**换装方法**

Posted on 2013年04月17日 by U3d / [Unity3D 基础教程](http://www.unitymanual.com/category/manual/unity3d-%e5%9f%ba%e7%a1%80%e6%95%99%e7%a8%8b)/被围观 270 次

游戏内的角色，能够像纸娃娃换装那样子让玩家可以为自己的角色改变外观，一直是相当受欢迎的功能；一般而言，我们建好的 3D 模型，如果要将其中一个部位换成另外一个形状，最直接的就是将该物件部位的 Mesh 替换掉，那么外观就改变了，但这种方法如果运用在需要做动作的模型上，将发现被置换掉的部位不会正常动作，更糟的状况可能连模型显示的位置及方向都是错误的，所以，直接变更 Mesh 的方法只适用于静态模型物件，为此，我们必须找出更深入的方法来做换装的功能，幸好，此部份 Unity 官方已经有提供相关范例可以参考。

Unity 官方提供的人物换装范例可以从官网下载 Character Customization，或是开启 Unity 编辑器的 Window > Asset Store 在 Complete Projects > Tutorials 找到 Character Customization 下载并汇入到自己的专案中。这个范例提供相当完整的示范，而且考虑到实作于游戏中时，不可能一次把全部的资源都载入，所以将模型、材质、纹理等资源都包成 Asset bundle，只在要使用到时才载入需要的部份；也因为如此，对於不了解 Asset bundle 的情况下，要透过这个范例来直接学习换装也相对变得困难；另外，范例中也对资源规范了特定形式的命名规则，主要也是为了建立 Asset bundle 内容资料及从 Asset bundle 取出资源而设计，在不了解这些规则之前，想要透过此范例学习换装，有一定程度的难度；当然，如果愿意使用与范例中 characters 目录中两个人物的模型、材质、纹理完全相同的命名规则及档案配置方式，几乎可以直接套用到自己的游戏中，而不太需要了解内部的运作方式。

在 Unity 开启 Asset store，在Asset store 中找出 Character Customization 范例。

虽然，官方这个范例能够直接套用，但制作游戏常会有不同的客制化需求，如果不了解相关原理和流程的话，可能就无法自由灵活的运用，所以，以下将利用这个范例并排除掉 Asset bundle 的部份，直接在场景中完成纸娃娃换装的方法。

首先，先来看看模型的结构，从 Projects 视窗将 CharacterCustomization > characters > Female > Female 拉到场景中，在 Hierarchy 视窗将物件展开来，会发现几个名称相同并使用数字区别的物件，它们分别代表人物各部位的模型，由此可知，整个人物模型档包含多个相同部位的模型，而 Famale\_Hips 则是整个人物的骨架结构，人物的动作则是设置在顶层物件(Female)的 Animation，所以这个模型档是个模型资源，而不是实际上要放到在场景中的目标物件。

每个部位有多个模型物件

了解模型档内容後，接下来先建立一个名为 TestChar 的 C# 程式档用来控制换装，为了方便测试，在 Projects 视窗将 CharacterCustomization > characters > Female > Female@walk 的 Animation Wrap Mode 改为 Loop，并在程式档的 Start() 内加入 animation.Play(“walk”)，如此在执行状态将会使人物不断的做走路的动作。

选择 Female@walk

Animation Wrap Mode 选择 Loop

Unity 官方这个范例，说穿了就是将模型档做为来源模型资源，然後再依照需求将各部位重新组合成一个新的目标模型，所以我们直接将人物模型 Female 拉两个到场景中，分别为它们命名为 Source 及 Target，依照以下步骤做些准备动作：

从 Projects 视窗将 CharacterCustomization > characters > Female > Per Texture Materials 依照名称把适当的材质球(Material) 拉给 Source 的每个部位(不包含 Female\_Hips 及其子物件)。

Source 物件是做为来源资源使用，实际在场景中不需要运作，所以直接点选 Source 物件并将 Inspector 视窗中 Source 名称栏位前的方框取消勾选来将它关闭。

