昵称：雪川大虫

园龄：3年

粉丝：122

www.cnblogs.com/tiger-xc/

blog.csdn.net/tiger_xc/

github.com/bayakala/

- 面临的一些问题
  - ✴ 巨大的源代码库
    - ✓ 代码重复使用率降低
    - ✓ 结构复杂、难理解、难预测、难掌控
    - ✓ 代码难以维护、难以管理
  - ✴ 函数式编程模式：组件（combinator）、组合（composition）
  - ✴ 海量的数据、新的数据处理要求
    - ✓ 无法有效处理数据：数据死锁、响应缓慢
  - ✴ 分布式大数据应用
  - ✴ 新硬件模式
    - ✓ 多核CPU、计算机集群
  - ✴ 多线程、多并发、分布式编程方式
  - ✴ 新计算模式
    - ✓ 云计算：计算资源集中化、应用系统平台化
  - ✴ 云平台应用

- scala
  - ✳ lambda expression - 匿名表达式
  - ✳ parametric types - 泛类型
  - ✳ pattern matching - 模式匹配
  - ✳ implicits - 隐式表达
  - ✳ immutable collections - 不可变集合
  - ✳ functions as values
  - ✳ partial functions
  - ✳ for-comprehension
  - ✳ …

✓ 函数式编程范式 - functional programming paradigm

✳ akka、spark

✓ scala >= java

  - ✳ 大数据技术方案
  - ✳ 丰富的开源资源
  - ✳ 庞大的动态社区

- 函数式编程 - functional programming ?

➡ M[P]

✓ 延迟运算 - delayed evaluation

```
val M[P] = M[p1]⊕M[p2]⊕M[p3]
… M[P].run
```

✳ no side effect

✳ immutability

✳ internal state eg. S1 => S2

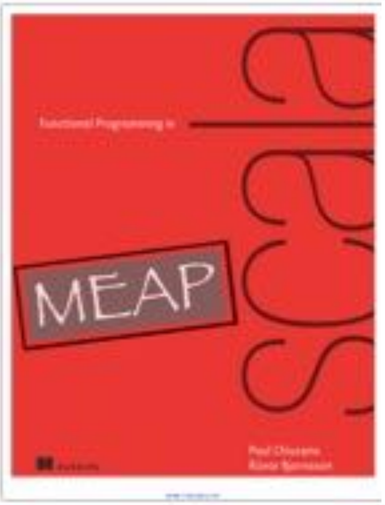
➡ List[P] = List[p1]++List[p2]++List[p3]

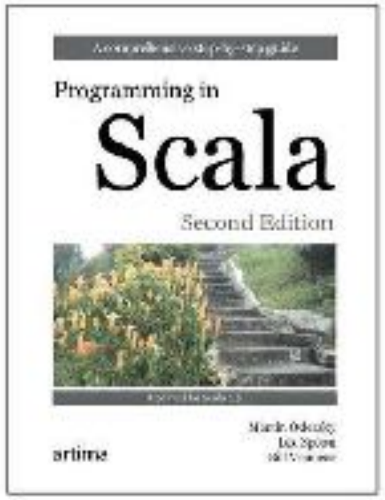
```
… run(List[P])
```

➡ Stream[P]

```
val Graph[P] = Source[p1] > Flow[p1,p2] > Flow[p2,p3]> Sink[P]
… Graph[P].run
```

- Scala函数式编程





Functional Programming in Scala

Programming in Scala

✳ 基础类型-basic abstractions

✓ Monoid

✓ Functor

✓ Applicative

✓ Monad

✳ 函数式编程组件库

✓ scalaz：haskel组件库的scala实现

✓ cats：another scalaz

X shapeless：type level programming

Functional Computation 函数式运算方式

组件	运算对象	函数款式	运算结果
Functor : map[A,B]	(F[A])	(f: A => B):	F[B]
Applicative : ap[A,B]	(F[A])	(f: F[A => B]):	F[B]
Monad : flatMap[A,B]	(F[A])	(f: A => F[B]):	F[B]
Traverse : traverse[G:Applicative,A,B]	(F[A])	(f: A => G[B]):	G[F[B]]

