SCALA FOR GINI - TEIL II

MIT MEHR MONADEN

ANDREAS NEUMANN

MONADE - FÜR FUNKTIONALE PROGRAMMIERER

All told, a monad in X is just a monoid in the category of endofunctors of X, with product × replaced by composition of endofunctors and unit set by the identity endofunctor.

Saunders Mac Lane in Categories for the Working Mathematician

MOANDEN FÜR DEN PRAGMATISCHEN ENTWICKLER

- Abläufe kappseln (Wie Objekte Methoden und Werte kappseln)
- Abläufe aneinaderreihen
- Seiteneffekte sichtbar machen

„WIE EINE MONADE“

- Folgende Operation definiert:
  - map
  - flatMap aka bind in other languages
  - filter
- Monadenregeln können verletzt werden, der Programmierer ist dafür selbst verantwortlich (Beispiel später: Try)
- => for Notation möglich

MAP, FLATMAP, FILTER

MAP

```
scala> List(1, 2, 3, 4) map (_ + 2)
res7: List[Int] = List(3, 4, 5, 6)


scala> List(1, 2, 3, 4) map ( x => x * x)
res8: List[Int] = List(1, 4, 9, 16)


scala> List(1, 2, 3, 4) map (_.toDouble)
res10: List[Double] = List(1.0, 2.0, 3.0, 4.0)
```

FILTER

```
import NaivePrime._

scala> (1 to 100) filter isPrime
(1 to 100).filter( i => isPrime(i) )
res2: scala.collection.immutable.IndexedSeq[Int] = Vector(2, 3, 5, 7, 11, 13, 17, 19,
23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97)

scala> (1 to 10) filter ( _ % 3 == 0)
res3: scala.collection.immutable.IndexedSeq[Int] = Vector(3, 6, 9)

scala> List("Januar","Februar","März","April","Mai") filter
(month => month.reverse.startsWith("r"))
res4: List[String] = List(Januar, Februar)
```

FLATMAP

```
scala> val words = "Gallia est omnis divisa in partes tres"
split (" ")
words: Array[String] = Array(Gallia, est, omnis, divisa, in, partes, tres)

scala> words.sliding(2,1).toList
res2: List[Array[String]] = List(Array(Gallia, est), Array(est, omnis), Array(omnis,
divisa), Array(divisa, in), Array(in, partes), Array(partes, tres))

scala> words.sliding(2,1).toList map(bigram => bigram map
(_.size))
res3: List[Array[Int]] = List(Array(6, 3), Array(3, 5), Array(5, 6), Array(6, 2),
Array(2, 6), Array(6, 4))

scala> words.sliding(2,1).toList flatMap(bigram => bigram map
(_.size))
res4: List[Int] = List(6, 3, 3, 5, 5, 6, 6, 2, 2, 6, 6, 4)
```

FOR

- Vereinfachung von geschachtelten map-, flatMap- und Filter-Operation
- angelehnt an Pseudocode
- unterstützt auch Zuweisungen und Pattern matching

FOR - MAP, FLATMAP, FILTER

- map

```
scala> words map f
res5: Array[Int] = Array(6, 3, 5, 6, 2, 6, 4)

scala> for (word <- words) yield f(word)
res6: Array[Int] = Array(6, 3, 5, 6, 2, 6, 4)
```

- filter

```
scala> words filter (_.size > 3) map f
res8: Array[Int] = Array(6, 5, 6, 6, 4)

scala> for (word <- words ; if word.size > 3) yield f(word)
res7: Array[Int] = Array(6, 5, 6, 6, 4)
```

- flatMap

```
scala> words.sliding(2,1).toList flatMap( bigram => bigram
map(word  => f(word) ) )
res11: List[Int] = List(6, 3, 3, 5, 5, 6, 6, 2, 2, 6, 6, 4)

scala> for {
    bigram <- words.sliding(2,1).toList
    word <- bigram
    if word.size > 3
  } yield f(word)
res10: List[Int] = List(6, 5, 5, 6, 6, 6, 6, 4)
```

```
scala> val f : String
=> Int = s => s.size
f: String => Int =
<function1>
```

NÜTZLICHE MONADEN

OPTION-MONADE

- Seiteneffekt: Operationen kann oder kann keinen Wert liefern
- Option[A]
- Ausprägungen:
  - Some[A]
  - None

