

../Programs.jl

Contents

[|- Project.toml](#)

[|- README.md](#)

/src

[|- Programs.jl](#)

/ch01

/args

[|- args.jl](#)

[|- main.jl](#)

/hello

[|- main.jl](#)

/ch03

/constant

[|- main.jl](#)

/variable

[|- main.jl](#)

/ch04

/primitive

[|- main.jl](#)

/structure

[|- main.jl](#)

/ch05

/big

[|- main.jl](#)

/float

[|- main.jl](#)

/int

[|- main.jl](#)

/operator

[|- main.jl](#)

/ch06

/operation

[|- main.jl](#)

/special

[|- main.jl](#)

/value

[|- main.jl](#)

/ch07

/parametric

/abst

[|- main.jl](#)

/comp

[|- main.jl](#)

/unionall

[|- main.jl](#)

/tuple

/named

[|- main.jl](#)

/normal

[|- main.jl](#)

/vararg

[|- main.jl](#)

/ch08

/dict

/basic

[|- main.jl](#)

/operation

[|- main.jl](#)

/index_and_iter

[|- main.jl](#)

/set

[|- main.jl](#)

/ch09

/basic

[|- main.jl](#)

/construction

[|- main.jl](#)

/index1

[|- main.jl](#)

/index2

[|- main.jl](#)

/iter

[|- main.jl](#)

/representation

[|- main.jl](#)

/search

[|- main.jl](#)

/type

| - main.jl

/view

| - main.jl

/Project.toml

[to top](#)

```
name = "Programs"
uuid = "e525bb1a-bb1e-11e9-07f5-1125a61c95e2"
authors = ["robert.hao <hypermind@outlook.com>"]
version = "0.1.0"
```

/README.md

[to top](#)

Programs.jl

The examples for book [Julia Programming Basics](https://github.com/hyper0x/JuliaBasics) (<https://github.com/hyper0x/JuliaBasics>).

/src/Programs.jl

[to top](#)

```
module Programs

greet() = print("Hello World!")

end # module
```

/src/ch01/args/args.jl

[to top](#)

```
# 提供与命令参数有关功能的文件。  
# - Julia version: 1.2.0  
# - Author: HaoLin  
# - Date: 2020-01-01
```

```
"""
```

```
    get_parameter(key::String, first::Bool=true)
```

根据参数获取指定的命令行参数值。

参数`key`代表命令行参数的名称。参数`first`代表一种获取策略。

如果参数`first`的值为`true`，那么无论有多少个同名的命令行参数，都只获取第一个。否则只获取最后一个。

```
"""
```

```
function get_parameter(key::String, first::Bool = true)
```

```
    if length(ARGS) == 0
```

```
        return "", -1
```

```
    end
```

```
    prefix = "--" * key
```

```
    key_end = 2 + length(key)
```

```
    value_begin = key_end + 2
```

```
    myargs = filter(a->a[1:key_end] == prefix ? true : false, ARGS)
```

```
    values = map(a->a[value_begin:end], myargs)
```

```
    first ? (values[1], 1) : (values[end], length(values))
```

```
end
```

[/src/ch01/args/main.jl](#)

[to top](#)

```
# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

module MyArgs

include("args.jl")

end

name, _ = MyArgs.get_parameter("name", true)
if name == ""
    name = "handsome"
end

println("Hey, $(name)!")
```

[/src/ch01/hello/main.jl](#)

[to top](#)

```
# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

println("Hey, Julia!")
```

[/src/ch03/constant/main.jl](#)

[to top](#)

```
# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

const A = 2020
println("Constant A: $(A)")
A = 2050 #= 这里会引发警告。 =#
println("Constant A: $(A) (redefined)\n")

b = 2070
println("Variable b: $(b)")
A = b #= 这里会引发警告。 =#
println("Constant A: $(A) (redefined)\n")

const C = 2020
println("Constant C: $(C)")
f() = C + 30
println("Invoke function f about C: $(f())")
C = 2030
println("Constant C: $(C) (redefined)")
println("Invoke function f about C: $(f())\n")

# C = "2020" #= 这里会报错。 =#

const D = "2020"
println("Constant D: $(D)")
D = "2020"
println("Constant D: $(D) (redefined)")
d = "2020"
println("Variable d: $(d)")
D = d
println("Constant D: $(D) (redefined)")
```

/src/ch03/variable/main.jl

[to top](#)

```
# 示例的演示文件。  
# - Julia version: 1.2.0  
# - Author: HaoLin  
# - Date: 2020-01-01
```

```
""""  
    get_uint32(x)
```

把参数`x`的值的类型转换为`UInt32`，并返回转换后的值。

```
""""
```

```
function get_uint32(x)  
    # 附加类型标注的变量 y。  
    y::UInt32 = x  
    y  
end
```

```
z = get_uint32(2020)  
# 下一行代码中的 z::UInt32 是类型断言。  
println("Invoke function get_uint32 with 2020: $(z::UInt32) (type:  
$(typeof(z)))")
```

```
# 下一行的 x 是全局变量，不能附加类型标注。  
# x::UInt32 = 2020 #= 这里会报错。 =#
```

```
# 使用 Ref{T} 类型的全局常量替代全局变量。  
const xref = Ref{UInt32}(2020)  
# 操作符 [] 可用于存/取引用的值。  
println("xref: $(xref[]::UInt32)")
```

/src/ch04/primitive/main.jl

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

primitive type MyUInt64 <: Unsigned 64 end

# 针对 MyUInt64 类型的实例构造方法。
# 用于根据一个 UInt64 类型的值构造出一个 MyUInt64 类型的值。
MyUInt64(x::UInt64) = reinterpret(MyUInt64, x)

# 针对 UInt64 类型的实例构造方法。
# 用于根据一个 MyUInt64 类型的值构造出一个 UInt64 类型的值。
UInt64(x::MyUInt64) = reinterpret(UInt64, x)

# 显示函数的新方法，针对 MyUInt64 类型。
Base.show(io::IO, x::MyUInt64) = print(io, UInt64(x))

```

~~~~~

请注意，函数中的表达式`UInt64(666)`和`MyUInt64(x)`表示的都是对构造函数的调用。  
构造函数与对应的类型同名，专用于类型的实例化。

构造函数也可以拥有衍生方法，以应对不同的构造原材料（如这里的`666`和`x`）。

比如，我们在前面用简易法定义了一个构造方法`MyUInt64(x::UInt64)`。

这个构造方法可以根据一个`UInt64`类型的值，构造出一个`MyUInt64`类型的实例。

正因为如此，下面的函数调用`MyUInt64(x)`才能够成功。

~~~~~

```

x = UInt64(666)
y = MyUInt64(x)
println(UInt64(y))

```

[/src/ch04/structure/main.jl](#)

[to top](#)


```
# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 代表用户的不可变复合类型。
struct User
    name::String
    reg_year::UInt16
    extra
end

