

# CobaltStrike二次开发环境初探

原创 六号刃部 酒仙桥六号部队

2020-12-09原文

这是 酒仙桥六号部队 的第 124 篇文章。

全文共计2938个字，预计阅读时长9分钟。

在我们使用cobaltstrike的过程中，会涉及到二次开发，从而使其功能上更加的健壮，不至于碰到杀软就软了的地步，本文从cobaltstrike的快速反编译到二次开发环境的准备作为二次开发cobaltstrike的起步。

IntelliJ

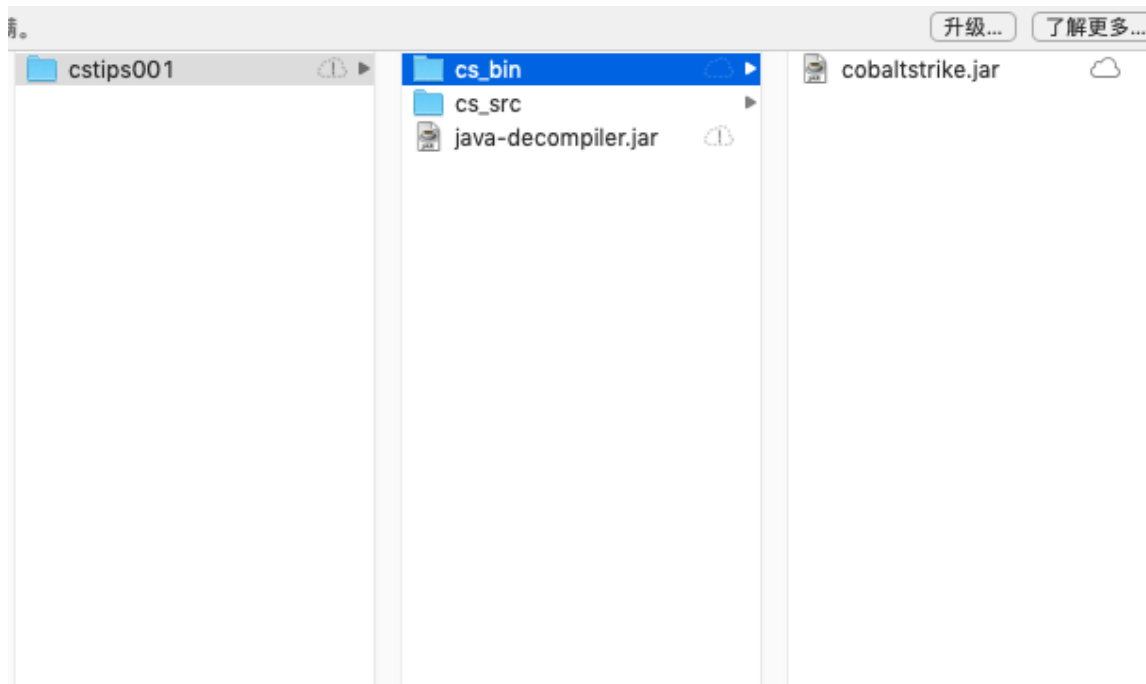
IDEA自带了一个反编译java的工具，有时候我们需要对cobaltstrike的整个jar包进行反编译，使用这个IntelliJ IDEA双击之类的反编译时要是对整个源码层面进行搜索并不是很方便，可使用其自带的反编译工具，可以做到批量的整个反编译。

## 一 CobaltStrike反编译

这里先在IntelliJ IDEA安装目录找到javadecompile.jar拷贝到一个准备好的目录，并且新建两个文件，一个cs\_bin里面放未反编译的cobaltstrike再建一个cs\_src文件，这个是空文件，是为了之后放反编译后的cobaltstrike

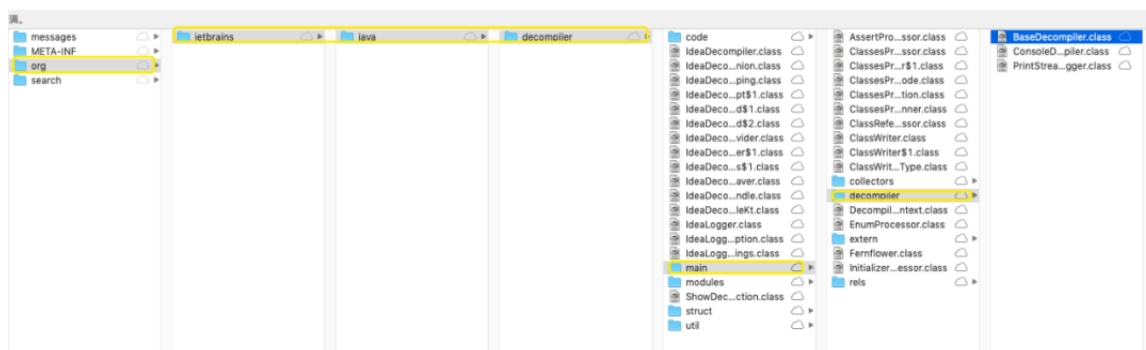
```
/Applications/IntelliJ IDEA.app/Contents/plugins/java-decompiler/lib/java-decompiler.jar //找到decompiler文件
```

```
cp java-decompiler.jar /Users/name/Desktop/wen/学习资料/java/cstips001/  
// 拷贝到准备好的目录
```



在 `java-decompiler` 中找到 `decompiler` 的路径，提取出来如下：

```
org/jetbrains/java/decompiler/main/decompiler/
```



把路径提取出来后，把反斜杠全部替换成 `.` 随之再其后加上 `ConsoleDecompilers`，如下就是提供反编译的这个类。



`org.jetbrains.java.decompiler.main.decompiler.ConsoleDecompile`

因为 `MANIFEST.MF` 中没有 `main class` 属性，没有指定主类，因此不能直接使用 `java -jar`，如果想要执行 `java` 包中具体的类，要使用 `java -cp` 输入如下命令：

```
java -cp java-decompiler.jar
```

```
org.jetbrains.java.decompiler.main.decompiler.ConsoleDecompiler
```

执行的时候会有提示。



让你加上 `-dgs=true` 后跟需要反编译的 `cobaltstrike` 和反编译之后要把 `cobaltstrike` 放入的目录，就是我们最开始建立的 `cs_src` 这个是存放反编译后的 `cobaltstrike`，运行这条命令即可对整个 `jar` 包开始反编译。

```
ava -cp java-decompiler.jar
```

```
org.jetbrains.java.decompiler.main.decompiler.ConsoleDecompiler
```

```
-dgs=true cs_bin/cobaltstrike.jar cs_src/
```

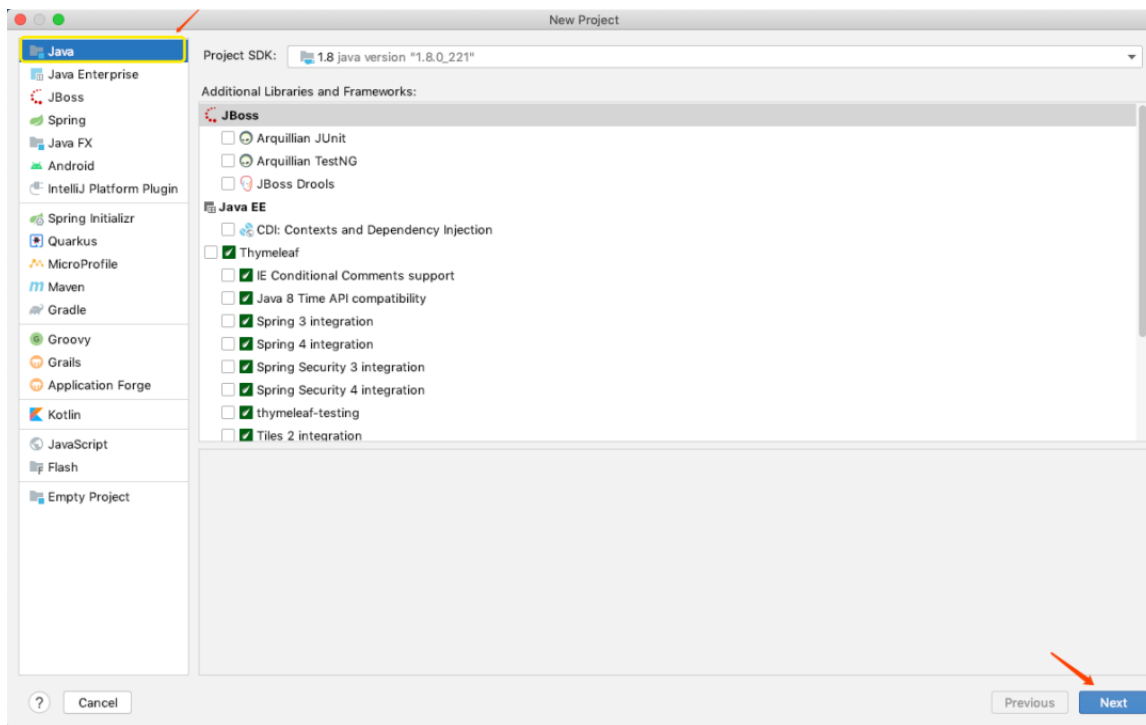
```
...@MacBook-Pro:~/Desktop/001-Exploit-Toolkits$ java -cp java-decompiler.jar org.javabins.java.decompiler.main.decompiler.ConsoleDecompiler -app=true -s_bin/cobaltstrike.jar -o...
INFO: Decompiling class aggressor/Aggressor
INFO: ... done
INFO: Decompiling class aggressor/AggressorClient
INFO: ... done
INFO: Decompiling class aggressor/bridges/AggressorBridge
INFO: ... done
INFO: Decompiling class aggressor/bridges/AggressorBridge
INFO: ... done
INFO: Decompiling class aggressor/bridges/Aliases
INFO: ... done
INFO: Decompiling class aggressor/bridges/AliasManager
INFO: ... done
INFO: Decompiling class aggressor/bridges/ArtifactBridge
INFO: ... done
INFO: Decompiling class aggressor/bridges/AttackBridge
INFO: ... done
INFO: Decompiling class aggressor/bridges/BeaconBridge
INFO: ... done
INFO: Decompiling class aggressor/bridges/BeaconTaskBridge
INFO: ... done
```

反编译后，会自动打包成 `jar` 包，右键解压后打开可以看到都是 `.java` 了，使用这个方法会非常方便，就不需要第三方工具，这个反编译出来的就可以直接放入 `IntelliJ IDEA` 中，可直接实现代码搜索，相关的交叉引用。

