01

Mais sobre diretivas

Transcrição

Começando deste ponto? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/angular-1/stages/11-alurapic.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

Pedidos de melhorias sempre acontecem

É bem provável que nossos usuários estejam satisfeitos, mas não é incomum a solicitação de melhorias depois da aplicação pronta e podemos antecipar uma delas! Percebam que após a inclusão ou alteração de uma foto o botão `salvar` ganha foco. Isso faz todo sentido, porque ele foi o último elemento da página que sofreu interação. Mas se o mesmo usuário quiser alterar uma nova foto em seguida? Com certeza ele terá que clicar no botão `voltar` para pesquisar a foto na página principal.

Podemos melhorar a experiência do usuário fazendo com que o botão `voltar` ganhe o foco logo após a operação de inclusão ou alteração ser concluída. Essa pequena mudança permitirá que a tecla `ENTER` retorne o usuário para a página principal rapidamente. Mas como faremos isso?

Fazendo as pazes com DOM, mas cada um no seu quadrado!

Que tal realizarmos manipulação de DOM dentro do nosso controller, por exemplo, usando jQuery? Definitivamente não! Se fizermos isso será muito complicado testar nosso controller com nosso framework de testes favorito, justamente pela dependência do DOM.

Angular permite manipulação de DOM não em nossos controllers, mas em nossas diretivas! Como diretivas possuem escopo isolado, elas podem ser testadas separadamente, sem impactar no código dos controllers. Já sabemos criar o esqueleto de uma diretiva, aliás chamaremos nossa nova diretiva de **meuFocus**.

Editando `public/js/directives/minhas-diretivas.js` :

```
// public/js/directives/minhas-diretivas.js

angular.module('minhasDiretivas', [])
    .directive('meuPainel', function() {
       // código omitido
    })
    .directive('minhaFoto', function() {
       // código omitido
    })
    .directive('meuBotaoPerigo', function() {
       // código omitido
    })
    .directive('meuFocus', function() {
        var ddo = {};
        ddo.restrict = "A";
        ddo.scope = {};
```

```
        return ddo;
    });
```

Bom, criamos o esqueleto da nossa diretiva, mas como será sua API? Bom, ela será usada como atributo no botão. Mesmo sem terminarmos nossa diretiva, vamos adicioná-la no botão `voltar`:

```
<!-- public/partials/foto.html -->
<!-- HTML anterior omitido -->

<a href="/" meu-focus focado="focado" class="btn btn-primary">Voltar</a>

<!-- HTML posterior omitido -->
```

Além da nossa diretiva, adicionamos o atributo `focado`, que faz parte da API da nossa diretiva. Ele deve apontar para a propriedade `$scope.focado` do controller. Se for `true`, o elemento ganha o foco, se for `false`, ele deixa de ganhar. Sendo assim, o papel de `FotoController` será atribuir o valor `true` toda vez que cadastrarmos uma nova foto. Já vamos realizar essa alteração:

```
// public/js/controllers/foto-controller.js

// código anterior omitido

$scope.submeter = function() {

            if ($scope.formulario.$valid) {
                cadastroDeFotos.cadastrar($scope.foto)
                .then(function(dados) {
                    $scope.mensagem = dados.mensagem;
                    if (dados.inclusao) $scope.foto = {};
                    // novidade aqui!
                    $scope.focado = true;
                })
                .catch(function(erro) {
                    $scope.mensagem = erro.mensagem;
                });
            }
        };
```

Diretivas, @&= e seus segredos

Um ponto importante: depois do nosso botão ganhar o foco, `$scope.focado` deve receber o valor `false`, caso contrário nosso botão ficará com foco eternamente! Atribuir este valor `false` será papel da diretiva. É por esta razão que não podemos usar `@` e nem `&` em seu escopo isolado, modificadores que já usamos em outras diretivas. Usaremos `=`, que criará uma relação bidirecional, isto é, `FotoController` e a diretiva `meuFocus` trabalharão com a mesma referência para `$scope.focado`. Se `focado` mudar na diretiva, mudará no controller, se mudar no controller, mudará na diretiva:

```
// public/js/directives/minhas-diretivas.js

// código anterior omitido

    .directive('meuFocus', function() {
```

```
        var ddo = {};
        ddo.restrict = "A";
        ddo.scope = {
            focado : '='
        };

        return ddo;
    });
```

Muito bem, mas isso não resolve o seguinte problema: precisamos ter acesso ao elemento do DOM para que possamos focá-lo em nossa página. Então, toda diretiva processada pelo Angular passa por duas fases: **compile** e **link**. O retorno da fase `compile` sempre devolve uma função de `link`, inclusive é apenas nesta função que podemos atribuir observadores que são executados sempre que o valor da propriedade observada mudar, recurso que precisamos. Nela, também temos acesso ao escopo isolado da diretiva e ao elemento do DOM no qual a diretiva foi adicionada.

```
// public/js/directives/minhas-diretivas.js

// código anterior omitido

    .directive('meuFocus', function() {
        var ddo = {};
        ddo.restrict = "A";
        ddo.scope = {
            focado : '='
        };
        ddo.link = function(scope, element) {

        };

        return ddo;
    });
```

Excelente, mas não confunda `scope` da diretiva com `$scope`. O primeiro é o escopo isolado da diretiva, o segundo é escopo de um controller, tudo bem?

