C++ 标准库

C++ 标准库提供了可以在标准 C++ 中使用的各类设施。

分类

语言支持库提供了 C++ 语言中特定部分需要的组件，例如内存分配（new/delete）和异常处理。

概念库描述了 C++ 程序为进行编译期模板实参验证和根据类型的属性分派函数而可能要用到的库组件。 (C++20 起)

诊断库提供了在 C++ 程序中报告错误的统一框架，包括预定义的异常类。

内存管理库提供了内存管理所需的组件，包括智能指针和有作用域分配器 (C++11 起)。

元编程库描述了用于模板和常量求值的设施，包括类型属性，整数序列 (C++14 起)和比例算术。 (C++11 起)

通用工具库包含了其他库元素需要用到的组件，例如动态存储管理所需的预定义的存储分配器，以及 C++ 程序中的用作基础设施的组件，例如 tuple 和 (C++11 起)函数包装器。

字符串库提供对操纵以内容为以下同一类型的字符序列表示的文本的支持：`char`，`char8_t`， (C++20 起)`char16_t`，`char32_t`， (C++11 起)`wchar_t` 或其他字符类型。

容器，迭代器，范围 (C++20 起)和算法库向 C++ 程序提供了最广泛使用的算法和数据结构的一个子集。

数值库提供了数值算法和复数组件以扩展对数值处理的支持。valarray 组件提供了对同时处理多个数值的支持，它可以在支持的平台上被实现为采用并行操作。随机数组件提供了生成伪随机数的组件。 (C++11 起)

时间库提供了通用的时间工具。

本地化库为文本处理提供了国际化的扩展支持。

输入/输出库提供了用于 C++ 程序的主要输入和输出机制的 iostream 组件。它们可以和库的其他元素一起使用，尤其是字符串，本地化和迭代器。

正则表达式库提供了正则表达式的匹配和搜索。 (C++11 起)

并发支持库提供了用于创建和管理线程的组件，包括原子操作，互斥锁和线程间通信。

库的内容

除非另外说明，C++ 标准库会按 C++ 标准库标头中的大纲的描述来提供实体和宏的定义。

除了 `operator new` 和 `operator delete` 以外的所有库实体都在命名空间 `std` 或命名空间 `std` 下嵌套的命名空间中定义（C 标准库设施的实体除外，见下文）。未指明在指定命名空间中定义的名字是直接在该命名空间中定义还是在该命名空间中的内联命名空间定义。 (C++11 起)

标头

C++ 标准库中的每个元素都在 *标头* 内声明或（合适地）定义。标头不一定是源文件，标头名中 < 和 > 之间的序列也不一定是合法的源文件名。

C++ 标准库提供了 *C++ 库标头* 和提供了 C 库设施的额外 C++ 标头（具体描述见标头页面）：

C++ 库标头				
<algorithm>	<iomanip>	<list>	<queue>	<string>
<bitset>	<ios>	<locale>	<set>	<strstream>
<complex>	<iosfwd>	<map>	<sstream>	<typeinfo>
<deque>	<iostream>	<memory>	<stack>	<utility>
<exception>	<istream>	<new>	<stdexcept>	<valarray>
<fstream>	<iterator>	<numeric>	<streambuf>	<vector>
<functional>	<limits>	<ostream>		
C++11 中添加的标头				
<array>	<condition_variable>	<mutex>	<scoped_allocator>	<type_traits>
<atomic>	<forward_list>	<random>	<system_error>	<typeindex>
<chrono>	<future>	<ratio>	<thread>	<unordered_map>
<codecvt>	<initializer_list>	<regex>	<tuple>	<unordered_set>
C++14 中添加的标头				
<shared_mutex>				
C++17 中添加的标头				
<any>	<filesystem>	<optional>	<string_view>	<variant>
<execution>	<memory_resource>			
C++20 中添加的标头				
<barrier>	<concepts>	<latch>	<semaphore>	<stop_token>
<bit>	<coroutine>	<numbers>	<source_location>	<syncstream>
<charconv>	<format>	<ranges>		<version>
<compare>				
C++23 中添加的标头				
<expected>	<flat_set>	<mdspan>	<spanstream>	<stdfloat>
<flat_map>	<generator>	<print>	<stacktrace>	

提供了 C 库设施的 C++ 标头			
<cassert>	<clocale>	<cstdarg>	<cstring>
<cctype>	<cmath>	<cstddef>	<ctime>
<cerrno>	<csetjmp>	<cstdio>	<cwchar>
<cfloat>	<csignal>	<cstdlib>	<cwctype>
<climits>			
C++20 中移除的标头			
<ciso646>			
C++11 中添加的标头			
<cfenv>	<cinttypes>	<cstdint>	<cuchar>
C++11 中添加，C++17 中弃用，C++20 中移除的标头			
<ccomplex>	<cstdalign>	<cstdbool>	<ctgmath>

