Scalaの便利ライブラリ Catsの紹介

Kyoto.なんか #6 (2024-08-24)

@Windymelt (windymelt [at] capslock.dev)

時に2024年

🈹 Scala 3.5.0 リリース

- パイプラインビルドの高速化
- ベストエフォートコンパイル
  - 一部がコケてもIDEがめちゃくちゃにならないようにする
- など
- ウェ〜イ

Scala 3

- もろもろが大改善されたメジャーバージョン
  - コンパイルが爆速に
  - 人々を苦しめた`implicit`が再整理されて使いやすくなった
  - `enum`の導入
  - intersection / union typesの導入
  - etc.
- 始めるなら今
  - なんでも訊いてくれ!!!

誰

- Windymelt
- Scala日本語コミュニティ「Scalaわいわいランド」主宰

それはそうと

Q. Scalaといえば？

Scalaといえば

- 元祖・FP/OOPハイブリッドプログラミング言語
- 主戦場はJVM
  - 最近は普通にJSにもコンパイルできる(!!)
  - ネイティブコンパイルもある(!!)
- 昨今の関数型ブームの先駆け
  - 諸説ありそう

Scalaの嬉しさ

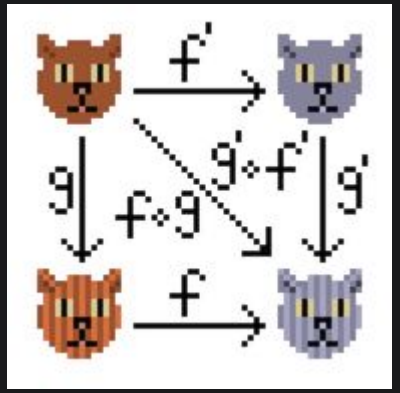
- 非常に強力なコレクションライブラリ
  - 「Scalaのアレが欲しい」との声をよく聞く(Windymelt調べ)
  - cf. Visual Scala Reference
- ほとんどの操作が実行時例外を投げない
  - 例外投げずに`Option`や`Either`を返す世界観
- FPとOOPの融合
  - 局所的にFP/OOPのいいとこ取りができる
- JVMの安定感

Scalaの苦労

- ScalaはFP寄りの言語だが、FPに全振りした言語ではない
- Scalaに最初から用意されているFP用の道具は割と素朴
  - FP用の語彙をなるだけ露出しないような努力があるっぽい
  - e.g. `Option`にも`List`にも`map`があるが、Functorと呼ばれることは(言語上は)ない
- ドメインが複雑になってくるとボイラープレートが増えてくる

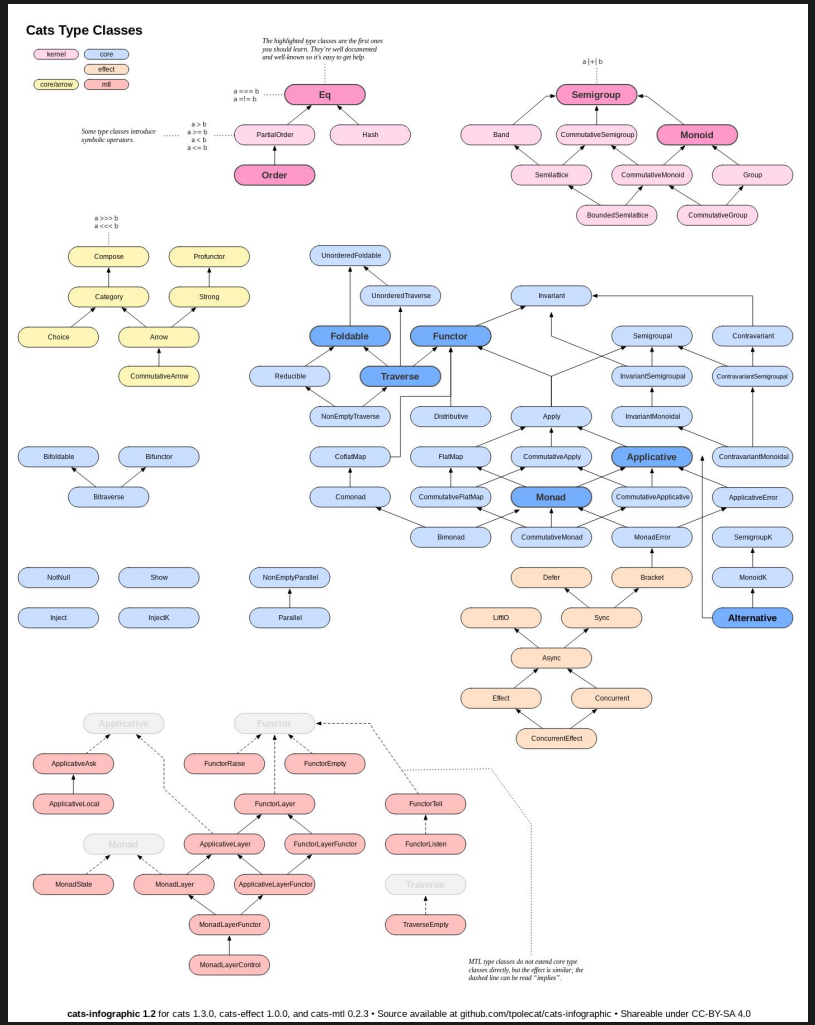
Cats

Cats



- Scala用のFP成分を強化するライブラリ
  - lodashみたいな便利ライブラリといった感じの位置
- ボイラープレート化しがちな複雑な箇所をうまいことシンプルに書けるようになる

どんなクラスがあるのかな〜



https://typelevel.org/cats/typeclasses.html

???🤔???

