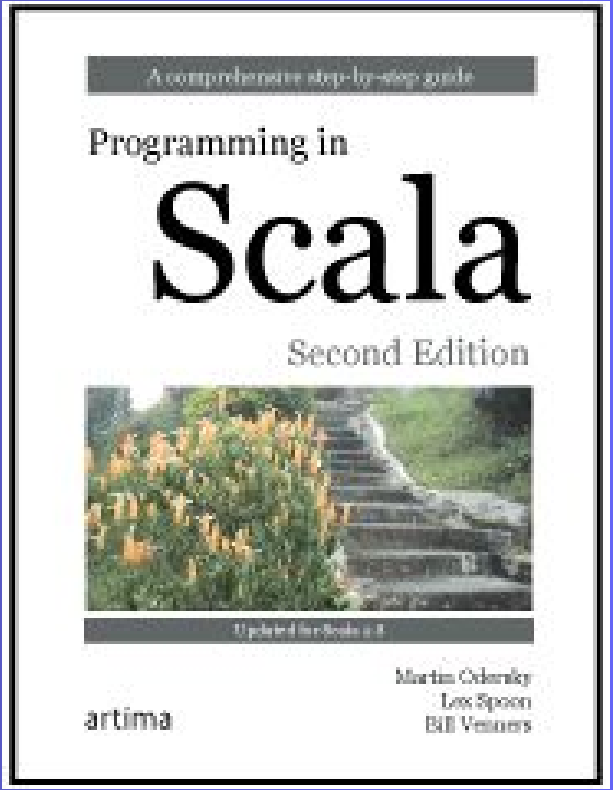
Stairway to Scala - Flight 16

Build tools & integrating with Java

Bill Venners

Dick Wall

escalatesoft.com



Flight 16 goal

A look at build tools, particularly sbt and activator, then a look at mixing Java and Scala (and using Java libraries).

Maven

- Has support for incremental compiler zinc
- Good support for other languages in addition to Scala
- Fast way to get a new Scala project

```
mvn archetype:generate
```

Check/update versions

```
mvn clean test
```

- Many archetypes, including web apps

Gradle

- Scala plugin:

```
apply plugin: ‘scala’
```

- DSL in Groovy

- Zinc support:

```
tasks.withType(ScalaCompile) {
    scalaCompileOptions.useAnt = false
}
```

- Dependencies:

```
dependencies {
  testCompile "org.scala-lang:scala-library:2.11.1"
}
```

Other Options

- Pants: Twitter’s Open Source Build Tool
  Python DSL
  http://pantsbuild.github.io/

- Apache Buildr
  Ruby DSL
  http://buildr.apache.org/

- Ant
  XML DSL :-)
  http://tutorials.jenkov.com/scala/compiling-with-ant.html

- sbt and activator

SBT

- Written in Scala, includes Scala like DSL
- Officially, name means nothing
- Fast Compile/Test, also Continuous
- https://scala-sbt.org

Using sbt

- Interactive mode
  - help & tasks

- Common commands:
  - clean
  - compile
  - project (for multiple project builds)
  - test & test:compile
  - publish, publish-local & publish-signed
  - console & test:console

- ~ commands

sbt structure and requirements

- src
  - main
    - scala
    - java
    - resources
  - test
    - scala
    - java
    - resources
- build.sbt // optional, common
- project // optional, common
  - Build.scala // optional - more power, other names
  - plugins.sbt // optional - other names
  - build.properties // optional

build.sbt

- Easiest way in to sbt
- Scala like DSL, simplified dialect
- 3 main operators:

:=	-	set a value
+=	-	add a value to existing
++=	-	add a sequence of values to existing

- Blank lines between expressions
- Can embed Scala code in {}s

example build.sbt

```
name := """scala-library-seed"""

organization := "com.example"

licenses += ("MIT", url("http://opensource.org/licenses/MIT"))

javacOptions ++= Seq("-source", "1.6", "-target", "1.6")

scalaVersion := "2.10.4"

crossScalaVersions := Seq("2.10.4", "2.11.2")

libraryDependencies ++= Seq(
  "org.scalatest" %% "scalatest" % "2.2.1" % "test"
)

bintraySettings

com.typesafe.sbt.SbtGit.versionWithGit
```

example plugins.sbt

```
resolvers += Resolver.url(
  "bintray-sbt-plugin-releases",
  url("http://dl.bintray.com/content/sbt/sbt-plugin-releases"))(
    Resolver.ivyStylePatterns)

addSbtPlugin("me.lessis" % "bintray-sbt" % "0.1.2")

resolvers += "jgit-repo" at "http://download.eclipse.org/jgit/maven"

addSbtPlugin("com.typesafe.sbt" % "sbt-git" % "0.6.4")
```

