APPRENEZ AngularJS

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#angularjs

Table des matières

À propos ... 1

Chapitre 1: Démarrer avec AngularJS ... 2

Remarques ... 2

Versions ... 2

Examples ... 9

Commencer ... 9

Présentation de toutes les constructions angulaires courantes ... 11

L'importance de la portée ... 12

Le plus simple possible Angular Hello World. ... 14

ng-app ... 14

Directives ... 14

Minification en angulaire ... 15

AngularJS Pour commencer Didacticiels vidéo ... 16

Chapitre 2: Angulaire MVC ... 19

Introduction ... 19

Examples ... 19

La vue statique avec contrôleur ... 19

démo mvc ... 19

Définition de la fonction de contrôleur ... 19

Ajout d'informations au modèle ... 19

Chapitre 3: angularjs avec filtre de données, pagination, etc. ... 20

Introduction ... 20

Examples ... 20

Angularjs affiche les données avec filtre, pagination ... 20

Chapitre 4: AngularJS des pièges et des pièges ... 21

Examples ... 21

La liaison de données bidirectionnelle cesse de fonctionner ... 21

Exemple ... 21

Choses à faire lors de l'utilisation de html5Mode ... 22

7 péchés capitaux d'AngularJS ... 23

Chapitre 5: Chargement paresseux 28
Remarques 28
Examples 28
Préparer votre projet pour un chargement paresseux 28
Usage 28
Utilisation avec routeur 29
UI-Router: 29
ngRoute: 29
Utiliser l'injection de dépendance 29
Utiliser la directive 30
Chapitre 6: Comment fonctionne la liaison de données 31
Remarques 31
Examples 31
Exemple de liaison de données 31
Chapitre 7: Composants 34
Paramètres 34
Remarques 35
Examples 35
Composants de base et crochets LifeCycle 35
Qu'est-ce qu'un composant? 35
Utilisation de données externes dans le composant: 35
Utilisation des contrôleurs dans les composants 36
Utiliser "require" comme objet 37
Composants en JS angulaire 37
Chapitre 8: Contrôleurs 39
Syntaxe 39
Examples 39
Votre premier contrôleur 39
Création de contrôleurs 41
Créer des contrôleurs, sécuriser la minification 41
L'ordre des dépendances injectées est important 41

Utilisation de contrôleurs dans JS angulaire 42
Création de contrôleurs angulaires sécuritaires 43
Contrôleurs imbriqués 44
Chapitre 9: Contrôleurs avec ES6 45
Examples 45
Manette 45
Chapitre 10: Décorateurs 46
Syntaxe 46
Remarques 46
Examples 46
Décorer service, usine 46
Directive décorer 47
Décorer le filtre 48
Chapitre 11: Demande $ http 49
Examples 49
Utiliser $ http dans un contrôleur 49
Utiliser la requête $ http dans un service 50
Calendrier d'une requête $ http 51
Chapitre 12: Des filtres 53
Examples 53
Votre premier filtre 53
Javascript 53
HTML 54
Filtre personnalisé pour supprimer des valeurs 54
Filtre personnalisé pour formater les valeurs 54
Effectuer un filtre dans un tableau enfant 55
Utilisation de filtres dans un contrôleur ou un service 56
Accéder à une liste filtrée depuis l'extérieur d'une répétition 57
Chapitre 13: Des promesses angulaires avec le service $ q 58
Examples 58
Utiliser $ q.all pour gérer plusieurs promesses 58
Utiliser le constructeur $ q pour créer des promesses 59

Report des opérations en utilisant $ q.defer 60
Utiliser des promesses angulaires avec le service $ q 60
Utiliser des promesses sur appel 61
Propriétés 61
Enveloppez la valeur simple dans une promesse en utilisant $ q.when () 63
$ q.when et son alias $ q.resolve 63
Évitez les $ q Anti-Pattern différé 64
Éviter ce anti-pattern 64
Chapitre 14: digestion en boucle 65
Syntaxe 65
Examples 65
liaison de données bidirectionnelle 65
$ digest et $ watch 65
l'arbre $ scope 66
Chapitre 15: directive classe ng 69
Examples 69
Trois types d'expressions de classe ng 69
1. ficelle 69
2. objet 69
3. tableau 70
Chapitre 16: Directives intégrées 71
Examples 71
Expressions angulaires - Texte vs Nombre 71
ngRépéter 71
ngShow et ngHide 75
ngOptions 76
ngModèle 78
ngClass 79
ngIf 79
JavaScript 80
Vue 80

DOM si currentUser n'est pas currentUser 80
DOM si currentUser est currentUser 80
Promesse de fonction 80
ngMouseenter et ngMouseleave 81
ngDisabled 81
ngDblclick 82
Directives intégrées Cheat Sheet 82
ngClick 84
ngRequired 85
ng-model-options 85
ngCloak 86
ngInclude 86
ngSrc 87
ngPattern 87
ngValue 87
ngCopy 88
Empêcher un utilisateur de copier des données 88
ngPaste 88
ngHref 88
ngList 89
Chapitre 17: Directives sur mesure 91
Introduction 91
Paramètres 91
Examples 93
Création et consommation de directives personnalisées 93
Modèle d'objet de définition de directive 94
Exemple de directive de base 95
Comment créer des composants utilisables à l'aide de directives 96
Directive de base avec modèle et une portée isolée 98
Construire un composant réutilisable 99
Décorateur de directive 100

Héritage de directive et interopérabilité 101
Chapitre 18: Directives utilisant ngModelController 103
Examples 103
Un contrôle simple: note 103
Un couple de contrôles complexes: éditer un objet complet 105
Chapitre 19: Distinguer Service vs Usine 109
Examples 109
Usine VS Service une fois pour toutes 109
Chapitre 20: Événements 111
Paramètres 111
Examples 111
Utiliser un système d'événements angulaires 111
$ scope. $ emit 111
$ scope. $ broadcast 111
Syntaxe: 112
Evénement enregistré propre à AngularJS 112
Usages et signification 113
Toujours désinscrire $ rootScope. $ Sur les écouteurs de l'événement scope $ destory 115
Chapitre 21: Filtres personnalisés 116
Examples 116
Exemple de filtre simple 116
example.js 116
exemple.html 116
Production attendue 116
Utiliser un filtre dans un contrôleur, un service ou un filtre 116
Créer un filtre avec des paramètres 117
Chapitre 22: Filtres personnalisés avec ES6 118
Examples 118
Filtre FileSize utilisant ES6 118
Chapitre 23: Fonctions d'assistance intégrées 120
Examples 120

angular.equals 120

angular.isString 120

angular.isArray 121

angular.merge 121

angular.isDefined et angular.isUndefined 121

angular.isDate 122

angular.isNumber 122

angular.isFunction 123

angular.toJson 123

angular.fromJson 124

angular.noop 124

angular.isObject 125

angular.isElement 125

copie angulaire 125

angular.identity 126

angulaire.pour chaque 126

Chapitre 24: Fournisseurs 128

Syntaxe 128

Remarques 128

Examples 128

Constant 128

Valeur 129

Usine 129

Un service 130

Fournisseur 130

Chapitre 25: Impression 132

Remarques 132

Examples 132

Service d'impression 132

Chapitre 26: Injection de dépendance 134

Syntaxe 134

Remarques 134

Examples.....134
Les injections.....134
Injections dynamiques.....135
$ inject Propriété Annotation.....135
Charger dynamiquement le service AngularJS en JavaScript vanilla.....135
Chapitre 27: Intercepteur HTTP.....137
Introduction.....137
Examples.....137
Commencer.....137
HttpInterceptor générique pas à pas.....137
Message Flash sur la réponse à l'aide de l'intercepteur http.....138
Dans le fichier de vue.....138
Fichier de script.....139
Pièges communs.....139
Chapitre 28: Le débogage.....141
Examples.....141
Débogage de base dans le balisage.....141
Utilisation de l'extension chrome ng-inspect.....142
Obtenir la portée de l'élément.....145
Chapitre 29: Le moi ou cette variable dans un contrôleur.....147
Introduction.....147
Examples.....147
Comprendre le but de la variable auto.....147
Chapitre 30: Les constantes.....149
Remarques.....149
Examples.....149
Créez votre première constante.....149
Cas d'utilisation.....149
Chapitre 31: Migration vers Angular 2+.....152
Introduction.....152
Examples.....152
Conversion de votre application AngularJS en une structure orientée composants.....152

Commencez à décomposer votre application en composants 152
Qu'en est-il des contrôleurs et des routes? 154
Et après? 154
Conclusion 155
Présentation des modules Webpack et ES6 155
Chapitre 32: Modules 156
Examples 156
Modules 156
Modules 156
Chapitre 33: ng-repeat 158
Introduction 158
Syntaxe 158
Paramètres 158
Remarques 158
Examples 158
Itération sur les propriétés de l'objet 158
Suivi et doublons 159
ng-repeat-start + ng-repeat-end 159
Chapitre 34: ng-style 161
Introduction 161
Syntaxe 161
Examples 161
Utilisation de style ng 161
Chapitre 35: ng-view 162
Introduction 162
Examples 162
ng-view 162
Enregistrement de navigation 162
Chapitre 36: Options de liaisons AngularJS (`=`, `@`, `&` etc.) 164
Remarques 164
Examples 164

@ liaison unidirectionnelle, liaison d'attribut....164
= liaison bidirectionnelle....164
& liaison de fonction, liaison d'expression....165
Liaison disponible via un échantillon simple....165
Attribut facultatif de liaison....166
Chapitre 37: Partage de données....167
Remarques....167
Examples....167
Utiliser ngStorage pour partager des données....167
Partage de données d'un contrôleur à un autre en utilisant le service....168
Chapitre 38: Portées angulaires....169
Remarques....169
Examples....169
Exemple de base de l'héritage $ scope....169
Éviter d'hériter des valeurs primitives....169
Une fonction disponible dans toute l'application....170
Créer des événements $ scope personnalisés....171
Utiliser les fonctions $ scope....172
Comment pouvez-vous limiter la portée d'une directive et pourquoi le feriez-vous?....173
Chapitre 39: Préparez-vous à la production - Grunt....175
Examples....175
Afficher le préchargement....175
Optimisation de script....176
Chapitre 40: Prestations de service....179
Examples....179
Comment créer un service....179
Comment utiliser un service....179
Créer un service avec angular.factory....180
$ sce - assainit et rend le contenu et les ressources dans des modèles....180
Comment créer un service avec des dépendances en utilisant la "syntaxe de tableau"....181
Enregistrement d'un service....181
Différence entre service et usine....182

Chapitre 41: Profilage de performance 186
Examples 186
Tout sur le profilage 186
Chapitre 42: Profilage et performance 189
Examples 189
7 améliorations simples de la performance 189
1) Utilisez ng-repeat avec parcimonie 189
2) Lier une fois 189
3) Les fonctions de portée et les filtres prennent du temps 190
4 spectateurs 191
5) ng-if / ng-show 192
6) Désactiver le débogage 192
7) Utiliser l'injection de dépendance pour exposer vos ressources 192
Lier une fois 193
Fonctions et filtres de portée 194
Observateurs 194
Alors, qu'est-ce que l'observateur? 195
ng-if vs ng-show 196
ng-if 196
ng-show 197
Exemple 197
Conclusion 197
Debounce votre modèle 197
Toujours désinscrire les auditeurs inscrits sur d'autres portées que l'étendue actuelle 198
Chapitre 43: Projet angulaire - Structure des répertoires 199
Examples 199
Structure du répertoire 199
Trier par type (à gauche) 199
Trier par élément (à droite) 200
Chapitre 44: Routage à l'aide de ngRoute 202
Remarques 202

Examples 202
Exemple de base 202
Exemple de paramètres de route 203
Définir un comportement personnalisé pour des itinéraires individuels 205
Chapitre 45: SignalR avec AngularJs 207
Introduction 207
Examples 207
SignalR et AngularJs [ChatProject] 207
Chapitre 46: Stockage de session 211
Examples 211
Gestion du stockage de session via le service à l'aide d'angularjs 211
Service de stockage de session: 211
Dans le contrôleur: 211
Chapitre 47: Tâches Grunt 212
Examples 212
Exécutez l'application localement 212
Chapitre 48: Tests unitaires 215
Remarques 215
Examples 215
Unité teste un filtre 215
Unité teste un composant (1.5+) 216
Unité teste un contrôleur 217
Unité teste un service 217
Unité teste une directive 218
Chapitre 49: ui-routeur 220
Remarques 220
Examples 220
Exemple de base 220
Vues multiples 221
Utilisation des fonctions de résolution pour charger des données 223
Vues imbriquées / États 224

Chapitre 50: Utilisation de directives intégrées 226
Examples 226
Masquer / Afficher les éléments HTML 226
Chapitre 51: Utiliser AngularJS avec TypeScript 228
Syntaxe 228
Examples 228
Contrôleurs angulaires en caractères dactylographiés 228
Utilisation du contrôleur avec la syntaxe ControllerAs 229
Utilisation de groupage / minification 230
Pourquoi la syntaxe ControllerAs? 231
Fonction du contrôleur 231
Pourquoi les contrôleurs? 231
Pourquoi $ scope 232
Chapitre 52: Validation de formulaire 233
Examples 233
Validation de formulaire de base 233
États de formulaire et d'entrée 234
Classes CSS 234
ngMessages 235
Approche traditionnelle 235
Exemple 235
Validation de formulaire personnalisé 236
Formulaires imbriqués 236
Validateurs asynchrones 237
Crédits 238

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: angularjs

It is an unofficial and free AngularJS ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official AngularJS.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec AngularJS

Remarques

AngularJS est un framework d'application Web conçu pour simplifier le développement d'applications côté client. Cette documentation concerne Angular 1.x , le prédécesseur de Angular 2, plus moderne, ou la documentation de Stack Overflow pour Angular 2 .

Versions

Version	Date de sortie
1.6.5	2017-07-03
1.6.4	2017-03-31
1.6.3	2017-03-08
1.6.2	2017-02-07
1.5.11	2017-01-13
1.6.1	2016-12-23
1.5.10	2016-12-15
1.6.0	2016-12-08
1.6.0-rc.2	2016-11-24
1.5.9	2016-11-24
1.6.0-rc.1	2016-11-21
1.6.0-rc.0	2016-10-26
1.2.32	2016-10-11
1.4.13	2016-10-10
1.2.31	2016-10-10
1.5.8	2016-07-22
1.2.30	2016-07-21
1.5.7	2016-06-15

Version	Date de sortie
1.4.12	2016-06-15
1.5.6	2016-05-27
1.4.11	2016-05-27
1.5.5	2016-04-18
1.5.4	2016-04-14
1.5.3	2016-03-25
1.5.2	2016-03-19
1.4.10	2016-03-16
1.5.1	2016-03-16
1.5.0	2016-02-05
1.5.0-rc.2	2016-01-28
1.4.9	2016-01-21
1.5.0-rc.1	2016-01-16
1.5.0-rc.0	2015-12-09
1.4.8	2015-11-20
1.5.0-beta.2	2015-11-18
1.4.7	2015-09-30
1.3.20	2015-09-30
1.2.29	2015-09-30
1.5.0-beta.1	2015-09-30
1.5.0-beta.0	2015-09-17
1.4.6	2015-09-17
1.3.19	2015-09-17
1.4.5	2015-08-28
1.3.18	2015-08-19

Version	Date de sortie
1.4.4	2015-08-13
1.4.3	2015-07-15
1.3.17	2015-07-07
1.4.2	2015-07-07
1.4.1	2015-06-16
1.3.16	2015-06-06
1.4.0	2015-05-27
1.4.0-rc.2	2015-05-12
1.4.0-rc.1	2015-04-24
1.4.0-rc.0	2015-04-10
1.3.15	2015-03-17
1.4.0-beta.6	2015-03-17
1.4.0-beta.5	2015-02-24
1.3.14	2015-02-24
1.4.0-beta.4	2015-02-09
1.3.13	2015-02-09
1.3.12	2015-02-03
1.4.0-beta.3	2015-02-03
1.3.11	2015-01-27
1.4.0-beta.2	2015-01-27
1.4.0-beta.1	2015-01-20
1.3.10	2015-01-20
1.3.9	2015-01-15
1.4.0-beta.0	2015-01-14
1.3.8	2014-12-19

Version	Date de sortie
1.2.28	2014-12-16
1.3.7	2014-12-15
1,3.6	2014-12-09
1.3.5	2014-12-02
1.3.4	2014-11-25
1.2.27	2014-11-21
1.3.3	2014-11-18
1.3.2	2014-11-07
1.3.1	2014-10-31
1.3.0	2014-10-14
1,3.0-rc.5	2014-10-09
1.2.26	2014-10-03
1.3.0-rc.4	2014-10-02
1.3.0-rc.3	2014-09-24
1.2.25	2014-09-17
1.3.0-rc.2	2014-09-17
1.2.24	2014-09-10
1.3.0-rc.1	2014-09-10
1.3.0-rc.0	2014-08-30
1.2.23	2014-08-23
1.3.0-beta.19	2014-08-23
1.2.22	2014-08-12
1.3.0-beta.18	2014-08-12
1.2.21	2014-07-25
1.3.0-beta.17	2014-07-25

Version	Date de sortie
1.3.0-beta.16	2014-07-18
1.2.20	2014-07-11
1.3.0-beta.15	2014-07-11
1.2.19	2014-07-01
1.3.0-beta.14	2014-07-01
1.3.0-beta.13	2014-06-16
1.3.0-beta.12	2014-06-14
1.2.18	2014-06-14
1.3.0-beta.11	2014-06-06
1.2.17	2014-06-06
1.3.0-beta.10	2014-05-24
1.3.0-beta.9	2014-05-17
1.3.0-beta.8	2014-05-09
1.3.0-beta.7	2014-04-26
1.3.0-beta.6	2014-04-22
1.2.16	2014-04-04
1.3.0-beta.5	2014-04-04
1.3.0-beta.4	2014-03-28
1.2.15	2014-03-22
1.3.0-beta.3	2014-03-21
1.3.0-beta.2	2014-03-16
1.3.0-beta.1	2014-03-08
1.2.14	2014-03-01
1.2.13	2014-02-15
1.2.12	2014-02-08

Version	Date de sortie
1.2.11	2014-02-03
1.2.10	2014-01-25
1.2.9	2014-01-15
1.2.8	2014-01-10
1.2.7	2014-01-03
1.2.6	2013-12-20
1.2.5	2013-12-13
1.2.4	2013-12-06
1.2.3	2013-11-27
1.2.2	2013-11-22
1.2.1	2013-11-15
1.2.0	2013-11-08
1.2.0-rc.3	2013-10-14
1.2.0-rc.2	2013-09-04
1.0.8	2013-08-22
1.2.0rc1	2013-08-13
1.0.7	2013-05-22
1.1.5	2013-05-22
1.0.6	2013-04-04
1.1.4	2013-04-04
1.0.5	2013-02-20
1.1.3	2013-02-20
1.0.4	2013-01-23
1.1.2	2013-01-23
1.1.1	2012-11-27

Version	Date de sortie
1.0.3	2012-11-27
1.1.0	2012-09-04
1.0.2	2012-09-04
1.0.1	2012-06-25
1.0.0	2012-06-14
v1.0.0rc12	2012-06-12
v1.0.0rc11	2012-06-11
v1.0.0rc10	2012-05-24
v1.0.0rc9	2012-05-15
v1.0.0rc8	2012-05-07
v1.0.0rc7	2012-05-01
v1.0.0rc6	2012-04-21
v1.0.0rc5	2012-04-12
v1.0.0rc4	2012-04-05
v1.0.0rc3	2012-03-30
v1.0.0rc2	2012-03-21
g3-v1.0.0rc1	2012-03-14
g3-v1.0.0-rc2	2012-03-16
1.0.0rc1	2012-03-14
0,10,6	2012-01-17
0,10,5	2011-11-08
0,10,4	2011-10-23
0.10.3	2011-10-14
0,10,2	2011-10-08
0,10,1	2011-09-09

Version	Date de sortie
0.10.0	2011-09-02
0.9.19	2011-08-21
0.9.18	2011-07-30
0.9.17	2011-06-30
0.9.16	2011-06-08
0.9.15	2011-04-12
0.9.14	2011-04-01
0.9.13	2011-03-14
0.9.12	2011-03-04
0.9.11	2011-02-09
0.9.10	2011-01-27
0.9.9	2011-01-14
0.9.7	2010-12-11
0.9.6	2010-12-07
0.9.5	2010-11-25
0.9.4	2010-11-19
0.9.3	2010-11-11
0.9.2	2010-11-03
0.9.1	2010-10-27
0.9.0	2010-10-21

Examples

Commencer

Créez un nouveau fichier HTML et collez le contenu suivant:

```
<!DOCTYPE html>
<html ng-app>
```

```
<head>
  <title>Hello, Angular</title>
  <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
</head>
<body ng-init="name='World'">
  <label>Name</label>
  <input ng-model="name" />
  <span>Hello, {{ name }}!</span>
  <p ng-bind="name"></p>
</body>
</html>
```

Démo en direct

Lorsque vous ouvrez le fichier avec un navigateur, vous verrez un champ de saisie suivi du texte `Hello, World!` . L'édition de la valeur dans l'entrée mettra à jour le texte en temps réel, sans avoir besoin de rafraîchir la page entière.

Explication:

1. Chargez la structure angulaire à partir d'un réseau de distribution de contenu.

   ```
   <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
   ```

2. Définir le document HTML en tant qu'application angulaire avec la directive `ng-app`

   ```
   <html ng-app>
   ```

3. Initialiser la variable de `name` utilisant `ng-init`

   ```
   <body ng-init=" name = 'World' ">
   ```

   Notez que ng-init doit être utilisé à des fins de démonstration et de test uniquement. Lors de la création d'une application réelle, les contrôleurs doivent initialiser les données.

4. Lier les données du modèle à la vue des contrôles HTML. Lier un `<input>` à la propriété `name` avec `ng-model`

   ```
   <input ng-model="name" />
   ```

5. Afficher le contenu du modèle en utilisant des accolades doubles `{{ }}`

   ```
   <span>Hello, {{ name }}</span>
   ```

6. Une autre manière de lier la propriété `name` consiste à utiliser `ng-bind` au lieu du guidon `"{{ }}"`

   ```
    <span ng-bind="name"></span>
   ```

Les trois dernières étapes établissent la *liaison de données bidirectionnelle* . Les modifications apportées à l'entrée mettent à jour le *modèle* , ce qui se reflète dans la *vue* .

Il y a une différence entre l'utilisation de guidons et de `ng-bind` . Si vous utilisez des guidons, vous pouvez voir le bon `Hello, {{name}}` lors du chargement de la page avant que l'expression ne soit résolue (avant le chargement des données) alors que si vous utilisez `ng-bind` , seules les données seront affichées. est résolu. Comme alternative, la directive `ng-cloak` peut être utilisée pour empêcher l'affichage des guidons avant leur compilation.

Présentation de toutes les constructions angulaires courantes

L'exemple suivant montre les constructions AngularJS courantes dans un fichier:

```
<!DOCTYPE html>
<html ng-app="myDemoApp">
  <head>
    <style>.started { background: gold; }</style>
    <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
    <script>
      function MyDataService() {
        return {
          getWorlds: function getWorlds() {
            return ["this world", "another world"];
          }
        };
      }

      function DemoController(worldsService) {
        var vm = this;
        vm.messages = worldsService.getWorlds().map(function(w) {
          return "Hello, " + w + "!";
        });
      }

      function startup($rootScope, $window) {
        $window.alert("Hello, user! Loading worlds...");
        $rootScope.hasStarted = true;
      }

      angular.module("myDemoApp", [/* module dependencies go here */])
        .service("worldsService", [MyDataService])
        .controller("demoController", ["worldsService", DemoController])
        .config(function() {
          console.log('configuring application');
        })
        .run(["$rootScope", "$window", startup]);
    </script>
  </head>
  <body ng-class="{ 'started': hasStarted }" ng-cloak>
    <div ng-controller="demoController as vm">
      <ul>
        <li ng-repeat="msg in vm.messages">{{ msg }}</li>
      </ul>
    </div>
  </body>
</html>
```

Chaque ligne du fichier est expliquée ci-dessous:

Démo en direct

1. `ng-app="myDemoApp"` , la directive ngApp qui initialise l'application et indique à angular qu'un élément DOM est contrôlé par un `angular.module` spécifique nommé `"myDemoApp"` ;
2. `<script src="angular.min.js">` est la première étape du démarrage de la bibliothèque AngularJS ;

Trois fonctions (`MyDataService` , `DemoController` et `startup`) sont déclarées, qui sont utilisées (et expliquées) ci-dessous.

3. `angular.module(...)` utilisé avec un tableau comme second argument crée un nouveau module. Ce tableau est utilisé pour fournir une liste des dépendances de module. Dans cet exemple, nous chaînons les appels sur le résultat de la fonction `module(...)` ;

4. `.service(...)` crée un service angulaire et renvoie le module pour le chaînage;

5. `.controller(...)` crée un contrôleur angulaire et renvoie le module pour le chaînage;

6. `.config(...)` Utilisez cette méthode pour enregistrer le travail à effectuer sur le chargement du module.

7. `.run(...)` que le code est exécuté au démarrage et prend un tableau d'éléments en tant que paramètre. Utilisez cette méthode pour enregistrer le travail qui doit être effectué lorsque l'injecteur a terminé de charger tous les modules.

   - le premier élément permet à Angular de savoir que la fonction de `startup` nécessite l' `$rootScope` service `$rootScope` en tant qu'argument;
   - le deuxième élément permet à Angular de savoir que la fonction de `startup` nécessite l' injection du service `$window` intégré en tant qu'argument;
   - le *dernier* élément du tableau, `startup` , est la fonction réelle à exécuter au démarrage;

8. `ng-class` est la directive ngClass pour définir une `class` dynamique et, dans cet exemple, utilise `hasStarted` sur `$rootScope` manière dynamique

9. `ng-cloak` est une directive pour empêcher le modèle HTML angulaire non rendu (par exemple " `{{ msg }}` ") d'être montré brièvement avant qu'Angular ait complètement chargé l'application.

10. `ng-controller` est la directive qui demande à Angular d'instancier un nouveau contrôleur de nom spécifique pour orchestrer cette partie du DOM;

11. `ng-repeat` est la directive qui permet à Angular d'itérer une collection et de cloner un modèle DOM pour chaque élément.

12. `{{ msg }}` présente l' interpolation : rendu sur place d'une partie de la portée ou du contrôleur;

L'importance de la portée

Comme Angular utilise HTML pour étendre une page Web et du Javascript pour ajouter de la logique, il est facile de créer une page Web en utilisant **ng-app** , **ng-controller** et certaines directives intégrées telles que **ng-if** , **ng-repeat** , etc. Avec la nouvelle syntaxe **controllerAs** , les nouveaux utilisateurs Angular peuvent associer des fonctions et des données à leur contrôleur au lieu d'utiliser `$scope` .

Cependant, il est tôt ou tard important de comprendre ce qu'est exactement ce `$scope` chose est. Il continuera à apparaître dans les exemples, il est donc important d'avoir une certaine compréhension.

La bonne nouvelle est que c'est un concept simple mais puissant.

Lorsque vous créez ce qui suit:

```
<div ng-app="myApp">
 <h1>Hello {{ name }}</h1>
</div>
```

Où habite le **nom** ?

La réponse est que Angular crée un objet `$rootScope` . Ceci est simplement un objet Javascript régulier et **name** est une propriété de l'objet `$rootScope` :

```
angular.module("myApp", [])
  .run(function($rootScope) {
    $rootScope.name = "World!";
  });
```

Et comme pour la portée globale en Javascript, ce n'est généralement pas une bonne idée d'ajouter des éléments à la portée globale ou à `$rootScope` .

Bien sûr, la plupart du temps, nous créons un contrôleur et mettons nos fonctionnalités requises dans ce contrôleur. Mais lorsque nous créons un contrôleur, Angular fait de la magie et crée un objet `$scope` pour ce contrôleur. Ceci est parfois appelé la **portée locale** .

Donc, en créant le contrôleur suivant:

```
<div ng-app="myApp">
  <div ng-controller="MyController">
    <h1>Hello {{ name }}</h1>
  </div>
</div>
```

permettrait à la portée locale d'être accessible via le paramètre `$scope` .

```
angular.module("myApp", [])
  .controller("MyController", function($scope) {
    $scope.name = "Mr Local!";
  });
```

Un contrôleur sans paramètre `$scope` peut simplement pas en avoir besoin pour une raison

quelconque. Mais il est important de comprendre que, **même avec la syntaxe controllerAs** , la portée locale existe.

Comme `$scope` est un objet JavaScript, Angular le configure comme par magie pour hériter de façon prototypique de `$rootScope` . Et comme vous pouvez l'imaginer, il peut y avoir une chaîne de portées. Par exemple, vous pouvez créer un modèle dans un contrôleur parent et lui associer la portée du contrôleur parent en tant que `$scope.model` .

Ensuite, via le prototype de chaîne, un contrôleur enfant peut accéder à ce même modèle localement avec `$scope.model` .

Rien de tout cela n'est évident au départ, car Angular fait juste sa magie en arrière-plan. Mais comprendre `$scope` est une étape importante pour apprendre comment Angular fonctionne.

Le plus simple possible Angular Hello World.

Angular 1 est à la base un compilateur DOM. Nous pouvons le transmettre au format HTML, soit en tant que modèle, soit en tant que page Web standard, puis le compiler.

Nous pouvons dire à Angular de traiter une région de la page comme une *expression en* utilisant la syntaxe du style de guidon `{{ }}` . Tout ce qui se trouve entre les accolades sera compilé, comme ceci:

```
{{ 'Hello' + 'World' }}
```

Cela va sortir:

```
HelloWorld
```

ng-app

Nous indiquons à Angular la partie de notre DOM à traiter comme modèle principal à l'aide de la *directive* `ng-app` . Une directive est un attribut ou un élément personnalisé que le compilateur de modèles angulaires sait gérer. Ajoutons maintenant une directive ng-app:

```
<html>
  <head>
    <script src="/angular.js"></script>
  </head>
  <body ng-app>
    {{ 'Hello' + 'World' }}
  </body>
</html>
```

J'ai maintenant demandé à l'élément body d'être le template racine. Tout ce qu'il contient sera compilé.

Directives

Les directives sont des directives de compilation. Ils étendent les capacités du compilateur Angular DOM. C'est pourquoi **Misko** , le créateur d'Angular, décrit Angular comme:

> "Un navigateur Web aurait été conçu pour les applications Web.

Nous créons littéralement de nouveaux attributs et éléments HTML et les compilons dans une application. `ng-app` est une directive qui active simplement le compilateur. Les autres directives incluent:

- `ng-click` , qui ajoute un gestionnaire de clic,
- `ng-hide` , qui cache conditionnellement un élément, et
- `<form>` , qui ajoute un comportement supplémentaire à un élément de formulaire HTML standard.

Angular est livré avec environ 100 directives intégrées qui vous permettent d'accomplir les tâches les plus courantes. Nous pouvons également écrire les nôtres, et ceux-ci seront traités de la même manière que les directives intégrées.

