

始终

不忘初心



# C++ 中的 mutable 关键字

📅 2017 年 05 月 25 日 | 📅 2020 年 02 月 04 日 | 📁 [Algorithm and Computer Science](#) | 👁 17161

📄 2k | ⌚ 4 分钟

此篇介绍 C++ 中的 mutable 关键字。

## 类中的 mutable

mutable 从字面意思上来说，是「可变的」之意。

若是要「顾名思义」，那么这个关键词的含义就有些意思了。显然，「可变的」只能用来形容变量，而不可能是「函数」或者「类」本身。然而，既然是「变量」，那么它本来就是可变的，也没有必要使用 mutable 来修饰。那么，mutable 就只能用来形容某种不变的东西了。

C++ 中，不可变的变量，称之为常量，使用 const 来修饰。然而，若是 const mutable 联用，未免让人摸不着头脑——到底是可变还是不可变呢？

事实上，mutable 是用来修饰一个 const 示例的部分可变的数据成员的。如果说得更清晰一点，就是说 mutable 的出现，将 C++ 中的 const 的概念分成了两种。

- 二进制层面的 const，也就是「绝对的」常量，在任何情况下都不可修改（除非用 `const_cast`）。
- 引入 mutable 之后，C++ 可以有逻辑层面的 const，也就是对一个常量实例来说，从外部观察，它是常量而不可修改；但是内部可以有非常量的状态。

当然，所谓的「逻辑 const」，在 C++ 标准中并没有这一称呼。这只是为了方便理解，而创造出来的名词。

显而易见，mutable 只能用来修饰类的数据成员；而被 mutable 修饰的数据成员，可以在 const 成员函数中修改。



这里举一个例子，展现这类情形。

```
1  class HashTable {
2      public:
3          //...
4          std::string lookup(const std::string& key) const
5          {
6              if (key == last_key_) {
7                  return last_value_;
8              }
9
10             std::string value{this->lookupInternal(key)};
11
12             last_key_ = key;
13             last_value_ = value;
14
15             return value;
16         }
17
18     private:
19         mutable std::string last_key_
20         mutable std::string last_value_;
21     };
```

这里，我们呈现了一个哈希表的部分实现。显然，对哈希表的查询操作，在逻辑上不应该修改哈希表本身。因此，HashTable::lookup 是一个 const 成员函数。在 HashTable::lookup 中，我们使用了 last\_key\_ 和 last\_value\_ 实现了一个简单的「缓存」逻辑。当传入的 key 与前次查询的 last\_key\_ 一致时，直接返回 last\_value\_；否则，则返回实际查询得到的 value 并更新 last\_key\_ 和 last\_value\_。

在这里，last\_key\_ 和 last\_value\_ 是 HashTable 的数据成员。按照一般的理解，const 成员函数是不允许修改数据成员的。但是，另一方面，last\_key\_ 和 last\_value\_ 从逻辑上说，修改它们的值，外部是无有感知的；因此也就不会破坏逻辑上的 const。为了解决这一矛盾，我们用 mutable 来修饰 last\_key\_ 和 last\_value\_，以便在 lookup 函数中更新缓存的键值。

## Lambda 表达式中的 mutable

C++11 引入了 Lambda 表达式，程序员可以凭此创建匿名函数。在 Lambda 表达式的设计中，捕获变量有几种方式；其中按值捕获（Capture by Value）的方式不允许程序员在 Lambda 函数的函数体中修改捕获的变量。而以 mutable 修饰 Lambda 函数，则可以打破这种限制。

```
1 int x{0};
2 auto f1 = [=]() mutable {x = 42;}; // okay, 创建了一个函数类型的实例
3 auto f2 = [=]() {x = 42;}; // error, 不允许修改按值捕获的外部变量的值
```

需要注意的是，上述 f1 的函数体中，虽然我们给 x 做了赋值操作，但是这一操作仅只在函数内部生效——即，实际是给拷贝至函数内部的 x 进行赋值——而外部的 x 的值依旧是 0。



俗话说，投资效率是最好的投资。如果您感觉我的文章质量不错，读后收获很大，预计能为您提高 10% 的工作效率，不妨小额捐助我一下，让我有动力继续写出更多好文章。

打赏

**本文作者：** Liam Huang

**本文链接：** <https://liam.page/2017/05/25/the-mutable-keyword-in-Cxx/>

**版权声明：** 本博客所有文章除特别声明外，均采用 [CC BY-NC-SA](#) 许可协议。转载请注明出处！

[# C++](#) [# Mutable](#)

◀ 在 VIM 中切换横竖分屏

GDB 入门教程：调试 ncurses 相关 bug 的完整范例



3 条评论

未登录用户 ▾



说点什么



① 支持 Markdown 语法

使用 GitHub 登录

预览



share7we 发表于 大约 1 年前



写的非常好



bj-yanfei 发表于 11 个月前



今天在看小争哥的一个阻塞队列的示例代码，看到mutable这个关键字，于是百度搜索一下。来到这里之后突然发现题主的名字好熟悉，原来我看的代码就是题主写的。在此膜拜大神。



Liam0205 发表于 11 个月前



@bj-yanfei 不敢当。欢迎多交流~

© 2013 – 2020 👤 Liam Huang | 🏠 1.3m | ☕ 39:53

由 [Hexo](#) & [NexT.Gemini](#) 强力驱动

👁 3006208