Source 前面的方框取消勾选，取消勾选后会弹出对话视窗询问是否希望关闭全部的子物件，点击 Deactivate Chidren。把 Target 物件中除了 Famale\_Hips 以外的子物件全部删除。把 TestChar 程式档拉给 Target 物件。Source 中的各部位名称应该都要有编号(例如 face-1 )，如果没有的话，加上编号。

完成以上的准备动作，接下来就要开始来写程式了，程式主要工作是先将 Source 中每个物件的 SkinnedMeshRenderer 资料取出储存在 data 变数中，data 的内容则是依照部位分类，接下来在 Target 加入 SkinnedMeshRenderer ，然后每个部位取出一个指定的资料，利用 CombineInstance class 及 Mesh.CombineMeshes() 将各部位模型合并，同时也重新排列材质，然後依照取出的 SkinnedMeshRenderer 的 bone 的名称，找到与 Target 的 Female\_Hips 子物件内名称相对应的物件重建骨架列表，最后将这些重新组合建立的资料给 Target 的 SkinnedMeshRenderer，如此就可完成换装的动作，以下为程式说明：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 001 | *//来源模型资源的物件* |
| 002 |  |
| 003 | **public** Transform source; |
| 004 |  |
| 005 | *//目标物件* |
| 006 |  |
| 007 | **public** Transform target; |
| 008 |  |
| 009 | *//模型资源资料* |
| 010 |  |
| 011 | **private** Dictionary<**string** , Dictionary<**string**,SkinnedMeshRenderer>> data = new Dictionary<**string**, Dictionary<**string**,SkinnedMeshRenderer>>(); |
| 012 |  |
| 013 | **void** Start () { |
| 014 |  |
| 015 | *//从来源模型资源取出各部位的 SkinnedMeshRenderer* |
| 016 |  |
| 017 | SkinnedMeshRenderer[] parts = source.GetComponentsInChildren<SkinnedMeshRenderer>(**true**); |
| 018 |  |
| 019 | **foreach**(SkinnedMeshRenderer part **in** parts){ |
| 020 |  |
| 021 | *//利用 ? 字元分隔档名做为资料结构的 key，档名为 部位?编号 储存为 [部位][编号]=SkinnedMeshRenderer资料* |
| 022 |  |
| 023 | **string**[] partName = part.name.Split('?'); |
| 024 |  |
| 025 | *// 在 data 加入资料* |
| 026 |  |
| 027 | **if**(!data.ContainsKey(partName[0])) data.**Add**(partName[0] , new Dictionary<**string**,SkinnedMeshRenderer>()); |
| 028 |  |
| 029 | data[partName[0]].**Add**(partName[1],part); |
| 030 |  |
| 031 | } |
| 032 |  |
| 033 | *//目标物件加入 SkinnedMeshRenderer* |
| 034 |  |
| 035 | SkinnedMeshRenderer targetSmr = target.gameObject.AddComponent<SkinnedMeshRenderer>(); |
| 036 |  |
| 037 | *//从目标物件取得骨架资料 (Female\_Hips 的全部物件)* |
| 038 |  |
| 039 | Transform[] hips = target.GetComponentsInChildren<Transform>(); |
| 040 |  |
| 041 | */\*\* 开始 重组模型 \*/* |
| 042 |  |
| 043 | *//初始化资料列表* |
| 044 |  |
| 045 | List<CombineInstance> combineInstances = new List<CombineInstance>(); |
| 046 |  |
| 047 | List<Material> materials = new List<Material>(); |
| 048 |  |
| 049 | List<Transform> bones = new List<Transform>(); |
| 050 |  |
| 051 | **foreach**(KeyValuePair<**string** , Dictionary<**string**,SkinnedMeshRenderer>> \_part **in** data){ |
| 052 |  |
| 053 | *//从资料中取得各部位指定编号的 SkinnedMeshRenderer* |
| 054 |  |
| 055 | SkinnedMeshRenderer smr = new SkinnedMeshRenderer(); |
| 056 |  |
| 057 | **switch**(\_part.Key){ |
| 058 |  |
| 059 | **case** “eyes”: |
| 060 |  |
| 061 | smr = \_part.**Value**[“1”]; |
| 062 |  |
| 063 | **break**; |
| 064 |  |
| 065 | **case** “face”: |
| 066 |  |
| 067 | smr = \_part.**Value**[“1”]; |
| 068 |  |
| 069 | **break**; |
| 070 |  |
| 071 | **case** “hair”: |
| 072 |  |
| 073 | smr = \_part.**Value**[“1”]; |
| 074 |  |
| 075 | **break**; |
| 076 |  |
| 077 | **case** “pants”: |
| 078 |  |
| 079 | smr = \_part.**Value**[“1”]; |
| 080 |  |
| 081 | **break**; |
| 082 |  |
| 083 | **case** “shoes”: |
| 084 |  |
| 085 | smr = \_part.**Value**[“1”]; |
| 086 |  |
| 087 | **break**; |
| 088 |  |
| 089 | **case** “top”: |
| 090 |  |
| 091 | smr = \_part.**Value**[“1”]; |
| 092 |  |
| 093 | **break**; |
| 094 |  |
| 095 | } |
| 096 |  |
| 097 | *//准备要组合的 Mesh* |
| 098 |  |
| 099 | CombineInstance ci = new CombineInstance(); |
| 100 |  |
| 101 | ci.mesh = smr.sharedMesh; |
| 102 |  |
| 103 | combineInstances.**Add**(ci); |
| 104 |  |
| 105 | *//排列新的材质列表* |
| 106 |  |
| 107 | materials.AddRange(smr.materials); |
| 108 |  |
| 109 | *//取得相对应名称的骨架物件来建立新的骨架列表* |
| 110 |  |
| 111 | **foreach**(Transform bone **in** smr.bones){ |
| 112 |  |
| 113 | **foreach**(Transform hip **in** hips){ |
| 114 |  |
| 115 | **if**(hip.name != bone.name) **continue**; |
| 116 |  |
| 117 | bones.**Add**(hip); |
| 118 |  |
| 119 | **break**; |
| 120 |  |
| 121 | } |
| 122 |  |
| 123 | } |
| 124 |  |
| 125 | } |
| 126 |  |
| 127 | *//合并 Mesh 并写入至 Target 的 SkinnedMeshRenderer* |
| 128 |  |
| 129 | targetSmr.sharedMesh = new Mesh(); |
| 130 |  |
| 131 | targetSmr.sharedMesh.CombineMeshes(combineInstances.ToArray() , **false** , **false**); |
| 132 |  |
| 133 | *// Target 的 SkinnedMeshRenderer 写入新骨架列表* |
| 134 |  |
| 135 | targetSmr.bones = bones.ToArray(); |
| 136 |  |
| 137 | *// Target 的 SkinnedMeshRenderer 写入新材质列表* |
| 138 |  |
| 139 | targetSmr.materials = materials.ToArray(); |
| 140 |  |
| 141 | */\*\* 重组模型 结束 \*/* |
| 142 |  |
| 143 | *//指定播放走路动作* |
| 144 |  |
| 145 | animation.Play(“walk”); |
| 146 |  |
| 147 | } |
| 148 |  |

