EPFL

sbt, Keys, and Scopes

Effective Programming in Scala

Scoping

We have seen that the source directories are different for the program and its tests.

However, there is a single key `sourceDirectory` (there is no such key as `testSourceDirectory`).

How does the test task find the test sources?

Scoping

We have seen that the source directories are different for the program and its tests.

However, there is a single key `sourceDirectory` (there is no such key as `testSourceDirectory`).

How does the test task find the test sources?

- A key can have $\underline{\text{different values}}$ in different **scopes**.

Scoping (2)

```
sbt:hello-sbt> Compile / sourceDirectory
src/main

sbt:hello-sbt> Test / sourceDirectory
src/test
```

Scoping (2)

```
sbt:hello-sbt> Compile / sourceDirectory
src/main

sbt:hello-sbt> Test / sourceDirectory
src/test
```

- There is a single concept of source directory, modeled by the key `sourceDirectory`, reused by both the `Compile` and `Test` configurations by scoping the key to the corresponding configuration.

Scoping (3)

Each key can be assigned a value along a **configuration** such as `Compile`, `Test`, or no specific configuration (a.k.a `Zero`).

When we look up the value of a key we can specify the configuration we are interested in. If no configuration is specified, sbt first tries with the `Compile` configuration and falls back to the `Test` configuration. For instance, `run` is equivalent to `Compile / run`.

Conversely, if we look up for `Compile / scalaVersion`, and that the key `scalaVersion` has no value in that scope, sbt falls back to a more general scope: it looks up in the `Zero` configuration.

Task Scoping

Configurations are one possible axis of key scoping.

Keys can also have different values according to a particular **task** key.

For instance, the task `unmanagedSources` lists all the project source files.

```
sbt:hello-sbt> show unmanagedSources
[info] * src/main/scala/hellosbt/Main.scala
```

The task can be configured by changing the value of the setting `includeFilter`:

```
sbt:hello-sbt> unmanagedSources / includeFilter
[info] ExtensionFilter(java,scala)
```

By default, sbt looks for source files with extension `.java` and `.scala`.

Task Scoping (2)

Let's also include `.sc` files!

```
// File build.sbt
unmanagedSources / includeFilter := new io.ExtensionFilter(
  ”java”,
  ”scala”,
  ”sc”
)
```

And then:

```
sbt:hello-sbt> show unmanagedSources / includeFilter
[info] ExtensionFilter(java,scala,sc)
```

Project Scoping

There is a third axis that can be used to assign values to sbt keys.

When a project contains sub-projects, each sub-project sets its own values for some keys. This is typically the case for the setting `baseDirectory`, which defines the root directory of each sub-project.

In our build definition example, we only have one project, so all our settings are scoped to this project. We can explicitly see that by prefixing the name of a key with the name of our project, `hello-sbt`:

```
sbt:hello-sbt> hello-sbt / sourceDirectory
[info] src
```

Project Scoping (2)

There is a special sub-project named `ThisBuild`, which means the “entire build”, so a setting applies to the entire build rather than a single project.

sbt falls back to `ThisBuild` when you look for the value of a key that has not been defined for a specific project.

This is a convenient way to define cross-project settings:

```
// Set the Scala version for all the projects in this build definition
ThisBuild / scalaVersion := ”3.0.0”
```

Combining Multiple Scope Axes

Here is how we can see the value of the `includeFilter` key according to multiple axes:

```
// current project, no configuration, unmanagedSource task
unmanagedSources / includeFilter

// hello-sbt project, no configuration, unmanagedSource task
hello-sbt / unmanagedSources / includeFilter

// hello-sbt project, Compile configuration, unmanagedSource task
hello-sbt / Compile / unmanagedSources / includeFilter
```

Summary

When the same concept (e.g., a source directory) is reused in several contexts such as configurations (e.g., the program and its tests), projects, or tasks, sbt encourages you to use a single setting key for this concept and to scope the value you assign to it to the desired context.