Scala

David Ainslie

Registered Traveller

Global Entry

Scala is huge! So where to begin?

The best thing I can do in an introduction such as this, is to make sure that everyone takes away at least one thing useful. So I’m going to talk quite high level, but also show a bit of code, but also keeping that code at a high level - maybe I can do future talks on more specific areas of this huge subject - we will briefly cover one such subject at the end.

Scala

The scalable language

Apparently, Scala got its name from “scalable”. It’s certainly the most scalable language I’ve ever seen. You can start from tiny applications, even shell scripts, up to easy and yet powerful web applications, and then all the way up to gigantic, distributed, big data crunching applications.

All applications have one thing in common (regardless of language) - they grow… and for me, Scala is the best language to manage that growth.
We don’t want to write; have problems; throw away; and then start again.
We want to manage, adapt, and constantly improve.

Sure Scala has a steep learning curve - but it is worth it.

Too many things can go wrong with your applications and developers need ways to minimise bad code. Scala does this at the syntactical level, making code readable, concise and maintainable. It is type safe, where the compiler does a ton of work for you, but at the same time, the code can be written as if it were a dynamic language.

SBT

Unlike other build tools where you might describe your build in
- XML or
- some simple scripting language

SBT allows you to describe your build, in essentially the language i.e. Scala, that your application is in.
Obviously this gives you greater power and flexibility, but also with the advantage of adding fancy extras.

- Reactive development environment. (Change source code, and sbt reruns your tests.)
- Interact with your code from a REPL
- Interact with your running application

Quick word on SBT before looking at Scala from tiny, medium to gigantic.
Most languages have a build tool, but usually that build tool, even though powerful, is still fairly simplistic. It will build your application, by managing its dependencies, compiling, running tests, and eventually creating some artifact that represents your final product, that can be run.
SBT is so much more.

