C++

c++(/ˌ 我的ɪˈ ˈ s p ɪ ˈ s/ "看见加号")是一种通用编程语言。它具有必需的、面向对象的和通用的编程功能，同时还为低级内存操作提供了便利。

C++

范式	多范式:过程、功能 、面向对象、通用[1]
设计人员	bjarne stroustrup
第一次出现	1985 年; 33 年前
稳定的释放	ioc/ieca14882:2017/2017 年 12 月 1 日; 11 个月前
打字纪律	静态的,名义的,部分推断
实现语言	c++ 或 c
文件名扩展	.c,. cc,. cpp,. cx,. cxx,. c ++, .h,. hh,. hpp,. hxx,. h +
网站	伊索克普公司
主要实现	
llvm clang, gcc, microsoft visual c++, embarcadero c++ builder,英特尔 c++ 编译器, ibm xl c++, edg	
受	
ada, algol 68, c, clu, ml, simula	
影响	
ada 95, c#,[2] c99,教堂,[3] clojure,[4]] d, java,[5] lua , 尼姆,[*引文需要*] perl, php, python, [6] 铁锈	
• wikibooks 的 c++ 编程	

它的设计偏向于系统编程和嵌入式、资源受限和大型系统，其设计亮点是性能、效率和使用灵活性。[7] c++ 在许多其他上下文中也被认为是有用的，其主要优势是软件基础结构和资源受限的应用程序，[7] 包括桌面应用程序、服务器（例如电子商务、 web 搜索或 sql 服务器）和性能关键型应用程序（例如电话交换机或空间探测器）。[8] c++ 是一种编译语言，它的实现可在许多平台上使用。许多供应商提供 c++ 编译器，包括自由软件基金会、 microsoft、英特尔和 ibm.

c++ 由国际标准化组织(iso) 标准化，最新的标准版本于 2017 年 12 月由 iso 批准并发布为 *iso/iec 1482:2017*（非正式称为 C++17）.[9] c++ 编程语言最初被规范化了 *1998 年作为 iso/iec 14882:1998*，然后由 C++03, C++11 和 C++14 标准修正。当前的 C++17 标准取代了这些具有新功能和扩大的标准库。在 1998 年首次标准化之前，bjarne stroustrup 自 1979 年以来一直在贝尔实验室开发 c++，作为 c 语言的延伸，因为他想要一种类似于 c 的高效和灵活的语言，c 语言也提供了程序组织的高级功能。[10] C++20 是以后的下一个计划的标准，保持一个新的版本的当前连续每三年[11] .

许多其他编程语言都受到了 c++ 的影响，包括 c#、 d、 java 和较新版本的 c。

历史

1979 年，丹麦计算机科学家 bjarne stroustrup 开始研究 c++ 的前身 "c 与类" .[12] 创造一种新语言的动机源于 stroustrup 在编程方面的经验，为他的博士论文。stroustrup 发现，simula 具有对大型软件开发非常有帮助的功能，但语言对于实际使用过于缓慢，而 bcpl 速度快，但太低级，不适合大型软件开发。当 stroustrup 开始在 at & amp; t 贝尔实验室工作时，他遇到了在分布式计算方面分析 unix 内核的问题。想起他的博士经验，stroustrup 着手提高 c 语言与 simula 一样的功能。[13] 选择 c 是因为它是通用的，快速的，便携的和广泛使用。除了 c 和 simula 的影响外，其他语言也影响了 c++，包括 algo:68、 ada、 clu 和 ml.

最初，stroustrup 的 "c 与类" 为 c 编译器、cpre（包括类、派生类、强类型、内联和默认参数）添加了功能。[10]

1983 年，"带类的 c" 被重命名为 "c++"（++是 c 中的增量运算符），添加了新功能，其中包括虚拟函数、函数名称和运算符重载、引用、常量、类型安全的自由存储内存分配（new/删除）、改进的类型检查和带有两个正向斜杠（///）的 bcpl 样式单行注释。此外，它还包括为 c++ 开发一个独立的编译器，.

1985 年，第一版的 *c++ 编程语言*发布，这成为该语言的明确参考，因为还没有一个官方的标准。[15] c++ 的第一个商业实现于同年 10 月发布。[12]

1989 年发布了 c++ 2.0，随后于 1991 年发布 *了更新的 c++ 编程语言*第二版。[16]2.0 中的新功能包括多个继承、抽象类、静态成员函数、const 成员函数和受保护的成员。1990 年，《*注释性 c++ 参考手册*》出版。这项工作成为未来标准的基础。后来的功能添加包括模板、异常、命名空间、新强制转换和布尔类型.

2.0 更新后，c++ 发展相对缓慢，直到 2011 年发布了 C++11 标准，增加了许多新功能，进一步扩展了标准库，并为 c++ 程序员提供了更多的功能。在 2014 年 12 月发布了少量 C++14 更新之后，C++17 引入了各种新的新增内容，并计划在 2020 年进行进一步修改。[17]

截至 2017 年，c++ 仍然是第三大最流行的编程语言，仅次于 java 和 c [18][19]

2018 年 1 月 3 日，stroustrup 被宣布为 2018 年查尔斯·史塔克·德雷珀工程奖获得者，"旨在构思和开发 c++ 编程语言"。[20]

词源

斯特鲁斯特鲁普认为: "这个名字象征着 c 的变化的进化性质"。[21] 这个名字被记入里克·马西蒂（中的 1983)[14] 并且在 1983 年 12 月第一次使用了。当 mascitti 在 1992 年被非正式地询问命名问题时，他表示，命名是用口若悬河的精神发出的。该名称来自 c 的++ 运算符(它增加变量的值）和使用 "+" 指示增强的计算机程序的常见命名约定。

在 c++ 的开发期间，该语言在获取其最终名称之前被称为 "新 c" 和 "c 与类" [14] [22].

