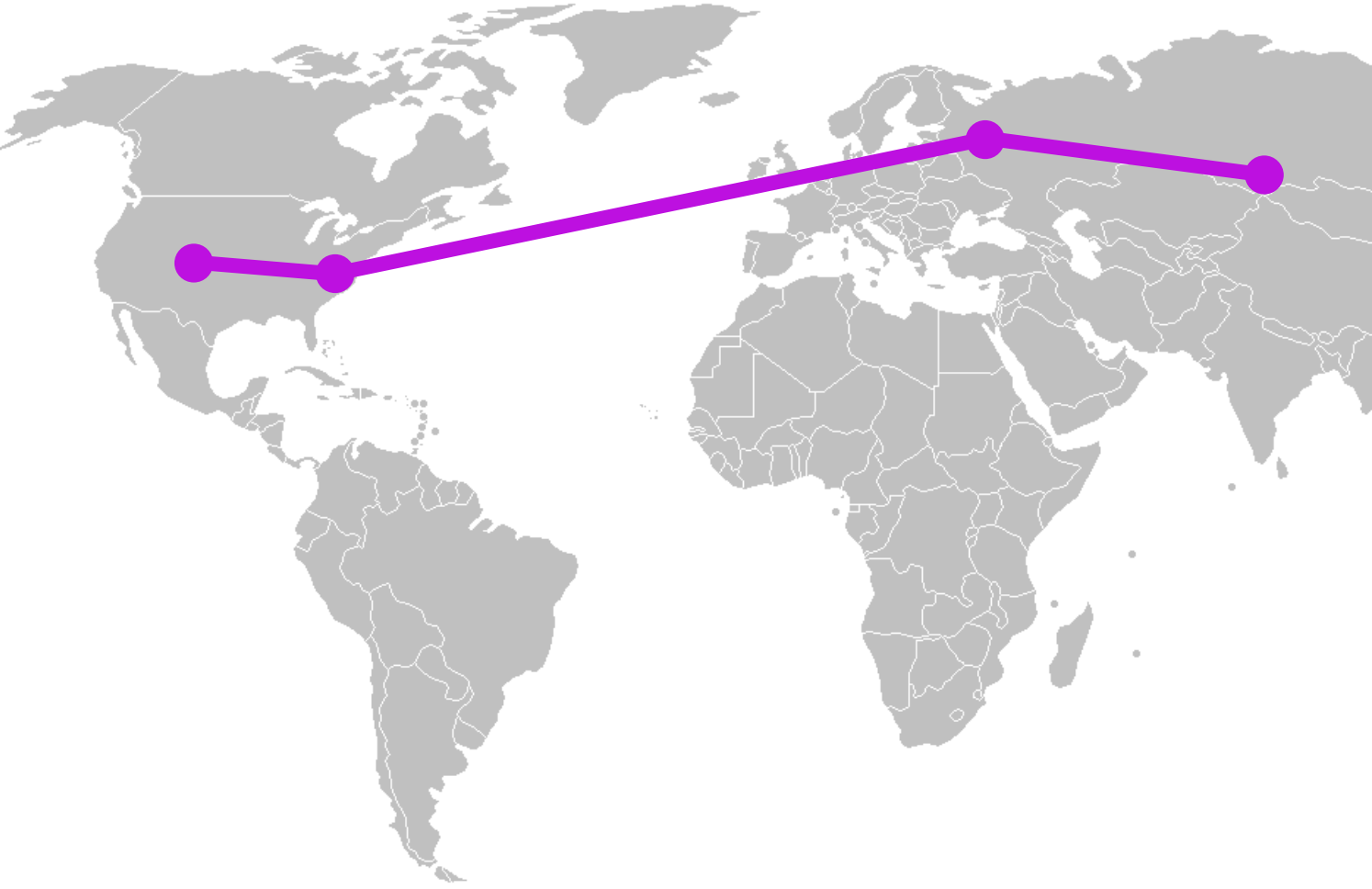
Typesafe Data Frames with Shapeless

Gleb Kanterov
@kanterov
gleb@kanterov.ru



```
class RDD[A] {
  def map[B](f: A => B): RDD[B]

  def filter(f: A => Boolean): RDD[A]

  def union(other: RDD[A]): RDD[A]

  def groupBy[B](f: A => B): RDD[(B, Seq[A])]

  def collect(): Array[A]
}
```

```
class DataFrame {
  def select(cols: Column*): DataFrame

  def filter(condition: Column): DataFrame

  def collect(): Array[Row]
}

def min(col: Column): Column
```

```
class Dataset[T: Encoder] {
  def select[U1: Encoder](
    c1: TypedColumn[T, U1]): Dataset[U1]

  def map[U : Encoder](
    func: T => U): Dataset[U]

  def filter[T](
    f: T => Boolean): Dataset[T]
}
```

Problems

- Runtime reflection to derive Encoder
- TypedColumn[T, U] is unsafe
- Not every type signature is possible without type-level programming

Shapeless

“When you are doing type-level programming you are suffering pain so that your users do not.”

