PHP.reboot

JVM Summit'10

In few words ...

- Secure: no eval, no magic quote
- DSLs for XML, JSON, SQL, Xpath, Xquery, URI, Perl 5 regex
  - Auto-escaping depending on the context
- Webserver & embeded SQL DB by default
- Dynamic typed with gradual type system
- Duck typing
- Interpreter, ahead of time & runtime compilers
- Works with 1.5/1.6/1.7 VMs

Hello World !

```
dom = document(
    http://search.twitter.com/search.atom?lang=en&q=Java)
titles = for node in dom//entry/title
              return node
<news>
{
  foreach(titles as title) {
    title.name = "description"
    echo title
  }
}
</news>
```

Gradual Typing

- You declare types *if you want*

```
function fibo(a) {
  if (a < 2)
    return 1
  return fibo(a - 1) + fibo(a – 2)
}
```

- or

```
function fibo(int a):int {
  if (a < 2)
    return 1
  return fibo(a - 1) + fibo(a – 2)
```

- or any combination

Toolchain

- Tatoo – parser generator (LALR)
  - Generate ASTs from grammar + type annotations
  - Keyword are *local* by default
  - Evaluate AST asap (fast reduce)
  - Manage multiple grammars, change grammar at runtime
- JSR 292 + lightweight loading
  - Share runtime logic between interpreter and bytecode
  - Method handles/invokedynamic allow to jump back and forth between evaluator and generated codes
  - Lambda/closure for free
- ASM – fast bytecode generation
  - Partial supports of JDK7 classfile format

Main ideas

- Compile to bytecode for speed
- But compilation takes time
  - Mixed strategy, first interpret then compile

- Performance:

  Java VMs have JITs

    JIT are amazing piece of software

  so don't do the job twice

  Generate same code as javac

Invokedynamic works but ...

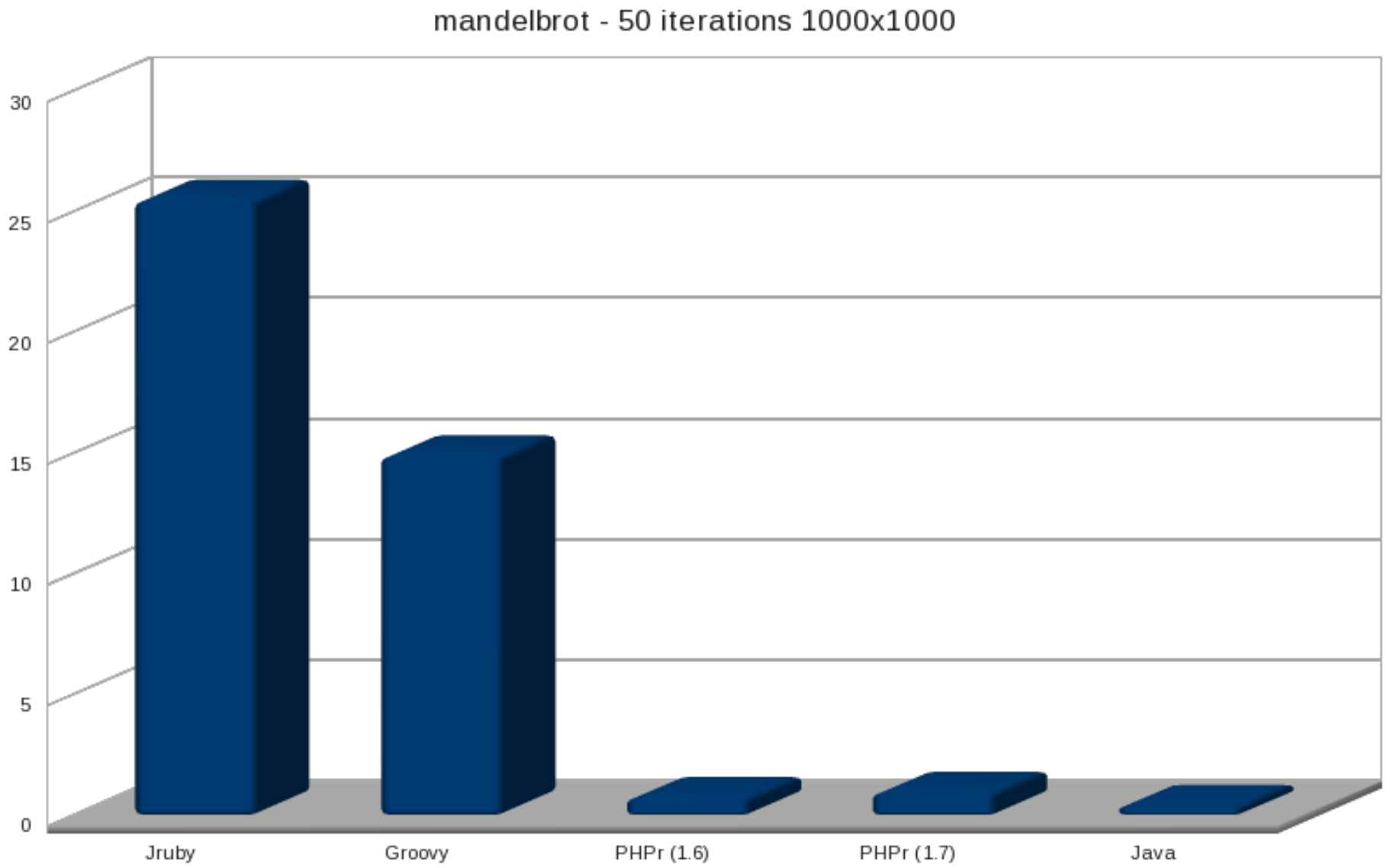
- In a perfect world, type profiling and escape analysis are sufficient
  - Invokedynamic on objects works great !
    - Inlining cache for dummies
  - Problem for calculation involving primitive types
    - Escape analysis is limited to inlining horizon
- **Runtime must do type profiling !**

