EBook Gratuito

APPRENDIMENTO

sbt

Free unaffiliated eBook created from
Stack Overflow contributors.

#sbt

Sommario

Di

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: sbt

It is an unofficial and free sbt ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official sbt.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capitolo 1: Iniziare con sbt

Osservazioni

Il Simple Build Tool (in breve SBT) può essere usato per costruire il codice del progetto Scala (o Java). Ciò include la gestione del codice, delle dipendenze e delle risorse che devono essere costruite, testate e / o compilate su un `.jar` o altro artefatto. È possibile creare attività personalizzate per gestire tutti questi processi.

Una nota sul nome; A volte SBT viene chiamato 'Scala Build Tool'. Anche se questo non era l'intento originale, è diventato anche comunemente usato. SBT può essere utilizzato per costruire qualsiasi progetto sulla JVM.

`.sbt` file `.sbt` o 'SBT build definitions' sono file interpretati appositamente, scritti in Scala, che vengono utilizzati da SBT per definire una build. `.scala` definizioni di build `.scala` possono anche essere scritte e importate in un file `.sbt` .

Le versioni precedenti alla `13.6` richiedevano che ogni file `.sbt` abbia ciascuna istruzione separata da una riga vuota. Senza la riga vuota, il file `.sbt` si interromperà.

Un pacchetto universale esiste nei formati ZIP e TGZ .

Versioni

Versione	Stato	Data di rilascio
0.13.12	Stabile	2016/07/17

Examples

Installa SBT su Linux

Le istruzioni complete possono essere trovate qui .

1. Installa il JDK .
2. Imposta la variabile di ambiente Java.

```
export JAVA_HOME=/usr/local/java/jdk1.8.0_102
echo $JAVA_HOME
/usr/local/java/jdk1.8.0_102
export PATH=$PATH:$JAVA_HOME/bin/
echo $PATH
...:/usr/local/java/jdk1.8.0_102/bin/
```

3. Installa Scala.

```
sudo wget http://www.scala-lang.org/files/archive/scala-2.11.8.deb
sudo dpkg -i scala-2.11.8.deb
sudo apt-get update
sudo apt-get install scala
```

4. Installa SBT.

```
wget https://bintray.com/artifact/download/sbt/debian/sbt-0.13.9.deb
sudo dpkg -i sbt-0.13.9.deb
sudo apt-get update
sudo apt-get install sbt
```

Distribuzioni Linux basate su RPM

- Scarica le definizioni del repository SBT e aggiungilo a YUM:

```
curl https://bintray.com/sbt/rpm/rpm | sudo tee /etc/yum.repos.d/bintray-sbt-rpm.repo
```

- Installa SBT in base alle definizioni precedentemente aggiunte a YUM:

```
sudo yum install sbt
```

Installa SBT su Windows

Installare

Gli installatori MSI possono essere trovati qui . Questa è l'ultima versione stabile . Scarica ed esegui per installare.

Verifica l'installazione

- Usa `WindowsKey + R` , digita `cmd` .
- In alternativa, accedere a `.sbt` (ad esempio, in `C:\Users\Hopper`) e digitare `cmd` nella barra degli indirizzi.
- Digitare `sbt about` per ottenere informazioni sulla versione, verificando che sia installata. Dovresti vedere qualcosa di simile a questo:

```
Java HotSpot(TM) 64-But Server VM warning: ignoring option MaxPermSize=256m; support was
removed in 8.0
[info] Set current project to root--sbt (in build file:/C:/Users/Hopper/.sbt/)
[info] This is sbt 0.13.8
...
```

Installa su Mac OSX

Le istruzioni ufficiali complete possono essere trovate qui .

