



Von imperativ zu funktional – Ein Scala-Beispiel

📅 26. August 2016 (<http://sebinside.de/2016/08/26/von-imperativ-zu-funktional-ein-scala-beispiel/>) 👤 seb (<http://sebinside.de/author/seb/>) 📁 Studium (<http://sebinside.de/category/studium/>)

Die Superstars der höheren Programmiersprachen sind Java und C++. Doch beides sind imperative Sprachen und erlauben funktionale Programmierung nur sehr eingeschränkt. Dass sich diese trotzdem sowohl für Performance als auch Verständlichkeit lohnen kann, möchte ich hier am Beispiel von Scala demonstrieren!

In einem früheren Blog-Post (<http://sebinside.de/2016/07/02/programmierparadigmen/>) habe ich bereits über verschiedene Programmierparadigmen gesprochen und die **imperative** Programmierung der **funktionalen** gegenüber gestellt. Nochmal kurz zur Erinnerung: Imperativ bedeutet Befehl nach Befehl (tue dies, dann tue das) während funktionale Programmierung deklarativ arbeitet und eher an mathematische Formeln erinnert.

Warum funktionale Programmierung?

Funktionale Programmierung hat einen wesentlichen Vorteil gegenüber „herkömmlicher“ imperativer Programmierung: Funktionen haben keine **Seiteneffekte**. Das bedeutet, dass die Ausgabe immer nur von der Eingabe abhängt. Das sieht auf den ersten Blick sehr trivial aus, kann bei sauberer Umsetzung die Anzahl unnötiger Programmfehler verringern. Ein Beispiel:

```
1  int giveMeANumber() {  
2  
3      if (crazyVal < 10) {  
4          crazyVal++;  
5          return 5;  
6  
7      } else {  
8          throw new Exception();  
9  
10     }  
11 }
```

Was ist die Variable **crazyVal** und wo kommt sie her? Keine Ahnung. Der Programmierer wird sich irgendwas dabei gedacht haben (oder auch nicht). Auf jeden Fall lässt sich diese Funktion weder sicher aufrufen noch testen, denn ihre Ausgabe ist maßgeblich von einer **Unbekannten** abhängig, die der Funktion nicht beim Aufruf übergeben wird. Das ist ein Seiteneffekt. Und der **stört**.

Wo wir es grade von **Performance** hatten: Natürlich kann so eine Funktion auch nicht vom Compiler oder der Laufzeitumgebung optimiert werden, da beim Funktionsaufruf das Verhalten nicht vorhergesagt werden kann. Die funktionale Sprache Haskell führt z.B. sogar intern Buche über kürzlich aufgerufene Funktionen und deren Ergebnis zur Optimierung, weil sich ein Ergebnis bei gleicher Eingabe ohne Seiteneffekte **nie** ändern wird (Ohne Probleme lassen sich so Beispiele mit 99% Speedup konstruieren).

Ein Beispiel für High-Level-Scala-Code

Aber funktionale Programmierung ermöglicht es nicht nur, Fehler zu vermeiden oder die Performance zu verbessern. Meistens ist funktionaler Code viel kompakter und besser zu verstehen (Naja, jedenfalls solange man nicht übertreibt, das ist überall so :)). ^

Im Folgenden wird ein Wert auf die Existenz eines Kleinbuchstabens getestet. Zunächst die klassische **imperative** Form in Java:

```
1 // Java
2 boolean hasLowerCase = false;
3
4 for (int i = 0; i < value.length(); i++) {
5     if (Character.isLowerCase(value.charAt(i)) {
6         hasLowerCase = true;
7         break;
8     }
9 }
```

Und jetzt die **funktionale** Lösung in Scala. Ich denke, hier ist kein weiterer Kommentar notwendig 😊

```
1 // Scala
2 val hasLowerCase = value.exists(_.isLowerCase)
```

Na gut, für alle die Scala interessiert, doch ein kleiner Kommentar: Variablen werden in Scala mit **val** oder **var** definiert (val ist äquivalent zu **readonly** var). Ein Typ muss nicht zwingend angegeben werden, den findet in den meisten Fällen die Scala-eigene Typinferenz. Die Funktion **exists** nimmt ein **Prädikat** entgegen und überprüft dieses für jedes Zeichen eines Strings (genauer: Für jedes Element einer Liste). Das Prädikat ist hier **isLowerCase**; der Unterstrich steht für ein anonymes Zeichen (hier eine Kurzschreibweise für ein schwergewichtigeren Lambda-Ausdruck).

