

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
资源引发崩溃	1.3.1
资源引用丢失	1.3.2
资源显示异常	1.3.3
资源差异比对	1.3.4
资源逆向	1.3.5
贴图	1.3.5.1
模型	1.3.5.2
资源编辑	1.3.6
资源防护	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。

贴图

资源引发崩溃

贴图

资源引用丢失

贴图

资源显示异常

贴图

资源差异比对

资源逆向

通俗一点讲，在本章节大家可以了解到如何从ab文件里面反编译资源，也就是从ab文件里面提取出来比较方便查看的文件格式。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，但是实现所有反编译是有成本的，并且不是所有译资源都是我们关心的，所以笔者暂时只实现了优先几个但是高频、实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的实现原理注定它是天然支持资源逆向的，大家在接下来的案例中会看到这一点。

为了增加反编译过程真实感，笔者决定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。

下载安装包

大家可以通过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，如果链接失效请自行从Google搜索结果里面寻找其他可用下载链接。

战歌竞技场 apk

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

```
unzip 10040714_com.tencent.hj.zqgame_a960942_1.5.151_j2e715.apk  
mv -v assets/AssetBundles/Android AssetBundles
```

上述命令可以解压apk安装包里面的所有资源，其中 AssetBundles 目录存储了游戏的ab资源，后续的资源逆向案例默认使用这些资源。

¹. [https://dlid4.myapp.com/myapp/1109006800/cos.release-75620/10040714_com.tencent.hjzqgame_a960942_1.5.151_j2e715.apk ↵](https://dlid4.myapp.com/myapp/1109006800/cos.release-75620/10040714_com.tencent.hjzqgame_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_201.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:275881937572619551
[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/artresource_captainpbr_captain_201.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格式转码里面会失败。

贴图格式转换

从上一步骤得到的贴图都是 *.tex 格式的文件，并不是我们常见的方便审阅的png或者jpg格式，它们是GPU渲染用到的贴图格式，所以还需要进行一次格式转换。在工程根目录放置了python脚本工具 textool.py，它可以自动批量把 *.tex 文件转换成 *.png 或 *.jpg，具体根据贴图是否包含透明通道来最终决定存储格式。

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) + '.' + ('png' if mode == 'RGBA' else 'jpg')
        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
```

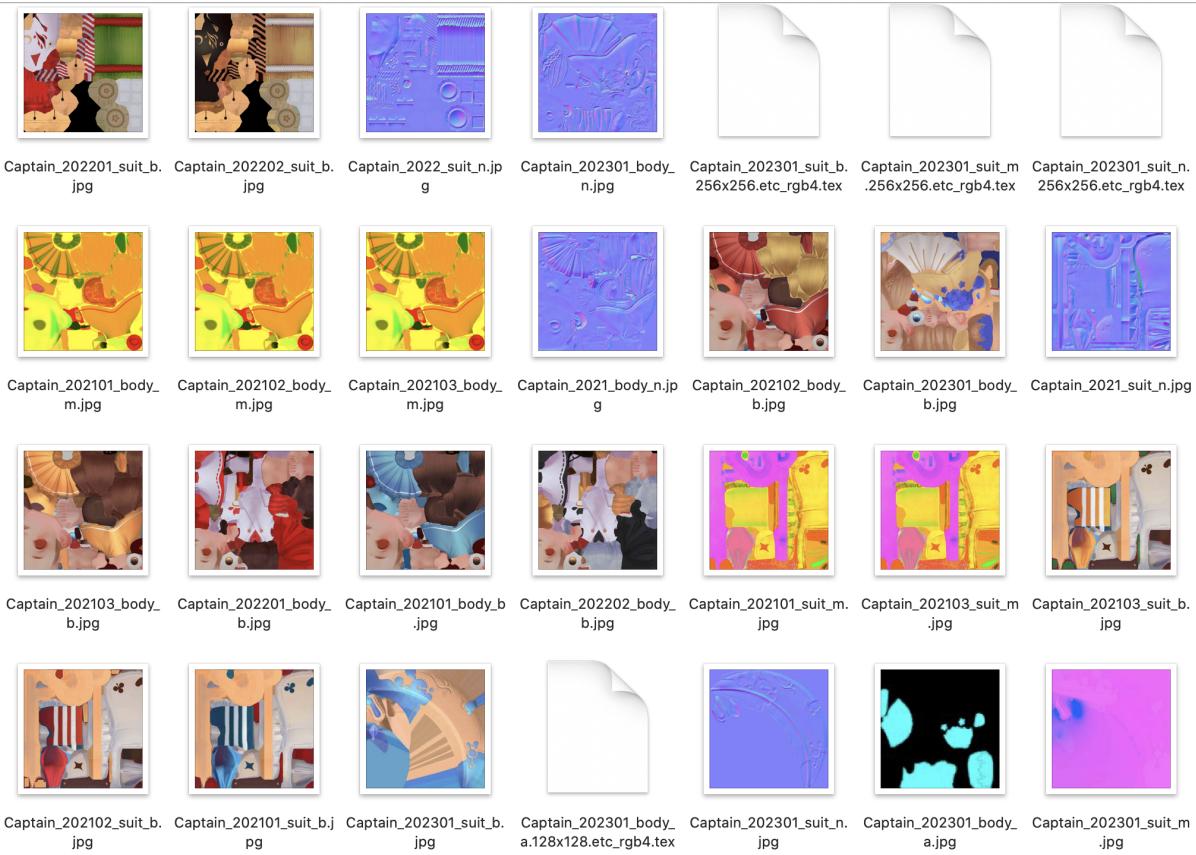
接下来通过textool转换 *.tex 贴图格式，转换后的图片文件存储在源 *.tex 文件的同级目录。

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