MacPorts

Installa MacPorts . Quindi, nel terminale esegui:

```
port install sbt
```

homebrew

Installa Homebrew . Quindi, nel terminale esegui:

```
brew install sbt
```

fonti

Scarica l'installazione di sbt *Tutte le piattaforme* (tgz) da SBT .

```
sudo su
cd /opt
mkdir sbt
cd sbt
curl https://dl.bintray.com/sbt/native-packages/sbt/0.13.13/sbt-0.13.13.tgz -o sbt-0.13.13.tgz
```

Quindi, esegui dopo

```
tar zxf sbt-0.13.13.tgz
ln -s sbt-0.13.13 latest
```

All'interno di $ HOME assicurati di aggiornare ~ / .profile - aggiungendo le seguenti righe

```
export SBT_HOME=/opt/sbt/latest
export PATH=$PATH:$SBT_HOME/bin
```

Verifica

Nel terminale esegui:

```
which sbt
```

Dovresti aspettarti un risultato simile a:

```
/opt/local/bin/sbt
```

Se non si ottiene output, sbt non è installato.

Importa progetto SBT in Eclipse

Ciò presuppone che sia installato sia Eclipse che SBT .

- Installa il plugin SBT per Eclipse dal marketplace di Eclipse.
- Nella directory della riga di comando passare alla directory principale del progetto.

  ```
  $ cd ~/home/sample/project
  ```

- Esegui sbt, che caricherà il progetto.

  ```
  $ sbt
  ```

- Compilare il progetto per garantire che le dipendenze siano ottenibili.

  ```
  > compile
  ```

- Esegui l'operazione di `eclipse` :

  ```
  > eclipse
  ```

- Entra in Eclipse e seleziona l'opzione di menu:

  ```
  File > New > Project From Existing Sources
  ```

- Nella procedura guidata, accedere alla directory del progetto e selezionarla. Eclipse gestirà il resto.

Leggi Iniziare con sbt online: https://riptutorial.com/it/sbt/topic/2351/iniziare-con-sbt

Capitolo 2: Compiti

Examples

Crea una semplice attività

Tutto ciò che è necessario per definire un'attività è una dichiarazione del suo tipo e una descrizione:

```
lazy val exampleTask = taskKey[Unit]("An example task that will return no value.")
```

Poiché `Unit` è il tipo, questa attività è composta interamente da effetti collaterali. Una volta definito, per implementare le azioni:

```
exampleTask := {
  val s: TaskStreams = streams.value
  s.log.info("The example task was executed.")
}
```

Se questi sono definiti in `build.sbt` , puoi caricare il progetto ed eseguirlo:

```
> exampleTask
[info] The example task was executed.
```

Leggi Compiti online: https://riptutorial.com/it/sbt/topic/7542/compiti

Capitolo 3: dipendenze

Examples

Aggiungi una dipendenza della libreria gestita

`libraryDependency` è l' `SettingKey` che gestisce le dipendenze della libreria 'gestita', che sono le dipendenze che vengono scaricate automaticamente, facendo corrispondere le versioni fornite. Per aggiungere una singola dipendenza:

```
libraryDependencies += "com.typesafe.slick" %% "slick" % "3.2.0-M1"
```

La prima parte, `"com.typesafe.slick"` , indica il pacchetto della libreria. La seconda parte, `"slick"` , è la libreria in questione. La parte finale, `"3.2.0-M1"` , è la versione. Poiché la libreria è unita da `%%` verrà utilizzata la versione di Scala fornita dalla `scalaVersion` .

Puoi aggiungere più librerie contemporaneamente usando `++=` :

```
libraryDependencies ++= Seq(
  "com.typesafe.slick" %% "slick" % "3.2.0-M1" % "compile",
  "com.typesafe.slick" %% "slick-hikaricp" % "3.2.0-M1",
  "mysql" % "mysql-connector-java" % "latest.release"
)
```

Ricorda la natura funzionale di Scala, che ti consente di calcolare le dipendenze. Ricorda solo di restituire un `Seq` :

```
libraryDependencies ++= {
  lazy val liftVersion = "3.0-RC3" //Version of a library being used
  lazy val liftEdition = liftVersion.substring(0,3) //Compute a value
  Seq(
    "net.liftweb" %% "lift-webkit" % liftVersion % "compile",  // Use var in Seq
    "net.liftmodules" %% ("ng_" + liftEdition) % "0.9.2" % "compile",  // Use computed var in
Seq
  )  // Because this is the last statement, the Seq is returned and appended to
libraryDependencies
}
```

Aggiungi un repository

Un repository è un luogo in cui SBT cerca `libraryDependencies` . Se la build si lamenta di non trovare una dipendenza, può mancare il repository corretto. All'interno di SBT, i repository sono elencati nel SettingKey `resolvers` :

```
resolvers += "Flyway" at "https://flywaydb.org/repo"
```

Ciò segue la sintassi di "Nome repository" in "posizione url".