Custom settings

```
val isAwesome = settingKey[Boolean]("Some boolean setting")

isAwesome := true

val totally = settingKey[String]("rating of totalness of the statement")

totally := "100% totally"

val totallyAwesome = settingKey[String]("How awesome is this project")

totallyAwesome := totally.value + {
  println("Checking project awesomeness")
  if (isAwesome.value) " awesome." else " not awesome."
}
```

Custom tasks

```
val checkAwesome = taskKey[Unit]("Check project awesomeness")

checkAwesome := {
  val _ = (compile in Test).value
  println("The project is " + totallyAwesome.value)
}
```

Multiple Projects

```
lazy val util = project

lazy val extras = project

lazy val prod = project.dependsOn(util, extras)

lazy val root = project.in(file("."))
  .aggregate(util, extras, prod)
  .settings(aggregate in update := false)
```

Activator

- Typesafe tool
- Keeps itself up to date
- Superset of sbt
- activator new
- activator ui

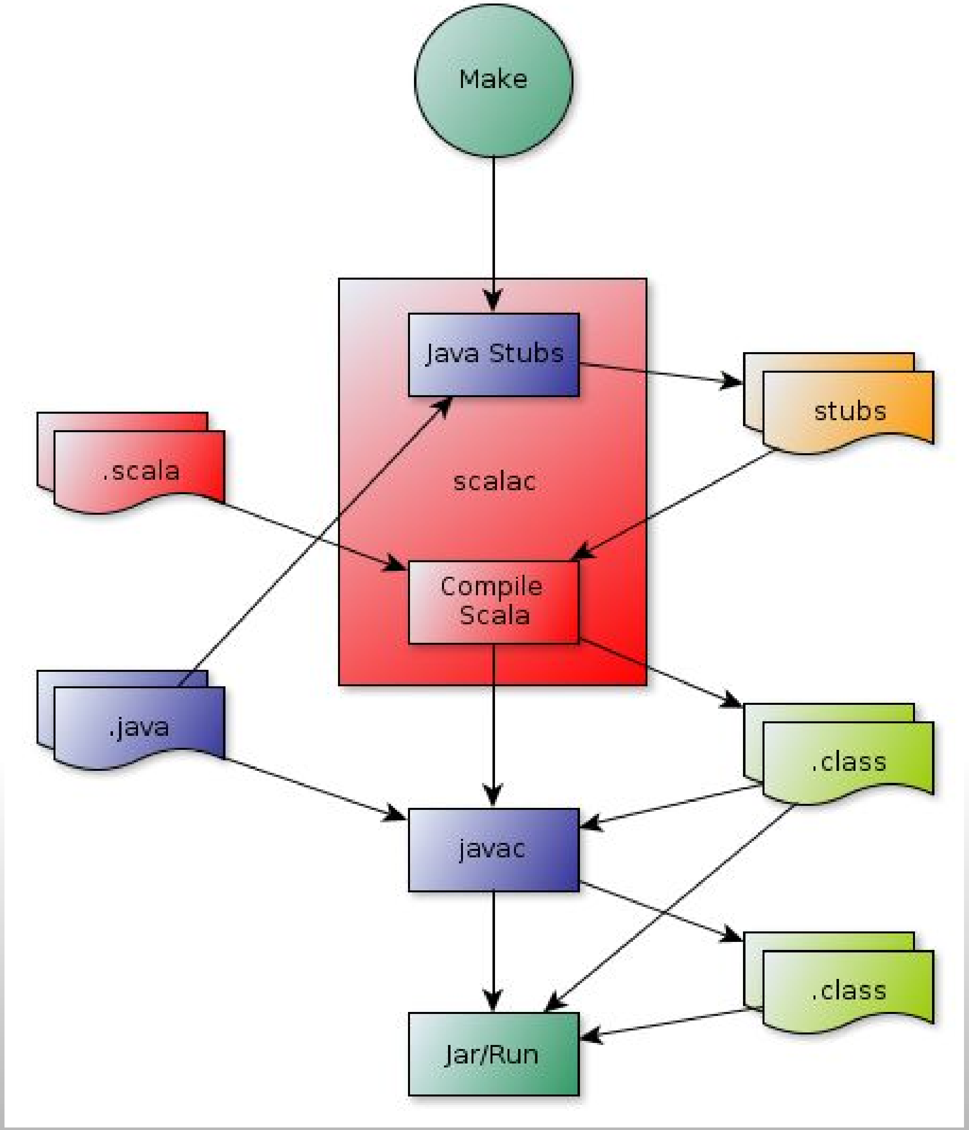
project/*.scala

- More power - full Scala capabilities
- Use for shared code/tasks/settings (between sbt files)
- Can be mixed with build.sbt
- Recommendation, use build.sbt until you need more, and then still keep build.sbt for everything you can

Using Java Libraries

- Scala can use any Java library in addition to Scala libs
- That includes (almost) any Java web framework
- But with mixed success

Scala / Java Compile Cycle



Calling Java from Scala

- Import any Java library
- Call Java methods just like Scala
- Can leave off ()s for empty params
- Can call using infix notation
- Can extend or "with" Java interfaces
- Can instantiate Java classes
- Scala handles conversion to/from primitives

Java 8 Support in Scala 2.12

- Scala 2.12 will require Java 8
- More work for Android (can maybe rewrite binary)
- Scala function literals will compile to method handles
- Scala will support SAMs - Single Abstract Methods
- FunctionN become Java FunctionlInterfaces
- @interface to guarantee trait can be used from Java
- Maybe integrate Java 8 Streams somehow

Nulls from Java

- Nulls discouraged in Scala

```
scala> val a = javaObj.methodCanReturnNull(x)

scala> a.toString                    // oops
java.lang.NullPointerException

