

Table of Contents

简介	1.1
第一章 编译安装	1.2
第二章 应用案例	1.3
资源逆向	1.3.1
贴图	1.3.1.1
模型	1.3.1.2
资源防护	1.3.2
资源引发崩溃	1.3.3
资源引用丢失	1.3.4
资源显示异常	1.3.5
资源差异比对	1.3.6
资源编辑	1.3.7
第三章 命令详解	1.4
savetree	1.4.1
gtt	1.4.2
dump	1.4.3
list	1.4.4
size	1.4.5
scanref	1.4.6
scantex	1.4.7
savefbx	1.4.8
savetex	1.4.9
saveta	1.4.10
saveobj	1.4.11
objref	1.4.12
mono	1.4.13
cmpref	1.4.14
cmpxtl	1.4.15
cmphash	1.4.16
external	1.4.17
rmmtree	1.4.18
lua	1.4.19
edit	1.4.20
第四章 进阶开发	1.5
项目架构	1.5.1

贴图

文件解析	1.5.2
对象序列化	1.5.3
命令系统	1.5.4
文件系统	1.5.5
LUA绑定	1.5.6

简介

愿景

abtool旨在提供基于Unity资源封装格式 AssetBundle 的C++开发框架以及预置工具集合，方便针对资源做任意的检测、编辑以及资源问题定位。

开发背景

笔者2020年初加入使命召唤手游项目，这是一款偏向内容运营的高品质手机游戏，有非常多的ab资源，从笔者加入项目时的1G左右增加到现在的5G左右，未来可以预期持续的增长。伴随着资源量的增加，资源相关的崩溃、显示问题越来越多，定位解决这些问题是一个常态化的工作。

笔者的工作内容主要是负责版本以及资源发布，在abtool出现之前定位这些问题非常困难，特别是资源引起的游戏崩溃问题，困难主要表现为：

1. 需要稳定的重现方法，通常需要花很多时间去摸索
2. 需要增加运行时调试信息来辅助判断，甚至需要真机调试，通常需要重新构建

处理这类问题遇到最大的麻烦是：即使解决了崩溃，你仍然无法确定是否还有其他类似的资源崩溃问题。我们有7000个左右的ab文件，排查所有的资源问题犹如大海捞针，每次发版本都是战战兢兢、如履薄冰，并且经常通宵攻坚，但也不总是有效，这种情况下只能延迟版本发布。

鉴于笔者丰富的工具开发经验，经历几次通宵后，笔者决定通过工具化寻求突破，最终的开发进度以及使用效果也是十分喜人，截文档撰写日起已有20多个内置命令，它们均是在解决资源问题过程中逐渐增加和完善的，具有很强的实用性。当然，通过后续的章节了解熟悉后，您也可以轻易开发出专属于您的工具命令。

文档更新

如果您需要访问最新的文档内容，建议您[查阅在线文档版本¹](#)，或者手动[下载当前文档的最新版本²](#)。

由于文档撰写比较匆忙，难免有所谬误，请多包涵。同时，也欢迎大家[Fork](#)笔者的[文档仓库³](#)并提交相应的PR，让我们一起来完善它，感激不尽！

¹ <https://larryhou.github.io/abtool-gitbook/> ↵

² <https://larryhou.github.io/abtool-gitbook/book.pdf> ↵

³ <https://github.com/larryhou/abtool-gitbook/> ↵

第一章 编译安装

由于笔者日常工作环境中很少使用其他操作系统，暂时abtool只支持针对macOS系统平台的编译运行，感兴趣的朋友可以自行适配其他系统平台，涉及平台差异的内容主要是以下几个外部链接库：

- libreadline.tbd
- libfbxsdk.a
 - Foundation.framework
 - libiconv.tbd
 - libxml2.tbd
 - libz.tbd

abtool源码基于C++14标准库实现，理论上处理好这些编译依赖问题即可完成适配。

首次编译abtool

如果您不是第一次使用abtool，也就是说您手上已经有了一份abtool工具，那么可以跳过该步骤，直接进行下一步操作。

1. Xcode编译

打开Xcode，使用组合键 $\text{⌘}+\text{B}$ 即可进行源码编译，之后编译可以在终端环境或者shell脚本里面随意使用，这是生成abtool工具的最简单的方式，需要注意的是请确保目标目录 `/usr/local/bin` 已被预先创建。

2. CMake编译

执行如下脚本，即可在 `build/bin` 目录得到abtool命令行工具。

```
# 当前cd目录为工程根目录
mkdir build
cd build
cmake ..
cmake --build .
```

生成TypeTree数据

TypeTree记录了资源对象数据的序列化信息，收集TypeTree的目的是为了把Unity的类型信息集成到abtool工具里面，只有这样abtool才有可能实现它的功能。

为了让大家快速体验整个工具编译过程，笔者在工程doc目录准备了 `QuickStart.unitypackage` 资源包，现在您只需要新建一个Unity工程，然后导入所有资源，通过Unity菜单 `abtool/Build Asset Bundles` 即可快速生成包含了TypeTree数据的ab文件，该资源包包含了能够让abtool源码正常编译的最小集合，具体来说是，资源里面包含了以下编译必须的资源对象类型：

- GameObject
- RectTransform
- Transform
- TextAsset
- Texture2D
- Cubemap

- Material
- Shader
- SpriteRenderer
- SkinnedMeshRenderer
- MeshRenderer
- ParticleSystemRenderer
- LineRenderer
- TrailRenderer
- MeshFilter
- Animator
- Mesh

如果您现有的项目资源已经覆盖了以上资源类型，那么可以放心使用abtool收集相应的TypeTree数据。然而QuickStart资源包只是覆盖了最小集合的资源类型，如果需要最大限度发挥abtool的功效，笔者强烈建议您扫描尽可能多的ab文件，从而可以收集到尽可能多的Unity类型数据，这样会让您定位资源问题更加得心应手，相信您在后续的日常使用中会深刻明白这一点。

```
# doc/resources目录存储了QuickStart资源编译的iOS/Android双平台的ab文件
cd doc/resources
# 由于QuickStart生成的ab文件后缀为ab，所以可以通过'*.ab'进行文件匹配
# 请根据实际项目的ab文件后缀做适当修改
find . -iname '*.ab' | xargs abtool savetree -a types.tte
```

上述脚本通过 `find` 命令查找所有的ab文件，并把这些文件通过 `xargs` 透传给abtool工具去处理，最终会在当前目录生成 `types.tte` 文件（当然也可以通过 `savetree` 的 `-a` 参数指定其他保存目录），这就是我们需要的Unity类型数据。

生成对象序列化代码

`types.tte` 是个二进制文件，把它转换成C++代码才能最终为abtool所用，通过下面这行命令可以轻松完成这个任务，整个过程就好比使用 `protoc` 编译 `*.proto` 文件一样。

```
# 当前cd目录为工程根目录
abtool gtt -a doc/resources/types.tte -o abtool/assetbundles/unity
```

由于上面的脚本是在工程根目录执行，并且代码的输出目录为 `abtool/assetbundles/unity`，所以当脚本执行完成后工程的代码就得到了更新。

最终编译abtool

通过上一步骤我们修改了Unity资源对象的序列化代码，所以还需要再次编译，这样我们就最终得到了功能完备的abtool，通过后续的章节可以逐渐窥探它强大的威力。

什么情况下需要重新编译abtool?

