Azzolini Riccardo 2021-04-26

Package

1 Package

In Scala le classi e i singleton object (e un altro tipo di definizioni, i trait, che verranno presentati a breve) sono organizzati in **package**. Come in Java, l'appartenenza di una classe a un package è dichiarata all'inizio del file sorgente (l'*unità di compilazione*) che la contiene mediante la clausola `package`. Ad esempio, un file contenente

```
package pfun.examples

object Hello {
  // ...
}
```

dichiara che il singleton object `Hello` appartiene al package `pfun.examples`.

Ogni *membro* di un package, cioè ogni risorsa (classe, object o trait) definita a un package, ha un **nome completo** (**fully qualified name**) formato dal nome del package seguito dal **nome semplice** della risorsa stessa (quello che viene specificato quando si definisce la risorsa), separati da un punto. Nell'esempio precedente, l'object definito ha nome semplice `Hello` e nome completo `pfun.examples.Hello`.

Per convenzione, i file sorgente dei package vengono tipicamente organizzati nel file system come in Java: a ogni package corrisponde una directory, e si trattano i caratteri `.` nei nomi dei package come un'astrazione del separatore di directory; ad esempio, il package `pfun.examples` corrisponde alla directory `pfun/examples`. Tecnicamente, però, Scala permette di organizzare i sorgenti in modo arbitrario, indipendentemente dai nomi dei package.

2 Utilizzo

Quando si ha una risorsa definita in un package, ad esempio

```
package pfun.util

class Rational {
  // ...
}
```

e la si vuole utilizzare in un altro package, è possibile far riferimento a essa per mezzo del suo nome completo,

```
object Test {
  val x = new pfun.util.Rational(1, 2)
}
```

oppure è possibile importarla e utilizzarla mediante il suo nome semplice (facendo però attenzione a evitare conflitti tra i nomi importati e/o definiti in un file, che avvengono secondo le stesse regole di Java):

```
import pfun.util.Rational

object Test {
  val x = new Rational(1, 2)
}
```

3 Clausole di importazione

L'importazione dei package o dei loro membri avviene tramite la **clausola di importazione**, `import`, che è molto più flessibile rispetto all'analoga clausola esistente in Java.

Le importazioni si possono suddividere in due principali forme:

- Un **named import** elenca esplicitamente i nomi di uno o più membri di un package da importare, usando eventualmente le parentesi graffe per specificare più membri di uno stesso package. Ad esempio, la clausola

  ```
  import pfun.util.Rational
  ```

  importa il singolo membro `pfun.util.Rational`, mentre la clausola

  ```
  import pfun.util.{Rational, Other}
  ```

  importa i membri `pfun.util.Rational` e `pfun.util.Other`.

- Un **wildcard import** importa tutti i membri di un package. A differenza di Java, che usa l'asterisco (`*`), in Scala le wildcard import sono indicate con l'underscore (`_`),

  ```
  import pfun.util._
  ```

  opzionalmente scritto tra parentesi graffe:

  ```
  import pfun.util.{_}
  ```

Le clausole di importazione possono occorrere in qualunque punto del codice sorgente, non solo all'inizio del file come in Java. Ciò permette di importare i membri di un package solo nello scope in cui effettivamente servono. Ad esempio:

```
object Test {
  import pfun.util.Rational

  // Qui la classe Rational è importata
  val x = new Rational(1, 2);
}

// Qui la classe Rational non è importata
```

3.1 Importazione dei membri di un oggetto

Scala permette di importare i *membri di un oggetto*, cioè di un singleton object o di un'istanza di una classe, in modo da poter accedere a tali membri senza bisogno di specificare il nome dell'oggetto. Alcuni esempi sono:

```
class Fruit(fruitName: String, fruitColor: String) {
  val name = fruitName
  val color = fruitColor
}
val apple = new Fruit("apple", "red")

import apple.{name, color}
println(name + " - " + color)
// oppure
import apple._
println(name + " - " + color)

object Fruits {
  val orange = new Fruit("orange", "orange")
  val pear = new Fruit("pear", "yellowish")
}

import Fruits._
println(pear.color)
```

Questo tipo di importazione può essere molto comodo se si deve accedere spesso ai membri di uno stesso oggetto, ma per evitare conflitti tra i nomi è spesso preferibile utilizzarla all'interno di un opportuno scope, piuttosto che al livello dell'intero file.

3.2 Importazione di un package

Un altro tipo di importazione che Scala consente (e Java no) è quella di un *intero package* (da non confondere con l'importazione di tutti i membri di un package), in seguito alla quale il package è utilizzabile attraverso il suo nome relativo (la parte del suo nome dopo l'ultimo punto).

Ad esempio, con la clausola

```
import java.util.regex
```

il package `java.util.regex` diventa accessibile tramite il nome semplice `regex` (di fatto, si crea un alias del package):

```
class AStarB {
  val pat = regex.Pattern.compile("a*b")
}
```

3.3 Renaming e hiding

Per risolvere più comodamente i conflitti tra i nomi, Scala permette di rinominare e nascondere i membri importati tramite delle apposite clausole:

- Una **renaming clause**, *original-name* `=>` *new-name*, importa il membro chiamato *original-name* non con il suo nome originale, bensì con il nome *new-name*. Ad esempio, un caso tipico di conflitto tra i nomi sono le classi `java.util.Date` e `java.sql.Date`; in Scala, è possibile importarle entrambe evitando conflitti se si rinomina almeno una delle due:

  ```
  import java.util.Date
  import java.sql.{Date => SDate}
  ```

  Così, il nome `Date` si riferisce a `java.util.Date`, mentre `java.sql.Date` va usata tramite il nome `SDate`.

  Le renaming clause possono anche essere usate nei wildcard import, se si vogliono importare tutti i membri rinominandone solo alcuni. Ad esempio, l'importazione

  ```
  import java.sql.{Date => SDate, _}
  ```

  importa `java.sql.Date` con il nome `SDate`, e tutti gli altri membri di `java.sql` con i loro nomi semplici originali.

- Le **hiding clause**, *original-name* `=> _`, possono essere usate nei wildcard import per indicare di importare tutti i membri tranne alcuni. Ad esempio, la clausola

  ```
  import java.sql.{Date => _, _}
  ```

  importa tutti i membri di `java.sql` tranne `java.sql.Date`.

Le renaming e hiding clause possono essere applicate a tutti i tipi di elementi che si possono importare, cioè non solo ai membri dei package ma anche agli interi package e ai membri degli oggetti. Ad esempio, la clausola

```
import java.{sql => S}
```

importa l'intero package `java.sql` con il nome `S` (dunque per indicare `java.sql.Date` si potrebbe scrivere `S.Date`), mentre la clausola

```
import apple.{name => fruitType}
```

importa con il nome `fruitType` il membro `name` dell'istanza di `Fruit` associata al nome `apple` (definito in precedenza), e la clausola

```
import Fruits.{pear => _, _}
```

importa tutti i membri del singleton object `Fruits` (anch'esso definito prima) tranne `pear`.

4 Importazioni automatiche

In ogni file sorgente Scala (e nell'interprete) sono importati automaticamente:

- tutti i membri del package `scala`, che definisce sostanzialmente i tipi di base (ad esempio `scala.Int` e `scala.Boolean`);
- tutti i membri del package `java.lang` (le classi che sono importate automaticamente anche in Java, come ad esempio `java.lang.Object`);
- tutti i membri del singleton object `scala.Predef`, che sono le funzioni che si hanno sempre a disposizione (ad esempio `scala.Predef.require`).

Tutto il resto richiede un'importazione esplicita.