```
template <typename T>
                                                              void func(const T& value) {}
                                                              func(10);
                                                类型区别问题
                                                              在这个例子中,T 将会被推断为 int, value 的类型则是 const int&,也就是一个常量引用
                                                              函数实参其实是一个右值,但是由于形参是一个常量引用,所以既可以接收左值,也可以接收右值
                                                                   - && 表示一个右值引用,int&& 表示类型为 int 的右值引用,当函数原型为 void (*f)(int &&) 时,我们只能传递右值,而无法传递左值
                                                                    但是,一旦把这玩意儿扔到函数模板里面,事情就变得微妙起来了
                                                                    template <typename T>
                                                                    void func(T&& value) {}
                                                                                                                     此时 func 函数模板既可以接收左值实参,也可
                                                                    int i = 10;
                                                                                                                     以接收右值实参
                                                                    func(i);
                                                                              // 可传递左值
                                                                    func(1024); // 也可传递右值
                                                - Universal Reference
                                                                                                                               int buffer = 1024;
                                                                                                                               func(buffer);
                                                                    当实参为左值时,T 将被推断为左值引用,同时,value 的类型也为左值引用
                                                                                                                                此时 T 的类型为 int&,value 的类型也为 int &
                                  万能引用
                                                                                                                                (这其中发生了引用折叠,后续详细补充)
                                                                                                                         func(1024);
                                                                     当实参为右值时,T 将被推断为右值的值类型,value 为右值引用
                                                                                                                          此时 T 的类型为 int,vlaue 的类型为 int&&
                                                                  必须定义于函数模板,且形式为 T&&
                                                                 template <typename T>
                                                                 class Buz {
                                                                 public:
                                                                                         · func 中的 T 并不是万能引用,因为类模板的类型参数无法推断,必须显式指定
                                                万能引用产生条件
                                                                   void func(T&& value) {}
                                                                 template <typename T>
                                                                                               此时 func 也只能接收右值
                                                                 void func(vector<T>&& value) {}
                                                                                                        template <typename T>
                                                                                                        void func(const T&& value) {}
                                                 const 修饰符将会剥夺一个引用成为万能引用的资格,此时为右值引用
                                                                                                        非万能引用,仅为右值引用
                                                                   Boost.TypeIndex 是 Boost 中一个很轻量的库,不需要链接,能够比较精准的知道一个表达式或变量的类型,比标准库中的 typeid 好使
万能引用与类型
                                                                          #include <boost/type_index.hpp>
    推断问题
                                                                          using boost::typeindex::type_id_with_cvr;
                                                    借助 boost 库
                                                                          template <typename T>
                                                                           void func(T& value) {
                                                                            cout<<"T = "<<type_id_with_cvr<T>().pretty_name()<<endl;</pre>
                                                                   使用
                                                                            cout<<"value = "<<type_id_with_cvr<decltype(value)>().pretty_name()<<endl;</pre>
                                                                           我们将会改变 func 该函数的函数形参,以观察在不同情况下不同实参类型所得到的不同推断即结果
                                                                       值传递的函数原型,形参 value 为实参的一个拷贝。通常来说,不管是 T 还是 value,都会忽略掉实参的 const 属性,但
                                                                       是指针却是一个例外
                                                                                                    const int buffer = 1024; — T = int, value = int
                                                                       const int buffer = 1024;
                                                                       const int *ptr = &buffer;
                                                                                                    const int *ptr = &buffer; — T = const int *, value = const int *
                                                                       const int * const ptr2 = &buffer;
                                                                                                    const int * const ptr2 = &buffer; — T = const int *, value = const int *
                                                   void func(T value)
                                                                       · 当指针作为形参时,该指针是否为顶层 const 无关紧要,因此会忽略掉第二个 const
                                                                                        int bar(int k) {}
                                                                       函数指针作为实参 =
                                                                                                       - T = int (*)(int), value = int (*)(int)
                                                                                        func(bar)
                                                                       也就是说,在这种情况下,T 的类型和 value 的类型基本保持一致
                                  类型推断问题
                                                                        此时 func 仅能接收左值,无法接收右值
                                                                                                   func(i); --- T = int, value = int&
                                                                                   int i = 1024;
                                                                       - 普通类型 -
                                                                                 - const int j = i;
                                                                                                   func(j); — T = const int, value = const int&
                                                                                   const int \&k = j;
                                                                                                   func(k); — T = const int, value = const int&
                                                                                                             - func(s); --- T = char [6], value = char (&) [6]
                                                                                      char s[] = "Hello";
                                                   void func(T& value)
                                                                        数组作为实参
                                                                                      const char cs[] = "World";
                                                                                                              func(cs); — T = const char [6], value = const char (&) [6]
                                                                                                       \subset T = int (int), value = int (&)(int)
                                                                                         int bar(int k) {}
                                                                        函数指针作为实参 =
                                                                                         func(bar)
                                                                                                         此时 T 就是一个可调用对象的原型,而 value 则是该原型的一个引用
                                                                        若实参为引用类型,例如上面的 k,是一个 int 类型的引用,那么引用类型在类型推断中被忽略,即 T 一定不会被推断成引用类型
                                                                        并且, const 属性由 T 获得,也就是说,如果在 func 函数内部对 value 进行修改的话,将会抛出异常
                                                   - void func(const T& value) —— 不再展开,情况和 void func(T& value) 基本类似,只不过此时 const 属性仅在 value 身上,而不会在 T 身上
                                                                                                         · 当实参为左值时,T 的类型本应该是 int &&&,将会被折叠成为 int&,即左值引用
                                                                                           func(buffer);
                                                                         int buffer = 1024;
                                                                                                         T = int&, value = int&
                                                    void func(T&& value) -
                                                                         - func(buffer);
                                                                         func(1024);
                                                                                           func(1024); \longrightarrow T = int, value = value = int&&
```