参考：<https://book.2cto.com/201306/25367.html>

STL模板库提供了一个函数std::move()，实际上std::move()并不能移动任何东西，唯一的功能是将一个左值强制转换为右值引用。

例：程序C++11\_test11

int c = 0;

// int&& rr1 = c; // error，右值引用不能绑定左值

int&& rr1 = *move*(c); // ok，move()将左值c强制转换为右值

调用std::move，就意味着承诺（只是承诺）：除了对c赋值或销毁外，不再使用它。

例：程序C++11\_test11

class Moveable

{

public:

Moveable() : ptr\_(new int(3))

{

*cout* << "Construct Moveable" << *endl*;

}

// 拷贝构造函数

Moveable(const Moveable& org) : ptr\_(new int(\*org.ptr\_))

{

*cout* << "Copy construct Moveable" << *endl*;

}

// 移动构造函数

Moveable(Moveable&& org) : ptr\_(org.ptr\_)

{

*cout* << "Move construct Moveable" << *endl*;

org.ptr\_ = nullptr;

}

~Moveable()

{

if (ptr\_ != nullptr)

{

*cout* << "delete ptr\_" << *endl*;

delete ptr\_;

ptr\_ = nullptr;

}

}

public:

int\* ptr\_;

};

int main(int argc, char\* argv[])

{

int c = 0;

// int&& rr1 = c; // error，右值引用不能绑定左值

int&& rr1 = std::*move*(c); // ok，move()将左值c强制转换为右值

{

Moveable m;

Moveable m1(m);

}

*cout* << *endl*;

{

Moveable m;

Moveable m2(std::*move*(m));

// error，抛出异常

// 对象m中的指针ptr\_的内存已被移动给了m2

// cout << \*m.ptr\_ << endl;

}

return 0;

}

输出为：

Construct Moveable

Copy construct Moveable

delete ptr\_

delete ptr\_

Construct Moveable

Move construct Moveable

delete ptr\_

从输出可以看出，使用移动构造后，内存少了一次开销。

使用std::move()移动后，源对象必须可以析构。

合成的移动构造函数：

只有当一个类没有定义任何自己版本的拷贝控制成员，且类的每个非static数据成员都可以移动时，编译器才会为他合成移动构造函数或移动赋值运算符。编译器可以移动内置类型成员，如果成员是一个类类型，且该类有对应的移动操作，编译也能移动这个成员。