|  |  |
| --- | --- |
| 1. 对象移动 2. 右值引用 3. 左值持久：右值短暂 4. 变量是左值 5. 标准库move函数 | Int I = 42;  Int &&rr = I;// 不能绑定到左值上  Int &&rr = I \* 42;//正确  Const int &r3 = i\*42;//const的引用可以绑定到右值上  Int &&r1 = 42;  Int &&r2 = r1;//错误，r1是左值的  Int &&r2 =std::move(r1);//正确 |
| 1. 对象移动 2. 移动而非拷贝对象可以大幅提升性能。 3. 另一个原因是unique\_ptr 与IO类这些类都包含不能被共享的资源。这些类型的对象不能拷贝但可以移动。 4. 旧C++标准中没有直接的方法移动对象。对象本身要求分配内存空间进行不必要的拷贝，或者对象较大 那么拷贝代价非常高 5. 右值引用 6. 右值引用就是必须绑定到右值的引用。通过&&而不是&来获得右值引用。 7. 重要形状：只能绑定到一个将要销毁的对象。因此，可以将一个右值引用的资源“移动”到另一个对象中 8. 左值表达式表示的是一个对象的身份，右值表达式表示的是对象的值 9. 常规引用称为“左值引用”，不能绑定到要求转换的表达式、字面常量或是返回右值的表达式 10. 右值引用可以以上，但是不能直接绑定到一个左值上。 11. 右值引用也是某个对象的另一个名字而已 12. 左值持久：右值短暂   右值只能绑定到临时对象  1) 所引用的对象将要被销毁  2) 该对象没有其它用户  右值引用的代码可以自由的接管所引用的对象的资源  Note:右值引用指向将要被销毁的对象。我们可以从绑定到右值引用的对象“窃取”状态   1. 变量是左值 2. 变量表达式都是左值，一个右值引用是左值，所以不能讲一个右值引用绑定到一个右值引用类型的变量上 3. 因为变量是持久的，到离开作用域才被销毁。   Note:变量是左值，所以不能将一个右值引用直接绑定到一个变量上，即使这个变量是右值引用类型也不行。   1. 标准库move函数 2. 虽然不能直接，但可以显式地将一个左值转换为右值引用类型， utility头文件中move 3. Move意味承诺：除了对rr1（被移动对象）赋值和销毁它外，将不再使用它。调用move后，不能对移后源对象的值做任何假设使用   Note:可以销毁赋值一个移后源对象，但不能使用。  3)) 直接使用std::move，不是用using声明。 | |