**[Unity安卓手游汉化笔记](https://www.cnblogs.com/guobaoxu/p/12055930.html)**

[华夏国宝](https://home.cnblogs.com/u/guobaoxu/) 著

这是我个人对于Unity逆向汉化的总结，这里默认游戏没有采用额外的安全措施，使用的Demo是使用Unity直接生成的，实际应用中，大部分需要先绕开游戏的安全机制，或者在汉化结束后需要进行重新签名等。

主要工具：

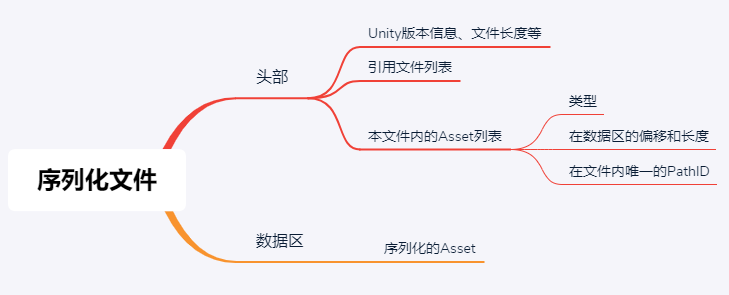
　　AssetStudio（地址：<https://www.perfare.net/tag/assetstudio>）

　　il2cppdumper：<https://www.perfare.net/tag/il2cppdumper>，和AssetStudio是一个作者，具体操作也在他的网站上有

　　UABE（地址：<https://7daystodie.com/forums/showthread.php?22675-Unity-Assets-Bundle-Extractor>）

一、基础知识之Unity的资源文件

　　这里说的资源文件不是做游戏过程中使用的资源，而是在序列化之后得到的资源文件，在Android平台下，Unity的资源文件都放在Asset/bin/Data文件夹下，也就是AssetStudio读取的文件夹，这下面的文件就是Unity的资源文件，子文件夹Manager里面放的是代码相关的文件，其他都是资源文件，如果读过Asset Studio的源码，类SerialiableFile解析的文件，就是最常用的资源文件格式，我在汉化过程中基本就是跟这种文件打交道了。这种序列化文件的基本构造是文件头部+数据区，数据区里放序列化的Asset对象，

****

　　Asset的类型很多，可以参见官网（<https://docs.unity3d.com/Manual/ClassIDReference.html>），如果对他的序列化方式感兴趣好奇，可以用UABE导出文本和二进制对照着看看

　　关于Asset对象的序列化，有两个地方需要关注：

　　① 首先是对象之间会有引用关系，因此有个“指针”，指针由两个部分组成，如下：

class PPtr {

Int32 fileID; // 在文件头部的引用文件列表中的第几个，0指代本文件

Int64 pathID; // 在该文件中的第几个对象，0表示空

}

　　② 然后是游戏制作过程中肯定会有自己的脚本，绑定在场景物体上的脚本大多会有参数，这些参数的数据会以静态资源的形式也写入资源文件。这部分会这么处理：

　　首先是类的信息（在哪一个DLL、命名空间、类名等）会放入MonoScript，然后具体某一个对象会放入一个MonoBehavior，MonoBehavior具体相关的东西我另外写成了博客，[Unity手游汉化笔记①：UABE+AssetStudio修改MonoBehavior](file:///D:\\Users\\Pozet\\Downloads\\www.cnblogs.com\\guobaoxu\\unity%E6%B1%89%E5%8C%96%E6%95%99%E7%A8%8B-%E5%8D%8E%E5%A4%8F%E5%9B%BD%E5%AE%9D\\12059225.html)

　　在汉化过程中，资源的替换主要就是MonoBehavior、图片（Texture2D）、TextAsset三种，后两种都有插件，操作简单，写了篇简单的博客混一下：[Unity手游汉化笔记⑤：图片和TextAsset替换](file:///D:\\Users\\Pozet\\Downloads\\www.cnblogs.com\\guobaoxu\\unity%E6%B1%89%E5%8C%96%E6%95%99%E7%A8%8B-%E5%8D%8E%E5%A4%8F%E5%9B%BD%E5%AE%9D\\12124226.html)

二、基础知识之脚本后端

　　熟悉正向开发的朋友应该知道，Android平台会有两种脚本后端，Mono和il2cpp。

　　① Mono的话比较简单，脚本的代码会被编译成DLL，然后放在assets\bin\Data\Managed文件夹下，用dnSpy等工具可以进行反编译和修改，在没有安全措施的情况下，dnSpy可以看到十分清晰的代码，修改也方便，用来汉化更是绰绰有余

　　② il2cpp就麻烦多了，脚本最终会被编译进lib/libil2cpp.so文件夹，同时，脚本内的一些字符串等信息会被放入assets\bin\Data\Managed\Metadata文件夹下面的global-metadata.dat文件里，如果用IDA对so文件反编译的话，也只能看到汇编代码。所以想要对il2cpp脚本后端的游戏进行大改是十分麻烦的事情，但是汉化主要聚焦在字符串的修改上，所以会相对简单的多。

　　从目前的趋势来看，il2cpp脚本后端使用频率会越来越高，毕竟难以反编译出源码，又是官方推荐。

三、工具介绍

　　Asset Studio：这个工具主要用来预览，是开源的，建议看看他的源码，并不复杂，可以让自己对Unity的资源部分有一个很好的理解，但是他没有解析所有的资源类型，之多常用的做了解析。

　　UABE：这个工具主要用来替换，这个并不是开源的，实际上看一眼前面提到的被序列化的文件就知道，要解析所有的Unity的资源是个大工程了。主要功能是导出导入，可以二进制形式，也可以文本形式，针对图片、TextAsset等还有专门的插件。

　　il2cppdumper：和AssetStudio是一个作者，也是开源的。如果结合前面两节，会发现一个问题，解析MonoBehavior是需要类的参数列表的，但是il2cpp后端的代码都被编译成so了，所以这里就需要一个工具，把参数列表从metadata和so里面导出来，这就是il2cppdumper的主要功能了，在使用前两个工具的时候也需要用到这个工具导出的DLL。具体使用方法在官网上有介绍。

四、汉化之常见UI框架

　　这里简单说一下文字汉化中会遇到的三个UI框架

　　1.NGUI：在UGUI出现以前比较通用的UI实现方案

　　2.UGUI：Unity自己做的UI，是官方推荐使用的

　　3.TextMeshPro：简称TMPro，专注于文本的展示，用这个做的文本，在放大之后不会出现毛边，所以很多人都喜欢用来做一些固定文字的展示（需要用户输入的就基本不用这个了）

　　三种都有共同的特点，都需要字体跟文本内容两方面的替换。

五、字体替换

　　三个UI框架都有自己的字体表现形式，NGUI对应BMFont，UGUI对应TTF，TMPro对应自己的字体，其中BMFont和TMPro字体是类似的，都是图片+一个Asset描述字符切片的位置，然后在绑定一个纹理用于渲染，TTF则是有专门的FontAsset类型（大概这就是亲儿子吧），每一种的替换另外写成博客，请看：

