Experiences Using Scala in Apache Spark

Patrick Wendell
March 17, 2015
Apache Spark

Cluster computing engine for big data

API inspired by Scala collections

Multiple language API’s (Scala, Java, Python, R)

Higher level libraries for SQL, Machine Learning, and Streaming
Data Sources

Spark Core

Spark SQL
Spark Streaming
MLlib
GraphX

Data Sources:
- hadoop
- cassandra
- Hive
- HBase
- PostgreSQL
- JSON
- MySQL
- elasticsearch
Contributors per Month to Spark

Most active project at Apache,
More than 500 known production deployments
A Tiny bit of Spark

Load error messages from a log into memory
then interactively search for various patterns
https://gist.github.com/ceteri/8ae5b9509a08c08c08a1132

```scala
> val lines = spark.textFile("hdfs://...")
> val errors = lines.filter(_.startsWith("ERROR"))
> val messages = errors.map(_.split("\t")).map(r => r(1))
> messages.cache()
> messages.filter(_.contains("mysql")).count()
```
About Databricks

Created by the founders of Spark

Databricks Cloud: A hosted analytics service based on Apache Spark

Most internal components written in Scala
About Me

UC Berkeley, then Databricks

Managing Spark team, releases, and roadmap

Systems and networking background (not PL)
This Talk

Our overall experience using Scala

Advice for future projects

Unsolicited feedback to the Scala community
Overall impressions of Scala
Using a new Programming Language is Like Falling in Love

When you meet, you are enamored by what’s interesting and different

Over time... you learn the quirks about your partner

The key to success is finding the right modes of interaction and fostering commitment from both sides
Why We Chose Scala

“The answer is pretty simple. When we started Spark, we had two goals — we wanted to work with the Hadoop ecosystem, which is JVM-based, and we wanted a concise programming interface similar to Microsoft’s DryadLINQ (the first language-integrated big data framework I know of, that begat things like FlumeJava and Crunch). On the JVM, the only language that would offer that kind of API was Scala…”

- Matei Zaharia in May 2014 [link]
Why We Chose Scala

Compatible with JVM ecosystem
  Massive legacy codebase in big data

DSL support
  Newer Spark API’s are effectively DSL’s

Concise syntax
  Rapid prototyping, but still type safe

Thinking functionally
  Encourages immutability and good practices
Perspective of a Software Platform

Users make multi-year investments in Spark

Large ecosystem of third party libraries

Hundreds of developers on the project (and developers come and go)
Important to Us

Backwards compatible, stable APIs

*APIs you don’t think of e.g. our build tool*

*We’ve never changed a major API in 4 years*

A simple and maintainable code base

*Code needs to be obvious and easy to fix*
Some Concerns with Scala

Easy to write dense and complex code

*Ethos that minimizing LOC is the end game*

Certain language concepts can be easily abused

*E.g. operator overloading, Option/Try, chaining*

Compatibility story is good, but not great

*Source compatibility is a big step towards improving this*
Announcing the Databricks Style Guide

“Code is written once by its author, but read and modified multiple times by lots of other engineers.”

https://github.com/databricks/scala-style-guide
Example: Symbolic Names

Do NOT use symbolic method names, unless you are defining them for natural arithmetic operations (e.g. +, -, *, /). Symbolic method names make it very hard to understand the intent of the functions.

```
// symbolic method names are hard to understand
channel ! msg
stream1 >>= stream2

// self-evident what is going on
channel.send(msg)
stream1.join(stream2)
```
Example: Monadic Chaining

class Person(val attributes: Map[String, String])
val database = Map[String, Person]
// values for any attribute can be “null”

def getAddress(name: String): Option[String] = {
  database.get(name).flatMap {
    elem =>
      elem.attributes.get("address")
        .flatMap(Option.apply)
        // handle null value
  }
}

def getAddress(name: String): Option[String] = {
  if (!database.contains(name)) {
    return None
  }
  database(name).attributes.get("address") match {
    case Some(null) => None // handle null value
    case Some(addr) => Option(addr)
    case None => None
  }
}
Example: Monadic Chaining

Do NOT chain (and/or nest) more than 3 operations.