OPTION - BENUTZUNG

- Die Option-Monade verhält sich wie die Listen-Monade
- **Some**(x) kann man sich wie eine Liste mit einem Element vorstellen
- **None** entspricht der leeren Liste

OPTION UND LISTE - VERGLEICH

```
// Like a list
scala> Some( 1 ) map (i => i + 0.5 )
res12: Option[Double] = Some(1.5)

scala> List( 1 ) map (i => i + 0.5 )
res13: List[Double] = List(1.5)


// Like an empty List
scala> val x : Option[Int] = None
x: Option[Int] = None

scala> x map (i => i+ 0.5)
res16: Option[Double] = None

scala> val y : List[Int] = List()
y: List[Int] = List()

scala> y map (i => i+ 0.5)
res17: List[Double] = List()
```

SZENARIO FÜR OPTION

- Wir wollen Wikipedia anfragen
- Welche Sprachversion von Wikipedia angefragt werden soll ist konfigurierbar, wir unterstützen nicht alle Sprachen
- der Suchterm soll auf Deutsch eingegeben werden und bei Bedarf übersetzt werden, wir kennen nicht alle Übersetzungen
- unsere Verbindung ist mist, deshalb kommen die Anfragen nur manchmal an

```
// Wikipedia URLs
val wiki = Map(
  "de" -> "http://de.wikipedia.org/wiki",
  "en" -> "http://en.wikipedia.org/wiki"
)

// primitive Translator
val translation = Map(
  "Monade" -> Map("de" -> "Monade", "en" -> "monad"),
  "Erdferkel" -> Map("en" -> "Aardvark")
)

// get Wiki Text in chosen language by providing german Search Term
def getWikiTexts(lang: String, searchTerm: String) : Option[String] = for {
  url <- wiki get lang
  translations <- translation get searchTerm
  translatedTerm <- translations get lang
  answer <- flakyConnection(s"$url/$translatedTerm")
} yield answer

// the connection might work
def flakyConnection(url: String) : Option[String] =
  if( Random.nextBoolean() )
   Some( Source.fromURL(url,"UTF-8").getLines().mkString("").replaceAll("<[^>+]>","") )
  else
    None

def example = for {
  search <- List("Erdferkel","Monade","Erdbeeren")
  language <- List("en","de")
  answer <- getWikiTexts(language,search)
} yield answer.take(50)
```

BEISPIEL OPTION

```
scala> getWikiTexts("en","Erdferkel")
res12: Option[String] = None

scala> getWikiTexts("en","Erdferkel")
res13: Option[String] =
Some(<!DOCTYPE html><html lang="en" dir="ltr" class="client-nojs"> ...

scala> getWikiTexts("de","Monade")
res15: Option[String] =
Some(<!DOCTYPE html><html lang="de" dir="ltr" class=„client-nojs">
<head><meta charset="UTF-8" /><title>Monade – Wikipedia</title><meta
name="generator" content="MediaWiki 1.23wmf5" />...
```

EITHER

- wenn None nicht genug ist
- liefert entweder den einen oder den anderen Wert
- Either[A,B]
  - Right(x : B) : Either[A,B]
  - Left(x: A) : Either[A,B]

SZENARIO

- Wir bowlen mit Walter und dem Dude

- Wir übertreten, obwohl es ein Ligaspiel ist und wollen volle Punktzahl eintragen

- Entweder wir **tragen Null** ein oder wir **betreten eine Welt des Schmerzes**

```scala
case class WorldOfPain(s: String)
case class MarkZero(s: String)

  def markItAnEight : Boolean = Random.nextBoolean()


  def bowlAndTouchLine : Either[WorldOfPain, MarkZero] =
    if ( markItAnEight )
      Left(WorldOfPain("you are entering a world of pain."))
    else
      Right(MarkZero("All right, it's fucking zero. Are you happy, you crazy fuck?"))

  def example = {
    println(
      """The Dude: Walter, ya know, it's Smokey, so his toe slipped over the line a little, big deal. It's just a game,
man.
        |   Walter Sobchak: Dude, this is a league game, this determines who enters the next round robin. Am I wrong?
Am I wrong?
        |   Smokey: Yeah, but I wasn't over. Gimme the marker Dude, I'm marking it 8.
        |   Walter Sobchak: [pulls out a gun] Smokey, my friend ...""".stripMargin)

    bowlAndTouchLine match {
      case Left(x) => println(x)
      case Right(MarkZero(text)) => {
        println("Smokey: " + text)
        println( println("Walter Sobchak: ...It's a league game, Smokey.")
        )
      }
    }
  }
```

