NatsuLang规范

1. 范围

这个文档说明了对NatsuLang的实现的需求，并定义了NatsuLang的概念及规范。

1. 术语与定义

仅在较小区域使用的术语将在第一次使用时就地定义。

* 1. 访问

（执行期行为）读取或修改一个对象的值。

* 1. 参数

（函数调用表达式）括号中由逗号分隔的列表中的表达式。

* 1. 参数

（抛出表达式）throw的操作数。

* 1. 阻塞

线程执行时等待某些状态被满足后再继续执行阻塞操作之后的操作。

* 1. 选择性支持

实现不需要支持的功能。

* 1. 诊断信息

一些实现输出的信息，内容由实现定义。

* 1. 实现定义行为

指合式的程序的决定于实现的行为，这个行为应当在实现的文档中说明。

* 1. 实现限制

由实现施加给程序的限制。

* 1. 签名

（函数）名称、参数类型列表、以及包含它的模块。

[注释：签名作为连接等的基础——注释结束]

* 1. 未定义行为

标准未施加任何要求的行为。

* 1. 未说明行为

指合式的程序的决定于实现的行为，实现不必说明这个行为。

* 1. 合式的程序

遵守语法规则、可诊断的语义规则的NatsuLang程序。

* 1. 病式的程序

并非合式的程序（2.12）。

1. 一般原则
   1. 实现承诺
      1. 可诊断规则包括本规范所有的句法与语义规则，除了包括显式标记为“不需要诊断”或者被描述为造成“未定义行为”的规则。
      2. 由标准库定义的名称位于std模块中，翻译单元通过引用这个模块以访问这些名称。
      3. 一个遵守规则的实现可能具有一些提供了不修改任何合式的程序的行为的扩展。实现应当诊断使用了这样的扩展的程序是否根据本规范是非合式的。之后可以编译和运行这样的程序。
      4. 所有的实现应当提供包含了所有不支持的可选支持功能的文档。
   2. 语法标记
      1. 在本文档所使用的标记中，语法类别将使用*斜体*指示，字面的词语及字符将使用定宽字体。可选项列在独立的行中，或者在由“以下其一”表示的下一行之中列出，如果可选项的文字过长并超出一行，则文字将会在下一行继续，并且开头多出一个缩进。可选的终结或非终结符号将会由“可选”下标指示。
      2. 语法分类的名称一般将会由以下的规则命名：
         1. ——*X名称* 是一个标识符，在当前上下文中决定它的含义（例：*类型名称*）。
         2. ——*X标识符* 是一个具有不依赖上下文的含义的标识符（例：*限定标识符*）。
         3. ——*X序列* 表示一个或多个*X*，且无中间分隔符（例：*声明序列* 表示一个声明的序列）。
         4. ——*X列表* 表示一个或多个*X*，且由中间分隔符分开（例：*标识符列表* 表示一个由逗号分隔开的标识符的序列）。
2. 词法约定
   1. 分离翻译
      1. 程序的文本被称为源代码，输入的源代码作为一个整体称为翻译单元。
   2. 翻译阶段
      1. 翻译的语法规则优先级由下列的阶段说明：
         1. 实现以实现定义的方式读入源代码，并映射到*基本源码字符集*。实现接受的源代码字符集由实现定义。
         2. 分解源代码到记号。
         3. 转换字符字面量及字符串字面量中的源码字符到执行字符集中的对应字符，若无法完成这样的转换，将会以实现定义的执行字符集的字符表示，但保证不会转换为空字符。
         4. 分析经过转换得到的记号，对于变量声明及函数声明获得声明的名称，缓存声明中的类型及初始化器信息，对于其他声明进行完全分析，缓存所有编译器动作。
         5. 所有源代码处理完毕后，分析所有已缓存的编译器动作及声明的类型及初始化器信息，在分析过程中若遇到引用已缓存的变量声明，不构成循环引用的情况下会递归地进行分析，对于函数声明允许循环引用。
   3. 字符集
      1. *基本源码字符集*包含98个字符：空格字符，表示横向缩进、纵向缩进、换页以及换行的控制字符，以及以下93个图形字符：

a b c d e f g h I j k l m n o p q r s t u v w x y z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

\_ { } [ ] # ( ) < > % : ; . ? \* + - / ^ & | ~ ! = , \ “ ‘ $ @

* + 1. *基本执行字符集*必须包含基本源码字符集的所有成员，包括表示警告、退格、以及硬回车的控制字符，以及空字符，其值为0。对于基本执行字符集，其中的每一个成员的值都必须非负并且与其他的成员的值互不相同。*执行字符集*是实现定义的基本执行字符集的超集。
  1. 记号

记号：

标识符

关键字

字面量

操作符

符号

* + 1. 记号分为5种：标识符、关键字、字面量、操作符以及符号。
  1. 注释
     1. 注释可以以/\*开始，以\*/结束，这被称为块注释。注释也可以以//开始，以换行符结束，这被称为行注释。[注释：注释无法嵌套，这意味着在以某种形式开始的注释中总是以遇到的第一个该形式定义的结束模式作为注释的结束，而在结束之前无法创建新的注释。——注释结束]
  2. 标识符

