OGRE 分析之文件系统（三）

Mythma
http://www.cppblog.com/mythma
Email: mythma@163.com

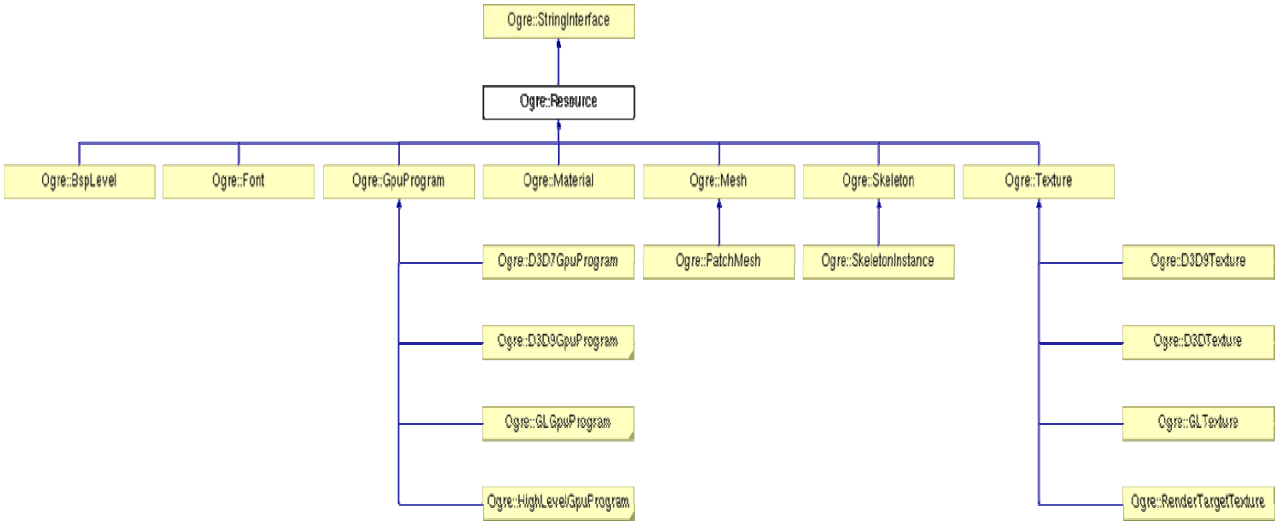
三、OGRE 的资源管理

资源的管理在游戏中占有十分重要的地位，因而 OGRE 对其提供了强大的支持。

1、资源

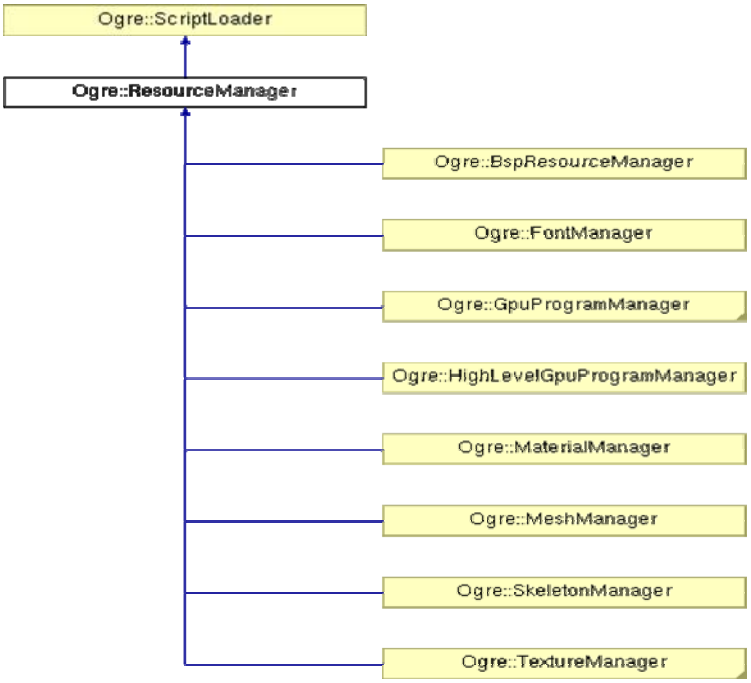
资源是程序自始至终加载和管理的数据对象。资源必须有一个唯一的名字标识，必须只被加载一次，并且能够被卸载以释放内存。

