如果你对逆向有所涉猎的话，可能听说过 Hook，利用 Hook 技术我们可以在某一逻辑的前后加入自定义的逻辑处理代码，几乎可以实现任意逻辑的修改。

在前面的 JavaScript 逆向实战课时我们也初步体会了 Hook 的功效，如果你对 Hook 的概念还不太了解，可以搜索一下“Hook 技术”相关的内容来了解下。

对于 App 来说，Hook 技术应用非常广泛。比如朋友圈微信步数的修改，其实就是通过 Hook 数据发送的方式实现步数的修改。比如处理安卓的 SSL Pining，用 Hook 技术也可以修改 SSL 证书校验结果，实现校验的绕过。对于 App 爬虫来说，我们也可以使用 Hook 一些关键的方法获取方法执行前后的结果，从而实现数据的截获。

那这些技术怎么来实现呢？这里就不得不提一个框架，叫作 Xposed。

Xposed 介绍

Xposed 框架（Xposed Framework）是一套开源的，在 Android 高权限模式下运行的框架服务，可以在不修改 App 源码的情况下影响程序运行（修改系统）的框架服务，基于它可以制作出许多功能强大的模块，且在功能不冲突的情况下同时运作。

Xposed 框架的原理是通过替换系统级别的 `/system/bin/app_process` 程序控制 zygote 进程，使得`app_process` 在启动过程中会加载 `XposedBridge.jar` 这个 jar 包，这个 jar 包里面定义了对系统方法、属性的一系列 Hook 操作，同时还提供了几个 Hook API 供我们编写 Xposed 模块来使用。我们编写一个 Xposed 模块时，引用 Xposed 提供的几个 Hook 方法就可以实现对系统级别任意方法和属性的修改了。

这么说可能有点抽象，下面我们来编写一个 Xposed 模块带你体会一下 Xposed 的用法，最后再使用 Xposed 来实现一个真实 App 执行逻辑的修改。

本节要求

由于 Xposed 是运行在 Android 平台上的，所以本节我们的环境就和 Android 相关。

开始学习本节课之前，需要做如下准备工作：

- 配置好 Android 开发环境，开发环境的搭建流程可以参考下面的几个链接：
    - https://juejin.im/post/5d255101e51d4556d86c7b4f
    - https://juejin.im/post/5d5eb3ed5188252ae10be138
- 准备一个已经 ROOT 的安卓设备并连接好 PC，可以使用虚拟机或真机，比如我使用的是虚拟机网易 Mumu，已经自带了 ROOT 功能，当然如果你有已经 ROOT 的真机也是可以的。
- 安装好 jadx、jadx-gui，这是一款用来反编译 apk 的软件，安装参考链接见：https://github.com/skylot/jadx。

准备好了如上环境之后，我们就开始 Xposed 模块的编写吧。

Xposed Installer 安装

有了如上的环境后，首先我们需要先安装 Xposed。

要安装 Xposed 我们需要借助于一个叫作 Xposed Installer 的 App，它就是用来安装 Xposed 框架的，利用它我们可以下载和安装 Xposed 框架，同时还能查看和管理 Xposed 模块，还能查看一些 Xposed 框架输出的日志信息等。

怎么安装呢？

我们可以先打开 Xposed 的官网 https://repo.xposed.info/module/de.robv.android.xposed.installer，然后就可以看到一些有关 Xposed 的提示信息。

这里提示了这么一条重要信息：

> For Android 5.0 or higher (Lollipop/Marshmallow), these versions don't work! Use this instead: http://forum.xda-developers.com/showthread.php?t=3034811

很明显，如果你的手机安卓版本在 5.0 以下，可以直接点击首页下方的 apk 下载，如果是 5.0 或更高版本，那么就到 http://forum.xda-developers.com/showthread.php?t=3034811 这个链接下载，后者的下载的真实链接是 https://forum.xda-developers.com/attachment.php?attachmentid=4393082&d=1516301692，可以直接点击下载。

