Insight? Practice!

Road to myself. 자기자신에게로 이르는 길.

- RSS

Search

Navigate...

- Blog
- Archives
- About

[번역] 하루만에 끝내는 AngularJS

Jan 20th, 2014

이글은 Todd라는 개발자가 작성한 "Ultimate guide to learning angular js in one day"라는 블로그 글을 번역한 것이다.

정말 하루만에 끝낼 순 없지만 -_- 그래도 좋은 시작점이 되리라 생각한다. 다같이 AngularJS에 빠져보자!

AngularJS 란?

Angular는 자바스크립트로 만든 client 측 MVC/MVVM 프레임워크로 모던 단일 페이지 웹 애플리케이션(혹은 웹사이트) 개발의 정수이다. 그리고 모던 웹을 다루는 새로운 방법이자 HTML5가 가져올 미래를 위한 큰 발판이다. 이 글은 필자가 Angular를 경험해보며 알게 된 것들과 조언, 우수 사례를 바탕으로 작성해본 이름하여 하루만에 끝내는 Angular 강좌다.

용어

Angular는 새로 등장하는 용어와 "MVC로 생각"하는 방법으로 인해 약간의 러닝커브가 있다. 여기서 MVC는 모델-뷰-컨트롤러 를 의미한다. 이제부터 Angular를 구성하는 필수적인 API와 용어를 살펴보자.

MVC

분명 MVC는 들어봤을 것이다. 프로그래밍 언어에서 애플리케이션/소프트웨어의 구조를 잡거나 아키텍처를 정하는 방법으로 많이 사용 되고 있다. 일단 간단히 의미를 살펴보자:

- 모델 : 보통 JSON으로 표현되는 애플리케이션의 특정한 데이터 구조를 말한다. 뷰가 서버와 통신하기 위해 꼭 필요한 내용이니 더 진행하기 전에 다음 JSON을 잘 살펴보자. 예를 들어 *User ID* 그룹은 다음과 같은 모델을 가질 수 있다:

```
{
  "users" : [{
    "name": "Joe Bloggs",
    "id": "82047392"
  },{
    "name": "John Doe",
    "id": "65198013"
  }]
}
```

이 정보를 jQuery의 *$.ajax* 메서드를 래핑한 Angular의 *$http* 를 통해 XHR(XMLHttp Request)로 서버에서 가져오거나 페이지를 로딩할 때 코드에서 직접 (데이터베이스에서) 읽어오도록 할 수 있다. 그리고 모델을 변경한 다음 다시 반영할 수도 있다.

- 뷰 : 뷰는 간단하다. HTML 혹은 렌더링된 결과를 말한다. MVC 프레임워크를 사용한다면 뷰를 갱신할 모델 데이터를 내려받은 뒤 HTML에서 해당 데이터를 보여줄 것이다.
- 컨트롤러 : 말 그대로 한번 생각해보자. 무언가를 조정한다. 근데 무엇을 조정할까? 데이터다. 컨트롤러는 서버 에서 직접 뷰 로 접근하는 일종의 중간 통로로서 필요할 때마다 서버와 클라이언트 통신으로 데이터를 변경한다.

AngularJS 프로젝트 설정 (아주 기초)

이제 실제 AngularJS 프로젝트를 만들어보자. 시작하기 전에 *ng-app* 선언으로 앱을 정의하는 부분이라든가 뷰와 통신하는 컨트롤러 또는 Angular에 내재된 DOM 바인딩 등 몇 가지 살펴볼 게 있다. 이제부터는 아주 기초적인 부분이다:

다음은 *ng-** 선언을 추가한 HTML이다:

```
<div ng-app="myApp">
    <div ng-controller="MainCtrl">
        <!-- controller logic -->
    </div>
</div>
```

그리고 Angular 모듈과 컨트롤러다:

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

myApp.controller('MainCtrl', ['$scope', function ($scope) {
  // Controller magic
}]);
```

더 진행하기 전에 모든 로직을 담을 *Angular* 모듈 을 하나 만들어보자. 모듈을 정의하는 방법은 다양하며 그 중 하나가 다음과 같이 로직을 묶는 방법이다(필자는 이런 방식을 별로 안좋아한다):

```
angular.module('myApp', [])
.controller('MainCtrl', ['$scope', function ($scope) {...}])
.controller('NavCtrl', ['$scope', function ($scope) {...}])
.controller('UserCtrl', ['$scope', function ($scope) {...}]);
```

필자가 해봤던 Angular 프로젝트를 생각해보면 전역 모듈을 만드는게 가장 좋은 방법이다. 하지만 이렇게 세미콜론을 쓰지않고 함수 체인을 갑자기 끊으면 불필요한 컴파일 에러를 만들어내기 때문에 비효율적이다. 다음 코드를 보자:

```
var myApp = angular.module('myApp', []);
myApp.controller('MainCtrl', ['$scope', function ($scope) {...}]);
myApp.controller('NavCtrl', ['$scope', function ($scope) {...}]);
```

```
myApp.controller('UserCtrl', ['$scope', function ($scope) {...}]);
```

새로 만드는 파일마다 *myApp* 을 네임스페이스처럼 사용할 수 있으니 바로 애플리케이션에 집중할 수 있을 것이다. 그렇다. 각 컨트롤러, 디렉티브, 팩토리 등 모든 것에 대해서 각각 새로운 파일을 만들 것이다 (이부분은 나에게 고마워해야 할 것임). 그리고 이들을 모두 엮어서 Grunt같은 실행기로 하나의 스크립트 파일을 DOM에 적용한다.

컨트롤러

이제 MVC와 기본적인 설정을 살펴봤으니 Angular의 컨트롤러를 어떻게 사용하는지 살펴보자.

다음 예제를 진행하기 전에 컨트롤러를 통해서 데이터를 DOM에 주입하는 아주 쉬운 단계부터 살펴보자. Angular는 HTML과 통신하기 위해 *{{ handlebars }}* 와 같은 템플릿 형식의 문법을 사용한다. HTML에 데이터를 하드코딩하지 않아야지만(이상적으로는) Angular를 제대로 사용하는 것이다. 다음은 DOM에 간단한 문자열을 넣는 예제다:

