Julia语言中双曲复数的实现

Anna V. Korolkova，[1]，*Migran N. Gevorkyan，[1]，†和 Dmitry S. Kulyabov[1,2,‡]俄罗斯人民友谊大学（*RUDN University*），俄罗斯联邦莫斯科市米克卢霍-马克莱亚街*6*号，邮编*117198*[2]联合核子研究所，俄罗斯联邦莫斯科州杜布纳市约里奥-居里街*6*号，邮编*141980*

背景： 双曲复数用于描述双曲空间。这类空间中一个著名的例子是闵可夫斯基空间，它在狭义相对论和电动力学问题中发挥着重要作用。然而，这类数在各种编程语言中并不常见。**目的：** 本文关注双曲复数在科学编程语言中的实现，尤其是在Julia语言中的实现。**方法：** Julia语言基于多重分派这一概念。该概念是面向对象编程语言中多态概念的扩展。为实现双曲复数，本文采用了Julia语言的多重分派方法。**结果：** 研究实现了一个用于处理双曲数的库。**结论：** 根据研究结果可以得出结论：科学编程语言中的多重分派概念便捷且自然。

Key词语：Julia 编程语言、多重分派、抽象数据类型、类型转换、参数化结 ric
stru构、双曲复数

arXiv:2301.01707v1 [cs.MS] 4 Jan 2023

I. 引言

Julia 编程语言 [1, 2] 是科学计算领域一种前景广阔的语言。目前，Julia语言已发展至稳定阶段。Julia 的设计旨在解决双语言问题。这一问题在于：快速原型开发、数据处理和可视化通常使用解释型动态语言或数学软件包（Python、Matlab 等），而进行密集数值计算时，则必须将程序改写为静态类型的编译型语言（C/C++、Fortran）。

Python 是这一问题的一个例证。它作为语言胶水接口而广受欢迎。人们为它编写了大量包装库，通过 Python 代码调用预编译库中的 C/C++ 和 Fortran 函数。例如，著名的 NumPy 库 [3]由 51% 的 C 代码和仅 47% 的 Python 代码组成（其余部分由 C++、Fortran、JavaScript 和 Unix shell 构成）。

Julia语言结合了动态类型解释型语言的灵活性与静态类型编译型语言的性能。

Julia语言的基础部分与其他科学编程语言非常相似，因此掌握起来并不困难。然而，Julia的核心围绕多重分派 [4], 这一概念构建，而这一概念在其他语言中并不常见。Julia与其他语言的本质区别就在于这一机制；要充分利用Julia的所有优势，就必须理解这一机制。

A. 论文结构

在本文中，作者特别注重阐释多重分派机制以及与之密切相关的其他机制。

在文章的第一部分，我们给出必要的定义，并通过简单示例阐释多重分派的概念，帮助你理解语言这一部分所涉及的语法，并掌握这种方法的核心思想。在第二部分，我们以在 Julia语言中实现双曲复数为例。这个例子不仅让你了解多重分派，还能接触类型转换机制、抽象类型层次结构、算术运算符重载，以及用户定义数据类型的指定。

*korolkova-av@rudn.ru †
gevorkyan-mn@rudn.ru ‡
kulyabov-ds@rudn.ru

II. 多重分派

A. 常见定义

动态分派 是一种机制，可让你在特定情况下从多种多态函数（或运算符）实现中选择应调用的实现 [5]。这种选择是在程序执行阶段进行的。多重分派 基于动态分派。在这种情况下，多态函数实现的选择依据是函数参数的类型、数量和顺序。这就是运行时多态分派的实现方式 [6, 7]。还要注意，除了“多重分派”这一术语外，也会使用多方法 这一术语。

多重分派机制与函数和运算符重载机制类似，例如 C++ 语言中实现的机制。不过，函数重载仅在编译时进行，而多重分派还应能在运行时工作（运行时多态）。

B. Julia 中的多重分派

为了说明多重分派机制，我们将在 Julia语言中给出以下代码示例。

```
函数 f(x,y)println("通用实现")返回
x  +y结束


函数 f(x)println("单个参数时")返
回 x结束


函数f(x::整数, y::整数)println("整数的实
现")返回 x  +y结束


函数f(x::字符串, y::字符串)println("字
符串的实现")返回 x *" " * y结束


函数 f(x::元组{整数, 整数}, y::元组{整数,整数})println("包含两个
整数元素的元组的实现")返回 (x[1],x[2], y[1], y[2])结束
```

在这个示例中，我们为 f 函数创建了五种实现，它们的签名各不相同。就 Julia语言而言，这意味着一个函数 f 现在有四种不同的方法。前两种方法中，我们没有使用类型注解，因此参数的类型将在编译时或运行时（如同解释型语言中一样）确定。还值得注意的是，Julia 使用动态即时编译（即时编译），因此对于用户而言，编译阶段与执行阶段并没有明确分开。

以下三个方法的参数都标注了类型，因此只有类型与标注匹配时才会调用它们。针对字符串的 f 函数使用了 * 拼接运算符。语言的创建者解释说，之所以选择乘号 * 而不是更传统的加号 + ，是因为字符串拼接不是交换运算，因此用乘号表示更合乎逻辑；而加号通常用于表示交换运算。

下面的代码片段展示了多重分派在编译时的工作方式。`@show` 宏用于打印函数名称以及传递给它的参数。

```
@show f(2.0, 1)
@show f(2, 2)
```

```
@show f(0x2,0x1) # 十六进制数@show f("文本","行")
@show f(3)@showf([1, 2], [3, 4])@showf((1, 2),
(3, 4))
```

