Julia 超新手教學 III part 1

by 杜岳華

Outline

- introduction
- type declaration
- composite type
- inner constructors
- parametric type
- abstract type
- type hierarchy
- incomplete initialization

Practice

RPG 遊戲

我們來試著實作一個小的 RPG 角色系統吧！（沒梗

應該會有劍士跟法師兩個角色...

Type system

大家手上可以用的型別（Type），大概是`Int64`、`String`、`Float64`、`Bool`...等等。

這些直接用來寫以上的演算法會非常繁複，而且不好閱讀，很難 debug。

我們希望有型別可以代表**劍士（Swordsman）**跟**法師（Wizard）**。

Type declaration

在動態語言中，只有值有型別，變數是沒有的

In [1]:
```
3::Int64
```

Out[1]:
```
3
```

以上程式碼確認了 3 是 Int64 型別，在這邊比較像斷言（assertion）

In [2]:
```
x = 3::Int64
```

Out[2]:
```
3
```

你可以檢查變數的型別為何？

In [3]:
```
typeof(x)
```

Out[3]:
```
Int64
```

Composite type

我們來實作劍士！

In [4]:
```
struct Swordsman
    hp   # 血量
    sp   # 魔力
    str  # 力量
    vit  # 體質
    agi  # 敏捷
    int  # 智力
    dex  # 靈巧
    luk  # 幸運
end
```

p.s. Type 的命名請開頭大寫並以駝峰式命名（camel case）

p.s. 在這邊型別可以綁定在變數上

In [5]:
```
sm = Swordsman(1000, 200, 200, 200, 100, 50, 100, 50)
```

Out[5]:
```
Swordsman(1000, 200, 200, 200, 100, 50, 100, 50)
```

In [6]:
```
sm.hp
```

Out[6]:
```
1000
```

In [7]:
```
sm.hp = sm.hp - 100
```

```
type Swordsman is immutable

Stacktrace:
 [1] setproperty!(::Swordsman, ::Symbol, ::Int64) at ./sysimg.jl:19
 [2] top-level scope at In[7]:1
```

Mutable and immutable type

In [8]:
```
mutable struct Swordsman2
    hp
    sp
    str
    vit
    agi
    int
    dex
    luk
end
```

In [9]:
```
sm2 = Swordsman2(1000, 200, 200, 200, 100, 50, 100, 50)
```

Out[9]:
```
Swordsman2(1000, 200, 200, 200, 100, 50, 100, 50)
```

In [10]:
```
sm2.hp
```

Out[10]:
```
1000
```

In [11]:
```
sm2.hp = sm2.hp - 100
```

Out[11]:
```
900
```

Inner constructors

In [12]:
```
mutable struct Swordsman2
    hp
    sp
    str
    vit
    agi
    int
    dex
    luk
    Swordsman2() = new(1000, 200, 200, 200, 100, 50, 100, 50)
end
```

```
struct Swordsman2
    ...

    function Swordsman2()
        ...
    end
end
```