1. 升级了Unity版本
2. 修改了Unity源码里面涉及资源对象的序列化的代码
3. 修改了 `AssetBundleArchive` 容器存储结构
4. 修改了 `SerializedFile` 存储结构
5. 如果需要abtool正常处理所有 `MonoBehaviour` 组件数据，那么您需要定期编译abtool，不过我们大部分情况下并不关心这部分数据。

第二章 应用案例

对于大多数来说，大家可能更希望abtool能有一些现实的应用场景，这样拿到工具后就可以着手做一些资源侧的检测优化工作，在本章节会列举几个常见的案例，通过案例来了解工具的使用。

资源逆向

资源逆向既是通常意义的资源反编译，也就是从ab文件里面提取出来方便浏览的资源。鉴于abtool集成了项目所有资源类型的序列化信息，理论上abtool可以反编译任意资源，但是实现情况是反编译所有资源有代价，并且也不是所有资源都是我们关心的，所以笔者暂时只实现了有限几个但高频使用的资源类型的反编译，比如：贴图、模型、Shader、二进制文件等。从ab文件反编译资源并非abtool的开发初衷，但是abtool的实现原理注定它可以轻松支持资源逆向目的。

为了增加abtool资源逆向功能的一般性，笔者选择在本案例中使用第三方线上运营游戏来做演示，大家可以依照步骤得到相同的结果。

声明：本案例使用的方法以及由该方法得到的资源仅用于学习交流，请勿用于其他非法目的，否则后果自负。

下载安装包

我们可以通过Google搜索 战歌竞技场 apk，然后下载相应的apk安装包。笔者使用的版本是 1.5.151，点击[链接¹](#)可直接进行下载，但是鉴于cdn链接的时效性，该文档并不保证该下载链接总是有效可用，如果下载失败请自行从Google搜索结果里面寻找其他链接进行下载。

战歌竞技场 apk

全部 视频 图片 新闻 更多

找到约 337,000 条结果 (用时 0.38 秒)

<https://chessrush.qq.com> › zlkdatasys › mct › play ▾

战歌竞技场腾讯游戏下载

即将启动. 启动或安装游戏《**战歌竞技场**》. 此浏览器不支持启动游戏请尝试其他浏览器打开此页面. 已安装, 立即启动 未安装, 立即下载. iOS 下载**Android** 下载.

<https://www.ruan8.com> › zhuanti › down ▾

战歌竞技场下载_战歌竞技场apk下载_战歌竞技场版本 ... - 软吧

软吧下载为广大玩家带来**战歌竞技场**下载, 提供各种类型最新版本下载给玩家, 汇总了当前最热门的游戏版本并提供有效下载地址.

解压ab资源

首先，用unzip命令行查看apk资源列表

```
unzip -l 10040714_com.tencent.hjqzqgame_a960942_1.5.151_j2e715.apk
```

从日志里面我们发现 assets/AssetBundles 目录存储了ab资源，现在我们可以继续用unzip提取ab资源

```
unzip -o 10040714_com.tencent.hjqzqgame_a960942_1.5.151_j2e715.apk 'assets/AssetBundles/*'  
cd assets
```

贴图

这样我们就得到了apk里面所有的ab资源，在后续资源资源逆向案例中如无特殊说明，均把解压出来的assets目录作为工具的工作空间，并默认使用AssetBundles目录里面的ab资源做演示。

编译

我们在前面章节已经学习了工具编译过程，由于该案例用到的ab资源属于某个特定Unity版本，所以需要依据编译流程手机资源TypeTree并重新编译abtool，否则您将无法正常通过abtool进行后续的资源逆向操作。

```
find . -iname '*.god' | xargs abtool savetree
```

¹ [https://dlid4.myapp.com/myapp/1109006800/cos.release-75620/10040714_com.tencent.hjqgame_a960942_1.5.151_j2e715.apk ↵](https://dlid4.myapp.com/myapp/1109006800/cos.release-75620/10040714_com.tencent.hjqgame_a960942_1.5.151_j2e715.apk)

贴图

选择ab文件

首先我们需要找到一个包含贴图资源的ab文件，如果您不确定是哪个ab满足要求，那么可以使用 `list` 命令收集所有进包的资源路径，然后反过来从贴图资源的路径查找相应的ab文件。

```
find AssetBundles -iname '*.*.god' | xargs abtool list -r
```

从结果里面我们选择 `artresource_captainpbr_captain_202.god` 作为演示资源。

```
[095/123] assets/artresource/captainpbr/captain_202/textures/captain_202103_suit_m.tga Texture2D i:13067992522968025633
[096/123] assets/artresource/captainpbr/captain_202/textures/captain_2021_body_n.tga Texture2D i:3883958129370872108
[097/123] assets/artresource/captainpbr/captain_202/textures/captain_2021_stuff_n.tga Texture2D i:11914906896687225448
[098/123] assets/artresource/captainpbr/captain_202/textures/captain_2021_suit_n.tga Texture2D i:17698705126682618081
[099/123] assets/artresource/captainpbr/captain_202/textures/captain_202201_body_a.tga Texture2D i:2758819737572619551
[100/123] assets/artresource/captainpbr/captain_202/textures/captain_202201_body_b.tga Texture2D i:10388116232065757511
[101/123] assets/artresource/captainpbr/captain_202/textures/captain_202201_body_d.tga Texture2D i:10679769261937420922
[102/123] assets/artresource/captainpbr/captain_202/textures/captain_202201_body_m.tga Texture2D i:7592280451348180021
[103/123] assets/artresource/captainpbr/captain_202/textures/captain_202201_suit_a.tga Texture2D i:8152769956827659214
[104/123] assets/artresource/captainpbr/captain_202/textures/captain_202201_suit_b.tga Texture2D i:14756172379838531154
[105/123] assets/artresource/captainpbr/captain_202/textures/captain_202201_suit_d.tga Texture2D i:18218461557286871300
[106/123] assets/artresource/captainpbr/captain_202/textures/captain_202201_suit_m.tga Texture2D i:13791270348976128462
[107/123] assets/artresource/captainpbr/captain_202/textures/captain_202202_body_b.tga Texture2D i:9804010988900091748
[108/123] assets/artresource/captainpbr/captain_202/textures/captain_202202_body_d.tga Texture2D i:10986088950313902645
[109/123] assets/artresource/captainpbr/captain_202/textures/captain_202202_body_m.tga Texture2D i:10320767474896706012
[110/123] assets/artresource/captainpbr/captain_202/textures/captain_202202_suit_b.tga Texture2D i:2592695568612982683
[111/123] assets/artresource/captainpbr/captain_202/textures/captain_202202_suit_d.tga Texture2D i:3746720729245558176
[112/123] assets/artresource/captainpbr/captain_202/textures/captain_202202_suit_m.tga Texture2D i:17681954939584535757
[113/123] assets/artresource/captainpbr/captain_202/textures/captain_2022_body_n.tga Texture2D i:11607926898818107768
[114/123] assets/artresource/captainpbr/captain_202/textures/captain_2022_suit_n.tga Texture2D i:3533547237365748242
[115/123] assets/artresource/captainpbr/captain_202/textures/captain_202301_body_a.tga Texture2D i:5710822887162919953
[116/123] assets/artresource/captainpbr/captain_202/textures/captain_202301_body_b.tga Texture2D i:11415350406488400628
[117/123] assets/artresource/captainpbr/captain_202/textures/captain_202301_body_d.tga Texture2D i:16515969976440694086
[118/123] assets/artresource/captainpbr/captain_202/textures/captain_202301_body_m.tga Texture2D i:86093989447636964
[119/123] assets/artresource/captainpbr/captain_202/textures/captain_202301_body_n.tga Texture2D i:9829224366313559506
[120/123] assets/artresource/captainpbr/captain_202/textures/captain_202301_suit_b.tga Texture2D i:1887866232146701256
[121/123] assets/artresource/captainpbr/captain_202/textures/captain_202301_suit_d.tga Texture2D i:8568020777195722661
[122/123] assets/artresource/captainpbr/captain_202/textures/captain_202301_suit_m.tga Texture2D i:17121284964933108453
[123/123] assets/artresource/captainpbr/captain_202/textures/captain_202301_suit_n.tga Texture2D i:18012581096705529419
```

