Fondamenti di Informatica (Elettronici)

THINK JULIA – Capitolo 10 (seconda parte)

16 novembre 2020

10. Array (seconda parte)[1]

1. Sintassi dot
2. Eliminazione (inserimento) di elementi
3. Array e stringhe
4. Oggetti e valori
5. Aliasing
6. Argomenti array
7. Debug
8. Glossario
9. Esercizi

[1] Tratto da https://benlauwens.github.io/ThinkJulia.jl/latest/book.html, disponibile sotto Licenza 'Creative Commons Attribution-NonCommercial 3.0 Unported'.

Section 1

Sintassi dot

Sintassi dot per creare una map

Per ogni operatore binario come ^, esiste un corrispondente operatore punto .^ che viene automaticamente definito per eseguire ^ elemento per elemento sugli array.

Ad esempio, "[1, 2, 3]^3" non è definito, ma "[1, 2, 3].^3" è definito come il calcolo del risultato elementwise "[1^3, 2^3, 3^3]":

```
julia> print([1, 2, 3] .^ 3)
[1, 8, 27]
```

Qualsiasi funzione Julia "f" può essere applicata elementwise a qualsiasi array con la sintassi "dot". Ad esempio, per rendere maiuscolo un array di stringhe, non abbiamo bisogno di un ciclo esplicito:

```
julia> t = uppercase.(["abc", "def", "ghi"]);
julia> print(t)
["ABC", "DEF", "GHI"]
```

Questo è un modo elegante per creare una map. La funzione capitalizeall può essere implementata da una sola riga:

```
capitalizeall(t) = uppercase.(t)
```

Section 2

Eliminazione (inserimento) di elementi

Funzioni `splice!` e `pop!`

Esistono diversi modi per eliminare elementi da un array. Se si conosce l'indice dell'elemento che si desidera, si può usare `splice!` :

```
julia> t = ['a', 'b', 'c'];
julia> splice!(t, 2)
'b': ASCII/Unicode U+0062 (category Ll: Letter, lowercase)
julia> print(t)
['a', 'c']
```

- `splice!` modifica l'array e restituisce l'elemento che è stato rimosso.

```
julia> t = ['a', 'b', 'c'];
julia> pop!(t)
'c': ASCII/Unicode U+0063 (category Ll: Letter, lowercase) julia>
print(t)
['a', 'b']
```

Funzioni `splice!` e `pop!`

Esistono diversi modi per eliminare elementi da un array. Se si conosce l'indice dell'elemento che si desidera, si può usare `splice!` :

```
julia> t = ['a', 'b', 'c'];
julia> splice!(t, 2)
'b': ASCII/Unicode U+0062 (category Ll: Letter, lowercase)
julia> print(t)
['a', 'c']
```

- `splice!` modifica l'array e restituisce l'elemento che è stato rimosso.
- `pop!` cancella e restituisce l'ultimo elemento:

```
julia> t = ['a', 'b', 'c'];
julia> pop!(t)
'c': ASCII/Unicode U+0063 (category Ll: Letter, lowercase) julia>
print(t)
['a', 'b']
```

popfirst! and pushfirst! functions

- popfirst! cancella e restituisce il primo elemento:

```
julia> t = ['a', 'b', 'c'];
julia> popfirst!(t)
'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase)
julia> print(t)
['b', 'c']
```

popfirst! and pushfirst! functions

- popfirst! cancella e restituisce il primo elemento:

```
julia> t = ['a', 'b', 'c'];
julia> popfirst!(t)
'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase)
julia> print(t)
['b', 'c']
```

- Le funzioni pushfirst! E push! inseriscono un elemento all'inizio, e rispettivamente alla fine dell'array.

Le funzioni deleteat! e insert!

- Se non hai bisogno del valore rimosso, puoi utilizzare la funzione deleteat!:

```
julia> t = ['a', 'b', 'c'];
julia> print( deleteat!(t, 2) )
['a', 'c']
```

```
julia> t = ['a', 'b', 'c'];
julia> print(insert!(t, 2, 'x'))
['a', 'x', 'b', 'c']
```

Le funzioni deleteat! e insert!

