Unity3d热更新

现在的手游安装有几种方式。一种是安装的时候就把程序和资源安装到本地。另外一种是只安装程序和少量的必要资源，然后在启动的时候再把缺少的资源下载完整。上述的两种安装模式，在更新资源上本质都是相同的。都是比较服务器资源的版本和本地资源的版本，以确定哪些资源要下载（包括需要更新的和新增的）。

试想一下，如果每次出一个新的版本，都需要玩家去应用商店去下载，在这个21世纪，流量紧缺的情况下，谁会愿意去为了一个小小的更新然后去下载一个完整的包呢？更有甚者，每当游戏出一个活动，那么得重新下多少次？因此热更新就这样应运而生。

而热更新一开始盛行于页游当中，后来端游也采取此方式，如今手游已经完全盖过了页游与端游的风采。手游包的体积也越来越大，而热更新就此在手游中出现。

1　原理

(1)首先将资源打包成AssetBundle。如果有两个对象共同依赖于同一个对象，需要采用依赖关系打包。AssetBundle需要根据不同的平台打包，各平台之间不能混用，如iOS和Android。

(2)为打包后的资源生成MD5值，上传服务器后，通过比对服务器端和客户端文件的MD5值，找出改变的文件，下载到本地。

(3)通过AssetBundle.CreateFromFile读取本地AssetBundle，因为该方法只能读取未压缩的AssetBundle，所以打包AssetBundle时，需要选择BuildAssetBundleOptions.UncompressedAssetBundle未压缩模式，然后使用LZMA或GZIP压缩后上传服务器。本地下载后需要解压缩保存在Application.persistentDataPath目录下。

2　资源打包AssetBundle

(1)新建一个cube，将其拉到Project视图里创建预设。

(2)在Assets目录下创建Scenes文件夹，创建场景并保存。

(3)新建ExportAssetBundles.cs，保存在Assets/Editor目录下。代码如下：

```
using UnityEditor;
using System.Collections;
using UnityEngine;
public class ExportAssetBundles : MonoBehaviour {
    [MenuItem("Build/ExportResource")]  //扩展菜单
    static void ExportResource()
    {
        //打开保存面板，获取用户选择的路径
        string path = EditorUtility.SaveFilePanel("Save Resource", "",
"New Resource", "assetbundle");
        if (path.Length != 0)
        {
            //选择的要保存的对象
            Object[] selection =
Selection.GetFiltered(typeof(Object), SelectionMode.DeepAssets);
            //打包
            BuildPipeline.BuildAssetBundle(Selection.activeObject,
selection, path, BuildAssetBundleOptions.CollectDependencies |
BuildAssetBundleOptions.CompleteAssets,
BuildTarget.StandaloneWindows);
        }
    }

    [MenuItem("Build/ExportScene")] //扩展菜单
    static void ExportScene()
    {
        //打开面板，选择用户保存的路径
```

```
            string path = EditorUtility.SaveFilePanel("Save Resource", "",
"New Resource", "unity3d");
            if (path.Length != 0)
            {
                //要打包的场景
                string[] scenes = {"Assets/Scenes/
AssetBoundleDemo.unity"} ;
                //打包
                BuildPipeline.BuildPlayer(scenes, path,
BuildTarget.StandaloneWindows,
BuildOptions.BuildAdditionalStreamedScenes);

            }
        }
}
```

(4)选中预设，运行ExportResource，弹出保存对话框，命名为cube.assetbundle。

(5)运行ExportScene，弹出保存对话框，命名为AssetBoundleDemo.unity3d。

小提示

1.AssetBundle的保存后缀名可以是assetbundle或者unity3d。

2.BuildAssetBundle要根据不同的平台单独打包，BuildTarget参数指定平台，如果不指定，默认的webplayer。

3 加载AssetBundle

下面通过一个示例演示如何加载AssetBundle。

```
using UnityEngine;
using System.Collections;
public class Load : MonoBehaviour {
    private string SceneUrl = "file:////Users/qcj-mac/Desktop/
AssetBoundleDemo.unity3d";
```

```
    private string BundleUrl= "file:////Users/qcj-mac/Desktop/
cube.assetbundle";
    void Start()
    {
        StartCoroutine(Download());
    }
    IEnumerator Download()
    {
        //下载AssetBundle，加载cube
        using(WWW www = new WWW(BundleUrl))
        {
            yield return www;
            AssetBundle bundle = www.assetBundle;
            Instantiate(bundle.LoadAsset("Cube"));
            bundle.Unload(false);
            yield return new WaitForSeconds(5);
        }
        using(WWW www = new WWW(SceneUrl))
        {
            yield return www;
            Application.LoadLevel("AssetBoundleDemo");
        }
    }
}
```

