Shapeless를 이용한 제네릭 프로그래밍

컴파일러 괴롭히고 단순 작업 줄이기

박지수 (CTO, 두물머리)

Algebraic Data Type

Product Type

`N(Prod(A, B)) = N(A) × N(B)`

```
case class CaseClassIsProduct(a: String, b: Int)

type TupleIsAlsoProduct = (a: String, b: Int)
```

Sum Type

`N(Sum(A, B)) = N(A) + N(B)`

```
// type EnumLike = EnumLikeA | EnumLikeB
sealed trait EnumLike
case class EnumLikeA(a1: String, a2: Boolean) extends EnumLike
case object EnumLikeB extends EnumLike

type EitherIsAlsoSumType = Either[EnumLikeA, EnumLikeB]
```

실제 사례

날짜들만 일주일씩 미룰 수 없을까?

```
val els1 = Els(
  common = Common(
    // ...
    settlementDate = LocalDate.of(2016, 2, 17),
    finalValuationDate = LocalDate.of(2019, 2, 12),
    issueDate = LocalDate.of(2016, 2, 18),
    maturityDate = LocalDate.of(2019, 2, 15)`),
  placement = Placement(
    // ...
    offeringPeriod = DateTimePeriod(
      LocalDateTime.of(2016, 2, 16, 9, 0),
      LocalDateTime.of(2016, 2, 17, 13, 0))),
  // ...
  earlyRedemption = List(
    EarlyRedemption(LocalDate.of(2016, 8, 11), /* ... */),
    EarlyRedemption(LocalDate.of(2017, 2, 14), /* ... */),
// ...
```

안 될 거야 없지만

매크로

- 강력하지만
- 잘 만들기 어려워요
- (아직은) 컴파일러를 어느 정도 알아야 (Tree, Ident, Apply...)
- 현재진행형
  - scala.reflect
  - scala.meta
  - scala.macros
  - SCP-014

안 될 거야 없지만

매크로

- 강력하지만
- 잘 만들기 어려워요
- (아직은) 컴파일러를 어느 정도 알아야 (Tree, Ident, Apply...)
- 현재진행형
  - scala.reflect
  - scala.meta
  - scala.macros
  - SCP-014

런타임 리플렉션

안 될 거야 없지만

매크로

- 강력하지만
- 잘 만들기 어려워요
- (아직은) 컴파일러를 어느 정도 알아야 (Tree, Ident, Apply...)
- 현재진행형
  - scala.reflect
  - scala.meta
  - scala.macros
  - SCP-014

런타임 리플렉션

- 네???????

Shapeless

스칼라 제네릭 프로그래밍 라이브러리

- https://github.com/milessabin/shapeless
  - Scrap your boilerplate
  - Higher rank polymorphism

제네릭 프로그래밍?

- CoRecursive: 008 - Generic Programming with Miles Sabin

  > "The kind of polymophism your language does not support **yet**"

실제 코드

```
import shapeless.{everywhere, Poly1}

class shiftDate(days: Int) extends Poly1 {
  implicit val dateDate = at[Date](_.plusDays(days))
  implicit val dateDateTime = at[DateTime](_.plusDays(days))
}

object shift5 extends shiftDate(5)
val els2 = everywhere(shift5)(els1)
```

실제 코드

```
import shapeless.{everywhere, Poly1}

class shiftDate(days: Int) extends Poly1 {
  implicit val dateDate = at[Date](_.plusDays(days))
  implicit val dateDateTime = at[DateTime](_.plusDays(days))
}

object shift5 extends shiftDate(5)
val els2 = everywhere(shift5)(els1)
```

끝

어디에 쓰나요

- Akka-http (구 spray): The Streaming-first HTTP server/module of Akka
- Breeze: A numerical processing library for Scala
- Circe: A JSON library for Scala powered by Cats
- Doobie: A principled JDBC layer for Scala
- Finch: Scala combinator library for building Finagle HTTP services
- Frameless: Expressive types for Spark
- Kittens: Automatic type class derivation for Cats
- Monocle: Optics library for Scala
- Pureconfig: A boilerplate-free library for loading configuration files
- Refined: Simple refinement types for Scala
- Scodec: Scala combinator library for working with binary data
- 등등 https://github.com/milessabin/shapeless/wiki/Built-with-shapeless

준비물

- Implicit parameter resolution
- HList
  - (+ Coproduct)
- Path-dependent type
  - Aux 패턴
- Isomorphism
- 지금 다 몰라도 괜찮아요

Implicit Parameter Resolution

Lexical scope

지역변수 선언, import, 바깥 스코프, 상속, 패키지 오브젝트 등

Implicit scope

동반객체, 패키지 오브젝트, 타입 생성자, 파라미터, 수퍼타입 등

참조: http://eed3si9n.com/revisiting-implicits-without-import-tax

HList

Homogenious List

```
sealed trait List[A]
case object Nil extends List[Nothing]
case class Cons[A](head: A, tail: List[A]) extends List[A]

import scala._
val xs: List[Int] =
  1 :: 2 :: 3 :: Nil
```

HList

Homogenious List

```
sealed trait List[A]
case object Nil extends List[Nothing]
case class Cons[A](head: A, tail: List[A]) extends List[A]

import scala._
val xs: List[Int] =
  1 :: 2 :: 3 :: Nil
```

Heterogenious List

```
sealed trait HList
case object HNil extends HList
case class HCons[H, T <: HList](head: H, tail: T) extends HList

import shapeless._
val hxs: Int :: Boolean :: Char :: HNil =
  1 :: true :: 'a' :: HNil
```

