Scripting on the JVM – Part I

Venkat Subramaniam
venkats@agiledeveloper.com

Abstract

You can take advantage of scripting or dynamic languages on the JVM. In this multi part article, we will take a look at how to mix Java and the scripting languages. We will use JavaScript and Groovy as examples, but you can use about two dozen languages on the JVM using similar approach. In this first part we will look at running JavaScript on the JVM.

Scripting Support

JSR-223[1] and the Scripting API[2] allow intermixing scripting languages and Java. The standard API provides a consistent way for you to use over two dozen languages. Let's begin by looking at a simple example. You can download and use jsr-223 Engine separately with Java 5. Java 6 has the scripting API bundled along with the JavaScript Rhino Engine. However, if you want to mix Groovy, JRuby, or other languages, you will still need the jsr-233 engine.

jrunscript

jrunscript is a nice little experimentation tool that allows you to play with some script on the JVM. Let's give it a try. On the command line, I type jrunscript and I get a response

```
js>
```

The tool tells us it is ready to run JavaScript. Let's quit by typing `quit()`.

```
js>quit()
>
```

Now try `jrunscript -q`:

```
>jrunscript -q
Language ECMAScript 1.6 implemention "Mozilla Rhino" 1.6
release 2

>
```

The tool recognizes JavaScript by default. Let's tweak that a bit.

```
>jrunscript -classpath %GROOVYSCRIPTPATH% -q
Language groovy 1.0 implemention "groovy" 1.0
Language ECMAScript 1.6 implemention "Mozilla Rhino" 1.6
release 2

>
```

Now, jrunscript is ready to recognize two languages: JavaScript and Groovy. What's that GROOVYSCRIPTPATH? I have set that to include two jars: the jsr-223-engines\groovy\build\groovy-engine.jar and the groovy-1.0\embeddable\groovy-all-1.0.jar.

To use Groovy instead of the default JavaScript, you can use the `-l` option as shown below:

```
>jrunscript -classpath %GROOVYSCRIPTPATH% -l groovy
groovy> println 'Hello Groovy!'
Hello Groovy!
groovy>
```

Let's get back to running some JavaScript. Start jruncript without any command line options and type some JavaScript in it as shown:

```
>jrunscript
js>
js> function Car() { this.miles = 0; }
js>
js> Car.prototype.drive = function(dist) { this.miles +=
dist; }
sun.org.mozilla.javascript.internal.InterpretedFunction@555
71e
js>
js> mycar = new Car();
[object Object]
js>
js> println(mycar.miles);
0
js>
js> mycar.drive(10);
js> println(mycar.miles);
10
js>
```

We created a function (class) `Car`. Added a method `drive()` to it. We then created an instance, and called the method on it. This is regular JavaScript, so what's the big deal. Well, this JavaScript is special–it's running on the JVM. You can call into Java if you like:

```
js> lst.add(1);
true
js> lst.add(2);
true
js> lst.add(5);
```

```
true
js> println (lst.size());
3
js> println (lst.getClass().getName());
java.util.ArrayList
js>
```

You can use as much Java as you like from JavaScript. What's better is that you're using not the same Java syntax, but the JavaScript syntax (or Groovy syntax, etc.)

What if you want to go beyond a few lines of code? You can type your code in a file and then run jrunscript with the file name as parameter. Let's create a list.js file as shown below:

```
importClass(java.util.ArrayList);

lst = new ArrayList();

with(lst)
{
  add(1);
  add(4);

  println (size());
}
```

That's JavaScript syntax to talk to Java `ArrayList`. Let's run it:

```
>jrunscript list.js
2
>
```

Let's write a little Swing application using JavaScript. We can see the idiomatic differences of mixing languages.

```
packages = new JavaImporter(javax.swing, java.awt,
java.awt.event);

with(packages)
{
  var frame = JFrame("Swing");
  frame.setSize(200, 100);
  var button = JButton("Click me");
  var label = JLabel("test");

  frame.getContentPane().add(button);
  frame.getContentPane().add(label);
```

```
  frame.getContentPane().setLayout(new FlowLayout());

  button.addActionListener(
   function() { label.setText("Script!"); }
     );

  frame.setVisible(true);

  java.lang.Thread.sleep(10000);
}
```

The `JavaImporter` allows us to import packages into a namespace (you may also use the `importPackage` or `importClass`). We create a Swing `JFrame`, create a button and label in it. The interesting part is the registration of event handler. In Java you would have created an inner class to implement `ActionListener` interface. Then you would have written a method `actionPerformed(ActionEvent ae)`. You don't need to do any of that here–it's smart enough to pick up the function you provide and implement the desired interface. We invoke `sleep()` to keep the thread alive (instead you can write a handler for close event).

The execution of this program (before and after clicking the button) is shown below:

Conclusion

In this article we took a preliminary look at calling Java from scripting languages using jrunscript. It also showed us how you can enjoy the idiomatic difference of JavaScript while utilizing the strengths of the Java API.

References

1. http://jcp.org/en/jsr/detail?id=223
2. https://scripting.dev.java.net