Scala
(at JP Morgan)

Joe Halliwell • 2015.11.30

Why should you listen to me?

Hello

2001	Started PhD in Artificial Intelligence (Python)
2003	Founded Blootag (J2ME Java)
2005	Co-founded Edinburgh Robotics (C, Java, Python)
2007	Finished PhD (phew!)
2008	Co-founded Winterwell Associates Ltd (A bit of everything)
	(lots of stuff happens)
2015	Joined JP Morgan (Python, Scala, Java)

Why should you care about JP Morgan?

JP Morgan

- Founded in 1871
- Now the biggest bank in the US
- $2.6T assets under management
- 250k employees
- Grown by 1,200 mergers
    - **2000: Chase Manhattan**
    - **2004: Bank One**
    - **2008: Bear Stearns acquisition**
    - **2008: Washington Mutual**

Tech at JP Morgan

- There's a lot of it
  - 7k applications
  - 60k servers
  - 32 datacentres
  - 30k engineers
- Full stack
- Glasgow Technology Centre

FP & Scala

Imperative vs Functional

Imperative

- Model: Turing machine
- Languages: C, Java, Go
- Iteratively modify state
- while/goto
- Objects/collections
- Boring

Functional

- Model: Lambda calculus
- Languages: Lisp, ML, Haskell
- Combine pure functions
- map/fold
- Algebraic datatypes/Monads
- Cool

Scala

“I [...] wanted to combine functional and object-oriented programming” - Odersky

- JVM language with good interop
- More concise and regular than Java
- Lots of nice productivity features
- Higher-order functions
- Algebraic types and pattern matching
- Much richer type system

Why Scala not Haskell?

- Java++
- JVM
- Tooling
- Ecosystem
- People
- Marketing
- Fear, uncertainty and doubt

Example: Case classes

Java

```
public class Person {
	private final long id;
	private String name;

	public Person(long id, String name) {
		this.id = id;
		this.name = name;
	}

	public long getId() {
		return this.id;
	}
	public String getName() {
		return this.name;
	}
	public String setName(String name) {
		this.name = name;
	}
	// FIXME: Add sensible equals(), hashCode()
	// TODO: Add toString(), copy()
}
```

Scala

```
case class Person(id: Long, var name: String)
```

Example: Lazy initialisation

Java

```java
public class App {

   private volatile ConnectionPool pool;
   public ConnectionPool getConnectionPool() {
      ConnectionPool cp = pool;
      if (cp == null) {
        synchronized(this) {
           if (cp == null) {
             cp = pool = new ConnectionPool();
           }
        }
      }
      return cp;
   }

}
```

Scala

```scala
class App {
  lazy val pool = new ConnectionPool()
}
```

Example: Mixins and call by value

Java

```java
public class Logger {
  public boolean quiet = false;
  public String log(String msg) {
    if (!quiet) System.out.println(msg);
  }
}

public class Person {
  static Logger LOG = new Logger();

  /*... 30 lines of boilerplate ...*/

  public void email(String msg) {
    if (!LOG.quiet) LOG.log(“Emailing “ + name);
  }
}
```

Scala

```scala
trait Logger {
  var quiet = false
  def log(msg: => String) = if (!quiet) println(msg);
}


case class Person(id: Long, var name: String) extends Logger {
  def email(msg: String) = {
    log(“Emailing “ + name)
  }
}
```

Example: Algebraic types

Haskell

```haskell
data List a = Nil | Cons a (List a)
map                        :: (a->b) -> (List a) -> (List b)
map f Nil                  =  Nil
map f (Cons x xs)          =  Cons (f x) (map f xs)
```

Scala

```scala
sealed trait List[+A] {
  def map[B](f: A => B): List[B] = this match {
     case Nil => Nil
     case Cons(head, tail) => Cons(f(head), tail.map(f))
  }
}

case object Nil extends List[Nothing]
case class Cons[A](head: A, tail: List[A]) extends List[A]
```

It’s not all good

- Compiler is sloooooooooow
- Uniform access can cause confusion e.g. `List(1, 2, 3).toSet()`
- Features are open to abuse e.g. operator overloading
- Unrestrained use of implicits can cause confusion
- Need to be careful about tail recursion
- Standard library is advanced e.g.

```
override def ++[B >: A, That](that: GenTraversableOnce[B])(implicit bf: CanBuildFrom[List[A], B, That])
```

- See what Paul Philips has to say

Learning more about Scala

- Functional Programming Principles in Scala https://www.coursera.org/course/progfun
- Effective Scala http://twitter.github.io/effectivescala/
- Cats: a functional programming library http://non.github.io/cats/
- Validation typeclass http://blog.lunatech.com/2012/03/02/validation-scala

What’s driving Scala adoption?

Spark

- MapReduce implementation
- Tries to minimize IO
- Suitable for interactive use
- Designed with machine learning in mind
- Really nice functional API

```
val data = sc.load("hdfs://namenode/input.txt")
val top20 = data.map(line => line.split(" "))
                .map(word => (word, 1))
                .reduceByKey((count1, count2) => count1 + count2)
                .top(10)(Ordering.by(_._2)
```

- Hive integration and SparkSQL

Case study: TPS

Existing system

- 100k lines of Bash, 200k lines of SQL
- Optimized for a proprietary database
- Processes historical data going back a few years
- Lots of table-driven matching logic with no up-to-date specification
- Takes a couple of days to run on 16 nodes

Proof of concept

- Take 10% with 6hr runtime and port to Spark
- Experiment with blending SQL/Java/Scala
- Bottom-up development, supported by full regression test suite

How did it go?

TPS port to Spark

Bad

- Less refactoring than we’d hoped
- Slower than we’d hoped
- Spark/SparkSQL is immature
- Finicky tuning

Good

- Completed in 2 months
- 2k lines of code
- 100% test coverage
- 1.5hr runtime
- Evidence of linear scaling
- Green light

Corporate vs Startup

Corporate	Bootstrapped startup
Static	Agile
Conservative	Innovative
Bureaucratic	Lightweight
Predictable, secure, 9-to-5	Insecure, constant crunch
Big problems	Small problems
Boring	Cool

Think like an owner

Thanks!

Want to find out more?

joe.halliwell@jpmchase.com