

Podstawy Groovy: typy i operacje

© Krzysztof Barteczko, PJWSTK 2012-2017

Skrypty

Co to są **skrypty**
i języki skryptowe ?

Perl, Python, REXX, Ruby, PHP, ...
VBScript, WinScripting, AppleScripting

No hard-and-fast definition of what a scripting language is exists, but generally scripting languages are interpreted languages that feature higher productivity than more traditional systems languages.

Zastosowania:

- programy uniwersalne
- makra w systemach (aplikacjach)
- integracja

Integracja:

- skrypty działające w ramach systemu
- skrypty - łączniki między różnymi aplikacjami
- skrypty w Internecie i mashupy

Scripting languages are glue languages. They stitch together existing software, such as objects, components, widgets, operating system commands, programs, functions, and other forms of existing code. Scripting languages are higher-level than traditional programming languages because they more easily leverage existing software.

Cytaty: The Quiet Revolution:Open Source Scripting
by Howard Fosdick 2009

Groovy

- * is an agile and dynamic language for the Java Virtual Machine
- * builds upon the strengths of Java but has additional power features inspired by languages like Python, Ruby and Smalltalk
- * supports Domain-Specific Languages and other compact syntax so your code becomes easy to read and maintain
- * makes writing shell and build scripts easy with its powerful processing primitives, OO abilities and an Ant DSL
- * increases developer productivity by reducing scaffolding code when developing web, GUI, database or console applications
- * seamlessly integrates with all existing Java objects and libraries
- * compiles straight to Java bytecode so you can use it anywhere you can use Java

Cytat ze strongy Groovy

Groovy

Groovy = Java – "boiler plate code"

- + dynamiczne typowanie
- + dynamiczne wykonanie kodu (skryptowanie)
- + znacznie poszerzona funkcjonalność bibliotek
- + rozbudowane oraz modyfikowalne instrukcje sterujące
- + przeciążanie operatorów
- + przyjemna składnia dla kolekcji
- + domknięcia i elementy programowania funkcyjnego
- + łatwe w użyciu parsery (XML/HTML, JSON)
- + rozbudowane rodzaje napisów i "templates"
- + wbudowane w składnię przetwarzanie wyr. regularnych
- + buildery (Swing, Ant, Html, Xml, JSON, Graphics ...)
- + metaprogramowanie fazy kompilacji i wykonania
- + doskonałe wsparcie do tworzenia DSL

Skrypty w języku Groovy

Skrypt:

- kod w jednym pliku źródłowym (lub jako napis w programie),
- nie ma potrzeby definiowania klas ani metod (ale można),
- nie ma potrzeby deklarowania zmiennych (ale można).

Można łatwo pisać proste programy.

Groovy:

```
name = 'World'
println "Hello $name"
```

Java:

```
public class Greeting {
    public static void main(String ... args) {
        String name = "World";
        System.out.println("Hello " + name);
    }
}
```

Średnik: w Groovy można pomijać, gdy w jednym wierszu jedna instrukcja. **Nawiasy** w wywołaniu metody/funkcji: można pomijać, gdy są argumenty wywołania. Różne rodzaje **literałów napisowych** ("...", '...') – zob. dalej o napisach.

Przykład praktyczny: kursy euro

```
import groovy.swing.SwingBuilder

def url = 'http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml'
def rates = new XmlParser().parse(url)
def map = [:]
rates.Cube.Cube.Cube.each {
    map[it.@currency] = it.@rate
}
new SwingBuilder().edt {
    frame(title: 'Euro rates', pack: true, visible: true) {
        panel() {
            comboBox(id: 'cb', border: titledBorder('Select currency'),
                prototypeDisplayValue: 'xxxxxxxxxxxxx',
                items: map.keySet().toList(),
                actionPerformed: {
                    def cur = it.source.selectedItem
                    lab.text = map[cur]
                })
            label(id: 'lab', preferredSize: cb.preferredSize,
                border: titledBorder('Rate'))
        }
    }
}
```



Dalej poznamy wszystkie zastosowane tu konstrukcje; tu warto zwrócić uwagę na prostotę programu (w Javie wymagałby on dużo więcej pracy i zajął dużo więcej miejsca).

