Lift

and some SQueryl

Tony Kay

Source and Slides
at
http://github.com/awkay/lift_squeryl_demo

Web Dev + Scala

- Some nice synergy:
  - Support for XML literals and first-class XML processing
  - Possible to build quality DSLs
    - Expressive, yet type-safe, interfacing with common dependencies (db, css, javascript)
  - Much easier to make the plumbing into nearly invisible abstractions

A bit on Persistence

- Relational database access from Java – Bleah
  - Annotate the hell out of things in Hibernate (or toil away in XML hell)
  - Throw complex queries around as strings (or worse, XML)
  - Opaque performance problems
  - Aggregation? Column calculations? DB functions?
  - Anyone else want to complain?

Two Quality DSLs

- Squeryl - an SQL look-alike, but syntax and type-checked by your compiler.
  - Resonates with my desire to have a DSL closely match the language I am used to in the problem domain.
  - Has integration with Lift Record
- Scala Query (a.k.a. Slick) - Integration of relational queries into scala `for` comprehensions.
  - Supports more than just relational databases
  - Somewhat opaque if you "think in SQL"
  - Nested queries are a bit unnatural

Writing Queries

- Type-safe - Schema via POSOs
- Syntax-checked
- Literals in Scala = Literals in SQL
- A little touchy on primitive comparisons
- Nullables via Option[T]

```
val acctQuery =
  from(accounts)( a =>
     where(a.last_name like "Abo%" and
           (a.id gt 0) and
           a.map(_.email) like "%gmail.com"
           )
     select(a.id, a.first_name, a.last_name)
   )
```

Processing Results

- Pull entire rows, or just desired columns
- Query object acts on database when mapped over, converted to a collection, etc.

```
// select(a.id, ...)
var fnames = query.map(row => row._1)

// select(a)
fnames = query.map(acct => acct.first_name)

// Convert to a collection and do further processing
fnames = query.toList.map(_._1)
```

Squeryl - Subqueries

- Can use subqueries

```
val accts_w_orders = from(order)(o =>
  where(...)
  select(o.account_id))
val acctQuery =
  from(accounts)( a =>
     where( (a.id in accts_w_orders) )
     select(a.id, a.first_name, a.last_name)
   )
```

Complex Queries

- Support for joins, functions, aggregates

```
join(accounts, invoices.leftOuter)( (a, i) =>
   where(a.last_name === "Smith")
   groupBy(a.id, a.first_name, a.last_name)
   compute(sum(i.total))
   on(a.id === i.account.id)
   )
```

Dynamic Clauses

- `None : Option[T]` causes removal of clause

```
val desiredLastName : Option[String] = None
val q =
  from(accounts)(a =>
     where(
        a.last_name === desiredLastName.?
        and ...)
     select(a)
     )
```

Other features

- Optional ORM-style API for some ops
  - One to many/many to many, etc.
  - save/update of object
- Direct DML (update/delete)
  - Updates can be written to affect a single column, or entire tables, as in SQL
- Pretty good schema generation support
- Extensible type system
- Extensible to use non-std db functions (e.g. PostgreSQL ageInYears(dateCol))

Learning More

- Docs on the main site are great!

Lift

What Makes Lift Compelling?

- Composable abstractions
- Web designer ***really*** deals with the porcelain
- Framework makes desktop-quality webapps easy
  - Impressive support of AJAX and Comet
- You get to work in Scala
- You can build amazing things with very little work

Leveraging Scala

- Most common pattern for dynamic interaction:
  - Pass in a lambda/PaF, which is called when an action is needed.
  - Used for normal requests (GET/POST), redirects, AJAX/Comet, form post-processing, etc.

```
var svar = ""
SHtml.a(() => { launchMissles }, Text("launch"))
SHtml.ajaxText("initial", (v) => { svar = v; update })
S.redirectTo("/page", () => { someRequestVar(4) })
...
```

Leveraging Scala

- Direct and natural ouput via XML support

```
class XSnippet {
  def render(n : NodeSeq) : NodeSeq = <span id="x">{ x }</span>
}
```

Leveraging Scala

- DSL for XML transformation (rendering.sc):

```
<div class="lift:TimeSnippet">
    <span class="date">Jan 3, 2013</span>
    <span class="time">10:34pm</span>
</div>

class TimeSnippet {
  def todaysDate: String = "" // ...
  def currentTime: String = "" // ...
  def render = ".date *" #> todaysDate.toString &
    ".time [class+]" #> "highlighted" &
    ".time *" #> currentTime
}

<div>
  <span class="date">Feb 8, 2013</span>
  <span class="time highlighted">8:15pm</span>
</div>
```

