Groovy Closure

Madhusudhanan.P.K

Topics

- What is and why closure?
- Method vs. Closure
- Closure as first-class objects
- Closure parameters & usage of parentheses
- Closure scope
- Closure usage areas

What is and Why Closure?

What is a Closure?

- A closure is a chunk of code within { ..}
- A closure behaves like a first-class object (just like String or Integer object)
  - It can be assigned to a variable (in the same way a String object can be assigned to a variable)
  - It can be passed around as a parameter of a method (in the same way a String object can be passed around as a parameter)
  - It can be a return value (in the same way a String object can be a return value)

Why Closure?

- More powerful and easier to use than Java anonymous inner class
- Enables simpler programming
  - 5.times {println "Hello world"}
  - [“Apple”, “Orange”, “Banana”].each {print it}
  - names.findAll { it.size() <= 3 } // names is an array of names
- There are certain things only closure can do
  - Higher order function (function that takes a function as an argument or returns a function)

Method vs. Closure

Method vs. Closure

- Similarities
  - Both contain chunk of code
  - Both are invoked in the same way
    - my_method(..)
    - my_closure(..) // my_closure.call() is also allowed
  - Both can receive parameters
  - In both, the last expression is the return value
- Differences
  - A closure is an object while a method is not - Closure is defined (becomes a closure object) only when you "bind" (assign) the block of code of {..} to a variable or pass as a parameter
  - The syntax through which they receive parameters are different
  - A closure takes a single parameter as “it”

Closure as First-class Objects

Closure as a First-class object #1

- Closure object can be assigned to a variable (just like String object can be assigned to a variable)

```
// A closure is defined and then assigned to a variable
def my_variable = { println "hello!" }
// Call closure
my_variable.call()        // => "hello!"
my_variable()             // => "hello!"

// By default closures take single parameter called "it"
def my_variable2 = { println it }
// Call closure
my_variable2.call("hello!")  // => "hello!"
my_variable2("hello!")  // => "hello!"
```

Closure as a First-class Object #2

- Closure object can be passed as a parameter (just like String object can be passed as a parameter)

```
// Define a target method that receives a parameter
def greetWithClosure1(closureAsParamater){
    closureAsParamater("Hello")
}

// Pass a closure as a parameter - the following work the same.
// If the closure is the last parameter, parentheses can be omitted
greetWithClosure1({println it})  // => Hello
greetWithClosure1 {println it}   // => Hello

def myClosure = {println it}
greetWithClosure1(myClosure) // => Hello
greetWithClosure1 myClosure  // => Hello
```

Closure as a First-class Object #3

- Closure object can be returned as a return value (just like String object can be returned as a return value)

```
// Define a method which returns a closure
def aMethod(String name){
    return {println "${name}"}
}

// A closure is returned and then is assigned to a variable
def aClosure = aMethod("Maxx Krish")

// Call the closure
aClosure()  // => Maxx
Krish
```

Closure Parameters & Usage of Parentheses

Closure Parameters

- By default closures take 1 default parameter called "it"

```
def square = { it * it }
println square(5)        // => 25
```

- You can also create closures with named parameters with ->

```
def square = { num -> num * num }
println square(5)        // => 25

def add = { a, b -> a+b }
println add( 5, 7 )      // => 12

printMapClosure = { key, value -> println key + "=" + value }
[ "Yue" : "Wu", "Mark" : "Williams" ].each(printMapClosure)
```

Parentheses when calling a closure

- When calling a closure (or a method) with parameters, the parameters do not need to be enclosed with parentheses( )

```
// Define a method
def index(parameter1, parameter2) {
  println "I am a method receiving ${parameter1} and ${parameter2}"
}
// Call the method
index "Maxx Krish",
11
// Define a closure and assign it to a variable
def index = {
  parameter1, parameter2 -> println "I am a closure receiving ${parameter1} and $
{parameter2}"
}
// Call the closure
index "Yo man", 22
index.call "Yo man", 22
```

Parentheses when closure is passed as a parameter

- If a closure is the last parameter or only parameter, there is no need to enclose the parameters with ( )
- If a closure is not the last parameter, the parameters need to be enclosed with ( )

```
// Define a method that takes a closure as a paramater
def greetWithClosure(greeting, name, closure){
    println "${greeting}, ${name}"
    closure(new Date())
}

// The following three work the same.  If the closure is
// the last parameter, there is not need to enclose the parameters with ( )
greetWithClosure("Goodbye", "Shelley", {println it})
greetWithClosure "Goodbye", "Shelley", {println it}
greetWithClosure("Goodbye", "Shelley") {println it}
```

Closure Scope

Closure Scope

- Closures can access variables defined in the same scope as the closure itself

```
// A closure object can access variables
// in the same scope when it gets created, so it can access
// “name” variable because it is defined in the same scope
def name = 'Maxx Krish'
def my_closure_variable = { println "hello, ${name}!" }

// Call closure and note that it can access name variable
my_closure_variable()        // => hello, Maxx Krish!

// Change the value of name variable
name = 'Bill'
my_closure_variable()        // => hello, Bill!
```

Method Closure Operator

Method Closure Operator

- A method can be converted to a closure using &. It is called method closure operator

```
def list = [ "apple", "orange", "banana"]

println "----- \"each\" takes a closure as an argument"
list.each {println it}

println "----- Define \"printSomething\" method"
String printSomething (String something){
    println something
}

println "----- Convert \"printSomething\" method into a closure using Method Closure Operator"
//list.each(printSomething) // Exception
list.each(this.&printSomething)
list.each this.&printSomething   //Without parentheses
```

Closure usage areas

Closure is used everywhere..

- Iterators
- Callbacks
- Specialized control structures
- Higher order functions (Functions that take/return functions)
- Dynamic method definition
- Resource allocation
- Threads
- Continuations

Closure Simplifies Coding

- *Number* class has *times* method which takes a closure as a parameter
  - 5.times {println "Hello world"}

Thank you!!!
maxx@zvarad.com