In [13]:
```
sm2 = Swordsman2()
```

Out[13]:
```
Swordsman2(1000, 200, 200, 200, 100, 50, 100, 50)
```

In [14]:
```
sm2.hp
```

Out[14]:
```
1000
```

Parametric type

參數化型別，讓型別變得像容器，可以容納不同型別，以達到 generic 的效果。

In [15]:
```
zeros(8, 8)
```

Out[15]:
```
8×8 Array{Float64,2}:
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

In [16]:
```
struct Container{T}
    board::Matrix{T}
    Container{T}(m, n) where T = new(zeros(T, m, n))
end
```

In [17]:
```
Container{Int64}(8, 8)
```

Out[17]:
```
Container{Int64}([0 0 ⋯ 0 0; 0 0 ⋯ 0 0; ⋯ ; 0 0 ⋯ 0 0; 0 0 ⋯ 0 0])
```

In [18]:
```
Container{Float64}(8, 8)
```

Out[18]:
```
Container{Float64}([0.0 0.0 ⋯ 0.0 0.0; 0.0 0.0 ⋯ 0.0 0.0; ⋯ ; 0.0 0.0 ⋯ 0.0 0.0; 0.0 0.0
⋯ 0.0 0.0])
```

In [19]:
```
Container{Bool}(8, 8)
```

Out[19]:
```
Container{Bool}(Bool[false false ⋯ false false; false false ⋯ false false; ⋯ ; false false
⋯ false false; false false ⋯ false false])
```

Methods

現在來實作攻擊跟補血兩個技能：

- 攻擊是能夠對對方造成傷害
- 補血是讓自己的 HP 增加

In [6]:

```
mutable struct Swordsman3
    hp::Int64
    sp::Int64
    str::Int64
    vit::Int64
    agi::Int64
    int::Int64
    dex::Int64
    luk::Int64
    Swordsman3() = new(1000, 200, 200, 200, 100, 50, 100, 50)
end
```

In [7]:
```
attack!(a::Swordsman3, b::Swordsman3) = (b.hp -= (0.8*a.str - 0.6*b.vit))
```

Out[7]:
```
attack! (generic function with 1 method)
```

In [8]:
```
heal!(a::Swordsman3, hp::Integer) = (a.hp += hp)
```

Out[8]:
```
heal! (generic function with 1 method)
```

In [9]:
```
sm = Swordsman3()
```

Out[9]:
```
Swordsman3(1000, 200, 200, 200, 100, 50, 100, 50)
```

In [10]:
```
sm2 = Swordsman3()
```

Out[10]:
```
Swordsman3(1000, 200, 200, 200, 100, 50, 100, 50)
```

In [11]:
```
attack!(sm, sm2)
```

Out[11]:
```
960.0
```

In [12]:
```
sm2
```

Out[12]:
```
Swordsman3(960, 200, 200, 200, 100, 50, 100, 50)
```

Abstract type

In [13]:
```
abstract type Role end

mutable struct Swordsman <: Role
    hp::Int64
    sp::Int64
    str::Int64
    vit::Int64
    agi::Int64
    int::Int64
    dex::Int64
    luk::Int64
    Swordsman() = new(1000, 200, 200, 200, 100, 50, 100, 50)
end
```

In [14]:
```
mutable struct Wizard <: Role
    hp::Int64
    sp::Int64
    str::Int64
    vit::Int64
    agi::Int64
    int::Int64
    dex::Int64
    luk::Int64
    Wizard() = new(500, 1000, 50, 50, 100, 200, 200, 100)
end
```

這時我們會說：Swordsman 是 Role 的子型別，或是 Swordsman 是一種Role

In [15]:
```
attack!(a::Role, b::Role) = (b.hp -= (0.8*a.str - 0.6*b.vit))
```

Out[15]:
```
attack! (generic function with 2 methods)
```

所有 Role 的子型別都可以用 attack! 這個方法。

In [16]:
```
剣士 = Swordsman()
```

Out[16]:
```
Swordsman(1000, 200, 200, 200, 100, 50, 100, 50)
```

In [17]:
```
法師 = Wizard()
```

Out[17]:
```
Wizard(500, 1000, 50, 50, 100, 200, 200, 100)
```

In [18]:
```
attack!(剣士, 法師)
```

Out[18]:
```
370.0
```

In [19]:
```
attack!(法師, 剣士)  # !?
```

Out[19]:
```
1080.0
```

Type hierarchy

這時 Swordsman 跟 Wizard 稱為 concrete type（具體型別），Role 則是 abstract type（抽象型別）。

concrete types 可以被 **實體化（instantiation）**，abstract type 則不行。

In [20]:
```
Role()
```