```
<div ng-app="myApp">
    <div ng-controller="MainCtrl">
        {{ text }}
    </div>
</div>

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

myApp.controller('MainCtrl', ['$scope', function ($scope) {

    $scope.text = 'Hello, Angular fanatic.';

}]);
```

다음은 실행한 결과다:

Result JavaScript HTML Edit in JSFiddle

Hello, Angular fanatic.

여기서 가장 중요한 개념은 특정 컨트롤러안에 모든 기능을 담는 *$scope* 라는 개념이다. *$scope* 는 DOM의 현재 요소/영역을 참조하며(*this* 와는 다르다), 요소안의 데이터와 로직을 주시하는 아주 멋진 관찰 기능을 가지고 있다. 이 기능으로 DOM에 자바스크립트 public/private 스코프를 멋지게 만들 수 있다.

$scope 개념이 처음에는 조금 이상해 보일지 몰라도 서버로부터 DOM을 만드는 아주 좋은 방법이다(정

적 데이터인 경우도 역시)! 예제를 보면 DOM으로 데이터를 어떻게 주입하는지에 대한 기본 개념을 익힐 수 있을 것이다.

이제 사용자의 로그인 세부 내용을 보여주기 위해 서버에서 조금 더 자세한 데이터를 받아왔다고 가정해 보자. 지금은 정적 데이터이고 실제 JSON으로 받아오는 건 나중에 살펴보겠다.

먼저 자바스크립트를 설정한다:

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

myApp.controller('UserCtrl', ['$scope', function ($scope) {

    // user details라는 네임스페이스를 사용하자. DOM에서 알아보기도 좋을 것이다.
    $scope.user = {};
    $scope.user.details = {
      "username": "Todd Motto",
      "id": "89101112"
    };

}]);
```

그 다음 화면에 보여주기 위해 DOM에 데이터를 지정한다:

```
<div ng-app="myApp">
    <div ng-controller="UserCtrl">
        <p class="username">Welcome, {{ user.details.username }}</p>
        <p class="id">User ID: {{ user.details.id }}</p>
    </div>
</div>
```

결과:

Result JavaScript HTML Edit in JSFiddle

Welcome, Todd Motto

User ID: 89101112

컨트롤러는 JSON 데이터로 서버와 통신하는 함수(이벤트 함수도!)와 데이터 만을 다룬다는 걸 기억하는 게 중요하다. DOM 조작을 컨트롤러에서 해선 안되며 jQuery도 일단은 생각하지 말자. DOM 조작은 디렉티브로 하면 되니까 조금 있다 다시 살펴보자.

중요팁: Angular 문서를 보면(이 글을 쓰는 지금) 컨트롤러를 생성하는 방법을 다음과 같이 설명하고 있다:

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

```
function MainCtrl ($scope) {
  //...
};
```

... 이렇게는 하지 말자. 모든 함수가 전역 함수가 되버려서 앱 안에 집어넣기도 어렵다. 또한 코드를 압축하기도 어렵고 테스트를 쉽게 실행하기도 힘들다. 따라서 전역 네임스페이스는 생성하지 말고 컨트롤러는 앱 안에 꼭 집어넣자.

디렉티브

디렉티브(Directives from existing scripts/plugins 포스트를 참고)의 가장 간단한 형태는 애플리케이션이 필요한 곳에 여러 번 사용할 수 있는 작은 HTML 조각 형태다. 디렉티브를 사용하면 애플리케이션에 별다른 노력없이도 쉽게 DOM을 주입하거나 사용자 정의 DOM의 상호작용을 적용할 수 있다. 디렉티브는 간단하지 않을 뿐더러 러닝커브가 생각보다 꽤 높긴 하지만 다음 절부터 읽어보면 분명 도움이 될 것이다.

그래서 디렉티브가 어디에 유용한 걸까? DOM 컴포넌트를 포함해서 많은 부분에 유용하다. 앱에서 사용하는 UI에 따라 다르긴 하지만 탭과 네비게이션 요소 등에 특히 유용하다. 이런 식으로 설명해보겠다. *ng-show* 나 *ng-hide* 를 생각해본적이 있다면 그게 바로 디렉티브다(DOM을 주입하지는 않지만).

이번 예제에서는 아주 간단한 버튼(*customButton* 이라는 이름의)을 생성해서 필자가 일일히 직접 타이핑하기 싫어하는 마크업을 한번 주입해 보겠다. DOM에 디렉티브를 정의하는 다양한 방법이 있지만 내가 사용한 방법은 다음과 같다:

```
<!-- 1: 속성으로 정의 -->
<a custom-button>Click me</a>

<!-- 2: 요소로 정의 -->
<custom-button>Click me</custom-button>

<!-- 3: 클래스로 정의(IE 구버전 호환을 위해) -->
<a class="custom-button">Click me</a>

<!-- 4: 주석으로 정의 (데모로는 별로 안좋긴 하다) -->
<!-- directive: custom-button -->
```

사용자 정의 요소는 HTML5의 웹 컴포넌트로 추가될 예정이라서 필자는 디렉티브를 속성 형태로 사용하는 걸 선호하지만 오래된 특정 브라우저에서 오류를 낸다고 한다.