# 代表个人的可变复合类型。
mutable struct Person
    name::String
    age::UInt8
    extra
end

u1 = User("Robert", 2000, "something")
u2 = User("Robert", UInt16(2000), "something")
println("u1: $(u1)")
println("u2: $(u2)")
println("u1 === u2? $(u1 === u2)")

p1 = Person("Robert", 30, "something")
p1.age = 37
p2 = Person("Robert", 37, "something")
println("p1: $(p1)")
println("p2: $(p2)")
println("p1 === p2? $(p1 === p2)")
```

/src/ch05/big/main.jl

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 任意精度的整数
# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

big_int = big"1234567890123456789012345678901234567890"
println("The big integer: $(big_int) ($(typeof(big_int)))\n")

float_number = 5 / 7
println("The float number: $(float_number) ($(typeof(float_number)))")
big_float = BigFloat(5 / 7)
println("The big float: $(big_float) ($(typeof(big_float)))\n")

current_precision = precision(BigFloat)
current_rounding = rounding(BigFloat)
println("The current precision of BigFloat: $(current_precision)")
println("The current rounding of BigFloat: $(current_rounding)\n")

new_precision = 35
new_rounding = RoundDown
setprecision(new_precision) # 等同于 setprecision(BigFloat, new_precision)
setrounding(BigFloat, new_rounding)
println("The new precision of BigFloat: $(new_precision)")
println("The new rounding of BigFloat: $(new_rounding)")

big_float2 = BigFloat(5 / 7)
println("The big float(2): $(big_float2) ($(typeof(big_float2)))\n")

new_rounding = RoundUp
setrounding(BigFloat, new_rounding)
println("The new rounding of BigFloat: $(new_rounding)")
big_float3 = BigFloat(5 / 7)
println("The big float(3): $(big_float3) ($(typeof(big_float3)))")

```

/src/ch05/float/main.jl

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 一个数组，容纳了所有可以代表浮点数的原语类型。
float_types = [Float16, Float32, Float64]

# 一个循环，以此为每个浮点数类型做相同的判断和输出。
for T in float_types
    # 打印：最大值、最小值以及它们的类型。
    val_max = typemax(T)
    val_min = typemin(T)
    type_max_more = typeof(val_max)
    type_min_less = typeof(val_min)
    println("$T: max: $(val_max) ($(type_max_more)); min: $(val_min)
    ($(type_min_less))")
end

```

/src/ch05/int/main.jl

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 一个数组，容纳了所有可以代表整数的原语类型。
int_types = [Int8, Int16, Int32, Int64, Int128,
             UInt8, UInt16, UInt32, UInt64, UInt128]

println("Do they wraparound for overflow of integer values?")
# 一个循环，以此为每个整数类型做相同的判断和输出。
for T in int_types
    # 判断：最大值加 1 是否等于最小值，以及最小值减 1 是否等于最大值。
    max_more = typemax(T) + 1
    min_less = typemin(T) - 1
    wraparound = max_more == typemin(T) && min_less == typemax(T)
    type_max_more = typeof(max_more)
    type_min_less = typeof(min_less)
    hint = "[types: $(type_max_more) (max_more); $(lpad(type_min_less, 7))
    (min_less)]"
    println("$$(lpad(T, 7)): $(rpad(wraparound, 5)) $(hint)")
end

```

/src/ch05/operator/main.jl

[to top](#)

```
# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 操作符 ===
println("$(NaN) === $(NaN): $(NaN === NaN)")
println("$(Inf) === $(Inf): $(Inf === Inf)")
println("$(Inf) === $(Inf): $(Inf === Inf)")
println("$(Inf) === $(Inf): $(Inf === Inf)")
println("$(0.0) === $(0.0): $(0.0 === 0.0)\n")

# 操作符 ==
println("$(NaN) == $(NaN): $(NaN == NaN)")
println("$(Inf) == $(Inf): $(Inf == Inf)")
println("$(Inf) == $(Inf): $(Inf == Inf)")
println("$(Inf) == $(Inf): $(Inf == Inf)")
println("$(0.0) == $(0.0): $(0.0 == 0.0)\n")

# 函数 isequal
println("$(NaN) is equals to $(NaN): $(isequal(NaN, NaN))")
println("$(Inf) is equals to $(Inf): $(isequal(Inf, Inf))")
println("$(Inf) is equals to $(Inf): $(isequal(Inf, Inf))")
println("$(Inf) is equals to $(Inf): $(isequal(Inf, Inf))")
println("$(0.0) is equals to $(0.0): $(isequal(0.0, 0.0))")
```

/src/ch06/operation/main.jl

[to top](#)

```
# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 获取字符串的尺寸和长度。
comment1 = "codeunit 函数会返回给定字符串对象的代码单元类型"
println("The value of comment1: $(comment1)\n")
size1, len1 = sizeof(comment1), length(comment1)

println("The size of comment1: $(size1)")
println("The size of \"a\": $(sizeof(\"a\"))")
println("The size of \"中\": $(sizeof(\"中\"))\n")

println("The length of comment1: $(len1)")
println("The length of \"a\": $(length(\"a\"))")
println("The length of \"中\": $(length(\"中\"))\n")

# 字符串的索引。
char_index1 = 13
char1 = comment1[char_index1]
println("The $(char_index1)th char in comment1: $(char1)")
println("The last char in comment1: $(comment1[end])")
char_index2 = thisind(comment1, size1)
println("The last char in comment1: $(comment1[char_index2])")
char_index3_1 = nextind(comment1, size1 - 5)
char_index3_2 = thisind(comment1, size1)
println("The last char in comment1:
$(comment1[char_index3_1:char_index3_2])\n")

# 字符串的拼接。
dup_chars_1 = "\xe2\x88"
dup_chars_2 = "\x80"
dup_chars_3 = "\xe2\x88\x80"
dup_chars = "\xe2\x88" * "\x80" * "\xe2\x88\x80"
println("The string: $(string(dup_chars_1, dup_chars_2, dup_chars_3))")
println("The string: $(dup_chars)")
println("Is string $(dup_chars) valid? $(isvalid(dup_chars) ? "Yes" : "No")\n")

# 字符串的搜索。
slogan1 = "Julia 编程入门很简单。"
println("The value of slogan1: $(slogan1)")
target1 = "入门"
println("The position of \"$(target1)\" in slogan1: $(findfirst(target1,
slogan1))\n")

slogan2 = "Julia 编程入门，跟着入门很简单。"
println("The value of slogan2: $(slogan2)")
target2 = "入门"
index2 = 19
println("The position of \"$(target2)\" in slogan2: $(findprev(target2,
slogan2, index2))",
```

```
        " (before index ${index2}))")
println("The position of \"$(target2)\" in slogan2: ${findnext(target2,
slogan2, index2))",
        " (after index ${index2}))")
target3 = '编'
index3 = findfirst(isequal(target3), slogan2)
println("The index of \"$(target3)\" in slogan2: ${index3}\n")

# 字符串的比较。
println("\"Julia\" < \"Julie\": ${\"Julia\" < \"Julie\"}")
println("\"Julia\" < \"Julian\": ${\"Julia\" < \"Julian\"}")
println("\"Michael\" < \"Mike\": ${\"Michael\" < \"Mike\"}\n")
```

