eBook Gratuit

APPRENEZ smalltalk

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#smalltalk

Table des matières

À propos

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

Chapitre 1: Démarrer avec smalltalk

Remarques

Cette section fournit une vue d'ensemble de ce qu'est smalltalk et pourquoi un développeur peut vouloir l'utiliser.

Il convient également de mentionner tous les grands sujets à l'intérieur de Smalltalk et de les relier aux sujets connexes. La documentation de Smalltalk étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Installation ou configuration

Le nom Smalltalk se réfère généralement à ANSI Smalltalk ou Smalltalk 80 (dont le premier est basé sur). Bien que la plupart des implémentations soient proches de la norme, elles varient selon les aspects (généralement appelés *dialectes*).

Chaque implémentation a sa propre méthode d'installation.

Les implémentations FOSS bien connues sont:

Pharo a commencé comme une fourche Squeak. (Windows / Linux / Mac OSX). Pharo a sa propre entrée de documentation à Stackoverflow Documentation, donc s'il vous plaît jeter un oeil là - bas

Squeak (Windows / Linux / Mac OSX)

GNU Smalltalk (Windows / Linux / Mac OSX)

Dolphin Smalltalk Commercialement à l'origine, maintenant open source libre. (Windows uniquement)

Cuis Smalltalk Une fourche Squeak axée sur la réduction de la complexité du système.

Les petites entreprises commerciales comprennent:

VisualWorks / Cincom Smalltalk Essai gratuit disponible.

VisualAge Smalltalk À l'origine par IBM, maintenant Instatiations. Essai gratuit disponible

Smalltalk / x (Gratuit pour un usage personnel?)

GemStone / s Edition communautaire gratuite disponible.

Autres dialectes Smalltalk

Ambre Smalltalk Un Smalltalk qui vit dans le navigateur.

Redline Smalltalk Smalltalk pour la JVM.

Liste des implémentations de Smalltalk sur world.st

Bonjour tout le monde dans Smalltalk

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

Cela va imprimer `Hello World!` à la fenêtre de transcription dans Smalltalk. `Transcript` est la classe qui vous permet d'imprimer dans la fenêtre Transcript en envoyant le message `show:` à cet objet. Les deux points indiquent que ce message nécessite un paramètre qui est dans ce cas une chaîne. Les chaînes sont représentées par des guillemets simples et des guillemets simples, car les guillemets sont réservés aux commentaires dans Smalltalk.

Lire Démarrer avec smalltalk en ligne: https://riptutorial.com/fr/smalltalk/topic/5316/demarrer-avec-smalltalk

Chapitre 2: Syntaxe Smalltalk

Examples

Littéraux et commentaires

Les commentaires

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

Cordes

```
'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."
```

Des symboles

```
#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"
```

Personnages

```
$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"
```

Nombres

Les nombres viennent dans toutes les variétés:

Entiers:

- Décimal

  dix

  -1

  0

  100000000000000000000000000000000000000000

- Hexadécimal

  16rAB1F

  16r0

  -16rFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

- ScaledDecimal

  17s0

  3.14159265s8

- Autre

  8r7731 "octal"

  2r1001 "binaire"

  10r99987 "décimal à nouveau!"

Point flottant

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

Fractions

Les fractions ne sont pas les mêmes que les nombres à virgule flottante, ce sont des nombres exacts (aucune erreur d'arrondi).

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

Tableaux

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

Tableaux d'octets

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

Tableaux dynamiques

Les tableaux dynamiques sont construits à partir d'expressions. Chaque expression à l'intérieur des accolades est évaluée à une valeur différente dans le tableau construit.

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

Blocs

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

Quelques notes:

Notez que les tableaux littéraux utilisent n'importe quel type et nombre de blancs comme séparateurs

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

Notez également que vous pouvez composer des littéraux

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

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

Il y a une certaine "tolérance" à la notation assouplie

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

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

Mais pas ici:

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

toutefois

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

Mais pas ici!

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

Comme vous pouvez le voir, il existe quelques incohérences:

- Alors que les pseudo-variables `true` , `false` et `nil` sont acceptées comme littéraux dans les tableaux, les pseudo-variables `self` et `super` sont interprétées comme des **symboles** (en utilisant la règle plus générale pour les chaînes non qualifiées).
- Bien qu'il ne soit pas obligatoire d'écrire `#(` pour démarrer un tableau imbriqué dans un tableau et que la parenthèse suffit, il est obligatoire d'écrire `#[` pour démarrer un objet `ByteArray` imbriqué.