Nous construisons une application Angular à partir d'une série de directives, reliées par HTML.

Minification en angulaire

Qu'est-ce que la minification?

C'est le processus de suppression de tous les caractères inutiles du code source sans changer sa fonctionnalité.

Syntaxe normale

Si nous utilisons une syntaxe angulaire normale pour écrire un contrôleur, après avoir minifié nos fichiers, cela va casser notre fonctionnalité.

Contrôleur (avant minification):

```
var app = angular.module('mainApp', []);
app.controller('FirstController', function($scope) {
    $scope.name= 'Hello World !';
});
```

Après avoir utilisé l'outil de minification, il sera minifié comme ci-dessous.

```
var app=angular.module("mainApp",[]);app.controller("FirstController",function(e){e.name=
'Hello World !'})
```

Ici, la minification a supprimé les espaces inutiles et la variable $ scope du code. Donc, lorsque nous utilisons ce code minifié, il ne va rien imprimer. Parce que $ scope est un élément crucial entre le contrôleur et la vue, qui est maintenant remplacé par la petite variable "e". Donc, lorsque vous exécutez l'application, elle va donner une erreur de dépendance "e" au fournisseur inconnu.

Il y a deux manières d'annoter votre code avec des informations de nom de service qui sont

sécurisées par minification:

Syntaxe d'annotation en ligne

```
var app = angular.module('mainApp', []);
app.controller('FirstController', ['$scope', function($scope) {
    $scope.message = 'Hello World !';
}]);
```

Syntaxe d'annotation de propriété $ inject

```
FirstController.$inject = ['$scope'];
var FirstController = function($scope) {
    $scope.message = 'Hello World !';
}

var app = angular.module('mainApp', []);
app.controller('FirstController', FirstController);
```

Après minification, ce code sera

```
var
app=angular.module("mainApp",[]);app.controller("FirstController",["$scope",function(a){a.message="Hel
World !"}]);
```

Ici, angular considérera la variable 'a' comme étant $ scope, et affichera la sortie 'Hello World!'.

AngularJS Pour commencer Didacticiels vidéo

Il y a beaucoup de bons tutoriels vidéo pour le framework AngularJS sur egghead.io

Browse Courses

all

WATCH LUKAS RUEBBELKE'S COURSE

Using Angular 2 Patterns in Angular 1.x Apps

Implementing modern component-based architectu in your new or existing Angular 1.x web application i breath of fresh air.

In this course, y...

0 of 13 lessons

WATCH AARON FROST'S COURSE

Introduction to Angular Material

Angular Material is an Angula native, UI component framew from Google. It is a reference implementation of Google's Material Design and provide...

0 of 7 lessons

WATCH KENT C. DODDS'S COURSE

AngularJS Authentication with JWT

JSON Web Tokens (JWT) are a more modern approach to authentication. As the web m to a greater separation betwe the client and server, JWT pro

0 of 7 lessons

WATCH JOEL HOOKS'S COURSE

Learn Protractor Testing for AngularJS

Protractor is an end-to-end te framework for AngularJS applications. It allows you do the browser and test the expe state of your ap...

0 of 10 lessons

- https://egghead.io/courses/angularjs-app-from-scratch-getting-started

- https://egghead.io/courses/angularjs-application-architecture
- https://egghead.io/courses/angular-material-introduction
- https://egghead.io/courses/building-an-angular-1-x-ionic-application
- https://egghead.io/courses/angular-and-webpack-for-modular-applications
- https://egghead.io/courses/angularjs-authentication-with-jwt
- https://egghead.io/courses/angularjs-data-modeling
- https://egghead.io/courses/angular-automation-with-gulp
- https://egghead.io/courses/learn-protractor-testing-for-angularjs
- https://egghead.io/courses/ionic-quickstart-for-windows
- https://egghead.io/courses/build-angular-1-x-apps-with-redux
- https://egghead.io/courses/using-angular-2-patterns-in-angular-1-x-apps

Lire Démarrer avec AngularJS en ligne: https://riptutorial.com/fr/angularjs/topic/295/demarrer-avec-angularjs

Chapitre 2: Angulaire MVC

Introduction

Dans **AngularJS,** le modèle **MVC** est implémenté en JavaScript et en HTML. La vue est définie en HTML, tandis que le modèle et le contrôleur sont implémentés en JavaScript. Il existe plusieurs manières de regrouper ces composants dans AngularJS, mais la forme la plus simple commence par la vue.

Examples

La vue statique avec contrôleur

démo mvc

Bonjour le monde

Définition de la fonction de contrôleur

```
var indexController = myApp.controller("indexController", function ($scope) {
    // Application logic goes here
});
```

Ajout d'informations au modèle

```
var indexController = myApp.controller("indexController", function ($scope) {
    // controller logic goes here
    $scope.message = "Hello Hacking World"
});
```

Lire Angulaire MVC en ligne: https://riptutorial.com/fr/angularjs/topic/8667/angulaire-mvc

Chapitre 3: angularjs avec filtre de données, pagination, etc.

Introduction

Exemple de fournisseur et requête sur les données d'affichage avec filtre, pagination, etc. dans Angularjs.

Examples

Angularjs affiche les données avec filtre, pagination

```
<div ng-app="MainApp" ng-controller="SampleController">
    <input ng-model="dishName" id="search" class="form-control" placeholder="Filter text">
    <ul>
        <li dir-paginate="dish in dishes |  filter : dishName | itemsPerPage: pageSize"
pagination-id="flights">{{dish}}</li>
    </ul>
    <dir-pagination-controls boundary-links="true" on-page-
change="changeHandler(newPageNumber)" pagination-id="flights"></dir-pagination-controls>
</div>
<script type="text/javascript" src="angular.min.js"></script>
<script type="text/javascript" src="pagination.js"></script>
<script type="text/javascript">

var MainApp = angular.module('MainApp', ['angularUtils.directives.dirPagination'])
MainApp.controller('SampleController', ['$scope', '$filter', function ($scope, $filter) {

    $scope.pageSize = 5;

    $scope.dishes = [
    'noodles',
    'sausage',
    'beans on toast',
    'cheeseburger',
    'battered mars bar',
    'crisp butty',
    'yorkshire pudding',
    'wiener schnitzel',
    'sauerkraut mit ei',
    'salad',
    'onion soup',
    'bak choi',
    'avacado maki'
    ];

    $scope.changeHandler = function (newPage) { };
}]);
</script>
```

Lire angularjs avec filtre de données, pagination, etc. en ligne:
https://riptutorial.com/fr/angularjs/topic/10821/angularjs-avec-filtre-de-donnees--pagination--etc-

Chapitre 4: AngularJS des pièges et des pièges

Examples

La liaison de données bidirectionnelle cesse de fonctionner

On devrait avoir en tête que:

1. La liaison de données d'Angular s'appuie sur l'héritage prototypique de JavaScript, donc sujet à l' observation de variables .
2. Une portée enfant hérite normalement de sa portée parent. Une exception à cette règle est une directive qui a une portée isolée car elle n'hérite pas de manière prototypique.
3. Il existe certaines directives qui créent une nouvelle portée enfant: `ng-repeat` , `ng-switch` , `ng-view` , `ng-if` , `ng-controller` , `ng-include` , etc.

Cela signifie que lorsque vous essayez de lier certaines données à une primitive située à l'intérieur d'une étendue enfant (ou vice versa), les choses risquent de ne pas fonctionner comme prévu. Voici un exemple de la facilité avec laquelle *"casser"* AngularJS.

Ce problème peut facilement être évité en suivant ces étapes:

1. Avoir un "." à l'intérieur de votre modèle HTML à chaque fois que vous liez des données
2. Utiliser la syntaxe `controllerAs` pour promouvoir l'utilisation de la liaison à un objet "pointillé"
3. $ parent peut être utilisé pour accéder aux variables de `scope` parent plutôt qu'à la portée enfant. comme dans `ng-if` on peut utiliser `ng-model="$parent.foo"` ..

Une alternative à ce qui précède consiste à lier `ngModel` à une fonction getter / setter qui mettra à jour la version mise en cache du modèle lorsqu'elle est appelée avec des arguments, ou la renverra lorsqu'elle sera appelée sans arguments. Pour utiliser une fonction getter / setter, vous devez ajouter `ng-model-options="{ getterSetter: true }"` à l'élément avec l'attribut `ngModal` et appeler la fonction getter si vous souhaitez afficher sa valeur dans l'expression (Exemple de travail).

Exemple

Vue:

```
<div ng-app="myApp" ng-controller="MainCtrl">
    <input type="text" ng-model="foo" ng-model-options="{ getterSetter: true }">
    <div ng-if="truthyValue">
        <!-- I'm a child scope (inside ng-if), but i'm synced with changes from the outside
scope -->
        <input type="text" ng-model="foo">
    </div>
```

```
    <div>$scope.foo: {{ foo() }}</div>
</div>
```

Manette:

```
angular.module('myApp', []).controller('MainCtrl', ['$scope', function($scope) {
    $scope.truthyValue = true;

    var _foo = 'hello'; // this will be used to cache/represent the value of the 'foo' model

    $scope.foo = function(val) {
        // the function return the the internal '_foo' varibale when called with zero
arguments,
        // and update the internal `_foo` when called with an argument
        return arguments.length ? (_foo = val) : _foo;
    };
}]);
```

Meilleure pratique : Il est préférable de garder les lecteurs rapidement car Angular est susceptible de les appeler plus fréquemment que les autres parties de votre code (référence).

Choses à faire lors de l'utilisation de html5Mode

Lorsque vous utilisez `html5Mode([mode])` il est nécessaire que:

1. Vous spécifiez l'URL de base de l'application avec un `<base href="">` dans la tête de votre `index.html` .
2. Il est important que la balise de `base` vienne avant les balises contenant des requêtes URL. Sinon, cela pourrait entraîner cette erreur - `"Resource interpreted as stylesheet but transferred with MIME type text/html"` . Par exemple:

```
<head>
    <meta charset="utf-8">
    <title>Job Seeker</title>

    <base href="/">

    <link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap.css" />
    <link rel="stylesheet" href="/styles/main.css">
</head>
```

3. Si vous ne voulez pas spécifier une balise de `base` , configurez `$locationProvider` pour ne pas exiger `base` balise de `base` en transmettant un objet définition avec `requireBase:false` à `$locationProvider.html5Mode()` comme ceci:

```
$locationProvider.html5Mode({
    enabled: true,
    requireBase: false
});
```

4. Afin de prendre en charge le chargement direct des URL HTML5, vous devez activer la

réécriture d'URL côté serveur. De AngularJS / Guide du développeur / Utilisation de $ location

> L'utilisation de ce mode nécessite la réécriture d'URL côté serveur. En gros, vous devez réécrire tous vos liens vers le point d'entrée de votre application (par exemple, `index.html`). Exiger une `<base>` est également important dans ce cas, car cela permet à Angular de différencier la partie de l'URL qui est la base de l'application et le chemin devant être traité par l'application.

Une excellente ressource pour des exemples de réécriture de requêtes pour différentes implémentations de serveurs HTTP peut être trouvée dans la FAQ de ui-router - Comment: configurer votre serveur pour qu'il fonctionne avec html5Mode . Par exemple, Apache

```
RewriteEngine on

# Don't rewrite files or directories
RewriteCond %{REQUEST_FILENAME} -f [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^ - [L]

# Rewrite everything else to index.html to allow html5 state links
RewriteRule ^ index.html [L]
```

nginx

```
server {
    server_name my-app;

    root /path/to/app;

    location / {
        try_files $uri $uri/ /index.html;
    }
}
```

Express

```
var express = require('express');
var app = express();

app.use('/js', express.static(__dirname + '/js'));
app.use('/dist', express.static(__dirname + '/../dist'));
app.use('/css', express.static(__dirname + '/css'));
app.use('/partials', express.static(__dirname + '/partials'));

app.all('/*', function(req, res, next) {
    // Just send the index.html for other files to support HTML5Mode
    res.sendFile('index.html', { root: __dirname });
});

app.listen(3006); //the port you want to use
```

7 péchés capitaux d'AngularJS

Vous trouverez ci-dessous la liste des erreurs souvent commises par les développeurs lors de l'utilisation des fonctionnalités d'AngularJS, de quelques leçons apprises et des solutions qui leur sont apportées.

1. Manipulation des DOM via le contrôleur

C'est légal, mais doit être évité. Les contrôleurs sont les endroits où vous définissez vos dépendances, liez vos données à la vue et développez une logique métier supplémentaire. Vous pouvez manipuler techniquement le DOM dans un contrôleur, mais chaque fois que vous avez besoin d'une manipulation identique ou similaire dans une autre partie de votre application, un autre contrôleur sera nécessaire. La meilleure pratique de cette approche consiste donc à créer une directive incluant toutes les manipulations et à utiliser la directive dans toute votre application. Par conséquent, le contrôleur laisse la vue intacte et fait son travail. Dans une directive, la fonction de liaison est le meilleur endroit pour manipuler le DOM. Il a un accès complet à la portée et à l'élément. En utilisant une directive, vous pouvez également profiter de la possibilité de réutilisation.

```
link: function($scope, element, attrs) {
    //The best place to manipulate DOM
}
```

Vous pouvez accéder aux éléments DOM dans la fonction de liaison de plusieurs manières, telles que le paramètre `element` , la méthode `angular.element()` ou le Javascript pur.

2. Liaison de données en transclusion

AngularJS est célèbre pour sa liaison de données bidirectionnelle. Cependant, il se peut que vous rencontriez parfois que vos données ne soient liées que dans un sens à l'intérieur des directives. Arrêtez-vous là, AngularJS n'a pas tort mais probablement vous. Les directives sont des endroits un peu dangereux car des champs d'enfants et des portées isolées sont impliqués. Supposons que vous ayez la directive suivante avec une transclusion

```
<my-dir>
    <my-transclusion>
    </my-transclusion>
</my-dir>
```

Et dans my-transclusion, vous avez des éléments liés aux données de la portée externe.

```
<my-dir>
    <my-transclusion>
        <input ng-model="name">
    </my-transclusion>
</my-dir>
```

Le code ci-dessus ne fonctionnera pas correctement. Ici, la transclusion crée une étendue enfant et vous pouvez obtenir la variable name, mais tout changement apporté à cette variable restera là. Donc, vous pouvez vraiment accéder à cette variable en tant que **$ parent.name** . Cependant, cette utilisation peut ne pas être la meilleure pratique. Une meilleure approche serait d'encapsuler

les variables à l'intérieur d'un objet. Par exemple, dans le contrôleur, vous pouvez créer:

```
$scope.data = {
    name: 'someName'
}
```

Ensuite, dans la transclusion, vous pouvez accéder à cette variable via un objet 'data' et voir que la liaison bidirectionnelle fonctionne parfaitement!

```
<input ng-model="data.name">
```

Non seulement dans les transclusions, mais dans toute l'application, c'est une bonne idée d'utiliser la notation en pointillés.

3. directives multiples ensemble

Il est en effet légal d'utiliser deux directives dans le même élément, à condition que vous obéissiez à la règle: deux portées isolées ne peuvent pas exister sur le même élément. D'une manière générale, lors de la création d'une nouvelle directive personnalisée, vous allouez une étendue isolée pour faciliter le passage des paramètres. En supposant que les directives myDirA et myDirB ont isolé les portées et que myDirC ne l’a pas, l’élément suivant sera valide:

```
<input my-dir-a my-dirc>
```

alors que l'élément suivant provoquera une erreur de console:

```
<input my-dir-a my-dir-b>
```

Par conséquent, les directives doivent être utilisées à bon escient, en tenant compte des domaines d'application.

4. Mauvaise utilisation de $ emit

$ emit, $ broadcast et $ on, ils fonctionnent dans un principe émetteur-récepteur. En d'autres termes, ils constituent un moyen de communication entre contrôleurs. Par exemple, la ligne suivante émet le 'someEvent' du contrôleur A, qui sera capturé par le contrôleur concerné B.

```
$scope.$emit('someEvent', args);
```

Et la ligne suivante attire le «someEvent»

```
$scope.$on('someEvent', function(){});
```

Jusqu'à présent, tout semble parfait. Mais rappelez-vous que si le contrôleur B n'est pas encore appelé, l'événement ne sera pas intercepté, ce qui signifie que les contrôleurs d'émetteur et de récepteur doivent être appelés pour que cela fonctionne. Encore une fois, si vous n'êtes pas certain de devoir utiliser $ emit, la création d'un service semble être une meilleure solution.

5. Mauvaise utilisation de $ scope. $ Watch

$ scope. $ watch est utilisé pour regarder un changement de variable. Chaque fois qu'une variable a changé, cette méthode est appelée. Cependant, une erreur courante est de changer la variable dans $ scope. $ Watch. Cela provoquera une incohérence et une boucle $ digest infinie à un moment donné.

```
$scope.$watch('myCtrl.myVariable', function(newVal) {
    this.myVariable++;
});
```

Donc, dans la fonction ci-dessus, assurez-vous de ne pas avoir d'opérations sur myVariable et newVal.

6. Méthodes de liaison aux vues

C'est l'un des péchés les plus mortels. AngularJS a une liaison bidirectionnelle et chaque fois que quelque chose change, les vues sont mises à jour plusieurs fois. Donc, si vous liez une méthode à un attribut d'une vue, cette méthode peut potentiellement être appelée cent fois, ce qui vous rend également fou lors du débogage. Cependant, seuls certains attributs sont créés pour la liaison de méthode, tels que ng-click, ng-blur, ng-on-change, etc., qui attendent des méthodes comme paramètres. Par exemple, supposons que vous ayez la vue suivante dans votre balisage:

```
<input ng-disabled="myCtrl.isDisabled()" ng-model="myCtrl.name">
```

Ici, vous vérifiez l'état désactivé de la vue via la méthode isDisabled. Dans le contrôleur myCtrl, vous avez:

```
vm.isDisabled = function(){
    if(someCondition)
        return true;
    else
        return false;
}
```

En théorie, cela peut sembler correct, mais techniquement, cela entraînera une surcharge, car la méthode fonctionnera d'innombrables fois. Pour résoudre ce problème, vous devez lier une variable. Dans votre contrôleur, la variable suivante doit exister:

```
vm.isDisabled
```

Vous pouvez relancer cette variable lors de l'activation du contrôleur

```
if(someCondition)
    vm.isDisabled = true
else
    vm.isDisabled = false
```

Si la condition n'est pas stable, vous pouvez lier ceci à un autre événement. Ensuite, vous devez

lier cette variable à la vue:

```
<input ng-disabled="myCtrl.isDisabled" ng-model="myCtrl.name">
```

Maintenant, tous les attributs de la vue ont ce qu'ils attendent et les méthodes ne seront exécutées que lorsque cela est nécessaire.

7. Ne pas utiliser les fonctionnalités d'Angular

AngularJS offre une grande commodité avec certaines de ses fonctionnalités, non seulement pour simplifier votre code, mais aussi pour le rendre plus efficace. Certaines de ces fonctionnalités sont énumérées ci-dessous:

1. **angular.forEach** pour les boucles (Attention, vous ne pouvez pas "casser", vous ne pouvez que vous empêcher d'entrer dans le corps, alors pensez à la performance ici.)
2. **angular.element** pour les sélecteurs DOM
3. **angular.copy** : Utilisez ceci lorsque vous ne devez pas modifier l'objet principal
4. **Les validations de formulaires** sont déjà géniales. Utilisez sale, vierge, touché, valide, requis, etc.
5. Outre le débogueur Chrome, utilisez également **le débogage à distance** pour le développement mobile.
6. Et assurez-vous d'utiliser **Batarang** . C'est une extension gratuite de Chrome où vous pouvez facilement inspecter les étendues
   .

Lire AngularJS des pièges et des pièges en ligne:
https://riptutorial.com/fr/angularjs/topic/3208/angularjs-des-pieges-et-des-pieges

Chapitre 5: Chargement paresseux

Remarques

1. Si vos dépendances chargées paresseuses nécessitent d’autres dépendances chargées paresseuses, assurez-vous de les charger dans le bon ordre!

```
angular.module('lazy', [
 'alreadyLoadedDependency1',
 'alreadyLoadedDependency2',
 ...
 {
  files: [
  'path/to/lazily/loaded/dependency1.js',
  'path/to/lazily/loaded/dependency2.js', //<--- requires lazily loaded dependency1
  'path/to/lazily/loaded/dependency.css'
  ],
  serie: true //Sequential load instead of parallel
 }
]);
```

Examples

Préparer votre projet pour un chargement paresseux

Après avoir inclus `oclazyload.js` dans votre fichier d'index, déclarez `ocLazyLoad` comme une dépendance dans `app.js`

```
//Make sure you put the correct dependency! it is spelled different than the service!
angular.module('app', [
 'oc.lazyLoad',
 'ui-router'
])
```

Usage

Pour charger des fichiers paresseux, injectez le service `$ocLazyLoad` dans un contrôleur ou un autre service

```
.controller('someCtrl', function($ocLazyLoad) {
 $ocLazyLoad.load('path/to/file.js').then(...);
});
```

Les modules angulaires seront automatiquement chargés en angulaire.

Autre variante:

```
$ocLazyLoad.load([
  'bower_components/bootstrap/dist/js/bootstrap.js',
```

```
  'bower_components/bootstrap/dist/css/bootstrap.css',
  'partials/template1.html'
]);
```

Pour une liste complète des variantes, visitez la documentation officielle

Utilisation avec routeur

UI-Router:

```
.state('profile', {
 url: '/profile',
 controller: 'profileCtrl as vm'
 resolve: {
  module: function($ocLazyLoad) {
   return $ocLazyLoad.load([
    'path/to/profile/module.js',
    'path/to/profile/style.css'
   ]);
  }
 }
});
```

ngRoute:

```
.when('/profile', {
     controller: 'profileCtrl as vm'
     resolve: {
      module: function($ocLazyLoad) {
       return $ocLazyLoad.load([
        'path/to/profile/module.js',
        'path/to/profile/style.css'
       ]);
      }
     }
    });
```

Utiliser l'injection de dépendance

La syntaxe suivante vous permet de spécifier des dépendances dans votre `module.js` au lieu d'une spécification explicite lors de l'utilisation du service

```
//lazy_module.js
angular.module('lazy', [
 'alreadyLoadedDependency1',
 'alreadyLoadedDependency2',
 ...
 [
  'path/to/lazily/loaded/dependency.js',
  'path/to/lazily/loaded/dependency.css'
 ]
]);
```

Note : cette syntaxe ne fonctionnera que pour les modules chargés paresseusement!

Utiliser la directive

```
<div oc-lazy-load="['path/to/lazy/loaded/directive.js',
'path/to/lazy/loaded/directive.html']">

<!-- myDirective available here -->
<my-directive></my-directive>

</div>
```

Lire Chargement paresseux en ligne: https://riptutorial.com/fr/angularjs/topic/6400/chargement-paresseux

Chapitre 6: Comment fonctionne la liaison de données

Remarques

Donc, bien que ce concept de liaison de données soit dans l'ensemble facile pour le développeur, il est assez lourd pour le navigateur puisque Angular écoute tous les changements d'événements et exécute le cycle Digest. Pour cette raison, chaque fois que nous attachons un modèle à la vue, assurez-vous que Scope est aussi optimisé que possible.

Examples

Exemple de liaison de données

```
<p ng-bind="message"></p>
```

Ce "message" doit être associé à la portée du contrôleur d'éléments en cours.

```
$scope.message = "Hello World";
```

Plus tard, même si le modèle de message est mis à jour, cette valeur mise à jour est reflétée dans l'élément HTML. Lorsque angular compile, le template "Hello World" sera attaché à innerHTML du monde actuel. Angular maintient un mécanisme de surveillance de toutes les directives attachées à la vue. Il a un mécanisme de cycle de digestion où il itère à travers le tableau Watchers, il mettra à jour l'élément DOM s'il y a un changement dans la valeur précédente du modèle.

Il n'y a pas de vérification périodique de Scope s'il y a un changement dans les objets qui lui sont attachés. Tous les objets attachés à la portée ne sont pas surveillés. Scope maintient prototypiquement un **$$ WatchersArray** . Scope ne parcourt que cet objet WatchersArray lorsque $ digest est appelé.

Angular ajoute un observateur au WatchersArray pour chacun de ces

1. {{expression}} - Dans vos modèles (et partout où il y a une expression) ou lorsque nous définissons ng-model.
2. $ scope. $ watch ('expression / function') - Dans votre JavaScript, nous pouvons simplement attacher un objet scope à angular à regarder.

La fonction **$ watch** prend en trois paramètres:

1. La première est une fonction d'observation qui renvoie simplement l'objet ou nous pouvons simplement ajouter une expression.

2. La seconde est une fonction d'écoute qui sera appelée en cas de changement

d'objet. Toutes les choses comme les modifications DOM seront implémentées dans cette fonction.

3. Le troisième étant un paramètre facultatif qui prend un booléen. Si c'est vrai, la profondeur angulaire regarde l'objet et si son faux Angular fait juste une référence en regardant l'objet. La mise en œuvre brutale de $ watch ressemble à ceci

```
Scope.prototype.$watch = function(watchFn, listenerFn) {
   var watcher = {
       watchFn: watchFn,
       listenerFn: listenerFn || function() { },
       last: initWatchVal  // initWatchVal is typically undefined
   };
   this.$$watchers.push(watcher); // pushing the Watcher Object to Watchers
};
```

Il y a une chose intéressante dans Angular appelé Digest Cycle. Le cycle $ digest commence à la suite d'un appel à $ scope. $ Digest (). Supposons que vous modifiez un modèle $ scope dans une fonction de gestionnaire via la directive ng-click. Dans ce cas, AngularJS déclenche automatiquement un cycle $ digest en appelant $ digest (). En plus de ng-click, il existe plusieurs autres directives / services intégrés qui vous permettent de changer de modèle (par exemple, ng-model, $ timeout, etc.) et déclenche automatiquement un cycle de digestion. L'implémentation brute de $ digest ressemble à ceci.

```
Scope.prototype.$digest = function() {
      var dirty;
      do {
          dirty = this.$$digestOnce();
      } while (dirty);
}
Scope.prototype.$$digestOnce = function() {
   var self = this;
   var newValue, oldValue, dirty;
   _.forEach(this.$$watchers, function(watcher) {
          newValue = watcher.watchFn(self);
          oldValue = watcher.last;   // It just remembers the last value for dirty checking
          if (newValue !== oldValue) { //Dirty checking of References
   // For Deep checking the object , code of Value
   // based checking of Object should be implemented here
             watcher.last = newValue;
             watcher.listenerFn(newValue,
                  (oldValue === initWatchVal ? newValue : oldValue),
                   self);
          dirty = true;
          }
     });
   return dirty;
 };
```

Si nous utilisons la fonction **setTimeout ()** de JavaScript pour mettre à jour un modèle de portée, Angular n'a aucun moyen de savoir ce que vous pourriez changer. Dans ce cas, il est de notre responsabilité d'appeler manuellement $ apply (), ce qui déclenche un cycle $ digest. De même, si vous avez une directive qui configure un écouteur d'événement DOM et modifie certains modèles

à l'intérieur de la fonction de gestionnaire, vous devez appeler $ apply () pour vous assurer que les modifications prennent effet. La grande idée de $ apply est que nous pouvons exécuter du code qui n’est pas au courant d’Angular, ce code pouvant encore changer les choses sur la portée. Si nous encapsulons ce code dans $ apply, il se chargera d'appeler $ digest ().
Implémentation approximative de $ apply ().

```
Scope.prototype.$apply = function(expr) {
       try {
         return this.$eval(expr); //Evaluating code in the context of Scope
       } finally {
         this.$digest();
       }
};
```

Lire Comment fonctionne la liaison de données en ligne:
https://riptutorial.com/fr/angularjs/topic/2342/comment-fonctionne-la-liaison-de-donnees

Chapitre 7: Composants

Paramètres

Paramètre	Détails
=	Pour utiliser la liaison de données bidirectionnelle. Cela signifie que si vous mettez à jour cette variable dans la portée de votre composant, la modification sera répercutée sur la portée parent.
<	Liaisons unidirectionnelles lorsque nous voulons simplement lire une valeur d'une étendue parent et ne pas la mettre à jour.
@	Paramètres de chaîne.
Et	Pour les rappels au cas où votre composant aurait besoin de générer quelque chose dans son étendue parent.
-	-
Crochets LifeCycle	**Détails** (nécessite angular.version> = 1.5.3)
$ onInit ()	Appelé sur chaque contrôleur après que tous les contrôleurs d'un élément aient été construits et que leurs liaisons ont été initialisées. C'est un bon endroit pour mettre le code d'initialisation pour votre contrôleur.
$ onChanges (changesObj)	Appelé à chaque mise à jour des liaisons unidirectionnelles. Le `changesObj` est un hachage dont les clés sont les noms des propriétés liées qui ont été modifiées et les valeurs sont un objet de la forme `{ currentValue, previousValue, isFirstChange() }` .
$ onDestroy ()	Appelé sur un contrôleur lorsque son étendue contenant est détruite. Utilisez ce hook pour libérer des ressources externes, des montres et des gestionnaires d'événements.
$ postLink ()	Appelé après que l'élément de ce contrôleur et ses enfants ont été liés. Ce hook peut être considéré comme analogue aux hooks ngAfterViewInit et ngAfterContentInit dans Angular 2.
$ doCheck ()	Appelé à chaque tour du cycle de digestion. Fournit une opportunité pour détecter et agir sur les changements. Toutes les actions que vous souhaitez entreprendre en réponse aux modifications que vous détectez doivent être appelées à partir de ce hook; implémenter ceci n'a aucun effet sur l'appel de $ onChanges.

Remarques

Le composant est un type particulier de directive qui utilise une configuration plus simple adaptée à une structure d'application basée sur des composants. Les composants ont été introduits dans Angular 1.5, les exemples de cette section **ne fonctionneront pas** avec les anciennes versions d'AngularJS.

Un guide complet du développeur sur les composants est disponible sur https://docs.angularjs.org/guide/component

Examples

Composants de base et crochets LifeCycle

Qu'est-ce qu'un composant?

- Un composant est fondamentalement une directive qui utilise une configuration plus simple et qui convient à une architecture basée sur des composants, ce qui est l'objet d'Angular 2. Considérez un composant comme un widget: un morceau de code HTML que vous pouvez réutiliser à différents endroits de votre application Web.

Composant

```
angular.module('myApp', [])
    .component('helloWorld', {
        template: '<span>Hello World!</span>'
    });
```

Balisage

```
<div ng-app="myApp">
  <hello-world> </hello-world>
</div>
```

Démo en direct

Utilisation de données externes dans le composant:

Nous pourrions ajouter un paramètre pour passer un nom à notre composant, qui serait utilisé comme suit:

```
angular.module("myApp", [])
  .component("helloWorld",{
    template: '<span>Hello {{$ctrl.name}}!</span>',
    bindings: { name: '@' }
```

```
  });
```

Balisage

```
<div ng-app="myApp">
  <hello-world name="'John'" > </hello-world>
</div>
```

Démo en direct

Utilisation des contrôleurs dans les composants

Jetons un coup d'œil à la façon d'ajouter un contrôleur.

