

Petri Kainulainen

≡ Menu

Do you want to create JPA repositories without writing any boilerplate code? If so, [read my Spring Data JPA tutorial!](#)

Creating a Runnable Binary Distribution With Maven Assembly Plugin

👤 Petri Kainulainen 📅 April 9, 2011

💬 78 comments

🔖 Maven

29

SHARES

Facebook

There are many ways that we can use when we want to share our applications with other people. For example, we can create a binary distribution that can be downloaded from our website.

This blog post describes how we can create a runnable binary distribution by using the Maven Assembly Plugin.



The requirements of our binary distribution are:

- The jar file that contains the classes of our example application must be copied to the root directory of the binary distribution.
- The manifest of the created jar file must configure the classpath and the main class of our application.
- The dependencies of our application must be copied to the *lib* directory.
- The startup scripts of our application must be copied to the root directory of the binary distribution.

Let's start by taking a quick look at our example application.

The Example Application

The example application of this blog post has only one class that writes the string 'Hello World!' to the log by using Log4j. The source code of the *HelloWorldApp* class looks as follows:

```
1  import org.apache.log4j.Logger;
2
3  public class HelloWorldApp
4  {
5      private static Logger LOGGER = Logger.getLogger(HelloWorldApp.class);
6
7      public static void main( String[] args )
8      {
9          LOGGER.info("Hello World!");
10     }
11 }
```

The properties file that configures Log4j is called *log4j.properties*, and it is found from the *src/main/resources* directory. Our *log4j.properties* file looks as follows:

```
1  log4j.rootLogger=DEBUG, R
2
3  log4j.appender.R=org.apache.log4j.ConsoleAppender
4
5  log4j.appender.R.layout=org.apache.log4j.PatternLayout
6  log4j.appender.R.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
```

The startup scripts of our example application are found from the *src/main/scripts* directory. This directory contains the startup scripts for the Windows and *nix operating

systems.

Let's move on and find out how we can create a jar file that configures the classpath and the main class of our example application.

Creating the Jar File

We can package our application into a jar file by using the [Maven Jar Plugin](#). We can configure the Maven Jar Plugin by following these steps:

1. Ensure that classpath of our example application is added to the created manifest file.
2. Specify that all dependencies of our application are found from the lib directory.
3. Configure the main class of our example application.

The configuration of the Maven Jar Plugin looks as follows:

```
1  <plugin>
2    <groupId>org.apache.maven.plugins</groupId>
3    <artifactId>maven-jar-plugin</artifactId>
4    <version>2.5</version>
5    <configuration>
6      <!-- Configures the created archive -->
7      <archive>
8        <!-- Configures the content of the created manifest -->
9        <manifest>
10         <!-- Adds the classpath to the created manifest -->
11         <addClasspath>true</addClasspath>
12         <!--
13           Specifies that all dependencies of our application are
14           from the lib directory.
15         -->
16         <classpathPrefix>lib/</classpathPrefix>
17         <!-- Configures the main class of the application -->
18         <mainClass>net.petrikainulainen.mavenassemblyplugin.HelloWo
19       </manifest>
20     </archive>
21   </configuration>
22 </plugin>
```

Additional Reading:

- [Maven Jar plugin – Manifest Customization](#)
- [Maven Jar plugin – Using a Default Manifest File](#)
- [Maven Jar plugin – Use Your Own Manifest File](#)

Let's move on and find out how we can assemble the binary distribution of our example application.

Assembling the Binary Distribution

We can assemble our binary distribution by using the [Maven Assembly Plugin](#). If we want to create a binary distribution that fulfills our requirements, we have to follow these steps:

1. Create an assembly descriptor that dictates the execution of the Maven Assembly Plugin.
2. Configure the Maven Assembly Plugin to use the created assembly descriptor.

First, we have create an assembly descriptor that dictates the execution of the Maven Assembly Plugin. We can create this assembly descriptor by following these steps:

1. Create an *assembly.xml* file to the *src/assembly* directory.
2. Configure the format of our binary distribution. Ensure that a zip file is created.
3. Copy the dependencies of our application to the *lib* directory.
4. Copy the startup scripts of our application from the *src/main/scripts* directory to the root directory of the binary distribution.
5. Copy the jar file of our example application from the *target* directory to the root directory of the binary distribution.

