Groovy Meta-Programming (Meta Object Protocol - MOP)

Madhusudhanan.P.K.

Topics

- What is and why meta-programming?
- Adding behavior during runtime using Expando class
- Adding behavior during runtime using ExpandoMetaClass
- Check method/property availability
- Dynamic method invocation
- Meta-programming hooks in Groovy
  - > Intercepting calls and accesses to existing methods and properties
  - > Intercepting calls and accesses to missing methods and properties
- Domain Specific Language (DSL)

What is & Why Meta-Programming (Meta Object Protocol)?

What is Meta-Programming?

- Meta-programming is the writing of computer programs that write or manipulate other programs (or themselves) as their data

Why Meta-Programming?

- Provides higher-level abstraction of logic
  - Easier to write code
  - Easier to read code
- Meta-programming feature of Groovy language makes it an excellent Domain Specific Language (DSL)

Adding Behavior during Runtime via Expando

Expando Class

- Create dynamically expandable bean via Expando class

```
println "----- Create a new Expando object"
def dog = new Expando()

println "----- Add properties to it during runtime"
dog.name = "My dog"
dog.greeting = "Hello"

println "----- Add behavior to it using closure during runtime"
dog.bark = {
   println "${name} says ${greeting}"
}

println "----- Let my dog say hello"
dog.bark()
```

Adding Behavior during Runtime via ExpandoMetaClass

ExpandoMetaClass class

- Groovy 1.1 includes a special MetaClass called an *ExpandoMetaClass* that allows you to dynamically add methods, constructors, properties and static methods using a neat closure syntax
  - ExpandoMetaClass is a MetaClass that behaves like an Expando, allowing the addition or replacement of methods, properties and constructors on the fly
- Every *java.lang.Class* is supplied with a special "*metaClass*" property that when used will give you a reference to an *ExpandoMetaClass* instance
- You can extend any class with new behavior

Example #1

- Add a behavior to the Dog class during runtime

```
println "----- Define Dog class"
class Dog{
}

println "----- Add bark() behavior to the Dog class"
Dog.metaClass.bark = {
    X -> println "${X} is barking!"
}

println "----- Call newly added metaClass method"
new Dog().bark("My dog")
```

Example #2

- Add a behavior to the String class during runtime (despite String is final class in Java)

```
// Add capitalize() metaClass method to the String class
String.metaClass.capitalize = {
    delegate[0].toUpperCase() +
    delegate[1..<(delegate.length())].toLowerCase()
}

// Call newly added metaClass method for String objects
println "abc".capitalize()      // "Abc"
println "ABC".capitalize()      // "Abc"
```

Check Method/Property Availability

Check Method/Property Availability

- java.util.List<MetaMethod> respondsTo(java.lang.Object obj, java.lang.String methodName)
  - Check if “methodName” method is available in the specified object
- java.util.List<MetaMethod> respondsTo(java.lang.Object obj, java.lang.String methodName, java.lang.Object[] argTypes)
  - Check if “methodName” method with “argTypes” arugment array is available in the specified object
- MetaProperty hasProperty(java.lang.Object obj, java.lang.String propertyName)
  - Check if “propertyName” property is available in the specified object

Dynamic Method Invocation

Dynamic Method Invocation

- You can invoke a method even if you don't know the method name until it is invoked:

```
class Dog {
  def bark() { println "woof!" }
  def sit() { println "(sitting)" }
  def jump() { println "boing!" }
}

def doAction( animal, action ) {
  animal."$action"()                 //action name is passed at invocation
}

def rex = new Dog()

doAction( rex, "bark" )              //prints 'woof!'
doAction( rex, "jump" )              //prints 'boing!'
```

Meta-Programming Hooks in Groovy: Intercepting Calls and Access to Existing Methods & Properties

Meta Programming Hooks

- invokeMethod
  - Intercept calls to existing methods
- get/setProperty
  - Intercept access to existing properties
- methodMissing
  - Intercept calls to missing methods
- propertyMissing
  - Intercept access to missing properties

invokeMethod – Enables AOP

```
// Usage of invokeMethod is to provide simple AOP style around advice to existing methods
class MyClass implements GroovyInterceptable {

   def sayHello(name){
     "Hello, ${name}"
   }

   def invokeMethod(String name, args) {
      System.out.println ("Beginning $name")
      def metaMethod = metaClass.getMetaMethod(name, args)
      def result = metaMethod.invoke(this, args)
      System.out.println ("Completed $name")
      return result
   }

}

myObj = new MyClass()
myObj.sayHello("Sang Shin")
```

invokeMethod – Enables DSL/Builder

```
// Usage of invokeMethod is to build a simple
// XML builder
class XmlBuilder {
   def out
   XmlBuilder(out) { this.out = out }
   def invokeMethod(String name, args) {
      out << "<$name>"
      if(args[0] instanceof Closure) {
           args[0].delegate = this
           args[0].call()
      }
      else {
          out << args[0].toString()
      }
      out << "</$name>"
   }
}
```

```
def xml = new XmlBuilder(new StringBuffer())
xml.html {
   head {
      title "Hello World"
   }
   body {
      p "Welcome!"
   }
}
```

Meta-Programming Hooks in Groovy: Intercepting Calls and Access to Missing Methods & Properties

methodMissing

- You can intercept a missing method and then add the desired behavior on the fly
  - This is how you can create your own methods during runtime
- Enables Domain Specific Language (DSL)
- This how Grails GORM supports
  - findByYourBirthPlace()
  - findByMyOwnSomething()

Example: methodMissing in GORM

- Dynamic finders in GORM uses methodMissing

```
class GORM {

   def dynamicMethods = [...] // an array of dynamic methods that use regex
   def methodMissing(String name, args) {
       def method = dynamicMethods.find { it.match(name) }
       if(method) {
          GORM.metaClass."$name" = { Object[] varArgs ->
             method.invoke(delegate, name, varArgs)
          }
          return method.invoke(delegate,name, args)
       }
       else throw new MissingMethodException(name, delegate, args)
   }
}
```

Example: methodMissing

```
import java.text.NumberFormat
def exchangeRates = ['GBP':0.501882, 'EUR':0.630159,
                     'CAD':1.0127, 'JPY':105.87] // (7/2/2008)

BigDecimal.metaClass.methodMissing = { String methodName, args ->
    conversionType = methodName[2..-1]
    conversionRate = exchangeRates[conversionType]

    if(conversionRate){
        NumberFormat nf = NumberFormat.getCurrencyInstance(Locale.US)
        nf.setCurrency(Currency.getInstance(conversionType))

        return nf.format(delegate * conversionRate)
    }
    "No conversion for USD to ${conversionType}"
}

println 2500.00.inGBP()
println 2500.00.inJPY()
println 2500.00.inXYZ()
```

Domain-Specific Language (DSL)

What is DSL?

- Martin Fowler defines a DSL as a "computer programming language focused on a particular domain."
- A DSL is a tiny specific-purpose language, in contrast to a large general-purpose language like the Java language
- Dave Thomas describes DSL as "a specialized language that domain experts invented as a shorthand for communicating effectively with their peers."
- Examples of DSL
  - SQL

Groovy Features That Enables DSL

- Meta-programing feature
  - You can add arbitrary methods and properties to any class
- Operator overloading
- Builder pattern

Thank you!

maxx@zvarad.com