- Se non hai bisogno del valore rimosso, puoi utilizzare la funzione deleteat!:

```
julia> t = ['a', 'b', 'c'];
julia> print( deleteat!(t, 2) )
['a', 'c']
```

- La funzione insert! Inserisce un elemento in un dato indice:

```
julia> t = ['a', 'b', 'c'];
julia> print(insert!(t, 2, 'x'))
['a', 'x', 'b', 'c']
```

Section 3

Array e stringhe

collect function

Una stringa è una sequenza di caratteri e un array è una sequenza di valori, ma un array di caratteri non è la stessa cosa di una stringa.

- Per convertire da una stringa a un array di caratteri, puoi usare la funzione `collect`:

```
julia> t = collect("spam");
julia> print(t)
['s', 'p', 'a', 'm']
```

La funzione `collect` spezza una stringa o un'altra sequenza in singoli elementi (crea un array).

Funzione "split"

Se vuoi dividere una stringa in parole, puoi usare la funzione `split`:

```
julia> t = split("pining for the fjords");
julia> print(t)
SubString{String}["pining", "for", "the", "fjords"]
```

Un argomento facoltativo chiamato delimitatore specifica quali caratteri utilizzare come confini delle parole.

L'esempio seguente utilizza un trattino (dash) come delimitatore:

```
julia> t = split("spam-spam-spam", '-');
julia> print(t)
SubString{String}["spam", "spam", "spam"]
```

Funzione `join`

"join" è l'inverso di "split".

Prende un array di stringhe e concatena gli elementi:

```
julia> t = ["pining", "for", "the", "fjords"];
julia> s = join(t, ' ')
"pining for the fjords"
```

In questo caso il delimitatore è un carattere space. Per concatenare stringhe senza spazi, non specificare un delimitatore.

Section 4

Oggetti e valori

Oggetti e valori 1/3

Un oggetto è qualcosa a cui una variabile può fare riferimento. Fino ad ora, era possibile utilizzare "oggetto" e "valore" in modo intercambiabile.

Se eseguiamo queste istruzioni di assegnazione:

```
a = "banana"
b = "banana"
```

Sappiamo che `a` e `b` si riferiscono entrambi a una stringa, ma non sappiamo se si riferiscono alla stessa stringa. Ci sono due possibili stati, mostrati nella Figura 10-2.

Figure 12. State diagrams.

In un caso, `a` e `b` si riferiscono a due oggetti diversi che hanno lo stesso valore.

Nel secondo caso si riferiscono allo stesso oggetto.

Oggetti e valori 2/3

Per controllare se due variabili si riferiscono allo stesso oggetto, puoi usare l'operatore ≡ (`\equiv TAB`) o `===`.

```
julia> a = "banana"
"banana"
julia> b = "banana"
"banana"
julia> a (`\equiv TAB`) b
true
```

In questo esempio, Julia ha creato un solo oggetto stringa e sia `a` che `b` si riferiscono ad esso. Ma quando crei due array, ottieni due oggetti:

```
julia> a = [1, 2, 3];
julia> b = [1, 2, 3];
julia> a (`\equiv TAB`) b
false
```

Quindi il diagramma di stato è simile al seguente.

Oggetti e valori 3/3

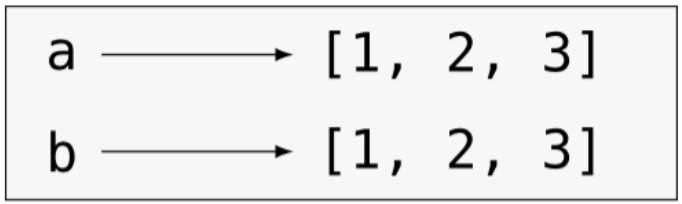

Figure 13. State diagram

In questo caso diremmo che i due array sono equivalenti, perché hanno gli stessi elementi, ma non identici, perché non sono lo stesso oggetto.

Se due oggetti sono identici, sono anche equivalenti, ma se sono equivalenti non sono necessariamente identici.

Per essere precisi, un oggetto ha un valore. Se valuti "[1, 2, 3]", ottieni un oggetto array il cui valore è una sequenza di numeri interi.

Se un altro array ha gli stessi elementi, diciamo che ha lo stesso valore, ma non è lo stesso oggetto.