自立实现有由实现定义的一套标头，对这套标头的最低限度要求参考此处。

C 标准库

使用 C++ 标准库也可以使用 C 标准库的设施，它们经适当调整以确保静态类型安全。对于这些库函数的描述大多依赖于它们在 C 标准库中的语义。

在某些情况下，标准 C++ 中指定的签名可能会与在 C 标准库中的签名不同，并且可能会声明额外的重载，但是没有另外说明的情况下它们的行为和前提条件不变。

为了与 C 标准库的兼容性，C++ 标准库提供了以下 C 标头。这些标头应该只在需要考虑互用性的时候使用。C++ 源文件可能需要包含这些标头其中之一以成为合法的 ISO C 源文件。没有考虑成为合法的 ISO C 源文件的源文件不应使用这些 C 标头。具体描述参考此处。

C 标头			
<assert.h>	<limits.h>	<stdarg.h>	<string.h>
<ctype.h>	<locale.h>	<stddef.h>	<time.h>
<errno.h>	<math.h>	<stdio.h>	<wchar.h>
<float.h>	<setjmp.h>	<stdlib.h>	<wctype.h>
<iso646.h>	<signal.h>		
C++11 中添加的标头			
<complex.h>	<inttypes.h>	<stdbool.h>	<tgmath.h>
<fenv.h>	<stdalign.h>	<stdint.h>	<uchar.h>
C++23 中添加的标头			
<stdatomic.h>			

除非另外说明，标头 c*xxx* 中的内容与 C 标准库中对应的标头 *xxx*.h 一致。但是在 C++ 标准库中，这些声明（除了在 C 中被定义为宏的名字）在命名空间 `std` 的命名空间作用域中。未指明这些名字（包括添加的重载）是否在全局命名空间作用域首次声明然后通过显式 using 声明注入命名空间 `std`。

在 C 中定义为宏的名字（`assert`，`offsetof`，`setjmp`，`va_arg`，`va_end` 和 `va_start`）在 C++ 标准库中也必须定义为宏，即使 C 允许实现将它们定义为函数。

在 C 中定义为函数的名字在 C++ 标准库中也必须定义为函数，即使在 C 中在函数原型外也可以为它们提供掩蔽宏。在 C++ 中唯一可以达成等价的内联行为的方法是将它们定义成 extern 内联函数。

在 C++ 是关键词或运算符的标识符不能在 C++ 标准库标头中定义为宏。特别是包含标准标头 <iso646.h> 没有任何效果。

与标准 C 中的安全函数关联的名字 (C++17 起)

只要包含了任何 C++ 标头，是否在全局命名空间声明以下在 C 标准附件 K 中的名字由实现定义（它们不会在命名空间 `std` 中声明）：

C 标准附件 K 中的名字			
abort_handler_s	mbstowcs_s	strncat_s	vswscanf_s
asctime_s	memcpy_s	strncpy_s	vwprintf_s
bsearch_s	memmove_s	strtok_s	vwscanf_s
constraint_handler_t	memset_s	swprintf_s	wcrtomb_s
ctime_s	printf_s	swscanf_s	wcscat_s
errno_t	qsort_s	tmpfile_s	wcscpy_s
fopen_s	RSIZE_MAX	TMP_MAX_S	wcsncat_s
fprintf_s	rsize_t	tmpnam_s	wcsncpy_s
freopen_s	scanf_s	vfprintf_s	wcsnlen_s
fscanf_s	set_constraint_handler_s	vfscanf_s	wcsrtombs_s
fwprintf_s	snprintf_s	vfwprintf_s	wcstok_s
fwscanf_s	snwprintf_s	vfwscanf_s	wcstombs_s
gets_s	sscanf_s	vprintf_s	wmemcpy_s
gmtime_s	mbstowcs_s	vscanf_s	vswscanf_s
abort_handler_s	strcat_s	vsnprintf_s	wmemmove
ignore_handler_s	strcpy_s	vsnwprintf_s	wprintf_s
localtime_s	strerrorlen_s	vsprintf_s	wscanf_s
L_tmpnam_s	strerror_s	vsscanf_s	
mbsrtowcs_s	strlen_s	vswprintf_s	

使用库

包含标头

C++ 标准库中的实体在标头中定义，翻译单元中有合适的 #include 预处理指令时可以使标头中的内容可以用于该翻译单元（即包含该标头）。

翻译单元可以以任何顺序包含库标头。每个标头可以被包含多次，效果和只包含一次该标头一致。除非该效果是包含 <cassert> 或 <assert.h>，此时效果依赖于当前语法位置 `NDEBUG` 的定义。

