Scala便利なConfig

Nextbeat

2024/05/24 (金)

富永 孝彦

今回はPureConfigというOSSが便利だよという話

今まで

今まではIxiaS側でTypesafe Configをラップ したConfigを使用していました。

```
val config = Configuration()

val host: Option[String] = config.get[Option[String]]("server.host")
val port: Option[Int]    = config.get[Option[Int]]("server.port")
```

これでも良いのですが、個人的にここら辺が微妙だった

- 1つずつconfigからの読み取りコードを書かないといけない
- 呼び出しタイミングによってはアプリケーション稼働中に例外が発生する
- 読み込んだ値を使ってモデルを構築する手間がある
- まとまりをキー情報でしか表現できない (`{まとまり}.{キー名}`)

こう思ってた

- そもそも1まとまりのものって個別に使うことってなくないか？まとまりなら最初からモデルになっておいて欲しい
- コンフィグレーションの値ってアプリケーション起動時に全部読み込めればそれで良くない？読み込めないものがあった時点で起動できない方がいいよね！

既存のConfigを拡張すれば可能だが...

ConfigLoaderの拡張がしんどい...
モデルごとに変換の処理が必ず必要となる

PureConfig

PureConfigは設定ファイルを読み込むためのScala ライブラリで、`HOCON`、`Java.properties`、または`JSON`などで書かれたTypesafe Config設定を、ボイラープレート不要の方法でScalaのネイティブクラスに読み込んでくれます。

Motivation

コンフィギュレーションのロードは、常に面倒でエラーの起きやすい手順で、一般的な方法はコンフィギュレーションの各フィールドをデシリアライズするコードを書くことでした。フィールドが多ければ多いほど、より多くのコードを書かなければならず（そしてテストし、保守しなければならない...）という問題がありました。

まさに

今更ながらPureConfigに入門してみる

使用感

こういうHttpサーバーを起動するために必要なモデルを構築する

```
case class Http(host: String, port: Int)
```

```
server {
  host = "127.0.0.1"
  port = 9000
}
```

今までだと

```
val config = Configuration()

val host: Option[String] = config.get[Option[String]]("server.host")
val port: Option[Int]    = config.get[Option[Int]]("server.port")


val http = Http(host, port)
```

PureConfigだと

```
val http = ConfigSource.default.at("server").load[Http]()
```

...以上！

素晴らしい

型の拡張は？

ホストやポートは `ip4s` の型を使いたいんだけど？

```
case class Http(host: Host, port: Port)
```

こんな感じでかける。

値の変換に失敗した理由を表す `FailureReason` を返してあげることで、エラーメッセージをいい感じに表示してくれる。

(デフォルトで色々な種類がある)

```
object Http {
  given ConfigReader[Host] = ConfigReader.fromString[Host] { str =>
    Host
    .fromString(str)
    .toRight(CannotConvert(str, "com.comcast.ip4s.Host", s"$str is not Host"))
  }
}
```

こんな設定値で

```
host = "example.com"
port = 8080
use-https = true
auth-methods = [
  { type = "private-key", pk-file = "/home/user/myauthkey" },
  { type = "login", username = "pureconfig", password = "12345678" }
]
```

モデルはこんな感じにしておく

```
case class Port(number: Int) extends AnyVal

sealed trait AuthMethod
case class Login(username: String, password: String) extends AuthMethod
case class Token(token: String) extends AuthMethod
case class PrivateKey(pkFile: java.io.File) extends AuthMethod

case class ServiceConf(
  host: String,
  port: Port,
  useHttps: Boolean,
  authMethods: List[AuthMethod]
)
```

こんなモデルにも拡張せずに対応できる！

いいね

```
ConfigSource.default.load[ServiceConf]
// res4: ConfigReader.Result[ServiceConf] = Right(
//   ServiceConf(
//     "example.com",
//     Port(8080),
//     true,
//     List(PrivateKey(/home/user/myauthkey), Login("pureconfig", "12345678"))
//   )
// )
```

色々なライブラリをサポートしている。
先ほどの `ip4s` もあり、先ほどの変換処理は本来は不要！

https://pureconfig.github.io/docs/library-integrations.html

自分はこんな感じで使ってるよ

```
case class Config(http: HttpConfig, database: DatabaseConfig, cookie: CookieConfig) derives ConfigReader
object Config:
  def read[F[_]: Sync]: F[Config] = ConfigSource.default.loadF[F, Config]()
```

Cats EffectなどのEffect Systemにも対応している！

※Scala3だと書き方が少し異なりモデルにConfigReaderを設定してあげる必要がある。

Configを構築できた時点で`database`の情報を表すモデルはあるので、持っている情報を使用して`toConnection`でコネクションを生成している

```
config <- Resource.eval(Config.read[IO])
master <- config.database.master.toConnection
slave  <- config.database.slave.toConnection
...
server <- EmberServerBuilder
            .default[IO]
            .withHost(config.http.host)
            .withPort(config.http.port)
            ....
```

※DatabaseConfigというモデルに`toConnection`というAPIを定義している。

他にも読み取りだけじゃなく書き込みもできたりします

いつ使うかは知らん

PureConfigで趣味プロジェクトがまたいい感じになった！
今までConfigは思考停止で同じような書き方しかしてなかったけど、他にも色々あるかも！

`circe-yaml` というものもあるらしい、 `circe` 使っている場合こっちの方が親和性は良いのかも？

良いScalaライフを〜