|  |  |
| --- | --- |
| 1. 模板特例化 2. 定义函数模板特例化 3. 函数重载与模板特例化 4. 类模板特例化 5. 类模板部分特例化 6. 特例化成员而不是类 | 模板特例化  2.  Template<>  Int compare(const char\* const &p1,const char\* const &p2)  {  Return strcmp(p1,p2);  }  Compare(“hi”,”mom”);  4.  Namespace std{  Template <>  Struct hash<Sales\_data>  {  Typedef size\_t result\_type;  Typedef Sales\_data argument\_type;  Size\_t operator()(const Sales\_data& s)const;  };  Size\_t  Hash<Sales\_data>::operator()(const Sales\_data& s)const  {  Return hash<string>()(s.bookNo)^hast<unsigned>()(s.units\_sold)^hash<double>()(s.revenue);  }  }  Unordered\_multiset<Sales\_data> SDset;  Template <class T> class std::hash;  Class Sales\_data{  Friend class std::hash<Sales\_data>;  }  5.  Template <class T>struct remove\_reference{  Typedef T type;  }  Template <class T> struct remove\_reference<T&>{  Typedef T type;  }  Template <class T> struct remove\_reference<T&&>{  Typedef T type;  }  6.  Template<>  Void Foo<int>::Bar(){…} |
| 1. 模板特例化 2. 单一模板，让能任何模板实参都是适合的，都能实例化，办不到。 3. 不希望使用通用模板版本时，可以定义类或函数模板的一个特例化版本 4. Compare调用非类型参数的版本compare(“hi”,”com”);非类型参数版本处理不了参数是p1和p2，字符指针的情况，无法将指针转换为一个数组的引用。   我们可以将第一个版本的compare定义一个模板特例，处理字符指针  它本身被调用是转换为char[]是不同的。   1. 定义函数模板特例化 2. Template后加<>空尖括号对。将为原模板的所有模板参数提供实参 3. 特例化版本的函数参数类型必须与声明的模板中对应的类型匹配。就是若声明中const T&,则特例化版本也要有const &,但对T可以自定义 4. 设置T为const char\*,结合const &则为const char\* const &,得到一个指向const char的const指针的引用 5. 函数重载与模板特例化 6. 当实例化特定版本实际上接管了编译器的工作，为原模板的一个特殊实例提供了定义。并非函数名的一个重载版本 7. 3个函数模板中，2个版本（第一个和第二个），对于compare(“hi”,”mom”)都是可行的，但是接收一个字符数组参数的版本更特例化，就是const char(&)[N]更特例化，会选择这个。得试试，是的 8. 若将接收字符指针的compare定义为普通的非模板函数，则这个更精确，会选择这个。只是前面的规则 9. 类模板特例化 10. 实例化hash特例化版本，必须在原模板定义所在的命名空间定义它，所以要开std空间添加成员 11. 一个特例化hash类： 12. 一个重载的调用运算符，接受一个容器关键字类型的对象，返回一个size\_t当哈希值？ 13. 两个类型成员，result\_type,argument\_type，调用运算符的返回和参数类型 14. 默认构造函数和拷贝赋值运算符   反正需要==运算符是为了判断相同的Sales\_data放在一起。  Hash是生成Sales\_data的哈希值，Size\_t是保存哈希值，argument是类型  unordered\_multiset会组合使用Sales\_data对应的特例化hash就是调用hash<Sales\_data>得到哈希值与Sales\_data上的等于运算符组合判断是否放在一起的意思吗？   1. Template<>开始，特例化版本为hash<Sales\_data>,哈希值为key\_type 2. 为Sales\_data定义的operator==(p497,同样用了3个成员)是兼容的。为了处理特定关键字类型，无序容器会组合使用key\_type对应的特例化hash版本和key\_type上的相等运算符。不懂。Key\_type是Sales\_data吗？ 3. 当将Sales\_data作为容器的关键字类型时，编译器就会自动使用此特例化版本 4. 由于hash<Sales\_data>使用Sales\_data私有成员，所以需要将它声明为Sales\_data的友元 5. 友元需要声明原模板版本 6. 类模板部分特例化 7. 与函数模板不同，类模板的特例化不必为所有模板参数提供实参。可以指定一部分而非所有模板参数，或是参数的一部分而非全部特性。 8. 一个类模板的部分特例化本身是一个模板，使用它时用户还必须在特例化版本中未指定的模板参数提供实参。(就是在类名后指定特定的实参)   Note:我们只能部分特例化类模板，而不能部分特例化函数模板   1. 部分特例化版本，就是用于左值引用和右值引用 2. 第一个模板定义了通用模板，将模板实参作为type成员的类型 3. 接下来的两个特例化版本的模板参数列表与原模板的模板参数列表相同，类名之后，要特例化的模板参数指定实参，这些实参与原始的模板中的参数按位置对应。 4. 部分特例化版本的模板参数列表是原始模板的参数列表的一个子集或者是一个特例化版本。模板参数的数目相同，类型不同 5. 得到type去除引用的实现方式就是使用部分特例化，用T&与T&&特例化，得到类型 6. 特例化成员而不是类 7. 只特例化特定成员函数而不是特例化整个模板 8. 当用int使用Foo时，Bar之外的成员像往常一样进行实例化。如果使用Foo<int>的成员Bar，则会使用我们定义的特例化版本 | |