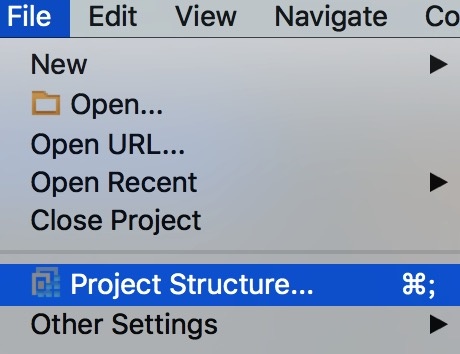
<https://www.cnblogs.com/qifengshi/p/6036870.html>

这里总结出用IDEA打包jar包的多种方式，以后的项目打包Jar包可以参考如下形式：

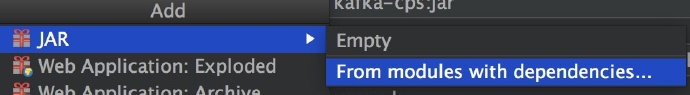
1. 用IDEA自带的打包形式
2. 用Maven插件maven-shade-plugin打包
3. 用Maven插件maven-assembly-plugin打包

## 用IDEA自带的打包方式：

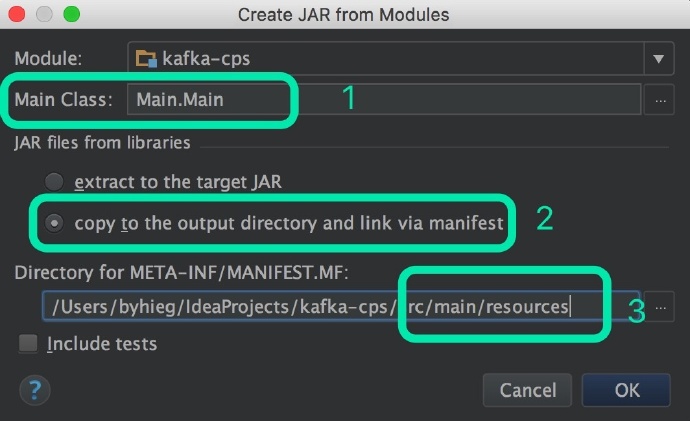
打开IDEA的file -> Project Structure，进入项目配置页面。如下图：



点击Artifacts，进入Artifacts配置页面，点击 + ，选择如下图的选项。

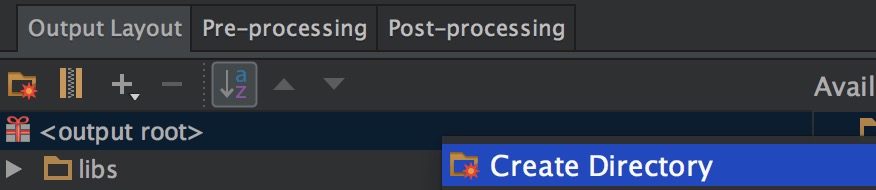


进入Create JAR from Modules页面，按照如下图配置。

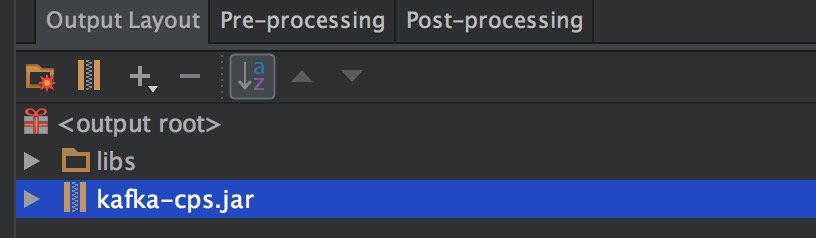


1. 第一步选择Main函数执行的类。
2. 第二步选择如图的选项，目的是对第三方Jar包打包时做额外的配置，如果不做额外的配置可不选这个选项（但不保证打包成功）
3. 第三步需要在src/main目录下，新建一个resources目录，将MANIFEST.MF文件保存在这里面，因为如果用默认缺省值的话，在IDEA12版本下会有bug。