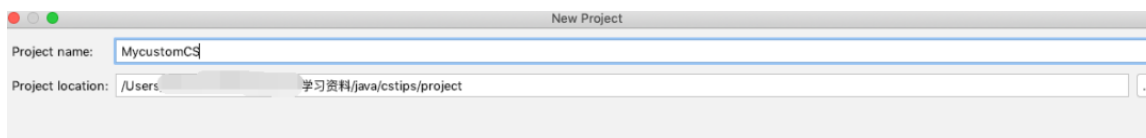
▶ aggressor	689.8 KB
▼ beacon	294.4 KB
BeaconC2.java	35.1 KB
BeaconCharsets.java	1.8 KB
BeaconCommands.java	3.0 KB
BeaconConstants.java	3.0 KB
BeaconData.java	3.7 KB
BeaconDNS.java	8.2 KB
BeaconDownloads.java	5.0 KB
BeaconElevators.java	1.4 KB
BeaconErrors.java	5.9 KB
BeaconExploits.java	1.4 KB
BeaconHTTP.java	4.2 KB
BeaconParts.java	1.6 KB
BeaconPayload.java	14.5 KB
BeaconPipes.java	1.6 KB
BeaconPivot.java	1.4 KB
BeaconRemoteExecMethods.java	1.5 KB
BeaconRemoteExploits.java	1.8 KB
BeaconSetup.java	4.1 KB
BeaconSocks.java	6.9 KB
BeaconTabCompletion.java	16.2 KB
▶ c2setup	5.1 KB
CheckinListener.java	686 B
CommandBuilder.java	3.0 KB
▶ dns	10.5 KB
▶ elevators	1.5 KB
EncodedCommandBuilder.java	1.5 KB

## 二 CobaltStrike二次开发环境

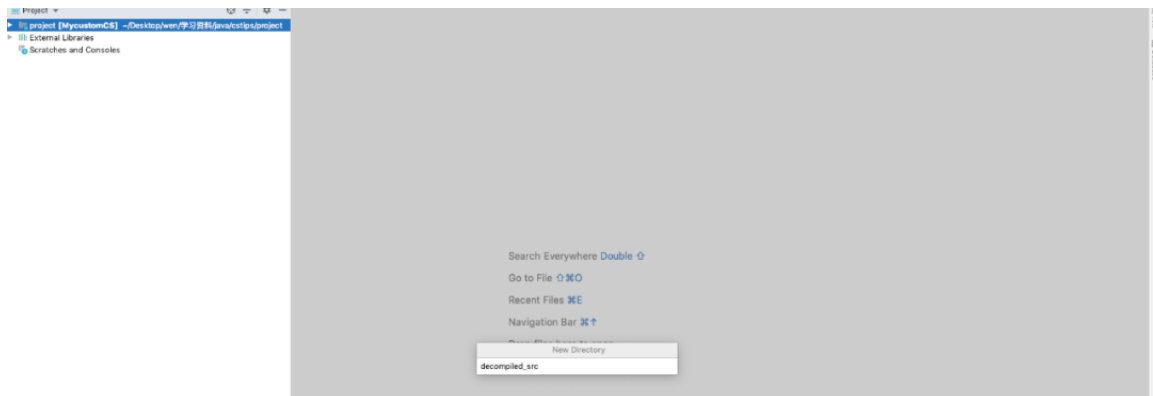
打 开 IntelliJ IDEA 选 择 Create New Project 一直选择Next。



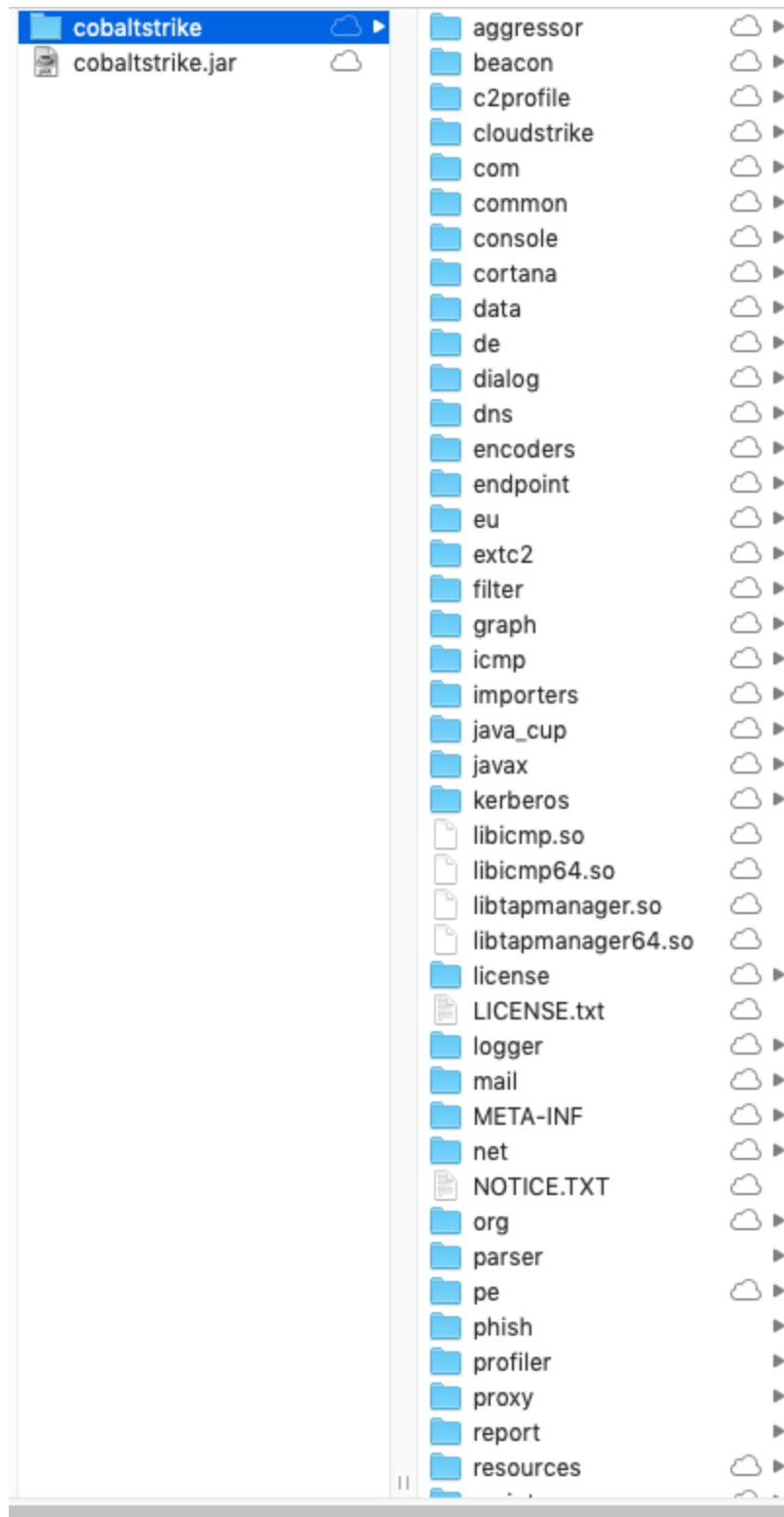
这里选择路径跟起个名。



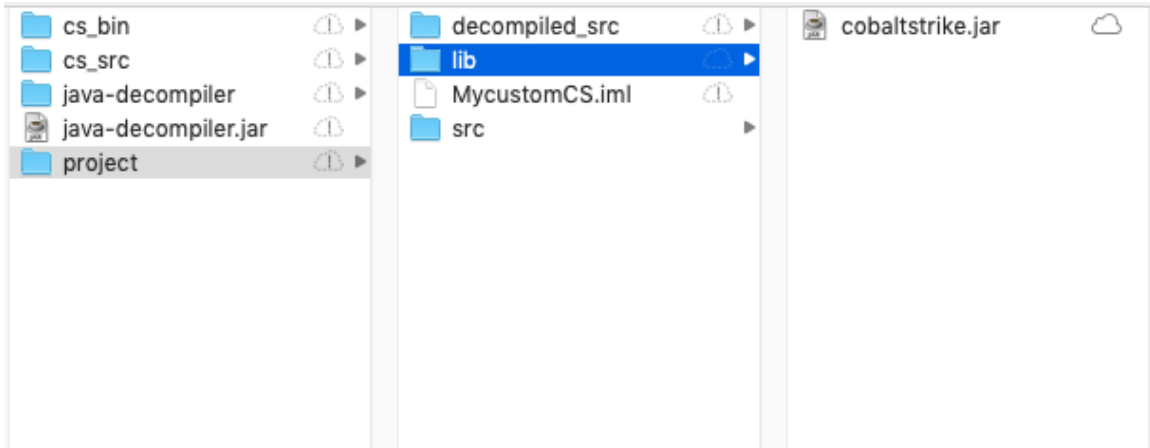
创建好后需要先建立两个文件夹，右击选择 New Directory 建立一个 decompiled\_src 文件夹，之后再建立一个 lib 文件夹。



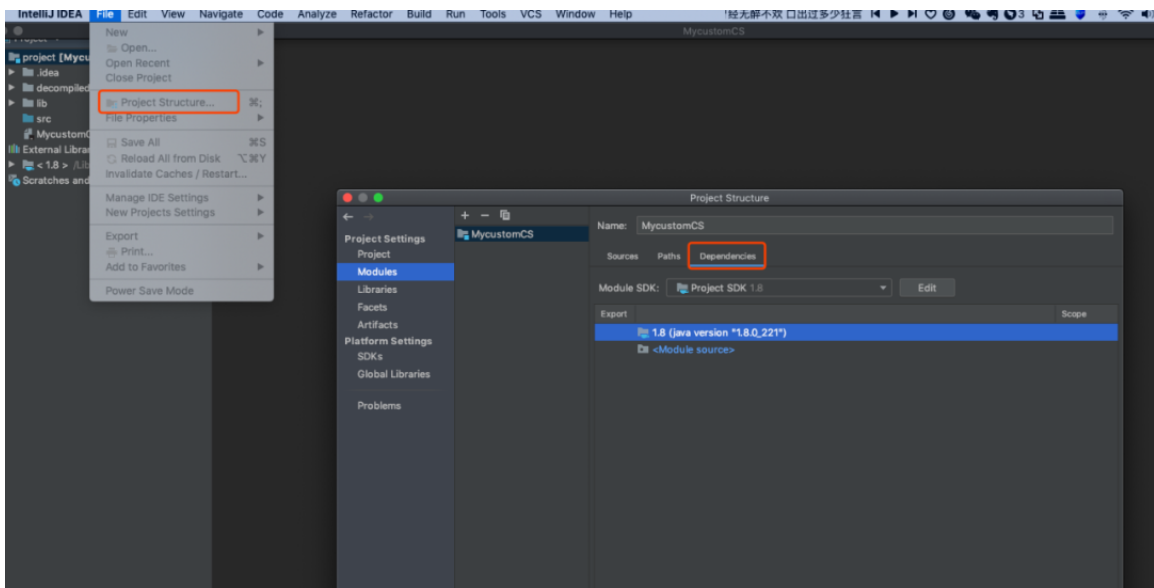
把在 `CS_Tips001` 中反编译好的 `CobaltStrike` 复制到 `decompiled_src` 中，然后把它解压出来，可看到一个完整的反编译后的目录。