Von imperativ zu funktional

So, und jetzt nochmal von vorne. Im Folgenden möchte ich einmal den kompletten Weg von imperativer zu funktionaler Programmierung durchgehen. Verwendet wird hierfür mal wieder Scala, das Beispiel ist mehr oder weniger an der Scala-Bibel (<http://amzn.to/2blzf2Z>) orientiert.

Der folgende Code gibt zeilenweise Argumente aus, wie sie einem Programm über die Commandozeile übergeben werden können.

Starten wir mit der **voll imperativen** Variante, wie man sie auch in Java schreiben könnte. Eine while-Schleife, die solange wiederholt wird, bis alle Argumente verarbeitet wurden:

^

```
1 def printArgs(args: Array[String]): Unit = {
2   var i = 0
3   while (i < args.length) {
4     println(args(i))
5     i+1
6   }
7 }
```

Das geht natürlich schöner. Im folgenden Beispiel wird die While-Schleife durch eine **For-Schleife** (genauer: Eine For-Each-Schleife) ersetzt. Statt 4 Zeilen nur noch 2 Zeilen Code, aber immer noch imperativ.

```
1 def printArgs(args: Array[String]): Unit = {
2   for (arg <- args)
3     println(arg)
4 }
```

Nun kommt der Sprung zum **funktionalen** Paradigma. Wurde gerade eben noch über alle Elemente von Hand iteriert, übernimmt nun die Sprache selbst die Iteration. Hierfür wird die Funktion *foreach* aufgerufen, die eine **anonyme Lambda-Funktion** annimmt und für jedes Element der Liste ausführt. Eine Lambda-Funktion ... äh, eigenes Buch. Aber letztlich ist es eine Schreibweise, um Funktionen mit Eingabe und Ausgabe-Parametern ohne Namen zu definieren.

Bildlich gesprochen: Nehme ein *arg* vom Typ *String* und werfe es in die Funktion *println()*.

```
1 def printArgs(args: Array[String]): Unit = {
2   args.foreach((arg: String) => println(arg))
3 }
```

Einer der großen Vorteile von Scala ist die **Typinferenz**. So kann die Typ-Angabe auch komplett weggelassen werden, der Compiler findet schon den besten Typ (hier: *String*). Nochmal einfacher Code.

```
1 def printArgs(args: Array[String]): Unit = {
2   args.foreach(arg => println(arg))
3 }
```

Und wo wir grade beim Weglassen sind: Eigentlich ist hier sogar die Lambda-Funktion überflüssig. Da hier der Fall sehr einfach ist (die genaueren Anforderungen lasse ich hier mal bei Seite), kann der Lambda-Ausdruck auch auf den eigentlichen Funktionsaufruf reduziert werden. Man spricht hier von **Unterversorgung** bzw. von **partiell angewandten Funktionen** (die es übrigens genauso wie auch das Lambda seit Version 8 auch in Java gibt!) ^

```
1 def printArgs(args:Array[String]): Unit = args.foreach(println)
```

Jetzt sind wir am Ende angekommen. **Vergleichen** wir diese eine Zeile mal mit der imperativen Code-Sequenz vom Beginn des Beispiels. Beide machen exakt dasselbe. Aber was ist leichter zu schreiben? Und besser zu verstehen?

Pur funktional

Na gut, eigentlich sind wir noch nicht am Ende angekommen. Unsere Funktion **printArgs** ist vom Typ Unit, gibt also keinen Wert zurück. Stattdessen wird ein Wert auf der Konsole ausgegeben. Und was ist das? Genau, ein **Seiteneffekt**.

