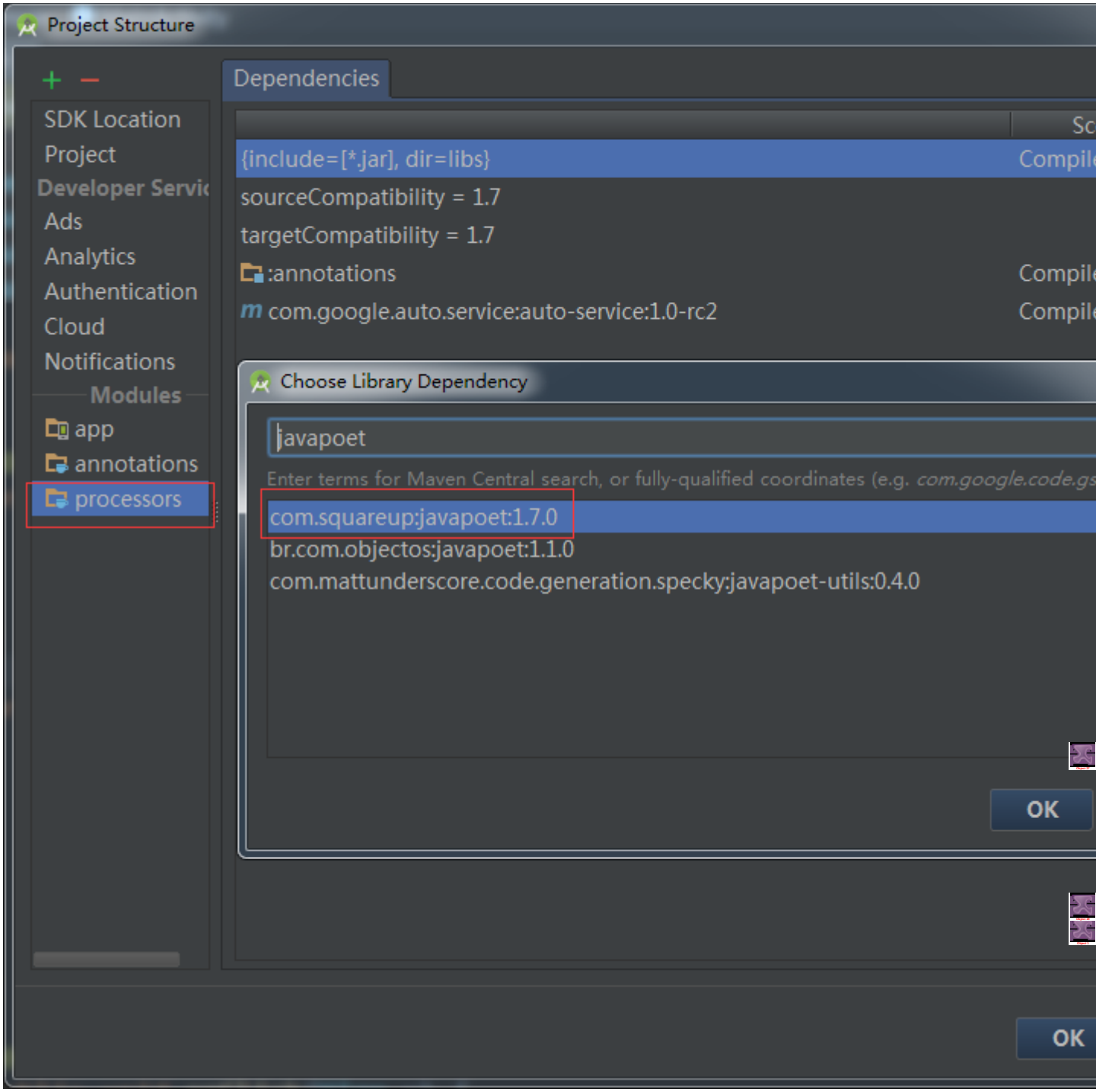
使用编译时注解时，需要在编译期间对注解进行处理，在这里我们没办法影响程序的运行逻辑，但我们可以进行一些需处理，比如生成一些功能性代码来辅助程序的开发，最常见的是生成.**Java** 源文件，并在程序中可以调用到生成的文件。这样我们就可以用注解来帮助我们处理一些固定逻辑的重复性代码（如 butterknife），提高开发的效率。

