以正确地使用模板。
10.1 On-Demand实例化(隐式实例化或自动实例化) 当C++编译器遇到模板特化的使用时,它会利用所给的实参替换对应的模板参数,从而产生该模板的特化。 这个过程是编译器自动进行的,并不需要客户端代码来引导(或者不需要模板定义来引导)。 on-demand实例化表明:在使用模板(特化)的地方,编译器通常需要访问模板和某些模板成员的整个定义
(也就是说,只有声明是不够的)。如下: template <typename t=""> class C; // (1) 这里只有声明</typename>
<pre>C<int>* p = 0; // (2)正确: 并不需要C<int>的定义 template<typename t=""></typename></int></int></pre>
class C { public: void f();
// (4) 类模板定义结束 void g(C <int>& c) // (5) 只使用类模板声明 { c.f(); // (6) 使用了类模板的定义,需要C::f()的定义</int>
}; •
下面是另一个需要进行(前面的)类模板实例化的表达式,因为编译器需要知道C <void>的大小: C<void>* p = new C<void>; 在源码中,有时候需要访问类模板成员,但在源码中这种需求并不总是显式可见的。例如,C++的重载解析</void></void></void>
在源码中,有的候需要访问类模板成页,但在源码中这种需求并不总是显式可见的。例如,C++的重氧解析规则会要求:如果候选函数的参数是class类型,那么该类型所对应的类就必须是可见的: □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
template <typename t=""> class C { public:</typename>
C(int); // 具有单参构造函数,可以被用于隐式类型转换 }; void candidate(C <double> const&); // (1) void candidate(int) {} // (2)</double>
int main() { candidate(42);
} 请 调用candidate(42)将会采用(2)处的重载声明。然而,编译器仍然可以实例化(1)处的声明,来检查产
生的实例能否成员该调用的一个有效候选函数。 10.2 延迟实例化 1. 模板实例化只对确实需要的部分进行实例化。换句话说,编译器会延迟模板的实例化。
2. 当隐式实例化类模板时,同时也实例化了该模板的每个成员声明,但并没有实例化相应的定义。 3. 然而,存在一些例外的情况:首先,如果类模板包含了一个匿名的union,那么该union定义的成员同时 被实例化了(匿名的union有它自身的特殊之处:它的成员可以被看成是外围类的成员。匿名成员可以看作影
一种构造,用来说明某些类成员共享同一个存储器)。 4.另一种例外情况发生在虚函数身上:作为实例化类模板的结果,虚函数的定义可能被实例化了,但也可能 还没有被实例化,这要依赖于具体的实现。实际上,许多实现都会实例化(虚函数)的定义,因为"实现虚函 数调用机制的内部结构"要求虚函数(的定义)作为链接实体存在。
5. 当实例化模板的时候,缺省的函数调用实参是分开考虑的。准确而言,只有这个被调用的函数(或成员函数)确实使用了缺省实参,才会实例化该实参。就是说,如果这个函数(或成员函数)不使用缺省调用实参 而是使用显式实参来进行调用,那么就不会实例化缺省实参。
10.3 C++的实例化模型 10.3.1 两阶段查找 上一篇中我们知道:当对模板进行解析的时候,编译器并不能解析依赖型名称。于是,编译器会在POI(poir
工 扁平弦问知道: 当为侯成处门解初时的原,编件部分行能解初代数型石标。了定,编件部公住FOI(poil of instantiation,实例化点)再次查找这些依赖型名称。另一方面,非依赖型名称是在首次看到模板的时候就 进行查找,因此在第1次查找时就可以诊断错误信息,于是,就有了两阶段查找这个概念: 第1阶段发生在模 板的解析阶段,第2阶段发生在模板的实例化阶段。 在第1阶段,当使用普通查找规则(在适当的情况下也会使用ADL)对模板进行解析时,就会查找非依赖型名
称。另外,非受限的依赖型名称(诸如函数调用中的函数名称,之所以说它是依赖型的,是因为该名称具有 个依赖型实参)也会在这个阶段进行查找,但它的查找结果是不完整的(就是说查找还没结束),在实例化 板的时候,还会再次进行查找。 第2阶段发生在模板被实例化的时候,我们也称此时发生的地点(或者源代码的某个位置)为一个实例化点
POI。依赖型受限名称就是在此阶段进行查找的(查找的目标是:运用模板实参代替参数之后所获得的特定等例化体);另外,非受限的依赖型名称在此阶段也会再次执行ADL查找。 10.3.2 POI
1. 非类型POI (非类型POI、非类型实例、非类型实体、非类型特化这里``非类型''可以理解为(个人理解)不是类的类 型。) 从上面我们知道,C++编译器会在模板客户端代码中的某些位置访问模板实体的声明或者定义。于是,当某 些代码构造引用了模板特化,而且为了生成这个完整的特化,需要实例化相应模板的定义时,就会在源代码
些代码构造引用了模板特化,而且为了生成这个完整的特化,需要实例化相应模板的定义时,就会在源代码产生一个实例化点(POI)。我们应该清楚,POI是位于源代码中的一个点,在该点会插入替换后的模板实例。例如:
class MyInt { public:
<pre>MyInt(int i); }; MyInt operator - (MyInt const&);</pre>
<pre>bool operator > (MyInt const&, MyInt const&); typedef MyInt Int; template <typename t=""> void f(T i)</typename></pre>
<pre>{ if(i > 0) { g(-i);</pre>
} // (1) void g(Int) {
// (2) f <int>(42); // 调用点 // (3)</int>
· // (4)
当C++编译器看到调用f <int>(42)时,它知道需要用MyInt替换T来实例化模板f:即生成一个POI。(2)和(3)处是临近调用点的两个地方,但它们不能作为POI,因为C++并不允许我们把::f<int>(Int)的定义在这里插入。另外,(1)处和(4)处的本质区别在于:在(4)处,函数g(Int)是可见的,而(1)处则不是;因此在(4)处函数g(-i)可以被解析。然而,如果我们假定(1)处作为POI,那么调用g(-i)将不能被</int></int>
