



# 燕良@游戏开发

--做游戏是一件快乐的事



目录视图

摘要视图

RSS 订阅

## 个人资料



房燕良

+ 加关注 发私信



访问： 125179次

积分： 1743

等级： **BLOG > 4**

排名： 第12304名

原创： 27篇 转载： 0篇

译文： 1篇 评论： 160条

## 引擎技术交流QQ群



游戏引擎能吃吗

(264656505)

## 文章搜索

## 常用网址

Unity3D官网  
Unreal Engine官网  
游戏蛮牛

## 文章分类

Unity3D (19)  
Unreal Engine (2)  
自研引擎 (2)  
其它 (5)

CSDN专家精选，微信开发学习路线大有看头！ 【博乐】点评美文，得C币 iOS开发前沿与Swift探秘 Swift初级教程大汇总

## 《炉石传说》架构设计赏析(4)：Asset管理

分类： Unity3D

2014-09-29 08:15 3190人阅读 评论(0) 收藏 举报

unity3d

炉石传说

目录(?)

[+]

欢迎转载，请注明作者【燕良@游戏开发】及原文地址：<http://blog.csdn.net/neil3d/article/details/39580197>

另外，欢迎大家来我的QQ群交流各种游戏引擎相关的技术：游戏引擎能吃吗(264656505)

话说，经过这段时间的学习和摸索，对于Unity3D的开发思路已经基本清晰了。唯独还剩下一个AssetBundle机制还没有搞透，这个涉及到前期项目的资源规划、资源管理代码的写法，以及自动更新机制的实现。

所以，还是想先把游戏逻辑的进一步分析押后，先来看一下《炉石传说》Asset管理。必须得说一下的是，目前分析都是PC版的程序集，对于移动端不一定完全合适，且当做一个案例分析吧。

本文主要讲述《炉石传说》的AssetBundle的管理机制。它的机制比较简单清晰，中规中矩，中间的分析过程就不讲了，直接展现其架构设计和代码逻辑组织。先从Asset管理相关的类讲起。

### class Asset：资源信息描述

Asset
-paths
+Directory : string
+Extensions : string
+Family : AssetFamily
+LoadAsync : bool
+Name : string
+Persistent : bool
+PreloadOnly : bool

AssetFamilyPathInfo
+exts
+format
+sourceDir

Asset类，并不管理直接的资源对象，而是保存的一个Asset相关的信息，具体请看上图。

另外，它还有一个“paths”变量，这是一个Dictionary，key是AssetFamily枚举，value是Assetbundle的路径和资源路径。下面的AssetFamily一节详细解释。

### enum AssetFamily - 资源分类



文章存档

2015年08月 (1)

2015年07月 (3)

2015年04月 (1)

2015年03月 (1)

2015年02月 (1)

