C++ 高级

2020年11月3日 19:51

模板

函数模板(类型参数化)

作用：建立一个通用函数，其函数返回值类型和形参类型可以不具体制定，用一个虚拟的类型来代表。

语法：

Template<typename T>

函数声明或定义

解释：

template——声明创建模板

type那么——表明其后面符号是一种数据类型，可以用class代替

T——通用的数据类型，名称可以替换，通常为大写字母

例：函数模板实现a，b交换（a，b不限数据类型）

```
Template<typename T>
Void mySwap(T &a, T &b)
{
    T temp = a;
    a = b;
    b = temp;
}
```

1.使用自动类型推导调用模板	2.使用显示指定类型调用
Int a = 10; Int b = 20; mySwap(a,b);	Int a=10; int b =20; mySwap<int>(a,b);

注意：

自动类型推导，必须推导出一致的数据类型T才可以使用

模板必须要确定出T的数据类型，才可以使用

普通函数和函数模板区别

普通函数调用时可以发生自动类型转换(隐式类型转换)

函数模板调用时，如果利用自动类型推导，不会发生隐式类型转换

如果利用显示指定类型的方式，可以发生隐式类型转换

总结：建议使用显式函数模板

普通函数和函数模板调用规则

1.如果函数模板和普通函数都可以调用，优先调用普通函数

2.可以通过空模板参数列表强制调用函数模板

3.函数模板可以发生重载

4.如果函数模板可以产生更好的匹配，优先调用函数模板

总结：既然提供了函数模板，最好不要提供普通函数，否则容易出现二义性

利用具体化的模板可以解决自定义类型的通用化，语法如下

```
template<> compare(Person &a,Person &b)
{
    ...
}
```

类模板

建立一个通用类，类中的成员数据类型可以不具体制定，用一个虚拟的类型来代表

语法

Template<typename T>

类

例：

```
Template<class Nametype,class Agetype>
Class Person
{
Public:
    Nametype m_name;
    Agetype m_age;
    Person(Nametype name,Agetype age)
    {
        This->name = name;
        This->age = age;
    }
}
main函数下调用：Person<string, int> p1("syq",999);
```

类模板和函数模板区别

1.类模板没有自动类型推导的使用方式

2.类模板在模板参数列表中可以有默认参数

类模板中成员函数和普通类中成员函数创建时机是有区别的

普通类中的成员函数一开始就可以创建

类模板中的成员函数在调用时才创建

类模板对象做函数参数

1、指定传入类型(常用)

```
Void printPerson1(Person<string,int>&p)
{
}
```

2、参数模板化

```
Template<class T1,class T2>
Void printPerson2(Person<T1,T2>&p)
{
}
```

3、整个类模板化

```
Template<class T>
Void printPerson3(T&p)
{
}
```

类模板+继承需注意

当子类继承的父类是一个类模板时，子类在声明的时候，要指定父类中T的类型

如果不指定，编译器无法给子类分配内存

如果想灵活指定父类中T的类型，子类也需变为类模板

类模板成员函数类外实现

构造函数

```
Template<class T1,class T2>
Person<T1,T2>::Person(T1 name,T2 age){}
```

成员函数

```
Template<class T1, class T2>
Void Person<T1 , T2>::showPerson(){}
```

类模板分文件编写

问题：类模板中成员函数创建时机是在调用，导致分文件编写时链接不到

解决：1.直接包含.cpp源文件

2.将声明和实现写到同一个文件中，并更改后缀名为.hpp，hpp是约定的名称，并不是强制

类模板与友元

1.全局函数 类内实现

```
Template<class T1,class T2>
```

```
Class Person
{
        Friend void printPerson(Person<T1,T2> p)
        {
                …
        }
        …
}
```

2.全局函数 类外实现

```
Template<class T1,class T2>
Class Person;

Template<class T1,class T2>
Void printPerson(Person<T1,T2> p)
{
        …
}

Template<class T1,class T2>
Class Person
{
        Friend void printPerson< >(Person<T1,T2> p);
        …
}
```

STL

STL基本概念

STL(Standard Template Libray,标准模板库)

STL从广义上分为：容器(container)，算法(algorithm)，迭代器(iterator)

