

只有有耐心圆满完成简单工作的人, 才能够轻而易举地完成困难的事。

Only those who have the patience to do simple things perfectly ever acquire the skill to do difficult things easily.

本博客文章如无特别说明则为原创,
转载请注明出处。

<	2011年9月						>
日	一	二	三	四	五	六	
28	29	30	31	1	2	3	
4	5	6	7	8	9	10	
11	12	13	14	15	16	17	
18	19	20	21	22	23	24	
25	26	27	28	29	30	1	
2	3	4	5	6	7	8	

常用链接


[我的随笔](#)
[我的评论](#)
[我参与的随笔](#)

留言簿(6)

[给我留言](#)
[查看公开留言](#)
[查看私人留言](#)

随笔分类

[C++专栏\(20\)](#) 

[C++博客](#) [首页](#) [新随笔](#) [联系](#) [聚合](#)  [管理](#)

随笔-60 评论-98 文章-0 trackbacks-0

C++关键字: mutable、volatile、explicit以及__based

mutable关键字

关键字mutable是C++中一个不常用的关键字,他只能用于类的非静态和非常量数据成员我们知道一个对象的状态由该对象的非静态数据成员决定,所以随着数据成员的改变,对象的状态也会随之发生变化!

如果一个类的成员函数被声明为const类型,表示该函数不会改变对象的状态,也就是该函数不会修改类的非静态数据成员.但是有些时候需要在该类函数中对类的数据成员进行赋值.这个时候就需要用到mutable关键字了

例如:

```
1 class Demo
2 {
3 public:
4     Demo(){}
5     ~Demo(){}
6 public:
7     bool getFlag() const
8     {
```

[C++Unit专栏\(2\)](#) [E文全翻\(2\)](#) [TDD\(5\)](#) [UI美学\(4\)](#) [XP敏捷\(5\)](#) [克隆cn.msn.com Step by Step\(4\)](#)[理越辩越明\(5\)](#) [面向对象\(3\)](#) [内存专题\(1\)](#) [去年今日\(1\)](#) [软件测试\(6\)](#) [软件设计\(14\)](#) [闲话连篇\(6\)](#) [心路历程\(4\)](#) [性能监测\(1\)](#) 

随笔档案

[2010年2月 \(1\)](#)[2009年6月 \(2\)](#)[2009年5月 \(4\)](#)[2009年2月 \(1\)](#)[2009年1月 \(1\)](#)[2008年10月 \(1\)](#)[2008年9月 \(2\)](#)[2008年8月 \(4\)](#)[2008年7月 \(3\)](#)[2008年6月 \(1\)](#)

```
9 |     m_nAccess + + ;
10 |     return m_bFlag;
11 | }
12 | private:
13 |     int m_nAccess;
14 |     bool m_bFlag;
15 | };
16 |
17 | int main()
18 | {
19 |     return 0;
20 | }
21 |
```

编译上面的代码会出现 error C2166: l-value specifies const object的错误说明在const类型的函数中改变了类的非静态数据成员.这个时候需要使用mutable来修饰一下要在const成员函数中改变的非静态数据成员

m_nAccess,代码如下:

```
1 | class Demo
2 | {
3 | public:
4 |     Demo(){}
5 |     ~Demo(){}
6 | public:
7 |     bool getFlag() const
8 |     {
9 |         m_nAccess + + ;
10 |         return m_bFlag;
11 |     }
12 | private:
13 |     mutable int m_nAccess;
14 |     bool m_bFlag;
15 | };
16 |
```

[2008年4月 \(5\)](#)[2008年3月 \(3\)](#)[2008年2月 \(1\)](#)[2008年1月 \(4\)](#)[2007年12月 \(4\)](#)[2007年10月 \(2\)](#)[2007年9月 \(1\)](#)[2007年8月 \(2\)](#)[2007年5月 \(2\)](#)[2006年10月 \(3\)](#)[2006年9月 \(9\)](#)[2006年8月 \(4\)](#)

相册

[UI Design](#)

我的博器

[HelloPDA.com](#)[好玩上线](#)[T恤吧](#)[老好啦！](#)[让项目管理落地生根](#)[CTO必读](#)

搜索

最新随笔

```
17 int main()
18 {
19     return 0;
20 }
21
```

这样再重新编译的时候就不会出现错误了!

volatile关键字

volatile是c/c++中一个鲜为人知的关键字,该关键字告诉编译器不要持有变量的临时拷贝,它可以适用于基础类型

如: int,char,long.....也适用于C的结构和C++的类。当对结构或者类对象使用volatile修饰的时候,结构或者类的所有成员都会被视为volatile.使用volatile并不会否定对CRITICAL_SECTION,Mutex,Event等同步对象的需要

例如:

```
1 int i;
2 i = i + 3;
```

无论如何,总是会有一小段时间,i会被放在一个寄存器中,因为算术运算只能在寄存器中进行。一般来说,volatile关键字适用于行与行之间,而不是放在行内。

我们先来实现一个简单的函数,来观察一下由编译器产生出来的汇编代码中的不足之处,并观察volatile关键字如何修正这个不足之处。在这个函数体内存在一个busy loop(所谓busy loop也叫做busy waits,是一种高度浪费CPU时间的循环方法)

```
1 void getKey(char* pch)
2 {
3     while (*pch == 0);
4 }
```

