转 C++中引用（&）的用法和应用实例

2018年11月05日 23:24:19 liudongdong19 阅读数：998 更多

一、什么是引用

引用，顾名思义是某一个变量或对象的**别名**，对引用的操作与对其所绑定的变量或对象的操作完全等价

```
语法：类型 &引用名=目标变量名；
```

特别注意：

1.&不是求地址运算符，而是起标志作用

2.引用的类型必须和其所绑定的变量的类型相同

```
1 #include<iostream>
2 using namespace std;
3 int main(){
4     double a=10.3;
5     int &b=a; //错误，引用的类型必须和其所绑定的变量的类型相同
6     cout<<b<<endl;
7 }
```

```
[Error] invalid initialization of reference of type 'int&' from expression of type 'double'
```

3.声明引用的同时**必须对其初始化**，否则系统会报错

```
1 #include<iostream>
2 using namespace std;
3 int main(){
4     int &a; //错误！声明引用的同时必须对其初始化
5     return 0;
6 }
```

```
[Error] 'a' declared as reference but not initialized
```

4.引用相当于变量或对象的别名，因此**不能再将已有的引用名作为其他变量或对象的名字或别名**

5.引用不是定义一个新的变量或对象，因此**内存不会为引用开辟新的空间存储这个引用**

```
#include<iostream>
using namespace std;
int main(){
    int value=10;
    int &new_value=value;
    cout<<"value在内存中的地址为："<<&value<<endl;
    cout<<"new_value在内存中的地址为："<<&new_value<<endl;
    return 0;
}
```

```
value在内存中的地址为：0x6ffe44
new_value在内存中的地址为：0x6ffe44
```

6.对数组的引用

语法：类型 **(&引用名)[数组中元素数量]=数组名；**

```
#include<iostream>
using namespace std;
int main(){
    int a[3]={1,2,3};
    int (&b)[3]=a;//对数组的引用
    cout<<&a[0]<<" "<<&b[0]<<endl;
    cout<<&a[1]<<" "<<&b[1]<<endl;
    cout<<&a[2]<<" "<<&b[2]<<endl;
    return 0;
}
```

```
0x6ffe20 0x6ffe20
0x6ffe24 0x6ffe24
0x6ffe28 0x6ffe28
```

7.对指针的引用

语法：类型 ***&引用名=指针名;//可以理解为：（类型*） &引用名=指针名，即将指针的类型当成类型***

```
#include<iostream>
using namespace std;
```

```
3 int main(){
4     int a=10;
5   int *ptr=&a;
6   int *&new_ptr=ptr;
7     cout<<&ptr<<" "<<&new_ptr<<endl;
8     return 0;
9 }
```

```
0x6ffe38 0x6ffe38
```

二、引用的应用

A.引用作为函数的参数

```
 1 #include<iostream>
 2 using namespace std;
 3 void swap(int &a,int &b){//引用作为函数的参数
 4     int temp=a;
 5     a=b;
 6     b=temp;
 7 }
 8 int main(){
 9     int value1=10,value2=20;
10     cout<<"----------------------交换前----------------------------"<<endl;
11     cout<<"value1的值为：10"<<endl;
12     cout<<"value2的值为：20"<<endl;
13     swap(value1,value2);
14     cout<<"----------------------交换后----------------------------"<<endl;
15     cout<<"value1的值为："<<value1<<endl;
16     cout<<"value2的值为："<<value2<<endl;
17     return 0;
18 }
```

```
----------------------交换前----------------------------
value1的值为：10
value2的值为：20
----------------------交换后----------------------------
value1的值为：20
value2的值为：10
```

特别注意：

1.当用引用作为函数的参数时，其效果**和用指针作为函数参数的效果相当。**当调用函数时，函数中的形参就会被当成实参变量或对象的一个别名来使用，也就是说**此时函数中对形参的各种操作实际上是对实参本身进行操作，而非简单的将实参变量或对象的值拷贝给形参。**