Section 5

Aliasing

Due riferimenti allo stesso oggetto

Se a si riferisce a un oggetto e si assegna `b = a`, entrambe le variabili si riferiscono allo stesso oggetto:

```
julia> a = [1, 2, 3];
julia> b = a;
julia> b (\equiv TAB) a
true
```

Il diagramma di stato ha il seguente aspetto:

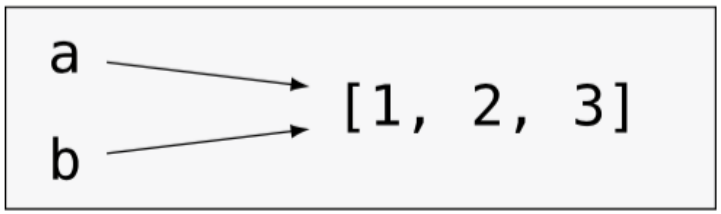

Figure 14. State diagram

Se l'oggetto con alias è modificabile

L'associazione di una variabile con un oggetto è chiamata riferimento. In questo esempio, ci sono due riferimenti allo stesso oggetto.

Un oggetto con più di un riferimento ha più di un nome, quindi diciamo che l'oggetto è alias. Se l'oggetto con alias è modificabile, le modifiche apportate con un alias influiscono sull'altro:

```
julia> b[1] = 42
42
julia> print(a)
[42, 2, 3]
```

Evita di creare alias con oggetti modificabili

WARNING

Sebbene questo comportamento possa essere utile, è soggetto a errori. In generale, è più sicuro evitare l'aliasing quando si lavora con oggetti modificabili.

Per gli oggetti immutabili come le stringhe, l'aliasing non è un grosso problema:

```
a = "banana"
b = "banana"
```

Non fa quasi mai differenza se `a` e `b` si riferiscono o meno alla stessa stringa.

Section 6

Argomenti array

I parametri della funzione ottengono riferimenti al loro argomento

Quando si passa un array a una funzione, la funzione ottiene un riferimento all'array.

Se la funzione modifica l'array, il chiamante vede la modifica. Ad esempio, `deletehead!` Rimuove il primo elemento da un array:

```
function deletehead!(t)
    popfirst!(t)
end
```

Here's how it is used:

```
julia> letters = ['a', 'b', 'c'];
julia> deletehead!(letters);
julia> print(letters)
['b', 'c']
```

Parametro e argomento sono alias

Il parametro `t` e la variabile `letters` sono alias per lo stesso oggetto. Il diagramma di stato è simile al diagramma di stack:

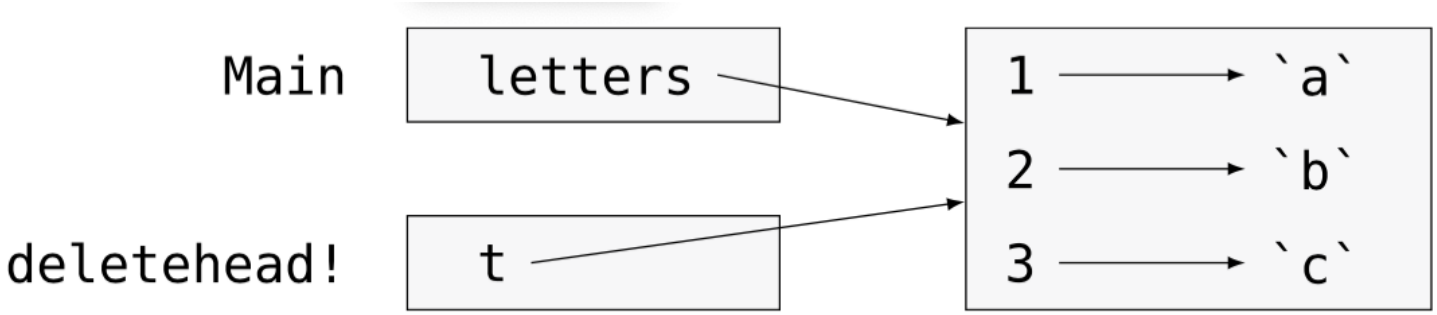

Figure 15. Stack diagram

Poiché l'array è condiviso da due frame, l'ho disegnato tra di loro. È importante distinguere tra operazioni che modificano array e operazioni che creano nuovi array.

