EBook Gratis

APRENDIZAJE smalltalk

Free unaffiliated eBook created from
Stack Overflow contributors.

#smalltalk

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: smalltalk

It is an unofficial and free smalltalk ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official smalltalk.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con smalltalk

Observaciones

Esta sección proporciona una descripción general de qué es smalltalk y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de smalltalk, y vincular a los temas relacionados. Dado que la Documentación para Smalltalk es nueva, es posible que deba crear versiones iniciales de los temas relacionados.

Examples

Instalación o configuración

El nombre Smalltalk generalmente se refiere a ANSI Smalltalk o Smalltalk 80 (en el cual se basa el primero). Si bien la mayoría de las implementaciones son cercanas al estándar, varían en diferentes aspectos (generalmente denominados *dialectos*).

Cada implementación tiene su propio método de instalación.

Las implementaciones conocidas de FOSS son:

Pharo comenzó como un tenedor Squeak. (Windows / Linux / Mac OSX). Pharo tiene su propia entrada de documentación en la Documentación de Stackoverflow, así que eche un vistazo allí

Squeak (Windows / Linux / Mac OSX)

GNU Smalltalk (Windows / Linux / Mac OSX)

Dolphin Smalltalk Originalmente comercial, ahora libre de código abierto. (Windows solamente)

Cuis Smalltalk Una horquilla Squeak enfocada en reducir la complejidad del sistema.

Smalltalks comerciales incluyen:

Prueba gratuita de VisualWorks / Cincom Smalltalk disponible.

VisualAge Smalltalk Originalmente por IBM, ahora Interpretaciones. Prueba gratuita disponible

Smalltalk / x (¿Gratis para uso personal?)

Edición de comunidad gratuita de GemStone / s disponible.

Otros dialectos de Smalltalk

Ámbar Smalltalk Un Smalltalk que vive en el navegador.

Redline Smalltalk Smalltalk para la JVM.

Lista de implementaciones de Smalltalk en world.st

Hola Mundo en Smalltalk

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

Esto imprimirá `Hello World!` a la ventana Transcripción en Smalltalk. `Transcript` es la clase que le permite imprimir en la ventana Transcripción enviando el mensaje `show:` a ese objeto. Los dos puntos indican que este mensaje requiere un parámetro que en este caso es una cadena. Las cadenas están representadas por comillas simples y comillas simples solo porque las comillas dobles están reservadas para comentarios en Smalltalk.

Lea Empezando con smalltalk en línea: https://riptutorial.com/es/smalltalk/topic/5316/empezando-con-smalltalk

Capítulo 2: Sintaxis de Smalltalk

Examples

Literales y comentarios

Los comentarios

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

Instrumentos de cuerda

```
'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."
```

Simbolos

```
#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"
```

Caracteres

```
$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"
```

Números

Los números vienen en todas las variedades:

Enteros:

- Decimal

  10

  -1

  0

  100000000000000000000000000000000000000

- Hexadecimal

  16rAB1F

  16r0

  -16rFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

- ScaledDecimal

  17s0

  3.14159265s8

- Otro

  8r7731 "octal"

  2r1001 "binario"

  10r99987 "decimal otra vez!"

Punto flotante

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

Fracciones

Las fracciones no son lo mismo que los números de punto flotante, son números exactos (sin error de redondeo).

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

Arrays

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

Matrices de bytes

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

Matrices dinámicas

Las matrices dinámicas se construyen a partir de expresiones. Cada expresión dentro de las llaves se evalúa a un valor diferente en la matriz construida.

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

Bloques

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

Algunas notas:

Tenga en cuenta que las matrices literales utilizan cualquier tipo y número de espacios en blanco como separadores

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

Tenga en cuenta también que puede componer literales

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

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

Hay cierta "tolerancia" a la notación relajada.

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

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

Pero no aquí:

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

sin embargo

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

¡Pero no aquí!

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

Como puedes ver hay un par de inconsistencias:

- Mientras que las pseudo variables `true` , `false` y `nil` se aceptan como literales dentro de matrices, las pseudo variables `self` y `super` se interpretan como **símbolos** (utilizando la regla más general para cadenas no calificadas).
- Si bien no es obligatorio escribir `#(` para iniciar una matriz anidada en una matriz y el paréntesis es suficiente, es obligatorio escribir `#[` para iniciar una `ByteArray` anidada.