翻译单元只能在定义或声明外包含标头，并且该标头内声明的实体只能在语法中这次包含之后引用。不需要诊断。

在模块单元中，只能在全局模块片段包含标头。 (C++20 起)

导入标头

C++ 库标头，或者对于自立实现，由实现提供的这些标头的子集，被统称为 *可导入的 C++ 库标头* 。 (C++20 起)

翻译单元中有合适的导入声明时可以使可导入的 C++ 库标头中的内容可以用于该翻译单元（即导入该标头）。

导入模块

C++ 标准库提供以下 *C++ 库模块* ：

- 具名模块 `std` 会导出由可导入的 C++ 库标头或提供了 C 库设施的额外 C++ 标头提供的在命名空间 `std` 中的声明（例如 <bit> 的 `std::rotr` 以及 <cstdio> 的 `std::puts`）。它还会导出由 <new> 提供的存储分配和解分配函数在全局命名空间的声明（例如 `::operator new`）。
- 具名模块 `std.compat` 会导出与具名模块 `std` 相同的声明，外加与由提供了 C 库设施的额外 C++ 标头提供的在命名空间 `std` 中的声明相对应的在全局命名空间中的声明（例如 `::fclose`）。 (C++23 起)

对于标准库中的每个声明：

- 未指明它隶属于哪个模块。
- 无论它是因为包含标头，导入标头单元还是导入 C++ 库模块而变得可及，它都会表示相同的实体。

功能特性测试宏	值	标准
`__cpp_lib_modules`	`202207L`	(C++23)

链接

C++ 标准库中的实体具有外部链接。除非另外说明，对象和函数默认具有 `extern "C++"` 链接。

来自 C 标准库并且声明为具有外部链接的名字具有 `extern "C"` 还是 `extern "C++"` 链接由实现定义。C++ 标准推荐在这种情况下使用 `extern "C++"`。

在库中定义且由 C++ 程序所需的对象和函数会在程序启动前包含到程序中。

标准库实现的要求

保证

每个 C++ 标头必须提供在以下地方出现的声明和定义：

- 该标头的概要中
- 在该标头的概要中它包含的所有其他标头的概要中

对于在多个标头中定义的类型或宏（例如 NULL），以任意顺序包含任意数量的这些标头都不会违反单一定义规则。

除非另外说明，所有由 C 标准库定义且展开到整型常量表达式的仿对象宏可以用于#if 预处理指令。

调用标准库的任意一个非成员函数签名实际上都会调用那个函数。因此符合标准的标准库实现都不能另外定义可能会被合法 C++ 程序调用的非成员函数。

不能将非成员函数签名声明为具有额外的默认实参。

除非另外说明，标准库中的函数在调用非运算符非成员函数时不会通过实参依赖查找使用来自其他命名空间的函数。

对于类（模板）定义中的每个函数（模板）友元声明，不会提供该函数（模板）的其他声明。

标准库函数签名只有在需要是 constexpr 的情况下才能声明为 `constexpr`。如果有标头提供了 constexpr 函数（包括构造函数）的非定义的声明，那么它也需要提供对应的定义。

除非另外说明，每个标准库函数应该满足下列各项要求以避免数据竞争：

- 除了通过函数实参，C++ 标准库函数不能（直接或间接地）访问能被当前线程以外的线程访问到的对象，包括 `this`。
- 除了通过非 const 函数实参，C++ 标准库函数不能（直接或间接地）修改能被当前线程以外的线程访问到的对象，包括 `this`。
  - 例如在没有进行同步处理的情况下，不能在内部使用具有静态存储期的对象，因为没有显式在线程间共享对象的程序也可能会因此产生数据竞争。 (C++11 起)
- C++ 标准库函数不能访问能通过它的实参或它的容器实参的元素间接访问到的对象，除非通过对容器元素调用该标准库函数的文档描述中要求调用的函数。
- 对通过调用标准库容器或字符串的成员函数获得的迭代器的操作可以访问底层容器，但不能修改它。
  - 尤其是会使迭代器失效的容器操作会和与该容器关联的迭代器操作冲突。
- C++ 标准库执行的所有对用户可见的操作都只能在当前线程进行。
  - 副作用不可见的操作可以并行处理。

对于每个在 C++ 标准库定义且要求从另一个在 C++ 标准库定义的类派生的类：

- 该基类在指定为 `virtual` 时必须是虚基类
- 该基类在没有指定为 `virtual` 时不能是虚基类
- 除非另外说明，名字不同的类型只能是不同的类型

除非另外说明，在 C++ 标准库中指定的类型都不能是指定为 final 的类类型。 (C++11 起)

如果在 C++ 标准库中定义的函数指定为需要（在特定情况下）抛出某个指定类型的异常，那么抛出的异常只能具有那个类型或者从那个类型派生的类型，这样对应那个基类类型的异常处理块才能捕获该异常。

