1 函数模板基本使用

1.1 template < class / typename T> 告诉编译器紧跟的代码里出现 T 不要报错

1.2 mySwap(T &a T &b) 类型也需要传入 ，类型参数化

1.3 myswap（a,b） 自动类型推导 按照 a b 的类型 来替换 T

1.4 myswap<int>(a,b) 显示指定类型

2 函数模板与普通函数的区别以及调用规则

2.1 区别 普通函数可以进行隐式类型转换 模板不可以

2.2 调用规则

2.2.1 c++编译器优先考虑普通函数

2.2.2 可以通过空模板实参列表的语法限定编译器只能通过模板匹配

2.2.3 函数模板可以像普通函数那样可以被重载

2.2.4 如果函数模板可以产生一个更好的匹配，那么选择模板

2.3 模板的机制

2.3.1 模板并不是万能的，不能通用所有的数据类型

2.3.2 模板不能直接调用，生成后的模板函数才可以调用

2.3.3 二次编译，第一次对模板进行编译，第二次对替换 T 类型后的代码进行二次编译

3 模板局限性

3.1 模板不能解决所有的类型

3.2 如果出现不能解决的类型，可以通过第三地具体化来解决问题

3.3 template<> 返回值 函数名<具体类型>（参数）

4 类模板

4.1 写法 template <T...> 紧跟着是类

4.2 与函数模板区别，可以有默认类型参数

4.3 函数模板可以进行自动类型推导，而类模板不可以

4.4 类模板中的成员函数 一开始不会创建出来，而是在运行时才去创建

5 类模板做函数的参数

5.1 三种方式

5.1.1 显示指定类型

5.1.2 参数模板化

5.1.3 整体模板化

5.2 查看类型名称

5.2.1 cout << typeid(T).name() << endl;

6 当类模板碰到继承

6.1 基类如果是模板类，必须让子类告诉编译器 基类中的 T 到底是什么类型

6.2 如果不告诉，那么无法分配内存，编译不过

6.3 利用参数列表 `class Child :public Base<int>`

7 类模板的类外实现成员函数

```
template <class T1, class T2>
```

7.1 Person<T1, T2>::Person(T1 name, T2 age)

8 类模板的分文件编写问题以及解决

8.1 .h .cpp 分别写声明和实现

8.2 但是由于 类模板的成员函数运行阶段才去创建，导致包含.h 头文件，不会创建函数的实现，无法解析外部命令

8.3 解决方案 保护 .cpp 文件 （不推荐）

8.4 不要进行分文件编写，写到同一个文件中，进行声明和实现，后缀名改为.hpp

8.5 约定俗成的

9 类模板碰到友元函数

9.1 友元函数类内实现

9.2 friend void printPerson(Person<T1 ,T2> & p)

9.3 友元函数类外实现

9.4 friend void printPerson<>(Person<T1, T2> & p); //没有<>普通函数 声明 加上 <>模板函数声明

9.5 让编译器看到 函数 并且看到这个 Person 类型

10