Trace loop body !

- Record
  - Runtime type of last value of each variable
  - Branches taken
- Typecheck speculatively using recorded variable types (fast forward typechecking)
  - If a conjecture is proved false, revert the variable type to Object and use invokedynamic
  - If body contains a function call, try to specialize it
    - Keep all specializations => fix point
  - If no record (branch not taken) continue
    - If conjecture is proved false,
      generate an escape trace for the whole branch
- Generate bytecode using typecheked types

Micro-benchmark – Mandelbrot

(time in ms)



Show me codes !

```
for(i = 0; i < 350; i = i + 1) {
  if (i == 100) {
      i = 102.0
  }
}
```

- With Ahead Of Time compiler
- With type inference and no escape to interpreter
- With type inference and escape

AOT code

```
    iconst_0
    invokestatic  Integer.valueOf:(I)LInteger;    // boxing
    astore 2
l0: aload 2
    sipush 350
    invokedynamic ge:(LObject;I)Z
    ifne l1
    aload 2
    bipush 100
    invokestatic  Integer.valueOf:(I)LInteger;     // boxing
    invokestatic  RT.ne:(LObject;LObject;)Z
    ifne l2
    ldc 102.0
    invokestatic  Double.valueOf:(D)LDouble;   // boxing
    astore 2
l2: aload 2
    iconst_1
    invokedynamic plus:(LObject;I)LObject;
    astore 2
    goto l0
l1: ...
```

Type inference without escape

```
trace(LEvalEnv;Ljava/lang/Object;LVar;)Z
l0: aload 1
    sipush 350
    invokedynamic ge(LObject;I)Z
    ifne l1
    aload 1
    bipush 100
    invokestatic Integer.valueOf(I)LInteger;   // boxing
    invokestatic RT.ne(LObject;LObject;)Z
    ifne l2
    ldc 102.0
    invokestatic Double.valueOf(D)LDouble;  // boxing
    astore 1
l2: aload 1
    iconst_1
    invokedynamic plus(LObject;I)LObject;
    astore 1
    goto l0
l1: aload 2       // post instruction, update interpreter variables
    aload 1
    invokevirtual Var.setValue (LObject;)V
    ...
```