随后把原始的未编译的CobaltStrike放到刚刚新建的lib中去。

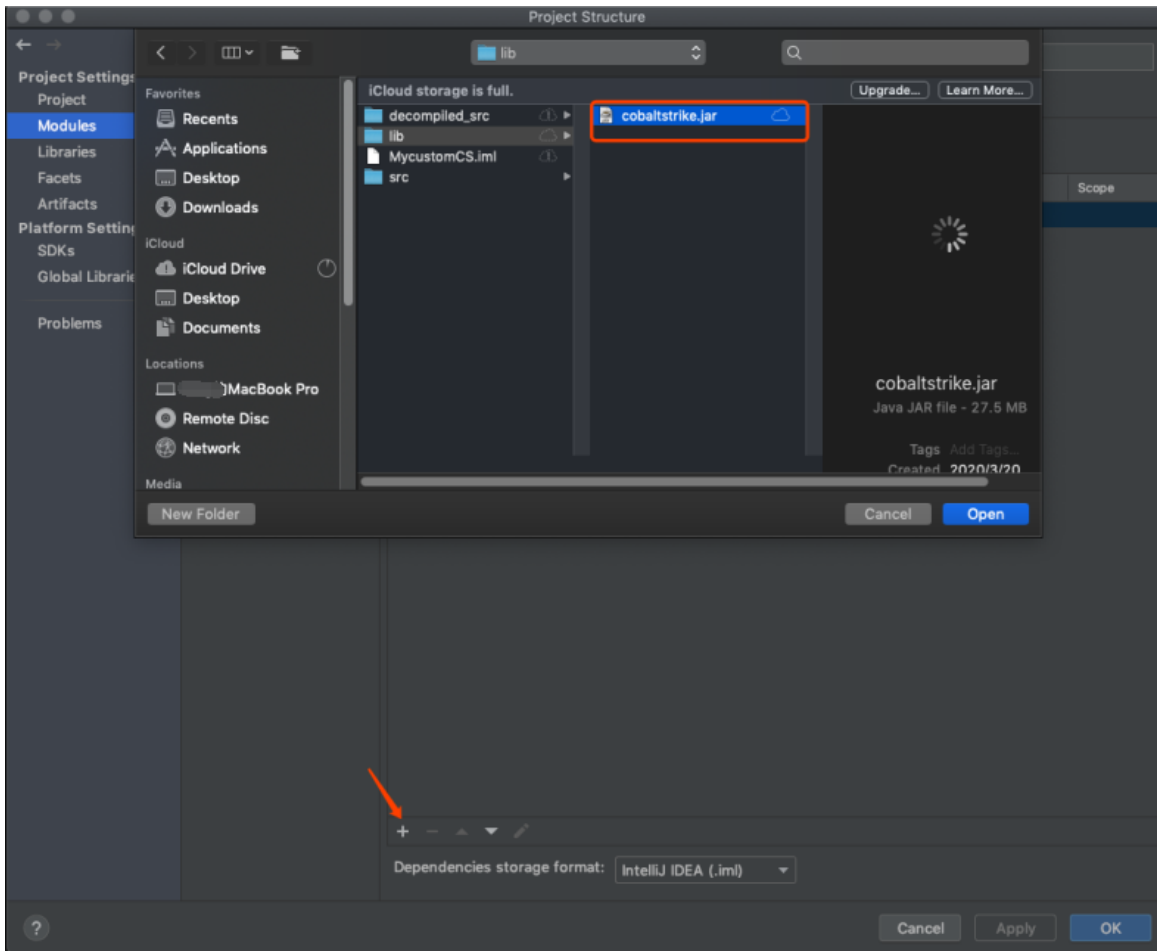


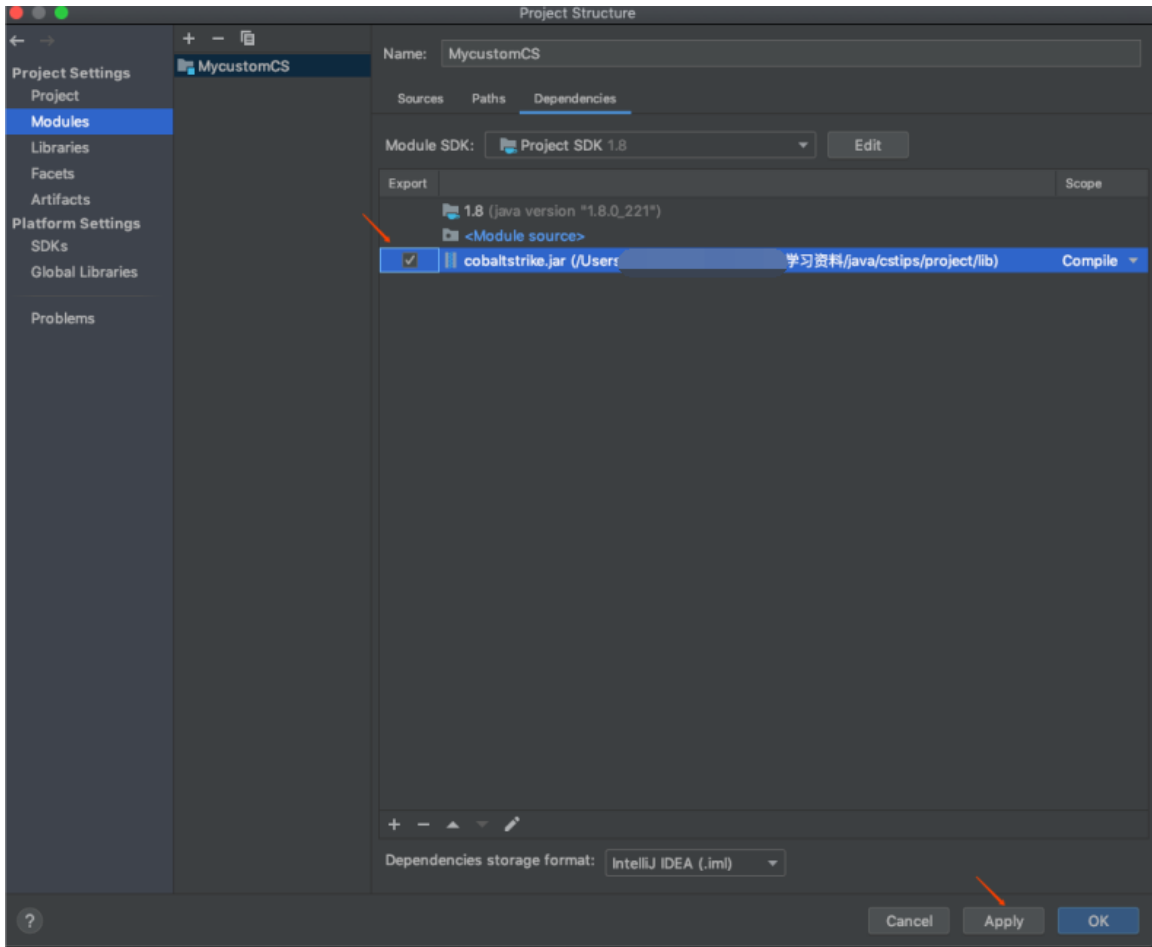
接下来需要对这个项目进行设置，点击 `File` 中的 `Project Structure` 在 `Modules` 对 `Dependencies` 进行设置。



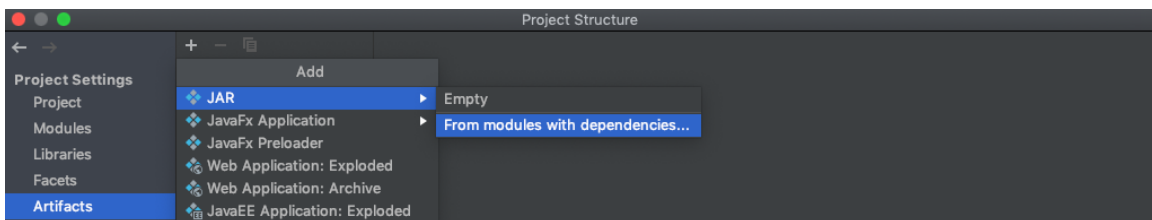
选择 `lib` 中的 `cobaltstrike.jar`，确认是 `Compile` 之后勾选一下，然后选择 `Apply`。