> 注意：由于时间上的变化，下载地址以官方介绍为准。

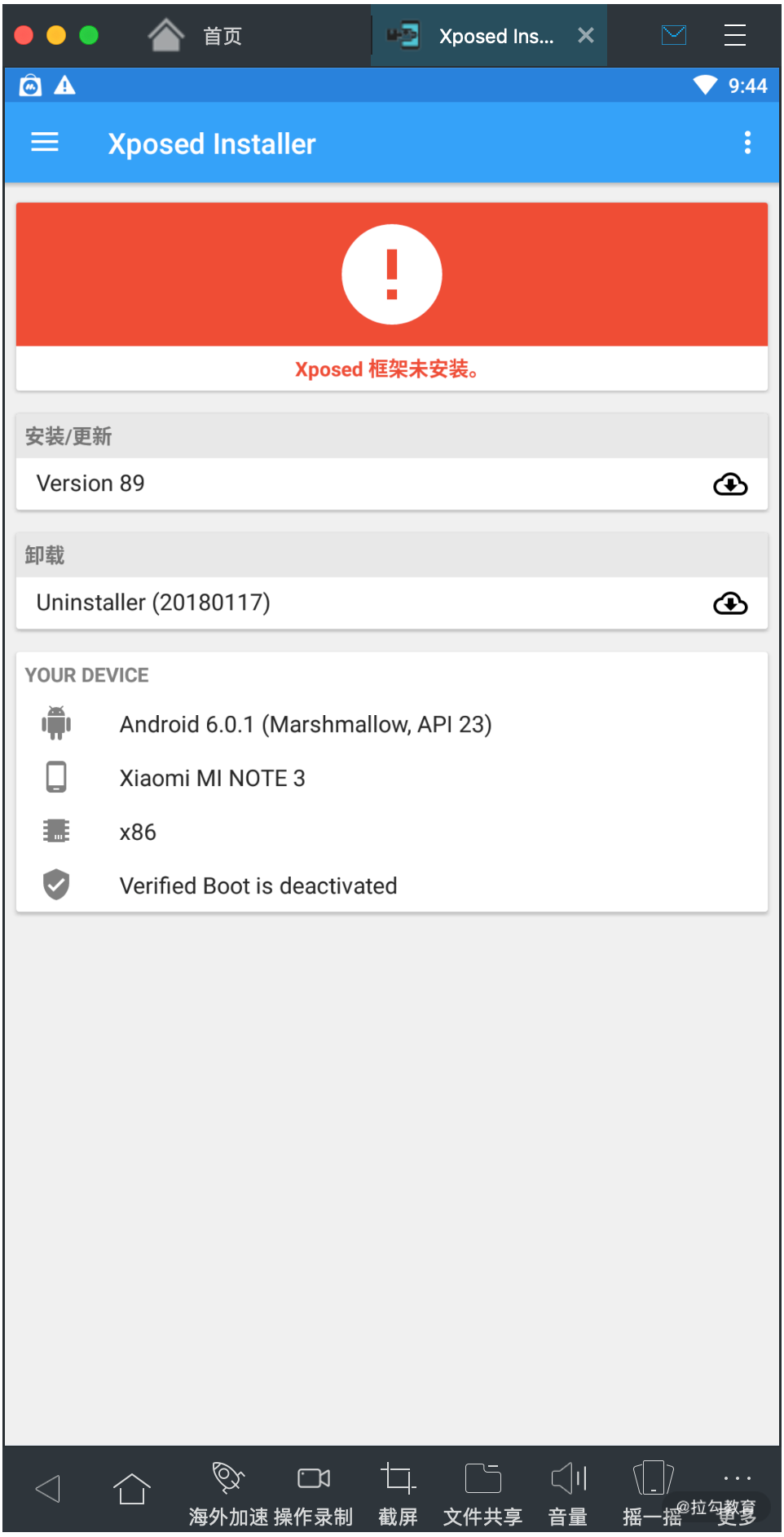
下载完成之后会得到一个 apk 文件，我们可以直接在真机或模拟机上安装。

安装完成之后便会出现这样的一个图标：

这就代表 Xposed Installer 已经安装成功了。

下面我们打开它，它可能会提示需要 ROOT 权限，授予即可。

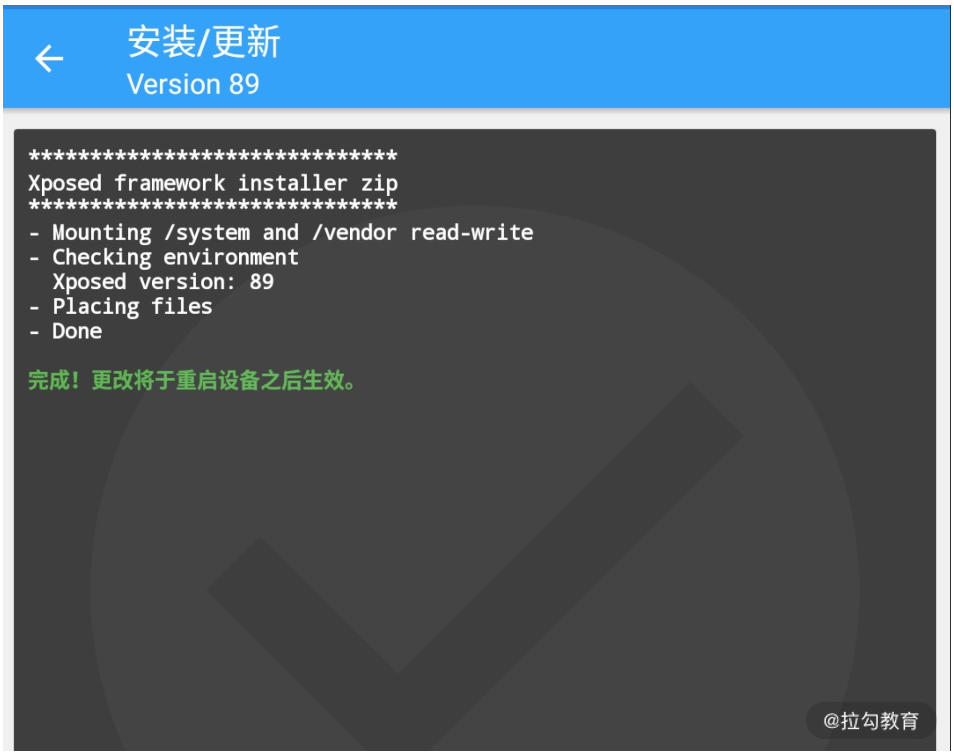
打开之后的界面可能如下所示：



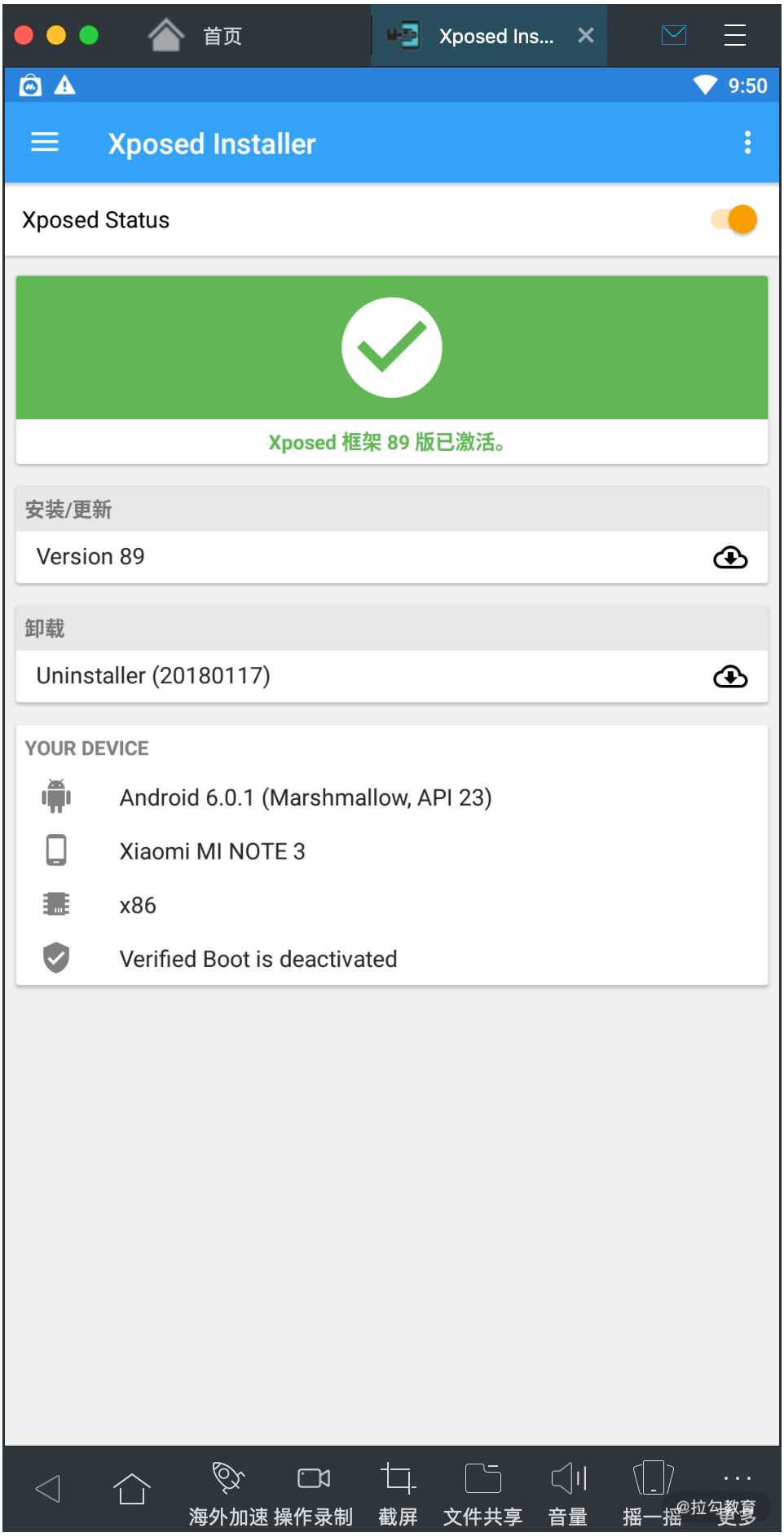
这里它提示 Xposed 模块未安装，所以我们需要安装一下。

点击安装，如图所示。

安装完成之后它会提示重启设备后生效，如图所示：



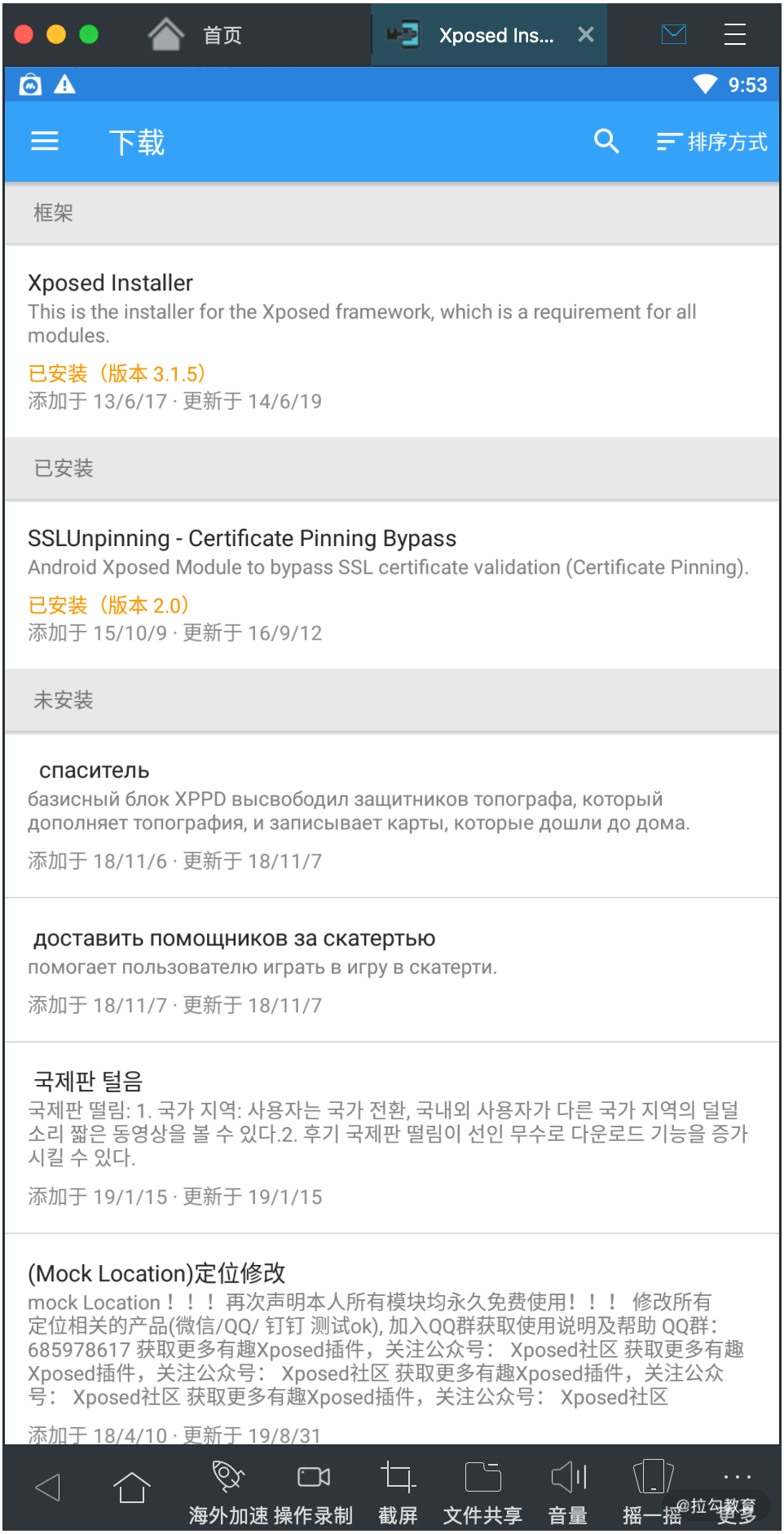
接着我们重启设备，便可以看到如下界面，代表 Xposed 框架已经安装并激活，如图所示：



这时候我们便可以开始编写 Xposed 模块了。

Xposed 模块

现在 Xposed 的生态非常庞大，基于 Xposed 开发的模块非常之多，点击下载菜单可以看到已有的发布的 Xposed 模块，如图所示：



五花八门的模块非常多，比如修改微信步数、修改系统定位、自动抢红包等等。当然我们也可以编写自己的模块来实现想要的功能。

这时候你可能就会问了，这模块究竟是干吗的？它到底是一个什么东西？

其实本质上来说，它就是一个安卓 App，开发一个 Xposed 模块其实流程上就和开发一个安卓 App 差不多，只不过相比 App 开发来说多了下面四个步骤：

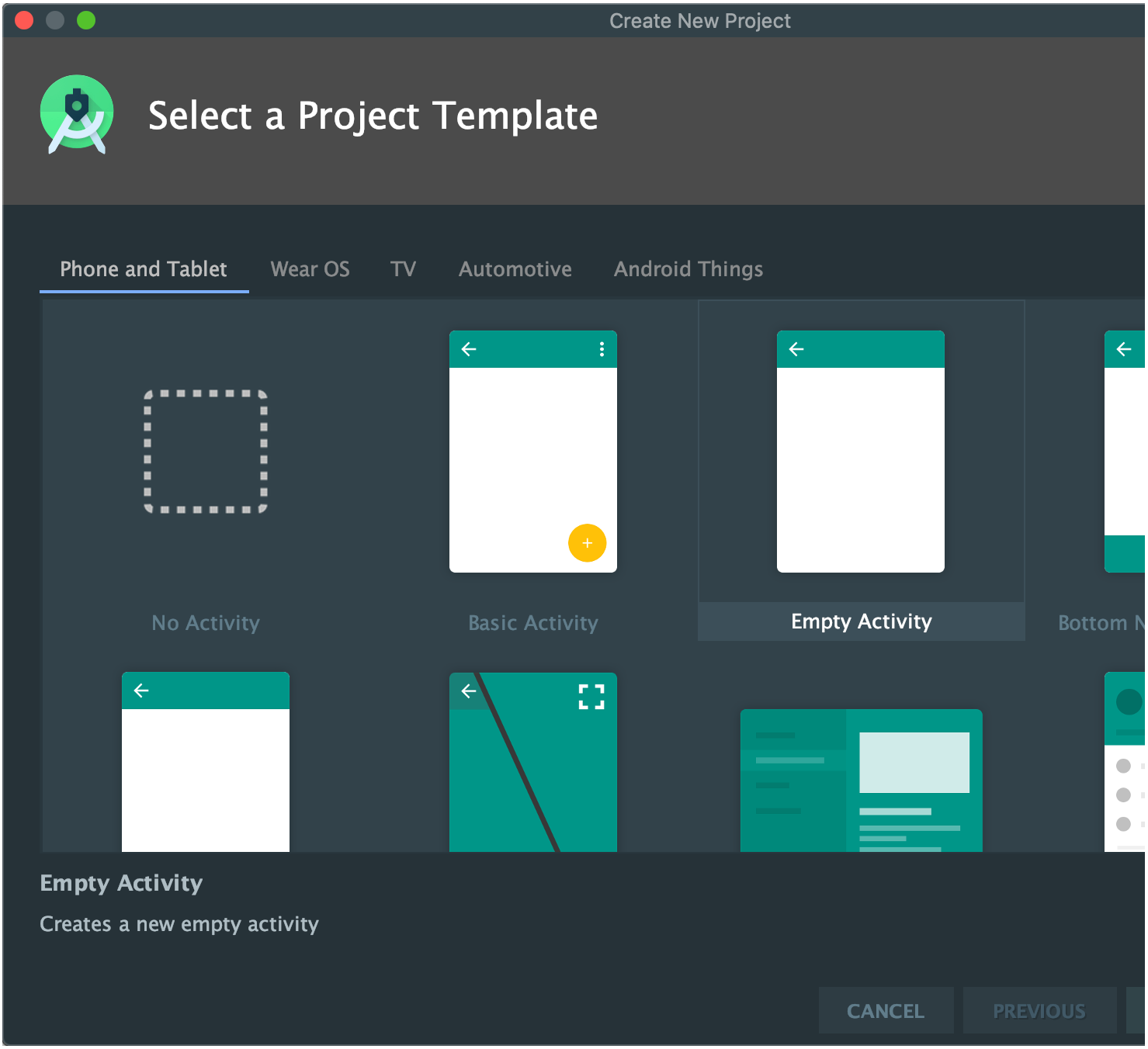
1. 这个 App 里面要加上一些标识，标明这个 App 是一个 Xposed 模块，以便安装之后 Xposed 框架可以识别出来。
2. 这个 App 里面需要引入 Xposed 的 jar 包，从而能实现 Hook 操作。
3. App 里面定义一些 Hook 操作，可以对本 App 或其他的 App 的逻辑进行修改。
4. 定义完这些 Hook 操作逻辑之后，还需要告诉 Xposed 框架哪些是我们自己定义的 Hook 操作逻辑，以便 Xposed 执行这些 Hook 逻辑。