展开

阅读排行

《炉石传说》架构设计赏

你不再需要TinyXML，推

使用Unity3D的50个技巧

分享一个小工具：Excel

2009年混合语言编程总

《炉石传说》架构设计赏

Unity3D-RPG项目实战 (

Unity 4.6的使用匿名dele

《炉石传说》架构设计赏

Unity3D-RPG项目实战 (

最新评论

次世代关卡制作流程：使用Unre

分享一个小工具：Excel表快速转

《炉石传说》架构设计赏析(2)：！

分享一个小工具：Excel表快速转

分享一个小工具：Excel表快速转

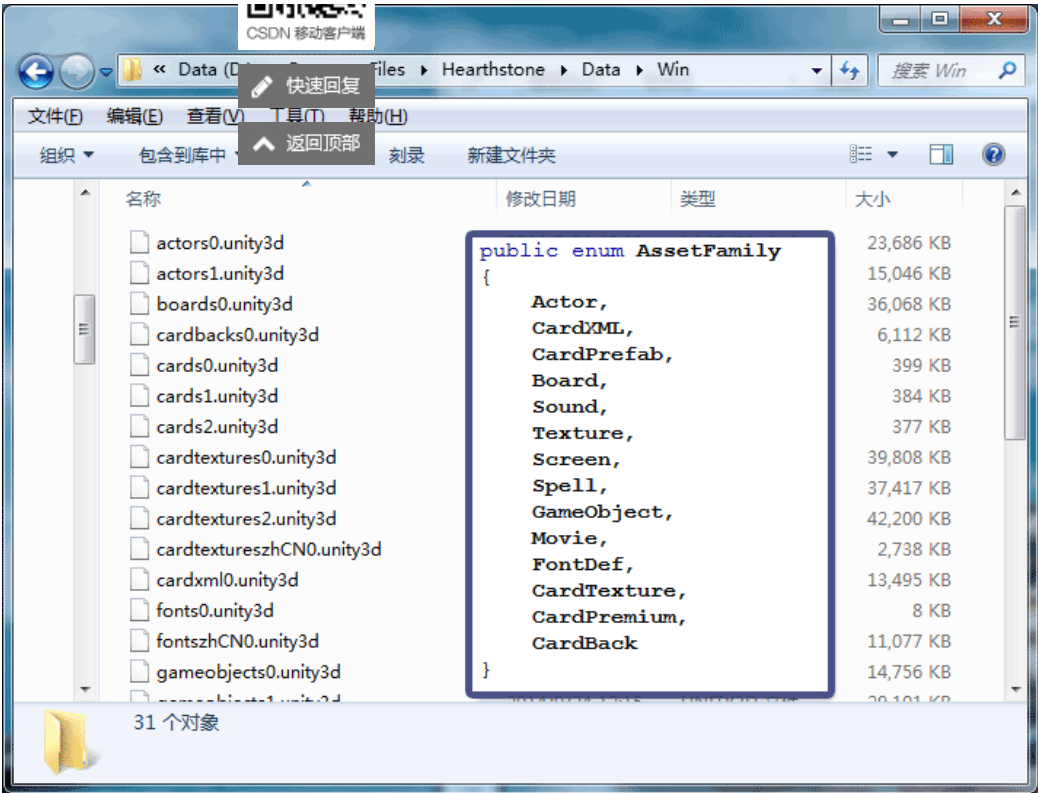
分享一个小工具：Excel表快速转

次世代关卡制作流程：使用Unre

次世代关卡制作流程：使用Unre

次世代关卡制作流程：使用Unre

次世代关卡制作流程：使用Unre



如上图所示：

- 炉石根据资源的不同类型进行分别的AssetBundle打包，一类资源对应一个或者多个资源包；（一类资源分多个包的规则不得而知）；
- 有的资源包真的本地化单独打包，例如“fonts0.unity3d”==》“fontszhCN0.unity3d”；
- 在程序中，资源包的分类对应枚举类型“AssetFamily”；
- 资源包的具体路径信息，存储在Asset.paths，这是一个静态变量；在初始化时，手动填写必要的信息，类似这样：

```
[csharp]
01. Dictionary<AssetFamily, AssetFamilyPathInfo> dictionary = new Dictionary<AssetFamily, As
();
02. AssetFamilyPathInfo info = new AssetFamilyPathInfo {
03.     format = "Data/Actors/{0}.unity3d",
04.     sourceDir = "Assets/Game/Actors"
05. };
06. info.exts = new string[] { "prefab" };
07. dictionary.Add(AssetFamily.Actor, info);
```

- 另外，还有一个class AssetBundleInfo是记录了每种AssetBundle对应的主文件名，以及包文件的个数、对应的对象类型等信息；详见下图：



class AssetLoader：资源加载



AssetLoader
<div><div>-m_downloadedFamilyBundles : AssetBundle</div><div>-m_downloadedLocalizedFamilyBundles : AssetBundle</div><div>-m_familyBundles : AssetBundle</div><div>-m_localizedFamilyBundles : AssetBundle</div><div>-m_sharedBundle : AssetBundle</div></div> <div><div>+LoadActor() : bool</div><div>+LoadBoard() : bool</div><div>+LoadCardBack()</div><div>+LoadCardPrefab() : bool</div><div>+LoadCardXml() : object</div><div>+LoadFile() : bool</div><div>+LoadFontDef() : bool</div><div>+LoadGameObject() : bool</div><div>+LoadGameObjectImmediately() : GameObject</div><div>+LoadMovie() : bool</div><div>+LoadSound() : bool</div><div>+LoadSpell() : bool</div><div>+LoadSpellTable() : bool</div><div>+LoadTexture() : bool</div><div>+LoadUIScreen() : bool</div><div>+PreloadActor()</div><div>+PreloadActorSpell()</div><div>+PreloadBundles()</div><div>+PreloadGameObject()</div><div>+PreloadSpell()</div><div>+ReloadUpdatableBundles()</div><div>+UnloadUpdatableBundles()</div><div>+GetBundleForFamily() : AssetBundle</div></div>

