加油！当自己的实力不能满足自己的目标时，**就静下心去学习！**

@[toc]

# 类模板

## 类模板语法

类模板作用：

* 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

**解释：**

template --- 声明创建模板

typename --- 表面其后面的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

**示例：**

#include <string>  
//类模板  
template<class NameType, class AgeType>   
class Person  
{  
public:  
 Person(NameType name, AgeType age)  
 {  
 this->mName = name;  
 this->mAge = age;  
 }  
 void showPerson()  
 {  
 cout << "name: " << this->mName << " age: " << this->mAge << endl;  
 }  
public:  
 NameType mName;  
 AgeType mAge;  
};  
  
void test01()  
{  
 // 指定NameType 为string类型，AgeType 为 int类型  
 Person<string, int>P1("孙悟空", 999);  
 P1.showPerson();  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结**：类模板和函数模板语法相似，在声明模板template后面加类，此类称为类模板

## 类模板与函数模板区别

类模板与函数模板区别主要有两点：

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

**示例：**

#include <string>  
//类模板  
template<class NameType, class AgeType = int>   
class Person  
{  
public:  
 Person(NameType name, AgeType age)  
 {  
 this->mName = name;  
 this->mAge = age;  
 }  
 void showPerson()  
 {  
 cout << "name: " << this->mName << " age: " << this->mAge << endl;  
 }  
public:  
 NameType mName;  
 AgeType mAge;  
};  
  
//1、类模板没有自动类型推导的使用方式  
void test01()  
{  
 // Person p("孙悟空", 1000); // 错误 类模板使用时候，不可以用自动类型推导  
 Person <string ,int>p("孙悟空", 1000); //必须使用显示指定类型的方式，使用类模板  
 p.showPerson();  
}  
  
//2、类模板在模板参数列表中可以有默认参数  
void test02()  
{  
 Person <string> p("猪八戒", 999); //类模板中的模板参数列表 可以指定默认参数  
 p.showPerson();  
}  
  
int main() {  
  
 test01();  
  
 test02();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 类模板使用只能用显示指定类型方式
* 类模板中的模板参数列表可以有默认参数

## 类模板中成员函数创建时机

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

**示例：**

class Person1  
{  
public:  
 void showPerson1()  
 {  
 cout << "Person1 show" << endl;  
 }  
};  
  
class Person2  
{  
public:  
 void showPerson2()  
 {  
 cout << "Person2 show" << endl;  
 }  
};  
  
template<class T>  
class MyClass  
{  
public:  
 T obj;  
  
 //类模板中的成员函数，并不是一开始就创建的，而是在模板调用时再生成  
  
 void fun1() { obj.showPerson1(); }  
 void fun2() { obj.showPerson2(); }  
  
};  
  
void test01()  
{  
 MyClass<Person1> m;  
   
 m.fun1();  
  
 //m.fun2();//编译会出错，说明函数调用才会去创建成员函数  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：类模板中的成员函数并不是一开始就创建的，在调用时才去创建

## 类模板对象做函数参数

学习目标：

* 类模板实例化出的对象，向函数传参的方式

一共有三种传入方式：

1. 指定传入的类型 --- 直接显示对象的数据类型
2. 参数模板化 --- 将对象中的参数变为模板进行传递
3. 整个类模板化 --- 将这个对象类型 模板化进行传递

**示例：**

#include <string>  
//类模板  
template<class NameType, class AgeType = int>   
class Person  
{  
public:  
 Person(NameType name, AgeType age)  
 {  
 this->mName = name;  
 this->mAge = age;  
 }  
 void showPerson()  
 {  
 cout << "name: " << this->mName << " age: " << this->mAge << endl;  
 }  
public:  
 NameType mName;  
 AgeType mAge;  
};  
  
//1、指定传入的类型  
void printPerson1(Person<string, int> &p)   
{  
 p.showPerson();  
}  
void test01()  
{  
 Person <string, int >p("孙悟空", 100);  
 printPerson1(p);  
}  
  
//2、参数模板化  
template <class T1, class T2>  
void printPerson2(Person<T1, T2>&p)  
{  
 p.showPerson();  
 cout << "T1的类型为： " << typeid(T1).name() << endl;  
 cout << "T2的类型为： " << typeid(T2).name() << endl;  
}  
void test02()  
{  
 Person <string, int >p("猪八戒", 90);  
 printPerson2(p);  
}  
  
//3、整个类模板化  
template<class T>  
void printPerson3(T & p)  
{  
 cout << "T的类型为： " << typeid(T).name() << endl;  
 p.showPerson();  
  
}  
void test03()  
{  
 Person <string, int >p("唐僧", 30);  
 printPerson3(p);  
}  
  
int main() {  
  
 test01();  
 test02();  
 test03();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 通过类模板创建的对象，可以有三种方式向函数中进行传参
* 使用比较广泛是第一种：指定传入的类型
* 如果想知道推导类型的名字 可以使用 typeid(T).name()