Funzione che modifica il parametro di input

Ad esempio, `push!` modifica un array, ma `vcat` crea un nuovo array. Ecco un esempio che utilizza `push!`:

```
julia> t1 = [1, 2];
julia> t2 = push!(t1, 3);
julia> print(t1)
[1, 2, 3]
```

`t2` è un alias di `t1`.

Ecco un esempio che utilizza `vcat`:

```
julia> t3 = vcat(t1, [4]);
julia> print(t1)
[1, 2, 3]
julia> print(t3)
[1, 2, 3, 4]
```

Fai attenzione agli alias

Il risultato di `vcat` è un nuovo array e l'array originale non è cambiato.

Questa differenza è importante quando si scrivono funzioni che dovrebbero modificare gli array. Ad esempio, questa funzione non elimina l'intestazione di un array:

```
function baddeletehead(t)
    t = t[2:end] # WRONG!
end
```

L'operatore `slice` crea un nuovo array e l'assegnazione fa riferimento a `t`, ma questo non influisce sul chiamante.

```
julia> t4 = baddeletehead(t3);
julia> print(t3)
[1, 2, 3, 4]
julia> print(t4)
[2, 3, 4]
```

Preferisco lasciare l'originale non modificato

All'inizio di `baddeletehead`, `t` e `t3` si riferiscono allo stesso array. Alla fine, `t` si riferisce a un nuovo array, ma `t3` si riferisce ancora all'array originale non modificato.

Un'alternativa è scrivere una funzione che crei e restituisca un nuovo array. Ad esempio, `tail` restituisce tutto tranne il primo elemento di un array:

```
function tail(t)
    t[2:end]
end
```

Questa funzione lascia l'array originale non modificato. Ecco come viene °:

```
julia> letters = ['a', 'b', 'c'];
julia> rest = tail(letters);
julia> print(rest)
['b', 'c']
```

Section 7

Debug

La maggior parte delle funzioni su array modifica l'argomento

L'uso incauto di array (e altri oggetti modificabili) può portare a lunghe ore di debug. Ecco alcune trappole comuni e modi per evitarle:

- Questo è l'opposto delle funzioni stringa, che restituiscono una nuova stringa e lasciano l'originale da solo.

La maggior parte delle funzioni su array modifica l'argomento

L'uso incauto di array (e altri oggetti modificabili) può portare a lunghe ore di debug. Ecco alcune trappole comuni e modi per evitarle:

- Questo è l'opposto delle funzioni stringa, che restituiscono una nuova stringa e lasciano l'originale da solo.
- Se sei abituato a scrivere codice stringa come questo:

La maggior parte delle funzioni su array modifica l'argomento

L'uso incauto di array (e altri oggetti modificabili) può portare a lunghe ore di debug. Ecco alcune trappole comuni e modi per evitarle:

- Questo è l'opposto delle funzioni stringa, che restituiscono una nuova stringa e lasciano l'originale da solo.
- Se sei abituato a scrivere codice stringa come questo:

La maggior parte delle funzioni su array modifica l'argomento

L'uso incauto di array (e altri oggetti modificabili) può portare a lunghe ore di debug. Ecco alcune trappole comuni e modi per evitarle:

- Questo è l'opposto delle funzioni stringa, che restituiscono una nuova stringa e lasciano l'originale da solo.
- Se sei abituato a scrivere codice stringa come questo:

```
new_word = strip(word)
```

- Si è tentati di scrivere codice array come questo:

La maggior parte delle funzioni su array modifica l'argomento

L'uso incauto di array (e altri oggetti modificabili) può portare a lunghe ore di debug. Ecco alcune trappole comuni e modi per evitarle:

- Questo è l'opposto delle funzioni stringa, che restituiscono una nuova stringa e lasciano l'originale da solo.
- Se sei abituato a scrivere codice stringa come questo:

```
new_word = strip(word)
```

- Si è tentati di scrivere codice array come questo:

La maggior parte delle funzioni su array modifica l'argomento

L'uso incauto di array (e altri oggetti modificabili) può portare a lunghe ore di debug. Ecco alcune trappole comuni e modi per evitarle:

- Questo è l'opposto delle funzioni stringa, che restituiscono una nuova stringa e lasciano l'originale da solo.
- Se sei abituato a scrivere codice stringa come questo:

```
new_word = strip(word)
```

- Si è tentati di scrivere codice array come questo:

```
t2 = sort!(t1)
```

Perché `sort!` restituisce l'array originale modificato `t1`, `t2` è un alias di `t1`.

