JavaScript as an Embedded DSL in Scala

Nada Amin, Grzegorz Kossakowski

Scala Days 2012

Lightweight Modular Staging

- Embedding DSLs as libraries in Scala
- Staging
- Rep[T] vs T

```
def prog1(b: Boolean, x: Rep[Int]) = if (b) x else x+1
def prog2(b: Rep[Boolean], x: Rep[Int]) = if (b) x else x+1
```

```
 prog1(true, x) //-> x
 prog2(b, x)    //-> If(b, x, Plus(x, Const(1)))
```

DSL program and generated code

Scala

```
def test(n: Rep[Int]): Rep[Array[Int]] =
  for (i <- range(0, n); j <- range(0, n)) yield i*j
```

JS

```
function test(x0) {
  var x6 = []
  for(var x1=0;x1<x0;x1++){
    var x4 = []
    for(var x2=0;x2<x0;x2++){
      var x3 = x1 * x2
      x4[x2]=x3
    }
    x6.splice.apply(x6, [x6.length,0].concat(x4))
  }
  return x6
}
```

Koch Snowflake Example

Scala

```
def snowflake = fun { (c: Rep[Context],
n: Rep[Int], x: Rep[Int], y: Rep[Int],
len: Rep[Int]) =>

  def leg: Rep[Int => Unit] = fun { n =>
    c.save();
    if (n == 0) {
      c.lineTo(len, 0);
    } else {
      c.scale(1.0/3,1.0/3);
      leg(n-1);
      c.rotate(60*deg);
      leg(n-1);
      c.rotate(-120*deg);
      leg(n-1);
      c.rotate(60*deg);
      leg(n-1);
    }
    c.restore();
    c.translate(len, 0);
  }

  c.save();
  c.translate(x,y);
  c.moveTo(0,0);
  leg(n);
  c.rotate(-120*deg);
  leg(n);
  c.rotate(-120*deg);
  leg(n);
  c.closePath();
  c.restore();
}
```

JS

```
function snowflake(c, n,
x, y, len) {

  function leg(n) {
    c.save();
    if (n == 0) {
      c.lineTo(len, 0);
    } else {
      c.scale(1/3,1/3);
      leg(n-1);
      c.rotate(60*deg);
      leg(n-1);
      c.rotate(-120*deg);
      leg(n-1);
      c.rotate(60*deg);
      leg(n-1);
    }
    c.restore();
    c.translate(len, 0);
  }

  c.save();
  c.translate(x,y);
  c.moveTo(0,0);
  leg(n);
  c.rotate(-120*deg);
  leg(n);
  c.rotate(-120*deg);
  leg(n);
  c.closePath();
  c.restore();
}
```

Koch Snowflake Example

Scala

```
def snowflake = fun { (c: Rep[Context],
n: Rep[Int], x: Rep[Int], y: Rep[Int],
len: Rep[Int]) =>

  def leg: Rep[Int => Unit] = fun { n =>
    c.save();
    if (n == 0) {
      c.lineTo(len, 0);
    } else {
      c.scale(1.0/3,1.0/3);
      leg(n-1);
      c.rotate(60*deg);
      leg(n-1);
      c.rotate(-120*deg);
      leg(n-1);
      c.rotate(60*deg);
      leg(n-1);
    }
    c.restore();
    c.translate(len, 0);
  }

  c.save();
  c.translate(x,y);
  c.moveTo(0,0);
  leg(n);
  c.rotate(-120*deg);
  leg(n);
  c.rotate(-120*deg);
  leg(n);
  c.closePath();
  c.restore();
}
```

JS

```
function snowflake(c, n,
x, y, len) {

  function leg(n) {
    c.save();
    if (n == 0) {
      c.lineTo(len, 0);
    } else {
      c.scale(1/3,1/3);
      leg(n-1);
      c.rotate(60*deg);
      leg(n-1);
      c.rotate(-120*deg);
      leg(n-1);
      c.rotate(60*deg);
      leg(n-1);
    }
    c.restore();
    c.translate(len, 0);
  }

  c.save();
  c.translate(x,y);
  c.moveTo(0,0);
  leg(n);
  c.rotate(-120*deg);
  leg(n);
  c.rotate(-120*deg);
  leg(n);
  c.closePath();
  c.restore();
}
```

Show it!