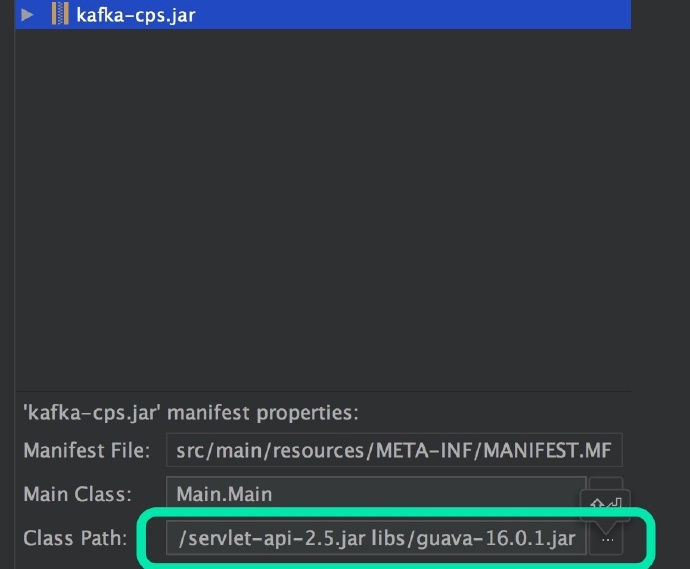
点击OK之后，出现如下图界面，右键点击<output root>，点击Create Directory,创建一个libs，将所有的第三方JAR放进libs目录下。