Instalacja

1. JDK z java.sun.com - program instalacyjny prowadzi za rękę
2. Dokumentacja z java.sun.com - unzip
3. Groovy z groovy.codehaus.org - unzip
4. (opcjonalnie) Wybrane IDE
5. Ustawić zmienne środowiskowe PATH, JAVA_HOME i GROOVY_HOME
6. Test:

Zapisać program w pliku Start.groovy:

```
println "I'm Groovy!"
```

Z wiersza poleceń:

```
groovy Start.groovy
```

Na konsoli uzyskamy napis:

```
I'm Groovy!
```

Praca z Eclipse

Eclipse pobrać można ze strony eclipse.org. Pobrane archiwum wystarczy rozpakować na dysku. Należy też doinstalować plug-in dla języka Groovy ("Software updates"). Zob:

<https://github.com/groovy/groovy-eclipse/wiki>

Na starcie Eclipse wybieramy Workspace - czyli obszar roboczy. Jest to wybrany przez nas katalog, w którym będą nasze programy. Workspace zawiera projekty. Programy umieszczane są w projektach. Zatem sekwencja działań jest następująca:

- * uruchomić Eclipse,
- * wybrać Workspace,
- * utworzyć nowy projekt ("Groovy project"),
- * w danym projekcie utworzyć program ("Groovy class").

Literały

Literały
liczbowe:

1
10
2.1
1000.33

Literały
boolowskie
- słowa
kluczowe:
true false

Literały
napisowe:

'Hello Groovy'
'1'
/I'm slashy string\x/
"I'm Gstring \$v"
'''Multiline
string'''
""Multiline GString
\$x = \$y""

Znaki - kodowanie Unicode

W Groovy literały są
obiektami odpowiednich
klas, np.

1 - typ (klasa) Integer
1.1 - typ BigDecimal
'x' - typ String
'1' - typ String
'ala ma kota' - typ String
"ala ma kota" - typ GString

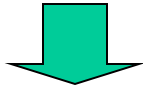
Integer - całkowite od -2147483648 do 2147483647

BigDecimal - dowolnie duże liczby z dowolną precyzją

Literały - znaki specjalne

Znaki specjalne	Zapis
Line feed (LF)	\n
Tabulacja (Tab)	\t
Backspace (BS)	\b
Carriage return (CR)	\r
Form feed (FF)	\f
Apostrof w '...'	\'
Cudzysłów w "..."	\"
Backslash	\\
Znak Unicode o kodzie NNNN	\uNNNN

```
print '\u03b1\u03b2\u03b3 '  
print '\\'  
print \"'  
print " alfa beta gamma "  
println '\nalfa\nbeta\ngamma'  
println "c:\\util\\bak"  
println /c:\util\bak/
```



```
αβγ \ ' alfa beta gamma '  
alfa  
beta  
gamma  
c:\util\bak  
c:\util\bak
```

Uwaga na:
znaki specjalne,
slashy string,
różnice w działaniu
print - bez przejścia do nowej linii,
println - z przejściem

By uzyskać na konsoli Eclipse znaki UTF:

"Run Config-Common-Console Encoding" UTF-8

MODYFIKATORY

float f = 1f, long l = 1L,

double d = 1d, BigDecimal = 1g

Zmienne i dynamiczne typowanie

W skryptach można nie deklarować zmiennych:

```
x = 10
```

Deklaracja zmiennej: `def` lub użycie konkretnego typu.

Zmienne nie zadeklarowane lub zadeklarowane z `def` są dynamicznie typowane:

```
def x = 10 // typ: Integer
```

```
x = 'Ala' // a teraz zmienił się na String
```

Użycie konkretnego typu w deklaracji zmiennej wymusza zgodność typów (typ zmiennej nie może być zmieniony).

```
Integer v = 1  
v = 'Groovy'
```

Błąd

Zamiast nazwy typu - słowo **def** (deklaracja zmiennej dowolnego typu):

```
def v = 1  
v = 'Groovy'
```

Ok

Redeklaracje niedozwolone:

```
def v  
...  
def v  
String s  
...  
Integer s
```

Błędy

W Groovy wszystko jest obiektem


Gdy używamy liczb, napisów, znaków, wartości boolowskich lub specjalnych konstrukcji składniowych (np. dla list, map, wyrażeń regularnych) odpowiednie obiekty tworzone są automatycznie.

W innych przypadkach musimy sami tworzyć obiekty za pomocą wyrażenia **new**, np.

```
d = new Date()
```

```
scanner = new Scanner(text)
```

```
file = new File('c:/temp/text.txt')
```



Argumenty wywołania
konstruktorów

Zmienne zawierają referencje do obiektów!

Hierarchie typów

Typy są wyznaczane przez klasy i przez interfejsy.

Klasy mogą dziedziczyć inne klasy i implementować interfejsy.

Object

Number extends Object

Integer extends Number

Literał 1 jest typu Integer i Number i Object

Object

AbstractMap extends Object implements Map

HashMap extends AbstractMap

Obiekt hash-mapa jest typu Object, Map, AbstractMap, HashMap

Typy proste

W Javie oprócz typów obiektowych występują typy proste.