이제 디렉티브를 어떻게 사용하고 주입하는지 알아봤으니 사용자 정의 버튼을 생성해보자. 애플리케이션의 전역 변수인 *myApp* 을 사용해서 디렉티브를 선언하는 방법이다:

```
myApp.directive('customButton', function () {
  return {
    link: function (scope, element, attrs) {
      // DOM 조작과 이벤트 설정은 여기서!
    }
  };
});
```

.directive() 메서드로 디렉티브를 선언하고 디렉티브 이름으로 'customButton'을 사용했다. 디렉티브 이름에 대문자를 사용하면 DOM에서는 하이픈으로 이를 구분해서 사용하게 된다(위 예제처럼).

디렉티브는 여러개의 속성을 가지는 객체를 반환한다. 처음 배우는 입장에서 필자가 가장 중요하게 생각하는 건 *restrict*, *replace*, *transclude*, *template*, *templateUrl*, *link* 속성이다. 이 속성들을 추가해보

자:

```
myApp.directive('customButton', function () {
  return {
    restrict: 'A',
    replace: true,
    transclude: true,
    template: '<a href="" class="myawesomebutton" ng-transclude>' +
                '<i class="icon-ok-sign"></i>' +
              '</a>',
    link: function (scope, element, attrs) {
      // DOM 조작과 이벤트 설정은 여기서!
    }
  };
});
```

결과:

Result JavaScript HTML Edit in JSFiddle

Click me

브라우저의 요소 검사 로 마크업이 잘 주입됐는지 확인해보는 걸 잊지 말자. 아이콘도 없고 멋진 폰트도 사용안했지만 어떻게 동작하는지는 알 수 있을 것이다. 자 다음은 디렉티브의 각 속성에 대한 설명이다:

- *restrict*: 어떻게 요소의 사용을 제한할 수 있을지 다시 한번 생각해보자. 오래된 IE를 지원해야하는 프로젝트를 진행중이라면 분명 속성/클래스 정의가 필요할 것이다. 'A'라고 지정하면 속성 으로만 사용할 수 있다는 의미이고 'E'는 요소, 'C'는 클래스, 'M'은 주석 으로만 사용할 수 있다는 것을 의미한다. 기본 값은 'EA'이고 이 처럼 여러 개의 제한을 동시에 걸수도 있다.
- *replace*: 디렉티브에 정의한 DOM의 마크업을 변경할 수 있음을 의미한다. 예제를 보면 처음의 DOM이 디렉티브의 템플릿으로 어떻게 변경됐는지 알 수 있을 것이다.
- *transclude*: 간단하게 말해서 집어넣는 것이다. transclude를 이용하면 기존의 DOM 내용을 디렉티브안에 복사할 수 있다. 'Click me'라는 문자열이 렌더링될 때 디렉티브로 옮겨진 것을 봤을 것이다.
- *template*: 템플릿은 주입할 마크업을 의미한다. HTML의 아주 작은 일부분을 정의할 때 특히 좋다. 주입된 템플릿은 Angular로 컴파일되며 이로 인해 handlebar 템플릿 태그도 사용할 수 있다.
- *templateUrl*: template 속성과 비슷하지만 <script> 태그 혹은 파일을 지정할 때 사용한다. HTML의 일부분을 다른 파일로 관리할 필요가 있을 때 템플릿 파일의 URL로 파일 이름과 경로

(보통 *templates* 디렉토리)를 표시해주면 된다.

```
myApp.directive('customButton', function () {
  return {
    templateUrl: 'templates/customButton.html'
    // 나머지 디렉티브 내용...
  };
});
```

그리고 다음은 템플릿 파일 내용이다(이름은 중요하지 않음):

```
<!-- customButton.html 내용 -->
<a href="" class="myawesomebutton" ng-transclude>
  <i class="icon-ok-sign"></i>
</a>
```

이렇게 했을 때 정말 좋은 점은 브라우저가 HTML 파일을 캐싱 한다는 점이다. 브라보! 캐싱되는걸 원하지 않는다면 <script> 태그안에 템플릿을 선언하면 된다:

```
<script type="text/ng-template" id="customButton.html">
<a href="" class="myawesomebutton" ng-transclude>
  <i class="icon-ok-sign"></i>
</a>
</script>
```

이렇게 하면 Angular에게 이 ID로 *ng-template* 을 선언했다고 알려주게 된다. 그러면 Angular는 *ng-template* 혹은 **.html* 파일을 찾기 시작할 것이다. 필자는 **.html* 파일을 선호하는데, 쉽게 관리할 수 있고 성능도 잘 나오며 DOM도 깔끔하게 유지할 수 있기 때문이다. 최소한 1개 이상 혹은 100개가 넘는 디렉티브를 사용할 것일고 이 중에서 분명 원하는 걸 쉽게 찾고 싶지 않겠나.

서비스

서비스는 종종 헷갈리는 부분이다. 경험에 비춰보면 서비스는 기능적인 큰 차이점을 제공하지 않으면서도 뭔가 더 좋아보이는 디자인 패턴이다. Angular 소스를 분석해보니 Angular는 같은 컴파일러를 사용하면서 많은 기능을 제공하는듯 하다. 분석해보니 서비스는 싱글톤 으로 사용해야하고 객체 리터럴이나 좀 더 복잡한 유즈 케이스처럼 더 복잡한 기능은 팩토리를 사용해야 한다.

다음 예제는 2개의 숫자를 곱하는 서비스이다:

```
myApp.service('Math', function () {
  this.multiply = function (x, y) {
    return x * y;
  };
});
```

컨트롤러안에서 서비스를 다음처럼 사용할 수 있겠다:

```
myApp.controller('MainCtrl', ['$scope', function ($scope) {
    var a = 12;
    var b = 24;

    // 결과는 288
    var result = Math.multiply(a, b);
}]);
```

맞다. 곱셈은 엄청 쉬워서 서비스가 필요하지도 않지만 핵심은 알 수 있었을 것이다.

