Scripting Your Java App

Kyle Cordes
Oasis Digital Solutions Inc.
kyle@kylecordes.com

St. Louis Java SIG
Nov. 9, 2006
kylecordes.com

This talk has no slides, only code and this handout which serves as a place to show URLs for future reference, and as a guide to the topics discussed.

Beans Scripting Framework

`http://jakarta.apache.org/bsf/projects.html`

The BSF came from IBM in the late 1990s, and was the first major multi-scripting-language plugin API. It supports many languages (BeanShell, Groovy, Jacl (Tcl), Rhino/JavaScript, Jython, etc.), with a focus on embedding script in HTML/XML. BSF is now an Apache project and supports non-web scripting as well.

BeanShell (JSR 274)

`http://beanshell.org/`

BeanShell supports a superset of the Java syntax; as a result you can use it as a more-pleasant-than-Java scripting language, or use it as a Java interpreter. BeanShell was written by Pat Niemeyer here in St. Louis.

Because of the Java syntax, BeanShell can be used to get code working in an interpreted way, or as part of a script plugin/extenstion, then very easily move that code in to your core application, unlike code written in a non-Java scripting language.

Another, more extreme use of BeanShell is to use it instead of `javac` for an entire application, to reduce the deployment size – with Java, compressed source code is often smaller than the equivalent compressed compiled classes!

Groovy (JSR 241)

Groovy is a relatively new language, whose specific syntax and semantics are being worked out in the JSR process. Its key advantage is that its design is much more compatible with how the JVM works than other non-Java languages: it is less dynamic. Thus it is likely to achieve better performance and more straightforward access to Java features, until some future JRE adds better dynamic language support.

Java 6.0 Scripting (JSR 223)

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

It appears that starting with Java 6.0 / 1.6, a scripting API and implementation will arrive "in the box" in the form of the **javax.script** package and bundled JavaScript (Rhino) implementation (as the "js" language). The API is similar to the BSF API, though more refined. The API makes it easy for a host application to support multiple script languages and integrate only once. Java 6 will be released Dec. 7, 2006.

"It's pretty hard to defend a delayed project when you are using Billy's Scripting Language. As a project manager or IT manager you could get fired for making that decision. However, if you build an application using MegaStandard Scripting Language X, you won't because you followed the recommendation of a standards body. Its not your fault it doesn't work as expected." (Richard Monson-Haefel's blog)

Other JSR 223 compliant scripting engines: AWK, BeanShell, Groovy, UGNL, Python, Ruby, Scheme, Tcl, many more.

Rhino (JavaScript)

You probably deploy with Java 5/1.5 or 1.4 now. Use JavaScript (the Rhino engine) today via its API and 693K js.jar file; you are prepared for the future, you will be able to use the JSR 223 API and in-the-box Rhino with little effort. All existing scripts should still work.

Experience Report

We have a large "enterprisey" product; as with many such products, there are some aspects of behavior that are inevitably customer-specific and hard to accommodate through a data-driven configuration mechanism. We use Rhino / JS scripting to add customer-specific behavior, for example in the form "if fields A and B have values C and D, then field E is required". It was surprisingly easy to add scripting support – perhaps a few pages of code to add plug points and invoke the script engine. Recommended.

What's a scripting language anyway?

Common answers, none of which are absolute, include "duck typing", the lack of a complex type system, interpreted execution, ability to "just load some text and eval() it", lack of binary compatibility concerns, and sandboxed execution.

Why script?

Scripting can provide more flexible configuration than the typical data-centric configuration approach; use it whenever you feel the temptation to invent just a little bit of an programming language, particularly in XML. (A rule to live by: do not write code in XML.)

Scripting is more important in commercial systems (which serve many customers) than in internal system; be wary of over-engineering, do not add a scripting language when a lookup table will do. Yet in a commercial system scripting provide a path to make your application "done", in spite of ever-shifting customer-specific needs.

Scripting serves as a "soft layer" in the "Alternate Hard and Soft Layers" design approach, which I illustrated with Lua at a talk last year; there are notes about the topic on my web site.

A scriptable application is generally also a testable, modular application.

Scripting opens the door to user-generated procedure content – witness the amazing proliferation of user-written scripts in Second Life.

What should my API look like?

With **rules scripting**, the user configures a logic behavior for a situation the application encounters, such as a "billing rule". Provide a carefully "shaped" (but not too complex) API, such as an entry point / script per product-type. The shaped API provides good isolation between scripts, important since rules are often filled in by users with very little programming experience.

Plugin scripting is more general, allowing the user to add capabilities and features to the application, within limits. Impose much less structure on the script; exposing very few entries points (perhaps just one), then let the script register interest in events as they require. This will make it possible for a scripter with some experience, to use an intentional design for their plugin, rather than contort it to meet your API. An **end-to-end plugin** spans the tiers (client/server/etc.) of the application; if your mechanism assists the script by carrying ad hoc attributes along with predefined data, such a plugin can be quite powerful. For example, it could define the entry of additional data, rules for the processing of that data, billing based on that data, etc.

External scripting, common in large desktop applications like Excel, steers the operation of the application with an API that resembles the model of a user manipulating the UI, with mild additional abstraction.

Be wary of what primitives you expose to the scripts; you may inadvertently expose your entire class structure – a large and complex "surface area" for which it will be very challenging to maintain future compatibility. **Sandbox** your scripts as much as feasible.

Questions?

oasisdigital.com