

Ruby Mutants

Check your torches at the door...



Except where otherwise noted, the content of this presentation is licensed under the Creative Commons Attribution-Share Alike 3.0 United States License (<http://creativecommons.org/licenses/by-sa/3.0/us/>).

Me

- Charles Oliver Nutter
- headius@headius.com
- @headius on Twitter
- JRuby Architect at Engine Yard
- JVM enthusiast

JRuby

- Ruby for the JVM
- VM for Ruby
- Call Java from Ruby
- Call Ruby from Java
- Labor of love

“Ruby for Java Devs”?

- JRuby is “a Ruby”, but...
 - Mostly-Java codebase
 - Java terrifies Rubyists
 - Java is verbose, inelegant
 - Java evolution moving slowly
- Can we use a different language?

Why Not X?

- Many JVM languages to choose from!
 - More elegant
 - Cleaner syntax
 - Shiny and new
- But do they fit requirements?

What Do I Want?

- As fast as Java for all cases
- No runtime library
- Easily extensible
- Beautiful, clean, flexible syntax
- Dynamic-feeling

Duby

- Static-typed, Ruby-like language
- Ruby syntax
- Pluggable transforms + macros
- Pluggable static types (w/ local inference)
- Pluggable backend (JVM, etc)

Syntax Tour

- Classes, literals, flow-control like Ruby
- Methods like Ruby with type decl
- Type imports (for JVM backend)
- new/initialize for object construction
- Method overloading
- Interface impl with “implements”

Ruby

```
def fib(a)
  if a < 2
    a
  else
    fib(a - 1) + fib(a - 2)
  end
end
```

Duby

```
def fib(a:int)
  if a < 2
    a
  else
    fib(a - 1) + fib(a - 2)
  end
end
```

Duby

```
def fib(a:int)
  if a < 2
    a
  else
    fib(a - 1) + fib(a - 2)
  end
end
```

Declare parameter types

Duby

```
def fib(a:int)
  if a < 2
    a
  else
    fib(a - 1) + fib(a - 2)
  end
end
```

Declare parameter types

Circular references
ok

Duby

```
def fib(a:int)
  if a < 2
    a
  else
    fib(a - 1) + fib(a - 2)
  end
end
```

Declare parameter types

Circular references
ok

Return type inferred

.class output

```
Compiled from "fib.duby"  
public class examples.fib extends java.lang.Object{  
    public static void main(java.lang.String[]);  
    public static int fib(int);  
    public examples.fib();  
}
```

.class output

```
public static int fib(int);
```

```
Code:
```

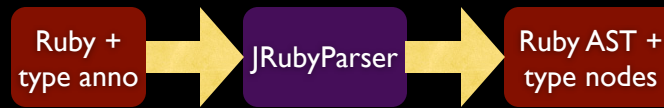
```
0: iload_0
1: iconst_2
2: if_icmplt 9
5: iconst_0
6: goto 10
9: iconst_1
10: ifeq 17
13: iload_0
14: goto 30
17: iload_0
18: iconst_1
19: isub
20: invokestatic #11; //Method fib:(I)I
23: iload_0
24: iconst_2
25: isub
26: invokestatic #11; //Method fib:(I)I
29: iadd
30: ireturn
```

.java output

```
// Generated from fib.duby
public class fib extends java.lang.Object {
    public static void main(java.lang.String[] argv) {
    }
    public static int fib(int a) {
        return (a < 2) ? a : (fib.fib((a - 1)) + fib.fib((a - 2)));
    }
}
```

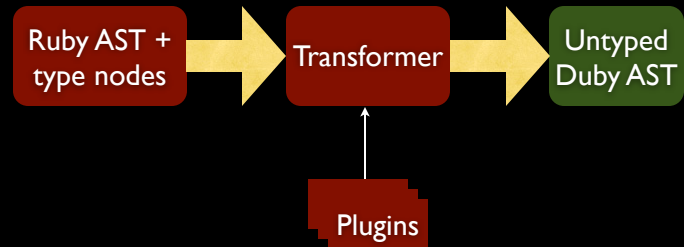
What's Happening?

