Scripting in the Java™ Platform

Roberto Chinnici
Senior Staff Engineer
Sun Microsystems, Inc.
Agenda

- Introduction to scripting
- JSR-223 and more
- Various uses of scripting
  - Testing, prototyping, embedded, whole app, components, frameworks
- Project Phobos
- Demos
Scripting Languages

- Dynamic typing, no compilation/deployment
- Various degrees of object orientation
- Python, PHP, Perl, Ruby, etc.
- Moved upstream from “scripts” to applications
- Full stacks: LAMP
- Associated to agile practices
- Recent popularity boosters: AJAX, Rails
“Java™” is both the language and the platform

Language choice is real

- http://www.robert-tolksdorf.de/vmlanguages.html

Now considerably easier with JSR-223

- Standard API for scripting engines

Many scripting engines available

- https://scripting.dev.java.net

New JDK 7 features driven by scripting
Benefits

- Security model
- Threading
- State-of-the-art garbage collector
- Zillions of libraries
- Enterprise connectivity
- Server infrastructure
“Soft” vs “Hard” Layers

- Core principle
- Lots of examples in the industry
- Application code is “soft”
- Platform and extensions are “hard”
- Different qualities for the different layers
- Scripting (soft) complements Java™ (hard)
JSR-223 Overview

- Pluggable, discoverable scripting engines
- Automatic registration
  - Just drop new engines in the classpath
- Bidirectional communication to/from scripts
- Multiple contexts (a.k.a. scopes)
  - Global, engine, session, request, page, ...
- Engines advertise their capabilities
Quick Language Survey

- Jython
  - First high-profile scripting language on the JVM
- Beanshell
- Groovy
  - First new language with its own JSR
- JRuby
- JavaScript
  - Scripting engine included in JDK 6
Specialized Languages

- Ant
- Jelly
- XSLT
- Jess
- ...
Languages Driving Innovation

- **GJ, Pizza**
  - Generics (JDK 5)
- **Jython, JRuby, others**
  - Closures (JDK 7?), continuations (?)
- **JavaScript, E4X**
  - Direct XML support (JDK 7?)
- **AspectJ**
  - AOP (?, Spring, EJB 3 interceptors)
Common Features

- Dynamic and duck typing
- Closures
- Mixins and/or multiple inheritance
- Dynamic/extensible objects
- Lots of literals (list, maps, long strings)
- Interactive prompt (console, REPL)
- Java™ runtime connectivity
Scripting and NetBeans™

- Lots of plugins available
- JavaScript editor
  - http://www.liguorien.org/jseditor
- Coyote Project
  - Jython, Groovy modules
    - https://coyote.dev.java.net
- JpyDbg (Jython)
Testing/Prototyping

- Low barrier to entry
- Interactively test code from the command line
- Use language introspection facilities
- Different consoles for different languages
- Or use `jrunscript` from the JDK
Jrunscript Tool

- Command line tool (JDK only)
- Execute scripts in any language
  
  jrunscript -f test.js
  jrunscript -e "print('hi!');"
  jrunscript -l groovy -f myapp

- Can be used to list all available engines
  
  jrunscript -q

- Shell-like interactive mode
  
  jrunscript / jrunscript -f -
Example: Jython

```python
>>> import java.lang.String
>>> dir(java.lang.String)
['CASE_INSENSITIVE_ORDER', 'bytes', 'codePointAt',
'codePointBefore', 'codePointCount', 'compareTo',
'compareToIgnoreCase', 'concat', 'contains',
'contentEquals', 'copyValueOf', 'endsWith',
'equalsIgnoreCase', 'format', 'getBytes', 'getChars',
'indexOf', 'intern', 'lastIndexOf', 'matches',
'offsetByCodePoints', 'regionMatches', 'replace',
'replaceAll', 'replaceFirst', 'split', 'startsWith',
'substring', 'toCharArray', 'toLowerCase', 'toUpperCase',
'trim', 'valueOf']
```
Example: JavaScript

```javascript
// direct class references possible with "java" prefix
var frame = new java.awt.Frame("hello");
frame.visible = true;
print(frame.title);

// access any package using "Packages" prefix
var document =
    new Packages.org.jdom.input.SAXBuilder().
        build(new java.io.FileInputStream("feed.xml"));

var feed =
    Packages.com.sun.syndication.io.SyndFeedInput().
        build(document);
```
Example: JavaScript

// implementing a Java interface
var obj = { run: function () { print("\nrunning"); } }
var r = new java.langRunnable(obj);
var t = new java.lang.Thread(r)
t.start()

// short version
var t = new java.lang.Thread(function() {
    print("\nrunning");
});
t.start()
“Open” Development

- JRuby classes are open (unless frozen)
- All classes, including “system” ones

```ruby
class Fixnum
  def answer
    42
  end
end
```

- Groovy JDK does the same thing with JDK classes

```java
java.lang.String.findAll(Closure)
```
Unit Tests

- The kind of code you want to write quickly
- In test-first development, a large fraction of all the code you write
- Groovy has JUnit extension package
- Use `groovyc` to compile your tests to Java™ as part of the build process
import groovy.util.GroovyTestCase

class SimpleFilterTest extends GroovyTestCase {

    void testCreateReport() {
        def gen = new MyReportGenerator()
        gen.year = "2006"
        def result = gen.generate()
        // "parse" helper method not shown here
        def doc = parse(result)
        // Gpath expression
        def node = doc.sales.find { it['@region'] == 'WEST' }
        assertNotNull(node)
    }
}

Full Scripts

- Obvious, but worth repeating

- Lots of opportunities:
  - automate tasks
  - code generation
  - anything with templates
  - set up databases, etc.

