C++11中的右值引用：T&&，但是通过T&&定义的不一定都是右值

C++中的右值没有地址，不能修改。在C++11中通过右值引用的临时对象

例：程序stl\_test5

int a = 5;

const int b = 6;

int c = 7;

// 非常量右值引用可以绑定到非常量右值

int&& r1 = 4; // 非常量右值

//int&& r2 = a; // error，右值引用不能绑定到左值

int&& r3 = a + c; // 临时对象，非常量右值

// 常量右值引用可以绑定到非常量右值和常量右值

// 常量右值引用本身没有现实意义

const int&& r4 = 5;

const int&& r5 = a + b; // 常量右值

//const int&& r6 = a; // error，不能绑定到左值

// 注意：绑定到右值的右值引用本身是一个左值

// 因为它有名字

//int&& r7 = r1; // error，不能绑定到左值

int&& r8 = *move*(r1); // ok，通过move将左值变成右值

右值引用的第一个重要作用是移动语义，解决C++在拷贝临时对象时不必要的资源拷贝。

例：程序stl\_test6

class P

{

public:

P()

{

*cout* << "构造函数" << *endl*;

}

P(const P& p)

{

*cout* << "拷贝构造函数" << *endl*;

}

~P()

{

*cout* << "析构函数" << *endl*;

}

};

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

{

*vector*<P> vec\_p;

P p1;

// vector容器在插入p1时

// 会首先调用拷贝构造函数

// 构造一个临时对象插入容器中

vec\_p.*push\_back*(p1);

return 0;

}

先了解C++11中的引用叠加规则。

例：程序stl\_test7