## 类模板与继承

当类模板碰到继承时，需要注意以下几点：

* 当子类继承的父类是一个类模板时，子类在声明的时候，要指定出父类中T的类型
* 如果不指定，编译器无法给子类分配内存 (继承下来)
* 如果想灵活指定出父类中T的类型，子类也需变为类模板

**示例：**

template<class T>  
class Base  
{  
 T m;  
};  
  
//class Son:public Base //错误，c++编译需要给子类分配内存，必须知道父类中T的类型才可以向下继承  
class Son :public Base<int> //必须指定一个类型  
{  
};  
void test01()  
{  
 Son c;  
}  
  
//类模板继承类模板 ,可以用T2指定父类中的T类型  
template<class T1, class T2>  
class Son2 :public Base<T2>  
{  
public:  
 Son2()  
 {  
 cout << typeid(T1).name() << endl;  
 cout << typeid(T2).name() << endl;  
 }  
};  
  
void test02()  
{  
 Son2<int, char> child1;  
}  
  
  
int main() {  
  
 test01();  
  
 test02();  
  
 system("pause");  
  
 return 0;  
}

总结：如果父类是类模板，子类需要指定出父类中T的数据类型

## 类模板成员函数类外实现

学习目标：能够掌握类模板中的成员函数类外实现

**示例：**

#include <string>  
  
//类模板中成员函数类外实现  
template<class T1, class T2>  
class Person {  
public:  
 //成员函数类内声明  
 Person(T1 name, T2 age);  
 void showPerson();  
  
public:  
 T1 m\_Name;  
 T2 m\_Age;  
};  
  
//构造函数 类外实现  
template<class T1, class T2>  
Person<T1, T2>::Person(T1 name, T2 age) {  
 this->m\_Name = name;  
 this->m\_Age = age;  
}  
  
//成员函数 类外实现  
template<class T1, class T2>  
void Person<T1, T2>::showPerson() {  
 cout << "姓名: " << this->m\_Name << " 年龄:" << this->m\_Age << endl;  
}  
  
void test01()  
{  
 Person<string, int> p("Tom", 20);  
 p.showPerson();  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：类模板中成员函数类外实现时，需要加上模板参数列表

## 类模板分文件编写

学习目标：

* 掌握类模板成员函数分文件编写产生的问题以及解决方式

问题：

* 类模板中成员函数创建时机是在**调用阶段**，导致分文件编写时链接不到

解决：

* 解决方式1：直接包含.cpp源文件（不要将声明与实现分开书写）
* 解决方式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