标识符：

非数字的标识符

标识符 非数字的标识符

标识符 数字字符

非数字的标识符：

非数字字符

非数字字符：以下其一

a b c d e f g h i j k l m

n o p q r s t u v w x y z

A B C D E F G H I J K L M

N O P Q R S T U V W X Y Z \_

数字字符：以下其一

0 1 2 3 4 5 6 7 8 9

* + 1. 标识符是一个任意长度的字母及数字的序列。大小写字母被视为不同的。
  1. 关键字
     1. 以下标识符被保留用于作为关键字。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| as | const | float | int128 | template | uint128 |
| auto | continue | float128 | long | this | ulong |
| bool | def | for | longdouble | throw | ulonglong |
| break | double | goto | longlong | true | ushort |
| catch | else | if | module | try | void |
| char | extern | import | return | typeof | while |
| class | false | int | short | uint |  |

* 1. 操作符和符号

操作符和符号：以下其一

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [ | … | ++ | ! | << | ^ | ; |
| ] | & | += | != | <= | ^= | = |
| ( | && | - | / | <<= | | | == |
| ) | &= | -- | /= | > | || | , |
| { | \* | -= | % | >> | |= | # |
| } | \*= | -> | %= | >= | ? | $ |
| . | + | ~ | < | >>= | : | @ |

* 1. 字面量
     1. 字面量的类型

字面量：

整数字面量

字符字面量

浮点字面量

字符串字面量

逻辑字面量

* + 1. 整数字面量

整数字面量：

二进制字面量 整数后缀可选

八进制字面量 整数后缀可选

十进制字面量 整数后缀可选

十六进制字面量 整数后缀可选

二进制字面量：

0b 二进制数字

0B 二进制数字

二进制字面量 二进制数字

八进制字面量：

0

八进制字面量 八进制数字

十进制字面量：

非0数字

十进制字面量 数字

十六进制字面量：

十六进制前缀 十六进制数字序列

二进制数字：

0

1

八进制数字：以下其一

0 1 2 3 4 5 6 7

非0数字：以下其一

1 2 3 4 5 6 7 8 9

十六进制前缀：以下其一

0x 0X

十六进制数字序列：

十六进制数字

十六进制数字序列 十六进制数字

十六进制数字：以下其一

0 1 2 3 4 5 6 7 8 9

a b c d e f

A B C D E F

整数后缀：

无符号型后缀 长型后缀

无符号型后缀 长长型后缀

长型后缀 无符号型后缀

长长型后缀 无符号型后缀

无符号型后缀：以下其一

u U

长型后缀：以下其一

l L

长长型后缀：以下其一

ll LL

* + - 1. 整数字面量是没有小数点以及指数部分的数字的序列。整数字面量可能具有一个前缀表示它的基数，以及一个后缀表示它的类型。二进制整数字面量以0b或者0B开头、由二进制数字的序列组成。八进制整数字面量以0开头、由八进制数字的序列组成。十进制整数字面量由除0以外的数字开头、由十进制数字的序列组成。十六进制整数字面量由0x或者0X开头、由十六进制数字的序列组成。
      2. 整数字面量的类型是以下相应的列表中第一个可以表示它的值的类型。

|  |  |  |
| --- | --- | --- |
| 后缀 | 十进制字面量 | 二进制、八进制或者十六进制字面量 |
| 无 | int  long  longlong | int  uint  long  ulong  longlong  ulonglong |
| u或U | uint  ulong  ulonglong | uint  ulong  ulonglong |
| l或L | long  longlong | long  ulong  longlong  ulonglong |
| 同时u或U及l或L | ulong  ulonglong | ulong  ulonglong |
| ll或LL | longlong | longlong  ulonglong |
| 同时u或U及ll或LL | ulonglong | ulonglong |

* + - 1. 如果整数字面量无法被列表中任何类型所表示，但是有扩展的整数类型可以表示它的值，则字面量可能具有该扩展整数类型。如果列表中所有类型都是有符号的，则扩展的整数类型也应当是有符号的。如果列表中所有类型都是无符号的，则扩展的整数类型也应当是无符号的。如果列表中既包含有符号的又包括无符号的类型，则扩展的整数类型可能是有符号或者无符号的。如果翻译单元中包含无法被任何可行类型表示的整数字面量，则程序为病式的。
    1. 字符字面量

字符字面量：

‘字符’

字符：

任何字符除了单括号’、反斜杠或者换行符

转义字符序列

转义字符序列：

简单转义字符序列

八进制转义字符序列

十六进制转义字符序列

简单转义字符序列：以下其一

\’ \” \? \\

\a \b \f \n \r \t \v

八进制转义字符序列：

\ 八进制数字

\ 八进制数字 八进制数字

\ 八进制数字 八进制数字 八进制数字

十六进制转义字符序列：

\x 十六进制数字

十六进制转义序列 十六进制数字

* + - 1. 字符字面量是一个被单括号包住的字符。它包含单个char类型可表示的字符，并具有类型char。
      2. 某些无法打印的字符、单引号、双引号、问号以及反斜杠可以如表格3表示。在字符字面量中单引号、反斜杠以及换行字符必须以转义序列表示。由反斜杠接单个或多个字符，但没有由表格3列出的由实现可选支持，具有实现定义的语义。转义序列表示单个字符。
      3. 转义序列\ooo由反斜杠接1、2或3个八进制数字组成，ooo用于表示转义到的字符的值。转义序列\xhhh由反斜杠接1个或多个十六进制数字组成，hhh用于表示转义到的字符的值。对于hhh中十六进制数字的个数没有限制。八进制数字和十六进制的序列由第一个并非八进制数字或十六进制数字的字符终止。如果字符字面量的值不在char的可表示范围内，则值为实现定义的。

