# 03\_unique\_ptr

## 前言

unique\_ptr采用的是独享所有权语义，对其持有的堆内存具有唯一拥有权。转移一个unique\_ptr将会把所有权全部从源指针转移给目标指针，源指针被置空；所以unique\_ptr不支持普通的拷贝和赋值操作，不能用在STL标准容器中；局部变量的返回值除外（因为编译器知道要返回的对象将要被销毁）；如果你拷贝一个unique\_ptr，那么拷贝结束后，这两个unique\_ptr都会指向相同的资源，造成在结束时对同一内存指针多次释放而导致程序崩溃。

## unique\_ptr的使用

### 初始化

1. 使用new关键字初始化

std::unique\_ptr<int> up1(new int(123));

1. reset函数初始化

std::unique\_ptr<int> up2;

up2.reset(new int(123));

1. make\_unique初始化

std::unique\_ptr<int> up3 = std::make\_unique<int>(123);

unique\_ptr是一个独占型的智能指针，它不允许其他的智能指针共享其内部的指针，不允许通过赋值将一个unique\_ptr赋值给另一个unique\_ptr。

unique\_ptr<int> up1 = make\_unique<int>(10);

unique\_ptr<int> up2 = up1; // 报错，独占指针不能赋值给另一个独占指针，共享指针也不行

shared\_ptr<int> sp1 = up1; // 报错

### unique\_ptr对象转移

使用移动构造可以把unique\_ptr对象转移给另一个对象。

int main()

{

std::unique\_ptr<int> up1(std::make\_unique<int>(123));

std::unique\_ptr<int> up2(std::move(up1)); //通过移动实现了复制操作

std::cout << ((up1.get() == nullptr) ? "up1 is NULL" : "up1 is not NULL") << std::endl;

std::unique\_ptr<int> up3;

up3 = std::move(up2); //通过移动实现了复制操作

std::cout << ((up2.get() == nullptr) ? "up2 is NULL" : "up2 is not NULL") << std::endl;

return 0;

}

运行结果：

up1 is NULL

up2 is NULL

以上代码利用std::move将up1持有的堆内存（值为123）转移给up2，再把up2转移给up3。最后，up1和up2不再持有堆内存的引用，变成一个空的智能指针对象。并不是所有的对象的std::move操作都有意义，只有实现了移动构造函数或移动赋值运算符的类才行，而std::unique\_ptr正好实现了这二者。

## unique\_ptr常用函数

构造函数：用于创建unique\_ptr实例。

reset()：释放当前管理的对象，并将unique\_ptr置为空。

release()：返回裸指针并放弃所有权，unique\_ptr变为空。

get()：返回裸指针，但不放弃所有权。

operator->() 和 operator\*()：重载了箭头运算符和解引用运算符，用于访问指向的对象。

## 总结

以上就是今天要讲的内容，后续会有更多内容。

## 参考资料

版权声明：本文参考了其他资料和CSDN博主的文章，遵循CC 4.0 BY-SA版权协议，现附上原文出处链接及本声明。

1. https://blog.csdn.net/haiguangboy/article/details/122278954
2. https://blog.csdn.net/weixin\_40026739/article/details/136397613