Scala in the Wild

How Bubbleye uses Scala

Jiří Jakeš

November 13, 2018

Scala Taiwan Meetup

Functional Programming Recap

Referential transparency

When I substitute expression with its result, the program will not change.

Referential transparency

When I substitute expression with its result, the program will not change.

```
System.out.println("Hello world".length +
    "Hello world".length)
```

```
val len = "Hello world".length
System.out.println(len + len)
```

```
val rnd = new Random(0L)
System.out.println(rnd.nextInt() + rnd.nextInt())
```

```
val rnd = new Random(0L)
val n = rnd.nextInt()
System.out.println(n + n)
```

Cats Effect

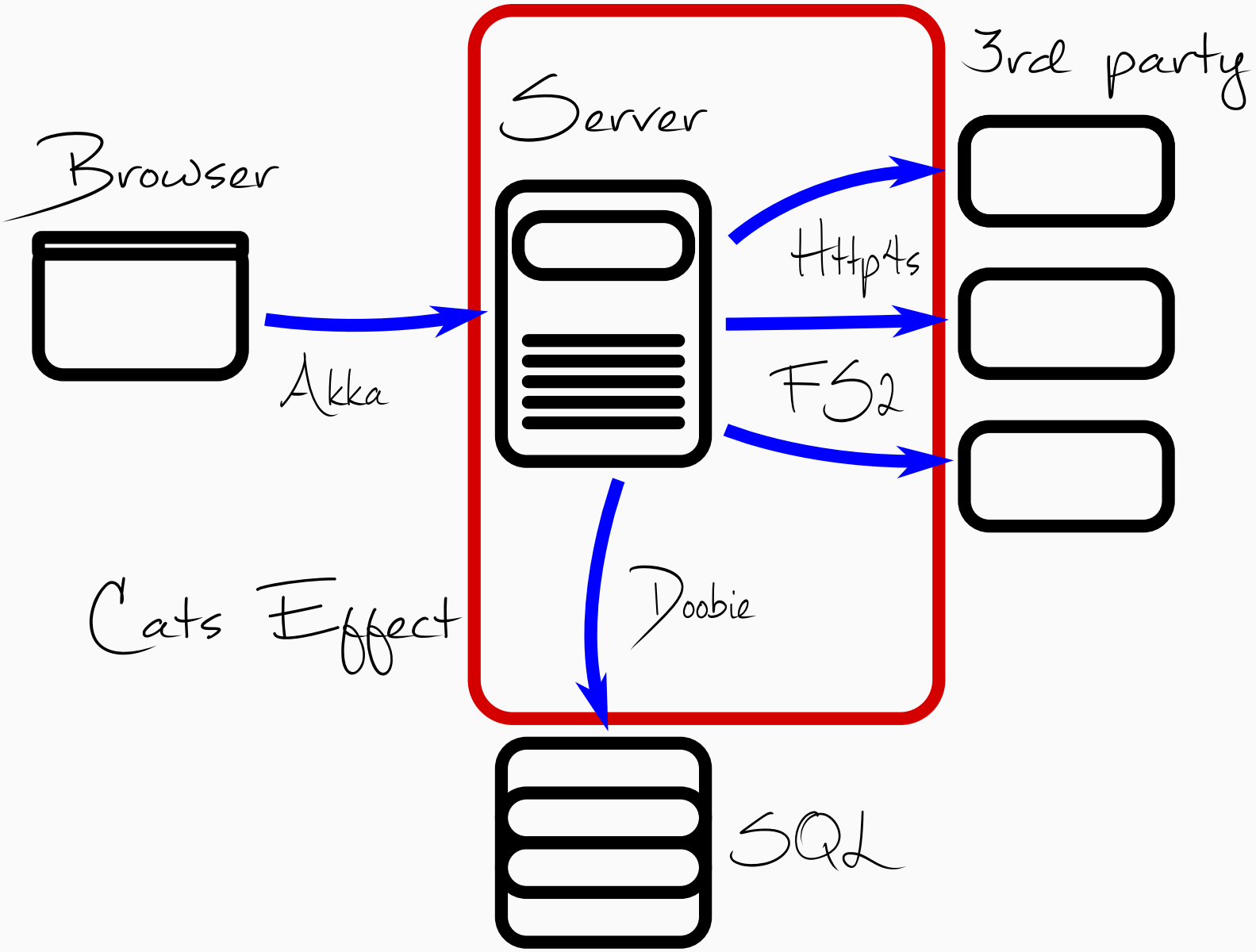
Cats Effect

- Purely functional set of abstractions for effectful types
- Synchronous and asynchronous computations
- Standard `IO` type

```scala
val rnd: IO[Int] = IO(Random.nextInt())

val program: IO[Unit] =
  for {
    rnd1 <- rnd
    rnd2 <- rnd
    _ <- IO(System.out.println(rnd1))
    _ <- IO(System.out.println(rnd2))
  } yield ()

def main(args: Array[String]): Unit =
  program.unsafeRunSync()
```

```scala
def rnd[F[_]: Sync]: F[Int] =
  Sync[F].delay(Random.nextInt())

def program[F[_]: Sync]: F[Unit] =
  for {
    rnd1 <- rnd
    rnd2 <- rnd
    _ <- Sync[F].delay(System.out.println(rnd1))
    _ <- Sync[F].delay(System.out.println(rnd2))
  } yield ()

def main(args: Array[String]): Unit = {
  program[IO].unsafeRunSync()
//  Await.ready(program[monix.Task].runAsync())
```

Doobie


Server
3rd party
Browser
Akka
Http4s
F52
Cats Effect
Doobie
SQL

Doobie

Pure functional JDBC layer for Scala and Cats/Cats Effect.

```scala
case class Company(name: String, zip: Int)
case class User(username: String, company: Company)

val userQ: Query0[User] =
  sql"""
     SELECT u.username, c.name, c.zip
     FROM users u
     LEFT JOIN companies c ON u.company = c.id
    """.query[User]
```

```scala
val userQ: Query0[User] = sql"...".query[User]

val c1: ConnectionIO[List[User]] =
  userQ.to[List]    // 0..*

val c2: ConnectionIO[NonEmptyList[User]] =
  userQ.nel         // 1..*

val c3: ConnectionIO[Option[User]] =
  userQ.option      // 0..1

val c4: fs2.Stream[ConnectionIO, User] =
  userQ.stream      // 0..*
```

```scala
val user: ConnectionIO[User] = ...
def address(u: User): ConnectionIO[Address] = ...
def save(a: Address, s: String): ConnectionIO[Unit] = ...

val stmt: ConnectionIO[Unit] =
  for {
    u <- user
    a <- address(u)
    _ <- save(a, "abc")
  } yield ()
```

```scala
val users: ConnectionIO[List[User]] = ...
def address(u: User): ConnectionIO[Address] = ...
def save(as: List[Address], s: String): ConnectionIO[Unit] =
                                                          ...
val stmt: ConnectionIO[Unit] =
  for {
    us <- users
    as <- us.traverse(address)
    _ <- save(as, "abc")
  } yield ()
```

```scala
val users: ConnectionIO[List[User]] = ...
def address(u: User): ConnectionIO[Address] = ...
def save(as: List[Address], s: String): ConnectionIO[Unit] =
                                                          ...
val stmt: ConnectionIO[Unit] =
  for {
    us <- users
    _ <- IO(System.out.println(us)).to[ConnectionIO]
    as <- us.traverse(address)
    _ <- save(as, "abc")
  } yield ()
```

```scala
