Literaturverzeichnis

* [Ein funktionaler Slang] https://www.bibsonomy.org/bibtex/21680c-8585c614573b21adf96ee8514b7/huluvu424242
* [Das Beste aus beiden Welten] https://www.bibsonomy.org/bibtex/-2f0407601a74ef17760765623be79cc11/huluvu424242

Grundlegende Konzepte in Vavr

* Alle Datenstrukturen sind inmutable (unveränderlich) - psynonym kann der Leser auf den Begriff persistente Datenstrukturen treffen. In diesem speziellen Fall ist dann meist nicht die Datenbankpersistenz gemeint, sondern die Unveränderlichkeit.
* Partielle Funktionen: Umgangsprachlich sind diese Funktionen nicht für alle Werte definiert z.B. die Division welche als Division durch 0 nicht definiert ist und eine entsprechende Exception werfen sollte.
* Lifting: Durch lifting lassen sich partielle Funktionen in totale Funktionen wandeln. Das funktioniert z.B. durch Rückgabe eines Option welcher einen konkreten Wert oder aber None als Ausprägung annimmt. Alle Werte bei denen Exceptions auftreten würden, werden dann auf None gemappt und die Exceptions silent verschluckt. Daher steht letztlich im Ergebnis für jedes X ein Ergebniswert bereit auch wenn dieser nur None also eine Art undefiniert ist.
* Cyrring erlaubt die Verringerung der Parameteranzahl einer Funktion indem diese als Ergebnis eine Funktion mit der gekürzten Parameterliste zurück liefert. Diese wird dann erneut mit Werten aufgerufen. Achtung: Auf der einen Seite wird die Lesbarkeit der Funktion verbessert, weil nicht mehr so viele Parameter zu übergeben sind aber auf der anderen Seite versteht man den Code auf der Seite des Aufrufers nicht mehr intuitiv, da hier eine Funktion mit Werten aufgerufen und das Ergebnis dann gleich wieder mit Werten aufgerufen wird. Das ergibt nur bei echten Funktionen mit echter Bedeutung Sinn. Zumindest wenn man diese Konstruktionen im Objektorientierten Umfeld einsetzt.. * Memoization: Erster Aufruf der Funktion erzeugt den Wert, weitere Aufrufe geben den gecachten Wert zurück.
* Ciricuit Breaker: Schützt Methodenaufrufe z.B. durch "im Fehlerfall 5x wiederholen, dann nicht mehr".
* Bestimmte Funktionen in Vavr wie z.B. map werden nur im Erfolgsfall ausgeführt und andere nur im Fehlerfall. Der Fehlerfall wird vom Konzept her von der linken Seite bzw. der None Seite, bzw. dem failure bereitgestellt wohingegen der Erfolgsfall von der rechten Seite bzw. der Some Seite bzw. der success Funktion bereitgestellt werden.

Datentypen

Option	(serialisierbar) Kapselung optionaler Werte, Vermeidung von Nullchecks (Some oder None)
Try	Erfolg oder Misserfolg, Ausführung für success oder recover möglich
Either	Links oder Rechts
Functi-on0...Fun-ction8	Funktionen mit 0 bis 8 Parametern vgl. Supplier
Tupel	Tupel mit bis zu 8 Parametern möglich

Lifting

```
Function2<Integer, Integer, Integer> divide=(a,b)->a/b;
Functi on2 <In teger, Integer, Option <In teg er>>
safeDivide = Functi on2.li ft( div ide);
```

Memorize

```
Function0<Double>
hashCache=Function0.of(Math::random).memoized();
double random Value1 = hashCa che.ap ply();
double random Value2 = hashCa che.ap ply();
then(r and omV alu e1).is Equ alT o(r and omV -
alue2);
```

Option

```
String helloWorld = Option.of("Hello")
.map( value -> value + " wor ld")
.getOr Els e(( )-> " Rec ove r");
```

Tupel

```
Tuple2<String, Integer> java8 =
Tuple.of("Java","8");
Tuple2 <St ring, Intege r> vavr2 = Java8.map(
 s -> s + " sla ng",
   i -> i / 4
);
String first = vavr2._1;
int second = vavr2._2;
```

Lazy

```
Lazy<Double> lazy=Lazy.of(Math::random);
lazy.i sEv alu ated(); // false
lazy.get() // 0.123 (random)
lazy.i sEv alu ated(); // true
lazy.get() // 0.123 (cached)
```

Try

```
// Serviceinterface mit Checked exceptions
public interface HelloW orl dSe rvice{
  String sayHel loW orl d(S tring name) throws
Busine ssE xce ption;
}
public class HelloW orl dSe rvice{
  Try <St rin g> sayHel loW orld( String name){
    return Try.of (() ->b ack end Dao.sa yHe llo -
Wor ld( inp ut));
 }
}
String result = helloW orl dSe rvi ce.s ay Hel -
loW orl d("R obe rt")
.map(value -> value + " and all reader s")
.recov er( thr owable -> " Handle exception and
recove r")
.get();
```

By **Huluvu424242**
(FunThomas424242)

cheatography.com/funthomas424242/
github.com/Huluvu424242

Published 17th February, 2019.
Last updated 17th February, 2019.
Page 2 of 2.