EITHER EXAMPLE

```
scala> example
The Dude: Walter, ya know, it's Smokey, so his toe slipped over the line a
little, big deal. It's just a game, man.
   Walter Sobchak: Dude, this is a league game, this determines who enters
the next round robin. Am I wrong? Am I wrong?
   Smokey: Yeah, but I wasn't over. Gimme the marker Dude, I'm marking it 8.
   Walter Sobchak: [pulls out a gun] Smokey, my friend ...
Smokey: All right, it's fucking zero. Are you happy, you crazy fuck?
Walter Sobchak: ...It's a league game, Smokey.
()

scala>

scala> example
The Dude: Walter, ya know, it's Smokey, so his toe slipped over the line a
little, big deal. It's just a game, man.
   Walter Sobchak: Dude, this is a league game, this determines who enters
the next round robin. Am I wrong? Am I wrong?
   Smokey: Yeah, but I wasn't over. Gimme the marker Dude, I'm marking it 8.
   Walter Sobchak: [pulls out a gun] Smokey, my friend ...
WorldOfPain(you are entering a world of pain.)
```

TRY-MONAD-LIKE

- Funktionales Error-Handling
- Ersetzt Either in vielen Implementierungen
- kein echte Monade, da Monadenregeln verletzt werden
- Try[A]
  - Success(x : A)
  - Failure(e: Exception)

TRY EXAMPLE SZENARIO

- Wir haben immer noch „crappy Hardware“ und unsere Verbindung ist echt mies
- Diesmal schmeißen wir aber mit Exceptions um uns wenn was nicht funktioniert
- Es kann passieren, dass keine Verbindung zustande kommt
- Es kann passieren, dass die Verbindung beim übertragen der Payload abbricht

```
case class Connection(toWhat: String) {

    def sendData(data: String): Try[String] = Try( send(data) )

    private def send(paylod: String) =
      if(Random.nextBoolean)
           s"Sent payload with ${paylod.size}"
      else
        throw new Exception("Couldn't send palyoad over Connection")

  }

  object Connection {

    def getConnection : Try[Connection] = Try ( connect )

    def connect =
      if(Random.nextBoolean)
        Connection("to the Internetz")
      else
        throw new Exception("Couldn't establish connection")

}

def sendHelloWorld  = for {
      connection <- Connection.getConnection
      r <- connection.sendData("Hallo Welt")
    } yield  r
```

```
scala> Connection.getConnection map (_.sendData("Hallo Welt"))
res17: scala.util.Try[scala.util.Try[String]] =
Success(Failure(java.lang.Exception: Couldn't send palyoad over
Connection))

scala> Connection.getConnection map (_.sendData("Hallo Welt"))
res18: scala.util.Try[scala.util.Try[String]] = Success(Success(Sent
payload with 10))

scala> Connection.getConnection map (_.sendData("Hallo Welt"))
res19: scala.util.Try[scala.util.Try[String]] =
Failure(java.lang.Exception: Couldn't establish connection)

scala> Connection.getConnection map (_.sendData("Hallo Welt"))
res20: scala.util.Try[scala.util.Try[String]] =
Success(Failure(java.lang.Exception: Couldn't send palyoad over
Connection))
```

FUTURE-MONADE

- Asynchrone Prozesse
- Future[A]
- nach on Complete
- Success(a: A)
- Failure(e: Exception)

FUTURE PLUMBING

- ExecutionContext :
- Aus der Doku: An ExecutionContext is an abstraction over an entity that can execute

```
import scala.concurrent._
import ExecutionContext.Implicits.global
```

- Duration: Implicit Conversions für Int

```
import scala.concurrent.duration._

// now valid code
5 seconds
1 hour
// langform
Duration(100, "millis")
```