- 第一行中，我们向函数传入了实数（浮点数）类型的参数，因此调用了通用实现。由于运算符 + 是为浮点数定义的，函数成功运行并得出了正确结果。

- 第二行和第三行调用了整数的方法。请注意，整数类型是抽象类型，包含语言核心中定义的、大小为 1 到 16 字节的有符号和无符号整数。默认情况下，十六进制数会被解释为无符号整数。

- 第四行调用了字符串的方法。第五行调用了单参数的方法。

- 第六行将两个数组作为参数传入。 + 运算是为数组定义的，因此函数运行时没有出错，并返回了逐元素求和的结果。

- 第七行中，函数参数是由两个整数构成的元组。由于我们为这种参数组合定义了方法，函数运行正常。

通用实现 f(2.0, 1) = 3.0 整数的实现 f(2, 2) = 4整数的实现 f(0x02, 0x01) = 0x03 字符串的实现 f("Text", "line") = "Text line" 单参数 f(3) = 3通用实现 f([1, 2], [3, 4]) = [4, 6]由两个整数元素组成的元组的实现 f((1, 2), (3,4)) = (1, 2, 3, 4)

上述示例在支持函数重载的语言中可以正常运行，但并未展示动态分派的具体机制，因为参数类型在编译阶段已知，并且翻译器可以获取这些类型。

为测试动态方法调用的效果，请看以下代码：

```
print("请输入一个整数: ")# 读取字符串并转换为
整数类型@show n  =parse(Int32, readline())ifn
 > 0x  = 1.2; y = 0.1elsex  = 1; y = 2endf(x,
y)
```

这里，变量值 x 和 y 的类型在编译时尚不确定，因为它们取决于用户在程序执行期间输入的数值。不过，当 x 和 y 为整数时，会调用相应的方法。

III. 双曲数

我们将使用双曲数来展示 Julia语言的多重分派功能，因此仅介绍其定义和基本算术运算。

双曲数 [8–11], 与椭圆数和抛物数一样，都是复数的推广。双曲数定义如下：

$$z = x + \mathrm{j}y,\ \mathrm{j}^2 = 1,\ \mathrm{j} \neq \pm 1.$$

数量 j 称为双曲虚数单位，数量 x 和 y 分别称为实部和虚部。

对于两个双曲数 $z_1 = x_1 + \mathrm{j}y_1$ 和 $z_2 = x_2 + \mathrm{j}y_2$，进行以下算术运算。

加法： $z_1 + z_2 = (x_1 + x_2) + \mathrm{j}(y_1 + y_2)$。

乘法： $z_1 z_2 = (x_1 x_2 + y_1 y_2) + \mathrm{j}(x_1 y_2 + x_2 y_1)$。

共轭： $z^* = x - \mathrm{j}y$。

逆数： $z^{-1} = \dfrac{x}{x^2 + y^2} - \mathrm{j}\dfrac{y}{x^2 - y^2}$。

除法： $\dfrac{z_1}{z_2} = \dfrac{x_1 x_2 - y_1 y_2}{x_2^2 - y_2^2} + \mathrm{j}\dfrac{x_1 y_1 - x_1 y_2}{x_2^2 - y_2^2}$.

双曲数的实现与复数的实现有许多相似之处。必须重载运算符 +、-、*，以及 /、开方、求幂、初等数学函数等。与此同时，为了说明多重分派的运作机制，我们主要关注算术运算。这是因为初等函数只接受一个参数，只需为其定义一种方法即可。对于算术运算符，则需要支持不同数值类型参数的各种组合。例如，应当能够将双曲数与整数、有理数、无理数相加，这不仅会影响多重分派，还会影响类型转换机制、抽象类型层次结构以及默认构造函数重载。

因此，我们将仅限于介绍算术运算本身的实现示例，不涉及双曲数各种初等函数中数学性更强的计算。

请注意，除了“双曲数”这一术语外，文献中还使用以下术语：双数、分裂复数、佩尔普莱克斯数、双曲数 [8, 12–15]。

IV. 在 JULIA 中实现双曲数

A. 声明数据结构

在 Julia 中实现双曲数的工作基于 Julia 官方代码库中复数的代码。我们还借鉴了抛物复数实现方面的成果 [16]。新的类型 Hyperbolic 使用不可变结构定义：

```
结构体双曲数{T<:实数}  <: 数“实部”
re::T“虚部”jm::T结束
```

该结构非常简单，仅包含两个参数化类型 T 的字段。这要求类型 T 是抽象类型 Real 的子类型（语法为 T<:Real）。类型 Hyperbolic 是抽象类型 Number的子类型（见图 1）。因此，双曲数被纳入了现有的数值类型层次结构中。

定义结构后，可以通过调用默认构造函数来创建一个新的 Hyperbolic 类型对象。例如，数 $h = 1 + \mathrm{j}3$ 表示如下：

```
h  =双曲数{Float64(1, 3)
```

创建后，可以访问已存在对象的字段值 结构的字段，例如 h。re 和 h。jm，但尝试更改 ject 将导致错误，因为结构体是不可变实体。

```
h  =双曲数(1,3)
```