哪些属于资源？从 Ogre::Resource 的继承体系可以看出：



2、资源管理

对于每一种 Resource，OGRE 都提供了相应的 ResourceManager，因此如下图所示：



Ogre::Resource 定义了各种资源的读取方式，Resource 可以通过类 Ogre::Resource 提供的方法加载和卸载，但 Resource 的最终的销毁却由 Ogre::ResourceManager 来管理。在构造 Ogre::Resource 的时候，必须得提供一个 ResourceManager 的引用。

具体的 ResourceManager 负责管理对应的资源池，并可以索引资源、查找资源、加载销毁资源以及内存的负荷监测等。

Ogre::ResourceManager 维护了一个 Resource 的 hash 表用以资源的快速查找，并提供了一个迭代器来遍历资源：

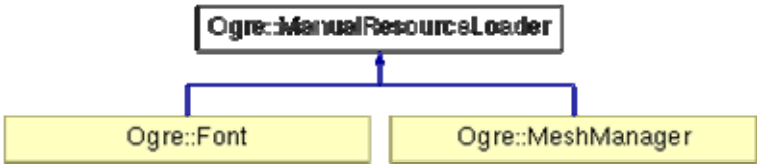
```
typedef HashMap< String, ResourcePtr >  ResourceMap
typedef std::map< ResourceHandle,ResourcePtr >  ResourceHandleMap
typedef MapIterator< ResourceHandleMap >  ResourceMapIterator
```

ResourceHandle 是一个无符号的长整型，初始为 1，每当加入一个资源的时候便加 1，用于作为 Resource 的 Handle。

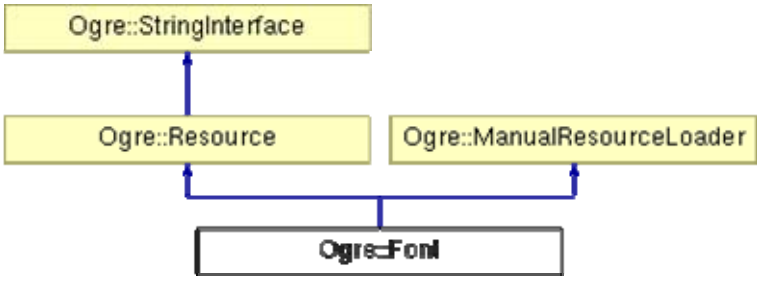
3、资源加载

Ogre::Resource 和 Ogre::ResourceManager 之间的关系从类的继承层次上可以看成是**抽象工厂模式**。在两者的实现代码中，广泛用到**模板方法模式**，很多纯虚函数都是由子类实现的。Resource 利用 ResourceManager 来通知资源的加载情况，ResourceManager 利用 Resource 加载具体的资源。

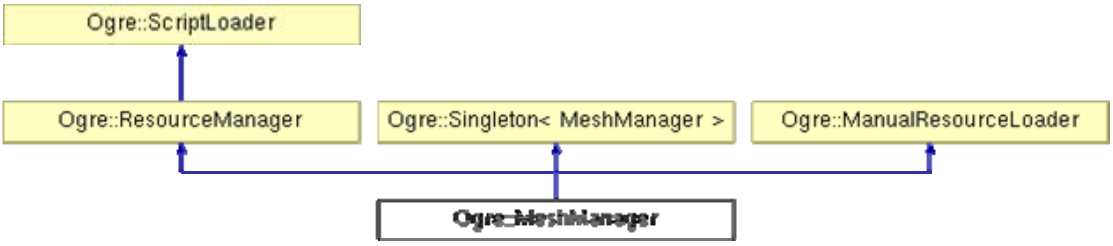
同一种 Resource 的加载方式可能有不同的加载方式，因此 Resource 不负责加载资源。OGRE 使用**桥接模式**来解决这种情况。对于这类资源，需要指定其加载方式为手动加载（Manual），并提供一个资源加载器 Ogre::ManualResourceLoader：



从另一个角度看 Ogre::Font：



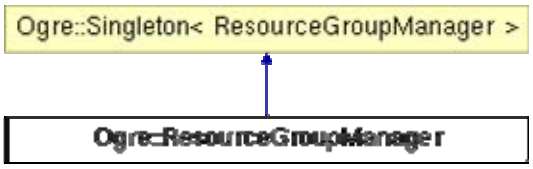
再看 Ogre::MeshManager：



可见 Font 集 Resource 和 ResourceLoader 于一身，而 MeshManager 则集 Resource 和 ResourceLoader 于一身。这样的结构看起来有一点混乱。

