Бесплатная электронная книга

учусь

sbt

Free unaffiliated eBook created from
Stack Overflow contributors.

#sbt

1
1: sbt 2
2
2
Examples 2
SBT Linux 2
Linux RPM 3
SBT Windows 3
3
3
Mac OSX 4
MacPorts 4
Homebrew 4
4
4
SBT Eclipse 5
2: 6
Examples 6
6
6
Pin Library Scala 7
Pin Library Scala 7
3: 8
Examples 8
8
4: 9
Examples 9
Scala 9
5: 10
10

Examples 10
..... 10
- 11
..... 11
..... 11
SBT REPL: 11
Scala 11
Scaladoc 11
6: 13
Examples 13
() 13
..... 13
..... 14
..... 15

Около

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

глава 1: Начало работы с sbt

замечания

Инструмент простой сборки (SBT для краткости) можно использовать для создания кода проекта Scala (или Java). Это включает в себя управление кодом, зависимостями и ресурсами, которые должны быть созданы, протестированы и / или скомпилированы в `.jar` или другой артефакт. Пользовательские задачи могут быть созданы для управления всеми этими процессами.

Заметка о названии; SBT иногда упоминается как «Инструмент сборки Scala». Хотя это не было первоначальным намерением, оно стало широко использоваться. SBT может использоваться для создания любого проекта на JVM.

`.sbt` или «определения SBT-сборки» - это специально интерпретированные файлы, написанные в Scala, которые используются SBT для определения сборки. `.scala` сборки `.scala` также могут быть записаны и импортированы в файл `.sbt` .

Версии до `13.6` требовали, чтобы в любом файле `.sbt` каждый оператор разделялся пустой строкой. Без пустой строки файл `.sbt` сломается.

Универсальный пакет существует в форматах ZIP и TGZ .

Версии

Версия	государственный	Дата выхода
0.13.12	стабильный	2016-07-17

Examples

Установка SBT в Linux

Полные инструкции можно найти здесь .

1. Установите JDK .
2. Установите переменную среды 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. Установите 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. Установите 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
```

Распределения Linux на основе RPM

- Загрузите определения хранилища SBT и добавьте его в YUM:

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

- Установите SBT в соответствии с определениями, ранее добавленными в YUM:

```
sudo yum install sbt
```

Установка SBT в Windows

устанавливать

Установщики MSI можно найти здесь . Это последняя стабильная версия . Загрузите и выполните установку.

Проверить установку

- Используйте `WindowsKey + R` , введите `cmd` .
- Кроме того, перейдите в `.sbt` (например, в `C:\Users\Hopper`) и введите `cmd` в адресной строке.
- Введите `sbt about` чтобы получить информацию `sbt about` версии, проверяя ее. Вы должны увидеть что-то вроде этого:

```
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
...
```

Установить на Mac OSX

Полные официальные инструкции можно найти здесь .

MacPorts

Установите MacPorts . Затем в терминале выполните:

```
port install sbt
```

Homebrew

Установите Homebrew . Затем в терминале выполните:

```
brew install sbt
```

источники

Загрузите установку sbt *All platform* (tgz) из 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
```

Затем выполните следующее

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

Внутри вашего ДОМЕНА обязательно обновите файл ~ / .profile - добавив следующие строки

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

верификация

В терминале выполните:

```
which sbt
```

Вы должны ожидать, что результат аналогичен:

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

Если вы не получаете выход, sbt не установлен.

Импорт проекта SBT в Eclipse

Предполагается, что вы установили как Eclipse, так и SBT .

- Установите плагин SBT для Eclipse с рынка Eclipse.
- В командной строке перейдите в каталог корневого каталога проекта.

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

- Выполните команду sbt, которая будет загружать проект.

  ```
  $ sbt
  ```

- Скомпилируйте проект, чтобы обеспечить доступность зависимостей.

  ```
  > compile
  ```

- Запустите задачу `eclipse` :

  ```
  > eclipse
  ```

- Перейдите в Eclipse и выберите пункт меню:

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

- В мастере перейдите в каталог проекта и выберите его. Eclipse справится с остальными.

Прочитайте Начало работы с sbt онлайн: https://riptutorial.com/ru/sbt/topic/2351/начало-работы-с-sbt

глава 2: зависимости

Examples

Добавить зависимость управляемой библиотеки

`libraryDependency` - это параметр `SettingKey` который управляет зависимыми библиотечными зависимостями, которые автоматически загружаются зависимостями, соответствующими поставляемым версиям. Чтобы добавить одну зависимость:

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

Первая часть, `"com.typesafe.slick"` , указывает на пакет библиотеки. Вторая часть, `"slick"` , - это библиотека, о которой идет речь. Последняя версия, `"3.2.0-M1"` , является версией. Поскольку к библиотеке присоединяется `%%` использоваться версия Scala, поставляемая `scalaVersion` установки `scalaVersion` .

