C++ const修饰函数、函数参数、函数返回值的用法

文章目录

前言

可能有些人会对下面的const的含义，有点迷糊

```
//11 区间排序，按右端点从大到小排序
struct Range
{
    int l, r;
    bool operator< (const Range &W) const
    {
        return r < W.r;
    }
};
```

本文会带你好好理解下const在不同地方的具体含义

1.const修饰函数

> int get() const { }

在类中将成员函数修饰为const表明在该函数体内，不能修改对象的数据成员而且不能调用非const函数。

- c++在函数后加const的意义(一般只出现在类的成员函数定义中)

我们定义的类的成员函数中，常常有一些成员函数不改变类的数据成员，也就是说，这些函数是"只读"函数，而有一些函数要修改类数据成员的值。如果把不改变数据成员的函数都加上const关键字进行标识，显然，可提高程序的可读性。其实，它还能提高程序的可靠性，已定义成const的成员函数，一旦企图修改数据成员的值，则编译器按错误处理。

- 为什么不能调用非const函数？因为非const函数可能修改数据成员，const成员函数是不能修改数据成员的，所以在const成员函数内只能调用const函数。

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

class A{
private:
    int i;
public:
        int get() const{
        // get函数返回i的值，不需要对i进行修改，则可以用const修饰。防止在函数体内对i进行修改。同时提高代码的可读性
        return i;
    }
    void set(int n){ // set函数需要设置i的值，所以不能声明为const
        i = n;
    }

};
```

2.const修饰函数参数

> void fun(const int* i) 或 void h(const Human & a){ }

防止传入的参数代表的内容在函数体内被改变，但**仅对指针和引用有意义**。因为如果是按值传递，传给参数的仅仅是实参的副本，即使在函数体内改变了形参，实参也不会得到影响。如：

```
void fun(const int i){
    i = 10;
}
```

在函数体内是不能改变的值的，但是没有任何实际意义。

const修饰的函数参数是指针时,代表在函数体内不能修改该指针所指的内容，起到保护作用，在字符串复制的函数中保证不修改源字符串的情况下，实现字符串的复制

```
void fun(const char * src, char * des){  // 保护源字符串不被修改，若修改src则编译出错。
    strcpy(des,src);
}
void main(){
    char a[10]="china";
    char b[20];
    fun(a,b);
    cout<<b<<endl;
}
```

而且const指针可以接收非const和const指针，而非const指针只能接收非const指针。

const修饰引用时：如果函数参数为用户自定义的类对象如：

```
void h(A a){
    ...
}
```

传递进来的参数a是实参对象的副本，要调用构造函数来构造这个副本，而且函数结束后要调用析构函数来释放这个副本，在空间和时间上都造成了浪费，所以**函数参数为类对象的情况，推荐用引用。**但按引用传递，造成了安全隐患，通过函数参数的引用可以修改实参的内部数据成员，所以用const来保护实参。

```
void h(const A & a){
    ...
}
```

3.const修饰函数返回值

C++ 返回值return为引用时报错为局部变量local variable

也是用const来修饰返回的指针或引用，保护指针指向的内容或引用的内容不被修改，也常用于运算符重载。归根究底就是使得函数调用表达式不能作为左值
加&后，函数调用表达式a.get()相当于是i的别名

- `int kk = a.get();` kk并不是的别名
- `int &kk = a.get();` kk引用a.get()，a.get()引用i，所以kk是的别名

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

class A {
private:
    int i;
public:
    A(){i=0;}
    int& get(){
        // int& 表示返回的类型是引用 可理解为返回值引用了i
        //作为引用或指针才会出现函数调用表达式作为左值
        return i;
    }

    // const int& get() { // 加const起保护作用
    //     return i;
    //}
};

void main(){
    A a;
    cout<<a.get()<<endl; // 数据成员值为0
    a.get()=1; // 尝试修改a对象的数据成员为1，而且是用函数调用表达式作为左值。
    cout<<a.get()<<endl; // 数据成员真的被改为1了，返回指针的情况也可以修改成员i的值
    // 所以为了安全起见最好在返回值加上const，使得函数调用表达式不能作为左值
}
```

现在再看下面这个例子就会很好理解了

```
// 区间排序，按右端点从大到小排序
struct Range
{
    int l, r;
    // 加const是为了保护数据，同时提高代码可读性
    bool operator< (const Range &W) const  // 前面的const表示不能修改引用的值 后面的const表示不能修改类的数据成员
    {
        // r = 4; wrong
        // W.r += 1; wrong
        return r < W.r;
    }
};
```

`const Range &W` 其实就是用Range类型定义了一个W

这跟你平时用 int W是一个道理，只不过这里的Range是一个类，所以最好加上引用&以及const

参考博客