4、资源组的管理

从上面的分析可以发现，各种 Resource 和各种 ResourceManager 组和起来，使用起来确实复杂。不过还好，OGRE 提供了另一个管理类：Ogre::ResourceGroupManager。ResourceGroupManager 把资源分成一个个的组，通过调用资源对应的 ResourceManager 来加载或卸载组中的资源。



Ogre::ResourceGroupManager 使用了单件模式，这就意味着全局的资源都存放在一处，其初始化之地在 Root 的构造函数之中。比较前面提到的 Archive 部分，可以发现，Ogre::ArchiveManager 与 Ogre::ResourceGroupManager 的功能（形式上的）十分相似。

1) Ogre::ResourceGroupManager 内部的主要数据结构：

```
typedef std::map< String, ResourceManager * > ResourceManagerMap
typedef std::multimap< Real, ScriptLoader * > ScriptLoaderOrderMap
typedef std::vector< ResourceGroupListener * > ResourceGroupListenerList
typedef std::map< String, Archive * > ResourceLocationIndex
typedef std::list< ResourceLocation * > LocationList
typedef std::list< ResourcePtr > LoadUnloadResourceList
typedef std::map< String, ResourceGroup * > ResourceGroupMap
```

其中：

a）ResourceGroup：主要有如下信息（具体参照 OGRE 源码）：该组的名字、该组中目录表、该组中种的资源表

b）ResourceLocation：资源的位置信息。由前面分析知道 Archive 用于目录管理，所以实际上就是一个 Archive。

c）ResourceGroupListener：用于返回资源加载情况的信息，如加载进度信息。

2) 资源的状态

资源在 ResourceGroupManager 中可有如下状态：

a) Undefined：在此状态 Resource 没有被实例化。此时 Resource 的路径已经加入资源路径，但只在文件系统中，Ogre 对 Resource 不做任何处理，即 OGRE 不会加载在此状态的资源。

进入此状态的条件：当调用 ResourceGroupManager 的 addResourceLocation 方法时；当资源本是有效的实例，但是调用了 ResourceManager::remove
或 ResourceGroupManager::clearResourceGroup 后。

b) Declared：此时 Resource 仍没有被实例化，只是加入资源组的已声明列表。当初生化资源组的时候，在此状态的资源，OGRE 将会创建该资源。

当调用 ResourceGroupManager::declareResource 后进入此状态。

c) Unload：此时 Resource 仍没有配实例化。但查找该资源可以找到，但资源没有使用很多内存，没有被加载。

当调用 ResourceGroupManager::initialiseResourceGroup 或是调用了 ResourceManager::create 后进入此状态；调用 Resource::unload 和 ResourceGroupManager::unloadResourceGroup 后。

d) Loaded：资源完全被加载。

调用 ResourceGroupManager::loadResourceGroup 后进入加载状态。

3）资源的加载顺序：

ResourceGroupManager::addResourceLocation
ResourceGroupManager::initialiseResourceGroup
ResourceGroupManager::loadResourceGroup

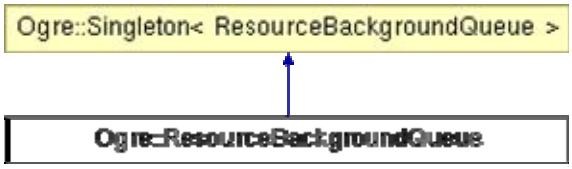
4）监控资源的加载过程

OGRE 提供了一个抽象类 Ogre::ResourceGroupListener 用于监控资源的加载进度。ResourceGroupManager 中维护了一个 ResourceGroupListener 的列表。在加载资源组的过程中，在每个资源加载之前，ResourceGroupManager 会向各 Listeners 发送 resourceLoadStarted 消息，在资源加载完成之后会发送 resourceLoadEnd 消息。

关于加载进度条，OGRE 的例子 deom_BSP 有演示。

5、多线程下的资源管理

多线程下，OGRE 的提供一个资源后台队列：Ogre::ResourceBackgroundQueue，用于资源的加载和卸载。



另外 OGRE 还提供了一个 Listener 监听后台资源的加载情况：Ogre::ResourceBackgroundQueueListener。