|  |  |  |
| --- | --- | --- |
| 换行符 | LF | \n |
| 水平制表符 | HT | \t |
| 垂直制表符 | VT | \v |
| 退格符 | BS | \b |
| 回车符 | CR | \r |
| 换页符 | FF | \f |
| 响铃 | BEL | \a |
| 反斜杠 | \ | \\ |
| 问号 | ? | \? |
| 单引号 | ’ | \’ |
| 双引号 | ” | \” |
| 八进制数字 | ooo | \ooo |
| 十六进制数字 | hhh | \xhhh |

* + 1. 浮点字面量

浮点字面量：

十进制浮点字面量

十六进制浮点字面量

十进制浮点字面量：

小数部分 指数部分可选 浮点后缀可选

十六进制浮点字面量：

十六进制前缀 十六进制小数部分 二进制指数部分 浮点后缀可选

小数部分：

数字序列

数字序列 . 数字序列

十六进制小数部分：

十六进制数字序列

十六进制数字序列 . 十六进制数字序列

指数部分：

e 符号可选 数字序列

E 符号可选 数字序列

二进制指数部分：

p 符号可选 数字序列

P 符号可选 数字序列

符号：以下其一

+ -

数字序列：

数字

数字序列 数字

浮点后缀：以下其一

f l F L

* + - 1. 浮点字面量由一个可选的表示基数的前缀、一个整数部分、一个可选的小数点及小数部分、一个e、E、p或P、一个可选的有符号整数指数以及一个可选的类型后缀。若无前缀，整数和小数部分都由一序列十进制数字组成，若前缀为0x或0X则由一序列十六进制数字组成。对于十进制浮点数指数部分是可选的。若无后缀则字面量的类型为double，若后缀为f或F则是float，若后缀为l或L则是longdouble。若结果的值不在该类型的可表示范围内时，程序将是病式的。
    1. 字符串字面量

字符串字面量：

“ 字符序列可选 “

字符序列：

字符

字符序列 字符

字符：

除了双括号”、反斜杠\和换行符以外的任何源码字符集

转义序列

* + - 1. 字符串字面量是由双括号括起的子序列字符。
    1. 逻辑字面量

逻辑字面量：

false

true

* + - 1. 逻辑字面量是关键字false和true。具有类型bool。

1. 基本概念
   1. 条款说明
      1. [注释：本条款介绍了NatsuLang语言的基本概念，它解释了对象和名称的区别，并介绍了声明和定义以及类型、范围、链接性和存储期的概念，论述了开始以及终结一个程序的机制。本条款的最后介绍了语言的基本类型以及它们的组合类型。——注释结束]
      2. *实体*是一个值、对象、函数、枚举值、类型、类成员或者模块。
      3. *名称*是一个表示了一个实体或者标签的*标识符*。
      4. 表示实体的名称将由*声明*引入。表示标签的名称将由goto语句或者带标签语句引入。
      5. *变量*由对象的声明引入。若变量的名称存在，则该名称表示引用的对象。
      6. 若表示名称的*标识符*由相同的字符序列组成，则两个名称是相同的。
      7. 在超过一个翻译单元中使用的名称可能潜在地引用了同一个实体，这个行为依赖于它的链接性。
   2. 声明和定义
      1. 声明可能引入一个或多个名称到翻译单元中，或者重新声明由之前的声明引入的名称。如果以上行为发生了，声明将说明这些名称的解读方式。
      2. 声明同时也是定义。
   3. 范围
      1. 定义域和范围
         1. 名称仅能在*可声明域*内声明。
         2. 由声明引入的名称从声明开始生效。
         3. 在相同的可声明域所有具有相同的非限定名称将全部引用相同的实体。
      2. 声明点
         1. 一个名称的*声明点*，从它完整的声明符之后、*初始化器*之前立刻开始。
         2. [注释：来自外部范围的名称将保持可见，直到出现隐藏了它的名称为止。

[示例：

def i = 2;

{

def a = i;

def i = 3;

}

内部域中的a具有初值2。——示例结束]——注释结束]

