**C++中的“引用”详解**

**前言：**在面向对象的程序设计语言中，我们经常听见一些名词，引用，地址，在函数传递参数的时候，我们又经常说值传递，引用传递，最容易让人搞混淆的就是“引用”和“地址”这两个概念了，对于C++和C#来说，引用一词从他们所呈现的表象来看的确很类似，但是本质实际上是不一样的，

***C++：****引用就是一个变量的别名；*

***C#：****引用可以用指针去理解，虽然C#没有指针，我们经常说某个变量所引用的数据，可以理解为某个变量所指向的数据。*

**一、先从C语言的交换两个变量说起**

C语言中：函数传参有传值和传址两种方式  
用swap函数举例：

**1.1 传值方式（创建了临时变量存放实参的值）**  
缺点：不能通过函数形参改变外部实参  
优点：不能改变外部的实参

1. void swap(int left,int right)//此代码不能完成两数的交换
2. {
3. int tmp = left;
4. left = right;
5. right = tmp;
6. }

**2.传址方式（创建了临时变量存放了实参的地址）**

缺点：每次访问实参都要解引用  
优点：可以改变外部实参

1. void swap(int\* left,int\* right)
2. {
3. int tmp = \*left;
4. \*left = \*right;
5. \*right = tmp;
6. }

而C++中就引入了引用的概念，下面详细介绍一下C++中的引用  
上面的代码就可以写成这样  
此时代码中的left和right就是实参的别名，通过交换left和right就能将传过来的实参进行交换。

1. void swap(int& left,int& right)
2. {
3. int tmp = left;
4. left = right;
5. right = tmp;
6. }

**总结：C++通过引用就可以达到C语言的指针作为参数的效果。**

**二、引用的简单概念**

引用不是新定义一个变量，而是给已存在变量取了一个别名，编译器不会为引用变量开辟内存空间，它和它引用的变量共用同一块内存空间。

定义引用类型的格式：

***类型 & 引用变量名(对象名) = 引用实体；***

***注意这里的空格是可选的，即***

* ***&符号与前后均可以有一个空格；如下：int & ra=a;***
* ***&符号与类型挨着，如下：int& ra=a;***
* ***&符号与引用名称挨着，如下：int &ra=a;***

到底怎么理解引用只是一个变量的别名呢？其实在生活中别名很好理解，比如 张三 有一个别名叫做  三儿 。那么这两个名称指的实际上就是同一个人，张三干嘛，三儿就在干嘛，两者是一起的，为了说明，我们看一个例子：

int main()

{

int a = 100;

int & ar = a;

cout << a << endl;

cout << ar << endl;

cout << &a << endl;

cout << &a << endl;

ar = 200; //改变引用变量

cout << a << endl;

cout << ar << endl;

cout << &a << endl;

cout << &a << endl;

getchar();

return 0;

}

/\*

100

100

012FF77C

012FF77C

200

200

012FF77C

012FF77C

\*/

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**2.1  总结引用的特点**

1. *引用变量的类型必须与它的实体类型一致（因为取别名要符合引用实体的身份，如果类型不一致则会报错）*
2. *引用变量使用必须要进行初始化（不然没有实体都不知道给谁取别名）*
3. *一个变量可以有多个引用（就相当于一个变量有好几个别名，这是可以的）*
4. *int a = 10;*
5. *int& ra = a;*
6. *int& rra = ra; ra,raa都是a的别名*
7. *引用一旦引用一个实体，再不能引用其他实体（同一个别名不能引用不同的人，否则就分不清谁是谁了）*
8. *引用不是指针，他就是一个变量，仅仅是一个别名；*

**总而言之：**

引用本身也是一个变量，但是这个变量又仅仅是另外一个变量一个别名，它不占用内存空间，它不是指针哦！不要混淆了，仅仅是一个别名，别名，别名，重要的事情说三遍。

**三、引用的更多使用**

**3.1 常引用**

变量可以使变量和常量，别名本质上也是变量，也可以是变量或者是常量，所以对应起来有四种情况，分别如下：

**（1）变引用——变量**

1. int a = 10; //可读可写
2. int& ra = a; //可读可写

**（2）常引用——变量**

1. int a = 10; //可读可写
2. int const& ra = a; //仅仅可读，不可写
4. ra=20; //编译不通过，ra是常量

**（3）变引用——常量**

1. int const a = 100; //常量
2. int & ar = a; //变量，编译没办法通过，因为本尊都是常量，别名自然不能是变量

**（4）常引用——常量**

1. int const a = 100; //常量
2. int const & ar = a; //常量，自身和别名都是常量，没有问题

**3.2 指针引用**

引用既然就是一个变量，那我同样也可以给指针变量去一个别名啊，参见下面的

int main()