Aber ganz ohne diese geht es eben doch nicht ... naja, außer man baut eine Funktion, die nichts ausgibt, sondern die formatierten Zeilen als Wert zurück liefert.

```
1 def formatArgs(args: Array[String]) = args.mkString("\n")
```

Die Sache mit der Performance

Zum Schluss noch ein paar Worte zur Performance. Funktionaler Programmierung wird oft nachgesagt, dass sie niemals mit herkömmlicher, imperativer C-Programmierung mithalten könnte. Letztlich genau der gleiche, **unnötige Krieg** wie zwischen JIT-compilierten Sprachen (Java, C#, usw.) und C++.

Ein konkretes Gegenbeispiel habe ich bereits mit der **Laufzeit-Optimierung** von Haskell (siehe oben) gegeben. Ein weiteres kommt aus der Welt von Java: In Java 8 ist es möglich, Parallelberechnung in Form von **Streams** zu definieren. Statt von Hand Threads zu erstellen und diese loslaufen zu lassen, übergibt man der Laufzeit-Umgebung einfach funktional die Arbeit und lässt diese alles optimieren.

Weiß ein System, das in diesem Moment den **vollen Einblick** auf aktuelle Prozesse hat, eventuell besser, wie ein Vorgang zu optimieren ist, als der Mensch, der nur mit diesem System programmiert hat? Eventuell **ein Gedanke wert**...! ^

Fazit, oder auch TL;DR


Funktionale Programmierung ist geil. Aber pur funktional ist meistens dann doch nicht ganz so einfach einsetzbar. Ansonsten bietet funktionaler Code aber mehr Power, bei weniger Arbeit und weniger Fehleranfälligkeit.


Okay, ich gebe zu: Ich rede hier grade von einer **perfekten Welt**. Natürlich kann man im funktionalen Paradigma genauso viel Müll hinschreiben, wie sonst auch. Aber: In mehr als einem Paradigma denken ermöglicht kreativere und deswegen vielleicht auch bessere Lösungen für Probleme.

Und allein deswegen ist es auf jeden Fall **einen Versuch wert!**

Teilen mit:

 E-Mail (<http://sebinside.de/2016/08/26/von-imperativ-zu-funktional-ein-scala-beispiel/?share=email&nb=1>)

 Facebook 1 (<http://sebinside.de/2016/08/26/von-imperativ-zu-funktional-ein-scala-beispiel/?share=facebook&nb=1>)

 Twitter (<http://sebinside.de/2016/08/26/von-imperativ-zu-funktional-ein-scala-beispiel/?share=twitter&nb=1>)

Gefällt mir:

★ Gefällt mir



6 Bloggern gefällt das.

[c](http://sebinside.de/tag/c/) (<http://sebinside.de/tag/c/>) [funktional](http://sebinside.de/tag/funktional/) (<http://sebinside.de/tag/funktional/>)

[funktionale programmierung](http://sebinside.de/tag/funktionale-programmierung/) (<http://sebinside.de/tag/funktionale-programmierung/>)

[haskell](http://sebinside.de/tag/haskell/) (<http://sebinside.de/tag/haskell/>) [imperativ](http://sebinside.de/tag/imperativ/) (<http://sebinside.de/tag/imperativ/>)

[imperative programmierung](http://sebinside.de/tag/imperative-programmierung/) (<http://sebinside.de/tag/imperative-programmierung/>)

[java](http://sebinside.de/tag/java/) (<http://sebinside.de/tag/java/>) [Programmieren](http://sebinside.de/tag/programmieren/) (<http://sebinside.de/tag/programmieren/>)

[Programmierparadigmen](http://sebinside.de/tag/programmierparadigmen/) (<http://sebinside.de/tag/programmierparadigmen/>)

[Programmiersprachen](http://sebinside.de/tag/programmiersprachen/) (<http://sebinside.de/tag/programmiersprachen/>) [scala](http://sebinside.de/tag/scala/) (<http://sebinside.de/tag/scala/>)

[studium](http://sebinside.de/tag/studium/) (<http://sebinside.de/tag/studium/>)



7 Gedanken zu „Von imperativ zu funktional – Ein Scala-Beispiel“



FELIX BEUSTER (HTTP://FIXEL.ME)