C 标准库中的函数不能抛出异常，除非该函数调用了由程序提供的函数（`qsort()` 和 `bsearch()` 满足这种情况），并且那个函数抛出了异常。

C++ 标准库中定义的析构操作不会抛出异常。C++ 标准库的所有析构函数的行为等同于它具有不抛出的异常说明。

如果 C++ 标准库中的函数通过 `std::error_code` 对象报告错误，那么该对象的 `category()` 成员在错误来自操作系统时必须返回 `std::system_category()`，或者在错误来自其他地方时必须返回到由实现定义的 `std::error_category` 对象的引用。对于这些错误类别，需要定义每个类别中 `value()` 的所有可能的值。

具有在 C++ 标准库中定义的类型的对象可以被移动。移动操作可以显式指明或隐式生成。除非另外说明，这类被移动后的对象会处于合法但未指明的状态。 (C++11 起)

具有在 C++ 标准库中定义的类型的对象可以移动赋值到自身。除非另外说明，这类移动赋值后的对象会处于合法但未指明的状态。

实现自由处理

未指明 C++ 标准库中定义的成员和非成员函数是否会被定义成内联函数。

对于非虚的 C++ 标准库成员函数，可以声明一组不同的成员函数签名，只要对该成员函数的任何调用在应当选中原函数签名组中的某个重载的情况下的行为与选中该重载本身的行为一致。这样就可以进行以下调整：

- 向形参添加默认实参
- 将有默认实参的成员函数替换成多个行为等价的成员函数
- 对成员函数名添加额外的签名

除非另外说明，由实现定义 C++ 标准库的哪些函数可以递归地重新进入。

C++ 标准库实现可以在线程间分享自己内部使用的对象，前提是这些对象对用户不可见且会受保护以避免数据竞争。 (C++11 起)

未指明 C++ 标准库的哪些函数签名和类是 C++ 标准库的另一个类的友元。

这里描述的名字和全局函数签名为实现保留。

C++ 标准库的类可以从一个具有为实现保留的名字的类派生。如果一个在 C++ 标准库定义的类要求从另一个在 C++ 标准库定义的类派生，那么该派生类既可以直接从要求的基类派生，也可以间接通过一系列具有为实现保留的名字的基类派生。

如果在 C++ 标准库中定义的函数没有指定为需要（在特定情况下）抛出异常但该函数不具有不抛出的异常说明，那么抛出的异常由实现定义，但异常类型必须是 `std::exception` 或者从 `std::exception` 派生的任意类型。

非虚函数的异常说明可以通过添加不抛出的异常说明来强化。

缺陷报告

下列更改行为的缺陷报告追溯地应用于以前出版的 C++ 标准。

缺陷报告	应用于	出版时的行为	正确行为
LWG 1 (https://cplusplus.github.io/LWG/issue1)	C++98	未指明来自 C 标准库并且声明为具有外部链接的名字的语言链接	由实现定义
LWG 119 (https://cplusplus.github.io/LWG/issue119)	C++98	可以强化虚函数的异常说明	仅限非虚函数
LWG 147 (https://cplusplus.github.io/LWG/issue147)	C++98	关于非成员函数的说明只考虑了全局函数	也会考虑非全局函数
LWG 225 (https://cplusplus.github.io/LWG/issue225)	C++98	标准库函数可能会因为实参依赖查找而调用其他命名空间的非成员函数	已禁止（除非另外说明）
LWG 336 (https://cplusplus.github.io/LWG/issue336)	C++98	`<strstream>` 不是 C++ 库标头	它是 C++ 库标头
LWG 343 (https://cplusplus.github.io/LWG/issue343)	C++98	未指定库标头的依赖	已（在概要中）指定
LWG 456 (https://cplusplus.github.io/LWG/issue456)	C++98	提供了 C 库设施的 C++ 标头只能在命名空间 `std` 中提供定义	可以在全局命名空间提供， 然后注入命名空间 `std`
LWG 465 (https://cplusplus.github.io/LWG/issue465)	C++98	在 C++ 中是关键词或运算符的标识符可以在 C++ 标准库标头中定义为宏（只有 `<ciso646>` 不允许）	所有 C++ 标准库标头中 都不能将它们定义为宏
LWG 1178 (https://cplusplus.github.io/LWG/issue1178)	C++98	C++ 标头需要包含其他提供了所需定义的标头	C++ 标头需要提供直接或间接地 包含在它的概要中的声明和定义
LWG 2013 (https://cplusplus.github.io/LWG/issue2013)	C++11	未指定标准库是否可以将在标准中未要求 是 constexpr 的函数声明为 `constexpr`	已禁止

来自“https://zh.cppreference.com/mwiki/index.php?title=cpp/standard_library&oldid=78517”