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      controller: function(){
        this.myName = 'Alain';
      }
  });
```

Balisage

```
<div ng-app="myApp">
  <hello-world name="John"> </hello-world>
</div>
```

CodePen Demo

Les paramètres transmis au composant sont disponibles dans la portée du contrôleur juste avant `$onInit` fonction `$onInit` ne soit appelée par Angular. Considérez cet exemple:

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      controller: function(){
        this.$onInit = function() {
          this.myName = "Mac" + this.name;
        }
      }
  });
```

Dans le modèle ci-dessus, cela rendrait "Bonjour John, je suis MacJohn!".

Notez que `$ctrl` est la valeur par défaut angulaire des `controllerAs` si aucun n'est spécifié.

Démo en direct

Utiliser "require" comme objet

Dans certains cas, vous devrez peut-être accéder aux données d'un composant parent à l'intérieur de votre composant.

Cela peut être réalisé en spécifiant que notre composant nécessite ce composant parent, le requis nous donnera une référence au contrôleur de composant requis, qui peut ensuite être utilisé dans notre contrôleur comme indiqué dans l'exemple ci-dessous:

> Notez que les contrôleurs requis sont garantis prêts uniquement après le hook $ onInit.

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      require: {
        parent: '^parentComponent'
      },
      controller: function () {
        // here this.parent might not be initiated yet

        this.$onInit = function() {
           // after $onInit, use this.parent to access required controller
           this.parent.foo();
        }

      }
  });
```

N'oubliez pas que cela crée un lien étroit entre l'enfant et le parent.

Composants en JS angulaire

Les composants d'angularJS peuvent être visualisés sous la forme d'une directive personnalisée (<html> dans une directive HTML, et quelque chose comme ceci sera une directive personnalisée <ANYTHING>). Un composant contient une vue et un contrôleur. Le contrôleur contient la logique métier liée à une vue, que l'utilisateur voit. Le composant diffère d'une directive angulaire car il contient moins de configuration. Une composante angulaire peut être définie comme ceci.

```
angular.module("myApp",[]).component("customer", {})
```

Les composants sont définis sur les modules angulaires. Ils contiennent deux arguments, One est le nom du composant et le second est un objet qui contient une paire de valeurs-clés, qui définit quelle vue et quel contrôleur il va utiliser comme ceci.

```
angular.module("myApp",[]).component("customer", {
    templateUrl : "customer.html", // your view here
    controller: customerController, //your controller here
    controllerAs: "cust"        //alternate name for your controller
})
```

"myApp" est le nom de l'application que nous construisons et le client est le nom de notre composant. Maintenant, pour l'appeler dans le fichier HTML principal, nous allons le mettre comme ça

```
<customer></customer>
```

Maintenant, cette directive sera remplacée par la vue que vous avez spécifiée et la logique métier que vous avez écrite dans votre contrôleur.

REMARQUE: Rappelez que le composant prend un objet comme second argument tandis que directive prend une fonction de fabrique comme argument.

Lire Composants en ligne: https://riptutorial.com/fr/angularjs/topic/892/composants

Chapitre 8: Contrôleurs

Syntaxe

- <htmlElement ng-controller = "controllerName"> ... </ htmlElement>
- <script> app.controller ('controllerName', controllerFunction); </ script>

Examples

Votre premier contrôleur

Un contrôleur est une structure de base utilisée dans Angular pour préserver la portée et gérer certaines actions dans une page. Chaque contrôleur est couplé à une vue HTML.

Vous trouverez ci-dessous un modèle de base pour une application angulaire:

```
<!DOCTYPE html>

<html lang="en" ng-app='MyFirstApp'>
    <head>
        <title>My First App</title>

        <!-- angular source -->
        <script src="https://code.angularjs.org/1.5.3/angular.min.js"></script>

        <!-- Your custom controller code -->
        <script src="js/controllers.js"></script>
    </head>
    <body>
        <div ng-controller="MyController as mc">
            <h1>{{ mc.title }}</h1>
            <p>{{ mc.description }}</p>
            <button ng-click="mc.clicked()">
                Click Me!
            </button>
        </div>
    </body>
</html>
```

Il y a quelques choses à noter ici:

```
<html ng-app='MyFirstApp'>
```

Définir le nom de l'application avec `ng-app` vous permet d'accéder à l'application dans un fichier Javascript externe, qui sera traité ci-dessous.

```
<script src="js/controllers.js"></script>
```

Nous avons besoin d'un fichier Javascript où vous définissez vos contrôleurs et leurs actions /

données.

```
<div ng-controller="MyController as mc">
```

L'attribut `ng-controller` définit le contrôleur pour cet élément DOM et tous les éléments qui sont enfants (récursivement) en dessous.

Vous pouvez avoir plusieurs du même contrôleur (dans ce cas, `MyController`) en disant `... as mc` , nous donnons à cette instance du contrôleur un alias.

```
<h1>{{ mc.title }}</h1>
```

La notation `{{ ... }}` est une expression angulaire. Dans ce cas, cela définit le texte interne de cet élément `<h1>` à la valeur de `mc.title` .

Remarque: Angular utilise la liaison de données bidirectionnelle, ce qui signifie que, quelle que soit la manière dont vous mettez à jour la valeur `mc.title` , celle-ci sera reflétée dans le contrôleur et la page.

Notez également que les expressions angulaires *ne* doivent *pas* faire référence à un contrôleur. Une expression angulaire peut être aussi simple que `{{ 1 + 2 }}` ou `{{ "Hello " + "World" }}` .

```
<button ng-click="mc.clicked()">
```

`ng-click` est une directive angulaire, liant dans ce cas l'événement click pour que le bouton déclenche la fonction `clicked()` de l'instance `MyController` .

En gardant cela à l'esprit, écrivons une implémentation du contrôleur `MyController` . Avec l'exemple ci-dessus, vous écrivez ce code dans `js/controller.js` .

Tout d'abord, vous devez instancier l'application Angular dans votre Javascript.

```
var app = angular.module("MyFirstApp", []);
```

Notez que le nom que nous transmettons ici est identique au nom que vous avez défini dans votre code HTML avec la directive `ng-app` .

Maintenant que nous avons l'objet app, nous pouvons l'utiliser pour créer des contrôleurs.

```
app.controller('MyController', function(){
    var ctrl = this;

    ctrl.title = "My First Angular App";
    ctrl.description = "This is my first Angular app!";

    ctrl.clicked = function(){
        alert("MyController.clicked()");
    };
});
```

Remarque: Pour tout ce que nous voulons faire partie de l'instance du contrôleur, nous utilisons le mot `this` clé `this` .

C'est tout ce qui est nécessaire pour construire un contrôleur simple.

Création de contrôleurs

```
angular
    .module('app')
    .controller('SampleController', SampleController)

SampleController.$inject = ['$log', '$scope'];
function SampleController($log, $scope){
    $log.debug('*****SampleController******');

    /* Your code below */
}
```

Note `.$inject` s'assurera que vos dépendances ne seront pas brouillées après minification. Aussi, assurez-vous qu'il est en ordre avec la fonction nommée.

Créer des contrôleurs, sécuriser la minification

Il existe différentes manières de protéger la création de votre contrôleur contre la minification.

La première est appelée annotation de tableau en ligne. Cela ressemble à ceci:

```
var app = angular.module('app');
app.controller('sampleController', ['$scope', '$http', function(a, b){
    //logic here
}]);
```

Le second paramètre de la méthode du contrôleur peut accepter un tableau de dépendances. Comme vous pouvez le voir, j'ai défini `$scope` et `$http` qui devraient correspondre aux paramètres de la fonction du contrôleur dans laquelle `a` sera la `$scope` et `b` serait `$http` . Notez que le dernier élément du tableau doit être votre fonction de contrôleur.

La seconde option utilise la propriété `$inject` . Cela ressemble à ceci:

```
var app = angular.module('app');
app.controller('sampleController', sampleController);
sampleController.$inject = ['$scope', '$http'];
function sampleController(a, b) {
    //logic here
}
```

Cela fait la même chose que les annotations de tableau en ligne mais offre un style différent pour ceux qui préfèrent une option à une autre.

L'ordre des dépendances injectées est important

Lorsque vous injectez des dépendances à l'aide du tableau, assurez-vous que la liste des dépendances correspond à la liste d'arguments correspondante transmise à la fonction de contrôleur.

Notez que dans l'exemple suivant, `$scope` et `$http` sont inversés. Cela provoquera un problème dans le code.

```
// Intentional Bug: injected dependencies are reversed which will cause a problem
app.controller('sampleController', ['$scope', '$http',function($http, $scope) {
    $http.get('sample.json');
}]);
```

Utilisation de contrôleurs dans JS angulaire

Dans Angular `$scope` est le lien entre le contrôleur et la vue qui aide à tous nos besoins de liaison de données. Controller As est une autre manière de lier le contrôleur et la vue et est principalement recommandé d'utiliser. Fondamentalement, ce sont les deux constructions de contrôleur dans Angular (c'est-à-dire $ scope et Controller As).

Différentes manières d'utiliser Controller as sont -

controllerAs Voir la syntaxe

```
<div ng-controller="CustomerController as customer">
    {{ customer.name }}
</div>
```

Contrôleur Syntaxe du contrôleur

```
function CustomerController() {
    this.name = {};
    this.sendMessage = function() { };
}
```

contrôleurs avec vm

```
function CustomerController() {
    /*jshint validthis: true */
    var vm = this;
    vm.name = {};
    vm.sendMessage = function() { };
}
```

`controllerAs` sont des sucres syntaxiques supérieurs `$scope` . Vous pouvez toujours vous lier aux méthodes View et toujours accéder à `$scope` . L'utilisation des `controllerAs` constitue l'une des meilleures pratiques suggérées par l'équipe centrale. Il y a beaucoup de raisons à cela, peu d'entre eux sont -

- `$scope` expose les membres du contrôleur à la vue via un objet intermédiaire. En définissant `this.*` , Nous pouvons exposer exactement ce que nous voulons exposer du contrôleur à la

vue. Il suit également la méthode JavaScript standard utilisée pour cela.

- En utilisant la syntaxe `controllerAs` , nous avons un code plus lisible et la propriété parent est accessible en utilisant le nom d'alias du contrôleur parent au lieu d'utiliser la syntaxe `$parent` .
- Il encourage l'utilisation de la liaison à un objet "pointillé" dans View (par exemple, customer.name au lieu de name), qui est plus contextuel, plus facile à lire et évite tout problème de référence pouvant survenir sans "dotting".
- Permet d'éviter l'utilisation d'appels `$parent` dans les vues avec des contrôleurs imbriqués.
- Utilisez une variable de capture pour cela lorsque vous utilisez la syntaxe `controllerAs` . Choisissez un nom de variable cohérent tel que `vm` , qui signifie ViewModel. Parce que `this` mot-clé est contextuel et, lorsqu'il est utilisé dans une fonction à l'intérieur d'un contrôleur, il peut changer de contexte. Capturer le contexte de ceci évite de rencontrer ce problème.

REMARQUE: l' utilisation de la syntaxe `controllerAs` ajoute à la référence de portée actuelle au contrôleur actuel, de sorte qu'elle soit disponible en tant que champ

```
<div ng-controller="Controller as vm>...</div>
```

`vm` est disponible sous la forme `$scope.vm` .

Création de contrôleurs angulaires sécuritaires

Pour créer des contrôleurs angulaires sans danger pour la minification, vous allez modifier les paramètres de la fonction du `controller` .

Le second argument de la fonction `module.controller` doit être passé à un **tableau** , où le **dernier paramètre** est la **fonction de contrôleur** , et chaque paramètre avant est le **nom** de chaque valeur injectée.

Ceci est différent du paradigme normal; cela prend la **fonction de contrôleur** avec les arguments injectés.

Donné:

```
var app = angular.module('myApp');
```

Le contrôleur devrait ressembler à ceci:

```
app.controller('ctrlInject',
    [
        /* Injected Parameters */
        '$Injectable1',
        '$Injectable2',
        /* Controller Function */
        function($injectable1Instance, $injectable2Instance) {
            /* Controller Content */
```

```
        }
    ]
);
```

Note: *Les noms des paramètres injectés ne sont pas obligés de correspondre, mais ils seront liés dans l'ordre.*

Cela se réduira à quelque chose de similaire à ceci:

```
var
a=angular.module('myApp');a.controller('ctrlInject',['$Injectable1','$Injectable2',function(b,c){/*
Controller Content */}]);
```

Le processus de minification remplacera chaque instance d' `app` par `a` , chaque instance de `$Injectable1Instance` avec `b` , et chaque instance de `$Injectable2Instance` avec `c` .

Contrôleurs imbriqués

Les contrôleurs d'imbrication enchaînent également la `$scope` . La modification d'une variable `$scope` dans le contrôleur imbriqué modifie la même variable `$scope` dans le contrôleur parent.

```
.controller('parentController', function ($scope) {
    $scope.parentVariable = "I'm the parent";
});

.controller('childController', function ($scope) {
    $scope.childVariable = "I'm the child";

    $scope.childFunction = function () {
        $scope.parentVariable = "I'm overriding you";
    };
});
```

Maintenant, essayons de les gérer tous les deux, imbriqués.

```
<body ng-controller="parentController">
    What controller am I? {{parentVariable}}
    <div ng-controller="childController">
        What controller am I? {{childVariable}}
        <button ng-click="childFunction()"> Click me to override! </button>
    </div>
</body>
```

Les contrôleurs d'imbrication peuvent avoir leurs avantages, mais une chose doit être prise en compte. L'appel de la directive `ngController` crée une nouvelle instance du contrôleur - ce qui peut souvent créer de la confusion et des résultats inattendus.

Lire Contrôleurs en ligne: https://riptutorial.com/fr/angularjs/topic/601/controleurs

Chapitre 9: Contrôleurs avec ES6

Examples

Manette

Il est très facile d'écrire un contrôleur angularJS avec ES6 si vous êtes familiarisé avec la **programmation orientée objet** :

```
class exampleContoller{

       constructor(service1,service2,...serviceN){
          let ctrl=this;
              ctrl.service1=service1;
              ctrl.service2=service2;
               .
               .
               .
              ctrl.service1=service1;
              ctrl.controllerName = 'Example Controller';
              ctrl.method1(controllerName)

       }

       method1(param){
          let ctrl=this;
               ctrl.service1.serviceFunction();
              .
              .
              ctrl.scopeName=param;
       }
       .
       .
       .
       methodN(param){
          let ctrl=this;
               ctrl.service1.serviceFunction();
              .
              .
       }


   }
   exampleContoller.$inject = ['service1','service2',...,'serviceN'];
   export default exampleContoller;
```

Lire Contrôleurs avec ES6 en ligne: https://riptutorial.com/fr/angularjs/topic/9419/controleurs-avec-es6

Chapitre 10: Décorateurs

Syntaxe

- décorateur (nom, décorateur);

Remarques

> **Decorator** est une fonction qui permet de modifier un service , une fabrique , une directive ou un filtre avant son utilisation. Decorator est utilisé pour remplacer ou modifier le comportement du service. La valeur de retour de la fonction décorateur peut être le service d'origine ou un nouveau service qui remplace ou encapsule et délègue le service d'origine.

Toute décoration ***doit être*** faite dans la phase de `config` l'application angulaire en injectant `$provide` et en utilisant la fonction `$provide.decorator` .

> La fonction décorateur a un objet `$delegate` injecté pour fournir un accès au service correspondant au sélecteur du décorateur. Ce `$delegate` sera le service que vous décorez. La valeur de retour de la fonction fournie au décorateur prendra la place du service, de la directive ou du filtre en cours de décoration.

On devrait envisager d'utiliser un décorateur uniquement si une autre approche n'est pas appropriée ou s'avère trop fastidieuse. Si les applications volumineuses utilisent le même service et que l'une d'entre elles modifie le comportement du service, il est facile de créer de la confusion et / ou des bogues dans le processus.

Un cas d'utilisation typique serait lorsque vous avez une dépendance à une tierce partie que vous ne pouvez pas mettre à niveau mais que vous avez besoin de travailler différemment ou de l'étendre.

Examples

Décorer service, usine

Ci - dessous exemple de décorateur de service, remplaçant `null` date renvoyée par le service.

```
angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('myService', function($delegate) {
      $delegate.getDate = function() { // override with actual date object
        return new Date();
      };
      return $delegate;
    });
```

```
  })
  .service('myService', function() {
    this.getDate = function() {
      return null; // w/o decoration we'll be returning null
    };
  })
  .controller('myController', function(myService) {
    var vm = this;
    vm.date = myService.getDate();
  });
```

```
<body ng-controller="myController as vm">
  <div ng-bind="vm.date | date:'fullDate'"></div>
</body>
```

> Saturday, August 6, 2016

Directive décorer

Les directives peuvent être décorées comme les services et nous pouvons modifier ou remplacer n'importe laquelle de ses fonctionnalités. Notez que la directive elle-même est accessible à la position 0 dans le tableau $ delegate et que le paramètre name dans le décorateur doit inclure le suffixe `Directive` (sensible à la casse).

Donc, si directive s'appelle `myDate` , on peut y accéder en utilisant `myDateDirective` utilisant `$delegate[0]` .

Voici un exemple simple où directive indique l'heure actuelle. Nous allons le décorer pour mettre à jour l'heure actuelle à intervalles d'une seconde. Sans décoration, il affichera toujours la même heure.

```
<body>
  <my-date></my-date>
</body>
```

```
angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('myDateDirective', function($delegate, $interval) {
      var directive = $delegate[0]; // access directive

      directive.compile = function() { // modify compile fn
        return function(scope) {
          directive.link.apply(this, arguments);
          $interval(function() {
            scope.date = new Date(); // update date every second
          }, 1000);
        };
      };

      return $delegate;
    });
```

```
  })
  .directive('myDate', function() {
    return {
      restrict: 'E',
      template: '<span>Current time is {{ date | date:\'MM:ss\' }}</span>',
      link: function(scope) {
        scope.date = new Date(); // get current date
      }
    };
  });
```

> Current time is 08:33

Décorer le filtre

Lors de la décoration des filtres, le paramètre name doit inclure le suffixe `Filter` (sensible à la casse). Si le filtre est appelé `repeat` , le paramètre decorator est `repeatFilter` . Ci-dessous, nous allons décorer un filtre personnalisé qui répète une chaîne donnée *n* fois, afin que le résultat soit inversé. Vous pouvez également décorer les filtres intégrés angulaires de la même manière, bien que cela ne soit pas recommandé, car cela peut affecter les fonctionnalités du framework.

```
<body>
  <div ng-bind="'i can haz cheeseburger ' | repeat:2"></div>
</body>

angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('repeatFilter', function($delegate) {
      return function reverse(input, count) {
        // reverse repeated string
        return ($delegate(input, count)).split('').reverse().join('');
      };
    });
  })
  .filter('repeat', function() {
    return function(input, count) {
      // repeat string n times
      return (input || '').repeat(count || 1);
    };
  });
```

> i can haz cheeseburger i can haz cheeseburger

> regrubeseehc zah nac i regrubeseehc zah nac i

Lire Décorateurs en ligne: https://riptutorial.com/fr/angularjs/topic/5255/decorateurs

Chapitre 11: Demande $ http

Examples

Utiliser $ http dans un contrôleur

Le service `$http` est une fonction qui génère une requête HTTP et renvoie une promesse.

Usage général

```
// Simple GET request example:
$http({
  method: 'GET',
  url: '/someUrl'
}).then(function successCallback(response) {
    // this callback will be called asynchronously
    // when the response is available
  }, function errorCallback(response) {
    // called asynchronously if an error occurs
    // or server returns response with an error status.
  });
```

Utilisation à l'intérieur du contrôleur

```
appName.controller('controllerName',
    ['$http', function($http){

    // Simple GET request example:
    $http({
        method: 'GET',
        url: '/someUrl'
    }).then(function successCallback(response) {
        // this callback will be called asynchronously
        // when the response is available
    }, function errorCallback(response) {
        // called asynchronously if an error occurs
        // or server returns response with an error status.
    });
}])
```

Méthodes de raccourci

`$http` service `$http` a également des méthodes de raccourci. Lisez à propos des méthodes http ici

Syntaxe

```
$http.get('/someUrl', config).then(successCallback, errorCallback);
$http.post('/someUrl', data, config).then(successCallback, errorCallback);
```

Méthodes de raccourci

- $ http.get

- $ http.head
- $ http.post
- $ http.put
- $ http.delete
- $ http.jsonp
- $ http.patch

Utiliser la requête $ http dans un service

Les requêtes HTTP sont largement utilisées de manière répétée sur chaque application Web. Il est donc judicieux d'écrire une méthode pour chaque requête commune, puis de l'utiliser à plusieurs endroits dans l'application.

Créez un `httpRequestsService.js`

httpRequestsService.js

```
appName.service('httpRequestsService', function($q, $http){

    return {
        // function that performs a basic get request
        getName: function(){
            // make sure $http is injected
            return $http.get("/someAPI/names")
                .then(function(response) {
                    // return the result as a promise
                    return response;
                }, function(response) {
                    // defer the promise
                    return $q.reject(response.data);
                });
        },

        // add functions for other requests made by your app
        addName: function(){
            // some code...
        }
    }
})
```

Le service ci-dessus effectuera une demande d'obtention à l'intérieur du service. Ce sera disponible à tout contrôleur où le service a été injecté.

Utilisation de l'échantillon

```
appName.controller('controllerName',
    ['httpRequestsService', function(httpRequestsService){

        // we injected httpRequestsService service on this controller
        // that made the getName() function available to use.
        httpRequestsService.getName()
            .then(function(response){
                // success
            }, function(error){
```

```
                // do something with the error
            })
    }])
```

En utilisant cette approche, nous pouvons maintenant utiliser **httpRequestsService.js à** tout moment et dans n'importe quel contrôleur.

Calendrier d'une requête $ http

Les requêtes $ http requièrent un temps qui varie selon le serveur, certaines peuvent prendre quelques millisecondes et d'autres peuvent prendre quelques secondes. Souvent, le temps requis pour récupérer les données d'une requête est critique. En supposant que la valeur de réponse est un tableau de noms, considérez l'exemple suivant:

Incorrect

```
$scope.names = [];

$http({
    method: 'GET',
    url: '/someURL'
}).then(function successCallback(response) {
            $scope.names = response.data;
        },
        function errorCallback(response) {
            alert(response.status);
        });

alert("The first name is: " + $scope.names[0]);
```

L'accès à `$scope.names[0]` juste en dessous de la requête $ http `$scope.names[0]` souvent une erreur - cette ligne de code s'exécute avant que la réponse ne soit reçue du serveur.

Correct

```
$scope.names = [];

$scope.$watch('names', function(newVal, oldVal) {
    if(!(newVal.length == 0)) {
        alert("The first name is: " + $scope.names[0]);
    }
});

$http({
    method: 'GET',
    url: '/someURL'
}).then(function successCallback(response) {
            $scope.names = response.data;
        },
        function errorCallback(response) {
            alert(response.status);
        });
```

En utilisant le service $ watch , nous `$scope.names` tableau `$scope.names` uniquement lorsque la

réponse est reçue. Lors de l'initialisation, la fonction est appelée même si `$scope.names` été initialisé auparavant, vérifiant par conséquent si le `newVal.length` est différent de 0 est nécessaire. Sachez que toute modification apportée à `$scope.names` déclenchera la fonction watch.

Lire Demande $ http en ligne: https://riptutorial.com/fr/angularjs/topic/3620/demande---http

Chapitre 12: Des filtres

Examples

Votre premier filtre

Les filtres sont un type spécial de fonction qui peut modifier la manière dont une page est imprimée ou peut être utilisée pour filtrer un tableau ou une action `ng-repeat` . Vous pouvez créer un filtre en appelant la méthode `app.filter()` , en lui passant un nom et une fonction. Voir les exemples ci-dessous pour plus de détails sur la syntaxe.

Par exemple, créons un filtre qui changera une chaîne en majuscules (essentiellement une enveloppe de la fonction javascript `.toUpperCase()`):

```
var app = angular.module("MyApp", []);

// just like making a controller, you must give the
// filter a unique name, in this case "toUppercase"
app.filter('toUppercase', function(){
    // all the filter does is return a function,
    // which acts as the "filtering" function
    return function(rawString){
        // The filter function takes in the value,
        // which we modify in some way, then return
        // back.
        return rawString.toUpperCase();
    };
});
```

Regardons de plus près ce qui se passe au-dessus.

Tout d'abord, nous créons un filtre appelé "toUppercase", qui est comme un contrôleur; `app.filter(...)` . Ensuite, la fonction de ce filtre renvoie la fonction de filtre réelle. Cette fonction prend un seul objet, qui est l'objet à filtrer, et devrait renvoyer la version filtrée de l'objet.

Note: *Dans cette situation, nous supposons que l'objet passé dans le filtre est une chaîne et que nous savons donc toujours utiliser le filtre uniquement sur les chaînes. Cela étant dit, une autre amélioration du filtre pourrait être faite pour parcourir l'objet (s'il s'agit d'un tableau) et pour rendre chaque élément majuscule.*

Maintenant, utilisons notre nouveau filtre en action. Notre filtre peut être utilisé de deux manières, soit dans un modèle angulaire, soit en tant que fonction javascript (en tant que référence angulaire injectée).

Javascript

Il vous suffit d'injecter l'objet `$filter` angulaire `$filter` dans votre contrôleur, puis de l'utiliser pour

récupérer la fonction de filtre en utilisant son nom.

```
app.controller("MyController", function($scope, $filter){
    this.rawString = "Foo";
    this.capsString = $filter("toUppercase")(this.rawString);
});
```

HTML

Pour une directive angulaire, utilisez le symbole pipe (`|`) suivi du nom du filtre dans la directive après la chaîne réelle. Par exemple, supposons que nous ayons un contrôleur appelé `MyController` qui contient une chaîne appelée `rawString` .

```
<div ng-controller="MyController as ctrl">
    <span>Capital rawString: {{ ctrl.rawString | toUppercase }}</span>
</div>
```

Note de l'éditeur: *Angular a un certain nombre de filtres intégrés, y compris "majuscule", et le filtre "toUppercase" est uniquement destiné à montrer comment fonctionnent les filtres, mais vous n'avez* pas *besoin de créer votre propre fonction majuscule.*

Filtre personnalisé pour supprimer des valeurs

Un cas d'utilisation typique d'un filtre consiste à supprimer des valeurs d'un tableau. Dans cet exemple, nous passons dans un tableau et supprimons tous les caractères nuls qui s'y trouvent, en renvoyant le tableau.

```
function removeNulls() {
    return function(list) {
        for (var i = list.length - 1; i >= 0; i--) {
            if (typeof list[i] === 'undefined' ||
                    list[i] === null) {
                list.splice(i, 1);
            }
        }
        return list;
    };
}
```

Cela serait utilisé dans le HTML comme

```
{{listOfItems | removeNulls}}
```

ou dans un contrôleur comme

```
listOfItems = removeNullsFilter(listOfItems);
```

Filtre personnalisé pour formater les valeurs

Un autre cas d'utilisation des filtres consiste à formater une valeur unique. Dans cet exemple, nous transmettons une valeur et nous renvoyons une valeur booléenne vraie appropriée.

```
function convertToBooleanValue() {
    return function(input) {
        if (typeof input !== 'undefined' &&
                input !== null &&
                (input === true || input === 1 || input === '1' || input
                        .toString().toLowerCase() === 'true')) {
            return true;
        }
        return false;
    };
}
```

Qui dans le HTML serait utilisé comme ceci:

```
{{isAvailable | convertToBooleanValue}}
```

Ou dans un contrôleur comme:

```
var available = convertToBooleanValueFilter(isAvailable);
```

Effectuer un filtre dans un tableau enfant

Cet exemple a été réalisé afin de démontrer comment effectuer un filtrage approfondi dans un tableau *enfant* sans nécessiter un filtre personnalisé.

Manette:

```
(function() {
  "use strict";
  angular
    .module('app', [])
    .controller('mainCtrl', mainCtrl);

  function mainCtrl() {
    var vm = this;

    vm.classifications = ["Saloons", "Sedans", "Commercial vehicle", "Sport car"];
    vm.cars = [
       {
          "name":"car1",
          "classifications":[
             {
                "name":"Saloons"
             },
             {
                "name":"Sedans"
             }
          ]
       },
       {
          "name":"car2",
          "classifications":[
```

```
            {
               "name":"Saloons"
            },
            {
               "name":"Commercial vehicle"
            }
         ]
      },
      {
         "name":"car3",
         "classifications":[
            {
               "name":"Sport car"
            },
            {
               "name":"Sedans"
            }
         ]
      }
    ];
  }
})();
```

Vue:

```
<body ng-app="app" ng-controller="mainCtrl as main">
  Filter car by classification:
  <select ng-model="classificationName"
          ng-options="classification for classification in main.classifications"></select>
  <br>
  <ul>
    <li ng-repeat="car in main.cars |
                 filter: { classifications: { name: classificationName } } track by $index"
                ng-bind-template="{{car.name}} - {{car.classifications | json}}">
    </li>
  </ul>
</body>
```

Vérifiez la `DEMO` complète.

Utilisation de filtres dans un contrôleur ou un service

En injectant `$filter` , tout filtre défini dans votre module Angular peut être utilisé dans des contrôleurs, des services, des directives ou même d'autres filtres.

```
angular.module("app")
  .service("users", usersService)
  .controller("UsersController", UsersController);

function usersService () {
  this.getAll = function () {
    return [{
      id: 1,
      username: "john"
    }, {
      id: 2,
      username: "will"
```

```
    }, {
      id: 3,
      username: "jack"
    }];
  };
}

function UsersController ($filter, users) {
  var orderByFilter = $filter("orderBy");

  this.users = orderByFilter(users.getAll(), "username");
  // Now the users are ordered by their usernames: jack, john, will

  this.users = orderByFilter(users.getAll(), "username", true);
  // Now the users are ordered by their usernames, in reverse order: will, john, jack
}
```

Accéder à une liste filtrée depuis l'extérieur d'une répétition

Parfois, vous souhaiterez accéder au résultat de vos filtres en dehors de `ng-repeat` , peut-être pour indiquer le nombre d'éléments filtrés. Vous pouvez le faire en utilisant `as [variablename]` syntaxe `as [variablename]` sur le `ng-repeat` .

