Scripting for the Java™ Platform

Mike Grogan
A. Sundararajan

Staff Engineers
Sun Microsystems

TS-1382

Goal of the Talk

Explain and demonstrate the Scripting support in Java™ SE 6 (code-named "Mustang")

Agenda

Scripting for the Java™ Platform
Scripting API
java.net Scripting Project
Scripting in Mustang (Java SE 6)
JavaScript-to-Java Communication
Development Tools, Future
Demo, Q&A

Scripting for the Java Platform

- "Java" technology is both the language and the platform
  - The platform includes Java VM and JDK™ APIs
  - Language choice—dynamically typed, scripting languages as well
- JSR 223—Scripting for the Java Platform
- Pluggable framework for third-party script engines
- javax.script package
- Optional javax.script.http package ("web scripting")

Scripting API

Scripting API

- Script Engine
- ScriptContext, Bindings
- ScriptEngineFactory
- ScriptEngineManager

ScriptEngine

- ScriptEngine interface—required
  - Execute scripts—“eval” methods
  - Map Java objects to script variables (“put” method)
- Invocable interface—optional
  - Invoke script functions/methods
  - Implement Java interface using script functions/methods
- Compilable interface—optional
  - Compile Script to intermediate form
  - Execute multiple times without recompilation

ScriptEngineManager

- Concrete class
- ScriptEngine discovery
  - Uses Services API
  - Thread context class loader
- ScriptEngine discovery
  - By name, extension, MIME type
- Explicit ClassLoader-based discovery as well
- Global scope Bindings visible to all ScriptEngines

Sample Code—“Hello World”

```
import javax.script.*;

public class Main {
    public static void main(String[] args) throws ScriptException {
        // create a script engine manager
        ScriptEngineManager factory = new ScriptEngineManager();
        // create JavaScript engine
        ScriptEngine engine = factory.getEngineByName("JavaScript");
        // evaluate JavaScript code from String
        engine.eval("print('hello world')");
    }
}
```

Sample Code—“eval” Script File

```
// create script engine manager
ScriptEngineManager manager = new ScriptEngineManager();

// create JavaScript engine
ScriptEngine engine = manager.getEngineByExtension(“.js”);

// evaluate a file (or any java.io.Reader)
engine.eval(new FileReader(“test.js”));
```

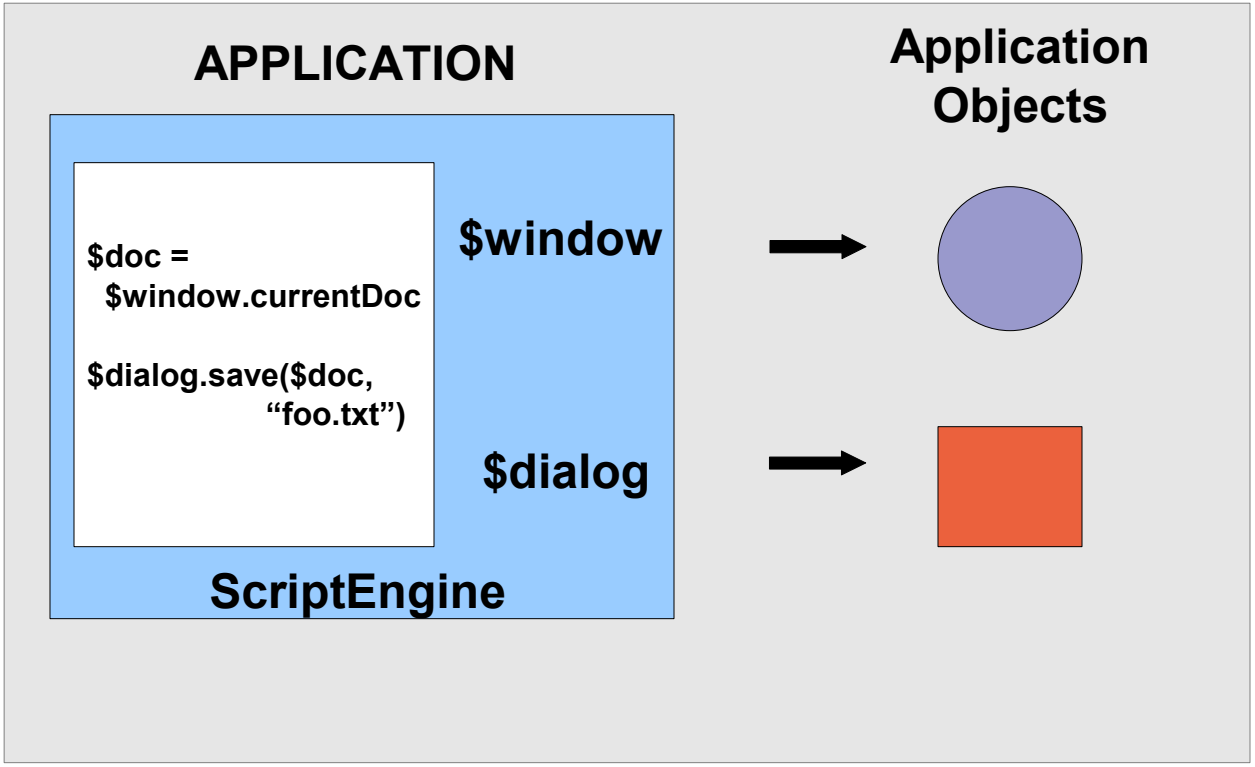
Sample Code—Invoking Functions

```
// JavaScript code in a String
String script = "function hello(name) { print('Hello, ' + name); }";
// evaluate script
engine.eval(script);

// JavaScript engine implements Invocable interface
Invocable inv = (Invocable) engine;

// invoke a global function called “hello”
inv.invoke("hello", new Object[] {"Scripting!!"} );
```

