随笔-123 文章-0 评论-158

C++ 模板详解（一）

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

Hadoop(8)
HBase(1)
IBM FileNet ECM(6)
IBM ILOG(5)
Java(27)
JavaScript(3)
Jboss(1)
Linux(3)
Mina(2)
Oracle(9)
Servlet(1)
Sqoop
Struts2(3)
Zookeeper
错误集锦(1)
设计模式(2)
数据结构(2)
算法(3)
系统架构(1)
线程安全(3)
音视频(1)
银行业务(1)

随笔档案(123)
2016年5月 (1)
2015年7月 (11)
2015年6月 (5)
2015年4月 (1)
2015年3月 (2)
2015年2月 (1)
2015年1月 (3)
2014年12月 (3)
2014年11月 (15)
2014年10月 (6)
2014年7月 (1)
2014年5月 (2)
2014年4月 (13)
2014年3月 (12)
2014年2月 (4)
2012年10月 (27)
2012年9月 (16)

最新评论
1. Re:java.lang.Object.clone()解读
为什么String没有实现cloneable接口，我的理解是：因为String是不可变类，这就意味着很多对String本身的改变其实都是创建了一个新的String。而实现复制，无非就是想得到浅复制或者.....
--杰-维斯布鲁克
2. Re:【linux】提醒"libc.so.6: version `GLIBC_2.14' not found"系统的glibc版本太低
这样有问题的，我是在centos6.5上操作的，升级glibc后系统会崩。先是语系无

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己确定。**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。

演示示例 1：

TemplateDemo.h

```
#ifndef TEMPLATE_DEMO_HXX
#define TEMPLATE_DEMO_HXX

template<class T> class A{
    public:
        T g(T a,T b);
        A();
};

#endif
```

TemplateDemo.cpp

```
#include<iostream.h>
#include "TemplateDemo.h"

template<class T> A<T>::A(){}

template<class T> T A<T>::g(T a,T b){
    return a+b;
}

void main(){
    A<int> a;
    cout<<a.g(2,3.2)<<endl;
}
```

编译结果：

```
1 --------------------Configuration: TemplateDemo - Win32 Debug--------------
2 Compiling...
3 TemplateDemo.cpp
4 G:\C++\CDaima\TemplateDemo\TemplateDemo.cpp(12) : warning C4244: 'argument' : conversion from 'const double' to 'int', possible loss of data
5
6 TemplateDemo.obj - 0 error(s), 1 warning(s)
```

运行结果： **5**

我们从上面的测试示例中可以看出，并非作者原作中的那么严密！此处仅是本人跟人测试结果！请大家本着实事求是的态度，自行验证！

2、非类型形参

2.1 、非类型模板形参：**模板的非类型形参也就是内置类型形参**，如**template<class T, int a> class B{};**其中**int a**就是非类型的模板形参。

2.2、 非类型形参在模板定义的内部是常量值，也就是说非类型形参在模板的内部是常量。

2.3、 非类型模板的形参只能是整型，指针和引用，像double，String, String **这样的类型是不允许的。但是**double &，double ***，对象的引用或指针是正确的。

2.4、 **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。

2.5 、注意：**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。

2.6、 **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。**

2.7 、**sizeof**表达式的结果是一个常量表达式，也能用作非类型模板形参的实参。

2.8 、当模板的形参是整型时调用该模板时的实参必须是整型的，且在编译期间是常量，比如**template <class T, int a> class A{};**如果有**int b，这时A<int, b> m;**将出错，因为**b**不是常量，如果**const int b，这时A<int, b> m;**就是正确的，因为这时**b**是常量。

2.9 、非类型形参一般不应用于函数模板中，比如有函数模板**template<class T, int a> void h(T b){}**，若使用**h(2)**调用会出现无法为非类型形参a推演出参数的错误，对这种模板函数可以用显示模板实参来解决，如用h<int, 3>(2)这样就把非类型形参a设置为整数3。显示模板实参在后面介绍。

2.10、 非类型模板形参的形参和实参间所允许的转换

1、允许从数组到指针，从函数到指针的转换。如：template <int *a> class A{}; int b[1]; A<b> m;即数组到指针的转换

2、const修饰符的转换。如：template<const int *a> class A{}; int b; A<&b> m; 即从int *到const int *的转换。

