关于异步加载 与 同步加载 效率的比较:

测试环境 Unity5.4 手机 安卓 华为meta8

测试用例说明:

统计 加载 16张 256\*256 RGBA32的图所需要加载的时间,资源都已经build好了 放在 StreamingAssets目录下

测试的时候为了避免文件缓存,每次测试都是重启手机后测试.

同步加载代码

public void LoadSync()

{

float start = Time.realtimeSinceStartup;

for (int i = 0; i < 16; i++)

{

string filename = Application.dataPath + "!assets/" + "a" + i.ToString() + ".png";

AssetBundle a = AssetBundle.LoadFromFile(filename);

Texture2D t = a.LoadAsset<Texture2D>(a.GetAllAssetNames()[0]);

}

float end = Time.realtimeSinceStartup;

SyncTime.text = "同步加载 Spend Time " + (end - start).ToString() + "s"

}

统计时间以成功得到Texture2D 为目标

在 meta8的测试加载时间约为1.8秒到2.2秒 之间

异步加载代码

public void LoadAsyn() //异步加载

{

StartCoroutine(LoadAsynWrap());

}

bool[] isloadover = new bool[16];

IEnumerator LoadAsynWrap()

{

float a = Time.realtimeSinceStartup;

for (int i = 0; i < isloadover.Length; i++)

{

isloadover[i] = false;

}

for(int i = 0; i < 16;i++)

{

string filename = Application.dataPath + "!assets/" + "a" + i.ToString() + ".png";

StartCoroutine(LoadOne(filename, i));

}

while (!Isalloadover())

{

yield return 1;

}

float b = Time.realtimeSinceStartup;

AsynTime.text = "异步加载 Spend Time " + (b - a).ToString() + "s";

}

bool Isalloadover()

{

for (int i = 0; i < isloadover.Length; i++)

{

if (!isloadover[i])

{

return false;

}

}

return true;

}

IEnumerator LoadOne(string filename,int index)

{

AssetBundleCreateRequest request = AssetBundle.LoadFromFileAsync(filename);

yield return request;

Texture2D t = request.assetBundle.LoadAsset<Texture2D>( request.assetBundle.GetAllAssetNames()[0]);

isloadover[index] = true;

}

在 meta8的测试加载时间约为0.5秒到0.6秒 之间

通过这个用例 我们可以发现,异步加载比同步加载的效率上有很大的提升.这里有几点需要说明的是

1. 在这个例子中,由于并没有其他的事情需要做,CPU处于相对闲置状态,因此启用多线程的时候,效率提升非常明显,但是在现实项目中,由于CPU 同时需要处理其他的事务,提升效果可能达不到这个比例.可以通过加载时限帧到更低,来提升加载速度.
2. 很多人对Coroutine 理解并不是特别正确,认为其本质是单线程机制,事实上UNTIY在内部很多处理已经是多线程机制.在本例中,

yield return request;

的作用是等待加载,具体的加载是在另外一个线程完成的. 而加载Texture2D的代码是在主线程上完成的.在本例中同时开辟了16个Coroutine ,相对于给后台线程同时推送了16个请求,并由后台去完成批量的加载.我没有UNITY的源代码,所以并不清楚具体后台是如何分配线程并完成的.但通过测试用例的结果,可以看出后台开了多线程的.

1. 这个测试用例通过同时启用批量的Coroutine而不是一个等一个的方式来加载资源,极大的提高了加载的效率.
2. 资源有限,同时开辟大量Coroutine 可能会导致抛出无法开始一个新的线程的异常,如果你需要使用这段代码,需要限制同时开辟Coroutine的数量,目前我在项目中使用的32个.