- 函数式运算示范

Functor: A => B

```
 def liftToStrong(name: String) = name.toUpperCase+”!"
 List("china","usa","japan").map(n => liftToStrong(n)).map(print)
          //> CHINA!USA!JAPAN!res0: List[Unit] = List((), (), ())
 case class Record(id: Int, content: String)
 case class Cache[A](data: A)
 implicit object cacheFunctor extends Functor[Cache] {
   def map[A,B](ca: Cache[A])(f: A => B): Cache[B] = Cache(f(ca.data))
 }
 val data = Cache[Record](Record(1,"I'm cached data”))
 def markRecord(r: Record) = Record(r.id + 1000, r.content + " updated!”)
 def saveToDB(r: Record) = println("saving record “+r.id)
 data.map(markRecord).map(saveToDB)   //> saving record 1001 res1:
Cache[Unit] = Cache(())
 val listOfcache = List(Cache(Record(1,"rec1")),Cache(Record(2,"rec2")))
 val listCacheFunctor = Functor[List] compose Functor[Cache]
 val mdata = listCacheFunctor.map(listCacheFunctor.map(listOfcache)
(markRecord))(saveToDB)
 //> saving record 1001 saving record 1002
 //> mdata:List[Cache[Unit]] List(Cache(()),Cache(()))
```

- 函数式运算示范

Applicative: F[A => B] >>> (F[A],F[B],F[C]…) => F[K]

```
trait Applicative[F[_]] extends Apply[F] { self =>
  def point[A](a: => A): F[A]
  override def map[A, B](fa: F[A])(f: A => B): F[B] =
    ap(fa)(point(f))
  override def apply2[A, B, C](fa: => F[A], fb: => F[B])(f: (A, B) => C): F[C] =
    ap2(fa, fb)(point(f))
```

```
trait Apply[F[_]] extends Functor[F] { self =>
  def ap[A,B]     (fa: => F[A])                             (f: => F[A => B]) :   F[B]
  def ap2[A,B,C]  (fa: => F[A], fb: => F[B])                (f: F[(A,B) => C]):   F[C] =
    ap(fb)(ap(fa)(map(f)(_.curried)))
  def ap3[A,B,C,D](fa: => F[A], fb: => F[B], fc: => F[C])   (f: F[(A,B,C) => D]): F[D] =
    ap(fc)(ap2(fa,fb)(map(f)(f => ((a:A,b:B) => (c:C) => f(a,b,c)))))
  …
  def apply2[A, B, C]    (fa: => F[A], fb: => F[B])               (f: (A, B) => C):    F[C] =
    ap(fb)(map(fa)(f.curried))
  def apply3[A, B, C, D] (fa: => F[A], fb: => F[B], fc: => F[C])  (f: (A, B, C) => D): F[D] =
    apply2(tuple2(fa, fb), fc)((ab, c) => f(ab._1, ab._2, c))
  …
   def lift2[A, B, C]  (f: (A, B) => C):      (F[A], F[B]) => F[C] =
    apply2(_, _)(f)
  def lift3[A, B, C, D](f: (A, B, C) => D):   (F[A], F[B], F[C]) => F[D] =
    apply3(_, _, _)(f)
  …
```