서비스(혹은 팩토리)를 생성할때는 의존성 주입을 사용해서 Angular에게 새로 만든 서비스의 존재를 알

려줘야 한다. 알려주지 않으면 컴파일 에러가 발생하거나 컨트롤러가 동작하지 않을 것이다. 컨트롤러 선언부분에 *function ($scope)* 를 봤을텐데 이게 바로 간단한 의존성 주입 방법이다. *function ($scope)* 앞에 있는 *['$scope']* 도 봤겠지만 이건 나중에 설명하겠다. 다음 예제는 의존성 주입을 통해 Angular에게 서비스가 필요하다고 알려주는 방법이다:

```
// Math를 주입한다
myApp.controller('MainCtrl', ['$scope', 'Math', function ($scope, Math) {
    var a = 12;
    var b = 24;

    // 결과는 288
    var result = Math.multiply(a, b);
}]);
```

팩토리

팩토리로 서비스를 만드는 건 이제 간단하다. 객체 리터럴을 팩토리안에서 생성하거나 다음처럼 몇 가지 메서드를 추가하면 된다:

```
myApp.factory('Server', ['$http', function ($http) {
  return {
    get: function(url) {
      return $http.get(url);
    },
    post: function(url) {
      return $http.post(url);
    },
  };
}]);
```

Angular의 XHR을 래핑한 코드를 작성해봤다. 컨트롤러에 의존성을 주입한 다음 이렇게 간단히 사용하면 된다:

```
myApp.controller('MainCtrl', ['$scope', 'Server', function ($scope, Server) {
    var jsonGet = 'http://myserver/getURL';
    var jsonPost = 'http://myserver/postURL';
    Server.get(jsonGet);
    Server.post(jsonPost);
}]);
```

혹시 서버 변경사항을 폴링하고 싶으면 *Server.poll(jsonPoll)* 을 설정하거나 *Server.socket(jsonSocket)* 을 사용할 수도 있겠다. 이렇게 컨트롤러에 서비스를 주입해서 사용하면 컨트롤러의 코드를 최소로 유지할 수 있다. 즉 나만의 도구를 만들어서 사용하는 것처럼 코드를 모듈화할 수 있는 길이 열리는 것이다.

필터

필터는 배열의 데이터와 함께 사용하며 루프 밖에서도 사용 할 수 있다. 데이터를 순회하면서 특정 조건에 만족하는 데이터만 추리고 싶을 때 필터를 사용하면 된다. 예를 들어 <input>에 입력된 값으로 사용자를 추리고 싶을 때처럼 말이다. 필터를 사용하는 방법은 컨트롤러 안에 선언하거나 메서드로 정의해서 사용해도 된다. 다음은 필터를 전역으로 선언한 방법이다:

```
myApp.filter('reverse', function () {
    return function (input, uppercase) {
        var out = '';
        for (var i = 0; i < input.length; i++) {
            out = input.charAt(i) + out;
        }
```

```
        if (uppercase) {
            out = out.toUpperCase();
        }
        return out;
    }
});

// 데이터를 제공하는 컨트롤러
myApp.controller('MainCtrl', ['$scope', function ($scope) {
    $scope.greeting = 'Todd Motto';
}]);
```

다음은 DOM에서 사용하는 방법이다:

```
<div ng-app="myApp">
    <div ng-controller="MainCtrl">
        <p>No filter: {{ greeting }}</p>
        <p>Reverse: {{ greeting | reverse }}</p>
    </div>
</div>
```

결과:

Result JavaScript HTML Edit in JSFiddle

No filter: Todd Motto

Reverse: ottoM ddoT

그리고 *ng-repeat* 안에서 다음과 같이 필터를 사용한다:

```
<ul>
  <li ng-repeat="number in myNumbers |filter:oddNumbers">{{ number }}</li>
</ul>
```

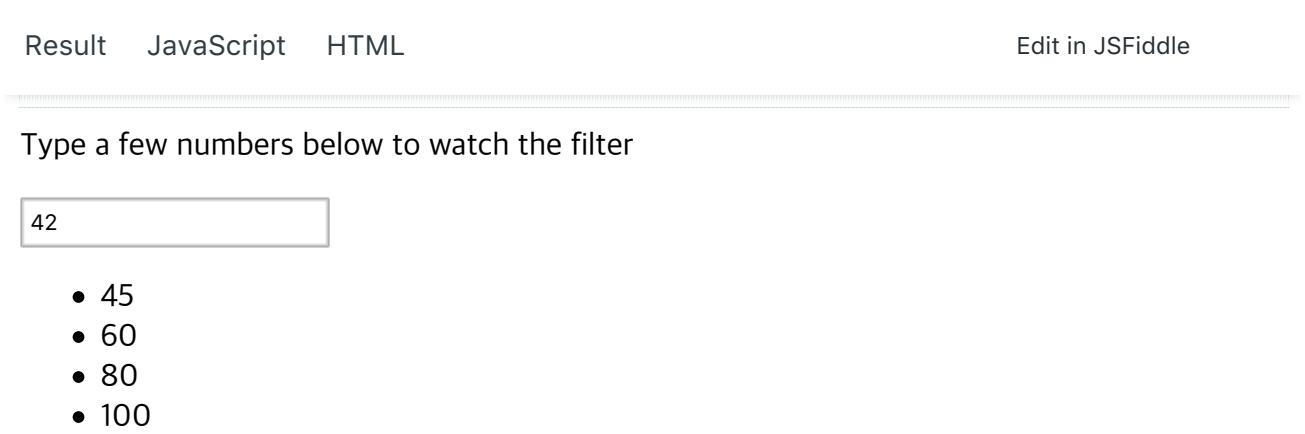
다음은 컨트롤러 안에서 필터를 선언하는 예제다:

```
myApp.controller('MainCtrl', ['$scope', function ($scope) {

    $scope.numbers = [10, 25, 35, 45, 60, 80, 100];

    $scope.lowerBound = 42;

    // 필터가 되어줘
    $scope.greaterThanNum = function (item) {
        return item > $scope.lowerBound;
    };

}]);
```