提取贴图资源

通过abtool的 `savetex` 命令可以一次性保存ab文件里面所有的贴图资源，默认输出到当前目录的 `_textures` 目录，也可以添加 `--output` 参数指定其他存放目录。

```
abtool savetex AssetBundles/Android/artresource_captainpbr_captain_202.god
```

贴图

```
|LARRYHOU-MC8:demo larryhou$ abtool-cr savetex AssetBundles/artresource_captainpbr_captain_202.god
[0] AssetBundles/artresource_captainpbr_captain_202.god
--textures/Captain_202202_body_b.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_202301_body_n.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_202102_body_d.512x512/etc2_rgba8.tex =262,144 256.00K
--textures/Captain_202202_body_m.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_202201_body_b.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_202201_body_d.512x512/etc2_rgba8.tex =262,144 256.00K
--textures/Captain_202103_body_d.512x512/etc2_rgba8.tex =262,144 256.00K
--textures/Captain_202102_suit_b.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_202202_body_d.512x512/etc2_rgba8.tex =262,144 256.00K
--textures/Captain_202301_body_b.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_2022_body_n.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_2021_stuff_n.64x64/etc_rgb4.tex =2,048 2.00K
--textures/Captain_202103_body_m.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_202101_body_d.512x512/etc2_rgba8.tex =262,144 256.00K
--textures/Captain_202103_suit_m.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_202101_suit_a.64x64/etc_rgb4.tex =2,048 2.00K
--textures/Captain_202101_suit_d.512x512/etc2_rgba8.tex =262,144 256.00K
--textures/Captain_202101_suit_b.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_202201_suit_m.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_202101_stuff_m.64x64/etc_rgb4.tex =2,048 2.00K
--textures/Captain_202201_suit_b.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_202101_stuff_b.64x64/etc_rgb4.tex =2,048 2.00K
--textures/Captain_202101_suit_m.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_202102_body_m.512x512/etc_rgb4.tex =131,072 128.00K
--textures/Captain_202301_body_d.512x512/etc2_rgba8.tex =262,144 256.00K
--textures/Captain_202103_suit_d.512x512/etc2_rgba8.tex =262,144 256.00K
--textures/Captain_202301_suit_m.256x256/etc_rgb4.tex =32,768 32.00K
--textures/Captain_202101_body_a.64x64/etc_rgb4.tex =2,048 2.00K
```

需要说明的是： savetex 保存的贴图有着固定命名规范，其格式为[filename].[宽]x[高].[贴图格式].tex，除了 filename，其余文件名内容是不能修改的，否则在接下来的贴图转码操作会失败。

贴图格式转换

从上一步骤得到的贴图都是 *.tex 格式的文件，并非用普通图片浏览工具可以直接查看文件格式，它们被做了特殊编码编码以便GPU渲染时可以被正常读取，所以还需要做贴图转码。在工程根目录放置了python脚本工具 textool.py，它可以批量地把 *.tex 文件转换成项目中常见的 *.tga 文件。

贴图

```
import re, struct
from tex2img import decompress_astc, decompress_etc, decompress_pvrtc
from PIL import Image

def main():
    import sys
    pattern = re.compile(r'[^/]+(\.\d+x\d+)\.([^.]+\.\tex$)')
    for filename in sys.argv[1:]:
        match = pattern.search(filename)
        if not match: continue
        # print('">>> {}'.format(filename))
        fp = open(filename, 'rb')
        texture_size = [int(x) for x in match.group(1).split('x')]
        texture_format = match.group(2)
        mode = 'RGBA'
        if texture_format.startswith('etc_'):
            image = decompress_etc(fp.read(), texture_size[0], texture_size[1], 0)
            mode = 'RGB'
        elif texture_format.startswith('etc2_'):
            image = decompress_etc(fp.read(), texture_size[0], texture_size[1], 3 if texture_format.startswith('etc2') else 0)
        elif texture_format.startswith('astc_rgb'):
            block_size = [int(x) for x in texture_format.split('_')[1].split('x')]
            image = decompress_astc(fp.read(), texture_size[0], texture_size[1], block_size[0], block_size[1])
        elif texture_format.startswith('pvrtc'):
            # https://github.com/powervr-graphics/Native_SDK/blob/3f88b0f3735774ab9fb718da0aeadd06acf68d21/fr
            image = decompress_pvrtc(fp.read(), texture_size[0], texture_size[1], 0 if texture_format[-1] == 'p' else 1)
        elif texture_format.startswith('rgba32'):
            image = fp.read()
        elif texture_format.startswith('rgb24'):
            image = fp.read()
            mode = 'RGB'
        elif texture_format.startswith('rgb565'):
            width, height = texture_size
            image = bytearray(width * height * 3)
            index = 0
            for r in range(height):
                for c in range(width):
                    v, = struct.unpack('<H', fp.read(2))
                    image[index+0] = (v >> 11 & 0x1F) * 255 // 0x1F # red
                    image[index+1] = (v >> 5 & 0x3F) * 255 // 0x3F # green
                    image[index+2] = (v >> 0 & 0x1F) * 255 // 0x1F # blue
                    index += 3
            image = bytes(image)
            mode = 'RGB'
        elif texture_format.startswith('rgba4444'):
            width, height = texture_size
            image = bytearray(width * height * 4)
            index = 0
            for r in range(height):
                for c in range(width):
                    v, = struct.unpack('<H', fp.read(2))
                    image[index+0] = (v >> 12 & 0xF) * 255 // 0xF # red
                    image[index+1] = (v >> 8 & 0xF) * 255 // 0xF # green
                    image[index+2] = (v >> 4 & 0xF) * 255 // 0xF # blue
                    image[index+3] = (v >> 0 & 0xF) * 255 // 0xF # alpha
                    index += 4
            image = bytes(image)
        elif texture_format.startswith('alpha8'):
            image = fp.read()
            mode = 'L'
        else: continue
        result = Image.frombytes(mode, tuple(texture_size), image, 'raw')
        savename = re.sub(r'(\.[^.]+)\{3\}$', '', filename) + '.tga'
        result.save(savename)
        print('+ {} => {}'.format(filename, savename))
```