Our assembly descriptor looks as follows:

```

1  <assembly>
2    <id>bin</id>
3    <!-- Specifies that our binary distribution is a zip package -->
4    <formats>
5      <format>zip</format>
6    </formats>
7
8    <!-- Adds the dependencies of our application to the lib directory -->
9    <dependencySets>
10     <dependencySet>
11       <!--
12         Project artifact is not copied under library directory sinc
13         it is added to the root directory of the zip package.
14       -->
15       <useProjectArtifact>false</useProjectArtifact>
16       <outputDirectory>lib</outputDirectory>
17       <unpack>false</unpack>
18     </dependencySet>
19   </dependencySets>
20
21   <fileSets>
22     <!--
23       Adds startup scripts to the root directory of zip package. The
24       scripts are copied from the src/main/scripts directory.
25     -->
26     <fileSet>
27       <directory>${project.build.scriptSourceDirectory}</directory>
28       <outputDirectory></outputDirectory>
29       <includes>
30         <include>startup.*</include>
31       </includes>
32     </fileSet>
33     <!--
34       Adds the jar file of our example application to the root direct
35       of the created zip package.
36     -->
37     <fileSet>
38       <directory>${project.build.directory}</directory>
39       <outputDirectory></outputDirectory>
40       <includes>
41         <include>*.jar</include>
42       </includes>
43     </fileSet>
44   </fileSets>
45 </assembly>

```

Second, we have to configure the Maven Assembly Plugin to use the assembly descriptor that we created in step one. We can do this by adding the following plugin declaration to the *plugins* section of our POM file:

```

1  <plugin>
2    <groupId>org.apache.maven.plugins</groupId>
3    <artifactId>maven-assembly-plugin</artifactId>
4    <version>2.5.1</version>
5    <configuration>
6      <!-- Configures the used assembly descriptor -->

```

```
7         <descriptors>
8             <descriptor>src/main/assembly/assembly.xml</descriptor>
9         </descriptors>
10     </configuration>
11 </plugin>
```

Additional Reading:

- [Maven Assembly Plugin – Assembly](#)
- [Maven Assembly Plugin – Usage](#)
- [Maven Assembly Plugin – Advanced Assembly Descriptor Topics](#)

We have now configured the Maven Assembly Plugin to create our binary distribution. Let's move on and find out what this really means.

What Did We Just Do?

We can create our binary distribution by running the command `mvn clean package assembly:single` at the command prompt. After we have done this, Maven creates the `maven-assembly-plugin-bin.zip` file to the `target` directory. This zip package is the binary distribution of our example application.

The name of the binary distribution is created by using the following formula: *[the value of the finalName element]-[the id of the used assembly].[the format of the distribution]*.

In other words, the name of the created binary distribution is `maven-assembly-plugin-bin.zip` because:

1. The value of the `finalName` element, which is found from our `pom.xml` file, is

maven-assembly-plugin.

2. The id of our assembly is *bin*.
3. We want to create a binary distribution that is packaged into a zip file.

When we run the command *unzip maven-assembly-plugin-bin.zip* at the command prompt, the *maven-assembly-plugin-bin* directory is created to the *target* directory. The content of the created directory is described in the following:

- The *maven-assembly-plugin-bin* directory contains the jar file of our example application and the startup scripts.
- The *maven-assembly-plugin-bin/lib* directory contains the *log4j-1.2.16.jar* file.

We can run our example application application by using the startup scripts found from the *maven-assembly-plugin-bin* directory. When we run our application, we see the following output:

```
1 | $ ./startup.sh
2 | 0 [main] INFO net.petrikainulainen.mavenassemblyplugin.HelloWorldApp -
```

Let's move on and summarize what we learned from this blog post.

Summary

This blog has taught us two things:

- We can customize the manifest of the created jar file by using the Maven Jar Plugin.
- We can create a binary distribution, which doesn't use the so called "fat jar" approach, by using the Maven Assembly plugin.