游戏运行时需要加载各种资源，基本上都是通过AssetLoader（也有个别情况适用了Resources.Load()）。接下来我们就重点看一下AssetLoader的实现思路。

AssetLoader对上层提供资源对象加载接口，对于每种类型的资源都提供一组函数，例如LoadCardPrefab，LoadActor等等。对于对象加载完成、加载进度等提供回调函数。这些函数只是一些简单的包装，其内部都调用到LoadCachedGameObject()或LoadCachedObject()这两个核心函数。

从这两个函数的流程可以看到，资源加载使用到了Cache机制：

- 首先从AssetCache中查找，如果找到了，则更新Cache项的时间戳，并调用回调；
- 如果没有找到，则向AssetCache添加一个Request，然后启动Coroutine：CreateCachedAsset()，它的调用步骤是：
  - 调用AssetCache.StartLoading();
  - 启动Coroutine：CreateCachedAsset\_FromBundle<RequestType>():
    - 使用AssetLoader.GetBundleForAsset()找到资源所属的AssetBundle；
    - 调用AssetBundle.LoadAsync()来真正加载资源；
    - 在加载的过程中，根据处理的结果调用：AssetCache.CacheRequest的OnLoadFailed()、OnLoadSucceeded()、OnProgressUpdate()等函数；
  - 在AssetCache查找此资源，如果找到了，则加载成功，调用回调函数；调用AssetCache.StopLoading();

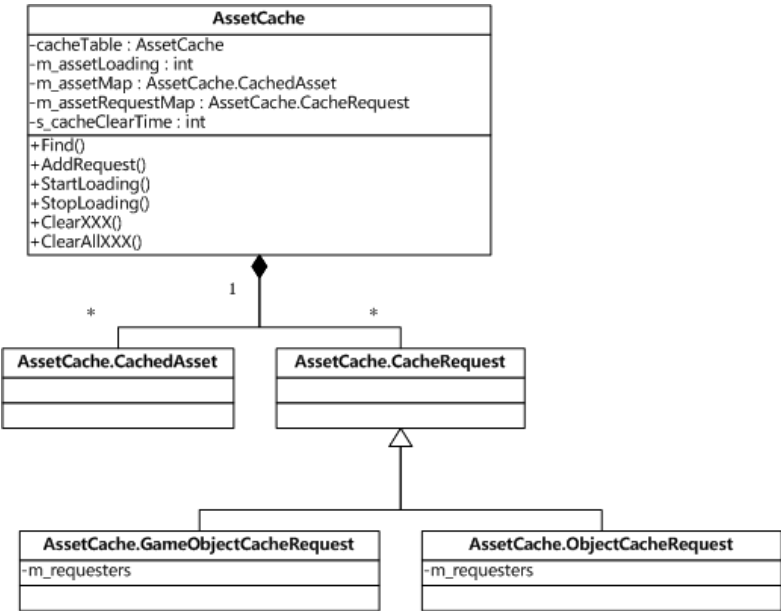
我们都知道在开发过程中，不能使用AssetBundle（每次启动都要打包，肯定收不了）。怀疑它的Editor模式相关的代码是用预编译宏处理来实现的，所以未出现在发布出来的程序集当中，类似这样：

```
[csharp]
01.  #if UNITY_EDITOR
02.      Obj = Resources.LoadAssetAtPath(assetPath, typeof(T));
03.      if (Obj == null)
04.          Debug.LogError ("Asset not found at path: " + assetPath);
05.          yield break;
06.  #else
```

class AssetCache：资源的Cache机制

前面在AssetLoader一节我们已经讲到了AssetCache机制，这里再做一个详细的阐述。





前面我们已经讲到:

- AssetCache中的资源项的时间戳, 由AssetLoader在资源加载请求时维护;
- AssetCache主要负责管理Cache数据, 而真正的资源加载动作还是在AssetLoader中执行;

AssetCache的资源淘汰主要由外部的各个模块根据自己认为需要的时机去调用, 例如:

- SceneManager.ClearCachesAndFreeMemory()
- LoadingScreen.ClearAssets()
- SoundMgr.UnloadSoundBundle()
- 等等

