<< Scilab的多型態資料結構 >> 作者: pipidog

這裡要解釋的是,如何在scilab中,利用一個變數儲存多型態的資料. 如將字串與數值存入同一個變數中,又或者將大小不同的資料陣列存入同一個變數中. Scilab 中常用的資料結構有五種:

list, cell, tlist, mlist, struct,
這五種資料結構中,我們可以依據他們的特性區分成有field以及沒有field.其中list 以及cell都是屬於沒有field的結構.而tlist,mlist以及struct則是屬於有field的結構.以下我們分別說明. 他們各自適合不同類型的問題. 下面我們分別做說明.

<< list (串列) >>
宣告變數a的資料結構是一個list:
→ a=list()
給定a資料
→ a(1)=ones(3,3) ; a(2)=eye(4,4); a(3)='Scilab'
要提出a (1)裡面的object中的matrix element :
→ a(1)(1,2)
ans =   1.
要知道a的大小:
→ size(a)
ans =   3.
注意,這裡會給出的是a的list大小,至於a(1)裡面的物件的大小必須要用:
→ size(a(1))
ans = 3   3
list可以透過index來插入資料於頭尾
→ a(0)='test' (則在a的頭插入資料'test'),
→ a($+1)='test' (則在a的尾插入資料'test')

若要清除某一元素的資料
→ a(1)=null()

但是常常我們需要處理到更複雜的資料結構,需要再list裡面再儲存list, 這時候我們常常需要增加list的維度. 但跟陣列資料可以直接宣告一個高維度的陣列不同, list並沒有辦法這樣做. 我們必須要針對每一個list a的每個元素,指定它也是一個list才行. 例如a(1)原本是ones(3,3),現在我們要讓他也變成一個list, 我們可以用:
→ a(1)=list()
則可以開始添加a(1)(1)=ones(3,3) , a(1)(2)= ones(4,4), …
但這個時候,並不存在a(2)(1)=ones(3,3) , a(2)(2)=ones(4,4) 的用法,原因是因為a(2)並不是一個list. 想要使用a(2)(1), a(2)(2),我們同樣的必須宣告a(2)=list()

從上面看見,list雖然可以簡單的儲存多樣性的資料,但是要將list的維度提高比較麻煩. 因此如果我們只是單純的想如同一個陣列一般的使用一個變數,但是變數中的每個陣列元素可以允許儲存多樣性的資料,其實有更好的選擇: 細胞陣列.

<< cell (細胞陣列) >>
cell 它能夠直接宣告不同維度,大小的矩陣讓使用者針對不同的矩陣元素放入不同的物件.

宣告一個 3x5的cell array
→ a=cell(3,5)
*注意, cell(3)的意思並不是cell(3,1) 而是cell(3,3)

同理,我們可以宣告cell為一個高維陣列
→ a=cell(3,5,7)

要存入資料,必須使用field name “entries”,這是由於cell array本身是一種mlist(稍後解釋),因此field name是必須的,另外也用來區別cell array與一般的數值矩陣. 這也是使用cell array比list方便的原因,因為list的表示往往容易跟一般的數值矩陣產生混淆,但是cell則避免了這樣的問題.

→ a(1,1).entries=ones(3,3)

要呼叫特定元素中的物件
→ a(1,1).entries(3,3)

上面的說明看似cell array比list更適合用來儲存多型態的資料. 但cell array仍有其限制. 其限制在於cell array是一個可以儲存多型態資料的矩陣. 因此它的指標,大小仍必須滿足矩陣的條件,這點在高維array時會有所差別. 例如list可以接受 a(1)存在高維度a(1)(1), a(1)(2), 但是對於a(2)則不具高維度. 就算同樣是高維度,list可允許a(1)和a(2)底下有不同數目的list. 例如a(1)存在a(1)(1),a(1)(2),但是對於a(2)則只存在a(2)(1)而沒有a(2)(2)的情形. 可是對cell array,一但我們宣告了a=cell(2,2),就無法讓a(2,2)這個元素不存在.
因此cell array雖然在使用上更直觀,但在儲存多型態的資料的自由度上則遠不如list方便. 所以我們可以說list可以完全取代cell,但是cell則無法完全取代list. 不過在資料結構不太複雜的情況下,用cell來解決問題仍是一個很好的選擇,但是熟悉list的使用仍是處理資料結構不能或缺的工具.