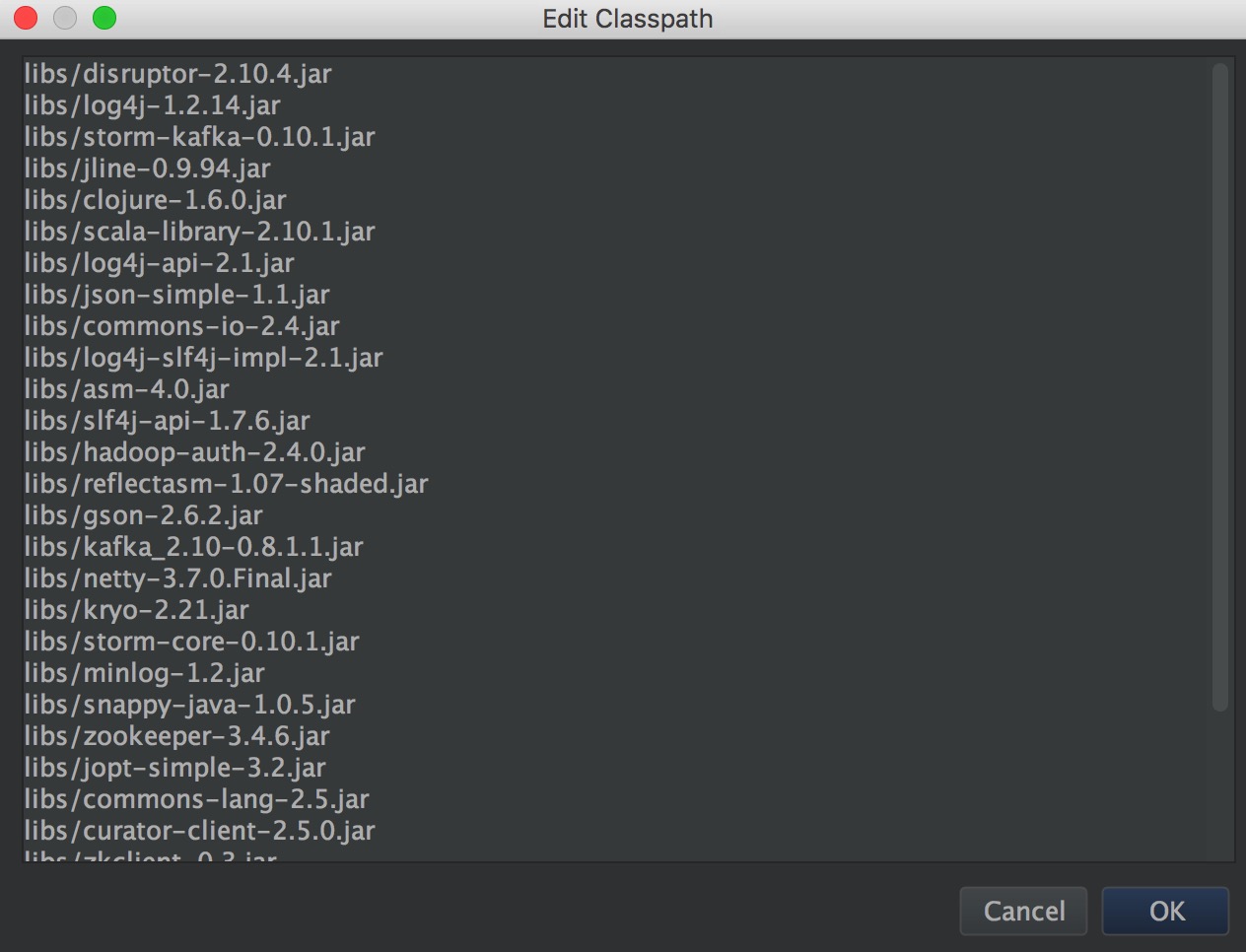
成功之后，如下图所示：

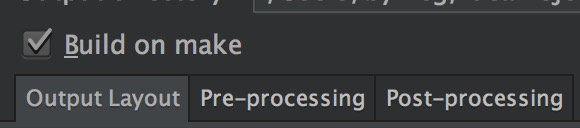


放入之后，点击我们要打成的jar的名字，这里面是kafka-cps.jar,选择classpath进行配置。

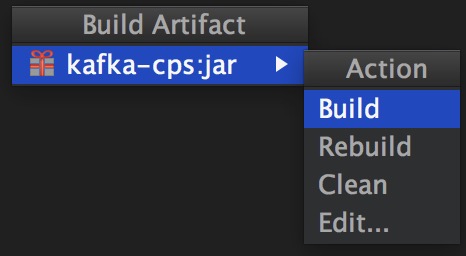


编辑的结果如下：

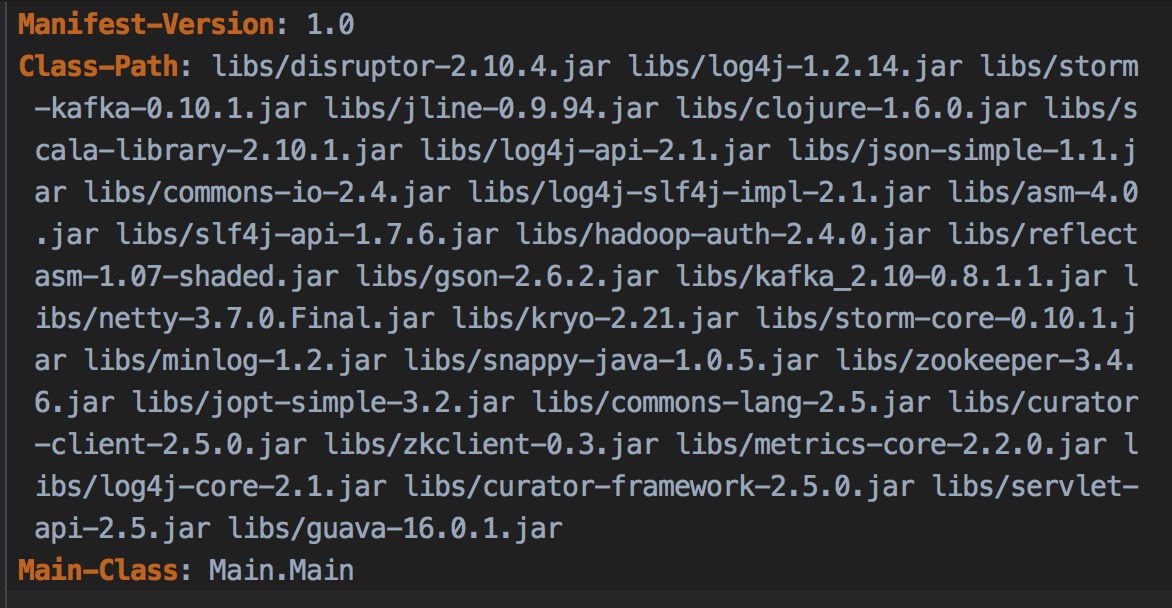


这里将所有的jar都写在libs/里面。点击OK，回到配置页面。  
同时还注意在配置页面，勾选build on make  


最后点击配置页面的OK，完成配置。回到IDEA,点击Build->Build Artifacts，选择build



就会生成我们需要的jar包。其位置在项目目录的out目录下/out/artifacts/kafka\_cps\_jar。  
下面放一个正确配置的清单文件内容



## 用maven-shade-plugin打包

上面的打包过程实在是过于的繁琐，而且也没有利用到maven管理项目的特色。为此，我们这里利用maven中的maven-shade-plugin插件。在pom.xml中，我们加入如下的信息来加入插件。