EIN FUTURE ERSTELLEN

- mit future

```
scala> val h = future { "Hello World!" }
h: scala.concurrent.Future[String] = scala.concurrent.impl.Promise$DefaultPromise@2caa89b0

scala> h.isCompleted
res8: Boolean = true

scala> h.value
res9: Option[scala.util.Try[String]] = Some(Success(Hello World!))
```

FUTURES UND PROMISES

- Promise ist ein EINMAL beschreibbarer Container für zukünftige Ergebnisse
- Da die Berechnungen scheiten können wird das Ergebnis in ein Try gepackt
- Vollendete Promises werden zu Futures
- Promise[A]
- p.succes.future
- p.failure.future

PROMISE ERFÜLLEN

```
scala> val p = Promise[String]
p: scala.concurrent.Promise[String] =
scala.concurrent.impl.Promise$DefaultPromise@37cf00f1

scala> p.success("Welcome to the World of Tomorrow!")
res4: p.type = scala.concurrent.impl.Promise
$DefaultPromise@37cf00f1

scala> p.success("!!!")
java.lang.IllegalStateException: Promise already completed.
    at scala.concurrent.Promise$class.complete(Promise.scala:55)
```

PROMISE SCHEITERN LASSEN

```
scala> val p2 = Promise[String]
p2: scala.concurrent.Promise[String] = scala.concurrent.impl.Promise
$DefaultPromise@612c4f6d



scala> p2.failure( new Exception("Cyrogenic Error") )
res7: p2.type = scala.concurrent.impl.Promise$DefaultPromise@612c4f6d
```

PROMISE ERFÜLLEN UM FUTURE ZU VOLLENDEN

```
scala> p.future
res11: scala.concurrent.Future[String] = scala.concurrent.impl.Promise
$DefaultPromise@37cf00f1


scala> p2.future
res12: scala.concurrent.Future[String] = scala.concurrent.impl.Promise
$DefaultPromise@612c4f6d


scala> p.future.value
res17: Option[scala.util.Try[String]] = Some(Success(Welcome to the World of Tomorrow!))


scala> p2.future.value
res15: Option[scala.util.Try[String]] = Some(Failure(java.lang.Exception: Cyrogenic Error))
```

WERT AUS EINEM FUTURE HOLEN

```
scala> val f = futureThatTakes(10)("Nach zehn Sekunden")
f: scala.concurrent.Future[String] = scala.concurrent.impl.Promise
$DefaultPromise@4a64534b

// 5 Sekunden

scala> f.isCompleted
res0: Boolean = false
scala> f.value
res1: Option[scala.util.Try[String]] = None

// 11 Sekunden

scala> f.isCompleted
res3: Boolean = true
scala> f.value
res4: Option[scala.util.Try[String]] = Some(Success(Nach zehn Sekunden))
```

FUTURE MIT CALLBACK

- f : Future
- f onComplete : Unit

```
scala> example
res0: List[Unit] = List((), (), ())

scala> Ich brauche fünf Sekunden.
Ich brauche zehn Sekunden.
Ich brauche elf Sekunden.
```

```
 def printFuture[T](f: Future[T]) = f onComplete {
    case Success(t) => println(t)
    case Failure(f) => throw f
  }

def example = {
    val a1 = futureThatTakes(10)("Ich brauche zehn Sekunden.")
    val a2 = futureThatTakes(5)("Ich brauche fünf Sekunden.")
    val a3 = futureThatTakes(11)("Ich brauche elf Sekunden.")

    val fs : List[Future[String]] = List(a1,a2,a3)

    fs map printFuture

  }
```

FUTURE ZURÜCK IN DEN SYNCHRONEN PROGRAMMABLAUF ZWEINGEN

- Wann fertig ist nicht vorhersehbar
- Man kann bestenfalls einen definierten Zeitraum auf das vollenden warten
- **Await.result(<Future>,< Wartezeit>)**

```
// Enough time

scala> Await.result(
  futureThatTakes(10)("Ich brauche zehn Sekunden."),
  10 seconds
)
res6: String = Ich brauche zehn Sekunden.


// Not enough Time

scala> Await.result(
  futureThatTakes(10)("Ich brauche zehn Sekunden."),
  5 seconds)
java.util.concurrent.TimeoutException: Futures timed out after [5 seconds]
    at scala.concurrent.impl.Promise$DefaultPromise.ready(Promise.scala:219)
```

