static那些事 - C++那些事


https://light-city.club/sc/basic_content/static/

光城 Mon Aug, 09 00:16

当与不同类型一起使用时，Static关键字具有不同的含义。我们可以使用static关键字：

静态变量： 函数中的变量，类中的变量

静态类的成员： 类对象和类中的函数

现在让我们详细看一下静态的这些用法：

静态变量

- 函数中的静态变量

当变量声明为static时，空间**将在程序的生命周期内分配**。即使多次调用该函数，静态变量的空间也**只分配一次**，前一次调用中的变量值通过下一次函数调用传递。这对于在C / C ++或需要存储先前函数状态的任何其他应用程序非常有用。

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

void demo()
{
    // static variable
    static int count = 0;
    cout << count << ' ';

    // value is updated and
    // will be carried to next
    // function calls
    count++;
}

int main()
{
    for (int i=0; i<5; i++)
        demo();
    return 0;
}
```

输出：

您可以在上面的程序中看到变量count被声明为static。因此，它的值通过函数调用来传递。每次调用函数时，都不会对变量计数进行初始化。

- 类中的静态变量

由于声明为static的变量只被初始化一次，因为它们在单独的静态存储中分配了空间，因此类中的静态变量**由对象共享。**对于不同的对象，不能有相同静态变量的多个副本。也是因为这个原因，静态变量不能使用构造函数初始化。

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

class Apple
{
public:
    static int i;

    Apple()
    {
        // Do nothing
    };
};

int main()
{
Apple obj1;
Apple obj2;
obj1.i =2;
obj2.i = 3;

// prints value of i
cout << obj1.i<<' '<<obj2.i;
}
```

您可以在上面的程序中看到我们已经尝试为多个对象创建静态变量i的多个副本。但这并没有发生。因此，类中的静态变量应由用户使用类外的类名和范围解析运算符显式初始化，如下所示：

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

class Apple
{
public:
    static int i;

    Apple()
    {
        // Do nothing
    };
};

int Apple::i = 1;

int main()
{
    Apple obj;
    // prints value of i
    cout << obj.i;
}
```

输出：

静态成员

- 类对象为静态

就像变量一样，对象也在声明为static时具有范围，直到程序的生命周期。

考虑以下程序，其中对象是非静态的。

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

class Apple
{
    int i;
    public:
        Apple()
        {
            i = 0;
            cout << 'Inside Constructor\n';
        }
        ~Apple()
        {
            cout << 'Inside Destructor\n';
        }
};

int main()
{
    int x = 0;
    if (x==0)
    {
        Apple obj;
    }
    cout << 'End of main\n';
}
```

输出：

```
Inside Constructor
Inside Destructor
End of main
```

在上面的程序中，对象在if块内声明为非静态。因此，变量的范围仅在if块内。因此，当创建对象时，将调用构造函数，并且在if块的控制权越过析构函数的同时调用，因为对象的范围仅在声明它的if块内。 如果我们将对象声明为静态，现在让我们看看输出的变化。

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

class Apple
{
    int i;
    public:
        Apple()
        {
            i = 0;
            cout << 'Inside Constructor\n';
        }
        ~Apple()
        {
            cout << 'Inside Destructor\n';
        }
};

int main()
{
    int x = 0;
    if (x==0)
    {
        static Apple obj;
    }
    cout << 'End of main\n';
}
```

输出：

```
Inside Constructor
End of main
Inside Destructor
```

您可以清楚地看到输出的变化。现在，在main结束后调用析构函数。这是因为静态对象的范围是贯穿程序的生命周期。

- 类中的静态函数

就像类中的静态数据成员或静态变量一样，静态成员函数也不依赖于类的对象。我们被允许使用对象和'.'来调用静态成员函数。但建议使用类名和范围解析运算符调用静态成员。

允许静态成员函数仅访问静态数据成员或其他静态成员函数，它们无法访问类的非静态数据成员或成员函数。

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

class Apple
{
    public:
        // static member function
        static void printMsg()
        {
            cout<<'Welcome to Apple!';
        }
};

// main function
int main()
{
    // invoking a static member function
    Apple::printMsg();
}
```

输出：