A brief introduction to scalaz-stream

Frank S. Thomas

September 18, 2013

What is it?

scalaz-stream is

- a library for stream processing and incremental I/O
- resource safe
- composable
- functional: stream transformations are pure values

Imperative I/O

```
def linesGt40k1(filename: String): Boolean = {
  val src = scala.io.Source.fromFile(filename)
  try {
    var count = 0
    val lines: Iterator[String] = src.getLines
    while (count <= 40000 && lines.hasNext) {
      lines.next
      count += 1
    }
    count > 40000
  }
  finally src.close
}
```

I/O with scalaz-stream

```
def linesGt40k2(filename: String): Boolean =
  scalaz.stream.io.linesR(filename).
    drop(40000).
    once.
    runLast.
    run.
    nonEmpty
```

Stream transformations with Process

```
trait Process[I,O]

case class Emit[I,O](
    head: Seq[O],
    tail: Process[I,O] = Halt[I,O]())
  extends Process[I,O]

case class Await[I,O](
    recv: I => Process[I,O],
    finalizer: Process[I,O] = Halt[I,O]())
  extends Process[I,O]

case class Halt[I,O]() extends Process[I,O]
```

- Process is a state machine with three possible states
- these must be interpreted by a driver for effects to occur

Combinators

Modify and compose processes:

- `def map[O2](f: O => O2): Process[I,O2]`
- `def flatMap[O2](f: O => Process[I,O2]): Process[I,O2]`
- `def append(p: => Process[I,O]): Process[I,O]`
- `def filter(f: I => Boolean): Process[I,I]`
- `def take(n: Int): Process[I,I]`
- `def takeWhile(f: I => Boolean): Process[I,I]`
- ... and many more

Feed the output of `this` to the input of `p2`:

- `def pipe[O2](p2: Process[O,O2]): Process[I,O2]`

Another Example

```
import scalaz.stream._
import scalaz.concurrent.Task

val converter: Task[Unit] =
  io.linesR("testdata/fahrenheit.txt").
     filter(s =>
       !s.trim.isEmpty && !s.startsWith("//")).
     map(line =>
       fahrenheitToCelsius(line.toDouble).toString).
     intersperse("\n").
     pipe(process1.utf8Encode).
     to(io.fileChunkW("testdata/celsius.txt")).
     run

val u: Unit = converter.run
```

References

- https://github.com/scalaz/scalaz-stream
- Chapter 15 of Functional Programming in Scala
  http://manning.com/bjarnason/
- Advanced Stream Processing in Scala
  http://www.youtube.com/watch?v=8fC2V9HX_m8