Optimization with escape trace
(phase 1)

```
trace(LEvalEnv;ILMethodHandle;LVar;)Z
l0: iload 1
    sipush 350
    if_icmpgt l1          // no loop peeling !
    iload 1
    sipush 100
    if_icmpne l2
    aload 2
    aload 0
    iload 1
    invokevirtual invoke(LEvalEnv;I)V
    iconst_0
    ireturn               // return false, trace escape !
l2: iload 1
    iconst_1
    iadd
    istore 1
    goto l0
l1: ...                   // post instruction, update interpreter variables
```

Escape to interpreter

- Need to restore:
  - Value stack
    - Typechecker prepare a blank stack + association between parameters position and stack position
  - ~~Interpreter call stack (walk on the AST)~~
- Solution =>
  An adhoc evaluator that exit from the trace
- Loop may be re-optimized later

Optimization after escape trace
(phase 2)

```
trace(LEvalEnv;DLVar;)V
l0: dload 1
    sipush 350
    i2d                 // no constant propagation !
    dcmpg
    ifge l1
    dload 1
    bipush 100
    i2d
    dcmpg
    ifne l2
    ldc 102.0
    dstore 1
l2: dload 1
    iconst_1
    i2d
    dadd
    dstore 1
    goto l0
l1: ...                 // post instruction, update interpreter variable
```

And with function calls ?

```
function fibo(a) {
  if (a < 2)
    return 1
  return fibo(a - 1) + fibo(a - 2)
}

i = 0
while (i < 200) {
  fibo(i % 7)
  i = i + 1
}
```

Fibo is called often

```
fibo(LEvalEnv;LObject;)LObject;
    aload 1
    iconst_2
    invokedynamic ge(LObject;I)Z
    ifne l0
    iconst_1
    invokestatic Integer.valueOf(I)LInteger;      // boxing
    areturn
l0: aload 0
    aload 1
    iconst_1
    invokedynamic minus(LObject;I)LObject;
    invokedynamic call:fibo(LEvalEnv;LObject;)LObject;
    aload 0
    aload 1
    iconst_2
    invokedynamic minus(LObject;I)LObject;
    invokedynamic call:fibo(LEvalEnv;LObject;)LObject;
    invokedynamic plus(LObject;LObject;)LObject;
    areturn
```

'while' is traced/compiled

```
trace(LEvalEnv;ILVar;)Z
l0: iload 1
    sipush 200
    if_icmpge l1
    aload 0
    aload 1
    bipush 7
    irem
    invokedynamic call:fibo(LEvalEnv;I)LObject;   // fibo is specialized !
    pop
    iload 1
    iconst_1
    iadd
    istore 1
    goto l0
l1: aload 2                  // post instruction, update interpreter variables
    aload 1
    invokestatic Integer.valueOf(I)LInteger;    // boxing
    invokevirtual Var.setValue(LObject;)V
    iconst_1
    ireturn
```

Specialized Fibo

```
fibo(LEvalEnv;I)Ljava/lang/Object;
    iload 1
    iconst_2
    if_icmpge l0
    iconst_1
    invokestatic Integer.valueOf(I)LInteger;      // boxing
    areturn
l0: aload 0
    iload 1
    iconst_1
    isub
    invokedynamic call:fibo(LEvalEnv;I)LObject;
    aload 0
    iload 1
    iconst_2
    isub
    invokedynamic call:fibo(LEvalEnv;I)LObject;
    invokedynamic plus(LObject;LObject;)LObject;
    areturn
```

JSR 292 backport

- Idea, ease the transition to the JSR 292 world
- Backport the API to work with 1.5/1.6 VMs
  - A weaver transform all calls to java.dyn (online/offline)
  - A runtime optimizer (online) ; a VM agent ; rewrites callsites at runtime to optimize invokedynamic/method handle invocation
- With the VM agent performance of 1.6 VM almost as par with 1.7 VM !
- Without, perf are equivalent to use reflection

Demos

- PHP.reboot on Android
- Test traceescape
- Test tracefunctionspecialized3

First a question for you:
Why your runtime doesn't do type profiling + type checking ?

Any questions ?

PHP.reboot

http://code.google.com/p/phpreboot/