[/src/ch06/special/main.jl](#)

[to top](#)

```
# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 原始字符串。
str_origin = raw"Julia\n\n"
println("The original string: $(str_origin)\n")

# 整数和浮点数。
str_bigint = big"1314"
println("The string representing a BigInt value: $(str_bigint)")

str_bigfloat = big"3.14e-2"
println("The string representing a BigFloat value: $(str_bigfloat)\n")

# 版本号。
str_version = v"1.0.0-beta1"
println("The string representing a version value: $(str_version)\n")

str_version1 = v"1.0.0"
str_version2 = v"1.0.0-"
str_version3 = v"1.0.0-alpha"
str_version4_1 = v"1.0.0-beta1"
str_version4_2 = v"1.0.0-beta.1"

println("$(str_version2) < $(str_version1): $(str_version2 < str_version1)")
println("$(str_version2) < $(str_version3): $(str_version2 < str_version3)")
println("$(str_version2) < $(str_version4_1): $(str_version2 < str_version4_1)\n")

println("$(str_version1) < $(str_version3): $(str_version1 < str_version3)")
println("$(str_version1) < $(str_version4_1): $(str_version1 < str_version4_1)")
println("$(str_version3) < $(str_version4_1): $(str_version3 < str_version4_1)")
println("$(str_version4_1) < $(str_version4_2): $(str_version4_1 < str_version4_2)\n")

str_version5 = v"1.0.0+"
str_version6_1 = v"1.0.0+win64"
str_version6_2 = v"1.0.0+win64.1"

println("$(str_version5) > $(str_version1): $(str_version5 > str_version1)")
println("$(str_version5) > $(str_version2): $(str_version5 > str_version2)")
println("$(str_version5) > $(str_version6_1): $(str_version5 > str_version6_1)")
println("$(str_version6_1) > $(str_version6_2): $(str_version6_1 > str_version6_2)\n")

# 正则表达式。
str_regex1 = r"\+((?:[0-9a-z-]+\.)*[0-9a-z-]+)"
```

```
println("A string representing a RegEx: ${str_regex1}")
build_info1 = "+win64.20200101"
println("The build information 1: ${build_info1}\n")

rm1 = match(str_regex1, build_info1)
println("The regex match named rm1: ${rm1}")
println("  rm1.match: ${rm1.match}")
println("  rm1.captures: ${rm1.captures}")
println("  rm1.offset: ${rm1.offset}")
println("  rm1.offsets: ${rm1.offsets}")
println("  rm1.regex: ${rm1.regex}\n")

str_regex2 = r"(.*\.) (\d{4})-(\d{2})-(\d{2})T(\d{2}):(\d{2})"
println("Another string representing a RegEx: ${str_regex2}")
build_info2 = "+win64.2020-01-01T21:01"
println("The build information 2: ${build_info2}")
str_sub2 = s"\1\2\3\4\5\6"
println("The substitution string: ${str_sub2}")
build_info2_new = replace(build_info2, str_regex2 => str_sub2)
println("The new value for build information 2: ${build_info2_new}\n")

# 字节数组。
ba1 = b"\u4e2d\xe5\x9b\xbd"
println("The string representing a byte array: ${ba1}")
println("The characters represented by this byte array: ${String(ba1)}\n")
```

[/src/ch06/value/main.jl](#)

[to top](#)


```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 字符相关的示例。
println("Chars: $('中') $('\\u4e2d') $('\\U004e2d')")

println("Chars: $('a') $('\\x61') $('\\141') \\n")

println("'a' < '中': $('a' < '中') \\n")

println("Int64('中'): $(Int64('中'))")
println("Char(20013): $(Char(20013)) \\n")

println("isvalid('中'): $(isvalid('中'))")
println("isvalid(Char(0x11ffff)): $(isvalid(Char(0x11ffff))) \\n")

code_point = codepoint('中')
println("codepoint('中'): $(code_point) (type: $(typeof(code_point))) \\n\\n")

# 字符串相关的示例。
str1 = " * CN          \\nUS\\nEN\\nRU \\n\\t\\n"
println("String 1:\\n$(str1)")

str2 = """
Julia
  Python
    Golang

      Java
      """
println("String 2:\\n$(str2)", '\\n')

comment1 = "codeunit 函数会返回给定字符串对象的代码单元类型"
println("The value of comment1: $(comment1)")
println("The code unit type of comment1: $(codeunit(comment1))")
println("The code unit number of comment1: $(ncodeunits(comment1))")

sn1 = 8
code_unit1 = codeunit(comment1, sn1)
println("The $(sn1)th code unit of comment1: $(code_unit1)\\n")

```

/src/ch07/parametric/abst/main.jl

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 代表商品的类型。
abstract type Goods end

# 代表首饰的类型。
abstract type Jewelry <: Goods end

# 代表项链的类型。
struct Necklace <: Jewelry end

# 代表戒指的类型。
struct Ring <: Jewelry end

# 代表储物空间的抽象类型。
abstract type StorageSpace{T} end

# 代表抽屉的类型。
mutable struct Drawer{T} <: StorageSpace{T}
    content::T
end

# 代表展柜的类型。
mutable struct Showcase{T <: Goods} <: StorageSpace{T}
    drawer1::Drawer{T}
    drawer2::Drawer{T}
end

showcase_jewelry = Showcase{Jewelry}(Drawer{Jewelry}(Necklace()),
Drawer{Jewelry}(Ring()))

#####
    describe(showcase::Showcase)::String

用于生成展柜描述的函数。
#####
function describe(showcase::Showcase)::String
    text = "A $(typeof(showcase.drawer1.content)); A
$(typeof(showcase.drawer2.content))"
end

println("The jewelry showcase: $(describe(showcase_jewelry))\n")

# 代表玩具的类型。
abstract type Toy <: Goods end

# 代表毛绒玩具的类型。

```

```

struct StuffedToy <: Toy end

# 代表电动玩具的类型。
struct ElectricToy <: Toy end

# 代表玩具箱的抽象类型。
abstract type ToyBox{T <: Toy} <: StorageSpace{T} end

# 代表纸板箱的类型。
mutable struct Carton{T <: Goods} <: ToyBox{T}
    content::T
end

# 下面这行代码会引发一个类型错误。
# toy_box = Carton{Goods}(StuffedToy())

toy_box = Carton{Toy}(StuffedToy())

