ThoughtWorks®

Creating a language on the JVM

With case studies from JRuby and Ioke

Ola Bini
ola.bini@gmail.com
http://olabini.com/blog

ThoughtWorks®

Why the JVM?

Memory management

Hotspot

Concurrency

Open Source, with many implementations

Mature code loading

High level abstractions

Platform independence

Reflective access to classes and objects

Tools

Many mature libraries

ThoughtWorks®

JRuby

Implementation of the Ruby language

Current version 1.2ish

Java 1.5

Mature, fast

Interpreter

JIT compiler

Best implementation of the Ruby language for most purposes

ThoughtWorks®

Ioke

An experiment

A programming language

Dynamic and strong typing

Prototype based object orientation

Homoiconic

Hosted on the JVM (Java 1.5+)

Current version: Ioke S ikj 0.2.0

ThoughtWorks®

Decisions

General purpose or special purpose?

- Scripting, Web, Rules, DSL, Game engine

Paradigm

- OO (what kind), Logic, Functional, Metaprogramming

Syntax or not

Statically typed or not

Other features

- Suggestions?

ThoughtWorks®

Parts of a language

Syntax

Type system

Execution model

Core types

Libraries

Integration/FFI

ThoughtWorks®

Syntax

What your language looks like

The text/outer representation that is used to describe your program

Representation of literals

- Numbers, strings, identifiers
- Array literals? Hash literals?

Method calls

Statement separators

Comments

ThoughtWorks®

Syntax: Lexing

Turn pieces of text into tokens

A token represents a unit of something

- OpenParenthesis, Identifier, Symbol, Integer
- Decimal, String, Comment, etc

The rule is that the lexer should be able to easily recognize what kind of token a piece of text is

Usually removes whitespace

- Whitespace-dependent languages an exception

ThoughtWorks®

Syntax: Parsing

- Arranges tokens together
- Make sure the arrangements makes sense
- Turns everything into a parse tree
  - An arrangement of the tokens into productions, generally
- Pure parser only says yes or no
  - It's mostly not so useful in that form
  - So most places have parsers output a tree structure
- Lexers and parsers are generally automatically generated
- JRuby uses handwritten lexer
- Ioke generates both lexer and parser

ThoughtWorks®

Type checking

What kind of type system?

- Nominal
- Structural

System F

Typed lambda calculus

Subtyping

How to handle parametric polymorphism?

- Variance?

ThoughtWorks®

Bottom types

What is a bottom type?

Something that type checks as a subtype of every type

null in Java and C#

Do you want one?

nil in a dynamic language is not the same thing

Bottom types means that you can always get runtime errors

Several languages go for an Option[T] or Maybe[T] instead

ThoughtWorks®

Scoping

- Dynamic
- Lexical
- Object
- Closures

ThoughtWorks®

Control structures

- Expressions
  - Precedence, associativity
  - Assignment
  - Initialization
  - Ordering within expressions
  - Short-circuiting
- Control flow
  - goto
  - Continuations
- Iteration
  - Recursion

ThoughtWorks®

Control structures - conditionals

if, unless

cond

case

conditional assignment

    ||= and ilk

Trinary if statement

Notions of truth

ThoughtWorks®

Data types

Numbers

Booleans

Strings

Arrays/Lists

Sets

Associative arrays

Primitive versions for efficiency?

Other types - IO, Ranges, Pairs, Time, etc

ThoughtWorks®

Equality

Reference types

Pointers

Aliases

Equality

Identity

Hashing

ThoughtWorks®

Parameter passing

Call by value (C, Scheme, Java, C++)

Call by reference (Perl, Visual Basic, C++)

Call by name

Call by need (Lazy evaluation, Haskell)

Return values

- Out values, Multiple return values

Optional arguments

- Where is the default value evaluated? When?

Keyword arguments

Rest arguments

ThoughtWorks®

Data model

- Modules, Namespaces, Packages
- Object orientation
  - Multiple inheritance
  - Class based?
  - Prototype based?
  - Polymorphism
    - Parametric
    - Subtyping
- Functional
- Procedural
- Something else?