通过注解处理器来生成 .java 源文件基本上都会使用 javapoet 这个库，JavaPoet 一个是用于产生 .java 源文件的辅助库，它可以很方便地帮助我们生成需要的.java 源文件，下面来看下具体使用方法。

JavaPoet

在使用前需要先引入这个库，和 AutoService 一样可以通过 AndroidStudio 直接添加，如下：



下面以最简单的 HelloWorld 例子来看下怎么使用 JavaPoet。

先定义一个注解：

```java
/**
 * JavaPoet HelloWorld 例子
 */
@Retention(RetentionPolicy.CLASS)
@Target(ElementType.TYPE)
public @interface JPHelloWorld {
}
```

在定义个注解处理器来处理这个注解：

```java
/**
 * 处理 HelloWorld 注解.
 */
@AutoService(Processor.class)
public class HelloWorldProcess extends AbstractProcessor {

    private Filer filer;

    @Override
    public synchronized void init(ProcessingEnvironment processingEnv) {
        super.init(processingEnv);
        // Filer 是个接口，支持通过注解处理器创建新文件
        filer = processingEnv.getFiler();
    }

    @Override
    public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnv) {
        for (TypeElement element : annotations) {
            if (element.getQualifiedName().toString().equals(JPHelloWorld.class.getCanonicalName())) {
                // 创建 main 方法
                MethodSpec main = MethodSpec.methodBuilder("main")
                        .addModifiers(Modifier.PUBLIC, Modifier.STATIC)
                        .returns(void.class)
                        .addParameter(String[].class, "args")
                        .addStatement("$T.out.println($S)", System.class, "Hello, JavaPoet!")
                        .build();
                // 创建 HelloWorld 类
                TypeSpec helloWorld = TypeSpec.classBuilder("HelloWorld")
                        .addModifiers(Modifier.PUBLIC, Modifier.FINAL)
                        .addMethod(main)
                        .build();

                try {
                    // 生成 com.example.HelloWorld.java
                    JavaFile javaFile = JavaFile.builder("com.example", helloWorld)
                            .addFileComment(" This codes are generated automatically. Do not modify!")
                            .build();
                    //   生成文件
                    javaFile.writeTo(filer);
```

```
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return true;
  }

  @Override
  public Set<String> getSupportedAnnotationTypes() {
    Set<String> annotations = new LinkedHashSet<>();
    annotations.add(JPHelloWorld.class.getCanonicalName());
    return annotations;
  }

  @Override
  public SourceVersion getSupportedSourceVersion() {
    return SourceVersion.latestSupported();
  }
}
```

