链滴

SBT 快速了解

作者：shiweichn

原文链接：https://ld246.com/article/1533310985596

来源网站：链滴

SBT 快速了解

---

1. SBT 目录结构和配置文件

1.1 目录结构

```
src/
  main/
    resources/
       <files to include in main jar here>
    scala/
       <main Scala sources>
    java/
       <main Java sources>
  test/
    resources
       <files to include in test jar here>
    scala/
       <test Scala sources>
    java/
       <test Java sources>
```

以上是基本的目录结构，结构和maven等构件工具标准目录基本一样。但是 在项目根路径下会一个 project 的目录，还有一个 build.sbt 的配置文件。

先说下 project 这个目录，这个目录下一般会有 project/ target/ build.properties plugins.sbt 四个东西，前俩是目录，后俩是文件。

- project/ ： SBT 主要借助项目根路径下的project目录中的一些内容来帮助我们构建项目的结构。而roject中的结构是借助于其子目录project来构建的。这个地方理论上可以一直嵌套下去。
- target/ ： 这里的内容应该就是给SBT用来构建项目用的。
- build.properties ： 这里暂时就存了个sbt的版本，更多作用待发掘。
- plugins.sbt ： SBT 插件的默认安装位置。想安装什么插件，写到这个文件里，SBT就会自动安装

例如：

```
addSbtPlugin("net.virtual-void" % "sbt-dependency-graph" % "0.8.2")
addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "0.14.3")
```

1.2 配置文件

SBT默认提供的配置文件是项目根目录下的 build.sbt文件，通过这个文件来配置构建整个项目的信息但是 还可以通过在project目录下的scala文件和build.sbt结合起来配置，但是project下的scala文件定要继承Build，才能被SBT认作为构建信息文件。最终SBT会将这两个文件中的内容合并到一起，作整个项目的构建信息。

---

2. 配置构建信息

2.1 jar包的依赖管理-引入jar

这应该是每个构件工具最基本的功能。如下

```
libraryDependencies += "com.typesafe.akka" % "akka-actor_2.12" % "2.5.6"
// 第一个字符串中的内容是`groupId`，第二个字符串中的内容是`artifactId`,最后一个是`version`。
```

也可以写成下面这个样子

```
libraryDependencies += {
  "com.typesafe.akka" % "akka-actor_2.12" % "2.5.6"
  "com.typesafe.akka" % "akka-agent_2.12" % "2.5.6"
  "com.typesafe.akka" % "akka-cluster_2.12" % "2.5.6"
}
```

2.2 jar包的依赖管理-排除jar

有时候引入一个jar包，会被依赖引入其他jar包。造成jar冲突。这个时候就要排除掉造成冲突的jar。下：

```
libraryDependencies += {
 "net.sf.json-lib" % "json-lib" % "2.4" exclude("commons-beanutils", "commons-beanutils")
}
```

使用 exclude 可以排除不想引入的jar。同样第一个参数是groupId，第二个是artifactId。

2.3 jar包的依赖管理-仓库中找不到jar

有时候仓库中找不到指定的jar，这个时候有两个办法，一个是指定本地的jar，另一个是使用from去络上下载指定的jar。

1.添加本地jar

首先在build.sbt中指定unmanagedBase的值，这样SBT在构建项目的时候会将这个目录的jar给添加项目的classpath中。

```
unmanagedBase := baseDirectory.value / "lib"
```

指定unmanagedBase 对应的路径为项目根目录下的 lib 目录。

2.使用 from,如下：

```
"com.typesafe.akka" % "akka-transactor-2.11" % "2.3.9" from "http://repo1.maven.org/maven
/com/typesafe/akka/akka-transactor_2.11/2.3.9/akka-transactor_2.11-2.3.9.jar"
```

2.4 jar包的依赖管理-jar包仓库地址

```
resolvers += Resolver.url("aliyun", url("http://maven.aliyun.com/nexus/content/groups/public
```

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

3. 编译参数和启动参数

在build.sbt 中， fork := true 时，sbt执行run命令时会新启动一个JVM来给项目运行，如果fork := false(默认为false)时，项目运行时会在sbt运行的这个JVM上。

```
javacOptions ++= Seq("-encoding", "UTF-8")
javacOptions ++= Seq("-source", "1.8", "-target", "1.8")

javaOptions += "-noverify"
javaOptions += ("-Drebel.dirs=" + target.value / "scala-2.12/classes")
javaOptions += "-Drebel.disable_update=true"
javaOptions += "-Xms500m"
javaOptions += "-Xmx500m"
javaOptions += "-XX:+UseG1GC"
javaOptions += "-XX:MaxGCPauseMillis=100"
```