3、提升转换。如：template<int a> class A{}; const short b=2; A<b> m; 即从short到int 的提升转换

4、整值转换。如：template<unsigned int a> class A{}; A<3> m; 即从int 到unsigned int的转换。

5、常规转换。

非类型形参演示示例1：

由用户自己亲自指定栈的大小，并实现栈的相关操作。

TemplateDemo.h

```
#ifndef TEMPLATE_DEMO_HXX
#define TEMPLATE_DEMO_HXX

template<class T,int MAXSIZE> class Stack{//MAXSIZE由用户创建对象时自行设置
    private:
        T elems[MAXSIZE];    // 包含元素的数组
        int numElems;    // 元素的当前总个数
    public:
        Stack();    //构造函数
        void push(T const&);    //压入元素
        void pop();        //弹出元素
        T top() const;    //返回栈顶元素
        bool empty() const{     // 返回栈是否为空
            return numElems == 0;
        }
        bool full() const{    // 返回栈是否已满
            return numElems == MAXSIZE;
        }
};

template <class T,int MAXSIZE>
Stack<T,MAXSIZE>::Stack():numElems(0){     // 初始时栈不含元素
    // 不做任何事情
}

template <class T,int MAXSIZE>
void Stack<T, MAXSIZE>::push(T const& elem){
    if(numElems == MAXSIZE){
        throw std::out_of_range("Stack<>::push(): stack is full");
    }
    elems[numElems] = elem;    // 附加元素
    ++numElems;                // 增加元素的个数
}

template<class T,int MAXSIZE>
void Stack<T,MAXSIZE>::pop(){
    if (numElems <= 0) {
        throw std::out_of_range("Stack<>::pop(): empty stack");
    }
    --numElems;                // 减少元素的个数
}

template <class T,int MAXSIZE>
T Stack<T,MAXSIZE>::top()const{
    if (numElems <= 0) {
        throw std::out_of_range("Stack<>::top(): empty stack");
    }
    return elems[numElems-1];  // 返回最后一个元素
}
```

```
50
51 #endif
```

TemplateDemo.cpp

```
 1 #include<iostream.h>
 2 #include <iostream>
 3 #include <string>
 4 #include <cstdlib>
 5 #include "TemplateDemo.h"
 6
 7 int main(){
 8     try {
 9         Stack<int,20>  int20Stack;  // 可以存储20个int元素的栈
10         Stack<int,40>  int40Stack;  // 可以存储40个int元素的栈
11         Stack<std::string,40> stringStack; // 可存储40个string元素的
栈
12
13         // 使用可存储20个int元素的栈
14         int20Stack.push(7);
15         std::cout << int20Stack.top() << std::endl;    //7
16         int20Stack.pop();
17
18         // 使用可存储40个string的栈
19         stringStack.push("hello");
20         std::cout << stringStack.top() << std::endl;    //hello
21         stringStack.pop();
22         stringStack.pop();    //Exception: Stack<>::pop<>: empty
stack
23         return 0;
24     }
25     catch (std::exception const& ex) {
26         std::cerr << "Exception: " << ex.what() << std::endl;
27         return EXIT_FAILURE;  // 退出程序且有ERROR标记
28     }
29 }
```

运行结果：

```
"G:\C++\CDaima\TemplateDemo\Debug\TemplateDemo.exe"
7
hello
Exception: Stack<>::pop(): empty stack
Press any key to continue
```

非类型形参演示示例2：

TemplateDemo01.h

```
 1 #ifndef TEMPLATE_DEMO_01
 2 #define TEMPLATE_DEMO_01
 3
 4 template<typename T> class CompareDemo{
 5     public:
 6         int compare(const T&, const T&);
```

```
 7 };
 8
 9 template<typename T>
10 int CompareDemo<T>::compare(const T& a,const T& b){
11     if((a-b)>0)
12         return 1;
13     else if((a-b)<0)
14         return -1;
15     else
16         return 0;
17 }
18
19 #endif
```

TemplateDemo01.cpp

```
1 #include<iostream.h>
2 #include "TemplateDemo01.h"
3
4 void main(){
5     CompareDemo<int> cd;
6     cout<<cd.compare(2,3)<<endl;
7 }
```

运行结果：　　-1

```
1 #include<iostream.h>
2 #include "TemplateDemo01.h"
3
4 void main(){
5     CompareDemo<double> cd;
6     cout<<cd.compare(3.2,3.1)<<endl;
7 }
```

运行结果：　　1

TemplateDemo01.h 改动如下：

```
 1 #ifndef TEMPLATE_DEMO_01
 2 #define TEMPLATE_DEMO_01
 3
 4 template<typename T> class CompareDemo{
 5     public:
 6         int compare(T&, T&);
 7 };
 8
 9 template<typename T>
10 int CompareDemo<T>::compare(T& a,T& b){
11     if((a-b)>0)
12         return 1;
13     else if((a-b)<0)
14         return -1;
15     else
16         return 0;
```

```
17 }
18
19 #endif
```

