AngularJS: $compile & Directives

By Rodric Haddad

Salut tout le monde!

Bienvenue aux bureaux de Google Montreal, au GDG #17
Content de vous voir ici.

Je m'appelle Rodric, et je suis votre présenteur pour cette session.
J'ai le plaisir de vous présenter un composant vital mais peu connu d'AngularJS, qui est le service $compile.

The Presenter

Je me dois de vous dire qui je suis, en une phrase ou deux.

Je suis un contributeur au project d'Angular. Ça fait environ 2 ans que je fais ça, commençant sur irc et github.
j'ai passé mon été comme Stagiaire chez Google en Californie, sur l'équipe d' Angular.
Et actuellement je travaille encore dans l'équipe, à distance, pendant que je continue mes études.

This Talk

Donc cette présentation je la en français. Mes diapos et examples sont en Anglais par contre.
Sacha m'avait demander si je peux faire la présentation en français. J'ai vu ça comme un défi personelle et donc j'ai dis oui.

Mon but est qu'à la fin de cette présentation vous comprenez tous ce qui a rapport avec les directives et le compilateur d'Angular.
Je vais couvrir l'API de $compile et celle des directives en détail, en donnant mes opinions et mes conseils sur certain aspect de tous ça.

J'assume que vous avez déjà jouer avec Angular, voir même développer une application ou deux avec, et donc des concepts
comme le DI ou les $scope d'Angular sont pas si étrangers que ça.

Alors pour nous guider, je suis aller sur la documentation de $compile, j'ai pris l'API complète des directives,
et je l'ai mits dans un fichier qu'on consultra à certaines étapes de la présentation.

On va essayer de garder les questions pour la fin. N'hésitez pas à me les poser en anglais ou en français.

La présentation est assez chargée. Je vous conseil de prendre en note vos questions pour pas les oublier, sur votre cell ou un bout de papier,
et je vais utiliser un fichier texte pour mes exemples, vous pouvez aussi notez la

numéro de ligne dans le fichier

Why $compile?

Okay, sautons dans le sujet.

Pourquoi $compile? J'aurai pu choisir n'importe quel autre service comme $http ou $parse.
Mais j'ai choisi ce service, parce que selon moi, c'est ce qui fait en sorte qu'Angular, c'est Angular.
La majorité de la "magie" qu'on dit existe dans ce framework, peut s'expliquer en comprenant comment le compilateur marche.
$compile s'occupe de compiler le DOM, donc les elements HTML. Alors cela englobe toutes nos directives, et comment ils sont reliés à nos scope.

Mais allons-y doucement, commencant par l'API de ce service.

$compile API

Ce service prend du HTML et retourne une fonction qu'on appelera une "linking function", traduit c'est une fonction de liaison.
Cette fonction peut par la suite être donner un scope et elle retournera un element où tous les directives sont actifs.

Une affaire à noter, cette API permet alors que la fonction de liaison puisse être appéler plusieurs fois avec des scopes differents.
En faite, c'est comment un directive comme ng-repeat marche.

Y a aussi une autre affaire qui doit se passer pour que tous cela soit utile, et c'est d' enregister nos directives dans nos modules.
Alors faut fournir un object de définition pour dicter les fonctionnalités de chaque directive.
Remarquer le camelCase

Mais juste pour que vous sachiez comment le compilateur obtient ces directives.
Il y a une méthode sur le provider de $compile qui est appelé en arrière plan. Donc les directives finissent toujours dans les mains de $compile.
Si vous êtes familier avec le DI d'Angular, ça ça doit pas vous surprendre.

Mais normalement on utilise le shortcut de module.directive, pourquoi se compliqué la vie.

Donc on voit qu'avec l'API de $compile, il faut lui donner les directives en premier, et

après on a deux étapes distincts. Une de compilation, et l’autre de liason.

The Compile Phase

Commençons avec la première, celle de compilation.

J'aimerai vous donnez les details de cette phase, donc qu'est-ce qui se passe dedans concretement.

Imaginons qu'on a une structure HTML avec quelque directives et leurs paramètres.
Le compilateur traverse cette structure, et pour chaque élément, fait une collection contenant ses attributs et directives.
Les directives sont par la suite trié par leur priorité, du plus grand au plus petit, c'est alors là que l'attribut priority d'un directive rentre en jeu.

Par la suite, le compilateur fait 3 affaires pour valider les directives:

Il regarde quels directives demande un scope.
Il s'assure que pas plus d'un directive demande un scope isolé, parce que conceptuellement ça fait pas sens.
Alors si vous voyez une erreur relié à deux directives qui demande un scope isolé en même temps, repensez votre usage des directives.

Aussi, le compilateur regarde qui veut faire un transclusion, on en parlera après de ce concept. Même chose, pas plus qu'un directive a le droit de demander un transclusion sur un element.

