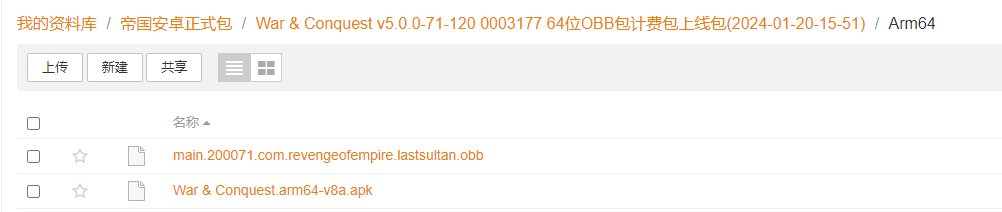
# OBB一些特殊逻辑

## 打包

帝国项目由于是之前提交的GP商店，而且包体打包超过了100MB，因此采用的是APK+OBB模式去提交，所以在可以遇见的未来，帝国包体依然要使用OBB方式，而非ABB格式。

使用OBB模式提交之后，由于包体大小依然很大，在优化包体的时候，采用了Split Architecture这样一种设计。这种方式可以在打包的时候，把代码分离城Armv7和Arm64两种架构，然后提交到GP的时候，以同一个包名，但是不同的架构和BuildCode提交。

下面是一个具体的例子：



从上面的图里可以看出，采用这种方式打包的话，Unity会生成两个APK，分别包含自身架构的代码，而非以前一样两个架构的代码都在其中，这样APK大小基本可以减少30MB左右。

而OBB文件一般都是AssetBundle的合集，本质上是不需要关心代码的架构版本的，所以其实在打包的时候只生成一个，但是客户端由于OBB文件需要包含BuildCode，所以复制了一份。

Unity在选择Split Architecture的时候，会默认Armv7的BuildCode+10000，而Arm64的BuildCode+20000。

这里有一个可能的隐患，就是如果正式包也是自动出包，有可能超过这个上限，导致一些问题，不过目前我们的项目中不会有这个问题；

## 游戏逻辑检测

在使用OBB模式去打开加载游戏的时候，虽然Google说他们会保证玩家下载完游戏一定是APK+OBB都完整的模式才能启动游戏，但是统计和经验告诉我们，还是有很多的办法能够删掉或者OBB不存在的情况下启动游戏的。

因此，我们在使用OBB打包的模式下，会新增一个OBBCheck场景，这个场景用来检测OBB是否存在，同时，在一些深度定制的手机操作系统上，如果没有访问SDCard的权限，有时候也没有办法检测到OBB是否存在，因此需要加上这样一个逻辑；

工程中的场景名就是OBBCheck，同时自动打包会处理是否要加入这个场景。

具体的代码叫OBBChecker，这个类，里面用到了一个第三方的Android权限管理插件，去检测OBB是否存在。

检测方式很简单，就是用Resource.Load加载一个OBB中的资源，如果加载成功，表示OBB是在的，如果加载不到，去判断权限。

加载成功就直接切换场景到Main场景，执行我们的代码逻辑。

## 历史

古早版本的Unity，在加载OBB的时候还要写一个特殊的代码，才能关联OBB，现在的Unity版本已经不需要了，直接自动关联，当成一个默认的来处理就行了。