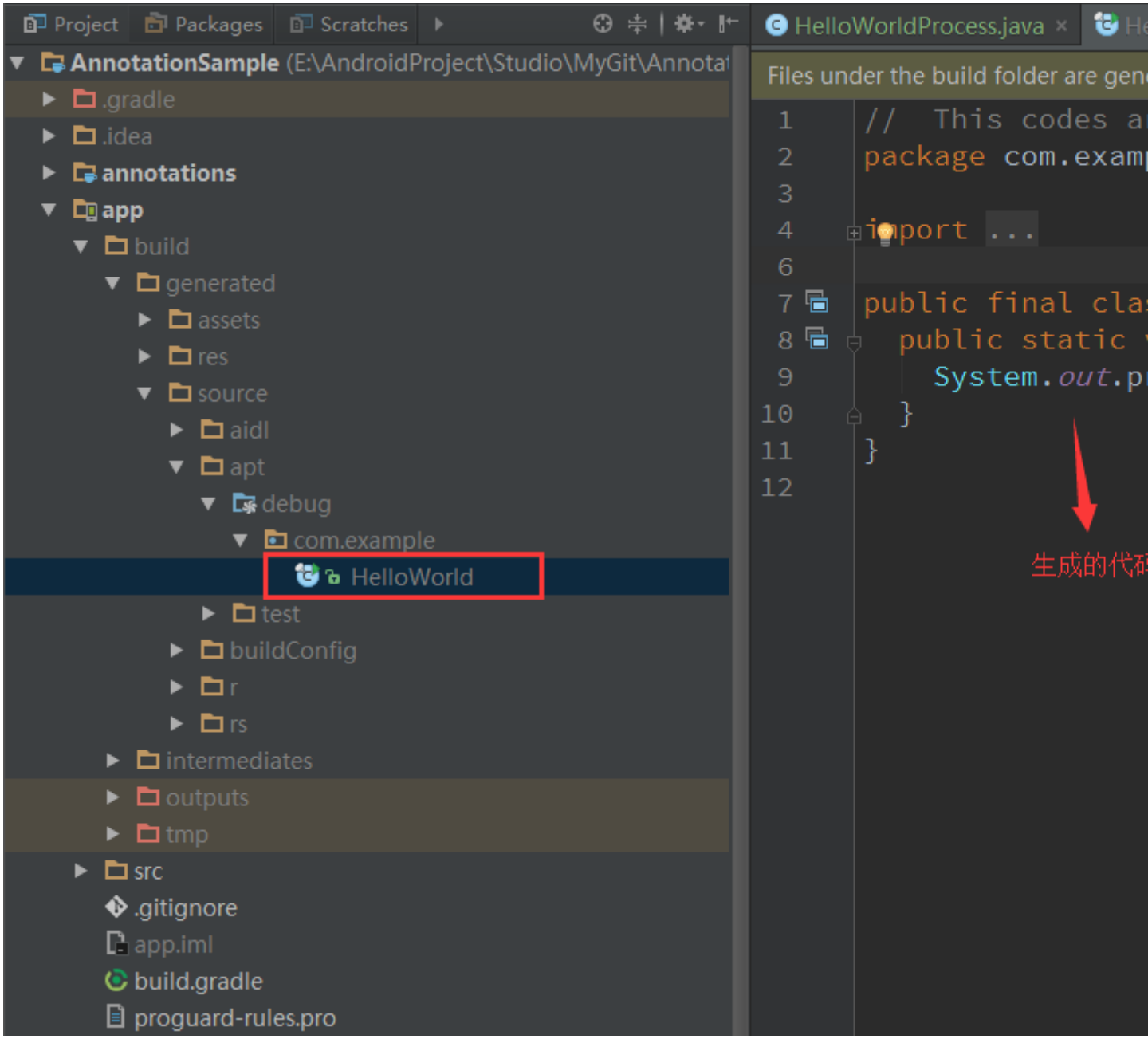
在用 JavaPoet 来生成一个 HelloWorld.java 文件之前，我们还必须在 init（）方法里获取到 Filer，这是一个用来辅助创建文件的接口，我们生成文件都通过它来处理。在 process（）方法先创建了个 MethodSpec 表示一个方法，再创建一个 TypeSpec 表示一个类并添加上前面创建的方法，最后用 JavaFile 来生成对应的 HelloWorld.java 并写入文件。

这是最简单的例子，整个语法结构也很清晰，相信做编程的看到这些使用方法都能猜到是做什么用的，我就没详细说了。除了这个例子，Github 上还有很多其它示例，如果你想很好地了解编译时注解的使用的话，还是很有必要把每个例子都过一遍，如果不想自己敲粘贴复制下很容易的。

在代码中使用定义的注解：

```
@JPHelloWorld
public class MainActivity extends AppCompatActivity{
    // ...
}
```

重新 Make 下工程就可以看到生成的 HelloWorld.java 文件了，目录如下：



可以看到已经成功生成了 HelloWorld.java 文件，注意生成文件所在的目录，现在我们就可以在项目中直接使用这个 java 类了。当然了，这个例子没有什么实际的使用价值，你可以参考其它例子来生成你想要的代码，用法是很多的。

这里只写了个最简单的例子，没有深入更详细的使用方法，等后面有时间再来整理个更详细的介绍。