# 术语表

本术语表是我们在此书中用到的重要技术名词的汇编。查看[StroustrupGlossary]以获取C++程序员使用的更具体有综合性和常用的名词。

**abstract class（抽象类）**

一种无法创建具体对象（实体）的类。抽象类用于单一类型中收集不同类的共同属性或定义一个聚合接口。因为抽象类作为基类使用，缩写ABC有时被用于*abstract base class*

**ADL**

*Argument-dependent lookup*（**参数依赖查找**）的缩写。ADL是一个过程，他在函数出现之处通过某种方式与函数参数有关的类和名称空间内查找一个函数的名称。由于历史原因，它有时被称为*extended Koenig lookup*或*Koenig lookup*

**alias template（别名模板）**

一个代表一族模板的构建。它定义了一个通过替换确定实体的模板参数生成实际类型别名的模式。一个别名模板可以是一个类成员。

**angle bracket hack<尖括号hack>**

一个需要编译器接收两个连续的>>符号作为两个封闭尖角括号的C++特性。例如，尖角括号技术造成了vector<list<int>>被当作单独的vector<list<int>>。

**angle brackets（尖括号）**

符号<和>用作界定符，而非小于或大于操作符。

**ANSI**

*America National Standard Institude*（美国国家标准协会）缩写。他是一个致力于为所有事物提供明确标准的非营利性民间团体。详见INCITS。

**argument（参数）**

替代了可编程参实体的一个参数的一个值（广义上）。例如，在函数调用ab(-3)中，参数是-3。在某些编程社区中，参数也成为实际参量（此处参量称为形式化参量）。请同时查看模板参数（*template argument*）。

**argument-dependent look（参数依赖查找）**

详见**ADL**。

**class（类）**

一类对象的描述。类定义了任何该类型对象的一系列特性。这包括数据（属性，数据成员）及其操作（方法，成员函数）。

**class template（类模板）**

代表了一组类的一个构建。它定义了一个通过替换确定实体的模板参数生成实际类的模式。类模板有时也称为*parameterized class*(参数化类)。

**class type（类类型）**

一个声明中带有class,struct或union的C++类型。

**collection class（集合类）**

一个用于管理一组对象的类。在C++中，集合类也成为容器。

**compiler（编译器）**

一个将翻译单元中的源码翻译为目标码（带有允许连接器决定决定跨翻译单元引用的符号注释的机器码）的程序或者库组件。

**complete type（完整类型）**

任何非不完整的类型：一个定义了的类，一个具有已知大小和完整元素的数组，一个具有已定义底层类型的枚举类型，以及除了void外的任何基础数据类型（选择性的带有const和/或volatile）。

**concept**

一个能被用于一个或更多模板参数的已命名集。请查阅附录E。

**constant-expression（常量表达式）**

一个编译器在编译时刻可以计算出值的表达式。我们有时称之为true constant来避免与constant expression（无连字符）混淆。后者包括了一个编译器在编译时刻不总是可以计算出值的表达式。

**const member function（常量成员函数）**

常量或临时变量可以调用的成员函数，因为它通常并不修改\*this对象的成员。

**container（容器）**

详见**collection class。**

**conversion function（转换函数）**

一个定义了某个类如何隐式转换为另一个类型的对象的的特殊成员函数。它的声明形式为operator *type()*。

**conversion operator（转换操作符）**

转换函数的同义词。后者是标准术语，但前者也常用。

**CPP file**

一个变量和非内联函数被定位的文件。程序的大部分可执行代码通常被置于CPP文件中。它们被命名为CPP文件的原因是他们有后缀.cpp。但由于历史原因后缀也可以是.C，.c，.cc,或者.cxx。详见header file和 translation unit。

**CRTP**

curiously recurring template pattern的同义词。它指一个拥有X最为模板参数的基类衍生的类X的编码模式。

**curiously recurring template pattern**

详见CRTP。

**decay（衰退）**

将指针和函数转换为指针的隐式转换。例如，字符串字面值”Hello”具有类型char const[6],但在很多C++环境中，它隐式转换为char const\*的指针（指向该字符串的第一个字符）。