Catsようわからんかった・・・

- デカいライブラリとかを見るとこうなりがち
- もったいない
- 今回はCatsの便利そうな機能を絞ってご紹介します
  - CatsひいてはScalaを布教する

Monoid

- 結合処理の一般化
- `Option`同士も「足す」ことができる

```
import cats.syntax.all.{*, given}

val o1: Option[Int] = Some(42)
val o2: Option[Int] = Some(666)
val o3: Option[Int] = None

o1 |+| o2 // => Some(708)
o2 |+| o3 // => Some(666)
```

Monoid

- 辞書同士も足せる
- 一般化されてるので複雑な構造のマージにべんり

```
import cats.syntax.all.{*, given}

val likeA = Map(
  “food” -> List(“sushi”),
  “drink” -> List(“beer”),
)

val likeB = Map(
  “drink” -> List(“tea”, “juice”),
  “watch” -> List(“niconico”),
)

likeA.combine(likeB)
// aka likeA |+| likeB
// =>
//   Map(
       drink -> List(beer, tea, juice),
       watch -> List(niconico),
       food -> List(sushi),
     )
```

どういう理屈やねん

- 元々足せるやつは足せる
  - `List[_]`を足す: 結合すればいい
  - `Int`を足す: 数字を足せばいい
- 足せるやつで構成された構造同士なら足せる(!!)
  - `Map[K, V]`のとき、`V`が足せるなら、`Map[K, V]`も足せる
    - e.g. `Map[String, List[String]]`
  - `Option[A]`のとき、`A`が足せるなら、`Option[A]`も足せる

便利

Applicative

- 関数適用の一般化
- こんな関数があるとして
- どうやって`register`を呼ぶか・・・

```
import cats.syntax.all.{*, given}

def validateAge(n: Int): Option[Int] =
  if n >= 20 then Some(n) else None

def validateName(str: String): Option[String] =
  if str.isEmpty then None else Some(str)

def validatePassword(str: String):
Option[String] =
  if str.length < 5 then None else Some(str)

def register(
  name: String,
  age: Int,
  password: String
): Unit = ()
```

Applicative

- オエーー

```
import cats.syntax.all.{*, given}

val (age, name, password) =
  (31, “Windymelt”, “foobar2000”)

validateAge(age) match
case Some(a) =>
  validateName(name) match
  case Some(n) =>
    validatePassword(password) match
    case Some(pw) =>
      Some(register(n, a, pw))
    case None => None
  case None => None
case None => None
```

Applicative

- `for`で書ける
- そもそも互いの検証は依存してなくないか？

```
import cats.syntax.all.{*, given}

val (age, name, password) =
  (31, “Windymelt”, “foobar2000”)

for
  a <- validateAge(age)
  n <- validateName(name)
  pw <- validatePassword(password)
yield register(n, a, pw)
```

Applicative

- `mapN`
  - `for`の依存がない版
- 引数が全て`Option`で渡ってくるときに便利

```
import cats.syntax.all.{*, given}

val (age, name, password) =
  (31, “Windymelt”, “foobar2000”)

(
  validateName(name),
  validateAge(age),
  validatePassword(password),
).mapN(register)
```

便利

Traverse

- `map`してからひっくり返すような処理
- `List`の要素が全部`Some`のとき`Some[List[A]]`にしてくれる

```
import cats.syntax.all.{*, given}

val ages = List(20, 23, 30, 31)

// 二十歳以上ならビールがもらえる処理
def getBeer(age: Int): Option[String] =
  if age >= 20 then Some(“beer”) else None

ages.traverse(getBeer)
// => Some(List(“beer”, “beer”, “beer”, “beer”))

// 普通にmapすると
ages.map(getBeer)
// => List(Some(“beer”), Some(“beer”), …)
```

便利

結局なんが嬉しいの？

- アプリケーションが発展してくるとデータの変換に取られる比重が増えていく
  - JSON詰め替え業
  - ドメインモデル
  - 要するに構造をガチャガチャいじることが増える
- `combine`とか`mapN`とか`traverse`とかは具体的な構造に縛られない

構造

陰ながら重要な要素、構造

- データ構造が複雑になるのはしゃーない
- 複雑な構造に対して手でチマチマ分岐やループをしていませんか!?!?
  - ありがち
  - 事故の元
- Catsは似た構造に同じアプローチをとれるようにしてくれる
  - 特定の構造を名指しするのではない
  - 「こういうメソッド生えてて規則を満たすならなんでも」
  - 強力な汎用性(型クラスのありがたみ)

型クラス

- Type class
- いわゆるクラスは継承関係を用いて振る舞いを変えられる
  - 汎用性があまりない
- 型クラスは要件を満たせば名乗っていい
  - e.g. mapメソッドを実装していて、合成が保たれて、単位元を保つならFunctorを名乗ってよい
  - 幅広い範囲に便利な操作を生やせる (ad-hoc polymorphism)

まとめると

- 手で分岐・ループするのに疲れたら型クラスという強力な抽象化があるぞ
- Scala 3は型クラスに対する強力なサポートを備えているぞ
- Catsが型クラスを強力にブーストするぞ