Getting Pushy

Server Push in Scala and Clojure

David Pollak

September, 2013, Strange Loop

Why the lab coat?

- Lost a bet to @fogus/@meangrape/@seriouspony
- Spreadsheet-Inator

@dpp

- Lift & Scala: 7 years
- Clojure & Plugh: This year
- Crazy passionate lawyer-trained dude

BRICK ALLOY

- My new Consultancy
- Delivering Cloud Systems that Drive Business Success

Across Address Spaces

- Transport
- Marshalling
- Program Semantics

Transport

- HTTP unless there's a good reason
- Traverses firewalls
- Understood semantics

Marshalling

- JSON unless...

  Your language is better...

  like Erlang & Clojure

Program Semantics

- Normal flow of control (RMI)?
- Hand-rolled per invocation?
- Futures/Actors | Channels | Queues?

About HTTP

- Limited Browser/Server Connections
  (1 to 8 per server)
- Proxies Unreliable
- Don't trust the wire

Key Goals

- "Native" & Lightweight Semantics
- Mostly transparent to transport vagaries

Respect the GUID

- Cross Address Space REF
  - Function
  - Component
  - Actor/Channel/Thingy
- Easy to serialize
- GC a challenge

Confirm(ish) Delivery

- Client and Server have delivery queues
- Server Queue based on clock value
- Client Queue based on ACK, but de-dupped by server

HTTP Push

- Used to be hard: Comet/Long Poll
- Marginally easier with Web Sockets
- Still Cross address space problem

Lift

- @dpp's Scala Libraries
- Most mature, best maintained libraries in Scala-land
- Actors, Futures, Monads, JSON, Web...
- Best Server Push!

Lift

- Ajax: Single HTTP channel, de-dupped
- Comet: Multiplexed, Versioned, Poll
- Server (mostly) cognizant of HTML

Lift 3: Data Focus

- Server knows about data
- Server cares much less about HTML
- No Change to underlying transport

Lift Data Chat


```
def render =
  for (sess <- S.session) yield
  Script(
  JsCrVar("serverFuncs",
    sess.buildRoundtrip(List[RoundTripInfo](

      "start" -> ((_: String) =>
                    PushyStream.stream),

      "chat" -> ((s: String) => {
        PushyStream.push(s)
        Stream.empty})))))
```

Lift Streaming


```
object PushyStream {
  private val in = new LinkedBlockingQueue[String]
  lazy val stream = Stream.continually(in.take())

  def push(s: String) = {
    in.add(s)
  }
}
```

Lift Client HTML

```
<div ng-controller="Chatter">
    <h2>Chat App</h2>

    <ul>
        <li ng-repeat="x in chats">{{x}}</li>
    </ul>

    <input ng-model="line">
    <button ng-click="send()">Chat</button>
</div>
```

Lift Client JS


```
$scope.chats = ["welcome"];

$scope.line = "";

$scope.send = function () {
  window.serverFuncs.chat($scope.line);
  $scope.line = "";
}

window.serverFuncs.start("").
then(function(v) {
  $scope.$apply(function() {
    $scope.chats.push(v);
  })});
```

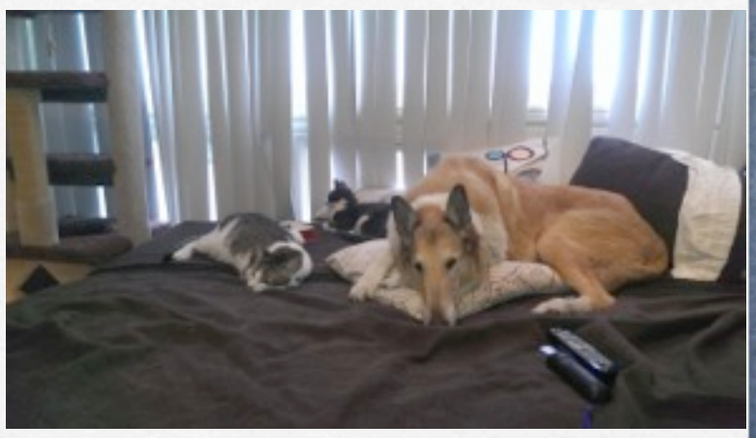
Plugh

- @dpp's Clojure libraries
- AngularJS focused web stuff
- Cross-address-space analytics stuff

