# Lift: 3 hours of Differences

*David Pollak*

*GeeCON 2011*

## About DPP

- *Coding since 1977*
- *Wrote spreadsheets, web frameworks, books, etc.*
- *Lift & Scala for almost 5 years*

## Lift Web Framework

- *Secure*
- *Scalable*
- *Simple*

## Today

- *Comet Chat Example*
- *Add REST*
- *Add Parameterized Menus*

# Chat

- *Real Time*
- *Short*
- *Demonstrates lots of Lift*

## View First

- *Valid Html5 or XHTML*
- *No code in template*
- *URL => View => Logic => Response*

## View

```
<div class="lift:comet?type=Chat">
  Some chat messages
  <ul>
    <li>A message</li>
    <li class="clearable">Hey</li>
    <li class="clearable">another line</li>
  </ul>
</div>
<div>
  <form class="lift:form.ajax">
    <input id="chat_in" class="lift:ChatIn"/>
    <input type="submit" value="Say Something"/>
  </form>
</div>
```

## Snippets

- *Transform DOM => DOM*
- *Bind state*
- *Associate functions with GUIDs*

## Ajax Input

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

# Actors

- *Simply Concurrency*
- *Async Mailbox*
- *Single Threaded Message Processing*

# HTTP vs. Comet

- *Comet -- Simulated Server Push*
  - *Today: Long Polling*
  - *Tomorrow: Web Sockets*
- *Ajax -- secure and without explicit routes*

## Comet

-
  ```
  class Chat extends CometActor with CometListener {
    private var msgs: Vector[String] = Vector()

    def registerWith = ChatServer

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

    def render = ClearClearable andThen "li *" #> msgs
  }
  ```

## Chat Server

-
  ```
  object ChatServer extends ChatRestServer {
    protected var msgs = Vector("Welcome")

    def createUpdate = msgs

    override def lowPriority = ({
      case s: String =>
        msgs :+= s
        updateListeners()
    }:  PartialFunction[Any, Unit]) orElse
    super.lowPriority
  }
  ```

## Helper classes

- *case classes: it's data*
- *implicit conversions -- makes code readable*
- *pattern matching*

## Helpers

-
  ```
  final case class ChatMessage(msg: String, pos: Int)

  object ChatMessage {
    implicit def toCM(p: (String, Int)): ChatMessage =
      ChatMessage(p._1, p._2)
  }

  final case class Messages(msgs: List[ChatMessage])

  final case class GetMessagesAfter(pos: Int, f: Vector[String] => Unit)
  ```

## The REST

- *Uses Scala's Pattern Matching*
- *Type Safe*
- *JSON Goodness (XML too)*

## REST Server

```
object ChatRest extends RestHelper {
  serve {
    case "chat" :: AsInt(pos) :: Nil Get _ =>
      ChatServer.msgAt(pos).flatMap(anyToJValue)

    case "chats" :: AsInt(pos) ::  Nil Get _ =>
      RestContinuation.async {
        reply => ChatServer !
        GetMessagesAfter(pos, msgs =>
          reply(vToR(msgs, pos)))}}

  def vToR(v: Vector[String], off: Int): LiftResponse =
    anyToJValue(Messages(v.toList.zipWithIndex.drop(off).
                         map(v => v: ChatMessage))) match {
        case Full(jv) => jv
        case _ => NotFoundResponse("Could not convert") }}
```

# REST Support

-
  ```
  trait ChatRestServer extends LiftActor with ListenerManager {
    protected def msgs: Vector[String]
    private var waiting: Vector[GetMessagesAfter] = Vector()

    override def updateListeners() {
      val len = msgs.length
      waiting = waiting.flatMap {
        case GetMessagesAfter(pos, f) if pos < len => f(msgs) ; Nil
        case gma => List(gma)}; super.updateListeners()}

    def msgAt(pos: Int) = (this !! pos).collect{case s: String => s}

    override def lowPriority = {
      case i: Int => reply(if (msgs.isDefinedAt(i)) msgs(i) else Empty)
      case gma@ GetMessagesAfter(pos, f) =>
        if (pos < msgs.length) f(msgs)
        else waiting :+= gma}}
  ```

## Boot.scala

- *Loaded once during, well, bootstrapping*
- *All configuration goes here*
- *Examples:*
  - `LiftRules.dispatch.append(ChatRest)`

  -
    ```
    def sitemap = SiteMap(
      Menu.i("Home") / "index",
      ViewMenu.menu,
    ```

## SiteMap

- *Menu Hierarchy*
- *Access Control*
- *Declarative*

## Menus

- `Menu.i("Home") / "index"`

-
  ```
  object ViewMenu {
    val menu =
      Menu.param[ChatMessage]("View", "View",
                              s => for {
                                p <- asInt(s)
                                m <- ChatServer.msgAt(p)
                              } yield ChatMessage(m, p),
                              _.pos.toString) / "view"
  }
  ```

## View a Message

- `The message is <span class="lift:ViewMsg">Message Goes Here</span>.`

## The Snippet

```
class ViewMsg(msg: ChatMessage) {
  def render = "* *" #> msg.msg
}
```

## Revised Chat

-
  ```
  def render = ClearClearable andThen
  "li *" #> (msgs.zipWithIndex.map {
    v => <a href={ViewMenu.menu.calcHref(v)}>{v._1}</a>
  }: Seq[NodeSeq])
  ```
- *OMG -- View in our code!!!*

## Revised View

-
  ```
  <ul>
    <li><a href="#">A message</a></li>
    <li class="clearable">Hey</li>
    <li class="clearable">another line</li>
  </ul>
  ```

## Revised (2nd) Chat

-
  ```
  def render = ClearClearable andThen
  "li *" #> msgs.zipWithIndex.map(v =>
    "a [href]" #> ViewMenu.menu.calcHref(v) andThen
    "a *" #> v._1)
  ```

## Conclusion

- *Lots of functionality & little code... all code in the slides*
- *Lots of type safety and resulting security*
- *Runs in your favorite web container*