Real World Groovy
How to Add Scripting Functionality to Your (Existing) Application

Eric Schreiner
Managing Director
Contecon Software GmbH
Other Groovy related sessions

- Eric Schreiner
  - Real World Groovy
  - How to Add Scripting Functionality to Your (Existing) Application

During the presentation: If you have questions – please ask
Prerequisites

- Basic understanding of Groovy
- Understanding of Java would be helpful.
Agenda

- Techniques to embed Groovy in a Java application
- Performance comparisons
  - creating objects
  - calling methods
- A “real world” example of adding Groovy functionality to an existing application
- Examine a running application with the Groovy "stethoscope"
Conventions used in this presentation

- *Blue italic* will be used for keywords and operators in the text (not in example code)
- All class names for Java classes used in my examples will start with a capital *J* to differ them from Groovy
- Additional example code is indicated in *green* at the bottom of the slides. *Grey* indicates that we will skip the sample during the presentation (time is limited)
Definitions

- **Wikipedia**
  Groovy is an object-oriented programming language for the Java Platform as an alternative to the Java programming language. It can be viewed as a scripting language for the Java Platform, as it has features similar to those of Python, Ruby, Perl, and Smalltalk. In some contexts, the name JSR 241 is used as an alternate identifier for the Groovy language.

- **Codehaus.org**
  Groovy is like a super version of Java. It can leverage Java's enterprise capabilities but also has cool productivity features like closures, builders and dynamic typing. If you are a developer, tester or script guru, you have to love Groovy.
Eclipse Setup for the examples

- Plugin: org.codehaus.groovy_1.0.1
- Compiler output location bin-groovy
Classpath

- Just embed groovy-all-major.minor.jar
  - e.g. groovy-all-1.5.6.jar
  - Easiest way to integrate
  - Found in the embeddable directory of distribution
- Or use .jar files from lib directory
  - More complex
  - May cause problems if your application is using different versions of the same .jar files
- At least a JRE 1.4 required
Embedding Groovy in a Java Application

- Groovyc compiler
  (all source code available before compilation)
- GroovyShell
- GroovyScriptEngine
- JSR-223 Scripting for the Java(6) platform
- Meta programming
Groovyc Example

- Compile *HelloWorld.groovy*
- Examine with *jad*
- Call it from Java
Class GroovyShell

- Simplest integration technique
- Uses *Binding* for passing parameters
- Just call the *evaluate* Method
  - `evaluate(File file)`
  - `evaluate(InputStream in)`
  - `evaluate(InputStream in, String fileName)`
  - `evaluate(String script)`
  - `evaluate(String script, String fileName)`

JGroovyShellExample.java, JGroovyShellBindingExample.java, JGroovyShellSwingExample.java
Class GroovyShell 2

- Generating dynamic classes
  - Can be used to generate classes (e.g. after parsing an XML file)
- The `parse` method of `GroovyShell`
  - Returns an instance of `Script`
  - Is more efficient because recompile is not required

`JGroovyShellDynamicExample.java`, `JGroovyShellParseExample.java`
Class GroovyShell 3

- The *run* method
  - Unlike evaluate, it executes Scripts and classes
  - If a main(Object[] args) or main(String[] args) exists, it's executed
  - If the class extends GroovyTestCase, a JUnit test runner executes it
  - If the class implements Runnable, it will be constructed with a String[] or default constructor. Then run() will be called.
GroovyShell configuration

- The class `CompilerConfiguration`
  - Optionally passed to the constructor of GroovyShell
  - `setScriptBaseClass()`
    see example
  - `setClasspath()`
    used to customize the classpath
  - `setSourceEncoding()`
    And many more
- It's also possible to define a custom classloader

`MyDebugScript.groovy`, `JGroovyShellConfigExample.java`
GroovyScriptEngine (GSE)

- Most complete solution to embed scripts in an application
- The GSE automatically tracks dependencies
- If a Script has been modified, all dependent scripts will be recompiled and reloaded

MyEvaluator.groovy, JGroovyScriptEngineSwingExample.java
JSR-223 integration


- `javax.script.ScriptEngineManager()`
  - `getEngineByExtension(String extension)`
  - `getEngineByMimeType(String mimeType)`
  - `getEngineByName(String shortName)`
    - For Groovy it's “groovy”

- Use the eval methods to run script
  ```java
  Object eval(Reader reader)
  Object eval(Reader reader, Bindings b)
  Object eval(String scriptReader)
  Object eval(String scriptReader, Bindings b)
  ```

See JSR-223 documentation for more info
Meta programming or how the magic works

- In Groovy every class implements the `GroovyObject` interface
  - public Object invokeMethod(String name, Object args);
  - public Object getProperty(String property);
  - public void setProperty(String property, Object newValue);
  - public MetaClass getMetaClass();
  - public void setMetaClass(MetaClass metaClass);

- If an ordinary Java class should be recognized as a Groovy class, it has to implement the GroovyObject interface.

