REAL WORLD FP

with

Cats-Effect and Fs2

JILEN

@ 水滴技术团队

公司介绍

- 微信公众号相关服务
- 实时处理较大量的微信消息推送
- 保存较大量的用户数据

水滴技术概况

- 后端几乎完全基于Scala
- 使用主流的Play Framework / Quill / Akka
- postgres / mysql / Kafka / ES
- 全异步
- FP - cats / cats-effect / shapeless

分享内容

- Cats-Effect介绍
- Cats-Effect工程实践
- Fs2实践

WHY?

提升知识水平...

Alexandru Nedelcu

alexandru

Scala and Haskell enthusiast, Typelevel contributor, author of Monix.io

Follow

Block or report user

România

alexn.org

Organizations

Followers you know

Overview | Repositories 54 | Stars 206 | Followers 442 | Following 3 | Gists 63

Pinned repositories

monix/monix

Asynchronous, Reactive Programming for Scala and Scala.js.

Scala ★ 1.3k 139

typelevel/cats-effect

The IO monad for Scala

Scala ★ 437 96

scala-best-practices

A collection of Scala best practices

★ 3.4k 474

funfix/funfix

Functional Programming Library for JavaScript, TypeScript and Flow ✨⚡

TypeScript ★ 427 23

monix/shade

Memcached client for Scala

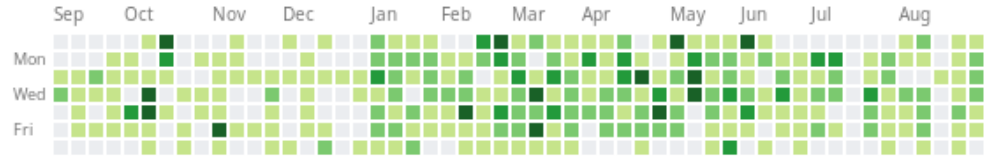
Scala ★ 94 20

stuff-classifier

simple text classifier(s) implemetation in ruby

Ruby ★ 448 100

1,825 contributions in the last year



Learn how we count contributions.

Less More

2018

2017

2016

2015

2014

@typelevel @monix @funfix More


2013
Activity overview
16%

Nedelcu Alexandru

Personal details

[Frame1]

Profile

I'm a passionate software developer with very good OOP programming skills. Able to work on own initiative, from concept to design, and from there to the web, or as part of a team. Proven leadership skills involving managing, developing and motivating teams to achieve their objectives. First class analytical, design and problem solving skills. Dedicated to maintaining quality standards.

Education
Computer Literacy

Programming Languages
C++, PHP, JavaScript, ActionScript

Databases
MySql, SqLite

DTP
Adobe Photoshop, Corel Draw

Authorware
Macromedia Flash, HTML, CSS, DHTML

ALEX NEDELCU

My first resume, circa 2004, please try not to laugh too hard. Apparently I had "good OOP programming skills", "proven leadership skills" and knew DHTML and Corel Draw, OMFG

JOHN A. DE GOES

- scalaz manitaner
- scalaz-zio 作者

大佬怎么说

John Ⓐ De Goes
@jdegoes

Following

I've been programming for more than 30 years. Procedural-OOP, dynamic-static, systems-apps. I've written millions of lines of code.

I've been there, done that, and in all my travels, I've found no better way to build correct, composable software than functional programming.

