移动构造函数

c++11新增
与拷贝构造相似,但不同
是真实的转移,原对象会丢失其内容,内容会被目标对象占有
当移动值的对象是临时变量的时候(未命名的变量){函数返回值, 类型转换的对象}. 好处: 对对象管理它们使用的存储空间很有用的.将A的内存转移给B,没有分配新的内存

```
// 移动构造函数和赋值
#include <iostream>
#include <string>
using namespace std;

class Example6 {
    string* ptr;
public:
    Example6 (const string& str) : ptr(new string(str)) {}
    ~Example6 () {delete ptr;}
    // 移动构造函数，参数x不能是const Pointer&& x,
    // 因为要改变x的成员数据的值;
    // C++98不支持，C++0x (C++11) 支持
    Example6 (Example6&& x) : ptr(x.ptr)
    {
        x.ptr = nullptr;
    }
    // move assignment
    Example6& operator= (Example6&& x)
    {
        delete ptr;
        ptr = x.ptr;
        x.ptr=nullptr;
        return *this;
    }
    // access content:
    const string& content() const {return *ptr;}
    // addition:
    Example6 operator+(const Example6& rhs)
    {
        return Example6(content()+rhs.content());
    }
};
int main () {
    Example6 foo("Exam");            // 构造函数
    // Example6 bar = Example6("ple"); // 拷贝构造函数
    Example6 bar(move(foo));      // 移动构造函数
                                  // 调用move之后，foo变为一个右值引用变量，
                                  // 此时，foo所指向的字符串已经被"掏空"，
                                  // 所以此时不能再调用foo
    bar = bar+ bar;               // 移动赋值，在这儿"="号右边的加法操作，
                                  // 产生一个临时值，即一个右值
```

```
                                    // 所以此时调用移动赋值语句
    cout << "foo's content: " << foo.content() << '\n'; // 没有输出,因为被移走了
    cout << "foo's content: " << bar.content() << '\n'; // 输出结果 ExamExam
    return 0;
}
```