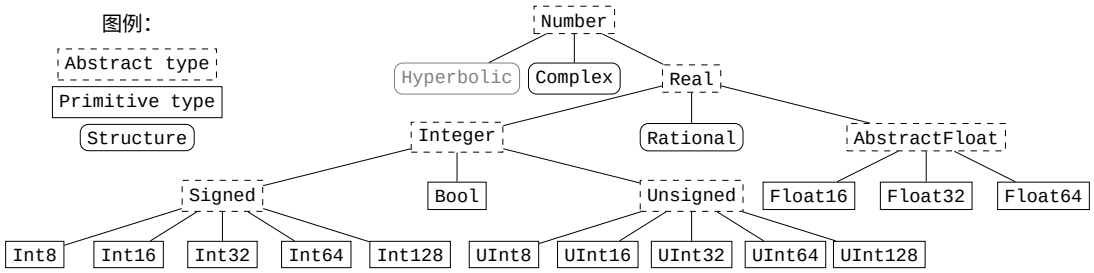

图 1。双曲数在 Julia 类型层次结构中的位置

不过，如果参数类型不同，默认构造函数就无法隐式转换并创建新对象。在这种情况下，必须显式指定参数化类型

```
    # Float64 和 Int64h  =Hyperbolic(1.0, 3)
# 错误h  = Hyperbolic{Float64}(1.0, 3)# 正确
```

B. 附加构造函数

默认构造函数是一个普通函数，其名称与类型名称相同。通过为此函数创建附加方法，您可以创建用于处理各种特殊情况的附加构造函数。

因此，例如，为了不必每次都指定参数化类型，您应该添加如下形式的新构造函数：

```
"""构造函数 No2"""
函数双曲数(x::实数, y::实数)返回双曲数
(promote(x, y)...)结束
```

该 提升 函数会将传递给它的参数转换为公共类型，并以元组形式返回结果。后缀运算符 ... 会解包该元组，并将其中的元素作为参数传递给构造函数。语言核心为 实数 抽象类型的所有子类型定义了类型转换规则，因此只要满足 T<: 实数规则，构造函数现在就能正确处理任意参数组合。例如，以下代码将正常运行：

```
    # 有理数和 Float64h  =双曲数(1//3, pi)>>双曲
数{Float64}(0.5, 3.141592653589793)
```

我们将一个有理数（类型为 有理数）和一个类型为 π的内置全局常量（数值 Float64 ）传递给构造函数。之后，类型转换规则生效，两个参数都被转换为更通用的 Float64类型。

声明另外两个构造函数 后，您就可以指定虚部为零的双曲数 :

```
"""构造函数 No3"""函数双曲数{T}(x::实数) where
{T<:实数}返回双曲数{T}(x, 0)结束"""构造函数 No4""
"函数 双曲数(x::实数)返回双曲数(提升(x, 0)...)结束
```

构造函数 No3 是使用 where 构造体声明的参数化函数。 T 是抽象类型 实数 的子类型。构造函数 No4 的工作方式与构造函数 No2 类似。

另外两个构造 构造函数将允许你将其他双曲数作为参数传递给构造函数 r.

```
"""构造函数 No5"""函数 双曲数{T}(h::双曲数) where{T<:实
数}双曲数{T}(h.re, h.jm)end

"""构造函数 No6"""
函数 双曲数(h::双曲数)返回双曲数(提升(h.re,
h.jm)...)结束
```

为了更加方便，你也可以为虚数单位 \mathrm{j} 单独定义一个常量：

```
常量 jm  =双曲数(0, 1)
```

C. 数据打印

为了能够以紧凑且易读的形式打印双曲数类型的值，你应该将相应的方法添加到 Base 模块中的 show 函数。

```
函数Base.show(io::IO, h::双曲数)
print(io, h.re,"+", h.jm, "j")end
```

函数 show 用于将数据打印到控制台，尤其是由 println和宏 @show调用。下面的代码和输出列表将假定已为双曲数添加了 show 方法。

D. 类型转换

在继续实现双曲数的算术运算方法之前，有必要定义类型转换规则。为此，请从 Base 模块中为函数 promote_rule 创建一个新方法。

函数 Base.promote_rule(::Type{双曲数{T}}, ::Type{S}) where {T<:实数, S<:实数} 返回双曲数{promote_type(T, S)} 结束 函数 Base.promote_rule(::Type{双曲数{T}}, ::Type{双曲数{S}}) where {T<:实数, S<:实数} ↪返回双曲数{promote_type(T, S)} 结束

作为提升_规则中指定了参数化类型的参数，这些类型应转换为一个包容类型。在我们的例子中，如果其中一种类型是实数的子类型，那么包容类型就是双曲数。为提升_规则添加方法后，就可以使用提升、提升_类型和转换函数。

```
>>h  =双曲数(1 //2)>>提升(h, 1)(1//2+0//1j,
1//1+0//1j)>>提升_类型(双曲数{Int64},
Float32)双曲数{Float32}
```

第一个函数我们已经很熟悉了。第二个函数可以推断出包容类型，但推断对象不是特定变量的值，而是类型本身。在 Julia 中，类型是一种第一类对象（类型为DataType），可以赋值给其他变量、作为函数参数传递，等等。

函数 convert 可以转换特定值的类型，例如：

```
>>转换(双曲数, 1)1+0j
```

添加类型转换的方法后，就可以开始添加算术运算的方法了。Julia 的一个特点是，算术运算不是以运算符的形式实现，而是以函数的形式实现。例如，以下调用都是正确的：

```
>>+(1,2)3>>+(1,2,3,4)10>>+((i for i in
1:10)...) # 从 1 到 10 的数字55
```