val users: ConnectionIO[List[User]] = ...
def address(u: User): ConnectionIO[Address] = ...
def save(as: List[Address], s: String): ConnectionIO[Unit] =
                                                          ...
val stmt: ConnectionIO[Unit] =
  for {
    us <- users
    _ <- IO(System.out.println(us)).to[ConnectionIO]
    as <- us.traverse(address)
    _ <- IO.raiseError(new Exception( ... )).to[ConnectionIO]
    _ <- save(as, "abc")
  } yield ()
```

```scala
val tr: Transactor[IO] =
  Transactor.fromDriverManager[IO](driver, url)

val users: ConnectionIO[List[User]] = ...

val usersIO: IO[List[User]] = users.transact(tr)
```

```sql
CREATE TABLE companies (
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  zip INT NULL
)

CREATE TABLE users (
  username TEXT NOT NULL,
  company BIGINT NOT NULL REFERENCES companies(id)
)
```

```scala
case class Company(name: String, zip: Int)
case class User(username: String, company: Company)

class QueryTest extends FunSuite with IOChecker {
  override val transactor: Transactor[IO] = ...

  test("User") {
    check(
      sql"""
          SELECT u.username, c.name, c.zip
          FROM users u
          LEFT JOIN companies c ON u.company = c.id
        """.query[User]
    )
  }
}
```

```
- User *** FAILED ***
  Query0[QueryTest.User] defined at QueryTest.scala:14
    SELECT u.username, c.name, c.zip
    FROM users u
    LEFT JOIN companies c ON u.company = c.id
    ✓ SQL Compiles and TypeChecks
    ✓ C01 username VARCHAR (text) NOT NULL  →  String
    ✓ C02 name     VARCHAR (text) NOT NULL  →  String
    × C03 zip      INTEGER (int4) NULL      →  Int
      Reading a NULL value into Int will result in a runtime failure.
      Fix this by making the schema type NOT NULL or by changing the
      Scala type to Option[Int]
```

```
- User *** FAILED ***
  Query0[QueryTest.User] defined at QueryTest.scala:14
    SELECT u.username, c.name
    FROM users u
    LEFT JOIN companies c ON u.company = c.id
    ✓ SQL Compiles and TypeChecks
    ✓ C01 username VARCHAR (text) NOT NULL  →  String
    ✓ C02 name     VARCHAR (text) NOT NULL  →  String
    × C03                                   →  Int
      Too few columns are selected, which will result in a runtime
      failure. Add a column or remove mapped Int from the result type.
```

```
- User *** FAILED ***
  Query0[QueryTest.User] defined at QueryTest.scala:14
    SELECT u.username, c.zip, c.name
    FROM users u
    LEFT JOIN companies c ON u.company = c.id
    ✓ SQL Compiles and TypeChecks
    ✓ C01 username VARCHAR (text) NOT NULL  →  String
    × C02 zip      INTEGER (int4) NULL      →  String
      INTEGER (int4) is ostensibly coercible to String according to the
      JDBC specification but is not a recommended target type. Expected
      schema type was CHAR or VARCHAR or LONGVARCHAR or NCHAR or
      NVARCHAR or LONGNVARCHAR.
      Reading a NULL value into String will result in a runtime failure.
      Fix this by making the schema type NOT NULL or by changing the
      Scala type to Option[String]
    × C03 name     VARCHAR (text) NOT NULL  →  Int
      VARCHAR (text) is ostensibly coercible to Int according to the
      JDBC specification but is not a recommended target type. Expected
      schema type was INTEGER.
```