BMFont字体：Unity手游汉化笔记③：UABE替换BMFont

TTF字体：Unity手游汉化笔记②：使用UABE替换TTF字体

TMPro字体：Unity手游汉化笔记④：UABE+AssetStudio替换TMPro字体

六、文字替换

　　文字的替换有两种，一种是资源文件中的，这种是在开发过程中，在检查器里直接输入的文字，一般是默认值或者固定的文本，另一种则是代码中的字符串，涉及代码，前面提到的两种脚本后端又需要拿出来说一说了。

　　资源文件中的修改很简单，不管是哪一个框架，所有的Text都是以脚本的形式呈现，如果你注意看Unity中的检查器，会发现他的组件名字都是“Text (Script)”，所以他的修改就是MonoBehavior的修改，还是看前面提到的那篇博客即可：Unity手游汉化笔记①：UABE+AssetStudio修改MonoBehavior

　　代码中字符串的修改就需要分脚本后端来说了

【1】Mono脚本后端

　　前面说了，Mono脚本后端把代码都放进了DLL里面，而DLL拿dnSpy直接修改就好了，很简单，具体过程我也写成了博客：Unity手游汉化笔记⑥：Mono脚本后端，修改代码中的字符串

【2】il2cpp脚本后端

　　这个就很麻烦了，代码被编译成了libil2cpp.so、global-metadata.dat两个文件，字符串在global-metadata.dat里，这个目前没有现成工具，好在il2cppdumper是开源的，里面涉及到了对字符串信息的提取，所以我自己撸了一个小工具来做修改，具体过程另外写成博客：Unity手游汉化笔记⑦：il2cpp脚本后端，修改代码中的字符串

**Unity手游汉化笔记①：UABE+AssetStudio修改MonoBehavior**

目录

一、使用工具

二、脚本在序列化文件中的表现形式

三、对MonoBehavior的参数进行修改

（1）寻找和定位

（2）修改之文本形式

（3）修改之二进制形式

MonoBehavior是开发过程中绑定在物体上的脚本被序列化后得到的，在汉化过程中，修改文字经常会需要修改这种类型的资源，字体的修改过程也会涉及到。

一、使用工具：

　　Unity版本：2018.4.5f1

　　AssetStudio（地址：<https://www.perfare.net/tag/assetstudio>）

　　UABE（地址：<https://7daystodie.com/forums/showthread.php?22675-Unity-Assets-Bundle-Extractor>）

　　il2cppdumper（地址：<https://www.perfare.net/tag/il2cppdumper>）

二、脚本在序列化文件中的表现形式

　　正向开发中，类中的public字段会放进检查器Inspector中（不考虑属性的控制的话），而在序列化的时候，首先会将类的信息序列化为一个MonoScript，绑定在场景物体上的具体对象则是序列化为MonoBehavior，一个类可以多次实例化，所以一个MonoScript会被多个MonoBehavior引用。

　　给一个具体的例子，写一个简单的类，代码如下

namespace MyNamespace

{

public class MyScript : MonoBehaviour

{

public bool myBool;

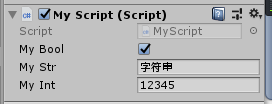
public string myStr;

public int myInt;

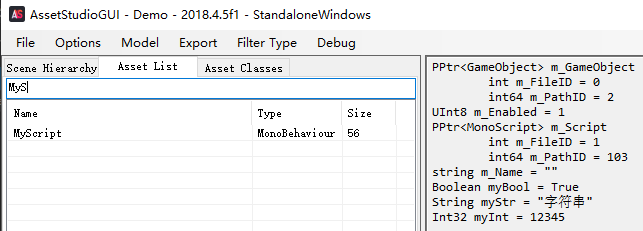
}

}

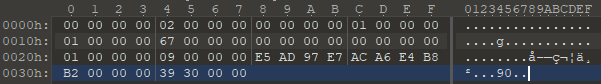
在检查器Inspector中会对应出现下面三个参数：



然后我们编译一下，再用Asset Studio预览，会找到下面这个



　　在右侧预览中，第一个指针指向的是这个MonoBehavior所绑定的场景对象，然后是Enabled，也就是检查器Inspector中名字左侧的勾，第二个指针指向的就是描述了类信息的MonoScript，值得注意的是，在MonoScript中，只描述了类的所属DLL、命名空间、类名等，并没有字段的信息，然后就是字段区域了，他的具体二进制形式也可以关注一下，用UABE导出一下二进制，内容如下



拆解一下，对应关系如下：

00 00 00 00 02 00 00 00 00 00 00 00 指向GameObject的指针

01 00 00 00 Enabled

01 00 00 00 67 00 00 00 00 00 00 00 指向MonoScript

00 00 00 00 这个MonoBehavior的名字

01 00 00 00 字段myBool，布尔值占4byte

09 00 00 00 E5 AD 97 E7 AC A6 E4 B8 B2 00 00 00 字段myStr，先长度，然后UTF-8编码，最后向4Byte对齐

39 30 00 00 字段myInt

可以看到在MonoScript的指针之后就是参数区域了，实际上如果把父类和非基本类型一并考虑进来，参数区域的序列化规则如下：

（1）先放父类的变量，再放入一个String（跟类有关系），再放入当前类内的字段；字段按照类内定义的顺序放

（2）基本类型会直接放，其中布尔值会用占4byte，可参考上面的例子

（3）String是先一个Int表示实际长度，再放入UTF-8编码的内容，最后补0，使得最后占据的总长度和4byte对齐

（4）Unity定义的资源会用PPtr来指向

（5）其他类的字段需要先有serializable属性才会进行序列化，规则类似（这还是个递归的过程）

　　如果想要深入理解，建议自己多做几个类看看

三、对MonoBehavior的参数进行修改

　　既然知道了在检查器Inspector中设置的字段会被序列化进资源文件，那么我们就可以从这里入手给他做修改。先对样例做点修改，把脚本绑到一个Text上，再在脚本里加一段代码：

void Start()

{

Text text = gameObject.GetComponent<Text>();

text.text = myStr;

}

这样运行起来之后效果如下：



接下来是修改的全过程，遇到的问题我尽可能记下来了，所以比较长

（1）寻找和定位

　　寻找主要利用AssetStudio，直接用AssetStudio加载Data文件夹，菜单栏-Filter Type-MonoBehaviour，筛选只看MonoBehavior类型，如果没有其他信息，就只能直接一个一个看了，第一次点击的时候会弹窗“Select Assembly Folder”，这里是要求选择DLL所在的文件夹。

【问题1】关于类的信息

　　前面说过了，类的信息会放在MonoScript中，但是MonoScript中只有类名，没有字段信息，所以要解析参数区域，还需要DLL，这很好理解。

　　但是涉及到DLL，又出现新的问题，Mono脚本后端的DLL就直接放在assets\bin\Data\Managed文件夹下，而il2cpp脚本后端则是编译成了so文件，所以针对il2cpp脚本后端，还需要另一个工具的帮助，il2cppdumper，他的具体操作建议看官网，利用il2cppdumper，可以得到DummyDLL，这里面没有实际的代码逻辑，但是有字段信息，足够使用了。

