ORACLE®

Speedy Scripting: Productivity and Performance

Per Bothner
Oracle / GNU

Agenda

- "Scripting" vs "programming"
- Does scripting language performance matter?
- Best of both styles?
- Some languages: Scala, JRuby, Clojure, Kawa
- Some benchmark numbers
- Some issues in language design

Who am I?

- Hired by Sun to work on JavaFX Script compiler
- Past technical lead for GCJ (gcc-compiler-based ahead-of-time compiler for Java), at Cygnus Solutions
- Lead for Kawa, one of the first compilers for a dynamic language on the JVM (starting 1996, at Cygnus)
- Various other Open-Source and GNU contributions

Disclaimer

- This mostly describes non-Sun/non-Oracle work
- Does not necessarily reflect Oracle views
- No language to be discussed (except Java!) is owned, endorsed, or supported by Oracle

Scripting Languages are Nice!

- Compilation is missing/optional/automatic
- real-eval-print interfaces (exploratory programming)
- Less boilerplate (don't need class + main method)
- Simpler type systems, optional or no type specifiers
- Flexible/dynamic name and type resolution
- Execution of "incomplete" program aids incremental development
- *Makes programmers productive and happy!*
- Popular JVM examples: JRuby, Jython, Groovy, ...

Compiled "Programming" Languages are Nice

- Fast execution
- Good compile-time error-checking
- Good types; sophisticated type systems
- Easier for tools to figure out what is going on
- People, too
- *Makes hardware, purchasing agents, and the environment happy!*
- Popular JVM examples: Java, Scala, ...

Everybody Wants to Combine the Best of Both

- Popular scripting languages are compiled to (some kind of) bytecode
- JSR-292 and other work to speed up dynamic languages
- Compiled languages support "scripting" (eval, javax.script) and a read-eval-print-loop

Does Script Language Performance Matter?

- Not for the classic small script for doing modest jobs
- But as people get comfortable with a scripting language, they use them for more and more and bigger and bigger tasks
- At some point people hit "the wall" [Norm Walsh] - the job is big enough that run-time is a problem
- Example: JavaScript performance is starting to matter

Of Course We Also Care About ...

- Programmer productivity
- Application robustness
- Extensibility and maintainability
- Finding (or better: avoiding) bugs

Semi-solution:
Use "Appropriate Language For Each Job"

- Sometimes needed, but can be a hassle
- Incompatible functionality, object models, type systems. (Need to use lowest-common-denominator values.)
- Many people aren't fluent in multiple languages

On the “Programming Language” end: Scala - "the Better Java"

- Strongly typed, research-based
- Similar to Java performance
- Very expressive - concise powerful programs
- May be intimidating to some

At the “Scripting” End: Popular Languages

- Jython
- JRuby
- Groovy
- PHP
- JavaScript
- Issues with performance, error-detection, "programming in the large"

Is There a “Golden Mean”?

Clojure

- A relatively new Lisp-like JVM language
- Uses immutable arrays, maps, etc
- Especially good for parallel (multi-threaded) execution

Thorn

- Interesting new language from IBM Research + Purdue
- Patterns
- Good parallelism story
- Syntactic extension
- OOPSLA '09 paper is recommended reading
- Not ready for production use

Kawa

- Dialect of Scheme - a mostly-functional language
- Scheme is standardized, many implementations
- Extensible syntax (hygienic macros)
- Continuous development and use since 1996

Kawa Performance

- True compiler
- Compiler can save class files, but is also fast enough for “load-and-go”
- Primarily lexical name binding, but dynamic is available
- First-class functions, efficient multiple inheritance
- Compile-type optimization and specialization
- Type specifiers are optional - otherwise uses type inference or run-time lookup/checking
- Fast!

Computer Languages Benchmark Game - ("shootout")

	Scala	Clojure	Jruby	Kawa
fannkuch	1.5	52.1	46.8	1.7
fasta			73.9	1.8
knucleotide	2.1	7.0	9.1	2.0
mandelbrot	1.2	2.6	timeout	1.1
nbody		5.1	115.3	1.0
pidigits	0.3		1.6	0.6
regexdna	1.6		3.0	1.0
spectralnorm	1.1		timeout	1.0
threadring			47.1	1.0

- Multiple solutions of multiple problems
- See http://shootout.alioth.debian.org
- Runtime relative to Java

Performance Factors in Language Design

- Type specifications should be available and optional
- Name scope should lexical (compiler-analyzable)
- Default scope should be script-local (to aid type-inference and other analysis)
- Access to primitive arithmetic, Java arrays, etc
- Think about performance issues from the start

Type Specifications

- Good for performance, catching errors, documentation
- Traditional static (strict) typing: Compiler rejects code if it cannot prove value matches target type
- This can be tedious, and seems to require non-trivial type systems (generics, to start with)
- Combine optional types with type inference
- Optimistic static typing: Compiler rejects code if it can prove value cannot match target type
- Lots of research: gradual typing, type inference, ...

Declarations

- Variable declarations are good
- Lexical scoping and name looking are good
- Compiler should be able to map variable reference to declaration
- Thus: Variable declaration need to be compact
- At least for immutable (final) variables
- Likewise: compiler should be able to figure out which (possibly-indirect) function is called - ... or (important!) complain about an undefined function
- Basis for most optimization and error-checking

Avoid String-based APIs

- Constructing XML data or SQL queries should be done functionally
- Using string-pasting is very error-prone - and risks injection attacks
- Using library functions or special syntax allows compiler to catch errors
- Example: XML literals evaluate to XML Nodes
- (Not primarily a performance issue, since ultimate result are often strings.)

Complex Object Construction

- Creating *e.g.* Swing components is very verbose
- Better some kind of keyword-based constructors
- Groovy has SwingBuilder
- Scala has library making good use of Scala features

Swing Frame with Button using Kawa

```
(JFrame
  title: "Hello!"
  (JButton
    text: "Click me!"
    tool-tip-text: "click here")))
```

- No Swing-specific library or “magic”
- Compiler looks for `set`, `add`, `valueOf` methods
- Static typing makes this easier – no runtime reflection

Single Abstract Method conversion

- One of the uses of closures (in Java and otherwise):
- A SAM class or interface has one abstract method
- Can use a lambda expression (closure object)
- Compiler creates anonymous class

```
(my-click-label:addActionListener
  (lambda (evt)
    (my-click-action)))
```

Expression-oriented Programming

- Use expressions that return results, not side-effects
- E.g.: `"Hello!"`
- not: `print "Hello!"`
- Expressions are less verbose and more composable
- Use list comprehensions rather than Java-style loops
- Example: Part of SQL's power is because you can compose query expressions
- Compiler can optimize

Avoid Side-effects

- Multi-CPU computers requires more parallelism
- Compilers need freedom to re-order or parallelize
- This is easier with functional / side-effect-free computations
- Thus functional languages becoming more important
- Haskell (GHC) is sometimes faster than Java in the "shootout"
- (Haskell is very powerful and compact – but probably too scary for most)

Conclusion

- Performance of scripting languages matter
- Language design matters for performance
- Both language designers and programmers need to think in terms of expressions, composable building blocks, and avoiding side-effects

Links

- Kawa: http://gnu.org/software/kawa/
- Clojure: http://clojure.org/
- Scala: http://scala-lang.org/
- JRuby: http://jruby.org/
- Thorn: http://thorn-lang.org/
- Haskell: http://haskell.org/
- or just use your favorite search engine
- Per Bothner http://per.bothner.com
  per@bothner.com or per.bothner@oracle.com


ORACLE®