哲学

在 c++ 的整个生命过程中，它的发展和演变一直遵循一套原则:[13]

- 它必须由实际问题驱动，其功能应该在现实世界的程序中立即有用。
- 每个功能都应该是可实现的（有相当明显的方法)。
- 程序员应该可以自由地选择自己的编程风格, c++ 应该完全支持这种风格。
- 允许一个有用的功能比防止每次可能滥用 c++ 更重要。
- 它应提供设施，将项目组织成明确界定的单独部分，并为组合单独开发的部分提供设施。

- 没有隐式违反类型系统(但允许显式冲突；即程序员明确请求的冲突)。
- 用户创建的类型需要具有与内置类型相同的支持和性能。
- 未使用的功能不应对创建的可执行文件产生负面影响(例如，性能较低)。
- c++ 下面不应有任何语言（汇编语言除外)).
- c++ 应与其他现有编程语言一起工作，而不是培养自己独立且不兼容的编程环境.
- 如果程序员的意图是未知的，允许程序员通过提供手动控制来指定它。

标准化

年	c++ 标准	非正式名称
1998 年	iso/ieci 14882:1998[23]	C++98
2003 年	iso/ieci 14882:2003[24]	C++03
2011 年	iso/iec 14882:2011 [25]	C++11，C++0x
2014 年	iso/iec 14882:2014[26]	C++14，C++1y
2017 年	iso/iec 14882:2017[9]	C++17，C++1z
2020 年	待定	C++20 [17]

c++ 由一个被称为 jtcnksc22/wg21 的 iso 工作组进行标准化。到目前为止，它已经发布了对 c++ 标准的五次修订，目前正在进行下一次修订，C++20.

1998 年，iso 工作组首次将 c++ *标准化为 iso/ieci 14882:1998*，非正式地称为 *C++98*。2003 年，它发布了一个新版本的 c++

标准，名为 *iso/iec 14882:2003*,它修复了在 C++98 中发现的问题.

该标准的下一次重大修订被非正式地称为 "C++0x"，但直到 2011 年才发布。[27]C++11 (14882:2011) 包括了许多增加到核心语言和标准库。[25]

2014 年, C++14 (也称为 C++1y) 作为 C++11 的一个小扩展发布，主要是错误修复和小改进。[28] 国际标准投票程序草案于 2014 年 8 月中旬完成。[29]

在 C++14 之后, iso c++ 委员会于 2017 年 7 月中旬完成了 C++17 的一项重要修订，称为 C++1z，并于 2017 年 12 月获得批准并公布。[30]

作为标准化过程的一部分, iso 还发布技术报告和规范:

- iso/iect18015:2006[31] 关于在嵌入式系统中使用 c++ 以及 c++ 语言和库功能对性能的影响,
- iso/ieci tr 197:2007[32] (也称为 c++ 技术报告 1) 关于库扩展主要集成到 C++11,
- iso/ieci tr 29124: 2010[33] 关于特殊的数学函数,
- io/ieci tr 24733:2011 [34] 关于十进制浮点算法,
- isoiec ts 18822[2015] [35] 标准文件系统库,
- isoiec ts 1957: 2015[36] 关于标准库算法的并行版本,

- isoiec ts 198441: 2015[37] 上的软件事务性内存,
- isoiec ts 19568: 2015 [38] 关于一组新的库扩展，其中一些已集成到 C++17,
- isoiec ts 19217: 2015 [39] 关于 c++概念

更多的技术规范正在开发中，有待批准，包括并发库扩展、网络标准库、范围和模块。[40]

语言

c++ 语言有两个主要组件：主要由 c 子集提供的硬件功能的直接映射和基于这些映射的零开销抽象。stroustrup 将 c++ 描述为 "一种轻量级的抽象编程语言［设计］，用于构建和使用高效和优雅的抽象";[7] 和 "提供硬件访问和抽象是 c++ 的基础。有效地做到这一点是它与其他语言的区别所在 "。[41]

c++ 继承了大多数 c 语法。下面是 bjarne stroustrup 的你好世界程序的版本，它使用 c++ 标准库流工具编写一条消息到标准输出:[42][43]

```
1 #include <iostream>2 3 int main()4 {5     std::cout << "Hello, world!\n";6 }
```

对象存储

与 c 中一样, c++ 支持四种类型的内存管理: 静态存储持续时间对象、线程存储持续时间对象、自动存储持续时间对象和动态存储持续时间对象。[44]

静态存储持续时间对象

静态存储持续时间对象是在输入 main() ）之前创建的（见下文异常），并在 main() ）出口之后按相反的创建顺序销毁。标准没有指定创建的确切顺序（尽管下面定义了一些规则），以便在如何组织实现方面实现一些自由。更正式地说，这类物体的寿命 "应持续到程序的持续期间"。[45]

静态存储持续时间对象分两个阶段初始化。首先，执行 "静态初始化"，并且只有在执行所有静态初始化*之后*，才执行 "动态初始化"。在静态初始化中，所有对象首先用零初始化;之后，所有具有常量初始化阶段的对象都使用常量表达式（即用文本或 constexpr 初始化的变量）进行初始化。虽然标准中没有指定静态初始化阶段，但它可以在编译时完成，并保存在可执行文件的数据分区中。动态初始化涉及通过构造函数或函数调用完成的所有对象初始化（除非函数用 constexpr 约定标记，以 C++11)。动态初始化顺序被定义为编译单元内的声明顺序(即同一文件)。对于编译单元之间的初始化顺序没有提供任何保证。

线程存储持续时间对象

此类型的变量与静态存储持续时间对象非常相似。主要区别在于创建时间就在线程创建之前，销毁是在线程连接之后完成的。[46]

自动存储持续时间对象

C++ 中最常见的变量类型是函数或块内的局部变量以及临时变量。[47] 自动变量的常见特征是，它们的生存期仅限于变量的范围。它们是在声明点创建并可能初始化的（有关详细信息，请参阅下文），并在离开作用域时按*相反*的创建顺序销毁。这是通过堆栈上的分配来实现的.

