Jupyter & Scala

Why hasn’t an official Scala kernel emerged yet?

Alexandre Archambault
github.com/alexarchambault
@alxarchambault
1. Scala kernels / shells / notebooks
2. Features
3. jupyter-scala
Scala kernels

- IScala: mattpap/IScala
- ISpark: tribbloid/ISpark
- Toree: apache/incubator-toree
- twosigma: twosigma/beakerx
- Spylon: maxpoint/spylon-kernel
- jupyter-scala: jupyter-scala/jupyter-scala
Default Scala REPL

$ scala

$ sbt console

• No pretty-printing

• Can’t add dependencies on-the-fly

• Shipped with compiler
Scala shells

• Spark
  
  $ bin/spark-shell

• Flink
  
  $ bin/start-scala-shell.sh

• scio (Scala API for Apache Beam)
  
  $ scio-repl

• ...

• …
scala-notebook

Bridgewater/scala-notebook

- Standalone kernel + notebook server
- UI based on IPython
- Server written in Scala
- akka instead of ZMQ
- Discontinued?
spark-notebook

spark-notebook/spark-notebook

- Fork of scala-notebook
- Switch to spark-repl
- Play
- UI based on Jupyter
Zeppelin

- Plenty of interpreters
- Plenty of *Scala* interpreters
  - Spark
  - Flink
  - Scio
  - …
Scala kernels / shells

- All based on the Scala default shell
  - parsing, compilation, running things
  - value printing
  - special commands
- Custom
  - Dependency management (adding / loading libraries)
  - Setting the right options
  - Interfacing with distributed frameworks (Spark, scio, …)
Ammonite

- Initiated by Li Haoyi in 2015
- ammonite.io
- Goals: shell, scripting
- “the IPython of Scala”
- Lots of user friendly features
  - IDE like completion
  - Pretty-printing
  - Syntax color for input
  - Smart way of doing “magics”
  - Heavy caching
  - zsh-like history
- But: doesn’t work with Spark / scio / etc.
Python

- Default shell
  
  $ python

- IPython
  
  $ ipython

- Jupyter notebook
  
  $ jupyter notebook
Why?
Dependency management

- No global install path for libraries
- No need of virtualenv-like things
- No need to ask users to pre-install dependencies
Dependency management

```
libraryDependencies +=
  "io.circe" %% "circe-core" % "0.8.0"
```

Transitive dependencies

```
com.github.mpilquist:simulacrum_2.12:0.10.0
io.circe:circere-core_2.12:0.8.0
io.circe:circere-numbers_2.12:0.8.0
org.scala-lang:scala-library:2.12.2
org.scala-lang:scala-reflect:2.12.0
org.typelevel:cats-core_2.12:0.9.0
org.typelevel:cats-kernel_2.12:0.9.0
org.typelevel:cats-macros_2.12:0.9.0
org.typelevel:macchinist_2.12:0.6.1
org.typelevel:macro-compat_2.12:1.1.1
```

Binaries

```
~/.coursier/cache/.../org/scala-lang/scala-library/2.12.2/scala-library-2.12.2.jar
~/.coursier/cache/.../org/typelevel/cats-kernel_2.12/0.9.0/cats-kernel_2.12-0.9.0.jar
~/.coursier/cache/.../org/scala-lang/scala-reflect/2.12.0/scala-reflect-2.12.0.jar
~/.coursier/cache/.../org/typelevel/cats-core_2.12/0.9.0/cats-core_2.12-0.9.0.jar
~/.coursier/cache/.../com/github/mpilquist/simulacrum_2.12/0.10.0/simulacrum_2.12-0.10.0.jar
~/.coursier/cache/.../org/typelevel/macro-compat_2.12/1.1.1/macro-compat_2.12-1.1.1.jar
~/.coursier/cache/.../org/typelevel/cats-macros_2.12/0.9.0/cats-macros_2.12-0.9.0.jar
~/.coursier/cache/.../org/typelevel/macchinist_2.12/0.6.1/macchinist_2.12-0.6.1.jar
~/.coursier/cache/.../io/circe/circe-numbers_2.12/0.8.0/circe-numbers_2.12-0.8.0.jar
~/.coursier/cache/.../io/circe/circe-core_2.12/0.8.0/circe-core_2.12-0.8.0.jar
```
Dependency management

Project A

```scala
libraryDependencies +=
  "io.circe" %% "circe-core" % "0.8.0"
```

Project B

```scala
libraryDependencies +=
  "io.circe" %% "circe-core" % "0.7.1"
```
Distributed frameworks

• Leverage that to “clone” their environment

• Dependencies (JAR files) sent to the other machines
Distributed frameworks

- No standard way of loading dependencies in the REPL
- No standard way of knowing the whole classpath
  - REPL build products in particular
  - spark-shell, scio-repl, etc. tweak the internals of the REPL to get the classpath
Serialization

• How to move things from machine to machine?

• For data: fast / efficient libraries (Kryo)

• For closures: Java serialization

  • mapping functions on streams, on RDDs, ...

  • user-defined function with spark SQL

```scala
val rdd: RDD[Foo] = ???
rdd.map { foo =>
  foo.bar // compute things
}
```
Java serialization

• Conservative: a class is not serializable by default
  • Fine for connections to databases, etc.
  • Need to explicitly mark classes as serializable
    • 343 “extends Serializable” or “with Serializable” in shapeless (github.com/milessabin/shapeless)
  • The whole ecosystem isn’t on par with this
Serialization

• Things are worse from a REPL perspective

• User code: wrapped by the REPL before compilation

```scala
val n = List(1, 2, 3)
```

becomes

```scala
object cmd1 {
  val n = List(1, 2, 3)
}
```

• What if one deserializes a singleton twice, 3, 4, 5, … times?

• Wrapping must be fine with serialization
Why many shells / kernels?

• Dependency management
  • Really practical in notebooks / REPLs
  • Requires glue code to interface with Spark, etc.

• Serialization
  • Whole ecosystem not on par with it
  • Worse if for a new REPL!
jupyter-scala

- Based on Ammonite
  - Much of the logic outside of the Jupyter kernel
- Modified Ammonite
  - Serialization-friendliness
  - Even more careful dependency management
- Bridges for Spark, etc.
import $ivy.`org.apache.spark::spark-sql:2.1.0`
import $ivy.`org.jupyter-scala::spark:0.4.2`
import org.apache.spark._
import org.apache.spark.sql._
import jupyter.spark.session._

val sparkSession = JupyterSparkSession.builder()
  .master("...")
  .config("spark.executor.instances", "20")
  .config("spark.executor.memory", "4g")
  .getOrCreate()
jupyter-scala

- Status
  - Not funded
  - Not many known major users
  - Relies on customized Ammonite (Ammonite itself quite new)
  - Lack of time
- Other Scala kernels could benefit from that
Thanks