就这么四步，这四步这么来实现呢，下面我们来一步步实现。

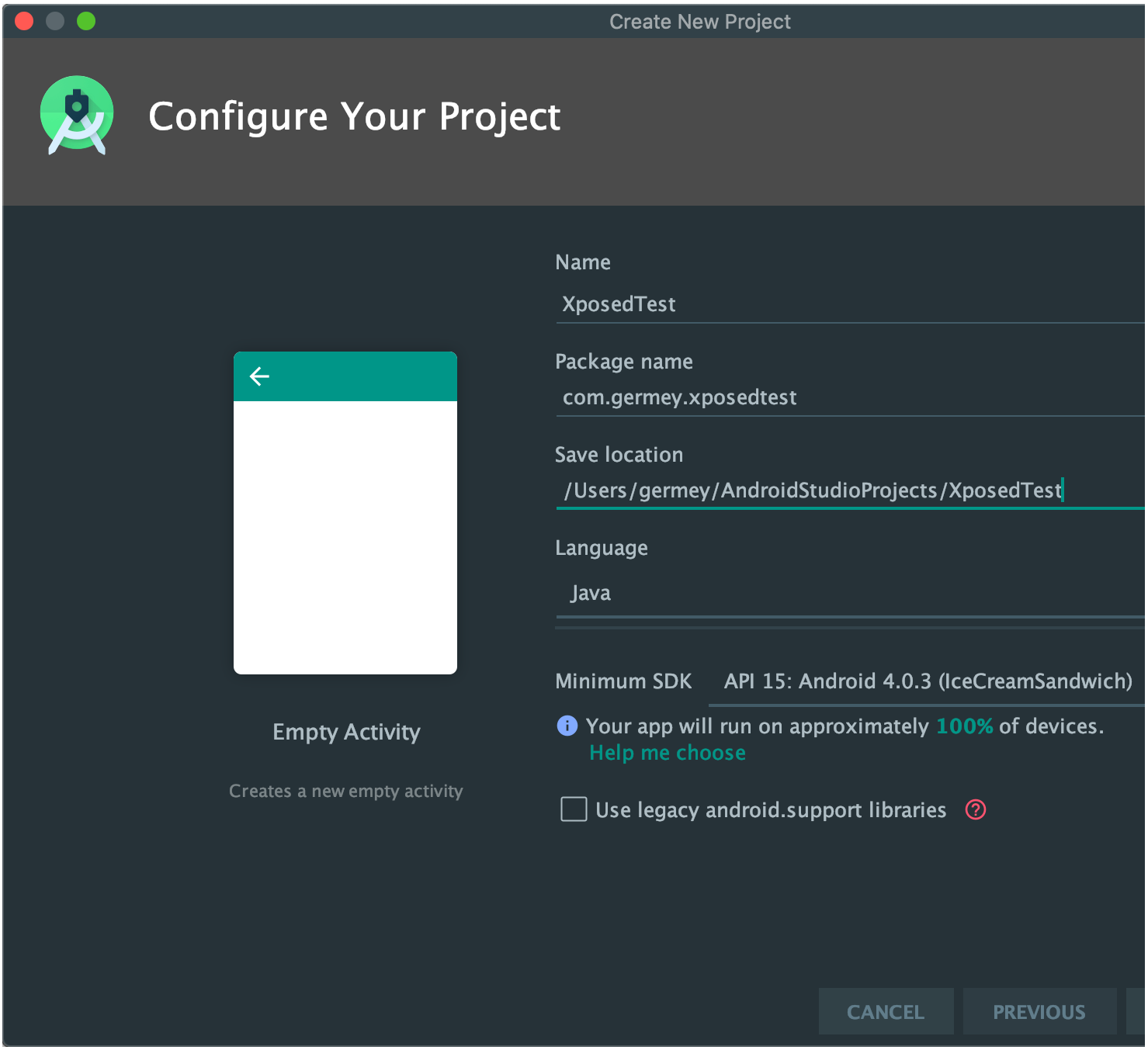
Xposed 模块开发

下面我们来根据上面的四步来进行一个 Xposed 模块开发吧。

首先我们用 Android Studio 新建一个安卓项目，第一步提示我们选择 Activity，直接选择默认的 Empty Activity 即可，如图所示：

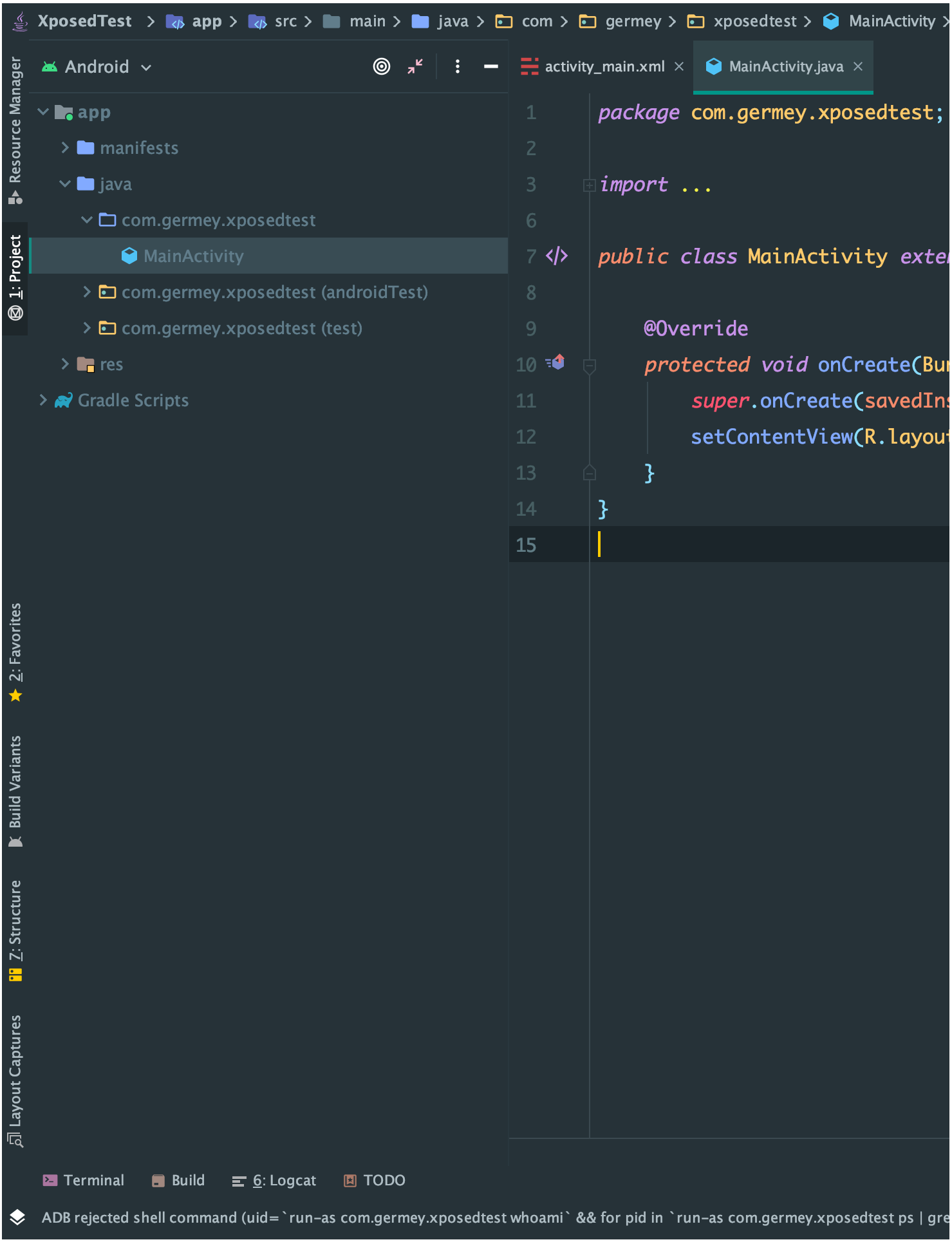


名称叫作 XposedTest，包名可以任意取，然后指定好项目路径和编写语言，同时指定最小 SDK 版本为 15，如图所示：



点击 FINISH，创建这个项目。

最后生成的界面如图所示：

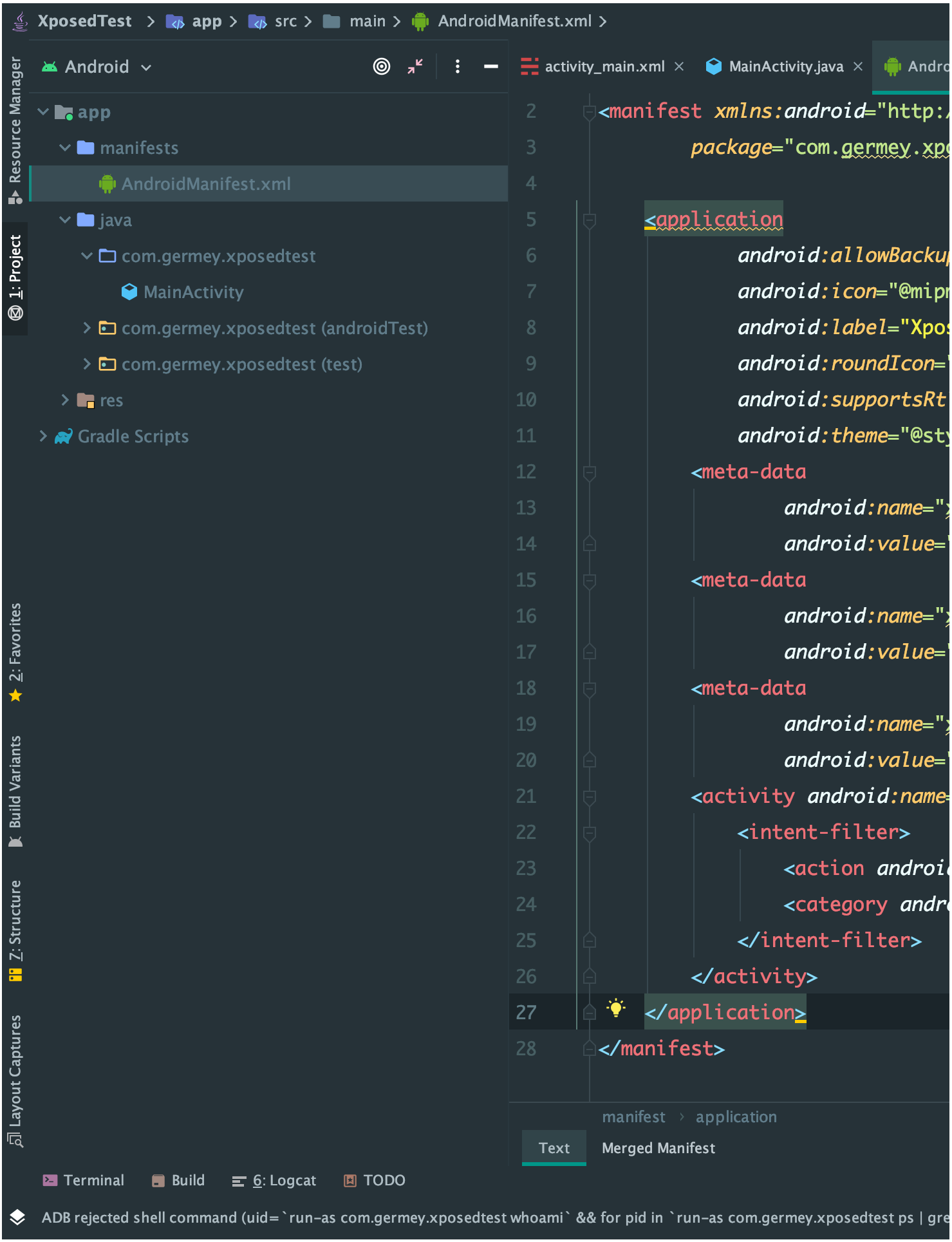


然后我们开始第一步的配置，先配置一些标识符，标识这是一个 Xposed 模块。

我们打开 AndroidManifest.xml 文件，添加如下内容：

```
<meta-data