```
MethodError: no constructors have been defined for Role

Stacktrace:
 [1] top-level scope at In[20]:1
```

所有型別都有父型別。

In [21]:
```
supertype(Swordsman)
```

Out[21]:
```
Role
```

In [22]:
```
supertype(Role)
```

Out[22]:
```
Any
```

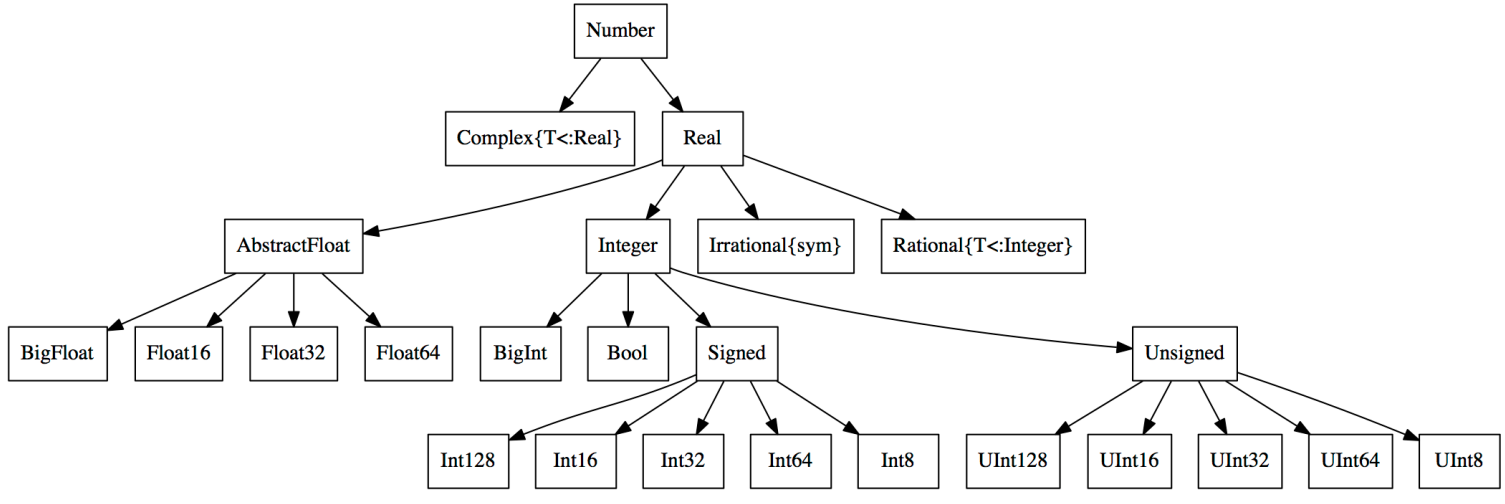
比較特別的是，**concrete type（具體型別）不會有子型別**，也就是concrete type不能被繼承，所以所有concrete types是位於繼承樹的最末端。

In [23]:
```
struct Swordsman2 <: Swordsman
end
```

```
invalid subtyping in definition of Swordsman2

Stacktrace:
 [1] top-level scope at none:0
```

Declared type



Incomplete initialization

In [24]:
```
mutable struct SelfReferential
    obj::SelfReferential
end
```

In [25]:
```
sr = SelfReferential()
```

```
MethodError: no method matching SelfReferential()
Closest candidates are:
  SelfReferential(!Matched::SelfReferential) at In[24]:2
  SelfReferential(!Matched::Any) at In[24]:2

Stacktrace:
 [1] top-level scope at In[25]:1
```

In [26]:
```
mutable struct SelfReferential2
    obj::SelfReferential2
    SelfReferential2() = (x = new(); x.obj = x)
end
```

In [27]:
```
sr2 = SelfReferential2()
```

Out[27]:
```
SelfReferential2(SelfReferential2(#= circular reference @-1 =#))
```

In [28]:
```
sr2 === sr2.obj
```

Out[28]:
```
true
```

In [29]:
```
sr2 === sr2.obj.obj
```

Out[29]:
```
true
```

Q & A