

Angular or Backbone: Go Mobile!

Part 1 Angular vs Backbone

Doris Chen Ph.D.
Senior Developer Evangelist
Microsoft
doris.chen@microsoft.com
@doristchen



Doris Chen, Ph.D.

- Developer Evangelist at Microsoft based in Silicon Valley, CA:
 - Video: <https://channel9.msdn.com/Niners/dorischen>
 - Blog: <http://blogs.msdn.com/b/dorischen/>
 - Twitter @doristchen
 - Email: doris.chen@microsoft.com
- Over 18 years of experience in the software industry focusing on web technologies
- Spoke and published widely at O'Reilly OSCON, Fluent, HTML5 Dev Conf, JavaOne, Google Developer Conference, Silicon Valley Code Camp and worldwide User Groups meetups
- Doris received her Ph.D. from the University of California at Los Angeles (UCLA)

Agenda

1. Angular vs Backbone (Part 1)
2. Demo: ToDo App in Angular and Backbone (Part 2)
3. Mobile Apps for JavaScript (Part 3)
Demo: ToDo App with Apache Cordova

Angular vs Backbone

Angular vs Backbone: common

- Client-side framework using the MV* design pattern
- Solve the problem of Single Page Web Applications
 - Structure
 - Modular
 - Support multiple clients
- Both open-sourced, under the permissive MIT license
- Have the concept of views, events, data models and routing

History



Angular

- Born in 2009 as a part of commercial product, called GetAngular
- Misko Hevery, one of founders, turn a web application
 - 17,000 lines in 6 months to 1,000 lines in 3 weeks using just GetAngular
 - Google starts sponsoring, becomes open-source Angular.js



Backbone

- Born in 2010, lightweight MVC framework/library
- As a lean alternative to heavy, full-featured MVC (MVVC) frameworks such as ExtJS

Learning curve and Flexibility



Angular



- Looks quite easy at first sight
 - After the very basics it is quite a steep learning curve from there
- Highly opinionated
 - Fighting the framework if you don't like the way it does certain things
 - Reading the documentation is not easy as a lot of Angular specific jargon
- Lack of examples





Backbone

- Basic of backbone is very easy to learn
- Not opinionated
 - most flexible framework with the less conventions and opinions
 - Need to make a lot of decisions when using Backbone
- Need to watch or read a few tutorials to learn some best Backbone practices
- probably need to learn another library on top of Backbone (e.g. Marionette or Thorax)

Community

Metric	 Angular	 Backbone
Stars on GitHub	29.8k	19.3k
Third-Party Modules	800 ngmodules	236 backplugs
StackOverflow Questions	49.5k	15.9k
YouTube Results	~75k	~16k
GitHub Contributors	928	230
Chrome Extension Users	150k	7k

	 Angular	 Backbone
Architecture	MV* (Model View Whatever)	MV + VC
Template Support	YES. doesn't require any other separate template engine.(Also doesn't support template engine)	Uses underscore.js(an embedded template engine which allows logic inside template code)
File Size	39.5K (no dependencies)	6.5K - <i>43.5K (w/underscore & jQuery)</i>
Nested Template Support	Yes	No
Data Binding	Yes	No
Routing	Yes	Yes
Compatible with other frameworks	Yes	Yes
Additional Features	Dependency Injection, Directives, Watch Expressions and 2-way Binding, HTML5 validations, Form Validations, and Filtering	

Angular: templates



- Simply HTML with binding expressions baked-in
- Binding expressions are surrounded by double curly braces

```
<ul>
  <li ng-repeat="framework in frameworks"
      title="{{framework.description}}">
    {{framework.name}}
  </li>
</ul>
```

Backbone: templates



Underscore is very basic and you usually have to throw JavaScript into the mix

```
<ul>
  <% _.each(frameworks, function(framework) { %>
    <li title="<%- framework.description %>">
      <%- framework.name %>
    </li>
  <% }); %>
</ul>
```

Angular: model



- Angular does not use observable properties, no restriction on implementing model
- No class to extend and no interface to comply
- Free to use whatever you want (including existing Backbone models)
- In practice, most developers use plain old JavaScript objects (POJO)

Backbone: model



```
app.TodoModel = Backbone.Model.extend({
  defaults: {
    title: '',
    done: false,
    location: 'Getting your location...'
  },

  toggleCompleted: function () {
    this.save({
      done: !this.get('done')
    });
  },

  sync: function () { return false; }
});
```