* + - 1. 枚举值的*声明点*从它的声明符定义之后立刻开始。
    1. 块范围
       1. 在块中的名称对于该块是本地的，它具有*块范围*。它的潜在范围从它的声明点开始，并且在块结束之时结束，在块范围中声明的变量是*本地变量*。
       2. 函数参数名称的潜在范围从它的*声明点*开始，在函数定义的块结束之时结束。
    2. 函数参数范围
       1. 函数参数具有函数参数范围。参数的潜在范围自它的声明点开始，止于当前函数声明符的结尾。
    3. 函数范围
       1. 只有标签具有函数范围，并且可以在函数的任何地方使用它们。
    4. 模块范围
       1. 模块的声明范围是模块体。声明在模块体内的实体是模块的成员，由这些声明引入声明范围的名称是模块的成员名称。模块的成员名称具有模块范围。它的潜在范围是整个模块的声明域。
       2. 模块成员可以由.成员访问符引用。
       3. 最外部的声明域也是一个模块，称为全局模块。声明在全局模块的名称具有全局模块范围或称全局范围。这样的名称的潜在范围是整个翻译单元的声明域。具有全局模块范围的名称被称为全局名称。
    5. 类范围
       1. 声明于类内的名称的潜在范围是类的整个声明域。
       2. 类成员的名称仅能以以下方式被使用：
          1. ——在类内使用
          2. ——在应用在该类或该类的实例或指针的.操作符后使用
    6. 枚举范围
       1. 枚举符具有枚举范围。它的潜在范围自它的声明点开始，至枚举的声明结束为止。
    7. 名称隐藏
       1. 名称可以被位于嵌套的声明域中的相同的名称隐藏。
       2. 若名称在当前范围内且未被隐藏，则被称为可见的。
  1. 名称查找
     1. 概述
        1. 名称查找规则对于所有名称相同，被查找到的名称应当总是表示相同的实体。
        2. 在表达式的上下文中查找到的名称指的是以非限定名在该表达式所在的范围中查找。
     2. 非限定名称查找
        1. 名称将会按照从开始查找点逐渐沿嵌套范围向外查找。如果无法查找到任何名称，则程序为病式的。
     3. 限定名称查找
        1. 概述
           1. 类或模块的成员或枚举符名称可以由应用在表示类、模块或枚举的嵌套名称说明符上的.成员访问操作符引用。如果无法查找到任何名称，则程序为病式的。
        2. 类成员
           1. 类成员名称若被隐藏，可以使用类名称及.操作符访问。
        3. 模块成员
           1. 模块成员名称若被隐藏，可以使用模块名称及.操作符访问。
  2. 程序和链接
     1. *程序*由一个或多个*翻译单元*链接而成。翻译单元由一序列声明组成：

翻译单元：

声明序列可选

* + 1. 若名称可能用于表示相同的实体、类型、模块或值等则被称为具有链接性：
       1. ——当名称具有外部链接性时，它所表示的实体可以在所定义的翻译单元及以外的翻译单元内的其他范围内被引用
       2. ——当名称具有内部链接性时，它所表示的实体仅能在当前定义的翻译单元的其他范围内被引用
       3. ——当名称无链接性时，它所表示的实体不能在其他范围内引用。
    2. 在块范围内以extern声明引入的名称可以具有链接性。如果没有或有超过一个符合要求的实体被找到时，程序是病式的。
  1. 启动和结束
     1. 入口函数
        1. 以Entry属性标识的函数是入口函数，仅能有一个函数被标识为入口函数，这样的函数不能接受参数及返回值（即返回类型应当为void）。执行程序时将会启动主线程执行入口函数。
     2. 结束
        1. 具有静态存储期的对象的析构函数将会在退出时以构造函数执行的反向顺序被执行。
        2. 具有自动存储期的对象的析构函数将会在离开当前域的时候以构造函数执行的反向顺序被执行。
  2. 存储期
     1. 概述
        1. 存储期时对象的一个定义了包含对象的存储的最小潜在生命期的属性。存储器由创建对象的方式决定：
           1. ——静态存储期
           2. ——自动存储期
           3. ——动态存储期
        2. 当存储期结束时，所有指向该存储的任何部分的指针都变为无效指针值。
     2. 静态存储期
        1. 所有不具有动态存储期的非局部变量都具有静态存储期。这样的实体的存储必须在整个程序的执行期间都有效。
     3. 自动存储期
        1. 未以extern声明的块范围变量具有自动存储期。这样的实体的存储至少在创建它们的块退出之前都有效。
        2. 若具有自动存储期的变量的初始化或析构具有副作用，则实现不得在块结束之前销毁它，更不能直接消除它。
     4. 动态存储期
        1. [注释：此部分未完成。——注释结束]
     5. 子对象的存储期
        1. 子对象的存储期与包含它的对象的存储期相同。
  3. 对象生存期
     1. 对象的生存期是对象的一个运行期属性。对象的生存期自初始化完成开始，至析构执行完成结束。
  4. 类型
     1. 初等类型
        1. 具有类型char的对象应当对于存储任何该实现的基本字符集都是足够大的。如果该对象存储了字符集中的字符，则该对象的值与该字符的字面量形式的值相等。char类型被视为无符号的。char类型的对象至少可以容纳0到255的表示。
        2. *标准有符号整数类型*有4种：“short”、“int”、“long”、“longlong”。允许存在实现定义的*扩展有符号整数类型*。标准和扩展有符号整数类型被统称为*有符号整数类型*。
        3. 对于每个标准有符号整数类型都有对应的*标准无符号整数类型*：“ushort”、“uint”、“ulong”，“ulonglong”，对应的每组整数类型都占有相同大小的存储，并且具有相同的对齐要求，以及相同的对象表示。对于每个扩展有符号整数类型也存在对应的扩展无符号整数类型。标准有符号整数类型和扩展有符号整数类型被统称为*无符号整数类型*。有符号整数类型的非负数表示范围是对应的无符号整数类型的子范围，对于具有相同的值的一对整数类型，它们具有相同的对象表示。标准有符号整数类型及标准无符号整数类型被统称为*标准整数类型*，扩展有符号整数类型及扩展无符号整数类型被统称为*扩展整数类型*。
        4. 类型bool的值只能为true或false。
        5. 类型bool、char以及有符号无符号整数类型被统称为*整数类型*。整数类型的表示必须以二进制命数系统定义值。
        6. *浮点类型*有3种：“float”、“double”、“longdouble”。浮点类型的表示遵守IEEE 754标准（IEC 60559:1989）中所定义的单精确度、双精确度表示。“longdouble”的表示为实现定义的。整数和浮点类型被统称为算术类型。
        7. 类型void不允许为对象具有的类型，不能表示任何值。当void作为函数返回类型时表示函数不返回任何值。
        8. [注释：即使标准定义了两个或多个基本类型具有相同的值表示，它们也是不同的类型。——注释结束]
     2. 组合类型
        1. 条款说明
           1. 组合类型可以被下列方式构造：