SEQ[FUTURE[A]] => FUTURE[SEQ[A]]

- Wenn man auf die Ergebnisse mehrere Futures zugreifen muss die voneinander abhängig sind

```
scala>   lazy val futureSeq = Future.sequence(
     |     List(
     |       futureThatTakes(10)("Ich brauche zehn Sekunden."),
     |       futureThatTakes(5)("Ich brauche fünf Sekunden."),
     |       futureThatTakes(12)("Ich brauche zwölf Sekunden.")
     |     )
     |   )
futureSeq: scala.concurrent.Future[List[String]] = <lazy>


scala>   Await.result(futureSeq, 12 seconds)
res8: List[String] = List(Ich brauche zehn Sekunden., Ich brauche fünf Sekunden., Ich brauche
zwölf Sekunden.)
```

PITFALLS COMBINING FUTURES

- Problem beim kombinieren der Futures

```
  def combineDoneWrong = {
    val combined = for {
        f1 <- futureThatTakes(10)("Ich brauche zehn Sekunden.")
        f2 <- futureThatTakes(5)("Ich brauche fünf Sekunden.")
    } yield f1 + f2

    Await.result(combined, 11 seconds)
  }
```

PITFALLS COMBINING FUTURES

- Richtig kombiniert

```
  def combineDoneRight = {
    val fF1 = futureThatTakes(10)("Ich brauche zehn Sekunden.")
    val fF2 = futureThatTakes(5)("Ich brauche fünf Sekunden.")

    val combined = for {
        f1 <- fF1
        f2 <- fF2
    } yield f1 + f2

    Await.result(combined, 11 seconds)
  }
```

PITFALLS COMBINING FUTURES

- falsch kombiniert

```
/**
  * scala> Await.result(combined, 11 seconds)
  * warning: there were 1 feature warning(s); re-run with -feature for details
  * java.util.concurrent.TimeoutException: Futures timed out after [11
seconds]
  */
```

- Richtig kombiniert

```
/**
  * scala> Await.result(combined, 11 seconds)
  * warning: there were 1 feature warning(s); re-run with -feature for details
  * res7: String = Ich brauche zehn Sekunden.Ich brauche fünf Sekunden.
  */
```

MAPS UND LISTEN : NÜTZLICHE FUNKTIONEN

- partition
- groupBy

PARTITION

- Liste aufteilen

```
scala>
(1 to 10) partition (_ % 2 == 0)

res1: (
  scala.collection.immutable.IndexedSeq[Int],
  scala.collection.immutable.IndexedSeq[Int]
  )
= (
  Vector(2, 4, 6, 8, 10),
  Vector(1, 3, 5, 7, 9)
)
```

GROUPBY

- Listenelemente gruppieren
- Ergebnis ist eine Map

```
scala> (1 to 10) groupBy (identity)
res2: scala.collection.immutable.Map[Int,scala.collection.immutable.IndexedSeq[Int]] =
Map(
   5 -> Vector(5),
   1 -> Vector(1), ...)

scala> (1 to 10) groupBy (_ % 4 == 0)
res3: scala.collection.immutable.Map[Boolean,scala.collection.immutable.IndexedSeq[Int]] =
 Map(
  false -> Vector(1, 2, 3, 5, 6, 7, 9, 10),
  true -> Vector(4, 8)
)

scala> "Lagerregal".toList.groupBy(x => x.toLower)
res12: scala.collection.immutable.Map[Char,List[Char]] =
Map(
  e -> List(e, e),
  a -> List(a, a),
  g -> List(g, g),
  l -> List(L, l),
  r -> List(r, r)
)
```

KOMPOSITION MIT STACKABLE TRAITS

SZENARIO

- Extraktoren liefern Extractions
- Extraktoren existieren in verschiedenen Ausprägungen, z.b. mit Datenbankanbindung
- Extraktoren sollen beliebig kombiniert werden können, Kombinationsreihenfolge ist wichtig