容器和算法之间通过迭代器进行无缝连接

STL几乎所有的代码都用了模板类或者模板函数

Vector

类似于数组，也成为单端数组，不同之处在于数组是静态空间，而vector可以动态扩展。

包含头文件include <vector>

声明vector<int> v;

放数据v.push_back(10);

每一个容器都有自己的迭代器，迭代器是用来遍历容器中的元素的

v.begin()返回迭代器，这个迭代器指向容器中第一个数据

v.end()返回迭代器，这个迭代器指向容器元素的最后一个元素的下一个位置

遍历方法：

```
Vector<int>::iterartor pBegin = v.begin();
Vector<int>::iterartor pEnd = v.end();
While (pBegin != pEnd)
{
        Cout << *pBegin <<endl;
        pBegin++;
}
```

```
For (it = v.begin(); it != v.end(); it++)
{
  cout << *it << end;;
}
```

```
//需要包含头文件algorithm
Void Myprint(int val){
        Cout < < val < < endl;
}
For_each(v.begin(),v.end(),Myprint);
```

assign对vector赋值：v3.assign(v1.begin(),v1.end());

V4.assign(10,100);//赋值10个100

Empty()判断容器是否为空，capacity()返回容器的容量，size()返回容器中元素的个数

Resize(int num)重新指定容器的长度为num，如果容器变长，则以默认值0填充新位置；如果容器变短，删除超出的元素

Resize(int num,elem)如果容器变长，以elem填充，其余同上

Push_back(ele);尾部插入元素ele

Pop_back()；删除最后一个元素

Insert(const_iterator pos,ele);迭代器指向位置pos插入元素ele

Insert(const_iterator pos,int count, ele);迭代器指向位置pos插入count个元素ele

Erase(const_iterator pos);删除迭代器指向的元素
Erase(const_iterator start,const_iterator end);删除迭代器从start到end之间的元素
Clear()删除容器中所有元素
At(int index)或者Operator[] 返回index所值的数据
Front()返回容器中第一个数据元素
Back()返回容器中最后一个数据元素
Swap(vec)容器互换，巧用交换收缩内存空间： vector<int>(v).swap(v);
Reserve(int len)；容器预留len个元素长度，预留位置不初始化，元素不能访问

String，字符串
string本质是一个类
assign各种重载用法，append各种重载用法
find，rfind，replace用法，compare用法，[]和at
insert插入，erase删除，substr截取子串

Deque
双端数组，可对头端进行插入删除操作
deque和vector区别
vector对于头部的插入和删除效率低，数据量越大，效率越低
deque相对而言对头部的插入删除速度会比vector快
vector访问元素时的速度会比deque快，这和两者内部实现有关
assign赋值操作
Empty()判断容器会否为空
Size()返回容器中元素个数
resize重新指定容器大小
Push_back(ele)尾插 push_front尾删
Pop_back()尾删 pop_front()头部删除
insert插入，clear()清空，erase()删除
At(int index)或者Operator[] 返回index所值的数据
Front()返回容器中第一个数据元素
Back()返回容器中最后一个数据元素
sort（beg，end）排序//要包含algorithm头文件

Stack
先进后出。
栈不允许有遍历行为。可以返回元素个数size，可以判断是否为空empty
入栈push(ele)，出栈pop()，top()返回栈顶元素

Queue
先进先出。
队列容器允许一段新增数据，一端允许删除数据。
入队push(ele)，出队pop()
Back()返回最后一个元素,front()返回第一个元素
Empty()判断队列是否为空，size()返回队列的大小

List
将数据进行链式存储。
链表是一种物理存储单元上非连续的存储结构，数据元素的逻辑顺序是通过链表中的指针链接实现的。
链表的组成：链表是由一系列结点组成

结点的组成：一个存储数据元素的数据域，另一个是存储下一个结点地址的指针域

优点：可以对任意位置进行快速插入或删除元素

缺点：容器遍历速度没有数组快，占用空间比数组大

STL提供的链表是双向的

构造函数：

List(T) lst;//list采用模板类实现，对象的默认构造形式

List(beg,end)//构造函数将beg到end区间中的元素拷贝给本身

List(n,elem);//构造函数将n个elem拷贝给本身

List(const list &lst);//拷贝构造函数

assign赋值，swap交换，size大小，empty判断是否为空，resize重新制定容器长度

Push_back(ele)尾插 push_front尾删

Pop_back()尾删 pop_front()头部删除

insert插入，clear()清空，erase()删除

Remove(elem);//删除容器中所有与elem匹配的元素

Front()返回容器中第一个数据元素

Back()返回容器中最后一个数据元素

Reverse()反转列表

Lst.Sort(comparePerson);链表排序

sort自定义数据类型要制定排序规则例子：

```
Bool comparePerson(Person &p1,Person &p2)
{
    If (p1.m_Age == p2.m_Age) {
        Return p1.m_height < p2.m_height;
    } else
    Return p1.m_Age < p2.m_Age;
}
```

