AngularJS LifeCycle

Primeiramente precisamos entender o que acontece em uma aplicação angularjs quando a mesma é iniciada.

Vamos abstrair toda a parte que navegador ler o que está escrito, assets, html em sua página e vamos pular direto pra parte onde é disparado o evento `document.ready` onde, em uma inicialização não manual, o angularjs irá buscar através da DOM a diretiva `ng-app`, ou utilizando o `angular.bootstrap`, é especificado manualmente onde a(s) aplicação(ões) irá(ão) executar.
Após encontrar o `ng-app`, o angularjs irá iniciar sua "mágica" que é o passo onde o que está no contexto atual é compilado, nesse processo o código definido nas diretivas encontradas é executado, e no final de cada execução é gerado a função `link` das respectivas diretivas.
Com as funções `link` geradas o angularjs as combina com o nosso conhecido escopo, no qual contem as variáveis e conteúdos que é preciso pra gerar nossa UI, com a combinação é gerado a `view`, que é o que o usuário irá ver e interagir. Nesse passo também é instanciado o escopo de cada `controller` e `subcontroller`, e agora que entra a parte que queremos... "É ele que a gente quer...", aqui é onde aparece os nossos amigos: `watcher` e `listener`, o angularjs irá criar um para cada diretiva que ele encontrou e compilou, assim o nosso querido framework sabe onde encontrar cada coisa e que dado vincular à aquela diretiva.
Beleza, até ai tudo bem... Mas como ele atualiza tudo dinamicamente??

The Digest Cycle

Bom para manter tudo atualizado o *angularjs* usa uma estratégia bem inteligente. Como sabemos o `model` da aplicação não atualiza randomicamente, ele irá mudar em resposta a eventos (requisições, cliques, focos...), logo o angularjs adicionar `watchers` a todos `binds` e `ng-models` e o mesmo pode verificar se os valores contidos naquele escopo difere do que está na UI, e assim pode atualizá-la,

Mas ele não fica em um *loop* verificando a cada segundo se está tudo atualizado.

Provavelmente qualquer pessoa "normal" se inclinaria para essa ideia, porém obviamente isso é bastante ineficiente.

Como foi falado, o nosso `model` não atualiza randomicamente, porém há certas ações que provavelmente irão modificar ele, logo cada `ng-model` que pode mudar tem o seu respectivo `watcher` e sempre que um evento é disparado o *angularjs* verifica os `watchers` e `bidings` procurando por alterações e fazendo as devidas atualizações necessárias, e essa operação é chamada de

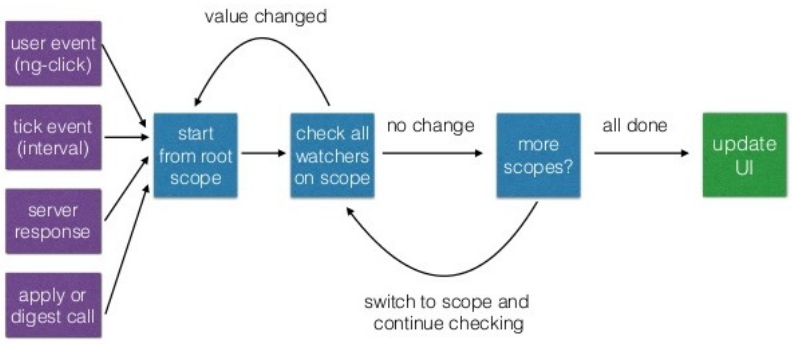
`Digest Cycle`.

O `digest cycle` é o que mantém a UI da nossa aplicação atualizada, e o trabalho que ele faz pode ser explicado nos seguintes passos:

- Sempre que um evento que pode mudar o `model` dispara, o angularjs executa o `digest cycle`.
- O `digest cycle` começa pelo $rootScope e vai verificando cada escopo procurando se existe alguma diference com o que está na UI.
- Se estiver tudo ok no escopo atual ele vai recursivamente até o seu escopo pai verificando todos adjacentes até o fim.
- Se por acaso o AngularJs verificar que um `watcher` reportou uma diferença ele é reiniciado.
- Ele reinicia pois uma alteração pode modificar um o estado de um `watcher` previamente verificado, e assim desencadear varias outras, ele somente para quando todos os `watchers` não reportarem modificação.

  Para prevenir um loop infinito o angularjs reinicia o `digest` apenas 10 vezes.
- Todas as atualizações da UI são acumuladas e assim que o ciclo é estabilizado, todas as atualizações são aplicadas.