贴图

```
if __name__ == '__main__':
    main()
```

在使用前建议把textool放到 /usr/bin/local/ 目录下，这样好处是不用每次都用一个很长的路径来访问这个工具了。

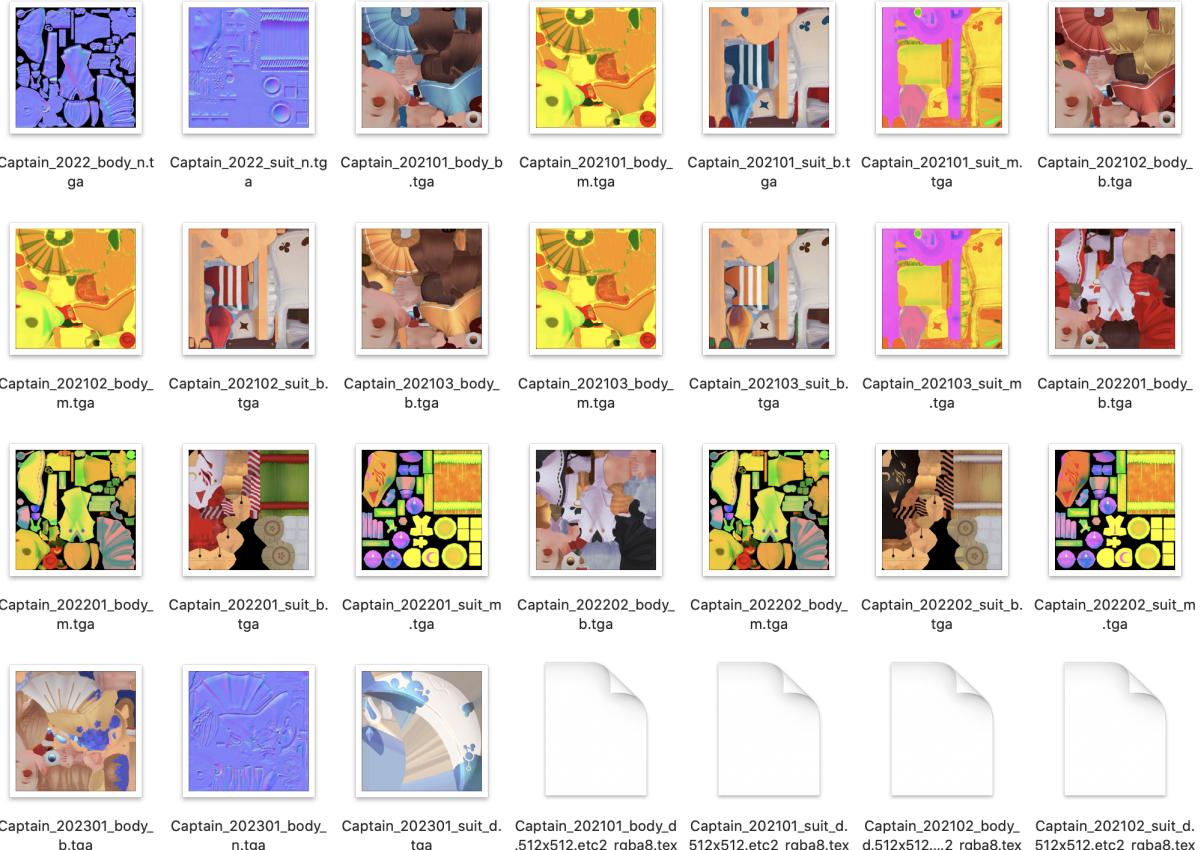
```
cp -fv textool.py /usr/local/bin/textool
```

接下来通过textool转换 *.tex 贴图格式，转换后的图片文件存储在源 *.tex 文件的同级目录。

```
textool __textures/*.tex
```

```
[LARRYHOU-MC8:demo larryhou$ textool __textures/*.tex
+ __textures/Captain_202101_body_a.64x64/etc_rgb4.tex => __textures/Captain_202101_body_a.tga
+ __textures/Captain_202101_body_b.512x512/etc_rgb4.tex => __textures/Captain_202101_body_b.tga
+ __textures/Captain_202101_body_d.512x512/etc2_rgba8.tex => __textures/Captain_202101_body_d.tga
+ __textures/Captain_202101_body_m.512x512/etc_rgb4.tex => __textures/Captain_202101_body_m.tga
+ __textures/Captain_202101_stuff_b.64x64/etc_rgb4.tex => __textures/Captain_202101_stuff_b.tga
+ __textures/Captain_202101_stuff_d.64x64/etc_rgb4.tex => __textures/Captain_202101_stuff_d.tga
+ __textures/Captain_202101_stuff_m.64x64/etc_rgb4.tex => __textures/Captain_202101_stuff_m.tga
+ __textures/Captain_202101_suit_a.64x64/etc_rgb4.tex => __textures/Captain_202101_suit_a.tga
+ __textures/Captain_202101_suit_b.512x512/etc_rgb4.tex => __textures/Captain_202101_suit_b.tga
+ __textures/Captain_202101_suit_d.512x512/etc2_rgba8.tex => __textures/Captain_202101_suit_d.tga
+ __textures/Captain_202101_suit_m.512x512/etc_rgb4.tex => __textures/Captain_202101_suit_m.tga
+ __textures/Captain_202102_body_b.512x512/etc_rgb4.tex => __textures/Captain_202102_body_b.tga
+ __textures/Captain_202102_body_d.512x512/etc2_rgba8.tex => __textures/Captain_202102_body_d.tga
+ __textures/Captain_202102_body_m.512x512/etc_rgb4.tex => __textures/Captain_202102_body_m.tga
+ __textures/Captain_202102_stuff_d.64x64/etc_rgb4.tex => __textures/Captain_202102_stuff_d.tga
```

打开 __textures 目录见证奇迹时刻。



贴图

textool工具依赖第三方贴图解码库tex2img¹，该工具封装了BinomialLLC/basis_universal²、Ericsson/ETCPACK³和powervr-graphics/Native_SDK⁴，感谢老哥K0lb3⁵提供的便利。笔者在此基础上增加了RGBA32、RGBA4444、RGB24、RGB565和Alpha8贴图格式的转码，经过这么一番整合，应该可以应付绝大部分的贴图转码。

1. <https://github.com/K0lb3/tex2img.git> ↵

2. https://github.com/BinomialLLC/basis_universal/ ↵

3. <https://github.com/Ericsson/ETCPACK> ↵

4. https://github.com/powervr-graphics/Native_SDK/tree/master/framework/PVRCore/texture ↵

5. <https://github.com/K0lb3> ↵

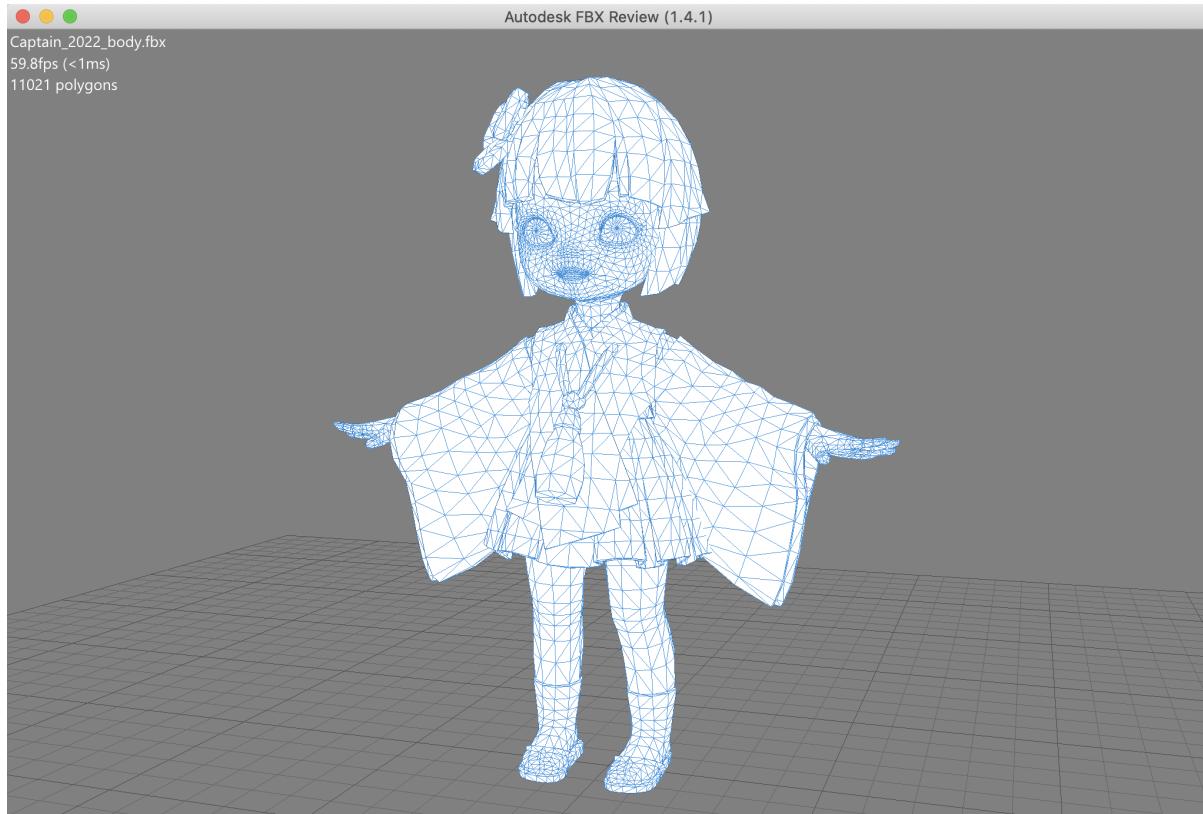
模型

在该案里面我们继续使用 `artresource_captainpbr_captain_202.god` 文件来演示，相比贴图逆向，模型逆向就简单多了，一行命令就可以把ab资源里面的模型导出为 `*.fbx` 文件。

```
$ abtool savefbx AssetBundles/Android/artresource_captainpbr_captain_202.god
```

```
LARRYHOU-MC8:assets larryhou$ abtool-cr savefbx AssetBundles/Android/artresource_captainpbr_captain_202.god
[0] AssetBundles/Android/artresource_captainpbr_captain_202.god
>> __fbx/Captain_2022_body.fbx
>> __fbx/Captain_2023_suit.fbx
>> __fbx/Captain_2022_suit.fbx
>> __fbx/Captain_2021_suit.fbx
>> __fbx/Captain_2021_stuff.fbx
>> __fbx/Captain_2023_body.fbx
>> __fbx/Captain_2021_body.fbx
```

