Dokumentacja języka EventScript

Projekt TKOM

Michał Sieczkowski

Opis języka

Język służy do programowania prostych zadań w postaci skryptów uruchamianych na urządzeniu mobilnym. Zapewnia podstawowe elementy znane z innych języków takie jak: zmienne, funkcje, instrukcje sterujące, pętle podstawowe typy danych oraz ich arytmetykę. Dla użytku programisty zdefiniowane są również metody pozwalające na sterowanie wybranymi funkcjami urządzenia mobilnego.

Typy danych

Poniższa tabela podsumowuje dostępne typy danych

Typ	Opis	Domyślna wartość
bool	typ boolowski	false
int	liczba całkowita 32-bitowa	0
float	liczba zmiennoprzecinkowa	0.0
string	napis ze znaków UTF-16	”” (pusty napis)
duration	okres z dokładnością do sekundy, jest to typ składający się de facto z 4 liczb całkowitych	0 dni 0 godzin 0 minut 0 sekund
datetime	data i czas z dokładnością do sekundy	aktualna data i czas

PRZYKŁADY LITERAŁÓW

```
false
1024
3.14
"Hello Wolrd!"
duration()              // == duration(0,0,0,0)
duration(0, 1, 0, 0)    //duration(seconds, minutes, days, years)
datetime()              // == now
datetime(2018, 1, 25, 13, 30, 59)
                        //datetime(year, month, dayOfMonth, hour, minute, second)
```

Zmienne

Możliwa jest deklaracja zmiennej w wyniku czego przyjmuje ona domyślną wartość.

```
variableDeclaration
    : VAR IDENTIFIER ':' type
    ;
```

PRZYKŁAD

```
var myNum : int             // myNum == 0
var currentTime : datetime  // currentTime == now
```

Zmienną można również zdefiniować przypisując do niej wartość, typ jest wnioskowany.

```
variableDefinition
    : VAR IDENTIFIER '=' expression
    ;
```

PRZYKŁAD

```
var myInt = 1024                    // definition
myInt = 512                         // assignment
var currentTime = datetime()        // definition
```

Instrukcje sterujące

Pętla:

```
FOR '(' forInit? ';' expression? ';' forUpdate=expressionList? ')'
blockOrStatement
```

PRZYKŁAD

```
for (var i = 0; i<5; ++i) {
    Speak(i)
}
```

Instrukcja warunkowa:

```
IF '(' expression ')' blockOrStatement (ELSE blockOrStatement)?
```

PRZYKŁAD

```
if (false) myInt = 30
else myInt = 40
```

Wyrażenia

Gramatyka języka definiuje m.in. wyrażenia arytmetyczne, logiczne, porównania, przypisania oraz wywołania funkcji. W sumie zdefiniowane jest 14 typów wyrażeń.

Język pozwala na:

- dodawanie, odejmowanie, mnożenie oraz dzielenie liczb całkowitych oraz zmiennoprzecinkowych
- dodawanie i odejmowanie okresu od daty
- obliczanie okresu jako różnicy między datami
- konkatenacje napisów z typami liczbowymi oraz czasowymi

```
expression
    : literal                                                   #literalExp
    | IDENTIFIER                                                #identifierExp
    | expression bop='.' literalFunctionCall                    #literalFuncExp
    | functionCall                                              #functionExp
    | builtInFunctionCall                                       #builtInFuncExp
```

```
    | prefix=('+'|'-'|'++'|'--') expression                    #unaryExp
    | prefix='!' expression                                    #negationExp
    | expression bop=('*'|'/'|'%') expression                  #multiplicativeExp
    | expression bop=('+'|'-') expression                      #additiveExp
    | expression bop=('<=' | '>=' | '>' | '<') expression      #relationalExp
    | expression bop=('==' | '!=') expression                  #equalityExp
    | expression bop='&&' expression                           #logicalAndExp
    | expression bop='||' expression                           #logicalOrExp
    | <assoc=right> expression bop='=' expression              #assignmentExp
    ;
```

Funkcje

Mogą przyjmować parametry, zwracają literał o pojedynczym typie danych lub krotki składające się z literałów o różnych typach danych.

```
function
    : FUNC IDENTIFIER '(' parameterList? ')' ('->' returnType)? block eos*
    ;

parameterList
    : parameter (',' parameter)*
    ;

parameter
    : IDENTIFIER ':' type
    ;

returnType
    : type
    | '(' type (',' type)* ')'
    ;

block
    : '{' NL? statement* '}' NL?
    ;
```

