**游戏图片分享系统的研发**

吴培健

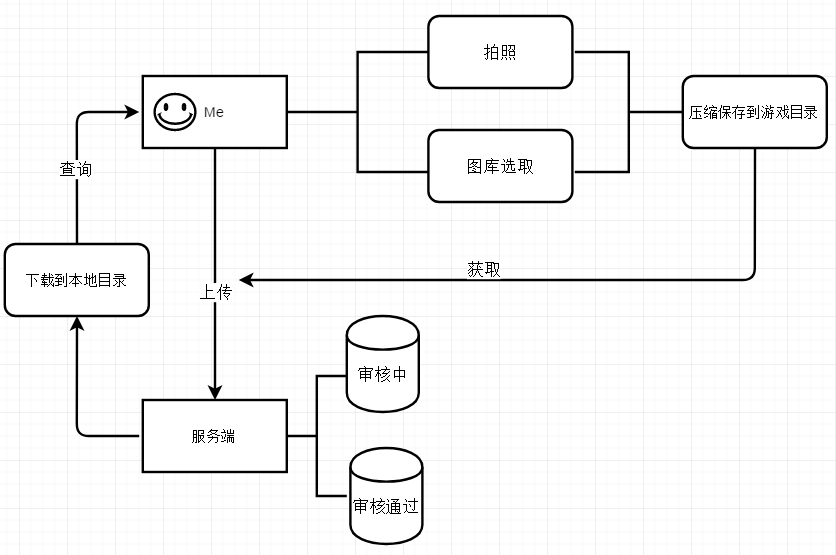
# 一、绪论

## 1. 引言

游戏中支持上传自定义图片是目前最普遍的做法，部分游戏只支持在内部提供有限数量的图片给玩家选择，作为玩家的头像，这种做法限制了玩家的个性化操作，玩家可选的头像比较单一，可能打开好友列表看到的是清一色的相同的头像。游戏支持上传自定义头像，可以很大程度上满足玩家展现自己的风采，审美的需求，对游戏中的社交活动也能起到一定积极作用。凡事都有两面性，由此会带来的问题也是需要考虑并且解决的，例如可能会有不怀好意的玩家会上传色情暴力的图片，如果处理不好会影响一个游戏的生命周期，所以图片审核流程必不可少，而且必须非常严格。

# 二、图片分享业务流程

## 1. 业务流程图



## 2. 说明

（1） 获取图片阶段：点击头像，弹出选择“拍照”，“图库中选择”，选择拍照，则调用系统摄像机，待拍照完毕，点击确定（一般是一个打钩的按钮），关闭照相界面，并跳转到图片缩放界面，缩放到合适大小，调整位置，点击确定。压缩图片大小（当前手机拍照处理的照片一般有几M到几十M不等，需要压缩至几百k），保存到手机目录。

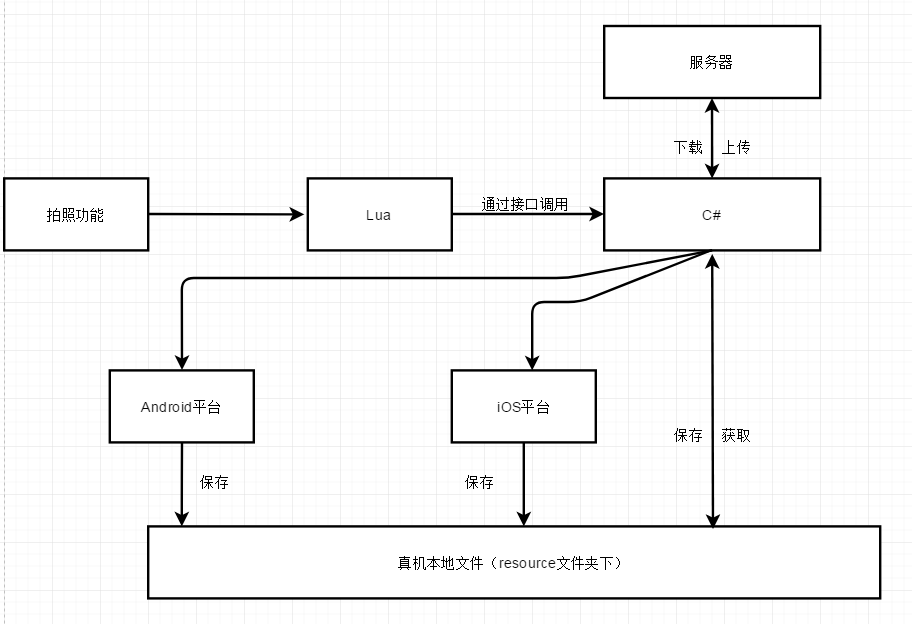
（2）图片上传阶段：点击上传按钮，获得步骤（1）中的图片的字节数组，调用服务器提供get/post请求链接将图片以字节数组的形式上传数据。