Et enfin, il verifie si les directives concernés ont un template, si oui, le template est

injecté directement dans l'HTML.
Si le template est un lien vers une ressource, un url, alors la requête est déclenché à ce moment pour obtenir le template.

Une fois que tout cela est fait, la méthode .compile est appelé pour chaque directive, et elle obtient en arguments l'élément en question et sa liste d'attribut.
Cette fonction peut retourné une fonction de liaison qu'on abordra après.

Ce process implique 2 affaires:
Que la fonction "compile" d'un directive ne peut pas ajouter d'autres directives à l' élément actuel.
Cela est car les attributs et directive de l'élément on déjà était collecter avant, alors le compilateur ne peut se rendre compte de l'ajout.

L'autre affaire, c'est que par contre, on peut ajouter des directives aux éléments enfants, parce que la compilation commence de haut en-bas,
alors le compilateur collectera tous qu'on ajouterai aux éléments enfants.

The Compile Phase++

Il y a 2 affaires que je n'ai pas mentionné relier à la compilation.

Le premier est `replace`, et ça c'est car ce n'est plus officiellement supporter dans AngularJS.
Mais vous risquer de le voir dans des anciens projets, donc faudrait que j'en parle.
Tous ce que `replace` change, c'est que si on a un template: à la place que celui-ci soit injecter dans l'élément du directive, il le replacera.
Ça permet de replacer un élément à l'aide d'un directive.
Cela peut créer des problèmes avec des conflits d'attributs dans certaines circonstances, c'est la raison pourquoi on ne le supporte plus.

Aussi, il y a `terminal`. Un attribut assez intéressant et des fois utile.
Quand le compilateur voit un directive terminal, après qu'il appelle sa fonction de compile, le compilateur arrête complétement la compilation de cette branche du DOM, la directive est terminal
La raison pourquoi c'est utile, c'est que à ce moment là un directive peut ajouter d'autres directives et recommencer la compilation.
J'ai un example: Il faudra utiliser l'argument de maxPriority du service

$compile, et on a ce qu’on veut.

Directives Declaration Styles

(`restrict`)

Tous au long de ma présentation j'ai assumé qu'on utilise nos directives comme attribut, mais on peut aussi les utiliser comme element, class de css ou commentaire d'HTML

Comme ceci

On a juste à spécifier le restrict d'un directive

La majorité des personnes se limite aux attributs et elements. C'est rare de voir des class ou des commentaires d'utiliser

Une affaire relativement drôle qu'on peut faire une fois qu'on connait comment le compilateur s'occupe de ça, c'est de s'amuser avec le restrict.

Pour que le compilateur fait son affaire, il utilise la fonction .indexOf de javascript des string.
Tous ce qu'il fait, c'est s'assurer que la lettre existe dans la chaine de characteres.

Donc en faite, on peut faire des affaires comme:

Puisque qu'on est sur le sujet des différentes formes de déclarer des directives, j'aimerai dire que l'interpolation elle-même
est une directive spécial que le compilateur lui-même définit, je parle des petites mustaches

Avec ça on conclu tous ce qui a rapport avec la phase de compilation. Et jusqu'à date on a couvert ces partie de l'API des directives.

Avant de sauter au Linking phase, j'aimera juste mentionner une affaire.

https://twitter.com/rodyhaddad/status/367006771748945921

The Linking Phase

Vous vous rappelez j'ai dis que votre que le service $compile retourne une fonction de liaison?
J'ai aussi dis que la fonction .compile des directives peut en retourner une aussi.

C'est car quand la linkingFn de $compile est appelé, ceux des directives sont appelé aussi.

L'ordre dans lequel ils sont appelés est simple.
La fonction .compile peut en faite retourner un object, qui aurai les méthode pre et post.
Le pre sera appelé pendant que la fonction de liaison descend l'arborescente de l'HTML, et post sera appelé en remontant.
Une autre façon de le mettre c'est preLink sera exécuter avant que les elements enfants soit lier au scope, et postLink l'inverse.

Et quand la fonction .compile retourne simplement une fonction et non un object, la fonction est pris comme une postLink.

Puisque la majorité des directives on juste besoin d'intervenir dans la phase de liaison et non de compilation, on peut tout de suite spécifier la fonction link sur l'objet du directive.
C'est un shortcut.

Par défaut c'est post, et on peut spécifier un object pour pre et post.

C'est rare qu'on aie besoin d'utiliser pre, souvent post fait super bien l'affaire. C'est juste dans le cas où on veut lier un élément à un scope avant ses enfants, avec ça on a cette possibilité.

Regardons plus en détails la fonction de liaison, commençant par son premier argument.

… slides (update = met à jour) (& = Esperluette)

Le deuxième argument, c'est l'élément sur lequel le directive est attaché évidemment.
C'est un élément jQuery, ou jqLite quand jQuery n'est pas disponible sur la page.

