softwarecave

← Integrating Hibernate with Spring

Accessing local files from a web browser using HTML5 File API →

Adding external/custom jars into Maven project

Posted on June 14, 2014

One of the strongest points of Maven is that it automatically manages project dependencies. The developer just needs to specify which dependencies and in which

version are needed and the Maven takes care of the rest including downloading and storing them at the right location and additionally packaging them into final artifact (e.g. WAR). This is very convenient and almost completely removes a need to hold additional jars in *lib/* project subdirectory.

However, there is a small assumption that all required dependencies are available at one or more public repositories. It is usually the case but sometimes you may need to use a jar which is not available there for some reason. Luckily, there are few popular approaches to overcome this problem which are described below.

Adding jar to public Maven repository

Theoretically, the best way would be to add a jar to a public Maven repository. However, if the jar is proprietary, it is usually impossible to get the permission from the company to do so.

Using system dependency

The second method is to add the required dependency with the *system* scope and additionally provide an absolute path to the a jar file placed somewhere on the local disc:

```
<dependencies>
  <dependency>
    <groupId>com.example</groupId>
    <artifactId>evalpostfix</artifactId>
    <version>1.0</version>
    <scope>system</scope>
    <systemPath>${basedir}/lib/evalpostfix-1.0.jar</systemPat
  </dependency>
</dependencies>
```

The problem with this approach is that this dependency will be completely ignored during packaging and forcing Maven to add it to the final artifact (e.g. WAR) would result in a very clumsy POM file.

Installing jar into local Maven repository

Much better solution is to add the required dependency manually to the local repository using command:

```
$ mvn install:install-file -Dfile=<path-to-file> \
    -DgroupId=<group-id> -DartifactId=<artifact-id> \
    -Dversion=<version> -Dpackaging=<packaging>
```

For example adding external jar *evalpostfix-1.0.jar* to the local repository could look like this:

```
$ mvn install:install-file -Dfile=evalpostfix-1.0.jar \
     -DgroupId=com.example -DartifactId=evalpostfix \
     -Dversion=1.0 -Dpackaging=jar
(...)
[INFO] --- maven-install-plugin:2.4:install-file (default-cli
[INFO] Installing /home/robert/informatyka/softwarecave/infix
[INFO] Installing /tmp/mvninstall2671284263455462989.pom to /
(...)
```

Once the dependency is available in the local repository it can be added to POM file like any other dependency:

```
<dependencies>
  <dependency>
    <groupId>com.example</groupId>
    <artifactId>evalpostfix</artifactId>
    <version>1.0</version>
```


Recent Posts

- Importing WSDL with Java and Maven
- Objects utility class in Java
- Default and static methods in interfaces in Java 8
- Database schema creation in JPA using SQL scripts
- Reading text file line by line in Java

Archives

- February 2017 (4)
- December 2014 (3)
- November 2014 (2)
- August 2014 (1)
- July 2014 (2)
- June 2014 (2)
- May 2014 (2)
- April 2014 (5)
- March 2014 (9)
- February 2014 (11)
- January 2014 (6)
- September 2011 (1)

Categories

- Agile
- AJAX
- Algorithms
- C++
- Code coverage
- cryptography

```
    </dependency>
  </dependencies>
```

This solution is still inconvenient because every new developer working on the project would have to run *mvn install:install* command on its own workstation.

Using internal Maven repository in a company

One of the best ideas is to setup an internal Maven repository in a company for storing such dependencies. The repository should be available to every developer working on a project though HTTP or other protocol supported by Maven. Of course, the repository server does not have to be available from outside of the company.

The required dependencies should be installed on the repository server using *mvn install:install-file* command:

```
$ mvn install:install-file -Dfile=evalpostfix-1.0.jar \
      -DgroupId=com.example -DartifactId=evalpostfix \
      -Dversion=1.0 -Dpackaging=jar \
      -DlocalRepositoryPath=/opt/mvn-repository/
```

The only difference from the command in the previous section is that it additionally specifies the path on the repository server where the jars and metadata should be stored.