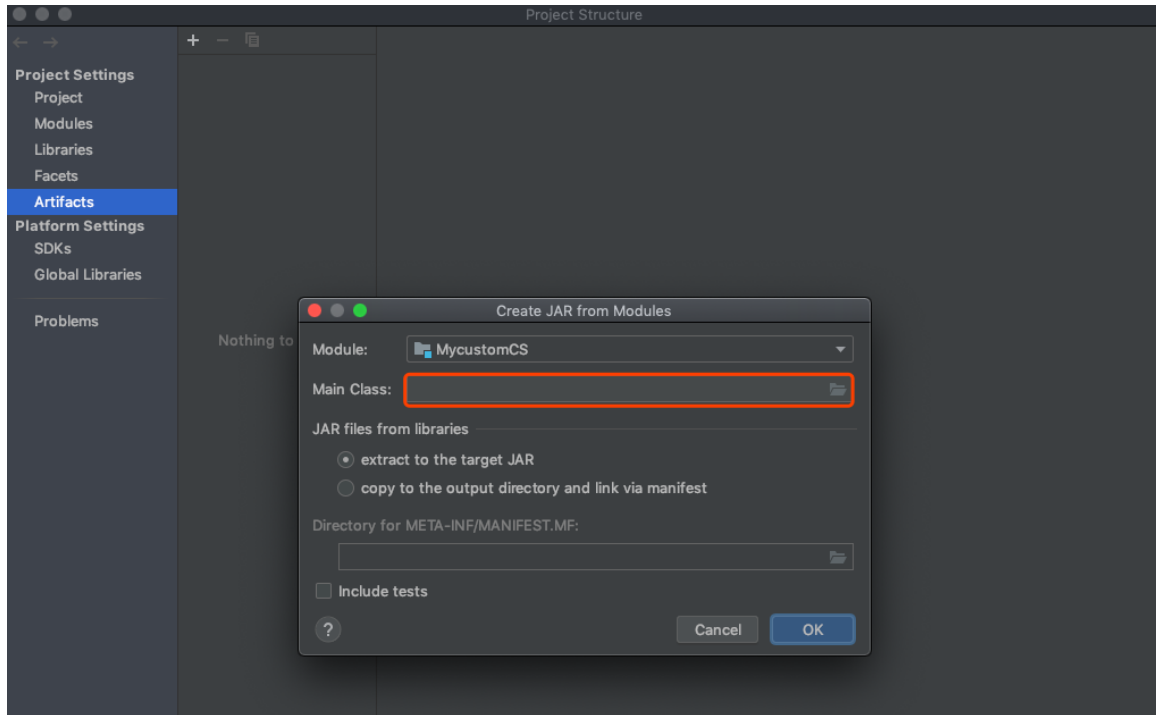




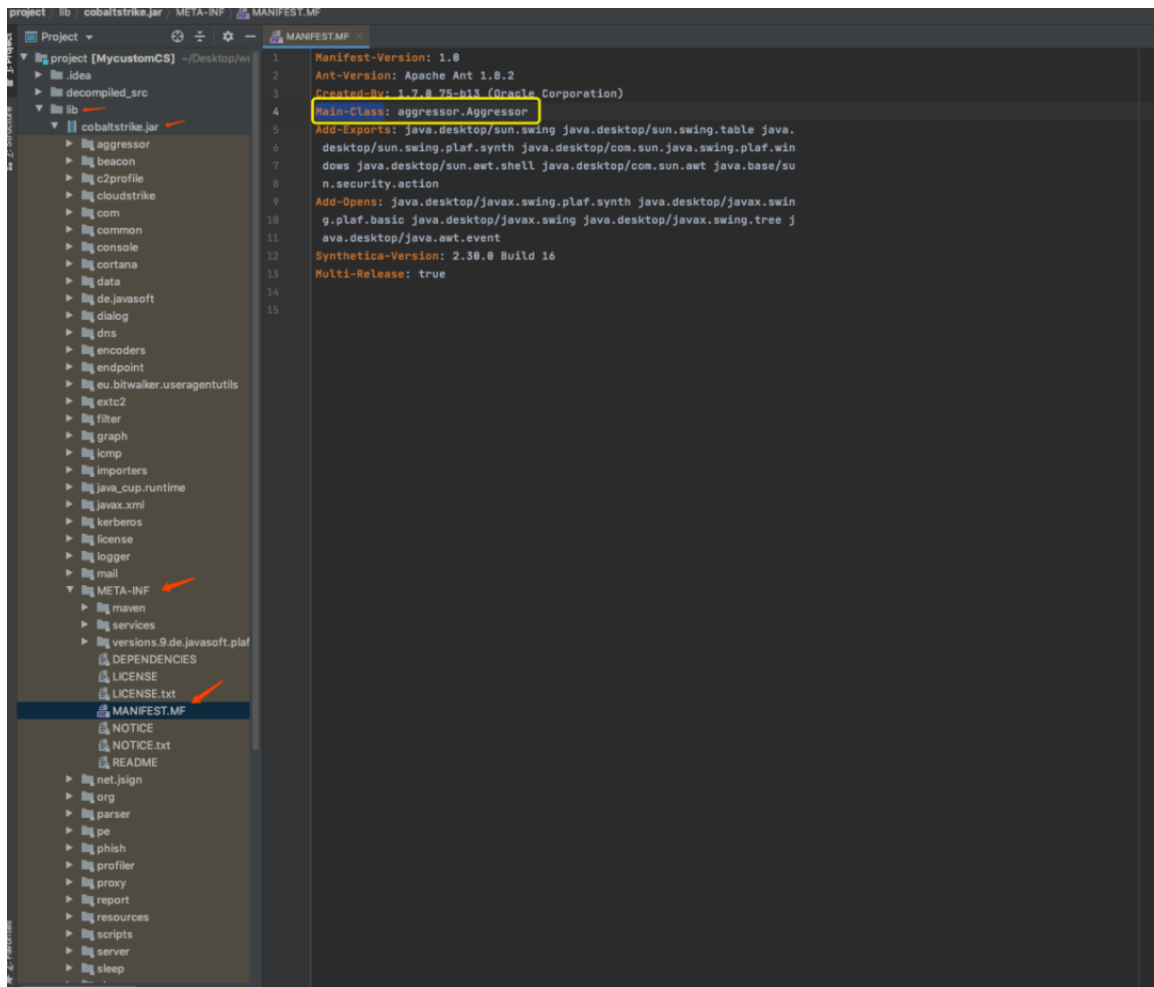
至此依赖关系设置完了，现在进入 `Artifacts-->JAR-->From modules with dependencies`