2.通常函数调用时，系统采用值传递的方式将实参变量的值传递给函数的形参变量。此时，系统会在内存中开辟空间用来存储形参变量，并将实参变量的值拷贝给形参变量，也就是说形参变量只是实参变量的副本而已；并且如果函数传递的是类的对象，系统还会调用类中的拷贝构造函数来构造形参对象。而**使用引用作为函数的形参时，由于此时形参只是要传递给函数的实参变量或对象的别名而非副本，故系统不会耗费时间来在内存中开辟空间来存储形参。**因此**如果参数传递的数据较大时，建议使用引用作为函数的形参，这样会提高函数的时间效率，并节省内存空间。**

3.使用指针作为函数的形参虽然达到的效果和使用引用一样，但**当调用函数时仍需要为形参指针变量在内存中分配空间**，而引用则不需要这样，故**在C++中推荐使用引用而非指针作为函数的参数**

4.**如果在编程过程中既希望通过让引用作为函数的参数来提高函数的编程效率，又希望保护传递的参数使其在函数中不被改变，则此时应当使用对常量的引用作为函数的参数。**

5.数组的引用作为函数的参数：C++的数组类型是带有长度信息的，**引用传递时如果指明的是数组则必须指定数组的长度**

```
#include<iostream>
using namespace std;
void func(int(&a)[5]){//数组引用作为函数的参数，必须指明数组的长度
//函数体
}
int main(){
    int number[5]={0,1,2,3,4};
    func(number);
    return 0;
}
```

B.常引用

语法：**const** 类型 **&**引用名=目标变量名；

常引用不允许通过该引用对其所绑定的变量或对象进行修改

```
#include<iostream>
using namespace std;
int main(){
    int a=10;
    const int &new_a=a;
    new_a=11;//错误！不允许通过常引用对其所绑定的变量或对象进行修改
    return 0;
}
```

```
[Error] assignment of read-only reference 'new_a'
```

特别注意：

先看下面的例子

```
#include<iostream>
#include<string>
using namespace std;
string func1(){
    string temp="This is func1";
    return temp;
}
void func2(string &str){
    cout<<str<<endl;
}
int main(){
    func2(func1());
    func2("Tomwenxing");
    return 0;
}
```

运行上面的程序编译器会报错

```
[Error] invalid initialization of non-const reference of type 'std::string& {aka std::basic_string<char>&}' from an rvalue of type 'std::string {aka std::basic_string<char>}'
[Error] in passing argument 1 of 'void func2(std::string&)'
[Error] invalid initialization of non-const reference of type 'std::string& {aka std::basic_string<char>&}' from an rvalue of type 'const char*'
[Error] in passing argument 1 of 'void func2(std::string&)'
```

这是由于func1()和“Tomwenxing”都会在系统中产生一个临时对象（string对象）来存储它们，而**在C++中所有的临时对象都是const类型的**，而上面的程序试图将const对象赋值给非const对象，这必然会使程序报错。如果在函数func2的参数前添加const，则程序便可正常运行了

```
#include<iostream>
#include<string>
using namespace std;
string func1(){
    string temp="This is func1";
    return temp;
}
void func2(const string &str){
```

```
 9     cout<<str<<endl;
10 }
11 int main(){
12     func2(func1());
13     func2("Tomwenxing");
14     return 0;
15 }
```

```
This is func1
Tomwenxing
```