그리고 이 필터를 *ng-repeat* 에서 다음과 같이 사용한다:

```
<li ng-repeat="number in numbers | filter:greaterThanNum">
  {{ number }}
</li>
```

결과:



지금까지 AngularJS와 API의 중요한 부분만 살펴봤다. 물론 수박 겉핥기 정도로 살펴본 것 뿐이지만 여러분만의 Angular 애플리케이션을 만드는 데는 충분할 것이다.

양방향 데이터 바인딩

양방향 데이터 바인딩이라는 말을 처음 들었을 때는 무슨 말인지 제대로 이해하지 못했다. 양방향 데이터 바인딩을 한 문장으로 표현하자면 완전히 동기화된 데이터 정도가 가장 좋겠다. 즉 모델 을 갱신하면 뷰에 반영되고, 뷰 를 갱신하면 모델 에 반영되는 형태를 말한다. 이는 별다른 작업 없이도 데이터가 동기화된다는 뜻이다. 예를 들어 <input> 하나에 *ng-model* 을 바인딩하고 값을 입력하기 시작하면 동시에 모델이 생성(기존에 존재하면 갱신)된다.

<input>을 하나 생성해서 ‘myModel’이라는 모델을 연결해보자. 그리고 이중괄호 문법으로 모델을 정의하면 뷰와 즉시 연동될 것이다:

```
<div ng-app="myApp">
    <div ng-controller="MainCtrl">
        <input type="text" ng-model="myModel" placeholder="Start typing..." />
        <p>My model data: {{ myModel }}</p>
    </div>
</div>

myApp.controller('MainCtrl', ['$scope', function ($scope) {
    // 빈 문자열로 초기화하고 모델 데이터를 읽어온다.
    $scope.myModel = '';
}]);
```

결과:

XHR/Ajax/$http 호출과 JSON 바인딩

지금까지 *$scope* 에 기본적인 데이터를 넣는 방법과 모델이 어떻게 양방향 데이터 바인딩으로 동작하는지를 알아봤으니 이제 실제 서버의 XHR 호출을 시도해볼 차례다. 웹사이트에 Ajax 요구사항이 없을 수도 있으니 필수는 아니겠지만, 웹 애플리케이션에서 데이터를 가져오는 부분부터 살펴보자.

로컬 환경에서 개발할 때는 보통 자바, ASP.NET, PHP 등으로 로컬 서버를 사용할 것이고 로컬 데이터베이스 혹은 실제 서버에 접속해서 API로 통신할 것이다. 분명 이 부분은 별반 다르지 않으리라 본다.

'달러 http'라고 입력하자. 이제부터 좋은 친구가 되어줄 것이다. *$http* 메서드는 Angular가 서버 데이터에 접근하는 기능을 멋지게 래핑한 메서드로 눈감고 사용할 수 있을 정도로 쉽다. 다음은 'GET' 요청을 보내고 서버에서 데이터를 받아오는 간단한 예제다. 문법이 jQuery와 꽤 비슷해서 금방 이해할 수 있을 것이다:

```
myApp.controller('MainCtrl', ['$scope', '$http', function ($scope, $http) {
  $http({
    method: 'GET',
    url: '//localhost:9000/someUrl'
  });
}]);
```

이렇게 하면 Angular는 콜백을 좀 더 효율적이고 읽기 쉬운 형태로 작성할 수 있는 *promise* 라는 걸 반환한다. Promise는 *.myPromise()* 처럼 점을 사용해서 함수 체인을 구성할 수 있는데 예상대로 성공했을 때와 실패했을 때의 핸들러를 제공한다:

```
myApp.controller('MainCtrl', ['$scope', function ($scope) {
  $http({
    method: 'GET',
    url: '//localhost:9000/someUrl'
  })
  .success(function (data, status, headers, config) {
    // 성공! 데이터를 가져왔어
  })
  .error(function (data, status, headers, config) {
    // 이런. 뭔가 잘못되었음! :(
  });
}]);
```

읽기도 쉽고 간지도 난다. 이제 DOM에 모델을 바인딩하고 모델 데이터를 갱신해서 뷰와 서버를 잘 엮어

보자. Ajax 호출로 DOM에 사용자 이름을 추가한다고 해보자.

먼저 데이터를 바인딩할 JSON 구조를 정해야 한다. 백엔드 개발자가 애플리케이션이 사용할 API를 만들테니 다음처럼 간단하게 시작해보자:

```
{
  "user": {
    "name": "Todd Motto",
    "id": "80138731"
  }
}
```

즉 서버가 객체 하나를 반환해주고 (다른 이름으로 'data'를 호출할 것이다 [promise 핸들러에 *data* 가 있음]) *data.user* 속성을 읽어와야 한다는 것을 의미한다. *data.user* 속성 안에는 *name* 과 *id* 가 있다. 접근하기도 쉬우니 'Todd Motto'라는 값을 돌려주는 *data.user.name* 을 찾아서 적용해보자!