API

```
//c: Rep[Context]
def leg: Rep[Int => Unit] = fun { n =>
  c.save();
  if (n == 0) {
    c.lineTo(len, 0);
  } else {
    c.scale(1.0/3,1.0/3);
    leg(n-1);
    c.rotate(60*deg);
    leg(n-1);
    c.rotate(-120*deg);
    leg(n-1);
    c.rotate(60*deg);
    leg(n-1);
  }
  c.restore();
  c.translate(len, 0);
}
```

```
trait Context
trait ContextOps {
  def save(): Rep[Unit]

  def lineTo(x: Rep[Int], y:
Rep[Int]): Rep[Unit]

  def scale(x1: Rep[Double], x2:
Rep[Double]): Rep[Unit]

  def rotate(x: Rep[Double]):
Rep[Unit]

//...
}
```

```
implicit def repToContextOps(x: Rep[Context]): ContextOps =
  repProxy[Context,ContextOps](x)
```

Rep[T] = T

```
trait Context
trait ContextOps {
  def save(): Rep[Unit]

  def lineTo(x: Rep[Int], y: Rep[Int]): Rep[Unit]

  def scale(x1: Rep[Double], x2: Rep[Double]): Rep[Unit]

  def rotate(x: Rep[Double]): Rep[Unit]

//...
}
```

Rep[T] = T

```
trait Context
trait ContextOps {
  def save(): Unit

  def lineTo(x: Int, y: Int): Unit

  def scale(x1: Double, x2: Double): Unit

  def rotate(x: Double): Unit

//...
}
```

Validation

```
val commentForm: Form[Comment] = Form(
  mapping(
    "firstname" -> nonEmptyText,
    "lastname" -> nonEmptyText,
    "company" -> optional(text),
    "email" -> email,
    "phone" -> optional(text verifying jsPattern("""[0-9.+]+""", "c.phone", "e.phone")),
    "message" -> nonEmptyText.verifying(jsConstraint("c.nice", "e.nice") { new { def eval(c: JS) = { import c._;
            (msg: Rep[String]) => {
              val words = msg.split(" ")
              def countWords(regex: String) =
                words.filter(regex.r.test(_)).length
              val hateCount = countWords("[Hh]ate|[Ss]uck")
              val loveCount = countWords("[Ll]ove|[Rr]ock")
              hateCount < loveCount
            }
     }}})
  )(Comment.apply)(Comment.unapply)
)
```

Validation

```
val commentForm: Form[Comment] = Form(
  mapping(
    "firstname" -> nonEmptyText,
    "lastname" -> nonEmptyText,
    "company" -> optional(text),
    "email" -> email,
    "phone" -> optional(text verifying jsPattern("""[0-9.+]+""", "c.phone", "e.phone")),
    "message" -> nonEmptyText.verifying(jsConstraint("c.nice", "e.nice") { new { def eval(c: JS) = { import c._;
               (msg: Rep[String]) => {
                  val words = msg.split(" ")
                  def countWords(regex: String) =
                    words.filter(regex.r.test(_)).length
                  val hateCount = countWords("[Hh]ate|[Ss]uck")
                  val loveCount = countWords("[Ll]ove|[Rr]ock")
                  hateCount < loveCount
               }
     }}})
  )(Comment.apply)(Comment.unapply)
)
```

Show it!

Typed Object Literals

```
def fetchTweets(username: Rep[String]) =
(ajax.get {
  new JSLiteral {
    val url = "http://api.twitter.com/1/statuses/user_timeline.json"
    val `type` = "GET"
    val dataType = "jsonp"
    val data = new JSLiteral {
      val screen_name = user
      val include_rts = true
      val count = 5
      val include_entities = true
    }
  }
}).as[TwitterResponse]

type TwitterResponse =
  Array[JSLiteral {val text: String}]
```

Scala

```
{
  url: "http://api.twitter.com/1/statuses/user_timeline.json/",
  type: "GET",
  dataType: "jsonp",
  data: {
    screen_name : username,
    include_rts : true,
    count : 5,
    include_entities : true
  }
}
```