TempalteDemo01.cpp

```
1 #include<iostream.h>
2 #include "TemplateDemo01.h"
3
4 void main(){
5     CompareDemo<int> cd;
6     int a=2,b=3;
7     cout<<cd.compare(a,b)<<endl;
8 }
```

非类型形参演示示例3：

TemplateDemo02.cpp

```
 1 #include<iostream.h>
 2
 3 template<typename T>
 4 const T& max(const T& a,const T& b){
 5     return a>b ? a:b;
 6 }
 7
 8 void main(){
 9     cout<<max(2.1,2.2)<<endl;//模板实参被隐式推演成double
10     cout<<max<double>(2.1,2.2)<<endl;//显示指定模板参数。
11     cout<<max<int>(2.1,2.2)<<endl;//显示指定的模板参数，会将函数函数直接转换为int。
12 }
```

运行结果：

```
"G:\C++\CDaima\TemplateDemo02\Debug\TemplateDemo02.exe"
2.2
2.2
2
Press any key to continue
```

cout<<max<int>(2.1,2.2)<<endl;//显示指定的模板参数，会将函数函数直接转换为int。此语句会出现警告：

```
1 --------------------Configuration: TemplateDemo02 - Win32 Debug--------------------
2 Compiling...
3 TemplateDemo02.cpp
4 G:\C++\CDaima\TemplateDemo02\TemplateDemo02.cpp(11) :
    warning C4244: 'argument' : conversion from 'const double' to 'const int', possible loss of data
```

```
5 G:\C++\CDaima\TemplateDemo02\TemplateDemo02.cpp(11) :
   warning C4244: 'argument' : conversion from 'const double' to
'const int', possible loss of data
6
7 TemplateDemo02.obj - 0 error(s), 2 warning(s)
```

分类: C++

标签: C++模板详解

zero516cn
关注 - 4
粉丝 - 191
+加关注

33 0

« 上一篇：C++引用详解
» 下一篇：C++ 模板详解（二）

posted @ 2012-10-25 15:30 zero516cn 阅读(123333) 评论(25) 编辑 收藏

评论列表

#1楼 2012-10-25 18:31 huals

楼主看的是不是c++模板这本书，不错啊，学习了

支持(0) 反对(0)

#2楼 2012-10-25 19:21 薰衣草的旋律

文章写的很好，有点深度，但是感兴趣的人好像不多哦。

支持(1) 反对(0)

#3楼 2012-10-25 20:59 hellmonky

楼主看的应该是c++模板编程那本书吧？写的不错，期待更新

支持(0) 反对(0)

#4楼[楼主] 2012-10-25 21:04 zero516cn

@ huals
作者：黄邦勇帅
你应该看的也是这个吧，但是上面实例太少，有很多些值得商榷的地方。就自己整理一下，也当学习了！

支持(0) 反对(0)

#5楼[楼主] 2012-10-25 21:05 zero516cn

@ 薰衣草的旋律
我也是刚开始学C++，以前是学的Java，开拓一下自己的视野，就当自己学习了。

支持(0) 反对(0)

#6楼[楼主] 2012-10-25 21:39 zero516cn

@ hellmonky
好的。很快的。。。

支持(0) 反对(0)

#7楼 2012-10-26 14:16 三更_雨

关于 1.2 、不能为同一个模板类型形参指定两种不同的类型。我在我的vs2010上试了下，貌似 函数模版 和 类模版的效果是不一样的。类模版是你这样的效果，如果你的类型指定的是int，也就是A<int> a;这种形式的话，就会提示将double的转为int会丢失数据之类的，如果用A<double> a的话，我这里编译完成之后，警告都没有了。 但是如果用函数模版的话，就会报error的错误。错误是：模板 参数"Type"不明确，可能是"double"或"int"。楼主可以试试。。。

支持(1) 反对(0)

#8楼[楼主] 2012-10-26 15:05 zero516cn

@ 三更_雨
好的。。。。

支持(0) 反对(0)

#9楼[楼主] 2012-10-26 15:30 zero516cn

@ 三更_雨
针对你的问题我谈一下我的理解：

对于“如果用A<double> a的话，我这里编译完成之后，警告都没有了”：
原因是从int类型到double类型，是自动类型转换，而不是强制类型转换；这也是为什么从double类型到int类型会有数据丢失的原因。