Applicative: F[A => B] >>> (F[A],F[B],F[C]…) => F[K]

```
val getsLen: String => Int = s => s.length          //> getsLen  : String => Int = <function1>
getsLen("abcd")                                      //> res4: Int = 4
val liftedLen = getsLen.point[List]                  //> liftedLen  : List[String => Int] =
List(<function1>)
Apply[List].ap(List("abcd"))(liftedLen)              //> res5: List[Int] = List(4)
```

```
trait Config[A] { def get: A }
object Config {
  def apply[A](a: A) = new Config[A] { def get = a }
  implicit val configFunctor = new Functor[Config] { def map[A,B](ca: Config[A])(f: A =>
B) = Config(f(ca.get)) }
  implicit  val confApplicative = new Applicative[Config] {
      def point[A](a: => A) = Config(a)
      def ap[A,B](ca: => Config[A])(cfab: => Config[A => B]) =cfab map (_(ca.get)) }
}
```

```
def map_[A,B](ca: Config[A])(f: A => B): Config[B] = Apply[Config].ap(ca)(f.point[Config])
def incr(i: Int): Int = i + 1                              //> incr: (i: Int)Int
Apply[Config].ap(Config(3))((incr _).point[Config]).get    //> res6: Int = 4
^(Config(1),Config(2)){_ + _}.get                          //> res7: Int = 3
^^(Config(1),Config(2),Config(3)){_ + _ + _}.get           //> res8: Int = 6
(Config(1) |@| Config(2) |@| Config(3)){_ + _ + _}.get     //> res9: Int = 6
(Config("hello") |@| Config(" ") |@| Config("world")) {_ + _ + _}.get   //> res10: String = hello
world
def greeting(hi: String, sp: String, w: String) = hi + sp + w
val configGreet = Apply[Config].lift3(greeting _)
     //> configGreet  : (Config[String],Config[String], Config[String]) => Config[String] =
<function3>
configGreet(Config("hello"),Config(" "), Config(“world!")).get          //> res11: String = hello
world!
```

Applicative: F[A => B] >>> (F[A],F[B],F[C]…) => F[K]

```
case class WebLogForm(usr: String, id: String, pwd: String)
def getUsr: Config[String] = Config("usr")        //> getUsr: => Config[String]
def getId: Config[String] = Config("id")          //> getId: => Config[String]
def getPwd: Config[String] = Config("pwd")        //> getPwd: => Config[String]
^^(getUsr,getId,getPwd)(WebLogForm(_,_,_))
      //> res12: Config[WebLogForm] = $$anonfun$main$1$Config$3$
$anon$4@359f7cdf
(getUsr |@| getId |@| getPwd)((a,b,c) => WebLogForm(a,b,c))
       //> res13:Config[WebLogForm] = $$anonfun$main$1$Config$3$
$anon$4@1fa268de
```

```
import java.sql.DriverManager
val connection = java.sql.DriverManager.getConnection("src","usr","pwd")

val sqlConnection =
Apply[Config].lift3(java.sql.DriverManager.getConnection)
//> sqlConnection: (Config[String], Config[String], Config[String]) =>
Config[java.sql.Connection] = <function3>

val conn =
sqlConnection(Config("Source"),Config("User"),Config("Password"))
```

- 函数式运算示范

Monad: A => F[B]

```
trait Monad[F[_]] extends Applicative[F] with Bind[F] { self =>
  override def map[A,B](fa: F[A])(f: A => B) = bind(fa)(a => point(f(a)))
…
```

```
trait Bind[F[_]] extends Apply[F] { self =>
  def bind[A, B](fa: F[A])(f: A => F[B]): F[B]
  override def ap[A, B](fa: => F[A])(f: => F[A => B]): F[B] = {
    val fa0 = Need(fa)
    bind(f)(x => map(fa0.value)(x))
  }
  def join[A](ffa: F[F[A]]) = bind(ffa)(a => a)
…
```

```
final class BindOps[F[_],A] private[syntax](val self: F[A])(implicit val F:
Bind[F]) extends Ops[F[A]] {
  def flatMap[B](f: A => F[B]) = F.bind(self)(f)
  def >>=[B](f: A => F[B]) = F.bind(self)(f)
  def join[B](implicit ev: A <~< F[B]): F[B] = F.bind(self)(ev(_))
…
```

Monad: A => F[B]

```
trait Config[A] { def get: A }
object Config {
  def apply[A](a: A) = new Config[A] { def get = a }
  implicit val confMonad = new Monad[Config] {
    def point[A](a: => A) = Config(a)
    def bind[A,B](ca: Config[A])(f: A => Config[B]) = f(ca.get)
  }
}
```

```
List(List(1,2),List(10,20)).flatMap(x => x.map(a => a))        //> res14: List[Int] =
List(1, 2, 10, 20)
(Config("hello") >>= { hi => Config(" ") >>= {
  sp => Config("World").map { world => hi + sp + world }}}).get  //> res15: String =
hello World
(Config(3) >>= (a => Config(2).map (b => a + b))) get           //> res16: Int = 5
(for {
  a <- Config(3)
  b <- Config(2)
  c <- Config(a+b)
} yield (c)).get                                                 //> res17: Int = 5
```

```
    fa.flatMap(a => fb.flatMap(b => fc.flatMap(c => fd.map(...))))
    for {
       a <- (fa: F[A])
       b <- (fb: F[A])
       c <- (fc: F[A])
    } yield { ... }
```

