

函数模板

郭 炜 刘家瑛



北京大学



泛型程序设计

- ▲ **Generic Programming**
- ▲ 算法实现时不指定具体要操作的数据的类型
- ▲ 泛型 — 算法实现一遍 → 适用于多种数据结构
- ▲ 优势: 减少重复代码的编写
- ▲ 大量编写模板, 使用模板的程序设计
 - 函数模板
 - 类模板



函数模板

- 为了交换两个int变量的值, 需要编写如下Swap函数:

```
void Swap(int & x, int & y)
{
    int tmp = x;
    x = y;
    y = tmp;
}
```



函数模板

- 为了交换两个double型变量的值, 还需要编写如下Swap函数:

```
void Swap(double & x, double & y)
{
    double tmp = x;
    x = y;
    y = tmp;
}
```

能否只写一个Swap, 就能交换各种类型的变量?



函数模板

用 函数模板 解决

`template`<class 类型参数1, class 类型参数2, ... >

返回值类型 模板名 (形参表)

{

函数体

}



函数模板

- 交换两个变量值的函数模板

```
template <class T>
```

```
void Swap(T & x, T & y)
```

```
{
```

```
    T tmp = x;
```

```
    x = y;
```

```
    y = tmp;
```

```
}
```



函数模板

```
int main(){  
    int n = 1, m = 2;  
    Swap(n, m);    //编译器自动生成 void Swap(int &, int &)函数  
    double f = 1.2, g = 2.3;  
    Swap(f, g);    //编译器自动生成 void Swap(double &, double &)函数  
    return 0;  
}
```

```
void Swap(int & x, int & y)  
{  
    int tmp = x;  
    x = y;  
    y = tmp;  
}
```

```
void Swap(double & x, double & y)  
{  
    double tmp = x;  
    x = y;  
    y = tmp;  
}
```



函数模板

- 函数模板中可以有不止一个类型参数

```
template<class T1, class T2>
```

```
T2 print(T1 arg1, T2 arg2)
```

```
{
```

```
    cout<< arg1 << " "<< arg2<<endl;
```

```
    return arg2;
```

```
}
```




函数模板

- 求数组最大元素的MaxElement函数模板

```
template <class T>
```

```
T MaxElement(T a[], int size) //size是数组元素个数
```

```
{
```

```
    T tmpMax = a[0];
```

```
    for( int i = 1; i < size; ++i )
```

```
        if( tmpMax < a[i] )
```

```
            tmpMax = a[i];
```

```
    return tmpMax;
```

```
}
```



函数模板

- 函数模板可以重载, 只要它们的形参表不同即可
- 例, 下面两个模板可以同时存在:

```
template<class T1, class T2>  
void print(T1 arg1, T2 arg2)  
{  
    cout<< arg1 << " "<< arg2<<endl;  
}
```

```
template<class T>  
void print(T arg1, T arg2)  
{  
    cout<< arg1 << " "<< arg2<<endl;  
}
```



函数模板

■ C++编译器遵循以下优先顺序:

Step 1: 先找参数完全匹配的普通函数(非由模板实例化而得的函数)

Step 2: 再找参数完全匹配的模板函数

Step 3: 再找实参经过自动类型转换后能够匹配的普通函数

Step 4: 上面的都找不到, 则报错



例: 函数模板调用顺序

```
template <class T>
```

```
T Max(T a, T b){
```

```
    cout << "Template Max 1" <<endl;
```

```
    return 0;
```

```
}
```

```
template <class T, class T2>
```

```
T Max(T a, T2 b){
```

```
    cout << "Template Max 2" <<endl;
```

```
    return 0;
```

```
}
```



```
double Max(double a, double b){  
    cout << "MyMax" << endl;  
    return 0;  
}  
int main()  
{  
    int i=4, j=5;  
    Max(1.2,3.4); //调用Max(double, double)函数  
    Max(i, j);    //调用第一个T Max(T a, T b)模板生成的函数  
    Max(1.2, 3);  //调用第二个T Max(T a, T2 b)模板生成的函数  
    return 0;  
}
```

运行结果:

MyMax

Template Max 1

Template Max 2



赋值兼容原则引起函数模板中类型参数的二义性

```
template<class T>
```

```
T myFunction(T arg1, T arg2)
```

```
{
```

```
    cout<<arg1<<" " <<arg2<<"\n";
```

```
    return arg1;
```

```
}
```

```
...
```

```
myFunction(5, 7);    //ok: replace T with int
```

```
myFunction(5.8, 8.4); //ok: replace T with double
```

```
myFunction(5, 8.4);  //error: replace T with int or double? 二义性
```



可以在函数模板中使用多个类型参数, 可以避免二义性

```
template<class T1, class T2>
```

```
T1 myFunction( T1 arg1, T2 arg2)
```

```
{
```

```
    cout<<arg1<<" " <<arg2<<"\n";
```

```
    return arg1;
```

```
}
```

```
...
```

```
myFunction(5, 7); //ok : replace T1 and T2 with int
```

```
myFunction(5.8, 8.4); //ok : replace T1 and T2 with double
```

```
myFunction(5, 8.4); //ok : replace T1 with int, T2 with double
```