

C++ 模板

模板是泛型编程的基础，泛型编程即以一种独立于任何特定类型的方式编写代码。

模板是创建泛型类或函数的蓝图或公式。库容器，比如迭代器和算法，都是泛型编程的例子，它们都使用了模板的概念。

每个容器都有一个单一的定义，比如 **向量**，我们可以定义许多不同类型的向量，比如 `vector<int>` 或 `vector<string>`。

您可以使用模板来定义函数和类，接下来让我们一起来看看如何使用。

函数模板

模板函数定义的一般形式如下所示：

```
template <class type> ret-type func-name(parameter list)
{
    // 函数的主体
}
```

在这里，type 是函数所使用的数据类型的占位符名称。这个名称可以在函数定义中使用。

下面是函数模板的实例，返回两个数中的最大值：

实例

```
#include <iostream>
#include <string>
using namespace std;
template <typename T>
inline T const& Max (T const& a, T const& b)
{
    return a < b ? b:a;
}
int main ()
{
    int i = 39;
    int j = 20;
    cout << "Max(i, j): " << Max(i, j) << endl;
    double f1 = 13.5;
    double f2 = 20.7;
    cout << "Max(f1, f2): " << Max(f1, f2) << endl;
    string s1 = "Hello";
    string s2 = "World";
    cout << "Max(s1, s2): " << Max(s1, s2) << endl;
    return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Max(i, j): 39
Max(f1, f2): 20.7
Max(s1, s2): World
```

类模板

正如我们定义函数模板一样，我们也可以定义类模板。泛型类声明的一般形式如下所示：

```
template <class type> class class-name {  
.  
.  
.  
}
```

在这里，**type** 是占位符类型名称，可以在类被实例化的时候进行指定。您可以使用一个逗号分隔的列表来定义多个泛型数据类型。

下面的实例定义了类 `Stack<>`，并实现了泛型方法来对元素进行入栈出栈操作：

实例

```
#include <iostream>  
#include <vector>  
#include <cstdlib>  
#include <string>  
#include <stdexcept>  
using namespace std;  
template <class T>  
class Stack {  
private:  
    vector<T> elems; // 元素  
public:  
    void push(T const&); // 入栈  
    void pop(); // 出栈  
    T top() const; // 返回栈顶元素  
    bool empty() const { // 如果为空则返回真。  
        return elems.empty();  
    }  
};  
template <class T>  
void Stack<T>::push (T const& elem)  
{  
    // 追加传入元素的副本  
    elems.push_back(elem);  
}  
template <class T>  
void Stack<T>::pop ()  
{  
    if (elems.empty()) {  
        throw out_of_range("Stack<>::pop(): empty stack");  
    }  
    // 删除最后一个元素  
    elems.pop_back();  
}  
template <class T>  
T Stack<T>::top () const
```

```
{
if (elems.empty()) {
throw out_of_range("Stack<>::top(): empty stack");
}
// 返回最后一个元素的副本
return elems.back();
}
int main()
{
try {
Stack<int> intStack; // int 类型的栈
Stack<string> stringStack; // string 类型的栈
// 操作 int 类型的栈
intStack.push(7);
cout << intStack.top() << endl;
// 操作 string 类型的栈
stringStack.push("hello");
cout << stringStack.top() << std::endl;
stringStack.pop();
stringStack.pop();
}
catch (exception const& ex) {
cerr << "Exception: " << ex.what() << endl;
return -1;
}
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
7
hello
Exception: Stack<>::pop(): empty stack
```

← C++ 命名空间

C++ 预处理器 →



3 篇笔记

✍ 写笔记