Pin Library per Project Version di Scala

Se il tuo progetto ha questo:

```
scalaVersion := 2.11  // Replace '2.11' with the version of Scala your project is running on
```

Quindi puoi usare `%%` per ottenere automaticamente la versione della libreria compilata rispetto alla versione di Scala utilizzata dal progetto:

```
libraryDependencies += "com.typesafe.slick" %% "slick" % "3.2.0-M1"
```

Nota che avere le due righe precedenti equivale ad avere questa riga:

```
libraryDependencies += "com.typesafe.slick" % "slick_2.11" % "3.2.0-M1"
```

Pin Library alla versione specifica di Scala

Una libreria può essere "fissata" a una versione specifica di Scala utilizzando l'operatore `%` tra `groupId` e `artifactId` (le prime due stringhe in una dipendenza di libreria). In questo esempio, aggiungiamo la libreria con l' `artifactId` di `slick` a Scala versione `2.10` :

```
 libraryDependencies += "com.typesafe.slick" % "slick_2.10" % "3.2.0-M1"
```

Leggi dipendenze online: https://riptutorial.com/it/sbt/topic/6760/dipendenze

Capitolo 4: Iniziare con lo sviluppo quotidiano

Examples

Esempio di sviluppo continuo giornaliero con Scala

```
# install sbt with homebrew (if you didn't)
brew install sbt

# - create a new scala project
# - name your project name when asked like: hello-world
sbt new sbt/scala-seed.g8

# go to the new project directory that you named
cd hello-world

# run sbt to open the sbt shell
sbt

# run your project in continuous running mode
~ run

# to continuously see the test outputs
# open up a new terminal tab, run sbt, type:
~ test

# to continuously compile
# open up a new terminal tab, run sbt, type:
~ compile
```

~ usato per operazioni continue in SBT come visto sopra.

Leggi Iniziare con lo sviluppo quotidiano online: https://riptutorial.com/it/sbt/topic/9842/iniziare-con-lo-sviluppo-quotidiano

Capitolo 5: Panoramica della costruzione

Osservazioni

La documentazione ufficiale è su www.scala-sbt.org .

Examples

Struttura della directory

La struttura standard per un progetto costruito da SBT è:

```
projectName/
    build.sbt
    project/
      <SBT sub-build information>
    src/
      main/
        scala/
           <Scala source files>
        java/
           <Java source files>
        resources/
           <Resource files>
      test/
        scala/
           <Scala test files>
        java/
           <Java test files>
        resources/
           <Resource files>
```

Altre directory possono esistere, ma la build si occupa principalmente di queste. Nella directory di base viene inserito `build.sbt` , i cui contenuti sono almeno:

- `name := <name of build>` : questo è il nome del progetto.
- `version := <version number>` : questa è la versione del progetto per il codice downstream a cui fare riferimento.
- `scalaVersion := <version of Scala>` : questa è la versione di Scala a cui è stato costruito il bytecode del progetto.

La directory del `project` è dove vengono posizionati i file `meta-build` (in contrapposizione a quelli di `proper-build`). Questa directory può avere il proprio file `build.sbt` che viene eseguito esattamente nello stesso modo, creando un ambiente per la `proper-build` SBT di build `proper-build` da eseguire. Questo è ricorsivo, quindi la directory del `project` può avere la propria directory di `project` dove si verifica un `meta-meta-build` e così via.

Al momento della creazione, SBT creerà una directory di `target` in cui vengono posizionati i file di classe e altri componenti.