C.引用作为函数的返回值

语法：类型 **&**函数名（形参列表）**{** 函数体 **}**

特别注意：

1.引用作为函数的返回值时，**必须在定义函数时在函数名前将&**

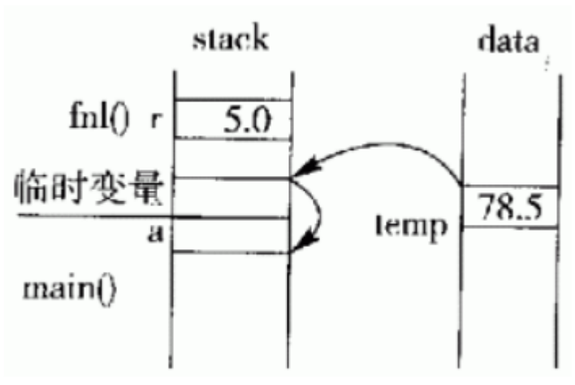
2.用引用作函数的返回值的**最大的好处是在内存中不产生返回值的副本**

```
 1 //代码来源：RUNOOB
 2 #include<iostream>
 3 using namespace std;
 4 float temp;
 5 float fn1(float r){
 6     temp=r*r*3.14;
 7     return temp;
 8 }
 9 float &fn2(float r){ //&说明返回的是temp的引用，换句话说就是返回temp本身
10     temp=r*r*3.14;
11     return temp;
12 }
13 int main(){
14     float a=fn1(5.0); //case 1：返回值
15     //float &b=fn1(5.0); //case 2:用函数的返回值作为引用的初始化值 [Error] invalid initialization of non-const reference of type 'float&' from an rvalue of type 'float'
                          //（有些编译器可以成功编译该语句，但会给出一个warning）
16     float c=fn2(5.0);//case 3：返回引用
17     float &d=fn2(5.0);//case 4：用函数返回的引用作为新引用的初始化值
18     cout<<a<<endl;//78.5
19     //cout<<b<<endl;//78.5
20     cout<<c<<endl;//78.5
21     cout<<d<<endl;//78.5
```

```
22     return 0;
23 }
```

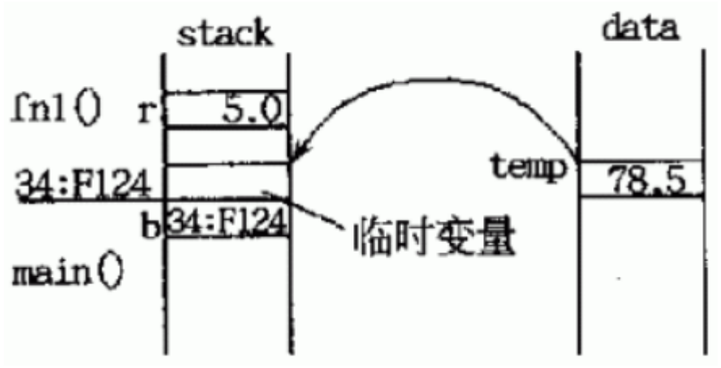
上例中4个case的说明解释：

case 1：用返回值方式调用函数（如下图，图片来源：伯乐在线）：



返回全局变量temp的值时，C++会在内存中创建临时变量并将temp的值拷贝给该临时变量。当返回到主函数main后，赋值语句a=fn1(5.0)会把临时变量的值再拷贝给变量a

case 2：用函数的返回值初始化引用的方式调用函数（如下图，图片来源：伯乐在线）

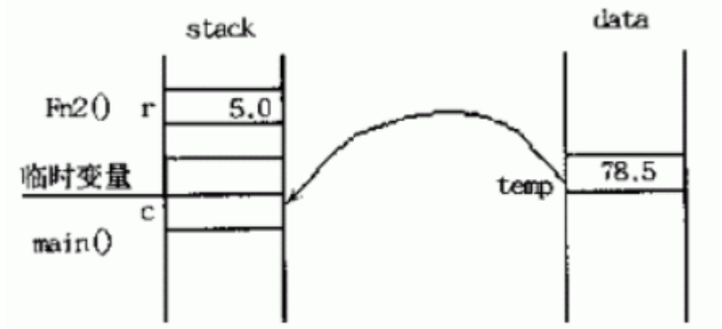


这种情况下，函数fn1()是以值方式返回到，返回时，首先拷贝temp的值给临时变量。返回到主函数后，用临时变量来初始化引用变量b，使得b成为该临时变量到的别名。由于临时变量的作用域短暂（在C++标准中，临时变量或对象的生命周期在一个完整的语句表达式结束后便宣告结束，也就是在语句float &b=fn1(5.0);之后），所以b面临无效的危险，很有可能以后的值是个无法确定的值。

