@DRPICOX

MVC - MODEL THE GREAT FORGOTTEN

TYPICAL ANGULAR CODE

AS SEEN ON SOME DOCS
(AND FROM SOME BEGINNERS)

TYPICAL ANGULAR CODE

<APP-BOOKS-LIST>

```
app.directive(‘appBooksList’, function($http) {
  return function link(scope) {
    $http.get(‘/api/v1/books’).then(function(response) {
      scope.books = response.data;
    });
  };
});
```

TYPICAL ANGULAR CODE

<APP-SCI-FI-BOOKS-LIST>

```
app.directive(‘appSciFiBooksList’, function($http) {
  return function link(scope) {
    $http.get(‘/api/v1/books’).then(function(response) {
      scope.books = response.data.filter(isSciFi);
    });
  };
});
```

REFACTORING - SERVICE

THERE IS REPLICATED CODE
¡LET'S DO A SERVICE!

REFACTORING - SERVICE

booksService

```
app.service(‘booksService’, function($http) {
  this.loadAll = function() {
    return $http.get(‘/api/v1/books’).then(function(response) {
      return response.data;
    });
  };
});
```

Acquires the responsibility for:

- R1: Know the API context Url
- R2: Create a remote connection
- R3: Transform response into usable object

REFACTORING - SERVICE

<APP-BOOKS-LIST> & <APP-SCI-FI-BOOKS-LIST>

```
app.directive(‘appBooksList’, function(booksService) {
  return function link(scope) {
    booksService.loadAll().then(function(books) {
      scope.books = books;
    });
  };
});
```

```
app.directive(‘appSciFiBooksList’, function(booksService) {
  return function link(scope) {
    booksService.loadAll().then(function(books) {
      scope.books = books;
    });
  };
});
```

PERFORMANCE - CACHING

OPS! WE ARE LOADING DATA TWICE
¡LET'S DO A CACHE!

PERFORMANCE - CACHING

booksService

```
app.service(‘booksService’, function($http) {
 this.loadAll = function() {
    return $http.get(‘/api/v1/books’, {cache:true}).then(function(response) {
      return response.data;
    });
 };
});
```

Notice that:

- Repairs all directives at once
- Acquires one new responsibility (R4: manage cache)

NEW FEATURE - ONE BOOK

WE HAVE A VIEW THAT ONLY SHOWS ONE BOOK

NEW FEATURE - ONE BOOK

<APP-BOOK-VIEW BOOK-ID=“ID”>

```
app.directive(‘appBookView’, function(booksService) {
  return function link(scope, element, attrs) {
    booksService.load(attrs.bookId).then(function(book) {
      scope.book = book;
    });
  };
});
```

NEW FEATURE - ONE BOOK

booksService (step1)

```
app.service('booksService', function($http) {
  this.loadAll = function() {
    return $http.get('/api/v1/books', {cache:true).then(function(response) {
      return response.data;
    });
  };
  this.load = function(id) {
    return $http.get('/api/v1/books/'+id).then(function(response) {
      return response.data;
    });
  };
});
```

Notice that:

- What about repeating loading of books (R4: manage cache)

NEW FEATURE - ONE BOOK

booksService (step2)

```
app.service('booksService', function($http) {
  this.loadAll = function() {
    return $http.get('/api/v1/books', {cache:true}).then(function(response) {
      return response.data;
    });
  };
  this.load = function(id) {
    return $http.get('/api/v1/books/'+id, {cache:true})
      .then(function(response) { return response.data; });
 };
});
```

- Uhmmmm… about R4… managing caches

NEW FEATURE - ONE BOOK

WE ARE LOADING TWICE THE SAME BOOK!

It happens when:

- Loading one book
- Loading all books

NEW FEATURE - ONE BOOK

OK... MAY BE IT'S OK LOAD TWICE
BUT NOT MORE TIMES
THE SAME BOOK

"We keep it simple..."

NEW FEATURE - UPDATE BOOK

WE HAVE A VIEW THAT MAKES AN UPDATE OF A BOOK AND SHOWS THE CHANGES INTRODUCED BY API

NEW FEATURE - UPDATE BOOK

<APP-BOOK-UPDATE BOOK-ID=“ID”>

```
app.directive(‘appBookUpdate’, function(booksService) {
  return function link(scope, element, attrs) {
    booksService.load(attrs.bookId).then(function(book) {
      scope.book = book;
    });
    scope.update = function() {
      booksService.update(scope.book).then(function(updatedBook) {
        scope.book = updatedBook;
      });
    };
  };
});
```

NEW FEATURE - ONE BOOK

booksService (step2)

```
app.service('booksService', function($http) {
  this.loadAll = function() {
    return $http.get('/api/v1/books', {cache:true}).then(function(response) {
      return response.data;
    });
  };
  this.load = function(id) {
    return $http.get('/api/v1/books/'+id, {cache:true})
      .then(function(response) { return response.data; });
  };
  this.update = function(book) {
    return $http.put('/api/v1/books/'+id, {cache:true})
      .then(function(response) { return response.data; });
  };
});
```

- Uhmmmm… about R4… managing caches

NEW FEATURE - UPDATE BOOK

SHIT!