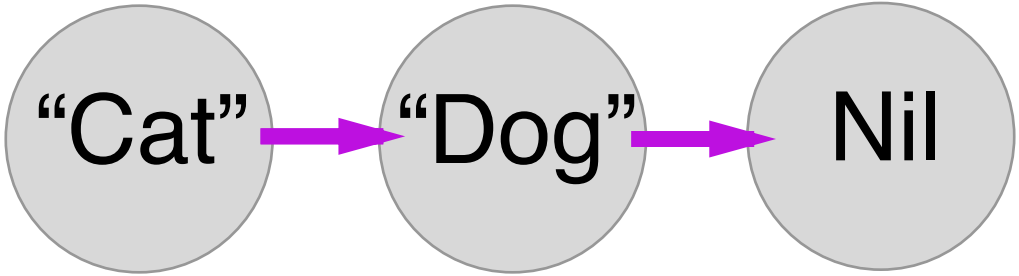
Daniel Spiewak

Frameless

Provide more typeful experience working with Apache Spark

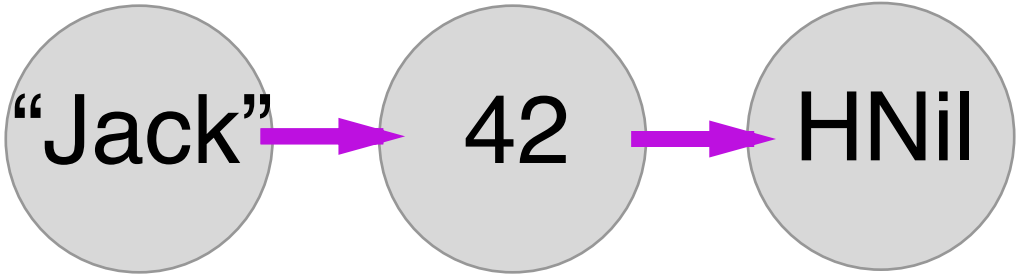
- Statically derived Encoder
- Columns are safely referenced
- Mirrors value-level computation to type-level for dataset methods

List[String]



HList

String :: Int :: HNil



```
case class Person(
  name: String,
  age: Int
)

new Generic[Person] {
  type Repr =
    String ::
    Int ::
    HNil

  def to(p: Person): Repr
  def from(h: Repr): Person
}
```

```
case class Person(
  name: String,
  age: Int
)

new LabelledGeneric[Person] {
  type Repr = Record.`
    'name -> String,
    'age  -> Int`.T

  def to(p: Person): Repr
  def from(h: Repr): Person
}
```

Safe column referencing

```
/** Evidence that type `T` has column `K`
    with type `V`.  */
@implicitNotFound(
  msg = "No column ${K} of type ${V} in ${T}")
trait Exists[T, K, V]

object Exists {
  implicit def derive[T, H <: HList, K, V](
    implicit
    lgen: LabelledGeneric.Aux[T, H],
    selector: Selector.Aux[H, K, V]
  ): Exists[T, K, V] = new Exists[T, K, V] {}
}
```

Safe column referencing

```
trait DataFrame {
  def col(name: String): Column
}

trait TypedDataset[A] {
  def col[U](c: Witness)(
    implicit e: Exists[A, c.T, U]
  ): TypedColumn[A, U]
}

// SIP-23: Literal-based singleton types
// 'name -> Witness { T = Symbol("name") }
ds.col('name)
```

```
trait JEncoder[A] {
  def enc(a: A): Json
}
```

