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. 病式的程序

并非合式的程序（2.13）。

* 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. 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. 分离编译
   2. 编译阶段
   3. 字符集
   4. 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. 表格1中的标识符被保留用于作为关键字。

表格1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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表格2中相应的列表中第一个可以表示它的值的类型。

表格2

|  |  |  |
| --- | --- | --- |
| 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。

表格3 转义序列

|  |  |  |
| --- | --- | --- |
| 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
    2. 字符串字面量
    3. 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。这样的字面量具有类型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. 函数原型范围
    3. 函数范围
    4. 模块范围
    5. 类范围
    6. 枚举范围
    7. 名称隐藏
  1. 名称查找
     1. 非限定名称查找
     2. 限定名称查找
        1. 类成员
        2. 模块成员
        3. 类成员访问
  2. 程序和链接
  3. 启动和结束
     1. 入口函数
     2. 结束
     3. 存储期
     4. 子对象的存储期
     5. 对象生存期
  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. 组合类型可以被下列方式构造：

——*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. 对齐

1. 标准转换
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-1)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 )

简单类型说明符 ( 表达式列表可选 )

类型名说明符 ( 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 --

表达式 as 类型标识符

表达式列表：

初始化器列表

* + 1. 下标
    2. 函数调用
    3. 类成员访问
    4. 自增与自减
    5. 类型转换
  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 转换表达式

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. 语句
2. 声明
3. 声明符
4. 类
5. 重载
6. 异常处理
7. 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

* 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-1)