找到我们需要的MonoBehavior之后，右键Show original File，可以看到他在level0文件。

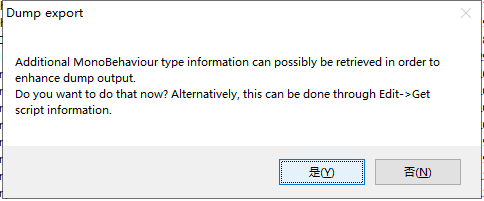
　　然后要用UABE找到具体的这个MonoBehavior，用UABE打开level0，有几个信息可以帮助我们定位到该MonoBehavior，首先是Type列，里面有完整的类名，而AssetStudio中也会有这个名字（AS左侧列表中的Name列，有名字就显示名字，没名字就显示类名）；其次是Size列，大小肯定是不会变的。找到该MonoBehavior所在的行如下



（2）修改之文本形式

　　修改靠的是UABE，因为UABE中有两种导入导出的方式，二进制RAW和文本形式，所以这里也分两种来说。

　　先用UABE导出文本，点击右侧的Export Dump，会弹窗提示：



　　这跟【问题1】是同一个问题，UABE会自动现在Data/Manager文件夹下找DLL，如果有DLL找不到，在第一次导出文本的时候就会提示这个框，要你给他找那些找不到的DLL。点击“是”，然后给他找缺少的DLL，然后就会发现一个新的问题，很多DLL根本没有，这很正常，因为Unity有很多的默认资源，这些资源里引用的类又不一定会编译进来，所以一般来说在Manager文件夹下有的、或者il2cppdumper导出来的给他选上就够了。导出来的文本如下：

0 MonoBehaviour Base

0 PPtr<GameObject> m\_GameObject

0 int m\_FileID = 0

0 SInt64 m\_PathID = 2

1 UInt8 m\_Enabled = 1

0 PPtr<MonoScript> m\_Script

0 int m\_FileID = 1

0 SInt64 m\_PathID = 103

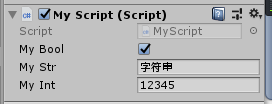
1 string m\_Name = ""

1 UInt8 myBool = 1

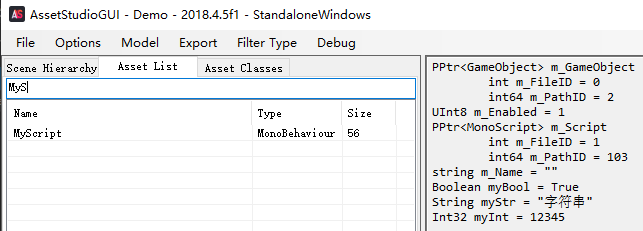
1 string myStr = "字符串"

0 int myInt = 12345

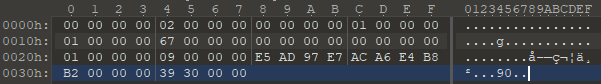
在检查器Inspector中会对应出现下面三个参数：



然后我们编译一下，再用Asset Studio预览，会找到下面这个



　　在右侧预览中，第一个指针指向的是这个MonoBehavior所绑定的场景对象，然后是Enabled，也就是检查器Inspector中名字左侧的勾，第二个指针指向的就是描述了类信息的MonoScript，值得注意的是，在MonoScript中，只描述了类的所属DLL、命名空间、类名等，并没有字段的信息，然后就是字段区域了，他的具体二进制形式也可以关注一下，用UABE导出一下二进制，内容如下



拆解一下，对应关系如下：

[复制代码](javascript:void(0);)

00 00 00 00 02 00 00 00 00 00 00 00 指向GameObject的指针

01 00 00 00 Enabled

01 00 00 00 67 00 00 00 00 00 00 00 指向MonoScript

00 00 00 00 这个MonoBehavior的名字

01 00 00 00 字段myBool，布尔值占4byte

09 00 00 00 E5 AD 97 E7 AC A6 E4 B8 B2 00 00 00 字段myStr，先长度，然后UTF-8编码，最后向4Byte对齐

39 30 00 00 字段myInt

[复制代码](javascript:void(0);)

可以看到在MonoScript的指针之后就是参数区域了，实际上如果把父类和非基本类型一并考虑进来，参数区域的序列化规则如下：

（1）先放父类的变量，再放入一个String（跟类有关系），再放入当前类内的字段；字段按照类内定义的顺序放

（2）基本类型会直接放，其中布尔值会用占4byte，可参考上面的例子

（3）String是先一个Int表示实际长度，再放入UTF-8编码的内容，最后补0，使得最后占据的总长度和4byte对齐

（4）Unity定义的资源会用PPtr来指向

（5）其他类的字段需要先有serializable属性才会进行序列化，规则类似（这还是个递归的过程）

　　如果想要深入理解，建议自己多做几个类看看

三、对MonoBehavior的参数进行修改

　　既然知道了在检查器Inspector中设置的字段会被序列化进资源文件，那么我们就可以从这里入手给他做修改。先对样例做点修改，把脚本绑到一个Text上，再在脚本里加一段代码：

void Start()

{

Text text = gameObject.GetComponent<Text>();

text.text = myStr;

}

这样运行起来之后效果如下：



接下来是修改的全过程，遇到的问题我尽可能记下来了，所以比较长

（1）寻找和定位

　　寻找主要利用AssetStudio，直接用AssetStudio加载Data文件夹，菜单栏-Filter Type-MonoBehaviour，筛选只看MonoBehavior类型，如果没有其他信息，就只能直接一个一个看了，第一次点击的时候会弹窗“Select Assembly Folder”，这里是要求选择DLL所在的文件夹。

【问题1】关于类的信息

　　前面说过了，类的信息会放在MonoScript中，但是MonoScript中只有类名，没有字段信息，所以要解析参数区域，还需要DLL，这很好理解。

　　但是涉及到DLL，又出现新的问题，Mono脚本后端的DLL就直接放在assets\bin\Data\Managed文件夹下，而il2cpp脚本后端则是编译成了so文件，所以针对il2cpp脚本后端，还需要另一个工具的帮助，il2cppdumper，他的具体操作建议看官网，利用il2cppdumper，可以得到DummyDLL，这里面没有实际的代码逻辑，但是有字段信息，足够使用了。

找到我们需要的MonoBehavior之后，右键Show original File，可以看到他在level0文件。

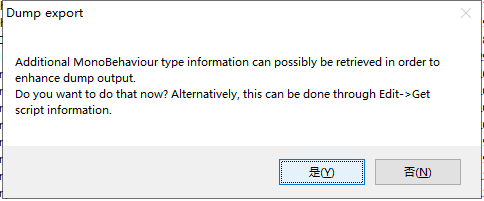
　　然后要用UABE找到具体的这个MonoBehavior，用UABE打开level0，有几个信息可以帮助我们定位到该MonoBehavior，首先是Type列，里面有完整的类名，而AssetStudio中也会有这个名字（AS左侧列表中的Name列，有名字就显示名字，没名字就显示类名）；其次是Size列，大小肯定是不会变的。找到该MonoBehavior所在的行如下