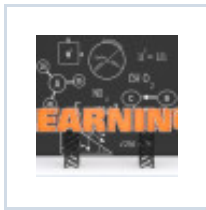
You can [get the example application of this blog post from Github](#).

If you enjoyed reading this blog post, you should follow me on Twitter:

Follow @petrikainulaine

1,135 followers

RELATED POSTS



LEARNING /

What I Learned This Week (Week 40/2013)



MAVEN /

Running Solr with Maven



UNCATEGORIZED /

10 Most Popular Blog Posts of 2014



About the Author

Petri Kainulainen is passionate about software development and continuous improvement. He is specialized in software development with the Spring Framework and is the author of [Spring Data](#) book.


[About Petri Kainulainen →](#)

Connect With Me



76 comments... [add one](#)



x78527 

May 29, 2011, 10:29

Thanks! That works for me.

REPLY



Jim Rayhawk 

July 26, 2011, 23:43

Great post!! This was exactly what I needed. Thanks for putting this together.

REPLY



Prash 

October 22, 2011, 17:57

Hi Petri,

That was a great tutorial..thanks a ton for your efforts

before i was struggling to put together my maven build as an app to invoke from batch script

your example provided just what i needed to do that all in one click

Thanks again

Regards

Prash

REPLY



Petri 

October 23, 2011, 12:33

Hi Prash,

It is good to hear that you found this tutorial useful. At least I know that I have managed to actually help someone :)

REPLY



Radu 

November 23, 2011, 11:28

Hey Petri, thanks a lot for this wonderful tutorial, I'm new to Maven and I was having a hard time trying to pack the binaries of my project into an archive with external jars and all.. your ideas served me greatly :)

Cheers!

Radu

REPLY



Petri 


November 23, 2011, 12:04

Radu,

it is good to hear that I could help you out.

REPLY



laurent 

December 12, 2011, 01:27

You helped me too. Tanks a lot.

REPLY



laurent 

December 12, 2011, 02:08

It can be useful to use 0755 so that the script file is executable.

Also, it is safer to use `${project.build.directory}` instead of `target`.

And since the jar is added at the root of the archive, you should use `false` in to prevent including it twice.

REPLY



Petri 

December 12, 2011, 13:54

Laurent,

Good to hear that I could help you out. Also, you gave some nice improvement ideas. I will update the code examples and the sample project as soon as possible.

REPLY



Petri 

December 18, 2011, 16:37

I moved the example application to GitHub and fixed the issues mentioned by Laurent.

REPLY



Jacob Lorensen 

June 15, 2012, 13:35

Hi. I may be extremely dense, and miss something obvious. But how do I get all the nice external jar files that I depend on in the maven build file included in the assembly generated jar file? It seems to me it only includes what is specifically present in my own project directories and not any dependencies.

Do I combine it with “one-jar” plugin, or?

REPLY



Jacob Lorensen 

June 15, 2012, 13:54

Forget it... had a bad-brain-day or something and forgot mvn assembly:single :(

[REPLY](#)**Petri**

June 15, 2012, 14:04

Jakob,

No worries. It happens to all of us =)

[REPLY](#)**Petri**

June 15, 2012, 14:03

Hi Jakob,

Thanks for your comment. You can configure this in the assembly descriptor. The *dependencySets* element of my assembly descriptor example configures the Maven assembly plugin to copy the dependencies of your project under the *lib* directory.

[REPLY](#)**jose antonio**

September 5, 2012, 00:12

Hi Petri,

I tried with your simple example app, but it gave me an exception at runtime

Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/log4j/Logger

```
at net.petrikainulainen.mavenassemblyplugin.HelloWorldApp.(HelloWorldApp.java:11)
Caused by: java.lang.ClassNotFoundException: org.apache.log4j.Logger
at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
```

I took a look inside the maven-assembly.jar and it didn't have a log4j.jar inside not even a lib/ dir in the target directory. Have you any idea what could it be?. Would you mind try out with your example? Thanks in advance, great post.

REPLY



Petri 

September 5, 2012, 08:17

Hi Jose,

What method were you using when you tried to run the example application (command line, IDE) and were you running it after you had assembled the binary distribution? I will describe two alternative ways of running this example:

1) From command line without using the Maven assembly plugin.

