# 函数的参数

1. 使用引用，避免不必要拷贝

在使用函数时常常用到形参，形参肯定要用到拷贝，当拷贝的类型为一个很大的数据时将会非常消耗性能，此时，我们可以使用引用来避免拷贝。

举例：当传递一个很长的字符串时，为了减少性能消耗可以使用引用

int func(string &a)

{

return a.size();

}

1. 尽量使用常量引用

在上述例子中我们可以发现，在这个函数中并没有对string进行改变，从安全和性能方面考虑，我们可以把函数参数改成const string &a，这对函数的功能完全没有影响。可以得出这样一个结论：

如果函数无需改变引用形参的值，最好将其声明为常量引用。

同时，把函数不会改变的形参定义成引用是一种比较常见的错误，这么做会导致函数的调用者误以为可以去修改实参的值。此外，使用引用而非常量引用也会极大地限制函数所以能接受的实参类型，比如，我们不能把const对象、字面量或者需要类型转换的对象传递给普通的引用形参。

举例：

int funcA(string &a)

{

return a.size();

}

int funcB(const string &a)

{

return a.size();

}

const string a = "dsadsadsadadsa";

string b = "dasdsadsadsa";

const string a = "dsadsadsadadsa";

string b = "dasdsadsadsa";

//cout << funcA("dsadsadsadadsa") << endl; //报错

cout << funcB("dsadsadsadadsa") << endl; //正确

//cout << funcA(a) << endl; //报错

cout << funcB(a) << endl; //正确

1. 数组形参

数组的两个特殊性质对我们在函数上使用的数组有影响，两个性质分别是：不允许拷贝数组，使用数组时会将其转化成指针。

因为不能拷贝数组，所以我们无法以值传递的方式使用数组参数。因为数组转换成指针，所以当我们为函数传递一个数组时，实际上传递的是指向数组首元素的指针。

举例：

void func(const int\* )

void func(const int[] )

void func(const int[2] )

尽管以上三种形式写法不同，但是实际上是相同的，都是const int\*类型，同时，尽管第三种形式用了const int[2]，但是实际上我们完全可以给这个函数传递两个元素以上的数组。

举例：

int funcB(const int a[2] )

{

return a[0];

}

int a[3] = {1,3,4};

cout << funcB(a) << endl;

因为数组是以指针的形式传递给函数的，所以一开始函数并不知道数组的确切尺寸，因此我们在使用时应该提供一些额外的信息来防止数组越界。我们可以使用标准库中的函数来防止数组越界

举例：

void funcA( const int \*beg, const int \*end)

{

while (beg != end)

{

cout << \*beg++ << endl;

}

}

int a[3] = {1,3,4};

funcA（begin(a), end(a)）;

在今后使用数组做形参时可以借鉴此方法，使用标准库中的begin和end函数，在函数体中将当前地址与最后一个元素地址的下一个地址相对比来确定是否越界。除此方法外，还可以在函数体外就使用begin和end函数求出数组大小（注意，两函数的返回值类型为size\_t），然后给函数传递数组首地址和数组长度。