**示例：**

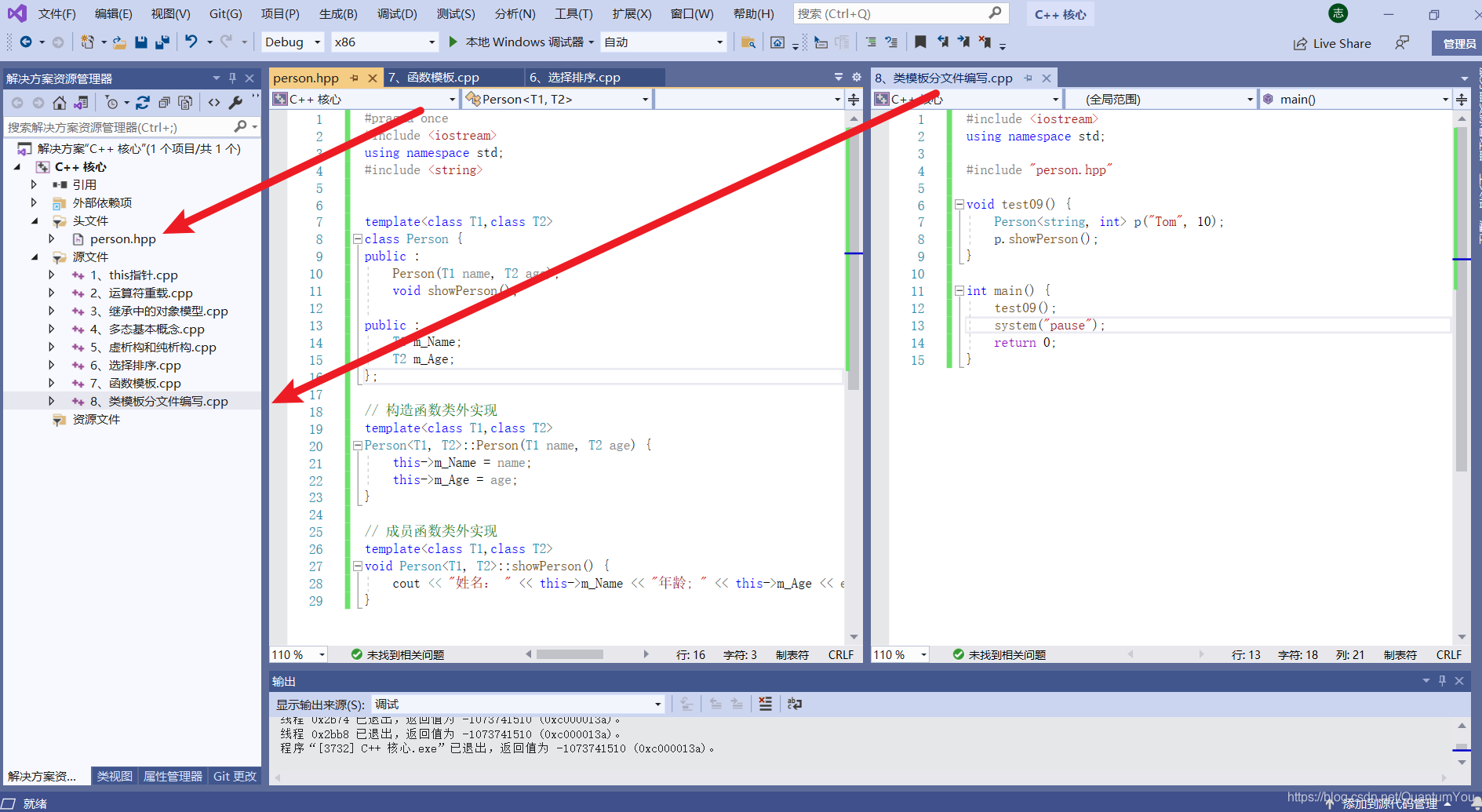
person.hpp中代码：

#pragma once  
#include <iostream>  
using namespace std;  
#include <string>  
  
template<class T1, class T2>  
class Person {  
public:  
 Person(T1 name, T2 age);  
 void showPerson();  
public:  
 T1 m\_Name;  
 T2 m\_Age;  
};  
  
//构造函数 类外实现  
template<class T1, class T2>  
Person<T1, T2>::Person(T1 name, T2 age) {  
 this->m\_Name = name;  
 this->m\_Age = age;  
}  
  
//成员函数 类外实现  
template<class T1, class T2>  
void Person<T1, T2>::showPerson() {  
 cout << "姓名: " << this->m\_Name << " 年龄:" << this->m\_Age << endl;  
}

类模板分文件编写.cpp中代码

#include<iostream>  
using namespace std;  
  
//#include "person.h"  
#include "person.cpp" //解决方式1，包含cpp源文件  
  
//解决方式2，将声明和实现写到一起，文件后缀名改为.hpp  
#include "person.hpp"  
void test01()  
{  
 Person<string, int> p("Tom", 10);  
 p.showPerson();  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：主流的解决方式是第二种，将类模板成员函数写到一起，并将后缀名改为.hpp



[参考链接](https://www.bilibili.com/video/BV1et411b73Z?p=180&spm_id_from=pageDriver)

## 类模板与友元

学习目标：

* 掌握类模板配合友元函数的类内和类外实现

全局函数类内实现 - 直接在类内声明友元即可

全局函数类外实现 - 需要提前让编译器知道全局函数的存在

**示例：**

#include <string>  
  
//2、全局函数配合友元 类外实现 - 先做函数模板声明，下方在做函数模板定义，在做友元  
template<class T1, class T2> class Person;  
  
//如果声明了函数模板，可以将实现写到后面，否则需要将实现体写到类的前面让编译器提前看到  
//template<class T1, class T2> void printPerson2(Person<T1, T2> & p);   
  
template<class T1, class T2>  
void printPerson2(Person<T1, T2> & p)  
{  
 cout << "类外实现 ---- 姓名： " << p.m\_Name << " 年龄：" << p.m\_Age << endl;  
}  
  
template<class T1, class T2>  
class Person  
{  
 //1、全局函数配合友元 类内实现  
 friend void printPerson(Person<T1, T2> & p)  
 {  
 cout << "姓名： " << p.m\_Name << " 年龄：" << p.m\_Age << endl;  
 }  
  
  
 //全局函数配合友元 类外实现  
 friend void printPerson2<>(Person<T1, T2> & p);  
  
public:  
  
 Person(T1 name, T2 age)  
 {  
 this->m\_Name = name;  
 this->m\_Age = age;  
 }  
  
  
private:  
 T1 m\_Name;  
 T2 m\_Age;  
  
};  
  
//1、全局函数在类内实现  
void test01()  
{  
 Person <string, int >p("Tom", 20);  
 printPerson(p);  
}  
  
  
//2、全局函数在类外实现  
void test02()  
{  
 Person <string, int >p("Jerry", 30);  
 printPerson2(p);  
}  
  
int main() {  
  
 //test01();  
  
 test02();  
  
 system("pause");  
  
 return 0;  
}

**总结**：建议全局函数做类内实现，用法简单，而且编译器可以直接识别