PRZYKŁAD

```
func add(a: int, b: int) -> int {
    return a + b
}

func addAndMultiply(a: int, b: int) -> (int, int) {
    return a + b, a * b
}
```

Krotki

Krotkę można uzyskać jedynie poprzez wywołanie funkcji. Można przypisać ją do zmiennej i dostać się do poszczególnych literałów za pomocą operacji ekstrakcji.

```
literalFunction
    : TO_STRING                #toStringFunc
    | TUPLE_EXTRACT            #tupleExtractFunc
    ;
```

```
TUPLE_EXTRACT:      '_' [1-9] Digit*;
```

PRZYKŁAD

```
var tuple = addAndMultiply(2, 3)
tuple._1                // == 5
tuple._2                // == 6
```

Metody sterujące urządzeniem

W gramatyce języka zdefiniowane są 2 grupy metod:

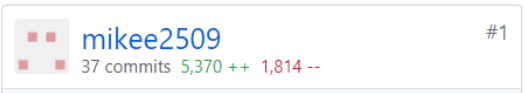
1. służące do sterowania funkcjami urządzenia mobilnego
2. służące do planowania wydarzeń tj. wywoływania przekazanych funkcji

```
// Built-in functions
RING:               'Ring';
SPEAK:              'Speak';
VIBRATE:            'Vibrate';
NOTIFY:             'Notify';
CALL:               'Call';
LAUNCH:             'Launch';
ADD_TO_CALENDAR:    'AddToCalendar';
SET_RINGER_VOLUME:  'SetRingerVolume';
SET_MEDIA_VOLUME:   'SetMediaVolume';
SET_ALARM_CLOCK:    'SetAlarmClock';
SET_WIFI:           'SetWifi';
SET_FLASHLIGHT:     'SetFlashlight';
SET_BRIGHTNESS:     'SetBrightness';

// Schedulers
ON_INTERVAL:        'OnInterval';
ON_TIME:            'OnTime';
ON_LOCATION:        'OnLocation';
ON_MESSAGE:         'OnMessage';
ON_WIFI_ENABLED:    'OnWifiEnabled';
ON_WIFI_DISABLED:   'OnWifiDisabled';
```

Niestety ze względu na ograniczoną ilość czasu jaką mogłem poświęcić na projekt oraz stopień skomplikowania implementowanej gramatyki, udało mi się zaimplementować tylko 2 przykładowe metody po jednej z obu kategorii.

```
// Invoke callback every 5 seconds, delay start for 2 minutes
OnInterval(callback, duration(5), duration(0, 2))


// Print string representation of value to log
Speak(value)
```

Pracochłonność implementacji języka oddaje statystyka dodanych i usuniętych linii kodu w trakcie projektu (wykluczając klasy generowane automatycznie):



Przykładowy skrypt

```
/* Print triangles of odd height */

var globalHeight = 1
OnInterval(action, duration(1), duration())

func action() {
    if (globalHeight % 2 == 0) {
        Speak("skipping even height = " + globalHeight)
    } else {
        printTriangle(globalHeight)
    }
    ++globalHeight
}

func printTriangle(height: int) {
    var line : string
    for (var i = 1; i <= height; ++i) {
        line = ""
        for (var j = 0; j < i; ++j) {
            line = line + "*"
        }
        Speak(line)
    }
}
```

Opis realizacji

Projekt został zrealizowany w języku Java (wersja 8) z wykorzystaniem narzędzia ANTLR. Do wstrzykiwania zależności wykorzystałem Spring framework. Testy oparłem o asercje dostarczone przez AssertJ.

Kod podzielony jest na dwie paczki:

- domain – klasy modelu, wyjątków
- parser – klasy „visitorów” interpretujące parsowany kod

Testy

Kod powstał w trybie TDD w wyniku czego powstało 49 testów jednostkowych.

Średnie pokrycie linii kodu w zaimplementowanych przeze mnie klasach wynosi aż 94%.

Dokładny raport z analizy pokrycia znajduje się w katalogu /coverage-report