9:01 AM - 7 Aug 2018

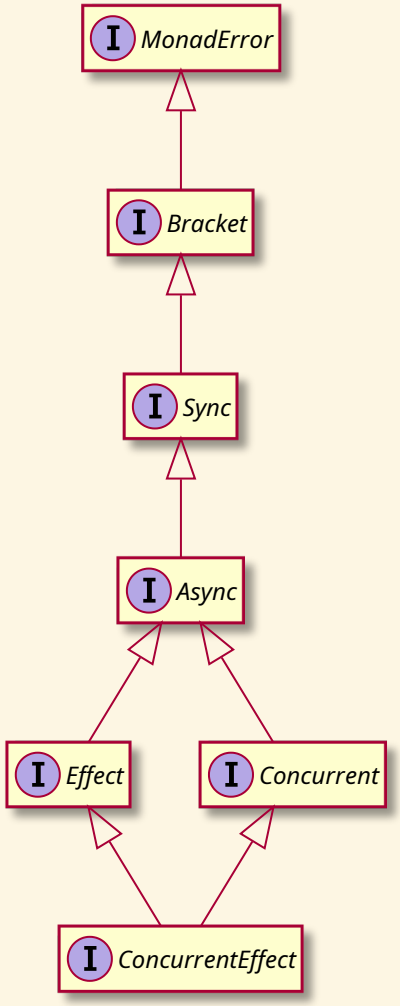
98 Retweets **425** Likes

CATS-EFFECT

CATS的IO MONAD实现

- 隔离推迟副作用
- 异步抽象

TYPECLASS


MonadError
Bracket
Sync
Async
Effect
Concurrent
ConcurrentEffect

WHY NOT FUTURE

- 不安全
- 性能差
- 难以推理

PARALLELISM WITH FUTURE

```
val f1 = Future {
  Thread.sleep(1000)
  1
}
val f2 = Future {
  Thread.sleep(1000)
  2
}
for {
  r1 <- f1
  r2 <- f2
} yield (r1, r2)
```

PARALLELISM WITH IO

```
def ioParallel = {
    //Through cats.Parallel
    val f1 = IO.sleep(1.seconds).as(1)
    val f2 = IO.sleep(1.seconds).as(1)
    (f1, f2).parTupled
}
```

CANCELABLE (FIBER)

```scala
def ioCancelable = {

    def setInterval[A](i: FiniteDuration, f: IO[A]): IO[Unit] = {
      def loop() = {
        IO.sleep(i) >> f.runAsync(_ => IO.unit) >> loop()
      }
      loop().start.flatMap(_.cancel)
    }

    for {
      h <- setInterval(i, IO(println("Hi"))).start
      _ <- IO.sleep(1.seconds)
      _ <- h
    } yield {}
}
```

CONCURRENCY

Purely functional, lock-free, non-blocking

- Ref - AtomicRefrence
- Deferred - Promise
- MVar - BlocingQueue(1)
- Semaphore

CATS-EFFECT实践

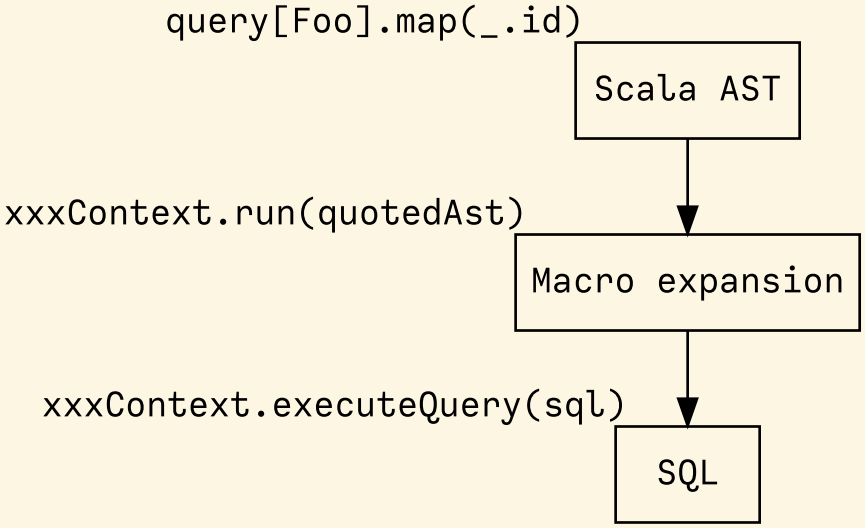
数据库操作

- doobie
- quill(mysql/postgres async)

PROS

- Slick紧耦合JDBC, Quill可以支持多个后端
- Quill编译时候生成SQL（可以在IDE/Console看到）
- 通过infix可以支持特定函数