**declaration(声明)**

将一个名称导入或重导入C++范围的C++构建。详见definition

**deduction（推导）**

根据模板使用的上下文中隐式决定模板参数的过程。完整的术语是*template argument deduction*（模板参数推导）。

**definition（定义）**

一个使已声明实体的细节被指明的声明，如果是变量，它会强制为已声明实体的存储空间被保存

**dependent base class（依赖性基类）**

一个取决于模板参数的基类。获取依赖性基类的成员时应该格外小心。同时请看two-phase lookup（二阶段检查）。

**dependent name（依赖性名称）**

取决于模板参数的含义的名称。例如，当A或T是模板参数时，A<T>::x就是一个依赖性名称。如果某个调用中的任何参数的类型取决于模板参数，该调用中的函数名称也是依赖型的。例如，如果T是一个模板参数，f((T\*)0)中的f是依赖性的。然而，模板参数的名称不被认为是依赖性的。同时请看two-phase lookup。

**digraph（合成符）**

一个在C++代码中等价于另一个单一字符的两个连续字符的组合。合成符的目的是允许C++键盘输入的源码中缺少确定的字符。尽管它的使用相对较少，合成符<:会在一个左尖括号后跟随一个域解析操作符（::），缺失需要的间隔符空格的时候偶然形成。C++11引入了词汇性的规则在那些情况下使合成符的解释无效。

**EBCO（空基类优化）**

empty base class optimization的缩写。在任何“空”基类子对象未占用任何内存时大部分现代编译器采用的优化。

**empty base class optimization**

详见EBCO

**explicit instantiation directive（显式实例化引导）**

唯一目的时创建一个point of instantiation(POI)的C++构建。

**explicit specialization（显示特化）**

对一个已替代模板的声明或定义一个替代性的定义的构建。原（generic）模板被称为主模板。如果替代性定义取决于一个或多个模板参数，就被称为部分特化。否则就时全特化。

**expression template（表达式模板）**

一个表达式模板被用于表示一个表达式的一部分。模板本是个代表了一类特定的操作。模板参数代表了一类该操作使用的操作符。

**forwarding reference（向前引用）**

T是可推导的模板参数时，形式为T&&的两个术语之一的右值引用。特殊规则在不同的普通右值引用时不同（6.1节，91页）。该术语在C++17中引入并替代了*universal reference*，因为该类引用主要使用于向前对象。然而需要注意它不是自动向前。也就是，该术语没有描述它是什么而是它的典型用途。

**friend name injection（友名称插入）**

当函数名称的唯一声明是友声明时，使其可见的过程。

**full specialization（全特化）**

请看explicit specialization。

**function object（函数对象）**

可以使用函数调用分析调用的对象。在C++中，这些是指向函数的指针、具有重载操作符（）的类（请看functor）、和具有生成一个指向函数指针或函数引用的转换函数的类。

**function template（函数模板）**

一个表示一族函数的构建。它指明了通过确定实体替代模板参数从而能生成一个实际函数的模式。注意函数模板是模板而非函数。函数模板有时也称为parameterized function（参数化函数）

**functor（算子）**

一个具有重载过的操作符（）的类类型的对象，它可以使用函数调用分析来调用。这包括封闭类型的lambda表达式。

**glvalue（泛左值）**

一类生成存储值位置的表达式（生成过可定位的值）。一个泛左值可以是左值或消亡值。详见B.2节。

**header file（头文件）**

通过#include引导最终成为编译单元一部分的文件。此类未见通常包含变量的声明和涉及到一个以上翻译单元的函数，同时还有类型的定义、内联函数、模板、常量和宏。它们命名的后缀通常有.hpp，.h，.H，.hh，或.hxx。tenement也被称为包含文件。同时请查看CPP 文件和翻译单元（translation unit）。

**INCITS（**美国国家信息技术标准委员会**）**

*International Committee for Information Technology Standards*(美国国家信息技术标准委员会)的缩写。是ANSI认可的美国标准发展组织（之前被称为X3）。一个叫J16的分委员会是C++标准化后的源动力。它和International Organization for Standardization(ISO)（美国国家标准化组织）合作密切。