——*指针*，其指向的对象具有指定的类型；

——*数组*，其元素具有指定的类型；

——*函数*，具有指定类型的参数，并且返回void或者指定类型的对象；

——*类*，包含多种类型的对象的序列，多种类型、枚举和用于操作这些对象的函数的集合，以及对访问这些实体的限制的集合；

——*枚举*，具有多个有名称的常量的集合；

* + - * 1. 构造组合类型的方式可以被递归地应用。
      1. 指针类型

指针类型：

类型名称 \*

* + - 1. 数组类型

数组类型：

类型名称 [ 常量表达式 ]

* + - 1. 函数类型

函数类型：

( 参数声明列表 ) 返回类型说明可选

参数声明列表：

参数声明

参数声明列表 , 参数声明

参数声明：

声明符

* + - 1. 类类型
         1. 见10。
      2. 枚举类型
  1. 对齐

1. 标准转换
   1. 条款说明
      1. 标准转换是内建的隐式转换。
   2. 整数转换
      1. 具有整数类型的值可以转换到另一个整数类型的值。
      2. 如果目标类型无法容纳原始类型表示的值，结果将会对2n取补。
      3. 如果目标类型是bool，见（），如果原始类型是bool，值false将会视为0、值true将会视为1。
   3. 浮点转换
      1. 具有浮点类型的值可以转换到另一个浮点类型的值。
      2. 如果目标类型无法容纳原始类型表示的值，结果将会对2n取补。
   4. 浮点-整数转换
      1. 具有浮点类型的值可以转换到整数类型的值，反之亦然。
      2. 如果目标类型无法容纳原始类型表示的值，结果将会对2n取补。
   5. bool转换
      1. 具有算术类型的值可以转换到bool的值。表示0的值将会转换到false，其他结果都会转换到true。
2. 表达式
   1. 条款说明
      1. [注释：条款6定义了表达式的语法、求值顺序以及含义。表达式是由操作符和操作数组成的，表示了一个计算的序列。表达式可以得到一个值作为结果，还可以造成副作用。——注释结束]
      2. 条款6定义了未被当前应用的类型重载的操作符的效果。重载的操作符不能修改*内建操作符*的规则，即应用在由标准定义的类型的操作符。这些内建操作符参与重载决议。
      3. 若在表达式的求值中，其结果没有在数学上定义或者不在这个类型的可表示范围内，则行为是未定义的。
   2. 初等表达式

初等表达式：

字面量

this

(表达式)

标识符表达式

* + 1. 字面量
       1. 字面量是一个初等表达式，它的类型取决于它的形式。
    2. This
       1. 关键字this在非静态成员函数被执行或非静态数据成员的初始化器被求值时表示一个对对象的引用。
       2. 表达式this不能在其他上下文出现。
    3. 括号
       1. 被括号包围的表达式(E)是一个初等表达式，它的类型与值都和E相同。被括号包围的表达式可以在和E可以被使用的上下文被使用，并且具有相同的意义，除非另有说明。
    4. 名称

标识符表达式：

未限定标识符

限定标识符

* + - 1. 概述
         1. 标识符表达式是初等表达式的限制表示。[注释：标识符表达式可以在.操作符后出现——注释结束]
         2. 标识符表达式用于表示一个类的非静态数据成员或者非静态成员函数时只能用在：

——作为类成员访问的一部分，当对象表达式引用成员的类[[1]](#footnote-0)或者由此派生的类，或者

——构成一个指向成员的引用，或者

——如果标识符表达式表示一个非静态数据成员并且它出现在不求值操作数中。

[示例:——示例结束]

* + - 1. 未限定名称

未限定标识符:

标识符

* + - * 1. *标识符*是一个被合适地声明的标识符表达式，表达式的类型是标识符的类型。结果是一个由标识符表示的实体。
      1. 限定名称

限定标识符：

嵌套名称说明符 未限定标识符

嵌套名称说明符：

类型名称.

模块名称.

typeof说明符.

嵌套名称说明符 标识符.