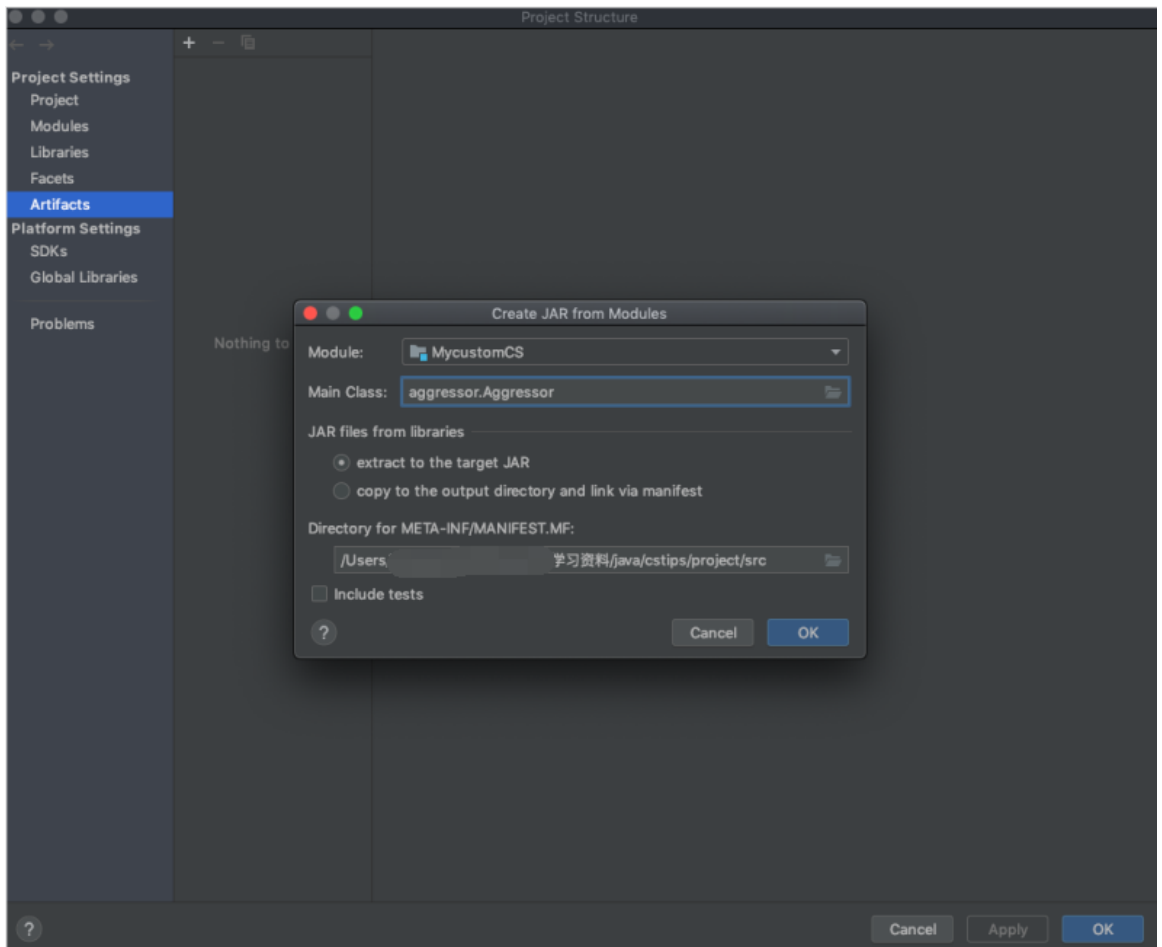
这里需要一个填写一个 `Main Class`



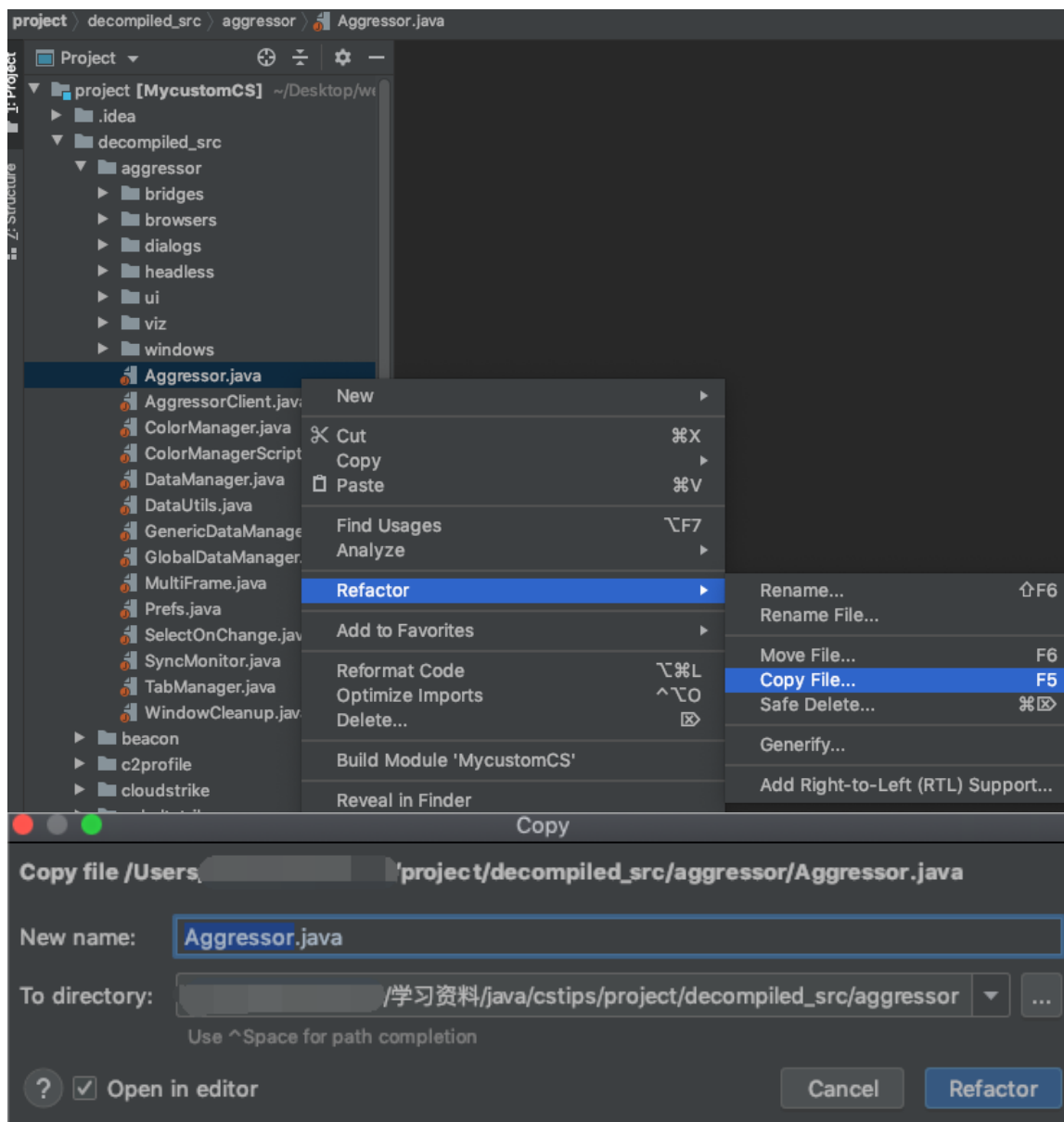
目前我们还不知道这个 Main Class 该填什么，可以点在 lib 中的 META-INF 里双击 MANIFEST.MF，我们可以看到 Main Class，复制 `aggressor.Aggressor`



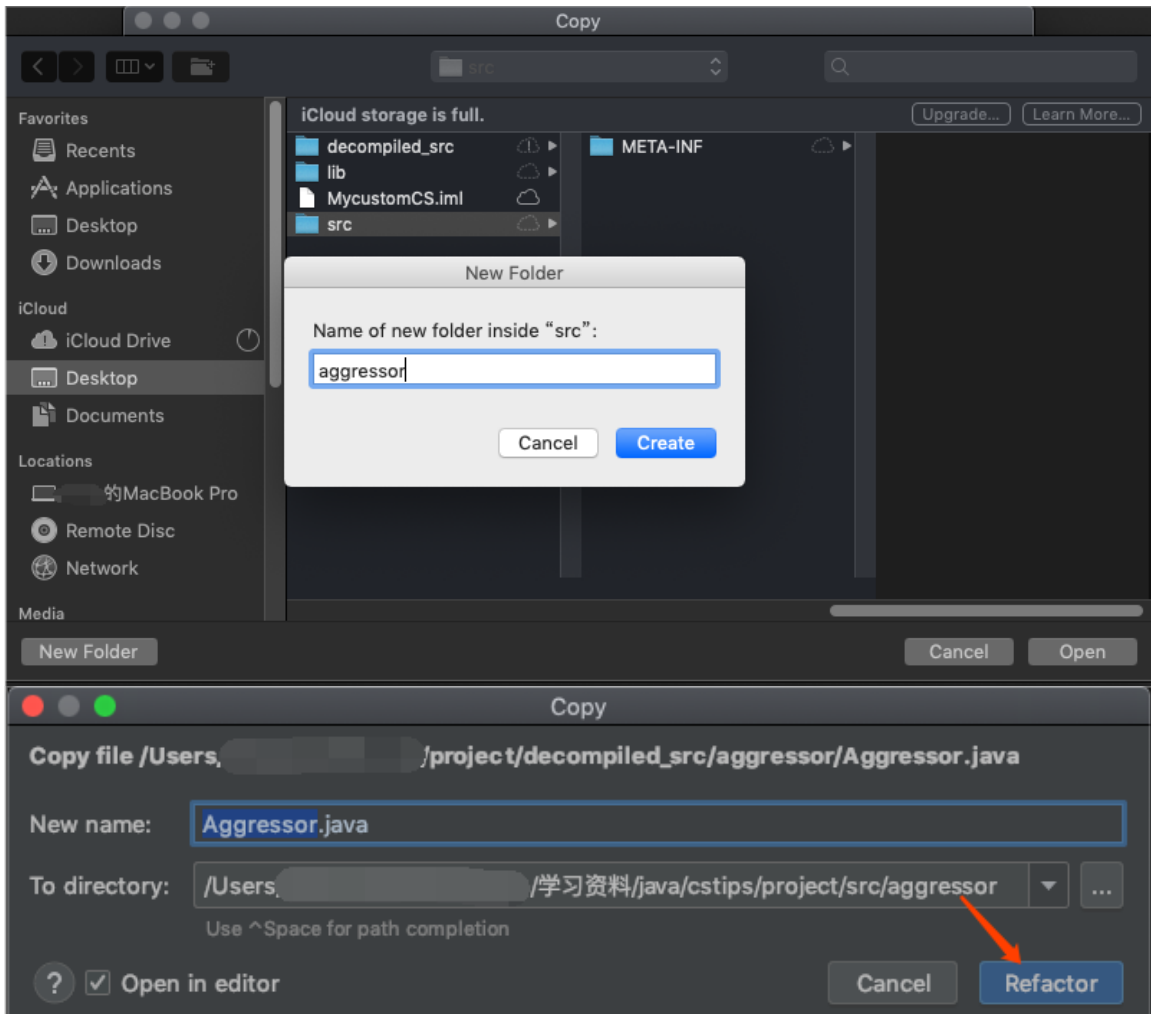
再次打开 Artifacts-->JAR-->From modules with dependencies 在 Main class 处填入 `aggressor.Aggressor` 选择OK, 这里就设置完成了。



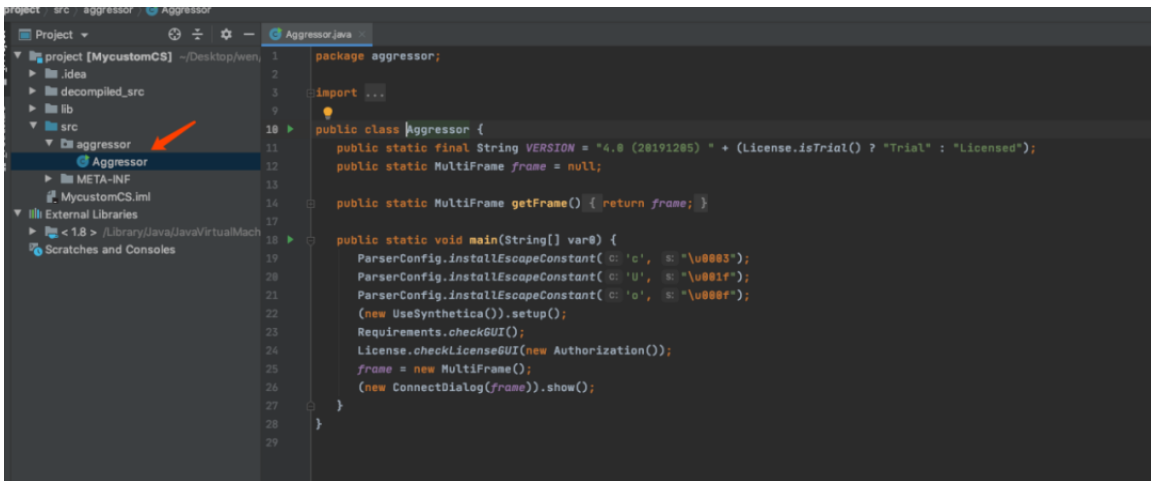
接下来在 `decompiled_src` 目录中找到已经反编译完的 `aggressor` 主类, 右击选择 `Refactor --Copy File`



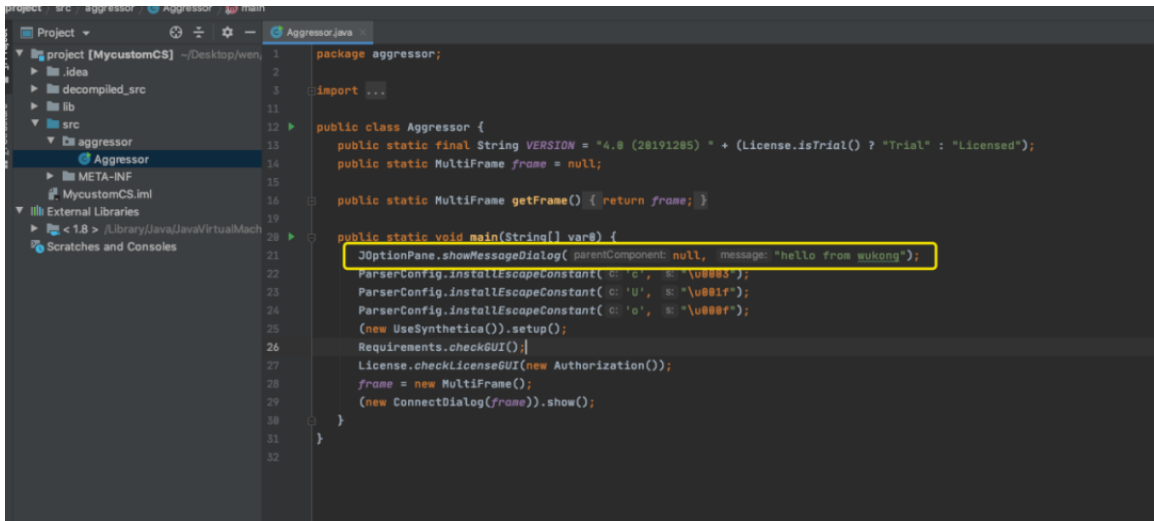
在 `directory` 点击添加，选择之前创建的 `src` 在其中添加一个 `aggressor` 名字要一致，最后点击 `Refactor`



