

简单，可复制

点点滴滴，尽在文中

:: 首页 :: 博文 :: 闪存 :: 新随笔 :: 联系 :: 订阅  :: 管理 ::  431 随笔 :: 0 文章 :: 539 评论 :: 0 引用

公告

史上最好的免费svn空间

昵称: ggjucheng

园龄: 6年9个月

粉丝: 1644

关注: 6

+加关注

博客地图

[c/c++笔记](#)

本人学习c/c++的一些笔记

[db笔记](#)

[mysql nosql](#)

[hadoop笔记](#)

本人工作中hadoop的心得

[internet笔记](#)

[互联网学习笔记](#)

[java笔记](#)

[java平台笔记](#)

[Linux/Unix笔记](#)

本人学习linux/unix的笔记

[TCP/IP笔记](#)

本人学习TCP/IP的心得和笔记

[web开发](#)

[html css js php etc.....](#)

[技术花絮](#)

C++ Template

引言

模板 (Template) 指C++程序设计语言中采用类型作为参数的程序设计, 支持通用程序设计。C++ 的标准库提供许多有用的函数大多结合了模板的观念, 如STL以及IO Stream。

函数模板

在c++入门中, 很多人会接触swap(int&, int&)这样的函数类似代码如下:

```
void swap(int&a , int& b) {  
    int temp = a;  
    a = b;  
    b = temp;  
}
```

但是如果是支持long,string,自定义class的swap函数,代码和上述代码差不多, 只是类型不同, 这个时候就是我们定义swap的函数模板,就可以复用不同类型的swap函数代码, 函数模板的声明形式如下:

```
template <class identifier> function_declaration;  
template <typename identifier> function_declaration;
```

swap函数模板的声明和定义代码如下:

```
//method.h  
template<typename T> void swap(T& t1, T& t2);  
  
#include "method.cpp"
```

非技术的技术

[其他笔记本](#)

比较零碎的技术文章归类

[学习指南](#)

IT技术学习路线,IT经典书籍学习和下载

友情链接

[IT短篇笑话](#)

百忙中, 可以看看it短篇笑话, 笑一笑, 放松下!

[相当好用的免费svn空间](#)

国内挺不错的svn免费空间, 很适合小团队使用

积分与排名

积分 - 1046749

排名 - 73

最新评论

1. [Re:Eclipse插件安装方式及使用说明](#)

很清楚, 谢谢, 受教

--=-u

2. [Re:linux awk命令详解](#)

讲得非常清楚, 谢谢楼主分享

--青儿哥哥

3. [Re:Linux vmstat命令实战详解](#)

可以看一下redhat的文档, Procs r: The number of processes waiting for run time. b: The number of process.....

--jcuan

4. [Re:JAVA正则表达式: Pattern类与Matcher类详解\(转\)](#)

Pattern

p=Pattern.compile("\\d+");
Matcher

m3=m.matcher("2223bb");m.

matches(); //匹配整个字符串

m.start();

--knn120

5. [Re:Linux网络流量实时监控](#)

ifstat iftop命令详解

666



//method.cpp

```
template<typename T> void swap(T& t1, T& t2) {  
    T tmpT;  
    tmpT = t1;  
    t1 = t2;  
    t2 = tmpT;  
}
```



上述是模板的声明和定义了, 那模板如何实例化呢, 模板的实例化是编译器做的事情, 与程序员无关, 那么上述模板如何使用呢, 代码如下:



```
//main.cpp  
#include <stdio.h>  
#include "method.h"  
int main() {  
    //模板方法  
    int num1 = 1, num2 = 2;  
    swap<int>(num1, num2);  
    printf("num1:%d, num2:%d\n", num1, num2);  
    return 0;  
}
```



这里使用swap函数, 必须包含swap的定义, 否则编译会出错, 这个和一般的函数使用不一样。所以必须在method.h文件的最后一行加入#include "method.cpp"。

类模板

--starRTC免费IM直播

阅读排行榜

1. linux awk命令详解(1198415)
2. Linux tcpdump命令详解(864292)
3. Linux netstat命令详解(589173)
4. linux grep命令详解(426653)
5. linux sed命令详解(377753)