- Make it a habit

- Natural way to develop a library of tools that work the way you do
import groovy.sql.Sql
import groovy.xml.MarkupBuilder

def sql = Sql.newInstance(...)

def xml = new MarkupBuilder()

def limit = '1984'
sql.eachRow("select * from customer where since < ${limit}") { customer |
    xml.customer(
        id: customer.id,
        since: customer.since,
        salutation: salutationOf(customer))
}
Scripts Caveat

- Many engines try to be drop-in replacement for “traditional” ones
  - Jython ↔ Python, JRuby ↔ Ruby
- But:
  - language version mismatches
  - some libraries don't work (e.g. they use C)
- Emulation an ongoing process
- May influence language specs (Python)
- Moral: surprises happen
Embedded Scripts

- “Soft” part of an application
  - Configuration
  - Extension points
  - Customization
  - Rules
  - Gluing components together
- Don't roll your own language
- Use an established one
- JSR-223 makes it easy
Key JSR-223 API Classes

- **ScriptEngineManager**
  - Entry point, manages engine registration

- **ScriptEngineFactory**

- **Script Engine**
  - Single engine with optional capabilities

- **ScriptContext**

- **CompiledScript**
ScriptEngine

- Basic engine functionality:
  - Execute scripts—“eval” methods
  - Map Java™ objects to script variables (“put” method)

- Invocable interface—optional
  - Invoke script functions/methods (macros)
  - Implement Java™ interface using script

- Compilable interface—optional
  - Compile once, execute multiple times
ScriptEngineManager

- Concrete class
- Engine discovery
  - Uses Services API
  - Thread context class loader
- Engine factory lookup
  - By name, extension, MIME type
- Explicit ClassLoader-based discovery as well
- Global scope visible to all engines
Application Configuration

- Long-standing solution: properties files
  
  ```
  com.acme.browser.connectionTimeout=5
  ```

- More recent solution: XML
  
  ```
  <acme:connection>
      <acme:timeout>5</acme:timeout>
  </acme:connection>
  ```

- Scripts are even better!
  
  ```
  connection.timeout = 5;
  ```
Benefits

- No custom XML parsing code, no data binding
- Expose configuration as objects
- Compact notation
- Powerful language constructs

```java
for (wiz in ext.wizards) {
    app.helpSystem.register(wiz);
}
```
- Latest features always available
Example:  Spring

```java
import org.springframework.context.support.ResourceBundleMessageSource;

bean = appContext.createRootBeanDefinition(
    ResourceBundleMessageSource.class,
    null, ['basenames':
        ['org/springframework/web/context/WEB-INF/${message-file}',
        'org/springframework/web/context/WEB-INF/more-context-messages'
    ]
)
appContext.beanFactory.registerBeanDefinition('messageSource', bean)
```

By Michael Henderson (http://www.behindthesite.com/)
Example

```java
import javax.script.*;

public void readConfig(String scriptName) throws ScriptException {
    String extension = ...;
    ScriptEngineManager mgr = new ScriptEngineManager();
    ScriptEngine engine = mgr.getEngineByExtension(extension);
    FileReader in = new FileReader(scriptName);
    engine.eval(new FileReader(scriptName));
}
```
Macros

- Use established scripting language as macro language for your application
- Expose application objects to scripts
- Analogy with browser
- Danger: many languages can find/load arbitrary classes
- All the usual caveats about accepting user input apply, esp. on the web
Example: Exposing Objects

// this is the script we want to run
String script = "application.setLogging(true);";

ScriptEngine engine = ...;
engine.put("application", app);
engine.eval(script);

// or

ScriptContext context = new SimpleScriptContext();
context.setAttribute("application", app, GLOBAL_SCOPE);
engine.eval(script, context);
“Reverse Macros”

- Make your application scriptable
- Define your own engine, offer exactly the capabilities you want to expose
- **Invocable** interface contains methods like:
  
  ```java
  invoke(String name, Object... args)
  invoke(Object target, String name, Object... args)
  ```

- Expose high-level operations as verbs
- Scripting consoles can invoke them
Scripting “components”

- Engine must support the `Invocable` interface
- Script returns object `x`
- Application tells the engine “give me a `Y` interface backed by `x`”
  