析,因为g(Int)在(1)处是不可见的。幸运的是,对于指向非类型特化的引用,C++把它的POI定义在``包 含这个引用的定义或声明之后的最近名字空间域"中。在我们的例子中,这个位置是(4)。 你可能会疑惑我们为什么在例子中使用类型MyInt,而不直接使用简单的int类型。这主要是因为:在POI执行的第2次查找 (指g(-i))只是使用了ADL。而基本类型int并没有关联名字空间,因此,如果使用了int类型,就不会发生
ADL查找,也就不能找到函数g。 2. 对于类特化,这个(POI)位置是不一样的。如下:
<pre>template <typename t=""> class S { public:</typename></pre>
T m; }; // (5) unsigned long h()
<pre>{ // (6) return (unsigned long)sizeof(S<int>); // (7)</int></pre>
} // (8)
如前所述,我们知道位置(6)和(7)不能作为POI,因为名字空间域类S <int>的定义不能出现在这两个位置(模板是不能出现在函数作用域内部的)。如果我们采用前面非类型实例的规则,那么POI应该在(8) 处,但这样的话,表达式sizeof(S<int>)会是无效的,因为要等到在编译到(8)之后,我们才能确定</int></int>
S <int>的大小,而代码sizeof(S<int>)位于(8)之前。因此,对于指向产生自模板的类实例的引用,它的 POI只能定义在"包含这个实例引用的定义或声明之前的最近名字空间域"。在我们的这个例子中,是指位置 (5)。</int></int>
注:书中还介绍了某些附带的实例化以及二次POI,详见书籍;
一个翻译单元通常会包含同个实例的多个POI。对于类模板实例而言,在每个翻译单元中,只有首个POI会被保留,而其他的POI则被忽略。对于非类型实例而言,所有的POI都会被保留。然而,对于上面的任何一种愉
一个翻译单元通常会包含同个实例的多个POI。对于类模板实例而言,在每个翻译单元中,只有首个POI会被保留,而其他的POI则被忽略。对于非类型实例而言,所有的POI都会被保留。然而,对于上面的任何一种情况,ODR原则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 当遇到POI的时候,(编译器要求)相应模板的定义必须是(基于某种方式)可见的。对于类特化而言,这就意味着:在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的。对于非类型的POI而言,也可能会采取上面的方式:当我们通常会把非类型模板的定义放在一个头文件中,然后在需要使用该定义的时
注:书中还介绍了某些附带的实例化以及二次POI,详见书籍; 一个翻译单元通常会包含同个实例的多个POI。对于类模板实例而言,在每个翻译单元中,只有首个POI会被保留,而其他的POI则被忽略。对于非类型实例而言,所有的POI都会被保留。然而,对于上面的任何一种情况,ODR原则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 当遇到POI的时候,(编译器要求)相应模板的定义必须是(基于某种方式)可见的。对于类特化而言,这意味着:在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的。对于非类型的POI而言,也可能会采取上面的方式:当我们通常会把非类型模板的定义放在一个头文件中,然后在需要使用该定义的时候,把这个头文件#include到这个翻译单元中。这种处理模板定义的源模型就是我们前面所谈到的包含模型,也是目前为止最广泛的实现方式。 对于非类型POI,还存在另一种实现方法:使用export关键字来声明非类型模板,而在另一个翻译单元中定该非类型模板。这就是我们前面所谈到的分离模型。
一个翻译单元通常会包含同个实例的多个POI。对于类模板实例而言,在每个翻译单元中,只有首个POI会被保留,而其他的POI则被忽略。对于非类型实例而言,所有的POI都会被保留。然而,对于上面的任何一种情况,ODR原则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 当遇到POI的时候,(编译器要求)相应模板的定义必须是(基于某种方式)可见的。对于类特化而言,这意味着:在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的。对于非类型的POI而言,也可能会采取上面的方式:当我们通常会把非类型模板的定义放在一个头文件中,然后在需要使用该定义的时候,把这个头文件#include到这个翻译单元中。这种处理模板定义的源模型就是我们前面所谈到的包含模型,也是目前为止最广泛的实现方式。对于非类型POI,还存在另一种实现方法:使用export关键字来声明非类型模板,而在另一个翻译单元中定该非类型模板。这就是我们前面所谈到的分离模型。 10.3.4 跨翻译单元查找模板中的名称分两阶段查找:
一个翻译单元通常会包含同个实例的多个POI。对于类模板实例而言,在每个翻译单元中,只有首个POI会被保留,而其他的POI则被忽略。对于非类型实例而言,所有的POI都会被保留。然而,对于上面的任何一种情况,ODR原则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 当遇到POI的时候,(编译器要求)相应模板的定义必须是(基于某种方式)可见的。对于类特化而言,这意味着:在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的。对于非类型的POI而言,也可能会采取上面的方式:当我们通常会把非类型模板的定义放在一个头文件中,然后在需要使用该定义的时候,把这个头文件#include到这个翻译单元中。这种处理模板定义的源模型就是我们前面所谈到的包含模型,也是目前为止最广泛的实现方式。对于非类型POI,还存在另一种实现方法:使用export关键字来声明非类型模板,而在另一个翻译单元中定该非类型模板。这就是我们前面所谈到的分离模型。 10.3.4 跨翻译单元查找模板中的名称分两阶段查找: 第1阶段发生在解析模板(也就是说,C++编译器第1次看到模板定义)的时候。在这个过程中,会使用普遍查找规则和ADL规则对非依赖型名称进行查找。另外,非受限的依赖型函数名称(这里的依赖型是指函数的参是依赖型的)会先使用普通查找规则进行查找,但只是把查找结果保存起来,并不会试图进行重载解析过一一这是在第2阶段的查找完成之后才进行的。