```
<ul>
  <li ng-repeat="item in vm.listItems | filter:vm.myFilter as filtered">
    {{item.name}}
  </li>
</ul>
<span>Showing {{filtered.length}} of {{vm.listItems.length}}</span>
```

Lire Des filtres en ligne: https://riptutorial.com/fr/angularjs/topic/1401/des-filtres

Chapitre 13: Des promesses angulaires avec le service $ q

Examples

Utiliser $ q.all pour gérer plusieurs promesses

Vous pouvez utiliser la fonction `$q.all` pour appeler une méthode `.then` après avoir `.then` un tableau de promesses et récupérer les données avec `.then` elles ont été résolues.

Exemple:

JS:

```
$scope.data = []

$q.all([
   $http.get("data.json"),
   $http.get("more-data.json"),
   ]).then(function(responses) {
     $scope.data = responses.map((resp) => resp.data);
   });
```

Le code ci - dessus fonctionne `$http.get` 2 fois pour les données dans les fichiers JSON locaux, lorsque les deux `get` la méthode complète , ils résolvent leurs promesses associées, lorsque toutes les promesses du tableau sont résolus, la `.then` méthode commence par les données de promesses à l' intérieur des `responses` argument de tableau.

Les données sont ensuite mappées pour pouvoir être affichées sur le modèle, nous pouvons alors afficher

HTML:

```
<ul>
   <li ng-repeat="d in data">
      <ul>
         <li ng-repeat="item in d">{{item.name}}: {{item.occupation}}</li>
      </ul>
   </li>
</ul>
```

JSON:

```
[{
  "name": "alice",
  "occupation": "manager"
}, {
  "name": "bob",
  "occupation": "developer"
```

```
}]
```

Utiliser le constructeur $ q pour créer des promesses

La fonction constructeur `$q` permet de créer des promesses à partir d'API asynchrones qui utilisent des rappels pour renvoyer des résultats.

> $ q (fonction (résoudre, rejeter) {...})

La fonction constructeur reçoit une fonction appelée avec deux arguments, `resolve` et `reject` qui sont des fonctions utilisées pour résoudre ou rejeter la promesse.

Exemple 1:

```
function $timeout(fn, delay) {
    return = $q(function(resolve, reject) {
         setTimeout(function() {
              try {
                 let r = fn();
                 resolve(r);
              }
              catch (e) {
                 reject(e);
              }
         }, delay);
     };
}
```

L'exemple ci-dessus crée une promesse à partir de l' API WindowTimers.setTimeout . Le framework AngularJS fournit une version plus élaborée de cette fonction. Pour en savoir plus, consultez la référence de l'API AngularJS $ timeout Service .

Exemple 2:

```
$scope.divide = function(a, b) {
    return $q(function(resolve, reject) {
      if (b===0) {
        return reject("Cannot devide by 0")
      } else {
        return resolve(a/b);
      }
    });
}
```

Le code ci-dessus montrant une fonction de division promis, il renverra une promesse avec le résultat ou rejetera avec une raison si le calcul est impossible.

Vous pouvez alors appeler et utiliser `.then`

```
$scope.divide(7, 2).then(function(result) {
    // will return 3.5
}, function(err) {
```

```
    // will not run
})

$scope.divide(2, 0).then(function(result) {
    // will not run as the calculation will fail on a divide by 0
}, function(err) {
    // will return the error string.
})
```

Report des opérations en utilisant $ q.defer

Nous pouvons utiliser `$q` pour différer les opérations à venir tout en ayant un objet prometteur en attente, en utilisant `$q.defer` nous créons une promesse qui sera résolue ou rejetée dans le futur.

Cette méthode n'est pas équivalente à l'utilisation du constructeur `$q` , car nous utilisons `$q.defer` pour promouvoir une routine existante qui peut ou peut ne pas renvoyer une promesse.

Exemple:

```
var runAnimation = function(animation, duration) {
    var deferred = $q.defer();
    try {
        ...
        // run some animation for a given duration
        deferred.resolve("done");
    } catch (err) {
        // in case of error we would want to run the error hander of .then
        deferred.reject(err);
    }
    return deferred.promise;
}

// and then
runAnimation.then(function(status) {}, function(error) {})
```

1. Assurez-vous de toujours renvoyer l'objet `deferred.promise` ou risquez une erreur lors de l'appel de `.then`
2. Assurez-vous de toujours résoudre ou rejeter votre objet différé ou `.then` peut ne pas fonctionner et vous risquez une fuite de mémoire

Utiliser des promesses angulaires avec le service $ q

`$q` est un service intégré qui aide à exécuter des fonctions asynchrones et à utiliser leurs valeurs de retour (ou exceptions) une fois le traitement terminé.

`$q` est intégré au mécanisme d'observation du modèle `$rootScope.Scope` , ce qui signifie une propagation plus rapide de la résolution ou du rejet dans vos modèles et évite `$rootScope.Scope` inutilement le navigateur, ce qui entraînerait une scintillement de l'interface utilisateur.

Dans notre exemple, nous appelons notre usine `getMyData` , qui renvoie un objet promis. Si l'objet est `resolved` , il renvoie un nombre aléatoire. S'il est `rejected` , il renvoie un rejet avec un message

d'erreur après 2 secondes.

Dans l'usine angulaire

```
function getMyData($timeout, $q) {
  return function() {
    // simulated async function
    var promise = $timeout(function() {
      if(Math.round(Math.random())) {
        return 'data received!'
      } else {
        return $q.reject('oh no an error! try again')
      }
    }, 2000);
    return promise;
  }
}
```

Utiliser des promesses sur appel

```
angular.module('app', [])
.factory('getMyData', getMyData)
.run(function(getData) {
  var promise = getData()
    .then(function(string) {
      console.log(string)
    }, function(error) {
      console.error(error)
    })
    .finally(function() {
      console.log('Finished at:', new Date())
    })
})
```

Pour utiliser des promesses, injectez `$q` comme dépendance. Ici, nous avons injecté `$q` dans la fabrique `getMyData` .

```
var defer = $q.defer();
```

Une nouvelle instance de différé est construite en appelant `$q.defer()`

Un objet différé est simplement un objet qui expose une promesse ainsi que les méthodes associées pour résoudre cette promesse. Il est construit en utilisant la fonction `$q.deferred()` et expose trois méthodes principales: `resolve()` , `reject()` et `notify()` .

- `resolve(value)` - Résout la promesse dérivée avec la valeur.
- `reject(reason)` - rejette la promesse dérivée avec la raison.
- `notify(value)` - fournit des mises à jour sur l'état de l'exécution de la promesse. Cela peut être appelé plusieurs fois avant que la promesse soit résolue ou rejetée.

Propriétés

L'objet promis associé est accessible via la propriété promis. `promise` - {Promise} - objet de promesse associé à ce différé.

Une nouvelle instance de promesse est créée lorsqu'une instance différée est créée et peut être récupérée en appelant `deferred.promise` .

L'objet de la `promise` est de permettre aux parties intéressées d'accéder au résultat de la tâche différée une fois terminé.

Méthodes de promesses -

- `then(successCallback, [errorCallback], [notifyCallback])` - Indépendamment du moment où la promesse était ou sera résolue ou rejetée, appelle l'un des rappels de succès ou d'erreur de manière asynchrone dès que le résultat est disponible. Les callbacks sont appelés avec un seul argument: le résultat ou la raison du rejet. De plus, le rappel de notification peut être appelé zéro ou plusieurs fois pour fournir une indication de progression, avant que la promesse ne soit résolue ou rejetée.
- `catch(errorCallback)` - raccourci pour promise.then (null, errorCallback)
- `finally(callback, notifyCallback)` - vous permet d'observer l'accomplissement ou le rejet d'une promesse, mais de le faire sans modifier la valeur finale.

L'une des caractéristiques les plus puissantes des promesses est la capacité de les enchaîner. Cela permet aux données de circuler dans la chaîne et d'être manipulées et mutées à chaque étape. Ceci est démontré avec l'exemple suivant:

Exemple 1:

```
// Creates a promise that when resolved, returns 4.
function getNumbers() {

  var promise = $timeout(function() {
    return 4;
  }, 1000);

  return promise;
}

// Resolve getNumbers() and chain subsequent then() calls to decrement
// initial number from 4 to 0 and then output a string.
getNumbers()
  .then(function(num) {
      // 4
      console.log(num);
      return --num;
  })
  .then(function (num) {
      // 3
      console.log(num);
      return --num;
  })
   .then(function (num) {
      // 2
```

```
        console.log(num);
        return --num;
    })
    .then(function (num) {
        // 1
        console.log(num);
        return --num;
    })
    .then(function (num) {
        // 0
        console.log(num);
        return 'And we are done!';
    })
    .then(function (text) {
        // "And we are done!"
        console.log(text);
    });
```

Enveloppez la valeur simple dans une promesse en utilisant $ q.when ()

Si tout ce dont vous avez besoin est d'encapsuler la valeur dans une promesse, vous n'avez pas besoin d'utiliser la syntaxe longue comme ici:

```
//OVERLY VERBOSE
var defer;
defer = $q.defer();
defer.resolve(['one', 'two']);
return defer.promise;
```

Dans ce cas, vous pouvez simplement écrire:

```
//BETTER
return $q.when(['one', 'two']);
```

> # $ q.when et son alias $ q.resolve
>
> Entoure un objet qui pourrait être une valeur ou une promesse (tierce) alors capable dans une promesse $ q. Ceci est utile lorsque vous traitez avec un objet qui peut ou peut ne pas être une promesse, ou si la promesse provient d'une source qui ne peut pas être fiable.
>
> - AngularJS $ q Service API Reference - $ q.when

Avec la sortie d'AngularJS v1.4.1

Vous pouvez également utiliser un alias cohérent ES6 `resolve`

```
//ABSOLUTELY THE SAME AS when
return $q.resolve(['one', 'two'])
```

Éviter ce anti-pattern

```
var myDeferred = $q.defer();

$http(config).then(function(res) {
   myDeferred.resolve(res);
}, function(error) {
   myDeferred.reject(error);
});

return myDeferred.promise;
```

Il n'est pas nécessaire de fabriquer une promesse avec `$q.defer` car le service $ http renvoie déjà une promesse.

```
//INSTEAD
return $http(config);
```

Renvoyez simplement la promesse créée par le service $ http.

Lire Des promesses angulaires avec le service $ q en ligne: https://riptutorial.com/fr/angularjs/topic/4379/des-promesses-angulaires-avec-le-service---q

Chapitre 14: digestion en boucle

Syntaxe

- $ scope. $ watch (watchExpression, callback, [comparaison profonde])
- $ scope. $ digest ()
- $ scope. $ apply ([exp])

Examples

liaison de données bidirectionnelle

Angular a de la magie sous son capot. il permet de lier les variables DOM aux vraies js.

Angular utilise une boucle, appelée " *boucle de digestion* ", appelée après toute modification d'une variable - appelant des rappels qui mettent à jour le DOM.

Par exemple, la directive `ng-model` associe un `keyup` eventListener à cette entrée:

```
<input ng-model="variable" />
```

Chaque fois que l'événement d' `keyup` déclenche, la *boucle de résumé* commence.

À un moment donné, la *boucle de résumé effectue une* itération sur un rappel qui met à jour le contenu de cette plage:

```
<span>{{variable}}</span>
```

Le cycle de vie de base de cet exemple résume (très schématiquement) le fonctionnement angulaire ::

1. Analyses angulaires html
    - `ng-model` directive `ng-model` crée un écouteur `keyup` en entrée
    - `expression` intérieur de l'intervalle ajoute un rappel au *cycle de digestion*
2. L'utilisateur interagit avec l'entrée
    - `keyup` listener commence le *cycle de digestion*
    - *cycle de digestion* appelle le rappel
    - Rappel des mises à jour

$ digest et $ watch

L'implémentation de la liaison de données à deux voies, pour obtenir le résultat de l'exemple précédent, pourrait être réalisée avec deux fonctions principales:

- **$ digest** est appelé après une interaction utilisateur (liaison DOM => variable)

- **$ watch** définit un rappel à appeler après les changements de variables (variable de liaison => DOM)

note: cet exemple est une démonstration, pas le code angulaire réel

---

```
<input id="input"/>
<span id="span"></span>
```

Les deux fonctions dont nous avons besoin:

```
var $watches = [];
function $digest(){
    $watches.forEach(function($w){
        var val = $w.val();
        if($w.prevVal !== val){
            $w.callback(val, $w.prevVal);
            $w.prevVal = val;
        }
    })
}
function $watch(val, callback){
    $watches.push({val:val, callback:callback, prevVal: val() })
}
```

Maintenant, nous pourrions maintenant utiliser ces fonctions pour connecter une variable au DOM (angular est livré avec des directives intégrées qui le feront pour vous):

```
var realVar;
//this is usually done by ng-model directive
input1.addEventListener('keyup',function(e){
    realVar=e.target.value;
    $digest()
}, true);

//this is usually done with {{expressions}} or ng-bind directive
$watch(function(){return realVar},function(val){
    span1.innerHTML = val;
});
```

Bien entendu, les implémentations réelles sont plus complexes et prennent en charge des paramètres tels que l' **élément** à lier et la **variable** à utiliser.

Un exemple courant peut être trouvé ici: https://jsfiddle.net/azofxd4j/

l'arbre $ scope

L'exemple précédent est suffisant lorsque nous devons lier un seul élément HTML, à une seule variable.

En réalité, nous avons besoin de lier plusieurs éléments à plusieurs variables:

```
<span ng-repeat="number in [1,2,3,4,5]">{{number}}</span>
```

Ce `ng-repeat` lie 5 éléments à 5 variables appelées `number` , avec une valeur différente pour chacun d'eux!

La façon dont angulaire atteint ce comportement utilise un contexte distinct pour chaque élément nécessitant des variables distinctes. Ce contexte est appelé une portée.

Chaque portée contient des propriétés, qui sont les variables liées au DOM, et les fonctions `$digest` et `$watch` sont implémentées en tant que méthodes de la portée.

Le DOM est un arbre, et les variables doivent être utilisées à différents niveaux de l'arborescence:

```
<div>
    <input ng-model="person.name" />
    <span ng-repeat="number in [1,2,3,4,5]">{{number}} {{person.name}}</span>
</div>
```

Mais comme nous l'avons vu, le contexte (ou la portée) des variables à l'intérieur de `ng-repeat` est différent du contexte au-dessus. Pour résoudre ce problème - angulaire implémente des étendues comme un arbre.

Chaque champ a un tableau d'enfants, et appelle sa `$digest` méthode fonctionnera tous ses enfants de `$digest` la méthode.

De cette façon - après avoir modifié l'entrée - `$digest` est appelé pour la portée du div, qui exécute alors le `$digest` pour ses 5 enfants - qui mettra à jour son contenu.

Une implémentation simple pour une portée pourrait ressembler à ceci:

```
function $scope(){
    this.$children = [];
    this.$watches = [];
}

$scope.prototype.$digest = function(){
    this.$watches.forEach(function($w){
        var val = $w.val();
        if($w.prevVal !== val){
            $w.callback(val, $w.prevVal);
          $w.prevVal = val;
        }
    });
    this.$children.forEach(function(c){
        c.$digest();
    });
}

$scope.prototype.$watch = function(val, callback){
    this.$watches.push({val:val, callback:callback, prevVal: val() })
}
```

note: cet exemple est une démonstration, pas le code angulaire réel

Lire digestion en boucle en ligne: https://riptutorial.com/fr/angularjs/topic/3156/digestion-en-boucle

Chapitre 15: directive classe ng

Examples

Trois types d'expressions de classe ng

Angular prend en charge trois types d'expressions dans la directive `ng-class` .

1. ficelle

```
<span ng-class="MyClass">Sample Text</span>
```

Spécifier une expression qui évalue une chaîne demande à Angular de la traiter comme une variable $ scope. Angular vérifie la portée de $ et recherche une variable appelée "MyClass". Quel que soit le texte contenu dans "MyClass", il deviendra le nom de classe réel appliqué à ce `<span>` . Vous pouvez spécifier plusieurs classes en séparant chaque classe d'un espace.

Dans votre contrôleur, vous pouvez avoir une définition qui ressemble à ceci:

```
$scope.MyClass = "bold-red deleted error";
```

Angular évaluera l'expression `MyClass` et trouvera la définition de $ scope. Il appliquera les trois classes "bold-red", "deleted" et "error" à l'élément `<span>` .

La spécification des classes de cette manière vous permet de modifier facilement les définitions de classe dans votre contrôleur. Par exemple, vous devrez peut-être modifier la classe en fonction d'autres interactions utilisateur ou de nouvelles données chargées à partir du serveur. De plus, si vous avez beaucoup d'expressions à évaluer, vous pouvez le faire dans une fonction qui définit la liste finale des classes dans une variable `$scope` . Cela peut être plus facile que d'essayer de placer de nombreuses évaluations dans l'attribut `ng-class` de votre modèle HTML.

---

2. objet

C'est le moyen le plus couramment utilisé pour définir des classes à l'aide de `ng-class` car il vous permet de spécifier facilement les évaluations qui déterminent la classe à utiliser.

Spécifiez un objet contenant des paires clé-valeur. La clé est le nom de classe qui sera appliqué si la valeur (un conditionnel) est évaluée comme vraie.

```
<style>
    .red { color: red; font-weight: bold; }
    .blue { color: blue; }
    .green { color: green; }
```

```
    .highlighted { background-color: yellow; color: black; }
</style>

<span ng-class="{ red: ShowRed, blue: ShowBlue, green: ShowGreen, highlighted: IsHighlighted
}">Sample Text</span>

<div>Red: <input type="checkbox" ng-model="ShowRed"></div>
<div>Green: <input type="checkbox" ng-model="ShowGreen"></div>
<div>Blue: <input type="checkbox" ng-model="ShowBlue"></div>
<div>Highlight: <input type="checkbox" ng-model="IsHighlighted"></div>
```

---

3. tableau

Une expression qui évalue un tableau vous permet d'utiliser une combinaison de **chaînes** (voir n ° 1 ci-dessus) et d' **objets conditionnels** (n ° 2 ci-dessus).

```
<style>
    .bold { font-weight: bold; }
    .strike { text-decoration: line-through; }
    .orange { color: orange; }
</style>

<p ng-class="[ UserStyle, {orange: warning} ]">Array of Both Expression Types</p>
<input ng-model="UserStyle" placeholder="Type 'bold' and/or 'strike'"><br>
<label><input type="checkbox" ng-model="warning"> warning (apply "orange" class)</label>
```

Cela crée un champ de saisie de texte lié à la variable d'étendue `UserStyle` qui permet à l'utilisateur de saisir n'importe quel nom de classe. Ceux-ci seront appliqués dynamiquement à l'élément `<p>` tant que type d'utilisateur. En outre, l'utilisateur peut cliquer sur la case à cocher liée aux données de la variable d'étendue d' `warning` . Cela sera également appliqué dynamiquement à l'élément `<p>` .

Lire directive classe ng en ligne: https://riptutorial.com/fr/angularjs/topic/2395/directive-classe-ng

Chapitre 16: Directives intégrées

Examples

Expressions angulaires - Texte vs Nombre

Cet exemple montre comment les expressions angulaires sont évaluées lors de l'utilisation de `type="text"` et de `type="number"` pour l'élément d'entrée. Considérez le contrôleur suivant et visualisez:

Manette

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

app.controller('ctrl', function($scope) {
    $scope.textInput = {
        value: '5'
    };
    $scope.numberInput = {
        value: 5
    };
});
```

Vue

```
<div ng-app="app" ng-controller="ctrl">
    <input type="text" ng-model="textInput.value">
    {{ textInput.value + 5 }}
    <input type="number" ng-model="numberInput.value">
    {{ numberInput.value + 5 }}
</div>
```

- Lorsque vous utilisez `+` dans une expression liée à une entrée de *texte* , l'opérateur **concaténera** les chaînes (premier exemple), affichant 55 sur l'écran `*` .
- Lorsque vous utilisez `+` dans une expression liée à la saisie d'un *numéro* , l'opérateur retourne la **somme** des nombres (deuxième exemple), affichant 10 sur l'écran `*` .

`*` - Jusqu'à ce que l'utilisateur modifie la valeur dans le champ de saisie, l'affichage changera ensuite en conséquence.

Exemple de travail

ngRépéter

`ng-repeat` est une directive intégrée dans Angular qui vous permet d'itérer un tableau ou un objet et vous permet de répéter un élément une fois pour chaque élément de la collection.

ng-repeat un tableau

```
<ul>
    <li ng-repeat="item in itemCollection">
       {{item.Name}}
    </li>
</ul>
```

Où:
item = **article** individuel dans la collection
itemCollection = Le tableau que vous itérez

ng-repeter un objet

```
<ul>
    <li ng-repeat="(key, value) in myObject">
       {{key}} : {{value}}
    </li>
</ul>
```

Où:
key = le nom de la propriété
valeur = la valeur de la propriété
myObject = l'objet que vous itérez

filtrer votre ng-repeat par entrée utilisateur

```
<input type="text" ng-model="searchText">
<ul>
    <li ng-repeat="string in stringArray | filter:searchText">
       {{string}}
    </li>
</ul>
```

Où:
searchText = le texte que l'utilisateur souhaite filtrer par liste
stringArray = un tableau de chaînes, par exemple `['string', 'array']`

Vous pouvez également afficher ou référencer les éléments filtrés ailleurs en attribuant un alias avec `as aliasName` , comme ceci:

```
<input type="text" ng-model="searchText">
<ul>
    <li ng-repeat="string in stringArray | filter:searchText as filteredStrings">
       {{string}}
    </li>
</ul>
<p>There are {{filteredStrings.length}} matching results</p>
```

ng-repeat-start et ng-repeat-end

Pour répéter plusieurs éléments DOM en définissant un point de départ et un point de fin, vous pouvez utiliser les directives `ng-repeat-start` et `ng-repeat-end` .

```
<ul>
    <li ng-repeat-start="item in [{a: 1, b: 2}, {a: 3, b:4}]">
        {{item.a}}
    </li>
    <li ng-repeat-end>
        {{item.b}}
    </li>
</ul>
```

Sortie:

- 1
- 2
- 3
- 4

Il est important de toujours fermer `ng-repeat-start` avec `ng-repeat-end` .

Les variables

`ng-repeat` expose également ces variables à l'intérieur de l'expression

Variable	Type	Détails
`$index`	Nombre	Égal à l'index de l'itération courante ($ index === 0 sera évalué à true au premier élément itéré; voir d' `$first`)
`$first`	Booléen	Évalue à true au premier élément itéré
`$last`	Booléen	Évalue à true au dernier élément itéré
`$middle`	Booléen	Évalue à true si l'élément se situe entre `$first` et `$last`
`$even`	Booléen	Évalué à true à une itération paire (équivalent à `$index%2===0`)
`$odd`	Booléen	Évalué à true à une itération impaire (équivalent à `$index%2===1`)

Considérations de performance

Rendu `ngRepeat` peut devenir lent, en particulier lors de l'utilisation de grandes collections.

Si les objets de la collection ont une propriété d'identifiant, vous devez toujours `track by` l'identificateur plutôt que l'objet entier, qui est la fonctionnalité par défaut. Si aucun identifiant n'est présent, vous pouvez toujours utiliser l' `$index` intégré.

```
<div ng-repeat="item in itemCollection track by item.id">
```

```
<div ng-repeat="item in itemCollection track by $index">
```

Portée de ngRepeat

`ngRepeat` crée toujours une étendue enfant isolée, il faut donc faire attention si la portée parent doit être accessible à l'intérieur de la répétition.

Voici un exemple simple montrant comment définir une valeur dans votre portée parent à partir d'un événement click à l'intérieur de `ngRepeat` .

```
scope val:  {{val}}<br/>
ctrlAs val: {{ctrl.val}}
<ul>
    <li ng-repeat="item in itemCollection">
        <a href="#" ng-click="$parent.val=item.value; ctrl.val=item.value;">
            {{item.label}} {{item.value}}
        </a>
    </li>
</ul>

$scope.val = 0;
this.val = 0;

$scope.itemCollection = [{
    id: 0,
    value: 4.99,
    label: 'Football'
},
{
    id: 1,
    value: 6.99,
    label: 'Baseball'
},
{
    id: 2,
    value: 9.99,
    label: 'Basketball'
}];
```

S'il n'y avait que `val = item.value` à `ng-click` il ne `val = item.value` pas à jour le `val` dans l'étendue parent en raison de l'étendue isolée. C'est pourquoi la portée parent est accessible avec `$parent` référence `$parent` ou avec la syntaxe `controllerAs` (par exemple `ng-controller="mainController as ctrl"`).

Imbriqué ng-repeat

Vous pouvez également utiliser ng-repeat imbriqué.

```
<div ng-repeat="values in test">
    <div ng-repeat="i in values">
      [{{$parent.$index}},{{$index}}] {{i}}
    </div>
</div>

var app = angular.module("myApp", []);
app.controller("ctrl", function($scope) {
```

```
  $scope.test = [
    ['a', 'b', 'c'],
    ['d', 'e', 'f']
  ];
});
```

Ici, pour accéder à l'index du parent ng-repeat à l'intérieur de child ng-repeat, vous pouvez utiliser `$parent.$index` .

ngShow et ngHide

La directive `ng-show` affiche ou masque l'élément HTML basé sur si l'expression qui lui est transmise est vraie ou fausse. Si la valeur de l'expression est falsifiée, elle se cachera. Si c'est véridique alors ça montrera.

La directive `ng-hide` est similaire. Cependant, si la valeur est falsifiée, elle affichera l'élément HTML. Lorsque l'expression est véridique, elle le cachera.

Exemple de travail JSBin

Contrôleur :

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

angular.module('app')
  .controller('ExampleController', ExampleController);

function ExampleController() {

  var vm = this;

  //Binding the username to HTML element
  vm.username = '';

  //A taken username
  vm.taken_username = 'StackOverflow';

}
```

Vue

```
<section ng-controller="ExampleController as main">

    <p>Enter Password</p>
    <input ng-model="main.username" type="text">

    <hr>

    <!-- Will always show as long as StackOverflow is not typed in -->
    <!-- The expression is always true when it is not StackOverflow -->
    <div style="color:green;" ng-show="main.username != main.taken_username">
      Your username is free to use!
    </div>

    <!-- Will only show when StackOverflow is typed in -->
```

```
    <!-- The expression value becomes falsy -->
    <div style="color:red;" ng-hide="main.username != main.taken_username">
      Your username is taken!
    </div>

    <p>Enter 'StackOverflow' in username field to show ngHide directive.</p>

 </section>
```

ngOptions

`ngOptions` est une directive qui simplifie la création d'une liste déroulante HTML pour la sélection d'un élément à partir d'un tableau qui sera stocké dans un modèle. L'attribut ngOptions est utilisé pour générer dynamiquement une liste d'éléments `<option>` pour l'élément `<select>` utilisant le tableau ou l'objet obtenu en évaluant l'expression de compréhension ngOptions.

Avec `ng-options` le balisage peut être réduit à une balise select et la directive crée la même sélection:

```
<select ng-model="selectedFruitNgOptions"
        ng-options="curFruit as curFruit.label for curFruit in fruit">
</select>
```

Il y a une autre façon de créer `select` options `select` en utilisant `ng-repeat` , mais il n'est pas recommandé d'utiliser `ng-repeat` car il est principalement utilisé pour des usages généraux, comme `forEach` juste pour faire une boucle. Alors que `ng-options` est spécifiquement conçu pour créer `select` options de balises `select` .

L'exemple ci-dessus utilisant `ng-repeat` serait

```
<select ng-model="selectedFruit">
    <option ng-repeat="curFruit in fruit" value="{{curFruit}}">
        {{curFruit.label}}
    </option>
</select>
```

EXEMPLE COMPLET

Voyons l'exemple ci-dessus en détail également avec quelques variations.

Modèle de données pour l'exemple:

```
$scope.fruit = [
    { label: "Apples", value: 4, id: 2 },
    { label: "Oranges", value: 2, id: 1 },
    { label: "Limes", value: 4, id: 4 },
    { label: "Lemons", value: 5, id: 3 }
];
```

```
<!-- label for value in array -->
<select ng-options="f.label for f in fruit" ng-model="selectedFruit"></select>
```

Balise d'option générée lors de la sélection:

```
<option value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Effets:

`f.label` sera l'étiquette de l' `<option>` et la valeur contiendra l'objet entier.

EXEMPLE COMPLET

```
<!-- select as label for value in array -->
<select ng-options="f.value as f.label for f in fruit" ng-model="selectedFruit"></select>
```

Balise d'option générée lors de la sélection:

```
<option value="4"> Apples </option>
```

Effets:

`f.value` (4) sera la valeur dans ce cas, alors que l'étiquette est toujours la même.

EXEMPLE COMPLET

```
<!-- label group by group for value in array -->
<select ng-options="f.label group by f.value for f in fruit" ng-
model="selectedFruit"></select>
```

Balise d'option générée lors de la sélection:

```
<option value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Effets:

Les options seront regroupées en fonction de leur `value` . Les options avec la même `value` tomberont dans une catégorie

EXEMPLE COMPLET

```
<!-- label disable when disable for value in array -->
<select ng-options="f.label disable when f.value == 4 for f in fruit" ng-
model="selectedFruit"></select>
```

Balise d'option générée lors de la sélection:

```
<option disabled="" value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Effets:

"Pommes" et "Limes" seront désactivés (impossible à sélectionner) à cause de la condition `disable when f.value==4` . Toutes les options avec la `value=4` seront désactivées

EXEMPLE COMPLET

```
<!-- label group by group for value in array track by trackexpr -->
<select ng-options="f.value as f.label group by f.value for f in fruit track by f.id" ng-
model="selectedFruit"></select>
```

Balise d'option générée lors de la sélection:

```
<option value="4"> Apples </option>
```

Effets:

Il n'ya pas de changement visuel lors de l'utilisation de `trackBy` , mais Angular détectera les modifications par l' `id` au lieu de par référence, ce qui est toujours une meilleure solution.

EXEMPLE COMPLET

```
<!-- label for value in array | orderBy:orderexpr track by trackexpr -->
<select ng-options="f.label for f in fruit | orderBy:'id' track by f.id" ng-
model="selectedFruit"></select>
```

Balise d'option générée lors de la sélection:

```
<option disabled="" value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Effets:

`orderBy` est un filtre standard AngularJS qui organise les options par ordre croissant (par défaut), afin que "Oranges" apparaisse en premier puisque son `id` = 1.

EXEMPLE COMPLET

Tous les `<select>` avec les `ng-options` `ng-model` doivent avoir `ng-model` attaché.

ngModèle

Avec ng-model, vous pouvez associer une variable à n'importe quel type de champ de saisie. Vous pouvez afficher la variable en utilisant des accolades doubles, par exemple `{{myAge}}` .

```
<input type="text" ng-model="myName">
<p>{{myName}}</p>
```

Au fur et à mesure que vous tapez dans le champ de saisie ou que vous le modifiez de quelque manière que ce soit, vous verrez instantanément la valeur dans le paragraphe mise à jour.

La variable ng-model, dans cet exemple, sera disponible dans votre contrôleur en tant que `$scope.myName` . Si vous utilisez la syntaxe `controllerAs` :

```
<div ng-controller="myCtrl as mc">
    <input type="text" ng-model="mc.myName">
    <p>{{mc.myName}}</p>
</div>
```

Vous devrez vous référer à la portée du contrôleur en pré-attendant l'alias du contrôleur défini dans l'attribut ng-controller à la variable ng-model. De cette façon, vous n'aurez pas besoin d'injecter `$scope` dans votre contrôleur pour référencer votre variable ng-model, la variable sera disponible sous la forme `this.myName` dans la fonction de votre contrôleur.

ngClass

Supposons que vous deviez afficher le statut d'un utilisateur et que vous ayez plusieurs classes CSS possibles. Angular permet de choisir facilement parmi une liste de plusieurs classes possibles qui vous permettent de spécifier une liste d'objets incluant des conditions. Angular est capable d'utiliser la classe correcte en fonction de la véracité des conditions.

Votre objet doit contenir des paires clé / valeur. La clé est un nom de classe qui sera appliqué lorsque la valeur (conditionnelle) est évaluée à true.