JS

Reified Classes

Scala

```
class Cell[A:Manifest] {
  private var value: Rep[A] = null
  private var defined: Rep[Boolean] =
    false
  private val queue = array[A => Unit]()

  def get(k: Rep[A => Unit]) = {
    if (defined) k(value)
    else queue.push(k)
  }

  def set(v: Rep[A]) = {
    if (defined) () // error
    else {
      value = v
      defined = true
      for (f <- queue) { f(v) } // spawn
    }
  }
}
```

JS

```
var x0 = function() {
  this.$init$()
}
x0.prototype.$init$ = function() {
  var x3 = this.value = null
  var x4 = this.defined = false
  var x5 = []
  var x6 = this.queue = x5
}
x0.prototype.get = function(x8) {
  //...
}
x0.prototype.set = function(x20) {
  //...
}
```

Reified Classes

Scala

```
class Cell[A:Manifest] {
  private var value: Rep[A] = null
  private var defined: Rep[Boolean] =
    false
  private val queue = array[A => Unit]()

  def get(k: Rep[A => Unit]) = {
    if (defined) k(value)
    else queue.push(k)
  }

  def set(v: Rep[A]) = {
    if (defined) () // error
    else {
      value = v
      defined = true
      for (f <- queue) { f(v) } // spawn
    }
  }
}
```

JS

```
var x0 = function() {
  this.$init$()
}
x0.prototype.$init$ = function() {
  var x3 = this.value = null
  var x4 = this.defined = false
  var x5 = []
  var x6 = this.queue = x5
}
x0.prototype.get = function(x8) {
  //...
}
x0.prototype.set = function(x20) {
  //...
}
```

Puzzle

JS

```
var xs = [1, 2, 3]
var i = 0
var msg = null
function f1() {
  if (i < xs.length) {
    window.setTimeout(f2, xs[i]*1000)
    msg = xs[i]
    i++
  } else {
    console.log("done")
  }
}
function f2() {
  console.log(msg)
  f1()
}
f1()
```

What does this code do?

Puzzle

JS

```
var xs = [1, 2, 3]
var i = 0
var msg = null
function f1() {
  if (i < xs.length) {
    window.setTimeout(f2, xs[i]*1000)
    msg = xs[i]
    i++
  } else {
    console.log("done")
  }
}
function f2() {
  console.log(msg)
  f1()
}
f1()
```

Puzzle

JS

```
var xs = [1, 2, 3]
var i = 0
var msg = null
function f1() {
  if (i < xs.length) {
    window.setTimeout(f2, xs[i]*1000)
    msg = xs[i]
    i++
  } else {
    console.log("done")
  }
}
function f2() {
  console.log(msg)
  f1()
}
f1()
```

Scala

```
val xs = array(1, 2, 3)
for (x <- xs.suspendable) {
  sleep(x * 1000)
  console.log(String.valueOf(x))
}
console.log("done")
```

Twitter

```
val users = array("gkossakowski", "odersky", "adriaanm")
for (user <- users.parSuspendable) {
  console.log("fetching " + user)
  val tweets = fetchTweets(user)
  console.log("finished fetching " + user)
  for (t <- tweets)
    console.log("fetched " + t.text)
}
console.log("done")
```

Twitter

```
val users = array("gkossakowski", "odersky", "adriaanm")
for (user <- users.parSuspendable) {
  console.log("fetching " + user)
  val tweets = fetchTweets(user)
  console.log("finished fetching " + user)
  for (t <- tweets)
    console.log("fetched " + t.text)
}
console.log("done")
```

Show it!

Under the hood

- We’ve used the CPS plugin in our DSL to build a non-trivial abstraction
- We discovered that we don’t need to reify the continuation monad - it’s enough to reify effects of the monad
- Staging allows us to strip out most of the abstractions at staging time so they don’t leak into generated JavaScript code

Under the hood

See our ECOOP 2012 paper for implementation details

The only equation in this talk

```
Rep[A => B] • Rep[B => C] == Rep[(A => B) • (B => C)]
```

Summary

- Type-Safe by re-using Scala's type system
- DSL can be evaluated in Scala, enabling sharing code between client and server
- Intuitive abstractions for objects
- DSL as a thin layer on top of JavaScript, on top of which abstractions can be built

http://github.com/js-scala

Questions?

http://github.com/js-scala