一个翻译单元通常会包含同个实例的多个POI。对于类模板实例而言,在每个翻译单元中,只有首个POI会被保留,而其他的POI则被忽略。对于非类型实例而言,所有的POI都会被保留。然而,对于上面的任何一种情况,ODR原则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 当遇到POI的时候,(编译器要求)相应模板的定义必须是(基于某种方式)可见的。对于类特化而言,这意味着:在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的。对于非类型的POI而言,也可能会采取上面的方式:当我们通常会把非类型模板的定义放在一个头文件中,然后在需要使用该定义的时候,把这个头文件#include到这个翻译单元中。这种处理模板定义的源模型就是我们前面所谈到的包含模型,也是目前为止最广泛的实现方式。对于非类型POI,还存在另一种实现方法:使用export关键字来声明非类型模板,而在另一个翻译单元中定该非类型模板。这就是我们前面所谈到的分离模型。 10.3.4 跨翻译单元查找模板中的名称分两阶段查找: 第1阶段发生在解析模板(也就是说,C++编译器第1次看到模板定义)的时候。在这个过程中,会使用普通查找规则和ADL规则对非依赖型名称进行查找,但只是把查找结果保存起来,并不会试图进行重载解析过一一这是在第2阶段的查找完成之后才进行的。 第2阶段发生在产生POI(实例化点)的时候。在这一点上,会使用普通查找规则和ADL规则来查找依赖型受限名称。而依赖型非受限名称(它已经在第1阶段使用普通查找规则查找了一次)则只(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段普通查找规则查找了一次)则只(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段普通查找规则查找了一次)则只(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段普通查找规则查找了一次)则只(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段普通查找规则查找了一次)则只(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段普通查找规则查找了一次)则只(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段普通查找规则查找了一次)则只(注意,只)使用AD规则进行查找,然后都可以由这种证明的证明,是是被证明的证明的证明的证明的证明的证明的证明的证明的证明的证明的证明的证明的证明的证
一个翻译单元通常会包含同个实例的多个POI。对于类模板实例而言,在每个翻译单元中,只有首个POI会祝保留,而其他的POI则被忽略。对于非类型实例而言,所有的POI都会被保留。然而,对于上面的任何一种情况,ODR原则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 当遇到POI的时候,(编译器要求)相应模板的定义必须是(基于某种方式)可见的。对于类特化而言,这意味着:在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的。对于非类型的POI而言,也可能会采取上面的方式:当我们通常会把非类型模板的定义放在一个头文件中,然后在需要使用该定义的时候,把这个头文件非向clude到这个翻译单元中。这种处理模板定义的源模型就是我们前面所谈到的包含模型,也是目前为止最广泛的实现方式。对于非类型POI,还存在另一种实现方法:使用export关键字来声明非类型模板,而在另一个翻译单元中定该非类型模板。这就是我们前面所谈到的分离模型。 10.3.4 跨翻译单元查找模板中的名称分两阶段查找: 第1阶段发生在解析模板(也就是说,C++编译器第1次看到模板定义)的时候。在这个过程中,会使用普通查找规则和ADL规则对非依赖型名称进行查找。另外,非受限的依赖型函数名称(这里的依赖型是指函数的参是规则和ADL规则对非依赖型名称进行查找,是把查找结果保存起来,并不会试图进行重载解析过一一这是在第2阶段的查找完成之后才进行的。 第2阶段发生在产生POI(实例化点)的时候。在这一点上,会使用普通查找规则和ADL规则来查找依赖型资限名称。而依赖型非受限名称(它已经在第1阶段使用普通查找规则查找了一次)则只(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段普通查找所获得的结果,组成一个候选函数集合,然后们助于重载解析,从该集合中选出(最佳的)被调用的函数。
一个翻译单元通常会包含同个实例的多个POI。对于类模板实例而言,在每个翻译单元中,只有首个POI会被保留,而其他的POI则被忽略。对于非类型实例而言,所有的POI都会被保留。然而,对于上面的任何一种情况,ODR原则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 当遇到POI的时候,(编译器要求)相应模板的定义必须是(基于某种方式)可见的。对于类特化而言,这意能着:在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的。对于非类型的POI而言,也可能会采取上面的方式:当我们通常会把非类型模板的定义放在一个头文件中,然后在需要使用该定义的时候,把这个头文件中,C的实现方式:这种处理模板定义的源模型就是我们前面所谈到的包含模型,也是目前为止最广泛的实现方法:使用export关键字来声明非类型模板,而在另一个翻译单元中定该非类型模板。这就是我们前面所谈到的分离模型。 10.3.4 跨翻译单元查找模板中的名称分两阶段查找: 第1阶段发生在解析模板(也就是说,C++编译器第1次看到模板定义)的时候。在这个过程中,会使用普通查找规则和ADL规则对非依赖型是指函数的参是依赖型的)会先使用普通查找规则进行查找,但只是把查找结果保存起来,并不会试图进行重载解析过一一这是在第2阶段的查找完成之后才进行的。 第2阶段发生在产生POI(实例化点)的时候。在这一点上,会使用普通查找规则和ADL规则来查找依赖型受限名称(它已经在第1阶段使用普通查找规则查找了一次)则只(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段使用普通查找规则查找了一次)则只(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段使用普通查找规则查找了一次)则只(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段使用普通查找规则看找了一次)则只(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段使通查找所获得的结果,组成一个候选函数集合,然后们助于重载解析,从该集合中选出(最佳的)被调用的函数。 下面用一个例子来解析所描述的一些概念: 1. 关于包含模型的简单例子:
一个翻译单元通常会包含同个实例的多个POI。对于类模板实例而言,在每个翻译单元中,只有首个POI会将保留,而其他的POI则被忽略。对于非类型实例而言,所有的POI都会被保留。然而,对于上面的任何一种的况,ODR原则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 当遇到POI的时候,(编译器要求)相应模板的定义必须是(基于某种方式)可见的。对于类特化而言,这意味着:在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的。对于非类型的POI而言,也可能会采取上面的方式:当我们通常会把非类型模板的定义放在一个头文件中,然后在需要使用该定义的时候,把这个头文件#include到这个翻译单元中。这种处理模板定义的源模型就是我们前面所谈到的包含模型,也是目前为止最广泛的实现方式。使用export关键字来声明非类型模板,而在另一个翻译单元中定该非类型模板。这就是我们前面所谈到的分离模型。 10.3.4 跨翻译单元查找 模板中的名称分两阶段查找:第1阶段发生在解析模板(也就是说,C++编译器第1次看到模板定义)的时候。在这个过程中,会使用普通查找规则和ADL规则对非依赖型名称进行查找。另外,非受限的依赖型函数名称(这里的依赖型是指函数的多是依赖型的)会先使用普通查技规则进行查找,但只是把查找结果保存起来,并不会试图进行重载解析过一一这是在第2阶段的查找完成之后才进行的。 第2阶段发生在产生POI(实例化点)的时候。在这一点上,会使用普通查找规则和ADL规则来查找依赖型受限名称。而依赖型非受限名称(它已经在第1阶段使用普通查找规则直找了一次)则只(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段普通查找所获得的结果,组成一个候选函数集合,然后们助于重载解析,从该集合中选出(最佳的)被调用的函数。 下面用一个例子来解析所描述的一些概念: 1、关于包含模型的简单例子: ***********************************
一个翻译单元通常会包含同个实例的多个POI。对于类模板实例而言,在每个翻译单元中,只有首个POI会将保留,而其他的POI则被忽略。对于非类型实例而言,所有的POI都会被保留。然而,对于上面的任何一种的况,ODR原则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 当遇到POI的时候,(编译器要求)相应模板的定义必须是(基于某种方式)可见的。对于类特化而言,这意味着:在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的。对于非类型的POI而言,也可能会采取上面的方式:当我们通常会把非类型模板的定义处放在一个头文件中,然后在需要使用该定义的时候,把这个头文件并include到这个翻译单元中。这种处理模板定义的源模型就是我们前面所谈到的包含模型,也是目前为止最广泛的实现方式。对于非类型模板。这就是我们前面所谈到的分离模型。 10.3.4 跨翻译单元查找 模板中的名称分两阶段查找: 第1阶段发生在解析模板(也就是说,C++编译器第1次看到模板定义)的时候。在这个过程中,会使用普询查找规则和ADL规则对非依赖型名称进行查找。另外,非受限的依赖型函数名称(这里的依赖型是指函数的参是依赖型的)会先使用普通查找规则和ADL规则对非依赖型名称进行重拨,但只是把查找结果保存起来,并不会试图进行重载解析过一一这是在第2阶段的查找完成之后才进行的。 第2阶段发生在产生POI(实例化点)的时候。在这一点上,会使用普通查找规则和ADL规则来查找依赖型受限名称。它已经在第1阶段使用普通查找规则和自数,只见(注意,只)使用AD规则进行查找,然后把ADL的查找结果结合第1阶段普通查找所获得的结果,组成一个候选函数集合,然后的助于重载解析,从该集合中选出(最佳的)被调用的函数。 下面用一个例子来解析所描述的一些概念: 1. 关于包含模型的简单例子:
一个翻译单元通常会包含同个实例的多个POI。对于类模板实例而言,在每个翻译单元中,只有首个POI会被保留,而其他的POI则被忽略。对于非类型实例而言,所有的POI都会被保留。然而,对于上面的任何一种的况,ODR原则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 当遇到POI的时候,《编译器要求》相应模板的定义必须是(基于某种方式)可见的。对于类特化而言,这意味着:在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的。对于非类型的POI而言,它可能会采取上面的方式:当我们通常会把非类型模板的定义放在一个头文件中,然后在需要使用该定义的时候,把这个头文件并Include到这个翻译单元中。这种处理模板定义的源模型就是我们前面所谈到的包含模型,也是目前为止最广泛的实现方式。对于非类型POI,还存在另一种实现方法:使用export关键字来声明非类型模板,而在另一个翻译单元中定该非类型模板。这就是我们前面所谈到的分离模型。 10.3.4 跨翻译单元查找 模板 (也就是说,C++编译器第1次看到模板定义)的时候。在这个过程中,会使用普通查找规则和ADI规则对非依赖型名称进行查找,另外,非受限的依赖型路数名称(这里的依赖型是指函数的参是依赖型的)会先使用普通查找规则进行查找,但只是把查找结果保存起来,并不会试图进行重载解析过一。这是在第二阶段的查找完成之后才进行的。第2阶段发生在产生POI(实例化点)的时候。在这一点上,会使用普通查找规则和ADI规则来查找依赖型受限名称(它已经在第1阶段使用普通查找规则查找了一次)则只(注意,只)使用ADI规则进行查找,然后把ADI的查找结果结合某价的结果,组成一个候选函数集合,然后们助于重载解析,从该集合中选出(最佳的)被调用的函数。 下面用一个例子来解析所描述的一些概念: 「文字 (中)