```
<style>
    .active { background-color: green; color: white; }
    .inactive { background-color: gray; color: white; }
    .adminUser { font-weight: bold; color: yellow; }
    .regularUser { color: white; }
</style>

<span ng-class="{
    active: user.active,
    inactive: !user.active,
    adminUser: user.level === 1,
    regularUser: user.level === 2
}">John Smith</span>
```

Angular vérifiera l'objet `$scope.user` pour voir le statut `active` et le numéro de `level` . Selon les valeurs de ces variables, Angular appliquera le style correspondant au `<span>` .

ngIf

`ng-if` est une directive similaire à **`ng-show`** mais insère ou supprime l'élément du DOM au lieu de le cacher simplement. Angular 1.1.5 introduit la directive ng-If. Vous pouvez utiliser la directive ng-if au-dessus des versions 1.1.5. Ceci est utile car Angular ne traitera pas les digests pour les éléments à l'intérieur d'un `ng-if` supprimé, `ng-if` réduit la charge de travail d'Angular, en particulier pour les liaisons de données complexes.

Contrairement à `ng-show` , la directive `ng-if` crée une étendue enfant qui utilise l'héritage prototype. Cela signifie que la définition d'une valeur primitive sur la portée enfant ne s'appliquera pas au parent. Pour définir une primitive sur la portée `$parent` propriété `$parent` de la portée enfant doit

être utilisée.

JavaScript

```
angular.module('MyApp', []);

angular.module('MyApp').controller('myController', ['$scope', '$window', function
myController($scope, $window) {
    $scope.currentUser= $window.localStorage.getItem('userName');
}]);
```

Vue

```
<div ng-controller="myController">
    <div ng-if="currentUser">
        Hello, {{currentUser}}
    </div>
    <div ng-if="!currentUser">
        <a href="/login">Log In</a>
        <a href="/register">Register</a>
    </div>
</div>
```

DOM si `currentUser` n'est pas `currentUser`

```
<div ng-controller="myController">
    <div ng-if="currentUser">
        Hello, {{currentUser}}
    </div>
    <!-- ng-if: !currentUser -->
</div>
```

DOM si `currentUser` est `currentUser`

```
<div ng-controller="myController">
    <!-- ng-if: currentUser -->
    <div ng-if="!currentUser">
        <a href="/login">Log In</a>
        <a href="/register">Register</a>
    </div>
</div>
```

Exemple de travail

Promesse de fonction

La directive ngIf accepte également les fonctions, qui exigent logiquement de renvoyer true ou false.

```
<div ng-if="myFunction()">
    <span>Span text</span>
</div>
```

Le texte span apparaîtra uniquement si la fonction renvoie true.

```
$scope.myFunction = function() {
    var result = false;
    // Code to determine the boolean value of result
    return result;
};
```

Comme toute expression angulaire, la fonction accepte tout type de variable.

ngMouseenter et ngMouseleave

Les directives `ng-mouseenter` et `ng-mouseleave` sont utiles pour exécuter des événements et appliquer un style CSS lorsque vous passez ou sortez de vos éléments DOM.

La directive `ng-mouseenter` exécute une expression dont un événement d'entrée de souris (lorsque l'utilisateur entre son pointeur de souris sur l'élément DOM dans lequel réside cette directive)

HTML

```
<div ng-mouseenter="applyStyle = true" ng-class="{'active': applyStyle}">
```

Dans l'exemple ci-dessus, lorsque l'utilisateur pointe son curseur sur le `div` , `applyStyle` devient `true` , ce qui applique la classe CSS `.active` à la `ng-class` .

La directive `ng-mouseleave` exécute une expression un événement de sortie de la souris (lorsque l'utilisateur éloigne le curseur de la souris de l'élément DOM dans lequel réside cette directive)

HTML

```
<div ng-mouseenter="applyStyle = true" ng-mouseleaver="applyStyle = false" ng-
class="{'active': applyStyle}">
```

En réutilisant le premier exemple, maintenant, lorsque l'utilisateur prend le pointeur de la souris en dehors du div, la classe `.active` est supprimée.

ngDisabled

Cette directive est utile pour limiter les événements d'entrée en fonction de certaines conditions existantes.

La directive `ng-disabled` accepte et exprime une valeur qui doit correspondre à une valeur

véridique ou à une valeur falsifiée.

`ng-disabled` est utilisé pour appliquer conditionnellement l'attribut `disabled` sur un élément d' `input` .

HTML

```
<input type="text" ng-model="vm.name">

<button ng-disabled="vm.name.length===0" ng-click="vm.submitMe">Submit</button>
```

`vm.name.length===0` est évalué à true si la longueur de l' `input` est 0, ce qui désactive le bouton, empêchant l'utilisateur de déclencher l'événement click de `ng-click`

ngDblclick

La directive `ng-dblclick` est utile lorsque vous souhaitez `ng-dblclick` un événement de double-clic à vos éléments DOM.

Cette directive accepte une expression

HTML

```
<input type="number" ng-model="num = num + 1" ng-init="num=0">

<button ng-dblclick="num++">Double click me</button>
```

Dans l'exemple ci-dessus, la valeur conservée à l' `input` sera incrémentée lorsque vous double-cliquerez sur le bouton.

Directives intégrées Cheat Sheet

`ng-app` Définit la section AngularJS.

`ng-init` Définit une valeur de variable par défaut.

`ng-bind` Alternative au modèle {{}}.

`ng-bind-template` Lie plusieurs expressions à la vue.

`ng-non-bindable` États que les données ne peuvent pas être liées.

`ng-bind-html` Lie la propriété HTML interne d'un élément HTML.

`ng-change` Evalue l'expression spécifiée lorsque l'utilisateur modifie l'entrée.

`ng-checked` Définit la case à cocher.

`ng-class` Définit la classe css dynamiquement.

`ng-cloak` Empêche l'affichage du contenu jusqu'à ce qu'AngularJS prenne le contrôle.

`ng-click` Exécute une méthode ou une expression lorsque l'utilisateur clique `ng-click` un élément.

`ng-controller` Associe une classe de contrôleur à la vue.

`ng-disabled` Contrôle la propriété désactivée de l'élément de formulaire

`ng-form` Définit un formulaire

`ng-href` dynamiquement les variables AngularJS à l'attribut href.

`ng-include` Utilisé pour récupérer, compiler et inclure un fragment HTML externe sur votre page.

`ng-if` Supprime ou recrée un élément dans le DOM en fonction d'une expression

`ng-switch` Commande de `ng-switch` conditionnelle en fonction de l'expression correspondante.

`ng-model` Lie les éléments input, select, textarea etc avec la propriété model.

`ng-readonly` Utilisé pour définir l'attribut readonly sur un élément.

`ng-repeat` Permet de parcourir chaque élément d'une collection pour créer un nouveau modèle.

`ng-selected` Utilisé pour définir l'option sélectionnée dans l'élément.

`ng-show/ng-hide` Affiche / `ng-show/ng-hide` éléments en fonction d'une expression.

`ng-src` Liez dynamiquement les variables AngularJS à l'attribut src.

`ng-submit` Bind angular expressions pour soumettre des événements.

`ng-value` Lier les expressions angulaires à la valeur de.

`ng-required` Lier les expressions angulaires aux événements de soumission.

`ng-style` Définit le `ng-style` CSS sur un élément HTML.

`ng-pattern` Ajoute le validateur de modèle à ngModel.

`ng-maxlength` Ajoute le validateur maxlength à ngModel.

`ng-minlength` Ajoute le validateur de minlength à ngModel.

`ng-classeven` Fonctionne conjointement avec ngRepeat et ne prend effet que sur les lignes impaires (paires).

`ng-classodd` Fonctionne conjointement avec ngRepeat et ne prend effet que sur les lignes impaires (paires).

`ng-cut` Permet de spécifier un comportement personnalisé pour un événement coupé.

`ng-copy` Utilisé pour spécifier le comportement personnalisé lors d'un événement de copie.

`ng-paste` Utilisé pour spécifier un comportement personnalisé sur un événement de collage.

`ng-options` Utilisé pour générer dynamiquement une liste d'éléments pour l'élément.

`ng-list` Permet de convertir une chaîne en liste en fonction du délimiteur spécifié.

`ng-open` Utilisé pour définir l'attribut open sur l'élément, si l'expression à l'intérieur de ngOpen est dans la vérité.

Source (édité un peu)

ngClick

La directive `ng-click` associe un événement click à un élément DOM.

La directive `ng-click` vous permet de spécifier un comportement personnalisé lorsqu'un élément de DOM est cliqué.

Il est utile lorsque vous souhaitez joindre des événements de clic sur les boutons et les gérer sur votre contrôleur.

Cette directive accepte une expression avec l'objet événement disponible en tant que `$event`

HTML

```
<input ng-click="onClick($event)">Click me</input>
```

Manette

```
.controller("ctrl", function($scope) {
    $scope.onClick = function(evt) {
        console.debug("Hello click event: %o ",evt);
    }
})
```

HTML

```
<button ng-click="count = count + 1" ng-init="count=0">
  Increment
</button>
<span>
  count: {{count}}
</span>
```

HTML

```
<button ng-click="count()" ng-init="count=0">
  Increment
</button>
<span>
  count: {{count}}
</span>
```

Manette

```
...

$scope.count = function(){
    $scope.count = $scope.count + 1;
}
...
```

Lorsque l'utilisateur clique sur le bouton, un appel de la fonction `onClick` "Hello click event" suivi de l'objet événement.

ngRequired

Le `ng-required` ajoute ou supprime l' `required` attribut de validation sur un élément, qui à son tour activer et désactiver le `require` clé de validation pour l' `input` .

Il est utilisé pour définir éventuellement si un élément d' `input` doit avoir une valeur non vide. La directive est utile lors de la conception de la validation sur des formulaires HTML complexes.

HTML

```
<input type="checkbox" ng-model="someBooleanValue">
<input type="text" ng-model="username" ng-required="someBooleanValue">
```

ng-model-options

`ng-model-options` permet de modifier le comportement par défaut de `ng-model` , cette directive permet d'enregistrer des événements qui se déclencheront lorsque le modèle ng sera mis à jour et d'attacher un effet de rebond.

Cette directive accepte une expression qui évaluera un objet de définition ou une référence à une valeur de portée.

Exemple:

```
<input type="text" ng-model="myValue" ng-model-options="{'debounce': 500}">
```

L'exemple ci-dessus associera un effet `myValue` rebond de 500 millisecondes sur `myValue` , ce qui entraînera la mise à jour du modèle 500 ms après que l'utilisateur aura fini de taper sur l' `input` (c'est-à-dire lorsque la mise à jour de `myValue` terminée).

Propriétés d'objet disponibles

1. `updateOn` : spécifie quel événement doit être lié à l'entrée

   ```
   ng-model-options="{ updateOn: 'blur'}" // will update on blur
   ```

2. `debounce` : spécifie un délai de quelques millisecondes vers la mise à jour du modèle

```
ng-model-options="{'debounce': 500}" // will update the model after 1/2 second
```

3. `allowInvalid` : un indicateur booléen autorisant une valeur non valide pour le modèle, contournant la validation de formulaire par défaut, ces valeurs seraient par défaut traitées comme `undefined` .

4. `getterSetter` : un indicateur booléen indiquant si le `ng-model` doit être traité comme une fonction getter / setter au lieu d'une valeur de modèle simple. La fonction va alors exécuter et renvoyer la valeur du modèle.

   Exemple:

```
<input type="text" ng-model="myFunc" ng-model-options="{'getterSetter': true}">

$scope.myFunc = function() {return "value";}
```

5. `timezone` : définit le fuseau horaire du modèle si l'entrée est de la `date` ou de l' `time` . les types

ngCloak

> La directive `ngCloak` est utilisée pour empêcher que le modèle HTML angulaire ne soit brièvement affiché par le navigateur dans sa forme brute (non compilée) pendant le chargement de votre application. - Voir la source

HTML

```
<div ng-cloak>
  <h1>Hello {{ name }}</h1>
</div>
```

`ngCloak` peut être appliqué à l'élément body, mais l'utilisation privilégiée consiste à appliquer plusieurs directives ngCloak à de petites parties de la page pour permettre un rendu progressif de la vue du navigateur.

La directive `ngCloak` n'a pas de paramètres.

Voir aussi: Prévention du scintillement

ngInclude

ng-include vous permet de déléguer le contrôle d'une partie de la page à un contrôleur spécifique. Vous pouvez le faire car la complexité de ce composant devient telle que vous souhaitez encapsuler toute la logique dans un contrôleur dédié.

Un exemple est:

```
  <div ng-include
       src="'/gridview'"
       ng-controller='gridController as gc'>
```

```
  </div>
```

Notez que le `/gridview` devra être servi par le serveur Web comme une URL distincte et légitime.

Notez également que l'attribut `src` accepte une expression angulaire. Cela pourrait être une variable ou un appel de fonction par exemple ou, comme dans cet exemple, une constante de chaîne. Dans ce cas, vous devez vous assurer d' **emballer l'URL source entre guillemets** , afin qu'elle soit évaluée comme une constante de chaîne. C'est une source commune de confusion.

Dans le `/gridview` html, vous pouvez faire référence au `gridController` comme s'il était enroulé autour de la page, par exemple:

```
<div class="row">
  <button type="button" class="btn btn-default" ng-click="gc.doSomething()"></button>
</div>
```

ngSrc

Utiliser le balisage angulaire comme `{{hash}}` dans un attribut src ne fonctionne pas correctement. Le navigateur ira chercher l'URL avec le texte littéral `{{hash}}` jusqu'à ce que Angular remplace l'expression à l'intérieur de `{{hash}}` . `ng-src` directive `ng-src` remplace l'attribut `src` origine de l'élément tag image et résout le problème

```
<div ng-init="pic = 'pic_angular.jpg'">
    <h1>Angular</h1>
    <img ng-src="{{pic}}">
</div>
```

ngPattern

La directive `ng-pattern` accepte une expression qui évalue un modèle d'expression régulière et utilise ce modèle pour valider une entrée textuelle.

Exemple:

Disons que nous voulons qu'un élément `<input>` devienne valide lorsque sa valeur (ng-model) est une adresse IP valide.

Modèle:

```
<input type="text" ng-model="ipAddr" ng-pattern="ipRegex" name="ip" required>
```

Manette:

```
$scope.ipRegex = /\b(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b/;
```

ngValue

Généralement utilisé sous `ng-repeat` ngValue est utile lors de la génération dynamique de listes de boutons radio à l'aide de ngRepeat

```
<script>
   angular.module('valueExample', [])
     .controller('ExampleController', ['$scope', function($scope) {
       $scope.names = ['pizza', 'unicorns', 'robots'];
       $scope.my = { favorite: 'unicorns' };
     }]);
</script>
 <form ng-controller="ExampleController">
   <h2>Which is your favorite?</h2>
     <label ng-repeat="name in names" for="{{name}}">
       {{name}}
       <input type="radio"
              ng-model="my.favorite"
              ng-value="name"
              id="{{name}}"
              name="favorite">
     </label>
   <div>You chose {{my.favorite}}</div>
 </form>
```

Plnkr de travail

ngCopy

La directive `ngCopy` spécifie le comportement à exécuter sur un événement de copie.

Empêcher un utilisateur de copier des données

```
<p ng-copy="blockCopy($event)">This paragraph cannot be copied</p>
```

Dans le contrôleur

```
$scope.blockCopy = function(event) {
    event.preventDefault();
    console.log("Copying won't work");
}
```

ngPaste

La directive `ngPaste` spécifie un comportement personnalisé à exécuter lorsqu'un utilisateur colle du contenu

```
<input ng-paste="paste=true" ng-init="paste=false" placeholder='paste here'>
pasted: {{paste}}
```

ngHref

ngHref est utilisé à la place de l'attribut href, si nous avons des expressions angulaires à l'intérieur

de la valeur href. La directive ngHref remplace l'attribut href d'origine d'une balise html en utilisant l'attribut href tel que tag, tag etc.

La directive ngHref s'assure que le lien n'est pas rompu même si l'utilisateur clique sur le lien avant qu'AngularJS n'ait évalué le code.

Exemple 1

```
<div ng-init="linkValue = 'http://stackoverflow.com'">
    <p>Go to <a ng-href="{{linkValue}}">{{linkValue}}</a>!</p>
</div>
```

Exemple 2 Cet exemple extrait dynamiquement la valeur href de la zone de saisie et la charge en tant que valeur href.

```
<input ng-model="value" />
<a id="link" ng-href="{{value}}">link</a>
```

Exemple 3

```
<script>
angular.module('angularDoc', [])
.controller('myController', function($scope) {
  // Set some scope value.
  // Here we set bootstrap version.
  $scope.bootstrap_version = '3.3.7';

  // Set the default layout value
  $scope.layout = 'normal';
});
</script>
<!-- Insert it into Angular Code -->
<link rel="stylesheet" ng-href="//maxcdn.bootstrapcdn.com/bootstrap/{{ bootstrap_version
}}/css/bootstrap.min.css">
<link rel="stylesheet" ng-href="layout-{{ layout }}.css">
```

ngList

La directive `ng-list` est utilisée pour convertir une chaîne délimitée depuis une entrée de texte vers un tableau de chaînes ou inversement.

La directive `ng-list` utilise un délimiteur par défaut de `", "` (espace virgule).

Vous pouvez définir le délimiteur manuellement en assignant à `ng-list` un délimiteur tel que `ng-list="; "` .

Dans ce cas, le délimiteur est défini sur un point-virgule suivi d'un espace.

Par défaut, `ng-list` a un attribut `ng-trim` qui est défini sur true. `ng-trim` quand faux, respectera l'espace blanc dans votre délimiteur. Par défaut, `ng-list` ne prend pas en compte les espaces blancs, sauf si vous définissez `ng-trim="false"` .

Exemple:

```
angular.module('test', [])
  .controller('ngListExample', ['$scope', function($scope) {
    $scope.list = ['angular', 'is', 'cool!'];
}]);
```

Un délimiteur de client est configuré pour être `;`. Et le modèle de la zone de saisie est défini sur le tableau créé dans la portée.

```
  <body ng-app="test" ng-controller="ngListExample">
    <input ng-model="list" ng-list="; " ng-trim="false">
  </body>
```

La zone de saisie affiche le contenu: `angular; is; cool!`

Lire Directives intégrées en ligne: https://riptutorial.com/fr/angularjs/topic/706/directives-integrees

Chapitre 17: Directives sur mesure

Introduction

Vous en apprendrez davantage sur la fonctionnalité de directives d'AngularJS. Vous trouverez ci-dessous des informations sur les directives, ainsi que des exemples élémentaires et avancés de leur utilisation.

Paramètres

Paramètre	Détails
portée	Propriété pour définir le champ d'application de la directive. Il peut être défini sur false, true ou comme une portée d'isolat: {@, =, <, &}.
scope: faux	La directive utilise la portée parent. Aucune portée créée pour la directive.
scope: true	La directive hérite prototypiquement de la portée parent en tant que nouvelle étendue enfant. S'il existe plusieurs directives sur le même élément demandant une nouvelle étendue, elles partageront une nouvelle étendue.
portée: { @ }	Liaison à sens unique d'une propriété de portée directive à une valeur d'attribut DOM. En tant que valeur d'attribut liée au parent, elle changera dans la portée de la directive.
scope: {=}	Liaison d'attribut bidirectionnelle qui modifie l'attribut dans le parent si l'attribut de directive change et inversement.
scope: {<}	Liaison à sens unique d'une propriété de portée directive et d'une expression d'attribut DOM. L'expression est évaluée dans le parent. Cela surveille l'identité de la valeur parente afin que les modifications apportées à une propriété d'objet dans le parent ne soient pas reflétées dans la directive. Les modifications apportées à une propriété d'objet dans une directive seront reflétées dans le parent, puisque les deux font référence au même objet
portée: { & }	Permet à la directive de transmettre des données à une expression à évaluer dans le parent.
compiler: fonction	Cette fonction est utilisée pour effectuer une transformation DOM sur le modèle de directive avant l'exécution de la fonction de lien. Il accepte `tElement` (le modèle de directive) et `tAttr` (liste des attributs déclarés sur la directive). Il n'a pas accès à la portée. Il peut renvoyer une fonction qui sera enregistrée en tant que `post-link` ou renvoyer un objet avec `pre` propriétés `pre` et `post` avec sera enregistré en tant que fonctions de `pre-`

Paramètre	Détails
	`link` et `post-link` .
lien: fonction / objet	La propriété link peut être configurée en tant que fonction ou objet. Il peut recevoir les arguments suivants: scope (directive scope), iElement (élément DOM où directive est appliquée), iAttrs (collection d'attributs d'élément DOM), controller (tableau de contrôleurs requis par directive), transcludeFn. Il est principalement utilisé pour configurer les écouteurs DOM, regarder les propriétés du modèle pour les modifications et mettre à jour le DOM. Il s'exécute après le clonage du modèle. Il est configuré indépendamment s'il n'y a pas de fonction de compilation.
fonction de pré-lien	Fonction de lien qui s'exécute avant toute fonction de lien enfant. Par défaut, les fonctions de lien de directive enfant s'exécutent avant les fonctions de lien de directive parent et la fonction de pré-lien permet au parent de créer le premier lien. Un cas d'utilisation est si l'enfant a besoin de données du parent.
fonction post-lien	Fonction de lien que les cadres après les éléments enfants sont liés au parent. Il est couramment utilisé pour attacher des gestionnaires d'événements et accéder aux directives enfants, mais les données requises par la directive enfant ne doivent pas être définies ici car la directive enfant a déjà été liée.
restrict: string	Définit comment appeler la directive depuis le DOM. Valeurs possibles (en supposant que le nom de notre directive est `demoDirective`): `E` - Nom de l'élément (`<demo-directive></demo-directive>`), `A` - Attribut (`<div demo-directive></div>`), `C` - Classe correspondante (`<div class="demo-directive"></div>`), `M` - Par commentaire (`<!-- directive: demo-directive -->`). La propriété `restrict` peut également prendre en charge plusieurs options, par exemple - `restrict: "AC"` restreindra la directive à *Attribute* OR *Class* . Si elle est omise, la valeur **par défaut** est `"EA"` (élément ou attribut).
exiger: 'demoDirective'	Localisez le contrôleur de demoDirective sur l'élément actuel et injectez son contrôleur comme quatrième argument de la fonction de liaison. Jeter une erreur si non trouvé.
require: '? demoDirective'	Essayez de localiser le contrôleur de demoDirective ou transmettez null au lien fn si non trouvé.
exiger: '^ demoDirective'	Localisez le contrôleur de demoDirective en recherchant l'élément et ses parents. Jeter une erreur si non trouvé.
require: '^^ demoDirective'	Localisez le contrôleur de demoDirective en recherchant les parents de l'élément. Jeter une erreur si non trouvé.
exiger: '? ^	Essayez de localiser le contrôleur de demoDirective en recherchant

Paramètre	Détails
demoDirective'	l'élément et ses parents ou transmettez null au lien fn s'il n'est pas trouvé.
exiger: '? ^^ demoDirective'	Essayez de localiser le contrôleur de demoDirective en recherchant les parents de l'élément, ou passez null au lien fn s'il n'est pas trouvé.

Examples

Création et consommation de directives personnalisées

Les directives sont l'une des fonctionnalités les plus puissantes d'angularjs. Les directives angularjs personnalisées sont utilisées pour étendre les fonctionnalités de HTML en créant de nouveaux éléments HTML ou des attributs personnalisés pour fournir certains comportements à une balise HTML.

directive.js

```
// Create the App module if you haven't created it yet
var demoApp= angular.module("demoApp", []);

// If you already have the app module created, comment the above line and create a reference
of the app module
var demoApp = angular.module("demoApp");


// Create a directive using the below syntax
// Directives are used to extend the capabilities of html element
// You can either create it as an Element/Attribute/class
// We are creating a directive named demoDirective. Notice it is in CamelCase when we are
defining the directive just like ngModel
// This directive will be activated as soon as any this element is encountered in html

demoApp.directive('demoDirective', function () {

  // This returns a directive definition object
  // A directive definition object is a simple JavaScript object used for configuring the
directive’s behaviour,template..etc
  return {
    // restrict: 'AE', signifies that directive is Element/Attribute directive,
    // "E" is for element, "A" is for attribute, "C" is for class, and "M" is for comment.
    // Attributes are going to be the main ones as far as adding behaviors that get used the
most.
    // If you don't specify the restrict property it will default to "A"
    restrict :'AE',

    // The values of scope property decides how the actual scope is created and used inside a
directive. These values can be either “false”, “true” or “{}”. This creates an isolate scope
for the directive.
    // '@' binding is for passing strings. These strings support {{}} expressions for
interpolated values.
    // '=' binding is for two-way model binding. The model in parent scope is linked to the
model in the directive's isolated scope.
    // '&' binding is for passing a method into your directive's scope so that it can be
called within your directive.
    // The method is pre-bound to the directive's parent scope, and supports arguments.
```

```
    scope: {
      name: "@",  // Always use small casing here even if it's a mix of 2-3 words
    },

    // template replaces the complete element with its text.
    template: "<div>Hello {{name}}!</div>",

    // compile is called during application initialization. AngularJS calls it once when html
page is loaded.
    compile: function(element, attributes) {
      element.css("border", "1px solid #cccccc");

      // linkFunction is linked with each element with scope to get the element specific data.
      var linkFunction = function($scope, element, attributes) {
        element.html("Name: <b>"+$scope.name +"</b>");
        element.css("background-color", "#ff00ff");
      };
      return linkFunction;
    }
  };
});
```

Cette directive peut ensuite être utilisée dans App en tant que:

```
<html>

   <head>
      <title>Angular JS Directives</title>
   </head>
   <body>
   <script src =
"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
   <script src="directive.js"></script>
    <div ng-app = "demoApp">
       <!-- Notice we are using Spinal Casing here -->
       <demo-directive name="World"></demo-directive>

      </div>
   </body>
</html>
```

Modèle d'objet de définition de directive

```
demoApp.directive('demoDirective', function () {
  var directiveDefinitionObject = {
    multiElement:
    priority:
    terminal:
    scope: {},
    bindToController: {},
    controller:
    controllerAs:
    require:
    restrict:
    templateNamespace:
    template:
    templateUrl:
    transclude:
```

```
    compile:
    link: function(){}
  };
  return directiveDefinitionObject;
});
```

1. `multiElement` - défini sur true et tous les noeuds DOM entre le début et la fin du nom de la directive seront collectés et regroupés en éléments de directive
2. `priority` - permet de spécifier l'ordre à appliquer aux directives lorsque plusieurs directives sont définies sur un seul élément DOM. Les directives avec des nombres plus élevés sont compilées en premier.
3. `terminal` - défini sur true et la priorité actuelle sera le dernier ensemble de directives à exécuter
4. `scope` - définit le champ d'application de la directive
5. `bind to controller` - lie les propriétés de l'étendue directement au contrôleur de la directive
6. `controller` - fonction constructeur du contrôleur
7. `require` - nécessite une autre directive et injecte son contrôleur comme quatrième argument de la fonction de liaison
8. `controllerAs` - référence de nom au contrôleur dans la portée de la directive pour permettre au contrôleur d'être référencé à partir du modèle de directive.
9. `restrict` - restreint la directive à Element, Attribute, Class ou Comment
10. `templateNameSpace` - définit le type de document utilisé par le modèle de directive: html, svg ou math. html est la valeur par défaut
11. `template` - balisage HTML qui remplace par défaut le contenu de l'élément de la directive, ou encapsule le contenu de l'élément directive si transclude est true
12. `templateUrl` - URL fournie de manière asynchrone pour le modèle
13. `transclude` - Extrait le contenu de l'élément où la directive apparaît et le met à la disposition de la directive. Le contenu est compilé et fourni à la directive en tant que fonction de transclusion.
14. `compile` - fonction pour transformer le modèle DOM
15. `link` - uniquement utilisé si la propriété compile n'est pas définie. La fonction de lien est responsable de l'enregistrement des écouteurs DOM ainsi que de la mise à jour du DOM. Il est exécuté après le clonage du modèle.

Exemple de directive de base

superman-directive.js

```
angular.module('myApp', [])
  .directive('superman', function() {
    return {
      // restricts how the directive can be used
      restrict: 'E',
      templateUrl: 'superman-template.html',
      controller: function() {
        this.message = "I'm superman!"
      },
      controllerAs: 'supermanCtrl',
      // Executed after Angular's initialization. Use commonly
      // for adding event handlers and DOM manipulation
```

```
      link: function(scope, element, attributes) {
        element.on('click', function() {
          alert('I am superman!')
        });
      }
    }
  });
```

superman-template.html

```
<h2>{{supermanCtrl.message}}</h2>
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular.js"></script>
  <script src="superman-directive.js"><script/>
</head>
<body>
<div ng-app="myApp">
  <superman></superman>
</div>
</body>
</html>
```

Vous pouvez en savoir plus sur les fonctions de `restrict` et de `link` de la directive sur la documentation officielle d'AngularJS sur les directives.

Comment créer des composants utilisables à l'aide de directives

Les directives AngularJS contrôlent le rendu du code HTML dans une application AngularJS. Ils peuvent être un élément HTML, un attribut, une classe ou un commentaire. Les directives sont utilisées pour manipuler le DOM, en attachant un nouveau comportement aux éléments HTML, à la liaison de données et à bien d'autres encore. Certains exemples de directives fournies par angulaire sont ng-model, ng-hide, ng-if.

De même, on peut créer sa propre directive personnalisée et la rendre utilisable. Pour créer une référence de directives personnalisées. Le sens derrière la création de directives réutilisables est de faire un ensemble de directives / composants écrit par vous comme angularjs nous fournit angular.js. Ces directives réutilisables peuvent s'avérer particulièrement utiles lorsque vous disposez d'une suite d'applications / d'applications nécessitant un comportement, une apparence et une convivialité cohérents. Un exemple d'un tel composant réutilisable peut être une barre d'outils simple que vous pouvez utiliser dans votre application ou dans différentes applications, mais que vous souhaitez qu'ils se comportent de la même manière.

Tout d'abord, créez un dossier nommé resutableComponents dans votre dossier d'application et créez le fichier refrableModuleApp.js

reusableModuleApp.js:

```
(function(){

 var reusableModuleApp = angular.module('resuableModuleApp', ['ngSanitize']);

 //Remember whatever dependencies you have in here should be injected in the app module where
it is intended to be used or it's scripts should be included in your main app
                               //We will be injecting ng-sanitize

 resubaleModuleApp.directive('toolbar', toolbar)

 toolbar.$inject=['$sce'];

 function toolbar($sce){

     return{
     restrict :'AE',
        //Defining below isolate scope actually provides window for the directive to take data
from app that will be using this.
        scope : {
                value1: '=',
                value2: '=',
                },