```
LARRYHOU-MC8:demo larryhou$ textool __textures/*.tex
+ __textures/Captain_202101_body_a.64x64/etc_rgb4.tex => __textures/Captain_202101_body_a.jpg
+ __textures/Captain_202101_body_b.512x512/etc_rgb4.tex => __textures/Captain_202101_body_b.jpg
+ __textures/Captain_202101_body_d.512x512/etc2_rgba8.tex => __textures/Captain_202101_body_d.jpg
+ __textures/Captain_202101_body_m.512x512/etc_rgb4.tex => __textures/Captain_202101_body_m.jpg
+ __textures/Captain_202101_stuff_b.64x64/etc_rgb4.tex => __textures/Captain_202101_stuff_b.jpg
+ __textures/Captain_202101_stuff_d.64x64/etc_rgb4.tex => __textures/Captain_202101_stuff_d.jpg
+ __textures/Captain_202101_stuff_m.64x64/etc_rgb4.tex => __textures/Captain_202101_stuff_m.jpg
+ __textures/Captain_202101_suit_a.64x64/etc_rgb4.tex => __textures/Captain_202101_suit_a.jpg
+ __textures/Captain_202101_suit_b.512x512/etc_rgb4.tex => __textures/Captain_202101_suit_b.jpg
+ __textures/Captain_202101_suit_d.512x512/etc2_rgba8.tex => __textures/Captain_202101_suit_d.png
+ __textures/Captain_202101_suit_m.512x512/etc_rgb4.tex => __textures/Captain_202101_suit_m.jpg
+ __textures/Captain_202102_body_b.512x512/etc_rgb4.tex => __textures/Captain_202102_body_b.jpg
+ __textures/Captain_202102_body_d.512x512/etc2_rgba8.tex => __textures/Captain_202102_body_d.png
+ __textures/Captain_202102_body_m.512x512/etc_rgb4.tex => __textures/Captain_202102_body_m.jpg
+ __textures/Captain_202102_stuff_d.64x64/etc_rgb4.tex => __textures/Captain_202102_stuff_d.jpg
+ __textures/Captain_202102_suit_b.512x512/etc_rgb4.tex => __textures/Captain_202102_suit_b.jpg
+ __textures/Captain_202102_suit_d.512x512/etc2_rgba8.tex => __textures/Captain_202102_suit_d.png
+ __textures/Captain_202103_body_b.512x512/etc_rgb4.tex => __textures/Captain_202103_body_b.jpg
+ __textures/Captain_202103_body_d.512x512/etc2_rgba8.tex => __textures/Captain_202103_body_d.png
+ __textures/Captain_202103_body_m.512x512/etc_rgb4.tex => __textures/Captain_202103_body_m.jpg
+ __textures/Captain_202103_stuff_d.64x64/etc_rgb4.tex => __textures/Captain_202103_stuff_d.jpg
+ __textures/Captain_202103_suit_b.512x512/etc_rgb4.tex => __textures/Captain_202103_suit_b.jpg
+ __textures/Captain_202103_suit_d.512x512/etc2_rgba8.tex => __textures/Captain_202103_suit_d.png
+ __textures/Captain_202103_suit_m.512x512/etc_rgb4.tex => __textures/Captain_202103_suit_m.jpg
+ __textures/Captain_2021_body_n.512x512/etc_rgb4.tex => __textures/Captain_2021_body_n.jpg
+ __textures/Captain_2021_stuff_n.64x64/etc_rgb4.tex => __textures/Captain_2021_stuff_n.jpg
+ __textures/Captain_2021_suit_n.512x512/etc_rgb4.tex => __textures/Captain_2021_suit_n.jpg
```

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](https://github.com/K0lb3/tex2img)，该工具封装了[BinomialLLC/basis_universal²](https://github.com/BinomialLLC/basis_universal)、[Ericsson/ETCPACK³](https://github.com/Ericsson/ETCPACK)和[powervr-graphics/Native_SDK⁴](https://github.com/powervr-graphics/Native_SDK)，感谢老哥[K0lb3](#)提供的便利，也建议大家给他点个赞。在此基础上笔者增加了RGBA32、RGBA4444、RGB24、RGB565和Alpha8贴图格式的解码，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 ↩

贴图

模型

贴图

资源编辑

贴图

资源防护

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