一个翻译单元通常会包含同个实例的多个POI、对于类模板实例而言,在每个翻译单元中,只有首个POI会排保留,而其他的POI则被忽略。对于非类型实例而言,所有的POI都会被保留。然而,对于上面的任何一种协 况,CDR原则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 10.3.3 包含模型与分离模型 10.3.3 包含模型与分离模型 12遇POI的时候,(编译程要求)相应模板的定义必须在它的POI之前就已经是可见的。对于类特化而言,这一意味者。在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的。对于非类型的POI而言,也可能会采取上面的方式:当我们通常会把非类型模板的定义放在一个头文件中,然后在需要使用该定义的时候,把这个头文件,们也但到这个翻译单元中,这种处理模成定义的源模型数是我们前面所谈到的包含模型,也是目前为止量厂泛的实现方式。使用export关键字来声明非类型模板,而在另一个翻译单元中定该非类型模板,这样是我们前面所谈到的分离模型。 10.3.4 跨翻译单元查找模板中的名称分两阶段查找: 第1阶段发生在解析模板(也就是说,C++编译器第1次看到模板定义)的时候。在这个过程中,会使用普通查找规则和ADU规则对非依赖型名模技有通关。另外,非受限的依赖型函数名称(这里的依赖型是指函数的参差化物度的查找规则进行查找,但只是把查找结果保存起来,并不会试图进行重截解析过一一定基在第2阶段的查找规划进行查找,但只是把查找结果保存起来,并不会试图进行重整解析就是一定基在第2阶段的查找规划会的管线线集结合面,的段量通查找规则通常发行一次)则只(注意,只)使用AD规则进行查找、然后把ADL的包括线线集结合面,前段音通查找规则通常及了一次)则只(注意,只)使用AD规则进行查找、然后是ADL的包括线线集结查,然后的助于重整解析,从该集合中选出(最佳的)被调用的函数。 下面用一个例子来解析所描述的一些概念: 1. 美于包含模型的简单例子: 「(()) // 编译,找不到到 // (2):fisint>(int)的POI // (2) // 编译,我不到到模板们的定义时,编译器注意中实现的是一个依赖型名称,为它的参数名称依赖于外部也是价格模板的扩展的多数(即实参的类型依赖于模型多数),因此,编译器会在(1)的图像
一个翻译单元通常会包含同个实例的多个POI,对于类模板实例而言。在每个翻译单元中,只有首个POI会相保留。而其他的POI则被忽略。对于非类型实例而言。所有的POI都会被保留。然而,对于上面的任何一种情况,ODR原则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 当遇到POI的时候,(编译器要求)相应模板的定义必须是(基于某种方式)可见的,对于类特化而言,这结
一个翻译单元通常会包含同个实例的多个POI、对于类模板实例而言、在每个翻译单元中,只有首个POI全体图。而其他的POI则都会要求:对缘留的任何一个POI处所出现的同种实例化体,都必须是等价的。 10.3.3 包含模型与分离模型 当遇到POI的时候,(编译器要求)相应模板的定义必须在它的POI之前就已经是可见的,对于类特化而言,这意味者,在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的,对于类型的POI而,这是味者,在同个翻译单元中,类模板的定义必须在它的POI之前就已经是可见的,对于类型的POI而言,这可能会取上面的方法;当我们需常会电头发现特核的定义必须是有关于一个头文件中,然在在强硬使用该交叉的时候,把这个头文件中的比如自然一个全球中,实有是可以是一个大型件中,然在是不要使用表交叉的时候,把这个头文件中的比如是现分,可能全取上面前了这的实现方式。 对于非类型POI,还存在另一种实现方法。使用export关键字来声码非类型模板,而在另一个翻译单元中定该非类型模板。这就是我们前面所谈到的分离模型。 10.3.4 得翻译单元重发
一个舒泽单元鸿常合包含同个实例的多个POI、对于类模板实例而言,在每个翻译单元中、只有首个POI公编码。 而其他的POI则被安慰。对于非难型实例而言,所有的POI都会按保留。然而、对于上面的任何一种的20、00 RR则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等例的20、20、20 RR则都会要求:对保留的任何一个POI处所出现的同种实例化体,都必须是等例的20、20、20、20、20、20、20、20、20、20、20、20、20、2
一个翻译单元通常会包含同个实例的多个POI,对于类模板实例而言,在每个翻译单元中,只有百个POI会解晶,而其他的POI则被数据,对于共取的任何一种POI分析。那必须是等的效果;对保留的任何一个POI分析出现的同种实例化体,都必须是等的为。 10.3.3 包含模型与分离模型
一个部译单元通常负色高向个实例的多个POI,对于果模板实例而言,在每个部译单元中,只有首个POI公院院。而其他的POID能数级,对于非类型实例而言,所有的POIO能表域保留。然而,对于上面的任何一种的 《 ORA《 DAN LAN LAN LAN LAN LAN LAN LAN LAN LAN L
一个部译单元途常会创合同个实例的多个POI。对于类模核实例而言。在每个部译单元中,只有首个POI公院展。而其他的POI则能多起。对于非类型实例而言。所有的POI公司报记的。如果是有的。对于上面的任何一种POI公司报记的。如果是有的。对于上面的任何一种POI公司报记的。如果是有的。对于类特化而言,这个这种是一个的一种POI公司报记的。如果是有的。对于类特化而言,这个这种是一个可以的时候。他们要要的一个可以是一个可以是一个可以是一个可以是一个可以是一个可以是一个可以是一个可以是
一个制译单元进常会包含两个实例的多个PO1、对于吴杲泰来则而言、在每个制译单元中,只有首个PO1公保信。而其他的PO11则表面表、对十年地类以表现而言、外有的PO1的全接保密、忽而,并于上面的任何一种经历。ODR原则的参考来:对常高的任何一个PO1处所出现。同年实验,这个对于生物的。 10.3.3 包含模型与含度模型 当点的PO10时候,《海祥每年大 州位原始的主义公务是(在于某种方式)可见的,对于主体的企而。这一点是是一个一个20处所中以是一个文件中,然后企业要使所适至公时可以完全是一个一个1000年以及一个文件中,然后企业要使所适至公时可以完全是一个一个1000年以及一个文件中,然后企业要使所适至公时对于某些PO1、中间已经会设定一种等于一、没种位是两个人文件中,然后企业要使所适至公时对于某些PO1、中间已经会设定一种等于一、这种位数据处分对那些基础的时间。使用于这些专用的上面一定的现象。 此至个全种技术,这是是一个一种实现方法。使用在20个人被手中,然后企业等使所适至公时对于某些PO1、有一个是一种实现方法。使用在的总域是一个人是一种实现方法。使用在的总域是一个人是一种实现方法,使用全型分域的企业发现,几个人们对中,会使用自动主线和设定的自动主线,另外,非安聚的体理是更数别的全发。