sbt

A neophytes perspective

Tim Carter

About Me

- Solution Architect at Verizon Enterprise Services developing B2B and Sales Force support applications
- Recent Experience is primarily with a traditional JEE stack
  - Java/Groovy
  - Spring, Hibernate, …
  - JSP, JSF, Struts
  - Sencha Ext/JS
  - Various Javascript frameworks
- Additional background with C/C++

- Scala - experience compensurate with font size

Topic Outline

- What is sbt ?
- Installation
- Directory Structure
- Anatomy of a sbt build definition
- Basic configuration
- Custom Settings / Tasks
- Scopes
- Plugins
- Testing

What is sbt ?

sbt is:

- Build Tool written in scala, for scala (and java)
- IDE
- REPL
- Task Engine

Alternative build tools

- Ant
- Maven
- Gradle
- …

Sbt Runtime

sbt runtime is nothing more than a wrapper that downloads what is needed:

```
./bin/sbt
./bin/sbt-launch-lib.bash
./bin/sbt-launch.jar
./bin/sbt.bat
./conf/sbtconfig.txt
./conf/sbtopts
```

Downloaded artifacts are stored in ~/.sbt/boot (configurable)

```
.sbt/boot/scala-2.11.1/
.sbt/boot/scala-2.11.1//com.typesafe.activator
.sbt/boot/scala-2.11.1//com.typesafe.activator/activator-launcher
.sbt/boot/scala-2.11.1//com.typesafe.activator/activator-launcher/1.2.12
.sbt/boot/scala-2.11.1//com.typesafe.activator/activator-launcher/1.2.12/activator-
common-1.0-6830c15252733edf977c869af798d113ad5ac80d.jar
.sbt/boot/scala-2.11.1//com.typesafe.activator/activator-launcher/1.2.12/activator-
launcher-1.2.12.jar
.sbt/boot/scala-2.11.1//com.typesafe.activator/activator-launcher/1.2.12/activator-
props-1.2.12.jar
.sbt/boot/scala-2.11.1//com.typesafe.activator/activator-launcher/1.2.12/activator-templates-
cache-1.0-6830c15252733edf977c869af798d113ad5ac80d.jar
.sbt/boot/scala-2.11.1//com.typesafe.activator/activator-launcher/1.2.12/activator-ui-
common-1.2.12.jar
(snip)
```

Build Definition

The build definition is made up a combination of setting and task definitions

There are three flavors of build definition:

- Bare .sbt build definition
- Multi-project .sbt build definition
- .scala build definition

During startup, sbt will scan for the following file patterns and combine key definitions into a comprehensive build definition:

build.sbt
project/*.sbt
project/*.scala
project/project/….

* actually, the file name doesn’t matter as long as it has .sbt or .scala

Directory Structure

sbt follows the Maven directory structure

Source Code

```
src/main/java
   <java sources>
src/main/scala
   <scala sources>
src/main/resources
   <property files, etc>

src/test/java
   <java test sources>
src/test/scala
   <scala test sources>
src/test/resources
   <test resource files>
```

Build Definition

```
build.sbt
project/
    build.properties
    *.scala
    *.sbt
```

Commands and Tasks

There is a technical distinction in sbt between tasks, which are “inside” the build definition, and commands, which manipulate the build definition itself.

Commands

- inspect [tree]
- project, projects
- set
- refresh - reload .sbt files in interactive mode

Common tasks

- clean
- compile
- test
- run
- package

sbt utility tasks

- console - scala REPL
- consoleProject - experiment with sbt build definition
- update - refresh dependencies

Keys

Setting key
value key of some type

Task key
Executable task

Input key
Task key with some command line parsing

Setting Keys

Setting key - value key

```
// declare
lazy val myVersion = settingKey[String](“version key”)
…

// define
myVersion := “1.2.0”
```

Some typical sbt keys:

- name
- version
- scalaVersion
- libraryDependences
- resolvers
- organization

sbt builtin keys defined here:

http://www.scala-sbt.org/0.13.7/api/index.html#sbt.Keys$

Task Key

Execute some build logic, which may return values as input to a dependent task

For example, these tasks simply print to the console

```
lazy val hello   = taskKey[Unit]("hello task key”)
lazy val goodbye = taskKey[Unit]("goodbye task key”)
…
hello   := println("hello, atlanta")
goodbye := println("goodbye, atlanta")

// add dependency
goodbye <<= goodbye.dependsOn(hello)
```

The sbt compile task depends on several subtasks to provide input:

```
compile:compile = Task[sbt.inc.Analysis]
[info]   +-compile:compile::compileInputs = Task[sbt.Compiler$Inputs]
[info]   | +-compile:classDirectory = target/scala-2.11/classes
[info]   | +-*/*:compileOrder = Mixed
[info]   | +-*:compilers = Task[sbt.Compiler$Compilers]
[info]   | +-compile:dependencyClasspath = Task[scala.collection.Seq[sbt.Attributed[jav..
[info]   | +-compile:incCompileSetup = Task[sbt.Compiler$IncSetup]
[info]   | +-*/*:javacOptions = Task[scala.collection.Seq[java.lang.String]]
[info]   | +-*/*:maxErrors = 100
[info]   | +-compile:scalacOptions = Task[scala.collection.Seq[java.lang.String]]
[info]   | +-*/*:sourcePositionMappers = Task[scala.collection.Seq[scala.Function1[xsbt..
[info]   | +-compile:sources = Task[scala.collection.Seq[java.io.File]]
(snip)
```

