问题

之前都没接触过左值引用和右值引用的概念.....

解答

在 C++ 中所有的值必属于左值、右值两者之一。

左值：可以取地址的，有名字的，非临时的
右值：不能取地址的，没有名字的，临时的

举个栗子：`int a = b + c`，a 就是左值，其变量名为 a，通过 `&a` 可以取得该变量的地址；而表达式 `b + c` 和函数返回值 `int fun()` 就是右值，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。

可见**临时值，函数返回的值**等都是右值；而**非匿名对象(包括变量)，函数返回的引用，const对象**等都是左值。

从本质上理解，创建和销毁由编译器幕后控制，程序员只能确保在本行代码有效的，就是右值(包括立即数)；而用户创建的，通过作用域规则可知其生存期的，就是左值(包括函数返回的局部变量的引用以及const对象)。

左值引用

所谓的左值引用就是对左值的引用。先看一下传统的左值引用：

```
int a = 10;
int &b = a;  // 定义一个左值引用变量
b = 20;      // 通过左值引用修改引用内存的值
```

*左值引用在汇编层面其实和普通的指针是一样的；*定义引用变量必须初始化，因为引用其实就是一个别名，需要告诉编译器定义的是谁的引用。

下面的这种是无法编译通过的，因为 10 是一个立即数，无法对一个立即数取地址，因为立即数并没有在内存中存储，而是存储在寄存器中。

```
int &c = 10;
```

这个问题可以这么解决：

```
const int& c = 10;
```

使用常引用来引用常量数字 10，因为此刻内存上产生了临时变量保存了10，这个临时变量是可以进行取地址操作的，因此 c 引用的其实是这个临时变量，相当于下面的操作：

```
const int temp = 10;
const int &var = temp;
```

结论：

> 左值引用要求右边的值必须能够取地址，如果无法取地址，可以用**常引用**。但使用常引用后，我们只能通过引用来读取数据，无法去修改数据，因为其被 const 修饰成常量引用了。

右值引用

右值引用是 C++11 新增的特性，右值引用用来绑定到右值，绑定到右值以后，本来会被销毁的右值的生存期会延长到与绑定到它的右值引用的生存期。（有点绕，多读两遍）

定义右值引用的格式如下：

```
类型 && 引用名 = 右值表达式;
int &&c = 10;
```

在汇编层面右值引用做的事情和常引用是相同的，即产生临时量来存储常量。但是，唯一一点的区别是，右值引用可以进行读写操作，而常引用只能进行读操作。

直接看下面这两段话很难理解，建议好好看下参考资料中的代码，写得非常好。

右值引用的存在并不是为了取代左值引用，而是充分利用右值(特别是临时对象)的构造来减少对象构造和析构操作以达到提高效率的目的。

带右值引用参数的拷贝构造和赋值重载函数，又叫**移动构造函数**和**移动赋值函数**，这里的移动指的是把临时量的资源移动给了当前对象，临时对象就不持有资源，为nullptr了，实际上没有进行任何的数据移动，没发生任何的内存开辟和数据拷贝。

注意：

右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，通常需要使用 `std::move()` 函数将左值强制转换为右值，如：

```
int val = 10;
int &&rrval = std::move(val);
```

但是这里需要注意：在调用完`std::move()`之后，不能再使用val，只能使用 rrval，这一点用于基本类型可能没什么直接影响，当应用到类函数的时候，用好`std::move()` 可以减少构造函数数的次数

参考资料

c++ 左值引用与右值引用