Designer-friendly Templates

- No code in the view!
  - Like Enhydra (1999) or Apache Wicket (2005+)
  - Keeps logic/model from creeping into hard-to-maintain locations
  - Makes interaction with web designers a breeze
- Views manipulated with node sequence transforms, or CSS selector transforms.
- Demo1

Simple AJAX

- AJAX interactions without the Javascript
  - Easily embed dynamic controls
  - Everything hooks to scala functions
  - Can easily output javascript functions that can interact with server, without having to know anything about routing.
    - Useful when hand-coded javascript UI needs to interact with the server (e.g. JQuery DnD).
- Demo 2

Help System Topic

- Anywhere in any page:

```
<a class="lift:Help?topic=confused">confusing thing</a>
```

- In help/confused[_lang[_country]].html

```
<div class="help">
    <h1>Confusing Thing</h1>
    <p>Wow, You sure look confused! Have you tried?</p>

    <ul>
        <li>Wiggling the mouse?</li>
        <li>Connecting the power?</li>
    </ul>

    <button class="lift:Help.dismiss">OK</button>
</div>
```

Sample – Help System

- A complete, production-ready, help system w/i18n:

```
object Help extends Loggable {
  def render(n: NodeSeq): NodeSeq = {
    val content = if(n.text.length > 0) n.text else "(?)"
    val topic = S.attr("topic")
    if (topic.isDefined)
      SHtml.a(() => {
          S.runTemplate(List("help", topic.get)).
          map(ns => ModalDialog(ns)) openOr Alert(S?"Help missing.")
        }, Text(content), "class" -> "help_link")
    else {
      logger.error("Missing topic in help request!")
      Text("")
    }
  }
  def dismiss = SHtml.ajaxButton(I.tr("OK"), () => Unblock,
                                 "class" -> "dismiss_button")
}
```

Simple Comet

- Push updates to browser
- Lift handles all connection details
- Same basic tools for rendering as AJAX interaction
- Uses a custom set of Lift actor classes
- Demo 3

Leveraging Scala

- Closure over variables allows conversational state without complications (Demo4)

```
class SearchSnippet {
  def search = {
    var query = ""

    def currentResults: NodeSeq = // ... based on query ...
    def updateResults(q: String): JsCmd = {
      query = q
      Replace("results", currentResults)
    }

    ".query_field" #> SHtml.ajaxText(query, updateResults _) &
      "#results" #> currentResults
  }
}
```

Wiring

- Lift support spreadsheet-like cells
- Cell updates are automatically piggybacked on other AJAX/Comet updates!
- See Demo 5

```
val numberOfItemsChecked = ValueCell(Database.items.count(_.complete))
val cashOnHand = ValueCell(12)
val moneyEarnedPerItem = 6
val moneyEarned = numberOfItemsChecked.lift(_ * moneyEarnedPerItem)
val projectedMoney = moneyEarned.lift(cashOnHand)(_ + _)
```

+

```
".cash_on_hand" #> WiringUI.asText(cashOnHand, JqWiringSupport.fade)
```

Mind-blowing Integration

- Put as many unrelated bits on a page as you want
- See Demo 6

RequestVar

- Type-safe data passing
- Scope to the request AND subsequent related AJAX/Comet interaction
- SessionVar also available

```
object SnippetA {
  object TheID extends RequestVar[Int](0)
  ...
}
...
class SnippetB {
  def render = ".link" #> SHtml.a(() => {
    S.redirectTo("/page", () => SnippetA.TheID(42))
  }, Text("click me"))
}
...
  <div class="lift:SnippetB"><a class="link"></a></div>
```

Secure

- Functions <=> Objects + Lift == Secure
  - All AJAX/Comet/Forms/Request parameters are encoded as handles to the objects (usually handler functions) in server memory.
  - Handles are what the client browser sees.
    - Nothing for an XSS attacker to attack
    - Encoding/decoding all data done for you, with proper escapes.
- No Query Parameters needed
  - You can still do REST, of course

Many More Nice Bits

- Persistence agnostic, but includes Record/Mapper
  - Record is like ActiveRecord.
  - Mapper is a richer ORM API.
- Screen/Wizard for succinct forms
- Parallel Snippet execution for reducing latency
- Long-running snippets and lazy loading
- Menu DSL with centralized URL access control, groups, dynamic content, etc

Learn More

- http://cookbook.liftweb.net/ has lots of useful tips
- Two online books:
  - Exploring Lift
  - Simply Lift
- Read the source of demos
- Explore the source of Lift itself
  - Maven+Scala IDE makes this a breeze
- My Demo code/maven pom.xml file on github
  - https://github.com/awkay/lift_squeryl_demo