C++ 模板详解

分类 编程技术

模板是C++支持参数化多态的工具，使用模板可以使用户为类或者函数声明一种一般模式，使得类中的某些数据成员或者成员函数的参数、返回值取得任意类型。

模板是一种对类型进行参数化的工具；

通常有两种形式：函数模板和类模板；

函数模板针对仅参数类型不同的函数；

类模板针对仅数据成员和成员函数类型不同的类。

使用模板的目的就是能够让程序员编写与类型无关的代码。比如编写了一个交换两个整型int 类型的swap函数，这个函数就只能实现int型，对double，字符这些类型无法实现，要实现这些类型的交换就要重新编写另一个swap函数。使用模板的目的就是要让这程序的实现与类型无关，比如一个swap模板函数，即可以实现int 型，又可以实现double型的交换。模板可以应用于函数和类。下面分别介绍。

注意： 模板的声明或定义只能在全局、命名空间或类范围内进行。即不能在局部范围，函数内进行，比如不能在main函数中声明或定义一个模板。

一、函数模板通式

1、函数模板的格式：

```
template <class 形参名, class 形参名, ......> 返回类型 函数名(参数列表)
{
    函数体
}
```

其中template和class是关见字，class可以用typename 关见字代替，在这里typename 和class没区别，<>括号中的参数叫模板形参，模板形参和函数形参很相像，模板形参不能为空，一但声明了模板函数就可以用模板函数的形参名声明类中的成员变量和成员函数，即可以在该函数中使用为最类型的地方都可以使用模板形参名。模板形参需要调用该模板函数时提供的模板实参来初始化模板形参，一旦编译器确定了实际的模板实参类型就称给实例化了函数模板的一个实例，比如swap的模板函数形式为：

```
template <class T> void swap(T& a, T& b){}.
```

当调用这样的模板函数时类型T就会被被调用时的类型所代替，比如**swap(a,b)**其中a和b是int 型，这时模板函数swap中的形参T就会被int 所代替，模板函数就变为**swap(int &a, int &b)**，而当**swap(c,d)**其中c和d是double类型时，模板函数会被替换为**swap(double &a, double &b)**，这样就实现了函数的实现与类型无关的代码。

2、注意：对于函数模板而言不存在 **h(int,int)** 这样的调用，**不能在函数调用的参数中指定模板形参的类型，对函数模板的调用应使用实参推演来进行**，即只能进行 **h(2,3)** 这样的调用，或者**int a, b; h(a,b)**。

函数模板的示例演示将在下文中涉及！

二、类模板通式

1、类模板的格式为：

```
template<class  形参名, class 形参名, ...>  class 类名
{ ... };
```

类模板和函数模板都是以template开始后接模板形参列表组成，模板形参不能为空，一但声明了类模板就可以用类模板的形参名声明类中的成员变量和成员函数，即可以在类中使用为最类型的地方都可以使用模板形参名来声明。比如

```
template<class T> class A{public: T a; T b; T fn(T c, T &d);};
```

在类A中声明了两个类型为T的成员变量a和b，还声明了一个返回类型为T带两个参数类型为T的函数fn。

2、类模板对象的创建：比如一个模板类A，则使用类模板创建对象的方法为**A<int> m;**在类A后面跟上一个<>实括号并在里面填上相应的类型，这样的话类A中凡是用到模板形参的地方都会被int 所代替，当类模板有两个模板形参时创建对象的方法为**A<int, double> m;**类型之间用逗号隔开。

//在类外部定义类的成员时**template**后面的模板形参应与要定义的类的模板形参一致

```
template<class T1,int a>     voidCi<T1,a>::g(){cout<<"class Ci g()"<<typeid(T1).name()<<endl;}
template<class T1,int &a>    voidCip<T1,a>::g(){cout<<"class Cip g()"<<typeid(T1).name()<<endl;}
```

//在类外部定义类的成员时，**template**后的模板形参应与要定义的类的模板形参一致

```
template<class T1,A<int> *m> voidCc<T1,m>::g(){cout<<"class Cc g()"<<typeid(T1).name()<<endl;}
template<class T1,double* a> voidCd<T1,a>::g(){cout<<"class Cd g()"<<typeid(T1).name()<<endl;}
```

//带有默认类型形参的模板类，在类的外部定义成员的方法，

//在类外部定义类的成员时**template**的形参表中默认值应省略

```
template<class T1,class T2>  voidC<T1,T2>::g(){cout<<"class D g()"<<endl;}
//template<class T1,class T2=int> void C<T1,T2>::g(){cout<<"class D k()"<<endl;} //错误，在类模板外部定义带有默认类型的形
参时，在**template**的形参表中默认值应省略。
//定义一实化与实现。
int a=2;  double&d=2.2; double*pe=&d;
A<int> mw; A<int> *pec=&mw; ( mw;
```

//main函数开始

```
int main()
{ // template<class T>void h(){} //错误，模板的声明或定义只能在全局，命名空间或类范围内进行。即不能在局部范围，函数内进行。
//A<2> m; //错误，对类模板不存在实参推演问题，类模板必须在实后引中明确指出其类型。
//类模板调用实例
A<int> ma; //输出"class A goucao int"创建int型的类模板A的对象ma。
B<int,int> mb; mb.g(); //输出"class B g() int int"创建类模板B的对象mb，并把类型形参T1和T2设计为int
//在类外部调用的调用
//调用非类型模板形参的实参必须是一个常量表达式，即他必须能在编译时计算出结果。任何局部对象，局部变量，局部对象的地址，局部
变量的地址都不是一个常量表达式，都不能用作非类型模板形参的实参。全局指针类型，全局变量，全局对象也不是一个常量表达式，不能用作非类型模板形参的实参。
//用作非类型模板形参的实参。
//全局变量的地址或引用，全局对象的地址或引用const类型变量是常量表达式，可以用作非类型模板形参的实参。
//调用类型int类型非类型形参的方法为C为Ci，声明形式为template<class T1,int a> class Ci
Ci<int,3>//正确，数值8是一个int型常量，输出"class Ci g() int"
const int a2=3; Ci<int,a2> mc11; mc11.g(); //正确，因为a2在这里是const型的常量，输出"class Ci g() int"
//Ci<int,a> mc1; //错误，int型变量a是局部变量，不是一个常量表达式。
//Ci<int,e> mc1; //错误，全局int型变量e也不是一个常量表达式。
//调用int&型非类型形参的方法类名为Cip，声明形式为template<class T1,int &a>class Cip
Cip<int,e> mcip;  //正确，对全局变量的引用或地址是常量表达式。
//Cip<int,a> mc1p1; //错误，局部变量的引用或地址不是常量表达式。
//调用double*类型的非类形形参类名为Cd，声明形式为template<class T3,double *a>class Cd
Cd<int,&ed> mcd;  //正确，全局变量的引用或地址是常量表达式。
//Cd<int,pe> mcd1;//正确，全局变量指针不是常量表达式。
//double dd=3.3; //错误，局部变量的地址不是常量表达式，不能用作非类型形参的实参
//Cd<int,&a> mcd;  //错误，非类型形参虽允许一些转换，但这个转换不成立。

//调用模板类型形参对象A<int> *的方法类名为Cc，声名形式为template<class T1,A<int>* m> class Cc
Cc<int,&mw> mcc; mcc.g();  //正确，全局对象的地址或者引用是常量表达式
//Cc<int,&ma> mcc;  //错误，局部变量的地址或引用不是常量表达式。
//Cc<int,pec> mcc2;  //错误，全局对象的指针不是常量表达式。

//调用非类型形参E&对象的引用的方法类名为Ce。声明形式为template<class T1,E &m> class Ce
E me1; //Cc<int,me1> mcc1;  //错误，局部对象不是常量表达式
Ce<int,mw> mce;  //正确，全局对象的指针或引用是常量表达式。
//非类型形参的转换示例，类名为C1
//非类型形参允许从数组到指针，从函数到指针的转换，const修饰符的转换，提升转换，整值转换，常规转换。
const short s=3; C1<int,s> mc14 //正确，虽然short型和int不完全匹配，但这里可以将short型转换为int型

*原文地址：*

*https://www.cnblogs.com/gw811/archive/2012/10/25/2738939.html*

*http://www.cnblogs.com/gw811/archive/2012/10/25/2736224.html*
```