TYPEN UND BASISTRAITS

- Extraction

```
case class Extraction(value: String, src: Strin
```

- Extrations

```
type Extractions =  Map[String,Extraction]
```

- Extractors

```
  trait Extractors {
    def process : Extractions
  }
```

REGELBASIERTE EXTRAKTOREN

- Regelbasierter Extraktor, liefter stets die gleichen Werten

```
class RuleExtractors extends Extractors{

  def process : Extractions = {
    // Message
    println("I'm a stupid Rule based Extraction System")

    Map(
      "name" -> Extraction("Andi","rule"),
      "amount" -> Extraction("2.50 €", "rule"),
      "bic" -> Extraction("ABCEDFG","rule")
    )
  }
}
```

TEMPLATING EXTRAKTOREN

- Template übeschreibt den Wert unter „amount“

```
trait TemplatingOverwrite extends Extractors{

  abstract override def process : Extractions = {
    println("I use mighty templates")
    super.process ++
      Map("amount" ->
        Extraction("2500 €“,"temlating")
      )
  }

}
```

EXTRAKTOREN MIT DATENBANK

- Schlägt BIC nach
- Ersetzt „name“ auf Basis des bicNameMappings

```
trait BICNameDatabase extends Extractors{

    lazy val oldExtractions =  super.process

    val bicNameDB = Map("ABCEDFG" -> "cristof")

    def lookUpExtraction : Option[Extraction] = for {
          Extraction(bic,_) <- oldExtractions get "bic"
          name              <- bicNameDB get bic
        } yield Extraction(name,"BICNameStore")


    abstract override def process : Extractions = {
      println("I use a BICStore to find the right names")
      lookUpExtraction map (ex => oldExtractions ++ Map("name" -> ex) )
          getOrElse oldExtractions
    }

  }
```

STACKABLE TRAITS - BESONDERHEITEN

- **abstract override** : Vorraussetzung für Stackable trait
- Basisklasse entweder als **trait** oder **abstract class**
- Für feinere Steuerung stehen **self type annotions** zur Verfügung

KOMPOSITION MIT TRAITS I

```
scala> val e = new RuleExtractors
e: stackable.RuleExtractors = stackable.package$RuleExtractors@3652158

scala> e.process
I'm a stupid Rule based Extraction System
res1: stackable.Extractions =
Map(name -> Extraction(Andi,rule), amount -> Extraction(2.50 €,rule),
 bic -> Extraction(ABCEDFG,rule))

scala>

scala> val e = new RuleExtractors with TemplatingOverwrite
e: stackable.RuleExtractors with stackable.TemplatingOverwrite =
$anon$1@18942c42

scala> e.process
I use mighty templates
I'm a stupid Rule based Extraction System
res2: stackable.Extractions =
Map(name -> Extraction(Andi,rule), amount -> Extraction(2500 €,temlating),
 bic -> Extraction(ABCEDFG,rule))
```

STACKABLE TRAITS

```
scala> val e = new RuleExtractors with TemplatingOverwrite with BICNameDatabase
e: stackable.RuleExtractors with
stackable.TemplatingOverwrite with stackable.BICNameDatabase = $anon$1@57172439

scala> e.process
I use a BICStore to find the right names
I use mighty templates
I'm a stupid Rule based Extraction System
res3: stackable.Extractions =
Map(name -> Extraction(cristof,BICNameStore), amount -> Extraction(2500 €,temlating),
 bic -> Extraction(ABCEDFG,rule))

scala>

scala> val e = new RuleExtractors with BICNameDatabase with TemplatingOverwrite
e: stackable.RuleExtractors with stackable.BICNameDatabase with
stackable.TemplatingOverwrite = $anon$1@17d561a8

scala> e.process
I use mighty templates
I use a BICStore to find the right names
I'm a stupid Rule based Extraction System
res4: stackable.Extractions =
Map(name -> Extraction(cristof,BICNameStore), amount -> Extraction(2500 €,temlating),
 bic -> Extraction(ABCEDFG,rule))
```

