内联函数

范 懿

目录

内联函数的动机

内联函数在 C++ 的类中有重要地位。

在 C++ 中，可以创建并不会被“**调用**”的短函数。

实际上，它们会在被触发的各处展开成代码。

函数被调用——相应变量入栈；调用结束——相应变量出栈。

☞ 内联函数并不会引发上述机制。

因此内联函数比普通函数省掉开销，使程序变得更快。

```
#include <iostream>
using namespace std;
inline int max(int a, int b){
    return a > b ? a : b;
}
int main(){
    cout << max(10, 20);
    cout << " " << max(99, 88);
    return 0;
}
```

对编译器而言，上述程序等价于下面的程序。

```
#include <iostream>
using namespace std;
int main()
{
    cout << (10 > 20 ? 10 : 20);
    cout << " " << (99 > 88 ? 99 : 88);
    return 0;
}
```

内联函数的使用建议

内联函数允许你写非常高效的代码

在类中，我们需要频繁调用界面函数来实现对私有变量的访问

因此，需要把它们设为内联函数

因为代码重复，所以，内联函数会导致程序过大

把小型的、频繁调用的、代码中出现不多的函数，设置为内联。

`inline`只是建议，编译器可能会选择无视这个关键字。

内联函数的注意事项

递归函数往往不会被编译器认可是内联。

如果一个函数不被编译器认可是内联，它将按一般方式执行。

```cpp
#include <iostream>
using namespace std;
class MyClass {
    int a, b;
    public:
    void init(int i, int j);
    void show();
};
// 创建内联函数
inline void MyClass::init(int i, int j){
    a = i;
    b = j;
}
// 再次创建内联函数
inline void MyClass::show(){
    cout << a << " " << b << "\n";
}
int main(){
    MyClass x;
    x.init(10, 20);
    x.show();
    return 0;
}
```

在类中定义内联函数

在类中定义短的函数是完全可行的。

当一个函数在类中被定义时，它被自动设为`inline`（如果可行）。

在类中定义的函数，可以在前面添加`inline`但没有必要。

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

class MyClass {
        int a, b;
    public:
        // 自动设为inline
        void init(int i, int j) { a = i; b = j; }
        void show() { cout << a << " " << b << "\n"; }
};

int main()
{
    MyClass x;
    x.init(10, 20);
    x.show(); // 这个设为内联意义不大，
    return 0; // 因为输出操作相对函数调用而言非常耗时
}
```

因为内联函数普遍比较短，所以上面这种风格是典型的。

```cpp
/*当然也可以写成这一页的风格*/
#include <iostream>
using namespace std;
class MyClass {
        int a, b;
    public:
        // 自动设为inline
        void init(int i, int j)
        {
            a = i;
            b = j;
        }
        void show()
        {
            cout << a << " " << b << "\n";
        }
};
```

专业的 C++ 代码很少把短函数写在类的定义之外。

构造函数和析构函数的情况和其他函数一样。