# Nexus Engine中的资源IO

## nresource\_manager组合各种IO策略

Nexus Engine第一个项目为MMORPG类型，但是我们不制作“无缝世界”的特性，而是采用“切换地图”的方式。但是游戏本身还是需要异步IO的，例如在游戏的过程中，由于NPC、其他玩家进入视野，需要加载他们的模型、贴图，这些操作都不应该影响主游戏线程，否则就会产生卡、顿的现象。

在nresource\_manager类中我们聚合了一个**nresource\_io**对象，然后我们实现了两个派生类：

1. nresource\_io\_basic，提供简单直接的阻塞资源IO模式，这种模式对于编辑器特别有用，可以大大简化编辑器的逻辑；
2. nresource\_async\_io，提供异步资源IO，同时也提供阻塞IO功能的策略，主要用于游戏运行时；



## 异步IO机制

### 问题分析

不了解的人会说“异步IO有什么难的，不就是搞一个线程在后面读磁盘嘛”！这话不能说他错，但是却不是问题的关键。异步IO机制的设计，首先重点是哪些对象需要执行异步IO，其次是磁盘数据读进来之后如何创建程序实际需要的对象，然后是如何使得上层逻辑避免多线程处理以及尽量减少异步处理。

首先我们来看个失败的例子，就是Unreal Engine 3。UE3的所谓的异步IO的支持是建立在Level Streamming机制之上的，它有一个单独的线程负责把数据块从磁盘的package中读取。由于Unreal的对象机制的限制，所有的对象创建必须在主线程中执行，这大大的降低了并行性；其次，由于渲染系统运行在另外一个单独的线程，所以渲染资源的创建又被派发到渲染线程中去执行，这极大的增加了程序复杂度。

其次分析一下我们上一代引擎Fantastic 3D的情况。我们的场景（或称之为关卡）是存盘游戏逻辑对象数据的，而且这部分数据并不大，一个超大型场景的话，这部分数据也就在2M左右，所以这部分数据我们是在游戏启动的时候加载的；而大量的数据是3D模型，贴图等资源，所以这部分数据支持异步IO的模式。整体的思路较之UE3更清晰，资源的IO对上层游戏逻辑的影响也很小。

综上所述，我们将延续我们之前的思路，也就是说尽量把异步IO的复杂度封装在资源系统内部，尽量减少向游戏逻辑层的扩散。

### 整体思路描述

首先在游戏启动的时候，可以通过engine\_config来配置使用哪个资源io类。如果异步io启动，则在引擎初始化时，启动一个线程开始等待“资源io请求”。

对于异步IO模式（nresource\_async\_io），在外部调用“nresource\_manager::load\_xxxxx”时，除非指定使用阻塞模式，否则这个操作将作为一个“资源io请求”被post到一个队列中，资源io线程被激活。资源io线程处理完该队列中所有请求后挂起，直至新的请求加入。

在资源io线程中加载资源的方式，是调用“nresource::load\_from\_file”，这和简单阻塞模式下是完全一样的，所以**需要nresource及其派生类是线程安全的**！资源加载主要是两部分操作，一部分是使用文件系统读磁盘，另外一部分是创建内部数据对象以及render resource，所以也需要涉及到的**相关系统（主要是File System和Render Resource）是线程安全的**！

### 资源IO完成通知

在资源加载完成（对象内部数据完全可用，并且所有子资源也加载完成）的时候，统一的post一个event到资源管理器。在每一帧update的时候，资源管理器负责向当前注册了资源事件的所有对象派发所有事件。这个设计达到了几个目的：

1. 异步IO与简单阻塞IO模式的统一；  
   既然引入了异步IO机制，那么上层完全避免异步处理也是不可能的，我们尽量的把它封装到actor component层；对于actor component，在绑定资源时和接到资源完成通知时，创建内部动态数据。另外阻塞模式下，加载完资源，也同样抛出事件，这使得以上机制得以执行。
2. 线程安全性不向上层扩张；

事件队列本身管理了线程安全，完成事件可以在IO线程或者主线程中post进来，但是全部都在主线程中派发，所以actor component所有代码仍然只执行在主线程中，不需要线程安全。

### 编辑器情景

首先编辑器只支持简单阻塞IO模式即可，这样能简单、可靠很多。

其次，因为actor component等需要绑定资源的对象都采用了相应“资源IO完成通知”的方式，所以在nresource类中增加了一个接口：“nresource::post\_edit\_change(bool ready)”，在编辑资源的过程中可以调用此函数，将nresource的ready标志设为true，并发送完成通知。

### 使用限制

* 如果一个资源自身使用了阻塞IO，而它的子资源使用了异步IO，则此时主资源的ready()测试返回false，并且上层无法收到资源完成通知（会收到子资源完成通知，但无法判断资源依赖关系）。
* nmaterial::set\_texture\_parameter()函数固定采用了EIO\_Block模式；

### 可能的改进

类似bounding box、动画长度等游戏逻辑相关的信息，可以管理在一个单独的其他文件中（当然这是冗余的），可以用完全阻塞的方式读入，以避免上层处理异步逻辑。可能存盘这些信息的地方主要有两个选择，一个是由底层资源系统统一处理，另外一个是存盘上层用到这些数据的地方，例如NPC模板表，物品模板表等。