DISPATCH

- DSL für asynchrone Webkommunikation
- basiert auf async-http-client

Periodic Table of Dispatch Operators

All operators of Scala's marvelous Dispatch library on a single page

Handle content in a function

^	<<? (values)	POST	>> ((in, charset) => result)	as_source
:/ (host, port)	/ (path)	PUT	>> ((in) => result)	as_str
:/ (host)	<<< (text)	DELETE	>~ ((source) => result)	>>> (out)
/ (path)	<<< (file, content_type)	HEAD	>- ((text) => result)	>:> ((map) => result)
url (url)	<<< (values)	secure	>>~ ((reader) => result)	>+ (block)
	<< (text)	<& (request)	<> ((elem) => result)	~> ((conversion) => result)
	<< (values)	>\ (charset)	</> ((nodeseq) => result)	>+> (block)
	<< (text, content_type)	to_uri	>## ((json) => result)	>! (listener)
	<< (bytes)		>|	
	<:< (map)			
	as (user, passwd)			
	as_! (user, passwd)			
	gzip			

>~ ((source) => result)

Constructs a Handler that accepts a function turning the content (available as a Source object) into some other value.

You might also be interested in the Dispatch book, here on Amazon.

DISPATCH - BENUTZUNG

```
scala> val x = url( "http://www.neumann.biz" )
X: dispatch.Req = Req(<function1>)

scala> Http( x )
res8: dispatch.Future[com.ning.http.client.Response] = scala.concurrent.impl.Promise$DefaultPromise@611be03c

scala> Http( x )
res10: dispatch.Future[com.ning.http.client.Response] = scala.concurrent.impl.Promise$DefaultPromise@79575519

scala> Http( x OK as.String)
res11: dispatch.Future[String] = scala.concurrent.impl.Promise$DefaultPromise@34407323

scala> val res = Http( x OK as.String)
res: dispatch.Future[String] = scala.concurrent.impl.Promise$DefaultPromise@5c4e5452

scala> res()
res15: String =
"<!DOCTYPE html>
<head>
<title>
Neumann.biz
</title>
<link href="/assets/neumann-cfa30a850dc468ea57b3a5fc541096a7.css" media="screen" rel="stylesheet" type="text/css" />
<link href="/assets/coderay-a93152a82bfe36822b966c91b60440b2.css" media="screen" rel="stylesheet" type="text/css" />
<meta content="authenticity_token" name="csrf-param" />
<meta content="tqK..
```

ASYNCHRONE WEBCALLS

- Worldcat API - Varianten von Büchern nach ISBN suchen
- http://xisbn.worldcat.org/xisbnadmin/doc/api.htm

SPECS 2

- Framework für Unit und Acceptance Tests
- s2 String interpolation für Tests
- viele Matcher für Unterschiedliche Testfälle: Sequenzen, Maps, String, DataTables
- HTML Output
- http://etorreborre.github.io/specs2/

AUFBAU EINE SPECIFICATION

```
class DispatchExampleSpec extends Specification{
  def is: Fragments = ???
}
```

```
class DispatchExampleSpec extends Specification with XmlMatchers {def is: Fragments = s2"""
works without net connection
  transform XML to Editions                                          $toEditions
  build api call                                                     $buildApiCall

needs working connection to the Internets
  get XML by ISBN                                                    $getEditionsXML
  compare ISBNs by differing titles                                  $getDifferingTitles

  """
```

HTML AUSGABE

testOptions in Test += Tests.Argument("html")

DispatchExampleSpec

works without net connection

- transform XML to Editions
- build api call

needs working connection to the Internets

- get XML by ISBN
- compare ISBNs by differing titles

Total for specification DispatchExampleSpec	
Finished in	28 ms
Results	4 examples, 0 failure, 0 error

HTML AUSGABE II

DispatchExampleSpec *(issues only)*

works without net connection

- transform XML to Editions
- build api call

needs working connection to the Internets

- get XML by ISBN
  - java.net.ConnectException: http://xisbn.worldcat.org/webservices/xid/isbn/978-0981531649?method=getEditions&format=xml&fl=title%2Cyear%2Clang%2Ced (NettyResponseFuture.java:329)
- compare ISBNs by differing titles
  - java.net.ConnectException: http://xisbn.worldcat.org/webservices/xid/isbn/978-0061020612?method=getEditions&format=xml&fl=title%2Cyear%2Clang%2Ced (NettyResponseFuture.java:329)

Total for specification DispatchExampleSpec	
Finished in	18 ms
Results	4 examples, 0 failure, 2 errors (+2)

DANKE FÜR EURE AUFMERKSAMKEIT :)