"""
    describe(toy_box::ToyBox)::String

用于生成玩具箱描述的函数。
"""
function describe(toy_box::ToyBox)::String
    text = "A $(typeof(toy_box.content)) ($(typeof(toy_box)))"
end

println("The toy box: $(describe(toy_box))")

```

[/src/ch07/parametric/comp/main.jl](#)

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 代表抽屉的类型。
mutable struct Drawer{T}
    content::T
end

# 代表展柜的类型。
mutable struct Showcase{T}
    drawer1::Drawer{T}
    drawer2::Drawer{T}
end

# 代表首饰的类型。
abstract type Jewelry end

# 代表项链的类型。
struct Necklace <: Jewelry
    # 此处忽略项链的具体特征。
end

# 代表戒指的类型。
struct Ring <: Jewelry
    # 此处忽略戒指的具体特征。
end

# 代表首饰店的类型。
mutable struct JewelryShop{T <: Jewelry}
    showcase1::Showcase{Necklace}
    showcase2::Showcase{Ring}
    showcase3::Showcase{Jewelry}
    showcase4::Showcase{T}
end

showcase1 = Showcase{Necklace}(Drawer(Necklace()), Drawer(Necklace()))
showcase2 = Showcase{Ring}(Drawer(Ring()), Drawer(Ring()))
showcase3 = Showcase{Jewelry}(Drawer{Jewelry}(Necklace()), Drawer{Jewelry}(Ring()))
showcase4_1 = Showcase{Jewelry}(Drawer{Jewelry}(Necklace()), Drawer{Jewelry}(Ring()))
showcase4_2 = Showcase{Ring}(Drawer(Ring()), Drawer(Ring()))

jewelryShop1 = JewelryShop{Jewelry}(showcase1, showcase2, showcase3, showcase4_1)
jewelryShop2 = JewelryShop{Ring}(showcase1, showcase2, showcase3, showcase4_2)

# 用于生成首饰描述的函数。

```

```

describe(jewelry::Jewelry) = "A ${typeof(jewelry)}"

# 用于生成抽屉描述的函数。
# 第一种写法。
# describe(drawer::Drawer{Jewelry}) = "${describe(drawer.content)}"
# describe(drawer::Drawer{Necklace}) = "${describe(drawer.content)}"
# describe(drawer::Drawer{Ring}) = "${describe(drawer.content)}"
# 第二种写法。
# describe(drawer::Drawer) = "${describe(drawer.content)}"
# 第三种写法。
describe(drawer::Drawer{<:Jewelry}) = "${describe(drawer.content)}"

"""
    describe(shop::JewelryShop)::String

用于生成店铺描述的函数。
"""
function describe(shop::JewelryShop)::String
    text = """
        Showcase 1: ${describe(shop.showcase1.drawer1)};
        ${describe(shop.showcase1.drawer2)}
        Showcase 2: ${describe(shop.showcase2.drawer1)};
        ${describe(shop.showcase2.drawer2)}
        Showcase 3: ${describe(shop.showcase3.drawer1)};
        ${describe(shop.showcase3.drawer2)}
        Showcase 4: ${describe(shop.showcase4.drawer1)};
        ${describe(shop.showcase4.drawer2)}
    """
end

println("Jewelry Shop 1:\n${describe(jewelryShop1)}")
println("Jewelry Shop 2:\n${describe(jewelryShop2)}")

```

[/src/ch07/parametric/unionall/main.jl](#)

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 代表首饰的类型。
abstract type Jewelry end

# 代表戒指的类型。
struct Ring <: Jewelry end

# 代表抽屉的类型。
mutable struct Drawer{T}
    content::T
end

println("The type of $(Drawer{Jewelry}) is $(typeof(Drawer{Jewelry})).")
println("The type of $(Drawer{}) is $(typeof(Drawer{})).\n")

"""
    parse_and_eval(expr_text::String)

用于解析并求值的函数。它会先把文本解析为表达式，再对该表达式进行求值，最后返回求值结果。
"""
function parse_and_eval(expr_text::String)
    eval(Meta.parse(expr_text))
end

expr_text1 = "Array{<:Jewelry,1}"
expr1 = parse_and_eval(expr_text1)
println("The expression \"$(expr_text1)\" represents \"$(expr1)\". (type: $(typeof(expr1)))")

expr_text2 = "Array{T,N} where T<:Jewelry where N"
expr2 = parse_and_eval(expr_text2)
println("The expression \"$(expr_text2)\" represents \"$(expr2)\". (type: $(typeof(expr2)))")

expr_text3 = "JewelryArray{N} = Array{<:Jewelry, N}"
expr3 = parse_and_eval(expr_text3)
println("The expression \"$(expr_text3)\" represents \"$(expr3)\". (type: $(typeof(expr3)))")

```

/src/ch07/tuple/named/main.jl

[to top](#)

```
# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 有名的元组。
named_tuple1 = (name="Robert", reg_year=2020, extra="something")
println("The named tuple called named_tuple1: $(named_tuple1)")
println("The type of named_tuple1: $(typeof(named_tuple1))")
elem_name1 = :reg_year
elem_value1 = named_tuple1[elem_name1]
println("The value of element named $(elem_name1) in named_tuple1:
$(elem_value1)\n")

named_tuple2 = NamedTuple{(:name, :reg_year, :extra)}(("Robert", 2020,
"something"))
println("The named tuple called named_tuple2: $(named_tuple2)")
println("named_tuple1 == named_tuple2: $(named_tuple1 == named_tuple2)\n")
```

/src/ch07/tuple/normal/main.jl

[to top](#)

```
# 示例的演示文件。  
# - Julia version: 1.2.0  
# - Author: HaoLin  
# - Date: 2020-01-01
```

```
# 普通的元组。
```

```
tt1 = Tuple{Real}  
tt2 = Tuple{Integer}  
tt3 = Tuple{Real,Char}  
tt4 = Tuple{Integer,Char}  
tt5 = Tuple{Real,AbstractChar}  
tt6 = Tuple{Integer, AbstractChar}  
tt7 = Tuple{Integer, String}
```

```
println("$ (tt1) >: $ (tt2) -> $ (tt1 >: tt2)")  
println("$ (tt3) >: $ (tt4) -> $ (tt3 >: tt4)")  
println("$ (tt5) >: $ (tt4) -> $ (tt5 >: tt4)")  
println("$ (tt3) >: $ (tt6) -> $ (tt3 >: tt6)")  
println("$ (tt5) >: $ (tt7) -> $ (tt5 >: tt7)")  
println("$ (tt3) >: $ (tt2) -> $ (tt3 >: tt2)")  
println("$ (tt1) >: $ (tt4) -> $ (tt1 >: tt4)\n")
```

```
tuple1 = (125, 3.1, '中', "编程")  
println("The tuple named tuple1: $(tuple1)")  
println("The type of tuple1: $(typeof(tuple1))")  
println("The first three element value of tuple1: $(tuple1[1:3])\n")
```

