In this step, you will add a feature to let your 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 app 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

Template

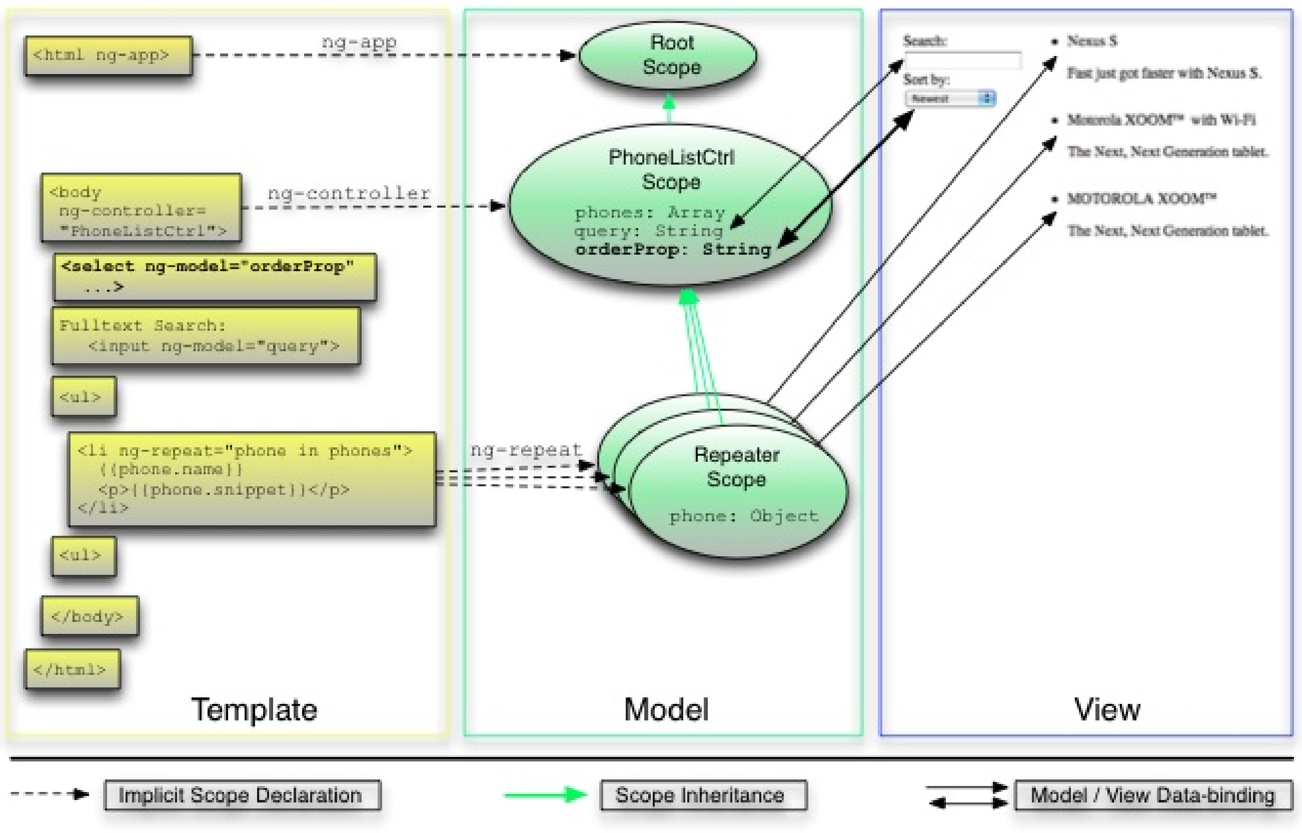
`app/index.html` :

```
Search: <input ng-model="query">
Sort by:
<select ng-model="orderProp">
  <option value="name">Alphabetical</option>
  <option value="age">Newest</option>
</select>


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

We made the following changes to the `index.html` template:

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



<html ng-app>
ng-app
Root Scope
<body ng-controller="PhoneListCtrl">
ng-controller
PhoneListCtrl Scope
phones: Array
query: String
orderProp: String
<select ng-model="orderProp" ...>
Fulltext Search: <input ng-model="query">
<ul>
<li ng-repeat="phone in phones"> {{phone.name}} <p>{{phone.snippet}}</p> </li>
ng-repeat
Repeater Scope
phone: Object
<ul>
</body>
</html>
Search:
Sort by:
Newest
Nexus S
Fast just got faster with Nexus S.
Motorola XOOM™ with Wi-Fi
The Next, Next Generation tablet.
MOTOROLA XOOM™
The Next, Next Generation tablet.
Template
Model
View
Implicit Scope Declaration
Scope Inheritance
Model / View Data-binding

- We then chained the `filter` filter with `orderBy` filter to further process the input into 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 `orderProp` model. `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 3, 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!

Controller

app/js/controllers.js :

```
var phonecatApp = angular.module('phonecatApp', []);

phonecatApp.controller('PhoneListCtrl', function ($scope) {
  $scope.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}
  ];

  $scope.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 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 our 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 app 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.

Test

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

test/unit/controllersSpec.js :

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

describe('PhoneListCtrl', function(){
  var scope, ctrl;

  beforeEach(module('phonecatApp'));

  beforeEach(inject(function($controller) {
    scope = {};
    ctrl = $controller('PhoneListCtrl', {$scope:scope});
  }));

  it('should create "phones" model with 3 phones', function() {
    expect(scope.phones.length).toBe(3);
  });


  it('should set the default value of orderProp model', function() {
    expect(scope.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 22.0: Executed 2 of 2 SUCCESS (0.021 secs / 0.001 secs)
```

Let's turn our attention to the end-to-end test.

`test/e2e/scenarios.js` :

```
...
it('should be possible to control phone order via the drop down select box', function() {

  var phoneNameColumn = element.all(by.repeater('phone in phones').column('{{phone.name}}'));
  var query = element(by.model('query'));

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

  query.sendKeys('tablet'); //let's narrow the dataset to make the test assertions shorter

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

  element(by.model('orderProp')).element(by.css('option[value="name"]')).click();

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

The end-to-end 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 `PhoneListCtrl` controller, remove the statement that sets the `orderProp` value and you'll see that Angular will temporarily add a new "unknown" option to the drop-down list and the ordering will default to unordered/natural order.

Add an `{{orderProp}}` binding into the `index.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 app, go to step 5 to learn about Angular services and how Angular uses dependency injection.

Previous | Live Demo | Code Diff | Next