  android:name="xposedmodule"
  android:value="true" />
<meta-data
  android:name="xposeddescription"
  android:value="Xposed Test" />
<meta-data
  android:name="xposedminversion"
  android:value="53" />
```

到 application 标签内，和 activity 标签并列，最终内容如图所示：

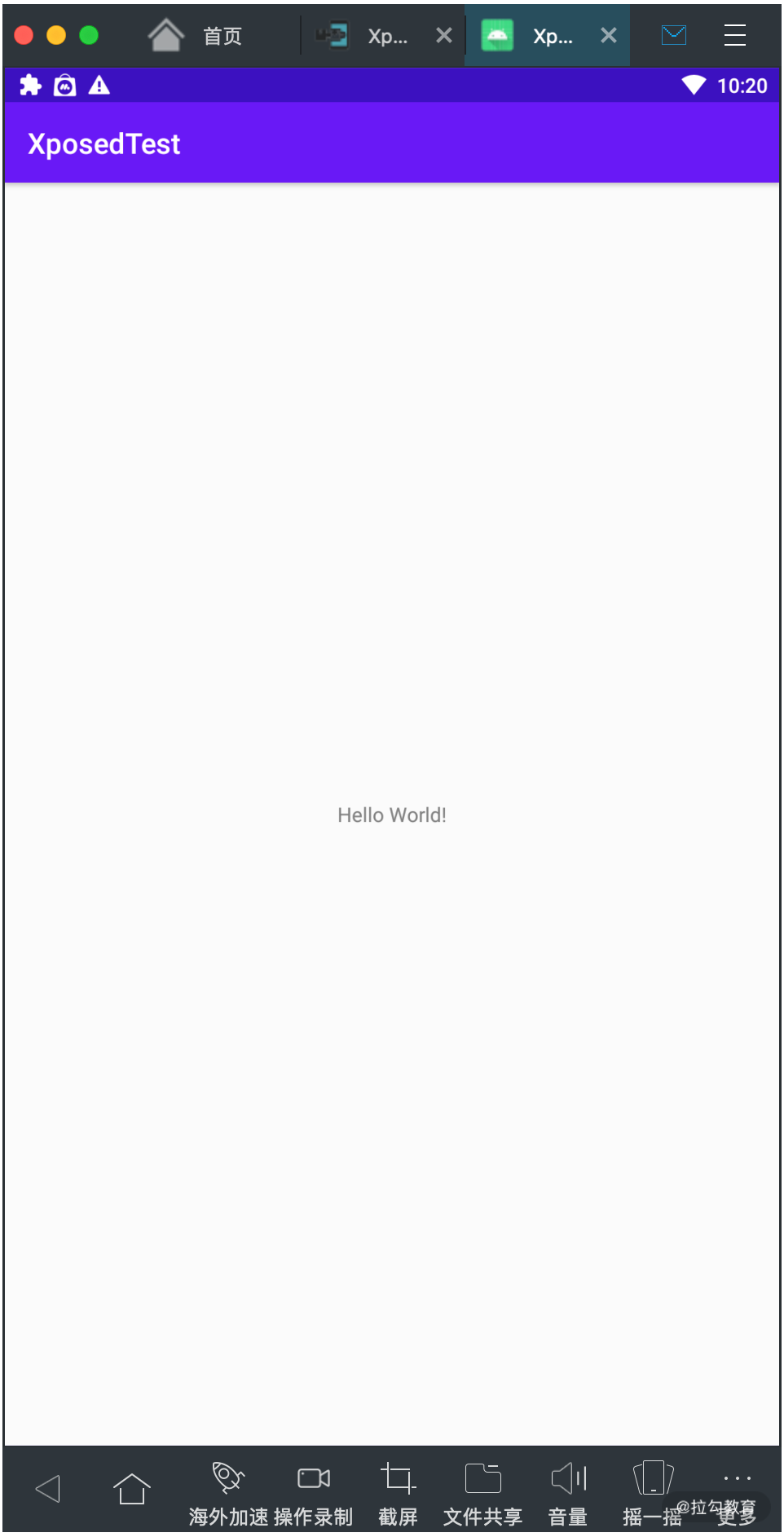


这里指定了三个 meta-data，分别如下。

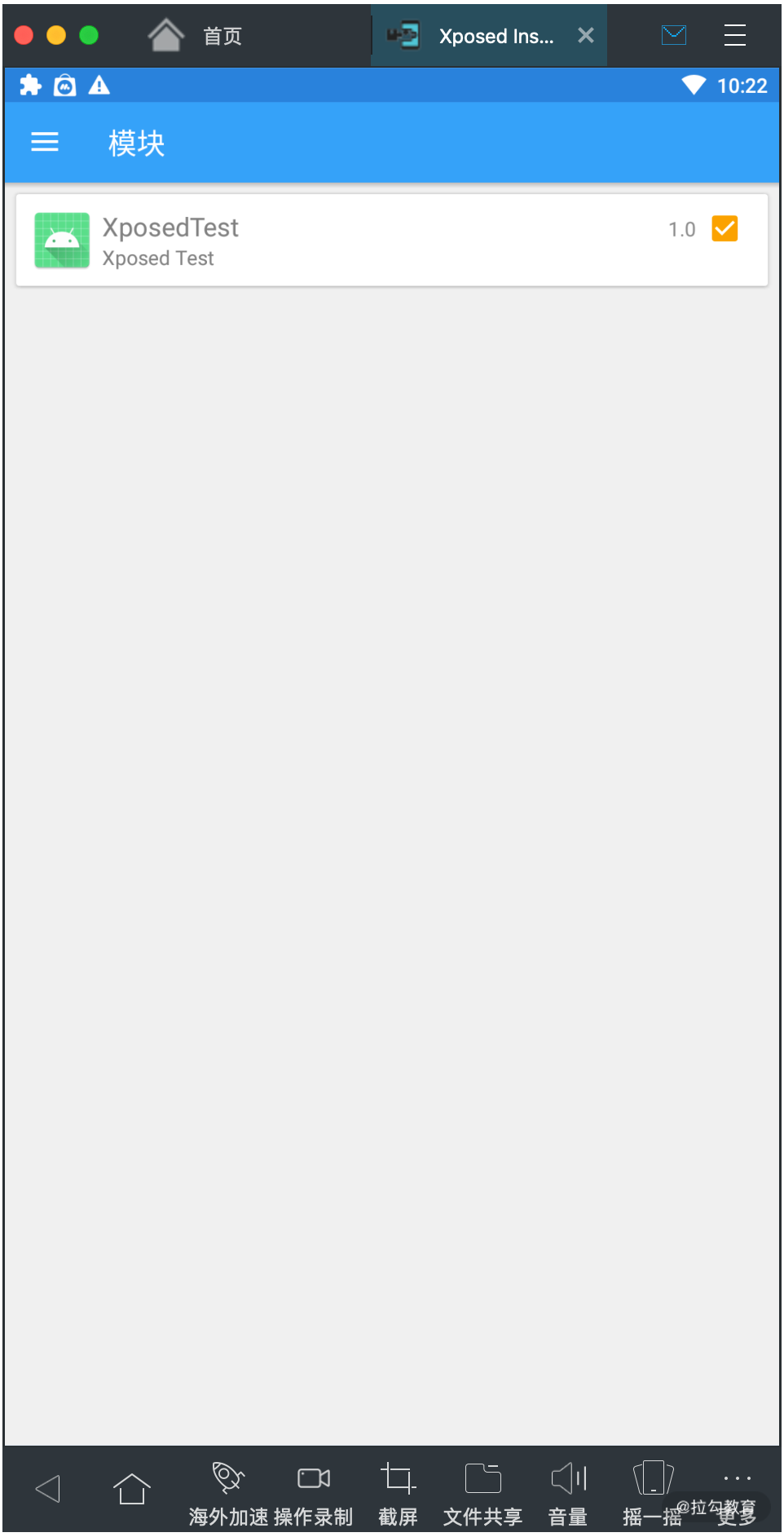
- xposedmodule：这里设置为 true，代表这是一个 Xposed 模块。
- xposeddescription：模块的描述，填写模块描述就好，就是一个字符串。
- xposedminversion：模块运行要求的 Xposed 最低版本号，这里是 53。

