version of an artifact to use across multiple projects.

Q8. What format does a maven repository use to uniquely identify an artifact in its repository?
A8. [groupId]-[artifactId]-[version]-[classifier].[type]

groupId: e.g. com.myapp
artifactId: e.g. myproject
version: e.g. 1.0.0-RC1.0-SNAPSHOT
classifier: e.g. sources, javadocs, bin and bundle (default: no classifier)
type: pom,jar,war,ear (default is jar)

Q9. What is an "attached artifact" and why would you need a "classifier"?
A9. Attached artifacts are additional related artifacts that get installed and deployed along with the "main" (e.g jar, war, or ear) artifact. These are most often, javadocs, sources, resouce bundles, properties files, etc, but can actually be any file.

Attached artifacts automatically share the same **groupId**, **ArtifactId** and **version** as the main artifact but they are distinguished with an additional **Classifier** field.

Q10. How do you inherit dependencies or plugins in Maven? How are sub modules aggregated?
A10. In Maven Inheritance, you will have a multi-module maven project where a centrally managed POM file contains all the "approved" versions of artifacts. In this "parent" pom was a "dependencyManagement" section with all the preferred versions of libraries within the organization. When you build a child of this parent, Maven will retrieve this parent to merge the parent pom with the child pom. You can have a look to the entire pom.xml by running the command mvn help:effective-pom. This parent pom is also known as the reactor pom. The child modules like myproject-ui, myproject-security, etc will be inheriting the versions from the parent.

Child pom.xml. The parent tag defines the parent to inherit form.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.
 xsi:schemaLocation="http://maven.apache.org/POM
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.myproject</groupId>
  <artifactId>myproject-ui</artifactId>
  <version>${myproject.version}</version>
  <packaging>war</packaging>
  <name>${project.artifactId}</name>
  <url>http://www.myorganization.com</url>
  <parent>
     <groupId>com.myproject</groupId>
     <artifactId>myproject-parent-build</artifac
     <version>${myproject.version}</version>
  </parent>
  ..
</project>
```

Like “dependencyManagement” the “pluginManagement” in parent contains plugin elements intended to configure child project builds that inherit from this one. However, this only configures plugins that are actually referenced within the plugins element in the children. The children have every right to override pluginManagement definitions.

The child modules are aggregated. The principle is that every command that you run on on the parent (i.e. reactor pom) will also be run on all the sub-modules. The order of the modules will be defined by the Reactor, which will look at inter-modules dependencies to find which module must be built before the others. If there are no dependencies, then it will take the list of the modules as they are defined in the parent pom.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.
 xsi:schemaLocation="http://maven.apache.org/POM
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.myproject</groupId>
  <artifactId>myproject-parent-build</artifactId
  <packaging>pom</packaging>
  <version>${myproject.version}</version>
  <name>${project.artifactId}</name>
  <url>http://www.myorganization.com</url>

  ...

  <modules>
```

```
    <module>myproject-security</module>
    <module>myproject-content</module>
    <module>myproject-data</module>
    <module>myproject-ui</module>
    <module>myproject-analytics</module>
  </modules>

</project>
```

Q. Is this inheritance of dependencies a good idea?
A. It is a good idea when you only have say 5-10 sub modules (i.e. child modules). But, if you have 50+ child modules inheriting from the parent dependency management or plugin management will cause a re-release of anything inheriting from it. For example, if you have 20+ projects inheriting org.jboss.seam jar, and if that jar goes from 2.2.2.Final to 2.3, then you will have to re-release the 20+ projects inheriting from it. It appears that as of version 2.0.9, there is a new scope on a dependency called import to aggregate dependencies without inheriting them.

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.myproject</groupId>
            <artifactId>myproject-parent-build</
            <version>${myproject.version}</versi
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        ....
    </dependencies>

```

So, like you favor composition or **aggregation** over **inheritance**, in maven pom also favor aggregation over inheritance.

Q11. What are the different dependency scopes have you used?
A11. compile, provided, test, and import.

Compile means that you need the JAR for compiling and running the app. For a web application, as an example, the JAR will be placed in the WEB-INF/lib directory.

Provided means that you need the JAR for compiling, but at run time there is already a JAR provided by the environment so you don't need it packaged with your app. For a web app, this means that the JAR file will not be placed into the WEB-INF/lib directory.

Test scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases.

import scope is used to aggregate the specified POM with the dependencies in that POM's "**DependencyManagement**" section. This scope is only used on a dependency of type pom in the section and available only from Maven version 2.0.9.

Q12. What is the purpose of Maven assemply plugin?
A12. Maven assembly plugin is primarily intended to allow users to aggregate the project output along with its dependencies, modules, site documentation, and other files into a single distribution archive like zip, tar, tar.gz, war, etc. Another example where this is handy is to create deployable files for different environments like myapp-1.10-SNAPSHOT-dev.zip, myapp-1.10-SNAPSHOT-test.zip, and myapp-1.10-SNAPSHOT-prod.zip where each containing environment related properties files packaged. There are 3 steps to create an assembly:

1) choose or write the assembly descriptor files to use like dev-assembly.xml, prod-assembly.xml, etc
2) configure the Assembly Plugin in your project's pom.xml, and include the above descriptor files
3) run "mvn assembly:single" on your project.

The dev-assembly file:

```
<assembly
 xmlns="http://maven.apache.org/plugins/maven-as
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
 xsi:schemaLocation="http://maven.apache.org/plu
 http://maven.apache.org/xsd/assembly-1.1.2.xsd"
  <id>dev</id>
```

```
    <includeBaseDirectory>false</includeBaseDirect
    <formats>
       <format>zip</format>
    </formats>
    <fileSets>
      <fileSet>
        <directory>config/dev</directory>
        <outputDirectory>/app-properties</outputDi
      </fileSet>
      <fileSet>
        <directory>jboss-config/dev/deploy</direct
        <outputDirectory>/ds</outputDirectory>
      </fileSet>
  </fileSets>
  <files>
    <file>
      <source>jboss-config/common/myapp-container
      <outputDirectory>/myapp-container-propertie
      <destName>my-app-dev.properties</destName>
    </file>
  </files>
</assembly>
```

The pom.xml file

```
<plugins>
     ....
    <plugin>
       <artifactId>maven-assembly-plugin</artifa
       <version>2.2.1</version>
       <executions>
         <execution>
            <id>assemble</id>
            <phase>package</phase>
            <configuration>
                <descriptors>
                <descriptor>assembly/dev-assembl
                <descriptor>assembly/test-assemb
                <descriptor>assembly/prod-assemb
              </descriptors>
            </configuration>
            <goals>
              <goal>single</goal>
            </goals>
         </execution>
       </executions>
    </plugin>
    ...
<plugins>
```

The main goal in the assembly plugin is the single goal. It is used to create all assemblies.

Now to generate different packages like myapp-1.10-SNAPSHOT-dev.zip, myapp-1.10-SNAPSHOT-test.zip, and myapp-1.10-SNAPSHOT-prod.zip

```
mvn package
```

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

0 Like Share

0 G+1

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**

‹ More Java annotations interview questions & answers

7 More Maven interview Questions & Answers ›

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