在这方面，为算术运算添加方法与为其他函数执行相应操作并无不同。

一元运算 + 和 - 的方法添加如下：

```
Base.:+(h::双曲数) =双曲数(+h.re, +
h.jm)Base.:-(h::双曲数) =双曲数(-h.re, -h.jm)
```

这是一个简写的函数声明。

同样，也会为二元加法、减法、乘法和除法添加方法。下面是加法和乘法的代码。

```
# 二元 +和 * 函数 Base.:+(x::双曲数, y::双曲数) xx =
x.re + y.re yy = x.jm + y.jm 双曲数(xx, yy) end
函数 Base.:*(x::双曲数, y::双曲数) xx = x.re* y.re +
x.jm * y.jm yy = x.re* y.jm + x.je * y.re return
双曲数(xx, yy) end
```

V. 结论

我们以双曲数的实现为例，考察了 Julia语言中多重分派的底层机制。这个例子让我们得以探讨该语言中的一些概念，例如数据类型层次结构、复合数据类型、类型转换机制、函数重载（按照 Julia语言的术语，为函数创建新的方法）等。

致谢

本文 得到了俄罗斯人民友谊大学战略学术领导力计划的支持 .

[1] J. Bezanson, A. Edelman, S. Karpinski, V. B. Shah, 《Julia：一种数值计算的新方法》，SIAM Review 59 (1) (2017年) 65–98。doi:10.1137/141000671。[2] M. N. 格沃尔基扬, D. S. 库利亚博夫, L. A. Sevastyanov, 《面向科学计算的 Julia 编程语言综述》，载于：第六届国际会议“科学与教育中的分布式计算与网格技术”，2014年，第27页。[3] T. E. Oliphant, 《NumPy 指南》，第2版，CreateSpace独立出版平台，2015年。[4] F. Zappa Nardelli, J. Belyakova, A. Pelenitsyn, B. Chung, J. Bezanson, J. Vitek, 《Julia 子类型：一种有理数式重构》，ACM 编程语言会议录 2 (OOPSLA) (2018年) 1–27。doi:10.1145/3276483。[5] K. Driesen, U. H¨olzle, J. Vitek, 《流水线处理器上的消息分派》，《计算机科学讲义》，S pringer Berlin Heidelberg，1995年。doi:10.1007/3-540-49538-x_13。[6] R. Muschevici, A. Potanin, E. Tempero, J. Noble, 《实践中的多重分派》，载于：OOPSLA’08：第23届ACM SIGPLAN会议：面向对象编程系统、语言与应用，ACM Press，2008年，第563–582页。doi:10.1145/1449764.1449808。[7] S. Gowda, Y. Ma, A. Cheli, M. Gw´o´zzd´z, V. B. Shah, A. Edelman, C. Rackauckas, 《通过多重分派实现高性能符号数值计算》，ACM 计算机代数通讯 55 (3) (2022年) 92–96。doi:10.1145/3511528.3511535。[8] I. M. 雅格洛姆, 《几何中的复数》，Academic Press，1968年。

[9] I. M. 雅格洛姆、B. A. 罗岑费尔德、E. U. 雅辛斯卡娅，《射影度量》，《俄罗斯数学综述》19 (5) (1964年) 49–107。doi:10.1070/RM1964v019n05ABEH001159。[10] D. S. 库利亚博夫、A. V. 科罗尔科娃、L. A. 谢瓦斯季亚诺夫，《用于相对论运算的复数》(2021年12月)。doi: 10.20944/preprints202112.0094.v1。[11] D. S. 库利亚博夫、A. V. 科罗尔科娃、M. N. 格沃尔基扬，双曲数作为爱因斯坦数，《物理学杂志：会议系列》1557 (2020年) 012027.1–5。doi:10.1088/1742-6596/1557/1/012027。[12] P. 菲尔斯塔德，通过佩尔普莱克斯数扩展狭义相对论，《美国物理学杂志》54 (5) (1986年) 416–422。doi:10.1119/1.14605。[13] W. 班德，关于通过佩尔普莱克斯数扩展相对论的评论，《美国物理学杂志》56 (5) (1988年) 469–469。doi:10.1119/1.15582。[14] J. 鲁尼，关于三类复数和平面变换，《环境与规划B：规划与设计》5 (1) (1978年) 89–99。doi:10.1068/b050089。[15] J. 鲁尼，力学中的广义复数，载于：M. Ceccarelli、V. A. Glazunov（编），《机器人与操纵器理论及实践进展》，《机构与机器科学》第22卷，Springer国际出版公司，楚格，2014年，页55–62。doi:10.1007/978-3-319-07058-2_7。[16] M. N. 格沃尔基扬、A. V. 科罗尔科娃、D. S. 库利亚博夫，Julia语言中广义复数的实现方法，载于：D. S. 库利亚博夫、K. E. 萨穆伊洛夫、L. A. 谢瓦斯季亚诺夫（编），第十届信息与电信技术及高科技系统数学建模国际会议（ITTMM-2020）框架下的信息技术与科学计算研讨会，《CEUR研讨会论文集》第2639卷，亚琛，2020年，页141–157。网址：http://ceur-ws.org/Vol-2639/paper-13.pdf

Julia语言中双曲复数的实现

A. B. Королькова,[1], * M. H. Геворкян,[1], † 和 Д. С. Кулябов[1,2,‡]
[1]俄罗斯人民友谊大学，*117198*，*6* [2]莫斯科，米克卢霍-马克莱街，楼
Объединённый институт ядерных исследований，约里奥-居里街*6*号，杜布纳，莫斯科州，俄罗斯，*141980*