         }
     template : '<ul>  <li><a ng-click="Add()" href="">{{value1}}</a></li>  <li><a ng-
click="Edit()" href="#">{{value2}}</a></li> </ul> ',
         link : function(scope, element, attrs){

              //Handle's Add function
              scope.Add = function(){


              };

              //Handle's Edit function
              scope.Edit = function(){

              };
     }
  }

});
```

mainApp.js:

```
(function(){
   var mainApp = angular.module('mainApp', ['reusableModuleApp']); //Inject resuableModuleApp
in your application where you want to use toolbar component

   mainApp.controller('mainAppController', function($scope){
      $scope.value1 = "Add";
      $scope.value2 = "Edit";

   });

 });
```

index.html:

```
<!doctype html>
<html ng-app="mainApp">
<head>
 <title> Demo Making a reusable component
<head>
 <body ng-controller="mainAppController">

    <!-- We are providing data to toolbar directive using mainApp'controller -->
    <toolbar value1="value1" value2="value2"></toolbar>

   <!-- We need to add the dependent js files on both apps here -->
    <script src="js/angular.js"></script>
    <script src="js/angular-sanitize.js"></script>

    <!-- your mainApp.js should be added afterwards --->
     <script src="mainApp.js"></script>

    <!-- Add your reusable component js files here -->
     <script src="resuableComponents/reusableModuleApp.js"></script>

 </body>
</html>
```

Directive sont des composants réutilisables par défaut. Lorsque vous créez des directives dans un module angulaire distinct, vous pouvez l'exporter et le réutiliser dans différentes applications angulaires. De nouvelles directives peuvent simplement être ajoutées à l'intérieur de reusableModuleApp.js et reusableModuleApp peut avoir son propre contrôleur, service, objet DDO dans la directive pour définir le comportement.

Directive de base avec modèle et une portée isolée

Création d' une directive personnalisée avec une *portée isolée* séparera le champ d' application **dans** la directive du champ **extérieur,** afin d'éviter que notre directive de changer accidentellement les données dans le champ parent et restreindre la lecture des données privées de la portée des parents.

Pour créer une étendue isolée tout en permettant à notre directive personnalisée de communiquer avec la portée externe, nous pouvons utiliser l'option `scope` qui décrit comment **mapper** les liaisons de la portée interne de la directive avec la portée externe.

Les liaisons réelles sont réalisées avec des **attributs** supplémentaires attachés à la directive. Les paramètres de liaison sont définis avec l'option `scope` et un objet avec des paires clé-valeur:

- Une **clé** qui correspond à la propriété de portée isolée de notre directive
- Une **valeur** indiquant à Angular comment lier la portée interne de la directive à un **attribut** correspondant

Exemple simple d'une directive avec une portée isolée:

```
var ProgressBar = function() {
```

```
  return {
    scope: { // This is how we define an isolated scope
      current: '=', // Create a REQUIRED bidirectional binding by using the 'current' 
attribute
      full: '=?maxValue' // Create an OPTIONAL (Note the '?'): bidirectional binding using 
'max-value' attribute to the `full` property in our directive isolated scope
    }
    template: '<div class="progress-back">' +
              '  <div class="progress-bar"' +
              '       ng-style="{width: getProgress()}">' +
              '  </div>' +
              '</div>',
    link: function(scope, el, attrs) {
      if (scope.full === undefined) {
        scope.full = 100;
      }
      scope.getProgress = function() {
        return (scope.current / scope.size * 100) + '%';
      }
    }
  }
}

ProgressBar.$inject = [];
angular.module('app').directive('progressBar', ProgressBar);
```

Exemple d'utilisation de cette directive et liaison de données depuis la portée du contrôleur vers la portée interne de la directive:

Manette:

```
angular.module('app').controller('myCtrl', function($scope) {
    $scope.currentProgressValue = 39;
    $scope.maxProgressBarValue = 50;
});
```

Vue:

```
<div ng-controller="myCtrl">
    <progress-bar current="currentProgressValue"></progress-bar>
    <progress-bar current="currentProgressValue" max-value="maxProgressBarValue"></progress-
bar>
</div>
```

Construire un composant réutilisable

Les directives peuvent être utilisées pour créer des composants réutilisables. Voici un exemple de composant "utilisateur":

userBox.js

```
angular.module('simpleDirective', []).directive('userBox', function() {
  return {
    scope: {
        username: '=username',
```

```
        reputation: '=reputation'
    },
    templateUrl: '/path/to/app/directives/user-box.html'
  };
});
```

Controller.js

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

myApp.controller('Controller', function($scope) {

    $scope.user = "John Doe";
    $scope.rep = 1250;

    $scope.user2 = "Andrew";
    $scope.rep2 = 2850;

});
```

myPage.js

```
<html lang="en" ng-app="myApp">
  <head>
    <script src="/path/to/app/angular.min.js"></script>
    <script src="/path/to/app/js/controllers/Controller.js"></script>
    <script src="/path/to/app/js/directives/userBox.js"></script>
  </head>

  <body>

    <div ng-controller="Controller">
        <user-box username="user" reputation="rep"></user-box>
        <user-box username="user2" reputation="rep2"></user-box>
    </div>

  </body>
</html>
```

user-box.html

```
<div>{{username}}</div>
<div>{{reputation}} reputation</div>
```

Le résultat sera:

```
John Doe
1250 reputation
Andrew
2850 reputation
```

Décorateur de directive

Parfois, vous pouvez avoir besoin de fonctionnalités supplémentaires d'une directive. Au lieu de

réécrire (copier) la directive, vous pouvez modifier le comportement de la directive.

Le décorateur sera exécuté pendant la phase $ inject.

Pour ce faire, fournissez un fichier .config à votre module. La directive s'appelle myDirective, vous devez donc configurer myDirectiveDirective. (ceci dans une convention angulaire [lire à propos des fournisseurs]).

Cet exemple changera le templateUrl de la directive:

```
angular.module('myApp').config(function($provide){
        $provide.decorator('myDirectiveDirective', function($delegate){
             var directive = $delegate[0]; // this is the actual delegated, your directive
             directive.templateUrl = 'newTemplate.html'; // you change the directive template
        return $delegate;
    })
});
```

Cet exemple ajoute un événement onClick à l'élément directive lorsque l'utilisateur clique dessus, cela se produit pendant la phase de compilation.

```
angular.module('myApp').config(function ($provide) {
        $provide.decorator('myDirectiveTwoDirective', function ($delegate) {
            var directive = $delegate[0];
            var link = directive.link; // this is directive link phase
            directive.compile = function () { // change the compile of that directive
                return function (scope, element, attrs) {
                    link.apply(this, arguments); // apply this at the link phase
                    element.on('click', function(){ // when add an onclick that log hello when
the directive is clicked.
                            console.log('hello!');
                    });
                };
            };
            return $delegate;
        });

    });
```

Une approche similaire peut être utilisée pour les fournisseurs et les services.

Héritage de directive et interopérabilité

Les directives angulaires js peuvent être imbriquées ou être interopérables.

Dans cet exemple, la directive Adir expose à la directive Bdir la portée de son contrôleur $, car Bdir nécessite Adir.

```
angular.module('myApp',[]).directive('Adir', function () {
        return {
            restrict: 'AE',
            controller: ['$scope', function ($scope) {
                    $scope.logFn = function (val) {
                            console.log(val);
```

```
                        }
                    }]
                }
        })
```

Assurez-vous de définir require: '^ Adir' (regardez la documentation angulaire, certaines versions ne nécessitent pas ^ character).

```
.directive('Bdir', function () {
        return {
            restrict: 'AE',
            require: '^Adir', // Bdir require Adir
            link: function (scope, elem, attr, Parent) {
            // Parent is Adir but can be an array of required directives.
                elem.on('click', function ($event) {
                    Parent.logFn("Hello!"); // will log "Hello! at parent dir scope
                    scope.$apply(); // apply to parent scope.
                });
            }
        }
    }]);
```

Vous pouvez imbriquer votre directive de cette manière:

```
<div a-dir><span b-dir></span></div>
<a-dir><b-dir></b-dir> </a-dir>
```

Il n'est pas obligatoire que les directives soient imbriquées dans votre code HTML.

Lire Directives sur mesure en ligne: https://riptutorial.com/fr/angularjs/topic/965/directives-sur-mesure

Chapitre 18: Directives utilisant ngModelController

Examples

Un contrôle simple: note

Construisons un contrôle simple, un widget d'évaluation, destiné à être utilisé comme:

```
<rating min="0" max="5" nullifier="true" ng-model="data.rating"></rating>
```

Pas de CSS sophistiqué pour l'instant; cela se traduirait par:

```
0 1 2 3 4 5 x
```

En cliquant sur un nombre, vous sélectionnez cette cote. et en cliquant sur le "x", la note est définie sur null.

```
app.directive('rating', function() {

    function RatingController() {
        this._ngModel = null;
        this.rating = null;
        this.options = null;
        this.min = typeof this.min === 'number' ? this.min : 1;
        this.max = typeof this.max === 'number' ? this.max : 5;
    }

    RatingController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            // KEY POINT 1
            ngModel.$render = this._render.bind(this);
        }
    };

    RatingController.prototype._render = function() {
        this.rating = this._ngModel.$viewValue != null ? this._ngModel.$viewValue : -
Number.MAX_VALUE;
    };

    RatingController.prototype._calculateOptions = function() {
        if( this.min == null || this.max == null ) {
            this.options = [];
        }
        else {
            this.options = new Array(this.max - this.min + 1);
            for( var i=0; i < this.options.length; i++ ) {
                this.options[i] = this.min + i;
            }
        }
```

```
    };

    RatingController.prototype.setValue = function(val) {
        this.rating = val;
        // KEY POINT 2
        this._ngModel.$setViewValue(val);
    };

    // KEY POINT 3
    Object.defineProperty(RatingController.prototype, 'min', {
        get: function() {
            return this._min;
        },
        set: function(val) {
            this._min = val;
            this._calculateOptions();
        }
    });

    Object.defineProperty(RatingController.prototype, 'max', {
        get: function() {
            return this._max;
        },
        set: function(val) {
            this._max = val;
            this._calculateOptions();
        }
    });

    return {
        restrict: 'E',
        scope: {
            // KEY POINT 3
            min: '<?',
            max: '<?',
            nullifier: '<?'
        },
        bindToController: true,
        controllerAs: 'ctrl',
        controller: RatingController,
        require: ['rating', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<span ng-repeat="o in ctrl.options" href="#" class="rating-option" ng-
class="{\'rating-option-active\': o <= ctrl.rating}" ng-click="ctrl.setValue(o)">{{ o
}}</span>' +
            '<span ng-if="ctrl.nullifier" ng-click="ctrl.setValue(null)" class="rating-
nullifier">&#10006;</span>'
    };
});
```

Points clés:

1. Implémentez `ngModel.$render` pour transférer la *valeur de vue* du modèle à votre vue.
2. Appelez `ngModel.$setViewValue()` chaque fois que vous estimez que la valeur de la vue doit être mise à jour.
3. Le contrôle peut bien entendu être paramétré; utilisez `'<'` liaisons de portée `'<'` pour les

paramètres, si dans Angular> = 1.5 pour indiquer clairement l'entrée - liaison unidirectionnelle. Si vous devez agir chaque fois qu'un paramètre change, vous pouvez utiliser une propriété JavaScript (voir `Object.defineProperty()`) pour enregistrer quelques montres.

Note 1: Afin de ne pas compliquer l'implémentation, les valeurs de notation sont insérées dans un tableau - les `ctrl.options` . Ce n'est pas nécessaire une implémentation plus efficace, mais aussi plus complexe, pourrait utiliser la manipulation DOM pour insérer / supprimer des notations lorsque le changement `min` / `max` .

Remarque 2: À l'exception des liaisons de portée `'<'` , cet exemple peut être utilisé dans Angular <1.5. Si vous êtes sur Angular> = 1.5, ce serait une bonne idée de transformer ceci en composant et d'utiliser le hook de cycle de vie `$onInit()` pour initialiser `min` et `max` , au lieu de le faire dans le constructeur du contrôleur.

Et un violon nécessaire: https://jsfiddle.net/h81mgxma/

Un couple de contrôles complexes: éditer un objet complet

Un contrôle personnalisé ne doit pas se limiter à des choses triviales comme les primitives; il peut éditer des choses plus intéressantes. Nous présentons ici deux types de contrôles personnalisés, l'un pour l'édition des personnes et l'autre pour l'édition des adresses. Le contrôle d'adresse est utilisé pour modifier l'adresse de la personne. Un exemple d'utilisation serait:

```
<input-person ng-model="data.thePerson"></input-person>
<input-address ng-model="data.thePerson.address"></input-address>
```

Le modèle de cet exemple est volontairement simpliste:

```
function Person(data) {
  data = data || {};
  this.name = data.name;
  this.address = data.address ? new Address(data.address) : null;
}

function Address(data) {
  data = data || {};
  this.street = data.street;
  this.number = data.number;
}
```

L'éditeur d'adresse:

```
app.directive('inputAddress', function() {

    InputAddressController.$inject = ['$scope'];
    function InputAddressController($scope) {
        this.$scope = $scope;
        this._ngModel = null;
        this.value = null;
        this._unwatch = angular.noop;
    }
```

```
    InputAddressController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            // KEY POINT 3
            ngModel.$render = this._render.bind(this);
        }
    };

    InputAddressController.prototype._makeWatch = function() {
        // KEY POINT 1
        this._unwatch = this.$scope.$watchCollection(
            (function() {
                return this.value;
            }).bind(this),
            (function(newval, oldval) {
                if( newval !== oldval ) { // skip the initial trigger
                    this._ngModel.$setViewValue(newval !== null ? new Address(newval) : null);
                }
            }).bind(this)
        );
    };

    InputAddressController.prototype._render = function() {
        // KEY POINT 2
        this._unwatch();
        this.value = this._ngModel.$viewValue ? new Address(this._ngModel.$viewValue) : null;
        this._makeWatch();
    };

    return {
        restrict: 'E',
        scope: {},
        bindToController: true,
        controllerAs: 'ctrl',
        controller: InputAddressController,
        require: ['inputAddress', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<div>' +
                '<label><span>Street:</span><input type="text" ng-model="ctrl.value.street"
/></label>' +
                '<label><span>Number:</span><input type="text" ng-model="ctrl.value.number"
/></label>' +
            '</div>'
    };
});
```

Points clés:

1. Nous modifions un objet; nous ne voulons pas changer directement l'objet donné par notre société mère (nous voulons que notre modèle soit compatible avec le principe d'immuabilité). Nous créons donc une montre peu profonde sur l'objet en cours d'édition et `$setViewValue()` jour le modèle avec `$setViewValue()` chaque fois qu'une propriété change. Nous transmettons une *copie* à notre parent.
2. Chaque fois que le modèle change de l'extérieur, nous le copions et sauvegardons la copie

dans notre champ d'application. Les principes d'immuabilité encore une fois, bien que la copie interne ne soit pas immuable, l'externe pourrait très bien l'être. De plus, nous reconstruisons la montre (`this_unwatch();this._makeWatch();`), pour éviter que l'observateur ne subisse les modifications que le modèle nous a `this_unwatch();this._makeWatch();` . (Nous voulons seulement que la montre se déclenche pour les modifications apportées dans l'interface utilisateur.)
3. Autre que les points ci-dessus, nous implémentons `ngModel.$render()` et appelons `ngModel.$setViewValue()` comme nous le ferions pour un contrôle simple (voir l'exemple de notation).

Le code du contrôle personnalisé de la personne est presque identique. Le modèle utilise `<input-address>` . Dans une implémentation plus avancée, nous pourrions extraire les contrôleurs dans un module réutilisable.

```
app.directive('inputPerson', function() {

    InputPersonController.$inject = ['$scope'];
    function InputPersonController($scope) {
        this.$scope = $scope;
        this._ngModel = null;
        this.value = null;
        this._unwatch = angular.noop;
    }

    InputPersonController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            ngModel.$render = this._render.bind(this);
        }
    };

    InputPersonController.prototype._makeWatch = function() {
        this._unwatch = this.$scope.$watchCollection(
            (function() {
                return this.value;
            }).bind(this),
            (function(newval, oldval) {
                if( newval !== oldval ) { // skip the initial trigger
                    this._ngModel.$setViewValue(newval !== null ? new Person(newval) : null);
                }
            }).bind(this)
        );
    };

    InputPersonController.prototype._render = function() {
        this._unwatch();
        this.value = this._ngModel.$viewValue ? new Person(this._ngModel.$viewValue) : null;
        this._makeWatch();
    };

    return {
        restrict: 'E',
        scope: {},
        bindToController: true,
        controllerAs: 'ctrl',
        controller: InputPersonController,
```

```
        require: ['inputPerson', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<div>' +
                '<label><span>Name:</span><input type="text" ng-model="ctrl.value.name" 
/></label>' +
                '<input-address ng-model="ctrl.value.address"></input-address>' +
            '</div>'
    };
});
```

Note: Ici, les objets sont typés, c’est-à-dire qu’ils ont des constructeurs appropriés. Ce n'est pas obligatoire le modèle peut être des objets JSON simples. Dans ce cas, utilisez simplement `angular.copy()` place des constructeurs. Un avantage supplémentaire est que le contrôleur devient identique pour les deux contrôles et peut facilement être extrait dans un module commun.

Le violon: https://jsfiddle.net/3tzyqfko/2/

Deux versions du violon ayant extrait le code commun des contrôleurs: https://jsfiddle.net/agj4cp0e/ et https://jsfiddle.net/ugb6Lw8b/

Lire Directives utilisant ngModelController en ligne: https://riptutorial.com/fr/angularjs/topic/2438/directives-utilisant-ngmodelcontroller

Chapitre 19: Distinguer Service vs Usine

Examples

Usine VS Service une fois pour toutes

Par définition:

Les services sont essentiellement des fonctions de constructeur. Ils utilisent le mot-clé 'this'.

Les usines sont des fonctions simples et renvoient donc un objet.

Sous la capuche:

Les usines appellent en interne la fonction de fournisseur.

Les services appellent en interne la fonction Usine.

Débat:

Les usines peuvent exécuter du code avant de retourner notre littéral d'objet.

Mais en même temps, les services peuvent également être écrits pour renvoyer un littéral d'objet et pour exécuter du code avant de retourner. Bien que cela soit contre-productif, les services sont conçus pour jouer le rôle de constructeur.

En fait, les fonctions de constructeur en JavaScript peuvent renvoyer ce qu'elles veulent.

Alors quel est le meilleur?

La syntaxe de constructeur des services est plus proche de la syntaxe de classe de ES6. La migration sera donc facile.

Résumé

Donc, en résumé, fournisseur, usine et service sont tous des fournisseurs.

Une fabrique est un cas particulier de fournisseur lorsque tout ce dont vous avez besoin dans votre fournisseur est une fonction $ get (). Il vous permet de l'écrire avec moins de code.

Un service est un cas particulier d'une fabrique lorsque vous souhaitez renvoyer une instance d'un nouvel objet, avec le même avantage d'écrire moins de code.

```
mod.provider("myProvider", fun
    this.$get = function() {
        return new function()
            this.getValue = fu
                return "My Val
            };
        };
    };
});
```

Lire Distinguer Service vs Usine en ligne: https://riptutorial.com/fr/angularjs/topic/7099/distinguer-service-vs-usine

Chapitre 20: Événements

Paramètres

Paramètres	Types de valeurs
un événement	Object {name: "eventName", targetScope: Scope, defaultPrevented: false, currentScope: ChildScope}
args	les données qui ont été transmises avec l'exécution de l'événement

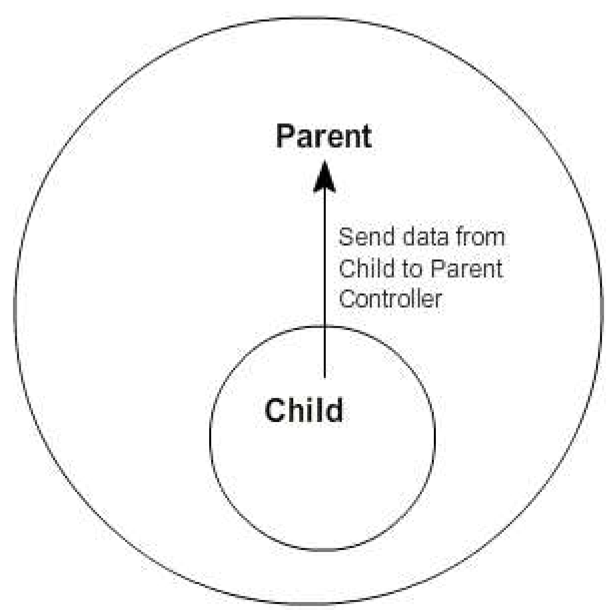
Examples

Utiliser un système d'événements angulaires

$ scope. $ emit

Utiliser `$scope.$emit` déclenchera un nom d'événement vers le haut dans la hiérarchie de la portée et notifiera la `$scope` .Le cycle de vie de l'événement commence à la portée sur laquelle `$emit` été appelé.

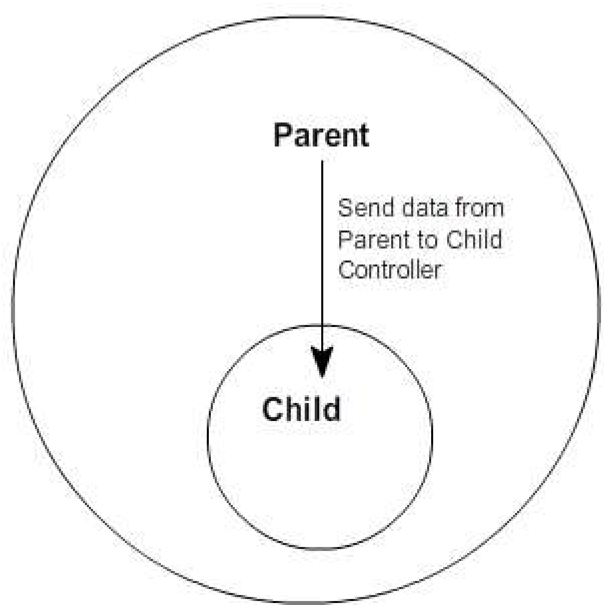
Fil de fer de travail:



$ scope. $ broadcast

Utiliser `$scope.$broadcast` déclenchera un événement dans la `$scope` . Nous pouvons écouter ces événements en utilisant `$scope.$on`

Fil de fer de travail:



Syntaxe:

```
// firing an event upwards
$scope.$emit('myCustomEvent', 'Data to send');

// firing an event downwards
$scope.$broadcast('myCustomEvent', {
  someProp: 'some value'
});

// listen for the event in the relevant $scope
$scope.$on('myCustomEvent', function (event, data) {
  console.log(data); // 'Data from the event'
});
```

Au lieu de `$scope` vous pouvez utiliser `$rootScope` . Dans ce cas, votre événement sera disponible dans tous les contrôleurs, quelle que soit la portée de ces contrôleurs.

Evénement enregistré propre à AngularJS

La raison de nettoyer les événements enregistrés, car même le contrôleur a été détruit, la gestion des événements enregistrés est toujours d'actualité. Le code fonctionnera donc de manière inattendue.

```
// firing an event upwards
$rootScope.$emit('myEvent', 'Data to send');

// listening an event
var listenerEventHandler = $rootScope.$on('myEvent', function(){
    //handle code
});

$scope.$on('$destroy', function() {
    listenerEventHandler();
});
```

Usages et signification

Ces événements peuvent être utilisés pour communiquer entre 2 ou plusieurs contrôleurs.

`$emit` distribue un événement vers le haut dans la hiérarchie de la portée, tandis que `$broadcast` distribue un événement vers le bas dans toutes les portées enfants. Ceci a été magnifiquement expliqué ici .

Il peut exister deux types de scénarios lors de la communication entre contrôleurs:

---

1. Lorsque les contrôleurs ont une relation parent-enfant. (nous pouvons principalement utiliser `$scope` dans de tels scénarios)

---

2. Lorsque les contrôleurs ne sont pas indépendants les uns des autres et doivent être informés de l'activité de chacun. (nous pouvons utiliser `$rootScope` dans de tels scénarios)

Par exemple: pour tout site de commerce électronique, supposons que nous ayons `ProductListController` (qui contrôle la page de liste des produits lorsque l'utilisateur clique sur une marque) et `CartController` (pour gérer les éléments du panier). Maintenant, lorsque nous cliquons sur le bouton **Ajouter au panier** , celui-ci doit également être informé par `CartController` , afin qu'il puisse refléter le nombre / les détails des nouveaux éléments du panier dans la barre de navigation du site Web. Cela peut être réalisé en utilisant `$rootScope` .

Avec `$scope.$emit`

```
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"></script>
    <script>
        var app = angular.module('app', []);

        app.controller("FirstController", function ($scope) {
            $scope.$on('eventName', function (event, args) {
```

```
                $scope.message = args.message;
            });
        });

        app.controller("SecondController", function ($scope) {
            $scope.handleClick = function (msg) {
                $scope.$emit('eventName', {message: msg});
            };
        });

    </script>
  </head>
  <body ng-app="app">
    <div ng-controller="FirstController" style="border:2px ;padding:5px;">
        <h1>Parent Controller</h1>
        <p>Emit Message : {{message}}</p>
        <br />
        <div ng-controller="SecondController" style="border:2px;padding:5px;">
            <h1>Child Controller</h1>
            <input ng-model="msg">
            <button ng-click="handleClick(msg);">Emit</button>
        </div>
    </div>
  </body>
</html>
```

Avec `$scope.$broadcast` :

```
<html>
  <head>
    <title>Broadcasting</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"></script>
    <script>
        var app = angular.module('app', []);

        app.controller("FirstController", function ($scope) {
            $scope.handleClick = function (msg) {
                $scope.$broadcast('eventName', {message: msg});
            };

        });

        app.controller("SecondController", function ($scope) {
            $scope.$on('eventName', function (event, args) {
                $scope.message = args.message;
            });
        });

    </script>
  </head>
  <body ng-app="app">
    <div ng-controller="FirstController" style="border:2px solid ; padding:5px;">
        <h1>Parent Controller</h1>
        <input ng-model="msg">
        <button ng-click="handleClick(msg);">Broadcast</button>
        <br /><br />
        <div ng-controller="SecondController" style="border:2px solid ;padding:5px;">
            <h1>Child Controller</h1>
            <p>Broadcast Message : {{message}}</p>
        </div>
```

```
    </div>
  </body>
</html>
```

Toujours désinscrire $ rootScope. $ Sur les écouteurs de l'événement scope $ destory

$ rootScope. $ sur les écouteurs restera en mémoire si vous naviguez vers un autre contrôleur. Cela créera une fuite de mémoire si le contrôleur est hors de portée.

Ne pas

```
angular.module('app').controller('badExampleController', badExample);

badExample.$inject = ['$scope', '$rootScope'];
function badExample($scope, $rootScope) {

    $rootScope.$on('post:created', function postCreated(event, data) {});

}
```

Faire

```
angular.module('app').controller('goodExampleController', goodExample);

goodExample.$inject = ['$scope', '$rootScope'];
function goodExample($scope, $rootScope) {

    var deregister = $rootScope.$on('post:created', function postCreated(event, data) {});

    $scope.$on('$destroy', function destroyScope() {
        deregister();
    });

}
```

Lire Événements en ligne: https://riptutorial.com/fr/angularjs/topic/1922/evenements

Chapitre 21: Filtres personnalisés

Examples

Exemple de filtre simple

Les filtres formulent la valeur d'une expression à afficher pour l'utilisateur. Ils peuvent être utilisés dans des modèles de vue, des contrôleurs ou des services. Cet exemple crée un filtre (`addZ`) puis l'utilise dans une vue. Tout ce que fait ce filtre est d'ajouter un «Z» majuscule à la fin de la chaîne.

example.js

```
angular.module('main', [])
    .filter('addZ', function() {
            return function(value) {
                return value + "Z";
            }
     })
    .controller('MyController', ['$scope', function($scope) {
        $scope.sample = "hello";
    }])
```

exemple.html

Dans la vue, le filtre est appliqué avec la syntaxe suivante: `{ variable | filter}` . Dans ce cas, la variable que nous avons définie dans le contrôleur, `sample` , est filtrée par le filtre que nous avons créé, `addZ` .

```
<div ng-controller="MyController">
   <span>{{sample | addZ}}</span>
</div>
```

Production attendue

```
helloZ
```

Utiliser un filtre dans un contrôleur, un service ou un filtre

Vous devrez injecter `$filter` :

```
angular
  .module('filters', [])
  .filter('percentage', function($filter) {
    return function (input) {
      return $filter('number')(input * 100) + ' %';
    };
```

```
});
```

Créer un filtre avec des paramètres

Par défaut, un filtre a un seul paramètre: la variable sur laquelle il est appliqué. Mais vous pouvez passer plus de paramètres à la fonction:

```
angular
  .module('app', [])
  .controller('MyController', function($scope) {
    $scope.example = 0.098152;
  })
  .filter('percentage', function($filter) {
    return function (input, decimals) {
      return $filter('number')(input * 100, decimals) + ' %';
    };
  });
```

Maintenant, vous pouvez donner une précision au filtre de `percentage` :

```
<span ng-controller="MyController">{{ example | percentage: 2 }}</span>
=> "9.81 %"
```

... mais les autres paramètres sont facultatifs, vous pouvez toujours utiliser le filtre par défaut:

```
<span ng-controller="MyController">{{ example | percentage }}</span>
=> "9.8152 %"
```

Lire Filtres personnalisés en ligne: https://riptutorial.com/fr/angularjs/topic/2552/filtres-personnalises

Chapitre 22: Filtres personnalisés avec ES6

Examples

Filtre FileSize utilisant ES6

Nous avons ici un filtre Taille de fichier pour décrire comment ajouter un filtre costum à un module existant:

```
let fileSize=function (size,unit,fixedDigit) {
return size.toFixed(fixedDigit) + ' '+unit;
};

  let fileSizeFilter=function () {
        return function (size) {
            if (isNaN(size))
                size = 0;

            if (size < 1024)
                return size + ' octets';

            size /= 1024;

            if (size < 1024)
               return fileSize(size,'Ko',2);

            size /= 1024;

            if (size < 1024)
               return  fileSize(size,'Mo',2);

            size /= 1024;

            if (size < 1024)
               return  fileSize(size,'Go',2);

            size /= 1024;
            return  fileSize(size,'To',2);
        };
    };
export default fileSizeFilter;
```

L'appel du filtre dans le module:

```
import fileSizeFilter from 'path...';
let myMainModule =
    angular.module('mainApp', [])
   .filter('fileSize', fileSizeFilter);
```

Le code HTML où nous appelons le filtre:

```
<div ng-app="mainApp">
```

```
    <div>
        <input type="text" ng-model="size" />
    </div>
    <div>
        <h3>Output:</h3>
        <p>{{size| Filesize}}</p>
    </div>
</div>
```

Lire Filtres personnalisés avec ES6 en ligne: https://riptutorial.com/fr/angularjs/topic/9421/filtres-personnalises-avec-es6

Chapitre 23: Fonctions d'assistance intégrées

Examples

angular.equals

La fonction `angular.equals` compare et détermine si 2 objets ou valeurs sont égaux, `angular.equals` effectue une comparaison profonde et renvoie true si et seulement si au moins 1 des conditions suivantes est remplie.

