c++ 左值引用与右值引用

叶余

一直在模仿，从来不专业

左值引用

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。

```
int &var = 10;
```

上述代码是无法编译通过的，因为10无法进行取地址操作，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，可以通过下述方法解决：

```
const int &var = 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var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;
```

根据上述分析，得出如下结论：

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。

那么C++11 引入了右值引用的概念，使用右值引用能够很好的解决这个问题。

右值引用

C++对于左值和右值没有标准定义，但是有一个被广泛认同的说法：

可以取地址的，有名字的，非临时的就是左值；
不能取地址的，没有名字的，临时的就是右值；

可见立即数，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)。

定义右值引用的格式如下：

```
类型 && 引用名 = 右值表达式;
```

右值引用是C++ 11新增的特性，所以C++ 98的引用为左值引用。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至与绑定到它的右值引用的生存期。

```
int &&var = 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。

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。