Mapping Script Variables to Application Objects



ScriptContext, Bindings

- ScriptContext—Script's view of host application
- ScriptContext contains one or more Bindings
- Bindings is subtype of Map<String, Object>
- Engine Scope Bindings
  - Script variables → application objects
- Global Scope Bindings
  - Variables shared across engines
- Writers for stdout, stderr
- Reader for stdin

Sample Code—Script Variables

```
// create script engine manager
ScriptEngineManager manager = new ScriptEngineManager();

// create JavaScript engine
ScriptEngine engine = manager.getEngineByName("JavaScript");
File f = new File("test.txt");

// expose File object as variable to script
engine.put("file", f);

// evaluate a script string
// script accesses "file" variable and calls method on it
engine.eval("print(file.getAbsolutePath())");
```

ScriptEngineFactory

- 1-1 with ScriptEngines
- Factory method—getScriptEngine
- Metadata methods
  - Script file extensions, mimetypes
  - Implementation-specific behavior (threading)
- Script generation methods
  - Generate method call
  - Generate “print” call

Scripting java.net Project

Scripting java.net Project

- http://scripting.dev.java.net
- BSD license for script engine code
- Groovy, Jelly, JRuby, Jexl, JudoScript, OGNL, Pnuts, Jython, JRuby, Scheme, Sleep, Jacl, XPath, XSLT
- The list is expected to grow
- Intend to develop useful scripting application and demo code as well
- Please join, contribute, share, and use!

Scripting in Mustang (Java SE 6)

Scripting in Mustang

- javax.script included
- javax.script.http not included
- JavaScript technology reference engine
- Based on Mozilla Rhino Engine
- Few optional components of Rhino not included
- Command-line language-independent script shell **jrunscript** (JDK only)

Scripting in Mustang—Rhino

- Using Rhino version 1.6R2
- E4X (ECMA Standard 357) not included
- javax.script.Compilable implemented by storing JavaScript interpreter bytecodes
- JavaAdapter that supports implementation of a single interface
- Renamed Rhino classes—sun.org.mozilla*
- Can drop later version of Rhino—but can't use that through javax.script API

JavaScript Technology-to-Java Technology Communication

JavaScript Technology-to-Java Technology

- Import Java packages and classes
  - importPackage(java.awt);
  - importClass(java.awt.Frame);
- Create Java-based Objects by “new ClassName”
  - var frame = new java.awt.Frame(“hello”);
- Call Java public methods from script
  - frame.setVisible(true);
- Access “JavaBean” properties like “fields”
  - print(frame.title);