> angular.equals (value1, value2)

1. Si les objets ou les valeurs passent la comparaison `===`
2. Si les deux objets ou les valeurs sont du même type et que toutes leurs propriétés sont également égales en utilisant `angular.equals`
3. Les deux valeurs sont égales à `NaN`
4. Les deux valeurs représentent le même résultat de l'expression régulière.

Cette fonction est utile lorsque vous avez besoin de comparer des objets ou des tableaux en fonction de leurs valeurs ou de leurs résultats plutôt que de simples références.

Exemples

```
angular.equals(1, 1) // true
angular.equals(1, 2) // false
angular.equals({}, {}) // true, note that {}==={} is false
angular.equals({a: 1}, {a: 1}) // true
angular.equals({a: 1}, {a: 2}) // false
angular.equals(NaN, NaN) // true
```

angular.isString

La fonction `angular.isString` renvoie true si l'objet ou la valeur qui lui est donné est du type `string`

> angular.isString (value1)

Exemples

```
angular.isString("hello") // true
angular.isString([1, 2]) // false
angular.isString(42) // false
```

C'est l'équivalent de la performance

```
typeof someValue === "string"
```

angular.isArray

La fonction `angular.isArray` renvoie true si et seulement si l'objet ou la valeur qui lui est transmis est du type `Array` .

> angular.isArray (valeur)

Exemples

```
angular.isArray([]) // true
angular.isArray([2, 3]) // true
angular.isArray({}) // false
angular.isArray(17) // false
```

C'est l'équivalent de

```
Array.isArray(someValue)
```

angular.merge

La fonction angular.merge prend toutes les propriétés énumérables de l'objet source pour étendre profondément l'objet de destination.

La fonction renvoie une référence à l'objet de destination maintenant étendu

> angular.merge (destination, source)

Exemples

```
angular.merge({}, {}) // {}
angular.merge({name: "king roland"}, {password: "12345"})
// {name: "king roland", password: "12345"}
angular.merge({a: 1}, [4, 5, 6]) // {0: 4, 1: 5, 2: 6, a: 1}
angular.merge({a: 1}, {b: {c: {d: 2}}}) // {"a":1,"b":{"c":{"d":2}}}
```

angular.isDefined et angular.isUndefined

La fonction `angular.isDefined` teste une valeur si elle est définie

> angular.isDefined (someValue)

C'est l'équivalent de la performance

```
value !== undefined; // will evaluate to true is value is defined
```

Exemples

```
angular.isDefined(42) // true
angular.isDefined([1, 2]) // true
angular.isDefined(undefined) // false
```

```
angular.isDefined(null) // true
```

La fonction `angular.isUndefined` teste si une valeur est indéfinie (c'est en fait l'opposé de `angular.isDefined`)

> angular.isUndefined (someValue)

C'est l'équivalent de la performance

```
value === undefined; // will evaluate to true is value is undefined
```

Ou juste

```
!angular.isDefined(value)
```

Exemples

```
angular.isUndefined(42) // false
angular.isUndefined(undefined) // true
```

angular.isDate

La fonction `angular.isDate` renvoie true si et seulement si l'objet qui lui est transmis est du type Date.

> angular.isDate (valeur)

Exemples

```
angular.isDate("lone star") // false
angular.isDate(new Date()) // true
```

angular.isNumber

La fonction `angular.isNumber` renvoie true si et seulement si l'objet ou la valeur qui lui est transmis est du type Number, cela inclut + Infinity, -Infinity et NaN

> angular.isNumber (valeur)

Cette fonction ne provoquera pas une contrainte de type telle que

```
"23" == 23 // true
```

Exemples

```
angular.isNumber("23") // false
angular.isNumber(23) // true
angular.isNumber(NaN) // true
```

```
angular.isNumber(Infinity) // true
```

Cette fonction ne provoquera pas une contrainte de type telle que

```
"23" == 23 // true
```

angular.isFunction

La fonction `angular.isFunction` détermine et renvoie true si et seulement si la valeur passée à est une référence à une fonction.

La fonction renvoie une référence à l'objet de destination maintenant étendu

> angular.isFunction (fn)

Exemples

```
var onClick = function(e) {return e};
angular.isFunction(onClick); // true

var someArray = ["pizza", "the", "hut"];
angular.isFunction(someArray ); // false
```

angular.toJson

La fonction `angular.toJson` prendra un objet et le sérialisera dans une chaîne au format JSON.

A la différence de la fonction native `JSON.stringify` , cette fonction supprime toutes les propriétés commençant par `$$` (car angulaire en général préfixe les propriétés internes avec `$$`)

```
angular.toJson(object)
```

Comme les données doivent être sérialisées avant de passer par un réseau, cette fonction est utile pour transformer les données que vous souhaitez transmettre en JSON.

Cette fonction est également utile pour le débogage car elle fonctionne de manière similaire à une méthode `.toString` .

Exemples:

```
angular.toJson({name: "barf", occupation: "mog", $$somebizzareproperty: 42})
// "{"name":"barf","occupation":"mog"}"
angular.toJson(42)
// "42"
angular.toJson([1, "2", 3, "4"])
// "[1,"2",3,"4"]"
var fn = function(value) {return value}
angular.toJson(fn)
// undefined, functions have no representation in JSON
```

angular.fromJson

La fonction `angular.fromJson` va désérialiser une chaîne JSON valide et retourner un objet ou un tableau.

> angular.fromJson (chaîne | objet)

Notez que cette fonction n'est pas limitée aux chaînes uniquement, elle affichera une représentation de tout objet transmis.

Exemples:

```
angular.fromJson("{\"yogurt\": \"strawberries\"}")
// Object {yogurt: "strawberries"}
angular.fromJson('{jam: "raspberries"}')
// will throw an exception as the string is not a valid JSON
angular.fromJson(this)
// Window {external: Object, chrome: Object, _gaq: Y, angular: Object, ng339: 3…}
angular.fromJson([1, 2])
// [1, 2]
typeof angular.fromJson(new Date())
// "object"
```

angular.noop

`angular.noop` est une fonction qui `angular.noop` aucune opération, vous passez `angular.noop` lorsque vous devez fournir un argument de fonction qui ne fera rien.

> angular.noop ()

`angular.noop` peut servir à fournir un rappel vide à une fonction qui `angular.noop` une erreur lorsque quelque chose d'autre qu'une fonction lui est transmis.

Exemple:

```
$scope.onSomeChange = function(model, callback) {
    updateTheModel(model);
    if (angular.isFunction(callback)) {
        callback();
    } else {
        throw new Error("error: callback is not a function!");
    }
};

$scope.onSomeChange(42, function() {console.log("hello callback")});
// will update the model and print 'hello callback'
$scope.onSomeChange(42, angular.noop);
// will update the model
```

Exemples supplémentaires:

```
angular.noop() // undefined
angular.isFunction(angular.noop) // true
```

angular.isObject

`angular.isObject` renvoie true si et seulement si l'argument qui lui est transmis est un objet, cette fonction renverra également true pour un tableau et renverra false pour `null` même si `typeof null` est `object` .

> angular.isObject (valeur)

Cette fonction est utile pour la vérification de type lorsque vous avez besoin d'un objet défini à traiter.

Exemples:

```
angular.isObject({name: "skroob", job: "president"})
// true
angular.isObject(null)
// false
angular.isObject([null])
// true
angular.isObject(new Date())
// true
angular.isObject(undefined)
// false
```

angular.isElement

`angular.isElement` renvoie true si l'argument qui lui est transmis est un élément DOM ou un élément `angular.isElement` jQuery.

> angular.isElement (elem)

Cette fonction est utile pour taper check si un argument passé est un élément avant d'être traité comme tel.

Exemples:

```
angular.isElement(document.querySelector("body"))
// true
angular.isElement(document.querySelector("#some_id"))
// false if "some_id" is not using as an id inside the selected DOM
angular.isElement("<div></div>")
// false
```

copie angulaire

La fonction `angular.copy` prend un objet, un tableau ou une valeur et en crée une copie `angular.copy` .

> angular.copy ()

Exemple:

Objets:

```
let obj = {name: "vespa", occupation: "princess"};
let cpy = angular.copy(obj);
cpy.name = "yogurt"
// obj = {name: "vespa", occupation: "princess"}
// cpy = {name: "yogurt", occupation: "princess"}
```

Tableaux:

```
var w = [a, [b, [c, [d]]]];
var q = angular.copy(w);
// q = [a, [b, [c, [d]]]]
```

Dans l'exemple ci-dessus, `angular.equals(w, q)` sera évalué à true car `.equals` teste l'égalité par valeur. Cependant, `w === q` évaluera à false car la comparaison stricte entre les objets et les tableaux se fait par référence.

angular.identity

La fonction `angular.identity` renvoie le premier argument qui lui est passé.

> angular.identity (argument)

Cette fonction est utile pour la programmation fonctionnelle, vous pouvez fournir cette fonction par défaut au cas où une fonction attendue ne serait pas passée.

Exemples:

```
angular.identity(42) // 42
```

```
var mutate = function(fn, num) {
    return angular.isFunction(fn) ? fn(num) : angular.identity(num)
}

mutate(function(value) {return value-7}, 42) // 35
mutate(null, 42) // 42
mutate("mount. rushmore", 42) // 42
```

angulaire.pour chaque

`angular.forEach` accepte un objet et une fonction itérateur. Il exécute ensuite la fonction itérateur sur chaque propriété / valeur énumérable de l'objet. Cette fonction fonctionne également sur les tableaux.

Comme la version JS de `Array.prototype.forEach` La fonction ne `Array.prototype.forEach` pas les propriétés héritées (propriétés de prototype), mais la fonction ne tentera pas de traiter une valeur `null` ou `undefined` et la renverra simplement.

> angular.forEach (objet, fonction (valeur, clé) {// fonction});

Exemples:

```
angular.forEach({"a": 12, "b": 34}, (value, key) => console.log("key: " + key + ", value: " +
value))
// key: a, value: 12
// key: b, value: 34
angular.forEach([2, 4, 6, 8, 10], (value, key) => console.log(key))
// will print the array indices: 1, 2, 3, 4, 5
angular.forEach([2, 4, 6, 8, 10], (value, key) => console.log(value))
// will print the array values: 2, 4, 6, 7, 10
angular.forEach(undefined, (value, key) => console.log("key: " + key + ", value: " + value))
// undefined
```

Lire Fonctions d'assistance intégrées en ligne:
https://riptutorial.com/fr/angularjs/topic/3032/fonctions-d-assistance-integrees

Chapitre 24: Fournisseurs

Syntaxe

- constante (nom, valeur);
- valeur (nom, valeur);
- factory (nom, $ getFn);
- service (nom, constructeur);
- fournisseur (nom, fournisseur);

Remarques

Les fournisseurs sont des objets singleton pouvant être injectés, par exemple, dans d'autres services, contrôleurs et directives. Tous les fournisseurs sont enregistrés en utilisant différentes "recettes", où le `Provider` est le plus flexible. Toutes les recettes possibles sont:

- Constant
- Valeur
- Usine
- Un service
- Fournisseur

Les services, les usines et les fournisseurs sont tous initialisés paresseux, le composant est initialisé uniquement si l'application en dépend.

Les décorateurs sont étroitement liés aux fournisseurs. Les décorateurs sont utilisés pour intercepter le service ou la création d'usine afin de modifier son comportement ou de le remplacer (en partie).

Examples

Constant

`Constant` est disponible à la fois dans les phases de configuration et d'exécution.

```
angular.module('app',[])
  .constant('endpoint', 'http://some.rest.endpoint') // define
  .config(function(endpoint) {
    // do something with endpoint
    // available in both config- and run phases
  })
  .controller('MainCtrl', function(endpoint) {       // inject
    var vm = this;
    vm.endpoint = endpoint;                          // usage
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{ ::vm.endpoint }}</div>
</body>
```

> point de terminaison = http: //some.rest.endpoint

Valeur

`Value` est disponible dans les phases de configuration et d'exécution.

```
angular.module('app',[])
  .value('endpoint', 'http://some.rest.endpoint') // define
  .run(function(endpoint) {
    // do something with endpoint
    // only available in run phase
  })
  .controller('MainCtrl', function(endpoint) {    // inject
    var vm = this;
    vm.endpoint = endpoint;                        // usage
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{ ::vm.endpoint }}</div>
</body>
```

> point de terminaison = http: //some.rest.endpoint

Usine

`Factory` est disponible en phase d'exécution.

> La recette Factory construit un nouveau service en utilisant une fonction avec zéro ou plusieurs arguments (ce sont des dépendances sur d'autres services). La valeur de retour de cette fonction est l'instance de service créée par cette recette.
>
> Factory peut créer un service de n'importe quel type, qu'il s'agisse d'une primitive, d'un littéral d'objet, d'une fonction ou même d'une instance d'un type personnalisé.

```
angular.module('app',[])
  .factory('endpointFactory', function() {
    return {
      get: function() {
        return 'http://some.rest.endpoint';
      }
    };
  })
  .controller('MainCtrl', function(endpointFactory) {
    var vm = this;
    vm.endpoint = endpointFactory.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

> point de terminaison = http: //some.rest.endpoint

Un service

`Service` est disponible en phase d'exécution.

> La recette Service produit un service tout comme les recettes Value ou Factory, mais en *appelant un constructeur avec le nouvel opérateur* . Le constructeur peut prendre zéro ou plusieurs arguments, qui représentent des dépendances nécessaires à l'instance de ce type.

```
angular.module('app',[])
  .service('endpointService', function() {
    this.get = function() {
      return 'http://some.rest.endpoint';
    };
  })
  .controller('MainCtrl', function(endpointService) {
    var vm = this;
    vm.endpoint = endpointService.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

> point de terminaison = http: //some.rest.endpoint

Fournisseur

`Provider` est disponible dans les phases de configuration et d'exécution.

> La recette du fournisseur est définie syntaxiquement comme un type personnalisé qui implémente une méthode `$get` .
>
> Vous devez utiliser la recette du fournisseur uniquement lorsque vous souhaitez exposer une API pour la configuration à l'échelle de l'application qui doit être effectuée avant le démarrage de l'application. Cela n'est généralement intéressant que pour les services réutilisables dont le comportement peut varier légèrement d'une application à l'autre.

```
angular.module('app',[])
  .provider('endpointProvider', function() {
    var uri = 'n/a';
```

```
    this.set = function(value) {
      uri = value;
    };

    this.$get = function() {
      return {
        get: function() {
          return uri;
        }
      };
    };
  })
  .config(function(endpointProviderProvider) {
    endpointProviderProvider.set('http://some.rest.endpoint');
  })
  .controller('MainCtrl', function(endpointProvider) {
    var vm = this;
    vm.endpoint = endpointProvider.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

> point de terminaison = http: //some.rest.endpoint

Sans phase de `config` , le résultat serait

> point final = n / a

Lire Fournisseurs en ligne: https://riptutorial.com/fr/angularjs/topic/5169/fournisseurs

Chapitre 25: Impression

Remarques

Créez une classe ng-hide dans un fichier css. ng-show / hide ne fonctionnera pas sans la classe.

Plus de détails

Examples

Service d'impression

Un service:

```
angular.module('core').factory('print_service', ['$rootScope', '$compile', '$http',
'$timeout','$q',
    function($rootScope, $compile, $http, $timeout,$q) {

        var printHtml = function (html) {
            var deferred = $q.defer();
            var hiddenFrame = $('<iframe style="display:
none"></iframe>').appendTo('body')[0];

            hiddenFrame.contentWindow.printAndRemove = function() {
                hiddenFrame.contentWindow.print();
                $(hiddenFrame).remove();
                deferred.resolve();
            };

            var htmlContent =   "<!doctype html>"+
                                "<html>"+
                                    '<head><link rel="stylesheet" type="text/css"
href="/style/css/print.css"/></head>'+
                                    '<body onload="printAndRemove();">' +
                                        html +
                                    '</body>'+
                                "</html>";

            var doc = hiddenFrame.contentWindow.document.open("text/html", "replace");
            doc.write(htmlContent);
            doc.close();
            return deferred.promise;
        };

        var openNewWindow = function (html) {
            var newWindow = window.open("debugPrint.html");
            newWindow.addEventListener('load', function(){
                $(newWindow.document.body).html(html);
            }, false);
        };

        var print = function (templateUrl, data) {

            $rootScope.isBeingPrinted = true;
```

```
            $http.get(templateUrl).success(function(template){
                var printScope = $rootScope.$new()
                angular.extend(printScope, data);
                var element = $compile($('<div>' + template + '</div>'))(printScope);
                var waitForRenderAndPrint = function() {
                    if(printScope.$$phase || $http.pendingRequests.length) {
                        $timeout(waitForRenderAndPrint, 1000);
                    } else {
                        // Replace printHtml with openNewWindow for debugging
                        printHtml(element.html());
                        printScope.$destroy();
                    }
                };
                waitForRenderAndPrint();
            });
        };

        var printFromScope = function (templateUrl, scope, afterPrint) {
            $rootScope.isBeingPrinted = true;
            $http.get(templateUrl).then(function(response){
                var template = response.data;
                var printScope = scope;
                var element = $compile($('<div>' + template + '</div>'))(printScope);
                var waitForRenderAndPrint = function() {
                    if (printScope.$$phase || $http.pendingRequests.length) {
                        $timeout(waitForRenderAndPrint);
                    } else {
                        // Replace printHtml with openNewWindow for debugging
                        printHtml(element.html()).then(function() {
                            $rootScope.isBeingPrinted = false;
                            if (afterPrint) {
                                afterPrint();
                            }
                        });
                    }
                };
                waitForRenderAndPrint();
            });
        };

        return {
            print : print,
            printFromScope : printFromScope
        }
    }
]);
```

Manette :

```
var template_url = '/views/print.client.view.html';
print_service.printFromScope(template_url,$scope,function(){
      // Print Completed
});
```

Lire Impression en ligne: https://riptutorial.com/fr/angularjs/topic/6750/impression

Chapitre 26: Injection de dépendance

Syntaxe

- myApp.controller ('MyController', function ($ scope) {...}); // code non minifié
- myApp.controller ('MyController', ['$ scope', fonction ($ scope) {...}]); // soutien minification
- function MyController () {}

  MyController. $ Inject = ['$ scope'];

  myApp.controller ('MyController', MyController); // $ injecter une annotation
- $ injector.get ("injectable"); // injection dynamique / à l'exécution

Remarques

Les fournisseurs ne peuvent pas être injectés dans des blocs d' `run` .

Les services ou les valeurs ne peuvent pas être injectés dans les blocs de `config` .

Veillez à annoter vos injections afin que votre code ne se brise pas lors de la minification.

Examples

Les injections

L'exemple le plus simple d'une injection dans une application angulaire - injecter `$scope` à un `Controller` angulaire:

```
angular.module('myModule', [])
.controller('myController', ['$scope', function($scope) {
    $scope.members = ['Alice', 'Bob'];
    ...
}])
```

Ce qui précède illustre une injection d'un `$scope` dans un `controller` , mais il en va de même si vous injectez un module dans un autre. Le processus est le même.

Le système Angular est chargé de résoudre les dépendances pour vous. Si vous créez un service par exemple, vous pouvez le répertorier comme dans l'exemple ci-dessus et il sera disponible pour vous.

Vous pouvez utiliser DI - Dependency Injection - où que vous définissiez un composant.

Notez que dans l'exemple ci-dessus, nous utilisons ce qu'on appelle "l'annotation de tableau en

ligne". Signification, nous écrivons explicitement comme des chaînes les noms de nos dépendances. Nous le faisons pour empêcher l'application de se briser lorsque le code est minifié pour Production. La minification du code modifie le nom des variables (mais pas les chaînes), ce qui interrompt l'injection. En utilisant des chaînes, Angular sait quelles dépendances nous voulons.

Très important - l'ordre des noms de chaînes doit être identique à celui des paramètres de la fonction.

Il existe des outils qui automatisent ce processus et en prennent soin pour vous.

Injections dynamiques

Il existe également une option permettant de demander dynamiquement des composants. Vous pouvez le faire en utilisant le service `$injector` :

```
myModule.controller('myController', ['$injector', function($injector) {
    var myService = $injector.get('myService');
}]);
```

Remarque: bien que cette méthode puisse être utilisée pour empêcher le problème de dépendance circulaire susceptible de briser votre application, il n'est pas recommandé de l'ignorer en l'utilisant. La dépendance circulaire indique généralement qu’il ya une faille dans l’architecture de votre application.

$ inject Propriété Annotation

De manière équivalente, nous pouvons utiliser l’annotation de la propriété `$inject` pour obtenir la même chose que ci-dessus:

```
var MyController = function($scope) {
  // ...
}
MyController.$inject = ['$scope'];
myModule.controller('MyController', MyController);
```

Charger dynamiquement le service AngularJS en JavaScript vanilla

Vous pouvez charger les services AngularJS dans JavaScript JavaScript en utilisant la méthode AngularJS `injector()` . Chaque élément jqLite récupéré appelant `angular.element()` possède une méthode d' `injector()` qui peut être utilisée pour récupérer l'injecteur.

```
var service;
var serviceName = 'myService';

var ngAppElement = angular.element(document.querySelector('[ng-app],[data-ng-app]') ||
document);
var injector = ngAppElement.injector();

if(injector && injector.has(serviceNameToInject)) {
```

```
    service = injector.get(serviceNameToInject);
}
```

Dans l'exemple ci-dessus, nous essayons de récupérer l'élément jqLite contenant la racine de l'application AngularJS (`ngAppElement`). Pour ce faire, nous utilisons la méthode `angular.element()` , en recherchant un élément DOM contenant l'attribut `ng-app` ou `data-ng-app` ou, s'il n'existe pas, nous nous tournons vers l'élément `document` . Nous utilisons `ngAppElement` pour récupérer l'instance d'injecteur (avec `ngAppElement.injector()`). L'instance d'injecteur est utilisée pour vérifier si le service à injecter existe (avec `injector.has()`) puis pour charger le service (avec `injector.get()`) dans la variable de `service` .

Lire Injection de dépendance en ligne: https://riptutorial.com/fr/angularjs/topic/1582/injection-de-dependance

Chapitre 27: Intercepteur HTTP

Introduction

Le service $ http d'AngularJS nous permet de communiquer avec un backend et de faire des requêtes HTTP. Il y a des cas où nous voulons capturer chaque demande et la manipuler avant de l'envoyer au serveur. D'autres fois, nous aimerions capturer la réponse et la traiter avant de terminer l'appel. Le traitement global des erreurs http peut aussi être un bon exemple de ce besoin. Les intercepteurs sont créés exactement pour de tels cas.

Examples

Commencer

Le service intégré `$http` Angular nous permet d'envoyer des requêtes HTTP. Souvent, il est nécessaire de faire des choses avant ou après une requête, par exemple en ajoutant à chaque requête un jeton d'authentification ou en créant une logique de traitement d'erreur générique.

HttpInterceptor générique pas à pas

Créez un fichier HTML avec le contenu suivant:

```
<!DOCTYPE html>
<html>
<head>
  <title>Angular Interceptor Sample</title>
  <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
  <script src="app.js"></script>
  <script src="appController.js"></script>
  <script src="genericInterceptor.js"></script>
</head>
<body ng-app="interceptorApp">
    <div ng-controller="appController as vm">
        <button ng-click="vm.sendRequest()">Send a request</button>
    </div>
</body>
</html>
```

Ajoutez un fichier JavaScript appelé "app.js":

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

interceptorApp.config(function($httpProvider) {
        $httpProvider.interceptors.push('genericInterceptor');
});
```

Ajoutez-en un autre appelé 'appController.js':

```
(function() {
```

```
    'use strict';

    function appController($http) {
        var vm = this;

        vm.sendRequest = function(){
            $http.get('http://google.com').then(function(response){
                console.log(response);
            });
        };
    }

    angular.module('interceptorApp').controller('appController',['$http', appController]);
})();
```

Et enfin le fichier contenant l'intercepteur lui-même 'genericInterceptor.js':

```
(function() {
    "use strict";

    function genericInterceptor($q) {
        this.responseError = function (response) {
            return $q.reject(response);
        };

        this.requestError = function(request){
            if (canRecover(rejection)) {
                return responseOrNewPromise
            }
            return $q.reject(rejection);
        };

        this.response = function(response){
            return response;
        };

        this.request = function(config){
            return config;
        }
    }

    angular.module('interceptorApp').service('genericInterceptor', genericInterceptor);
})();
```

Le 'genericInterceptor' couvre les fonctions possibles que nous pouvons remplacer en ajoutant un comportement supplémentaire à notre application.

Message Flash sur la réponse à l'aide de l'intercepteur http

Dans le fichier de vue

Dans la base html (index.html) où nous incluons généralement les scripts angulaires ou le code HTML partagé dans l'application, laissez un élément div vide, les messages flash apparaîtront à l'intérieur de cet élément div

```
<div class="flashmessage" ng-if="isVisible">
    {{flashMessage}}
</div>
```

Fichier de script

Dans la méthode de configuration du module angulaire, injectez le httpProvider, le httpProvider possède une propriété de tableau d'intercepteur, poussez l'intercepteur personnalisé. Dans l'exemple actuel, l'intercepteur personnalisé intercepte uniquement la réponse et appelle une méthode attachée à rootScope.

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

    interceptorTest.config(['$httpProvider',function ($httpProvider) {

        $httpProvider.interceptors.push(["$rootScope",function ($rootScope) {
            return {     //intercept only the response
                        'response': function (response)
                                    {

$rootScope.showFeedBack(response.status,response.data.message);

                                      return response;
                                    }
                      };
        }]);

    }])
```

Comme seuls les fournisseurs peuvent être injectés dans la méthode de configuration d'un module angulaire (c'est-à-dire httpProvider et non rootscope), déclarez la méthode associée à rootscope dans la méthode run du module angulaire.

Affichez également le message dans $ timeout afin que le message ait la propriété flash, qui disparaît après une heure limite. Dans notre exemple, ses 3000 ms.

```
interceptorTest.run(["$rootScope","$timeout",function($rootScope,$timeout){
     $rootScope.showFeedBack = function(status,message){

        $rootScope.isVisible = true;
        $rootScope.flashMessage = message;
        $timeout(function(){$rootScope.isVisible = false },3000)
    }
}]);
```

Pièges communs

En essayant d’injecter **$ rootScope ou tout autre service** dans la méthode de **configuration** du module angulaire, le cycle de vie de l’application angulaire ne le permet pas et une erreur de fournisseur inconnu sera générée. Seuls les **fournisseurs** peuvent être injectés dans la méthode de **configuration** du module angulaire

Lire Intercepteur HTTP en ligne: https://riptutorial.com/fr/angularjs/topic/6484/intercepteur-http

Chapitre 28: Le débogage

Examples

Débogage de base dans le balisage

Test de la portée et sortie du modèle