자바스크립트 (코드 안에 주석을 통해 설명하겠다):

```
myApp.controller('UserCtrl', ['$scope', '$http', function ($scope, $http) {

  // 사용자 객체를 생성
  $scope.user = {};

  // 빈 문자열로 초기화
  $scope.user.username = '';

  // 서버에 사용자 이름을 요청
  $http({
    method: 'GET',
    url: '//localhost:9000/someUrlForGettingUsername'
  })
  .success(function (data, status, headers, config) {
    // 서버로부터 받아온 사용자 이름을 모델에 할당!
    $scope.user.username = data.user.name;
  })
  .error(function (data, status, headers, config) {
    // 이런. 뭔가 잘못되었음! :(
  });
}]);
```

DOM에서는 다음과 같이 설정하면 된다:

```
<div ng-controller="UserCtrl">
  <p>{{ user.username }}</p>
</div>
```

이제 사용자 이름이 출력될 것이다. 자 이제 정말 흥미로운 선언적 데이터 바인딩에 대해 살펴보자.

선언적 데이터 바인딩

Angular의 철학은 기능이 풍부한 동적 HTML을 생성해서 웹 클라이언트 측에서는 상상할 수 없었을 만큼 많은 일을 보이지 않게 처리해주는 것이다. 이게 바로 Angular가 하려고 하는 일이다.

이제 메일 목록과 각 메일의 제목, 보낸 날짜를 Ajax 요청으로 가져와서 DOM에 그리는 기능을 구현한다고 생각해보자. Angular의 힘을 느껴볼 시간이다. 먼저 메일에 대한 컨트롤러를 만들자:

```
myApp.controller('EmailsCtrl', ['$scope', function ($scope) {

  // 이메일 객체를 생성
  $scope.emails = {};
```

```
  // 서버에서 데이터를 받아온 것처럼 꾸며보자.
  // 그냥 객체의 배열이다.
  $scope.emails.messages = [{
        "from": "Steve Jobs",
        "subject": "I think I'm holding my phone wrong :/",
        "sent": "2013-10-01T08:05:59Z"
    },{
        "from": "Ellie Goulding",
        "subject": "I've got Starry Eyes, lulz",
        "sent": "2013-09-21T19:45:00Z"
    },{
        "from": "Michael Stipe",
        "subject": "Everybody hurts, sometimes.",
        "sent": "2013-09-12T11:38:30Z"
    },{
        "from": "Jeremy Clarkson",
        "subject": "Think I've found the best car... In the world",
        "sent": "2013-09-03T13:15:11Z"
    }];

}]);
```

HTML에 이걸 넣을 필요 없다. 동적인 HTML 조각을 만들기 위해 애플리케이션이 무엇을 해야 하는지 선언하는 선언적 바인딩을 사용할 시간이다. Angular가 기본으로 제공하고 어떤 콜백이나 상태 변경 없이도 데이터를 순회하며 결과를 렌더링하는 *ng-repeat* 디렉티브를 사용해보자:

```
<ul>
  <li ng-repeat="message in emails.messages">
    <p>From: {{ message.from }}</p>
    <p>Subject: {{ message.subject }}</p>
    <p>{{ message.sent | date:'MMM d, y h:mm:ss a' }}</p>
  </li>
</ul>
```

결과:

Result JavaScript HTML Edit in JSFiddle

My Inbox:

From: Steve Jobs

Subject: I think I'm holding my phone wrong :/

Oct 1, 2013 5:05:59 PM

From: Ellie Goulding

date 필터 도 추가했으니 UTC 날짜로 그려주는 걸 볼 수 있을 것이다.

선언적 바인딩의 강력함을 더 확인하려면 Angular가 제공하는 *ng-** 디렉티브를 공부해보자. 서버와 모델, 뷰, 데이터를 그려주는 부분이 어떻게 잘 조화되는지 알 수 있을 것이다.

Scope 함수

선언적 바인딩에 이어서 scope 함수도 사용하면 멋진 애플리케이션을 만들 수 있다. 이제 데이터에서 메일 중 하나를 삭제 하는 기능을 구현해보자:

```
myApp.controller('MainCtrl', ['$scope', function ($scope) {

  $scope.deleteEmail = function (index) {
    $scope.emails.messages.splice(index, 1)
  };

}]);
```

고급 팁: 모델에서 데이터 를 지우는 동작을 생각해보는 건 중요하다. 실제 DOM과 연관된 요소를 지우는 게 아니기 때문이다. Angular는 MVC 프레임워크로서 양방향 바인딩과 콜백없이 모든걸 처리해준다. 우리가 해줘야 할 일은 데이터에 반응하는 코드를 현명하게 작성하는 것 뿐이다!

*ng-** 디렉티브를 통해 Scope에 함수를 바인딩해보자. 여기서는 *ng-click* 디렉티브를 사용한다:

```
<a ng-click="deleteEmail($index)">Delete email</a>
```

이 방법은 내부에 클릭 핸들러를 정의하는 것과 여러 가지 면에서 많이 다르다. 이유는 추후 살펴보기로 하자. *$index* 를 매개변수로 넘긴 게 보일 텐데 Angular가 어떤 메일을 지워야 하는지 알려주기 위함이다(얼마나 많은 코드와 로직이 필요없는지 보라!).

결과 (메일이 삭제된다!):

Result JavaScript HTML Edit in JSFiddle

My Inbox:

From: Steve Jobs

Subject: I think I'm holding my phone wrong :/

Oct 1, 2013 5:05:59 PM

Delete email

From: Ellie Goulding

선언적 DOM 메서드

이제 *DOM* 메서드 로 넘어가보자. 역시 디렉티브이며 보통 스크립트 로직으로 작성해서 DOM에 기능을 제공하는 형태다. 이를 잘 설명할 수 있는 예제로 간단한 토글 네비게이션이 좋겠다. *ng-show* 와 *ng-click* 을 사용해서 깔끔한 토글 네비게이션을 만들어보자:

```
<a href="" ng-click="toggle = !toggle">Toggle nav</a>
  <ul ng-show="toggle">
    <li>Link 1</li>
    <li>Link 2</li>
    <li>Link 3</li>
```

```
</ul>
```

이 코드는 컨트롤러가 없는 MVVM을 의미하여 나중에 다시 살펴보도록 한다.