**include file（包含文件）**

详见头文件

**incomplete type（不完整类型）**

一个已被声明但未被定义的类，不完整元素类型的数组或未知大小的数组，底层类型没有定义的枚举类型，或者void（选择性的带有const 和/或 volatile）。

**indirect call（间接调用）**

将调用的函数直到调用实际发生时才被知道的函数调用（在运行时）。

**initializer（初始化器）**

指明如何初始化一个对象的构建。例如，在

Std::complex<float> z1 = 1.0,z2(0.0,1.0);

中，初始化器是 =1.0 和 （0.0，1.0）。

**initializer list（初始化列表）**

一个在括号中以逗号隔开的表达式列表，用于初始化对象和引用。初始化列表通常用于初始化变量，但也会，举个例子，在构造器定义中初始化成员和基类。该初始化可以直接发生，或通过一个介入性的std::initializer\_list对象初始化。

**injected class name（注入类名称）**

就像其自身定义范围内该类的名称一样可见。对于类模板，如果名称后未跟随一个模板参数列表，该模板的名称在模板范围内部被当作类名称使用。

**Instance（实例）**

术语实例在C++编程中有两重含义：从面向对象术语中拿出来的含义是一个类的实例：一个是一个类的实现的对象。例如，在C++中，std::cout 是类 std::ostream 的一个实例。另一个含义（并且这是本书中一直想表达的）是一个模板实例：一个类：一个函数，或一个通过使用确定参数替代所有模板参数值而获得的成员函数。在这种意义中，一个实例亦被称为特化，即使后者常常被用错为显示特化。

**Instantiation（实例化）**

在模板定义中替换模板参数来生成一个具体的实体（函数，类，变量或别名）。如果只有声明而不是定义被替代，有时会使用部分模板特化来描述。同时请查看substitution。创建一个类的实例（对象）的替代性意义在本书中未被使用（请看instance）。

**ISO（美国国家标准化组织）**

International Organization for Standardization的全球性缩写。一个被称为WG21的ISO工作组是为C++11的标准化和发展做出努力的源动力。

**Iterator（迭代器）**

一个知道如何横跨一列元素的对象。通常情况下，这些对象属于一个集（详见collection class）。

**linkable entity（可连接实体）**

下列中的任意一个：一个函数或成员函数、一个全局变量或一个静态数据成员、包括任何从模板中生成的东西，对链接器来说是可见的**。**

**liker（链接器）**

一种连接编译过的翻译单元并横跨这些单元决定可连接实体的引用的程序或操作系统服务。

**lvalue（左值）**

一类为存储值生成位置，且假定该位置不会移动的表达式。典型的例子是表示有名称对象的表达式（变量或成员）和字符串字面值。详见附录B.1节。

**member class template（成员类模板）**

一个代表一族成员类的构建。它是声明在其他类或类模板定义中的类模板。他有自己一系列的模板参数（与类模板的成员类不同）。

**member function template（成员函数模板）**

一个代表一族成员函数的构建。它有自己一系列的模板参数（与类模板的成员函数不同）。它和函数模板十分相似，但所有模板参数被替代时，他的结果是一个成员函数（而不是普通函数）。成员函数模板不能是虚的（virtual）。

**member template（成员模板）**

一个成员类模板，成员函数模板或静态数据成员模板。

**Modern C++（现代C++）**

本书中使用到的的词，指在C++11标准化及以后的C++语言。（例如，C++11，C++14，C++17）

**nondependent name（非依赖性名称）**

一个不依赖于模板参数的名称。请看*dependent name* 和 *two-phase lookup*。

**ODR（单定义原则）**

One-definition rule（单定义原则）的缩写。该规则将某些约束放入C++程序中出现的定义中。详见10.4节和附录A。

**one-definition rule（单定义原则）**

请看ODR

**overload resolution（重载解析）**

当多个候选者存在时（通常他们有相同名称）选择调用哪个函数的过程。同时请看附录C。

**parameter（参数）**

在某一点上用于替代某个实际“值”（一个变量）的占位符实体。对于宏参数和模板参数来说，该替代出现在编译时刻。对于函数调用参数，它发生在运行时刻。在某些编程社区中，参数也被称为形式化参数（变量被称为实际变量）。同时请看*argument* 和 *template argument*。