注意：所有不支持随机访问迭代器的容器，不可以用标准算法。

不支持随机访问迭代器的容器，内部会提供对应的一些算法。

Set/multiset容器，集合

所以元素都会在插入时自动被排序

本质：set/multiset属于关联式容器，底层结构是用二叉树实现的

set和multiset区别：set不允许容器中有重复的元素，multiset允许容器中有重复的元素

插入数据只有insert函数

size返回容器数目，empty判断容器是否为空，swap(st)交换两个容器

clear删除所有元素，

Erase(pos);删除pos迭代器所指的元素，返回下一个元素的迭代器

Erase(beg,end);删除区间beg到end所有元素，返回下一个元素的迭代器

Erase(elem);删除容器中值为elem的元素

Find(key) 查找key是否存在，若存在，返回该键的元素的迭代器；如果不存在，返回set.end();

Count(key) 统计key的元素个数

pair对组创建

成对出现的数据，利用对组可以返回两个数据

两种创建方式：pair<type,type> p (val1,val2);

Pair<type,type> o =make_pair(val1,val2);

set容器排序，默认从小到大，改变排序规则

内置数据类型，从大到小(仿函数重载（）运算符)

```
Class MyCompare
{
    Public:
    Bool operator()(int v1,int v2)
    {
```

```
            Return v1>v2;
        }

    }
    Set<int,MyCompare>s2;
    自定义数据类型，必须指定排序规则
    Class Person
    {
        Public
        string m_Name;
        Int m_age;
    }

    Class MyCompare
    {
        Public:
        Bool operator()(const Person &p1,const Person &v2)
        {
            Return p1.m_Age>p2,m_Age;
        }

    }
    Set<Person,MyCompare>s2;
```

Map/multimap容器

map中所有元素都是pair

pair中第一个元素为key（键值），起到索引作用，第二个元素为value（实值）

所有元素都会根据元素的键值key自动排序

本质：属于关联式容器，底层使用二叉树实现

优点：可以根据key值款速找到value值

map和multimap区别：

map不允许容器中有重复key值元素

multimap允许容器中有重复key值元素

构造map容器map<type，type> mp

size（）返回大小 empty（）判断容器是否为空 swap（st）交换

插入

1.m.insert(pair<int,int>(1,10))，需要使用对组pair

2.m.insert(make_pair(1,10))

3.m.insert(map<int,int>::value_type(3.30))

4.m[4] = 40;//不推荐

删除

m.erase（ele）；iterator删除

m.erase（key）；键删除

m.erase（beg，end）；区间删除

find（key） 查找key是否存在，如果存在返回迭代器；如果不存在，返回map.end（）

count（key） 统计key的元素个数

排序 map<int,int,MyCompare> m；MyCompare为仿函数重写（）同上

STL-函数对象

概念：重载函数调用操作符的类，其对象称为函数对象

函数对象使用重载的()时，行为类似函数调用，也叫仿函数

本质：函数对象（仿函数）是一个类，不是一个函数

特点：

函数对象在使用时，可以想普通函数一样调用，有参数，有返回值

函数对象可以有自己的状态
函数对象可以作为参数传递

谓词
概念:返回bool类型的仿函数称为谓词
如果operator（）接受一个参数，那么叫做一元谓词
如果operator（）接受两个参数，那么叫做二元谓词

内建函数对象头文件 #include<functional>
算数仿函数
Negate 一元仿函数 取反仿函数
例子
Negate<int> n;
Cout << n(50) <<endl;
输出：-50
Plus 二元仿函数 加法仿函数
Minus 减法仿函数
multiplies乘法仿函数
Divides 除法仿函数
Modulus 取模仿函数
例子
Plus<int> p;
Cout << p(10,30) << endl;
输出：40;
关系仿函数
Equal_to 等于
Not_equal_to 不等于
Greater 大于，最常用
Greater_equal 大于等于
Less 小于
Less_equal 小于等于
逻辑仿函数
Logical_and 逻辑与
Logical_or 逻辑或
Logical_not 逻辑非