Clojure core.async

- Messaging between endpoints
- synchronous and async send/receive
- Substrate for Plugh

Plugh Client

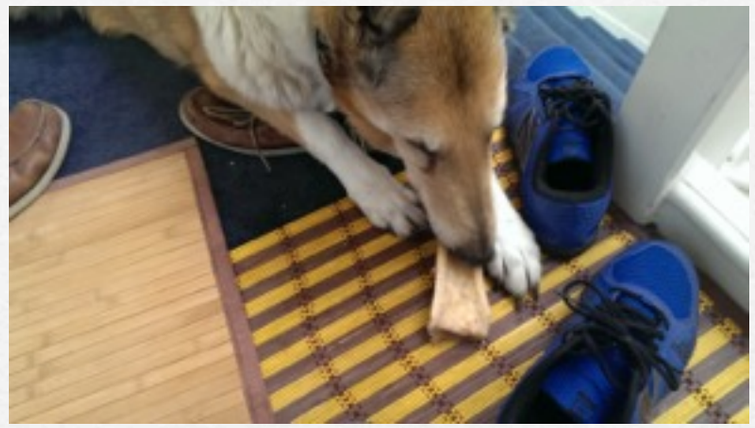


```
(s-set :chats (clj->js []))
(s-set :line "")

(defn.scope send []
  (let [msg (:line $scope)]
    (go
      (>! server-chan {:msg msg})))
  (assoc! $scope :line ""))

(let [rc (chan)]
  (go (>! server-chan {:add rc}))

  (go (while true
      (let [chats (<! rc)]
        (in-scope (doseq [m chats]
           (.push (:chats $scope) m)))))))
```

Plugh Server

```
(while true
  (let [v (<! server-chan)]
    (when-let [n (:add v)]
      (swap! listeners #(conj % n))
      (>! n @chats))

    (when-let [chat-msg (:msg v)]
      (swap! chats #(conj % chat-msg))
      (doseq [ch @listeners]
        (>! ch [chat-msg])))))))
```

Spreadsheet-Inator



```
(:name (.toLowerCase it))
(:age it)
(:age5 (/ (to-num (:age it)) 5))
(:#totals :age sum, :age5 avg)
(:#filter -> it :name .-length (> 4))
```

Questions?

Lift Chat View


```
<div>
    <h2>Chat App</h2>

    <ul data-lift="comet?type=Chat">
        <li>Chat Line</li>
        <li class="clearable">Line 2</li>
        <li class="clearable">Line 3</li>
        <li class="clearable">Line 4</li>
    </ul>

    <form data-lift="form.ajax">
      <input data-lift="ChatIn" id="chat_in">
      <button>Chat</button>
    </form>
</div>
```

Lift Chat Server


```
object ChatServer extends LiftActor with
                   ListenerManager {
  private var chats = Vector("Welcome")

  def createUpdate = chats

  override def lowPriority = {
    case s: String => chats :+= s
                      updateListeners()
  }
}
```

Lift Chat Pusher

```
class Chat extends CometActor with CometListener {
  private var chats = Vector("")

  def registerWith = ChatServer

  def render = ClearClearable andThen
               "li *" #> chats

  override def lowPriority = {
    case v: Vector[String] => chats = v; reRender()
  }
}
```

Lift Chat Ajax

```
object ChatIn {
  def render = SHtml.onSubmit(s => {
    ChatServer ! s
    SetValById("chat_in", "")
  })
}
```