**parameterized class（参数化类）**

一个类模板或嵌套在类模板中的类。他们都是参数化的，因为直到模板参数被确定时他们才对应一个独立的类。

**parameterized function（参数化函数）**

一个函数或成员函数模板或模板的成员函数。他们都是参数化的因为直到模板参数被确定时它们才对应一个单一函数（或成员函数）。

**partial specialization（部分特化）**

为模板的明确替换声明或定义一个代替性定义的构建。原始的（通用）模板被称为主模板。替代性的定义依然依赖于模板参数。目前，该构建只为类模板存在。同时请看explicit specialization。

**POD（**朴素旧数据**）**

“朴素旧数据（类型）”的缩写。POD类型是可以不使用任何C++特性定义的类型（比如虚成员函数、获取关键字，以及此类）。例如，所有普通的C构建的strcut都是一个POD。

**POI（实例化点）**

Point of instantiation的缩写。一个POI是一个模板使用模板参数替换模板参数来概念性的扩展的源码的位置。实践中，该扩展不要出现在每个POI。同时请看显式实例化推导。

**point of instantiation**

请看POI

**policy class（策略类）**

为原组件描述可配置行为的类和类模板的成员。策略常以模板参数传递。例如，一个排序模板可能会有顺序策略。策略类也被称为策略模板或策略。同时请看*traits template*。

**polymorphism（多态）**

一个操作（由其名称识别）应用于多个不同类对象的能力。在C++中，传统面向对象编程的多态（也成为运行时或动态多态）通过在衍生类中重写的虚函数实现。此外，C++模板允许静态多态。

**precompiled header（预编译器头文件）**

一种可以让编译器快速加载的源码的处理形式。源码下的预编译头文件必须先是翻译单元的一部分（例如，它不能在翻译单元中间的某处开始）。通常情况下，预编译头文件对应几个头文件。使用预编译头文件可以持续性的减少C++编写的大型程序的组建所用的时间。

**primary template（主模板）**

不是部分特化的模板。

**prvalue（纯右值）**

执行初始化的一类表达式。纯右值能被假定用于分配例如1或true的纯数学值或临时变量（尤其是那些以值返回的值）。任何C++11前的右值在C++11中是纯右值。请看value category 和 B.2节。

**qualified name（量化名称）**

一个包含范围修饰符（：：）的名称。

**reference counting（引用计数）**

一个持续计算有多少实体引用一个特定资源的资源管理策略。当计数为0时，该资源可被丢弃。

**rvalue（右值）**

不是左值的一类值。一个右值可以是纯右值（一个临时变量）或消亡值（一个带有std::move()标记的左值）。在C++11前被称为右值的，在C++11中被称为纯右值。请看value category 和B.2节。

**SFINAE**

*Substitution Faliure Is Not A Error*（替换失败不是错误）的缩写。一种尝试以有效方式替换模板参数时静默丢弃模板而非触发编译错误的方法。重载集的其他模板之后获得被选中的机会，只要它们的替换成功。

**source file（源文件）**

一个头文件或CPP文件。

**specialization（特化）**

使用实际值替换模板参数的结果。一个特化可能通过实例化或显式实例化创建。该术语有时错误地等价于*eplicit specialization*。同时请看*instacne*。

**static data member template（静态数据成员模板）**

一个类或类模板地变量模板。

**substitution（替换）**

以实际类型、值或模板替换模板化实体中模板地参数地过程。在重载解析时，例如，只有最低限度的替换来建立候选函数的类型（的行为）被执行，并且入股该替换导致无效构建，则会使用SFINAE被使用。同时请看instantiation。