26. August 2016 um 18:42 Uhr (<http://sebinside.de/2016/08/26/von-imperativ-zu-funktional-ein-scala-beispiel/#comment-53>)

Irgendwann wenn ich mal Zeit habe, schaue ich mir auch mal Scala an. Vielleicht findet sich ja dann auch abseits der Uni mal ein Anwendungsfall ^^

ANTWORTEN (<HTTP://SEBINSIDE.DE/2016/08/26/VON-IMPERATIV-ZU-FUNKTIONAL-EIN-SCALA-BEISPIEL/?REPLYTOCOM=53#RESPOND>)

ONESIMPLE DOT

28. August 2016 um 12:51 Uhr (<http://sebinside.de/2016/08/26/von-imperativ-zu-funktional-ein-scala-beispiel/#comment-54>)

Echt cool und interessant das ganze 😊 Weiter so! c:

ANTWORTEN (<HTTP://SEBINSIDE.DE/2016/08/26/VON-IMPERATIV-ZU-FUNKTIONAL-EIN-SCALA-BEISPIEL/?REPLYTOCOM=54#RESPOND>)

ROMAN GRÄF

29. August 2016 um 14:48 Uhr (<http://sebinside.de/2016/08/26/von-imperativ-zu-funktional-ein-scala-beispiel/#comment-55>)

Kann es sein das du in der printAcvs funktion die mit
`args.foreach(arg => println(arg))`
arbeitet die schließende Klammer vergessen oder ist das in Scala zulässig

ANTWORTEN (<HTTP://SEBINSIDE.DE/2016/08/26/VON-IMPERATIV-ZU-FUNKTIONAL-EIN-SCALA-BEISPIEL/?REPLYTOCOM=55#RESPOND>)

FRIESEN BETON

31. August 2016 um 23:17 Uhr (<http://sebinside.de/2016/08/26/von-imperativ-zu-funktional-ein-scala-beispiel/#comment-56>)

Schon spannend, wie man funktional abkürzen kann, wenn man wie ich bisher quasi nur imperativen Code kennt bzw. nur imperativ programmieren kann 😊 Da ist's gleich doppelt so ärgerlich, dass man in Studium und Kürze auch Beruf nur mit dem Monster Fortran arbeitet ... 😞

ANTWORTEN (<HTTP://SEBINSIDE.DE/2016/08/26/VON-IMPERATIV-ZU-FUNKTIONAL-EIN-SCALA-BEISPIEL/?REPLYTOCOM=56#RESPOND>)

**GERD SATTLER**

18. September 2016 um 12:00 Uhr (<http://sebinside.de/2016/08/26/von-imperativ-zu-funktional-ein-scala-beispiel/#comment-60>)

Hey Skate,
ich weiß dass das hier ein bisschen der falsche Ort ist um zu fragen, aber was hältst du eigentlich von Linux und OpenSource?

ANTWORTEN ([HTTP://SEBINSIDE.DE/2016/08/26/VON-IMPERATIV-ZU-FUNKTIONAL-EIN-SCALA-BEISPIEL/?REPLYTOCOM=60#RESPOND](http://sebinside.de/2016/08/26/von-imperativ-zu-funktional-ein-scala-beispiel/?replytocom=60#respond))

SEB (HTTP://SEBINSIDE.DE)

20. September 2016 um 23:42 Uhr (<http://sebinside.de/2016/08/26/von-imperativ-zu-funktional-ein-scala-beispiel/#comment-61>)

Viel, deswegen ist auch 90% meiner Arbeit Open-Source – sogar die Artefakte meiner Bachelor-Arbeit 😊

ANTWORTEN ([HTTP://SEBINSIDE.DE/2016/08/26/VON-IMPERATIV-ZU-FUNKTIONAL-EIN-SCALA-BEISPIEL/?REPLYTOCOM=61#RESPOND](http://sebinside.de/2016/08/26/von-imperativ-zu-funktional-ein-scala-beispiel/?replytocom=61#respond))

Pingback: 14 coole Scala-Features (Teil 2 von 3) – SebInside (<http://sebinside.de/2016/09/28/14-coole-scala-features-teil-2-von-3/>)

KOMMENTAR VERFASSEN

Gib hier deinen Kommentar ein ...