Run the following command at the root directory of the project:

```
mvn exec:java -
Dexec.mainClass="net.petrikainulainen.mavenassemblyplugin.Hello
WorldApp"
```

See [3 ways to run Java main from Maven](#) for more details.

2) From command line after the binary distribution has been created.

First, you have to create the binary distribution. You can do this by running the following command at command prompt (again, run this at the root directory of the project):

```
mvn assembly:single
```

Second, you have to unzip the created zip package. You can find the zip package from the target directory. Look for a file called `maven-assembly-plugin-bin.zip`.

Third, you have to run the application. You can run the application by using the provided start up scripts (either `startup.bat` or `startup.sh`). Remember to run these scripts from the directory in which you unzipped the binary distribution. You can do this by running the following commands at command prompt:

```
cd target  
cd maven-assembly-plugin  
startup.bat (or startup.sh)
```

I will add this information to the actual blog entry as well (Thanks for pointing this out). If this did not solve your problem, please let me know.

REPLY



Nadia

September 6, 2012, 21:07

Thanks for putting this tutorial together, wish I have found it before, it was helpful.

[REPLY](#)**Petri**

September 6, 2012, 21:20

Hi Nadia,

It is great to hear that I could help you out.

[REPLY](#)**gibffe**

October 18, 2012, 17:05

sweet

[REPLY](#)**Ulrich Knaack**

December 7, 2012, 13:39

Hello,

I downloaded your example and tried it, but maven doesn't build a zip-File.
I'm using maven 3.0.4 , eclipse Juno SR1 and m2eclipse 0.12.1. I'm not using the embedded version of maven within m2eclipse but the stand-alone one.

Can you help me?

Regards, Ulrich

REPLY



Petri 

December 7, 2012, 13:57

Hi Ulrich,

I have got a few additional questions for you:

- Are you trying to use the Maven Assembly plugin directly from Eclipse?
- If you run the command `mvn assembly:single` at the command prompt, what happens? This should create file called `maven-assembly-plugin-bin.zip` to the target directory.

Also, please note that my example application creates the binary distribution only when the correct goal of the Maven Assembly plugin is executed. In real life scenario it might be better to create the distribution during the package lifecycle phase.

REPLY



David Lilljegen 

January 18, 2013, 00:53

Superb tutorial worked like a charm!

REPLY



Petri 



David,

January 18, 2013, 09:26

Thank you! It is great to hear that this blog entry was useful to you.

[REPLY](#)

jayasankar

February 5, 2013, 20:05

Great article; very helpful

[REPLY](#)

Petri

February 5, 2013, 20:17

Thank you. It is good to hear that I could help you out.

[REPLY](#)

Ramesh

February 6, 2013, 09:19

Great post :)

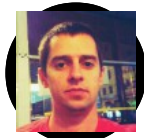
[REPLY](#)

**Petri**

February 7, 2013, 21:56

Thank you!

REPLY

**Danilo**

February 19, 2013, 22:24

Thanks a lot man, you saved my life.

Danilo – Brazil

REPLY

**Petri**

February 21, 2013, 18:35

You are welcome!

Thank you for leaving a comment. It is always nice to get feedback from my readers.

REPLY

**Noah Camp**

March 1, 2013, 21:28

Wow... THANK YOU! After days of searching and reading posts on StackOverflow, I was unable to come up with a suitable approach to a jar packaging requirement I have. This

approach is exactly what I needed since I'm dealing with RSA jars that throw security exceptions if unpacked/repackaged into an uber-jar.

As a newer person to maven, I found this article extremely helpful.

Again, thank you!

REPLY



Petri 

March 2, 2013, 21:30

You are welcome!

REPLY



Thiago Henrique 

March 16, 2013, 04:44

Great article! Very util, nice good!

REPLY



Petri 

March 16, 2013, 10:31

Thank you. I am happy to hear that you like it.

REPLY

**Girish** 

March 25, 2013, 13:53

Very useful. Thanks a lot.

REPLY

**Petri** 

April 16, 2013, 10:11

You are welcome!

REPLY

**Phantom1024** 

April 16, 2013, 01:17

Thank you for this article. I had been searching for days.