결과 (토글된다!):

표현식

Angular에서 마음에 드는 부분 중에 하나가 자바스크립트의 for문을 사용해서 반복되는 코드를 작성하는 부분이다.

혹시 이렇게 작성해본적 있지 않는가?

```
elem.onclick = function (data) {
  if (data.length === 0) {
    otherElem.innerHTML = 'No data';
  } else {
    otherElem.innerHTML = 'My data';
  }
};
```

이 코드는 데이터의 상태에 따라 DOM을 수정하는 코드로 *GET* 요청에 대한 콜백으로 사용될법하다. Angular를 사용하면 이렇게 자바스크립트를 따로 작성하지 않아도 이 코드를 충분히 구현할 수 있다!

```
<p>{{ data.length > 0 && 'My data' || 'No data' }}</p>
```

이렇게 작성하면 콜백없이도 애플리케이션에서 데이터를 풀링하거나 읽어온 뒤 자신을 동적으로 갱신한다. 데이터가 없으면 알려줄 것이고 데이터가 있어도 말해줄 것이다. Angular는 양방향 바인딩이라는 마법으로 이러한 경우를 모두 자동으로 처리해준다.

결과:

Result JavaScript HTML Edit in JSFiddle

Test 1: No data

Test 2: My data

동적 뷰와 라우팅

단일 페이지 웹 애플리케이션(혹은 웹사이트!)에는 헤더, 푸터, 사이드바, 본문이 있고 URL에 따라 내용이 표시되는 게 보통이다.

Angular를 사용하면 동적 뷰 를 통해 이를 쉽게 설정할 수 있다. 동적 뷰를 사용하는 방법은 URL을 기준으로 *$routeProvider* 를 통해 특정 뷰를 얻어온 다음 적용하면 된다. 간단한 예를 살펴보자:

```
myApp.config(['$routeProvider', function ($routeProvider) {

  /**
   * $routeProvider
   */
  $routeProvider
  .when('/', {
    templateUrl: 'views/main.html'
  })
  .otherwise({
    redirectTo: '/'
  });

}]);
```

URL이 '/' (사이트의 루트) '이면' *main.html* 가 주입된다는 것을 알 수 있다. 초기 뷰로 *index.html* 대신 *main.html* 을 설정하는 게 좋은데 왜냐하면 *index.html* 페이지를 이미 단일 페이지 셋업에 사용했기 때문이다. 그리고 다른 URL에 대해서 뷰를 추가하는 것도 무척 쉽다:

```
myApp.config(['$routeProvider', function ($routeProvider) {

  /**
   * $routeProvider
   */
  $routeProvider
  .when('/', {
    templateUrl: 'views/main.html'
  })
  .when('/emails', {
    templateUrl: 'views/emails.html'
  })
  .otherwise({
    redirectTo: '/'
```

```
  });

}]);
```

이로서 이메일 목록을 보여주는 *emails.html* 을 간단하게 추가했다. 결국 매우 복잡한 애플리케이션을 아주 적은 노력으로 만들 수 있다는 것이다.

$routeProvider 서비스에는 공부할만한 게 더 많이 있지만 여러분의 몫으로 남겨둔다. 그리고 Ajax 호출이 진행 중일 때 이벤트를 보내는 *$http* 인터셉터같은 것도 있다. 새로운 데이터를 받아오는 동안에 로딩표시를 보여주는 용도로 사용할 수 있다.

전역 static 데이터

Gmail은 JSON으로 작성된 많은 양의 초기 데이터를 한 페이지에서 처리한다(오른쪽 클릭 - 페이지 소스 보기). 페이지에 데이터를 즉시 반영하고 싶으면 Angular를 사용해보자. 렌더링 속도까지 빨라질 것이다.

필자가 앱을 개발할 때는 자바 태그를 DOM안에 넣었고 렌더링될 때 서버로부터 데이터를 받아왔다. [필자가 Java 경험이 없어서 아래처럼 선언했지만 어떤 언어든 사용가능하다.] 다음은 페이지에 JSON을 작성해서 컨트롤러에 넣고 즉시 바인딩하는 방법이다:

```
<!-- index.html 내용 (물론 페이지 맨 아래) -->
<script>
window.globalData = {};
globalData.emails = <javaTagHereToGenerateMessages>;
</script>
```

페이지가 해석되는 동안 자바 태그가 데이터를 렌더링할 것이고 Angular는 이메일 목록을 즉시 렌더링할 것이다. 이제 컨트롤러에 데이터를 넣어보자:

```
myApp.controller('EmailsCtrl', ['$scope', function ($scope) {

    $scope.emails = {};

    // 초기 데이터를 설정!
    $scope.emails.messages = globalData.emails;

}]);
```

압축

Angular 코드 압축에 대한 이야기를 해볼까 한다. 아마 자바스크립트 코드를 압축해본 적이 있을테고 이로 인해 오류가 난 적도 있을 것이다!

