Title: Spring Boot add files to classpath from command line execution

Post Body:

I am using Netbeans 8.2 to develop Spring applications. This specific app with which I am having trouble is a Spring Boot 1.5.3 app. I have a spring xml file and an application.properties that I keep in /config under the root project directory.

I am passing the spring xml file to my project via the `@ImportResource` annotation and a value property like `@ImportResource(value='${config.xmlfile}')`.

When I click the 'Run Project' button in Netbeans my Spring app starts up and it correctly finds the application.properties file in my /config folder. However, any classpath references to other files in that folder are lost. For example, setting the config.xml file to `classpath:config/file.xml` or `classpath:file.xml` both fail to find the file but `file:config/file.xml` works.

Similarly, when running from the command line I have the following as my structure:

```
app/ |-- bin |   `-- app-run.sh |-- config |   |-- application.properties |   |-- log4j2.xml |   |-- file.xml `-- app-exec.jar
```

I am using the `spring-boot-maven-plugin` to make the jar as follows:

```
<plugin>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-maven-plugin</artifactId>      <version
```

and my app-run.sh script executes the following:

```
exec /bin/java -cp :.:../config/*:../app-exec.jar  -Dlogging.config=../config/log4j2.xml  -Dspring.config.location=../config/a
```

where /bin/java represents the location where I have java installed. The classpath set in -cp does not seem to be working here. Similarly to when running through the IDE, setting the config.xml file to classpath:config/file.xml or classpath:file.xml both fail to find the file but file:../config/file.xml works.

I would like to be able to set the classpath in both the IDE and from command line so that I can access files in Spring using classpath reference to make things easier. I do NOT want to put them all in `src/main/resources` and have them be packaged in the jar, as I need to edit these after packaging and deployment.

Does anybody have any ideas or helpful hints? Thanks in advance!

Accepted Answer:

Updated answer:

You can follow the practice in my original answer, but we recently dropped this for a simpler and cleaner option that is more standard (the 'Java' way). We made the change because we needed to dynamically load dependent libraries at runtime that were not available at compile time (in their exact version). In our case we wanted to load dependent jars only from separate folder(s) and not from an executable jar. We ended up having duplicate dependencies in the executable jars and in separate folder(s), so we decided to drop the executable jar Properties Launcher and instead only load dependencies from separate folders. This is often **NOT** the best option and should be evaluated for your use case. I prefer reading the standard Java classpath.

To run a Spring Boot app without an executable jar, we used Maven Assembly to put the dependent jars in a /libs directory and dropped the spring-boot-maven-plugin. The steps and some code for this are below:

1. Remove the spring-boot-maven-plugin that creates the executable jar in ZIP format

   Add the following to your assembly XML

   ```
   <dependencySets> <dependencySet> <outputDirectory>where you want the libs to go</outputDirectory>
   <useProjectArtifact>whether you want to include the project artifact here</useProjectArtifact> </dependencySet>
   </dependencySets>
   ```

   Run your code from the main class and include the dependent jar folder(s) on the classpath. Use the standard classpath notation on your OS and not the custom, awkward PropertiesLauncher loader path syntax

   ```
   java -cp <standard-classpath> <main-class>
   ```

An example of an actual call: `java -cp $CLASSPATH:./lib/*:./cfg/*:my-app.jar Application.class`

In this way you execute the Spring Boot app via standard java execution call, no custom Spring loading syntax. You just need to ensure that all of your dependencies are available on the classpath at runtime. We found this much easier to maintain and made this the standard for all of our apps.

Original answer:

After some researching, and thanks to @TuyenNguyen's helpful answer I was able to get the following working:

I added the following to my spring-boot-maven-plugin so that when I run from the command line it uses the `PropertiesLauncher` instead of the `JarLauncher`:

```
<configuration>      <mainClass>${mainClass}</mainClass>      <layout>ZIP</layout> //THIS IS THE IMPORTANT PART </configuratio
```

See here and here for more about the `PropertiesLauncher` options. It allows you to set the classpath, among other things. See here, here, and here for where I found the answer to this problem. Using format ZIP makes the `PropertiesLauncher` be used.

From there, I was able to use this command to launch the application as I intended:

```
java -Dloader.path=../config,../ -Dloader.config.location=classpath:application.properties -jar ../app-exec.jar
```

Another important note: when specifying the `-Dloader.path` make sure to use comma-separated values and only directories and files, as described here. Also, be sure to put the -D args before you specify -jar jar or they will not be set.

If anyone has any suggestions or edits to further improve this answer or the original question in order to help additional users, please let me know or make the edits yourself!

Highest Rated Answer:

If you don't put your files in `src/main/resources` then you can put it inside any folder that you want, BUT you must set your folder as a resources folder. Because `classpath` is always point to resources folder. Once you make your folder as a resource folder, it will be packaged into the jar. If you want to edit your resource file, just using 7 zip tool to open your jar -> edit files -> save -> it will update your change in the jar.

Another solution is create a folder, put all files you want to edit and not packaged in that, then set classpath manually to that folder every time you run, but the way you set above is not correct, try this solution for set classpath correct way.