Http4s, fs2

Http4s

Typeful, functional, streaming HTTP for Scala.

Fs2

Compositional, streaming I/O library for Scala.

```scala
import fs2.Stream

val rng = new Random(0L)

val program: IO[Unit] =
  Stream
    .eval(IO(rng.nextInt()))    // Stream[IO, Int]
    .map(math.abs)
    .repeat
    .evalMap(i ⇒ IO(System.out.println(i)))
    .take(5)                    // Stream[IO, Unit]
    .compile                    // Stream.ToEffect[IO, Unit]
    .drain                      // IO[Unit]
```

```scala
val client: Client[IO] = ...

val data: Stream[IO, Byte] = ...

val request =
  Request[IO](
    method = Method.POST,
    uri = "https://www.example.com/api",
  ).withEntity(Multipart(Vector(
    Part.fileData("image", "landscape.jpg", data)
  )))

val response: IO[String] =
  client.fetch(request)(response ⇒ ... )
```

```scala
val s3Client: AmazonS3 = ...
val ec: ExecutionContext = ...

val stream: Stream[IO, Byte] =
  fs2.io.readInputStreamAsync(
    IO(
      s3Client
        .getObject("bucket", "key")
        .getObjectContent
    ),
    chunkSize = 1048576,
    ec
  )
```