（3）服务端审核：服务端收到图片数据，将图片放入审核中的数据库，等待人工审核。审核通过后，审核中数据库的图片会被复制到审核通过的数据库。每个玩家在审核中，审核通过的数据库中分别有且只有一张图片存在。

（4）图片下载阶段：通过调用XPhoto插件的接口加载图片，分两种情况：查看玩家自己的头像，首先向服务端请求审核中的图片，存在则将审核的图存放在游戏目录，若不存在，向服务端请求审核通过的图片，存在则将图片保存到本地，若不存在，根据玩家职业显示对应的默认图片。查看其他玩家的头像，向服务器请求审核通过的图片，存在则将图片保存到本地，若不存在，根据玩家职业显示对应的默认图片。

# 三、图片分享框架

## 1. 框架图



## 2. 说明

（1） C#中间层：C#层作为平台层和Lua层交互的桥梁。

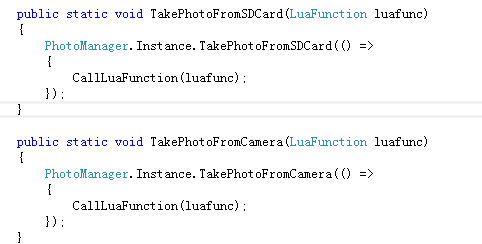
首先是C#和平台接口交互，C#抽象出接口用来拍照，从图库中选取图片，在基类PlatformSdkMgr中定义：

public virtual void TakePhotoFromSDCard(string path, string filename, string cropSize = "200"){}

public virtual void TakePhotoFromCamera(string path, string filename, string cropSize = "200"){}

C#和具体平台交互实现细节放在相应子类中实现，Android平台交互类AndroidSdkMgr继承自PlatformSdkMgr，实现了从C#和Android互相发送消息，具体交互细节这里不展开讨论。从C#发送消息告诉Android端需要拍照或者从图库中选取图片，接着相关的平台调用自己的原生接口进行拍照，选图。同理IOSSdkMgr也继承自PlatformSdkMgr，实现从C#到iOS的交互，同样将平台相关的操作交给iOS系统处理。各个平台将玩家选中的图片都放在相同的游戏内部目录，C#就可以无视平台的差异等同的对玩家选好的图片进行上传下载处理。

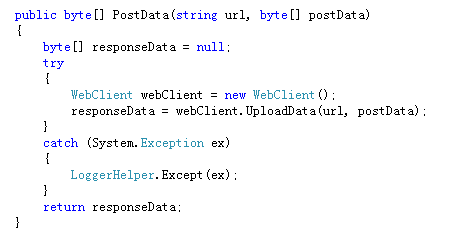
Lua层和C#层交互还是利用tolua框架，在LuaFacade中定义Lua使用的接口：



C#上传下载选中的图片：这里是纯C#操作，也就是简单的在C#中发送get/post请求，图片利用io操作转换成字节数组的形式进行传送。具体实现封装在PhotoManager中，代码过长这里只贴关键代码：