定义好这三个参数之后，把这个 App 安装到手机，Xposed 就能识别出这个 App 是一个 Xposed 模块了。

我们点击运行按钮，在手机上运行这个 App。这时候可以看到手机上出现了如下界面，如图所示：



这时候我们打开 Xposed Installer 的模块界面，就发现它检测到了这个模块，如图所示：



在这里我们把它勾选上，就成功启用了这个 Xposed 模块。不过值得注意的是，需要重启设备才能生效。可以手动启动设备或者通过 Xposed Installer 首页的重启选项来进行重启。

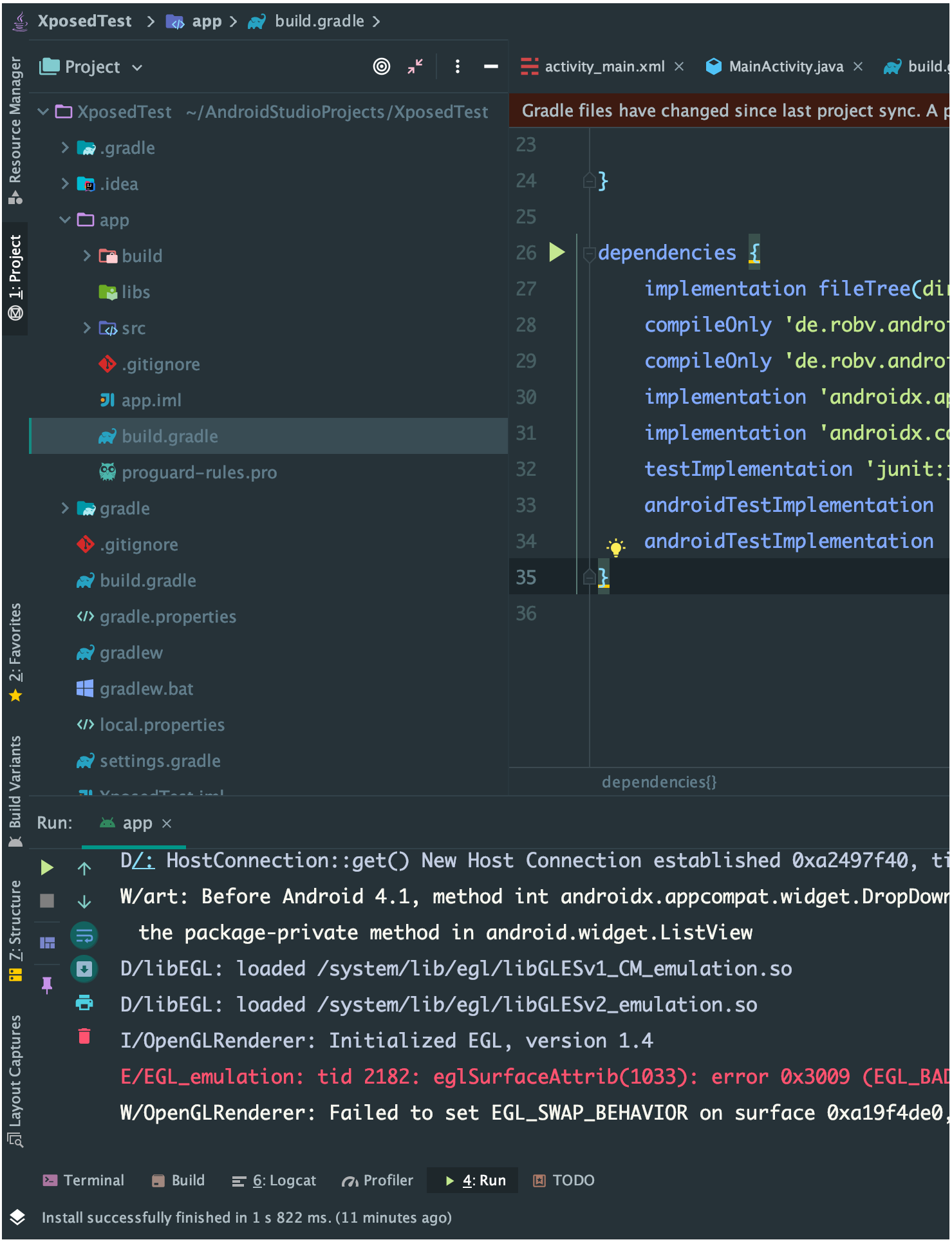
但是，现在启用了也没什么用啊，因为这个模块里面什么功能都还没有呢。

接下来我们再在项目中引入 Xposed 相关的 SDK，这样我们才能调用 Xposed 提供的一些 Hook 操作方法，实现 Hook 操作。

打开 app/build.gradle 文件，在 dependencies 区域添加如下两行代码：

```
compileOnly 'de.robv.android.xposed:api:82'
compileOnly 'de.robv.android.xposed:api:82:sources'
```

这是 Xposed 的 SDK，添加之后 Android Stuido 会检测到项目配置发生的变化，在右上角会提示一个“Sync Now”的选项，我们点击之后，新添加的 Xposed SDK 便会自动下载和安装，如图所示：



好，现在 Xposed 的 SDK 就安装成功了，下面我们就能使用里面的方法来实现逻辑的 Hook 了。

那怎么来实现逻辑的 Hook 呢？Hook 什么呢？那总得有点逻辑吧？哪来的逻辑呢？自己先写一个吧。

说干就干，这里我们就添加一个鼠标响应事件，点击之后触发一个算式计算的逻辑。

首先我们修改下页面内容，把当前的文本框设置成一个按钮，以便点击触发，修改 app/src/main/res/layout/activity_main.xml 文件，内容替换为如下内容：

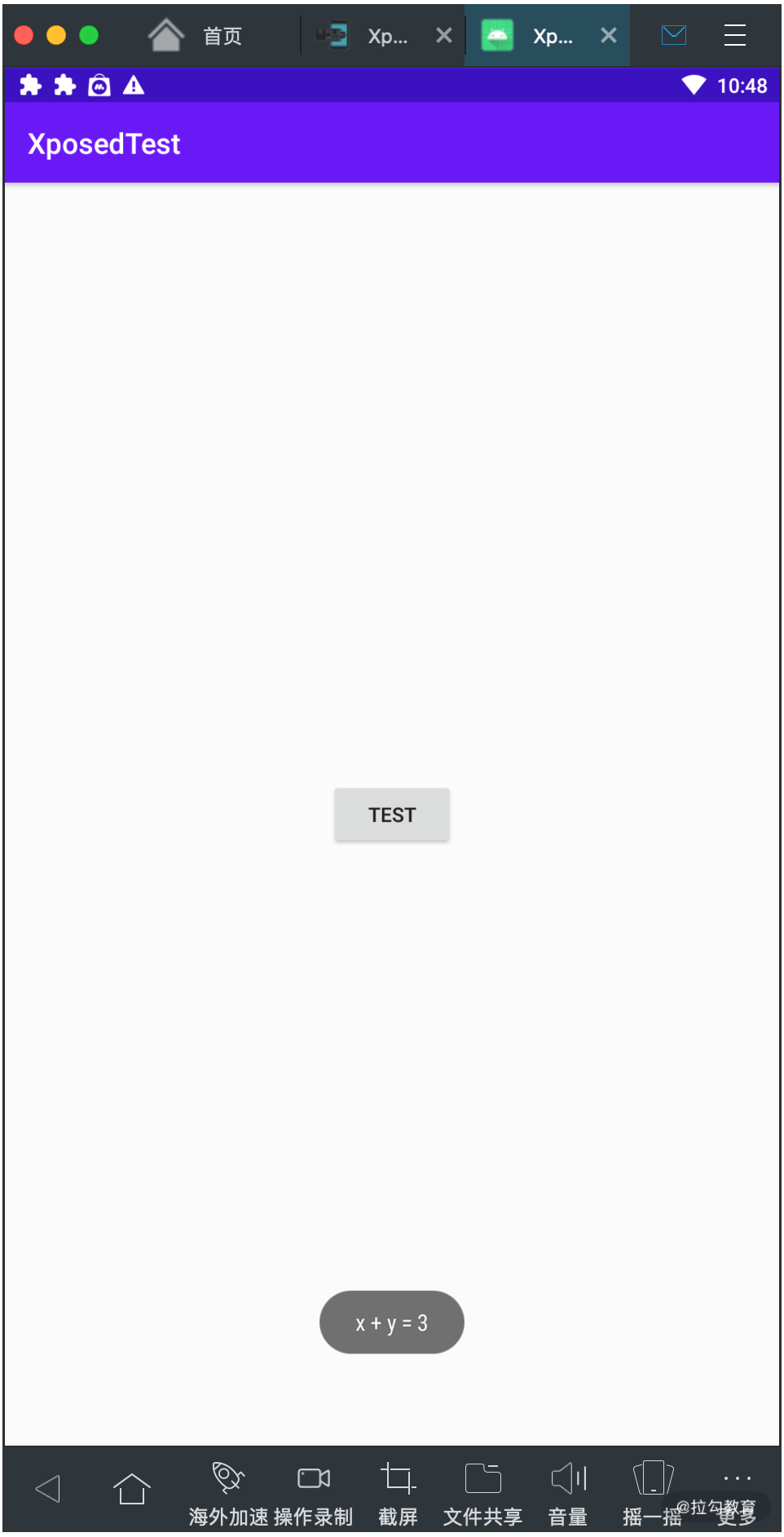
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">
    <Button
            android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Test"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

这时候重新运行 App 就会出现一个按钮了，而不是一行文本框。接下来再修改下 MainActivity.java 文件，内容如下：

```
package com.germey.xposedtest;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
    private Button button;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button = findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Toast.makeText(MainActivity.this, showMessage(1, 2), Toast.LENGTH_SHORT).show();
            }
        });
    }
    public String showMessage(int x, int y) {
        return "x + y = " + (x + y);
    }
}
```