写完程式后，记得把场景中的 Source 及 Target 两个物件分别拉给附属在 Target 物件上的 TestChar script 的 source 及 target 栏位；

程式动作都在 Start() 内进行，是因为最初目标物件并没有模型等资料，所以要先依照指定的各部位资料把人物建立出来并使它动作，而 smr = \_part.Value[“1”]; 的 “1” 则是表示指定此部位的 “1” 模型资料，所以只要改变各部位的这个值，就能为人物配置不同的造型，当然，前题是来源模型资源必须要有这个编号的物件才行；以上程式码主要是测试及解说流程用，在实作上应该把标示 /\*\* 重组模型 \*/ 这一段程式独立出来，在需要换装时，给予各部位指定编号来执行。

以上是 Unity 官方范例中处理换装的方法，它把各部位模型、材质等资料重新组合合并成单一的模型并重建骨架列表，如此即使看起来人物身上有其中一个部位被置换了，仍能持续正常动作；当查看 Target 物件时会发现它的子物件仍然维持不变，只有 Target 物件本身在 Inspector 视窗中的 Component 多出了 Skinned Mesh Renderer 及各部位的 Material，如果查看 SkinnedMeshRenderer 的 Mesh 栏位 也会发现看不到任何的 Mesh。

Target 物件的内容

这种做法的来源模型与材质数量必须相对应，否则模型的贴图将会变得不正常，也就是说如果裤子的 material 有两个，其他部位的 materail 只有一个，那麽结果模型上的贴图将与预期的不同；为了使各部位的 material 使用上更为弹性，前面的程式将做些修改，使它的各部位都是独立的 GameObject，如下所示：

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 001 | *//来源模型资源的物件* |
| 002 |  |
| 003 | **public** Transform source; |
| 004 |  |
| 005 | *//目标物件* |
| 006 |  |
| 007 | **public** Transform target; |
| 008 |  |
| 009 | *//模型资源资料* |
| 010 |  |
| 011 | **private** Dictionary<**string** , Dictionary<**string**,Transform>> data = new Dictionary<**string**, Dictionary<**string**,Transform>>(); |
| 012 |  |
| 013 | *//目标物件的骨架* |
| 014 |  |
| 015 | **private** Transform[] hips; |
| 016 |  |
| 017 | *//目标物件各部位的 SkinnedMeshRenderer 资料(参照)* |
| 018 |  |
| 019 | **private** Dictionary<**string** , SkinnedMeshRenderer> targetSmr = new Dictionary<**string**, SkinnedMeshRenderer>(); |
| 020 |  |
| 021 | **void** Start () { |
| 022 |  |
| 023 | *//从来源模型资源取出各部位的 SkinnedMeshRenderer* |
| 024 |  |
| 025 | SkinnedMeshRenderer[] parts = source.GetComponentsInChildren<SkinnedMeshRenderer>(**true**); |
| 026 |  |
| 027 | **foreach**(SkinnedMeshRenderer part **in** parts){ |
| 028 |  |
| 029 | *//利用 ? 字元分隔档名做为资料结构的 key，档名为 部位?编号 储存为 [部位][编号]=Transform资料* |
| 030 |  |
| 031 | **string**[] partName = part.name.Split('?'); |
| 032 |  |
| 033 | *// 在 data 加入资料* |
| 034 |  |
| 035 | **if**(!data.ContainsKey(partName[0])){ |
| 036 |  |
| 037 | data.**Add**(partName[0] , new Dictionary<**string**,Transform>()); |
| 038 |  |
| 039 | *//建立新的 GameObject 并使用部位名称来命名，指定为目标物件的子物件* |
| 040 |  |
| 041 | GameObject partObj = new GameObject(); |
| 042 |  |
| 043 | partObj.name = partName[0]; |
| 044 |  |
| 045 | partObj.transform.parent = target; |
| 046 |  |
| 047 | *//为新建立的 GameObject 加入 SkinnedMeshRenderer，并将此 SkinnedMeshRenderer 存入 targetSmr* |
| 048 |  |