另外, 程序启动时会自动更新资源包 (在Login.OnAssetsVersion()中启动), 主要是通过UpdateManager和Downloader两个类来处理。

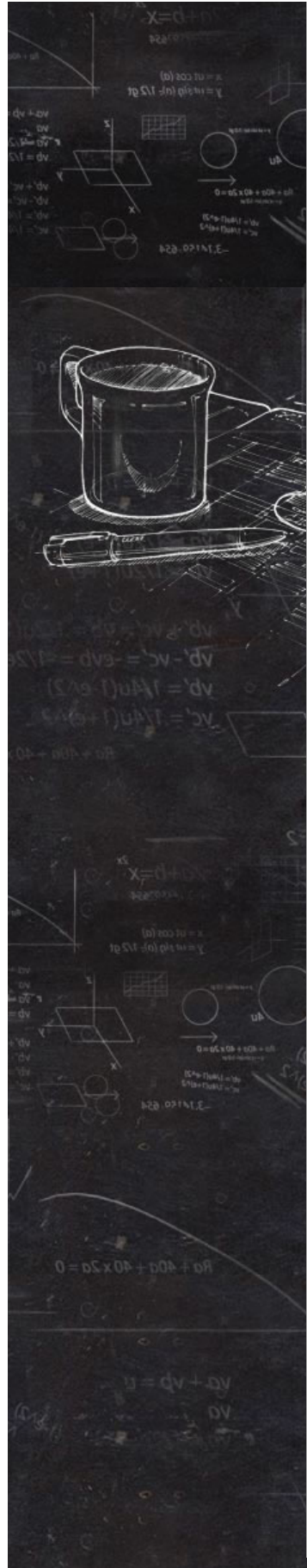
OK, 总结一下炉石的资源管理机制:

- 对游戏资源按照类型分包, 每一类资源包可以有多个;
- 在游戏运行时使用Cache机制;

最后, 还是顺便炫一下战绩:



版权声明: 本文为博主原创文章, 未经博主允许不得转载。



^ 上一篇

推荐一款Visual Studio的LUA插件

v 下一篇

《炉石传说》架构设计赏析(5)：卡牌&技能的静态数据组织

顶

1

踩

0

主题推荐

unity3d

架构设计

游戏引擎

游戏开发

dictionary

本地化

猜你在找

- Cocos2d-Lua手游开发基础篇

■ 《炉石传说》架构设计赏析5卡牌&技能的静态数据
- Qt基础与Qt on Android入门

■ 《炉石传说》架构设计赏析7使用ProtocolBuffers
- 数据结构和算法

■ 《炉石传说》架构设计赏析6卡牌&技能数据的运行
- 3G Android实战开发从入门到精通

■ 《炉石传说》架构设计赏析6卡牌&技能数据的运行
- Python编程基础视频教程(第五季)

■ 《炉石传说》架构设计赏析6卡牌&技能数据的运行

准备好了么？跳吧！

更多职位尽在 CSDN JOB

■ 产品经理（后台管理）	我要跳槽	■ 开发管理工程师（互联网）	我要跳槽
钰诚国际控股集团	20-35K/月	重庆东银控股集团有限公司	6-10K/月
■ 智能客服系统开发（管理）岗	我要跳槽	■ 系统管理工程师	我要跳槽
中信建投证券股份有限公司	15-25K/月	招商银行信用卡中心	25-35K/月

New Message

To gizmacam@gmail.com |

查看评论

暂无评论

您还没有登录,请[登录](#)或[注册](#)

\* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题

Hadoop

AWS

移动游戏

Java

Android

iOS

Swift

智能硬件

Docker

OpenStack

VPN

Spark

ERP

IE10

Eclipse

CRM

JavaScript

数据库

Ubuntu

NFC

WAP

jQuery

BI

HTML5

Spring

Apache

.NET

API

HTML

SDK

IIS

Fedora

XML

LBS

Unity

Splashtop

UML

components

Windows Mobile

Rails

QEMU

KDE

Cassandra

CloudStack

FTC

coremail

OPhone

CouchBase

云计算

iOS6

Rackspace

Web App

SpringSide

Maemo

Compuware

大数据

aptch

Perl

Tornado

Ruby

Hibernate

ThinkPHP

HBase

Pure

Solr

Angular

Cloud Foundry

Redis

Scala

Django

Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持