FBX文件可以通过[Autodesk FBX Review¹](#)打开



也可以用其他3D工具打开，比如我们可以在Blender里面查看模型的法线、UVs等信息。

贴图



结合贴图逆向得到的贴图，那么我们可以尝试用基础贴图 `Captain_202201_body_b.tga` 简单渲染一下。

贴图



呃，跟预期的好像不太一样，这简直像鬼一样……别着急，把贴图上下翻转下就能正常显示了。

贴图



¹. <https://www.autodesk.com/products/fbx/fbx-review> ↵

资源防护

通过资源逆向案例我们见识了abtool强大的资源反编译能力，但是这个时候我们不应该兴奋，而是应该无比忧虑才对：因为第三方工具可以在没有项目仓库权限的情况下轻易获取游戏资源，这些都是项目团队日夜攻坚、长时间累积优化的结果，如果被用于非法目的，对游戏是非常不利的。问题来了：既然ab资源如此容易破解，那么该如何保护游戏资产？

打包ab资源时关掉TypeTree

我们先来看下ab打包接口

```
public static AssetBundleManifest BuildAssetBundles(
    string outputPath,
    AssetBundleBuild[] builds,
    BuildAssetBundleOptions assetBundleOptions,
    BuildTarget targetPlatform);
```

第三个枚举参数 `BuildAssetBundleOptions` 用来控制ab的打包行为，其中枚举值 `DisableWriteTypeTree` 可以关闭 TypeTree。

```
/// <summary>
///   <para>Do not include type information within the AssetBundle.</para>
/// </summary>
DisableWriteTypeTree = 8,
```

由于abtool绝大部分功能都基于TypeTree，那是不是关闭TypeTree资源就安全了？没那么简单！TypeTree是由Unity生成，换句话说，如果拿到相同版本的Unity也是可以轻易获取TypeTree的，在这种情况下，关掉TypeTree的意义仅仅是防止了破解 `MonoBehaviour`，防护等级是很弱的！换句话说，如果您用了Unity公开发行的版本（标准版），那么您的游戏资产完全是在裸奔的！

修改关键资源序列化

如果您的项目有源码，那么可以把一些关键资源的序列化字段改一下。比如 `Texture2D` 这个资源类型，它的数据结构大致如下

```

struct Texture2D: public Object {
    std::string m_Name; // 1
    int32_t m_ForceFallbackFormat; // 2
    bool m_DownscaleFallback; // 3
    int32_t m_Width; // 4
    int32_t m_Height; // 5
    int32_t m_CompleteImageSize; // 6
    int32_t m_TextureFormat; // 7
    int32_t m_MipCount; // 8
    bool m_IsReadable; // 9
    int32_t m_ImageCount; // 10
    int32_t m_TextureDimension; // 11
    GLTextureSettings m_TextureSettings; // 12
    int32_t m_LightmapFormat; // 13
    int32_t m_ColorSpace; // 14
    TypelessData m_TexData; // 15
    StreamingInfo m_StreamData; // 16
};

```

在资源对象序列化过程中，`string / map / set / vector` 等数据类型均被当做数组来处理：先用4字节存储数组长度，然后按顺序存储数组元素，对于`string`它的数组元素类型是`char`。基于此，我们可以在标准版的Unity源码里面做一些微小改动，比如把上述类型序列化顺序交换位置，或者在这些字段的序列化之前写入一个很大的整形，这样通过标准版Unity反编译资源就会导致崩溃或者无限循环。该保护措施的本质是修改资源类型的数据结构，使其与标准版Unity生成TypeTree产生差异，从而导致通过标准版Unity破解资源的方法失效。

修改 AssetBundleArchive 存储结构

每个ab资源在Unity里面会都会通过一个`AssetBundleArchive`容器来存储，它的作用是压缩资源对象数据并提供文件寻址功能，并没有什么特别的，如果您的项目有源码完全可以自行设计一个实现类似功能的资源容器，比如王者荣耀、原神游戏做了类似的设计。

修改 SerializedFile 存储结构

除了定制资源容器，还可以通过修改容器内资源的存储方式，比如修改`SerializedFile`的metadata数据的组织方式。

加密

如果觉得上述源码修改过于复杂，那么可以对ab文件做二进制的加密，也可以选择对其部分内容做加密，前提是不要有过多运行时开销，比如LOL手游做了类似的设计。

资源引发崩溃

一般情况下资源导致游戏崩溃并不多见，随着游戏资源量的增加以及构建时间的增加，就会需要用到Unity的ab增量编译，但是如果增量编译过程中发生了Unity闪退或者系统崩溃，那么就有很大几率导致编译出来的资源出问题，并且Unity下次构建时也无法自我修复。

举个例子，我们项目的ab资源增加到3G的时候遇到一次比较严重的由热更资源引发的崩溃，由于每次崩溃的时机各有不同，定位起来非常困难，最后通过一个能够稳定复现的崩溃定位到原因是：材质球A引用的贴图T在另外一个ab文件里面，但是通过材质球A的引用路径去加载后得到的是却是另外一个材质球B，可以理解为材质球A的贴图指针位置不是贴图T，那么通过材质球A的贴图指针尝试访问贴图T的时候就崩溃了。然后根据这个特征，笔者开发了 `scanref` 工具，它的作用是扫描所有引用了材质球、贴图、Mesh的对象，然后通过引用路径找到目标资源对象，并校验目标资源的类型是否跟预期一致。

```
LARRYHOU-MC8:iMSDK_Garena_SA_iOS_193_AssetBundle larryhou$ abtool scanref
archive:/cab-00a615392edb6a8c75e792bb4d60c290/cab-00a615392edb6a8c75e792bb4d60c290 dynamic/module/weaponskin_bundle_12!7!forcedownload!cod_models$weapons$mainweapon$mainweapon_061_v17365510.pak
- archive:/cab-f8adecc20aec1172be68dc0f5781b6dc/cab-f8adecc20aec1172be68dc0f5781b6dc i:3 missing
archive:/cab-00de96edie3a030462756b657221e6e9/cab-00de96edie3a030462756b657221e6e9 dynamic/module/vehiclelesskin_bundle_01!7!forcedownload!cod_models$vehicles$br_motorcycle5510.pak
- archive:/cab-2102a399eb2d16395b45a5a344073458/cab-2102a399eb2d16395b45a5a344073458 i:3 missing
archive:/cab-013d2ae3ffd3963f2505995b4803b5c/cab-013d2ae3ffd3963f2505995b4803b5c dynamic/module/role_bundle_08!7!forcedownload!cod_models$avatars$seal6_003_bluewhite.pak
T archive:/cab-935fbdb22f82316cda48c391a5d38e03b/cab-935fbdb22f82316cda48c391a5d38e03b i:4 expect=Mesh:43 actual=Texture2D:28
T archive:/cab-935fbdb22f82316cda48c391a5d38e03b/cab-935fbdb22f82316cda48c391a5d38e03b i:5 expect=Mesh:43 actual=Texture2D:28
T archive:/cab-935fbdb22f82316cda48c391a5d38e03b/cab-935fbdb22f82316cda48c391a5d38e03b i:13 expect=Texture2D:28 actual=Mesh:43
T archive:/cab-935fbdb22f82316cda48c391a5d38e03b/cab-935fbdb22f82316cda48c391a5d38e03b i:16 expect=Texture2D:28 actual=Mesh:43
T archive:/cab-935fbdb22f82316cda48c391a5d38e03b/cab-935fbdb22f82316cda48c391a5d38e03b i:17 expect=Avatar:90 actual=Mesh:43
T archive:/cab-935fbdb22f82316cda48c391a5d38e03b/cab-935fbdb22f82316cda48c391a5d38e03b i:22 expect=Material:21 actual=Avatar:90
- archive:/cab-935fbdb22f82316cda48c391a5d38e03b/cab-935fbdb22f82316cda48c391a5d38e03b i:23 missing
```