{

int a = 100;

int \*p = &a;

int \* &rp = p;

cout << a << endl;

cout << \*p << endl;

cout << \*rp << endl; //这里为什么要将\*放在前面，因为p的类型是 int \* 作为一个整体哦！！

cout << p << endl;

cout << rp << endl;

getchar();

return 0;

}

/\*

100

100

100

012FF84C

012FF84C

\*/

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

我们发现这里的指针变量p和它的引用（别名）rp是完全一样的。但是由于引用的目的跟指针的目的是类似的，所以一般不需要对指针再起别名了。（参见两数交换的函数）

***总而言之一句话：***

***引用变量就是别名、别名、别名。***

**四、引用与函数的结合使用**

**4.1 引用变量作为函数参数**

把传值和传址的优点结合起来了，写起来会比较方便，**同时对形参修改了会体现到我们的外部实参上（因为形参就是实参的别名**），同时传引用的效率比传值的效率高，传引用写起来也比传址方便。

1. void swap(int& left,int& right)
2. {
3. int tmp = left;
4. left = right;
5. right = tmp;
6. }
7. //函数调用,由于引用仅仅是一个别名，对于形参的操作会影响到实参
8. a=10;，需要特别注意
9. b=20;
10. swap(a,b);

由于引用变量作为函数参数，对形参修改了会体现到我们的外部实参上（因为形参就是实参的别名），这需要特别注意，但是如果我不希望改变外面的实参呢？  
传了引用之后，在函数内部进行操作就会把实参修改怎么办？

此时就采用const引用

1. void testfun(const int& a)
2. {
3. //a = 10;
4. //此时a就不能改，因为a是一个常量的引用，不允许修改
5. }

**总结：**所以我们在构建函数的时候，还是要根据实际的需求，来决定到底是传值、传指针、还是传引用。不能一概而论。

**4.2 引用变量作为函数的返回值**

**（1）不要返回局部变量的引用——一个严重的问题**

先看一个代码：

1. using namespace std;
3. int& test1()
4. {
5. int n = 5;
6. return n;
7. }
9. int main()
10. {
11. int i = test1();
12. cout << i << endl;
14. getchar();
15. return 0;
16. }

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**注意：**

不同的编译器对于返回局部变量的引用有所区别对待：

* *对于gcc和g++， 编译报警告，运行的时候会出现错误*
* *对于msvc：编译报警告，warning C4172: 返回局部变量或临时变量的地址: n，但是运行的时候不会出现错误，而是像正常的运行一样，比如上面的代码结果为5.*

为什么不要返回局部变量的引用呢？

因为当该函数调用结束之后，该函数内部创建的局部变量出了作用域会被销毁，为这个函数开辟的栈帧也会被系统回收，在调用下一个函数之前会对这一部分栈空间里的垃圾数据进行清理，因此你也会失去对这个空间的管控能力。函数调用结束之后，所有的局部变量都销毁了，哪里来的别名这一说法。

**（2）返回全局变量的引用**

1. int c; //定义全局变量
2. int & add(int a, int b)
3. {
4. c = a + b;
5. return c; //这里的返回值就是一个int类型的变量，并不是一个引用类型啊，这是不是和int &不兼容？
6. }

不兼容问题并不会存在，由于引用变量并不会占用内存，它实际上就是c，所以返回引用变量就是c。

当然我想下面这样写，更规范，也是没问题的

1. int c; //定义全局变量
2. int &rc=c; //先给c把别名起好
3. int & add(int a, int b)
4. {
5. c = a + b;
6. return rc; //rc 就是c
7. }

怎么调用呢？

如下就像普通函数调用即可：

1. int main()
2. {
3. int a = 100;
4. int b = 200;
5. int result = add(a, b); //就像普通函数调用即可
6. cout << result << endl;
8. getchar();
9. return 0;
10. }

当然也可以这样做，这样看起来更加规范一些：

1. int main()
2. {
3. int a = 100;
4. int b = 200;
5. int result;
6. int &rresult=result //先给返回值起一个别名
7. rresultadd(a, b);
8. cout << rresult << endl;
10. getchar();
11. return 0;
12. }