Morden cpp note — const volatile mutable

Friday, December 4, 2020 8:38 AM

✱ const and volatile

```
const int MAX_LEN = 1024;
const std::string NAME = "metroid";
```

const 和宏定义的区别：const定义在常量的预处理阶段不存在，直到运行阶段才出现。所以类似于只读变量。

```
//需要加上 volatile,
const volatile int MAX_LEN = 1024;
auto ptr = (int*) (& MAX_LEN);
*ptr = 2048;
cout << MAX_LEN << endl; // 输出2048.
```

加上volatile后，编译器不会再把MAX_LEN替换为1024，而是去内存里取值，而它已通过指针被强制修改。

const 可以理解为 read only,
volatile 会禁止编译器的优化，除非必要，应少用.

在C++里，还有引用类型和指针类型，加上const就成了常量引用和常量指针.

```
int x = 100;
const int& rx = x;
const int* px = &x;
```

const& 被称为万能引用，可以引用任何类型，即不管是值，指针，左引用还是右引用，而且会给变量附上const特性，变量就成了常量，只能读，禁止写。编译器会帮你检查所有对它的操作，发出警告，在编译阶段防止修改.

因此，在设计函数的时候，尽可能用它作为入口参数，保证效率和安全。

const用于指针，常见的用法是const放在声明的最左边，表示指向常量的指针。指针指向的是只读变量，不允许修改.

```
string name = "uncharted";
const string* ps1 = & name; //指向常量
*ps1 = "spiderman"; // 错误，不允许修改.
```

另一种用法是，const在*右边，表示指针不能被修改，而指向的变量可以被修改.

```
string* const ps2 = &name; // 指向变量，但指针本身不能被修改.
*ps2 = "spiderman"; // 正确.允许修改.
```

为了代码的可读性，建议不用* const形式.

- 与类相关的const用法.

const 成员函数.

```
class DemoClass final
{
private:
    const long MAX_SIZE = 256; // const成员变量
    int m_value; // 成员变量

public:
    int get_value() const   //const成员函数
    {
      return m_value;
    }
};
```

const放在函数的后面，表示这个函数是一个常量，如果在前面，代表返回值是const int.
const 成员函数的意思，并不是函数不可修改，而是执行中是const的，不会修改对象的状态（即成员变量），也就是说，成员函数是一个只读操作。

	const	非 const
对象(实例)	(cost T) 对象只读，只能调用const成员函数.	可以修改对象， 调用任意成员函数
引用	(const T&) 引用的对象只读，只能调用 const成员函数.	
指针	(const T*) 指针指向的对象只读，只能调用const成员函数.	
成员函数	(func() const) 不允许修改成员变量	可以修改成员变量

可以借鉴一下标准库，比如vector，它的empty(), size(), capacity() 等查看基本属性都是const，而reserve(), clear(), erase() 则是非const.

- mutable.

mutable只能用来修饰类里面的成员变量，表示变量即使是在const对象里，也是可以修改的。标记为mutable的成员不会改变对象的状态，也就是不影响对象的常量属性，所以允许const成员函数改写mutable成员变量.

mutable像是给const对象打了一个补丁，让它部分可变。因为对象与普通的int, double不同，内部会有很多成员变量表示状态，但因为封装的特性，外界只能看到一部分状态，判断对象是否const，应该由外部可观测的状态特征来决定.

Eg. 对象内部用mutex来保证线程安全，或者有一个缓冲区来暂存数据，再或者有一个原子变量做引用计数，这些属于内部的私有实现细节，外面看不到，变与不变不会改变外界看到的常量性。这时，如果const成员函数不允许修改它们，就说不过去了。

所以，对于有特殊作用的成员变量，可以加上mutable修饰，解除const的限制，让任何成员函数都可以操作它。

```
class DemoClass final
{
private:
   mutable mutex_type m_mutex; // mutable成员变量

public:
   void save_data() const   // const成员函数.
   {
     //do sth with m_mutex
   }
};
```

和volatile一样，mutable也要少作，慎用.

总结：尽可能多用const，让代码更安全。

- 常量/变量
  - volatile
    - 禁止编译器优化
    - 影响性能
  - const
    - 修饰变量，只读
    - 修饰成员函数，不改变对象状态.
    - 可以被编译器优化.
    - const* 常量指针
    - const& 是万能引用
  - mutable
    - 修饰成员变量
    - 不影响对象的常量性.

因为函数参数是"传值"语义，所以对于简单的值类型如int, double，不用 const& 也不会影响效率.

const_cast是C++的四个转型操作符之一，专门用来去除"常量性"，可以用在某些特殊场景，比如调用纯C接口，但应少用.

成员函数有一个隐含的this参数，所以从语义上来说，const成员函数实际上是传入了一个 const this 指针，但因为C++语法限制，无法声明const this，所以就把const放到了函数后面.

依据应用场景，有的成员函数可能既是const又是非const，所以就会用两种重载形式，比如vector的front(), at()等，如果是const对象编译器就会用const版本.

C++11引入了新关键字，constexpr，能够表示编译阶段的常量，

- C++指针和引用的区别：指针是内存地址，引用是变量别名，指针可以为空，而引用不能为空.