这样 `aggressor` 就自动的被拷贝到 `src` 目录里去了，这里可以看一下，如图。



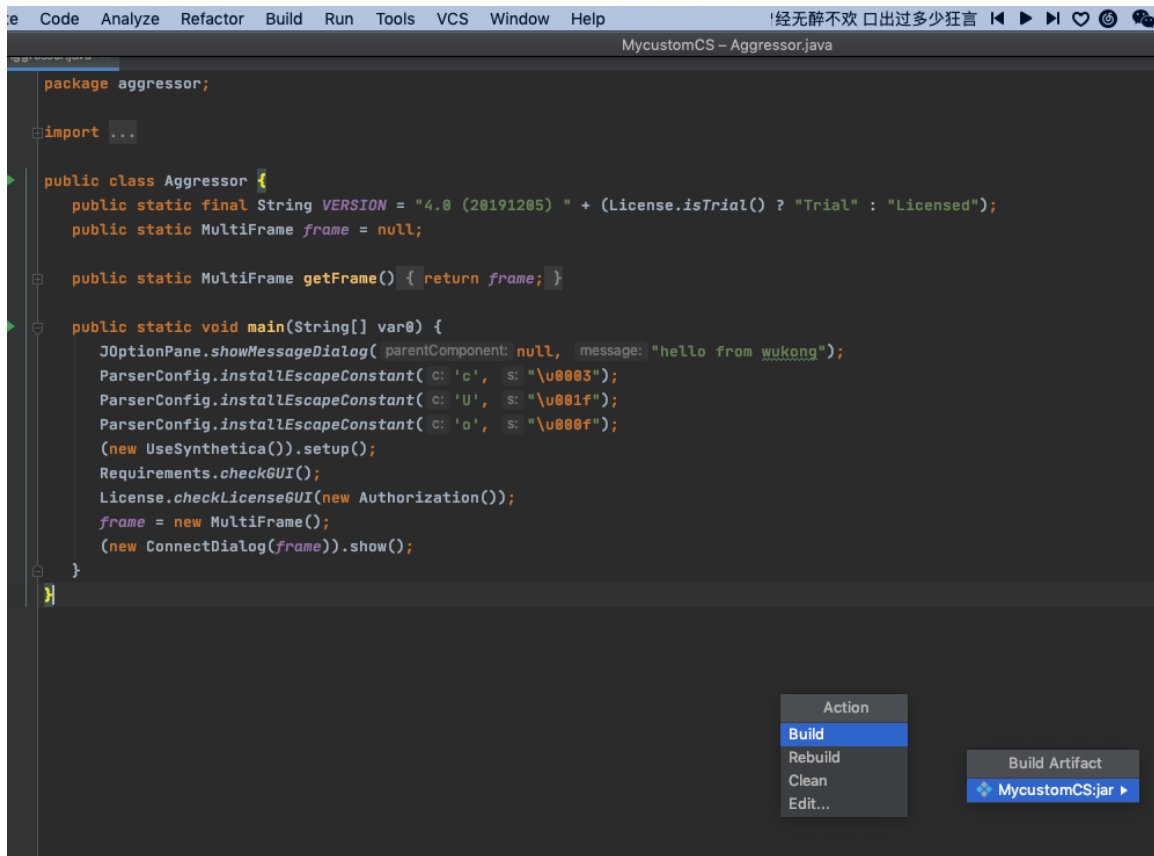
测试一下，修改文件，保存。



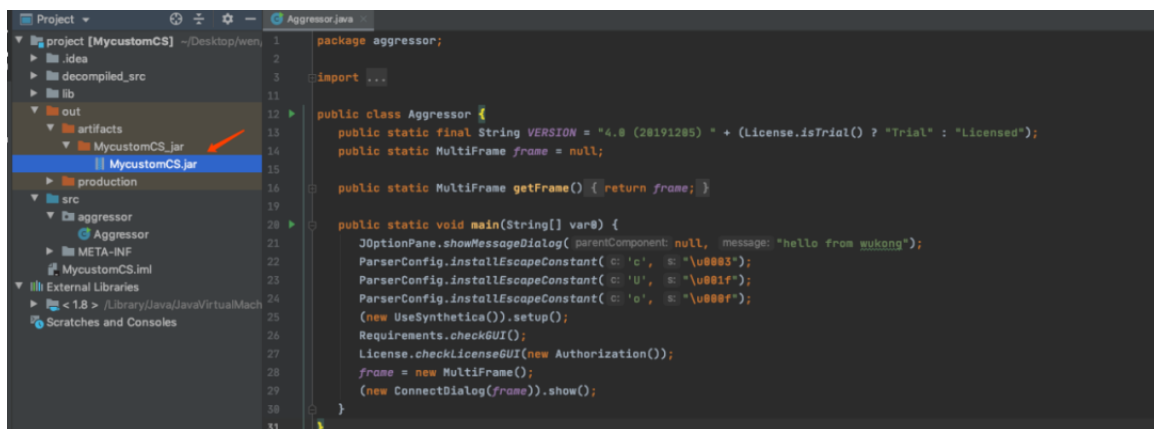
```
1 package aggressor;
2
3 import ...
4
5
6
7
8
9
10
11
12 public class Aggressor {
13     public static final String VERSION = "4.0 (20191205) * + (License.isTrial() ? "Trial" : "Licensed");
14     public static MultiFrame frame = null;
15
16     public static MultiFrame getFrame() { return frame; }
17
18
19
20
21     public static void main(String[] var0) {
22         JOptionPane.showMessageDialog( parentComponent: null, message: "hello from wukong");
23         ParserConfig.installEscapeConstant( @'G', @'\u0047');
24         ParserConfig.installEscapeConstant( @'U', @'\u0055');
25         (new UseSynthetica()).setup();
26         Requirements.checkGUI();
27         License.checkLicenseGUI(new Authorization());
28         frame = new MultiFrame();
29         (new ConnectDialog(frame)).show();
30     }
31 }
32
```

到这里我们的整个准备工作就完成了，之后就是我们要修改哪个文件，就可以在完整的源码中找到那个文件，然后右键Refactor然后Copy File到这个目录然后进行修改，修改完成之后就可以选Build-->Build Artifacts -->Build进行编译。

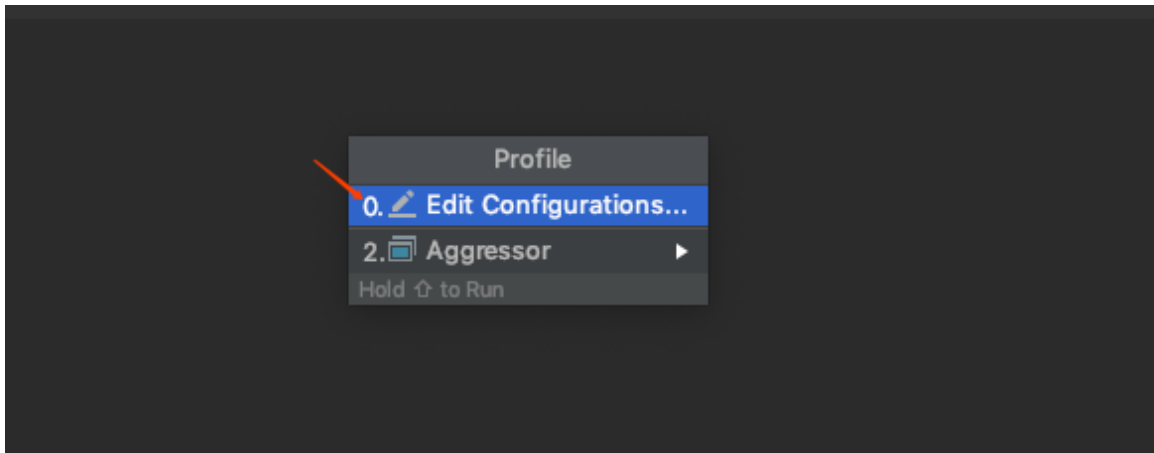




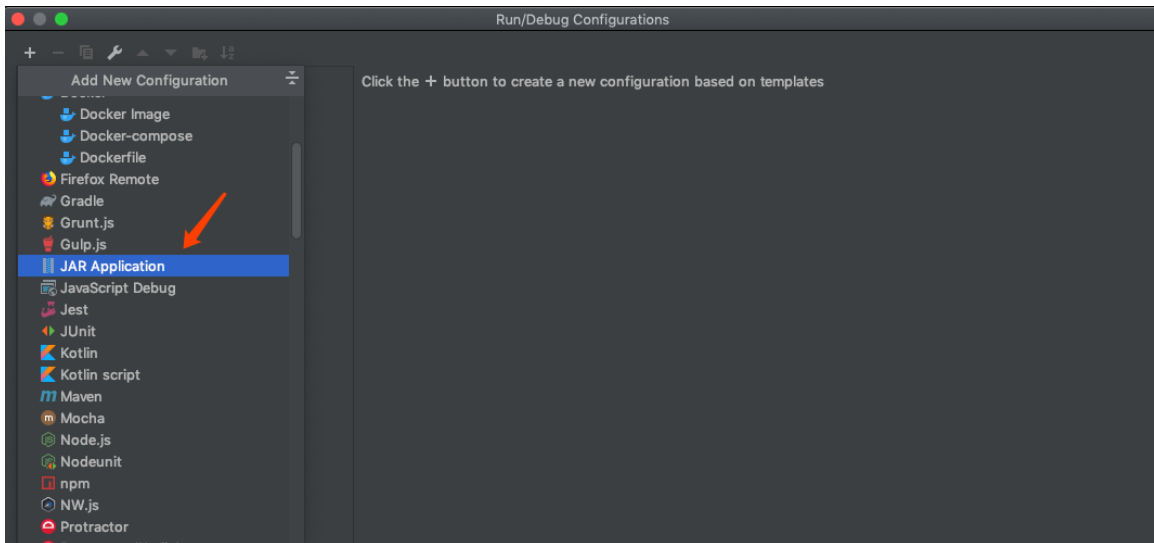
当提示 Build completed successfully in 4 s 227 ms (a minute ago) 的时候，会生成一个 out 文件夹，其中可看我们的编译好的 MycustomCS.jar