当你在VC开发环境中将最优化选项都关闭之后,编译这个程序,将获得以下结果(汇编代码)

1. Visual Studio VS C++Builder07 不同的公司, 同样的Bug
2. [初探Xerces系列]DTD
3. [初探Xerces系列]DOM Lev3 Core的关键特性整理
4. [初探Xerces系列]IDE之惑
5. [初探Xerces系列]对外提供CLI
6. [初探Xerces系列]目录结构也可以辅助提高代码可读性
7. [初探Xerces系列]xerces-c-3.0.1在CB07下编译不过的解决办法
8. 获取版本信息Version的完整代码 (从msdn扩展而来)
9. 初始化const, static, const static, static const成员变量
10. 【zz】设计Qt风格的C++API

最新评论 XML

1. re: UI美学

其实UI美学应该属于信息美学中的一类,因为UI本身是一种提供信息的平台,交互是属于信息的一个属性,因此他和infographic一样是一种信息美学.

--懵中人

2. re: [克隆cn.msn.com Step by Step]Multi Page

评论内容较长,点击标题查看

--MartinaHOLCOMB35

3. re: C++关键字: mutable、volatile、explicit以及__based

今天偶尔看到, 讲的很透彻谢谢

--smileEvday

4. re: 初始化const, static, const static, static const成员变量

```

1 ; while (*pch == 0)
2 $L27
3 ; Load the address stored in pch
4 mov eax, DWORD PTR _pch$[ebp]
5 ; Load the character into the EAX register
6 movsx eax, BYTE PTR [eax]
7 ; Compare the value to zero
8 test eax, eax
9 ; If not zero, exit loop
10 jne $L28
11 ;
12 jmp $L27
13 $L28
14 ;}

```

这段没有优化的代码不断的载入适当的地址, 载入地址中的内容, 测试结果。效率相当的低, 但是结果非常准确现在在我们再来看看将编译器的所有最优化选项开关都打开以后, 重新编译程序, 生成的汇编代码, 和上面的代码

比较一下有什么不同

```

1 ;{
2 ; Load the address stored in pch
3 mov eax, DWORD PTR _pch$[esp-4]
4 ; Load the character into the AL register
5 movsx al, BYTE PTR [eax]
6 ; while (*pch == 0)
7 ; Compare the value in the AL register to zero
8 test al, al
9 ; If still zero, try again
10 je SHORT $L84
11 ;
12 ;}

```

从代码的长度就可以看出来, 比没有优化的情况要短的多。需要注意的是编译器把MOV指令放到了循环之外。这在单线程中是一个非常好的优化, 但是, 在多线程应用程序中, 如果另一个线程改变了变量的值, 则循环永远不会结

明显不对, const 成员变量需要在初始化列表中初始化, 基础中的基础, 以前一个腾讯的人面试我, 他也不知道这个。

--pw

5. re: 我和充斥臭味代码的战争

这绝对是项目管理问题@abettor

--stepinto

阅读排行榜

1. C++关键字: mutable、volatile、explicit以及__based(11001)
2. 初始化const, static, const static, static const成员变量(6176)
3. 读懂常见IRP:IRP_MJ_CLEANUP\IRP_MJ_CLOSE\IRP_MJ_CREATE(4312)
4. 我是笨人——读Rob Pike的《Notes on C Programming》(附全文链接)(3442)
5. 获取版本信息Version的完整代码(从msdn扩展而来)(3331)

评论排行榜

1. 我和充斥臭味代码的战争(26)
2. [克隆cn.msn.com Step by Step]Multi Page(10)
3. Visual Studio VS C++Builder07不同的公司, 同样的Bug(9)
4. 关于实战测试驱动开发的一点感想。(7)
5. UI美学(7)

束。被测试的值永远被放在寄存器中, 所以该段代码在多线程的情况下, 存在一个巨大的BUG。解决方法是重新

写一次getKey函数, 并把参数pch声明为volatile,代码如下:

```

1 void getKey(volatile char* pch)
2 {
3     while (*pch == 0);
4 }

```

这次的修改对于非最优化的版本没有任何影响, 下面请看最优化后的结果:

```

1 {
2     ; Load the address stored in pch
3     mov eax, DWORD PTR _pch$[esp-4]
4     ; while (*pch == 0)
5     $L84:
6     ; Directly compare the value to zero
7     cmp BYTE PTR [eax], 0
8     ; If still zero, try again
9     je SHORT $L84
10    ;
11 }

```

这次的修改结果比较完美, 地址不会改变, 所以地址声明被移动到循环之外。地址内容是volatile, 所以每次循环之中它不断的被重新检查。把一个const volatile变量作为参数传递给函数是合法的。如此的声明意味着函数不能改变变量的值, 但是变量的值却可以被另一个线程在任何时间改变掉。

explicit关键字

我们在编写应用程序的时候explicit关键字基本上是很少使用, 它的作用是"禁止单参数构造函数"被用于自动型别转换, 其中比较典型的例子就是容器类型, 在这种类型的构造函数中你可以将初始长度作为参数传递给构造函数。

