第12讲 类模板

目录 CONTENTS

12.1 类模板的概念

- 类模板（Class Template）是这样一种通用类：在定义类时不指明某些数据成员、成员函数的参数及返回值的数据类型，而是用类型参数取代。

- 类模板也称为参数化的类，它可以生成多个成员和功能相似的类，这些类的区别仅仅是某些数据成员、成员函数的参数或返回值的数据类型不同。

12.2 类模板的定义

- 类模板的定义格式为：

```
template<typename 形参名字1, typename 形参名字2>
class XXX
{

}
```

12.2 类模板的定义

- 类模板成员函数

如果类模板成员函数直接写在类模板定义中，隐式声明为内联函数。

如果类模板成员函数定义在类模板之外，必须以关键字template开始，后边接类模板参数列表（包含非参数类型），格式如下：

```
template<typename 类型参数>
函数返回值类型 类模板名<类型参数>::成员函数名(形参表)
{
    函数体
}
```

12.2 类模板的定义

- 类模板的实例化

  类模板<类型参数> 对象名

类模板并不是类，只有给定类型参数后，才会定义具体的类，用以创建对象。

【例1】 建立求最大值的模板类

```
template <typename T>
class Max     //声明类模板Max
{
    private:
        T item1;       //类型为T,T在该类的对象生成时具体化
        T item2;
public:
    Max(){}
    Max(T t1,T t2){item1=t1;item2=t2;}
    T GetMaxItem(){if(item1>item2) return item1;else return item2;}
};
int main()
{
     Max<int> nmyMax(1,2);
     Max<double> dblmyMax(1.2,1.3);
     cout<<nmyMax.GetMaxItem()<<endl;
     cout<<dblmyMax.GetMaxItem()<<endl;
     return 0;
}
```

【例2】 建立求最大值的模板类

```
template <typename T>
class Max     //声明类模板Max
{
    private:
        T item1;      //类型为T,T在该类的对象生成时具体化
        T item2;
public:
    Max(){}
    Max(T t1,T t2){item1=t1;item2=t2;}
    T GetMaxItem();
};
template<typename T>
T Max<T>::GetMaxItem()
{
    if(item1>item2)
        return item1;
    else
        return item2;
}
```

【例2】 定义堆栈类模板

```
template<typename T>
class Stack
{
    T items[10];
    int top;
public:
    Stack();
    bool Push(const T& item);
    bool Pop(T& item);
    bool isEmpty();
    bool isFull();
};

template<typename T>
Stack<T>::Stack()
{
    top = 0;
}
```

【例2】 定义堆栈类模板

```
template<typename T>
bool Stack<T>::Push(const T& item)
{
    if(!isFull())
    {
        items[top++] = item;
        return true;
    }
    else
        return false;
}

template<typename T>
bool Stack<T>::isFull()
{
    return top == 10;
}
```

【例2】 定义堆栈类模板

```
bool Stack<T>::Pop(T& item)
{
    if(!isEmpty())
    {
        item = items[--top];
        return true;
    }
    else
        return false;
}

template<typename T>
bool Stack<T>::isEmpty()
{
    return top == 0;
}
```

【例2】 定义堆栈类模板

```
int main()
{
    Stack<string> sts;
    string s1 = "hello";
    sts.Push(s1);

    Stack<int> sti;
    sti.Push(5);
    int i;
    sti.Pop(i);
    cout<<i;
    system("pause");
    return 0;
}
```

12.3 自定义大小Stack

- 在类模板的构造函数中传递栈大小。

```
template<typename T>
class Stack
{
    T* items;
    int size;
    int top;
public:
    Stack(int size = 10);
    bool Push(const T& item);
    bool Pop(T& item);
    bool isEmpty();
    bool isFull();
};
```

```
template<typename T>
Stack<T>::Stack(int size)
{
    this->size = size;
    items = new T[this->size];
    top = 0;
}

int main()
{
    Stack<int> s1(5);
    Stack<double> s2(6);
    return 0;
}
```

12.4 非类型参数构建数组类模板

- 定义数组模板

```
template<typename T, int n>
class Array
{
    T items[n];
    int size;
public:
    Array() {size=n;}
    T& operator[](int i);
};
```

```
template<typename T, int n>
T& Array<T,n>::operator[](int i)
{
    return items[i];
}

int main()
{
    Array<int, 5> s3;
    Array<double, 6> s4;
    return 0;
}
```

12.4 类型参数与非类型参数

- 两者的区别？

```
int main()
{
    Stack<int> s1(5);
    Stack<int> s2(6);
    return 0;
}
```

声明了一个类，创建了2个对象

```
int main()
{
    Array<int, 5> s3;
    Array<int, 6> s4;
    return 0;
}
```

声明了2个类，分别创建了1个对象

12.5 递归使用模板

- 模板可以递归使用，以构建更为复杂的结构。

```
Array< Array<int, 5>, 10> aa;
```

12.5 递归使用模板

```
int main()
{
    Array< Array<int, 5>, 10> data;
    Array<int, 10> sum;
    Array<double, 10> ave;
    // 计算
    for(int i=0;i<10;i++)
    {
        sum[i] = 0;
        for(int j=0;j<5;j++)
        {
            data[i][j] = (i+1)*(j+1);
            sum[i] += data[i][j];
        }
        ave[i] = sum[i]/5;
    }
    // 显示
    for(int i=0;i<10;i++)
    {
        for(int j=0;j<5;j++)
        {
            cout<<data[i][j]<<"     ";
        }
        cout<<": sum = "<<sum[i]<< ",
    ave = "<<ave[i]<<endl;
    }
    system("pause");
    return 0;
}
```

12.6 多个类型参数

同函数模板相类似，类模板也可以有多个类型参数。

```
#include <iostream>
using namespace std;
template <typename T1, typename T2>
class CA
{   T1 m_x;
    T2 m_y;
public:
    CA(T1 x, T2 y)
    {     m_x = x;
          m_y = y;          }
    void show()
    { cout<<"m_x="<<m_x<<"  "<<"m_y="<<m_y<<endl; }
};
int main()
{   CA<char, double>  ca1('a', 123.45);
    CA<int, float>  ca2(100, 3.14159f);
    ca1.show();
    ca2.show();
    return 0;
}
```

声明包含两个类型参数的模板

定义类模板CA

通过类模板实例化定义对象ca1，
类型参数T1和T2分别被char和double所取代

程序的运行结果：
m_x=a m_y=123.45
m_x=100 m_y=3.14159

通过类模板实例化定义对象ca2，
类型参数T1和T2分别被int和float所取代

12.7 函数模板作为类模板成员

类模板中的成员函数还可以是一个函数模板。成员函数模板只有在被调用时才会被实例化。

```
template <class T>
class A
{
public:
    template <class T2>
    void Func(T2 t) { cout << t; }  //成员函数模板
};
int main()
{
    A<int> a;
    a.Func('K');  //成员函数模板Func被实例化
    a.Func("hello");
    return 0;
}
```