ATTENZIONE

Prima di utilizzare le funzioni e gli operatori di array, è necessario leggere attentamente la documentazione e quindi testarli in modalità interattiva.

Scegli un idioma e seguilo

Parte del problema con gli array è che ci sono troppi modi per fare le cose. Ad esempio, per rimuovere un elemento da un array, puoi usare `pop!`, `Popfirst!`, `Delete_at`, o anche un'assegnazione di slice.

Per aggiungere un elemento, puoi usare `push!`, `Pushfirst!`, `Insert!` O `vcat`. Supponendo che `t` sia un array e `x` sia un elemento dell'array, questi sono corretti:

```
insert!(t, 4, x)
push!(t, x)
append!(t, [x])
```

E questi sono sbagliati:

```
insert!(t, 4, [x])
push!(t, [x])
vcat(t, [x])
# WRONG!
# WRONG!
# WRONG!
```

Fare copie per evitare lo aliasing.

Se vuoi usare una funzione come `sort!` che modifica l'argomento, ma devi mantenere anche l'array originale, puoi fare una copia:

```
julia> t = [3, 1, 2];
julia> t2 = t[:]; # t2 = copy(t)
julia> sort!(t2);
julia> print(t)
[3, 1, 2]
julia> print(t2)
[1, 2, 3]
```

In questo esempio potresti anche usare la funzione predefinita `sort`, che restituisce un nuovo array ordinato e lascia integro l'originale:

```
julia> t2 = sort(t);
julia> println(t)
[3, 1, 2]
julia> println(t2)
[1, 2, 3]
```

Section 8

Glossario

Glossario I

array Una sequenza di valori.

Glossario I

array Una sequenza di valori.

elemento Uno dei valori in un array (o un'altra sequenza), chiamato anche item.

Glossario I

array Una sequenza di valori.

elemento Uno dei valori in un array (o un'altra sequenza), chiamato anche item.

array annidato Un array che è un elemento di un altro array.

Glossario I

array Una sequenza di valori.

elemento Uno dei valori in un array (o un'altra sequenza), chiamato anche item.

array annidato Un array che è un elemento di un altro array.

accumulatore Una variabile utilizzata in un ciclo per sommare o accumulare un risultato.

Glossario I

array Una sequenza di valori.

elemento Uno dei valori in un array (o un'altra sequenza), chiamato anche item.

array annidato Un array che è un elemento di un altro array.

accumulatore Una variabile utilizzata in un ciclo per sommare o accumulare un risultato.

assegnazione aumentata Un'istruzione che aggiorna il valore di una variabile utilizzando un operatore come "=."

Glossario I

array Una sequenza di valori.

elemento Uno dei valori in un array (o un'altra sequenza), chiamato anche item.

array annidato Un array che è un elemento di un altro array.

accumulatore Una variabile utilizzata in un ciclo per sommare o accumulare un risultato.

assegnazione aumentata Un'istruzione che aggiorna il valore di una variabile utilizzando un operatore come "=."

Operatore "dot" Operatore binario applicato elemento per elemento agli array.

Glossario I

array Una sequenza di valori.

elemento Uno dei valori in un array (o un'altra sequenza), chiamato anche item.

array annidato Un array che è un elemento di un altro array.

accumulatore Una variabile utilizzata in un ciclo per sommare o accumulare un risultato.

assegnazione aumentata Un'istruzione che aggiorna il valore di una variabile utilizzando un operatore come "=."

Operatore "dot" Operatore binario applicato elemento per elemento agli array.

sintassi "dot" Sintassi utilizzata per applicare una funzione elementwise a qualsiasi array.

Glossario I

array Una sequenza di valori.

elemento Uno dei valori in un array (o un'altra sequenza), chiamato anche item.

array annidato Un array che è un elemento di un altro array.

accumulatore Una variabile utilizzata in un ciclo per sommare o accumulare un risultato.

assegnazione aumentata Un'istruzione che aggiorna il valore di una variabile utilizzando un operatore come "=."