| 049 | targetSmr.**Add**(partName[0] , partObj.AddComponent<SkinnedMeshRenderer>()); |
| 050 |  |
| 051 | } |
| 052 |  |
| 053 | data[partName[0]].**Add**(partName[1],part.transform); |
| 054 |  |
| 055 | } |
| 056 |  |
| 057 | *//从目标物件取得骨架资料 (Female\_Hips 的全部物件)* |
| 058 |  |
| 059 | hips = target.GetComponentsInChildren<Transform>(); |
| 060 |  |
| 061 | */\*\* 开始 重组模型 \*/* |
| 062 |  |
| 063 | **foreach**(KeyValuePair<**string** , Dictionary<**string**,Transform>> \_part **in** data){ |
| 064 |  |
| 065 | **switch**(\_part.Key){ |
| 066 |  |
| 067 | **case** “eyes”: |
| 068 |  |
| 069 | ChangePart(“eyes” , “1”); |
| 070 |  |
| 071 | **break**; |
| 072 |  |
| 073 | **case** “face”: |
| 074 |  |
| 075 | ChangePart(“face” , “1”); |
| 076 |  |
| 077 | **break**; |
| 078 |  |
| 079 | **case** “hair”: |
| 080 |  |
| 081 | ChangePart(“hair” , “1”); |
| 082 |  |
| 083 | **break**; |
| 084 |  |
| 085 | **case** “pants”: |
| 086 |  |
| 087 | ChangePart(“pants” , “1”); |
| 088 |  |
| 089 | **break**; |
| 090 |  |
| 091 | **case** “shoes”: |
| 092 |  |
| 093 | ChangePart(“shoes” , “1”); |
| 094 |  |
| 095 | **break**; |
| 096 |  |
| 097 | **case** “top”: |
| 098 |  |
| 099 | ChangePart(“top” , “1”); |
| 100 |  |
| 101 | **break**; |
| 102 |  |
| 103 | } |
| 104 |  |
| 105 | } |
| 106 |  |
| 107 | */\*\* 重组模型 结束 \*/* |
| 108 |  |
| 109 | *//指定播放走路动作* |
| 110 |  |
| 111 | target.animation.Play(“walk”); |
| 112 |  |
| 113 | } |
| 114 |  |
| 115 | **private** **void** ChangePart(**string** part , **string** item){ |
| 116 |  |
| 117 | *//从资料中取得各部位指定编号的 SkinnedMeshRenderer* |
| 118 |  |
| 119 | SkinnedMeshRenderer smr = data[part][item].GetComponent<SkinnedMeshRenderer>(); |
| 120 |  |
| 121 | *//取得相对应名称的骨架物件来建立新的骨架列表* |
| 122 |  |
| 123 | List<Transform> bones = new List<Transform>(); |
| 124 |  |
| 125 | **foreach**(Transform bone **in** smr.bones){ |
| 126 |  |
| 127 | **foreach**(Transform hip **in** hips){ |
| 128 |  |
| 129 | **if**(hip.name != bone.name) **continue**; |
| 130 |  |
| 131 | bones.**Add**(hip); |
| 132 |  |
| 133 | **break**; |
| 134 |  |
| 135 | } |
| 136 |  |
| 137 | } |
| 138 |  |
| 139 | *// 更新指定部位 GameObject 的 SkinnedMeshRenderer 内容* |
| 140 |  |
| 141 | targetSmr[part].sharedMesh = smr.sharedMesh; |
| 142 |  |
| 143 | targetSmr[part].bones = bones.ToArray(); |
| 144 |  |
| 145 | targetSmr[part].materials = smr.materials; |
| 146 |  |
| 147 | } |
| 148 |  |

