Retour d'expérience

Cordova avec Durandal, Knockout et Breeze

[sf≡ir]

Ahmed Radjdi

- Développeur Front End

Programme

1. Durandal & Knockout
    - Durandal key concepts
    - Knockout key concepts
    - Durandal + Knockout : Two-level architecture
2. Breeze
    - Breeze key concepts
    - Offline-first with Breeze
3. Test and build front ressources for each platform

Two-level architecture with Durandal & Knockout

Knockout

“Knockout is a **JavaScript MVVM library** that makes it easier to **create rich, desktop-like user interfaces** with JavaScript and HTML. It uses observers to **make your UI automatically stay in sync** with an underlying data model, along with a powerful and extensible set of declarative bindings to **enable productive development.**”

http://knockoutjs.com

Knockout key features

Declarative Bindings

```
<div data-bind="text: message"></div>
<script>
  var viewModel = {
    message: 'Welcome'
  };

  ko.applyBindings(viewModel);
</script>
```

Automatic UI Refresh - Two-way data-binding

```
<div data-bind="text: message"></div>
<script>
  var viewModel = {
    message: ko.observable('Welcome')
  };

  ko.applyBindings(viewModel);
</script>
```

Knockout key features

Dependency Tracking

```
<input data-bind="value: firstName"/>
<input data-bind="value: lastName"/>
<div data-bind="text: fullName"></div>

<script>
  var vm = {
    firstName: ko.observable("Hello"),
    lastName: ko.observable("World")
  };

  vm = ko.computed(function() {
    return vm.firstName() + " " + vm.lastName();
  });

  ko.applyBindings(vm);
</script>
```

Knockout key features

[sf≡ir]

Templating

```
<ul data-bind="foreach: people">
  <li data-bind="text: name"></li>
</ul>

<script>
  var vm = {
    people: ko.observableArray([{
      name: 'John'
    }, {
      name: 'Bob'
    }, {
      name: 'Rebecca'
    }])
  };

  ko.applyBindings(vm)
</script>
```

Knockout key features

Component

```
<my-list></my-list>

<script>
  ko.components.register('my-list', {
      viewModel: { require: 'path/to/myList.js' },
      template: { require: 'text!path/to/myList.html' }
  });
</script>
```

```
// myList.js
define(['knockout'], function(ko) {
  var vm = {
    people: ko.observableArray([
      {
        name: 'John'
      },
      ...
    ]);
  };
  return vm;
});
```

```
<!-- myList.html -->
<ul data-bind="foreach: people">
  <li data-bind="text: name"></li>
</ul>
```

Durandal

“Durandal is a cross-device, cross-platform **client framework written in JavaScript** and designed to **make Single Page Applications** (SPAs) easy to create and maintain. It's built on jQuery, Knockout and RequireJS and **offers broad browser support** for SPA app development.”

DURANDAL

http://durandaljs.com/

Durandal key features

Modularization with Require

- Plain Old Java Object (POJO)

```
define({
  message: 'Hello, World!'
});
```

- Singleton

```
define(function() {
  var helloMessage = 'Hello, World!';
  return {
    message: helloMessage
  };
});
```

- Constructor

```
define(function() {
  var ctor = function(message) {
    this.message = message;
  };
  return ctor;
});
```

Durandal key features

Lifecycle

- Activation

```
define(function() {
  return {
    canDeactivate: function() {
      // some logic
      return true; // false
    },
    canActivate: function() {
      // some logic
      return true; // false
    },
    deactivate: function() {
      // some logic
      return true; // false or Promise
    },
    activate: function() {
      // some logic
      return true; // false or Promise
    }
  };
});
```

- Composition

```
define(function() {
  return {
    binding: function() {
      // some logic
      return true; // false
    },
    bindingComplete: function() {
      // some logic
    },
    attached: function() {
      // some logic
    },
    compositionComplete: function() {
      // some logic
    },
    detached: function() {
      // some logic
    }
  };
});
```

Durandal key features

- Routing
  - Route Parameters and Query Strings
  - Child Routers, Handling Unknown Routes
- Composition
  - Composing a POJO
  - Composing explicit Models and Views

Durandal + Knockout

Two-level architecture

- First level by Durandal with page and sub-page of application
- Second level with Knockout by creating a component library

Offline-first with Breeze

Breeze

“Breeze is a JavaScript library that helps you **manage data in rich client applications**. Breeze.js **communicates with any service** that speaks HTTP and JSON and runs natively on any JS client.”

http://www.getbreezenow.com/

Breeze key features

Query like LINQ

```
// Define query for customers, starting with 'A'
var query = breeze.EntityQuery
   .from("Customers")
   .where("CompanyName", "startsWith", "A")
   .orderBy(“CompanyName");
```

Async with promises on remote serve or from cache

```
// Execute query asynchronously on remote server or from cache
manager.fetchStrategy(breeze.FetchStrategy.FromLocalCache);
manager.fetchStrategy(breeze.FetchStrategy.FromServer);

// returns a promise ... with success/fail callbacks
var promise = manager.executeQuery(query)
   .then(querySucceeded)
   .fail(queryFailed);
```

Re-query from cache sync

```
// execute query synchronously on local cache
var customers = manager.executeQueryLocally(query);
```

Breeze key features

Knockout friendly

```
<!-- Knockout template -->
<ul data-bind="foreach: results">
  <li>
    <span data-bind="text:FirstName"></span>
    <span data-bind="text:LastName"></span>
  </li>
</ul>

<script>
  // bound to employees from query
  manager.executeQuery(breeze.EntityQuery.from("Employees"))
    .then(function(data) {
      ko.applyBindings(data);
    });
</script>
```

Breeze key features

[sf≡ir]

Change tracking

```
// save all changes (if there are any)
if (manager.hasChanges()) {
  manager.saveChanges()
     .then(saveSucceeded)
     .fail(saveFailed);
}
```

Save changes offline

```
var changes = manager.getChanges();
var exportData = manager.exportEntities(changes);

window.localStorage.setItem("changes", exportData);

// ... later ...

var importData = window.localStorage.getItem("changes");
manager.importEntities(importData);
```

Offline-first with Breeze

When application starts :

- Need a first query to get data
- Change FetchStrategy when connection change
- Export data to local storage when application quit
- Import data to cache when application start

When displaying data :

- Show cache data first
- Then refresh with remote data

Demo

Thank you

Questions ?