Tagless Final

Pepe García

2020-04-20

Tagless Final

In recent years, there has been a growing interest in modelling software in Scala using *Tagless Final*.

Tagless Final is a technique invented as a solution for the Expression Problem.

Tagless Final

It has changed a lot from the way it was ideated in the beginning to what we use now , but the idea is to provide the functionality packaged in typeclasses.

let's see how can we declare our TF Algebras:

Tagless Final

Carrier operations: Model

First, we'll need some model around the operations:

```
object model {
  type Order = Unit
  case class NewOrderResponse(trackingCode: String)
  type Tracking = Unit
  type DropoffPoint = Unit
}
```

Tagless Final

Carrier operations: Typeclass

```
import model._

trait CarrierIntegration[F[_]] {
  def newOrder(order: Order): F[NewOrderResponse]
  def getTracking(
    trackingCode: String
  ): F[Option[Tracking]]
  def getDropoffPoints(): F[List[DropoffPoint]]
}
```

Tagless Final

Carrier Operations: Implementation

```
import scala.concurrent.Future

object CarrierIntegration {

  implicit val impl: CarrierIntegration[Future] =
    new CarrierIntegration[Future] {
      def newOrder(order: Order): Future[NewOrderResponse] =
        Future.successful(NewOrderResponse("trackingId"))
      def getTracking(
        trackingCode: String
      ): Future[Option[Tracking]] =
        Future.successful(Some(()))
      def getDropoffPoints(): Future[List[DropoffPoint]] =
        Future.successful(Nil)
    }
```

Tagless Final

Carrier Operations: Usage

In order to use our TF Algebra, we can import it as any other typeclass!

```
import cats._
import cats.implicits._

def callCarrier[F[_]: Monad](
  implicit CI: CarrierIntegration[F]
): F[Unit] = for {
  response <- CI.newOrder(())
  _ <- CI.getTracking(response.trackingCode)
} yield ()
```