EBook Gratuito

APPRENDIMENTO smalltalk

Free unaffiliated eBook created from
Stack Overflow contributors.

#smalltalk

Sommario

Di

Capitolo 1: Iniziare con smalltalk

Osservazioni

Questa sezione fornisce una panoramica di cosa è smalltalk e perché uno sviluppatore potrebbe volerlo utilizzare.

Dovrebbe anche menzionare tutti i soggetti di grandi dimensioni all'interno di smalltalk e collegarsi agli argomenti correlati. Poiché la Documentazione per Smalltalk è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Examples

Installazione o configurazione

Il nome Smalltalk di solito si riferisce a ANSI Smalltalk o Smalltalk 80 (di cui il primo è basato su). Mentre la maggior parte delle implementazioni sono vicine allo standard, variano in diversi aspetti (di solito indicati come *dialetti*).

Ogni implementazione ha il proprio metodo di installazione.

Le ben note implementazioni di FOSS sono:

Pharo Iniziato come una forchetta Squeak. (Windows / Linux / Mac OSX). Pharo ha la propria voce di documentazione in StackOverflow Documentation, quindi per favore guarda qui

Squeak (Windows / Linux / Mac OSX)

GNU Smalltalk (Windows / Linux / Mac OSX)

Dolphin Smalltalk Originariamente commerciale, ora open source gratuito. (Solo Windows)

Cuis Smalltalk Una forcella Squeak con particolare attenzione alla riduzione della complessità del sistema.

Smalltalks commerciali includono:

Versione di prova gratuita di VisualWorks / Cincom Smalltalk disponibile.

VisualAge Smalltalk Originariamente da IBM, ora Instatiations. Prova gratuita disponibile

Smalltalk / x (gratuito per uso personale?)

GemStone / s Edizione community gratuita disponibile.

Altri dialoghi Smalltalk

Amber Smalltalk A Smalltalk che vive nel browser.

Redline Smalltalk Smalltalk per JVM.

Elenco delle implementazioni Smalltalk su world.st

Ciao mondo in Smalltalk

```
Transcript show: 'Hello World!'.
```

Questo stamperà `Hello World!` alla finestra Transcript in Smalltalk. `Transcript` è la classe che consente di stampare sulla finestra Transcript inviando il messaggio `show:` a quell'oggetto. I due punti indicano che questo messaggio richiede un parametro che è in questo caso una stringa. Le stringhe sono rappresentate da virgolette singole e virgolette singole solo poiché le virgolette doppie sono riservate ai commenti in Smalltalk.

Leggi Iniziare con smalltalk online: https://riptutorial.com/it/smalltalk/topic/5316/iniziare-con-smalltalk

Capitolo 2: Sintassi Smalltalk

Examples

Letterali e commenti

coments

```
"Comments are enclosed in double quotes. BEWARE: This is NOT a string!"
"They can span
 multiple lines."
```

stringhe

```
'Strings are enclosed in sigle quotes.'
'Single quotes are escaped with a single quote, like this: ''.'
''''  "<--This string contains one single quote"

'Strings too can span
multiple lines'

''   "<--An empty string."
```

simboli

```
#thiIsaSymbol  "Symbols are interned strings, used for method and variable names,
                and as values with fast equality checking."
#'hello world' "A symbol with a space in it"
#''            "An empty symbol, not very useful"
#+
#1             "Not the integer 1"
```

Personaggi

```
$a     "Characters are not strings. They are preceded by a $."
$A     "An uppercase character"
$      "The spacecharacter!"
$→     "An unicode character"
$1     "Not to be confused with the number 1"
```

Numeri

I numeri sono disponibili in tutte le varietà:

Interi:

- Decimale

  10

  -1

  0

  100000000000000000000000000000000000000000

- esadecimale

  16rAB1F

  16r0

  -16rFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

- ScaledDecimal

  17s0

  3.14159265s8

- Altro

  8r7731 "ottale"

  2r1001 "binario"

  10r99987 "decimale di nuovo!"