Backbone: model and collection



```
var TodoCollection = Backbone.Collection.extend({  
    model: app.TodoModel,  
});  
app.todoCollection = new TodoCollection;
```

- Backbone is built around observable properties, **forced** to extend Backbone.Model and Backbone.Collection
- Backbone does not support observing functions, property needs to **reset** when any change happens

Angular: good things



- **Two-way data bindings, dependency injection**, easy-to-test code, extending the HTML dialect by using **directives**, out of the box **filters, reusable services** and **factories**
- Minimizes drastically the boilerplate code, no direct DOM manipulation
- The automatic **Dirty Checking**
 - No need getters and setters
 - modify property and angular automatically detects the change and notify all the watchers

2-way bindings and directives



```
<div class="templateWrapper" ng-repeat="todoItem in todos">
  <div class="templateContainer">
    <input class="templateTitle"
      ng-class="{crossedOut: todoItem.done}" type="text"
      ng-text-change="changeToDoText(todoItem)"
      ng-model="todoItem.text" />
    <h3 class="templateAddress">{{todoItem.address}}</h3>
  </div>
</div>
```


Angular: more good things



- Building application blocks into: **Controllers, Directives, Factories, Filters, Services and Views (templates)**.
 - **Views** UI, **Controllers** logic behind UI, **Services** communication with backend and common functionality, **Directives** reusable components and extending HTML by defining new elements, attributes and behaviors
- **Promises** play a main role in Angular

Custom directives



```
<input class="templateTitle"
      ng-class="{crossedOut: toItem.done}" type="text"
      ng-text-change=" changeToDoText(toDoItem)"
      ng-model="toDoItem.text" />
```

```
angular.module('xPlat.directives')
  .directive('ngTextChange', function () {
    return {
      restrict: 'A',
      replace: 'ngModel',
      link: function (scope, element, attr) {
        element.on('change', function () {
          scope.$apply(function () {
            scope.$eval(attr.ngTextChange);
          });
        });
      }
    };
  });
```

Custom directives, continued



```
scope.changeToDoText = function (todoItem) {  
    //Notice .then Promise pattern is used here  
    getAddress().then(function (address) {  
        todoItem.address = address;  
        return storage.update(todoItem);  
    }, function (errorMessage) {  
        todoItem.address = errorMessage;  
        return storage.update(todoItem);  
    });  
}
```

Angular: bad things



- Extremely **opinionated**
 - Frustrated: prior experience in creating UI with Javascript is rendered almost useless
- Do everything according to the "Angular way"
- Directives could be super complicated and odd
- Expression language too powerful
 - `<button ng-click="(oldPassword && checkComplexity(newPassword) && oldPassword != newPassword) ? (changePassword(oldPassword, newPassword) && (oldPassword=(newPassword='')) : (errorMessage='Please input a new password matching the following requirements: ' + passwordRequirements))">Click me</button>`
- Scope could be confusing
- Need more built-in support

Backbone: good things



- Compact, minimal, **not opinionated**
- Resembles classic Javascript the most
- Has the least learning curve
- Huge community (ecosystem) and lots of solutions on StackOverflow
- Many popular applications
 - Twitter, Foursquare, LinkedIn Mobile, Soundcloud, Pitchfork, Pandora, Pinterest, Flixster, AirBNB

Backbone: bad things



- Hands off: Need to develop own
 - Application Architecture / Layout Structure / Memory management/
- Too much option of bringing third party plugins
 - Cost time to do research, it's not so minimal and becomes opinionated
- Lacking features compared to the newer frameworks
 - **No data binding**
 - a lot of boiler plate code to make it work well
 - doesn't allow easy iteration through UI variants
 - **No nested model**
 - `userModel.get("name").first = "Tom"; userModel.set("name.first","Tom");` -- can't do
- Views manipulate the DOM directly, making them really hard to unit-test, more fragile and less reusable.

Backbone vs Angular

```
updateCrossOut: function () {  
    if (this.model.get('done')) {  
        this.$input.addClass('crossedOut');  
        ...  
    }  
    else {  
        this.$input.removeClass('crossedOut');  
        ...  
    }  
},
```

```
<input class="templateTitle"  
ng-class="{crossedOut: todoItem.done}"... ng-model="todoItem.text" />
```