例如:

你可以声明这样一个构造函数

```
1 class Array
2 {
3 public:
4     explicit Array(int size);
5     ... ..
6 };
```

在这里explicit关键字起着至关重要的作用,如果没有这个关键字的话,这个构造函数有能力将int转换成Array.一旦这种情况发生,你可以给Array支派一个整数值而不会引起任何的问题,比如:

```
1 Array arr;
2 ...
3 arr = 40;
```

此时,C++的自动型别转换会把40转换成拥有40个元素的Array,并且指派给arr变量,这个结果根本就不是我们想要的结果.如果我们将构造函数声明为explicit,上面的赋值操作就会导致编译器报错,使我们可以及时发现错误.需要注意的是:explicit同样也能阻止"以赋值语法进行带有转型操作的初始化";

例如:

```
1 Array arr(40);//正确
2 Array arr = 40;//错误
```

看一下以下两种操作:

```
1 X x;
2 Y y(x);//显式类型转换
```

另一种

```
1 X x;  
2 Y y = x; //隐式类型转换
```

这两种操作存在一个小小的差别,第一种方式通过显式类型转换,根据型别x产生了型别Y的新对象;第二种方式通过隐式转换产生了一个型别Y的新对象.explicit关键字的应用主要就是上面所说的构造函数定义种,参考该关键字的应用可以看看STL源代码,其中大量使用了该关键字

__based关键字

该关键字主要用来解决一些和共享内存有关的问题,它允许指针被定义为从某一点开始算的32位偏移值,而不是内存种的绝对位置

举个例子:

```
1 typedef struct tagDEMOSTRUCT {  
2     int a;  
3     char sz[10];  
4 } DEMOSTRUCT, * PDEMOSTRUCT;  
5  
6 HANDLE hFileMapping = CreateFileMapping(...);  
7 LPVOID lpShare = (LPDWORD)MapViewOfFile(...);  
8  
9 DEMOSTRUCT __based(lpShare)* lpDemo;  
10
```

上面的例子声明了一个指针lpDemo,内部储存的是从lpShare开始的偏移值,也就是lpHead是以lpShare为基准的偏移值.

上面的例子种的DEMOSTRUCT只是随便定义的一个结构,用来代表任意的结构.

虽然__based指针使用起来非常容易,但是,你必须在效率上付出一定的代价.每当你用__based指针处理数据,CPU都必须为它加上基地址,才能指向真正的位置.

posted on 2008-04-09 10:13 [创建更好的解决方案](#) 阅读(11001) [评论\(4\)](#) [编辑](#) [收藏](#) [引用](#) 所属分类: [C++专栏](#)

评论:

re: C++关键字：mutable、volatile、explicit以及__based 2009-04-11 14:47 | ncy_wisdom
侯捷的书上有吧，算原创吗？ [回复](#) [更多评论](#)

re: C++关键字：mutable、volatile、explicit以及__based 2009-05-04 17:50 | cloved
先转载,然后慢慢学习. [回复](#) [更多评论](#)

re: C++关键字：mutable、volatile、explicit以及__based[未登录] 2010-04-16 16:53 | 菜鸟
造诣很高，多指教 [回复](#) [更多评论](#)

re: C++关键字：mutable、volatile、explicit以及__based 2011-09-13 10:53 | smileEvday
今天偶尔看到，讲的很透彻谢谢 [回复](#) [更多评论](#)

[刷新评论列表](#)

[找优秀程序员，就在博客园](#)

标题

姓名

主页

验证码 * 3092

内容(提交失败后,可以通过“恢复上次提交”恢复刚刚提交的内容)

☒ Remember Me?

提交

[登录](#) [使用高级评论](#) [新用户注册](#) [返回首页](#) [恢复上次提交](#)

[使用Ctrl+Enter键可以直接提交]

[【推荐】超50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库](#)



相关文章:

[Visual Studio VS C++Builder07 不同的公司, 同样的Bug](#)

[\[初探Xerces系列\]对外提供CLI](#)

[\[初探Xerces系列\]目录结构也可以辅助提高代码可读性](#)

2015年2月12日

C++关键字: mutable、volatile、explicit以及__based - 只有有耐心圆满完成简单工作的人, 才能够轻而易举地完成困难的事。 - C++博客

[\[初探Xerces系列\]xerces-c-3.0.1在CB07下编译不过的解决办法](#)

[获取版本信息Version的完整代码 \(从msdn扩展而来\)](#)

[初始化const, static, const static, static const成员变量](#)

[BCB的两个问题](#)

[狗, 哈士奇, 跳蚤, 继承, 聚合, UpCast和DownCast](#)

[重构故事——Chapter03](#)

[重构故事——Chapter02](#)

网站导航: [博客园](#) [IT新闻](#) [BlogJava](#) [知识库](#) [程序员招聘](#) [管理](#)

Powered by: [博客园](#) 模板提供: [沪江博客](#) Copyright ©2015 创建更好的解决方案