* + - * 1. 在嵌套名称说明符中由typeof说明符表示的类型应当是类或者枚举类型。
        2. 若嵌套名称说明符表示了一个类，其后是该类或该类的基类的成员的名称，则如此表示的是一个限定标识符。结果是表示的成员，结果的类型是该成员的类型。
        3. 若嵌套名称说明符表示了一个枚举，其后是这个枚举的枚举值的名称，则如此表示的是一个引用了该枚举值的限定标识符。结果是该枚举值，结果的类型是这个枚举的类型。
    1. Lambda表达式
  1. 后缀表达式
     1. 概述
        1. 后缀表达式由左向右结合。

后缀表达式：

初等表达式

后缀表达式 [ 表达式或大括号初始化列表 ]

后缀表达式 ( 表达式列表可选 )

后缀表达式 . 标识符表达式

后缀表达式 ++

后缀表达式 --

后缀表达式 as 类型名称

表达式列表：

初始化器列表

* + 1. 下标
       1. 跟随着以方括号包含的表达式的后缀表达式是下标表达式。方括号之外的表达式应当具有类型T的数组类型或者类型T的指针类型，另一个应当具有整数类型。其结果应当是类型T。对于表达式E1[E2]，若E1是数组，此表达式的行为是取得E1的第E2个元素；若E1是指针，此表达式相当于\*((E1)+(E2))。E1总是先于E2被执行。
    2. 函数调用
       1. 跟随着以圆括号包含的表达式列表的后缀表达式是函数调用表达式。圆括号之外的表达式应当具有函数类型或者函数指针类型。通过函数指针调用并非该函数指针表示的函数类型的函数将会导致未定义行为。对于隐式类成员访问的情况下，隐式对象将会是this指向的对象。[注释：此情况下的f()被解释为this.f()。——注释结束]
       2. 函数调用的结果是被执行的return语句的操作数（若存在），类型是被调用函数的返回类型。
       3. 允许递归地调用函数。
    3. 类成员访问
       1. 跟随着.和标识符表达式的后缀表达式是类成员访问表达式。.之前的表达式将会先于类成员访问表达式被执行。
       2. 类成员访问表达式的结果是根据名称查找得到的成员，若未能找到成员则结果是未定义的。
    4. 自增与自减
       1. 后缀自增和自建表达式的值是操作数自增或自减之前的值，操作数必须可被修改，自增操作使得操作数的值增加1，自减操作使得操作数的值减1。
    5. 类型转换
       1. 跟随着as和类型名称的后缀表达式是类型转换表达式。结果的类型是转换到的类型，其值由具体执行的转换决定。
  1. 一元表达式
     1. 概述
        1. 具有一元操作符的表达式由右向左结合。

一元表达式：

后缀表达式

++ 类型转换表达式

-- 类型转换表达式

一元操作符 一元表达式

new表达式

delete 表达式

一元操作符：以下其一：

+ - ! ~

* + 1. 一元操作符
       1. 一元+操作符的操作数应当具有数学类型，结果是操作数的值，结果的类型是操作数的类型。
       2. 一元-操作符的操作数应当具有数学类型，结果是操作数取负的值，对于位数为n的无符号数的负值，结果是2n减去该操作数的值，结果的类型为操作数的类型。
       3. 逻辑取反操作符!的操作数应当具有bool类型，若操作数的值为false，则结果为true，否则结果为false，结果的类型为bool。
       4. 按位取反操作符~的操作数应当具有整数类型，结果是操作数的按位取反值，结果的类型为操作数的类型。
    2. 自增与自减
       1. 前缀自增和自减表达式的值是操作数自增或自减之后的值，操作数必须可被修改，自增与自减的语义与后缀自增自减相同。[注释：++x相当于x+=1。——注释结束]
    3. new表达式
    4. delete表达式
  1. 乘除操作符
     1. 乘除操作符\*、/以及%从左向右结合。

乘除表达式：

一元表达式

乘除表达式 \* 一元表达式

乘除表达式 / 一元表达式

乘除表达式 % 一元表达式

* + 1. \*和/的操作数应当具有算术类型；%的操作数应当具有整数类型。操作数将会应用常用算术转换后进行计算，并由此决定结果的类型。
  1. 加减操作符
     1. 加减操作符+和-从左向右结合。+的操作数应当具有算术类型，或者其一具有指针类型另一具有整数类型；-的操作数应当具有算术类型，或者都具有相同的指针类型，或者左操作数具有指针类型另一具有整数类型。操作数将会应用常用算术转换后进行计算，并由此决定结果的类型。

加减表达式：

乘除表达式

加减表达式 + 乘除表达式

加减表达式 - 乘除表达式

* + 1. 若操作数都为算术类型，+操作符的结果是操作数的和，-操作符的结果是操作数的差。
    2. 指针类型操作数加减整数类型的结果如下：其类型为指针指向的类型，假设P是指针类型的操作数，x是一个数组，P指向x的第i个元素（例如P=&x[i]），x具有n个元素，J是一个具有值j的表达式，且0 ≤i+j≤n，则表达式P+J及J+P的结果是指向x[i + j]的元素的指针，否则结果是未定义的。类似地，若0 ≤i-j≤n，则P-J的结果是指向x[i - j]的元素，否则结果是未定义的。
    3. 两个指向相同的数组对象的指针相减的结果如下：其类型为[注释：需要定义PtrDiffType——注释结束]，假设P和Q是指针类型的操作数，x是一个数组，P和Q分别指向x[i]和x[j]，则表达式P-Q的结果是i - j。
    4. 对空指针值加减0的结果仍是空指针值，两个空指针值相减结果是0。
  1. 按位移动操作符
     1. 按位移动操作符从左向右结合。对于表达式E1 << E2和E1 >> E2，E1先于E2执行。