// ========================
上面我們說明了list 跟cell的使用,他們主要的目的都是為了"儲存”多型態的變數”. 但是在物件導向與資料型態的觀念中,我們不能不處理”多載 (overloading)”的問題. 所謂多載指的是,使用者要自訂一種資料型態,這個型態的資料會伴隨著一連串的”操作”. 當系統遇到不同型態的資料時,會根據它的型態來決定該怎麼進行這個操作.
例如有兩個變數a, b, 當我宣告a=2, b=3 的時候, a+b 就是把這兩個數字相加變成5. 但是當我宣告a=ones(2,2), b=3的時候,a+b 就變成在計算[2,2 ; 2,2] + [3,3 ; 3,3], 而當 a=’Hello,’, b=’ John!’時, a+b就成了’Hello, John!’. 也就是說, 同樣是一個加號” + “,但是這個操作遇到常數,矩陣,跟字串時,系統會自動根據資料的型態產生常數相加,矩陣相加,字串相加的動作. 這種同一個操作遇到不同型態會產生不同動作的過程, 在物件導向的語言中就叫做”多載”.
當我們在使用Scilab時,為了讓程式更容易維護,功能更多樣性,有時候會需要引入多載的功能,讓自己定義屬於自己的資料型態. 例如我可能會希望定義一種資料型態叫做”Person”, Person底下

儲存了一個人的名字,身高,體重等等. 那麼當我把兩個Person變數,例如Mary跟John相加 Son=Mary+John,會產生一個新的Person,叫做Son,Son裡面同樣也儲存了Son這個人的身高,體重. 而身高體重的數值可能是我透過一種特殊的遺傳預測方式,根據John和Mary的資料得出的預測身高跟體重. 因此我們可以簡單的把複雜的遺傳演化問題,變成一堆變數相加的問題. 例如 Son1=Mary+John , Son2=Amy+Kevin, 那麼Mary和John的孩子Son1跟Amy和Kevin的孩子Son2如果結婚生子,它們生出的後代的特性就可以簡單地透過Grandson=Son1+Son2得出.這就是物件導向的重要應用. 關於多載,介紹的篇幅所需不少,也許有機會再介紹,有需要的人可以透過scilab中的help查詢"overload"來取得相關介紹. 下面我們僅針對Scilab中,為了這個多載而定義出了另外三類的結構,tlist, mlist, struct來逐一介紹.

// ==========

tlist, mlist, struct這三類結構都有一個共同的特徵,就是它們的變數都存在field,所謂field也就是在一的變數名底下,我們可以存在子名. 例如變數Mary, 可以存在Mary.name, Mary.height, Mary.weight分別儲存Mary的名字,身高,體重. 事實上,它們可以儲存的資料並沒有特殊限制. 例如Mary.height可以是一個矩陣,一個字串,甚至也可以是一個list.

<< tlist, mlist >>
tlist是typed-list (型態串列), mlist是matrix-list(矩陣串列),它們的使用方式如下:

```
-->M=tlist(['person', 'name', 'value'],['Mary'],[175, 65])
 M  =


       M(1)

!person  name  value  !

       M(2)

 Mary

       M(3)

    175.    65.
```

上面的指令定義了一種資料型態叫"person",在person這個型態底下,有兩個子描述,一個叫name,一個叫做value,而後把後面緊接著的資料設給前面的子描述. 所以會有M.name='Mary'以及M.value=[175,65]的結果. 至於person只是對於這種型態資料的一個名稱,如同矩陣叫matrix,常數叫constant一樣. 這種型態我們叫person. 在實際的應用上,這個名稱並不太重要,他只是幫助我們認識這個型態.事實上宣告型態的當下就給定資料不是必需的. 我們可以只定義第一個字串矩陣即可. 日後如果要給定資料,只需要使用 M.name='Mary'; M.value=[1.75,65]就可以達到同樣效果.

同樣的,對於mlist結構,我們也有:

```
-->M=tlist(['person', 'name', 'value'],['Mary'],[175, 65])
 M  =


       M(1)

!person  name  value  !

       M(2)

 Mary

       M(3)

    175.    65.
```

看到這裡,相信許多人應該一頭霧水了. 這跟之前的tlist不是完全一樣嗎? 事實上它們還是有點不太一樣. 如果M是宣告成tlist,那麼當我們打M(1)時會出現:

```
-->M(1)
 ans  =

!person  name  value  !
```

可是如果M(1)是宣告成mlist,那麼當我們打M(1)則會出現未定義的錯誤:

```
-->M(1)
     !--error 144
Undefined operation for the given operands.
check or define function %l_e for overloading.
```

