


Scala:
Byte-code Fancypants

David Pollak

JVM Language Summit 2009

http://github.com/dpp/jvm_summit_2009

About DPP

- Author *Beginning Scala*
- *BDFL Lift*
- *Wrote some spreadsheets*

What is Scala?

- ~~A pile of short-cuts for common patterns~~
- An object oriented language that's byte-code and class compatible with Java
- A functional language with a powerful type system
- A fast, concise general purpose language


Scala History

- Wirth PhD's Odersky
- Odersky & Wadler do Pizza, GJ, & Generics
- Odersky does Scala 2003
- Scala 2.0 2006
- Lift 2007
- "It's hot baby" 2009 (books, talks, etc.)


Scala: Java NG

- ```
  public class Person {
      public final String name;
      public final int age;
      Person(String name, int age) {
          this.name = name;
          this.age = age;
      }
  }
  ```

- ```
  class Person(val name: String, val age: Int)
  ```

Scala: Java NG

- ```
  import java.util.ArrayList;
      ...
      Person[] people;
      Person[] minors;
      Person[] adults;
      {  ArrayList<Person> minorsList = new ArrayList<Person>();
         ArrayList<Person> adultsList = new ArrayList<Person>();
         for (int i = 0; i < people.length; i++)
            (people[i].age < 18 ? minorsList : adultsList)
                     .add(people[i]);
         minors = minorsList.toArray(people);
         adults = adultsList.toArray(people);
      }
  ```

- ```
  val people: Array[Person]
  val (minors, adults) = people partition (_.age < 18)
  ```

Auto Boxing/Unboxing

- class BoxBox {
  def in(i: Int): List[Int] = i :: Nil
  def out(lst: List[Int]): Int = lst.head
}

Boxing Bytecode

- public scala.collection.immutable.List in(int);
  Code:
  0: iload_1
  1: istore_2
  2: getstatic #18; //Field scala/collection/immutable/Nil$.MODULE
$:Lscala/collection/immutable/Nil$;
  5: iload_2
  6: invokestatic #24; //Method scala/runtime/
BoxesRunTime.boxToInteger:(I)Ljava/lang/Integer;
  9: invokevirtual #28; //Method scala/collection/immutable/Nil$.$colon
$colon:(Ljava/lang/Object;)Lscala/collection/immutable/List;
  12: areturn

Unboxing Bytecode

- public int out(scala.collection.immutable.List);
  Code:
  0: aload_1
  1: invokeinterface #43, 1; //InterfaceMethod scala/collection/generic/IterableTemplate.head:()Ljava/lang/Object;
  6: invokestatic #47; //Method scala/runtime/BoxesRunTime.unboxToInt:(Ljava/lang/Object;)I
  9: ireturn

Functions

- object Invoker {
  def invoke(f: () => Int): Int = f()
  }

  class CallInvoker {
  def intFunc(): Int = 5
  def doCall() = Invoker.invoke(intFunc _)
  }

Invoker: Java-ish

- final class Invoker extends java.lang.Object with ScalaObject {
  def invoke(f: Function0): Int = scala.Int.unbox(f.apply());
};

CallInvoker: Java-ish

- class CallInvoker extends java.lang.Object with ScalaObject {
  def intFunc(): Int = 5;
  def doCall(): Int = Invoker.invoke({
  {
  (new jvm_summit.CallInvoker$$anonfun$doCall$1
(CallInvoker.this): Function0)
  }
  });
  def this(): jvm_summit.CallInvoker = {
  CallInvoker.super.this();
  ()
  }
 };

Synthetic Function

- final <synthetic> class CallInvoker$$anonfun$doCall$1 extends java.lang.Object with Function0 with ScalaObject {
  override def toString(): java.lang.String = scala.Function0$class.toString(CallInvoker$$anonfun$doCall$1.this);

  final def apply(): Int = CallInvoker$$anonfun$doCall$1.this.$outer.intFunc();

  <synthetic> <paramaccessor> private[this] val $outer: jvm_summit.CallInvoker = _;

  final <bridge> def apply(): java.lang.Object =
      scala.Int.box(CallInvoker$$anonfun$doCall$1.this.apply());

  def this($outer: jvm_summit.CallInvoker): jvm_summit.CallInvoker$$anonfun$doCall$1 = {
      .....
  }
 }
}

Multiple Apply?

- public final java.lang.Object apply();
  Code:
  0: aload_0
  1: getfield #22; //Field $outer:Ljvm_summit/CallInvoker;
  4: astore_1
  5: aload_0
  6: invokevirtual #39; //Method apply:()I
  9: invokestatic #45; //Method scala/runtime/BoxesRunTime.boxToInteger:(I)Ljava/lang/Integer;
  12: areturn

public final int apply();
  Code:
  0: aload_0
  1: getfield #22; //Field $outer:Ljvm_summit/CallInvoker;
  4: astore_1
  5: aload_0
  6: getfield #22; //Field $outer:Ljvm_summit/CallInvoker;
  9: invokevirtual #50; //Method jvm_summit/CallInvoker.intFunc:()I
  12: ireturn

Structural Types

- object Structural {
  def getLen(in: {def length(): Int}): Int =
  in.length

  def main(in: Array[String]) {
  getLen("Hello")
  getLen(new Lenny)
  }
  }
  class Lenny {
  def length() = 55
  }