```
target1 = '中'  
isequal1 = isequal(target1)  
println("The position of \"$(target1)\" in tuple1: $(findfirst(isequal1,  
tuple1)))")  
index1_1 = 4  
index1_2 = 2  
println("The position of \"$(target1)\" in tuple1: $(findprev(isequal1, tuple1,  
index1_1))",  
" (before index $(index1_1))")  
println("The position of \"$(target1)\" in tuple1: $(findnext(isequal1, tuple1,  
index1_2))",  
" (after index $(index1_2))\n")
```

```
println("Double tuple1: $((tuple1..., tuple1...))")  
tuple2 = ([1, 2, 3], [4, 5, 6, 7])  
println("The tuple named tuple2: $(tuple2)")  
println("Double tuple2: $((tuple2..., tuple2...))\n")
```

```
println("isbitstype(Tuple{Int64,Float64,Char}) ->  
$(isbitstype(Tuple{Int64,Float64,Char}))")  
println("isbitstype(Tuple{Float64,String}) ->  
$(isbitstype(Tuple{Float64,String}))")  
println("isbitstype(Tuple{Real}) -> $(isbitstype(Tuple{Real}))\n")
```

```
tuple2_2 = (tuple2..., tuple2...)
```



```
println("The tuple named tuple2_2 based on tuple2: $(tuple2_2)")
println("Modify the value of the element in position tuple2[2][1]...")
tuple2[2][1] = tuple2[2][1] * 10
println("The tuple named tuple2: $(tuple2)")
println("The tuple named tuple2_2: $(tuple2_2)\n")
```

/src/ch07/tuple/vararg/main.jl

[to top](#)

```
# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 可变参数的元组。
vtt1 = Tuple{Vararg{String}}
println("A type of vararg tuple called vtt1: $(vtt1)")
println("isa(), vtt1 -> $(isa(), vtt1)")
println("isa(\"Julia\",), vtt1 -> $(isa(\"Julia\",), vtt1)")
println("isa(\"Julia\"), vtt1 -> $(isa(\"Julia\"), vtt1)\n")

vtt2 = Tuple{Vararg{Int},5}
println("Another type of vararg tuple called vtt2: $(vtt2)")
tt1 = Tuple{Int,Int,Int,Int,Int}
println("A type of normal tuple called tt1: $(tt1)")
println("vtt2 == tt1 -> $(vtt2 == tt1)\n")

tuple1 = (1,2,3,4,5)
println("The tuple named tuple1: $(tuple1)")
println("isa(tuple1, vtt2) -> $(isa(tuple1, vtt2)")
tuple2 = (1,2,3,4,5,6,7,8,9,0)
println("The tuple named tuple2: $(tuple2)")
println("isa(tuple1, vtt2) -> $(isa(tuple2, vtt2))\n")
```

/src/ch08/dict/basic/main.jl

[to top](#)

```
# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 自定义的键类型。
mutable struct MyKey
    code::String
    sn::Int128
end

key1 = MyKey("mykey", 1); println("Key 1: $(key1)")
key2 = MyKey("mykey", 1); println("Key 2: $(key2)\n")

println("key1 == key2 -> ", key1 == key2)
println("isequal(key1, key2) -> ", isequal(key1, key2), "\n")

# 下面为 MyKey 类型定义 == 方法。
import Base.==
==(x::MyKey, y::MyKey) = x.code == y.code && x.sn == y.sn

println("key1 == key2 -> ", key1 == key2)
println("isequal(key1, key2) -> ", isequal(key1, key2), "\n")

println("hash(key1) == hash(key2) -> ", hash(key1) == hash(key2), "\n")

# 下面为 MyKey 类型定义 hash 方法。
import Base.hash
hash(k::MyKey, h::UInt) = hash(k.sn, hash(k.code, h))

println("hash(key1) == hash(key2) -> ", hash(key1) == hash(key2), "\n\n")

println("Change the value of the field sn in key2 to 2...")
key2.sn = 2
dict1 = Dict{MyKey, Int64}()
dict1[key1] = 10; dict1[key2] = 20
println("Dict 1: $(dict1)\n")

judge1(key, has) =
    "Is there a key $(key) in the dictionary dict1? $(has)"

println(judge1(key1, haskey(dict1, key1)))
println(judge1(key2, haskey(dict1, key2)), "\n")

println("Change the value of the field sn in key1 to 2...")
key1.sn = 2
println(judge1(key1, haskey(dict1, key1)), "\n")

println("Change the value of the field sn in key2 to 3...")
```

```
key2.sn = 3
println(judge1(key2, haskey(dict1, key2)), "\n\n")

# 构造字典的几种方式。
println("The dict (1): ", Dict([(1, "a"), (2, "b"), (3, "c")]))
println("The dict (2): ", Dict([1, "d"], [2, "e"], [3, "f"]))
println("The dict (3): ", Dict([1, "i"], [2, "j"], [3, "k"]))
println("The dict (4): ", Dict(1=>"x", 2=>"y", 3=>"z"), "\n")

pair1 = 1=>"u"
println("The pair (1): $(pair1) (type: $(typeof(pair1)))")
pair2 = Pair(1, "v")
println("The pair (2): $(pair2) (type: $(typeof(pair2)))\n")
```

/src/ch08/dict/operation/main.jl

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 存取键值对。
dict2 = Dict{String, Int64}{}
println("Store pair \"a\"=>1 into dictionary dict2...")
dict2["a"] = 1
println("Store pair \"b\"=>2 into dictionary dict2...")
dict2["b"] = 2
println("Dict 2 (1): $(dict2)")

function judge2(dict, key)
    has = haskey(dict, key)
    val = get(dict, key, "[nothing]")
    "Is there a key \"$(key)\" in the dictionary $(dict)? $(has) (value:
$(val))"
end

println(judge2(dict2, "a"))
println(judge2(dict2, "c"), "\n")

println("Store pair \"c\"=>2 into dict2 (if not present)...")
get!(dict2, "c", 3)
println("Dict 2 (2): $(dict2)")
println("Pop the pair for key \"c\": $(pop!(dict2, "c", "[default]"))")
println("Pop the pair for key \"d\": $(pop!(dict2, "d", "[default]"))")
println("Delete the pair for key \"b\" from dict2: $(delete!(dict2, "b"))")
println("Empty dictionary dict2. Dict (3): $(empty!(dict2))\n\n")

# 迭代。
dict3 = Dict{"a"=>1, "b"=>2, "c"=>3, "d"=>4, "e"=>5}
println("Dict 3: $(dict3)\n")

for p in dict3
    println("The pair in dict3: $(p) [$(typeof(p))]" )
end
println()

for (k,v) in dict3
    println("The key & value in dict3: ($k, $v) [$(typeof(k)), $(typeof(v))]" )
end
println("\n")

# 获取列表。
dict4 = Dict{"a"=>1, "b"=>2, "c"=>3}
println("Dict 4: $(dict4)\n")

keys_dict4 = keys(dict4)