在建立 data 变数内容时，同时为每个部位建立 GameObject，另外也把变更部位内容的程式码独立出来为 ChangePart 方法，如此在每次需要变更该部位时，只要指定部位名及编号就可以直接为该部位换装，而不需要将每个部位都重建；因为每个部位都是 GameObject 实体，我们在 Hierarchy 或 Scene 视窗中点选该部位也可以清楚的从 Inspector 视窗中看到此部位内容，正因如此，每个部位就可以自由配置 Material 的数量了。

从以上程式中会发现换装除了把 Mesh 和 Material 从来源取出给目标置换之外，有个关键的地方是重建骨架列表，为什麽要重建骨架列表呢？最主要是变更 Mesh 之後的 SkinnedMeshRenderer.bones 及 SkinnedMeshRenderer.sharedMesh.bindposes 数量有可能会不同而产生错误讯息 Number of bind poses doesn't match number of bones in skinned mesh，即使数量相同而没有错误讯息，SkinnedMeshRenderer.sharedMesh.bindposes 内的 Matrix4x4[] 资料也会因为数值不正确而发生执行期模型扭曲成奇怪形状的问题；这部份可以将 Female 模型档汇入到 3DS Max 中查看，以鞋子为例，在 Modify 视窗中，可以很明显看出 shoes-1 和 shoes-2 的 Bones 列表内容是不同的，所以在为模型物件变更 Mesh 的同时必须重建骨架列表。

以上的说明主要是用于了解换装所需要的做法，实作时，不太可能把游戏中的角色全身各部位的模型资料全部都载入做为来源资料，例如游戏中的武器有100种，角色背包中有3种武器，但为了换装却把100种武器都载入到游戏中，而实际上此角色最多也只能变换背包中的3种武器而已，这样无疑是浪费了97种武器所占用的资源；所以在了解如何换装後，实作时应该尽量像官方范例那样把来源资源包装起来，只取出需要的资源来进行换装。