Le troisième, c'est la collection d'attributs relié à l'élément, et elle a quelque méthode à noter.

Je veux juste mentionner $animate.
Utiliser ça pour faire des beau directives svp.

Je vais sauter les controlleurs pour l'instant pour aller tout de suite au transcludeFn.
Je suis mieux d'expliquer qu'est-ce que la transclusion pour commencer.
Par définition, la transclusion veut dire d'inclure un document dans un autre document par référence
Examples
…
Notons que le scope est garder intact avec la transclusion à l'aide de ng-transclude.

Maintenant qu'on a une idée c'est quoi la transclusion, on peut plus facilement comprendre la transcludeFn.

En faite, c'est comme une fonction de liaison

Ça: <!-- Ancre -->

Regardons ng-repeat comme example

Beaucoup de personnes ont de la misère avec le concept de transclusion.
J'ai essayer de l'expliquer diffèremment en incluant le concept de linking function.
Si vous avez compris la transclusion, ma mission ici est déjà accomplis et je dormirais bien ce soir.

Prenons une petite minute pour regarder l'API complète

Let’s Talk 2.0

Okay, on a couvert les deux phases de $compile, la phase de compilation et la phase de liaison.

J’aimerai prendre une petite minute pour parler d’Angular 2.0

Si vous n’avez pas encore entendu parler, c’est la nouvelle version d’Angular qui est actuellement en développement.
C’est loin d’être fini, mais c’est le résultat d’apprentissage obtenu par la communauté et par l’équipe dans les 6 dernières années qu’Angular existait.

Un concept centrale de cette nouvelle version est les WebComponents, qu’on nomera tout simplement Components.
Si vous avez vu la présentation à ng-europe sur 2.0, vous comprenez sûrement de quoi je parle.

Il me reste juste les controlleurs des directives à couvrir, et j’aimerai le faire en gardant en-tête le but d’utiliser les directives comme component,
comme des composants concret de notre application.

Components

Commençons par des exemples de components

Dialog, c'est un composant
Autocomplete, c'est un composant, on peut l'imaginer avoir un API
Select,
La seule affaire qui manque est le binding à un model

Les directives d'Angular viennent alors pour aider dans ça, et pour définir nos propres components si on veut.
Vous l'avait surement entendu avant, les directives c'est pour apprendre à l'HTML des nouvelles affaires, teach it new tricks.

Controllers

Donc ou est-ce que les controlleurs rentre en question?
Comme que <select> à son propre API, Angular nous permet de définir un API pour nos directives à l'aide de controlleur.

En faite, c'est un controller comme ceux auxquels on est habitué, où on peut injecter tous les services qu'on connait et qu'on aime.

Il y a quelque services spécifique aux controlleurs des directives, et en faite, c'est les même que ceux passé à la fonction de liaison.
$scope, $element, $attrs, $transclude.

Vous pouvez aussi spécifier le controller comme string, et l'enregistrer dans un module.
Ça aide dans l'organisation du code quand le controlleur d'un directive grandis.

Un controlleur peut être demander par n'importe qu'elle autre directives à l'aide de require, et devrait agir comme un API aux fonctionnalités du directive.
Donc on voit ici que ce otherDirective require le controlleur de myDirective qui se trouve sur le meme élément

À noter, les directives demander sont injecter dans la linking function.
Et un directive qui veut son propre controlleur peut le demander.

Un directive peut aussi demander pour plusieurs controlleurs en même temps à l'aide d'un array

Il y a aussi une micro-syntaxe pour controller quel controller on veut.

Un belle example d'un controlleur est celui ngModel

Mais ngModel n'est pas là pour rendre un élément comme un component, c'est juste pour aider avec les forms

Normalement quand on veut un component, on utilise controllerAs, et cela expose le controlleur à la vue.

Selon moi, un directive parfait est un où son linking function qui s'occupe juste des bindings entre le monde extérieur et le scope du directive,
et son controlleur contient toute la logique pour rendre le directive fonctionnel.
Idéalement, avec un API publique pour que le directive joue bien avec les autres.

Conclusion

Questions? Questions

@rodyhaddad

On vient de couvrir presque toutes l'API des directives. (voir fichier)

Le seul attribut que je vois c'est le templateNamespace.
Vous en aurez juste besoin si vous developer des directives pour du svg.
Le compilateur marche dans de l'svg comme dans de l'html.
Alors là en spécifiant cela au compilateur, il s'assure que tous les éléments créer sont dans le bon namespace.

Ex: ng-repeat dans svg

Alors voilà, j'ai conclu pour le coté tecnique des directives et du compilateur.
Je vous ai donner mon avis sur plusieurs affaires,

Je pense, que c'est le temps pour des questions, en anglais ou français, les deux marchent :)