背景。 双曲复数用于描述双曲空间。此类空间的著名实例之一是闵可夫斯基空间，它在狭义相对论和电动力学问题中发挥着重要作用。然而，这类数在各种编程语言中并不常见。**目的**。 在科学编程语言中实现双曲复数颇有意义，尤其是在 Julia 语言中。**方法**。 Julia语言以多重分派（multiple dispatch）这一概念为基础。该概念是面向对象编程语言中多态概念的扩展。实现双曲复数时，采用了 Julia 语言的多重分派方法。**结果**。 最终获得了一个用于实现双曲数的库。**结论**。 根据研究结果，可以得出结论：多重分派这一概念在科学编程语言中既方便又自然。

关键词：Julia 编程语言，多重分派，抽象数据类型，类型转换，参数化结构体，双曲复数

arXiv:2301.01707v1 [cs.MS] 4 Jan 2023

I. 引言

Julia 编程语言 [1, 2] ——是一种前景广阔的语言，旨在用于科学计算。目前，Julia 语言已达到稳定状态。根据开发者的设计，Julia 解决了两种语言问题。这个问题在于，为了快速原型开发、数据处理和可视化，人们会使用解释型动态语言或数学软件包（Python、Matlab 等）；而进行密集数值计算时，则不得不将程序改写为具有静态类型的编译型语言（C/C++、Fortran）。

以 Python 语言为例，可以看到这一问题的具体表现。Python 因作为接口“胶水语言”而广受欢迎。人们用它编写了大量包装库，利用 Python 代码调用预先编译的库中的 C/C++ 和 Fortran 函数。例如，著名的 NumPy [3] 有 51% 的代码由 C 语言编写，只有 47% 由 Python 编写（其余部分由 C++、Fortran、JavaScript 和 Unix shell 共同构成）。

Julia 语言兼具解释型动态类型语言的灵活性，以及静态类型编译型语言的性能。

Julia 语言的基础部分与其他科学编程语言非常相似，因此学习起来并不困难。然而，Julia 的核心围绕多重分派 (multiple dispatch) [4], 这一概念构建，而这一概念在其他语言中很少见。正是这一机制使 Julia 与其他语言有着显著区别；理解它对于充分利用 Julia 的所有优势至关重要。

A. 文章结构

在本文中，作者着重阐释了多重分派机制及与之密切相关的其他机制。

在文章的第一部分，我们给出必要的定义，并通过简单示例阐释多重分派的概念，帮助读者理解与语言这部分相关的语法，并

*korolkova-av@rudn.ru †
gevorkyan-mn@rudn.ru ‡
kulyabov-ds@rudn.ru

领会这一方法的精髓。在第二部分，我们以 Julia 语言实现双曲复数为例。这个示例不仅涉及多重分派，还涉及类型转换机制、抽象类型层次结构、算术运算符重载以及用户定义数据类型的定义。

II. 多重分派

A. 一般定义

动态分派 （dynamic dispatch）是一种机制，用于从多种多态函数（或运算符）的实现中，选择在特定情况下应调用哪一种 [5]。选择具体实现是在程序运行阶段进行的。多重分派 建立在动态分派的基础上。在这种情况下，多态函数实现的选择取决于函数参数的类型、数量和顺序。由此实现运行时多态（runtime polymorphic dispatch） [6, 7]。还需指出，除了“多重分派”这一术语外，也使用术语多方法。

多重分派机制类似于函数和运算符重载机制，例如 C 语言中的实现++。然而，函数重载只在编译阶段进行，而多重分派在程序运行阶段也必须能够工作（运行时多态）。

B. Julia 中的多重分派

为了说明多重分派机制，下面给出一段 Julia 语言代码示例。

```
function f(x,y)println("通用
实现")return x  +yend


function f(x)println("单参数的实
现")return xend


functionf(x::Integer,
y::Integer)println("整数实现")return x
 +yend

functionf(x::String,
y::String)println("字符串实现")
return x *" " * yend

函数 f(x::Tuple{Int, Int}, y::Tuple{Int,Int})println("适用于由两个整
数元素组成的元组的实现")返回 (x[1],x[2], y[1], y[2])end
```

在这个示例中，我们为函数 f 创建了五种实现，它们的签名各不相同。用 Julia 语言的术语来说，这意味着函数 f 现在有四种不同的方法。前两种方法中，我们没有使用类型注释，因此参数的类型将在编译阶段或程序运行阶段确定（与解释型语言一样）。还值得注意的是，Julia 使用动态 JIT 编译（即时编译），因此对于用户而言，编译阶段与运行阶段并没有明确区分。

接下来三种方法的参数都带有类型注释，因此只有参数类型与注释相符时才会调用这些方法。在字符串的 f 方法中，使用乘号 * 作为连接运算符。语言创建者选择乘号 * 而非更传统的加号 + ，理由是字符串连接属于不可交换运算，因此使用乘号而不是通常用于表示可交换运算的加号更为合理。

下面编译阶段的演示方法重载分派

宏 @show 用于打印函数名及其传入的参数
@show f(2.0,1) @show f(2, 2) @show f(0x2,0x1) # 十六进制
数 @show f("字符串","文本") @show f(3) @show f(
[1, 2], [3, 4]) @show f((1, 2), (3, 4))

B.