這是什麼意思呢? 前面我們說到,mlist跟tlist的設計是為了要讓使用者自訂資料型態,進行多載的. 那些原本系統內建的操作,例如加法,乘法,輸入一個變數後,系統會如何顯示這個變數的資訊,都是可以重定義的. 事實上還不只上面這些,包括你要如何提取出一筆資料的方式都可以被重定義. 例如一個矩陣,我們要提取他的某一維度的所有資料,常常會用這個語法:
a(:,1),意思是我們要提取出變數a的第一行的所有資料. 而這個提取的方式"(:,1)"甚至是更簡單的a(1,2)裡面的"(1,2)"事實上也是可以被重定義的.tlist跟mlist最大的不同是,tlist仍使用系統預設的資料提取方式,而mlist則是讓使用者連提取方式都可以自訂了. 因此mlist可以說是最基本的一種資料結構的定義.幾乎不存在任何限制. 那既然如此,又為什麼需要tlist呢?原因很簡單,因為使用者不見得希望對於資料型態做出如此徹底的重定義.很多

時候我們僅僅是想重新定義資料型態的加法,乘法的． 我們並不希望對一個資料的一切都完全重定義,過高的自由度往往只會浪費使用者不必要的時間．

<< strcut (結構變數) >>
事實上把struct和tlist,mlist放在一起講是不對的.它應該和list與cell放在一起講. 因為他們都是用來儲存變數而不是用來處理多載的,但是因為他和tlist,mlist常常讓使用者混淆,所以我決定把它放在最後,在使用者了解了tlist與,mlist之後再講,方便比較.
struct跟mlist,和tlist一樣,都是具有field name的結構. 但是和tlist, mlist不同,struct並沒有使用多載的能力,事實上他只是一種基於mlist定義出來的資料型態 (cell array也是). 它可以讓使用者針對一個變數的不同需求定義子變數名,如此而已. 有人問,那既然已經有tlist跟mlist可以做同樣的事情了,我們又為什麼需要struct呢? 同樣的理由,方便. 因為很多時候我們只是要把變數中的資料進行分類,並不想要重定義多載. struct讓我們可以可以繼續使用系統內建的那些操作的定義,只單純的使用子變數名的功能. 因此在不需要使用多載的情況下,使用struct是最好的選擇.
使用struct,最簡單的就是直接當一般變數使用. 使用struct
student1.name='Mary'
student1.value=[175,65]
另外也可以一口氣給定field name與資料
student1=struct('name','Mary','value',[1.75,65])
這樣做可以讓我們把很多原本零散的變數整合到同一個變數名底下,我們只需要student1這個變數,就可以同時擁有底下的name, value等field的值. 這對寫程式的變數取名很有幫助.

要注意的是,struct跟mlist, tlist很大的不同是,struct可以動態的增加field name
例如我可以直接宣告student1.sex='female'.

而mlist與tlist則是在宣告後就無法再增加field name了. 例如我們透過mlist與tlist宣告出一種型態叫person,如果再宣告之時沒有sex這項屬性,日後也無法再添加了.
這說明struct本質上是一種儲存變數的方式,而mlist, tlist則是用來定義一種"資料型態"!

另外要特別一提的是,struct跟cell一樣,都不是scilab核心內建的基本資料結構.它們都是一種基於mlist的"應用". 事實上不只如此,就連我們常在使用的高維陣列 (3個以上指標的陣列)都是一種mlist的應用. 雖然使用上感覺不出什麼差別,但是對於了解scilab的資料結構是必須知道的.

// ==========
總結一下我們所介紹的資料結構: list, cell, struct, mlist, tlist. 前面三種都是一種儲存多型態資料的方式, list跟cell都不具有field name的功能, 在使用上,list的自由度最高,但若要建構高維度的list較麻煩. cell很簡單,很直觀,使用上也能夠清楚的和一般的數值陣列做出區別.但自由度不如list高. 如果有能力還是應該盡量掌握使用list的能力. cell 則適合較簡單的資料結構使用. 而struct雖然和mlist與tlist一樣有field name,但本質上它的功用還是僅限於儲存資料. 但是field name可以幫助我們更容易的定義變數以及使用變數,可以多加使用.

tlist和mlist則是為了多載而設計出的結構.兩者的差別在於mlist提供了重定義資料提取的空間,而tlist則使用系統預設的. 對於沒有多載需求的情況下,我們能夠徹底掌握list與struct也就夠了.