4 AssetBundle依赖关系

如果一个公共对象被多个对象依赖，我们打包的时候，可以有两种选取。一种是比较省事的，就是将这个公共对象打包到每个对象中。这样会有很多弊端：内存被浪费了；加入公共对象改变了，每个依赖对象都得重新打包。AssetBundle提供了依赖关系打包。

5 更新资源流程

(1)生成配置文件

在资源打包AssetBundle后，需要计算资源文件的MD5值，生成配置文件。下面给出一个例子：

```
using System;
using System.IO;
using System.Text;
namespace MD5
{
    class MainClass
    {
        public static string resPath = "/Users/qcj-mac/Documents/workspace/MD5/MD5/Res";
        public static void Main (string[] args)
        {
            Console.WriteLine ("Hello World!");

            //获取Res文件夹下所有文件的相对路径和MD5值

            string[] files = Directory.GetFiles(resPath, "*", SearchOption.AllDirectories);
            StringBuilder versions = new StringBuilder();
            for (int i = 0, len = files.Length; i < len; i++)
            {
                string filePath = files[i];
//                  string extension = filePath.Substring(files[i].LastIndexOf("."));
//                  if (extension == ".unity3d")
```

```
//                  {
                        string relativePath =
filePath.Replace(resPath, "").Replace("\\", "/");
                        string md5 = MD5File(filePath);

versions.Append(relativePath).Append(",").Append(md5).Append("\ n");
//                  }
            }
            //生成配置文件
            FileStream stream = new FileStream(resPath +
"version.txt", FileMode.Create);
            byte[] data =
Encoding.UTF8.GetBytes(versions.ToString());
            stream.Write(data, 0, data.Length);
            stream.Flush();
            stream.Close();
        }

        public static string MD5File(string file)
        {
            try
            {
                FileStream fs = new FileStream(file,
FileMode.Open);
                System.Security.Cryptography.MD5 md5 = new
System.Security.Cryptography.MD5CryptoServiceProvider();
                byte[] retVal = md5.ComputeHash(fs);
```

```
                fs.Close();
                StringBuilder sb = new StringBuilder();
                for (int i = 0; i < retVal.Length; i++)
                {
                    sb.Append(retVal[i].ToString("x2"));
                }
                return sb.ToString();
            }
            catch (Exception ex)
            {
                throw new Exception("md5file() fail, error:" +
ex.Message);
            }
        }
    }
}
```

(2)版本比较

先加载本地的version.txt，将结果缓存起来。下载服务器的version.txt，与本地的version进行比较，筛选出需要更新和新增的资源。

(3)下载资源

依次下载更新的资源，如果本地已经有旧资源，则替换之，否则就新建保存起来。

(4)更新本地版本配置文件version.txt

用服务器的version.txt替换掉本地的version.txt。这样做是为了确保下次启动的时候，不会再重复更新了。

关于上述的流程，下面有个小例子。这里没有用到web服务器，而是将本地的另外一个文件夹作为资源服务器目录。这里的目录只针对PC下的版本进行测试。如果要在手机平台上，需要记得更新相关的路径。

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.IO;

public class ResUpdate : MonoBehaviour
{
	public static readonly string VERSION_FILE = "version.txt";
	public static readonly string LOCAL_RES_URL = "file://" +
Application.dataPath + "/Res/";
	public static readonly string SERVER_RES_URL = "file:////Users/
qcj-mac/Documents/workspace/MD5/MD5/";
	public static readonly string LOCAL_RES_PATH =
Application.dataPath + "/Res/";

	private Dictionary<string,string> LocalResVersion;
	private Dictionary<string,string> ServerResVersion;
	private List<string> NeedDownFiles;
	private bool NeedUpdateLocalVersionFile = false;

	void Start()
	{
		//初始化
		LocalResVersion = new Dictionary<string,string>();
		ServerResVersion = new Dictionary<string,string>();
		NeedDownFiles = new List<string>();