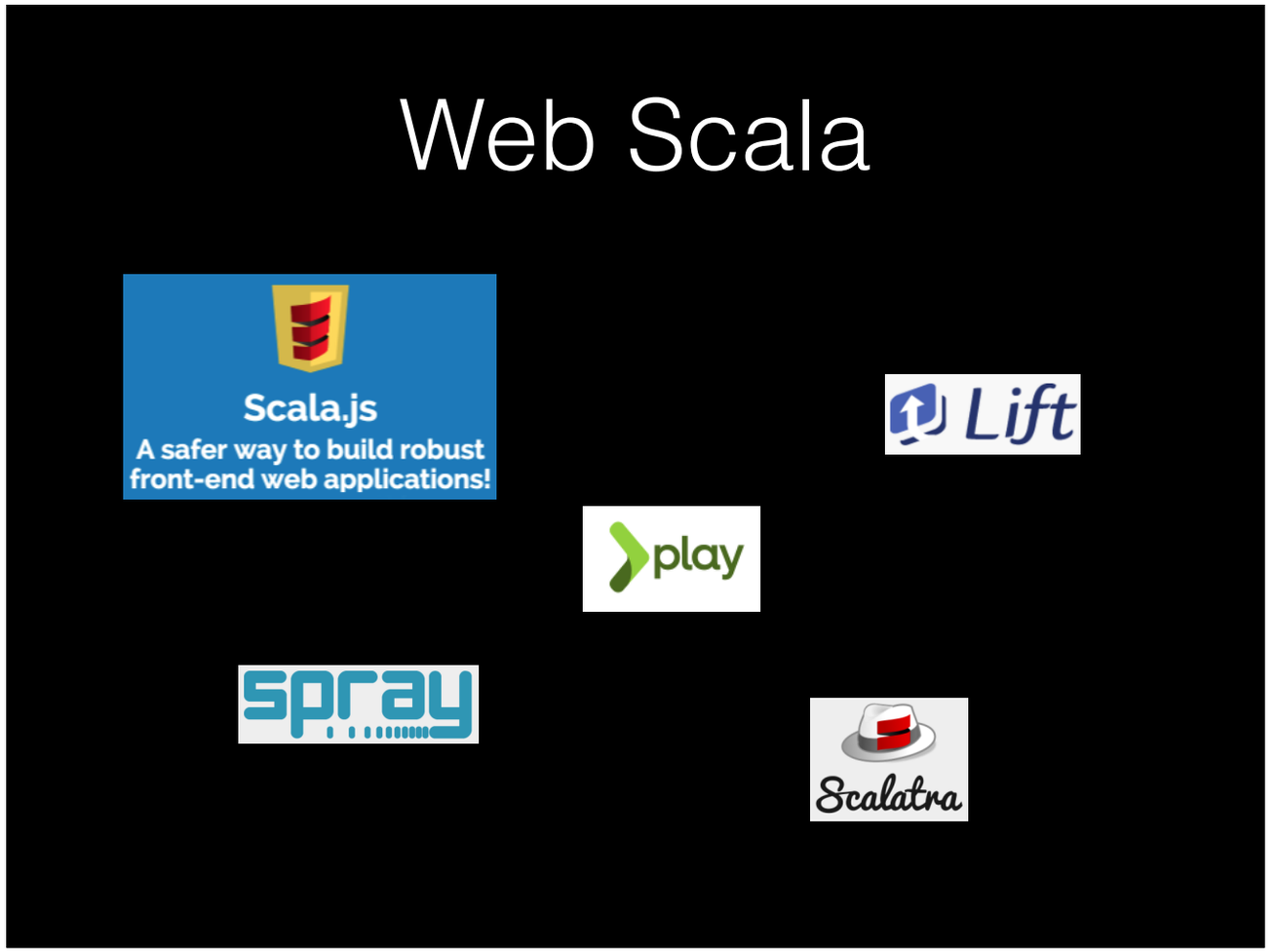
Tiny Scala

- Replace Bash
- Note, there is a brilliant presentation at http://tinyurl.com/beyondbash
- Run sbt - Scala's interactive build tool - then just type

e.g. within a terminal/console that has ammonite as a dependency (as this presentation project has):

> sbt test:console
> val listed = ls!
> listed.<TAB> shows us all methods available on "listed" - the same as context help in your IDE
> listed.zipWithIndex map { case (line, index) => s"$index) $line" } foreach println

Why would you want to use Ammonite-Shell instead of Bash? Possible reasons include:
You can never remember the syntax to write an if-statement in Bash



Play is a reactive web framework, created for the demands of modern web applications.

A notable competitor is Node.js.
JavaScript got itself into an unassailable position on the client i.e. browser.
Now I'm one of those who likes to criticise JavaScript. I'll quote a famous quote:
Historians who reflect on JavaScript's emergence as a dominant programming language in the 21st century may find themselves quoting Donald Rumsfeld: "You go to war with the army you have, not the army you might wish to have."
And then of course, someone thought it would be a good idea to have JavaScript on the backend as well, for a consistent technology across an application stack.
Nice idea - in theory! I'll say no more.

What is most interesting, is the number of languages that can now be compiled down to JavaScript. Scala with Scala.js has taken advantage of this, to allow Scala to be a full stack technology. Many others have done this, an example being Clojure and ClojureScript.

Note that I mention some other Scala web libraries.
Lift for example is apparently the most secure web framework - it's used by a lot of banks.

And Spray, which we use (as well as Play) is a highly performant toolkit for building REST/HTTP based integration layers on top of Scala and Akka (which I'll mention in a moment).

Big Scala

- Akka - A…MAZE…ING (see next slide)
- Spark - It's the big data killer application
- Algebird - Used to build aggregation systems
- Scalding - Makes Hadoop user friendly
- Kafka - Unbelievably powerful messaging

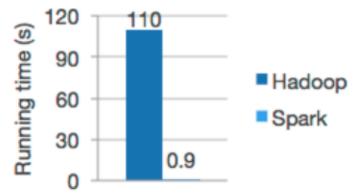
Akka

Build powerful concurrent & distributed applications more easily

Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.



Logistic regression in Hadoop and Spark

Spark

http://www.slideshare.net/deanwampler/why-scala-is-taking-over-the-big-data-world

Kafka

Designed to allow a single cluster to serve as the central data backbone for a large organization. It can be elastically and transparently expanded without downtime.

Show me some Scala code

A sidenote

Scala has too many good points, to cover in this introduction.

However, one advantage, can actually be a disadvantage.

The advantage is that Java code can seamlessly be used in Scala. This is great; we can use many super 3rd party Java libraries. But you shouldn't start writing your Scala code like Java.