◀ Programmierparadigmen (<http://sebinside.de/2016/07/02/programmierparadigmen/>)

14 coole Scala-Features (Teil 1 von 3) ▶ (<http://sebinside.de/2016/09/21/14-coole-scala-features-teil-1-von-3/>)

SOCIAL MEDIA

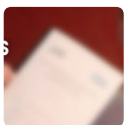


(htt (htt (htt (htt
 p://t p://f p://y (htt
 witt aceb outu p://g
 er.c ook. be.c ithu
 om/ com om/ b.co
 skat /seb skat m/s
 e70 insid e70 ebin
 2) e) 2) side)

BLOG VIA E-MAIL ABONNIEREN

Gib Deine E-Mail-Adresse an, um diesen Blog zu abonnieren und Benachrichtigungen über neue Beiträge via E-Mail zu erhalten.

BELIEBTE BEITRÄGE



Rückblick auf den Bachelor (<http://sebinside.de/2017/04/13/rueckblick-auf-den-bachelor/>)

13 Apr , 2017



Programmierparadigmen
 (<http://sebinside.de/2016/07/02/programmierparadigmen/>)

02 Jul , 2016



Webseiten kommen und gehen (<http://sebinside.de/2016/06/15/webseiten-kommen-und-gehen/>)

15 Jun , 2016

KATEGORIEN



📁 Allgemein (<http://sebinside.de/category/allgemein/>)

📁 Coding (<http://sebinside.de/category/coding/>)

📁 Minecraft (<http://sebinside.de/category/minecraft/>)

📁 Studium (<http://sebinside.de/category/studium/>)

Suche ...



NEUESTE KOMMENTARE

💬 Bowser bei Hallo. äh.. Welt! (<http://sebinside.de/2016/03/22/hallo-ae-h-welt/#comment-138>)

💬 daeaevaeaed bei Rückblick auf den Bachelor (<http://sebinside.de/2017/04/13/rueckblick-auf-den-bachelor/#comment-137>)

💬 coernerbrot bei Programmierparadigmen
(<http://sebinside.de/2016/07/02/programmierparadigmen/#comment-136>)

💬 seb (<http://sebinside.de>) bei Eine neue Plugin-Architektur für Code Overflow
(<http://sebinside.de/2018/01/16/eine-neue-plugin-architektur-fuer-code-overflow/#comment-135>)

💬 Sven (<https://usacookie.de>) bei Rückblick auf den Bachelor
(<http://sebinside.de/2017/04/13/rueckblick-auf-den-bachelor/#comment-134>)

ARCHIV

Januar 2018 (<http://sebinside.de/2018/01/>)

Juni 2017 (<http://sebinside.de/2017/06/>)

Mai 2017 (<http://sebinside.de/2017/05/>)

April 2017 (<http://sebinside.de/2017/04/>)

Oktober 2016 (<http://sebinside.de/2016/10/>)

September 2016 (<http://sebinside.de/2016/09/>)

August 2016 (<http://sebinside.de/2016/08/>)

Juli 2016 (<http://sebinside.de/2016/07/>)

Juni 2016 (<http://sebinside.de/2016/06/>)

April 2016 (<http://sebinside.de/2016/04/>)


März 2016 (<http://sebinside.de/2016/03/>)

SOCIAL MEDIA



(<http://twitter.com/sebinside>) (<http://facebook.com/sebinside>) (<http://youtube.com/sebinside>) (<http://github.com/sebinside>)

FEED

 (<http://sebinside.de/feed/>) RSS - Beiträge (<http://sebinside.de/feed/>)

 (<http://sebinside.de/comments/feed/>) RSS - Kommentare (<http://sebinside.de/comments/feed/>)



Ich heiße Sebastian, studiere Informatik am KIT in Karlsruhe und betreibe den YouTube Kanal skate702 (<http://youtube.com/skate702>). Auf diesem Blog geht es um Technik, Software und Minecraft!