Article
Vitaliy Serdtsev · Dec 8, 2017 6m read

Looking for the equivalent of first-class functions in the Caché DBMS

> First-class function[wiki]
> In computer science, a programming language is said to have first-class functions if it treats functions as first-class citizens. This means the language supports passing functions as arguments to other functions, returning them as the values from other functions, and assigning them to variables or storing them in data structures. Some programming language theorists require support for anonymous functions (function literals) as well. In languages with first-class functions, the names of functions do not have any special status; they are treated like ordinary variables with a function type.

This post continues the article " Declarative development in Caché "

```
[2, 3, 5, 7, 11, 13, 17].forEach(function(i) {
  console.log(i);
});
```

How to do something like this in Caché using COS?

Below are some exercises on this topic.

The **test.forEach** class intended for working with collection lists was developed to see how COS – one of the server languages integrated into Caché — can be used to write laconic, easy-to-read and flexible code.

The code of the class contained some COS capabilities:

- Args... (passing of an arbitrary number of arguments to a method/procedure/program);
- XECUTE or **$XECUTE** (execution of arbitrary COS commands);
- $COMPILE (compilation/syntax check);
- $CLASSMETHOD (execution of an arbitrary class method with an arbitrary number of arguments);
- $METHOD (execution of an arbitrary method of a class instance with an arbitrary number of arguments);
- Built-in class library.

> **Attention!** The examples below assume that Undefined=2.

This mode can be enabled in the terminal.

```
USER>set old=$system.Process.Undefined(2)
```

Run the tests and don ' t forget to put everything back in its place afterward

```
USER>do $system.Process.Undefined(old)
```

Let ' s go!

Class and example source codes.

Initialization of a ForEach object

Initialization is performed with the help of the test.forEach.%New(val,sep) method.

The first parameter "**val**" accepts a collection of literals, or a list, or a collection of objects.

The second parameter "**sep**" is a delimiter for a collection of literals.

1. Initialization of a collection of literals

```
USER>set tmp=##class(test.forEach).%New("2,3,5")
or
USER>set tmp=##class(test.forEach).%New($listbuild(2,3,5))
```

2. Initialization of a collection of literals with a random delimiter For example, with ";" acting as a delimiter

```
USER>set tmp=##class(test.forEach).%New("2;zxc;5;asd,ert",";")
or
USER>set tmp=##class(test.forEach).%New($listbuild(2,"zxc",5,"asd,ert"))
```

3. Initialization of a list of objects

```
USER>set list=##class(%ListOfObjects).%New()
USER>for i="f1","f2","f3",7 do list.Insert(##class(test.myclass).%New(i))
USER>set tmp=##class(test.forEach).%New(list)
```

> **Attention!** The test.forEach class in the %New method expects a collection instantiated from %Collection.AbstractList

Use cases

The **test.myclass** class has several methods that we will be calling for each element of a collection.

For example, *Dump* – it shows information about an element and the passed parameters, *Sum* — sums up the arguments and shows the result.

An example with a collection of numbers

Initializing the collection:

```
USER>set tmp=##class(test.forEach).%New("2,3,5")
```

Let ' s run the following in the terminal:

```
USER>do tmp.Do("test.myclass:Dump")
Element = 2, Output argument = "", Additional arguments = ""
Element = 3, Output argument = "", Additional arguments = ""
Element = 5, Output argument = "", Additional arguments = ""

USER>set r="result" do tmp.Do("test.myclass:Dump",.r,"p1","p2")
Element = 2, Output argument = "result", Additional arguments = "p1,p2"
```

```
Element = 3, Output argument = "result", Additional arguments = "p1,p2"
Element = 5, Output argument = "result", Additional arguments = "p1,p2"
```