W języku Groovy można używać typów prostych, ale dane tych typów są i tak przekształcane na obiekty odpowiadających im klas.

```
byte b = 1
long l = 1111111
char c = 'x'
int i = 111
[ b, l, c, i ].each {
    println it.class.name
}
```

Wynik:

```
java.lang.Byte
java.lang.Long
java.lang.Character
java.lang.Integer
```

nazwa typu prostego	nazwa klasy	znaczenie
byte	Byte	liczby całkowite
short	Short	
int	Integer	
long	Long	
float	Float	liczby rzeczywiste
double	Double	
char	Character	znaki Unicode
boolean	Boolean	wartości logiczne: prawda, fałsz



Autoboxing!

Operatory

Operatory

arytmetyczne:

* mnożenie
/ dzielenie
+ dodawanie
- odejmowanie
** potęgowanie (G)
% reszta z dzielenia
++ zwiększanie
-- zmniejszanie

Operatory relacyjne:

< mniejsze
<= mniejsze lub równe
>= większe lub równe
> większe
== czy równe
!= czy nie równe

Przypisania:

= (i pochodne)

Operatory

logiczne:

! negacja
|| alternatywa
&& koniunkcja

I wiele innych

Operator instanceof:

stwierdzenie typu:

x instanceof A

czy x jest typu A lub dowolnego jego podtypu

Groovy pozwala na definiowanie operatorów w klasach i w wielu jego standardowych klasach jest to zrobione.

Operacje na liczbach - cechy szczególne

Przy dzieleniu dwóch liczb całkowitych następuje promocja do BigDecimal i wynik jest zgodny z "arytmetyką".

Java: $1/3 == 0$

Groovy: $1/3 = 0.33333333$

Literały rzeczywiste są typu BigDecimal, wobec tego operacje są dokładne.

Java: $3 * 0.2 != 0.6$

Groovy: $3 * 0.2 = 6$

Potęgowanie:

Java: konieczne użycie funkcji z klasy Math

Groovy: operator `**`

Złożone operatory przypisania

Złożone operatory przypisania mają postać:

`op=`

gdzie `op` - to jeden z operatorów

`* / % + - << >> >>> & ^ |`

Złożone operatory przypisania, stosowane w następujący sposób:

`x op= wyrażenie`

są wygodną formą skrócenia zapisu:

`x = x op (wyrażenie)`

gdzie:

`x` - dowolna zmienna

`wyrażenie` - dowolne wyrażenie

`op` - symbol operatora

Na przykład, zamiast:

```
numOfChildren = numOfChildren + 2
```

możemy napisać:

```
numOfChildren += 2
```

Zwiększanie i zmniejszanie

`++` zwiększa o jeden wartość argumentu (zmiennej)
`--` zmniejsza

Przyrostkowa forma operatorów (znak operatora po argumentach) modyfikuje wartość argumentu po jej wykorzystaniu w wyrażeniu.

Przedrostkowa (znak operatora przed argumentem) - przed wykorzystaniem tej wartości.

Np.

```
def n, i = 1  
n = i++      // przyrostkowa forma operatora ++
```

zmienna `i` zostanie zwiększona o 1, ale zmiennej `n` zostanie nadana wartość zmiennej `i` sprzed zwiększenia, czyli po wykonaniu instrukcji: `n` będzie równe 1, a `i` będzie równe 2.

```
def n, i = 1;  
n = ++i;      // przedrostkowa forma operatora ++
```

zmienna `i` zostanie zwiększona o 1 i ta nowa jej wartość zostanie przypisana zmiennej `n`, czyli po wykonaniu instrukcji: `n = 2`, `i = 2`.

Odnośniki do dokumentacji

Skrypty:

http://users.pja.edu.pl/~kb/PJPadd/docs/structure.html_scripts_versus_classes.html

Typy proste i autoboxing:

http://users.pja.edu.pl/~kb/PJPadd/docs/objectorientation.html_primitive_types.html

Typy liczbowe: http://users.pja.edu.pl/~kb/PJPadd/docs/syntax.html_numbers.html

Operatory: <http://groovy-lang.org/operators.html>

Ćwiczenia

1. Zainstalować JDK, Groovy, Eclipse IDE, Groovy Plug-in dla Eclipse
2. Napisać w wybranym edytorze tekstowym program typu "hello world" w Groovy. Uruchomić go z konsoli. Wypróbować program GroovyConsole z instalacji Groovy
3. Praca z Eclipse: workspace, tworzenie projektów i programów Groovy, nawigacja, uruchamianie programów (na najprostszym przykładzie "hello world")