- 第一行中，我们向函数传入了浮点类型的参数，因此调用了通用实现。由于浮点数定义了运算符 +，函数得以成功执行并返回了正确结果。

- 第二行和第三行调用了整数的方法。请注意，类型Integer 是抽象类型，包含语言核心中定义的、大小为 1 至 16 字节的有符号和无符号整数。以十六进制记数法表示的数默认会被解释为无符号整数。

- 第四行调用了字符串方法。第五行调用了单参数方法。
- 第六行将两个数组作为参数传入。操作 + 针对数组定义，因此函数执行成功并返回了逐元素求和。

- 第七行中，函数参数是由两个整数构成的元组。由于我们为这种参数组合定义了方法，因此函数正确执行。

f(的通用实现2.0, 1) = 3.0整数的实现 f(2, 2)
= 4整数的实现 f(0x02, 0x01) = 0x03字符串的实现
f(" 字符串 "," 文本 ") = " 字符串文本 " 单参数 f(
3) = 3f(的通用实现[1, 2], [3, 4]) = [4, 6]两
个整数构成的元组的实现

元素

f((1, 2), (3, 4)) = (1, 2, 3, 4)

这些示例在支持函数重载的语言中也能正确运行，并不体现动态分派的特性，因为参数类型在编译阶段已知，且编译器可以访问这些类型。

为了验证方法的动态调用确实有效，我们来看下面的代码:

print("请输入一个整数: ")# 读取字符串并转换为
整数类型@show n =parse(Int32, readline())如果
n > 0x = 1.2; y = 0.1否则x = 1; y = 2结束
f(x, y)

这里，变量 x 和 y 的值类型在编译阶段尚不确定，因为它们取决于用户在程序运行时输入的数字。不过，对于 x 和 y 均为整数的情况，会调用相应的方法。

III. 双曲复数

我们将使用双曲数来展示 Julia 语言多重分派的功能，因此仅介绍其定义和基本算术运算。

双曲数 [8–11],与椭圆数和抛物数一样，是复数的推广。双曲数可以定义如下：

$$z = x + \mathrm{j}y,\ \mathrm{j}^2 = 1,\ \mathrm{j} \neq \pm 1.$$

将 j 称为双曲虚单位，而 x 和 y 分别称为实部和虚部。

对于两个双曲数 $z_1 = x_1 + \mathrm{j}y_1$ 和 $z_2 = x_2 + \mathrm{j}y_2$，有以下算术运算。

加法： $z_1 + z_2 = (x_1 + x_2) + \mathrm{j}(y_1 + y_2)$。

乘法： $z_1 z_2 = (x_1 x_2 + y_1 y_2) + \mathrm{j}(x_1 y_2 + x_2 y_1)$。

共轭： $z^* = x - \mathrm{j}y$。

倒数： $z^{-1} = \dfrac{x}{x^2 + y^2} - \mathrm{j}\dfrac{y}{x^2 - y^2}$。

除法： $\dfrac{z_1}{z_2} = \dfrac{x_1 x_2 - y_1 y_2}{x_2^2 - y_2^2} + \mathrm{j}\dfrac{x_1 y_1 - x_1 y_2}{x_2^2 - y_2^2}$。

双曲数的实现与复数的实现大体相似。需要重载运算符 +、-、* 和 /、开方函数、幂运算、初等数学函数等。为了说明多重分派的工作机制，算术运算尤其值得关注。这是因为初等函数只接受一个参数，因此只需为它们定义一个方法。而对于算术运算符，则需要考虑不同数值类型参数的组合。例如，应当能够将双曲数与整数、有理数、无理数相加，这不仅涉及多重分派，还涉及类型转换机制、抽象类型层次结构以及默认构造函数的重载。

因此，我们仅举例说明算术运算的实现，不涉及对双曲数的各种初等函数进行数学上更复杂的计算。

需要指出的是，除了“双曲数”这一术语，文献中还使用“双数”“分裂复数”“双曲型复数”等术语（double numbers、split complex numbers、perplex numbers、hyperbolic numbers） [8, 12–15]。

IV. 在 JULIA 中实现双曲数

A. 数据结构声明

在 Julia 中实现双曲数时，我们以 Julia 官方代码库中的复数代码为基础。此外，我们还借鉴了实现抛物型复数时取得的成果 [16]。新的 Hyperbolic 类型通过不可变结构定义：

```
结构体双曲数{T<:实数}  <:  数“实部”
re::T“虚部”jm::T结束
```

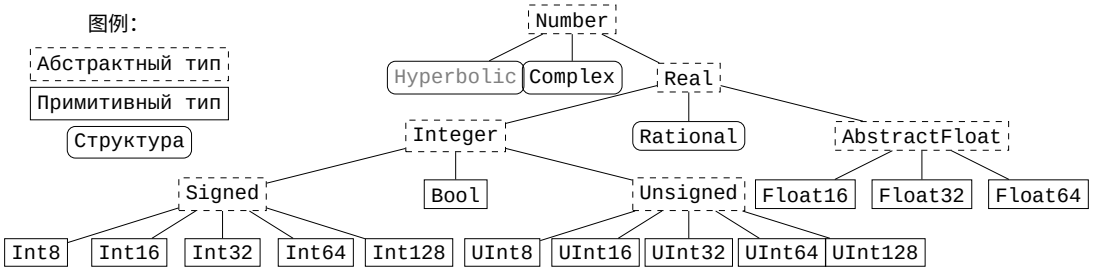

图 1. 双曲数在 Julia 类型层次结构中的位置

该结构很简单，只包含两个参数类型 T 的字段。与此同时，要求类型 T 是抽象类型 实数 的子类型（语法为 T<:实数）。类型 双曲数 本身是抽象类型 数 的子类型（见图 1）。因此，双曲数被纳入了现有的数值类型层次结构中。