当执行点通过声明点时，将创建局部变量。如果变量具有构造函数或初始值设定项，则这将用于定义对象的初始状态。当在其中声明的局部块或函数关闭时，局部变量将被销毁。局部变量的 C++ 析构函数在对象生存期结束时调用，允许使用称为 raii 的自动资源管理规程，这在 C++ 中被广泛使用。

在创建父对象时创建成员变量。数组成员按顺序从数组的0初始化为最后一个成员。当父对象按创建的相反顺序销毁时，成员变量将被销毁。即如果父母是 "自动对象"，那么当它超出范围，引发其所有成员的破坏时，它将被销毁。

临时变量是作为表达式计算的结果创建的，并在包含表达式的语句已完全计算（通常在；在;语句末尾）时销毁。

动态存储持续时间对象

主要文章:新的和删除（c++）

这些对象具有动态生命周期，创建 **new** 对象并通过调用 **delete** 销毁。[48]

模板

c++ 模板支持通用编程。c++ 支持函数、类、别名和变量模板。模板可以按类型、编译时常量和其他模板进行参数化。模板是通过编译时*的实例*化实现的。为了实例化模板，编译器将特定参数替换为模板的参数，以生成具体的函数或类实例。某些替换是不可能的;这些被 "替换失败不是错误" (sfinae) 一语描述的重载解析策略所消除。模板是一个功能强大的工具，可用于泛型编程、模板元编程和代码优化，但这种功能意味着成本。模板的使用可能会增加代码的大小，因为每个模板实例化都会生成模板代码的副本：但是，如果手动编写代码，将生成的代码量相同或较少。[49] 这与其他语言（如 java）中看到的运行时泛型形成鲜明对比，在这些语言中，该类型在编译时被擦除，并且保留了一个模板体。

模板不同于宏：虽然这两个编译时语言功能都支持条件编译，但模板并不局限于词法替换。模板了解其配套语言的语义和类型系统，以及所有编译时类型定义，并可以执行高级操作，

包括基于严格类型检查参数的评估的编程流控制。宏能够根据预定的条件对编译进行条件控制，但不能实例化新类型、递归或执行类型计算，实际上仅限于编译前文本替换和文本包含/排除。换句话说，宏可以基于预定义的符号控制编译流，但不能独立地实例化新的符号。模板是用于静态多态性（见下文）和泛型编程的工具.

此外，模板是 c++ 中的一种编译时间机制，它是 turing-sit 完成的，这意味着计算机程序可以表示的任何计算，在运行时之前，可以通过模板元程序以某种形式进行计算。

总之，模板是在不知道用于实例化模板的特定参数的情况下编写的编译时参数化函数或类。实例化后，生成的代码等效于专门为传递的参数编写的代码。通过这种方式，模板提供了一种方法，可以将函数和类的一般、广泛适用的方面（在模板中编码）与特定方面（用模板参数编码）分离，而不会因为抽象而牺牲性能。

对象

主要文章: c++ 类

c++ 将面向对象编程（oop）功能引入到 c。它提供类，它们提供了 oop（和一些非 oop）语言中常见的四个功能:抽象、封装、继承和多态性。与其他编程语言中的类相比, c++ 类的

一个显著特征是支持确定性析构函数，而确定性析构函数又为资源获取提供了支持，即初始化(raii) 概念。

封装

封装是信息的隐藏，以确保数据结构和运算符按预期用途使用，并使开发人员更明显地使用该模型。c++ 提供了将类和函数定义为其主要封装机制的能力。在类中，成员可以声明为公共、受保护或私有，以显式强制封装。类的公共成员可供任何函数访问。私有成员只能对属于该类成员的函数以及由类（"朋友"）显式授予访问权限的函数和类进行访问。除了类本身和任何朋友之外，从类继承的类的成员也可以访问受保护的成员。

面向对象的原则确保了所有的封装，并且只封装了访问类型的内部表示的函数。c++ 通过成员函数和朋友函数支持此原则，但它不强制执行此原则。程序员可以将类型的部分或全部表示声明声明为公共表示，并且允许他们使公共实体不属于某个类型的表示形式。因此，c++ 不仅支持面向对象的编程，还支持其他分解范式，如模块化编程.

通常认为，将所有数据都公开是私有的或受保护的，并只公开那些作为类用户最小接口一部分的函数，通常被认为是一种良好做法。这可以隐藏数据实现的细节，允许设计器以后从根本上改变实现，而不以任何方式更改接口。[5] [51]

继承

继承允许一种数据类型获取其他数据类型的属性。基类的继承可以声明为公共继承、受保护继承或私有继承。此访问指定符确定不相关的派生类是否可以访问基类的继承的公共成员和受保护的成员。只有公共继承才相当于通常所说的 "继承"。另外两种形式的使用频率要低得多。如果省略访问指定符，则 "类" 将私下继承，而 "结构" 则公开继承。基类可以声明为虚拟类;这称为虚拟继承。虚拟继承可确保继承图中只存在基类的一个实例，从而避免了多个继承的一些模糊问题。

多重继承是大多数其他语言中找不到的 c++ 功能，它允许从多个基类派生一个类;这允许更复杂的继承关系。例如，"飞猫" 类可以同时继承 "猫" 和 "飞哺乳动物"。其他一些语言（如 c# 或 java)通过允许继承多个接口，同时将基类的数量限制为一个（接口与类不同,仅提供成员函数的声明，不提供实现或成员数据)。c# 和 java 中的接口可以在 c++ 中定义为只包含纯虚拟函数的类，通常称为抽象基类或 "abc"。这种抽象基类的成员函数通常在派生类中显式定义，而不是隐式继承。c++ 虚拟继承具有称为显性的歧义解析功能.