如果真的希望用函数的返回值来初始化一个引用，应当先创建一个变量，将函数的返回值赋给这个变量，然后再用该变量来初始化引用：

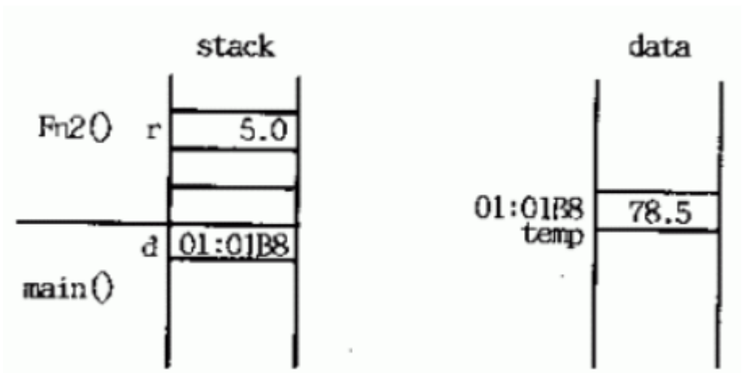
```
1 int x=fn1(5.0);
2 int &b=x;
```

case 3:用返回引用的方式调用函数（如下图，图片来源：伯乐在线）



这种情况下，函数fn2()的返回值不产生副本，而是直接将变量temp返回给主函数，即主函数的赋值语句中的左值是直接从变量temp中拷贝而来（也就是说c只是变量temp的一个拷贝而非别名），这样就避免了临时变量的产生。尤其当变量temp是一个用户自定义的类的对象时，这样还避免了调用类中的拷贝构造函数在内存中创建临时对象的过程，提高了程序的时间和空间的使用效率。

case 4:用函数返回的引用作为新引用的初始化值的方式来调用函数（如下图，图片来源：伯乐在线）



这种情况下，函数fn2()的返回值不产生副本，而是直接将变量temp返回给主函数。在主函数中，一个引用声明d用该返回值初始化，也就是说此时d成为变量temp的别名。由于temp是全局变量，所以在d的有效期内temp始终保持有效，故这种做法是安全的。

3.**不能返回局部变量的引用。**如上面的例子，如果temp是局部变量，那么它会在函数返回后被销毁，此时对temp的引用就会成为“无所指”的引用，程序会进入未知状态。

4.**不能返回函数内部通过new分配的内存的引用。**虽然不存在局部变量的被动销毁问题，但如果被返回的函数的引用只是作为一个临时变量出现，而没有将其赋值给一个实际的变量，那么就可能造成这个引用所指向的空间（有new分配）无法释放的情况（由于没有具体的变量名，故无法用delete手动释放该内存），从而造成内存泄漏。因此应当避免这种情况的发生

5**当返回类成员的引用时，最好是const引用。**这样可以避免在无意的情况下破坏该类的成员。

6.可以用函数返回的引用作为赋值表达式中的左值

```
#include<iostream>
using namespace std;
int value[10];
int error=-1;
int &func(int n){
    if(n>=0&&n<=9)
        return value[n];//返回的引用所绑定的变量一定是全局变量，不能是函数中定义的局部变量
    else
```

```
 9         return error;
10 }
11
12 int main(){
13     func(0)=10;
14     func(4)=12;
15     cout<<value[0]<<endl;
16     cout<<value[4]<<endl;
17     return 0;
18 }
```

```
10
12
```

D.用引用实现多态

在C++中，**引用是除了指针外另一个可以产生多态效果的手段**。也就是说一个基类的引用可以用来绑定其派生类的实例

```
class Father;//基类（父类）
class Son：public Father{.....}//Son是Father的派生类
Son son;//son是类Son的一个实例
Father &ptr=son;//用派生类的对象初始化基类对象的使用
```

特别注意：

ptr只能用来访问派生类对象中从基类继承下来的成员。如果基类（类Father）中定义的有虚函数，那么就可以通过在派生类（类Son）中重写这个虚函数来实现类的多态。

三、总结

1.在引用的使用中，单纯给某个变量去别名是毫无意义的，**引用的目的主要用于在函数参数的传递中，解决大块数据或对象的传递效率和空间不如意的问题**

2.用引用传递函数的参数，能**保证参数在传递的过程中不产生副本**，从而提高传递效率，同时**通过const的使用，还可以保证参数在传递过程中的安全性**

3.引用本身是目标变量或对象的别名，对引用的操作本质上就是对目标变量或对象的操作。因此**能使用引用时尽量使用引用而非指针**

转载：https://www.cnblogs.com/duwenxing/p/7421100.html

想对作者说点什么