Railway-Oriented-Programming in Clojure – Funktionales Error-Handling

Jan-Philipp Willem, Fakultät für Informatik, Hochschule Mannheim

Zusammenfassung—Das Verhalten eines Programms nach einem Fehlerfall zu definieren, wurde bereits in vielen Programmiersprachen zu lösen versucht. In der Praxis wird oft nur an den "Happy-Path" gedacht, oder gar gescheut den Code durch eine komplizierte Fehlerbehandlung aufzublähen. Scott Wlaschin hat versucht den Either-Type (Monade) mithilfe des Konzepts von einzelnen Abschnitten einer Eisenbahnstrecke auf eine andere Art zu erklären und damit eine funktionale Fehler-Beachtende Herangehensweise vorgeschlagen. Im Folgenden soll eine Umsetzung dieser Ideen in der Sprache Clojure und dessen Möglichkeiten beschrieben werden.

1 Einleitung

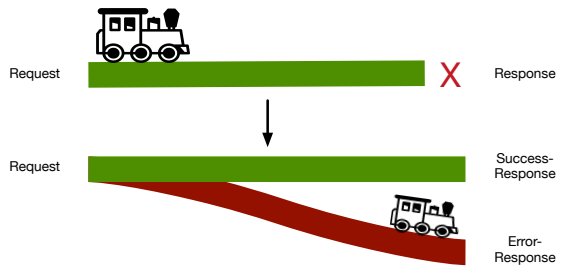

Abbildung 1: Bisherige Fehlerbehandlung und Nutzen von Zwei-Wege-Schienenteilen

Seit es losging.

1.1 „Happy-path"

1.2 „Error-by-design"

2 Railway-Oriented-Programming

Es sollen zunächst folgende grundlegende Schienen-Teile betrachtet werden:

- Result: Success/Failure
- switch
- switch-functions
- bind/bind-pipe
- switch-compose

Die Grundüberlegung besteht in der Erwartung, dass eine Funktion von nun an nicht nur einen einzigen Rückgabewert besitzt. Es soll zwischen einem gutartigen Ergebnis (**Success**) und dem Fehlerfall (**Failure**) unterschieden werden können. Dies ist durch ein Leichtes erzielbar, indem man einen **Result**-Typ definiert, welcher das eigentliche Ergebnis einer Funktion oder eine Fehlermeldung enthält.
Eine Funktion die unseren Result-Typ nutzt, soll im Weiteren als eine **switch-function** bezeichnet werden, da sie wie eine Weiche in der Bahnfahrt reagiert, welche den Datenfluss von dem Success-Path auf den Failure-Path leitet.
Um eine bestehende Funktion in einen ROP-Datenfluss zu integrieren, kann sie mit **switch** vereint werden, womit ihr eigentliches Ergebnis in ein Success konvertiert wird.

Um zwei switch-functions miteinander zu verbinden, wird die Funktion **bind** genutzt, welche die Success- und Failure-Paths miteinander verbindet. Da die Sprache Clojure ein Lisp ist, liegt es nahe, bind auch für die Übergabe einer Liste mit mehreren switch-functions zu definieren. So führt **bind-pipe** (»=) die übergebenen Forms nacheinander aus und leitet jeweils das Ergebnis der nächsten Funktion als Parameter zu. Es entsteht ein in sich geschlossenes Zwei-Wege-Schienensystem. Dabei fühlt sich die Vorgehensweise an, als ob man das thread-first macro (->) genutzt hätte, welches man in anderen Sprachen auch als pipe oder left-to-right-composition bezeichnet.
Mit **switch-compose** kann man switch-functions im Vergleich zu bind auf eine andere Art miteinander verbinden. Es kann so aus jeweils zwei switch-functions eine Komposition gebildet werden, welche sich benutzen lässt, als wäre es eine einzige Funktion. Die Verschachtelungstiefe ist dabei beliebig groß zu wählen. Es sind beide Wege für jeweilig andere Use-Cases sinnvoll.

2.1 Weitere Schienenteile

In Erweiterung zu den Grundlegenden Schienenteilen sind weitere folgende denkbar:

- try/catch
- log
- tee
- map-parallel
- ..

3 Fazit

Abschließend ist zu sagen, dass das Konzept der Fehlerbehandlung über einen Result-Typ und dessen dazugehörigen Hilfsfunktionen am Anfang im Vergleich zu dem bisherigen Weg sehr kompliziert wirkt. Weiterhin ist bei der Arbeit mit einem Team darauf zu achten, dass alle Entwickler neue Funktionen in den ROP-Datenfluss integrieren sollten. Bei einer konsistenten Anwendung, kann es im Gegenzug wirklich hilfreich sein, da man nun nicht mehr raten muss, ob eine Funktion bei einem Fehler nun nichts tut, Null oder bspw. einen negativen Wert zurückgibt. Man erhält so eine

stabile und leicht wartbare Basis für eigene Implementierungen. Es kann bspw. bei einer Formular-Validierung für ein Feld oder eine Überprüfung jeweilig ein Schritt in unserem ROP-Datenfluss dargestellt werden.
Als ein gutes Beispiel sind die Grund-Funktionen von Elm [7] zu betrachten, welche grundsätzlich alle bei kritischen Stellen eine Result- oder Maybe-Monade für den Rückgabewert nutzen.
Gegen Ende des Projektes wurde der Schreiber dieser Arbeit auf zwei weitere interessante Umsetzungen von ROP in Clojure aufmerksam. [3] bedient sich einer Clojure-Library [6] welche Konzepte der Kategorie-Theorie umsetzt. Und eine weitere [5], welche bereits als Paket in Leiningen verfügbar ist.

LITERATUR

[1] D. Higginbotham, *Clojure for the Brave and True*, https://braveclojure.com
[2] Scott Wlaschin, *Railway-Oriented-Programming – a functional approach to error handling*, https://fsharpforfunandprofit.com/rop/
[3] ah45, *railway oriented programming*, https://gist.github.com/ah45/7518292c620679c460557a7038751d6d
[4] Clojure Core Team, *Core.match – a Pattern Matching Function*, https://github.com/clojure/core.match/wiki/Basic-usage
[5] HughPowell, *railway-oriented-programming*, https://github.com/HughPowell/railway-oriented-programming
[6] Funcool, *Cats – Category Theory and Algebraic abstractions in Clojure*, https://github.com/funcool/cats
[7] Elm, *Gut gelößtes Sprachkonzept mit Maybe & Either in Elm*, https://guide.elm-lang.org/error_handling

Die Links wurden zuletzt am 07.07.2017 besucht.