Virgola mobile

```
3.14 1.2e3     "2 floating-point numbers"
```

frazioni

Le frazioni non sono le stesse dei numeri a virgola mobile, sono numeri esatti (nessun errore di arrotondamento).

```
4/3            "The fraction 4/3"
355/113        "A rational approximation to pi"
```

Array

```
#(#abc 123)    "A literal array with the symbol #abc and the number 123"
```

Array di byte

```
#[1 2 3 4]       "separators are blank"
#[]              "empty ByteArray"
#[0 0 0 0 255]   "length is arbitrary"
```

Matrici dinamiche

Gli array dinamici sono costruiti dalle espressioni. Ogni espressione all'interno delle parentesi valuta un valore diverso nell'array costruito.

```
{self foo. 3 + 2. i * 3}   "A dynamic array built from 3 expressions"
```

blocchi

```
[ :p | p asString ] "A code block with a parameter p.
                     Blocks are the same as lambdas in other languages"
```

Alcune note:

Nota che gli array letterali usano qualsiasi tipo e numero di spazi vuoti come separatori

```
#(256 16rAB1F 3.14s2 2r1001 $A #this)
"is the same as:"
#(256
  16rAB1F
  3.14s2
  2r1001
  $A #this)
```

Nota anche che puoi comporre letterali

```
#[255 16rFF 8r377 2r11111111]       (four times 255)

#(#[1 2 3] #('string' #symbol))     (arrays of arrays)
```

C'è una certa "tolleranza" alla notazione rilassata

```
#(symbol) = #(#symbol)              (missing # => symbol)

#('string' ($a 'a'))                (missing # => array)
```

Ma non qui:

```
#([1 2 3]) ~= #(#[1 2 3])           (missing # => misinterpreted)
```

però

```
#(true nil false)                   (pseudo variables ok)
```

Ma non qui!

```
#(self) = #(#self)                  (missing # => symbol)
```

Come puoi vedere ci sono un paio di incongruenze:

- Mentre le pseudo variabili `true` , `false` e `nil` sono accettate come letterali all'interno degli array, le pseudo variabili `self` e `super` sono interpretate come **simboli** (usando la regola più generale per le stringhe non qualificate).
- Mentre non è obbligatorio scrivere `#(` per l'avvio di un array nidificato in un array e la parentesi è sufficiente, è obbligatorio scrivere `#[` per avviare un `ByteArray` nidificato.

Alcuni di questi sono stati presi da:

http://stackoverflow.com/a/37823203/4309858

Invio di messaggi

In Smalltalk quasi tutto ciò che fai è *inviare messaggi* agli oggetti (chiamati metodi di chiamata in altre lingue). Esistono tre tipi di messaggi:

Messaggi unari:

```
#(1 2 3) size
"This sends the #size message to the #(1 2 3) array.
 #size is a unary message, because it takes no arguments."
```

Messaggi binari:

```
1 + 2
"This sends the #+ message and 2 as an argument to the object 1.
 #+ is a binary message because it takes one argument (2)
 and it's composed of one or two symbol characters"
```

Messaggi di parole chiave:

```
'Smalltalk' allButFirst: 5.
"This sends #allButFirst: with argument 5 to the string 'Smalltalk',
 resulting in the new string 'talk'"

3 to: 10 by: 2.
"This one sends the single message #to:by:, which takes two parameters (10 and 2)
 to the number 3.
 The result is a collection with 3, 5, 7, and 9."
```

Più messaggi in una dichiarazione sono valutati dall'ordine di precedenza

```
unary > binary > keyword
```

e da sinistra a destra.

```
1 + 2 * 3 " equals 9, because it evaluates left to right"
1 + (2 * 3) " but you can use parenthesis"

1 to:   #(a b c d) size   by:   5 - 4
 "is the same as:"
1 to: ( #(a b c d) size ) by: ( 5 - 4 )
```

Se si desidera inviare molti messaggi allo stesso oggetto, è possibile utilizzare l'operatore a cascata ; (punto e virgola):

```
OrderedCollection new
  add: #abc;
  add: #def;
  add: #ghi;
  yourself.
```

"Questo prima invia il messaggio #new alla classe OrderedCollection (#new è solo un messaggio, non un operatore) .Dispone a un nuovo OrderedCollection, quindi invia la nuova raccolta per tre volte il messaggio #add (con argomenti diversi), e il messaggio te stesso. "

Classi e metodi

Classi e metodi sono generalmente definiti nell'IDE Smalltalk.

Classi

Una definizione di classe ha qualcosa del genere nel browser:

```
XMLTokenizer subclass: #XMLParser
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'XML-Parser'
```

Questo è in realtà il messaggio che il browser invierà per creare una nuova classe nel sistema. (In

questo caso è `#subclass:instanceVariableNames:classVariableNames:poolDictionaries:category:` ma ce ne sono altri che creano anche nuove classi).