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-shade-plugin</artifactId>

<version>1.4</version>

<configuration>

<createDependencyReducedPom>true</createDependencyReducedPom>

</configuration>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>shade</goal>

</goals>

<configuration>

<transformers>

<transformer

implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">

<mainClass>Main.Main</mainClass>

</transformer>

</transformers>

</configuration>

</execution>

</executions>

</plugin>

这里面配置了一个｀configuration｀标签内容，在此标签下面 有一个transformer标签，用来配置Main函数的入口( <mainClass>Main.Main</mainClass>)，当然此标签内容很复杂，不是上面写的那么简单，上面之所以如此简单，是因为在所有类中(包括第三方Jar)只有一个Main方法。如果第三方jar中有Main方法，就要进行额外的配置，上面这么配置，不一定能执行成功。

具体可以参见[maven插件](http://blog.csdn.net/defonds/article/details/43233131)。

在加入这段代码到pom.xml之后，我们就可以用maven的命令去打包了。其指令如下：

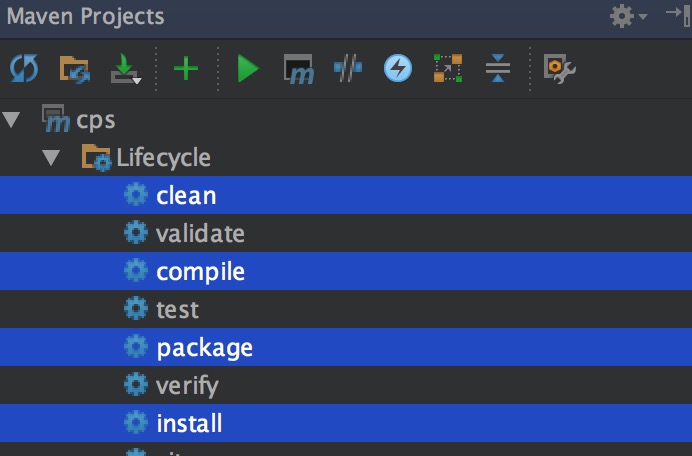
mvn clean compile //清除之前target编译文件并重新编译

mvn clean package //对项目进行打包(因为配置过插件，所以jar包是可执行的)

mvn clean install //安装项目，然后就可以使用了

然后通过java -jar cps-1.0-SNAPSHOT.jar运行。

如果使用IDEA的话，可以通过自带的maven管理工具代替执行上面的命令。如下图所示，依次点击蓝色的部分。



## 用maven-assembly-plugin打包

上面的方法，我们还需要点击很多命令去打包。这次利用一个新的插件，可以打包更简单。同样，在pom.xml中加入如下代码。上文的maven-shade-plugin插件代码可以删除。最好不要写2个插件代码。

<plugin>

<artifactId>maven-assembly-plugin</artifactId>

<version>2.4</version>

<configuration>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

<archive>

<manifest>

<mainClass>Main.Main</mainClass>

</manifest>

</archive>

</configuration>

<executions>

<execution>

<id>make-assembly</id>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

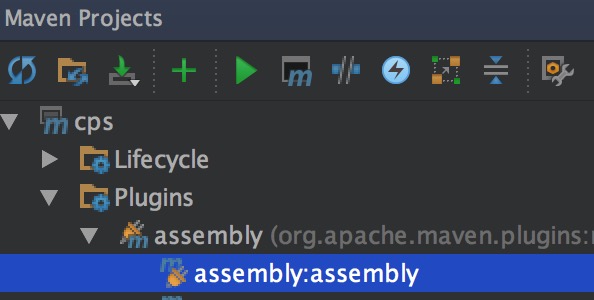
</executions>

</plugin>

这里同样配置了一个manifest标签来配置Main函数的入口。然后通过如下指令来实现打包。

mvn assembly:assembly

如果使用IDEA的话，可以通过自带的maven管理工具代替执行上面的命令。如下图所示，点击蓝色的部分。



然后通过执行java -jar cps-1.0-SNAPSHOT-jar-with-dependencies.jar运行。