Une partie de ceci a été prise de:

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

Envoi de message

Dans Smalltalk, presque tout ce que vous faites consiste à *envoyer des messages* à des objets (appelés méthodes d'appel dans d'autres langues). Il existe trois types de messages:

Messages unaires:

```
#(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."
```

Messages binaires:

```
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"
```

Messages de mots clés:

```
'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."
```

Plusieurs messages dans une instruction sont évalués par l'ordre de priorité

```
unary > binary > keyword
```

et de gauche à droite.

```
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 vous souhaitez envoyer plusieurs messages au même objet, vous pouvez utiliser l'opérateur en cascade ; (point-virgule):

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

"Cela envoie d'abord le message #new à la classe OrderedCollection (#new est juste un message, pas un opérateur). Il en résulte une nouvelle OrderedCollection. Il envoie ensuite la nouvelle collection trois fois le message #add (avec différents arguments), et le message vous-même. "

Classes et méthodes

Les classes et les méthodes sont généralement définies dans l'IDE Smalltalk.

Des classes

Une définition de classe ressemble à ceci dans le navigateur:

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

Il s'agit en fait du message que le navigateur vous enverra pour créer une nouvelle classe dans le système. (Dans ce cas, c'est `#subclass:instanceVariableNames:classVariableNames:poolDictionaries:category:` mais il y en a d'autres qui `#subclass:instanceVariableNames:classVariableNames:poolDictionaries:category:` également de nouvelles classes).

La première ligne indique quelle classe vous êtes en train de sous-classer (dans ce cas, c'est XMLTokenizer) et le nom que la nouvelle sous-classe aura (#XMLParser).

Les trois lignes suivantes sont utilisées pour définir les variables que la classe et ses instances auront.

Les méthodes

Les méthodes ressemblent à ceci dans le navigateur:

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

    ^firstArgument doSomethingWith: secondArgument
```

Le `^` (caret) est l'opérateur de retour.

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

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

ce n'est pas la bonne façon d'exécuter une exponentiation, mais elle montre une définition de *message binaire* (ils sont définis comme n'importe quel autre message) et certaines *variables temporaires de méthode* (ou temporelles de méthode, *temp1* et *temp2*) plus un argument de bloc (*i*).

Boucles dans Smalltalk

Pour cet exemple, une `Ordered Collection` sera utilisée pour afficher les différents messages pouvant être envoyés à un objet `OrderedCollection` pour effectuer une boucle sur les éléments. Le code ci-dessous instanciera un `OrderedCollection` vide en utilisant le message `new` puis le `OrderedCollection` avec 4 numéros en utilisant le message `add:`

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

Tous ces messages prendront un bloc comme paramètre qui sera évalué pour chacun des

éléments de la collection.

1. `do:`
   C'est le message d'énumération de base. Par exemple, si nous voulons imprimer chaque élément de la collection, nous pouvons y parvenir en tant que tel:

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

   Chacun des éléments à l' intérieur de la collection seront définis comme les souhaits de l' utilisateur en utilisant cette syntaxe: `:each` Ce `do:` boucle imprimera tous les éléments de la collection à la `Transcript` fenêtre.

2. `collect:`
   Le message `collect:` permet de faire quelque chose pour chaque élément de la collection et place le résultat de votre action dans une nouvelle collection
   Par exemple, si nous voulions multiplier chaque élément de notre collection par 2 et l'ajouter à une nouvelle collection, nous pouvons utiliser le message `collect:` tant que tel:

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

3. `select:`
   Le message `select:` permet de créer une sous-collection dans laquelle les éléments de la collection d'origine sont sélectionnés en fonction de certaines conditions. Par exemple, si nous voulions créer une nouvelle collection de nombres impairs provenant de notre collection, nous pouvons utiliser le message `select:` tant que tel:

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

   Comme `each odd` retourne un booléen, seuls les éléments qui rendent le booléen true seront ajoutés à `oddCollection` qui aura `#(1 3)` .

4. `reject:`
   Ce message fonctionne en sens inverse pour `select:` et rejette tous les éléments qui rendent le retour booléen `true` . Ou, en d'autres termes, il sélectionnera tous les éléments qui rendent le retour booléen `false` . Par exemple, si nous voulions construire la même `oddCollection` comme dans l'exemple précédent. Nous pouvons utiliser le `reject:` tant que tel:

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

   `oddCollection` aura à nouveau `#(1 3)` comme éléments.

Ce sont les quatre techniques de dénombrement de base de Smalltalk. Cependant, n'hésitez pas à parcourir la classe `Collections` pour plus de messages pouvant être implémentés.

Lire Syntaxe Smalltalk en ligne: https://riptutorial.com/fr/smalltalk/topic/5422/syntaxe-smalltalk

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec smalltalk	Bananeweizen, Community, fede s., Leandro Caniglia, Stephan Eggermont, ybce
2	Syntaxe Smalltalk	aka.nice, fede s., ybce