（2）修改之文本形式

　　修改靠的是UABE，因为UABE中有两种导入导出的方式，二进制RAW和文本形式，所以这里也分两种来说。

　　先用UABE导出文本，点击右侧的Export Dump，会弹窗提示：



　　这跟【问题1】是同一个问题，UABE会自动现在Data/Manager文件夹下找DLL，如果有DLL找不到，在第一次导出文本的时候就会提示这个框，要你给他找那些找不到的DLL。点击“是”，然后给他找缺少的DLL，然后就会发现一个新的问题，很多DLL根本没有，这很正常，因为Unity有很多的默认资源，这些资源里引用的类又不一定会编译进来，所以一般来说在Manager文件夹下有的、或者il2cppdumper导出来的给他选上就够了。导出来的文本如下：

[复制代码](javascript:void(0);)

0 MonoBehaviour Base

0 PPtr<GameObject> m\_GameObject

0 int m\_FileID = 0

0 SInt64 m\_PathID = 2

1 UInt8 m\_Enabled = 1

0 PPtr<MonoScript> m\_Script

0 int m\_FileID = 1

0 SInt64 m\_PathID = 103

1 string m\_Name = ""

1 UInt8 myBool = 1

1 string myStr = "字符串"

0 int myInt = 12345

[复制代码](javascript:void(0);)

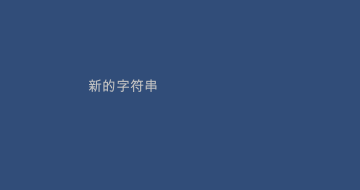
可以看到和AssetStudio中的基本一样，如果你的中文字符串是乱码，说明你没有采用UTF-8编码

【问题2】编码问题

　　Unity序列化文件中的字符串用的是UTF-8编码

　　改一下参数，再回到UABE，点击Import Dump，选择修改后的文本，可以看到Modified列中出现了星号。接下来就是保存了，点击OK，这里有个小问题，UABE只能另存为，而不能直接写回源文件，所以要保存到别的地方，再手动复制替换。

　　替换后，如果没有其他安全措施，那就可以直接运行，效果如下：



（3）修改之二进制形式

　　二进制形式其实很少用，能直接改文本根本不想写十六进制，但是以防万一还是测试了。

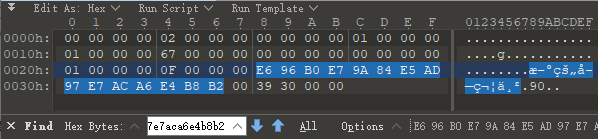
　　还是先用UABE打开level0并定位到要改的MonoBehavior，这次点击右侧的Export Raw，用010 Editor打开，工具其实随意，只要能进行十六进制的编辑就可以了。关于参数区域，前面（二、脚本在序列化文件中的表现形式）已经说过了，但是那是已知类的字段的情况，再不知道的情况怎么做呢？其实也不难，直接调成UTF-8编码，虽然都是乱码，但是字符串部分是没有乱的！用010Editor的话效果如下：



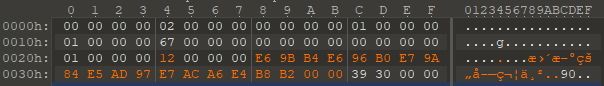
这里有两种方式回到十六进制：

　　① 010 Editor的话，选中字符串，再切回十六进制显示，选中是保持着的

　　② 在百度找个[在线编码转换](http://stool.chinaz.com/hex)，（注意【问题2】编码问题）得到这串的编码“%e6%96%b0%e7%9a%84%e5%ad%97%e7%ac%a6%e4%b8%b2”，只保留十六进制数“e696b0e79a84e5ad97e7aca6e4b8b2”，然后利用编辑器搜索功能搜索一下，一样可以定位到



　　确定了字符串的位置，接下来就是修改，回顾一下前面的规则，字符串是“先长度、再Byte数组、最后补0对齐4byte”的规则，所以前面的“0F 00 00 00”是长度，还是那个[在线编码转换](http://stool.chinaz.com/hex" \t "_blank)，得到“更新的字符串”的编码“%e6%9b%b4%e6%96%b0%e7%9a%84%e5%ad%97%e7%ac%a6%e4%b8%b2”，只保留十六进制数“e69bb4e696b0e79a84e5ad97e7aca6e4b8b2”，长度为18个Byte（十六进制就是0x12），按照规则写进去：



注意，字符串后面的参数应该向后移动，不能被覆盖。

　　改完了就是写回，回到UABE，右侧Import Raw，选中已经修改完的dat文件，保存，覆盖，再运行。效果如下：



**Unity手游汉化笔记②：使用UABE替换TTF字体**

目录

一、分析

二、思路

三、具体实践

四、总结

Unity版本：2018.4.5f1

工具：UABE（地址：<https://7daystodie.com/forums/showthread.php?22675-Unity-Assets-Bundle-Extractor>）

一、分析

　　在Unity中，可以看到一个TTF文件在导入后，可以展开成下面这样



 也就是一个大哥带两个小弟，合理猜测在这个“大哥”的二进制块中会包含指向两个“小弟”的指针

二、思路

　　首先实验只替换“大哥”，同时要保持两个指向小弟的指针不变，如果可以正常显示就结束，如果不行就把两个“小弟”在一起替换

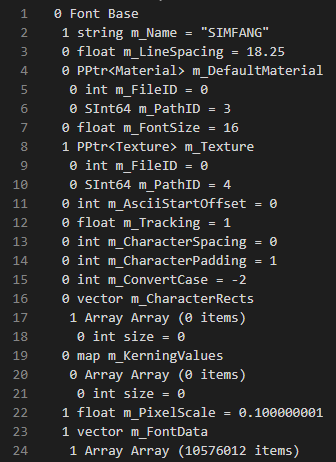
三、具体实践

【1】Demo

　　用Unity导出两个Demo，内容上都只有一个Text，但用了不同的字体，目标是把仿宋Demo的字体改成楷体

【2】用UABE导出

　　在UABE里可以直接看到字体名字的FONT类型资源，先试着DUMP出文本，仿宋的文本DUMP出来之后非常大，足足有300M，用VS Code打开（用记事本肯定打不开的），可以看到内容如下

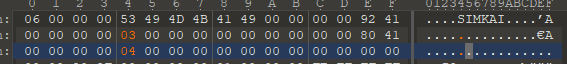


　　显然两个指针是要保留的，但是其他数据不用动，如果继续用文本，感觉实在太慢了，所以转而用RAW。于是分别DUMP出仿宋Demo和楷体Demo的字体，得到两个dat格式的文件。

【3】对二进制文件的处理

　　对楷体的dat文件进行处理，替换那两个指针，换成和仿宋的dat一样

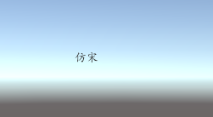
　　根据DUMP文本，首先是字体名字，一个变长的String，然后一个32位数，随后是第一个指针，然后再一个32位数，第二个指针，只需要把楷体的这两个指针改掉即可