ThoughtWorks®

Logic programming

Declarative

Unification

Search order

Imperative control flow

ThoughtWorks®

Evaluation

- Interpreter
  - AST visitor pattern
- Internal Byte-code Compiler
- Virtual machine
  - Evaluates language specific byte codes
- Java Byte-code compiler
  - Loads code in a new class
    - Needs a new classloader because of GC
- Continuation Passing style

ThoughtWorks®

Representation of code

AST at runtime

Bytecode at runtime

Lisp

- S-expressions (lists of lists and atoms, basically)

Ioke

- Tree of Message objects

Ruby

- Strings
- ParseTree returns lists of symbols (in a complicated format)

ThoughtWorks®

Internal representation

- Wrap everything?
  - JRuby: RubyObject, RubyString, RubyFixnum, etc
- Allow raw Java objects
  - Don't depend on a specific class/interface signature
  - Can't expect objects to keep references to their runtime
- How to represent core types
  - Directly, RubyString is a subclass of RubyObject
    - Small amount of efficiency
  - Indirectly, IokeObject points to an IokeData
    - Composition instead of inheritance
    - Allows implementation of become

ThoughtWorks®

Handling errors

- Exceptions
- Conditions
- Global error-handler
  - Traps and signals
- Special return values
- Kill execution

ThoughtWorks®

Concurrency

- Shared memory
  - Transactional memory
- Message passing
  - Futures, transparent or oblique
  - Agents/Actors

ThoughtWorks®

Text manipulation

String types

Regular expressions

Functional view on strings - see them as lists

- Use map/filter etc
- Destructuring

SNOBOL

ThoughtWorks®

Java Integration

- Call Java methods
  - Static ones too
- Automatically coerce arguments to correct types
  - Choose among overloaded methods
- Construct Java objects
- Keep references to Java objects
- Coerce data into Java primitives
- Work with Java arrays
- Automatically coerce return value to language types?
- Implement Java types in host language

ThoughtWorks®

Implementing interfaces

Use Java dynamic proxy

Returns an object that implements all interfaces given

And invokes one method for any call

Convenient and easy

Might be slow

Doesn’t generalize to class extensions

“Quick and dirty”

JRuby had this a few years back - doesn’t anymore

Ioke never went through this stage

ThoughtWorks®

Extending classes

Override any method that is not final

Call protected super members

Call super for the current method

Differentiate between implementing overloaded methods

Dispatch calls to super constructor

You need to generate bytecode to do this

ThoughtWorks®

Byte code compilation

Something like ASM works well

You need to understand Java bytecodes

And the runtime model of Java

Tricky bits:

- Loading code without running out of memory
- Stack height
- Exceptions
- Constants

```
foo.bar.Quux.class
==
ldc “foo.bar.Quux”
invoke_static java/lang/Class forName (Ljava/lang/String;)Ljava/lang/Class;
```

ThoughtWorks®

Bridging the Java object model

How does a Java object look like in your language?

How do you construct new objects?

How do you convert from and to Java objects

- And primitives
- And arrays

Do you have classes in your language?

- Are Java classes the same kind of thing?
- Are Java classes first class at all?

Two approaches - masquerade Java objects as your language

- Or make Java objects magic - special

ThoughtWorks®

Java object model

Classes are not objects

But you can get at a representation for a class

Methods are not objects

But you can get at a representation for a method

Fields are not objects

But you can get at a representation for a field

An object is created from a class

An object has the same methods and fields as its class

The methods can not be changed per object

The fields can be changed per object

ThoughtWorks®

Ruby object model

- Everything is an object - sort of
  - Classes and modules are objects
  - But classes have a magic allocate-method
- Any object has any kind of instance variables
  - Instance variables are bindings from names to objects
- Any object has any number of methods
- An object has an associated class
  - That class can be a named one
  - Or an anonymous singleton class - generated when doing object specific changes
- Plus some fudging under the covers with modules

ThoughtWorks®

Ioke object model

Everything is an object

Any object can mimic zero or more other objects

An object has zero or more cells

- A cell is a binding from a name to an object of some kind

A method is an object

A macro is an object

A piece of code is a collection of objects

ThoughtWorks®

Limitations of the JVM

Statically typed bytecodes

Primitive values

- But not tagged values

No tuples

No support for continuations

No support tail call optimizations

No support for multiple return values

Garbage collection of classes

- Makes code loading impractical

Startup cost

ThoughtWorks®

JSR 292, DaVinci, JVM languages

JSR292 - invoke dynamic JSR

DaVinci - a place to experiment with newer language features

JVM languages - a place to discuss libraries related to above mentioned features

Q and A