评论排行榜

1. linux awk命令详解(40)
2. Linux tcpdump命令详解(27)
3. C++指针详解(24)
4. linux sed命令详解(23)
5. Linux netstat命令详解(21)

推荐排行榜

1. linux awk命令详解(112)
2. Linux tcpdump命令详解(65)
3. Linux netstat命令详解(62)
4. Linux GCC常用命令(53)
5. Linux入门——适合初学者(52)

考虑我们写一个简单的栈的类，这个栈可以支持int类型，long类型，string类型等等，不利用类模板，我们就要写三个以上的stack类，其中代码基本一样，通过类模板，我们可以定义一个简单的栈模板，再根据需要实例化为int栈，long栈，string栈。



```
//stack.h
template <class T> class Stack {
public:
    Stack();
    ~Stack();
    void push(T t);
    T pop();
    bool isEmpty();
private:
    T *m_pT;
    int m_maxSize;
    int m_size;
};
```

```
#include "stack.cpp"
```



```
//stack.cpp
template <class T> Stack<T>::Stack() {
    m_maxSize = 100;
    m_size = 0;
    m_pT = new T[m_maxSize];
}
template <class T> Stack<T>::~~Stack() {
    delete [] m_pT ;
}

template <class T> void Stack<T>::push(T t) {
    m_size++;
    m_pT[m_size - 1] = t;
```

```
}  
template <class T> T Stack<T>::pop() {  
    T t = m_pT[m_size - 1];  
    m_size--;  
    return t;  
}  
template <class T> bool Stack<T>::isEmpty() {  
    return m_size == 0;  
}  
}
```



上述定义了一个类模板--栈，这个栈很简单，只是为了说明类模板如何使用而已，最多只能支持100个元素入栈，使用示例如下：



```
//main.cpp  
#include <stdio.h>  
#include "stack.h"  
int main() {  
    Stack<int> intStack;  
    intStack.push(1);  
    intStack.push(2);  
    intStack.push(3);  
  
    while (!intStack.isEmpty()) {  
        printf("num:%d\n", intStack.pop());  
    }  
    return 0;  
}
```



模板参数

模板可以有类型参数，也可以有常规的类型参数int，也可以有默认模板参数，例如

```
template<class T, T def_val> class Stack{...}
```

上述类模板的栈有一个限制，就是最多只能支持100个元素，我们可以使用模板参数配置这个栈的最大元素数,如果不配置，就设置默认最大值为100，代码如下：



```
//stack.h
template <class T,int maxsize = 100> class Stack {
    public:
        Stack();
        ~Stack();
        void push(T t);
        T pop();
        bool isEmpty();
    private:
        T *m_pT;
        int m_maxSize;
        int m_size;
};

#include "stack.cpp"
```



```
//stack.cpp
template <class T,int maxsize> Stack<T, maxsize>::Stack() {
    m_maxSize = maxsize;
    m_size = 0;
    m_pT = new T[m_maxSize];
}

template <class T,int maxsize> Stack<T, maxsize>::~~Stack() {
    delete [] m_pT ;
}

template <class T,int maxsize> void Stack<T, maxsize>::push(T t) {
    m_size++;
    m_pT[m_size - 1] = t;
```

```
}  
template <class T,int maxsize> T Stack<T, maxsize>::pop() {  
    T t = m_pT[m_size - 1];  
    m_size--;  
    return t;  
}  
template <class T,int maxsize> bool Stack<T, maxsize>::isEmpty() {  
    return m_size == 0;  
}  
}
```



使用示例如下:



```
//main.cpp  
#include <stdio.h>  
#include "stack.h"  
int main() {  
    int maxsize = 1024;  
    Stack<int,1024> intStack;  
    for (int i = 0; i < maxsize; i++) {  
        intStack.push(i);  
    }  
    while (!intStack.isEmpty()) {  
        printf("num:%d\n", intStack.pop());  
    }  
    return 0;  
}
```