NEW FEATURE - UPDATE BOOK

SHIT!

What about:

- Cache?
- What if we have a list side-by side, it is updated?
- What if we have view and update side-by-side?

NEW FEATURE - UPDATE BOOK

SHIT!

How to solve it?

- New services?
- Callbacks?
- RxJS?
- More caches?
- Events?

NEW FEATURE - UPDATE BOOK

WHAT IS THE REAL PROBLEM?

NEW FEATURE - UPDATE BOOK

WHAT IS THE REAL PROBLEM?

R4 (manage caches) is hidden another responsibility:

- Single source of true

We need a model!

WHAT A MODEL IS NOT?

- A model is not styles in the web
- A model is not objects in the scope
- A model is not objects stored in the browser (local,websql,…)
- A model are not instances of any sort of data
- A model is not plain data

WHAT IS A MODEL?

SINGLE SOURCE OF TRUE

- You can find your model
- There is only one instance for each “concept”
  (all books with the same id are the same object)
- Models may be computed from other models
- It does not matters how many views do you have,
  all views use the same models
  (as do services also)

That is something that ember programmers knows very well.

REFACTOR - ADD MODEL

Book

```
app.value(‘Book’, function Book(id) {
  this.id = id;
  this.isSciFi = function() { return !!this.tags.sciFi; };
  this.take = function(info) { angular.extend(this, info); };
});
```

We discover new responsibilities:

- R5: update and extract information of a book

REFACTOR - ADD MODEL

bookDictionary

```
app.value(‘bookDictionary’, function(Book) {
  this.list = []; this.map = {};
  this.getOrCreate() = function(id) {
    var book = this.map[id];
    if (!book) {
      book = new Book(id);
      this.list.push(book); this.list.map[id] = book;
    };
    return book;
  };
  this.takeAll = function(bookInfos) {
    bookInfos.forEach(function(bookInfo) {
      this.getOrCreate(bookInfo.id).take(bookInfo);
    }, this);
  };
});
```

Note that list

is also a model.

We discover new responsibilities:

- R6: manage book instances to avoid replicates
- R7: store books so anyone can access to them

REFACTOR - ADD MODEL

booksService

```
app.service(‘booksService’, function($http) {
  this.loadAll = function() {
    return $http.get(‘/api/v1/books’, {cache:true}).then(function(response) {
      return response.data;
    });
  };
  this.load = function(id) {
    return $http.get(‘/api/v1/books/’+id, {cache:true})
      .then(function(response) { return response.data; });
  };
  this.update = function(book) {
    return $http.put(‘/api/v1/books/’+id, {cache:true})
      .then(function(response) { return response.data; });
  };
});
```

- Uhmmmm… it has caches…
- Uhmmmm… it must interact with models
- Uhmmmm… it must speak api

Too many responsibilities!

REFACTOR - ADD MODEL

booksRemote

```
app.service(‘booksRemote’, function($http) {
  this.retrieveAll = function() {
    return $http.get(‘/api/v1/books’,{cache:true}).then(_getData);
  };
  this.retrieve = function(id) {
    return $http.get(‘/api/v1/books/’+id,{cache:true}).then(_getData);
  };
  this.store = function(book) {
    return $http.put(‘/api/v1/books/’+id,{cache:true}).then(_getData);
  };
  function _getData(response) { return result.data; }
});
```

Acquires the responsibility for:

- R1: Know the API context Url
- R2: Create a remote connection
- R3: Transform response into usable object
- R4: Manage cache (¿when data must be refreshed?)

REFACTOR - ADD MODEL

booksService (loadAll)

```
app.service(‘booksService’, function(booksDictionary, booksRemote) {
 this.loadAll = function() {
    return booksRemote.retrieveAll().then(function(books) {
      booksDictionary.takeAll(books);
      return booksDictionary.list;
    });
 };
  …
});
```

LoadAll returns always the same instance of list.

REFACTOR - ADD MODEL

booksService (load)

```
app.service(‘booksService’, function(booksDictionary, booksRemote) {
  …
  this.load = function(id) {
    return booksRemote.retrieve(id).then(function(bookInfo) {
      var book = booksDictionary.getOrCreate(id);
      book.take(bookInfo);
      return book;
    });
  };
  …
});
```

Load(id) returns always the same instance of book for the same id.

And the instance will be contained by the list in loadAll.