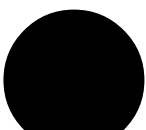
REPLY

**Petri** 

April 16, 2013, 10:12

You are welcome. I am happy hear that I could help you out. :)

REPLY

**Jirka** 



November 14, 2013, 14:52

Thanks for the article! It seems to be exactly what I'm looking for.

I have one problem – when I package your “assembly-plugin” example, there is created only maven-assembly-plugin.jar file but not you mentioned directory structure. I am a beginner with Maven, what am I doing wrong? Do need these directories exist before I do a package command?

REPLY



Petri 

November 14, 2013, 18:33

Hi,

You are not doing anything wrong.

The *package* goal doesn't create the binary distribution. It simply packages the project and creates the jar file. You can create the binary distribution by running the following command at command prompt:

```
mvn assembly:single
```

I will add this information to the README of the example application. Thanks for pointing this out.

REPLY



Jirka 

November 19, 2013, 14:50

Hi Petri,

one more thing. When I call `mvn assembly:single` plugin ZIP file is created but doesn't contain jar file, please see this screenshot –

http://s23.postimg.org/xpy2spqij/Screenshot_2013_11_19_13_47_34.png

Do I need to call another command to create a jar file? I have tried `mvn compile`, `mvn package` but no jar has been created. Thanks again.

REPLY



Petri



November 19, 2013, 18:27

Actually,

If you use the command `mvn assembly:single`, the *package* phase is not invoked. I didn't expect this. I tested this myself now (I used Maven 3.1.1) and it seems that you have two options:

1. Run the command `mvn package assembly:single` at command prompt.
2. Run the command `mvn assembly:assembly` at command prompt (This will invoke the *package* phase as well).

REPLY



Jirka



November 19, 2013, 20:16

Yes, “`mvn assembly:assembly`” works for me, thanks. You made my day!

**Jirka**

November 19, 2013, 15:05

Petri, back to my last comment – I forgot to call “mvn package” command, now It works like a charm! You can delete my comment, thanks.

REPLY

**Petri**

November 19, 2013, 18:31

If you don't mind, I will not delete it because I forgot that running the *mvn assembly:single* command doesn't invoke the *package* phase (see my answer to your previous comment).

REPLY

**Jirka**

November 19, 2013, 20:23

Maybe one suggestion to improve your code – is there some chance how to use `artifactId` or name for new jar file name? Your code works but generates “maven-assembly-plugin.jar” / “maven-assembly-plugin.zip”. It's not a problem, it works, but... :)

REPLY

**Petri**

November 19, 2013, 20:39

You can change the file names by using the `finalName` element. For example, if you change it to: `<finalName>foo</finalName>`, the names of the created files are:

foo.jar and *foo.zip*.

REPLY



google.com



December 24, 2013, 00:27

Your way of describing everything in this piece of writing is genuinely nice, all be capable of easily be aware of it, Thanks a lot.

REPLY



Bryan Karsh



January 1, 2014, 01:33

Hi Petri —

Thanks for the great blog. I have a question that is stumping me. I am using the assembly plugin with a bin distribution — it works great. My script makes use of a flexible shell script in /bin, config files in /etc, logs in /log, jars in /lib. — everything is great.

However, I have to add some new functionality that requires shading. I've never worked with the shade plugin. I am also not super excited to completely revamp what I have now with the assembly plugin set up.

My question is — any tips on shading a handful of dependency jars, but keep the rest of the assembly configuration in tact (including some external jars, my /etc config files, etc).

Any help much appreciated!

-b

REPLY

Petri 

January 1, 2014, 01:50

Hi Bryan,

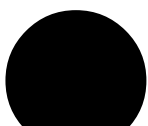
I haven't used to Maven Shade plugin either but it seems that it is possible to [include and exclude the artifacts which are added to the "Uber jar" created by the Shade plugin.](#) It is also possible to [include and exclude the artifacts added to the binary distribution.](#)

I would try to solve this problem by following these steps:

- Configure the Maven Shade plugin to include only the artifacts which must be shaded.
- Configure the Maven Assembly plugin to exclude the shaded artifacts.
- Configure the Maven Assembly plugin to add the "Uber jar" created by the Maven Shade plugin to the binary distribution instead of the "regular" jar.
- Ensure that the shading is done before the binary distribution is created.