模板专门化

当我们要定义模板的不同实现,我们可以使用模板的专门化。例如我们定义的stack类模板,如果是char*类型的栈,我们希望可以复制char的所有数据到stack类中,因为只是保存char指针,char指针指向的内存有可能会失效,stack弹出的堆栈元素char指针,指向的内存


可能已经无效了。还有我们定义的swap函数模板，在vector或者list等容器类型时，如果容器保存的对象很大，会占用大量内存，性能下降，因为要产生一个临时的大对象保存a，这些都需要模板的专门化才能解决。

函数模板专门化

假设我们swap函数要处理一个情况，我们有两个很多元素的vector<int>,在使用原来的swap函数，执行tmpT = t1要拷贝t1的全部元素，占用大量内存，造成性能下降，于是我们系统通过vector.swap函数解决这个问题,代码如下：


```
//method.h
template<class T> void swap(T& t1, T& t2);

#include "method.cpp"
```




```
#include <vector>
using namespace std;
template<class T> void swap(T& t1, T& t2) {
    T tmpT;
    tmpT = t1;
    t1 = t2;
    t2 = tmpT;
}

template<> void swap(std::vector<int>& t1, std::vector<int>& t2) {
    t1.swap(t2);
}
```



template<>前缀表示这是一个专门化,描述时不用模板参数，使用示例如下：



```
//main.cpp
#include <stdio.h>
#include <vector>
#include <string>
#include "method.h"
```

```
int main() {
    using namespace std;
    //模板方法
    string str1 = "1", str2 = "2";
    swap(str1, str2);
    printf("str1:%s, str2:%s\n", str1.c_str(), str2.c_str());

    vector<int> v1, v2;
    v1.push_back(1);
    v2.push_back(2);
    swap(v1, v2);
    for (int i = 0; i < v1.size(); i++) {
        printf("v1[%d]:%d\n", i, v1[i]);
    }
    for (int i = 0; i < v2.size(); i++) {
        printf("v2[%d]:%d\n", i, v2[i]);
    }
    return 0;
}
```



vector<int>的swap代码还是比较局限，如果要用模板专门化解决所有vector的swap，该如何做呢，只需要把下面代码

```
template<> void swap(std::vector<int>& t1, std::vector<int>& t2) {
    t1.swap(t2);
}
```

改为

```
template<class V> void swap(std::vector<V>& t1, std::vector<V>& t2) {
    t1.swap(t2);
}
```

就可以了，其他代码不变。


类模板专门化

请看下面compare代码：





```
//compare.h
template <class T>
class compare
{
public:
bool equal(T t1, T t2)
{
    return t1 == t2;
}
};
```



```

#include <iostream>
#include "compare.h"
int main()
{
    using namespace std;
    char str1[] = "Hello";
    char str2[] = "Hello";
    compare<int> c1;
    compare<char *> c2;
    cout << c1.equal(1, 1) << endl;           //比较两个int类型的参数
    cout << c2.equal(str1, str2) << endl;     //比较两个char *类型的参数
    return 0;
}
```



在比较两个整数，compare的equal方法是正确的，但是compare的模板参数是char*时，这个模板就不能工作了，于是修改如下：

```

//compare.h
#include <string.h>
template <class T>
class compare
```

```
{
    public:
    bool equal(T t1, T t2)
    {
        return t1 == t2;
    }
};

template<>class compare<char *>
{
    public:
    bool equal(char* t1, char* t2)
    {
        return strcmp(t1, t2) == 0;
    }
};
```



main.cpp文件不变，此代码可以正常工作。

模板类型转换

还记得我们自定义的Stack模板吗，在我们的程序中，假设我们定义了Shape和Circle类，代码如下：

```
//shape.h
class Shape {

};

class Circle : public Shape {

};
```

然后我们可以这么使用：



```
//main.cpp
#include <stdio.h>
#include "stack.h"
```

```
#include "shape.h"

int main() {
    Stack<Circle*> pcircleStack;
    Stack<Shape*> pshapeStack;
    pcircleStack.push(new Circle);
    pshapeStack = pcircleStack;
    return 0;
}
```