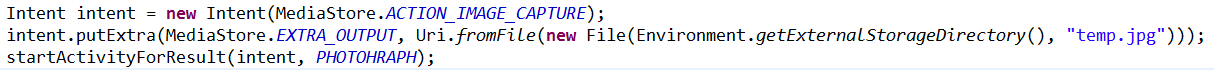
public void UploadPhoto(string cross\_uuid, EndRemoteUrlCallback callback){…}

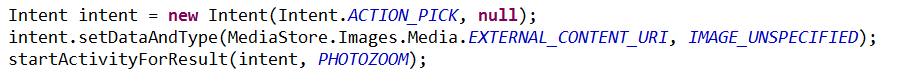
public int LoadPhoto(string name, bool isPlayer, string version, GameObject go, Action<Texture2D> cb){}



（2） 平台层（Android，iOS）：仅以Android为例做说明

C#端发送拍照，从图库获取图片的请求到达Android端，Android端立即发送Intent打开拍照或者图库的Activity：





选择好需要的图片返回MainActivity，MainActivity收到返回结果继续发送Intent打开图片缩放的Activity，完成图片缩放：



缩放完成，需要对图片进行优化操作，压缩（图片过大不利于传输）：

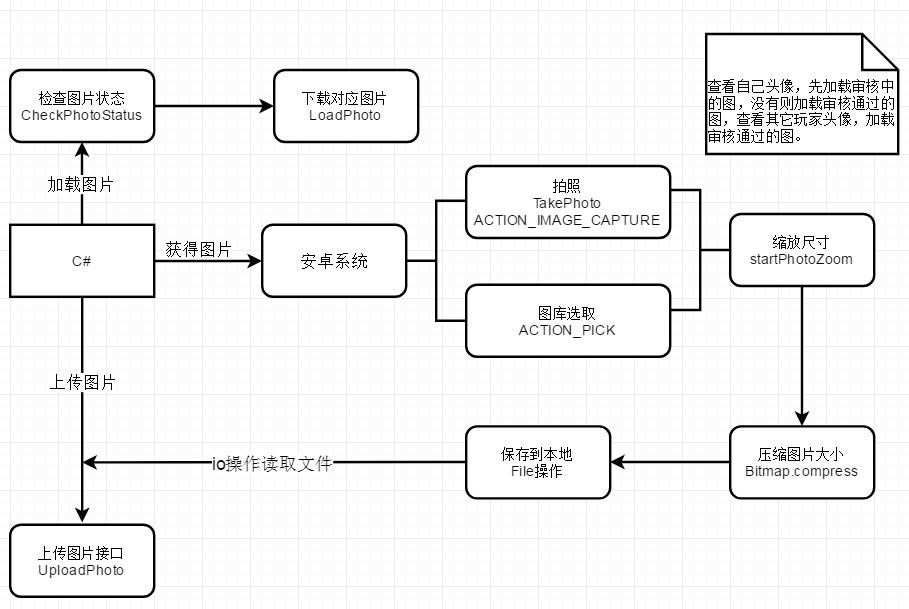
C:\Users\ADMINI~1\AppData\Local\Temp\1553841478(1).png

需要注意的是压缩过程不能在主线程中执行，否则可能会引起ANR。压缩完成使用File将图片保存在本地。

（3）Lua层：前面两步如果布置好了，到了这一步则是相当简单了，调用（1）在LuaFacade中暴露的接口就能轻松调用到Android/iOS接口，实现拍照，上传，加载头像的操作。

可以根据策划的具体需求适当的在（1）（2）中做出调整。

## 3. Unity-Android图片分享流程图



# 四、小结

游戏图片分享系统的功能比较单一，实现起来比较简单。比较复杂的功能是下载，可以用MD5的形式计算是否需要更新图片，而不需要每次查看都从服务器下载一次图片。最好是MD5信息能作为游戏玩家的信息（类似名字），这样可以少发一次检查MD5的请求。下载的图片都缓存在本地Photo目录下。