Algo de esto fue tomado de:

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

Envío de mensajes

En Smalltalk, casi todo lo que hace es *enviar mensajes* a objetos (denominados métodos de llamada en otros idiomas). Hay tres tipos de mensajes:

Mensajes unarios:

```
#(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."
```

Mensajes binarios:

```
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"
```

Mensajes de palabras clave:

```
'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."
```

Varios mensajes en una declaración son evaluados por el orden de precedencia

```
unary > binary > keyword
```

y de izquierda a derecha.

```
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 )
```

Si desea enviar muchos mensajes al mismo objeto, puede usar el operador en cascada ; (punto y coma):

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

"Esto primero envía el mensaje #new a la clase OrderedCollection (#new es solo un mensaje, no un operador). Da como resultado una nueva OrderedCollection. Luego envía la nueva colección tres veces el mensaje #ad (con diferentes argumentos), y el mensaje tu mismo ".

Clases y metodos

Las clases y los métodos se definen generalmente en el IDE de Smalltalk.

Las clases

Una definición de clase se ve algo así en el navegador:

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

Este es realmente el mensaje que el navegador le enviará para que cree una nueva clase en el

sistema. (En este caso es `#subclass:instanceVariableNames:classVariableNames:poolDictionaries:category:` pero hay otros que también crean nuevas clases).

La primera línea muestra qué clase está subclasificando (en este caso es XMLTokenizer) y el nombre que tendrá la nueva subclase (#XMLParser).

Las siguientes tres líneas se utilizan para definir las variables que tendrán la clase y sus instancias.

Métodos

Los métodos se ven así en el navegador:

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

    ^firstArgument doSomethingWith: secondArgument
```

El ^ (caret) es el operador de retorno.

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

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

esta no es la forma correcta de hacer la exponenciación, pero muestra una definición *binaria de mensajes* (que están definidas como cualquier otro mensaje) y algunas *variables temporales método* (o temporales método, *Temp 1* y *Temp 2)* más un argumento bloque *(i).*

Bucles en Smalltalk

Para este ejemplo, se utilizará una `Ordered Collection` para mostrar los diferentes mensajes que se pueden enviar a un objeto `OrderedCollection` para recorrer los elementos.
El siguiente código creará una `OrderedCollection` vacía usando el mensaje `new` y luego lo `OrderedCollection` con 4 números usando el mensaje `add:`

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

Todos estos mensajes tomarán un bloque como parámetro que se evaluará para cada uno de los elementos dentro de la colección.

1. `do:`
   Este es el mensaje básico de enumeración. Por ejemplo, si queremos imprimir cada

elemento de la colección, podemos lograrlo como tal:

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

Cada uno de los elementos dentro de la colección se definirá como el usuario lo desee con esta sintaxis `:each` Este bucle `do:` imprimirá todos los elementos de la colección en la ventana `Transcript` .

2. `collect:`
   El mensaje de `collect:` permite hacer algo para cada elemento de la colección y coloca el resultado de su acción en una nueva colección
   Por ejemplo, si quisiéramos multiplicar cada elemento de nuestra colección por 2 y agregarlo a una nueva colección, podemos usar el mensaje de `collect:` :

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

3. `select:`
   El mensaje de `select:` permite crear una subcolección donde se seleccionan los elementos de la colección original en función de alguna condición que sea verdadera para ellos. Por ejemplo, si quisiéramos crear una nueva colección de números impares de nuestra colección, podemos usar el mensaje `select:` como tal:

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

   Como `each odd` devuelve un valor booleano, solo los elementos que hacen que el retorno booleano sea verdadero se agregarán a `oddCollection` que tendrá `#(1 3)` .

4. `reject:`
   Este mensaje funciona al contrario para `select:` y rechaza cualquier elemento que haga que el retorno booleano sea `true` . O, en otras palabras, seleccionará cualquier elemento que haga que el retorno booleano sea `false` . Por ejemplo, si quisiéramos construir la misma `oddCollection` como el ejemplo anterior. Podemos utilizar `reject:` como tal:

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

   `oddCollection` nuevamente tendrá `#(1 3)` como sus elementos.

Estas son las cuatro técnicas básicas de enumeración en Smalltalk. Sin embargo, no dude en navegar por la clase `Collections` para ver más mensajes que pueden implementarse.

Lea Sintaxis de Smalltalk en línea: https://riptutorial.com/es/smalltalk/topic/5422/sintaxis-de-smalltalk

Creditos

S. No	Capítulos	Contributors
1	Empezando con smalltalk	Bananeweizen, Community, fede s., Leandro Caniglia, Stephan Eggermont, ybce
2	Sintaxis de Smalltalk	aka.nice, fede s., ybce