面向对象和泛型编程

请你回答一下什么是右值引用，跟左值又有什么区别？ —这里描述的有问题，感觉把简单的问题复杂化了，详情见c++prime plus笔记

右值引用是C++11中引入的新特性 , 它实现了转移语义和精确传递。它的主要目的有两个方面：

1. 消除两个对象交互时不必要的对象拷贝，节省运算存储资源，提高效率。
2. 能够更简洁明确地定义泛型函数。

左值和右值的概念：

左值：能对表达式取地址、或具名对象/变量。一般指表达式结束后依然存在的持久对象。

右值：不能对表达式取地址，或匿名对象。一般指表达式结束就不再存在的临时对象。

右值引用和左值引用的区别：

1. 左值可以寻址，而右值不可以。
2. 左值可以被赋值，右值不可以被赋值，可以用来给左值赋值。
3. 左值可变,右值不可变（仅对基础类型适用，用户自定义类型右值引用可以通过成员函数改变）。

左值、右值 在C++11中所有的值必属于左值、右值两者之一，右值又可以细分为纯右值、将亡值。在C++11中可以取地址的、有名字的就是左值，反之，不能取地址的、没有名字的就是右值（将亡值或纯右值）。举个例子，int a = b+c, a 就是左值，其有变量名为a，通过&a可以获取该变量的地址；表达式b+c、函数int func()的返回值是右值，在其被赋值给某一变量前，我们不能通过变量名找到它，＆(b+c)这样的操作则不会通过编译。

右值、将亡值 在理解C++11的右值前，先看看C++98中右值的概念：C++98中右值是纯右值，纯右值指的是临时变量值、不跟对象关联的字面量值。临时变量指的是非引用返回的函数返回值、表达式等，例如函数int func()的返回值，表达式a+b；不跟对象关联的字面量值，例如true，2，”C”等。

C++11对C++98中的右值进行了扩充。在C++11中右值又分为纯右值（prvalue，Pure Rvalue）和将亡值（xvalue，eXpiring Value）。其中纯右值的概念等同于我们在C++98标准中右值的概念，指的是临时变量和不跟对象关联的字面量值；将亡值则是C++11新增的跟右值引用相关的表达式，这样表达式通常是将要被移动的对象（移为他用），比如返回右值引用T&&的函数返回值、std::move的返回值，或者转换为T&&的类型转换函数的返回值。

将亡值可以理解为通过“盗取”其他变量内存空间的方式获取到的值。在确保其他变量不再被使用、或即将被销毁时，通过“盗取”的方式可以避免内存空间的释放和分配，能够延长变量值的生命期。

左值引用、右值引用 左值引用就是对一个左值进行引用的类型。右值引用就是对一个右值进行引用的类型，事实上，由于右值通常不具有名字，我们也只能通过引用的方式找到它的存在。

右值引用和左值引用都是属于引用类型。无论是声明一个左值引用还是右值引用，都必须立即进行初始化。而其原因可以理解为是引用类型本身自己并不拥有所绑定对象的内存，只是该对象的一个别名。左值引用是具名变量值的别名，而右值引用则是不具名（匿名）变量的别名。

左值引用通常也不能绑定到右值，但常量左值引用是个“万能”的引用类型。它可以接受非常量左值、常量左值、右值对其进行初始化。不过常量左值所引用的右值在它的“余生”中只能是只读的。相对地，非常量左值只能接受非常量左值对其进行初始化。

```
int &a = 2;         // 左值引用绑定到右值，编译失败

int b = 2;          // 非常量左值
const int &c = b; // 常量左值引用绑定到非常量左值，编译通过
const int d = 2;  // 常量左值
const int &e = c; // 常量左值引用绑定到常量左值，编译通过
const int &b =2;  // 常量左值引用绑定到右值，编程通过
```

右值值引用通常不能绑定到任何的左值，要想绑定一个左值到右值引用，通常需要std::move()将左值强制转换为右值，例如：

```
int a;
int &&r1 = c;              // 编译失败
int &&r2 = std::move(a);  // 编译通过
```

下表列出了在C++11中各种引用类型可以引用的值的类型。值得注意的是，只要能够绑定右值的引用类型，都能够延长右值的生命期。

表 3-1　C++11 中引用类型及其可以引用的值类型

引用类型	可以引用的值类型				注　记
	非常量左值	常量左值	非常量右值	常量右值	
非常量左值引用	Y	N	N	N	无
常量左值引用	Y	Y	Y	Y	全能类型，可用于拷贝语义
非常量右值引用	N	N	Y	N	用于移动语义、完美转发
常量右值引用	N	N	Y	Y	暂无用途