Operation

```scala
val config: Conf = ...

def fun1(conf: Conf): A = ...
def fun2(a: A, conf: Conf): B = ...
def fun3(a: A, b: B, conf: Conf): C = ...

def fun(conf: Conf): C = {
  val a: A = fun1(conf)
  val b: B = fun2(a, conf)
  val c: C = fun3(a, b, conf)
  c
}

val c: C = fun(config)
```

```scala
case class Reader[A, B](run: A ⇒ B) {
  // map, flatMap, ...
}

val funR1: Reader[Conf, A] = Reader(fun1(_))
def funR2(a: A): Reader[Conf, B] = Reader(fun2(a, _))
def funR3(a: A, b: B): Reader[Conf, C] = Reader(fun3(a, b, _))

val funR: Reader[Conf, C] =
  for {
    a <- funR1
    b <- funR2(a)
    c <- funR3(a, b)
  } yield c

val c: C = funR.run(config)
```

```scala
// monad iff F monad
case class ReaderT[F[_], A, B](run: A ⇒ F[B]) {
  // map, flatMap, ...
}

val funR1: ReaderT[IO, Conf, A] = ...
def funR2(a: A): ReaderT[IO, Conf, B] = ...
def funR3(a: A, b: B): ReaderT[IO, Conf, C] = ...

val funR: ReaderT[IO, Conf, C] =
  for {
    a <- funR1
    b <- funR2(a)
    c <- funR3(a, b)
  } yield c

val c: IO[C] = funR.run(config)
```

```scala
def fun: (Vector[String], C) = {
  var log: Vector[String] = Vector.empty

  val a: A = funA()
  log = log :+ "Performed A"

  val b: B = funB(a)
  log = log :+ "Performed B"

  val c: C = funC(a, b)
  log = log :+ "Performed C"

  (log, c)
}
```

```scala
case class Writer[L, V](run: (L, V)) {
  // map, flatMap, ...
}

val funA: Writer[Vector[String], A] =
  Writer(Vector("Performed A"), ... )
def funB(a: A): Writer[Vector[String], B] =
  Writer(Vector("Performed B"), ... )
def funC(a: A, b: B): Writer[Vector[String], C] =
  Writer(Vector("Performed C"), ... )

def fun: Writer[Vector[String], C] =
  for {
    a <- funA
    b <- funB(a)
    c <- funC(a, b)
  } yield c

val (log, c) = fun.run
```

```scala
// monad iff F monad
case class WriterT[F[_], L, V](run: F[(L, V)]) {
  // map, flatMap, ...
}
```

Communication with 3rd party servers

- Short conversation with server
- Session data?
- Log?
- Error reporting?

```
class ReaderWriterStateT[F[_], E, L, S, A]( ... )

type RWST[F[_], E, L, S, A] =
                              ReaderWriterStateT[F, E, L, S, A]

/*

 F[_] = effect (RWST monad iff F monad)
 E    = environment (ReaderT)
 L    = log (WriterT)
 S    = state (StateT)
 A    = value

*/
```

```
EitherT[RWST[F, E, L, S, ?], Throwable, A]
```

```
trait Operations[F[_], E] {
  type Operation[A] =
   EitherT[RWST[F, E, TimedLog[String], Int, ?], Throwable, A]

  ...
}

class TimedLog[A](as: Vector[LogMessage[A]])

case class LogMessage[A](time: LocalDateTime,
                         nesting: Option[Int],
                         source: String,
                         line: Int,
                         message: A)
```

```scala
trait Operations[F[_], E] {
  type Operation[A] =
   EitherT[RWST[F, E, TimedLog[String], Int, ?], Throwable, A]

  def pure[A](a: A): Operation[A] = ...

  def liftF[A](fa: F[A}): Operation[A] = ...

  def env: Operation[E] = ...

  def fails[A](msg: String): Operation[A] = ...

  def auditBlock(log: String): Operation[Unit] = ...
  def audit(log: String): Operation[Unit] = ...
  def auditUnblock(log: String): Operation[Unit] = ...

   ...
}
```

```scala
def obtainFromServer: Operation[Thing] = ...
def processThing(t: Thing): Operation[Result] = ...
def sendToServer(r: Result): Operation[Response] = ...

val program: Operation[Result] =
  for {
    _ <- auditBlock(s"Doing something on server")
    thing <- obtainFromServer
    _ <- audit(s"Obtained: $thing")
    result <- processThing(thing)
    _ <- audit(s"Result: $result")
    response <- sendToServer(result)
    _ <- audit(s"Response from server: $response")
    _ <- auditUnblock("Done.")
  } yield result

val result: F[Result] = program.runAudited(SESSION)
```

```
2018-11-11T12:10:20.284 | 0.464 | 0.000 | Doing something on server
2018-11-11T12:10:20.748 | 0.006 | 0.464 |   Obtained: ...
2018-11-11T12:10:20.754 | 0.264 | 0.470 |   Result: ...
2018-11-11T12:10:21.017 | 0.001 | 0.733 |   Response from server: ...
2018-11-11T12:10:21.019 |       | 0.735 | Done.
```

```
2018-11-11T12:13:52.151 | 0.544 | 0.000 | Doing something on server
2018-11-11T12:13:52.695 | 0.005 | 0.544 |   Obtained: ...
2018-11-11T12:13:52.700 | 0.030 | 0.549 |   Result: ...
2018-11-11T12:13:52.730 |       | 0.579 | ERROR: Something wrong
```

Other

Not covered

- Tagless final
- Cats Effect concurrency primitives

Resources

- https://typelevel.org/cats/
- https://typelevel.org/cats-effect/
- https://http4s.org/
- http://fs2.io/
- https://tpolecat.github.io/doobie/

Thank you for attention