Lift 3 - High & Lift-ed Up

Lift 3, AGYNAMIX Torsten Uhlmann

About Me

- Torsten Uhlmann, studied Computer Science
- Software Developer since 1995
- Pascal, C/C++, Perl, Java since 1998, Scala since 2010
- Freelancer since 2005
- Java projects for customers & own products
- Little Lift How-to for starters
- tuhlmann@agynamix.de , @agynamix

What We Do

- Overview of new Lift 3 features
- Showcasing (some) Best Practices
- Presenting Sample Application
- https://github.com/tuhlmann/lift-3-demo
- Contains source and presentation
- Questions - I even try to answer

Best Practices

- http://www.eltimn.com/blog/002-javascript-apps-with-lift-best-practices
- minify JS in production, use unminified in development
- same with Less/CSS
- Define SiteMap so you can reuse it (Site.scala) for linking
- Small JavaScript files, with individual namespaces, for instance per Lift view. Lift-Extras offers helpers

```
// Login.js
App.namespace("views.user");
App.views.user.Login = (function($) {
```

```
<script>
$(document).ready(function() {
  App.views.user.Login.init();
});
</script>
```

Best Practices

- Lift Modules
  - Lift Extras
  - Lift Mongoauth
    - Lift Mapperauth
- Giter8 projects:
  - g8 eltimn/lift-mongo
  - g8 tuhlmann/lift-bootstrap (needs update)

Markdown

- Integrated Markdown Support
- Markdown files in SiteMap (security)
- Easy to process content from other sources, not just files in webapp/
- Markdown parser can be used for user input.
- `MarkdownParser.parse(String)` to turn input text into a NodeSeq

Futures

- Pass a LAFuture to a REST endpoint to make use of continuations
  - Thread is not blocked, using the containers continuation support to register a callback to deliver result when ready

```
case "delay" :: Nil Get _ =>
  LAFuture(() => {
    Thread.sleep(2000)
    <b>Waiting for 2 seconds</b>
  })
```

- Easy conversion between Scala Futures (returned by Dispatch) and LAFutures
- Example: Ask Wikipedia with Dispatch

Roundtrip Promises

- REST:
  - Additional connection to server, connection starvation may happen
  - REST call may run long, **you** take care of security
- Promises:
  - Define a JS object with functions bound to server-side functionality.
  - Client calls - call is forwarded to server, client receives a promise. Uses Lift‘s connection error handling
  - When result is available, callback on promise is executed.

Roundtrip Promises

- Define object on the server

```
// Associate the server functions with client-side functions
for (sess <- S.session) {
  val script = JsCrVar("findSuggestions",
                sess.buildRoundtrip(List[RoundTripInfo]("find" -> doFind _)))
  S.appendGlobalJs(script)
}
```

- Call on client

```
$( "#query_rt" ).autocomplete({
  minLength: 2,
  delay: 200,
  source: function( request, response ) {
    var term = request.term;
    findSuggestions.find(term).then(response);
  }
});
```

Streaming Promises

- If result is a Stream, then client processes results as they become available. On the server:

```
def doFind(param: String): Stream[String] = {
  val words = Languages.l.filter(_.toLowerCase startsWith param.toLowerCase()).sorted

  from (1) take words.size map (num => {
    Thread.sleep(1000)
    words(num-1)
  }): Stream[String]

}
```

- Client:

```
streamingPromise.find(query).then(function(response){
  $('#streaming-search-result').append('<li>'+response+'</li>');
});
```

Client Server Actors

- Actors across address space
  - Send message to server actor - is transferred to client and JS function called
  - call a JS function on the client - is transferred to server actor
  - Parameter is anything that Lift can convert to/from Json
  - Easy setup, easy use

Client Server Actors

- An actor that sends to client Javascript function

```
// get a server-side actor that when we send
// a JSON serializable object, it will send it to the client
// and call the named function with the parameter
val clientProxy = sess.serverActorForClient("window.actorsBridge.messageFromServer")
```

- An actor that receives from client

```
// Create a server-side Actor that will receive messages when
// a function on the client is called
val serverActor = new ScopedLiftActor {

  override def lowPriority = {
    case JString(str) =>
      ClientServerActor ! ChatMessage(user.username.is, str)
```

- Create a Javascript function that we can call

```
S.appendJs(SetExp(JsVar("window.actorsBridge"),
JsExtras.CallNew("App.angular.ActorsBridge", sess.clientActorFor(serverActor)) ))
```

- Creates Javascript:

```
window.actorsBridge = new App.angular.ActorsBridge(function(x) {
  liftAjax.lift_ajaxHandler('F242485830911FLIPVS=' +
  encodeURIComponent(JSON.stringify(x)), null, null, null);});
```

Data Attr Processor

- You can add your processors to data attributes
- In you html file:

```
<div data-scaladays="ScalaDays 2013!">
  <p>Welcome to the Lift 3 Demo at</p>
</div>
```

- Define the processor in Boot:

```
// Add a DataProcessor
LiftRules.dataAttributeProcessor.append {
  case ("scaladays", attributeStr, enclosedNodes, itsSession) =>
        ("p *+" #> attributeStr).apply(enclosedNodes)
}
```

- Also works with Futures

Thank you!

- Thank you for listening!

- Any questions?

Lift 3, AGYNAMIX Torsten Uhlmann