图中的日志是什么含义呢？那是纯正的崩溃的味道！

```
T archive:/cab-935fbdb22f82316cda48c391a5d38e03b/cab-935fbdb22f82316cda48c391a5d38e03b i: 4 expect=Mesh: 43 actual
```

针对其中一行日志简单解释下，`archive: /` 开头一串文本是ab里面 `SerializedFile` 的路径，可以用来寻址。资源 `dynamic/module/role_bundle_08!7!forcedownload!cod_models$avatars$seal6_003_bluewhite.pak` 里面id 为 767 的 `SkinnedMeshRenderer` 对象拥有一个外部资源指针 `PPtr<Mesh>`，指向另外一个ab资源文件(`archive:/cab-935fbdb22f82316cda48c391a5d38e03b/cab-935fbdb22f82316cda48c391a5d38e03b`)中索引为 `m_PathID=4` 的对象，但是目标引用路径的资源其实是个 `Texture2D` 对象。

上述 `SkinnedMeshRenderer` 数据文本展开显示

```

<SkinnedMeshRenderer: 137> id=767
    m_GameObject: PPtr<GameObject>
        m_FileID = 0
        m_PathID = 169
    m_Enabled: bool = 1
    m_CastShadows: uint8_t = 1
    m_ReceiveShadows: uint8_t = 1
    m_ReceiveNoSSShadows: uint8_t = 0
    m_DynamicShadows: uint8_t = 1
    m_MotionVectors: uint8_t = 2
    m_LightProbeUsage: uint8_t = 1
    m_RefractionProbeUsage: uint8_t = 3
    m_LightmapIndex: uint16_t = 65535
    m_LightmapIndexDynamic: uint16_t = 65535
    m_LightmapTilingOffset: Vector4f
        x: float = 1
        y: float = 1
        z: float = 0
        w: float = 0
    m_Materials: vector<PPtr<Material>>
        [ 0]: PPtr<Material>
            m_FileID = 0
            m_PathID = 3
    m_StaticBatchInfo: StaticBatchInfo
        firstSubMesh: uint16_t = 0
        subMeshCount: uint16_t = 0
    m_StaticBatchRoot: PPtr<Transform>
        m_FileID = 0
        m_PathID = 0
    m_ProbeAnchor: PPtr<Transform>
        m_FileID = 0
        m_PathID = 0
    m_LightProbeVolumeOverride: PPtr<GameObject>
        m_FileID = 0
        m_PathID = 0
    m_SortingLayerID: int32_t = 0
    m_SortingLayer: int16_t = 0
    m_SortingOrder: int16_t = 0
    m_Quality: int32_t = 0
    m_UpdateWhenOffscreen: bool = 0
    m_SkinnedMotionVectors: bool = 0
    m_Mesh: PPtr<Mesh>
        m_FileID = 4
        m_PathID = 4
    m_Bones: vector<PPtr<Transform>>
        [ 0]: PPtr<Transform>
            m_FileID = 0
            m_PathID = 703
    m_BindShapeWeights: vector<float>
    m_RootBone: PPtr<Transform>
        m_FileID = 0
        m_PathID = 703
    m_AABB: AABB
        m_Center: Vector3f
            x: float = -0.145836
            y: float = 0.0210291
            z: float = 0.00812059
        m_Extent: Vector3f
            x: float = 0.109583
            y: float = 0.15261
            z: float = 0.10819
    m_DirtyAABB: bool = 0

```

通过引用路径找到的资源却是 Texture2D 对象

```
$ abtool edit dynamic/module/common\!7\!forcedownload\!cod_models\$avatar\$seal6_003.pak
[0] dynamic/module/common\!7\!forcedownload\!cod_models\$avatar\$seal6_003.pak
$ lua file:dump_object(4)
<Texture2D:28> id=4
  m_Name:string = C_T_M_Seal6_Shotgun_003_N
  m_Width:int32_t = 1024
  m_Height:int32_t = 1024
  m_CompleteImageSize:int32_t = 626288
  m_TextureFormat:int32_t = 50
  m_MipCount:int32_t = 11
  m_IsReadable:bool = 0
  m_StreamingMipmaps:bool = 1
  m_StreamingMipmapsPriority:int32_t = 2
  m_StreamingGroupID:int32_t = 567068459
  m_ImageCount:int32_t = 1
  m_TextureDimension:int32_t = 2
  m_TextureSettings:GLTextureSettings
    m_FilterMode:int32_t = 2
    m>Aniso:int32_t = 1
    m_MipBias:float = 0
    m_WrapMode:int32_t = 0
  m_LightmapFormat:int32_t = 0
  m_ColorSpace:int32_t = 0
  m_TexData:TypelessData
    size = 0
    data =
  m_StreamData:StreamingInfo
    offset:uint32_t = 158432
    size:uint32_t = 626288
    path:string = archive:/CAB-935fbdb22f82316cda48c391a5d38e03b/CAB-935fbdb22f82316cda48c391a5d38e03b.ress
```

把 Texture2D 对象强转成 Mesh 对象赋值给 SkinnedMeshRenderer 对象，最终就会导致崩溃。庆幸的现在可以通过 abtool scanref 扫描游戏的所有资源，30秒就可以扫描3G左右的ab资源，可以说是非常高效的。

```
find . -iname '*.pak' | xargs abtool scanref
```

scanref 会把扫描数据缓存到当前目录的 assets.ref 文件，下次运行的时候 scanref 会自动读取缓存文件，这样不用再次扫描ab资源就可以快速得到相同的结果，直接把耗时降到2秒。

```
abtool scanref
```

上面的操作还只是发现资源问题，其实我们更想知道这样的问题如何解决，根据问题的严重性有两种解决方法：

1. 如果只有少量ab资源存在引用问题，那么可以把扫描出来的ab资源从构建机的输出目录删掉，下次构建时 Unity会修复这些资源；
2. 如果有很多ab资源都存在引用问题，可以通过ab资源回退的方式来解决，前提是每次构建后都归档了相应的 ab资源。具体操作是，下载历史构建的ab资源，运行 abtool scanref 找到一个没有资源引用问题的资源版本，然后用这个版本的ab资源替换构建机上的ab资源，并重新运行一次ab打包流程。

到现在为止，我们知道如何通过abtool发现资源问题，也拥有解决问题的方法，但是相信大家依然会有疑问：资源崩溃问题是如何产生的？

就上面 SkinnedMeshRenderer 错误引用 Texture2D 的例子来说，可以解释为：由于依赖关系变更，A和B两个ab资源都需要重新打包，但是ab资源B打包完成后，由于意外原因导致Unity打包流程中断，本来应该被重新打包的A资源，因为Unity的闪退而被忽略打包，所以A保持为老资源状态并引用了一个错误的资源，并且下次构建的时候 Unity也无法识别这种异常情况。