【4】写回楷体Demo

　　用UABE打开仿宋Demo的资源文件，Import Raw，选择修改后的楷体dat文件，写回，重新打包

【5】结果



　　仿宋Demo的字体被成功替换成楷体

四、总结

　　完整的操作过程是：

　　① 分别得到新字体.dat，旧字体.dat

　　② 把新字体.dat的两个指针改成和旧字体.dat一致

　　③ 把新字体.dat写入旧字体的资源文件里去，代替旧字体

　　麻烦的点：新字体需要专门去建一个Unity工程并编译来得到二进制描述，而且版本要相近，本文demo中的指针是一个32位数表示文件、一个64位数表示对象，但是我记得某个版本有变化，所以最好版本相近甚至相同。

**Unity手游汉化笔记③：UABE替换BMFont**

目录

一、Demo

二、分析思路

三、替换

四、总结

五、补充

工具：

　　Unity版本：2018.4.5f1

　　Bitmap Font Generator：简称BMFont，就是常用的那个做位图字体的工具，地址：<http://www.angelcode.com/products/bmfont/>

　　AssetStudio：地址：<https://www.perfare.net/tag/assetstudio>

　　UABE：地址：<https://7daystodie.com/forums/showthread.php?22675-Unity-Assets-Bundle-Extractor>

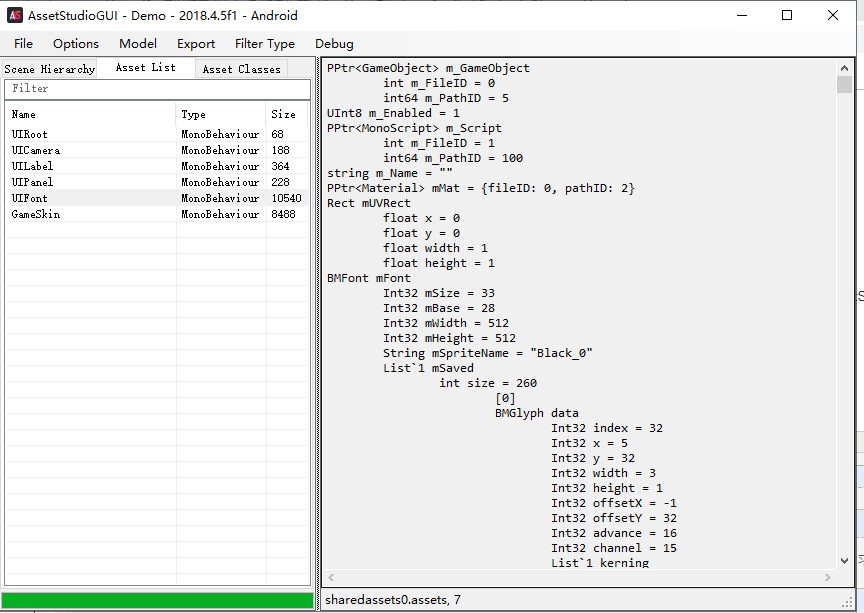
一、Demo

　　BMFont主要是NGUI在用，所以先做一个简单的Demo，用UGUI在屏幕上写一行字，采用黑体做成的BMFont，先用BMFont软件做，再导入Unity，Demo效果如下。目标是给他换成别的字体。



二、分析思路

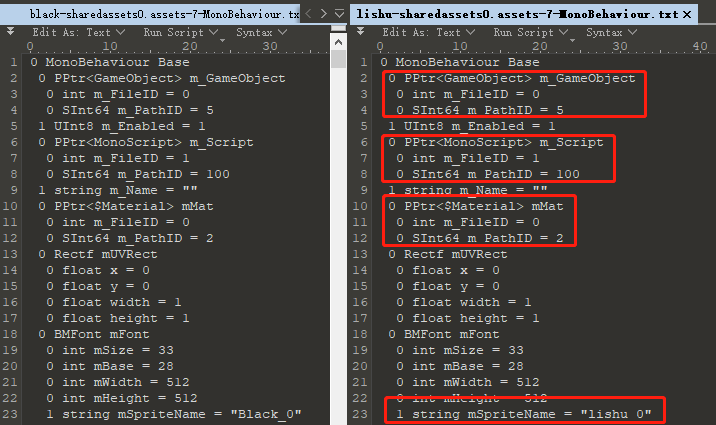
　　BMFont由两部分组成，一张图+一个文本（fnt后缀名），但是在导入Unity的时候，NGUI插件还加了一个Prefab和一个Material，不管怎么样，先用AssetStudio来预览，首先图是找到了，这个肯定是要改的，也比较简单，然后是找到了一个MonoBehavior，参数预览如下：



　　显然BMFont的文本被改写成了这个MonoBehavior，那么替换就很简单了，再做一个含有新字体的APK，先用UABE导出需要替换的这两个部分，再导回去就结束了

三、替换

　　和TTF不同，BMFont一般不会包含很多字符，所以用UABE来DUMP文本的话并不会很大。首先是这MonoBehavior的替换，分别Dump出新旧字体的参数文本，把新字体里面的指针都改成和旧字体一样，如下图红圈覆盖部分，然后把新字体的参数导入给旧字体。



　　图片替换就比较简单了，直接找到图片对应的Texture2D，右侧Plugins-Edit-Load新图，即可。

　　最后保存重新打包APK就完成了，最终效果如下



四、总结

　　完整操作：

　　① 用新字体做一个简易Demo

　　② 用新字体的MonoBehavior和图片分别替换

　　③ 实际上在MonoBehavior里面有一个指针指向了Material，如果替换之后效果不理想，可以把他也替换了，和替换MonoBehavior的时候相似，内部的指针必须保持

五、补充

　　在实际应用中，我曾经遇到了文本和图片还保留着的情况，测试过，没有了MonoBehavior要替换，取而代之的是TextAsset和Texture2D要换，这两个UABE都有插件，替换起来很方便。而且只需要用BMFont做出新字体就可以了，不需要用Unity。

**Unity手游汉化笔记④：UABE+AssetStudio替换TMPro字体**

　　TMPro，完整的叫TextMeshPro，原本是个插件，后来Unity逐渐做成了内置，主要特点是在放大之后不会模糊，逐渐成为主流，但是他的字库是需要自己制作的，而不是直接用的字体文件，随之而来的缺点就是字库没办法做得足够完整，也失去了动态字体。关于TMPro的汉化，分成两个部分，文字和字体，本文讲字体替换，文字的修改请看这篇：Unity手游汉化笔记①：UABE+AssetStudio编辑MonoBehavior类型Asset

 一、使用工具：

　　Unity版本：2018.4.5f1  
　　AssetStudio（地址：[https://www.perfare.net/tag/assetstudio](https://www.perfare.net/tag/assetstudio" \t "_blank)）  
　　UABE（地址：[https://7daystodie.com/forums/showthread.php?22675-Unity-Assets-Bundle-Extractor](https://www.perfare.net/tag/assetstudio" \t "_blank)）