- There is also the abstract class `GroovyObjectSupport` that can be extended
The MetaClass

- Every GroovyObject has an association with **MetaClass**
- **MetaClass** provides all meta-information about a Groovy class
  - Methods
  - Fields
  - Properties
- **MetaClass** also provides the Methods that do the real work of method invocation
  - `public abstract Object invokeConstructor(Object[] args);`
  - `public abstract Object invokeMethod(Object o, String methodName, Object[] arguments);`
  - `public abstract Object invokeStaticMethod(Object object, String methodName, Object[] arguments);`
The MetaClassRegistry

- The MetaClass is stored and received from a central Store – the \textit{MetaClassRegistry}

\begin{center}
\begin{tikzpicture}
    \node (meta) {MetaClassRegistry};
    \node[below of=meta] (get) {getMetaClass (class): MetaClass};
    \node[below of=get] (set) {setMetaClass (class, metaClass)};
    \node[below of=set] (invoke) {\texttt{MetaClass}};
    \node[above of=invoke] (interface) {\texttt{GroovyObject}};

    \draw[->] (get) -- (meta);
    \draw[->] (set) -- (meta);
    \draw[->] (invoke) -- (meta);
    \draw[->] (interface) -- (invoke);
\end{tikzpicture}
\end{center}
Method invocation and interception

- Use `GroovyInterceptable` as a marker Interface to indicate that we override `invokeMethod`
- Overriding the `invokeMethod` method
  - Only works on Groovy Objects
  - Not really reusable when implemented as an abstract class
- Intercepting method calls with `ProxyMetaClass`
  - Also works on non Groovy Objects
- Groovy Categories (http://groovy.codehaus.org/Groovy+Categories)

OverrideInvokeMethodExample.groovy, MethodProxyExample.groovy
JFileGetter.java, TypeFileContent.groovy
Groovy Performance - Constructors

- JBenchmarkConstructor compares object creation between Java and Groovy

- If you want to play with it, make sure that you compile GroovyBench.groovy manually
### Groovy Performance - Constructors

#### Results

- **Loopcount**: 10,000,000 (ten million)
- **Java Constructor**: 109ms
- **Constructor on compiled Groovy Class**: 7907ms
- **GroovyScriptEngine Constructor**: 8610ms
- **GroovyScriptEngine II (just 10000 loops) Constructors**: 4062ms (normalized by factor: 4062000ms)
- **GroovyShell (just 1000) Constructors**: 8672ms (normalized by factor: 86720000ms)

*although a cache is used you have to consider between comfort and performance*

*similar*
Groovy Performance - Methods

- **JBenchmarkMethodInvocation** compares method invocation between Java and Groovy
- If you want to play with it, make sure that you compile GroovyBench.groovy manually

*JBenchmarkMethodInvocation.java, JBench.java, GroovyBench.groovy*
Results

- Loopcount: 10,000,000 (ten million)
- Java voidTest(): 31ms
- Java booleanTest(): 47ms
- Java concatTest(): 3735ms
- Groovy voidTest(): 93ms
- Groovy booleanTest(): 1703ms
- Groovy concatTest(): 26360ms

*JBenchmarkMethodInvocation.java, JBench.java, GroovyBench.groovy*
Groovy Performance - Methods