Conhecendo o oráculo do data binding: $watcher!

Um ponto que precisamos resolver: só podemos executar o código da diretiva quando `focado` mudar, ação que será inicialmente empreendida pelo nosso controller. Nossa diretiva só saberá que ela mudou se adicionarmos um observador, na linguagem do Angular, um **watcher**:

```
// public/js/directives/minhas-diretivas.js

// código anterior omitido

    .directive('meuFocus', function() {
        var ddo = {};
        ddo.restrict = "A";
        ddo.scope = {
            focado : '='
        };
```

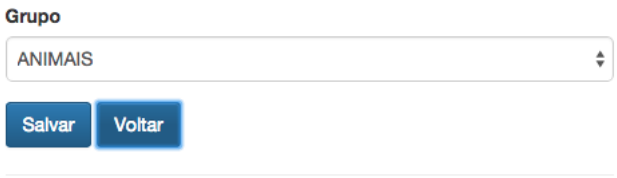
```
        ddo.link = function(scope, element) {
            scope.$watch('focado', function() {

                // executado toda vez que o valor mudar
                if (scope.focado) {

                   // se mudou e é verdadeiro, o elemento deve ganhar o foco
                }
            });
        };

        return ddo;
    });
```

O `scope.$watch` recebe dois parâmetros. O primeiro indica qual propriedade do escopo privado da diretiva desejamos observar por mudanças e o segundo a função que será executada sempre que a propriedade mudar. Se `scope.focado` for `true`, focaremos nosso elemento:

```
// public/js/directives/minhas-diretivas.js

// código anterior omitido

    .directive('meuFocus', function() {
        var ddo = {};
        ddo.restrict = "A";
        ddo.scope = {
            focado : '='
        };

        ddo.link = function(scope, element) {
            scope.$watch('focado', function() {

                if (scope.focado) {
                    element[0].focus();
                    scope.focado = false;
                }
            });
        };

        return ddo;
    });
```

O `element` é um elemento DOM, porém encapsulado pelo jqLite. O que é isso? É uma API de manipulação de DOM usada pelo Angular que tenta seguir o padrão da API `Sizzle`, mesmo padrão usando pelo jQuery, jbMobi ou Zepto. O problema é que o jqLite não possui a função `.focus()` que o jQuery possui. Neste caso, quando usamos `element[0]` significa que estamos acessando diretamente o elemento do DOM encapsulado pelo jqLite, sendo assim, podemos chamar a função `focus()` diretamente no elemento.

Será que funciona? Vamos selecionar uma foto qualquer e clicar em salvar:



Funciona perfeito! O mais bacana desse exemplo é que ele nos mostra de perto como funciona o mecanismo de data binding do Angular.

Consultar oráculos sai caro, há alternativa?

Conhecemos o `$watcher`, o "oráculo" do data binding do Angular, mas não podemos abusar dele em nossa diretivas, porque há um custo computacional nisso, o que pode tornar a atualização de nossa view bem lenta.

Não é o nosso caso, mas sempre que formos criar uma diretiva devemos nos perguntar se há outra forma de conseguirmos a funcionalidade sem o uso de watchers.

Menor custo com barramento de eventos

Em nossa diretiva, no lugar de usarmos watchers, trabalharemos com o barramento de eventos (*event bus*) do Angular.

Nosso primeiro passo será editar `FotoController` e substituir `$scope.focado` pelo disparo do evento **fotoCadastrada**, através de **$scope.$broadcast**:

```
// public/js/controllers/foto-controller.js

// código anterior omitido

$scope.submeter = function() {

            if ($scope.formulario.$valid) {
                cadastroDeFotos.cadastrar($scope.foto)
                .then(function(dados) {
                    $scope.mensagem = dados.mensagem;
                    if (dados.inclusao) $scope.foto = {};
                    // novidade aqui!
                    $scope.$broadcast('fotoCadastrada');
                })
                .catch(function(erro) {
                    $scope.mensagem = erro.mensagem;
                });
            }
        };
```

O nome do evento poderia ser qualquer um, mas deixamos claro que é um evento disparado quando a foto é incluída ou alterada. O `$broadcast` desce na hierarquia de elementos no escopo do controller, diferente do seu irmão `$emit`, que sobe na hierarquia.