Cheat Sheet

Questo foglio presuppone che tu sia nella directory principale del progetto, contenente `build.sbt` . `$` indica un prompt dei comandi e `>` indica i comandi eseguiti nella console SBT.

Compilare un progetto

```
$ sbt compile
```

Prova un progetto

```
$ sbt test
```

Inserisci SBPL REPL:

```
$ sbt
```

Inserisci Console Scala con progetto incorporato disponibile

```
$ sbt
> console
```

Genera Scaladoc

Questo è un esempio di esecuzione di un 'Task' SBT . Il sito SBT ha ulteriori informazioni sulla generazione di documentazione Scaladoc .

```
$ sbt doc
```

o:

```
$ sbt
> doc
```

Leggi Panoramica della costruzione online: https://riptutorial.com/it/sbt/topic/6761/panoramica-della-costruzione

Capitolo 6: progetti

Examples

Più progetti nella stessa build (sottoprogetti)

A volte una build combina più directory sorgente, ognuna delle quali è il proprio 'progetto'. Ad esempio, potresti avere una struttura di build come questa:

projectName / build.sbt project / src / main / ... test / ... core / src / main / ... test / ... webapp / src / main / ... test / ...

Nel progetto precedente, il codice in `projectName/src` è considerato il progetto `root` . Ci sono altri due moduli, o "sottoprogetti", `core` e `webapp` .

La configurazione di un sottoprogetto è simile alla configurazione del progetto root, tranne per il fatto che la sottodirectory è specificata nel progetto. Questo esempio mostra un progetto root che aggrega un progetto `core` e `webapp` .

```
lazy val root = (project in file(".")).aggregate(core,webapp).dependsOn(core, webapp)

lazy val core = (project in file("core"))

lazy val webapp = (project in file("webapp")).dependsOn(core)
```

I valori passati a `file()` sono le directory relative alla radice del progetto.

Il progetto `webapp` dipende dal progetto `core` , che è indicato dalla clausola `dependsOn` , che prende il valore `core` specificato nella riga sopra. `dependsOn` e `lazy` evaluation assicurano che le dipendenze siano disponibili prima che i progetti le utilizzino. In questo caso, la `webapp` dipende dal `core` , quindi il `core` verrà compilato prima che la build tenti di compilare `webapp` .

`aggregate` rende le attività definite in un progetto disponibili per il progetto che lo aggrega. Ad esempio, l'esecuzione della `compile` nel progetto `root` eseguirà anche la `compile` in `core` e `webapp` .

Configura macro in un progetto

Nel file `build.sbt` (o dove il progetto è definito se si trova in un'altra posizione), aggiungere la seguente impostazione:

```
scalacOptions += "-language:experimental.macros"
```

Ad esempio, un progetto potrebbe essere definito in questo modo:

```
lazy val main = project.in(file("."))  // root project
  .settings(scalacOptions += "-language:experimental.macros",
            addCompilerPlugin("org.scalamacros" % "paradise" % "2.1.0" cross
```

```
CrossVersion.full))
```

Nell'esempio sopra, il plugin `paradise` è incluso per fornire il supporto completo di Scala `2.10.x`

Impostazioni di visualizzazione

Quando nella console SBT, per elencare tutte le impostazioni definibili per un progetto:

```
settings
```

Oppure, per ottenere le impostazioni di un sottoprogetto (ad esempio, nome `webapp`):

```
project webapp
settings
```

La prima riga in alto naviga nel sottoprogetto specifico.

Per mostrare il valore di un'impostazione specifica (ad esempio, `organization`):

```
show organization
```

Questo mostrerà il valore di quell'impostazione.

Leggi progetti online: https://riptutorial.com/it/sbt/topic/6790/progetti

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con sbt	Altius, andriosr, Ani Menon, Community, Eugene Yokota, James , karel, kn_pavan, mko, Nathaniel Ford
2	Compiti	Nathaniel Ford
3	dipendenze	Nathaniel Ford
4	Iniziare con lo sviluppo quotidiano	Inanc Gumus
5	Panoramica della costruzione	Nathaniel Ford
6	progetti	Nathaniel Ford