I have no idea if this will work but based on the documentation of these two plugins, it should work. If you get it to work, it would be interesting to hear about it!

REPLY

Adil Fulara 



January 20, 2014, 13:21

Hi Petri,

Greetings !! As always your blog is very helpful.

Do you have any plans to make this work with multi-module project ?

Keep up the good work.

Adil.

REPLY



Petri 

January 20, 2014, 18:10

Thank you for your kind words. I really appreciate them.

About your question:

I can write a separate tutorial about using the Maven Assembly plugin in a multi-module projects. Do you have any special requirements which I should take into account?

REPLY



Adil Fulara 

January 22, 2014, 15:29

Hi Petri,

Perfect !! I have been trying it without success.

Regarding what i want is more of an open ended question.

I am trying to architect a template where i have

1. A module for API (interfaces) (JAR)
2. A module or two for Implementation (ex: DB solution, In memory solution) which uses the API module (JAR)
3. UI module to leverage API and Implementation for User Interaction
4. Module for Main Entry (Not sure if this is needed or not but this would be like a public static main function and any pre or post run configurations like schema migrations etc.)
5. Parent module to have the assembly xml to combine all the dependencies into lib and provide a startup script along with README and other documentation.

I tried to use the moduleset and tried to create a sample project. so far i have been unsuccessful. Partly due to my limited knowledge of advanced maven knowledge.

Let me know if the above seems like a wrong approach to creating a template for rapid app development.

Regards,
Adil Fulara.

PS: How do I subscribe to update from you on this page ? I didnt get any email regarding your follow up comment.

REPLY



Petri 

January 22, 2014, 23:06

Hi Adil,

I got a pretty good idea about your requirements. I will try to create a similar project structure and write a blog post about it.

At the moment it is not possible to get an email notification when someone answers to your comment. I will contact you by using your email address tomorrow.

REPLY



Adil Fulara 

January 23, 2014, 08:15

Awesome Petri..!!

I have made it a routine to check your blog regularly. Your articles are clear, precise and to the point. I always recommend my friends to check your tutorials too.

Feel free to share your project on github and let me know if i can assist. After all it's all a learning process and good to know where I made mistakes.

Regards,
Adil Fulara.



vince 

January 31, 2014, 00:03

This is very nice and clear. I want to go one step further though. I have the ProjectArtifact

with a name like `${artifactid}-${version}.jar`, but on the archive I want this to be just `${artifactid}.jar`. Any idea how to do this?

REPLY



Petri

February 1, 2014, 20:49

I haven't personally tried this but you could try to use the `outputFileNameMapping` element [as suggested here](#).

REPLY



kumar

February 10, 2014, 13:50

Hi Petri,
how can i run the wicket programs with out using maven and eclipse like IDEs.Please help.

REPLY



Petri

February 10, 2014, 20:48

Hi Kumar,

If you have the war file of the application, you can deploy it to a servlet container. If you have to create the war file, you have to use a some kind of build tool (Ant, Gradle, or Maven).

REPLY



Ivaylo 

March 19, 2014, 15:54

Many thanks!!!

REPLY



Petri 

March 19, 2014, 16:01

You are welcome!

REPLY



BergIT 

April 5, 2014, 08:20

Thank you for you explain, but you didn't add in the begin of tuto what artifactID we should choose when we create new Maven Project

REPLY



Petri 

April 6, 2014, 11:19

There are no universal rules for selecting the `artifactId` of a Maven project. Just

pick something which describes the project in question or follow the guidelines of your employer (if it is a work project).

REPLY



kris 

June 5, 2014, 01:53

very useful article and neat pom files. I hacked(?) it to generate zip file for multi module project. Wondering if you can extend this article to multi module setup (github code works for single module)

REPLY



Petri 

June 5, 2014, 18:08

Thank you for your comment. I added your blog post idea to my Trello board, and I will write the blog post in the future.

REPLY



Julian 

October 3, 2014, 14:20

Thank you for this nice tutorial. It was really helpful.