这里我们定义了一个 Button，然后使用 findViewById 方法从视图里面获取到了这个 Button 对象，同时我们为这个 Button 添加了一个点击事件，点击之后会生成一个 Toast 提示，其内容为 showMessage 方法返回的结果。
这个 showMessage 方法接收两个参数，是 int 类型的 x 和 y，返回的结果是一个字符串，即“x + y =”这个字符串再拼接上二者计算得到的结果，其实就是一个算数表达式。

这里 showMessage 在调用的时候我们传入了 1 + 2，所以最后 showMessage 显示的结果应该为“x + y = 3”，我们重新运行下 App，然后点击 TEST 按钮，可以看到如下运行结果：



这样我们的一个基本的逻辑就定义好了。

定义好了之后呢，下一步我们就来用 Xposed 进行 Hook 吧，我们在 MainActivity.java 同级新建一个 Java Class，内容如下：

```
package com.germey.xposedtest;
import de.robv.android.xposed.IXposedHookLoadPackage;
import de.robv.android.xposed.XC_MethodHook;
import de.robv.android.xposed.XposedBridge;
import de.robv.android.xposed.XposedHelpers;
import de.robv.android.xposed.callbacks.XC_LoadPackage;
public class HookMessage implements IXposedHookLoadPackage {
    public void handleLoadPackage(XC_LoadPackage.LoadPackageParam loadPackageParam) throws Throwable {
        if (loadPackageParam.packageName.equals("com.germey.xposedtest")) {
            XposedBridge.log("Hooked com.germey.xposedtest Package");
            Class clazz = loadPackageParam.classLoader.loadClass(
                    "com.germey.xposedtest.MainActivity");
            XposedHelpers.findAndHookMethod(clazz, "showMessage", int.class, int.class, new XC_MethodHook() {
                protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
                    XposedBridge.log("Called beforeHookedMethod");
                    param.args[0] = 2;
                    XposedBridge.log("Changed args 0 to " + param.args[0]);
                }
                protected void afterHookedMethod(MethodHookParam param) throws Throwable {
                    XposedBridge.log("Called afterHookedMethod");
                }
            });
        }
    }
}
```

这里我们就定义了 Hook 的相关逻辑了，这里梳理几个关键的点：

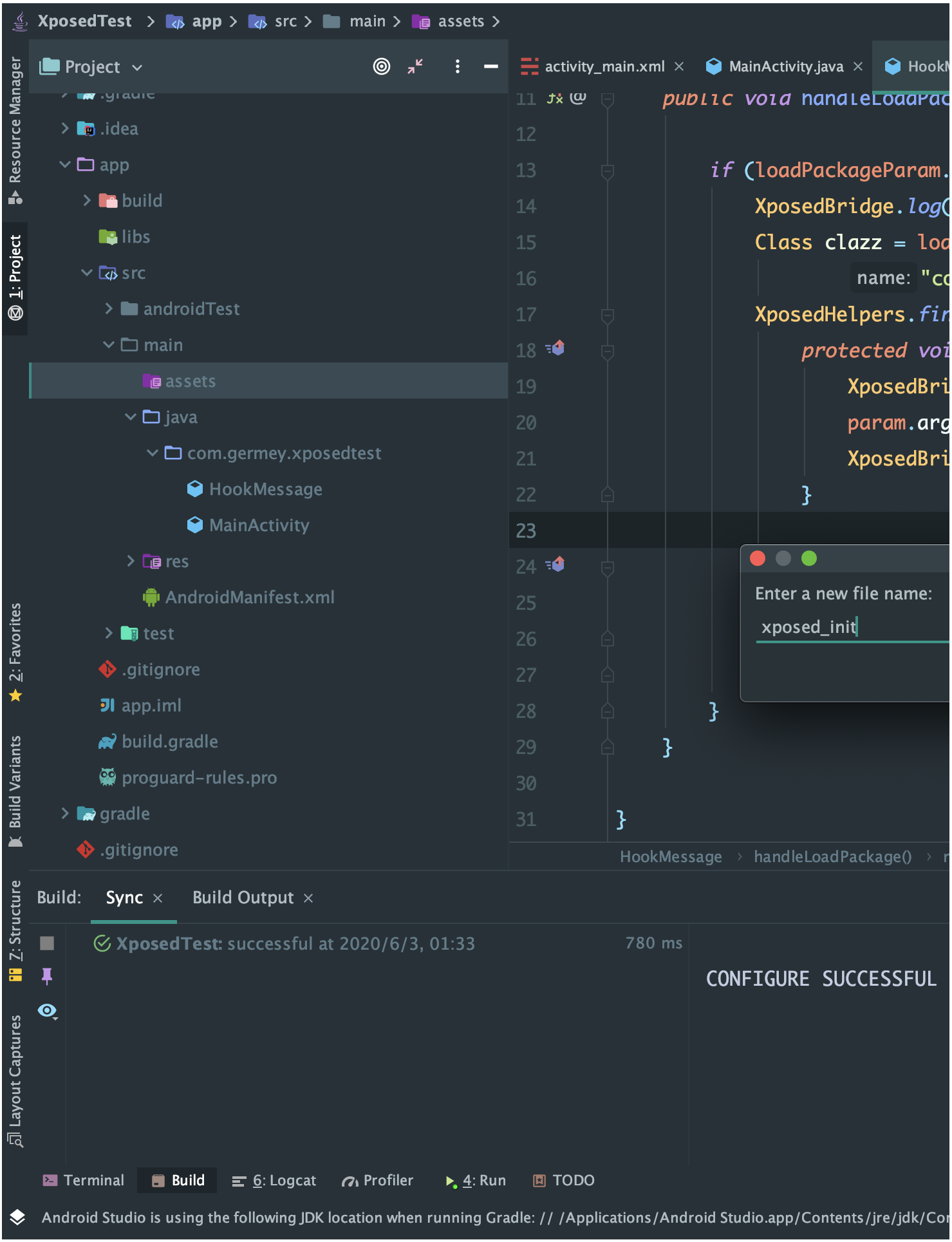
- 这里的 class 实现了 IXposedHookLoadPackage 接口，需要定义 handleLoadPackage 这个方法，这个方法会在每个 App 包加载时执行。
- 在 handleLoadPackage 里面我们通过 `loadPackageParam.packageName` 获取到了 App 包名，然后判断是否是我们当前 App 的包名。这里包名可以是任意 App 的包名，不一定是当前 App 的包名，只不过是因为我们为了方便，在当前 App 里面定义了一个逻辑，所以这里我们 Hook 的是当前 App 的逻辑，才填写了当前 App 的包名。
- 利用 loadClass 方法并指定 class 的路径可以动态地加载这个 class，是一个 Class 对象。
- 利用 XposedHelpers 提供的 findAndHookMethod 方法可以从 class 里面查找对应的方法，这里需要传入的参数分别为 Class 对象，方法名称，方法的参数类型，处理方法。这里方法的参数类型是有几个写几个，比如这里 showMessage 有两个 int 类型的参数，这里就需要顺次写两个 int.class，如果是其他的类型也是分别写类型再加 class 的声明。
- XC_MethodHook 里面定义了我们施行 Hook 的真正逻辑，这里通常可以实现两个方法，分别叫作 beforeHookedMethod 和 afterHookedMethod，分别代表在被 Hook 方法（这里为 showMessage）执行前的操作和执行后的操作，同时二者都有一个 MethodHookParam 类型的参数，里面包含了方法执行的参数和结果等信息。
- 一般来说 beforeHookedMethod 方法可以用来修改被 Hook 方法的参数内容，或者直接定义被 Hook 方法的运行流程。afterHookedMethod 可以用来对被 Hook 方法进行后处理，比如对被 Hook 方法的结果进行拦截、保存、转发、修改等操作。
- XposedBridge 里的 log 方法可以记录将 Log 信息记录到 Xposed Installer 里面，我们通过 Xposed Installer 里面日志页面就可以看到对应结果，方便做调试使用。

这里我们先对 beforeHookedMethod 处理，对 param 的 args 属性做了处理，这里的 args 属性是一个列表，就是 showMessage 方法的调用参数，因为我们之前传入的是 1 和 2，所以这里 args 属性的值其实就是 `[1, 2]`，那这里我们是把它改写了一下，把第一个内容改写成了 2，那这里 args 其实就会变成 `[2, 2]` 了。

好，现在 Hook 的逻辑我们已经实现好了，还差最后一步，那就是告诉 Xposed 我们的 Hook 逻辑是定义在哪里了，我们需要新建一个 Xposed 入口文件。

