# 客户端内存泄漏

无尘

##### AFile泄漏

* 1. 泄漏原因：
     + AFile是用来存储模型动画数据的类，占用的内存空间较多。
     + 诺亚所有的界面都是单例，一旦被创建就不会释放内存
     + 有些界面需要显示模型，模型需要加载AFile，如果模型不释放，AFile也会一直存在于内存中
     + 诺亚显示模型的界面有20多个，如果都不释放会造成大量的内存泄露
  2. 解决方法：
     + 为所有需要显示模型的界面写了基类，在基类中定义了初始化模型和释放模型的虚函数
     + 在界面显示时调用模型初始化函数，在界面析构时调用模型释放函数
     + 整理所有的模型显示界面，实现相应的接口

##### 性能优化导致的泄漏

* 1. 泄漏原因：
     + 每个AFile中存储了50 – 80个骨骼，每个骨骼对应了一个CFBGraphics\_MovableKey类型的对象

class CFBGraphics\_MovableKey

{

int m\_nTransforms; // 帧数

CFBGraphics\_Key\_Matrix\* m\_pTransform; // 骨骼变换矩阵数组

int\* m\_pFrameIndices;

}

class CFBGraphics\_Key\_Matrix

{

float fFrame;

D3DXMATRIX Matrix;

}

然而帧数是整数，因此fFrame每一帧都要转换成整数再使用

* + - C++的float转int的效率很低，所以我在CFBGraphics\_MovableKey加载的时候new了一个int型数组，预先把所有的fFrame从float转成int型，避免每一帧都做转换。
    - 然后这个int型数组没有在AFile析构的时候delete掉，造成了比较多的内存碎片
  1. 建议：
     + 非必要情况不要使用new创建临时变量，除了内存泄漏的隐患外，new出来的变量存在堆区，访问速度也比较慢。
     + 尽量在new对象的时候马上写好对应的delete语句

##### 资源文件查询失败导致文件异步加载节点泄漏

* 1. 泄漏原因：
     + 诺亚的资源文件是异步加载的，主线程需要加载文件时，会先创建一个AsynLoading\_Node对象，里面保存了文件路径，再把它交给子线程
     + 子线程被唤醒以后，会通过AsynLoading\_Node中存储的文件名加载数据
     + 资源文件中保存了AsynLoading\_Node对象，这个节点会随资源文件的释放一起被释放
     + 主线程有可能在子线程加载数据的时候释放资源文件，如果此时删除AsynLoading\_Node会导致宕机，为此诺亚里面加入了一个保护机制

主线程

子线程

Load File

Create Node

m\_bOK = false

m\_bDiscarded = false

if m\_bDiscarded

Delete Node

Query File

Read File

if bQuerySuc

m\_bOK = true

Release File

if m\_bOK

Delete Node

else

m\_bDiscarded = true

Do Other Things …

Sleep

if m\_bDiscarded

Delete Node

唤醒

Release File

if m\_bOK

Delete Node

else

m\_bDiscarded = true

Block

Sleep

* + - 如果Query File失败，m\_bOK不会被置为true，那么AsynLoading\_Node就泄露了