通过上面示例可以在项目编译或启动时做一些操作。这些参数通过sbt的命令都可以看到。如：

```
D:\gitRepos\SGameServer>sbt
"C:\Users\Administrator\.sbt\preloaded\org.scala-sbt\sbt\"1.0.2"\jars\sbt.jar"

[info] Loading project definition from D:\gitRepos\SGameServer\project
[info] Set current project to SGameServer (in build file:/D:/gitRepos/SGameServer/)
> show javaOptions
is windows platform
[info] * -noverify
[info] * -Drebel.dirs=D:\gitRepos\SGameServer\target\scala-2.12\classes
[info] * -Drebel.disable_update=true
[info] * -Xms500m
[info] * -Xmx500m
[info] * -XX:+UseG1GC
[info] * -XX:MaxGCPauseMillis=100
[info] * -agentpath:D:\gitRepos\SGameServer\jrebel64.dll
[success] Total time: 0 s, completed 2017-10-17 17:42:47
```

在build.sbt这个文件中还可以写逻辑代码，如：

```
lazy val switchTask = taskKey[String]("Switch Jrebel By Platform !")
switchTask := System.getProperty("os.name").toUpperCase
javaOptions += {
  if (switchTask.value.contains("WINDOWS")) {
    println("is windows platform")
    "-agentpath:" + baseDirectory.value / "jrebel64.dll"
  } else {
    println("is linux platform")
    "-agentpath:" + baseDirectory.value / "libjrebel64.so"
  }
}
```

通过这个可以在不同系统下给运行指定不同的参数。

3.1 编译项目和启动项目

编译：

在 sbt的console中执行 compile这个命令即可编译整个项目，若使用~ compile可以实现当有文件动并保存后进行自动编译。

启动：

在 sbt中执行run或runMain都可以启动项目，前者不用指定程序入口，在console中，sbt会将所以口展示给你选择。后者需要指定程序入口。

至于停止运行项目，好像只有Ctrl+C来结束，而且Linux环境下，在sbt的console启动项目再执行Ctrl C会将sbt进程与项目进程都给结束。但是windows环境下这样操作只能结束sbt进程。

---

4. SBT 插件

在plugins.sbt文件中可以进行插件添加。如下：

```
addSbtPlugin("net.virtual-void" % "sbt-dependency-graph" % "0.8.2")
addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "0.14.3")
```

形式就是这个样子，第一个插件是可以将项目当前的jar依赖关系生产一个网页，方便查看和排查。第个插件是用来将项目打成jar包的。更多功能和更多插件待发掘。

5. SBT Console

在项目的根路径是运行sbt，进入 SBT的控制台。通过help命令可以查看帮助。如：

```
> help
  about                    Displays basic information about sbt and the build.
  tasks                    Lists the tasks defined for the current project.
  settings                 Lists the settings defined for the current project.
 ......
```

SBT 的配置定义主要是在 Keys.scala 这个文件中，所以可以通过sbt 的console可以查看这些配置值如：

```
> name
[info] SGameServer
> version
[info] 1.0
> baseDirectory
[info] D:\gitRepos\SGameServer
> unmanagedBase
[info] D:\gitRepos\SGameServer\lib
>
```

sbt console中的 reload 命令可以将 构建信息重新载入 当前sbt环境，不管是 build.sbt 还是 build.scala中的。所以改了构建信息后，需要在console中reload下才会生效。

6. SBT Tasks

SBT中的Tasks部分，内容比较多。我们也可以自定义task。查看所有任务：

```
> tasks

This is a list of tasks defined for the current project.
It does not list the scopes the tasks are defined in; use the 'inspect' command for that.
Tasks produce values.  Use the 'show' command to run the task and print the resulting value.

  compile          Compiles sources.
  doc              Generates API documentation.
  dependencyGraph  Prints the ascii graph to the console
  run              Runs a main class, passing along arguments provided on the command line.
  ......
```

单独使用tasks 命令只能看到SBT提供的任务，使用 -v 参数可以看到我们自定义的参数。

在sbt直接指向任务名称即可调用任务，其实执行compile 、run等命令就是在执行任务。这不过这些SBT提供的任务而已。

自定义任务：

```
lazy val switchTask = taskKey[String]("Switch Jrebel By Platform !")
switchTask := System.getProperty("os.name").toUpperCase
```

在console直接调用 switchTask 这个任务就执行了。如：

```
> switchTask
[success] Total time: 0 s, completed 2017-10-17 18:03:48
```

最后

以上就是SBT的基本配置信息。其实 Keys.scala这个文件很重要，如果有什么需求不知道怎么配置，以看下这个类中的定义。Tasks这块内容很多，也很有用，有时间可以多研究一下，每个任务可以配自己的Scope。但是好像任务一定要手动调用才能执行。插件也能帮我们省很多事。console下的help提供基本的帮助。官方文档+help+Keys.scala 应该就可以解决大部分问题。

http://www.scala-sbt.org/1.x/docs/index.html

https://unmi.cc/tag/sbt/

https://github.com/ReactivePlatform/Notes/issues/13