```
Posted on 2020-05-16 17:29 浅墨浓香 阅读(465) 评论(1) 编辑 收藏 举报
```

8.5 Compile-Time if 8.5 编译期if

第8章 编译期编程: 8.5 编译期if

那部分(如果有的话)。

} 

Partial specialization, SFINAE, and std::enable\_if allow us to enable or disable templates as a whole. C++17 additionally introduces a compile-time if statement that allows is to enable or disable specific statements based

on compile-time conditions. With the syntax if constexpr(...), the compiler uses a compile-time expression to decide whether to apply the then part or the else part (if any). 偏特化、SFINAE以及std::enable\_if允许我们从整体上启用或禁用模板。C++17还引入的<mark>编译期if语句</mark>,它允许根据编译

template<typename T, typename... Types>

std::cout << firstArg << '\n';

Section 1.1.3 on page 6). For example:

实例化】(请参阅第6页的1.1.3节)。例如:

if constexpr (std::is\_integral\_v<T>) {

if (t > 0) {
foo(t - 1); // OK

//undeclared();

expression that yields a Boolean value. For example:

foo(42); // OK

template<typename T, std::size\_t SZ> void foo(std::array<T, SZ> const& coll) if constexpr (!isPrime(SZ)) { //special additional

See Section 14.6 on page 263 for further details.

specialization or operator ?: for selections).

that are callable in compile-time contexts.

certain compile-time constraints.

更多细节请参阅第263页14.6节。

if constexpr (std::numeric\_limits<char>::is\_signed{

page 125, to perform additional code if a given size is not a prime number:

undeclared(42); // error if undeclared() not declared

template<typename T> void foo(T t)

else {

} } 

达式。例如:

} 

8.6 Summary 8.6 小结

int main()

}else {

void print (T const& firstArg, Types const& ... args)

期的条件启用或禁用特定语句。借助if constexpr()语法,编译器使用编译期表达式来决定是使用then的那部分还是else的

As a first example, consider the variadic function template print() introducedin Section 4.1.1 on page 55. It prints its arguments (of arbitrary types) using recursion. <u>Instead of providing</u> a separate function to end the

recursion, the constexpr if feature allows us to decide locally whether to continue the recursion: 作为第1个例子,考虑第55页4.1.1节介绍的可变参数模板print()。它使用递归的方法来打印其参数(任意类型)。

if constexpr (sizeof...(args) > 0) {
 print(args...); //只有在sizeof...(args)>0时才会提供代码 (从C++17开始)

instantiated. Thus, a corresponding function is not required to exist and the recursion ends.

Here, if print() is called for one argument only, args becomes an empty parameter pack so that sizeof...(args) becomes 0. As a result, the recursive call of print() becomes a discarded statement, for which the code is not

在这里,如果只给print()传递一个参数,则<mark>args</mark>变为一个<mark>空参数包</mark>,因此<mark>sizeof...(args)</mark>为<mark>0。</mark>这样,if语句里面的递归 print()语句就会被丢弃,也就是说这部分代码不会被实例化。因此,不需要一个单独的函数也可以结束递归。 The fact that the code is not instantiated means that only the first translation phase (the definition time) is performed, which checks for correct syntax and names that don't depend on template parameters (see

事实上,上面所说的代码不会被实例化,意思是这些代码只会进行第1个翻译阶段,这时只会做一个语法检查以及和模板 参数无关的名称检查,但不会实例化代码出来【译注:实例化是在第2阶段进行的。即constexpr if语句只影响第2阶段的

//undeclared(t); // 依赖型名称,第1阶段不会检查是否己声明undeclared(t)。在第2阶段实例化时,

static\_assert(false, "no integral"); // 总是assert (即使else语句被丢弃时一样)

static\_assert(!std::is\_integral\_v<T>, "no integral"); //OK

Note that if constexpr can be used in any function, not only in templates. We only need a compile-time

注意,if constexpr的使用并不限于模板函数,它可以用于任何函数中。它所需要的只是一个能够返回布尔值的编译期表

static\_assert(false, "unsigned"); // always asserts (even if discarded) static\_assert(!std::numeric\_limits<char>::is\_signed, "char is unsigned"); //OK

With this feature, we can, for example, use our isPrime() compile-time function, introduced in Section 8.2 on

利用这一特性,我们可以让第125页8.2节中介绍的编译期函数isPrime在其参数不是质数时执行一些额外的代码:

• Templates provide the ability to compute at compile time (using recursion to iterate and partial

• With constexpr functions, we can replace most compile-time computations with "ordinary functions"

• Templates are used only if needed and substitutions in function template declarations do not result in

• With partial specialization, we can choose between different implementations of class templates based on

模板提供了在编译期进行计算的能力(<mark>使用递归进行<mark>遍历</mark>以及使用偏特化或?:运算符进行<mark>选择</mark>)</mark>

通过constexpr函数,可以将大多数<mark>编译期计算</mark>替换为<mark>编译期上下文中可调用的"普通函数"</mark>

// error: 如果未声明undeclared时,在第1阶段翻译时就会报错

这里如果print()只有一个实参,args为空参数包,sizeof...(args)为0,说 归调用print()的语句变为废弃语句,不会实例化,因此不会要求对应的 函数存在,递归结束。不会实例化意味着只执行Two-Phase Translation的第一阶段,检查语法正确性和不依赖于模板参数的名称

//当r不为整型时,会进入else语句。此时如果发现undeclared(t)未声明,则会报错。 //(在第2阶段时,如果r为整型由于编译期if的特点,该语句被丢弃,因此不会报错)。

handling if the passed array has no prime number as size

constexpr if特性允许我们在本地决定是否继续递归,而不用再单独定义一个函数来结束递归:

invalid code. This principle is called SFINAE(substitution failure is not an error). 模板只有<mark>在被需要的时候</mark>才会<mark>被使用,对函数模板声明进行替换不会产生无效代码</mark>,这个原则称为SFINAE(替换失 败不是错误) • SFINAE can be used to provide function templates only for certain types and/or constraints. SFINAE可以被用来只为某些类型或限制条件提供函数模板。 • Since C++17, a compile-time if allows us to enable or discard statements according to compile-time

通过特化,可以根据某种编译期限制条件选择不同的类模板实现。

conditions (even outside templates) 从C++17开始,编译期if允许我们根据编译期条件启用或丢弃某些语句(甚至可用于非模板中) 分类: C++模板编程

好文要顶 关注我 收藏该文 6 1 浅墨浓香 关注 - 0

粉丝 - 248 +加关注 « 上一篇: 第8章 编译期编程: 8.4 SFINAE(替换失败并不是错误) » 下一篇: 第9章 在实践中使用模板: 9.1 包含模型