```

```

println("The keys in dict4: $(keys_dict4) (type: $(keytype(dict4)))")
println("Is there a key \"a\" in the key list of dict4? $(in("a",
keys_dict4))\n")

values_dict4 = values(dict4)
println("The values in dict4: $(values_dict4) (type: $(valtype(dict4)))\n\n")

# 合并。
dict5 = Dict{"a"=>10, "b"=>20, "c"=>30}
println("Dict 5: $(dict5)")
println("Merge dict3, dict4 and dict5: $(merge(dict3, dict4, dict5))\n")

dict6 = Dict{"a"=>1.0, "d"=>4.0, "e"=>5.0}
println("Dict 6: $(dict6)")
println("Merge dict5 and dict6: $(merge(dict5, dict6))")
println("Merge dict5 and dict6 with combine `+`: $(merge(+, dict5, dict6))\n")

dict7 = Dict{1=>"a", 2=>"b", 3=>"c"}
println("Dict 7: $(dict7)")
dict8 = Dict{1.0=>'a', 3.0=>'c'}
println("Dict 8: $(dict8)")
println("Merge dict7 and dict8: $(merge(dict7, dict8))")
println("Merge dict7 and dict8 with combine `string`: $(merge(string, dict7,
dict8))\n")

dict9 = Dict{1=>"x", 2=>"y", 4=>"z"}
println("Dict 9 (1): $(dict9)")
println("Merge dict9 and dict7 (change dict9): $(merge!(dict9, dict7))")
println("Merge dict9 and dict8 with combine `*` (change dict9): $(merge!(*,
dict9, dict8))")
println("Dict 9 (2): $(dict9)\n")

```

/src/ch08/index_and_iter/main.jl

[to top](#)

```
# 示例的演示文件。  
# - Julia version: 1.2.0  
# - Author: HaoLin  
# - Date: 2020-01-01
```

```
""""  
    app(f, args...)
```

完全基于 `applicable` 函数的别名函数。只是为了缩短函数名称而已。

```
""""  
app(f, args...) = applicable(f, args...)
```

```
# 可索引对象。  
println("Is the string an indexable object? ", app(getindex, "1", 1))  
println("Is the tuple an indexable object? ", app(getindex, (1,), 1))  
println("Is the dict an indexable object? ", app(getindex, Dict{1=>"a"}, 1))  
println("Is the set an indexable object? ", app(getindex, Set{[1]}, 1))  
println("Is the array an indexable object? ", app(getindex, [1], 1), "\n")
```

```
# 可迭代对象。  
println("Is the string an iterable object? ", app(iterate, "1"))  
println("Is the tuple an iterable object? ", app(iterate, (1,)))  
println("Is the dict an iterable object? ", app(iterate, Dict{1=>"a"}))  
println("Is the set an iterable object? ", app(iterate, Set{[1]}))  
println("Is the array an iterable object? ", app(iterate, [1]), "\n")
```

`/src/ch08/set/main.jl`

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 构造集合。
println("The set (1): $(Set([1, 2, 4]))")
println("The set (1): $(Set((1, 2, 4)))")
println("The set (2): $(Set(Dict{1=>"x", 2=>"y", 4=>"z"}))")
println("The set (1): $(Set((1, 2.0, "4")))\n\n")

# 操作集合。
set1 = Set([1, 2, 3])
println("Set 1: $(set1)")
set2 = Set([2, 3, 4])
println("Set 2: $(set2)")
set3 = Set([3, 4, 5])
println("Set 3: $(set3)")
set4 = Set([4, 5, 6])
println("Set 4: $(set4)\n")

println("union with set1, set2 and set3: ", union(set1, set2, set3))
println("intersect with set1, set2 and set3: ", intersect(set1, set2, set3))
println("setdiff with set1, set2 and set3: ", setdiff(set1, set2, set3))
println("setdiff with set2, set1 and set3: ", setdiff(set2, set1, set3))
println("setdiff with set3, set1 and set1: ", setdiff(set3, set2, set1))
println("symdiff with set1 and set2: ", symdiff(set1, set2))
println("symdiff with set1, set2 and set3: ", symdiff(set1, set2, set3))
println("symdiff with set1, set2, set3 and set4: ", symdiff(set1, set2, set3,
set4), "\n")

```

/src/ch09/basic/main.jl

[to top](#)

```
# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

array2d = [[1,2,3,4,5] [6,7,8,9,10] [11,12,13,14,15] [16,17,18,19,20]
[21,22,23,24,25] [26,27,28,29,30]]
println("The array named array2d: $(array2d)\n")

println("The element type of array2d: $(eltype(array2d))")
println("The dimensions of array2d: $(ndims(array2d))")
println("The length of array2d: $(length(array2d))")
println("The size of array2d: $(size(array2d))\n")

println("The summary for array2d: $(summary(array2d))\n")
```

/src/ch09/construction/main.jl

[to top](#)


```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

"""
    basic_info(array::Array)

用于获取数组的基本信息的函数。
"""
basic_info(array::Array) = "$(array) ($(summary(array)))"

# 构造函数。
temp_array = Array{Int64}(undef, 4, 3, 2)
println("Array{Int64}(undef, 4, 3, 2) -> $(basic_info(temp_array))\n")

temp_array = Array{Int64, 4}(undef, (4, 3, 2, 2))
println("Array{Int64, 4}(undef, (4, 3, 2, 2)) -> $(basic_info(temp_array))\n")

temp_array = Array{Union{Nothing, String}}(nothing, 2, 2)
println("Array{Union{Nothing, String}}(nothing, 2, 2) ->
$(basic_info(temp_array))\n")

temp_array = Array{Union{Missing, Int64}}(missing, 2, 3)
println("Array{Union{Missing, Int64}}(missing, 2, 3) ->
$(basic_info(temp_array))\n\n")