Imagem para entender melhor todo esse ciclo:



FONTE: https://stackoverflow.com/questions/50594246/is-there-a-config-option-to-lower-the-amount-of-angularjs-1-x-digest-cycles

$scope.$apply() e $scope.$digest()

A similaridade entre estes dois métodos é basicamente chamar o que foi explicado anteriormente, **o digest cycle**, a diferença entre eles está baseada nos escopos.
Enquanto o $scope.$apply() força a execução do `digest cycle` direto do $rootScope (Basicamente toda a aplicação atual) o $scope.$digest() força a execução do `digest` apenas em o escopo atual.
Com isso em mente é possível imaginar em quais casos é bom ou não utilizar o $scope.$apply() e $scope.$digest(), a dica é: Em certos casos não precisamos verificar todos os `watchers` de nossa aplicação, apenas o escopo atual.
É preciso tomar cuidado ao utilizar componentes de terceiros que utilizam um `life cycle` diferente da sua aplicação, e nesse caso o mais indicado é utilizar o `$scope.$apply()` pois a atualização deve ser feita através do `$rootScope`, e não apenas no escopo atual com o `$scope.$digest()`.

Data-Binding: Entendendo a mágica do AngularJS

Explicando o `$digest()`

Ao demarcar tags HTMLs com expressões AngularJS (ex: **{{ message }}**, **data-ng-model="message"**), o *framework* irá associar um **watcher** para atualizar a *view* toda vez que o **$scope** *model* alterar. Um exemplo de criar um **watcher** abaixo:

```
$scope.$watch('message', function(newValue, oldValue) {

     // Atualiza o DOM Element

});
```

O segundo parâmetro da função é um *listener*, onde irá ser chamado quando o valor de **message** for

alterado.

OK, agora sabemos que quando demarcamos com expressões o AngularJS criar *watchers*. Agora a pergunta que não quer calar, quando que é chamado esses *watchers*? A resposta é... **$digest()**!

O ciclo **$digest** inicia todos os *watchers* criados para verificar se ocorreu mudanças no **$scope** *model* para serem atualizados. Ele é executado a partir da função **$scope.$digest()**. Um exemplo é quando usamos diretivas do AngularJS (ex: **ng-click**) com associação ao **$scope** *model*, automaticamente o AngularJS chama a função para inicializar o ciclo. Existem diretivas/serviços (ex: **ng-model**, **$http**) que disparam automaticamente o ciclo **$digest**.

Explicando o `$apply()`

"Uai" credo, surgiu outro cara na parada. Quem é esse fera? Calma galera, ele é nosso amigo. Lembra que o **$scope.$digest()** inicia o ciclo **$digest** relacionado ao **$scope** *model*? Então, o **$scope.$apply()** chama o **$rootScope.$digest()** que é o **$scope** *model* pai de todos os **$scopes**. "Tá" mas explica isso direito... em

outras palavras ao invés de chamar o **$digest()** só para **$scope** *model* específico, ele chama para todos os que existirem para ver se não estamos perdendo nada.

Agora, vamos assumir que você adicionou um **ng-click** em um botão passando o nome de uma função no **$scope** *model*. O AngularJS irá encapsular essa função dentro de uma chamada **$scope.$apply()**. Então ele vai chamar a função e no final irá executar o **$digest()**. Após isso irá ocorrer o que dissemos, irá iniciar os *watchers* criados e verificará os valores se foram modificados. Se forem modificados, irá executar seus *listeners*. :)

Quando chamar manualmente o ciclo $digest?

Existem algumas situações onde é necessário a chamada do ciclo manualmente. Um dos casos mais comuns são quando usamos algum plugin JQuery (ex: JQuery DatePicker). No AngularJS, existem diretivas/serviços que fazem parte do contexto onde ele automaticamente chama o ciclo **$digest()**. Agora esses tipos de situações que não seja dentro do AngularJS, trabalham fora do contexto do AngularJS, ou seja, o AngularJS não sabe das mudanças geradas e com isso não irá disparar o ciclo **$digest()**. Nesses momentos, devemos realizar a chamada manual para inicializar o ciclo.

Vamos supor que você utilize um **setTimeout()** para atualizar uma variável do **$scope** *model*:

```
angular.module('app', []).controller('appController', function($scope) {
   var _this = this;


   function getMessage() {
      setTimeout(function() {
         _this.message = 'Mensagem de 3 segundos';
      }, 3000);
   }
   getMessage();
});
```

Note que a resposta não aparece na tela porque o AngularJS jamais saberá o que ocorreu mesmo alterando o **$scope** *model*.

Para que o AngularJS reconheça a mudança, devemos chamar o **$scope.$apply()**. Segue abaixo o exemplo modificado:

```
angular.module('app', []).controller('appController', function($scope) {
   var _this = this;
   function getMessage() {
      setTimeout(function() {
         $scope.$apply(function() {
            _this.message = 'Mensagem de 3 segundos';
         })
      }, 3000);
   }
   getMessage();
});
```

PS: Ao invés de utilizar o `setTimeout()`, devemos utilizar o serviço $timeout que o AngularJS disponibiliza e que já realiza a chamada do `$scope.$apply()` ao termino do timeout.

Conclusão

Depois de toda essa explicação, o ponto principal é que o AngularJS detecta as mudanças que você realizou dentro do contexto. Se não detectar mudanças, então provavelmente ela está fora do contexto dele e então é necessário a chamada manualmente do ciclo `$digest`.