定义结构后，可以通过调用默认构造函数创建 Hyperbolic 类型的新对象。例如，数 $h = 1 +\mathrm{j}3$ 可按如下方式定义：

```
h  =双曲数{Float64(1, 3)
```

创建后，可以通过以下方式访问结构体的字段：h.re 和 h.jm。不过，尝试修改已有对象的字段值会导致错误，因为结构体是不可变实体。

如果构造函数的两个参数具有相同的 T 类型，就可以省略花括号中的类型说明，因为它会根据传入参数的类型自动推断。

```
h  = 双曲数(1,3)
```

然而，如果参数的类型 不同，默认构造函数将无法进行隐式类型转换并创建 新对象。在这种情况下，必须显式指定参数化类型

```
    # Float64 和 Int64h  =双曲数(1.0, 3) #
错误h  = 双曲数{Float64}(1.0, 3)# 正确
```

B. 额外构造函数

默认构造函数是一个普通函数，其名称与类型名称相同。为这个函数添加额外的方法，就可以创建额外的构造函数来处理各种特殊情况。

例如，为了避免每次都指定参数化类型，可以添加一个如下形式的新构造函数：

```
"""构造函数 No2"""
函数双曲数(x::实数, y::实数)return双
曲数(promote(x, y)...)end
```

函数 提升 会将传入的参数转换为共同类型，并以元组形式返回结果。后缀运算符 ... 会解包元组，并将其元素作为参数传递给构造函数。在语言核心中，已为抽象类型 实数的所有子类型定义了类型转换规则，因此现在该构造函数可以正确处理任意参数组合，关键是要满足规则 T<:实数。例如，以下代码可以正常运行：

```
    # 有理数和 Float64h  =双曲数(1//3, pi)>>双曲
数{Float64(0.5, 3.141592653589793)
```

我们向构造函数传入了一个有理数（类型Rational）和一个内置的全局常量（数值π），其类型为 Float64。随后，类型转换规则生效，两个参数都被转换为更通用的类型Float64 。

声明另外两个构造函数，就能创建虚部为零的双曲数：

```
"""构造函数 No3"""函数双曲数{T}(x::实数) where
{T<:实数}返回双曲数{T}(x, 0)结束"""构造函数 No4""
"函数 双曲数(x::实数)返回双曲数(提升(x, 0)...)结束
```

构造函数 No3 是参数化函数，使用 `where` 构造。参数 `T` 是抽象类型 实数 的子类型。构造函数 No4 的工作方式与构造函数 No2 类似。

另外两个构造函数允许将其他双曲数作为构造函数参数传入。

```
"""构造函数 No5""" 函数 双曲数{T}(h::双曲数) where {T<:实
数} 双曲数{T}(h.re, h.jm) 结束 """构造函数 No6""" 函数 双曲
数(h::双曲数) 返回双曲数(提升(h.re, h.jm)...) 结束
```

为了更加方便，也可以为虚数单位 \mathrm{j} 创建一个单独的常量：

```
const jm  =双曲数(0, 1)
```

C. 数据输出

为了能够以紧凑且易读的形式打印双曲数类型的值，应为 Base 模块中的 show 函数添加相应的方法。

```
函数Base.show(io::IO,
h::Hyperbolic)print(io, h.re,"+", h.jm, "
j")end
```

函数 `show` 用于将数据打印到控制台，具体来说，它会被函数 `println`和宏 `@show`调用。在下面的代码及其运行结果示例中，假定已经添加了针对双曲数的 `show` 方法。

D. 类型转换

在开始实现双曲数算术运算的方法之前，需要先定义类型转换规则。为此，应在 Base 模块中为 promote_rule 函数创建一个新方法。

```
function Base.promote_rule(::Type{Hyperbolic{T}}, ::Type{S}) where {T<:实数, S<:实数}
returnHyperbolic{promote_type(T, S)} end function Base.promote_rule(::Type{Hyperbolic{T}}, ::
Type{Hyperbolic{S}}) where {T<:实数, S<:实数} ↪returnHyperbolic{promote_type(T, S)} end
```

在 提升_规则中，指定需要转换为同一共同超类型的参数化类型。在我们的情况下，若其中一种类型是 实数 的子类型，则可以实现这一点，此时共同超类型将是 双曲数 类型。

为 promote_rule 添加方法后，就可以使用 promote、promote_type 和 convert 函数。

```
>>h =双曲数(1 // 2)>>提升(h, 1)(1//2+0//
1j, 1//1+0//1j)>>提升_类型(双曲数{Int64},
Float32)双曲数{Float32}
```

第一个函数我们已经熟悉了。第二个函数则允许推导出变量类型本身（而非具体变量值）的共同父类型。Julia 中的类型是一等对象（类型 DataType），可以将其赋值给其他变量、作为参数传递给函数等。

convert 函数允许转换具体值的类型，例如 :

```
>>convert(双曲数, 1)1+0j
```

E. 双曲数的算术运算

添加类型转换方法后，就可以开始添加算术运算的方法了。Julia 的特点是，算术运算并非以运算符的形式实现，而是以函数的形式实现。因此，以下调用都是有效的：

```
>>+(1,2)3>>+(1,2,3,4)10>>+((i for i in
1:10)...) # 从 1 到 10 的数字55
```

