Funktionale Programmierung (in Scala)

Jan Albert

1. Dezember 2018

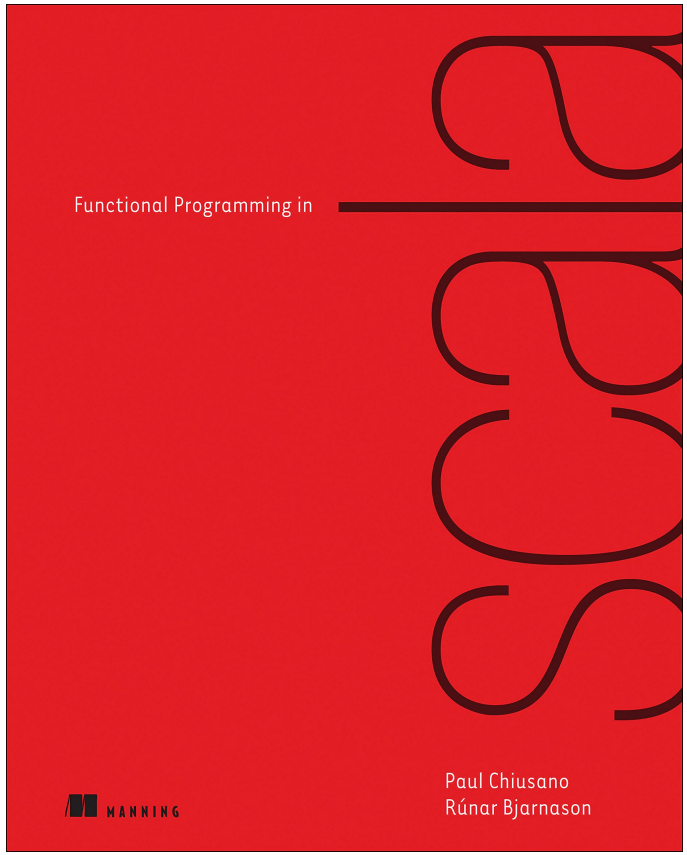
Inhaltsverzeichnis

Buch

Functional Programming in Scala

Paul Chiusano, Runar Bjarnason

Manning, 2014



Was ist Funktionale Programmierung?

Idee: Benutzt ausschließlich "reine Funktionen" d. h. Funktionen, welche keine Seiteneffekte haben.

Beispiele für Seiteneffekte:

- Verändern/Modifizieren einer Variable
- Verändern/Modifizieren einer Datenstruktur
- Eine Exception werfen
- Konsolen Eingabe/Ausgabe
- Lesen/Schreiben aus/von einer Datei

Reine Funktion

Definition (Reine Funktion)

Eine *reine Funktion* mit Eingabetyp A und Ausgabetyp B (Schreibweise: $A \Rightarrow B$) ist eine Berechnung, welche jeden Wert a vom Typ A genau einen Wert b vom Typ B zuordnet, sodass b nur aus dem Wert von a bestimmt wird.

Beispiele:

- Eine Funktion intToString vom Typ Int $\Rightarrow$ String bildet jede ganze Zahl auf einen String ab und macht nichts anderes.
- Die Addition von ganzen Zahlen.

Ausdruck

Definition (Ausdruck)

Jeder Teil eines Programms, welcher zu einem Ergebnis zusammengefasst werden kann d. h. alles was man in den Scala-Interpreter tippen kann und ein Ergebnis liefert, nennt man einen *Ausdruck*.

Beispiel: $2 + 3$ ist ein Ausdruck, welcher die reine Funktion $+$ vom Typ (Int, Int) $\Rightarrow$ Int auf 2 und 3 anwendet.

Referenziell Transparent (RT)

Definition (Referenziell Transparent (RT))

Ein Ausdruck e ist *Referenziell Transparent (RT)*, wenn für alle Programme p, alle Vorkommnisse von e in p durch das Ergebnis von e ersetzt werden können, ohne die Bedeutung von p zu ändern. Eine Funktion ist rein, wenn der Ausdruck $f(x)$ referenziell transparent für alle referenziell transparenten x ist.

Beispiel für RT

```
scala> val x = "Hello, World"
x: java.lang.String = Hello, World

scala> val r1 = x.reverse
r1: java.lang.String = dlroW ,olleH

scala> val r2 = x.reverse
r2: java.lang.String = dlroW ,olleH
```

Beispiel für RT

```
scala> val r1 = "Hello, World".reverse
r1: java.lang.String = dlroW ,olleH

scala> val r2 = "Hello, World".reverse
r2: String = dlroW ,olleH
```

Gegenbeispiel für RT

```
scala> val x = new StringBuilder("Hello")
x: java.lang.StringBuilder = Hello

scala> val y = x.append(", World")
y: java.lang.StringBuilder = Hello, World

scala> val r1 = y.toString
r1: java.lang.String = Hello, World

scala> val r2 = y.toString
r2: java.lang.String = Hello, World
```

Gegenbeispiel für RT

```
scala> val x = new StringBuilder("Hello")
x: java.lang.StringBuilder = Hello

scala> val r1 = x.append(", World").toString
r1: java.lang.String = Hello, World

scala> val r2 = x.append(", World").toString
r2: java.lang.String = Hello, World, World
```

Beispiel mit Seiteneffekt

```
1 class Cafe {
2   def buyCoffee(cc: CreditCard): Coffee = {
3     val cup = new Coffee()
4     cc.charge(cup.price)
5     cup
6   }
7 }
```

RT im Beispiel

RT im Beispiel

Der Returntype von cc.charge(cup.price) "verschwindet" in buyCoffee. Das Ergebnis von buyCoffee(aCreditCard) ist nur cup, was äquivalent zu new Coffee() ist. Wenn buyCoffee eine reine Funktion wäre, so müsste für jedes Programm p, sich p(buyCoffee(aCreditCard)) und p(new Coffee()) gleich verhalten.

Beispiel ohne Seiteneffekt

```
1 class Cafe {
2   def buyCoffee(cc: CreditCard): (Coffee, Charge) = {
3     val cup = new Coffee()
4     (cup, Charge(cc, cup.price))
5   }
6 }
```

Zusammenfassung

- Idee: Verwende reine Funktionen d.h. Funktionen ohne Seiteneffekte.
- Mittels RT lässt sich überprüfen, ob eine reine Funktion vorliegt.
- Reine Funktionen ermöglichen "lokales schließen".
- Reine Funktionen sind kontextunabhängig.
- Reine Funktionen sind leichter testbar, wiederverwertbar und modular.

Vielen Dank
für eure Aufmerksamkeit.