One question:

Why is it preferable to create a zip containing the libs and runnable scripts instead of

creating a simple large jar-file with all dependencies included and which can be executed simply by double-clicking?

Thank you

REPLY



Petri 

October 3, 2014, 21:46

Hi Julian,

If you are creating a binary distribution for an application, I think that you can use the approach that makes sense to you (the single jar approach might be a better choice).

On the other hand, if you are creating a binary distribution for a library or a framework, packaging its dependencies into a single jar doesn't make any sense because you must not decide how the users of your library (or framework) should manage their dependencies.

REPLY



Julian 

October 3, 2014, 22:52

Thank you very much for your answer. Now i get the point :)

REPLY



Petri 

October 4, 2014, 16:50

You are welcome!

REPLY



Tom 

October 8, 2014, 16:19

As others have already mentioned, awesome tutorial. Very thorough.

REPLY



Petri 

October 8, 2014, 21:04

Thank you for your kind words. I really appreciate them.

REPLY



Anirban 

November 9, 2014, 06:26

Nice one, helped me !:)

REPLY



Petri 

November 9, 2014, 20:34

Thanks! I am happy to hear that this blog post was useful to you.

REPLY



Prateej Gupta 

January 16, 2015, 00:12

This was immensely helpful.
Precise and to-the-point tutorial!
Thanks a lot!

REPLY



Petri 

January 16, 2015, 23:23

You are welcome! I am happy to hear that this tutorial was useful to you.

REPLY



Dizzi 

January 16, 2015, 01:13

Thank you. You've gave me at least few hours of sleep.

REPLY



Petri 

January 16, 2015, 23:24

You are welcome. I am happy to hear that this blog post helped you to get some sleep.

REPLY

Leave a Comment

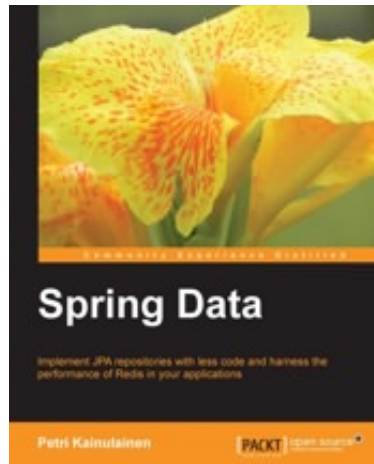
PREVIOUS POST: [WICKET HTTPS TUTORIAL PART THREE: CREATING A SECURE FORM
SUBMIT FROM A NON SECURE PAGE](#)

NEXT POST: [CREATING PROFILE SPECIFIC CONFIGURATION FILES WITH MAVEN](#)

CONNECT WITH ME



SPRING DATA BOOK



Implement JPA repositories with less code and harness the performance of Redis in your applications.

READ
MORE

TUTORIALS

[Getting Started with Gradle](#)

[Spring Data JPA Tutorial](#)

[Spring Data Solr Tutorial](#)

[Spring MVC Test Tutorial](#)

[Spring Social Tutorial](#)

[Using jOOQ with Spring](#)

[Writing Clean Tests](#)

[Writing Tests for Data Access Code](#)

SEARCH

FROM THE BLOG

[Recent](#)[Popular](#)[Favorites](#)[We Are Gonna Need It](#)[Getting Started with Gradle: Creating a Multi-Project Build](#)[2014 Annual Review](#)[My Favorite Blog Posts of 2014](#)[10 Most Popular Blog Posts of 2014](#)

CATEGORIES

[Programming](#) (104)[Apache Hadoop](#) (3)[Apache Wicket](#) (4)[Asciidoctor](#) (1)[Clean Code](#) (1)[Gradle](#) (5)[jOOQ](#) (4)[Maven](#) (7)[Solr](#) (9)[Spring Framework](#) (49)[Testing](#) (12)[Tips and Tricks](#) (16)[Unit Testing](#) (5)[Reviews](#) (3)[Software Development](#) (63)[Design](#) (10)[General](#) (7)[Learning](#) (21)[Processes](#) (20)[Technology Evaluation](#) (5)[Uncategorized](#) (6)[Writing](#) (1)



© 2010-Present Petri Kainulainen

All post images are property of their respective owners and may not be reused.

[Sitemap](#)