按位移动表达式：

加减表达式

按位移动表达式 << 加减表达式

按位移动表达式 >> 加减表达式

* + 1. E1 << E2的结果是E1左移E2个比特位，空出的比特将会由0填充。若E1具有无符号类型，结果将为E1×2E2，若超出可表示范围则会对2N取补（N为结果的类型的位数），若E1具有有符号类型，则结果的值将是E1作为无符号类型的表示做相同的运算的结果作为有符号类型的表示的值。
    2. E1 >> E2的结果是E1右移E2个比特位，若E1是无符号类型，则结果将是E1/2E2的整数部分，若E1是有符号类型且是负数，则结果是实现定义的。
  1. 关系操作符
     1. 关系操作符从左向右结合。

关系表达式：

按位移动表达式

关系表达式 < 按位移动表达式

关系表达式 > 按位移动表达式

关系表达式 <= 按位移动表达式

关系表达式 >= 按位移动表达式

* + 1. 对于指针的比较，若两个指针指向同一数组的不同元素，则指向具有较高下标的元素的指针更大，其他情况下结果由实现定义。
  1. 相等操作符
     1. 相等操作符从左向右结合。

相等表达式：

关系表达式

相等表达式 == 关系表达式

相等表达式 != 关系表达式

* + 1. 对于指针，当且仅当指针表示相同的对象的地址，或者都为空时才为相等，其他情况下皆为不相等。
  1. 按位与操作符

按位与表达式：

相等表达式

按位与表达式 & 相等表达式

* 1. 按位或操作符

按位或表达式：

按位与表达式

按位或表达式 | 按位与表达式

* 1. 异或操作符

异或表达式：

按位或表达式

异或表达式 ^ 按位与表达式

* 1. 逻辑与操作符

逻辑与表达式：

异或表达式

逻辑与表达式 && 异或表达式

* + 1. 逻辑与操作符的两个操作数都会转换到bool，如果两个操作数的结果都是true则结果是true，否则都是false。逻辑与操作符遵循短路求值规则，若第一个操作数结果是false则不会对第二个操作数求值。
    2. 若第二个操作数被求值则总是后于第一个操作数求值。
  1. 逻辑或操作符

逻辑或表达式：

逻辑与表达式

逻辑或表达式 || 逻辑与表达式

* + 1. 逻辑或操作符的两个操作数都会转换到bool，如果两个操作数的结果有一是true则结果是true，否则是false。逻辑或操作符遵循短路求值规则，若第一个操作数结果是true则不会对第二个操作数求值。
    2. 若第二个操作数被求值则总是后于第一个操作数求值。
  1. 条件操作符

条件表达式：

逻辑或表达式

逻辑或表达式 ? 表达式 : 赋值表达式

* + 1. 条件与操作符的第一个操作数会转换到bool并总是被求值，若结果是true则对第一个操作数求值，否则对第二个操作数求值，结果的类型是第一个操作数和第二个操作数的公共类型，值是被求值的表达式转到结果类型的值。
    2. 以左类型指代第二操作数的类型，以右类型指代第三操作数的类型，则决定公共类型的规则：

——若左类型与右类型相同，则结果为此类型，否则

——若左类型与右类型其一是浮点类型而另一是整数类型，则结果为上文所指的浮点类型，否则

——若左类型与右类型同属浮点类型或者同属整数类型，则结果为具有较高等级的类型。

* + 1. 若第二操作数及第三操作数都是可改变的，且类型相同，则结果同样可改变。
  1. 赋值及复合赋值操作符

赋值表达式：

条件表达式

逻辑或表达式 赋值操作符 初始化器

赋值操作符：以下其一

= \*= /= %= += -= >>= <<= &= ^= |=

* + 1. 赋值及复合赋值操作符从右向左结合，所有赋值及复合赋值操作符都要求左操作数是可改变的。右操作数先于左操作数求值。结果是左操作数，并且可改变。
    2. 对于复合赋值操作符，E1 op= E2相当于E1 = E1 op E2，但E1只会被求值一次。

1. 语句
   1. 条款说明

语句：

表达式语句

复合语句

if 语句

迭代语句

跳转语句

声明语句

* + 1. 除非另有说明，语句按序列执行。
  1. 表达式语句

表达式语句：

表达式可选;

* + 1. 表达式语句中的表达式是弃值表达式。所有表达式语句的副作用都会在下一个语句执行之前完成。无表达式的表达式语句被称为空语句。
  1. 复合语句

复合语句：

{ 语句序列可选 }

语句序列：

语句

语句序列 语句

* + 1. 多个语句通过包含在{}中可以作为单个语句出现在需求单个语句的地方。
    2. 复合语句定义了块范围(5.3.3)。
  1. if语句

if 语句：

if ( 表达式 ) 语句

if ( 表达式 ) 语句 else 语句

* + 1. 若条件求值结果是true，则执行第一个子语句，如果存在else部分并且条件求值结果是false则执行第二个子语句。else部分与最近的未结合else的if结合。
    2. if语句的子语句隐式定义了块范围。[示例：

if (b)

def a : int;