  ```
  Y y = engine.getInterface(x, Y.class);
  ```
- Adaptation of `x` to `Y` up to the engine
- Commonly, methods are mapped by name
Whole Application

- Just write it all in a scripting language
- UI toolkits often easier to use with scripting (e.g. closures for events)
  ```java
  importPackage(java.awt.event);
  button.addActionListener(function(evt) { ... } });
  ```
- Some languages have compilation to bytecode via a separate tool
  - jythonc, groovyc, jsc
Server-side Components

- Example: write a servlet in scripting
- Compile and package it normally
- Not very dynamic though
- Often, a little magic is all that is needed
  - Jython's PyServlet
  - Groovy's GroovyServlet
- Adapt the idea to other components
- Key: single interface or base class
Example: Jython Servlet

from javax.servlet.http import HttpServlet

class JythonServlet1 (HttpServlet):
    
def doGet(self,request,response):
        self.doPost (request,response)
    
def doPost(self,request,response):
        toClient = response.getWriter()
        response.setContentType("text/html")
        toClient.println(            "<html><head><title>Servlet Test</title>" +            "<body><h1>Servlet Test</h1></body></html>"
A Sweet Spot - XML

- Traditionally a pain point for developers
- Groovy introduced `MarkupBuilder`
- Now builders galore (ant, Swing, ...)
- Also XPath-like constructs
- Highlight dynamic qualities of scripting languages
- More radical approaches:
  - E4X language extension to JavaScript
Example - Groovy

def builder = new NodeBuilder()

builder.diet(type:'healthy') {
    food {
        name('bagel')
        serving(grams: 100)
        calories(120)
        sodium(210)
    }
    food {
        name('truffles')
        serving(grams: 40)
        calories(350)
        fat(25)
        notes('just kidding')
    }
}
function (title, summary, type, content) {
    default xml namespace = “http://www.w3.org/2005/Atom”;
    var doc = <entry>
        <title>{title}</title>
        <summary>{summary}</summary>
        <content type={type}>{content}</content>
    </entry>;
    return doc.toXMLString();
}

// Xpath-like access: find all images in a feed

for each (var i in feed..::*::img) {
    // ...
}
Web Frameworks

- High profile: Rails on JRuby or the many Python web frameworks
- Promising but not 100% there yet
- Language and library compatibility issues
- Native frameworks: Grails, others
- Or use Project Phobos
Project Phobos

- Lightweight web application framework
- Supports multiple scripting languages
- Complementarity
  - Runs in a vanilla web application
- Current focus is on JavaScript
- Bottom-up framework development
- https://phobos.dev.java.net/
Overview

PROJECT GLASSFISH (APPLICATION SERVER)

JAVA™ PLATFORM

PHOBOS ADAPTER

SCRIPTING ENGINES

AJAX LIBRARIES

SCRIPTING LIBRARIES

TEMPLATES, STATIC CONTENT, ADDITIONAL LIBRARIES...

SCRIPTS

VIEWS

CONTROLLERS
Programming Model

- Less prescriptive than ROR
- Meaningful directory structure
- HTTP centric
  - Map out the URLs
  - Attach logic to them
  - Test out interactively, repeat
- REST modeled directly
- Background tasks, AJAX support, etc.
Plain Scripts

- Servlet-like programming
- No need to extend servlet base class
- Request/response bound to variables

```java
response.setStatus(200);
response.setContentType("text/html");
writer = response.getWriter();
writer.println(<html><head><title>Hello from Javascript</title></head><body>Hello from Javascript!</body></html>);
writer.flush();
```
Controllers

- MVC framework
- Controllers are JavaScript “classes”
- Actions are “methods”
- Views are scripts

```html
<html>
  <head><title><%= model.title %></title></head>
  <body><%= model.text %></body>
</html>
```
Resources

- REST framework
- Resources are "classes"
- Methods are HTTP methods
- Code deals with HTTP entities
  - content type, payload, extension headers
- Most HTTP aspects offloaded to framework
- Currently looking at WADL
AJAX

- All jMaki components available
- Layout standardized
- Dojo Toolkit bundled
- JSON natural data format
  - JavaScript on both clients and servers
- jMaki XmlHttpProxy support
- Investigate COMET using Grizzly ARP
Mozilla Rhino Extensions

- Much more powerful language
- JavaAdapter
- JSAdapter
  - on-the-fly properties
- Assignable __proto__ property
  - dynamic inheritance
- Continuations
- E4X support
JDK 7 Early Plan

- New invokedynamic bytecode
  - Not used by the Java™ language
  - May lead to hotswapping capabilities
- Some new Java™ language features being investigated
  - Closures, direct XML support
- Possibly more scripting engines
  - Send us your feedback!
Conclusion

- Scripting can improve your productivity
- Lots of languages to choose from
  - Evaluate implementations for speed, robustness, features
  - IDE support a primary consideration too
- JSR-223 makes embedding scripts easy
- Web applications ground zero for scripting
Scripting in the Java™ Platform

Roberto Chinnici
Senior Staff Engineer
Sun Microsystems, Inc.