这种对自由的主线和自由主线规则实现方式。对于实现的自由主线和实现方式,对于公式即还有主动的对一定是在非常的企业有关于一次,现于《这种自动主线和发行主义》,另外,非安聚的体理是是指控制的一一定是在非常的企业的主线,而是是一个数据的企业是是是一个人们的企业是是一个人们的企业是是一个人们的企业是是一个人们的企业是是一个人们的企业是一个一个人们的企业是一个人的企业是一个人们的企业是一个一个一个人们的企业是一个一个人们的企业是一个一个是一个人们的企业是一个人们的企业是一个一个人们的企业是一个人们的企业是一个人们的企业的企业是一个人是一个人们的企业是一个一个企业的企业是一个人们的企业是一个人们的企业是一个一个人们的企业是一个一个企业的企业是一个一个企业的企业是一个一个企业的企业是一个一个企业的企业是一个一个企业的企业是一个一个企业的企业是一个企业的企业是一个企业的企业是一个企业的企业是一个企业的企业的企业是一个企业的企业是一个企业的企业是一个企业的企业是一个企业的企业是一个企业的企业是一个企业的企业是一个企业的企业是一个企业的企业是一个企业的企业是一个企业的企业是一个企业的企业是一个企业的企业是一个企业是一个企业的企业的企业是一个企业的企业是一个企业的企业是一个企业的企业的企业的企业的企业是一个企业的企业是一个企业的企业是一个企业的企业的企业的企业是一个企业的企业是一个企业的企业的企业是一个企业的企业,企业是一个企业的企业的企业是一个企业的企业是一个企业的企业是一个企业的企业是一个企业的企业的企业是一个企业的企业是一个企业的企业的企业是一个企业的企业的企业的企业的企业的企业的企业的企业的企业的企业的企业的企业是一个企业的企业的企业的企业的企业的企业的企业的企业的企业的企业的企业的企业的企业的企
一个翻译单元进席会包含码个实例的多个PO1。对于吴昊极来例而后。在每个翻译单元中,只有首个PO1公债保,而更新的PO1则被超级。对于非典型之间而言。所有的PO1的合金储保施,实现,对于上面的任何一种经验,ORICE则随着企業。对成自由任何一个PO1处代出现的国种技术任务。实现,对于上面的任何一种经验的PO1分间,(编译器要以:标应的任何一个PO1处代出现的国种技术任务。实现,对于电影的元、可是到的国际。(编译器要以:标应的任何一个PO1处代出现的国种技术任务。这一是强硬的10.03.3 有线性型分离是一个关键,并未被的证义分别是(是于某种方式)可见的,对于未实的形式。这里是那分元。并即仍通常之不中,实现是是一个人类性,然后在需要进现该区义的理论,是一个人工作,然后需要进现该区义的对于主要的2、1分在分子中心比较的影响,是一个人工作,是一个人工作,但是一个人工作的一种实现方式。对于未实的几个工作任何一种实现方式。对于未实的几个任任中,一个人工作,是一个人工作的工作企业方式。这一是自己的方式。这一是自己的方式,这种类型性极,这是一个人工作的一种关于未发的。10.3 4 有线单单元 是转使 (也就是说,C++错符器单以是到增级定义)的时候。在这个过程中,会实现于这种政策的成员,并不是一种实现方式,这种主要的特别是一个人工作的。另外,并更多的的政策,并不会说此对重要的主意规则是一个人工作的。另外,并更多的对于是一个人工作的,是有规则的主意规则是一个人是有一种实现的主意规则是一个人是有一种实现的主意规则是一个人是不是一种实现的主意规则是一个人是通数集合,然后的工程处理的主要规则是一个代意通数集合,然后的工程处理的主要规则是一个代意通数集合,然后的工程处理的主要规则是一个人是通数集合,然后的工程处理的主要规则是一个人是通数的主要规则是一个人是通数生态。然后的工程分对处于一个例子来取得对能到的一些态念: 10.4 117.5 117.6 117.5 117.
一个部洋申八選常会包含同个实例的多个POI、对于类属核类例而高、在何个部洋单元中,只有音中POI会解解。而其他的POI则核定数。对于由原它何一种情况。而我他的POI则核定数。对于由原它何一种情况。而我他的POI则核定数。对于由原它何一种情况。而我他的POI则核定数。对于是原它何一种情况。可以是原因的原则,对于上国的它何一种情况。可以是原因的原则,对于是原心的自己。 10.3.3 含态性型 50.94世型 11.0.3.3 含态性型 50.94世元 11.0.3.3 含态性型 50.94世元 11.0.3.3 含态性型 50.94世元 11.0.3.3 电影中学中心中心原则 12.94世元 11.0.3.3 电影中等中心中,这种处理域检定义的原体型 12.94世元 12.0.3 电影中学中心中心 12.94世元 12.0.3 电影中学中心 12.94世元 12.0.3 电影中学中心 12.94世元 12.0.3 电影中学中心 12.94世元 12.0.3 电影中学中心 12.94世元 12.0.3 电影中心 12.94世元 12.0.3 电影中心 12.94世元 12.0.3 电影中心 12.94世元 12.0.3 电影中心 12.94世元 12.0.3 电影响应 12.0
一个部学生元语宗会包含向个实例的多个POL,对于英格敦实例的点。布色个部学中元中,只有首个POL会情况,而起影中OUI的或多观,对于其实实例的方。
一个商伴並用途供養包含的个类例的多个PO1、对于类似及美丽而言、在每个部件单元中、只有有个PO1会解析,所有的PO1余情况,仍可以的是多数。对于类型级而高,所有的PO1余情况,对于其他的16一种体。 《公司及前期检查的的股份数。以于有类型级而高,所有的PO1余情况,然后,对于其他的任一种体。 《公司及前期检查分别的基础。以于有类型级而一个PO10余情况就的种类似化。 然后是并依然一种大学中,以下有关的一种大学中,是一个专人的一个文学中,对于其类型的PO10元件,这种可以不是一个大学中,以下有关键的PO10余代度,但还是我们用的发生的一个PO10余代型的不是一个大学中,对于其实型的PO10元件,并可是关键性的一种关于,实验检验更多少的各种一个文学中,对于其实型的PO10元件,是一个专人文学中,对于其实型的PO10元件,并不是一个文学中,对于其实型的可以不是一个文学中,对于其实型的可以不是一个文学中,对于其实型的一个专人文学中,对于其实型的,是一个专人文学中,对于其实型的,是一个专人文学中,对于其实型的,是一个专人文学中,实现了这种形式,是一个专人文学中,不是一个文学中,实现了这种形式,是一个专人文学中,不是一个文学中,实现的是一种技术的企业的主要。 10.3.4 自然经验中的任何分别的是可以,在一个专人工作的一个专人工作的,是一个专人工作的一个专人工作的一个专人工作的一个专人工作的一个专人工作的一个专人工作的一个专人工作的一个专人工作的一个专人工作的一个专人工作的一个专人工作的一个专人工作的一个专人工作的一个专人工作的一个专人工作的一个专人工作的工作。 10.1 在 2.1 是一个专人工作的工作,是一个专人工作的一个专人工作的工作。 10.2 是一个可以不同的有能是一个专人工作的一个专人工作的一个专人工作的工作。 10.3 是一个可以不同的有能是一个专人工作的工作。 10.3 是一个可以不同的有能是一个专人工作的工作。 10.3 是一个可以不同的有能是一个专人工作的工作。 10.3 是一个可以不同的有能是一个专人工作的工作。 10.3 是一个可以不同的工作。 10.3 是一个专人工作的工作。 10.4 是一个专人工作的工作。 10.5 是工业的工作。 10.4 用类文型之,是一个专业工作的工作。 10.5 是工业的工作。 10.5 是工作的工作。 10.5 是工作的