在 main 文件夹新建一个 assets folder，如图所示：

然后在 assets 文件夹下新建一个 xposed_init 文件，不需要有任何后缀，如图所示：

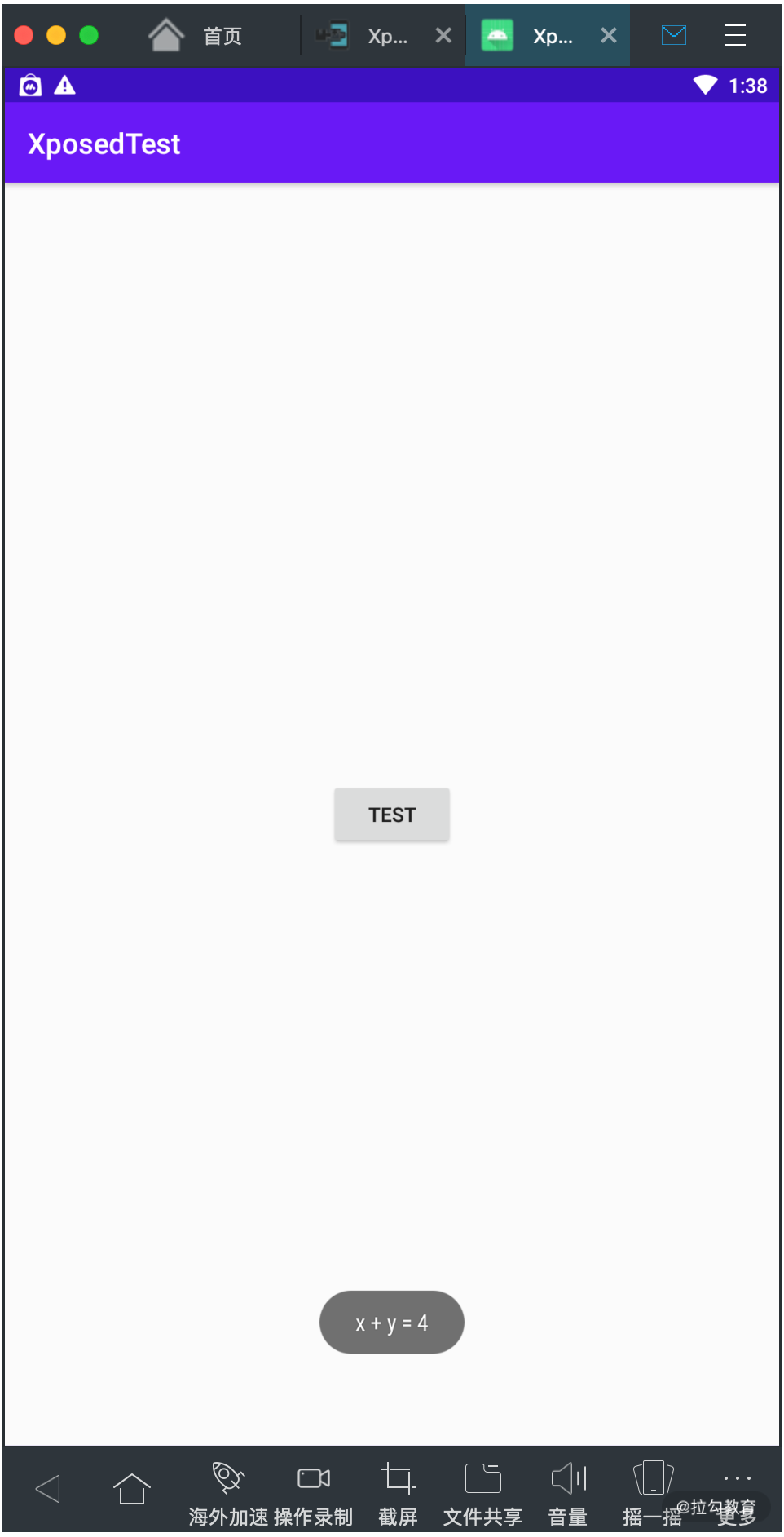


其内容就把 Hook 的这个类的路径写好就好了，内容如下：

```
com.germey.xposedtest.HookMessage
```

好，这样保存之后，Xposed 就能自动读取这个 xposed_init 文件来执行我们自定义的 Hook 逻辑了。
最后，我们就来重新运行看下效果吧。记得安装完成之后重启一下 Xposed，否则是没有效果的。

重启模块之后，点击 TEST 按钮，可以看到就出现了如下效果，如图所示：

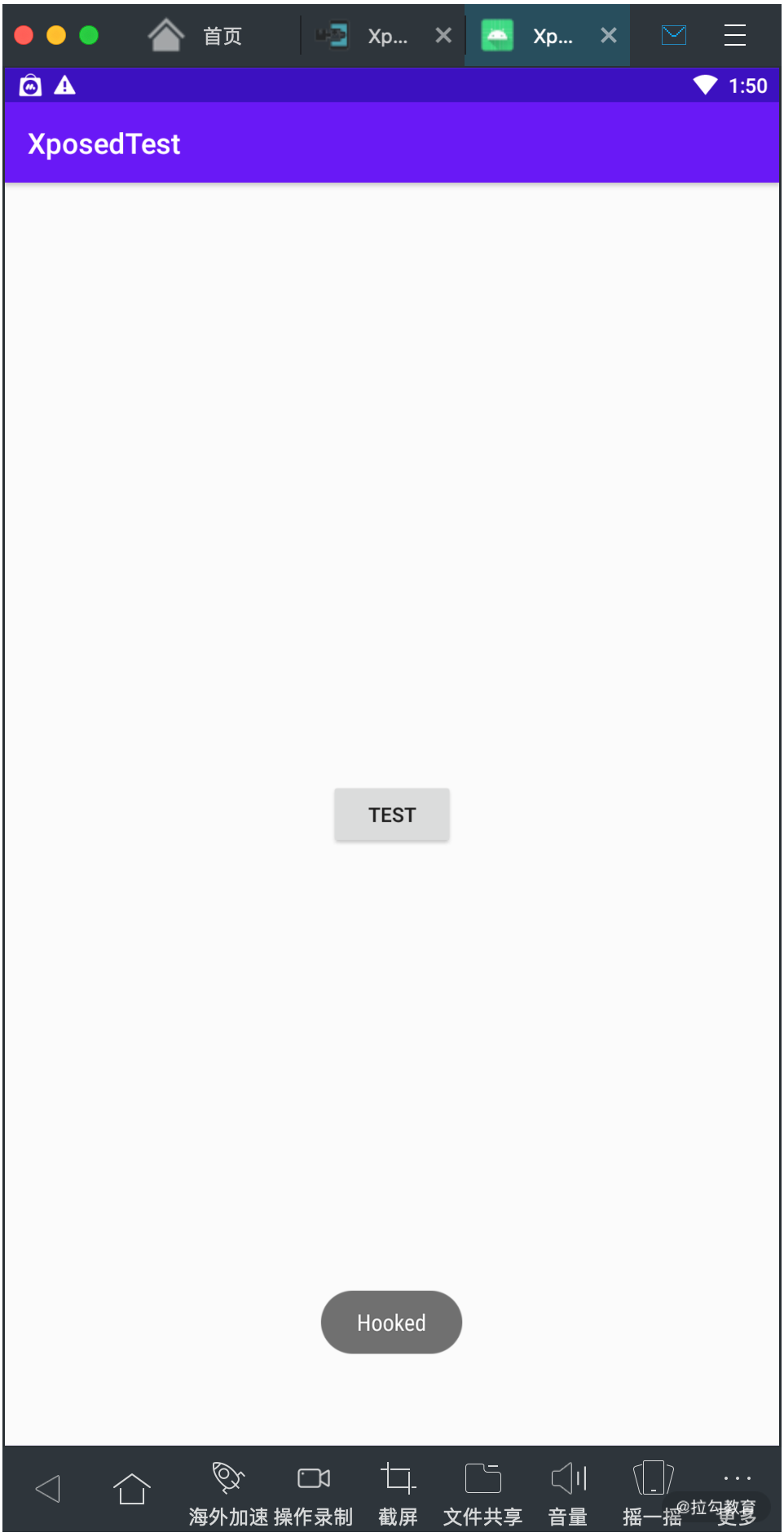


这里可以看到，最后的运行效果就不一样了，出现了“x + y = 4”的这个现象，这说明通过 beforeHookedMethod 的定义，我们成功把 args 的第一个参数，也就是 x 修改成了 2，而第二个参数没有修改，还是 2，最后就相当于 showMessage 调用之前，两个参数就被修改成了 2 和 2，最后答案就是 4 了。这下我们就体会到了 beforeHookedMethod 的用法了。

刚才我们是用了 beforeHookedMethod 来实现了参数替换的效果，接下来我们再来体会一下 afterHookedMethod 的用法，它可以对方法的返回结果进行后处理，比如这里我们把 afterHookedMethod 修改为如下内容：

```
protected void afterHookedMethod(MethodHookParam param) throws Throwable {
    XposedBridge.log("Called afterHookedMethod");
    param.setResult("Hooked");
}
```

这里我们增加了 param 的 setResult 方法的调用，利用它我们可以直接将方法的返回值修改掉。
重新运行这个模块，然后重启手机，同样地还是点击 TEST 按钮，这时候我们发现其结果就变成了如下内容，如图所示：



可以看到最后方法的返回结果被修改了，这正是 afterHookedMethod 所起的作用。

由此，我们通过 beforeHookedMethod 和 afterHookedMethod 的修改可以实现 showMessage 在调用前和调用后的修改。

好，最后我们再来看下日志，打开 Xposed Installer 的日志页面，可以看到内容如图所示：

可以看到这里就输出了我们用 XposedBridge 的 log 方法输出的内容。

OK，到此为止我们就实现了 Xposed 的 Hook 逻辑了，通过这个案例你应该就能体会到 Xposed 的效用了。

真实 App 的修改

好，下面我们再来使用一个真实的 App 为样例来进行一下 Hook 操作吧，我们来实现通过 Hook App 的某个方法来达到修改 App 的运行效果的作用。

这个 App 的下载地址为：https://app1.scrape.center/。

下载完成之后，我们安装一下，可以看到不断下拉的过程中会有电影数据加载出来，如图所示。

通过观察我们可以发现每刷新一次，就会加载出 10 条数据，一共 100 条。

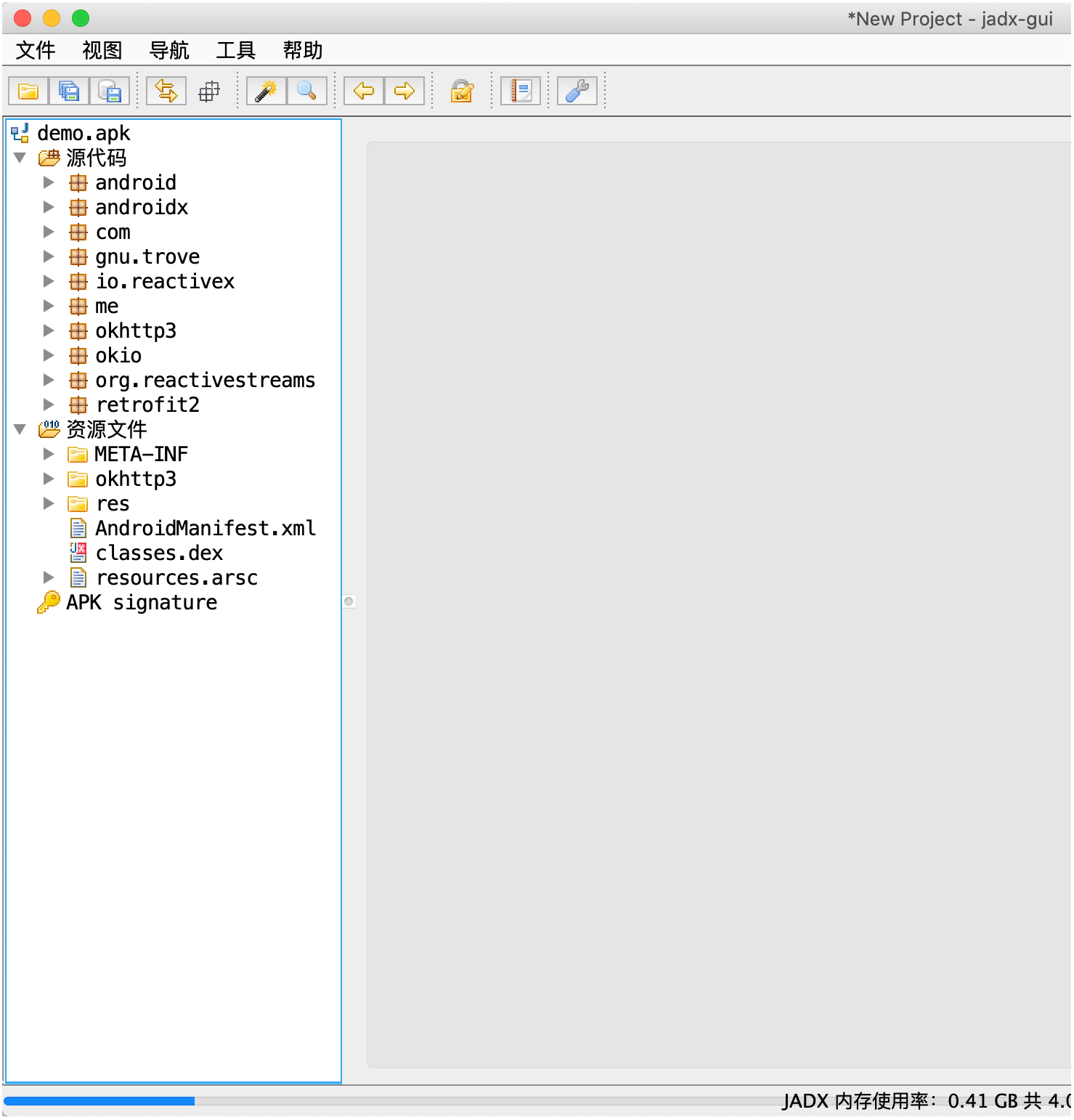
那这个加载数量我们可以通过 Xposed 修改吗？比如修改成一次加载 5 条能做到吗？当然是可以的。