# 另外的一些可以创建多维数组的函数。
temp_array = zeros(Float32, 4, 3)
println("zeros(Float32, 4, 3) -> $(basic_info(temp_array))\n")

temp_array = ones(Float32, 4, 3)
println("ones(Float32, 4, 3) -> $(basic_info(temp_array))\n")

temp_array = fill(1.0f-3, 2, 3)
println("fill(1.0f-3, 2, 3) -> $(basic_info(temp_array))\n")

```

[/src/ch09/index1/main.jl](#)

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 线性索引。
array2d = [[1,2,3,4,5] [6,7,8,9,10] [11,12,13,14,15] [16,17,18,19,20]
[21,22,23,24,25] [26,27,28,29,30]]
println("The array named array2d: $(array2d) ($(summary(array2d)))\n")

temp_result1 = array2d[1]
println("array2d[1] -> $(temp_result1)")
temp_result1 = array2d[[1,3,5]]
println("array2d[[1,3,5]] -> $(temp_result1)")
temp_result1 = array2d[1:6]
println("array2d[1:6] -> $(temp_result1)\n\n")

# 笛卡尔索引。
println("array2d[CartesianIndex(3, 2, 1)] -> $(array2d[CartesianIndex(3, 2, 1)])\n")

array3d = reshape(array2d, (3,5,2))
println("The array named array3d: $(array3d) ($(summary(array3d)))\n")

temp_result2 = array3d[:, :, 1]
println("array3d[:, :, 1] -> $(temp_result2) ($(summary(temp_result2)))")
temp_result2 = array3d[:, [1,2], 1]
println("array3d[:, [1,2], 1] -> $(temp_result2) ($(summary(temp_result2)))\n")

temp_result2 = array3d[:, [1,2], :]
println("array3d[:, [1,2], :] -> $(temp_result2) ($(summary(temp_result2)))\n")

temp_result2 = array3d[:, [1,2], [1]]
println("array3d[:, [1,2], [1]] -> $(temp_result2) ($(summary(temp_result2)))\n")

temp_result2 = array3d[:, 1, :]
println("array3d[:, 1, :] -> $(temp_result2) ($(summary(temp_result2)))")
temp_result2 = array3d[1, :, :]
println("array3d[1, :, :] -> $(temp_result2) ($(summary(temp_result2)))\n\n")

```

/src/ch09/index2/main.jl

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 二维数组。
array2d = [[1,2,3,4,5] [6,7,8,9,10] [11,12,13,14,15] [16,17,18,19,20]
[21,22,23,24,25] [26,27,28,29,30]]
println("The array named array2d (1): $(array2d) ($(summary(array2d)))\n")

array2d_copy = copy(array2d)
println("The array named array2d_copy (1): $(array2d_copy)\n")

array2d_copy[5] = 50;
println("array2d_copy[5] = 50")
array2d_copy[[1,3]] = [10, 30];
println("array2d_copy[[1,3]] = [10, 30]")
array2d_copy[7:9] = [70, 80, 90];
println("array2d_copy[7:9] = [70, 80, 90]\n")

println("The array named array2d_copy (2): $(array2d_copy)\n")

println("The array named array2d (2): $(array2d)\n\n")

# 三维数组。
array3d = reshape(array2d, (3,5,2))
println("The array named array3d (1): $(array3d) ($(summary(array3d)))\n")

array3d_copy = copy(array3d)
println("The array named array3d_copy (1): $(array3d_copy)\n")

array3d_copy[:, :, 1] = zeros{Int64, 3, 5};
println("array3d_copy[:, :, 1] = zeros{Int64, 3, 5}")
array3d_copy[:, 3:4, 2] = ones{Int64, 3, 2};
println("array3d_copy[:, 3:4, 2] = ones{Int64, 3, 2}")
array3d_copy[:, [1,5], 2] = fill(2, 3, 2)
println("array3d_copy[:, [1,5], 2] = fill(2, 3, 2)\n")

println("The array named array3d_copy (2): $(array3d_copy)\n")

println("The array named array3d (2): $(array3d)\n")

```

/src/ch09/iter/main.jl

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

array2d = [[1,2,3,4,5] [6,7,8,9,10] [11,12,13,14,15] [16,17,18,19,20]
[21,22,23,24,25] [26,27,28,29,30]]
println("The array named array2d: $(array2d) ($(summary(array2d)))\n")

println("Iterate array2d:")
for e in array2d
    print("$e\t")
end
println("\n")

println("Iterate eachindex(array2d):")
for i in eachindex(array2d)
    print("$i: $(array2d[i])\t")
end
println("\n")

println("Iterate pairs(array2d):")
for (i, v) in pairs(array2d)
    print("$i: $(v)\t")
end
println("\n\n")

temp_result = pairs(IndexStyle(array2d), array2d)
println("pairs(IndexStyle(array2d), array2d) -> $(temp_result)
($(summary(array2d)))\n")

temp_result = pairs(IndexCartesian(), array2d)
println("pairs(IndexCartesian(), array2d) -> $(temp_result)
($(summary(array2d)))\n")

```

[/src/ch09/representation/main.jl](#)

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 一般表示法：维数和类型。
temp_array = [1, 2, 3, 4, 5]
println("[1, 2, 3, 4, 5] -> $(temp_array) ($(summary(temp_array)))")
temp_array = [1; 2; 3; 4; 5]
println("[1; 2; 3; 4; 5] -> $(temp_array) ($(summary(temp_array)))")
temp_array = [1 2 3 4 5]
println("[1 2 3 4 5] -> $(temp_array) ($(summary(temp_array)))\n")

temp_array = [[1]; [2,3]; 4; 5]
println("[[1]; [2,3]; 4; 5] -> $(temp_array) ($(summary(temp_array)))")
temp_array = [[1], [2,3], 4, 5]
println("[[1], [2,3], 4, 5] -> $(temp_array) ($(summary(temp_array)))\n")

temp_array = Int8[1, 2, 3, 4, 5]
println("Int8[1, 2, 3, 4, 5] -> $(temp_array) ($(summary(temp_array)))")
temp_array = Int8[[1]; [2,3]; 4; 5]
println("Int8[[1]; [2,3]; 4; 5] -> $(temp_array) ($(summary(temp_array)))\n\n")

# 一般表示法：多维数组。
temp_array = [[1] [2 3] 4 5]
println("[[1] [2 3] 4 5] -> $(temp_array) ($(summary(temp_array)))")
temp_array = [[1;2] [3;4] [5;6]]
println("[[1;2] [3;4] [5;6]] -> $(temp_array) ($(summary(temp_array)))")
temp_array = [[1 2]; [3 4]; [5 6]]
println("[[1 2]; [3 4]; [5 6]] -> $(temp_array) ($(summary(temp_array)))\n")