- Use JAD to examine GroovyBench.groovy

```java
public String concat(String s1, String s2, String s3) {
    Class class1 = GroovyBench.class;
    Class class2 = groovy.lang.MetaClass.class;
    StringBuffer sb = ((StringBuffer) (ScriptBytecodeAdapter.invokeNewN(class1, java.lang.StringBuffer.class, ((Object) new Object[] { s1 })))));
    ScriptBytecodeAdapter.invokeMethodN(class1, sb, "append", new Object[] { s2 });
    ScriptBytecodeAdapter.invokeMethodN(class1, sb, "append", new Object[] { s3 });
    return (String)ScriptBytecodeAdapter.castToType(ScriptBytecodeAdapter.invokeMethod0(class1, sb, "toString"), java.lang.String.class);
}

public boolean booleanTest(boolean b) {
    Class class1 = GroovyBench.class;
    Class class2 = groovy.lang.MetaClass.class;
    return DefaultTypeTransformation.booleanUnbox((Boolean)ScriptBytecodeAdapter.castToType(DefaultTypeTransformation.box(b), java.lang.Boolean.class));
}
```

```java
JBenchmarkMethodInvocation.java, JBench.java, GroovyBench.groovy
```
How to Add Scripting Functionality to an existing Java Application

- Project definition
  - We have a ten year old legacy Java software
  - Without changing the existing application we should do the following:
    - Provide a modern scripting framework to our clients
    - the scripting framework should not be dependent on a specific scripting language
    - provide clean method names
    - hide certain methods and Classes
**Structure of existing App**

package: `de.css2008.myexistingjavaapp`

- **JEmployee**
  - some getters (): String
  - some setters()
  - getSalary(): double
  - setSalary(double d)
  - other Methods

- **JEmployeeDAO**
  - putEmployee(JEmployee)
  - getEmployeeById(Integer id): JEmployee
  - getAllEmployees(): Collection<JEmployee>

*JEmployee.java, JEmployeeDAO.java*
Structure of existing App

Goals

- Hide existing implementation
- Prohibit access to methods
- Instantiation should not be possible for script
- Provide new Interface e.G with closures

package: 
\texttt{de.css2008.myexistingjavaapp}

<table>
<thead>
<tr>
<th>JEmployee</th>
</tr>
</thead>
<tbody>
<tr>
<td>some getters (): String</td>
</tr>
<tr>
<td>some setters()</td>
</tr>
<tr>
<td>getSalary(): double</td>
</tr>
<tr>
<td>setSalary(double d)</td>
</tr>
<tr>
<td>other Methods</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JEmployeeDAO</th>
</tr>
</thead>
<tbody>
<tr>
<td>putEmployee(JEmployee)</td>
</tr>
<tr>
<td>getEmployeeById(Integer id):JEmployee</td>
</tr>
<tr>
<td>getAllEmployees():Collection&lt;JEmployee&gt;</td>
</tr>
</tbody>
</table>

\texttt{JEmployee.java, JEmployeeDAO.java}
Structure of Toolkit

package: de.css2008.myexistingjavaapp

<table>
<thead>
<tr>
<th>JEmployee</th>
</tr>
</thead>
<tbody>
<tr>
<td>some getters (): String</td>
</tr>
<tr>
<td>some setters()</td>
</tr>
<tr>
<td>getSalary(): double</td>
</tr>
<tr>
<td>setSalary(double d)</td>
</tr>
<tr>
<td>other Methods</td>
</tr>
</tbody>
</table>

package: de.css2008.mygroovytoolkit

<table>
<thead>
<tr>
<th>JTkEmployee</th>
</tr>
</thead>
<tbody>
<tr>
<td>some getters (): String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JToolkit</th>
</tr>
</thead>
<tbody>
<tr>
<td>all ToolkitMethods (): String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GroovyTkSimpleExample</th>
</tr>
</thead>
<tbody>
<tr>
<td>the scriptcode</td>
</tr>
</tbody>
</table>

GroovyTkSimpleExample.groovy, JTkEmployee.java, JToolkit.java
Using the Delegating Meta Class

- In Groovy there are two ways to set the DelegatingMetaClass
  - Dynamically by using
    `InvokerHelper.getMetaRegistry().setMetaClass(Class, DelegatingMetaClass)`
  - Providing a custom meta class at a well known location:
    `groovy.runtime.metaclass.[YOURPACKAGE].[YOURCLASS]MetaClass`

in our example: `de.css2008.mygroovytoolkit`

- Toolkit
The challenge

- Just try GroovyTkProblemExample
References

- Groovy
  - Codehaus
    http://groovy.codehaus.org/
  - about Groovy
    http://aboutgroovy.com/
  - JSR241
  - developerWorks

- Books
  - Groovy in Action (very good)
  - Programming Groovy
  - Groovy Programming
Contact me

Eric Schreiner

Eric.Schreiner@contecon.de
http://www.contecon.de

Please fill out the evaluations
Thank you

....if you have questions

ask now