一个创促基本层常含色彩的"各种的"与例的多个POL,对于最大的类似的。 然后,对于上面的平台中的经验解析,可以由于这种POL的经验的专业,对性与的作为一种创办。 PREMPORE 类似,而是这种企业,然后,对于上面的平台中心特别。 PREMPORE 类似,而是这种企业,是是是是是一个是是是是一个是是是是一个是是是是是一个是是是是是一个是是是是是一个是是是是是一个是是是是是一个是是是是是是
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・
一种形型工资产会协会用产品资金的企业。
一种经历工程的企业的国际,实现是实现的情况,对于这种经历,有效的公司等级,是一个分型是实现,并不同的任何一种 12. OODE的服务会实验,对特别的证明一个FOOL的情况的用户的公司会观,而是一种可以一种 12. OODE的服务会实验,对特别的证明一个FOOL的情况的用户的公司会观,而是一种可以一种 12. OODE的服务会实验,对特别的证明一个FOOL的情况的用户实现化分表。但是一种更多的。13. 可以是一种更多的。13. 可以是一种更是一种更多的。13. 可以是一种更多的。13. 可以是一种更多可以是一种更多的。13. 可以是一种更多的。13. 可以是一种更多的。13. 可以是一种更多的。13.
一一种生产则如果各种公司中国和特别的。为于中国的经济中、在2000年的,并不是1900年代的。由此的的经验,用的经验的经验,对于1900年代种,2003年的期间的共和,对自由的任任一个P012年代的国际,并不是1900年代的一种,2003年的期间的关系,对自由的任任一个P012年代的国际,是1900年代,1900年的国际,是1900年代,1900年的国际,1900年的国际,1900年的国际,1900年的国际,1900年的国际,1900年的国际,1900年的国际,1900年的国际,1900年的国际,1900年的国际,1900年的国际,1900年的国际,1900年的国际,1900年的国际,1900年的国际,1900年的国际的企业,1900年的国际企业,1
一番中华中央地区企员经济个全的设施中心,并不是他们的企业,但如此的企业的企业,不是个有关中心,现在个的公司经验。 《日本的政治经验等》,但我们的经济一种的总域证明中华中的总域证明的企业,在总域是自然。 《日本的政治经验等》,但我们的经济一种的总域证明的专种。例如,这个现在是一个成为,就会是一个公司。 《日本的政治经验》,但是是一个中的总域证明的专种。例如,对于其他的。可以为,对于其他的。 《日本的政治经验》,是是是一个专种的企业的企业,对于是一个政治中,或是但是有自然的可以,对于其他的。 《日本的政治经验》,是是是一个政治和证明的专种。 《日本的政治经验》,是是一个政治的企业是一个政治中,或是但两种政治经验,而且是一个政治中,或是但是一个政治中,或是是一个政治的企业,对于是一个政治的企业,对于是一个企业,对于是一个企业,是一个政治的企业,对于是一个企业,是一个政治是一个政治的企业,对于是一个企业,是一个政治的企业,对于是一个企业,是一个政治是一个企业,是一个政治是一个企业,是一个专种中不中的企业,就是一个政治是一个企业,是一个专种中不中的企业,就是一个政治是一个企业,是一个专种中不中的企业,就是一个专业,是一个专种中不中的企业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业,是一个专业是一个一个企业。这一个企业,是一个专业的企业,是一个专业的企业,是一个专业的企业,是一个专业的企业,是一个专业的企业,是一个专业的企业,是一个专业的企业,是一个专业的企业,是一个企业是一个企业是一个企业是一个企业是一个企业是一个企业是一个企业是一个企业
一番時代中央部の公民会介での対象を介では、日本書類が表現において、日本音が由来では、「大田 (中国) (中国) (中国) (中国) (中国) (中国) (中国) (中国)
- 中国地域中国企業企会的企业。例如,如果的股份的人。对于和股份的股份的人。但是是有效的人。但是是是有效的人。但是是有效的人。但是是是有效的人。但是是是有效的人。但是是是有效的人。但是是是有效的人。但是是是是是是是是是是是是是是是是是是是是是是是是是是是是是是是是是是是是
一般語彙者是中国出版的。如于有些企业的。
一个创新的企业。并且他们的证明的,不是可以会现的。我们的证明的证明。然后,对于正的的证明。然后,对于正面的的证明。我们的现在则是的,不是可以会现的。我们的现在是的一个现在,我们是一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们就是我们的一个现在,我们就是我们的一个现在,我们就是我们的一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们们是我们的一个现在,我们们是我们们是我们们是我们们是我们的一个现在,我们们是我们们是我们们是我们们是我们们是我们们是我们们是我们们是我们们是我们们
学研究中央社会の研究内で国际的で対しませな。1990日の日本地ので、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019日の日本のでは、2019
一番製造工業の指数の影響の「中華の企業」というであっています。 2. の地の利用が関係が、対すを関係が関係であった。「おからのはおから、他の大きな情報を関係。 2. の地の利用が支援、が開発的によったのは利用である。例如の対象を関係。 3. の地の利用が支援、が開発的によったのは利用である。例如の対象を関係。 3. の地の利用が支援、が開発的によったの影響を関係していまった。 3. の地の対象を対象を対象を対象を対象を対象を対象を対象を対象を対象を対象を対象を対象を対
- 一会が担当。 地名日本の日本の日本の日本の日本の日本の日本の日本の日本の日本の日本の日本の日本の日