但要修改的话，我们得需要知道 App 的一些逻辑，或者它的一些包路径，方法名之类的内容。

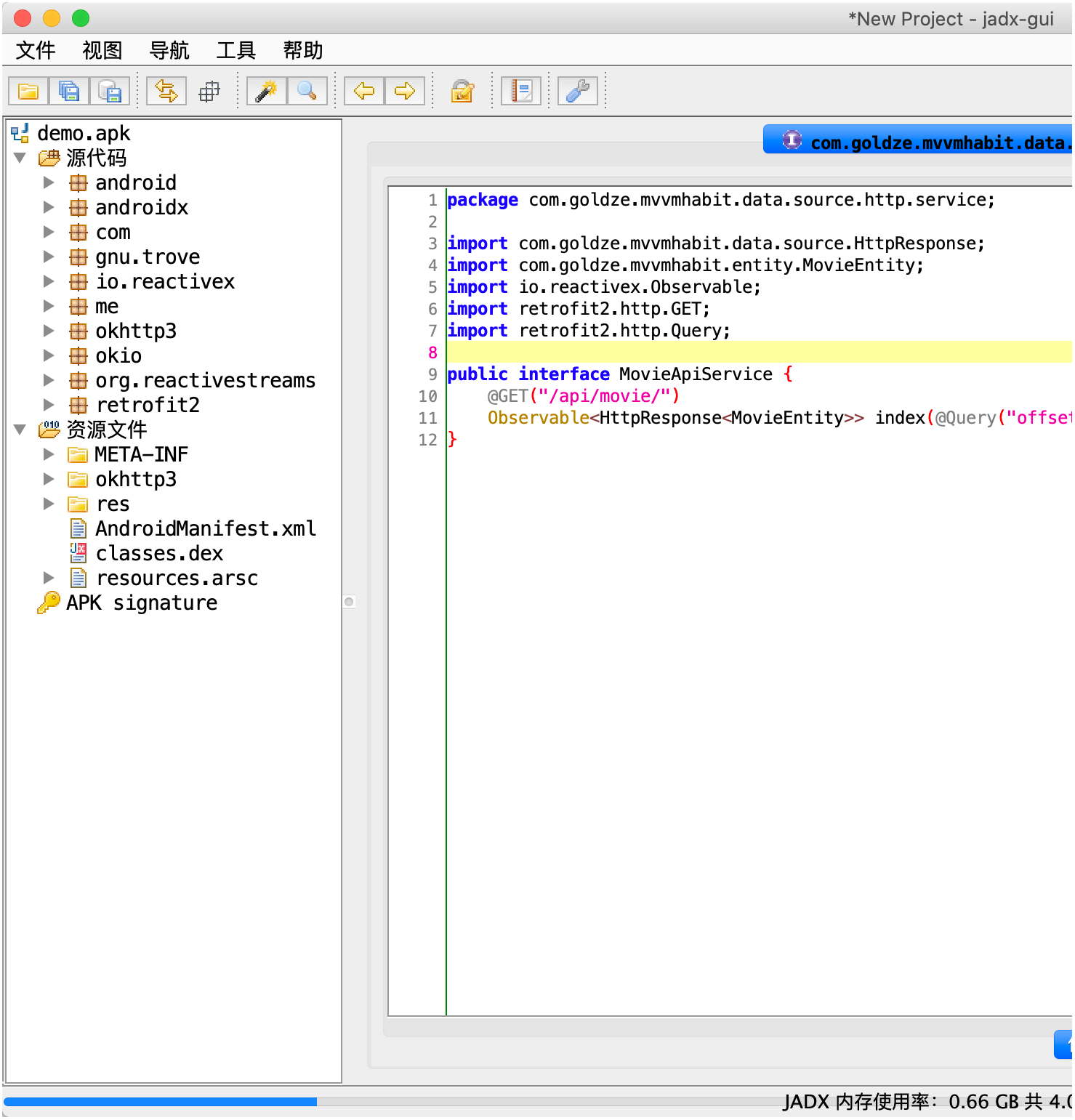
那这个怎么来处理呢？这时候我们可能就需要对 App 的安装包进行反编译了。

由于这个 App 比较简单，没有设置加固，所以这里我们直接使用 jadx 或 jadx-gui 就可以把源码反编译出来了。

这里我使用的是 jadx-gui，我们打开 jadx-gui，然后直接打开 apk 文件，就可以看到反编译后的结果了，如图所示：



通过 App 的一些抓包操作可以找到 API 请求的 URL Path 为 /api/movie，我们使用 jadx-gui 搜索下这个入口，就能找到其原始的一些定义，如图所示：



好，这时候呢我们可以大体定位到一些数据的操作就在 com.goldze.mvvmhabit.data 这个路径下，我们打开看看这里，同时借助于一些方法的交叉引用分析，可以大致分析到此处 index 方法的调用是在 com.goldze.mvvmhabit.data.MainRepository 里的 index 方法里。

实现如下：

```
public Observable<HttpResponse<MovieEntity>> index(int page, int limit) {
    return this.mHttpDataSource.index(page, limit);
}
```

这里其实就很清楚了，它接受两个参数，一个是 page，一个 limit，最后调用了 mHttpDataSource 的 index 方法获得了 Response 的结果，并封装成一个 Observable 对象。
好，那我们就来 Hook 一下这个方法吧。我们还是在刚才的 XposedTest 这个项目下，另外新建一个 class，名字叫作 HookAPI，同时还是按照之前的方法来修改一下包名、类名、方法名等等，内容修改如下：

```
package com.germey.xposedtest;
import de.robv.android.xposed.IXposedHookLoadPackage;
import de.robv.android.xposed.XC_MethodHook;
import de.robv.android.xposed.XposedBridge;
import de.robv.android.xposed.XposedHelpers;
import de.robv.android.xposed.callbacks.XC_LoadPackage;
public class HookAPI implements IXposedHookLoadPackage {
    public void handleLoadPackage(XC_LoadPackage.LoadPackageParam loadPackageParam) throws Throwable {
        if (loadPackageParam.packageName.equals("com.goldze.mvvmhabit")) {
            XposedBridge.log("Hooked com.goldze.mvvmhabit Package");
            Class clazz = loadPackageParam.classLoader.loadClass(
                    "com.goldze.mvvmhabit.MainRepository");
```

```
            XposedHelpers.findAndHookMethod(clazz, "index", int.class, int.class, new XC_MethodHook() {
                protected void beforeHookedMethod(MethodHookParam param) throws Throwable {
                    XposedBridge.log("Called beforeHookedMethod");
                    param.args[1] = 5;
                    XposedBridge.log("Changed args 0 to " + param.args[0]);
                }
                protected void afterHookedMethod(MethodHookParam param) throws Throwable {
                    XposedBridge.log("Called afterHookedMethod");
                }
            });
        }
    }
}
```

这里我们就利用 beforeHookedMethod 方法把 args 的第二个参数修改成了 5，这样 limit 每次调用就不再是 10 了，而是 5，照理来说每次加载就会返回 5 条数据了。
另外我们还需要在 xposed_init 里面定义好这个入口文件，内容修改如下：

```
com.germey.xposedtest.HookMessage
com.germey.xposedtest.HookAPI
```

好，接下来重新安装这个 Xposed 模块，然后重启手机，接着再运行这个 Xposed 模块。
下面我们再重新打开刚才被 Hook 的 App，看到如下的加载效果，如图所示：

可以看到这里加载的数据就变成了 5 条，而不再是原来的 10 条，每次上拉刷新也都是 5 条数据了。

到此，我们就通过 Xposed 通过 Hook 的方式修改了 App 的运行效果，而没有去修改原始 App 的任何一行代码。

Xposed API

最后再来看下 Xposed 提供的 API。

其实刚才所说的 Hook 操作只是 Xposed 的其中一个 API，即 findAndHookMethod 的用法。我们可以查看 Xposed 的所有的 API，链接为 https://api.xposed.info/reference/de/robv/android/xposed/XposedHelpers.html。

在这里我们可以看到有如下的一些 API，这里简单列举一下。

- callStaticMethod：调用静态方法。
- findAndHookConstructor：查找并 Hook 构造方法。
- findClassIfExists：查找某个类是否存在。
- findField：获取成员变量。

有很多 API 是有类似或重合的功能的，这里就不再一一列举了，感兴趣的话你可以看官方的文档说明。

另外也非常推荐你研究一下 Xposed 里面的各个 package 的用法，API 见 https://api.xposed.info/reference/de/robv/android/xposed/package-summary.html。

另外也欢迎你多去研究一些优秀的 Xposed 模块，比如 https://devsjournal.com/best-xposed-modules.html 里面列举了几款很受欢迎的 Xposed 模块，另外还有 Xposed 中文站 http://xposed.appkg.com/，你可以找一些优秀的模块的源码来研究一下，收获会非常大的。

总结

本文我们介绍了 Xposed 的基本理念，并通过案例来实现了 Xposed Hook App 的流程。

Xposed 的功能非常强大，利用它，App 尽在我们掌控之中，为所欲为不再是奢望。

本节代码：https://github.com/Python3WebSpider/XposedTest。

参考链接

- https://forum.xda-developers.com/showthread.php?t=3034811