这里是无法编译的，因为Stack<Shape*>不是Stack<Circle*>的父类，然而我们却希望代码可以这么工作，那我们就要定义转换运算符了，Stack代码如下：

```
//stack.h

template <class T> class Stack {
public:
    Stack();
    ~Stack();
    void push(T t);
    T pop();
    bool isEmpty();
    template<class T2> operator Stack<T2>();
private:
    T *m_pT;
    int m_maxSize;
    int m_size;
};

#include "stack.cpp"
```



```
template <class T> Stack<T>::Stack() {
    m_maxSize = 100;
```



```
m_size = 0;
m_pT = new T[m_maxSize];
}
template <class T> Stack<T>::~~Stack() {
    delete [] m_pT ;
}

template <class T> void Stack<T>::push(T t) {
    m_size++;
    m_pT[m_size - 1] = t;
}

template <class T> T Stack<T>::pop() {
    T t = m_pT[m_size - 1];
    m_size--;
    return t;
}

template <class T> bool Stack<T>::isEmpty() {
    return m_size == 0;
}

template <class T> template <class T2> Stack<T>::operator Stack<T2>() {
    Stack<T2> StackT2;
    for (int i = 0; i < m_size; i++) {
        StackT2.push((T2)m_pT[m_size - 1]);
    }
    return StackT2;
}
```



```
//main.cpp
#include <stdio.h>
#include "stack.h"
#include "shape.h"
int main() {
```

```
Stack<Circle*> pcircleStack;
Stack<Shape*> pshapeStack;
pcircleStack.push(new Circle);
pshapeStack = pcircleStack;
return 0;
}
```



这样, Stack<Circle>或者Stack<Circle*>就可以自动转换为Stack<Shape>或者Stack<Shape*>,如果转换的类型是Stack<int>到Stack<Shape>, 编译器会报错。

其他

一个类没有模板参数, 但是成员函数有模板参数, 是可行的, 代码如下:



```
class Util {
public:
    template <class T> bool equal(T t1, T t2) {
        return t1 == t2;
    }
};

int main() {
    Util util;
    int a = 1, b = 2;
    util.equal<int>(1, 2);
    return 0;
}
```



甚至可以把Util的equal声明为static,代码如下:



```
class Util {
public:
```

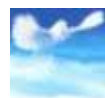
```
        template <class T> static bool equal(T t1, T t2) {  
            return t1 == t2;  
        }  
};  
  
int main() {  
    int a = 1, b = 2;  
    Util::equal<int>(1, 2);  
    return 0;  
}
```

分类: [C/C++](#)标签: [C/C++_基础](#)

好文要顶

关注我

收藏该文



ggjucheng

关注 - 6

粉丝 - 1644

[+加关注](#)[« 上一篇: C++异常处理](#)[» 下一篇: 学会用core dump调试程序错误\(转\)](#)posted on 2011-12-18 21:29 [ggjucheng](#) 阅读(60732) 评论(5) [编辑](#) [收藏](#)

6

0

评论

#1楼 2012-01-05 12:52 Stephen_Liu

比较基础, 可以考虑写写typelist和traits技巧。

支持(0) 反对(0)

[回复](#) [引用](#)

#2楼[楼主] 2012-01-05 13:01 ggjucheng

@ Stephen_Liu

刚开始学模板而已，就总结下模板的简单用法，希望通过这个文章让一点都不懂C++模板的人能一眼学会，然后直接复制编程，这个是我的文章初衷。你的建议我后面会考虑写的，谢谢。

支持(1) 反对(0)

[回复](#) [引用](#)**#3楼 2012-12-27 17:12 reflyer**

你的文章写得很好，很容易看懂，谢谢了

支持(0) 反对(0)

[回复](#) [引用](#)**#4楼 2014-10-17 14:25 峰子_仰望阳光**

楼主写的太好了！学习啦！

支持(0) 反对(0)

[回复](#) [引用](#)**#5楼 2017-06-12 11:31 虎嗅蔷薇S**

template<> 模板专门化不带参数 编译根本无法通过

支持(0) 反对(0)

[回复](#) [引用](#)[刷新评论](#) [刷新页面](#) [返回顶部](#)

发表评论

昵称:

评论内容:

