# **C++模板**

　　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内置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他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，即可以在类中使用内置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 hy(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**hy** 。

　　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**<>** 尖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。

　　3、对于类模板，模板形参的类型必须在类名后的尖括号中明确指定。比如**A<2> m;** 用这种方法把模板形参设置为**int** 是错误的（**编译错误：error C2079: 'a' uses undefined class 'A<int>'** ），**类模板形参不存在实参推演的问题。** 也就是说不能把整型值**2** 推演为**int** 型传递给模板形参。要把类模板形参调置为**int** 型必须这样指定**A<int> m** 。

　　4、在类模板外部定义成员函数的方法为：

**template** **<模板形参列表> 函数返回类型 类名<模板形参名>::函数名(参数列表){函数体}，**

比如有两个模板形参**T1** ，**T2** 的类**A** 中含有一个**void h()** 函数，则定义该函数的语法为：

**template<class T1,class T2> void A<T1,T2>::h(){}。**

注意：当在类外面定义类的成员时**template** 后面的模板形参应与要定义的类的模板形参一致。

　　5、再次提醒注意：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。

## **三、模板的形参**

**有三种类型的模板形参** **：类型形参，非类型形参和模板形参。**

### **1、类型形参**

　　　　1.1 、类型模板形参：**类型形参由关见字class或typename后接说明符构成** ，如**template<class T> void h(T a){}** ;其中**T** 就是一个类型形参，类型形参的名字由用户自已确定。**模板形参表示的是一个未知的类型** 。模板类型形参可作为类型说明符用在模板中的任何地方，与内置类型说明符或类类型说明符的使用方式完全相同，即可以用于指定返回类型，变量声明等。

**作者原版：** **1.2、** 不能为同一个模板类型形参指定两种不同的类型，比如**template<class T>void h(T a, T b){}，** 语句调用**h(2, 3.2)**将出错，因为该语句给同一模板形参**T** 指定了两种类型，第一个实参**2** 把模板形参T指定为**int** ，而第二个实参**3.2** 把模板形参指定为**double** ，两种类型的形参不一致，会出错。**（针对函数模板）**

　　　　作者原版：1.2针对函数模板是正确的，但是忽略了类模板。下面将对类模板的情况进行补充。

**本人添加1.2补充版（针对于类模板）、** 当我们声明类对象为：**A<int> a** ，比如**template<class T>T g(T a, T b){}** ，语句调用**a.g(2, 3.2)** 在编译时不会出错，但会有警告，因为在声明类对象的时候已经将**T** 转换为**int** 类型，而第二个实参**3.2** 把模板形参指定为**double** ，在运行时，会对**3.2** 进行强制类型转换为**3** 。当我们声明类的对象为：**A<double> a** ,此时就不会有上述的警告，因为从**int** 到**double** 是自动类型转换。

## **四、类模板的默认模板类型形参**

**1、可以为类模板的类型形参提供默认值，但不能为函数模板的类型形参提供默认值。函数模板和类模板都可以为模板的非类型形参提供默认值。**

　　2、类模板的类型形参默认值形式为：**template<class T1, class T2=int> class A{};**为第二个模板类型形参**T2**提供**int**型的默认值。

　　3、类模板类型形参默认值和函数的默认参数一样，如果有多个类型形参则从第一个形参设定了默认值之后的所有模板形参都要设定默认值，比如**template<class T1=int, class T2>class A{};**就是错误的，因为**T1**给出了默认值，而**T2**没有设定。

　　4、在类模板的外部定义类中的成员时**template** 后的形参表应省略默认的形参类型。比如**template<class  T1, class T2=int> class A{public: void h();}**; 定义方法为**template<class T1,class T2> void A<T1,T2>::h(){}**。

**定义类模板类型形参：**