资源引用丢失

在增量编译过程中如果Unity闪退了，除了会产生导致崩溃的问题资源，也会导致一些不那么严重、但是非常奇怪的渲染结果：屏幕渲染出现紫块或者渲染效果异常，通过 `scanref` 也可以发现这样的问题，如下日志里面还有 `missing` 标记。

```
archive: /cab-f8642b11cae21bfbe1fac93206d98fba/cab-f8642b11cae21bfbe1fac93206d98fba dynamic/module/vehiclesski
- archive: /cab-12279edb29aa918860d5fb88556056ca/cab-12279edb29aa918860d5fb88556056ca i: 4 missing
```

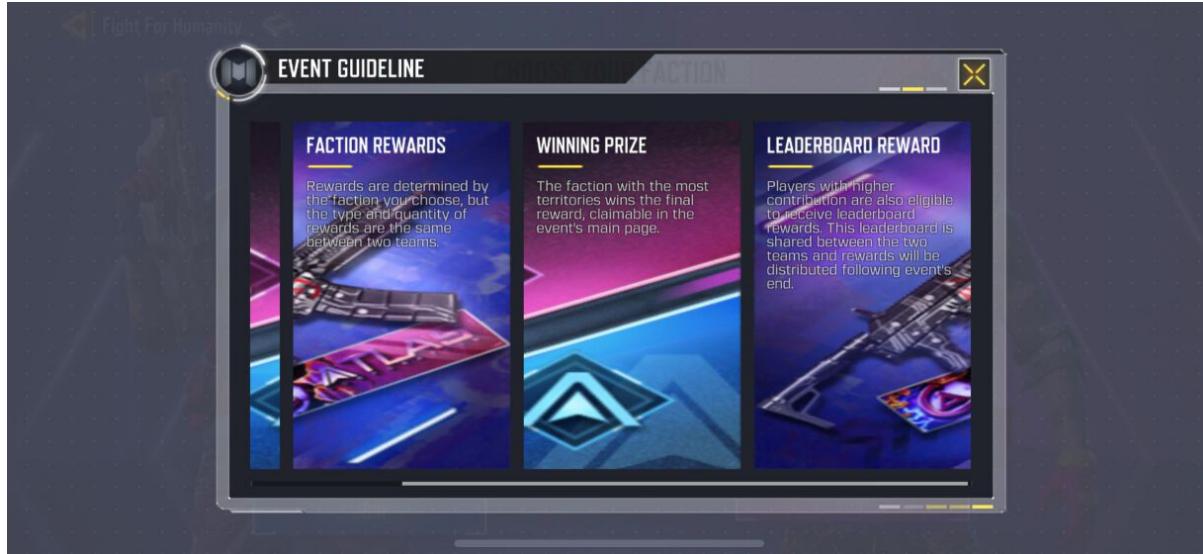
屏幕紫块

假如ab资源A引用的材质球M在另外一个ab文件B里面，由于M的打包配置变化导致它应该被打包到ab资源C里面，但是由于Unity闪退B和C被正常打包而A没有被正常打包，这种情况下A资源尝试去B资源里面加载M，但是M已经从B里面移除并被打包到了C里面，从而出现材质球加载失败的问题，就会出现紫块。

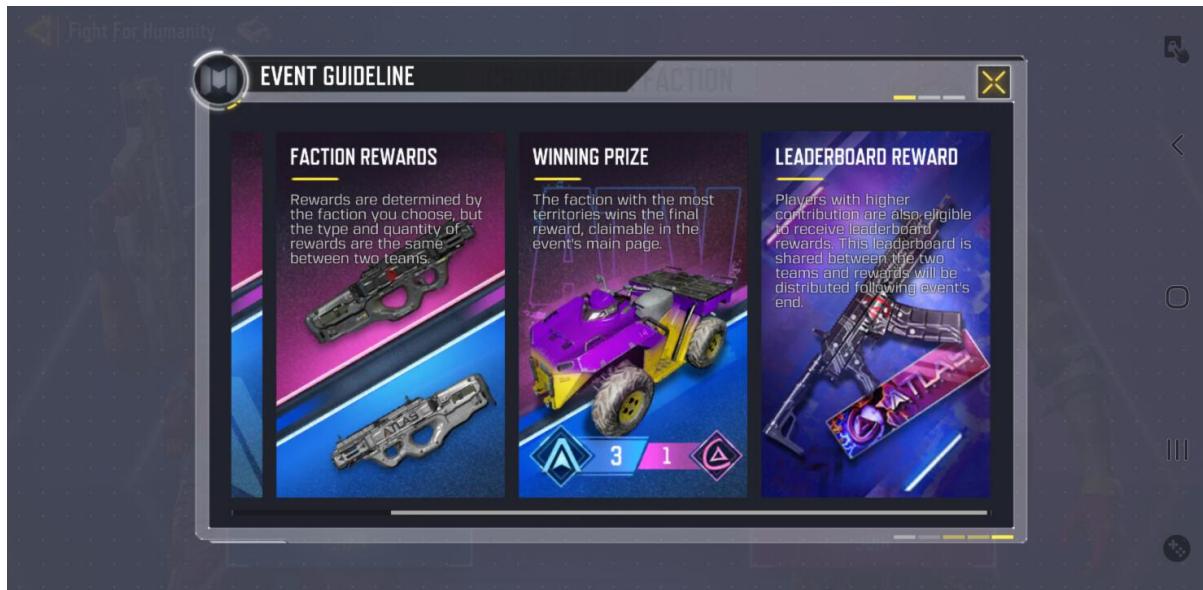
渲染异常

资源显示异常

某天我被拉去查一个iOS资源显示问题，如下



正常应该是这样的



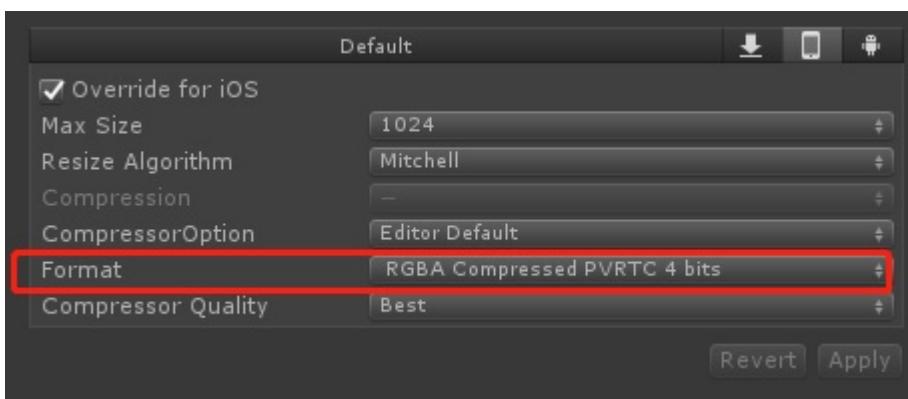
从表象来看是图片被横向拉伸了，美术同学和开发同学都说资源没问题，Editor验收也正常，安卓的真机也没问题。如果是你该怎么定位这个问题？

我来定位的话，就简单多了！工具在手，首先反编译下贴图：通过 abtool savetex 得到 Update_4_1.1024x1024.pvrtc_rgba4.tex，再用textool转码得到tga文件，下图为tga转了jpg的显示。

贴图



从结果来看，图片确实被拉伸了，原图是512x1024尺寸的，实际变成1024x1024尺寸了，这是由于PVRTC不支持长方形的贴图导致的：对于非正方形的贴图，会被PVRTC强制拉伸成正方形再进行编码。当然，最终打开Unity那一刻也确认下笔者的判断。