操作员和操作员超载

不能重载的运算符

算子	象征
范围解析运算符	::
条件运算符	?:
点运算符	.

成员选择操作员	.*
"大小" 运算符	**sizeof**
"类型" 运算符	typeid

主要文章: C 和 C++ 中的操作员

C++ 提供了 35 个以上的运算符，包括基本算术、位操作、间接、比较、逻辑操作等。对于用户定义的类型，几乎所有运算符都可以重载，但有几个显著的例外,.如成员访问（和.**）以及条件运算符。丰富的可重载运算符集对于使 C++ 中的用户定义类型看起来像内置类型至关重要。

可重载运算符也是许多高级 C++ 编程技术（如智能指针)的重要组成部分。重载运算符不会更改涉及运算符的计算的优先级，也不会更改运算符使用的操作数（但运算符可能会忽略任何操作数，但它将在执行前进行计算)。重载的 "&&" 和"||" 运算符失去了短路评估属性。

多 态 性

参见:多态性（计算机科学)

多态性使许多实现的一个公共接口，以及对象在不同情况下的不同行为。

C++ 支持多种*静态*(在编译时解析）和*动态*(在运行时解析）多态性，并得到上述语言功能的支持。编译时多态性不允许某些运行时决策，而运行时多态性通常会导致性能下降。

静态多态性

函数重载允许程序声明具有相同名称但具有不同参数（即*临时多态性*）的多个函数。这些函数是由其形式参数的数量或类型来区分的。因此，相同的函数名称可以引用不同的函数，具体取决于它所使用的上下文。函数返回的类型不用于区分重载函数，并将导致编译时错误消息。

声明函数时，程序员可以为一个或多个参数指定默认值。这样做允许在调用函数时选择省略具有默认值的参数，在这种情况下将使用默认参数。当调用函数时，参数少于声明的参数，则显式参数将按从左到右的顺序与参数匹配，参数列表末尾的任何不匹配参数都将被分配给它们的默认参数。在许多情况下，在单个函数声明中指定默认参数比提供具有不同参数数的重载函数定义更为可取。

C++ 中的模板为编写泛型多态代码（即参数多态性）提供了一种复杂的机制。特别是，通过奇怪的重复模板模式，可以实现一种静态多态性形式，这种形式与重写虚拟函数的语法非常相似。由于 C++ 模板具有类型感知性和 tur 冲击力–完整，因此它们还可用于让编译器解析递归条件，并通过模板元编程生成实质性程序。与某些观点相反，模板代码在使用正确的编译器设置编译后不会生成批量代码。[49]

动态多态性

继承

另请参见:子类型

对于 c++ 中的基类类型的变量指针和引用也可以引用该类型的任何派生类的对象。这允许数组和其他类型的容器保存指向不同类型的对象的指针（引用不能直接保存在容器中）。这将启用动态（运行时）多态性，其中引用的对象的行为可能因其（实际的派生）类型而异。

c++ 还提供 dynamic_cast 运算符，它允许代码通过基本引用指针安全地尝试将对象转换为派生程度更高的类型:*向下转换*。*尝试*是必要的，因为通常不知道引用了哪个派生类型。（*转换*为更一般类型时，始终可以通过 static_cast 在编译时执行检查，因为祖先类是在派生类的接口中指定的，所有调用方都可以看到。dynamic_cast 依赖于运行时类型信息(rtti)，程序中的元数据支持区分类型及其关系。如果对指针的 dynamic_cast 失败，则结果为 nullptr 常量，而如果目标是引用（不能为空），则强制转换将引发异常。*已知*具有特定派生类型的对象可以通过 static_cast 进行强制转换，绕过 rtti 和 dynamic_cast 的安全运行时类型检查，因此只有在程序员非常确信强制转换是,并将永远是，有效的。

虚拟成员功能

通常，当派生类中的函数重写基类中的函数时，要调用的函数由对象的类型决定。当给定函数的两个或多个定义之间的参数数量或类型不存在差异时，将重写给定的函数。因此，在编译时，可能无法确定对象的类型，因此也不可能确定要调用的正确函数，因为只给定一个基类指针;因此，该决定被推迟到运行时。这称为动态调度。虚拟成员函数或*方法*[52] 允许根据对象的实际运行时类型调用函数的最具体实现。在 c++ 实现中，这通常使用虚拟函数表完成。如果对象类型已知，则可以通过在函数调用之前预置完全限定的类名来绕过此操作，但通常在运行时解析对虚拟函数的调用。

除了标准的构件函数外，运算符重载和析构函数也可以是虚拟的。根据经验，如果类中的任何函数都是虚拟的，则析构函数也应该是虚拟的。由于对象在创建时的类型在编译时是已知的，因此构造函数和扩展副本构造函数不能是虚拟的。但是，当指向派生对象的指针作为指向基对象的指针传递时，可能会出现需要创建对象副本的情况。在这种情况下，常见的解决方案是创建 clone() (或类似）虚拟函数，该函数在调用时创建并返回派生类的副本。

成员函数也可以通过在右括号和分号之前以= 0 追加它来使其成为 "纯虚拟" 函数。包含纯虚拟函数的类称为*抽象类*。对象不能从抽象类创建;它们只能来自。任何派生类都将虚拟函数

作为纯函数继承，并且必须提供它（以及所有其他纯虚拟函数）的非纯定义，然后才能创建派生类的对象。尝试创建具有纯虚拟成员函数或继承的纯虚拟成员函数的类的对象的程序格式不正确。

兰姆达表达式

c++ 提供对匿名函数(也称为 lambda 表达式）的支持，其形式如下:

```
[capture](parameters) -> return_type { function_body }
```

[capture]列表支持闭包的定义。这种 lambda 表达式在标准中被定义为未命名函数对象的语法糖。一个示例 lambda 函数可以定义如下:

```
[](int x, int y) -> int { return x + y; }
```