Input Key

Task capable of parsing input parameters

```
lazy val demo = inputKey[Unit]("demo input task")
```

Ultra basic example

```
demo := {
  // get the result of parsing
  val args: Seq[String] = spaceDelimited("<args>").parsed

  println("The current Scala version is " + scalaVersion.value)
  println("The arguments to demo were:")
  args foreach println
}
```

```
$ sbt “demo a b”

The current Scala version is 2.11.4
The arguments to demo were:
a
b
```

Basic Build Configuration

build.sbt

```
name         := “hello”
version      := “1.0”
scalaVersion := “2.11.6”


resolvers +=
  "Sonatype OSS Snapshots" at “https://oss.sonatype.org/content/repositories/snapshots"

// Individually
libraryDependencies += "org.scalatest" %% "scalatest" % vScalaTest % "test"
libraryDependencies += "junit" % "junit" % "4.10" % "test"

// as a sequence
libraryDependencies ++= Seq(
    jdbc, anorm, cache,
    "org.webjars" %% "webjars-play" % "2.3.0-2"
  )
```

project/build.properties

```
sbt.version=0.13.7
```

Library Dependencies

Specify additional Maven repositories

```
resolvers +=
  "Sonatype OSS Snapshots" at “https://oss.sonatype.org/content/repositories/snapshots"
```

Specify additional maven dependencies

```
// Individually
libraryDependencies ++= "org.webjars" %% "webjars-play" % "2.3.0-2"

// As a sequence
libraryDependencies ++= Seq(
    jdbc, anorm, cache,  // vals exported by play plugin
    "org.webjars" %% "webjars-play" % "2.3.0-2"
  )
```

*Also note that **%%** before the artifact name will fetch the version specific to scalaVersion*

The following are identical, given [scalaVersion := “2.11.x”]

```
"org.webjars" %% "webjars-play"      % “2.3.0-2”  // scalaVersion added to name
“org.webjars” %  “webjars-play_2.11" % “2.3.0-2”
```

Custom Settings and Tasks

build.sbt with custom settings and tasks

```
import sbt.Keys._

name         := "basic+",
version      := "1.0",
scalaVersion := “2.11.4",

// declare custom settings
lazy val foo     = settingKey[String]("foo key")

// declare custom tasks
lazy val hello   = taskKey[Unit]("hello task key")
lazy val goodbye = taskKey[Unit]("goodbye task key")

// define task values
foo      := "bar"
hello    := println("hello, atlanta”),
goodbye := println("goodbye, atlanta")

// add dependency
goodbye <<= goodbye.dependsOn(hello)

libraryDependencies += "org.scalatest" %% "scalatest" % "2.1.6" % "test"
```

Projects

The following depicts a multi-project structure consisting of sub-modules within the main project folder:

```
import sbt.Keys._
import Dependencies._
import BuildSettings._

name := "multi"


lazy val commonSettings = Seq(
     scalaVersion := "2.11.4",
     version := "1.0",
     foo := "bar",
     hello := println( s"hello, ${foo.value}" ),
     libraryDependencies ++= libs ++ testLibs
)

lazy val root = (project in file("."))
     .dependsOn(core) // root has a code dependency on core
     .aggregate(core) // build core and root together
     .settings(commonSettings: _*)
     .settings(
          description := "root project"
     )

.lazy val core = (project in file("core"))
     .settings(commonSettings: _*)
     .settings(
          foo := "baz",
          description := "core project"
     )
```

Scopes

Each key can have an associated value in more than one context, called a “scope.”

Scope Axis

- Project
- Configurations
  - Compile
  - Test
  - Runtime

For example, the following will add a source file to the compile source path:

```
sources in Compile += file("src/other/scala/Other.scala")
```

Scopes are specified like this:

```
{<build-uri>}<project-id>/config:intask::key
```

Example Scoped Settings

```
fullClasspath (default scope)
test:fullClasspath (within test scope)
util/compile:libraryDependencies (compile libs within util project)
```

http://www.scala-sbt.org/0.13/tutorial/Scopes.html

Testing

The standard source locations for testing are:

- Scala sources in src/test/scala/
- Java sources in src/test/java/
- Resources for the test classpath in src/test/resources/

Supported test frameworks:

- specs2
- ScalaCheck
- ScalaTest

To include scalacheck, include the following (note the “test” scope):

```
// Add scalacheck dependency in the test configuration
libraryDependencies += "org.scala-tools.testing" %% "scalacheck" % "1.9" % “test"
```

Test related tasks:

```
test
testOnly
```

Plugins

A plugin extends the build definition, add settings, tasks, or boilerplate behavior

project/plugins.sbt (name does not matter)

```
addSbtPlugin("com.typesafe.play" % "sbt-plugin" % "2.3.8")
addSbtPlugin("com.typesafe.sbt" % "sbt-coffeescript" % "1.0.0")
addSbtPlugin("com.typesafe.sbt" % "sbt-less" % "1.0.0")
…
```

build.sbt

```
…
libraryDependencies ++= Seq(
  // vals provided by play plugin
  jdbc,   anorm,  cache, ws,
)
```

Available plugins:
http://www.scala-sbt.org/0.13/docs/Community-Plugins.html

Resources

http://jsuereth.com/scala/2013/06/11/effective-sbt.html

http://danielwestheide.com/talks/scaladays2014/slides

https://twitter.github.io/scala_school

http://www.scala-sbt.org/0.13/tutorial/index.html

https://jazzy.id.au/2015/03/04/sbt-declarative-dsl.html

This presentation

https://github.com/twcarter/sbt-preso

??