C/C++ 中 volatile 关键字详解

1、为什么用volatile?

避免编译器优化，实现对特殊地址的稳定访问

C/C++ 中的 volatile 关键字和 const 对应，用来修饰变量，通常用于建立语言级别的 **memory barrier**。这是 BS 在 "The C++ Programming Language" 对 volatile 修饰词的说明：

> A volatile specifier is a hint to a compiler that an object may change its value in ways not specified by the language so that aggressive optimizations must be avoided.

volatile 关键字是一种类型修饰符，用它声明的类型变量表示可以被某些编译器未知的因素更改，比如：操作系统、硬件或者其它线程等。**遇到这个关键字声明的变量，编译器对访问该变量的代码就不再进行优化，从而可以提供对特殊地址的稳定访问。**

声明时语法：`int volatile vInt;`

当要求使用 volatile 声明的变量的值的时候，系统总是重新从它所在的内存读取数据，即使它前面的指令刚刚从该处读取过数据。而且读取的数据立刻被保存。例如：

```
volatile int i=10;
int a = i;
...
// 其他代码，并未明确告诉编译器，对 i 进行过操作
int b = i;
```

volatile 指出 i 是随时可能发生变化的，每次使用它的时候必须从 i的地址中读取，因而编译器生成的汇编代码会重新从i的地址读取数据放在 b 中。

而优化做法是，由于编译器发现两次从 i读数据的代码之间的代码没有对 i 进行过操作，它会自动把上次读的数据放在 b 中。而不是重新从 i 里面读。这样以来，如果 i是一个寄存器变量或者表示一个端口数据就容易出错，所以说 **volatile 可以保证对特殊地址的稳定访问。**

注意，在 VC 6 中，一般调试模式没有进行代码优化，所以这个关键字的作用看不出来。下面通过插入汇编代码，测试有无 volatile 关键字，对程序最终代码的影响，输入下面的代码：

实例1

```
#include <stdio.h>

void main()
{
    int i = 10;
    int a = i;

    printf("i = %d", a);

    // 下面汇编语句的作用就是改变内存中 i 的值
    // 但是又不让编译器知道
    __asm {
        mov dword ptr [ebp-4], 20h
    }
```

```
15
16      int b = i;
17      printf("i = %d", b);
18  }
```

然后，在 Debug 版本模式运行程序，输出结果如下：

```
1  i = 10
2  i = 32
```

然后，在 Release 版本模式运行程序，输出结果如下：

```
1  i = 10
2  i = 10
```

输出的结果明显表明，Release 模式下，编译器对代码进行了优化，第二次没有输出正确的 i 值。下面，我们把 i 的声明加上 volatile 关键字，看看有什么变化：

实例2

```
1  #include <stdio.h>
2
3  void main()
4  {
5      volatile int i = 10;
6      int a = i;
7
8      printf("i = %d", a);
9      __asm {
10         mov dword ptr [ebp-4], 20h
11     }
12
13     int b = i;
14     printf("i = %d", b);
15 }
```

分别在 Debug 和 Release 版本运行程序，输出都是：

```
1  i = 10
2  i = 32
```

用途

这说明这个 volatile 关键字发挥了它的作用。其实不只是内嵌汇编操纵栈"这种方式属于编译无法识别的变量改变，另外更多的可能是多线程并发访问共享变量时，一个线程改变了变量的值，怎样让改变后的值对其它线程 visible。一般说来，volatile用在如下的几个地方：

- 1) 中断服务程序中修改的供其它程序检测的变量需要加 volatile；
- 2) 多任务环境下各任务间共享的标志应该加 volatile；
- 3) 存储器映射的硬件寄存器通常也要加 volatile 说明，因为每次对它的读写都可能由不同意义；

2、volatile 指针

和 const 修饰词类似，const 有常量指针和指针常量的说法，volatile 也有相应的概念：

修饰由指针指向的对象、数据是 const 或 volatile 的：

```
const char* cpch;
volatile char* vpch;
```

注意：对于 VC，这个特性实现在 VC 8 之后才是安全的。

指针自身的值——一个代表地址的整数变量，是 const 或 volatile 的：

```
char* const pchc;
char* volatile pchv;
```

注意：

- (1) 可以把一个非volatile int赋给volatile int，但是不能把非volatile对象赋给一个volatile对象。
- (2) 除了基本类型外，对用户定义类型也可以用volatile类型进行修饰。
- (3) C++中一个有volatile标识符的类只能访问它接口的子集，一个由类的实现者控制的子集。用户只能用const_cast来获得对类型接口的完全访问。此外，volatile向const一样会从类传递到它的成员。

3、多线程下的volatile

有些变量是用 volatile 关键字声明的。当两个线程都要用到某一个变量且该变量的值会被改变时，应该用 volatile 声明，该关键字的作用是防止优化编译器把变量从内存装入 CPU 寄存器中。如果变量被装入寄存器，那么两个线程有可能一个使用内存中的变量，一个使用寄存器中的变量，这会造成程序的错误执行。volatile 的意思是让编译器每次操作该变量时一定要从内存中真正取出，而不是使用已经存在寄存器中的值，如下：

```
volatile  BOOL  bStop  =  FALSE;
```

(1) 在一个线程中：

```
while(  !bStop  )  {  ...  }
bStop  =  FALSE;
return;
```

(2) 在另外一个线程中，要终止上面的线程循环：

```
bStop  =  TRUE;
while(  bStop  );  //等待上面的线程终止，如果bStop不使用volatile申明，那么这个循环将是一个死循环，因为bStop已经读取到了寄存器中，寄存器中bStop的值永远不会变成FALSE，加上volatile，程序在执行时，每次均从内存中读出bStop的值，就不会死循环了。
```

这个关键字是用来设定某个对象的存储位置在内存中，而不是寄存器中。因为一般的对象编译器可能会将其的拷贝放在寄存器中用以加快指令的执行速度，例如下段代码中：

```
...
for(int nMyCounter = 0; nMyCounter<100; nMyCounter++)  {
...
}
...
```

在此段代码中，nMyCounter 的拷贝可能存放到某个寄存器中（循环中，对 nMyCounter 的测试及操作总是对此寄存器中的值进行），但是另外又有段代码执行了这样的操作：`nMyCounter -= 1;` 这个操作中，对 nMyCounter 的改变是对内存中的 nMyCounter 进行操作，于是出现了这样一个现象：nMyCounter 的改变不同步。