template<typename T>

template<typename E, int N>

void f2(E(&)[N]); // 数组,引用参数

void f(double*);

template<typename T1, typename T2, typename T3> void f3(T1 (T2::*) (T3*)); // 类的成员函数指针

// 演绎T为int**.

f3(&S::f); // 演绎T1=void,T2=S,T3= double.

复杂的类型声明都是产生自(比它)基本的构造(例如指针、引用、数组、函数声明子(declarators);成 员指针声明子、template-id等);匹配过程是从最顶层的构造开始,然后不断递归各种组成元素(即子构 造)。我们可以认为:大多数的类型声明构造都可以使用这种方式进行匹配,这些构造也被称为演绎的上下

(2) 除了非类型参数之外,模板参数还包含其他成分的非类型表达式。例如,诸如S<I+1>的类型名称就不 能被用来演绎I(如果是S<I>就可以)。另外,我们也不能通过匹配诸如int(&)[sizeof(S<T>)]类型的参数

具有这些约束是很正常的,因为通常而言,尽管有时候会很容易地忽略受限的类型名称,但演绎过程并不是唯 一的(甚至不一定是有限的)。而且,一个不能演绎的上下文并没有自动地表明:所对应的程序就是错误的,

存在两种特殊情况,其中用于演绎的实参-参数对(A,P)并不是分别来自于函数调用的实参和函数模板的参 数。第1种情况出现在取函数模板地址的时候。在这种情况下,P是函数模板声明子的参数化类型(即下面的f

在上面的代码中,P就是void(T, T),而A是void(char, char)。用char替换T,该演绎过程是成功的。另外,pf

在这种情况下,实参-参数对(A, P)涉及到我们试图进行转型的实参和转型运算符的返回类型。下面的代码

在此,我们试图把S转型为int(&)[20];因此,类型A为int[20],而类型P为T[N]。于是,用类型int替换T,

通常,模板演绎过程会试图找到函数模板参数的一个匹配,以是参数化类型P等同于类型A。然而,当找不到

(1) 如果原来声明的参数是一个引用参数,那么被替换的P类型可以比A类型多一个const或者volatile限定

(3) 当演绎过程不涉及到转型运算符模板的时候,被替换的P类型可以是A类型的基类;或者当A是指针类型

只有在精确匹配不存在的情况下,才会出现这种宽松的匹配。即使这样,只有在前面添加的几种转型中能找到

模板实参演绎只能应用于函数模板和成员函数模板,是不能应用于类模板的。另外,对于类模板的构造函数,

如例子所示,缺省调用实参是可以依赖于模板参数的。但是,只有在没有提供显式实参的情况下,才会实例化

// 因为T=S,所以T()就是无效的了。于是缺省调用实参T()也就不需要进行实例化,因为已经提供了一个显

对于缺省调用实参而言,即使不是依赖型的,也不能用于演绎模板实参。这意味着下面的C++程序是无效

●推荐

刷新评论 刷新页面 返回顶部

posted @ 2016-01-25 14:30 小天_y 阅读(2100) 评论(0) 编辑 收藏 举报

导反对

一种替换,并且借助这种替换可以匹配A类型和P类型,演绎过程才能是成功的。

// 错误:不能从构造函数的调用实参12演绎类模板参数T

和普通函数一样,在函数模板中也可以指定缺省的函数调用实参。例如:

这种依赖型的缺省实参——这也使得下面例子有效的一条规则:

(2) 如果A类型是指针类型或者成员指针类型,那么它可以进行限定符转型(就是说,添加const或者

时,P可以是一个指针类型,它所指向的类型是A所指向的类型的基类。如:

(1) 受限的类型名称。例如,一个诸如Q<T>::X的类型名称不能被用来演绎模板参数T。

的类型),而A是被赋值(或者初始化)的指针(即下面的pf)所代表的函数类型。例如:

template<typename T, int N> operator T[N]&();

f2(b); // 演绎E为bool, N为42.

文。然而,某些构造就不能作为演绎的上下文,如:

(上面两点不理解, why???)

11.3 特殊的演绎情况

template<typename T>

void (*pf)(char, char) = &f;

被初始化为"特化f<char>"的地址。

另一种特殊情况和转型运算符模板一起出现。如:

void f(T, T);

class S

public:

清楚地说明了这种情况:

void g(S s)

f(s);

11.4 可接受的实参转型

template<typename T>

template<typename T> class D : public B<T>

void g(D<long> dl)

void f(int (&)[20]);

用20替换N之后,该演绎就是成功的。

这种匹配的时候,下面的几种变化就是可接受的:

volatile限定符),转化为被替换的P类型;

template<typename T> void f(B<T>*);

也不能根据实参来演绎类模板参数。如:

 $S(T b) : a(b) {}$

template <typename T>

public:

private: Ta;

11.6 缺省调用实参

template<typename T>

*loc = val;

void init(T* loc, T const& val = T())

f(&dl); // 成功演绎: 用long替换T

{

} ;

}

符。

{ };

{ } ;

};

}

};

式参数

}

的:

class S {

public:

ss(0, 0);int main()

S(int, int);

init(&s, S(7, 42));

template<typename T> void $f(T x = 42) \{ \}$

分类: C++ Template

<u>小天_y</u> 关注 - 63 粉丝 - 96

好文要顶

+加关注

f<int>(); // 正确: T= int

关注我

« 上一篇: <u>C++ template —— 模板中的名称(三)</u> » 下一篇: <u>C++ template —— 模板特化 (五)</u>

f(); // 错误: 不能根据缺省调用实参42来演绎T

收藏该文

int main()

 $S \times (12);$

class S

11.5 类模板参数

class B

或者前面分析的参数不能再次进行类型演绎。

void f1(T*)

class S

public:

void g(int*** ppp)

bool b[42];

f1(ppp);

{

} ;

}

类演绎T。