07 const/volatile/mutable:常量/变量究竟是怎么回事

const与volatile

合称cv修饰符

const

与宏的区别:const定义的常量在预处理阶段不存在,直到运行阶段才会出现

它是运行时"变量",只是不允许修改,"只读变量"　　可以通过volatile来破坏它的"常量性"

```
const volatile int n = 1;
auto ptr = (int*)(&n);
*ptr = 2;
cout << n;  //2
```

const虽然不是"真正的常数",但编译器会优化, 将所有const常量替换成原始值

没有volatile的const,即使你改了const常量的值,但这个值运行期间用不到,因为它在编译阶段被优化掉了　　相当于所有的n在编译阶段都替换成了1,你就修改不了了

volatile

"不稳定的",提示编译器别动老子,所以会禁止优化

尽量少用volatile,不过可用volatile修饰在多个线程中使用的原生类型变量,见博客中的例子

基本const用法

常量引用　　cosnt &:万能引用,常用于修饰函数的入参:既有效率,也安全

常量指针

指针常量

与类相关的const用法

const成员函数

eg: int get_value() const　　表示函数的执行是const的,不会修改成员变量

即成员函数是一个"只读操作"

如果操作const变量的成员函数不是const的,则不允许调用　　既然对象是const的,那么它所有相关操作必然是const

	const	非const
对象（实例）	(const 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

子主题

借鉴标准库:　　vector的empty(),size(),capacity()是const的

reserve(),clear(),erase()是非const的

mutable

可变的,会变的　　和volatile用法大相径庭

volatile可修饰任何变量

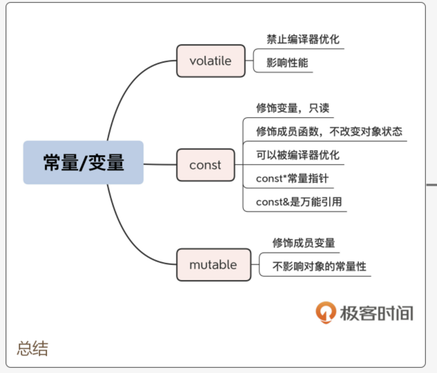
mutable只能修饰类成员变量,表示即使在const对象中也是可以修改的

标记为mutable的成员不会影响对象的常量性,所以允许const成员函数改写mutable成员变量

尽管前面说const成员函数不允许修改成员变量,那些指的是非mutable的成员变量,如果是mutable的,就可以修改之.

给有特殊作用的成员变量加上mutable,让任何成员函数都可以操作它

总结



尽可能多的使用const,尤其是多线程中,把只读属性持续传递出去