REFACTOR - ADD MODEL

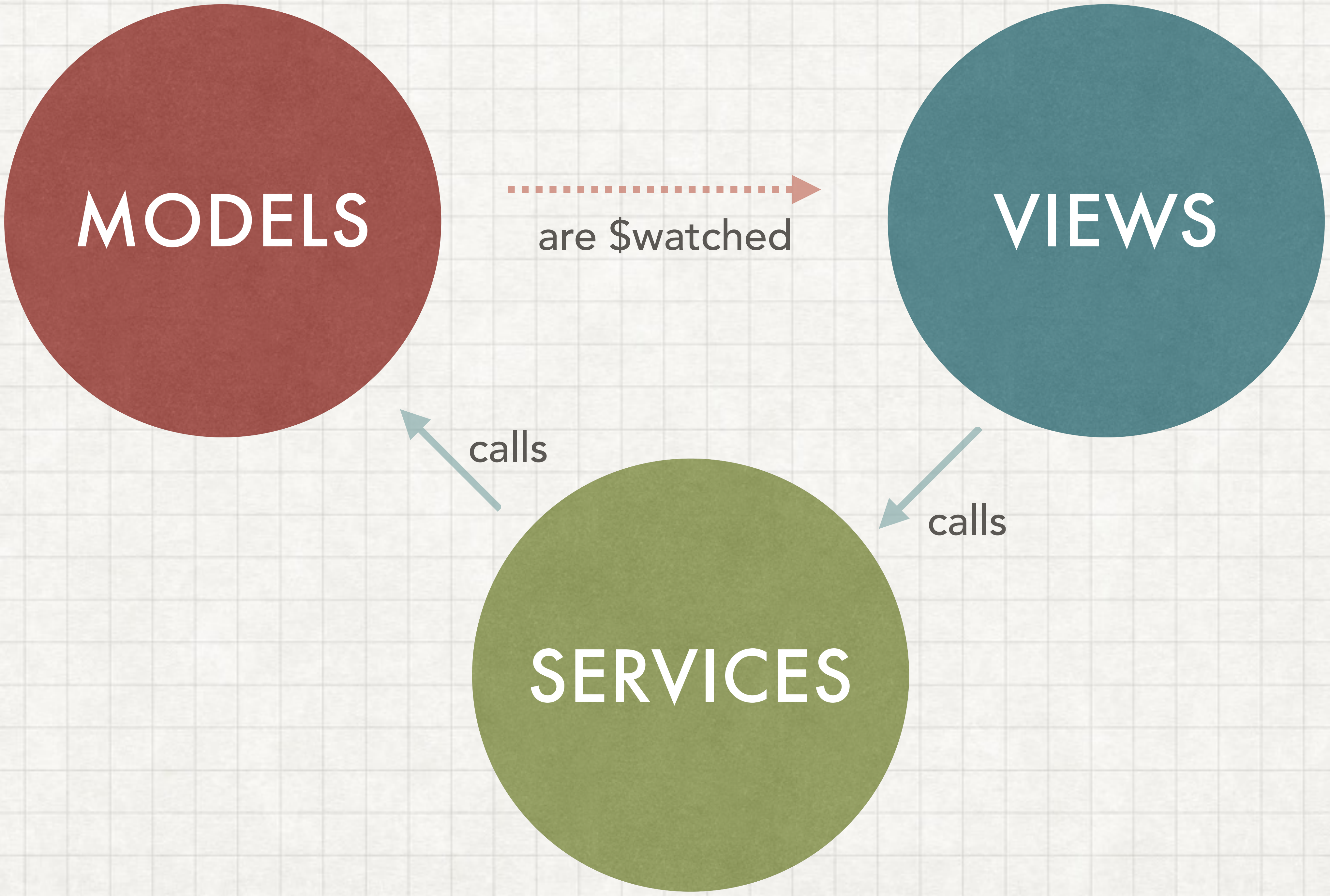
booksService (update)

```
app.service(‘booksService’, function(booksDictionary, booksRemote) {
  …
  this.update = function(book) {
    return booksRemote.update(book).then(function(bookInfo) {
      var book = booksDictionary.getOrCreate(id);
      book.take(bookInfo);
      return book;
    });
  };
});
```

Update returns always the same instance than load and loadAll, regardless the input. And also updates the same instance.

REFACTOR - ADD MODEL

three layers



NEW FEATURE - UPDATE BOOK

DONE!

NEW FEATURE - UPDATE BOOK

DONE?

What about Sci-Fi list?

- Is computed

  (it has to be recomputed each time that a book o list changes)
- A filter is expensive
- Watch a derive function is expensive
- Watch all books (aka $watch(,,true)) is expensive

NEW FEATURE - UPDATE BOOK

SHIT!

Solved in next talk!

DISCLAIMER

THINGS WRONG THAT I DID TO FIT THINGS IN SLIDES

- Directives should use controller and bindToController
- Directives should never load resources, this should be done by routes or under demand
- I usually like to use .factory to simplify and homogenize code (over .service neither .value)
- In a previous talk I said that model was plain JS data, sorry (plain data is the form, but does not defines a model)
- Use better variable names