修复也很简单，改成ASTC格式就可以了。

贴图

资源差异比对

贴图

资源编辑

第三章 命令详解

截止文档撰写日起已有20多个内置命令，它们均是在解决资源问题过程中逐渐增加和完善的，具有很强的实用性。大部分命令运行过程中输出到终端的日志都是有颜色样式的，这个设计主要是根据信息的重要性做不同的高亮突出显示，方便在日志里面找到有用的信息。当然，也强烈建议您把终端设置为黑色背景样式，不然颜色显示会比较奇怪，因为黑色背景为终端显示样式的调试环境。

```
LARRYHOU-MC8:abtool larryhou$ ./build/bin/abtool list doc/resources/android/quickstart.ab
[0] doc/resources/android/quickstart.ab
[1/1] quickstart.ab
    archive:/cab-438d38142b45d040df1fcfd2d05a1cf/cab-438d38142b45d040df1fcfd2d05a1cf assets:11 objects:304
        [01/11] assets/quickstart/book.json TextAsset i:6263331711779796716
        [02/11] assets/quickstart/charactor/ch29_1001_diffuse.png Texture2D i:16325549884401675245
        [03/11] assets/quickstart/charactor/ch29_1001_glossiness.png Texture2D i:10610311816000281347
        [04/11] assets/quickstart/charactor/ch29_1001_normal.png Texture2D i:702652466962523341
        [05/11] assets/quickstart/charactor/ch29_1001_specular.png Sprite i:10291185036364045368
        [06/11] assets/quickstart/charactor/ch29_body.mat Material i:12245866026436902592
        [07/11] assets/quickstart/charactor/ch29_nonpbr-flair.fbx Avatar i:5307880407919849257
        [08/11] assets/quickstart/charactor/flair.controller AnimatorController i:4643326605353963186
        [09/11] assets/quickstart/prefabs/ch29_nonpbr-flair.prefab GameObject i:11058135177181279887
        [10/11] assets/quickstart/prefabs/renderer.prefab GameObject i:505592247217905686
        [11/11] assets/quickstart/skybox.jpg Cubemap i:13196032094151524373
    [1/1] archive:/cab-438d38142b45d040df1fcfd2d05a1cf/cab-438d38142b45d040df1fcfd2d05a1cf objects:304 assets:11 quickstart.ab
[#] objects:304 assets:11
```

然而，在有些情况下，我们需要对工具输出的日志做进一步分析，这个时候我们是不希望有颜色高亮的，因为这些颜色都是通过[颜色控制符¹](#)实现的，这会让日志里面多出一些方括号`[`的字符，如下图显示看起来比较杂乱，有可能会让下游的分析工具产生不符合预期的结果。

```
[0] doc/resources/android/quickstart.ab
[1/1] quickstart.ab
    archive:/cab-438d38142b45d040df1fcfd2d05a1cf/cab-438d38142b45d040df1fcfd2d05a1cf [2m assets:11 objects:304 [0m
        [01/11] assets/quickstart/book.json [33mTextAsset [0m [2m i:6263331711779796716 [0m
        [02/11] assets/quickstart/charactor/ch29_1001_diffuse.png [33mTexture2D [0m [2m i:16325549884401675245 [0m
        [03/11] assets/quickstart/charactor/ch29_1001_glossiness.png [33mTexture2D [0m [2m i:10610311816000281347 [0m
        [04/11] assets/quickstart/charactor/ch29_1001_normal.png [33mTexture2D [0m [2m i:702652466962523341 [0m
        [05/11] assets/quickstart/charactor/ch29_1001_specular.png [33mSprite [0m [2m i:10291185036364045368 [0m
        [06/11] assets/quickstart/charactor/ch29_body.mat [33mMaterial [0m [2m i:12245866026436902592 [0m
        [07/11] assets/quickstart/charactor/ch29_nonpbr-flair.fbx [33mAvatar [0m [2m i:5307880407919849257 [0m
        [08/11] assets/quickstart/charactor/flair.controller [33mAnimatorController [0m [2m i:4643326605353963186 [0m
        [09/11] assets/quickstart/prefabs/ch29_nonpbr-flair.prefab [33mGameObject [0m [2m i:11058135177181279887 [0m
        [10/11] assets/quickstart/prefabs/renderer.prefab [33mGameObject [0m [2m i:505592247217905686 [0m
        [11/11] assets/quickstart/skybox.jpg [33mCubemap [0m [2m i:13196032094151524373 [0m
    [1/1] archive:/cab-438d38142b45d040df1fcfd2d05a1cf/cab-438d38142b45d040df1fcfd2d05a1cf [33m objects:304 assets:11 [0m [2m quickstart.ab [0m
[#] objects:304 assets:11
```

不过工程根目录里的 `nocolor.cpp` 的小工具可以轻松去掉终端的颜色样式，该工具含代码格式只有26行C++代码，非常轻量高效。

贴图

```
#include <iostream>
#include <string>

int main( int argc, char* argv[])
{
    std::string pipe;
    while ( std::getline( std::cin, pipe))
    {
        auto cursor = pipe.begin();
        for ( auto iter = pipe.begin(); iter != pipe.end(); iter++)
        {
            if (*iter == '\e' && *( iter+1) == '[' )
            {
                ++iter; // [
                ++iter; // d
                ++iter; // m
                if (*iter != 'm') { ++iter; }
                continue;
            }

            *cursor++ = *iter;
        }
        *cursor = 0;
        std::cout << pipe.data() << std::endl;
    }
    return 0;
}
```

可以通过如下终端命令快速编译。

```
clang++ -std=c++11 nocolor.cpp -o/usr/local/bin/nocolor
```

使用起来也十分方便，只需命令末尾追加管道。

```
abtool list doc/resources/android/quickstart.ab | nocolor
```

```
LARRYHOU-MC8:abtool larryhou$ ./build/bin/abtool list doc/resources/android/quickstart.ab | nocolor
[0] doc/resources/android/quickstart.ab
[1/1] quickstart.ab
archive:/cab-438d38142b45d040df1fcfd2d05a1cf/cab-438d38142b45d040df1fcfd2d05a1cf assets:11 objects:304
[01/11] assets/quickstart/book.json TextureAsset i:626331711779796716
[02/11] assets/quickstart/charactor/ch29_1001_diffuse.png Texture2D i:16325549884401675245
[03/11] assets/quickstart/charactor/ch29_1001_glossiness.png Texture2D i:10610311816000281347
[04/11] assets/quickstart/charactor/ch29_1001_normal.png Texture2D i:702652465962523341
[05/11] assets/quickstart/charactor/ch29_1001_specular.png Sprite i:10291185036364045368
[06/11] assets/quickstart/charactor/ch29_body.mat Material i:12245866026436902592
[07/11] assets/quickstart/charactor/ch29_nonpbr-flair.fbx Avatar i:5307880407919849257
[08/11] assets/quickstart/charactor/flair.controller AnimatorController i:4643326605353963186
[09/11] assets/quickstart/prefabs/ch29_nonpbr-flair.prefab GameObject i:11058135177181279887
[10/11] assets/quickstart/prefabs/renderer.prefab GameObject i:505592247217905686
[11/11] assets/quickstart/skybox.jpg Cubemap i:13196032094151524373
[1/1] archive:/cab-438d38142b45d040df1fcfd2d05a1cf/cab-438d38142b45d040df1fcfd2d05a1cf objects:304 assets:11 quickstart.ab
[#] objects:304 assets:11
```

¹ https://misc.flogisoft.com/bash/tip_colors_and_formatting ↩

贴图

savetree

贴图

gtt

贴图

dump

贴图

list

贴图

size

贴图

scanref

贴图

scantex

贴图

savefbx

贴图

savetex

贴图

saveta

贴图

saveobj

贴图

objref

贴图

mono

贴图

cmpref

贴图

cmpxtl

贴图

cmphash

贴图

external

贴图

rmtree

贴图

lua

贴图

edit

第四章 进阶开发

贴图

项目架构

贴图

文件解析

贴图

对象序列化

贴图

命令系统

贴图

文件系统

贴图

LUA绑定