异常处理

异常处理用于将存在的运行时问题或错误进行通信，从检测到该问题的位置到可以处理该问题的位置。[53] 它允许这在一个统一的方式和分开地从主要代码做，同时检测所有错误。[54] 如果发生错误，将引发（引发）异常，然后由最近的适当异常处理程序捕获。异常会导致当前作用域退出，并且每个外部作用域（传播），直到找到合适的处理程序，从而依次调用

这些退出作用域中任何对象的析构函数。[55] 同时，异常被显示为一个对象运载关于被检测到的问题的数据。[56]

请注意，许多 c++ "样式"，如 google 的,[57] 禁止在 c++ 程序中使用异常，从而限制了语言。

导致异常的代码被放置在 try 块中。异常在单独的 catch 块(处理程序）中处理;每个 try 块可以有多个异常处理程序，因为它在下面的示例中可见。[58]

```
1 #include <iostream> 2 #include <vector> 3 #include <stdexcept> 4  5 int main() { 6  try { 7  std::vector<int> vec{3, 4, 3, 1}; 8  int i{vec.at(4)}; // Throws an exception, std::out_of_range (indexing for vec is from 0-3 not 1-4) 9  }10  // An exception handler, catches std::out_of_range, which is thrown by vec.at(4)11  catch (std::out_of_range &e) {12 std::cerr << "Accessing a non-existent element: " << e.what() << '\n';13  }14 // To catch any other standard library exceptions (they derive from std::exception)15 catch (std::exception &e) {16  std::cerr << "Exception thrown: " << e.what() << '\n';17  }18  // Catch any unrecognised exceptions (i.e. those which don't derive from std::exception)19  catch (...) {20  std::cerr << "Some fatal error\n";21  }22 }
```

也可以使用 throw 关键字有目的地引发异常;这些例外是以通常的方式处理的。在某些情况下，由于技术原因，不能使用例外。其中一个例子是嵌入式系统的关键组件，在该组件中，必须保证每个操作都能在指定的时间内完成。这不能在异常的情况下确定，因为没有任何工具来确定处理异常所需的最长时间。[59]

标准库

主要文章: c++ 标准库

c++标准由两部分组成: 核心语言和标准库。c++ 程序员期望后者在 c++ 的每个主要实现上都是新的;它包括聚合类型（向量、列表、地图、集合、队列、堆栈、数组、元组)、算法(查找每个，二进制搜索、随机 _ 洗牌等)，输入输出设施(iostream，用于从和写入控制台和文件)、文件系统库、本地化支持、用于自动内存管理的智能指针、正则表达式支持、多线程库、 atomics 支持（允许变量在没有任何外部同步的情况下，最多一次由一个线程读取或写入时间实用程序（测量、获取当前时间等)，这是一种将错误报告转换为 c++ 异常的系统，随机数生成器和 c 标准库的稍有修改的版本（使其符合 c++ 类型系统)。

c++ 库的很大一部分基于标准模板库(stl)。stl 提供的有用工具包括容器作为对象（如向量和列表）的集合、提供类似于数组的容器访问的迭代器以及执行的算法操作，如搜索和排序。

此外，还提供了（多）映射（关联数组）和（多）集，所有这些都导出兼容的接口。因此，可以使用模板编写使用任何容器或迭代器定义的任何序列的泛型算法。与 c 中一样，通过使用 #include 指令来包含标准标头来访问库的功能。c++ 标准库提供 105 个标准标头，其中 27 个不推荐使用。

该标准采用了最初由亚历山大·斯捷潘诺夫设计的 stl，他多年来一直在试验通用算法和容器。当他从 c++ 开始时，他终于找到了一种语言，在这种语言中，可以创建通用算法（例如 stl 排序），这种算法的性能甚至比 c 标准库 qsort 要好，这得益于 c++ 功能，例如使用内联和编译时绑定而不是函数指针。该标准并不将其称为 "stl"，因为它只是标准库的一部分，但该术语仍被广泛用于将其与标准库的其他部分（输入输出流、国际化、诊断、c 库子集等）区分开来。[60]

大多数 c++ 编译器和所有主要编译器都提供了 c++ 标准库的符合标准的实现。

兼容性

为了给编译器供应商更大的自由，c++ 标准委员会决定不强制实施名称处理、异常处理和其他特定于实现的功能。此决定的缺点是，不同编译器生成的对象代码预期不兼容。但是，有人试图使特定计算机或操作系统的编译器标准化（例如 c++ abi），[61] 尽管它们现在似乎已基本放弃。

带 c

更多信息：c 和 c++ 的兼容性

c++ 通常被认为是 c 的超集，但这并不是严格意义上的事实。[62] 大多数 c 代码可以很容易地在 c++ 中正确编译，但有一些差异会导致某些有效的 c 代码无效或在 c++ 中的行为不同。例如，c 允许隐式转换从 void*到其他指针类型，但 c++ 不这样做（出于类型安全的原因）。此外，c++ 还定义了许多新关键字，如 new 关键字和 class，这些关键字可用作 c 程序中的标识符（例如，变量名）。

1999 年修订的 c 标准（c99）消除了一些不兼容性，该标准现在支持 c++ 功能，如行注释（//）和与代码混合的声明。另一方面，c99 引入了一些 c++ 不支持的新功能，这些功能在 c++ 中不兼容或冗余，例如可变长度数组、本机复数类型(但是，std::complex：c++ 标准中的复杂类)库提供类似的功能，但不兼容代码)、指定的初始值设定项、复合文本和 restrict 关键字。[63] c99 引入的一些功能包含在 c++ 标准的后续版本中，C++11 (在那些不是冗余的功能中)。[64] [65] [66] 然而，C++11 标准引入了新的不兼容性，例如不允许将字符串文本分配给字符指针，这仍然有效 c。