QUILL INTERNAL



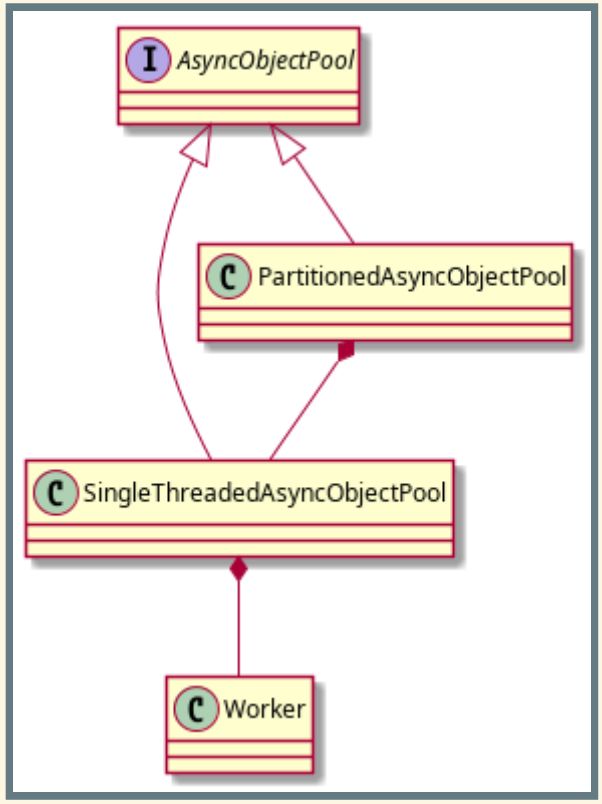
CONS

- 不支持复杂的join（无法正确进行Beta-Reduction，短时间内很难修复）
- 会生成带空格的ident（会导致老版本sbt增量编译无法工作）
- 可能会出现maximum string literal length exceeded

MYSQL-ASYNC的问题

- 设计上比较复杂
- 作者不再维护
- ConnectionPool实现非常error-prone
- 不会关闭PreparedStatement(mysql)

POOLING WITH MYSQL-ASYNC



FIX MYSQL-ASYNC POOLING

```
final case class State[F[_], A](
  queue: Vector[A],
  deq:   Vector[Deferred[F, A]]
)
class Queue[F[_], A](ref: Ref[F, State[F, A]])
  (implicit F: ConcurrentEffect[F], T: Timer[F]) {

  def enqueue(a: A): F[Unit]
  def timedDequeue(timeout: FiniteDuration): F[Option[A]]

}
```

ENQUEUE

```
// final case class State[F[_], A](queue: Vector[A], deq:   Vector[Deferred[
// ref: Ref[F, State[F, A]]
def enqueue(a: A): F[Unit] = {
    ref.modify { s =>
      if (s.deq.isEmpty) {
        (s.copy(queue = s.queue :+ a), None)
      } else {
        (s.copy(deq = s.deq.tail), Some(s.deq.head))
      }
    }.flatMap {
      case Some(h) =>
        F.runAsync(h.complete(a))(_ => IO.unit).to[F]
      case None =>
        F.unit
    }
}
```

HTTP CLIENT

```
implicit class AhcSyntax[F[_]](req: BoundedRequestBuilder)(implicit F: Concu
  def run() = F.cancelable[Response] { k =>
    val future = req.execute(new AsyncCompletionHandler[Unit] {
      override onThrowable(Throwable t) = {
        k(Left(t))
      }
      override onCompleted(res: Response) = {
        k(Right(res))
      }
    })
    F.delay(future.cancel())
  }
}
```

BLOCKING CODE

```
def shift[F[_], A](f: => A)(ec: ExecutionContext)(implicit S: ContextShift[F
  S.evalOn(ec)(F.delay(f))
}
```

代码组织

定义ALG(TAGLESS FINAL)

```scala
trait UserAlg[F[_]] {
  def add(a: User): F[Long]
  def get(id: Long): F[Option[User]]
}
```