二、字体替换

1.用Unity做一个简单的Demo

　　效果如下：



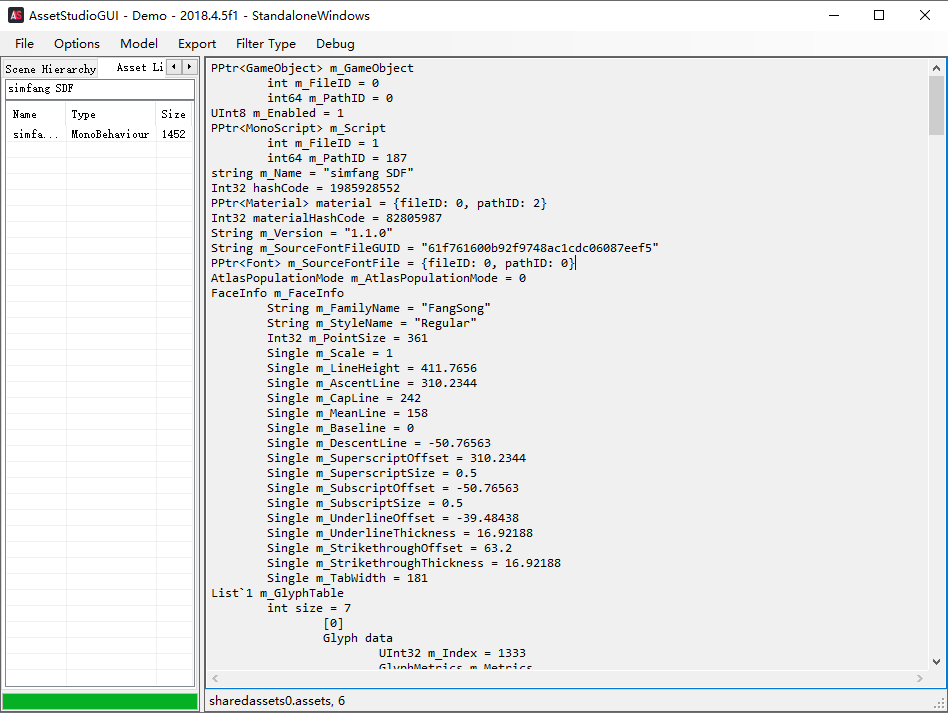
 　　做Demo的过程中你应该会发现，一个TMPro的字体以一个SDF结尾的Asset文件呈现，展开后如下：



　　实际上还有一个MonoBehaviour，整体类似于BMFont，所以我在替换的时候和BMFont思路类似，通过维持设置来保证纹理不变化，只替换MonoBehavior和图片

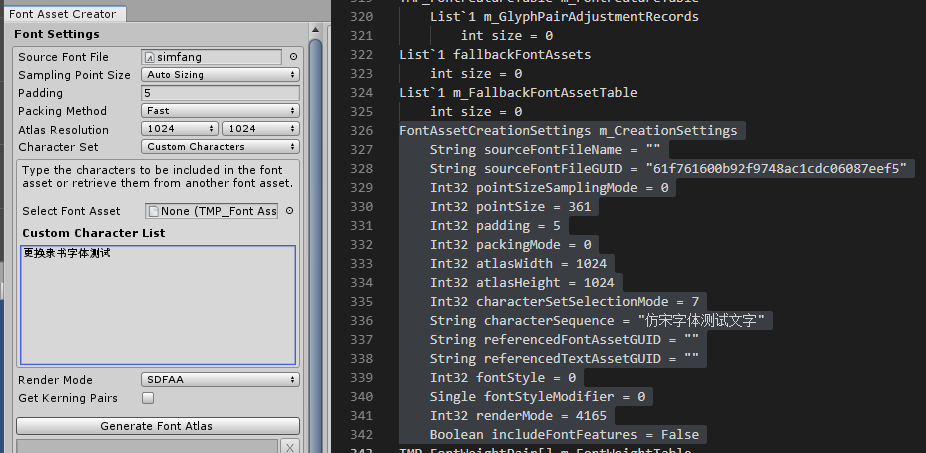
2.获得新字体

　　先用Asset Studio来预览，打开并加载Data文件夹（Android的话在Asset/bin/Data），在MonoBehavior分类下，会找到一个simfang SDF，这就是我们的字体最关键的部分了，一般TMPro的字体都会以SDF结尾，他的参数列表很长，如下：



　　我们根据这里面的一些信息来制作新字体的Demo，使用相同版本的Unity来制作。

　　在Unity中，先从菜单栏-Window-TextMeshPro-FontAssetCreator调出Font Asset Creator，这里的设置在MonoBehavior中体现在了m\_CreationSettings部分，可以把Asset Studio中的参数列表放进一个编辑器里，用搜索功能找到他，我们要保持设置，具体如下：



　　Sampling Point Size 对应右侧pointSizeSamplingMode和pointSize

　　Padding对应右侧padding

　　Packing Method对应右侧packingMode

　　Atlas Resolution对应atlasWidth和atlasHeight

　　Character Set及其随后小方框对应右侧characterSetSelectionMode、characterSequence、referencedFontAssetGUI、referencedTextAssetGUID

　　Render Mode对应右侧renderMode（看不懂对应关系，好在一般都是默认值）

　　Get Kerning Pairs对应右侧includeFonrFeatures

　　Generate Font Atlas后就得到了图片，Save之后就得到了Asset文件，随便弄个Text，把字体绑上去，然后生成整个项目

3.替换

　　剩下的步骤就和替换BMFont的时候基本一样了，用UABE导出新旧字体的图片、MonoBehavior参数文本，图片直接导回去，MonoBehavior要做以下修改：

　　① 指针部分，把参数文本复制到带搜索功能的编辑器里，先搜索关键词“pathid”（关闭大小写匹配），有几个部分：

　　　　最开头的m\_GameObject和m\_Script：这是MonoBehavior都需要的，应该保持旧的数值

　　　　material：这是一个关键的东西，字体绑定的纹理，这里也要维持旧的数值

　　　　m\_AtlasTextures：指向图片，也要维持旧的数值

　　　　其他的匹配项都是0，不用管。

　　② 哈希值，搜索关键词“hash”和“guid”，各有两处匹配，这两个我也选择了保留旧的数值，以免出错

4.结果



# Unity手游汉化笔记⑤：图片和TextAsset替换

用处：

　　TextAsset的最常见用途之一是多语言，所以汉化过程中有时候会改到这个东西

　　图片就不用说了，不是所有的文字都是用Text显示的，汉化过程中多少会有图片出现

工具：

　　AssetStudio（地址：<https://www.perfare.net/tag/assetstudio>）

　　UABE（地址：<https://7daystodie.com/forums/showthread.php?22675-Unity-Assets-Bundle-Extractor>）

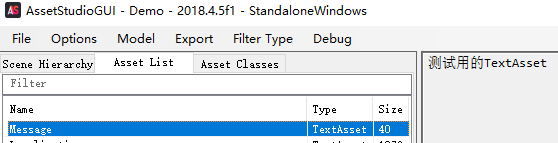
一、TextAsset

Demo：用Unity做的简单例子，读取文本中的文字并直接显示

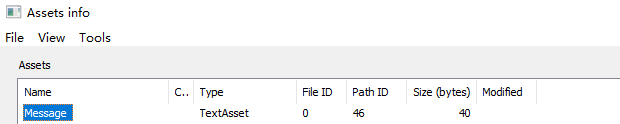


修改过程：

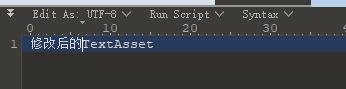
　　先用Asset Studio加载assets/bin/Data文件夹，菜单栏Filter Type-TextAsset，在右侧预览中可以看到内容，从而确定我们要改的TextAsset是哪一个