To get the best out of Scala, you must treat it as its own language - this is very important for Java developers coming to Scala. The same can also be true for Java developers writing JavaScript.

Scala is…

- Functional Programming
- Object Oriented Programming
- Test Driven Development
- Expression Oriented Programming
- Event Driven Development
- Aspect Oriented Programming
- Reactive and Functional Reactive Programming
- etc…

Many people try to say what Scala is, in one line, usually “It is a hybrid language, specifically Functional and Object Oriented, that runs on the JVM”.
Then you have others, such as the Haskell and Clojure developers, that say, “it is not a functional language, as you can write non-functional code”. That is true, but that is also the point.
For every aspect of programming, there is something more suitable that something else. E.g. when modelling your business domain, object oriented programming is most suitable. Functional programming suits the behaviour. Reactive programming suits your web API and also interfacing with other external systems. Your cross cutting concerns should utilise Aspect Oriented programming. And the list goes on, and so does the list of “Scala is…”.
And the one thing that probably all these methodologies go hand in hand with; is Test Driven Development.

Of all the many reasons why functional programming is really taking off, is that it perfectly suits Test Driven Development. And for me, Scala really emphasises this.

And the easier it is to write tests, the more likely developers will, not only write tests, but will indeed do Test Driven Development.

Oh! And just to note, TDD is so easy in Scala, and most of the libraries are written with an emphasis on testability, you test web application and web services endpoints with just a few lines of code, right up to actually writing tests for your clusters in a manner that they still look just like unit tests.

So in a nutshell, Scala brings together testable, functional, expressions, and immutability as if they are one and the same.

Scala Random Basics 1

Multiple each item in a list by 2

> sbt

> (1 to 10) map { _ * 2 }

In the functional world; for every item in the list, we want to apply a function:

given the item, multiple by 2

_ * 2 is our function (the underscore being our item, that I have not named, because I don't care about the name), and as with all functions, there is an "input", then something "processes the input", and the end result is an "output" i.e.

input	=>	output
number	function processing	output
1	1 * 2	2
2	2 * 2	4
3	3 * 2	6

Scala Random Basics 2

Sum a List of Numbers

> sbt

> (1 to 10) reduce { _ + _ }

> (1 to 10) sum

Scala Random Basics 3

Filter list of numbers

Filter a list of numbers into two categories based on a criteria using partition.This example creates two lists of students based on their test scores, where a pass is 60 and above:

> sbt

> val (passed, failed) =
        List(49, 58, 76, 82, 88, 90) partition { _ >= 60 }

Scala Random Basics 4

Happy Birthday function

> sbt

```
> def happyBirthday(name: String) = (1 to 4).map { line =>
    s"Happy Birthday ${if (line == 3) s"dear $name" else "to You"}"
  } foreach println

> happyBirthday(“Batman")
```

This function show expression oriented programming.
Expression oriented programming, actually fits perfectly with functional programming.
Everything in expression oriented programming results in “something” and that is how pure functions operate, in that, for some input, there is always an output, and more specifically, the output is always the same, for a given input i.e. there is no dependency on the outside world.
And then, this all goes perfectly with “test driven development”, since you want your tests to always give the same result, no matter what else is going on i.e. tests should be completely independent.

Scala Random Basics 5

Fetch and Parse an XML web service

Note that XML is a native structure to Scala, parsing an XML feed comes with no effort:

> sbt

> import scala.xml._

> val xml = XML.load(“http://www.devdaily.com/rss.xml")

> xml \ "channel" \ “title"

> (xml \ "channel" \ “title”) text

Functional Programming

If you want to get the best out of Scala, you have to write functional code.
This is a big area and certainly needs a talk all of its own.
And as a side note, a lot of the functional syntax and ideas was taken from Haskell.

What is “map”?
Well, “map” is a function. So we are giving a function to a function.

Again, for another talk, one thing we do in functional programming is function composition.

Maybe you remember form school maths, seeing something like f(g(x)) which is:

x => g(x) => f(g(x))

Scala Code

Pick a subject!

Futures! Why, because I was going to give a presentation on future. And the code I present is quite straightforward, and all in “tests” showing how easy it is to write tests in Scala. It’s as if it is a part of the language, encouraging developers to write good code.

The code can be seen at:

https://github.com/UKHomeOffice/scala-presentation

along with the slides named “presentation.key”