Other examples with numbers

```
USER>kill r do tmp.Do("test.myclass:Sum",.r) write r
10

USER>kill r do $system.OBJ.DisplayError(tmp.Do("test.myclass:Sum",.r,5))
ERROR #5001: There was an error in element: 5

USER>do $system.OBJ.DisplayError(tmp.Do("PrintLn"))
ERROR #5654: Method '2:PrintLn' does not exist

USER>do $system.OBJ.DisplayError(tmp.Do("test.myclass:PrintLn"))
ERROR #5001: Method PrintLn is not a method of class test.myclass

USER>set r=10 do tmp.Do(,.r) write r
20 ; 10 +2+3+5

USER>kill r do tmp.Do(,.r) write r
10 ; 2+3+5

USER>set r=-10 do tmp.Do("+",.r) write r
0 ; -10 +2+3+5

USER>set r=1 do tmp.Do("*",.r) write r
30 ; 2*3*5

USER>kill r do tmp.Do("_",.r,"^") write r
^2^3^5 ; concatenation with the delimiter

USER>do tmp.Do("min",.r) write r
2 ; min

USER>do tmp.Do("max",.r) write r
5 ; max

USER>do tmp.Do("avg",.r) write r
3.333333333333333334 ; avg, (2+3+5)/3

USER>kill r do tmp.Do($listbuild("set args(1,1)=args(1,1)+el"),.r) write r
10 ; 2+3+5

USER>set r="r" do tmp.Do($listbuild("do sub^prog(el,args...)"),.r,"p1","p2")
--------
el = 2
args=1
args(1)=3
args(1,1)="r"
args(1,2)="p1"
args(1,3)="p2"

--------
el = 3
args=1
args(1)=3
```

```
args(1,1)="r"
args(1,2)="p1"
args(1,3)="p2"

--------
el = 5
args=1
args(1)=3
args(1,1)="r"
args(1,2)="p1"
args(1,3)="p2"

USER>set r="r" do $system.OBJ.DisplayError(tmp.Do($listbuild(
"do1 sub^prog(el,args...)"),.r,"p1","p2"))
ERROR #5745: Compile Failed!
```

Use case for a collection of objects

Initialization:

```
USER>set list=##class(%ListOfObjects).%New()
USER>for i="f1","f2","f3" do list.Insert(##class(test.myclass).%New(i))
USER>set tmp=##class(test.forEach).%New(list)
```

Check the result in the terminal:

```
USER>do tmp.Do("test.myclass:Dump")
Element = "f1", Output argument = "", Additional arguments = ""
Element = "f2", Output argument = "", Additional arguments = ""
Element = "f3", Output argument = "", Additional arguments = ""

USER>do tmp.Do("PrintLn")
"f1"
"f2"
"f3"

USER>do tmp.Do("PrintLn",,"Element = ")
Element = "f1"
Element = "f2"
Element = "f3"

USER>kill r do tmp.Do("Concat",.r,"**") write r
**f1**f2**f3

USER>kill r do $system.OBJ.DisplayError(tmp.Do("Concat",.r,"f3"))
ERROR #5001: There was an error in element: f3

USER>do $system.OBJ.DisplayError(tmp.Do("PrintLn1"))
ERROR #5654: Method 'test.myclass:PrintLn1' does not exist

USER>do $system.OBJ.DisplayError(tmp.Do("Sum",.r))
ERROR #5001: Method Sum is not a method of an instance of class test.myclass

USER>do tmp.Do("SetField",,"blablabla"), tmp.Do("PrintLn",,"Element = ")
Element = "blablabla"
Element = "blablabla"
```

```
Element = "blablabla"

USER>do tmp.Do($listbuild("do el.PrintLn(.args)"))
"blablabla"
"blablabla"
"blablabla"

USER>do tmp.Do($listbuild("write ""field="",el.field,!"))
field=blablabla
field=blablabla
field=blablabla
```

We have left without attention a few other collection types, for example arrays, globals, tables, threads. But at least you know “ how it works ” now...

This is a translation of an old article. Thanks @Evgeny Shvarov for the help in translation.

#API #Object Data Model #ObjectScript #Caché

Source URL:https://community.intersystems.com/post/looking-equivalent-first-class-functions-cach%C3%A9-dbms