Path-dependent Type

```
trait Fun[In] {
  type Out
  def apply(x: In): Out
}

def func[In](f: Func[In], x: Int): x.Out = f(x)
```

Composing Path-dependent Type

```
def composeWrong[A](
                     fa: Fun[A],
                     fb: Fun[fa.Out]
                   ): Fun[A] { type Out = fb.Out } = ???

// Error: illegal dependent method type: parameter may
// only be referenced in a subsequent parameter section
```

Composing Path-dependent Type

```
def composeWrong[A](
                     fa: Fun[A],
                     fb: Fun[fa.Out]
                   ): Fun[A] { type Out = fb.Out } = ???

// Error: illegal dependent method type: parameter may
// only be referenced in a subsequent parameter section
```

```
def compose[A](fa: Fun[A])
              (fb: Fun[fa.Out]): Fun[A] { type Out = fb.Out } =
  new Fun[A] {
    type Out = fb.Out
    def apply(x: A): fb.Out = fb(fa(x))
  }
```

Implicit Parameter의 경우

```
def composeImplicit1[A](implicit fa: Fun[A])
                       (fb: Fun[fa.Out]) = ???

// Error: ';' or newline expected

def composeImplicit2[A](implicit fa: Fun[A])
                       (implicit fb: Fun[fa.Out]) = ???

// Error: ';' or newline expected
```

Aux Pattern

```
// trait Fun { ... }
object Fun {
  type Aux[A, B] = Foo[A] { type Out = B }
}
```

Aux Pattern

```
// trait Fun { ... }
object Fun {
  type Aux[A, B] = Foo[A] { type Out = B }
}
```

```
def composeImplicit[A, B, C](
  implicit
  fa: Fun.Aux[A, B],
  fb: Fun.Aux[B, C]
): Fun.Aux[A, C] = new Fun[A] {
  type Out = C
  def apply(x: A): C = fb(fa(x))
}
```

참조: http://gigiigig.github.io/posts/2015/09/13/aux-pattern.html

Isomorphism

> 수학에서, 동형 사상(同型寫像, 문화어: 동형넘기기, 영어: isomorphism)은 서로 구조가 같은 두 대상 사이에, 모든 구조를 보존하는 사상이다. 두 대상 사이에 동형 사상이 존재하는 경우 서로 동형(同型, 영어: isomorphic)이라고 하며, 서로 동형인 두 대상은 구조가 같아 구조로서 구별할 수 없다. https://ko.wikipedia.org/wiki/동형_사상

```
case class ScalaNight(year: Int, location: Address, isFun: Boolean)
type ScalaNightTuple = (Int, Address, Boolean)
type ScalaNightHList = Int :: Address :: Boolean :: HNil
```

```
sealed trait Bool
case object True extends Bool
case object False extends Bool

type BoolEither = Either[True, False]
type BoolCoproduct = True :+: False :+: CNil
```

코드를 봅시다

좋기만 한가요

좋기만 한가요

그럴리가요

- 컴파일 시간 증가
- (상대적으로) 난해한 코드
- ADT <-> `HList/Coproduct` 변환으로 인한 런타임 오버헤드
- 컴파일러 버그를 밟을 때가 있음

좋기만 한가요

그럴리가요

- 컴파일 시간 증가
- (상대적으로) 난해한 코드
- ADT <-> `HList/Coproduct` 변환으로 인한 런타임 오버헤드
- 컴파일러 버그를 밟을 때가 있음

일부 대안

- Magnolia: fast and unintrusive typeclass derivation
- scalaz-deriving: deriving for scala data types
- 그 외 각종 매크로 라이브러리

좋기만 한가요

그럴리가요

- 컴파일 시간 증가
- (상대적으로) 난해한 코드
- ADT <-> `HList/Coproduct` 변환으로 인한 런타임 오버헤드
- 컴파일러 버그를 밟을 때가 있음

일부 대안

- Magnolia: fast and unintrusive typeclass derivation
- scalaz-deriving: deriving for scala data types
- 그 외 각종 매크로 라이브러리
- `Ctrl+C`, `Ctrl+V`

언제

안 쓰는 게 나은가요

- 코드 중복이 거슬리지만 잘 모아놔서 한꺼번에 고치기 어렵지 않을 때
- 런타임 성능이 많이 중요할 때

쓰면 좋은가요

- 아주 복잡한 ADT를 다뤄야 할 때
- 타입 선언을 해봤자 런타임에 자꾸 깨질 때 (직렬화 등)
- 지금 붙여넣는 비슷한 코드 더미를 도저히 유지보수할 자신이 없을 때

(주)두물머리

스칼라 개발자를 모십니다

(주)두물머리

스칼라 개발자를 모십니다

데이터/인프라 엔지니어도 모십니다

(주)두물머리

스칼라 개발자를 모십니다

데이터/인프라 엔지니어도 모십니다

웹 프론트엔드 전문가도요

(주)두물머리

스칼라 개발자를 모십니다

데이터/인프라 엔지니어도 모십니다

웹 프론트엔드 전문가도요

https://github.com/doomoolmori/colleagues

(주)두물머리

스칼라 개발자를 모십니다

데이터/인프라 엔지니어도 모십니다

웹 프론트엔드 전문가도요

https://github.com/doomoolmori/colleagues

recruit@doomoolmori.com

recruit@doomoolmori.com

감사합니다.

recruit@doomoolmori.com