Scripting on the JVM – Part II

Venkat Subramaniam
venkats@agiledeveloper.com

Abstract

In Part I we looked at using jrunscript to call from JavaScript to Java. In this Part II we'll look at calling JavaScript from within Java. We will call into a script from Java, pass parameters, and also get result back from the script.

Scripting API

If you like to call into one of the two dozen scripting languages from within Java, you can use the Scripting API[1,2]. The scripting API provides a standard interface (as defined in JSR 223) for Java applications to interact with scripting engines.

A script engine takes care of mapping calls from Java to the underlying methods in the script you provide. Let's start with an example.

Get ScriptEngine

To use the scripting engine, you interact with an instance of `ScriptEngine`. You can obtain an instance of `ScriptEngine` from `ScriptEngineManager`. You can give it the name of the languages you're interested in or the file extension used in that language.

```
ScriptEngineManager scriptEngineManager =
                  new ScriptEngineManager();
ScriptEngine engine1 =
                  scriptEngineManager.getEngineByName("javascript");
System.out.println(engine1);

ScriptEngine engine2 = scriptEngineManager.getEngineByExtension("js");
System.out.println(engine2);
```

The above code reports:

```
com.sun.script.javascript.RhinoScriptEngine@32fb4f
com.sun.script.javascript.RhinoScriptEngine@1113708
```

The Rhino Engine for JavaScript is already part of Java 6. If you're using Java 6 and want to use JavaScript you don't need anything more. If you're using Java 5 or want to use other languages with Java 6, you need to download JSR 223 Engine.

Call JavaScript

To call JavaScript, you can use the `ScriptEngine` to `evaluate` your script. In the example below, I will hard code the script. However, in a real application I may get the script from a file or some form of input at runtime.

```
ScriptEngineManager scriptEngineManager = new ScriptEngineManager();
ScriptEngine engine =
            scriptEngineManager.getEngineByName("javascript");
```

```
try
{
    engine.eval("println('Hello from JavaScript');");
}
catch (ScriptException e)
{
    System.out.println(e);
}
```

The output from the above code is:

```
Hello from JavaScript
```

Bindings

If you want to pass some data to the script, you can do that easily using the Bindings. You can create any number of Bindings you like or you can use the one attached to the engine already. Let's explore this further.

```
engine.put("name", "Bill");
engine.eval("println('Hello ' + name + ' from JavaScript');");

engine.put("name", "Vivek");
engine.eval("println('Hello ' + name + ' from JavaScript');");
```

The output from the above code is

```
Hello Bill from JavaScript
Hello Vivek from JavaScript
```

You can also create separate bindings and use them for different evaluations.

```
Bindings bindings1 = engine.createBindings();
bindings1.put("name", "Bill");

Bindings bindings2 = engine.createBindings();
bindings2.put("name", "Vivek");

engine.eval("println('Hello ' + name);", bindings1);
engine.eval("println('Hello ' + name);", bindings2);
engine.eval("println('Hello ' + name);", bindings1);
```

The output from the above code is:

```
Hello Bill
Hello Vivek
Hello Bill
```

Return Objects

You can also receive the response from the script.

```
Bindings bindings1 = engine.createBindings();
bindings1.put("name", "Bill");
```

```
Object result = engine.eval("name + ' today is ' + new Date();",
                              bindings1);

System.out.println(result);
```

The output from the above code is shown below:

```
Bill today is Mon Dec 31 2007 04:43:11 GMT-0700 (MST)
```

Conclusion

In this Part-II we saw how to invoke JavaScript from within your Java code. We have merely scratched the surface. We will look at invoking methods and functions in Part III.

References

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