AngularJS 코드를 압축하는 건 쉽다. 함수 앞의 배열에 주입해야하는 의존관계만 잘 정의하면 된다:

```
myApp.controller('MainCtrl',
['$scope', 'Dependency', 'Service', 'Factory',
function ($scope, Dependency, Service, Factory) {

  // 코드

}]);
```

압축되고 나면 다음과 같다:

```
myApp.controller('MainCtrl',
```

```
['$scope', 'Dependency', 'Service', 'Factory',
function (a,b,c,d) {

  // a = $scope
  // b = Dependency
  // c = Service
  // d = Factory

  // $scope 별칭이 사용됨
  a.someFunction = function () {...};

}]);
```

주입하는 의존 객체의 순서에 주의하자. 순서가 달라지면 분명 여러분은 물론 팀에 골치아픈 일이 생길 것이다.

MVC와 MVVM의 차이점

AngularJS 포스트를 마무리 지으면서 AngularJS의 자부심인 MVC/MVVM의 차이점에 대해 간단히 다뤄볼까 한다:

- *MVC*: 컨트롤러와 통신한다, 모델-뷰-컨트롤러
- *MVVM*: 기술적으로는 자기자신과 통신하는 선언적 데이터 바인딩이다. 모델-뷰-뷰-모델. 모델은 뷰와 통신하고 뷰는 모델과 통신한다. Angular의 양방향 데이터 바인딩은 별다른 작업없이도 스스로 알아서 통신한다. 또한 컨트롤러없이 로직을 작성할 수도 있다!

예를 들어 다음은 데이터를 제공하는 컨트롤러없이도 *ng-repeat* 을 생성하는 예제다:

```
<li ng-repeat="number in [1,2,3,4,5,6,7,8,9]">
  {{ number }}
</li>
```

테스트해보니 잘 동작하긴 하지만 깔끔하게 작성하려면 항상 컨트롤러를 사용하길 추천한다.

결과:

Result JavaScript HTML Edit in JSFiddle

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

HTML5 웹 컴포넌트

이전에 봤던것처럼 AngularJS에서도 사용자 정의 요소를 만들 수 있다:

```
<myCustomElement></myCustomElement>
```

이건 사실 HTML5의 미래를 웹에 구현한 것으로 Angular를 사용하면 HTML5의 웹 컴포넌트와 `<template>`요소를 비슷하게 사용할 수 있다. 웹 컴포넌트는 뷰를 생성하기 위한 동적 자바스크립트를 주입할 수 있는 사용자 정의 요소로 구성된다. - Angular를 사용하면 이 멋진 기능을 지금 구현할 수 있는 것이다. 즉 Angular는 이를 먼저 생각하고 다가올 웹 기술을 미리 검증한 것이다 - 경의를 표한다.

스코프 주석

내 생각에 스코프 주석은 업무를 도와주는 역할을 멋지게 해낸다. HTML에 다음과 같은 주석을 사용하는 것과 비교해보면 더욱 그렇다:

```
<!-- header -->
<header>
  Stuff.
</header>
<!-- /header -->
```

Angular를 소개할때면 DOM 대신 뷰와 스코프를 생각하라고 말하곤 한다. 고의로 컨트롤러간의 데이터를 공유하지 않는 한 스코프는 사실 말 그래도인 닫힌 범위 라서 다른 곳에서는 데이터를 접근할 수 없다. 따라서 한 스코프의 영역을 스코프 주석으로 구분하는 게 훨씬 도움이 된다:

```
<!-- scope: MainCtrl -->
<div class="content" ng-controller="MainCtrl">

</div>
<!-- /scope: MainCtrl -->
```

AngularJS 디버깅

Angular를 개발하고 디버깅할때는 구글이 추천하는 아주 멋진 크롬 확장 기능을 사용하면 좋다. 이름은 Batarang이고 여기서 받을 수 있다.

자 그럼, 즐거운 코딩되시길.

추가로 읽어볼 것들

- 이 글에 대한 발표자료 SpeakerDeck in slides
- 나만의 디렉티브를 작성하는 방법 code your own Directive

Posted by Soomong Jan 20th, 2014 AngularJS, Translation

« 2013년 회고

Comments

댓글 32건 **Insight? Practice!** 1 로그인

♡ 추천 7 공유 최신순

토론 참여하기

柳永根【SW】 • 3달 전
좋은 글이네요.. 같이 공부하는 친구들에게도 참고 할 수 있도록 알려줘야 겠네요.
^ | v • 답글 • 공유 ›

jason Jeon • 10달 전
감사합니다 개념이 팍팍~
^ | v • 답글 • 공유 ›

U02 • 일년 전
감사합니다~최근에 앙귤라 보고있엇는데 도움되엇어요
^ | v • 답글 • 공유 ›

HongJinBom • 일년 전
감사합니다 ㅎㅎㅎ
^ | v • 답글 • 공유 ›

김은경 • 2년 전
이글 읽으니까 AngularJS의 장점과 사용 방법이 한눈에 들어와서 너무 좋아요. 깔끔하게 정리해주신 것도 한몫하는 것 같습니다. :-)
^ | v • 답글 • 공유 ›

아라한사 • 2년 전
잘 보고 갑니다 ^^
^ | v • 답글 • 공유 ›

UCHAN • 2년 전
templateUrl 에 해당 Dom ID 값을 넣었는데 에러가 메세지가..

GET http://123.140.74.000/scs/setting/conf/templates/setting/server/resetWarning.html 404 (NOT FOUND)

해당 dom url 값도 넣어봣지만 같은 에러가 나네요

에러가 발생하는데 이런경우는 뭐가 문제일까요....ㅜ
^ | v • 답글 • 공유 ›

UCHAN • 2년 전
번역을...대단하세요
angular 공부중이였는데 큰 도움이 됬습니다

Soomong

장인 개발자를 꿈꾸는 견습개발자
신나고 즐겁게 개발하기 위해 노력중

Recent Posts

- [번역] 하루만에 끝내는 AngularJS
- 2013년 회고
- [책] Spring in Action
- 흰 띠를 매다
- [책] 피플웨어

Book