Structural Types Bytes

- public int getLen(java.lang.Object);
  Code:
  0: aload_1
  1: astore_2
  2: aconst_null
  3: astore_3
  4: aload_2
  5: invokevirtual #51; //Method java/lang/Object.getClass:()Ljava/lang/Class;
  8: invokestatic #55; //Method reflMethod$Method1:(Ljava/lang/Class;)Ljava/lang/reflect/Method;
  11: aload_2
  12: iconst_0
  13: anewarray #29; //class java/lang/Object
  16: invokevirtual #61; //Method java/lang/reflect/Method.invoke:(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
  19: astore_3
  20: aload_3
  21: checkcast #63; //class java/lang/Integer
  24: invokestatic #68; //Method scala/runtime/BoxesRunTime.unboxToInt:(Ljava/lang/Object;)I
  27: ireturn
  28: astore 4
  30: aload 4
  32: invokevirtual #74; //Method java/lang/reflect/InvocationTargetException.getCause:()Ljava/lang/Throwable;
  35: athrow
  Exception table:
  from to target type
  4 20 28 Class java/lang/reflect/InvocationTargetException


Tail Calls

- class CallMe {
  final def count(in: Int): Int =
    if (in >= 1000) in else count(in + 1)
}

Tail Call Bytecode

- public final int count(int);
  Code:
  0: iload_1
  1: ldc #13; //int 1000
  3: if_icmplt 8
  6: iload_1
  7: ireturn
  8: iload_1
  9: iconst_1
  10: iadd
  11: istore_1
  12: goto 0

Uniform Access

- trait Access {
    def name: String
  }

  class Person extends Access {
    var name = "David"
  }

Uniform Access: Code

- ```
  abstract trait Access extends java.lang.Object {
    def name(): java.lang.String
  };
  class Person extends Object with jvm_summit.Access with ScalaObject {
    private[this] var name: java.lang.String = _;
    <accessor> def name(): java.lang.String = Person.this.name;
    <accessor> def name_=(x$1: java.lang.String): Unit =
                    Person.this.name = x$1;
    def this(): jvm_summit.Person = {
      Person.super.this();
      Person.this.name = "David";
      ()
    }
  }
  ```
  ```

Laziness

- class LazyPerson extends Access {
  lazy val name =
  System.currentTimeMillis.toString
}

Lazy Code

- ```
  <stable> <accessor> lazy def name(): java.lang.String = {
    if (LazyPerson.this.bitmap$0.&(1).==(0))
      {
        LazyPerson.this.synchronized({
          if (LazyPerson.this.bitmap$0.&(1).==(0))
            {
              LazyPerson.this.name = scala.Long.box(System.currentTimeMillis()).toString();
              LazyPerson.this.bitmap$0 = LazyPerson.this.bitmap$0.|(1);
              ()
            };
          scala.runtime.BoxedUnit.UNIT
        });
        ()
      };
    LazyPerson.this.name
  };
  ```
  ```

Case Class

- case class Namey(name: String, age: Int)
- For Free: hashCode, equals, toString, pattern matching, extractors, product, copy (type-safe clone), named parameters:

```
val n = Namey("David", 45)
val older = n.copy(age = n.age + 1)
```

Pattern Matching

- def check(in: Any): String = in match {
  case i: Int => "Integer: "+i
  case d: Double if d > 0.0D => "Double: "+d
  case Namey(n, 45) => "Name: "+n
  case x => "Dunno "+x
}

Pattern Matching code

```
def check(in: java.lang.Object): java.lang.String = {
  var temp6: java.lang.Object = in;
  if (temp6.$isInstanceOf[Int]())
    {
        var temp7: Int = scala.Int.unbox(temp6);
            "Integer: ".+(scala.Int.box(temp7.+(3)))
    }
  else
    if (temp6.$isInstanceOf[Double]())
      {
          var temp8: Double = scala.Double.unbox(temp6);
              val d: Double = temp8;
              if (PatPat.this.gd2$1(d))
                {
                    "Double: ".+(scala.Double.box(temp8.*(2.0)))
                }
              else
                {
                    val x: java.lang.Object = scala.Double.box(temp8);
                    body%3(x){
                      "Dunno ".+(x)
                    }
                }
  }
else
  if (temp6.$isInstanceOf[jvm_summit.Namey]())
    {
        var temp9: jvm_summit.Namey = temp6.$asInstanceOf[jvm_summit.Namey]();
          var temp10: java.lang.String = temp9.name();
            if (scala.Int.box(temp9.age()).==(scala.Int.box(45)))
              {
                  "Name: ".+(temp10)
              }
            else
              body%3(temp9)
    }
else
    body%3(temp6)
```

Functions & Vars

- class Variable {
  def doSomething(f: Int => Unit) = f(42)

  def refVar() {
    var x = 0
    doSomething(y => x += y)
    assert(x == 42)
  }
}

Vars: how they're done

- ```
  def refVar(): Unit = {
    var x$1: scala.runtime.IntRef = new scala.runtime.IntRef(0);
    Variable.this.doSomething({
      (new jvm_summit.Variable$$anonfun$refVar$1(Variable.this, x$1))
    });
    scala.this.Predef.assert(x$1.elem.==(42))
  };
  ```

- ```
  final def apply(y: Int): Unit =
    Variable$$anonfun$refVar$1.this.x$1.elem =
      Variable$$anonfun$refVar$1.this.x$1.elem.+(y);
  ```

Conclusion

- Simple Scala constructs:
  - Are common Java patterns
  - Expand into complex code
  - Reasonably optimized by JVM
- Make writing maintainable code much easier & faster

Questions