要将 c 和 c++ 代码混合在一起，在 c 和 c++ 中同时使用的任何函数声明或定义都必须通过 c 链接声明，方法是将其放置在 extern "C" {/*...*/} .. */} 块中。这样的函数可能不依赖于特征，取决于名称处理(即函数重载)。

批评

主要文章: c++ 的批评

尽管它被广泛采用，一些著名的程序员批评了 c++ 语言，包括 linus torvalds,[67]richard stallman,[68] joshua bloch, ken 汤普森,[69][70][71]] 和 donald knuth。 [72] [73]

c++ 最常被批评的一点是，它被认为是一种语言的复杂性，有人批评说，在实践中，大量的非正交特征需要将代码限制在 c++ 的子集上，从而避免了常见的可读性优势风格和成语。正如 joshua bloch 所说:

> 我认为 c++ 被推送到了它的复杂性阈值之外，然而却有很多人在编程它。但你要做的是强迫人们把它子集起来。因此，我所知道的使用 c++ 的几乎每一家商店都说: "是的，我们使用的是 c++，但我们没有执行多实现继承，也没有使用运算符重载。由于生成的代码的复杂性太高，您不会使用的功能只是一堆。而且我觉得当你必须开始这样做的时候是不好的。你失去了这个程序员的可移植性，每个人都可以阅读其他人的代码，我认为这是一件非常好的事情。

donald knuth(1993，评论标准化前的 c++)，谁说 edsger dijkstra "认为编程在 c++" 将使他身体不适 ":[72][73]

> 我今天和他们在一起的问题是.....。c++ 太复杂了。目前，我不可能编写可移植代码，我相信这些代码会在很多不同的系统上工作，除非我避免所有的异国情调功能。每当 c++ 语言设计人员对如何解决某些问题有两个相互竞争的想法时，他们都会说: "好吧，我们会两个都做"。所以这种语言太巴洛克了，不符合我的口味。

肯·汤普森，谁是斯特鲁斯特鲁普的同事在贝尔实验室，给出了他的评估:[70][71]

> 它当然有它的优点。但总的来说，我认为这是一种糟糕的语言。它把很多事情做得都有一半，它只是一个相互排斥的想法垃圾堆。我认识的每个人，无论是个人的还是公司的，都会选择一个子集，这些子集是不一样的。因此，传输算法不是一种好的语言--比如，"我写了它;在这里，把它。它太大了，太复杂了。而且显然是由一个委员会建造的。斯特鲁斯特鲁普多年来一直在竞选，远远超出了他对语言所做的任何技术贡献，使其被采用和使用。他用鞭子和椅子管理着所有的标准委员会。而他对任何人都说 "不"。他把所有的特点都放在了这种语言中。它的设计并不干净--它只是所有的东西的结合。我认为它遭受了巨大的痛苦。

然而 brian kernighan,也是贝尔实验室的同事，对这一评估提出异议:[74]

> c++ 具有巨大的影响力。...很多人说 c++ 太大，太复杂等，但事实上，它是一个非常强大的语言，几乎所有的东西，在那里有一个真正合理的理由：它不是有人做随机发明，它实际上是人们试图解决真正的 wo 世界上的问题。现在，我们今天认为理所当然的很多程序，我们只是使用，都是 c++ 程序。

斯特鲁斯特鲁斯特鲁自己评论说: "在 c++ 中，有一种更小、更干净的语言难以走出来"。[75]

其他投诉可能包括缺乏反射或垃圾收集、编译时间慢、感知到的特征蠕变、[76] 和冗长的错误消息，特别是来自模板的错误消息元编程。[77]

另请参见

引用

1. stroustrup, bjarne(1997 年). "1"。*c++ 编程语言*(第三版)。国际标准书号 0-201-88954-4. oclc59193992.

2. naugler, david (2007 年 5 月)。"c++ 和 java 程序员的 c# 2.0: 会议研讨会"。*大学计算科学杂志*。**22**(5)。尽管 c# 受到 java 的强烈影响，但它也受到了 c++ 的强烈影响，最好被视为 c++ 和 java 的后代.

3. "教堂规格（鸣谢)"(pdf)。cay inc. 2015 年 10 月 1 日。检索 2016 年 1 月 14 日.

4. "富有的希基问答由迈克尔·福格斯"的影响。检索 2017–01–11 .

5. 哈利。h. chaudhary (2014 年 7 月 28 日)。"破解 java 编程访谈:: 2000 + java 面试 que\ ans"的影响。检索 2016 年 5 月 29 日.

6. "9。类–python 3.6.4 文档 ".*docs.python.org* 的影响。检索 2018–01–09 .

7. ^ [a] [b] [c] stroustrup, b. (2014 年 5 月 6 日)。"讲座: c++ 的精髓。爱丁堡大学 "。检索 2015 年 6 月 12 日.

8. stroustrup, bjarne (2014 年 2 月 17 日)。"c++ 应用程序"。*斯特鲁斯特鲁普网站*的影响。检索 2014 年 5 月 5 日.

9. ^ [a] [b] "iso/ieciec 14882:2017".国际标准化组织。

10. "bjarne stroustrup 的主页 "。*www.stroustrup.com*.

11. "c++;它的走向 ".

12. ^ ᵃ ᵇ stroustrup, bjarne (2010 年 3 月 7 日)。"bjarne stroustrup 的常见问题: c++ 是什么时候发明的？"*斯特鲁斯特鲁普网站*的影响。2010 年 9 月 16 日检索.

13. ^ ᵃ ᵇ 斯特鲁斯特鲁普，比耶恩。"在现实世界中和为现实世界发展一种语言: c + + 1991–2006"(pdf).

14. ^ ᵃ ᵇ ᶜ 斯特鲁斯特鲁普，比耶恩。"c++ 的历史: 1979–1991"(pdf).

15. 斯特鲁斯特鲁普，比耶恩。"c++ 编程语言"(第一个 ed。的影响。2010 年 9 月 16 日检索.