与

if (b)

{

def a : int;

}

等价，在if语句后，a不再存在于范围中。——示例结束]

* 1. 迭代语句
     1. 条款说明

迭代语句：

while ( 表达式 ) 语句

do 语句 while ( 表达式 ) ;

for (初始化语句 表达式可选 ; 表达式可选 ) 语句

* + - 1. 迭代语句表示循环行为。
      2. 迭代语句的子语句隐式定义了块范围。[示例：

while (--i)

def a : int;

与

while (--i)

{

def a : int;

}

等价，在while语句之后，a不再存在于范围中。——示例结束]

* + 1. while 语句
       1. while语句中的子语句将会在条件表达式的值为false之前一直重复执行，每次执行子语句之前都会测试一次条件表达式。
    2. do 语句
       1. do语句的子语句将会在条件表达式的值为false之前一直重复执行，每次执行子语句之后都会测试一次条件表达式。
    3. for 语句
       1. for语句的语法如下：

for (初始化语句 条件表达式可选 ; 自增表达式可选 ) 语句

语义上等价于

{

初始化语句

while (条件表达式)

{

语句

自增表达式;

}

}

除了continue语句将会在重新对条件表达式求值之前执行自增表达式以外。

* 1. 跳转语句
     1. break语句
        1. break语句仅能在迭代语句的子语句中出现，语义是立即终止迭代，从迭代体中跳出。
     2. continue语句
        1. continue语句仅能在迭代语句的子语句中出现，语义是立即跳转到迭代体的结尾。
     3. return语句
        1. return语句的语义是将控制流从函数体返回到调用者。操作数将会作为返回值。
  2. 声明语句
     1. 声明语句引入一个或多个新名称到块中，声明语句具有形式：

声明语句：

变量声明

* + 1. 具有自动存储期的变量将会在每一次声明语句执行时被初始化，并在离开当前块时被销毁。

1. 声明
   1. 条款说明
      1. 声明通常说明了解读名称的方式。声明有以下形式：

声明序列：

声明

声明序列 声明

声明：

变量声明

函数声明

模块声明

类声明

空声明

* 1. 变量声明

变量声明：

def 声明符 ;

声明符：

标识符 类型部分可选 初始化器可选

类型部分：

: 类型名称

初始化器：

= 表达式

= { 初始化器列表 }

* + 1. 变量声明向当前声明域引入了一个自动存储期的变量，类型部分和初始化器不可同时忽略，若初始化器具有第二种形式则类型部分不可忽略。当类型部分忽略且初始化器具有第一种形式时，变量的类型将会由表达式推导。
  1. 函数声明

函数声明：

def 函数声明符

函数声明符：

标识符 函数类型部分 函数体

函数类型部分：

: 函数类型

函数体：

复合语句

* + 1. 函数声明向当前声明域引入了一个函数。
  1. 模块声明

模块定义：

module 标识符 { 模块体 }

模块体：

声明序列可选

* + 1. 模块定义仅能出现在模块范围内。
    2. 若定义一个具有名称name的模块，且在当前上下文中查找name能查找到其他具有相同名称的模块，则此模块定义将会导入到查找到的模块，即具有相同名称的模块将会视为同一个模块，使用此名称将可以引用所有相同名称模块中的名称。
  1. 类声明
     1. 见10。
  2. 空声明

空声明：

;

1. 类
   1. 条款说明
      1. 类是一种类型。每个类定义都会引入一个新的类型。

类定义：

类头部 { 成员声明可选 }

类头部：

class 标识符

* 1. 类成员
     1. 类定义中的成员声明声明了类的所有成员，无法通过编译器动作以外的其他方式增加新的成员。
     2. 数据成员指由成员声明引入的非函数成员。成员函数指由成员声明引入的函数成员。
     3. 数据成员和成员函数可以声明为static的，声明为static时称为静态成员。成员不可声明为extern的。

1. 重载
2. 异常处理
3. 编译器动作

编译器动作：

$ 编译器动作命名空间说明符可选 编译器动作标识符 ( 参数列表可选 )

编译器动作命名空间说明符：

命名空间标识符 .

编译器动作命名空间说明符 命名空间标识符 .

参数列表：

参数

参数列表 , 参数

参数：

类型

语句

表达式

声明

标识符

* 1. 概述
     1. 编译器动作可以请求零个或多个参数，具体请求的参数个数由具体的编译器动作决定，参数的个数及形式可能与动作、其他参数或上下文有关。
     2. 为编译器动作传递参数时不会执行作为参数的语句或对表达式求值，但在动作执行时可能会执行语句或对表达式求值，并可能由此产生一系列副作用，具体的行为由具体的动作决定。
     3. 编译器动作可以在要求类型、语句、表达式或声明的上下文中出现，并可能输出零个或多个类型、语句、表达式或声明，输出的结果将会视为位于动作的位置上。
     4. 多数标准预定义的编译器动作位于Compiler命名空间下，实现可能提供其他扩展的编译器动作，由实现提供的扩展编译器动作的行为是实现相关的。
  2. 标准预定义的编译器动作
     1. Dump
        1. Dump将输入的参数原样输出，不改变任何语义，没有任何副作用。

1. 包括隐式的this。 [↑](#footnote-ref-0)