Вы можете сразу добавить несколько библиотек, используя `++=` :

```
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"
)
```

Помните функциональный характер Scala, позволяющий вычислять зависимости. Не забудьте вернуть `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
}
```

Добавить репозиторий

Репозиторий - это место, где SBT ищет `libraryDependencies` . Если сборка жалуется на то, что не найдет зависимость, в ней может отсутствовать правильный репозиторий. Внутри SBT репозитории перечислены в `resolvers` SettingKey:

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

Это следует за синтаксисом «Название репозитория» в 'url location'.

Pin Library для версии проекта Scala

Если у вашего проекта есть следующее:

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

Затем вы можете использовать `%%` для автоматического получения версии библиотеки, скомпилированной с версией Scala, используемой в проекте:

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

Обратите внимание, что наличие вышеуказанных двух строк эквивалентно наличию этой одной строки:

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

Pin Library для конкретной версии Scala

Библиотеку можно «привязать» к определенной версии Scala, используя оператор `%` между `groupId` и `artifactId` (первые две строки в зависимости от библиотеки). В этом примере мы привязываем библиотеку с `artifactId` `slick` к версии Scala версии `2.10` :

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

Прочитайте зависимости онлайн: https://riptutorial.com/ru/sbt/topic/6760/зависимости

глава 3: Задачи

Examples

Создание простой задачи

Все, что необходимо для определения задачи, - это объявление ее типа и описания:

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

Поскольку `Unit` является типом, эта задача полностью состоит из побочных эффектов. После определения для реализации действий:

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

Если они определены в `build.sbt` , вы можете загрузить проект и выполнить его:

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

Прочитайте Задачи онлайн: https://riptutorial.com/ru/sbt/topic/7542/задачи

глава 4: Начало работы с ежедневной разработкой

Examples

Ежедневный пример непрерывного развития с 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
```

~ используется для непрерывных операций в sbt, как показано выше.

Прочитайте Начало работы с ежедневной разработкой онлайн:
https://riptutorial.com/ru/sbt/topic/9842/начало-работы-с-ежедневной-разработкой

глава 5: Обзор сборки

замечания

Официальная документация находится на сайте www.scala-sbt.org .

Examples

Структура каталога

Стандартная структура для проекта, созданного SBT:

```
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>
```

Другие каталоги могут существовать, но сборка касается прежде всего этих. В базовом каталоге `build.sbt` , содержимое которого как минимум:

- `name := <name of build>` : это имя проекта.
- `version := <version number>` : Это версия проекта для ссылки на нижестоящий код.
- `scalaVersion := <version of Scala>` : это версия Scala, с которой построен байт-код проекта.

Каталог `project` - это место, где размещаются `meta-build` (в отличие от файлов с `proper-build`). Этот каталог может иметь собственный файл `build.sbt` который выполняется точно таким же образом, создавая среду для `proper-build` SBT `proper-build` . Это рекурсивно, поэтому каталог `project` может иметь свой собственный каталог `project` где происходит `meta-meta-build` , и так далее.

После создания SBT создаст `target` каталог, в который помещаются файлы классов и другие компоненты.

Чит-лист

В этом листе предполагается, что вы находитесь в корневой директории проекта, содержащей `build.sbt` . `$` указывает командную строку и `>` указывает, что команды запускаются внутри консоли SBT.

Скомпилировать проект

```
$ sbt compile
```

Проверить проект

```
$ sbt test
```

Введите SBT REPL:

```
$ sbt
```

Войдите в консоль Scala со встроенным проектом.

```
$ sbt
> console
```

Создание Scaladoc

Это пример выполнения задачи «SBT» . Сайт SBT содержит дополнительную информацию о создании документации Scaladoc .

```
$ sbt doc
```

или же:

```
$ sbt
> doc
```

Прочитайте Обзор сборки онлайн: https://riptutorial.com/ru/sbt/topic/6761/обзор-сборки

глава 6: проектов

Examples

Несколько проектов в одной сборке (подпроекты)

Иногда сборка объединяет несколько исходных каталогов, каждый из которых является их собственным «проектом». Например, у вас может быть такая структура сборки, как это:

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

В вышеупомянутом проекте код в `projectName/src` считается `root` проектом. Есть еще два модуля, или «подпроекты», `core` и `webapp` .

Настройка подпроекта аналогична настройке корневого проекта, за исключением того, что в проекте указан подкаталог. В этом примере показан корневой проект, который объединяет `core` и проект `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)
```

Значения, переданные в `file()` являются каталогами относительно корня проекта.

Проект `webapp` зависит от `core` проекта, который обозначается предложением `dependsOn` , которое берет значение `core` указанное в строке выше. `dependsOn` и `lazy` оценка гарантируют доступность зависимостей до их использования. В этом случае `webapp` зависит от `core` , поэтому `core` будет скомпилировано до того, как сборка попытается скомпилировать `webapp` .

`aggregate` задает задачи, определенные в одном проекте, доступным для проекта, который его агрегирует. Например, выполнение `compile` в `root` проекте также выполняет `compile` в `core` и `webapp` .

Настройка макросов в проекте

В файле `build.sbt` (или где проект определен, если он находится в другом месте), добавьте следующий параметр:

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

Например, проект может быть определен следующим образом:

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

В приведенном выше примере, `paradise` плагин включен для того , чтобы обеспечить полную поддержку Scala `2.10.x` .

Настройки экрана

Когда в консоли SBT вы можете указать все определяемые настройки для проекта:

```
settings
```

Или, чтобы получить настройки подпроекта (например, named `webapp`):

```
project webapp
settings
```

Первая строка над навигацией переходит к конкретному подпроекту.

Чтобы показать значение определенного параметра (например, `organization`):

```
show organization
```

Это отобразит значение этой настройки.

Прочитайте проектов онлайн: https://riptutorial.com/ru/sbt/topic/6790/проектов

кредиты

S. No	Главы	Contributors
1	Начало работы с sbt	Altius, andriosr, Ani Menon, Community, Eugene Yokota, James , karel, kn_pavan, mko, Nathaniel Ford
2	зависимости	Nathaniel Ford
3	Задачи	Nathaniel Ford
4	Начало работы с ежедневной разработкой	Inanc Gumus
5	Обзор сборки	Nathaniel Ford
6	проектов	Nathaniel Ford