16. 斯特鲁斯特鲁普，比耶恩。"c++ 编程语言"(第二个 ed。的影响。2010 年 9 月 16 日检索.

17. ^ ᵃ ᵇhttps://herbsutter.com/2016/06/30/trip–report–summer–iso–c–standards–meeting–oulu/ "C++17 后的下一个标准将 C++20"

18. "最新消息。tiobe 指数 tiobe–软件质量公司。n. p., n. d. web。2017 年 6 月 5 日。

19. 磷虾，保罗"java、c、c 面临着日益激烈的人气竞争。信息世界。infoworld, 2017 年 2 月 10 日。Web。2017 年 6 月 5 日。

20. https://www.nae.edu/177355.aspx"计算机科学先锋 bjarne stroustrup 获得 2018 年查尔斯·史坦克·德雷珀工程奖"

21. "bjarne stroustrup 的常见问题解答–" c++ "的名称从何而来？"的影响。检索 2008 年 1 月 16 日.
22. "c 代表 c++ 程序员"。东北大学。2010 年 11 月 17 日原稿 存档。检索 2015 年 9 月 7 日.
23. "iso/ieciec 14882:1998".国际标准化组织。
24. "iso/ieciec 14882:2003".国际标准化组织。
25. ^ *a b* "iso/ieciec 14882:2011".国际标准化组织。
26. "iso/ieciec 14882:2014".国际标准化组织。
27. "我们有一个国际标准: C++0x 获得一致批准"。*萨特磨坊*.
28. "c++ 的未来".
29. "我们有 C++14!: 标准 c++".
30. 行程报告: 夏季 iso c++ 标准会议（多伦多）
31. "iso/iec 译 tr 180015:2006"。国际标准化组织。
32. "iso/ieco tr 19768:2007"。国际标准化组织。
33. "iso/ieco tr 29124: 2010"。国际标准化组织。
34. "iso/ieco tr 24733:2011".国际标准化组织。
35. "iso/iec ts 18822: 2015".国际标准化组织。
36. "isoiec ts 1957: 2015".国际标准化组织。
37. "isoiec ts 1984:2015"。国际标准化组织。
38. "isoiec ts 19568:2015"。国际标准化组织。
39. "isoiec ts 19217: 2015".国际标准化组织。

40. 查看 2016 年 1 月 16 日访问 https://en.cppreference.com/w/cpp/experimental 的名单。

41. b. stroustrup (sergio de simone 访谈) (2015 年 4 月 30 日)。"斯特鲁斯特鲁普: 对 C++17 的思考--访谈"的影响。检索到 2015 年 7 月 8 日.

42. stroustrup, bjarne (2000 年)。*c++ 编程语言*(特别版)。阿迪森-韦斯利。第 46 页。国际标准书号 0-201-70073-5.

43. 斯特鲁斯特鲁普, 比耶恩。"c++ 编程语言 (第 3 版) 的未解决问题"。此代码直接从 bjarne stroustrup 的错误页面 (第 633 页) 复制。他谈到'\n'的使用, 而不是 std::endl。另请参阅 " void main ()",以了解隐式 return 0;在 main 函数中的解释吗? 此隐式返回在其他函数中*不可*用。

44. ISO/IEC.*编程语言— C++11 草稿 (n3797)* *§3.7 存储期限 [基图. stc]*

45. ISO/IEC.*编程语言— C++11 草稿 (n3797)* *§3.7.1 静态存储持续时间 [basic.stc.static]*

46. ISO/IEC.*编程语言— C++11 草稿 (n3797)* *§3.7.2 线程存储持续时间 [basic.stc.thread]*

47. ISO/IEC. *编程语言— C++11 草稿（n3797）§3.7.3 自动存储持续时间 [basic.stc.auto]*

48. ISO/IEC. *编程语言— C++11 草稿（n3797）§3.7.4 动态存储持续时间 [basic.stc.dynamic]*

49. ^ [a] [b] "没有人理解 c++：第 5 部分：模板代码 bloat"。articles.emptycrate.com/：空程序软件。旅行。东西。2008 年 5 月 6 日。2010 年 3 月 8 日 检索。有时，您会阅读或听到有人谈论导致代码膨胀的 c++ 模板。前几天我在想这个问题，心里想，"自我，如果代码做的事情完全一样，那么编译的代码就不可能真的更大了，不是吗"[...] 那么编译的代码大小呢？每一个都是用命令 g + + & lt; 文件名 & gt;. cpp-o3 编译的。非模板版本：8140 字节，模板版本：8028 字节！

50. 萨特，赫伯;亚历山大，安德烈 *c ++ 编码标准：101 条规则、准则和最佳实践*。阿迪森-韦斯利。

51. 亨里克森，马茨;nyquist，erik（1997 年）。*工业实力 c++*。普伦蒂斯·霍尔国际标准书号 0-13-12965-5.

52. stroustrup，bjarne（2000 年）。*c++ 编程语言*(特别版)。阿迪森-韦斯利。第 310 页。国际标准书号 0-201-7003-5. 虚拟成员函数有时被称为*方法*.

53. mycroft, alan (2013 年)。"c 和 c++ 异常模板" (pdf)。*剑桥计算机实验室–2013–14 课程材料*的影响。检索 2016 年 8 月 30 日.

54. stroustrup, bjarne (2013 年)。*c++ 编程语言*。艾迪生·韦斯利第 345 页。国际标准书号 9780321563842.

55. stroustrup, bjarne (2013 年)。*c++ 编程语言*。艾迪生·韦斯利第 363–365 页。国际标准书号 9780321563842.

56. stroustrup, bjarne (2013 年)。*c++ 编程语言*。艾迪生·韦斯利第 345 页，363 页。国际标准书号 9780321563842.