Once it is finished, the dependency can be added to the POM file. Additionally, the location of the new repository server is provided:

```
<repositories>
  <repository>
    <id>Internal company repository</id>
    <url>http://mvnrepo.company.com/</url>
  </repository>
```


- Database
- Defensive programming
- Git
- Guava
- Hibernate
- HTML
- Java
- Java EE
- JavaScript
- JPA
- JSF
- JTA
- Linux
- Maven
- Software development practices
- Spring
- Version control
- Web-Services
- XML

RSS - Posts
RSS - Comments

Meta

- Register
- Log in
- Entries RSS
- Comments RSS
- WordPress.com

```
</repositories>
(...)
<dependencies>
  <dependency>
    <groupId>com.example</groupId>
    <artifactId>evalpostfix</artifactId>
    <version>1.0</version>
  </dependency>
</dependencies>
```

The advantages of this approach should be clearly visible:

- new developers can start building the project without any additional preparation tasks
- no need to send jars though emails, IM or downloading them from the Internet
- reduced or completely removed need for build instructions
- all external jars are managed in a single place
- dependencies and the server can be shared by multiple projects

Using in-project Maven repository

The idea is quite similar to using internal repository server but this time the repository is stored in a directory (e.g. called *lib*) located in a project root directory. After creating the directory and installing jar files there using *mvn install:install-file* command, the dependencies and the repository can be referenced from a POM file:

```
<repositories>
  <repository>
    <id>Internal company repository</id>
    <url>file://${basedir}/lib</url>
  </repository>
</repositories>
(...)
<dependencies>
```

```
    <dependency>
      <groupId>com.example</groupId>
      <artifactId>evalpostfix</artifactId>
      <version>1.0</version>
    </dependency>
  </dependencies>
```

The created repository including jars, pom files and checksums must be stored in a version control system so that it is available to other developers. The biggest issue with this solution is that it clutters VCS repository with files that such never be placed there (e.g. jars).

Conclusion

Choosing the right solution is not always easy. Personally, I would first try to add the jar into public or at least internal Maven repository in a company. If it is not be possible, I would go for in-project Maven repository and use the other methods as a last resort.

Share this:

Twitter Facebook 1 Google Pocket Pinterest Tumblr LinkedIn Reddit Email

Like 

One blogger likes this.

Related

Creating executable jar file with Maven
In "Java"

Test driven development
In "Java"

Using JUnit, JaCoCo and Maven for code coverage
In "Code coverage"

About Robert Piasecki

Husband and father, Java software developer, Linux and open-source fan.

View all posts by Robert Piasecki →

This entry was posted in Java, Maven and tagged Java, Maven. Bookmark the permalink.

← Integrating Hibernate with Spring

Accessing local files from a web browser using HTML5 File API →

11 Responses to *Adding external/custom jars into Maven project*

dineshramitc *says:*

June 14, 2014 at 22:44

Reblogged this on Dinesh Ram Kali..

Reply

Johnf358 *says:*

August 21, 2014 at 05:00

I'm trying to find sites that have already fantastic useful information on what's popular and what is the optimum makeup products is.. afebbabebabf

Reply

Pritish Shah *says:*

October 1, 2014 at 02:41

Nice blog. I am adding jars by following Installing jar into local Maven repository section. Also, I have provided scope "provided" for those dependencies. However, maven still adding those jar files under my war file. Is there any specific reason for that? For other jars that is coming from official maven repository, it's not including under war file. Do you have any idea?

Reply

Fabio da Silva *says:*

September 3, 2015 at 05:57

No use the scope tag, works to me.

Reply

Pingback: *Maven | Brain Overflow*

Fabio da Silva *says:*

September 3, 2015 at 05:54

Very good article. Congratulations. This article helped me.
I created in my blog
http://fabiophx.blogspot.com.br/2015/09/adicionando-jars-externos-com-maven.html
a link to yours.

Reply

Nitin *says:*

October 13, 2015 at 13:45

very good blog. its usefull for me

Reply

Bernhard *says:*

January 3, 2016 at 11:32

I could not find a suitable answer to this problem. With the help of your article, I solved it in two minutes. Your teaching style is perfect. I bookmarked your site and

continue following it and you:)

Reply

Scott *says:*

September 22, 2016 at 21:30

One issue I am unable to find an easy, non-time consuming method for is the following scenario. We need to use a proprietary SDK set of libraries (SAP product) and to use this SDK, I need around 315 jars just to add about 40 lines of code. I only need these libraries for compiling because the target server already contains the libraries on the classpath. Without maven installing each of the 315 jars to my local repo and then scope them as provided, what is a better more feasible alternative??

Reply

Siempre Indeciso *says:*

March 7, 2017 at 13:29

How could we have an hybrid situation? I mean: taking it from external company repository (nighty builds) and, if version is not adequate (we might have changed the source code but we have not committed it), compiling the jar from sources and putting the jar into the local repository (.m2 folder).
Thanks!!!

Reply

Kenny EasyRealm *says:*

May 22, 2017 at 01:01

It took me hours to figure this out until I arrived at your blog. No one mentioned that the option of importing jars to local Maven repo. Great material! Shows that you've great understanding of the concept..

Reply

Leave a Reply

Enter your comment here...

softwarecave

Blog at WordPress.com.