对于“但是如果用函数模版的话，就会报error的错误”：
造成这个问题的原因是：函数模板类型的确定是通过实参推演，所以，对于同样的T类型必须要一致。

谢谢您的提醒纠正了我的一个错误，我会在文章中完善的。谢谢！！！

支持(1) 反对(0)

#10楼 2012-10-26 15:33 三更_雨

不知道你想用 非类型形参演示示例2说明什么问题。对于你那个错误，是因为在main中调用的时候，将常量2和3，赋值给了非常量的引用型变量。如果你在main中定义成 int a = 2, b = 3; cout<<cd.compare(a,b)<<endl; 这种形式，就不会存在错误。至少在我这里没错。

文章写的很好，最近正好想看看模版的知识，就看到你这篇了。THX楼主啦。。

支持(0) 反对(0)

#11楼 2012-10-26 15:35 三更_雨

@ 516Forever713
恩啊。。我想也是这个问题。。应该是你说的这样。
函数模板类型的确定是通过实参推演，所以，对于同样的T类型必须要一致。

支持(1) 反对(0)

#12楼[楼主] 2012-10-26 16:08 zero516cn

@ 三更_雨
我吧，学 C++今天也才是第8天，前几天快速过了一遍C++基础（以前是做Java的，在学C++感觉容易一点），这几天就一直在研究这个模板，我也是从网上下载了一篇文章，但是上面实例太少了，就自己加些实例，让网友吐吐槽，大家的眼镜总是雪亮的吗，要不是你质疑，我在一些问题上也是处于死胡同，一个人考虑问题总是有欠缺的，特别是像我这种刚开始学的。

支持(0) 反对(1)

#13楼 2012-10-26 17:05 三更_雨

@ 516Forever713
原来是这样啊，学到第八天就开始看模版了，很厉害的说啊。。

支持(0) 反对(0)

#14楼[楼主] 2012-10-26 17:07 zero516cn

@ 三更_雨
我QQ:1215972660
有兴趣加一下！

支持(0) 反对(0)

#15楼 2012-10-26 17:07 三更_雨

@ 516Forever713
花这么多时间，还着色写这篇文章实在是大赞啊。。。

支持(0) 反对(0)

#16楼 2014-03-26 17:51 网名还没想好

写得不错

支持(0) 反对(0)

#17楼 2015-02-07 16:28 elson.zeng

learn, thank you!

支持(0) 反对(0)

#18楼 2015-04-01 13:51 Dingo妹

xue xi le ,zan ,xie xie

支持(0) 反对(0)

#19楼 2015-05-16 14:39 impluse

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(){}。

请教博主，这个类模板外定义的成员函数为什么在类名后要加<T1,T2>，看不出来也没看明白T1,T2的作用呢

支持(0) 反对(0)

#20楼 2015-06-21 16:56 puppylpg

为了赞作者一笔特意注册了博客园的账号！写的详实！

支持(0) 反对(0)

#21楼 2015-06-23 23:13 kiss_xiaojie

谢谢楼主，学习了！！

支持(0) 反对(0)

#22楼 2015-07-09 21:37 flyingdust21

学习了，顶，谢谢

支持(0) 反对(0)

#23楼 2016-04-30 16:01 JRSmith

@ impluse
我也有这个疑问，请问您现在明白了吗？能否指点一下，谢谢。

支持(0) 反对(0)

#24楼 2016-04-30 16:40 JRSmith

您好博主，问一下“改动之后的TemplateDemo01.h”这部分实验是为了说明什么问题？改动之后的代码去掉了那几个形参的const，我想问一下，有没有const这个对比实验是想证明什么问题？谢谢了。

支持(0) 反对(0)

#25楼 2016-10-13 15:57 弯路回正

您好 我想请教下在函数模板作函数参数是否可以

比如我有一个比较函数

```
template<typename T> bool cmp(T a, T b)
{
return a>b;
}

double a=5, b=10;
sort(a, b, cmp);
```

这样是否可行呢

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新IT新闻:

· 黑客揭秘Uber漏洞 可让你终生免费打车
· AWS的S3故障回顾和思考
· 硅谷富豪的隐秘一面：广积粮建地堡 去新西兰买房
· 癌症检测公司GRAIL获9亿美元B轮融资，腾讯跟投
· IBM发布首个认知影像产品 预警医生看不到的险情
» 更多新闻...

最新知识库文章:

· 垃圾回收原来是这么回事
· 「代码家」的学习过程和学习经验分享
· 写给未来的程序媛
· 高质量的工程代码为什么难写
· 循序渐进地代码重构
» 更多知识库文章...