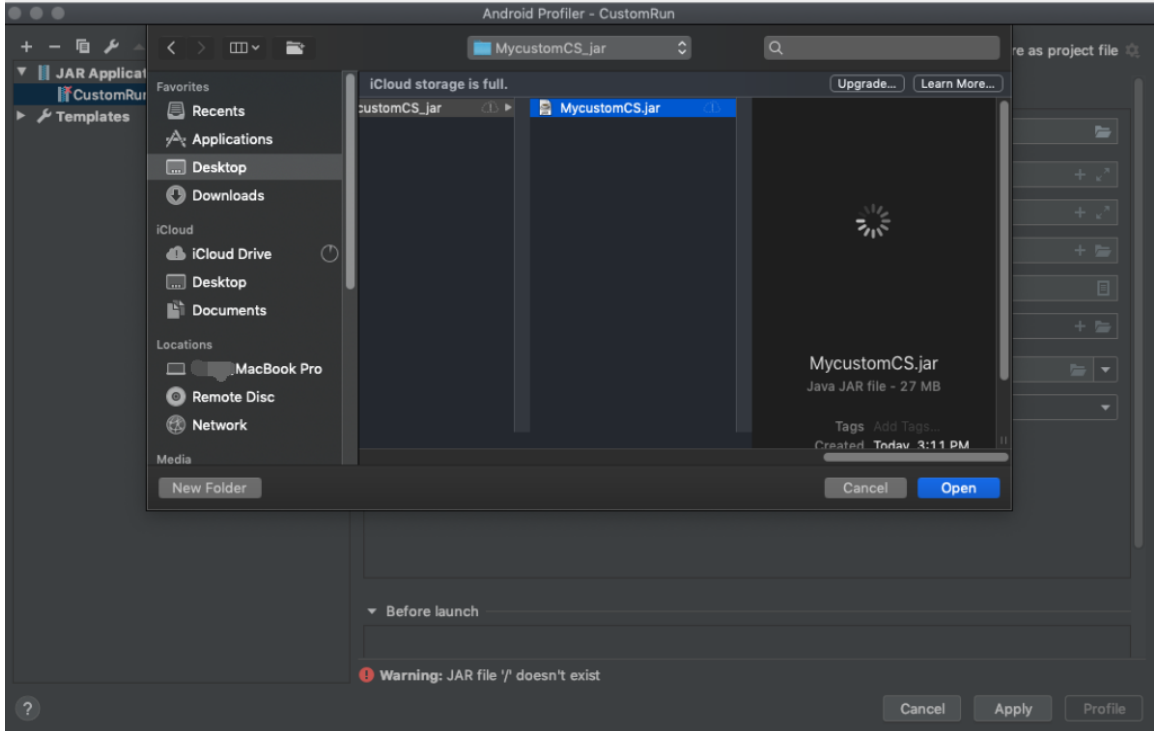
在每次调试运行的时候，不需要切换到命令行环境，可以直接选择 Run 中的 Profile 设置参数。



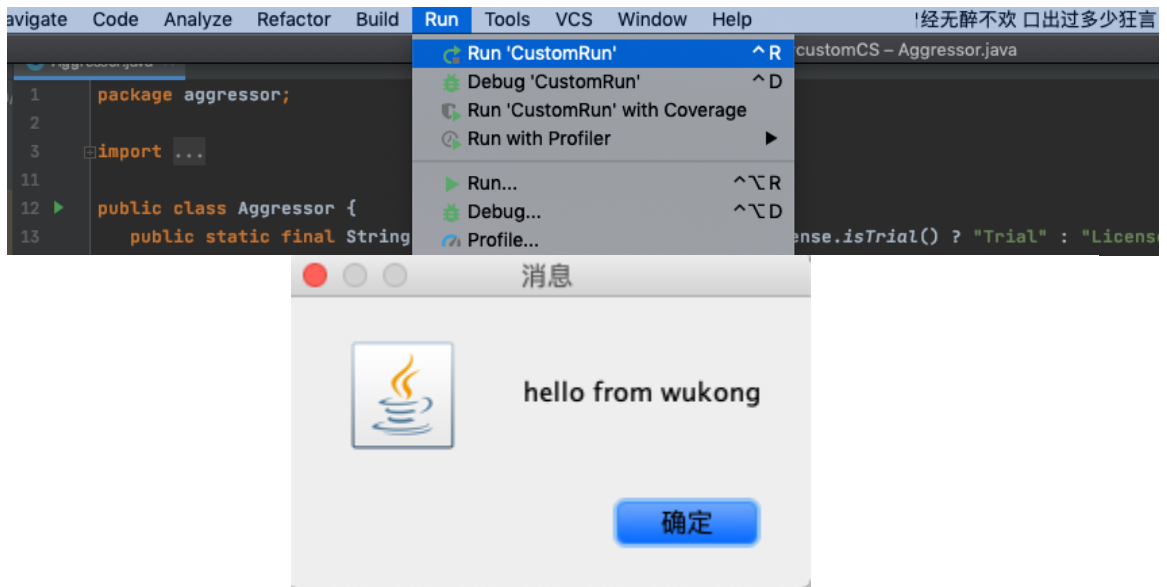
选择 **+** 号，在 `JAR Application` 添加一个配置文件。