Accessing Java-Based Packages

- "Packages" global variable to access Java-based packages
- Examples
  - Packages.java.util.Vector
  - Packages.javax.swing.JFrame
- "java" is shortcut for "Packages.java"
- java.lang is not imported by default (unlike Java technology)
  - Conflict with JavaScript Object, Boolean, Math etc.
- Package and class names checked lazily

JavaImporter

JavaImporter to avoid polluting global namespace

```
var SwingGui = JavaImporter(Packages.javax.swing,
                            Packages.javax.swing.event,
                            Packages.javax.swing.border,
                            java.awt.event);
with (SwingGui) {
    var mybutton = new JButton(“test”);
    var myframe = new Jframe(“test”);
 }
```

Creating, Using Java-Based Arrays

- Need to use reflection

```
var a = java.lang.reflect.Array.newInstance(java.lang.String, 5);
```

- Element access and length access is normal

```
a[0] = “scripting is great!”;
print(a.length);
```

- In most cases, you can use JavaScript-based arrays
- Can pass a script array when a Java-based method expects a Java-based array (auto conversion)

Overload Resolution

- In most scenarios, you do not need to do anything—correct overload based on arg types is selected
- User can explicitly select particular variant :

```
var out = java.lang.System.out;
// select a particular println function
out["println(java.lang.Object)"]("hello");
```

- More details at

  http://www.mozilla.org/js/liveconnect/lc3_method_overloading.html

Implementing Java-Based Interface

- Implement Java-based interfaces—anonymous class-like

```
var x  = new java.lang.Runnable() {
        run: function() {
            print("running...\n")
        }
    };
```

- Implement Java-based Interfaces—explicit JavaAdapter

```
importPackage(java.awt.event);
var o = { actionPerformed: function(evt) { print("clicked"); } };
var listener = new JavaAdapter(ActionListener,o);
```

Implementing Java-Based Interface (Cont.)

- When an interface with single method is expected, you can pass a script function (auto conversion)
- Example:

```
function func() {
    print("I am func!");
}
// pass script function for java.lang.Runnable argument
var t = new java.lang.Thread(func);
t.start();
```

Development Tools, Future

Development Tools

- JavaScript-based editor for NetBeans™ 5.0
  - http://www.liguorien.org/jseditor/
- NetBeans JavaScript support soon
- Coyote Project
  - NetBeans module for Jython and Groovy
  - https://coyote.dev.java.net/
- JpyDbg
  - NetBeans module for Jython
  - http://jpydbg.sourceforge.net/

Future

- Groovy JSR 241
  - http://www.jcp.org/en/jsr/detail?id=241
- BeanShell JSR 274
  - http://www.jcp.org/en/jsr/detail?id=274
- invokedynamic bytecode JSR 292
  - http://www.jcp.org/en/jsr/detail?id=292
  - Used for better compilation of dynamically-typed scripts
  - Also to Investigate support for (flexible) hotswap

DEMO

jrunscript

jrunscript

- “Experimental” JDK software-only command line tool
- Scripting language independent shell
  - -l <language> : choose language, default is “JavaScript”
  - -e <script> : eval given script string
  - -f <file> : eval given script file , '-' means stdin
  - -cp <classpath> : Classpath for script engine and application classes
  - -q : list all script engines available
  - -Dfoo=bar : define a Java System property
  - Interactive mode when '-e' and '-f' are not used

jrunscript (Cont.)

- Built-in JavaScript programming language functions
  - File system utils—pwd, cd, rm, mkdir, mkdirs, find, grep
  - Net utils—'cat', 'cp' accept URLs
  - XML utils—XMLDocument, XSL Transform functions
  - Process utils—exec, exit
  - load—load script files, URLs and streams
- 'engine' global variable—can access current scriptengine object
- jrunscript <script> <script arguments>
  - Global 'arguments' array for script arguments

DEMO

jconsole script shell plugin

DEMO

SwiXML + Scripting

Q&A

Scripting for the Java™ Platform

Mike Grogan
A. Sundararajan

Staff Engineers
Sun Microsystems

TS-1382