Agora, vamos mudar nossa diretiva no que diz respeito à sua API. Primeiro, vamos alterar `foto.html`:

```
<!-- public/partials/foto.html -->
<!-- HTML anterior omitido -->

<a href="/" meu-focus class="btn btn-primary">Voltar</a>

<!-- HTML posterior omitido -->
```

Note que agora temos apenas a diretiva `meu-focus` sem qualquer outro atributo. Simples assim? Vamos alterar nossa diretiva:

```
// public/js/directives/minhas-diretivas.js

// código anterior omitido

    .directive('meuFocus', function() {
        var ddo = {};
        ddo.restrict = "A";
       // não tem mais scope
        ddo.link = function(scope, element) {
             scope.$on('fotoCadastrada', function() {
                 element[0].focus();
             });
        };

        return ddo;
    });
```

As mudanças são notáveis: primeiro, não precisamos configurar um escopo isolado. Em seguida, usamos `scope.$on` para ouvir o evento `fotoCadastrada` que será disparado pelo nosso controller. Quando o evento for disparado, manipulamos o DOM focando nosso botão. O resultado deve continuar o mesmo: nosso botão `voltar` ganhará o foco.

Interessante, mas uma pergunta que não quer calar: se alguém esquecer de disparar o evento no controller? E se outras diretivas também estiverem interessadas no evento `fotoCadastrada` ? Parece que um bom local para dispararmos esse evento é dentro do nosso serviço `cadastroDeFotos` . Não haverá acoplamento do produtor do evento e de quem o consumirá, algo interessante sob o ponto de vista da manutenção do nosso código.

O problema é que dentro de um serviço não temos acesso ao `$scope` , só controllers têm. Porém, podemos injetar dentro do nosso serviço o `$rootScope` , escopo pai de todos os escopos para a partir dele dispararmos nosso evento.

Primeiro, vamos remover o disparo do evento em `FotoController` :

```
// public/js/controllers/foto-controller.js

// código anterior omitido

$scope.submeter = function() {

            if ($scope.formulario.$valid) {
                cadastroDeFotos.cadastrar($scope.foto)
                .then(function(dados) {
                    $scope.mensagem = dados.mensagem;
                    if (dados.inclusao) $scope.foto = {};
```

```
                    // o broadcasting foi removido
                })
                .catch(function(erro) {
                    $scope.mensagem = erro.mensagem;
                });
            }
        };
```

Agora, vamos editar `public/js/services/meus-servicos`. No serviço `cadastroDeFotos`, injetaremos `$rootScope` e a partir dele dispararemos o evento `fotoCadastrada` quando a foto for incluída ou alterada. Como ele é o pai de todos os escopos dos controllers, os `$scope` também serão afetados pelo disparo do evento:

```
angular.module('meusServicos', ['ngResource'])
    .factory('recursoFoto', function($resource) {

        return $resource('/v1/fotos/:fotoId', null, {
            'update' : {
                method: 'PUT'
            }
        });
    })
    .factory("cadastroDeFotos", function(recursoFoto, $q, $rootScope) {

        // novidade
        var evento = 'fotoCadastrada';

        var service = {};
        service.cadastrar = function(foto) {
            return $q(function(resolve, reject) {

                if(foto._id) {
                    recursoFoto.update({fotoId: foto._id}, foto, function() {

                        // novidade
                        $rootScope.$broadcast(evento);

                        resolve({
                            mensagem: 'Foto ' + foto.titulo + ' atualizada com sucesso',
                            inclusao: false
                        });
                    }, function(erro) {
                        console.log(erro);
                        reject({
                            mensagem: 'Não foi possível atualizar a foto ' + foto.titulo
                        });
                    });

                } else {
                     recursoFoto.save(foto, function() {

                        // novidade
                        $rootScope.$broadcast(evento);

                        resolve({
                            mensagem: 'Foto ' + foto.titulo + ' incluída com sucesso',
                            inclusao: true
```

```
                    });
                }, function(erro) {
                    console.log(erro);
                    reject({
                        mensagem: 'Não foi possível incluir a foto ' + foto.titulo
                    });
                });
            }
        });
    };
    return service;
});
```

Show! Agora é cair dentro dos exercícios!

O que aprendemos neste capítulo?

- manipulação de DOM em diretivas
- observar mudanças em propriedades através de $watchers
- o modificador `=`
- barramento de eventos do Angular