　　右键Show original File，得知在文件resources.asset中，用UABE打开这个文件，定位到该TextAsset，通过名字、类型、大小，很容易定位到的



　　在UABE右侧点击Plugins，弹出的窗口里两个选项，导入和导出，选择Export to .txt，导出为txt文件，用带有**【UTF-8编码】**的编辑器（我用的010 Editor）打开，然后修改，保存，效果如下：



　　回到UABE，再次点击Plugins，选择Import from .txt，选择修改后的txt文件，点击OK，保存。注意UABE只能另存为，无法直接写回源文件，所以要保存到另一个位置，然后手动复制替换。

　　再次运行，效果如下：



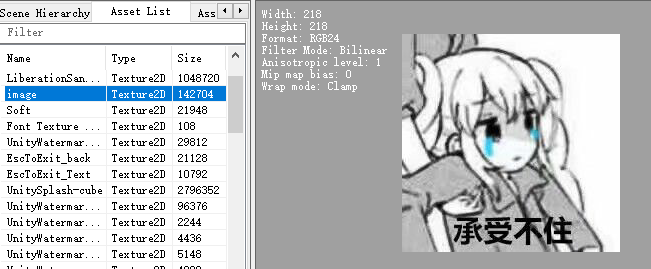
二、图片替换

Demo：用Unity做的简单例子，就UI/Image上一张图片直接显示，没有什么骚操作，效果如下

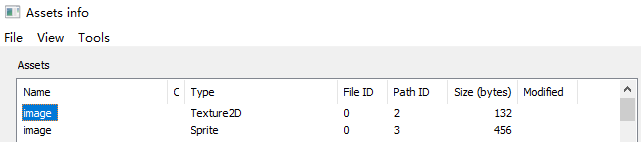


修改过程：

　　用AssetStudio加载assets/bin/Data文件夹，菜单栏Filter Type-Texture2D，在右侧预览中可以看到内容，从而确定我们要改的图片是哪一个



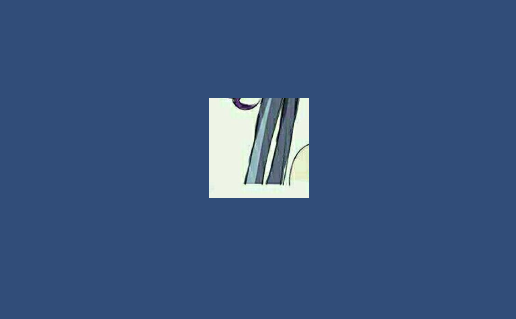
　　右键Show original File，得知源文件为sharedassets0.asset，用UABE打开它，定位到该文件，同样主要靠名字、类型可以找到，大小的话两边稍稍有点不一样，但是名字加类型够了



 　　在UABE右侧，点击Plugins，如果担心定位错了，可以选择Export to png，检查一下。修改的话选择Edit，再选择Load，导入准备好的新图片，关于图片，最好是尺寸一致的。

　　如果你仔细看，会发现还有一个同名的Sprite资源，他会引用Texture2D的一部分，UI/Image直接使用的是Sprite，而不是Texture2D，也就是说，他们的职责分配是，Texture2D是源图，Sprite选择图片的区域，Image是显示。有缘查一查正向开发中Sprite的应用，可以得到更多信息。

　　Edit完，点击OK保存，覆盖源文件，重新启动。这里我用了一个尺寸大得多的图，它的效果如下：



　　可以看到只有一部分被显示了，如果不嫌麻烦，可以再去改一改Sprite里的参数，同样用UABE，但是最方便还是保持尺寸相同。

**Unity手游汉化笔记⑥：Mono脚本后端，修改代码中的字符串**

　　在汉化过程中，不可避免的，会遇到要修改代码中字符串的情况，而Android端的代码有两种脚本后端，Mono和il2cpp，本文针对Mono来讲。首先比较基本的，Mono脚本后端会把代码编译进DLL里，然后放在assets\bin\Data\Managed文件夹下，在没有其他安全措施的情况下，我们可以直接用dnSpy来处理

　　dnSpy：<https://github.com/0xd4d/dnSpy>

　　一个开源工具，主要用于C#代码的逆向，目前已经支持直接编辑DLL中的代码

下面开始介绍具体过程：

一、样例

　　一个简易的样例，页面上一个UI/Text，绑定一个脚本，脚本代码如下

[复制代码](javascript:void(0);)

1 public class MyScript : MonoBehaviour

2 {

3 void Start()

4 {

5 var text = gameObject.GetComponent<Text>();

6 text.text = "my string";

7 }

8 }

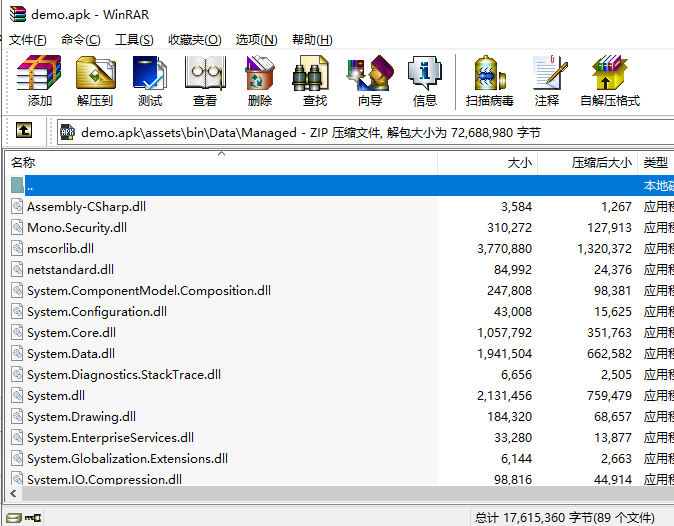
[复制代码](javascript:void(0);)

运行效果如下：

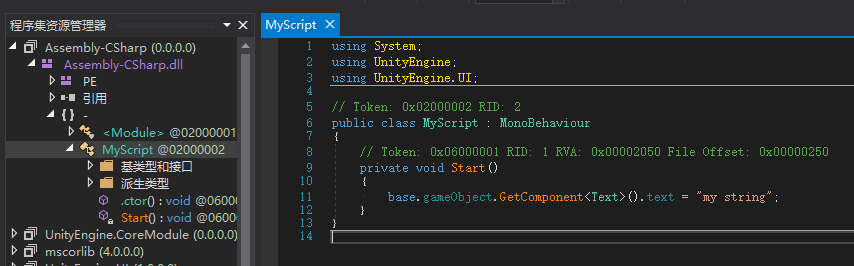


二、修改

　　因为我们不考虑其他安全措施，所以对apk的处理就简单粗暴一下，直接用解压软件打开，把assets\bin\Data\Managed文件夹整个拖出来，可以看到他的内容如下：