在 `Path` 到 `JAR` 中选择 `out` 文件中我们修改并编译好的 jar 包，选择好后点击 `Apply`



最后在Run中选择Run CustomRun即可看到消息窗。



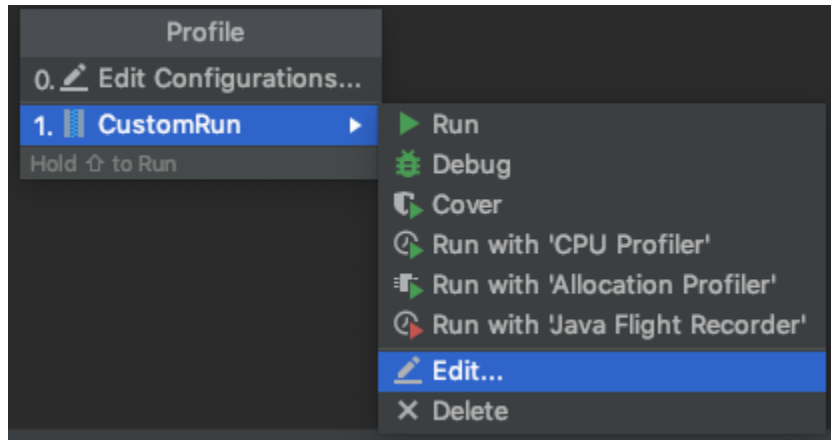
点击确定，发现弹出提示，没关系，继续点击确定。

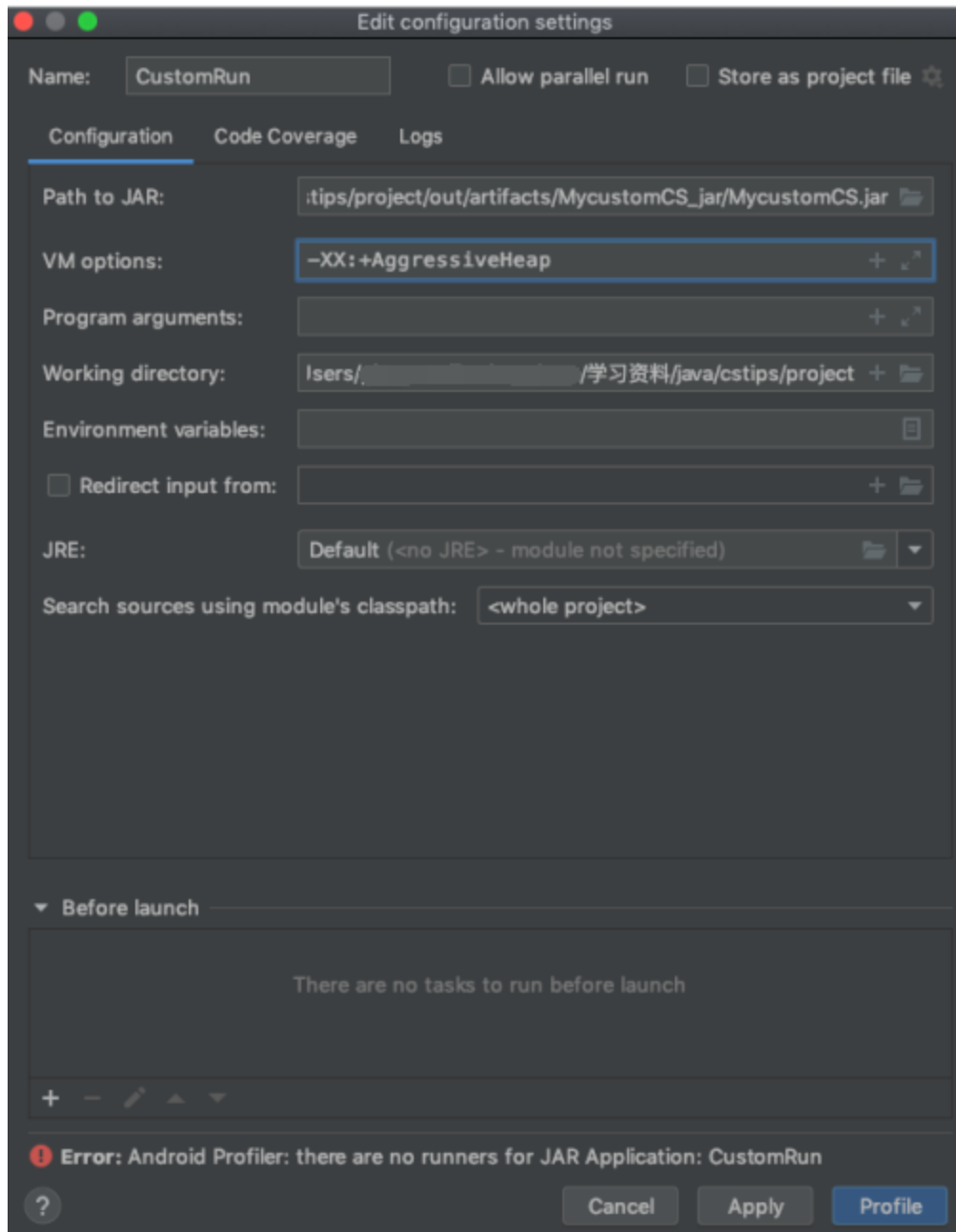


我们把 `-XX:+AggressiveHeap` 复制下来。

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/java -Dfile.encoding=UTF-8 -jar /Users/.../.../java/custom/project/out/artifacts/MyCustom3.jar/MyCustom3.jar  
[-] Java: -XX:+AggressiveHeap option not set. Use the Cobalt Strike Launcher. Don't click the .jar file!  
Process finished with exit code 0
```

再放到 Run 的 Profile 这里就直接选择之前创建的 CustomRun 填入 VM options 最后选择 Apply

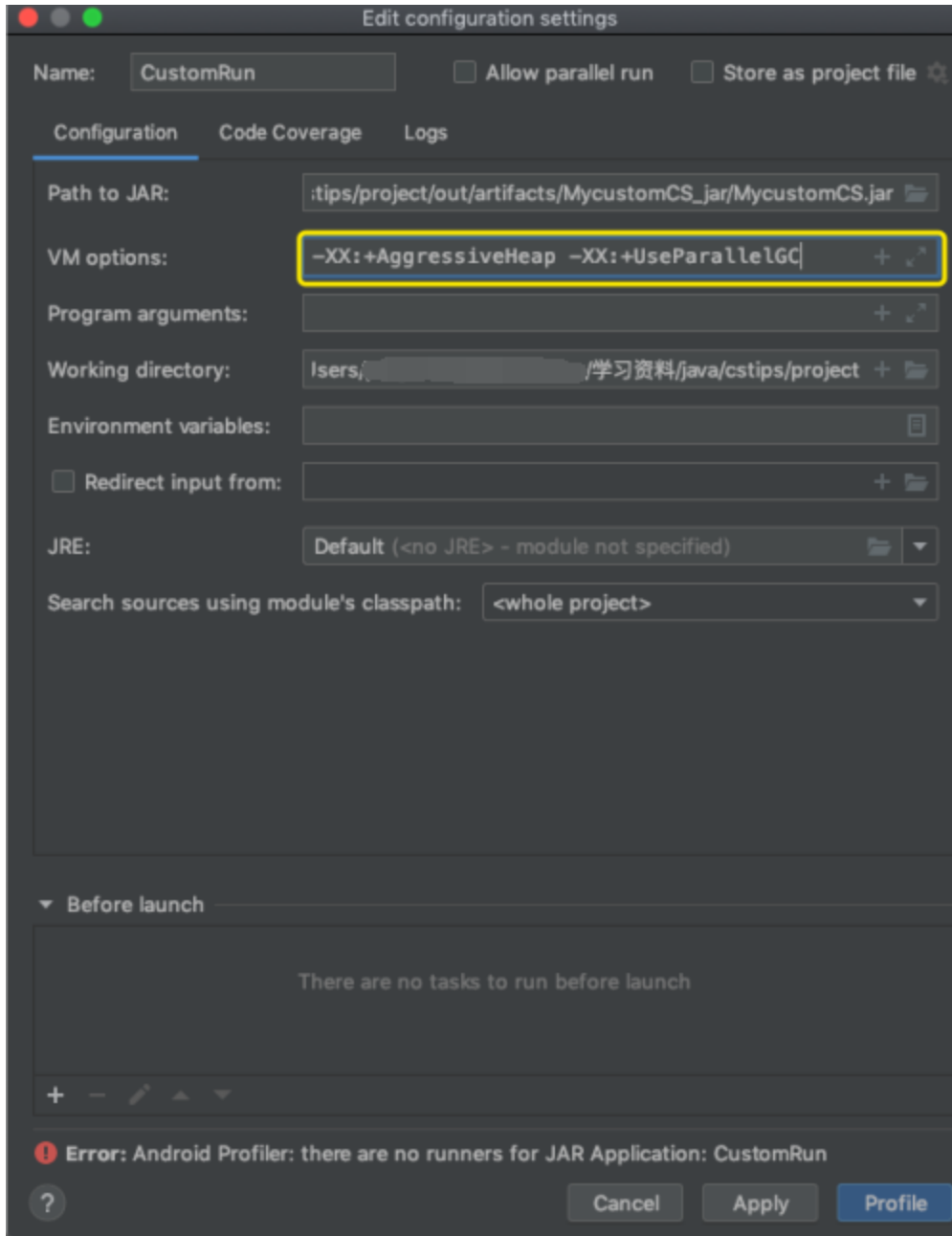




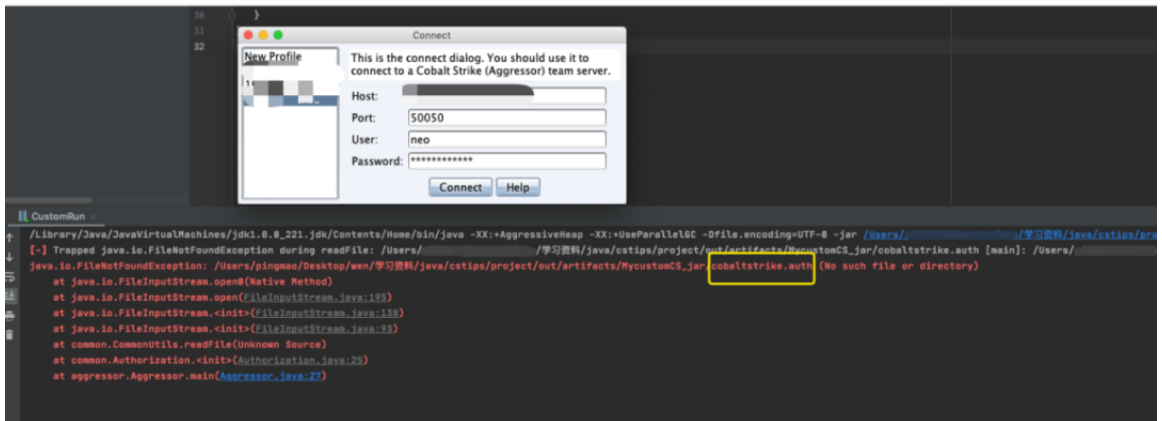
再次Run运行，继续复制 `-XX:+UseParallelGC`

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/java -XX:+AggressiveHeap -Dfile.encoding=UTF-8 -jar /Users/.../project/out/artifacts/MyCustomCS_jar/MyCustomCS.jar  
[-XJVM] Java -XX:+UseParallelGC option not set. Use the Cobalt Strike Launcher. Don't click the .jar file!  
Process finished with exit code 0
```

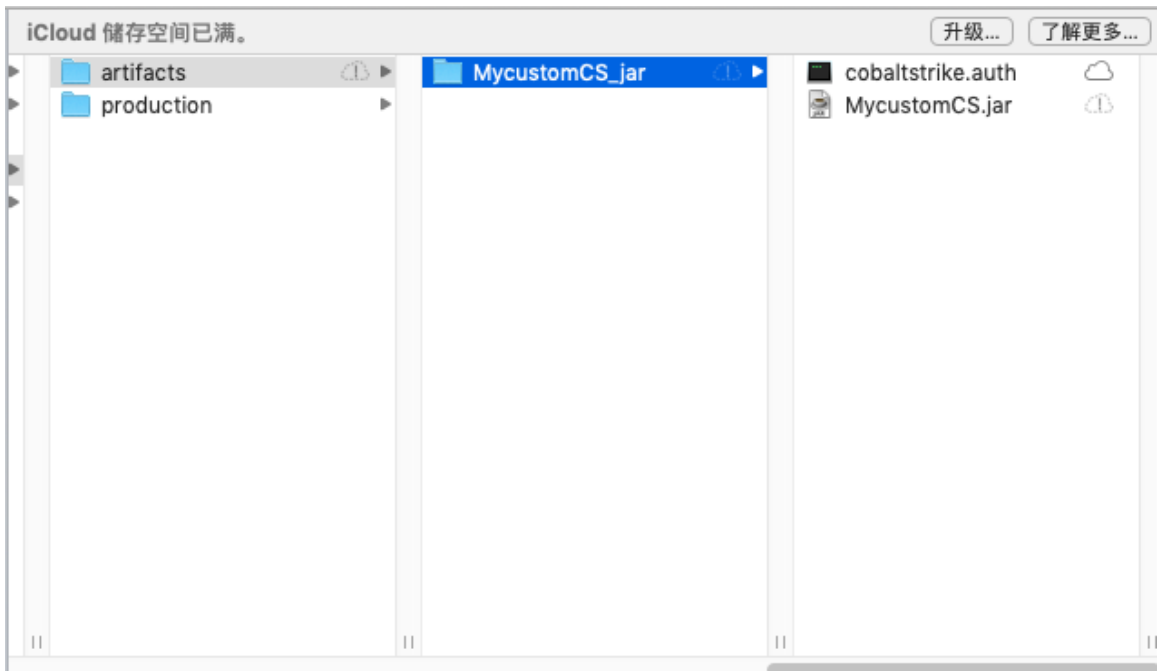
继续在添加到 `VM options` 中，记得要用空格隔开。



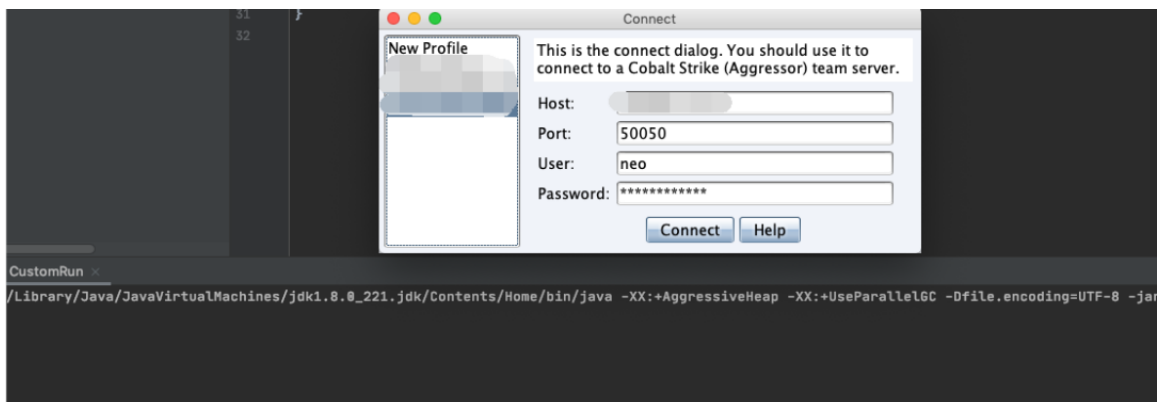
再次运行，提示 `auth` 文件找不到。



这里把初始的 `cobaltstrike.auth` 文件复制到 `MycustomCS.jar` 同目录下。



最后运行，到这里我们需要进行二次开发的环境就搭建好了，接下来就可以根据我们的需要通过关键字在源码中定位到具体的功能实现的代码，从而进行修改，或作一些功能上的增强。



### 三 总结

本文使用 IntelliJ IDEA自带的java反编译工具，对cobaltstrike进行了快速的反编译，并展示了修改后的效果，权当抛砖引玉，具体的插件开发各位师傅可以结合具体场景进行编写。

#### 参考资料

红队学院CSTips







知其黑 守其白

分享知识盛宴，闲聊大院趣事，备好酒肉等你



长按二维码关注 酒仙桥六号部队

精选留言

---

用户设置不下载评论