Operatore "dot" Operatore binario applicato elemento per elemento agli array.

sintassi "dot" Sintassi utilizzata per applicare una funzione elementwise a qualsiasi array.

operazione di riduzione Uno schema di elaborazione che attraversa una sequenza e accumula gli elementi in un unico valore risultato.

Glossario I

array Una sequenza di valori.

elemento Uno dei valori in un array (o un'altra sequenza), chiamato anche item.

array annidato Un array che è un elemento di un altro array.

accumulatore Una variabile utilizzata in un ciclo per sommare o accumulare un risultato.

assegnazione aumentata Un'istruzione che aggiorna il valore di una variabile utilizzando un operatore come "=."

Operatore "dot" Operatore binario applicato elemento per elemento agli array.

sintassi "dot" Sintassi utilizzata per applicare una funzione elementwise a qualsiasi array.

operazione di riduzione Uno schema di elaborazione che attraversa una sequenza e accumula gli elementi in un unico valore risultato.

operazione "map" Un modello di elaborazione che attraversa una sequenza ed esegue un'operazione su ogni elemento.

Glossario I

array Una sequenza di valori.

elemento Uno dei valori in un array (o un'altra sequenza), chiamato anche item.

array annidato Un array che è un elemento di un altro array.

accumulatore Una variabile utilizzata in un ciclo per sommare o accumulare un risultato.

assegnazione aumentata Un'istruzione che aggiorna il valore di una variabile utilizzando un operatore come "=."

Operatore "dot" Operatore binario applicato elemento per elemento agli array.

sintassi "dot" Sintassi utilizzata per applicare una funzione elementwise a qualsiasi array.

operazione di riduzione Uno schema di elaborazione che attraversa una sequenza e accumula gli elementi in un unico valore risultato.

operazione "map" Un modello di elaborazione che attraversa una sequenza ed esegue un'operazione su ogni elemento.

Glossario II

oggetto Qualcosa a cui una variabile può fare riferimento. Un oggetto ha un tipo e un valore.

Glossario II

oggetto Qualcosa a cui una variabile può fare riferimento. Un oggetto ha un tipo e un valore.

equivalente Avere lo stesso valore.

Glossario II

oggetto Qualcosa a cui una variabile può fare riferimento. Un oggetto ha un tipo e un valore.

equivalente Avere lo stesso valore.

identico Essere lo stesso oggetto (che implica equivalenza).

Glossario II

oggetto	Qualcosa a cui una variabile può fare riferimento. Un oggetto ha un tipo e un valore.
equivalente	Avere lo stesso valore.
identico	Essere lo stesso oggetto (che implica equivalenza).
riferimento	L'associazione tra una variabile e il suo valore.

Glossario II

oggetto Qualcosa a cui una variabile può fare riferimento. Un oggetto ha un tipo e un valore.

equivalente Avere lo stesso valore.

identico Essere lo stesso oggetto (che implica equivalenza).

riferimento L'associazione tra una variabile e il suo valore.

aliasing Una circostanza in cui due o più variabili si riferiscono allo stesso oggetto.

Glossario II

oggetto Qualcosa a cui una variabile può fare riferimento. Un oggetto ha un tipo e un valore.

equivalente Avere lo stesso valore.

identico Essere lo stesso oggetto (che implica equivalenza).

riferimento L'associazione tra una variabile e il suo valore.

aliasing Una circostanza in cui due o più variabili si riferiscono allo stesso oggetto.

argomenti opzionali Argomenti che non sono obbligatori.

Glossario II

oggetto Qualcosa a cui una variabile può fare riferimento. Un oggetto ha un tipo e un valore.

equivalente Avere lo stesso valore.

identico Essere lo stesso oggetto (che implica equivalenza).

riferimento L'associazione tra una variabile e il suo valore.

aliasing Una circostanza in cui due o più variabili si riferiscono allo stesso oggetto.

argomenti opzionali Argomenti che non sono obbligatori.

delimitatore Un carattere o una stringa utilizzati per indicare dove deve essere suddivisa una stringa.

Section 9

Esercizi

aaaa

aaaa