temp_array = [ [[1.0;2.0] [1.1;2.1]]; [[3.0;4.0] [3.1;4.1]]; [[5.0;6.0] [5.1;6.1]] ]
println("[ [[1.0;2.0] [1.1;2.1]]; [[3.0;4.0] [3.1;4.1]]; [[5.0;6.0] [5.1;6.1]]
] -> ",
        temp_array, " (",
        summary(temp_array), ")\n")

```

/src/ch09/search/main.jl

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 一维数组。
temp_array1 = [1,2,3,4,5,6,7,8,9]
println("The temporary array 1: $(temp_array1)\n")

temp_result1 = findfirst(isequal(7), temp_array1)
println("findfirst(isequal(7), temp_array1) -> $(temp_result1)")
temp_result1 = findfirst(isequal(27), temp_array1)
println("findfirst(isequal(27), temp_array1) == nothing -> $(temp_result1 ==
nothing)\n")

# 二维数组。
array2d = [[1,2,3,4,5] [6,7,8,9,10] [11,12,13,14,15] [16,17,18,19,20]
[21,22,23,24,25] [26,27,28,29,30]]
println("The array named array2d: $(array2d) ($(summary(array2d)))\n")

temp_result2 = findfirst(isequal(27), array2d)
temp_result2_v = array2d[temp_result2]
println("findfirst(isequal(27), array2d) -> $(temp_result2) (value:
$(temp_result2_v))")
temp_result2 = findnext(iseven, array2d, CartesianIndex(2, 6))
temp_result2_v = array2d[temp_result2]
println("findnext(iseven, array2d, CartesianIndex(2, 6)) -> $(temp_result2)
(value: $(temp_result2_v))")
temp_result2 = findprev(iseven, array2d, CartesianIndex(2, 6))
temp_result2_v = array2d[temp_result2]
println("findprev(iseven, array2d, CartesianIndex(2, 6)) -> $(temp_result2)
(value: $(temp_result2_v))\n")

# 布尔值数组。
array2d_bool = Bool[0 0 1 0 0 1; 1 0 1 0 0 0; 0 0 0 1 0 0; 1 0 0 0 1 1; 0 1 0 1
0 0]
println("The array named array2d_bool: $(array2d_bool)
($(summary(array2d_bool)))\n")

temp_result3 = findlast(array2d_bool)
temp_result3_v = array2d_bool[temp_result3]
println("findlast(array2d_bool) -> $(temp_result3) (value: $(temp_result3_v))")
temp_result3 = findprev(array2d_bool, CartesianIndex(3, 6))
temp_result3_v = array2d_bool[temp_result3]
println("findprev(array2d_bool, CartesianIndex(3, 6)) -> $(temp_result3)
(value: $(temp_result3_v))")
temp_result3 = findall(array2d_bool)
temp_result3_v = array2d_bool[temp_result3]
println("findall(array2d_bool) -> $(temp_result3) (value:
$(temp_result3_v))\n\n")

```

最小值和最大值：一维数组。

```
temp_array2 = [115,65,18,2,117,-102,123,66,-93,-102]
println("The temporary array 2: $(temp_array2)")
```

```
temp_array3 = [115,65,18,2,117,-102,123,66,NaN,-102]
println("The temporary array 3: $(temp_array3)\n")
```

```
temp_result4 = findmin(temp_array2)
println("findmin(temp_array2) -> $(findmin(temp_array2))")
```

```
temp_result4 = findmin(temp_array3)
println("findmin(temp_array3) -> $(findmin(temp_array3))\n")
```

最小值和最大值：二维数组。

```
temp_array5 = findmax(array2d, dims=1)
println("findmax(array2d, dims=1) -> $(temp_array5)")
temp_array5 = findmax(array2d, dims=2)
println("findmax(array2d, dims=2) -> $(temp_array5)")
temp_array5 = findmax(array2d, dims=(1,2))
println("findmax(array2d, dims=(1,2)) -> $(temp_array5)\n")
```

/src/ch09/type/main.jl

[to top](#)

```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

# 数组类型的继承关系。
println("Array <: AbstractArray -> $(Array <: AbstractArray)")
println("Array{Float64,3} <: Array{Real,3} -> $(Array{Float64,3} <:
Array{Real,3})\n")

# 数组类型的别名类型。
println("Vector{Int} == Array{Int,1} -> $(Vector{Int} == Array{Int,1})")
println("Matrix{Int} == Array{Int,2} -> $(Matrix{Int} == Array{Int,2})\n")

# 数组长度的最大值。
println("Array{T,typemax(Int64)} where T -> $(Array{T,typemax(Int64)} where
T)\n")

# 数组的类型。
type1 = Array{Int64,1}
println("isa([1], Array{Int64,1}) -> $(isa([1], type1))")
println("isa([1,2,3], Array{Int64,1}) -> $(isa([1,2,3], type1))")
println("isa([1,2,3,4,5], Array{Int64,1}) -> $(isa([1,2,3,4,5], type1))\n")

# 多维数组与数组的嵌套。
println("The type of [[1,2] [3,4]]: $(typeof([[1,2] [3,4]]))")
println("The type of [[1,2], [3,4,5]]: $(typeof([[1,2], [3,4,5]]))\n")

```

[/src/ch09/view/main.jl](#)

[to top](#)


```

# 示例的演示文件。
# - Julia version: 1.2.0
# - Author: HaoLin
# - Date: 2020-01-01

array4d = reshape(Vector{Int}(1:36), (3,3,2,2))
println("The array named array4d: $(array4d) ($(summary(array4d)))\n")

array4d_view1 = view(array4d, 26)
println("The view named array4d_view1: $(array4d_view1)
$(summary(array4d_view1))")
println("The size of array4d_view1: $(size(array4d_view1))")
println("The dimensions of array4d_view1: $(ndims(array4d_view1))")
println("The element type of array4d_view1: $(eltype(array4d_view1))")
println("The unique element value in array4d_view1: $(array4d_view1[])\n")

array4d_view1[] = 260
println("array4d_view1[] = 260")
println("The unique element value in array4d_view1: $(array4d_view1[])")
println("The element value of index 26 in array4d: $(array4d[26])\n")

array4d_view2 = view(array4d, [1,3,5])
println("The view named array4d_view2: $(array4d_view2)
$(summary(array4d_view2))")
println("The element values in array4d_view2: $(array4d_view2[[1, 2, 3]])\n")

array4d_view2[[1,2,3]] = [10, 30, 50]
println("array4d_view2[[1,2,3]] = [10, 30, 50]")
println("The element values in array4d_view2: $(array4d_view2[:])\n")

array4d_view3 = view(array4d, :, 1, 2, 2)
println("The view named array4d_view3: $(array4d_view3)
$(summary(array4d_view3))\n")

array4d_view3[:] = [280, 290, 300]
println("array4d_view3[:] = [280, 290, 300]")
println("The element values in array4d_view3: $(array4d_view3[:])")
println("The first vector of the last 2-dimensional array in array4d:
$(array4d[:, 1, 2, 2])\n")

println("parent(array4d_view3) == array4d -> $(parent(array4d_view3) ==
array4d)\n")

pindeces = parentindices(array4d_view3)
println("parentindices(array4d_view3) -> $(pindeces)")
array4d_view3_a = array4d[CartesianIndices(pindeces)]
println("array4d[CartesianIndices(parentindices(array4d_view3))] -> ",
array4d[CartesianIndices(pindeces)])
println("vec(array4d[CartesianIndices(parentindices(array4d_view3))]) -> ",
vec(array4d_view3_a))
println("array4d[:, 1, 2, 2] -> $(array4d[:, 1, 2, 2])\n")

```