If it takes more than 5 seconds to figure out what the logic is, try hard to think about how you can expression the same functionality without using monadic chaining.

A chain should almost always be broken after a flatMap.
Other Topics in Maintaining a Large Scala Project
Binary Compatibility

*Source compatibility:* Third party code will compile cleanly against newer versions of your library.

*Binary compatibility:* Third party code compiled against older versions of your library can link/run with newer versions.
Binary Compatibility

Can’t change or remove function signatures

Can’t change argument names

Need to cross compile for different Scala versions
Adding concrete members to traits breaks binary compatibility:

<table>
<thead>
<tr>
<th>v1</th>
<th>v2 (broken)</th>
</tr>
</thead>
</table>
| trait person {
    def name: String
} | trait person {
    def name: String
    def age: Option[Int] = None
} |

It's okay if you don't expect third parties to extend the trait. Otherwise, use abstract class.
Return Types

Explicitly list return type in public API’s, otherwise type inference can silently change them between releases:

<table>
<thead>
<tr>
<th>v1</th>
<th>v2 (broken)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>def getRdd = {</code></td>
<td><code>def getRdd = {</code></td>
</tr>
<tr>
<td><code>new MappedRDD()</code></td>
<td><code>new FilteredRDD()</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>

```python
def getRdd: RDD = {
    `new MappedRDD()`
}
```
Verifying Binary Compatibility

MIMA tool from Typesafe – useful primitive but very outdated

We’ve built tooling around it to support package private visibility

Building a better compatibility checker would be a great community contribution!
Java API’s

Conjecture: The most popular Scala projects in the future will have Java API’s
Exposing Java API’s in Scala

Not that hard, but you need to avoid:
- Default argument values
- Implicits in API’s
- Returning Scala collections
- Symbolic method names

Need to runtime unit test everything using Java

With some encouragement, Scala team has helped fix Java compatibility bugs
Performance in Scala

Understand *when* low-level performance is important

When it’s important:
- Prefer Java collections over Scala collections
- Prefer while loops over for loops
- Prefer private[this] over private
IDE’s

I DON'T ALWAYS WRITE SCALA

BUT WHEN I DO, I PREFER INTelliJ
Most Spark engineers switched to IntelliJ from Eclipse in ~2013
Build Tools

For a long time we supported an SBT and a Maven build.

Our “official” build is now Maven, but we use the sbt-pom-reader plugin to support SBT.

SBT has improved substantially since we made this decision.
## Build Tools

<table>
<thead>
<tr>
<th></th>
<th>SBT</th>
<th>Maven</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Invoke scalac</strong></td>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td><strong>Multiple modules</strong></td>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td><strong>Continuous compile</strong></td>
<td>Easy</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Publishing artifacts</strong></td>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td><strong>Build language</strong></td>
<td>Scala</td>
<td>XML</td>
</tr>
<tr>
<td><strong>Plug-ins</strong></td>
<td>SBT plugins</td>
<td>MOJO</td>
</tr>
<tr>
<td><strong>Dependency resolution</strong></td>
<td>Ivy-style</td>
<td>Maven-style</td>
</tr>
<tr>
<td><strong>Anything else</strong></td>
<td>Medium</td>
<td>Hard</td>
</tr>
</tbody>
</table>
Getting Help with Scala

The best Scala book you’ve never heard of

Scala issue tracker
https://issues.scala-lang.org/secure/Dashboard.jspa
Conclusions

Scala has a large surface area. For best results, we’ve constrained our use of Scala.

Keeping your internals and (especially) API simple is really important.

Spark is unique in its scale, our conventions may not apply to your project.
Thank you. Questions?