Phase I: Parse Ruby



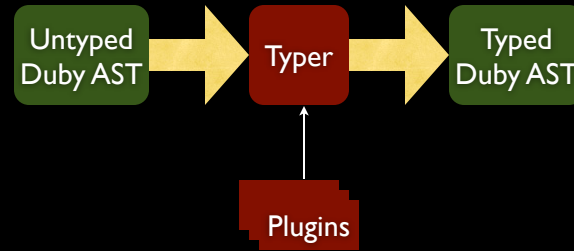
What's Happening?

Phase 2: Transform to DUBY AST



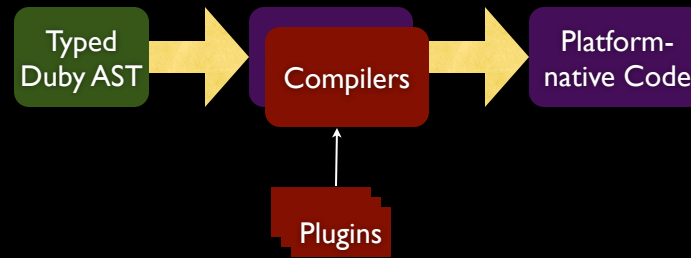
What's Happening?

Phase 3: Infer Types



What's Happening?

Phase 4: Generate Native Code



Sidebar: Bitescript

- DSL/API for emitting JVM bytecode
- Used by Surinx, Duby compilers
- Ruby-based, requires JRuby
- Nice Ruby wrapper around ASM

Do It Live!

(Things may break)

But I Like Dynlangs!

- Static typing is not always the right answer
 - TDD is especially cumbersome
- Many static libs fall back on reflection
- JVMs starting to support dynamic calls

Surinx

- Dynamically-typed, Ruby-like JVM language
 - Ruby syntax
 - Dynamic typing (all calls dynamic)
 - Java/JVM type system
 - Requires invokedynamic (JSR-292)

Ruby

```
def fib(a)
  if a < 2
    a
  else
    fib(a - 1) + fib(a - 2)
  end
end
```

Duby

```
def fib(a:int)
  if a < 2
    a
  else
    fib(a - 1) + fib(a - 2)
  end
end
```

Surinx

```
def fib(a)
  if a < 2
    a
  else
    fib(a - 1) + fib(a - 2)
  end
end
```

.class Output

```
public static java.lang.Object fib(java.lang.Object);
Code:
  0: aload_0
  1: ldc2_w    #10      // long 21
  4: invokestatic #17      // Method java/lang/Long.valueOf:(J)Ljava/lang/Long;
  7: invokedynamic #36, 0 // NameAndType __lt__: (Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object;
 12: getstatic  #42      // Field java/lang/Boolean.FALSE:Ljava/lang/Boolean;
 15: if_acmpeq  22
 18: aload_0
 19: goto      63
 22: aconst_null
 23: aload_0
 24: ldc2_w    #43      // long 11
 27: invokestatic #17      // Method java/lang/Long.valueOf:(J)Ljava/lang/Long;
 30: invokedynamic #46, 0 // NameAndType "-": (Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object;
 35: invokedynamic #47, 0 // NameAndType fib: (Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object;
 40: aconst_null
 41: aload_0
 42: ldc2_w    #10      // long 21
 45: invokestatic #17      // Method java/lang/Long.valueOf:(J)Ljava/lang/Long;
 48: invokedynamic #46, 0 // NameAndType "-": (Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object;
 53: invokedynamic #47, 0 // NameAndType fib: (Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object;
 58: invokedynamic #49, 0 // NameAndType "+": (Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object;
 63: areturn
```

Everything ends up Object, all dynamic invocations

Whither DUBY?

- Optional static types?
 - Gradually specialize for perf
 - No perf penalty as in Groovy
- Dynamic invocation?
 - When target can't be statically inferred

Merge!

- DUBY will gain SurinX features
 - No need for two languages
 - Reuse infrastructure
- Hybrid static/dynamic
- They're your types...
use them when YOU need them!

What About JRuby?

- Doby, Surinx are idea testbeds
 - Ruby with optional static types?
 - Unifying Ruby and Java types for JRuby
- Doby, Surinx are *not Ruby*
- No core library, frameworks, apps
- Doby, Surinx are good practice :)

Your Turn

- Join the project, help out!
 - www.github.com/headius/duby
- Mind the semantic gap!
- Watch Duby and JRuby this coming year!

Contact

- www.kenai.com/projects/duby
- www.kenai.com/projects/jvmscript
(bitescrpt)
- www.duby-lang.org (soon)
- www.jruby.org
- headius@headius.com