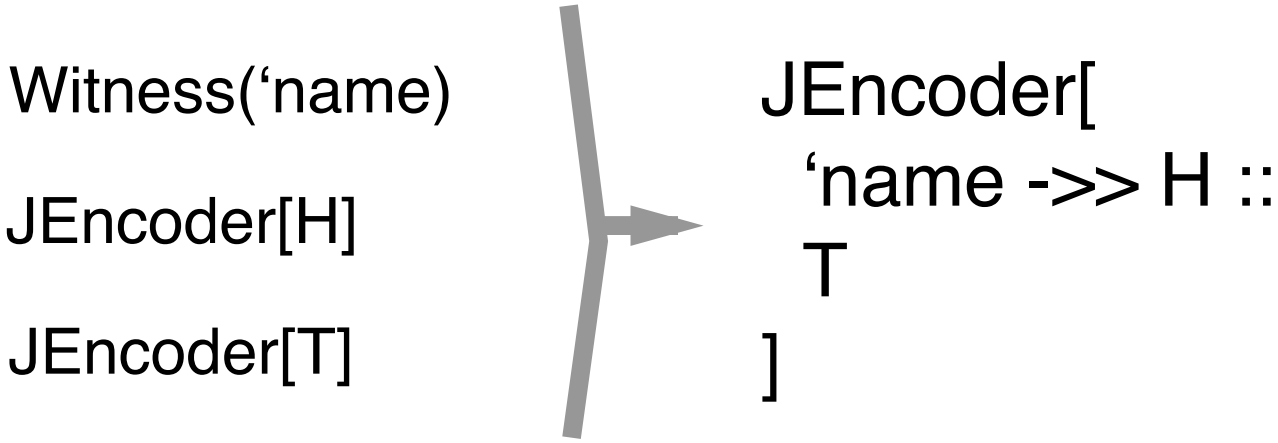
Base

JEncoder[String] → “Jack”

JEncoder[Int] → 42

JEncoder[HNil] → {}

Rule



```
val hnil: JEncoder[HNil]
val int: JEncoder[Int]
val string: JEncoder[String]

// 'age ->> Int :: HNil
val ageHNil = JEncoder { case h :: t =>
  { "age": int.enc(h) } |+| hnil.enc(t) }

// 'name ->> String :: 'age ->> Int :: HNil
val nameAgeHNil = JEncoder { case h :: t =>
  { "name": string.enc(h) } |+| e1.enc(t) }

val generic: LaballedGeneric[Person]

val person = JEncoder[Person] { person =>
  nameAgeHNil.enc(generic.to(person)) }
```

```
class ExpressionEncoder[A]
  extends Encoder[A] {

  def toRow(path: Expr): Expr

  def fromRow(path: Expr): Expr
}
```

```
// ExpressionEncoder[Person]

def toRow(path: Expression) = {
  val name =
    Invoke(path, "name", StringType)
  val age =
    Invoke(path, "age", IntegerType)

  CreateNamedStruct(
    Literal("name") ::
    stringEnc.toRow(name) ::
    Literal("age")  ::
    intEnc.toRow(age) ::
    Nil
  )
```

Statically derived encoders

- String, Int, Long, Double, etc.
- SQLDate, SQLTimestamp
- Vector[A], Option[A]
- products (case classes)
- (soon) sealed hierarchies (coproducts)
- using Injection

```
def select[U1](
  c1: TypedColumn[T, U1]
): TypedDataset[U1]

def select[U1, U2](
  c1: TypedColumn[T, U1],
  c2: TypedColumn[T, U2]
): TypedDataset[(U1, U2)]

def select[U1, U2, U3](
  c1: TypedColumn[T, U1],
  c2: TypedColumn[T, U2],
  c3: TypedColumn[T, U3]
): TypedDataset[(U1, U2, U3)]
```

```
def selectMany[U <: HList, Out <: HList](
  cols: U
)(
  implicit
  cm: Comapped.Aux[U, TypedColumn[A, ?], Out],
  tupler: Tupler[Out]
): TypedDataset[Out]

cols = col('name) :: col('age) :: HNil

U = TypedColumn[T, U1] ::
    TypedColumn[T, U2] ::
    HNil

Out = U1 :: U2 :: HNil

tupler.Out  = (U1, U2)
```

ProductArgs

```
selectMany(col('name) :: col('age) :: HNil)
```

```
selectMany(col('name), col('age))
```

```
Intersection.Aux[
  Record.`
    'name    -> String,
    'age     -> Int`.T,

  Record.`
    'name    -> String,
    'address -> String`.T,

  Record.`'name -> String`.T
]
```

```
Union.Aux[
  Record.`
    'name    -> String,
    'age     -> Int`.T,
  Record.`
    'name    -> String,
    'address -> String`.T,

  Record.`
    'name    -> String,
    'age     -> Int,
    'address -> String`.T
]
```

Frameless

github.com/adelbertc/frameless

Contributions are welcome!

Thank you! Questions?