v1.5.6-build.4831 (snapshot

/ Tutorial (tutorial)
/ 6 - Two-way Data Binding (tutorial/step_06)

content/tutorial/step_06.ngdoc?message=docs(tutorial%2F6 - Two-way Data Binding)%3A%20describe%20your%20change...)

Previous (tutorial/step_05) ▶ Live Demo (http://angular.github.io/angular-phonecat/step-6/app) Code Diff (https://github.com/angular/angular-phonecat/compare/step-5...step-6)
Next (tutorial/step_07)

In this step, we will add a feature to let our users control the order of the items in the phone list. The dynamic ordering is implemented by creating a new model property, wiring it together with the repeater, and letting the data binding magic do the rest of the work.

- In addition to the search box, the application displays a drop-down menu that allows users to control the order in which the phones are listed.

Workspace Reset Instructions ➤

The most important changes are listed below. You can see the full diff on GitHub (https://github.com/angular/angular-phonecat/compare/step-5...step-6).

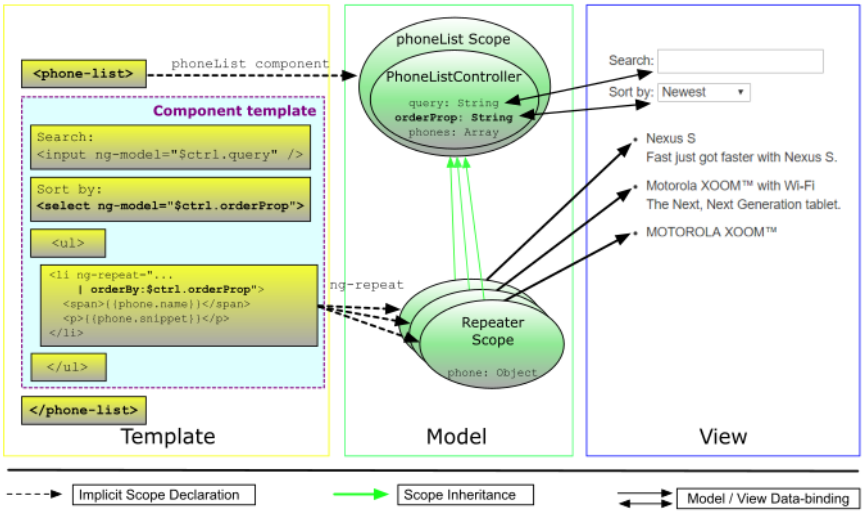
Component Template

app/phone-list/phone-list.template.html :

```
<div class="container-fluid">
  <div class="row">
    <div class="col-md-2">
      <!--Sidebar content-->

      <p>
        Search:
        <input ng-model="$ctrl.query">
      </p>

      <p>
        Sort by:
        <select ng-model="$ctrl.orderProp">
          <option value="name">Alphabetical</option>
          <option value="age">Newest</option>
        </select>
      </p>

    </div>
    <div class="col-md-10">
      <!--Body content-->

      <ul class="phones">
        <li ng-repeat="phone in $ctrl.phones |
filter:$ctrl.query | orderBy:$ctrl.orderProp">
          <span>{{phone.name}}</span>
          <p>{{phone.snippet}}</p>
        </li>
      </ul>

    </div>
  </div>
</div>
```

We made the following changes to the `phone-list.template.html` template:

- First, we added a `<select>` element bound to `$ctrl.orderProp`, so that our users can pick from the two provided sorting options.



- We then chained the `filter` filter with the orderBy (api/ng/filter/orderBy) filter to further process the input for the repeater. `orderBy` is a filter that takes an input array, copies it and reorders the copy which is then returned.

  Angular creates a two way data-binding between the select element and the `$ctrl.orderProp` model. `$ctrl.orderProp` is then used as the input for the `orderBy` filter.

As we discussed in the section about data-binding and the repeater in step 5 (tutorial/step_05), whenever the model changes (for example because a user changes the order with the select drop-down menu), Angular's data-binding will cause the view to automatically update. No bloated DOM manipulation code is necessary!