- 函数式运算示范

Traverse: (F[A])(f: A => G[B]) >>> G[F[B]]

```
traverse[G:Applicative,A,B](F[A])(f: A => G[B]): G[F[B]]
sequence[G:Applicative,A](F[G[A]]): G[F[A]]
```

```
 trait Book
 trait Author
 import concurrent._
 def books(author: Author): Future[Book] = ???
    //> books: (author: Author)scalaz.concurrent.Future[Book]

 def listFutureBooks(authors: List[Author]) = authors.map(books)
   //> listFutureBooks: (authors:
List[Author])List[scalaz.concurrent.Future[Book]]

 def futureListBooks(authors: List[Author]) = authors.traverse(books)
    //> futureListBooks: (authors:
List[Author])scalaz.concurrent.Future[List[Book]]

 def futureListBooks_s(authors: List[Author]) =
listFutureBooks(authors).sequence
  //> futureListBooks_s: (authors:
List[Author])scalaz.concurrent.Future[List[Book]]
```

- 函数库组件 - combinator library components

  - ✓ Monoid
  - ✓ Functor
  - ✓ Applicative
    - ✓ Free Applicativte：monadic programming - parallel
  - ✓ Monad
  - ✓ Monad Transformer
    - ✓ Reader Monad：dependency injection
    - ✓ Writer Monad：logger
    - ✓ State Monad：case class State[S,+A](run: S => (A, S))
    - ✓ Free Monad：monadic programming - sequencial
      - ✓ IO Monad
      - ✓ Slick DBIO …

✳ Scala Future

- programming patterns - 编程模式

✳ parametric types

✓ code reuse

✳ type class pattern

✓ ad hoc polymorphism

✓ dynamic type extension

✓ implicits

✳ magnet pattern

✓ free method overloading

✓ spray-api DSL response completion

✳ iterator pattern

✓ pull-model stream

✳ actor pattern

✓ message driven

✳ dependency injection

✓ cake pattern

✓ scala self-typing

✓ reader monad

✓ inversion of contral ioc

- scala libraries - 编程工具库

✳ Streams - 程序流程控制、数据交换

- ✓ scalaz-stream-FS2
- ✓ akka-stream
  - ✓ reactive-streams

✳ Database

- ✓ Slick
- ✓ ScalikeJDBC
- ✓ Cassandra-scala
- ✓ MongoDB-scala

✳ Distributed application infrastructure

- ✓ akka
  - message driven - 消息驱动
  - resilient - HA高可用
  - responsive - 高响应
  - elastic - 可拓展

- ✓ akka-actor
- ✓ aka-cluster
- ✓ aka-streams
- ✓ aka-http
- ✓ aka-persistence

• 开源项目 - Open Source Projects

✓ FunDA - Functional Data Acess - a Slick Extension
http://github.com/bayakala/FunDA

- auxiliary to slick
- scalaz-streams-FS2
- reactive-streams
- recordset traversing
- muti-thread、non-blocking
- parallel loading、updating

◉ DISK - Distributed I.T System Development Kit
??? http://github.com/bayakala/DISK

▸ 一套分布式集群运算架构
▸ 一套scala-api
▸ 无须了解分布式框架细节，使用API实现分布式编程

✓ 只需通过配置文件进行运算框架和环境调整
✓ 自动集群任务均衡分配
✓ Stream-Flow程序流程控制
✓ 提供JDBC、cassandra、MongoDB数据处理引擎

- akka-cluster - 分布式集群运算环境
- akka-stream - 程序运算控制与数据交换
- akka-http - 系统集成
- RabitMQ - 消息保证送达系统 AMQ

- 期望

  - ✓ 结交朋友
  - ✓ 了解新技术、新应用
  - ✓ 互相帮助
  - ✓ 共同发展