STL-常用算法
算法主要用到的头文件
<algorithm>是所有STL头文件中最大的一个，范围涉及到比较、交换、查找、遍历、复制、修改等
<numeric> 体积很小，只包括几个在序列上面进行简单数学运算的模板函数
<functional> 定义了一些模板类，用以声明函数对象

常用算法
For_each：for_each(beg(),end(),print01); 其中print01是普通函数
for_each(beg(),end(),print02())；其中print02是成员函数
Transform 搬运容器到另一个容器中，目标容器需要提前开辟空间
Transform(beg1,end1,beg2,_func)
Beg1原容器开始迭代器，end1原容器结束迭代器，beg2目标容器开始迭代器，func函数或者函数对象
Find 查找元素，find(beg,end,value)，自定义数据类型需要重载==
Find_if 按条件查找元素，find_if(beg,end,_pred)，pres为函数或者谓词（返回bool类型的仿函数）
Adjacent_find 查找相邻重复元素，返回相邻元素的第一个位置的迭代器 Adjacent_find（beg，end）

Binary_search 二分查找法 binary_serch(beg,endmvalue)，注意在无序序列中不可用
Count 统计元素个数，count（beg，end，value）
Count_if 按条件统计元素个数 count（beg，end，_pred）

C++异常处理

异常是程序在执行期间产生的问题。C++ 异常是指在程序运行时发生的特殊情况，比如尝试除以零的操作。

异常提供了一种转移程序控制权的方式。C++ 异常处理涉及到三个关键字：try、catch、throw。

- throw: 当问题出现时，程序会抛出一个异常。这是通过使用 throw 关键字来完成的。
- catch: 在您想要处理问题的地方，通过异常处理程序捕获异常。catch 关键字用于捕获异常。
- try: try 块中的代码标识将被激活的特定异常。它后面通常跟着一个或多个 catch 块。

```
try
{
   // 保护代码
}catch( ExceptionName e1 )
{
   // catch 块
}catch( ExceptionName e2 )
{
   // catch 块
}catch( ExceptionName eN )
{
   // catch 块
}
```

多线程

多线程是多任务处理的一种特殊形式，多任务处理允许让电脑同时运行两个或两个以上的程序。一般情况下，两种类型的多任务处理：基于进程和基于线程。

- 基于**进程**的多任务处理是**程序**的并发执行。
- 基于**线程**的多任务处理是**同一程序的片段**的并发执行。

创建线程

```
#include <pthread.h>
pthread_create (thread, attr, start_routine, arg)
```

在这里，pthread_create 创建一个新的线程，并让它可执行。下面是关于参数的说明：

参数	描述
thread	指向线程标识符指针。
attr	一个不透明的属性对象，可以被用来设置线程属性。您可以指定线程属性对象，也可以使用默认值 NULL。
start_routine	线程运行函数起始地址，一旦线程被创建就会执行。
arg	运行函数的参数。它必须通过把引用作为指针强制转换为 void 类型进行传递。如果没有传递参数，则使用 NULL。

创建线程成功时，函数返回 0，若返回值不为 0 则说明创建线程失败。

终止线程

使用下面的程序，我们可以用它来终止一个 POSIX 线程：

```
#include <pthread.h>
pthread_exit (status)
```

在这里，pthread_exit用于显式地退出一个线程。通常情况下，pthread_exit()函数是在线程完成工作后无需继续存在时被调用。

如果 main()是在它所创建的线程之前结束，并通过 pthread_exit()退出，那么其他线程将继续执行。否则，它们将在 main()结束时自动被终止。

连接和分离线程

我们可以使用以下两个函数来连接或分离线程：

```
pthread_join (threadid, status)
pthread_detach (threadid)
```

pthread_join()子程序阻碍调用程序，直到指定的 threadid 线程终止为止。当创建一个线程时，它的某个属性会定义它是否是可连接的（joinable）或可分离的（detached）。只有创建时定义为可连接的线程才可以被连接。如果线程创建时被定义为可分离的，则它永远也不能被连接。