		//加载本地version配置
```

```
            StartCoroutine(DownLoad(LOCAL_RES_URL +
VERSION_FILE, delegate(WWW localVersion)
                                {
                    //保存本地的version
                    ParseVersionFile(localVersion.text, LocalResVersion);
                    //加载服务端version配置
                    StartCoroutine(this.DownLoad(SERVER_RES_URL +
VERSION_FILE, delegate(WWW serverVersion)
                                        {
                        //保存服务端version
                        ParseVersionFile(serverVersion.text,
ServerResVersion);
                        //计算出需要重新加载的资源
                        CompareVersion();
                        //加载需要更新的资源
                        DownLoadRes();
                    } ));

            } ));
        }

        //依次加载需要更新的资源
        private void DownLoadRes()
        {
            if (NeedDownFiles.Count == 0)
            {
                UpdateLocalVersionFile();
                return;
            }

            string file = NeedDownFiles[0];
            NeedDownFiles.RemoveAt(0);
```

```
            StartCoroutine(this.DownLoad(SERVER_RES_URL + file,
delegate(WWW w)
                                {
                    //将下载的资源替换本地就的资源
                    ReplaceLocalRes(file, w.bytes);
                    DownLoadRes();
                } ));
        }

        private void ReplaceLocalRes(string fileName, byte[] data)
        {
            string filePath = LOCAL_RES_PATH + fileName;
            FileStream stream = new FileStream(LOCAL_RES_PATH +
fileName, FileMode.Create);
            stream.Write(data, 0, data.Length);
            stream.Flush();
            stream.Close();
        }

        //显示资源
        private IEnumerator Show()
        {
            Debug.Log(LOCAL_RES_URL + "cube.assetbundle");

            WWW asset = new WWW("file:////Users/qcj-mac/Documents/
workspace/MD5/MD5/Res/cube.assetbundle");
            yield return asset;
            AssetBundle bundle = asset.assetBundle;
            Instantiate(bundle.LoadAsset("Cube"));
            bundle.Unload(false);
        }

        //更新本地的version配置
```

```
    private void UpdateLocalVersionFile()
    {
        if (NeedUpdateLocalVersionFile)
        {
            StringBuilder versions = new StringBuilder();
            foreach (var item in ServerResVersion)
            {

versions.Append(item.Key).Append(",").Append(item.Value).Append("\
n");
            }

            FileStream stream = new
FileStream(LOCAL_RES_PATH + VERSION_FILE, FileMode.Create);
            byte[] data =
Encoding.UTF8.GetBytes(versions.ToString());
            stream.Write(data, 0, data.Length);
            stream.Flush();
            stream.Close();
        }
        //加载显示对象
        StartCoroutine(Show());
    }

    private void CompareVersion()
    {
        foreach (var version in ServerResVersion)
        {
            string fileName = version.Key;
            string serverMd5 = version.Value;
            //新增的资源
            if (!LocalResVersion.ContainsKey(fileName))
            {
```

```
                NeedDownFiles.Add(fileName);
            }
            else
            {
                //需要替换的资源
                string localMd5;
                LocalResVersion.TryGetValue(fileName, out
localMd5);
                if (!serverMd5.Equals(localMd5))
                {
                    NeedDownFiles.Add(fileName);
                }
            }
        }
        //本次有更新，同时更新本地的version.txt
        NeedUpdateLocalVersionFile = NeedDownFiles.Count > 0;
    }

    private void ParseVersionFile(string content,
Dictionary<string,string> dict)
    {
        if (content == null || content.Length == 0)
        {
            return;
        }
        string[] items = content.Split(new char[] { '\n' });
        foreach (string item in items)
        {
            string[] info = item.Split(new char[] { ',' });
            if (info != null && info.Length == 2)
            {
                dict.Add(info[0], info[1]);
            }
```

```
            }

        }

        private IEnumerator DownLoad(string url, HandleFinishDownload
finishFun)
        {
            WWW www = new WWW(url);
            yield return www;
            if (finishFun != null)
            {
                finishFun(www);
            }

            www.Dispose();
        }
        public delegate void HandleFinishDownload(WWW www);
}
```