ADT WITH FREE

```scala
sealed trait UserOpA[A]
case class Add(u: User) extends UserOpA[Long]
case class Get(id: Long) extends UserOpA[Option[User]]
type UserOp[A] = Free[UserOpA, A]

def add(u: User): UserOp[Long] = Free.liftF[UserOpA, Long](new Add(u))
def get(id: Long): UserOp[Option[User]] = Free.liftF[UserOpA, Option[User]](

def init(u: User) = {
  get(u.id).flatMap {
    case Some(u) => Free.pure(u)
    case None => add(u).map(id => u.copy(id = id))
  }
}
```

ALGBERA WITH F

```
class AlgWithFApp[F[_]](alg: UserAlg[F])(implicit F: Monad[F]) {
  def init(user: User) = alg.get(user.id).flatMap {
    case None => alg.add(user).map(id => user.copy(id = id))
    case Some(h) => F.pure(h)
  }
}
```

用类型处理错误

```scala
sealed trait UserLoginErr extends Exception
object UserLoginErr {
  case class NotExists(email: String) extends UserLoginErr
  case object PasswordIncorrect extends UserLoginErr
}
trait UserAlg[F[_]] {
  def login(email: String, pass: String): F[Either[UserLoginErr, Unit]]
}
```

FS2

Streaming your data with **Stream**

STREAM是什么

- 标准库的 **Stream** - 可能是无限长的队列
- fs2.Stream - 和标准库类似，但是这些元素可以通过 **eval** 副作用 **F** 获得

什么是简单

- 优雅(概念少)
- 复杂(概念多)

ELEGANT

```scala
type Pipe[F[_], I, O] = Stream[F, I] => Stream[F, O]
type Sink[F[_], I] = Pipe[F, I, Unit]
trait Topic {
  def publish: Sink[F, A]
  def subscribe: Stream[F, A]
}
trait Queue[F[_], A] {
  def enqueue: Sink[F, A]
  def dequeue: Stream[F, A]
}
```

POWERFUL

- Combinators（scan/fold/split…）
- Stateful transofrm with Pull

STREAMING QUERY

```scala
case class User(id: Long)

def readFrom(minId: Long): F[Seq[User]] = ???
def sendMsg(u: User): F[Unit] = ???

def stream() = {
  def loop(from: Long): Stream[F, User] =
    Stream.eval(readFrom(from)).flatMap {
      case us if !us.isEmpty => Stream.emits(us) ++ loop(us.map(_.id).max)
      case us => Stream.empty
    }
  }
  loop(0L)
}
stream().evalMap(sendMsg)
```

PRALLEL PROCESS

```
stream().mapAsync(100)(sendMsg)
```

STREAMING MYSQL BINLOG

```
def stream[F[_]](cli: BinaryLogClient)(implicit F: ConcurrentEffect[F]) = {

    def register(queue: Queue[F, Event]) = F.delay {
      cli.registerEventListener(new BinaryLogClient.EventListener() {
        override def onEvent(event: Event) {
          F.toIO(queue.enqueue1(event)).unsafeRunSync() //Blocking
        }
      })
      cli.connect(3000) //Spawns in new Thread
    }

    Stream.bracket {
      Queue.bounded[F, Event](1000).flatTap(register)
    } {
      _ => F.delay(cli.disconnect())
    }.flatMap(q => q.dequeueAvailable)

}
```

BACKPURESS WITH QUEUE

- bounded
- unbounded
- circularBuffer

MERGE

```
def merge[F[_]: ConcurrentEffect, A] {
    def fromQuery: Stream[F, A] = ???
    def fromRealtime: Stream[F, A] = ???
    def stream = fromQuery.merge(fromRealtime)
}
```

PARJOIN

```scala
def parJoin[F[_]: ConcurrentEffect, A] = {
    def conns: Stream[F, Con]
    def request(c: Con): Stream[F, Msg]
    def reply(m: Msg): F[Unit]
    def run() = conns.map(request).parJoin(1000).evalMap(reply)
}
```

TRANSFORM WITH PULL

MORE

- Signal
- Topic

THANKS