关键字过滤笔记

# **1 预编译头**

预编译头是Visual Studio提供的预编译功能，具体在项目属性——C++——预编译头中设置。需要StdAfx.h和StdAfx.cpp。**预编译头的原理是编译时将StdAfx编译为中间件.pch，之后在引用#include “StdAfx.h”的文件中将.pch与其进行链接，可以大量减少编译时间。**

需要注意的是，编译器在编译StdAfx时十分耗时，因此StdAfx之中主要用于存放不经常修改的引用，如stl库或者相对于该项目来说是外部的项目。此外，在引用第三方库时，如果第三方库并未使用预编译头的话，需要手动对其.cpp修改，设置其不使用预编译头。

使用预编译头的.cpp都需要#include “StdAfx.h”

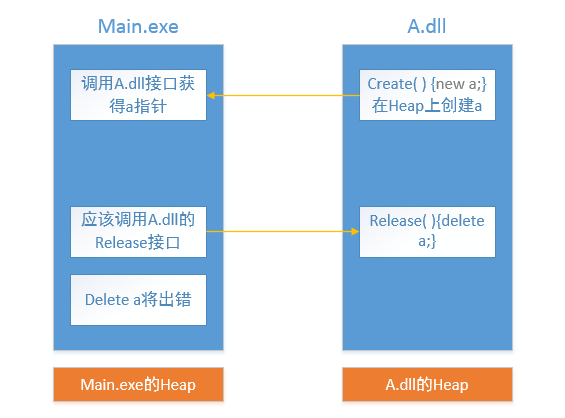
# 2 健壮性

良好的健壮性需要关注每个可能失败的调用点，对于可能失败的情况首先要告知外部调用者此次调用是否成功，成功之后再关注数据。如MultiByteToWideChar（）失败时返回0，因此首先需要对其进行判断。

# 3 new、delete的代价

**临时new、delete**的组合虽然不会造成内存泄露，但是很容易引发内存碎片的问题，该问题在服务端上比较严重。

对于如MultiByteToWideChar（）需要临时创建随机大小的空间的问题，可以使用C++数据结构的resize（）解决，如string.resize（），取string的地址可以通过&string[0]以进行，但是必须保证string[0]是存在的元素，如果string无元素可能引发错误。甚至可能造成以下问题：

图11-1

如果直接在main中delete a的话，程序会认为a建立于main的堆上，会尝试把a的内存归还予main.exe的heap，此时将导致出错。因为存在着2个heap。

# 4 循环 vs 迭代器

当然是选择迭代器区间版本！

# 5 头文件的代码

除非内联情况，否则不要在头文件中编写代码：可能会出现行号错误这类匪夷所思的错误。

# 6 二段构造的三层封装

如果构造函数会失败，需要使用二段构造结构。在使用二段构造结构时，需要做三层封装如下：

底层：A::A（）

中间层：A::Create(参数)——>封装底层

顶层：A::Create（）————>封装中间层

（二段构造结构一般在跨模块时才会使用，同一模块内一般不需要这么做。）

# 7 类的布局

类的布局应该是public函数、public变量、private函数、private变量。（该向外部暴露时暴露，该隐藏时隐藏）

# 8 XML的读写

XML不应该假定其顺序，读取时应该指定到元素标识，而非从上向下数第几个。

XML配置时需要考虑到是否出现需要配置重复信息的情况，应该尽量避免这种情况的出现。

# 9 全局变量

尽最大可能不要使用全局变量，因为在巨大工程中全局变量很容易出现冲突，并且在使用过程中需要关注每个引用。

# 10 类的职责

各类应当职责分明，如Node归属Automaton内部，那么应该由Automaton对Node进行管理。而Automaton属KeyFilter内部，KeyFilter不应该亲自操作Node。

# 11 堆与栈

1. 在对象不接近于1M大小，或者不需要递归多次令栈逼迫上限的情况下，可以使用栈内存就尽量使用栈内存，堆内存容易造成内存碎片。
2. 类成员只要可置于内部，在不导致 1 的情况下，同样应该定义为对象形式而非指针形式。
3. 对于STL容器，不要对子元素使用指针，将容器定义在堆上即可。
4. Class A

{

B m\_b;

C m\_c;

}

这种情况一般new顶层A即可，B、C都会被创建于堆上。

## 11.1 类指针成员

类中含有指针成员——>需要析构函数——>需要拷贝函数

# 12 先行判断

关于指针的先行判断十分重要：

1. 初始化为NULL。
2. Delete时一定必须首先判断是否为NULL，delete之后置为NULL。

# 13 错误处理

错误信息最好一层一层包裹外抛，以知晓是哪一层发生了错误。

# 14 类的完整性

即使不打算通过构造函数使用类的话，也应该保证类的构造函数是可以使用的，即保证类的完整性。