C++ debug 处理

2018 年 04 月 20 日 15:45:22 乌班图 ysm 阅读数：441

https://blog.csdn.net/u012278016/article/details/79903465
assert

断言表示为一些布尔表达式，程序员相信在程序中的某个特定点该表达式值为真。可以在任何时候启用和禁用断言验证，因此可以在测试时启用断言，而在部署时禁用断言。

assert() 宏用法
注意：assert 是宏，而不是函数。在 C 的 assert.h 头文件中。

assert 宏的原型定义在<assert.h>中，其作用是如果它的条件返回错误，则终止程序执行，原型定义：
#include <assert.h>
void assert(int expression); // 0 错误　非 0 正确
assert 的作用是先计算表达式 expression，如果其值为假（即为 0），那么它先向标准错误流 stderr 打印一条出错信息，然后通过调用 abort 来终止程序运行；否则，assert()无任何作用。宏 assert（）一般用于确认程序的正常操作，其中表达式构造无错时才为真值。完成调试后，不必从源代码中删除 assert()语句，因为宏 NDEBUG 有定义时，宏 assert()的定义为空。[1]
请看下面的程序清单 badptr.c：

```
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
int main( void )
{
   FILE *fp;
   fp = fopen( "test.txt", "w" );//以可写的方式打开一个文件，如果不存在就创建一个同名文件
   assert( fp ); //所以这里不会出错
   fclose( fp );
   fp = fopen( "noexitfile.txt", "r" );//以只读的方式打开一个文件，如果不存在就打开文件失败
   assert( fp ); //所以这里出错
   fclose( fp ); //程序永远都执行不到这里来
   return 0;
}
```

//--

使用 assert 的缺点是，频繁的调用会极大的影响程序的性能，增加额外的开销。在调试结束后，可以通过在包含#include <assert.h>的语句之前插入 #define NDEBUG 来禁用 assert 调用，示例代码如下：

```
#include <stdio.h>
#define NDEBUG
#include <assert.h>
```

用法总结与注意事项：

1)在函数开始处检验传入参数的合法性
如:

```
int resetBufferSize(int nNewSize)
{
   //功能:改变缓冲区大小,
   //参数:nNewSize 缓冲区新长度
   //返回值:缓冲区当前长度
   //说明:保持原信息内容不变 nNewSize<=0 表示清除缓冲区
   assert(nNewSize >= 0);
   assert(nNewSize <= MAX_BUFFER_SIZE);
   ...
}
```

2)每个 assert 只检验一个条件,因为同时检验多个条件时,如果断言失败,无法直观的判断是哪个条件失败

```
/***不好***/
assert(nOffset>=0 && nOffset+nSize<=m_nInfomationSize);
/****好****/
assert(nOffset >= 0);
assert(nOffset+nSize <= m_nInfomationSize);
```

3)不能使用改变环境的语句,因为assert 只在DEBUG 个生效,如果这么做,会使用程序在真正运行时遇到问题
错误: assert(i++ < 100)
这是因为如果出错，比如在执行之前 i=100,那么这条语句就不会执行，那么 i++ 这条命令就没有执行。
正确: assert(i < 100)

```
i++;
```

4)assert 和后面的语句应空一行,以形成逻辑和视觉上的一致感

5)有的地方,assert 不能代替条件过滤
注意：当对于浮点数：

```
#include<assert.h>
float pi=3.14f;
assert (pi==3.14f);
```

在 switch 语句中总是要有 default 子句来显示信息(Assert)。

```
int number = SomeMethod();
switch(number)
{
   case 1:
      Trace.WriteLine("Case 1:");
      break;
   case 2:
      Trace.WriteLine("Case 2:");
      break;
   default :
      Debug.Assert(false);
      break;
}
```

```
//=============================================================
CCAssert(cond,msg);
```

cond 和 assert(cond);中一样

VS C++一些 debug[持续更新中。。。]

2018 年 08 月 12 日 17:35:19 _yxy_ 阅读数：63

https://blog.csdn.net/u011643312/article/details/81070587

1、vs2015 中，“error LNK2019: 无法解析的外部符号 __imp__”等问题原因是：缺少相应的库。

项目属性设置中添加相应的库 的方法为：项目->属性->链接器->输入->附加依赖项 中加入相应的库，用分号（;）与其他的库分开。实际中通过增加 ws2_32.lib，解决了该问题。

2、 每次编译加载一堆 dll 符号慢的问题，解决方法：工具->选项->调试->符号 取消 Microsoft 服务器前面的√

3、写 C++构造函数的时候，总是会在.h 文件里直接声明。因为没有用到，所以可能都没有.cpp 文件的实现，或者直接都没有实现，这里就会出现经常出现的问题：

譬如：

错误 52 error LNK2019: 无法解析的外部符号 "public: __thiscall DialogHandler::DialogHandler(char const *)" (??0DialogHandler@@QAE@PBD@Z)，该符号在函数 "public: __thiscall DialogHandlerShop::DialogHandlerShop(void)" (??0DialogHandlerShop@@QAE@XZ) 中被引用 D:\work\cocos2dx_game_popstar_TV\proj.win32\SceneGame.obj

原因是：构造函数没有实现，只是做了声明才导致的！可以在构造函数声明后面加一对大括号。