**template（模板）**

代表一族类型、函数、成员函数或变量的的构建。他确定了从那个实际类型、函数、成员函数或变量能通过确定实体替代模板参数来生成的模式。在本书中，该术语不包括函数、类、静态数据成员和只通过实际的类模板的成员参数化的类型别名。请看*alias template，class template，parameterized template，function template* 和*parameterized function*。

**template argument（模板参数）**

模板中的占位符。最常见的一种模板参数时代表类型的类型参数。非类型参数代表特定类型的常量值，模板模板参数代表类型模板。同时请看parameter。

**template argument deduction（模板实体）**

请看deduction。

**template-id（模板id）**

模板名后跟随一个使用尖括号指明的模板参数的组合。

**template parameter（模板参数）**

模板中常见的占位符。最常见的模板参数时代表类型的类型参数。非类型参数代表确定类型的常量值，并且模板模板参数代表类型模板。同时请看*parameter*。

**template entity（模板实体）**

在一个模板中定义或创建的模板或实体。后者包括像一个类模板的普通成员函数或者是出现在一个模板中的封闭类型的lambda表达式。

**traits template（特征模板）**

一个具有描述模板参数特性（特征）的成员的类模板。特征模板的常用目的是避免过多数量的模板参数。同时请看*policy class*。

**translation unit（翻译单元）**

一个CPP文件和它通过使用#include指令包含的所有的头文件和标准库头文件，减去通过状态编译指令例如#if排除的程序文本。为了简单性，它也可以被认为是预处理一个CPP文件的结果。请看*CPP file*和*header file*

**true constant（真值常量）**

一个在编译时可以被编译器计算值的表达式。请看*constant-expression*

**tuple（元组）**

C struct概念的泛化，这样成员可以按序获取。

**two-phase lookup（两阶段检查）**

对模板中的名称使用的名称检查方法。这两个阶段是（1）模板定义的过程和（2）模板对确定模板参数的实例化。非依赖性名称只在第一阶段检查，在该阶段中非依赖性基类不被考虑。带有范围修饰符（::）的依赖性名称只在第二阶段检查。没有范围修饰符的依赖性名称可能在两个阶段都检查，但在第二阶段中，只有参数依赖检查被执行。

**type alias（类型别名）**

一个类型的替代性名称，使用typedef声明，一个别名声明或一个别名模板的实例引入。

**type template（类型模板）**

一个类模板，成员类模板，或别名模板。

**universal reference（通用引用）**

形式为T&&的右值引用的两个术语之一，其中T是可推断的模板参数。与普通右值引用不同的特殊规则会使用（详见6.1节）。该术语被Scott Meyers 同时以左值引用和右值引用的普通术语制造出来。因为“普通的”是，嗯，太普遍了。C++17标准引入了*forwading reference*来替代它。

**user-defined conversion（用户定义转换）**

编程者定义的类型转换。它可以被具有一个参数或转换函数调用的构造器。除非构造器或转换函数以关键字explicit声明，否则该类型转换可以隐式发生。

**value category（值类别）**

一个表达式的分类。传统的值类别左值和右值继承于C。C++11引入了替换性的类别：泛左值（generalized lvalues），它的计算定义了存储对象，以及出右值（pure rvalues），它的计算初始化对象。而外的划分将泛左值分为左值（localizable value）和消亡值（eXpiring values）。此外，C++11中右值用作消亡值和纯右值的通常类别（C++11前，右值为C++11中的纯右值）。请看附录B获取相关细节。

**variable template（变量模板）**

代表一族变量或静态数据成员的构建。它指明了通过特定实体替换模板参数来生成实际变量和静态数据成员的模式。

**whitespace（空格）**

在C++中，这是在源码中分隔符号（标识符，字面值，符号，等等）的空。除了传统的空格，换行和水平制表符，这也包括注释。其它空格符（该页面传入了一个control字符）有时也是有效的空格符。

**xvalue（消亡值）**

生成一个假定不再需要的但已存储对象的一类表达式。典型的例子是带有std::move()标记的左值。请看*value category* 和 B.2节。