57. https://google.github.io/styleguide/cppguide.html#Exceptions

58. stroustrup, bjarne (2013 年)。*c++ 编程语言*。艾迪生·韦斯利第 344 页，370 页。国际标准书号 9780321563842.

59. stroustrup, bjarne (2013 年)。*c++ 编程语言*。艾迪生·韦斯利第 349 页。国际标准书号 9780321563842.

60. graziano lo russo (2008 年)。"对斯捷潘诺夫的采访"。*stlport. org* 的影响。检索 2015 年 10 月 8 日.

61. "c++ abi 摘要"。2001 年 3 月 20 日。2006 年 5 月 30 日检索.

62. "bjarne stroustrup 的常见问题解答--c 是 c++的子集吗？"的影响。检索 2014 年 5 月 5 日.

63. "c9x一新 c 标准"的影响。检索 2008 年 12 月 27 日.

64. "C++0x 海合会的支持"的影响。2010 年 10 月 12 日检索.

65. "vc10 中 C++0x 核心语言功能：表"的影响。2010 年 10 月 12 日检索.

66. "clang-C++98 、 C++11 和 C++14 状 态 " 。 Clang.llvm.org.检索 2013 年 6 月 10 日.

67. "关于：[rfc] 转换内置邮件. c 以使用更好的字符串库"(邮寄名单)。2007 年 9 月 6 日。检索 2015 年 3 月 31 日.

68. "关于：努力吸引更多的用户？"(邮寄名单)。2010 年 7 月 12 日。检索 2015 年 3 月 31 日.

69. andrew binstock (2011年5月18日)。"多布博士：采访肯·汤普森"的影响。2014 年 2 月 7 日检索.

70. ^ ᵃ ᵇ peter seibel (2009 年 9 月 16 日)。*工作中的程序员：对编程工艺的思考*。阿压特 475–476 页。国际标准书号 978-1-432-19488-4.

71. ^ ᵃ ᵇhttps://gigamonkeys.wordpress.com/2009/10/16/coders-c-plus-plus/

72. ^ a b https://www.drdobbs.com/architecture-and-design/an-interview-with-donald-knuth/228700500
73. ^ a b http://tex.loria.fr/litte/knuth-interview
74. brian kernighan (2018 年 7 月 18 日)。 *brian kernighan 问答-计算机*.
75. http://www.stroustrup.com/bs_faq.html#really-say-that
76. pike, rob (2012 年)。"少是指数多".
77. kreinin, yossi (2009年10月13日)。"有缺陷的 c++" 的影响。检索 2016 年 2 月 3 日.

进一步阅读

- 亚伯拉罕，大卫;古尔托沃伊，阿列克西 *c++ 模板元编程：来自提升和超越的概念、工具和技术*。阿迪森-韦斯利。国际标准书号 0-321-22725-5.
- 亚历山大，安德烈(2001) . *现代 c++ 设计：应用的通用编程和设计模式*。阿迪森-韦斯利。国际标准书号 0-201-70431-5.
- 亚历山德雷斯库，安德烈;萨特，赫伯 *c++ 设计和编码标准：写作程序的规则和准则*.阿迪森-韦斯利。国际标准书号 0-321-11358-6.

- becker, pete c++ *标准库扩展：教程和参考*.阿迪森–韦斯利。国际标准书号 0–321–41299–0.
- 破碎，弗兰克 (2010 年)。 *c++ 批注*。格罗宁根大学。国际标准书号 90–367–0470–7.
- coplien, james o.(1994) [重印以更正，出版原始的年 1992 年]。*高级 c++：编程样式和习语*。国际标准书号 0–2010–5485–0.
- dewhurst, stephen c. (2005 年)。*c++ 通用知识：基本的中间编程*.阿迪森–韦斯利。国际标准书号 0–321–32192–8.
- 资讯科技工业议会(2003 年 10 月 15 日)。*编程语言—c++*(第二版)。日内瓦: iso/iec。14882:2003(E)。
- josuttis, nicolai m. (2012 年)。*c++ 标准库，教程和参考*(第二版)。阿迪森–韦斯利。国际标准书号 0–321–62321–5.
- 柯尼希，安德鲁;moo, barbara e. (2000 年)。*加速 c++—通过示例进行实际编程*。阿迪森–韦斯利。国际标准书号 0–201–70353.
- lippman, stanley b.;lajoie, jose;moo, barbara e. (2011 年)。*c++ 底漆*(第五版)。阿迪森–韦斯利。国际标准书号 0–321–71411–3.

- lippman, stanley b. (1996 年)。*在 c++ 对象模型中*。阿迪森–韦斯利。国际标准书号 0–201–83454–5.
- meyers, scott(2005). *有效 c++*（第三版）。阿迪森–韦斯利。国际标准书号 0–321–33487–6.
- stroustrup, bjarne(2013). *c++ 编程语言*(第四版)。阿迪森–韦斯利。国际标准书号 978–0–321–56384–2.
- stroustrup, bjarne(1994). *c++ 的设计与演化*。阿迪森–韦斯利。国际标准书号 0–2–54330–3.
- stroustrup, bjarne(2014). 使用 *c++ 的编程原则和实践*（第二版）。阿迪森–韦斯利。国际标准书号 978–0–321–99278–9.
- 萨特，赫伯更多*特殊的 c++: 40 个新的工程难题、编程问题和解决方案*。阿迪森–韦斯利。国际标准书号 0–00–70434–x.
- 萨特，赫伯(2004). *卓越的 c++ 样式*。阿迪森–韦斯利。国际标准书号 0–201–76602–8.
- vandevoorde, david;josuttis, nicolai m. (2003 年)。*c++ 模板：完整的指南*。阿迪森–韦斯利。国际标准书号 0–201–73484–2.

外部链接

- jtctusc22/wg21— iso/ iec c++ 标准工作组

- 标准 c++ 基金会——个非营利组织，促进标准 c++ 的使用和理解。bjarne stroustrup 是该组织的主任。