一般来说，开发者开发的代码在Assembly-CSharp.dll中，但是还是要把整个文件夹拖出来，因为会有引用关系的。把所有的DLL都放进dnSpy里，我的Demo很简单，所以直接展开Assembly-CSharp.dll的话，就可以直接看到反编译后的代码了，效果如下：



　　可以看到和源码甚至相差无几，直接在方法内的代码处右键，这里右键菜单里有两个“编辑函数”，齿轮的那个是方法的一些属性用的，我们用【另一个】，在弹出的编辑框里直接改代码就好了，和正向开发中的一样，然后点击右下角【编译】按钮。

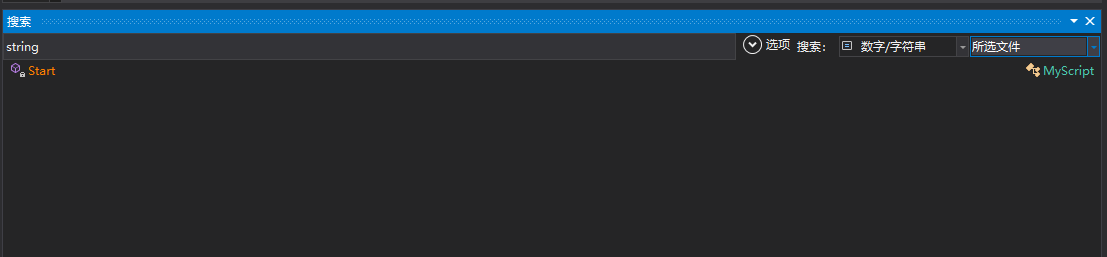
　　然后要保存回DLL，菜单栏-文件-保存模块，确定。

　　最后压回apk，因为DEMO没有安全措施，所以直接压缩后就可以安装了，运行起来，效果如下：



三、补充

　　Demo因为太简单了，直接展开Assembly-CSharp.dll就找到了要修改的类，但是实际上遇到的游戏都不会这么简单，所以搜索上也是一个麻烦的事情，dnSpy中按Ctrl+Shift+K，可以弹出搜索框，类型选择【数字/字符串】，范围选【所选文件】，可以很快搜索到。



　　可以看到，如果没有安全措施，用的又是Mono后端，那么你的代码可以说是基本裸奔的状态，要修改简直易如反掌，所以Mono脚本后端现在多被用在开发过程中快速迭代使用，而不用于最终上线版本。

**Unity手游汉化笔记⑦：il2cpp脚本后端，修改代码中的字符串**

　　对于Mono脚本后端，代码被编译进DLL，如果没有安全措施，利用dnSpy就可以很轻松的修改其中的字符串。

　　对于il2cpp脚本后端，代码中的字符都在global-metadata.dat文件中。用[il2cppdumper](https://www.perfare.net/tag/il2cppdumper" \t "_blank)，通过libil2cpp.so和global-metadata.dat导出很多信息，其中stringliteral.json文件中就有代码中的字符串，但是这个工具是用来dump信息的，不是用来修改的，所以我参考他的源码，做了一个简单的工具来对代码进行修改，下面是操作过程。

一、Demo制作

　　Demo很简单，页面上添加一个UI/Text，绑定一个脚本，然后导出apk之前，记得先在Player Settings里面设置好使用il2cpp脚本后端。

脚本代码如下：

1 public class MyScript : MonoBehaviour {

2 void Start () {

3 var text = gameObject.GetComponent<Text>();

4 text.text = "Test String";

5 }

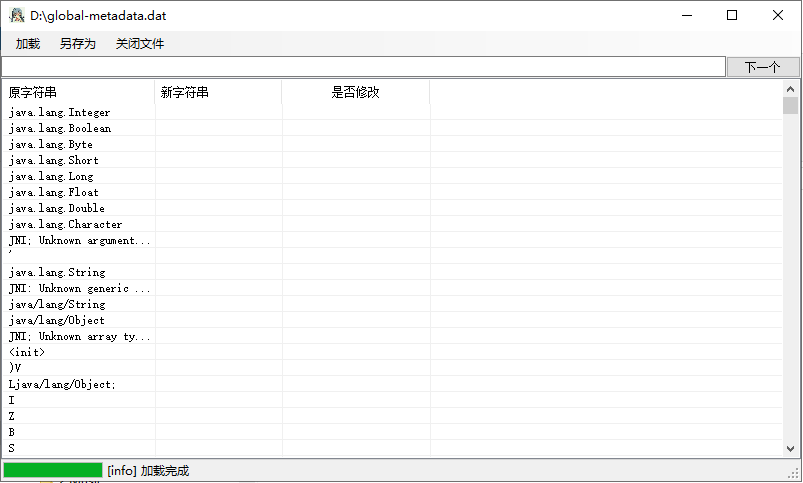
6 }

运行效果（没有截全屏）如下：

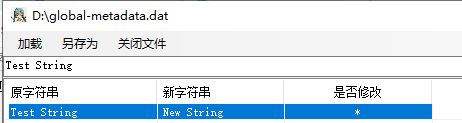


二、修改

　　Demo是Unity直接导出的，没有安全措施，直接用WinRAR打开，在\assets\bin\Data\Managed\Metadata下找到global-metadata.dat文件，用我自己做的一个小工具（<https://github.com/JeremieCHN/MetaDataStringEditor>）打开他，效果如下：



上面的输入框是搜索用的，搜索要改的字符串，回车或者点击下一个找到它，然后右键该字符串，编辑（可以中文，但要考虑字体支不支持），保存，效果如下

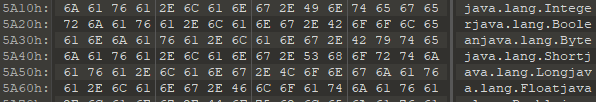


我没有做成写回源文件的方式，要另存为，手动覆盖一下，然后装回去。最终效果（没有截全屏）如下：



三、基本原理

　　global-metadata.dat里面的信息很多，在il2cppdumper里面可以看到，字符串的区域有两个，代码中的字符串在其中一个，这个区域的特点是这样，首先在文件头部有一个列表，里面放了每一个字符串的长度和在数据区的偏移量，同时在数据区，字符串是紧凑放置的，也就是说字符串和字符串之间没有分割符号，用010 Editor打开文件的话，可以找到字符串的区域，



而另一个区域则是每一个字符串的结尾都会有一个\0，两个区域的功能应该是不同的，我的猜测是，我改的这个区域是代码中的字符串，而另一个区域则是类名、方法名等。

　　具体修改过程也比较简单，对于头部的列表，因为修改前后字符串的数量是不变的，所以这个列表所占据的空间也是不变的，数据区的话大概率是会变的，我的做法是，在修改之后先计算数据区的长度，如果可以放入原来数据区所在的空间，就直接放，否则在文件尾另外开辟一块空间来放，这样我就不需要去重排后面的其他数据了。关于字符串的编码，还是Unity常用的UTF-8编码。