La prima riga mostra quale classe stai sottoclassi (in questo caso è XMLTokenizer) e il nome che la nuova sottoclasse avrà (#XMLParser).

Le prossime tre righe sono usate per definire le variabili che la classe e le sue istanze avranno.

metodi

I metodi hanno questo aspetto nel browser:

```
aKeywordMethodWith: firstArgument and: secondArgument
    "Do something with an argument and return the result."

    ^firstArgument doSomethingWith: secondArgument
```

Il `^` (caret) è l'operatore di ritorno.

```
** anInteger
    "Raise me to anInteger"
    | temp1 temp2 |

    temp1 := 1.
    temp2 := 1.
    1 to: anInteger do: [ :i | temp1 := temp1 * self + temp2 - i ].
    ^temp1
```

questo non è il modo giusto di fare esponenziazione, ma mostra una definizione di *messaggio binario* (sono definiti come qualsiasi altro messaggio) e alcune *variabili temporanee del metodo* (o *tempari di* metodo, *temp1* e *temp2*) più un argomento di blocco (*i*).

Loop in Smalltalk

Per questo esempio, verrà utilizzata una `Ordered Collection` per mostrare i diversi messaggi che possono essere inviati a un oggetto `OrderedCollection` per eseguire il loop sugli elementi.
Il codice seguente istanzia un oggetto `OrderedCollection` vuoto usando il messaggio `new` e poi lo popola con 4 numeri usando il messaggio `add:`

```
anOrderedCollection := OrderedCollection new.
anOrderedCollection add: 1; add: 2; add: 3; add: 4.
```

Tutti questi messaggi prenderanno un blocco come parametro che verrà valutato per ciascuno degli elementi all'interno della raccolta.

1. `do:`
   Questo è il messaggio di enumerazione di base. Ad esempio, se vogliamo stampare ogni elemento della collezione, possiamo ottenerlo come tale:

```
anOrderedCollection do:[:each | Transcript show: each]. "Prints --> 1234"
```

Ciascuno degli elementi all'interno della collezione sarà definito come l'utente desidera utilizzare questa sintassi `:each` This `do:` loop stamperà ogni elemento della collezione nella finestra `Transcript` .

2. `collect:`
   Il messaggio `collect:` ti consente di fare qualcosa per ogni oggetto della collezione e mette il risultato della tua azione in una nuova collezione
   Ad esempio, se volessimo moltiplicare ogni elemento della nostra collezione per 2 e aggiungerlo a una nuova raccolta, possiamo usare il messaggio `collect:` come tale:

```
evenCollection := anOrderedCollection collect:[:each | each*2]. "#(2 4 6 8)"
```

3. `select:`
   Il messaggio `select:` consente di creare una sotto-raccolta in cui gli elementi della collezione originale sono selezionati in base ad alcune condizioni che sono vere per loro. Ad esempio, se volessimo creare una nuova raccolta di numeri dispari dalla nostra raccolta, possiamo usare il messaggio `select:` come tale:

```
oddCollection := anOrderedCollection select:[:each | each odd].
```

   Dal momento che `each odd` restituisce un booleano, solo gli elementi che rendono il ritorno booleano vero verranno aggiunti a `oddCollection` che avrà `#(1 3)` .

4. `reject:`
   Questo messaggio funziona in modo opposto per `select:` e rifiuta qualsiasi elemento che renda `true` il booleano. O, in altre parole, selezionerà tutti gli elementi che rendono `false` ritorno booleano. Ad esempio se volessimo costruire lo stesso `oddCollection` come nell'esempio precedente. Possiamo usare `reject:` come tale:

```
oddCollection := anOrderedCollection reject:[:each | each even].
```

   `oddCollection` avrà ancora `#(1 3)` come suoi elementi.

Queste sono le quattro tecniche di enumerazione di base in Smalltalk. Tuttavia, non esitare a consultare la classe `Collections` per ulteriori messaggi che potrebbero essere implementati.

Leggi Sintassi Smalltalk online: https://riptutorial.com/it/smalltalk/topic/5422/sintassi-smalltalk

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con smalltalk	Bananeweizen, Community, fede s., Leandro Caniglia, Stephan Eggermont, ybce
2	Sintassi Smalltalk	aka.nice, fede s., ybce