



Lift 3 - High & Lift-ed Up

About Me

- 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
- <http://www.agynamix.de>
- tuhlmann@agynamix.de, Twitter: @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($) {
```

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

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
 - When result is available, callback on promise is executed.

Roundtrip Promises

- Define object on the server

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

- Call on client

```
$( "#typeahead_rt" ).typeahead({
  minLength: 2,
  source: function( query, process ) {
    | findSuggestions_typeahead.find(query).then(process);
  }
});
```

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 your 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?