scala> val b = Option(javaObj.methodCanReturnNull(x))

scala> b.toString                    // safe
None
```

Nulls to Java

- Methods that expect nulls?

```
scala> val a: Option[String] = Some("Hello")

scala> val b: Option[String] = None

scala> val r1 = javaObj.nullCapable(a.orNull)

scala> val r2 = javaObj.nullCapable(b.orNull)
```

Working with Java Collections

```
scala> val jl = new java.util.ArrayList[Int]
scala> jl.add(1); jl.add(2); jl.add(3)

scala> jl.map(_ * 2)
<console>:7: error: value map is not a member of java.util.
ArrayList[Int]
       jl.map(_ * 2)
          ^

scala> import scala.collection.JavaConverters._

scala> jl.asScala.map(_ * 2)
res1: scala.collection.mutable.Buffer[Int] = ArrayBuffer(2, 4, 6)
```

Implicit conversions not always enough?

```
// Java method signature:
public List<Integer> someJavaFunc(List<Integer> list) { ... }

scala> val sl = List(1, 2, 3)
scala> val r1 = obj.someJavaFunc(sl.asJava)
error: type mismatch;
found    : java.util.List[Int]
required : java.util.List[java.lang.Integer]

scala> val jl = sl.map( new java.lang.Integer(_) )
scala> val r2 = obj.someJavaFunc(jl.asJava)

(works)
```

Using Java Interfaces/Inner Classes

```
// java
public interface Predicate {
  boolean apply(Object o);
}

scala> val isString = new Predicate {
     |   def apply(o: AnyRef): Boolean =
     |     o match {
     |       case s: String => true
     |       case _ => false
     |     }
     | }
scala> isString.apply("Hello")
res4: Boolean = true
```

Using Option from Java

```
// java

Option<String> something = Option.apply(it);
Option<String> nothing = Option.empty();

scalaObj.fnWithOptional(something);
scalaObj.fnWithOptional(nothing);
```

Using Scala Objects/Traits in Java

```
// Scala

trait DoSomethingToString {
  def doIt(s: String): String
}

// Java

class Shout implements DoSomethingToString {
  public String doIt(String s) {
    return s.toUpperCase();
  }
}
```

General Advice

- Java calling Scala
  - Provide empty trait based API around Scala implementation
  - Avoid function literals
  - Convert between nullable and Option
- Scala calling Java
  - Remember scala.collection.JavaConverters
  - Use implicit conversions (respectfully)
  - Remember the REPL

Exercises for Flight 16