因此，为算术运算添加方法与为其他函数添加方法的过程并无不同。

Доба一元运算 + 和 - 的方法定义如下 :

```
Base.:+(h::双曲数) =双曲数(+h.re, +
h.jm)Base.:-(h::双曲数) =双曲数(-h.re, -h.jm)
```

这里使用了函数声明的简写。

同样地，也可以添加二元加法、减法、乘法和除法的方法。下面给出加法和乘法的代码。

```
# 二元 +与 * 函数 Base.:+(x::双曲数, y::双曲数) xx =
x.re + y.re yy = x.jm + y.jm 双曲数(xx, yy) end
函数 Base.:*(x::双曲数, y::双曲数) xx = x.re* y.re +
x.jm * y.jm yy = x.re* y.jm + x.je * y.re return
双曲数(xx, yy) end
```

V. 结论

我们以双曲数的实现为例，探讨了 Julia 语言的核心机制——多重分派。这个示例涉及了该语言中的一些概念，例如数据类型层次结构、复合数据类型、类型转换机制、函数重载（用 Julia 语言的术语来说，就是为函数创建新的方法）等。

致谢

本出版物在俄罗斯人民友谊大学战略学术领导力计划的支持下完成。

---

[1] Bezanson J., Edelman A., Karpinski S., Shah V. B. Julia：一种数值计算的新方法 // SIAM Review. ---- 2017年 ---- 1月 ---- Vol. 59, no. 1. ---- P. 65--98.[2] Gevorkyan M. N., Kulyabov D. S., Sevastyanov L. A. 面向科学计算的 Julia 编程语言综述 // 第六届“分布式计算与网格技术在科学与教育中的应用”国际会议. ---- 2014年 ---- P. 27.[3] Oliphant T. E. NumPy指南. ---- 第2版. ---- CreateSpace独立出版平台, 2015年. ---- ISBN: 978- 1517300074.[4] Zappa Nardelli F., Belyakova J., Pelenitsyn A., Chung B., Bezanson J., Vitek J. Julia 子类型：一种有理重构 // ACM 编程语言会议录. ---- 2018年 ---- 10月 ---- Vol. 2, no. OOPSLA. ---- P. 1--27.[5] Driesen K., H\"olzle U., Vitek J. 流水线处理器上的消息分派 // ECOOP'95 --- 面向对象编程，第九届欧洲会议，\r 丹麦奥胡斯，1995年8月7日至11日 / 编者：M. Tokoro, R. Pareschi. ---- 《计算机科学讲义》. Springer Berlin Heidelberg, 1995年. ---- 253--282 p. ---- ISBN: 9783540601609.[6] Muschevici R., Potanin A., Tempero E., Noble J. 多重分派的实践应用 // OOPSLA'08：第23届ACM SIGPLAN面向对象编程系统、语言与应用会议论文集. ---- ACM Press, 2008年. ---- 10. ---- P. 563--582.[7] Gowda S., Ma Y., Cheli A., Gw\'o\'zzd\'z M., Shah V. B., Edelman A., Rackauckas C. 通过多重分派实现高性能符号数值计算 // ACM 计算机代数通讯. ---- 2022年 ---- 1月 ---- Vol. 55, no. 3. ---- P. 92--96. [8] Яглом И. М. 复数及其在几何学中的应用 // 《数学、数学教学、应用与历史》. — 1961年. — 《数学启蒙》第2辑第6卷. — С. 61–106. — 获取地址：http://mi.mathnet. ru/mp680. [9] Яглом И. М., Розенфельд Б. А., Ясинская Е. У. 射影度量 // 《数学科学进展》. — 1964年. — Т. 19, No 5 (119). — С. 51–113.[10] Kulyabov D. S., Korolkova A. V., Sevastianov L. A. 用于相对论运算的复数. ---- 2021年. ---- 12月.[11] Kulyabov D. S., Korolkova A. V., Gevorkyan M. N. 作为爱因斯坦数的双曲数 // 《物理学杂志：会议系列》. ---- 2020年 ---- 5月 ---- Vol. 1557. ---- P. 012027.[12] Fjelstad P. 通过佩尔普莱克斯数拓展狭义相对论 // 《美国物理学杂志》. ---- 1986年 ---- 5月 ---- Vol. 54, no. 5. ---- P. 416--422.[13] Band W. 关于通过佩尔普莱克斯数拓展相对论的评论 // 《美国物理学杂志》. ---- 1988年 ---- 5月 ---- Vol. 56, no. 5. ---- P. 469--469.[14] Rooney J. 三类复数与平面变换 // 《环境与规划B：规划与设计》. ---- 1978年. ---- Vol. 5, no. 1. ---- P. 89--99.[15] Rooney J. 力学中的广义复数 // 《机器人与操纵器理论及实践进展》 / 编者：M. Ceccarelli, V. A. Glazunov. ---- 楚格：Springer国际出版公司, 2014年. ---- 《机构与机器科学》第22卷. ---- P. 55--62.[16] Gevorkyan M. N., Korolkova A. V., Kulyabov D. S. Julia语言中广义复数的实现方法 // 第十届国际会议“信息与电信技术及高科技系统数学建模”（ITTMM-2020）框架内的信息技术与科学计算研讨会 / 编者：D. S. Kulyabov, K. E. Samouylov, L. A. Sevastianov. ---- CEUR研讨会论文集第2639卷. ---- 亚琛，2020年 ---- 4月 ---- P. 141--157. ---- 获取地址：http://ceur-ws.org/Vol-2639/paper-13.pdf.