Component Controller

app/phone-list/phone-list.components.js :

```
angular.
  module('phoneList').
  component('phoneList', {
    templateUrl: 'phone-list/phone-list.template.html',
    controller: function PhoneListController() {
      this.phones = [
        {
          name: 'Nexus S',
          snippet: 'Fast just got faster with Nexus S.',
          age: 1
        }, {
          name: 'Motorola XOOM™ with Wi-Fi',
          snippet: 'The Next, Next Generation tablet.',
          age: 2
        }, {
          name: 'MOTOROLA XOOM™',
          snippet: 'The Next, Next Generation tablet.',
          age: 3
        }
      ];

      this.orderProp = 'age';
    }
  });
```

- We modified the `phones` model - the array of phones - and added an `age` property to each phone record. This property is used to order the phones by age.

- We added a line to the controller that sets the default value of `orderProp` to `age`. If we had not set a default value here, the `orderBy` filter would remain uninitialized until the user picked an option from the drop-down menu.

This is a good time to talk about two-way data-binding. Notice that when the application is loaded in the browser, "Newest" is selected in the drop-down menu. This is because we set `orderProp` to `'age'` in the controller. So the binding works in the direction from our model to the UI. Now if you select "Alphabetically" in the drop-down menu, the model will be updated as well and the phones will be reordered. That is the data-binding doing its job in the opposite direction — from the UI to the model.

Testing

The changes we made should be verified with both a unit test and an E2E test. Let's look at the unit test first.

`app/phone-list/phone-list.component.spec.js`:

```
describe('phoneList', function() {

  // Load the module that contains the `phoneList` component
before each test
  beforeEach(module('phoneList'));

  // Test the controller
  describe('PhoneListController', function() {
    var ctrl;

    beforeEach(inject(function($componentController) {
      ctrl = $componentController('phoneList');
    }));

    it('should create a `phones` model with 3 phones',
function() {
      expect(ctrl.phones.length).toBe(3);
    });

    it('should set a default value for the `orderProp`
model', function() {
      expect(ctrl.orderProp).toBe('age');
    });

  });

});
```

The unit test now verifies that the default ordering property is set.

We used Jasmine's API to extract the controller construction into a `beforeEach` block, which is shared by all tests in the parent `describe` block.

You should now see the following output in the Karma tab:

```
Chrome 49.0: Executed 2 of 2 SUCCESS (0.136 secs / 0.08
secs)
```

Let's turn our attention to the E2E tests.

e2e-tests/scenarios.js :

```
describe('PhoneCat Application', function() {

  describe('phoneList', function() {

    ...

    it('should be possible to control phone order via the
drop-down menu', function() {
      var queryField = element(by.model('$ctrl.query'));
      var orderSelect =
element(by.model('$ctrl.orderProp'));
      var nameOption =
orderSelect.element(by.css('option[value="name"]'));
      var phoneNameColumn = element.all(by.repeater('phone
in $ctrl.phones').column('phone.name'));

      function getNames() {
        return phoneNameColumn.map(function(elem) {
          return elem.getText();
        });
      }

      queryField.sendKeys('tablet');   // Let's narrow the
dataset to make the assertions shorter

      expect(getNames()).toEqual([
        'Motorola XOOM\u2122 with Wi-Fi',
        'MOTOROLA XOOM\u2122'
      ]);

      nameOption.click();

      expect(getNames()).toEqual([
        'MOTOROLA XOOM\u2122',
        'Motorola XOOM\u2122 with Wi-Fi'
      ]);
```

```
});

...
```

The E2E test verifies that the ordering mechanism of the select box is working correctly.

You can now rerun `npm run protractor` to see the tests run.

Experiments

- In the `phoneList` component's controller, remove the statement that sets the `orderProp` value and you'll see that Angular will temporarily add a new blank ("unknown") option to the drop-down list and the ordering will default to unordered/natural order.
- Add a `{{$ctrl.orderProp}}` binding into the `phone-list.template.html` template to display its current value as text.
- Reverse the sort order by adding a `-` symbol before the sorting value: `<option value="-age">Oldest</option>`

Summary

Now that you have added list sorting and tested the application, go to step 7 (tutorial/step_07) to learn about Angular services and how Angular uses dependency injection.

Previous (tutorial/step_05) **Live Demo** (http://angular.github.io/angular-phonecat/step-6/app) **Code Diff** (https://github.com/angular/angular-phonecat/compare/step-5...step-6)