```
<div ng-app="demoApp" ng-controller="mainController as ctrl">
    {{$id}}
    <ul>
        <li ng-repeat="item in ctrl.items">
            {{$id}}<br/>
            {{item.text}}
        </li>
    </ul>
    {{$id}}
    <pre>
       {{ctrl.items | json : 2}}
    </pre>
</div>


angular.module('demoApp', [])
.controller('mainController', MainController);

function MainController() {
    var vm = this;
    vm.items = [{
        id: 0,
        text: 'first'
    },
    {
        id: 1,
        text: 'second'
    },
    {
        id: 2,
        text: 'third'
    }];
}
```

Parfois, il peut être utile de voir s'il existe une nouvelle portée pour résoudre les problèmes de portée. `$scope.$id` peut être utilisé dans une expression partout dans votre balisage pour voir s'il y a un nouveau $ scope.

Dans l'exemple, vous pouvez voir que l'extérieur de ul-tag a la même portée ($ id = 2) et à l'intérieur de `ng-repeat` il y a de nouvelles étendues enfants pour chaque itération.

Une sortie du modèle dans une pré-balise est utile pour voir les données actuelles de votre modèle. Le filtre `json` crée une belle sortie formatée. La pré-balise est utilisée car à l'intérieur de cette balise, tout caractère de nouvelle ligne `\n` sera correctement affiché.

démo

Utilisation de l'extension chrome ng-inspect

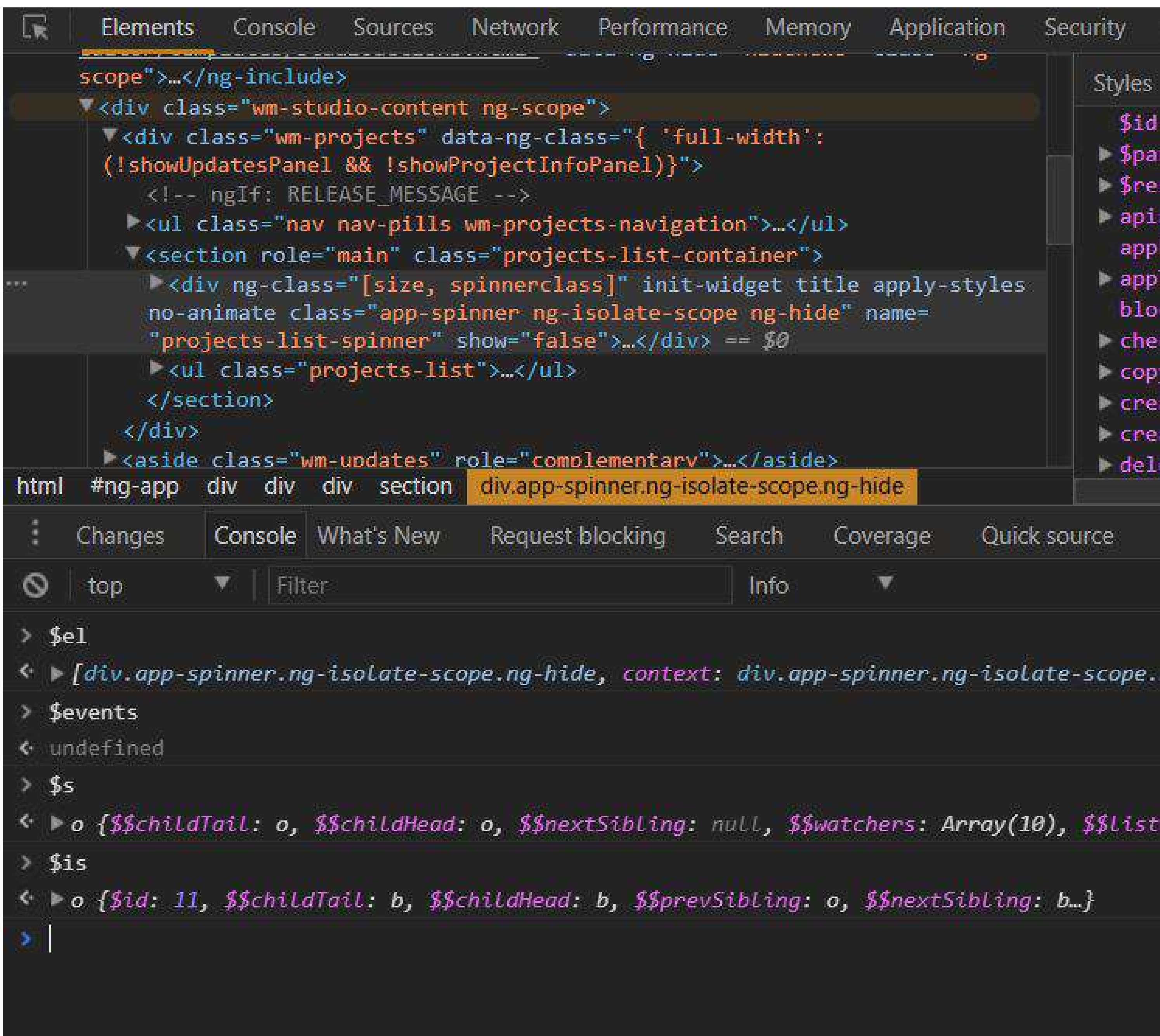
ng-inspect est une extension Chrome légère pour le débogage des applications AngularJS.

Lorsqu'un nœud est sélectionné dans le panneau des éléments, les informations relatives à la portée sont affichées dans le panneau ng-inspect.

Expose peu de variables globales pour un accès rapide à `scope/isolateScope` .

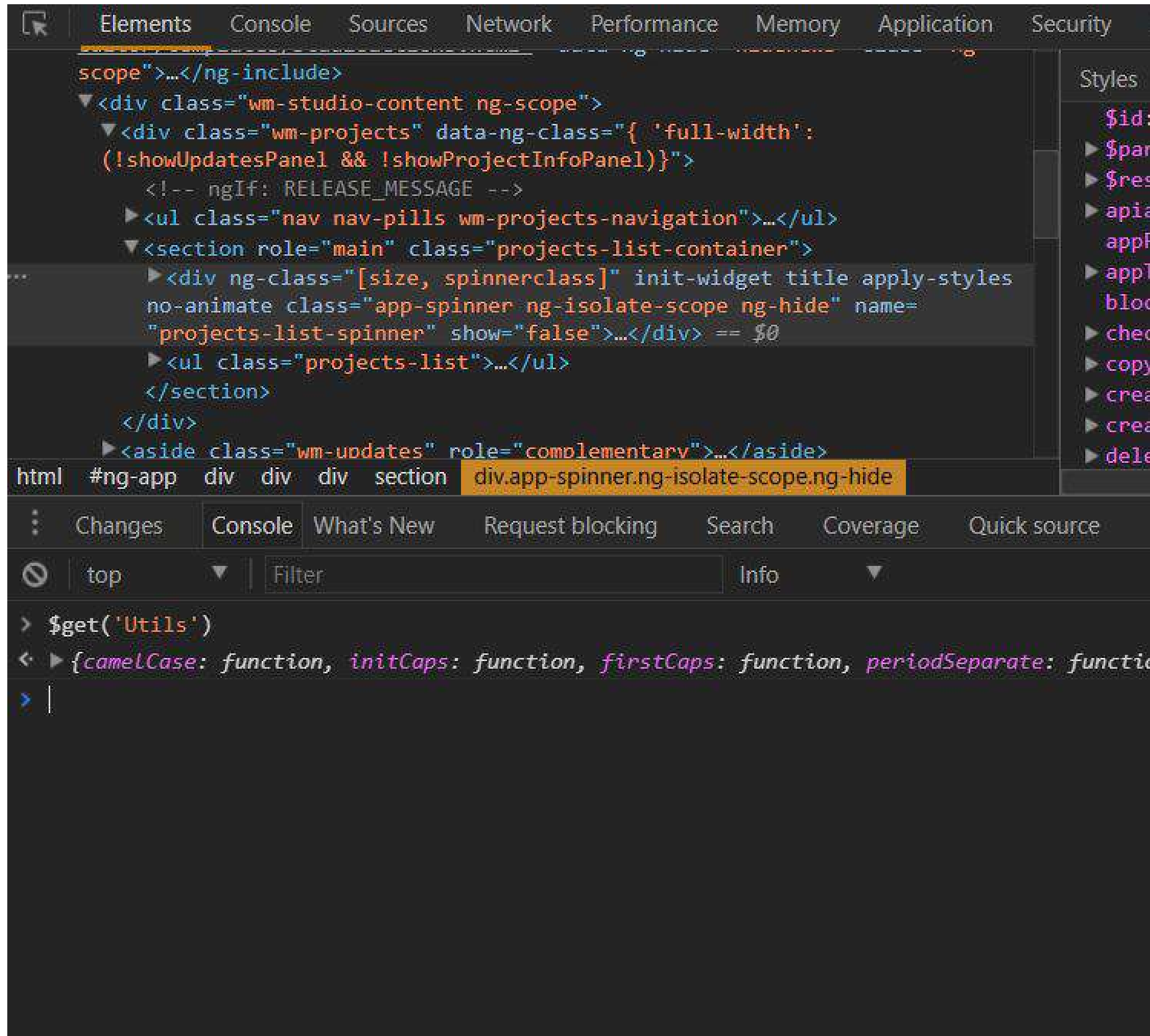
```
$s      -- scope of the selected node
$is     -- isolateScope of the selected node
$el     -- jQuery element reference of the selected node (requiers jQuery)
```

```
$events -- events present on the selected node (requires jQuery)
```



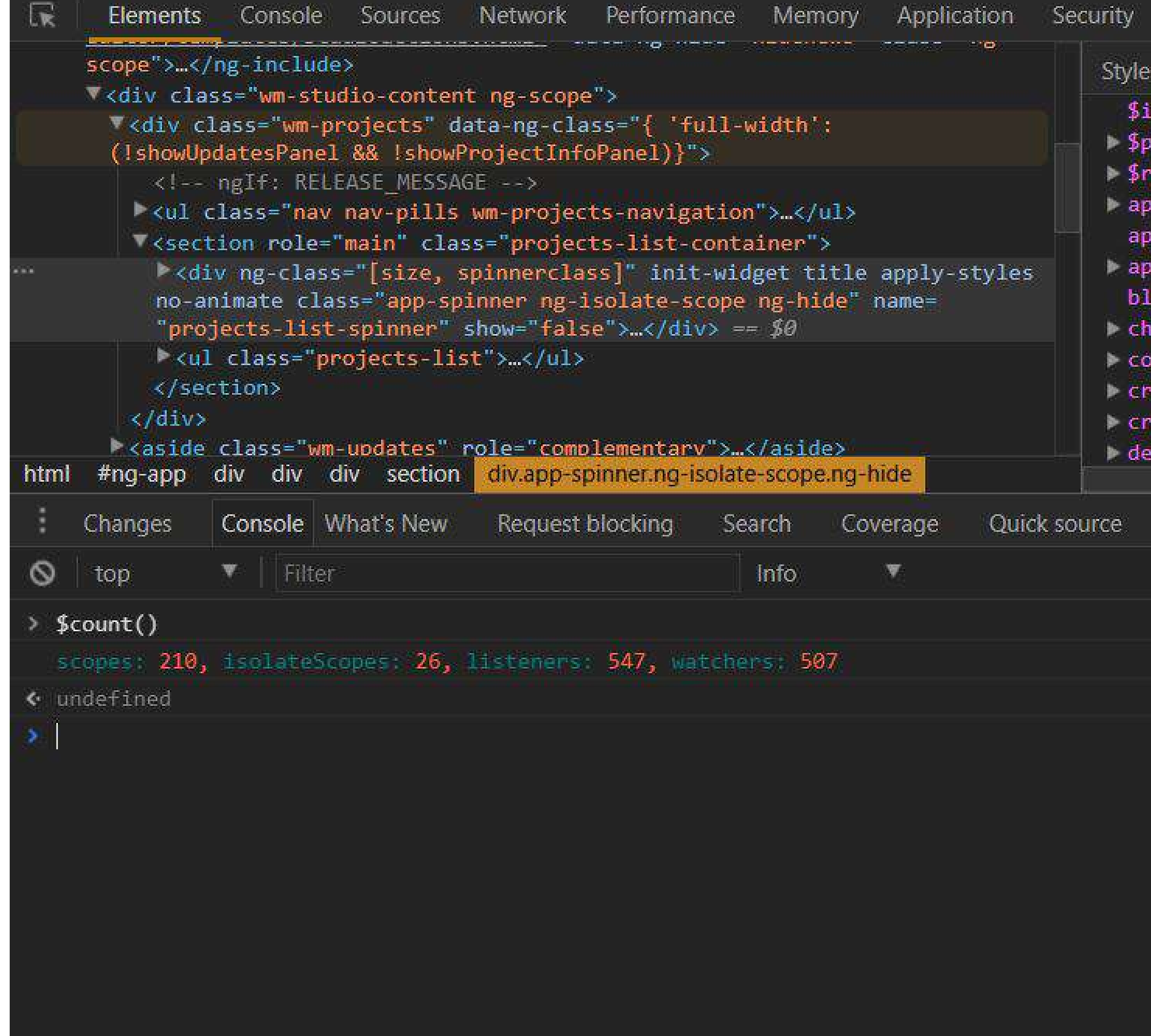
Permet d'accéder facilement aux services / usines.

Utilisez `$get()` pour récupérer l'instance d'un service / d'une usine par son nom.



Les performances de l'application peuvent être surveillées en comptant le nombre de portées, d'isolateScopes, d'observateurs et d'auditeurs de l'application.

Utilisez `$count()` pour obtenir le nombre de portées, isolateScopes, les observateurs et les écouteurs.



> Remarque: cette extension ne fonctionnera que lorsque le paramètre debugInfo est activé.

Téléchargez ng-inspect ici

Obtenir la portée de l'élément

Dans une application angulaire, tout tourne autour de la portée, si l'on peut obtenir une étendue d'éléments, il est facile de déboguer l'application angulaire. Comment accéder à la portée de l'élément:

```
angular.element(myDomElement).scope();
e.g.
angular.element(document.getElementById('yourElementId')).scope()  //accessing by ID
```

Obtenir la portée du contrôleur: -

```
angular.element('[ng-controller=ctrl]').scope()
```

Un autre moyen facile d'accéder à un élément DOM à partir de la console (comme mentionné par jm) consiste à cliquer dessus dans l'onglet "elements", et il sera automatiquement stocké sous la forme $ 0.

```
angular.element($0).scope();
```

Lire Le débogage en ligne: https://riptutorial.com/fr/angularjs/topic/4761/le-debogage

Chapitre 29: Le moi ou cette variable dans un contrôleur

Introduction

Ceci est une explication d'un modèle commun et généralement considéré comme la meilleure pratique que vous pouvez voir dans le code AngularJS.

Examples

Comprendre le but de la variable auto

Lorsque vous utilisez "controller as syntax", vous devez donner à votre contrôleur un alias dans le code HTML lors de l'utilisation de la directive ng-controller.

```
<div ng-controller="MainCtrl as main">
</div>
```

Vous pouvez ensuite accéder aux propriétés et méthodes de la variable ***principale*** qui représente notre instance de contrôleur. Par exemple, accédons à la propriété de ***salutation*** de notre contrôleur et affichons-la à l'écran:

```
<div ng-controller="MainCtrl as main">
    {{ main.greeting }}
</div>
```

Maintenant, dans notre contrôleur, nous devons définir une valeur pour la propriété de salutation de notre instance de contrôleur (par opposition à $ scope ou autre chose):

```
angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;

  self.greeting = "Hello World";
})
```

Pour que l'affichage HTML soit correct, nous avons dû définir la propriété de salutation à ***l'*** intérieur de notre corps de contrôleur. Je crée une variable intermédiaire nommée ***self*** qui contient une référence à cela. Pourquoi? Considérez ce code:

```
angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;
```

```
  self.greeting = "Hello World";

  function itsLate () {
    this.greeting = "Goodnight";
  }

})
```

Dans ce code ci-dessus, vous pouvez vous attendre à ce que le texte à l'écran se mette à jour lorsque la méthode *itsLate* est appelée, mais ce n'est pas le cas. JavaScript utilise des règles de portée au niveau des fonctions pour que le "this" dans itsLate fasse référence à quelque chose de différent de "this" en dehors du corps de la méthode. Cependant, nous pouvons obtenir le résultat souhaité si nous utilisons la variable ***auto*** :

```
 angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;

  self.greeting = "Hello World";

  function itsLate () {
    self.greeting = "Goodnight";
  }

})
```

C'est la beauté d'utiliser une variable "self" dans vos contrôleurs - vous pouvez y accéder n'importe où dans votre contrôleur et vous pouvez toujours être sûr qu'elle fait référence à votre instance de contrôleur.

Lire Le moi ou cette variable dans un contrôleur en ligne:
https://riptutorial.com/fr/angularjs/topic/8867/le-moi-ou-cette-variable-dans-un-controleur

Chapitre 30: Les constantes

Remarques

UPPERCASE votre constante : Ecrire constante dans le capital est une meilleure pratique courante utilisée dans de nombreuses langues. Il est également utile d'identifier clairement la nature des éléments injectés:

Lorsque vous voyez `.controller('MyController', function($scope, Profile, EVENT))` , vous savez instantanément que:

- `$scope` est un élément angulaire
- `Profile` est un service personnalisé ou une usine
- `EVENT` est une constante angulaire

Examples

Créez votre première constante

```
angular
  .module('MyApp', [])
  .constant('VERSION', 1.0);
```

Votre constante est maintenant déclarée et peut être injectée dans un contrôleur, un service, une usine, un fournisseur et même dans une méthode de configuration:

```
angular
  .module('MyApp')
  .controller('FooterController', function(VERSION) {
    this.version = VERSION;
  });
```

```
<footer ng-controller="FooterController as Footer">{{ Footer.version }}</footer>
```

Cas d'utilisation

Il n'y a pas de révolution ici, mais la constante angulaire peut être utile spécialement lorsque votre application et / ou votre équipe commencent à se développer ... ou si vous aimez simplement écrire de beaux codes!

- **Code refactor.** Exemple avec les noms d'événement. Si vous utilisez beaucoup d'événements dans votre application, vous avez un peu partout les noms des événements. Quand un nouveau développeur rejoint votre équipe, il nomme ses événements avec une syntaxe différente, ... Vous pouvez facilement empêcher cela en regroupant les noms de vos événements dans une constante:

```
angular
  .module('MyApp')
  .constant('EVENTS', {
    LOGIN_VALIDATE_FORM: 'login::click-validate',
    LOGIN_FORGOT_PASSWORD: 'login::click-forgot',
    LOGIN_ERROR: 'login::notify-error',
    ...
  });
```

```
angular
  .module('MyApp')
  .controller('LoginController', function($scope, EVENT) {
    $scope.$on(EVENT.LOGIN_VALIDATE_FORM, function() {
      ...
    });
  })
```

... et maintenant, les noms de vos événements peuvent bénéficier de l'autocomplétion!

- **Définir la configuration** Localisez toutes vos configurations au même endroit:

```
angular
  .module('MyApp')
  .constant('CONFIG', {
    BASE_URL: {
      APP: 'http://localhost:3000',
      API: 'http://localhost:3001'
    },
    STORAGE: 'S3',
    ...
  });
```

- **Isoler les pièces.** Parfois, il y a des choses dont vous n'êtes pas très fier ... comme la valeur codée par exemple. Au lieu de les laisser dans votre code principal, vous pouvez créer une constante angulaire

```
angular
  .module('MyApp')
  .constant('HARDCODED', {
    KEY: 'KEY',
    RELATION: 'has_many',
    VAT: 19.6
  });
```

... et refactor quelque chose comme

```
$scope.settings = {
  username: Profile.username,
  relation: 'has_many',
  vat: 19.6
}
```

à

```
$scope.settings = {
  username: Profile.username,
  relation: HARDCODED.RELATION,
  vat: HARDCODED.VAT
}
```

Lire Les constantes en ligne: https://riptutorial.com/fr/angularjs/topic/3967/les-constantes

Chapitre 31: Migration vers Angular 2+

Introduction

AngularJS a été totalement réécrit en utilisant le langage TypeScript et renommé simplement Angular.

Il est possible de faire beaucoup pour une application AngularJS afin de faciliter le processus de migration. Comme le dit le guide de mise à niveau officiel , plusieurs "étapes de préparation" peuvent être effectuées pour restructurer votre application, la rendant meilleure et plus proche du nouveau style angulaire.

Examples

Conversion de votre application AngularJS en une structure orientée composants

Dans le nouveau framework angulaire, les **composants** sont les principaux **composants** constitutifs de l'interface utilisateur. Ainsi, l'une des premières étapes permettant à une application AngularJS d'être migrée vers le nouvel Angular est de la transformer en une structure plus orientée composants.

Des composants ont également été introduits dans l'ancienne AngularJS à partir de la version **1.5+** . L'utilisation de composants dans une application AngularJS ne rendra pas seulement sa structure plus proche du nouvel Angular 2+, mais la rendra également plus modulaire et plus facile à entretenir.

Avant d'aller plus loin, je recommande de consulter la page de documentation officielle d'AngularJS sur les composants , où leurs avantages et leur utilisation sont bien expliqués.

Je préférerais mentionner quelques astuces sur la façon de convertir l'ancien code orienté `ng-controller` vers le nouveau style orienté `component` .

Commencez à décomposer votre application en composants

Toutes les applications orientées composants ont généralement un ou plusieurs composants qui incluent d'autres sous-composants. Alors pourquoi ne pas créer le premier composant qui contiendra simplement votre application (ou un gros morceau de celle-ci).

Supposons que nous ayons un morceau de code assigné à un contrôleur, nommé `UserListController` , et que nous voulons en créer un composant, que nous `UserListComponent` .

HTML actuel:

```
<div ng-controller="UserListController as listctrl">
    <ul>
        <li ng-repeat="user in myUserList">
            {{ user }}
        </li>
    </ul>
</div>
```

JavaScript actuel:

```
app.controller("UserListController", function($scope, SomeService) {

    $scope.myUserList = ['Shin', 'Helias', 'Kalhac'];

    this.someFunction = function() {
        // ...
    }

    // ...
}
```

nouveau HTML:

```
<user-list></user-list>
```

nouveau JavaScript:

```
app.component("UserList", {
    templateUrl: 'user-list.html',
    controller: UserListController
});

function UserListController(SomeService) {

    this.myUserList = ['Shin', 'Helias', 'Kalhac'];

    this.someFunction = function() {
        // ...
    }

    // ...
}
```

Notez que nous n'injectons plus `$scope` dans la fonction contrôleur et que nous `this.myUserList` maintenant `this.myUserList` au lieu de `$scope.myUserList` ;

nouveau fichier de modèle `user-list.component.html` **:**

```
<ul>
    <li ng-repeat="user in $ctrl.myUserList">
        {{ user }}
    </li>
</ul>
```

Notez comment nous faisons maintenant référence à la variable `myUserList` , qui appartient au contrôleur, en utilisant `$ctrl.myUserList` partir du HTML au lieu de `$scope.myUserList` .

C'est parce que, comme vous l'avez probablement compris après avoir lu la documentation, `$ctrl` dans le modèle fait maintenant référence à la fonction du contrôleur.

Qu'en est-il des contrôleurs et des routes?

Si votre contrôleur était lié au modèle en utilisant le système de routage au lieu de `ng-controller` , si vous avez quelque chose comme ceci:

```
$stateProvider
  .state('users', {
    url: '/users',
    templateUrl: 'user-list.html',
    controller: 'UserListController'
  })
  // ..
```

vous pouvez simplement changer votre déclaration d'état pour:

```
$stateProvider
    .state('users', {
      url: '/',
      template: '<user-list></user-list>'
    })
    // ..
```

Et après?

Maintenant que vous avez un composant contenant votre application (qu'elle contienne l'intégralité de l'application ou une partie de celle-ci, comme une vue), vous devez maintenant commencer à diviser votre composant en plusieurs composants imbriqués, en encapsulant des parties dans de nouveaux sous-composants. , etc.

Vous devriez commencer à utiliser les fonctionnalités du composant comme

- Liaisons des **entrées** et des **sorties**
- **des crochets de cycle de vie** tels que `$onInit()` , `$onChanges()` , etc ...

Après avoir lu la documentation des composants que vous devriez déjà savoir comment utiliser toutes les fonctionnalités de composants, mais si vous avez besoin d' un exemple concret d'une véritable application simple, vous pouvez vérifier cela .

En outre, si vous disposez de certaines fonctions contenant beaucoup de code logique dans le contrôleur de votre composant, vous pouvez envisager de transférer cette logique dans des

services .

Conclusion

Adopter une approche basée sur les composants vous permet de faire évoluer votre AngularJS vers le nouveau framework Angular, mais le rend également meilleur et beaucoup plus modulaire.

Bien sûr, il y a beaucoup d'autres étapes à franchir pour aller plus loin dans la nouvelle direction Angular 2+, que je vais énumérer dans les exemples suivants.

Présentation des modules Webpack et ES6

En utilisant un ***chargeur de module*** tel que Webpack, nous pouvons bénéficier du système de module intégré disponible dans **ES6** (ainsi que dans **TypeScript**). Nous pouvons alors utiliser les fonctionnalités d' importation et d' exportation qui nous permettent de spécifier les morceaux de code que nous pouvons partager entre les différentes parties de l'application.

Lorsque nous introduisons ensuite nos applications dans la production, les chargeurs de modules facilitent également leur conditionnement dans des lots de production avec les piles incluses.

Lire Migration vers Angular 2+ en ligne: https://riptutorial.com/fr/angularjs/topic/9942/migration-vers-angular-2plus

Chapitre 32: Modules

Examples

Modules

Le module sert de conteneur de différentes parties de votre application, telles que des contrôleurs, des services, des filtres, des directives, etc. Les modules peuvent être référencés par d'autres modules via le mécanisme d'injection de dépendance d'Angular.

Créer un module:

```
angular
    .module('app', []);
```

Array `[]` passé dans l'exemple ci-dessus est la *liste des modules* `app` dépend, s'il n'y a pas de dépendances alors nous passons Emrayy Array ie `[]` .

L'injection d'un module comme dépendance d'un autre module:

```
angular.module('app', [
    'app.auth',
    'app.dashboard'
]);
```

Référencement d'un module:

```
angular
    .module('app');
```

Modules

Le module est un conteneur pour différentes parties de vos applications - contrôleur, services, filtres, directive, etc.

Pourquoi utiliser les modules
La plupart des applications ont une méthode principale qui instancie et connecte les différentes parties de l'application.
Les applications angulaires n'ont pas de méthode principale.
Mais dans AngularJs, le processus déclaratif est facile à comprendre et on peut empaqueter du code en tant que modules réutilisables.
Les modules peuvent être chargés dans n'importe quel ordre car les modules retardent l'exécution.

déclarer un module

```
var app = angular.module('myApp', []);
// Empty array is list of modules myApp is depends on.
// if there are any required dependancies,
// then you can add in module, Like ['ngAnimate']

app.controller('myController', function() {

  // write your business logic here
});
```

Chargement de module et dépendances

1. Blocs de configuration: - s'exécuter pendant la phase de fournisseur et de configuration.

```
angular.module('myModule', []).
config(function(injectables) {
  // here you can only inject providers in to config blocks.
});
```

2. Run Blocks: - s'exécutent après la création de l'injecteur et sont utilisés pour démarrer l'application.

```
angular.module('myModule', []).
run(function(injectables) {
  // here you can only inject instances in to config blocks.
});
```

Lire Modules en ligne: https://riptutorial.com/fr/angularjs/topic/844/modules

Chapitre 33: ng-repeat

Introduction

La directive `ngRepeat` instancie un modèle une fois par article d'une collection. La collection doit être un tableau ou un objet. Chaque instance de modèle reçoit sa propre étendue, où la variable de boucle donnée est définie sur l'élément de collection en cours et `$index` est défini sur l'index ou la clé de l'élément.

Syntaxe

- `<element ng-repeat="expression"></element>`
- `<div ng-repeat="(key, value) in myObj">...</div>`
- `<div ng-repeat="variable in expression">...</div>`

Paramètres

Variable	Détails
`$index`	offset de l'itérateur de *nombre* de l'élément répété (0..length-1)
`$first`	*booléen* true si l'élément répété est le premier dans l'itérateur.
`$middle`	*booléen* true si l'élément répété est entre le premier et le dernier dans l'itérateur.
`$last`	*booléen* true si l'élément répété est le dernier dans l'itérateur.
`$even`	*booléen* true si la position `$index` l'itérateur `$index` est pair (sinon faux).
`$odd`	*booléen* true si la position `$index` l'itérateur `$index` est impair (sinon faux).

Remarques

AngularJS fournit ces paramètres sous la forme de variables spéciales disponibles dans l'expression ng-repeat et à l'intérieur de la balise HTML sur laquelle la ng-repeat se déroule.

Examples

Itération sur les propriétés de l'objet

```
<div ng-repeat="(key, value) in myObj"> ... </div>
```

Par exemple

```
<div ng-repeat="n in [42, 42, 43, 43]">
  {{n}}
</div>
```

Suivi et doublons

`ngRepeat` utilise $ watchCollection pour détecter les modifications dans la collection. Lorsqu'un changement se produit, `ngRepeat` effectue alors les modifications correspondantes dans le DOM:

- Lorsqu'un élément est ajouté, une nouvelle instance du modèle est ajoutée au DOM.
- Lorsqu'un élément est supprimé, son instance de modèle est supprimée du DOM.
- Lorsque les articles sont réorganisés, leurs modèles respectifs sont réorganisés dans le DOM.

Doublons

- `track by` toute liste pouvant inclure des valeurs en double.
- `track by` accélère également considérablement les changements de liste.
- Si vous n'utilisez pas `track by` dans ce cas, vous obtenez l'erreur suivante: `[ngRepeat:dupes]`

```
$scope.numbers = ['1','1','2','3','4'];

<ul>
  <li ng-repeat="n in numbers track by $index">
    {{n}}
  </li>
</ul>
```

ng-repeat-start + ng-repeat-end

AngularJS 1.2 `ng-repeat` gère plusieurs éléments avec `ng-repeat-start` et `ng-repeat-end` :

```
// table items
$scope.tableItems = [
    {
        row1: 'Item 1: Row 1',
        row2: 'Item 1: Row 2'
    },
    {
        row1: 'Item 2: Row 1',
        row2: 'Item 2: Row 2'
    }
];

// template
<table>
    <th>
        <td>Items</td>
    </th>
    <tr ng-repeat-start="item in tableItems">
        <td ng-bind="item.row1"></td>
    </tr>
    <tr ng-repeat-end>
        <td ng-bind="item.row2"></td>
```

```
    </tr>
</table>
```

Sortie:

Articles
Point 1: Rangée 1
Point 1: Rangée 2
Item 2: Rangée 1
Item 2: Rangée 2

Lire ng-repeat en ligne: https://riptutorial.com/fr/angularjs/topic/8118/ng-repeat

Chapitre 34: ng-style

Introduction

La directive 'ngStyle' vous permet de définir le style CSS sur un élément HTML de manière conditionnelle. Tout comme nous pourrions utiliser l'attribut de *style* sur l'élément HTML dans les projets non-AngularJS, nous pouvons utiliser `ng-style` dans les styles angularjs qui sont basés sur des conditions booléennes.

Syntaxe

- `<ANY ng-style="expression"></ANY >`
- `<ANY class="ng-style: expression;"> ... </ANY>`

Examples

Utilisation de style ng

L'exemple ci-dessous modifie l'opacité de l'image en fonction du paramètre "status".

```
<img class="img-responsive"  ng-src="{{imagesrc}}"
    ng-style="{'opacity' : (status == 2) ? 1 : 0.5}">
```

Lire ng-style en ligne: https://riptutorial.com/fr/angularjs/topic/8773/ng-style

Chapitre 35: ng-view

Introduction

ng-view est une directive intégrée à la construction que angular utilise comme conteneur pour basculer entre les vues. {info} ngRoute ne fait plus partie du fichier angular.js de base, vous devrez donc inclure le fichier angular-route.js après le fichier javascript angulaire de base. Nous pouvons configurer une route en utilisant la fonction «when» de $ routeProvider. Nous devons d'abord spécifier la route, puis dans un deuxième paramètre, fournir un objet avec une propriété templateUrl et une propriété de contrôleur.

Examples

ng-view

`ng-view` est une directive utilisée avec `$route` pour afficher une vue partielle dans la présentation de la page principale. Ici, dans cet exemple, Index.html est notre fichier principal et lorsque l'utilisateur arrive sur "/" route, le templateURL home.html sera affiché dans Index.html où `ng-view` est mentionné.

```
angular.module('ngApp', ['ngRoute'])

.config(function($routeProvider){
  $routeProvider.when("/",
    {
      templateUrl: "home.html",
      controller: "homeCtrl"
    }
  );
});

angular.module('ngApp').controller('homeCtrl',['$scope', function($scope) {
  $scope.welcome= "Welcome to stackoverflow!";
}]);

//Index.html
<body ng-app="ngApp">
    <div ng-view></div>
</body>

//Home Template URL or home.html
<div><h2>{{welcome}}</h2></div>
```

Enregistrement de navigation

1. Nous injectons le module dans l'application

```
  var Registration=angular.module("myApp",["ngRoute"]);
```

2. maintenant nous utilisons $ routeProvider de "ngRoute"

```
    Registration.config(function($routeProvider) {

});
```

3. enfin nous intégrons la route, nous définissons "/ add" le routage vers l'application au cas où l'application "/ add" la détournerait vers regi.htm

```
 Registration.config(function($routeProvider) {
    $routeProvider
    .when("/add", {
        templateUrl : "regi.htm"
    })
});
```

Lire ng-view en ligne: https://riptutorial.com/fr/angularjs/topic/8833/ng-view

Chapitre 36: Options de liaisons AngularJS (`=`, `@`, `&` etc.)

Remarques

Utilisez ce lanceur pour jouer avec des exemples.

Examples

@ liaison unidirectionnelle, liaison d'attribut.

Transmettez une valeur littérale (pas un objet), telle qu'une chaîne ou un nombre.

La portée enfant a sa propre valeur, si elle met à jour la valeur, la portée parent a sa propre valeur (la portée de l'enfant ne peut pas modifier la valeur de la portée parens). Lorsque la valeur de la portée parent est modifiée, la valeur de la portée enfant sera également modifiée. Toutes les interpolations apparaissent chaque fois lors d'un appel abrégé, pas seulement lors de la création d'une directive.

```
<one-way text="Simple text." <!-- 'Simple text.' -->
         simple-value="123" <!-- '123' Note, is actually a string object. -->
         interpolated-value="{{parentScopeValue}}" <!-- Some value from parent scope. You
can't change parent scope value, only child scope value. Note, is actually a string object. --
>
         interpolated-function-value="{{parentScopeFunction()}}" <!-- Executes parent scope
function and takes a value. -->

         <!-- Unexpected usage. -->
         object-item="{{objectItem}}" <!-- Converts object|date to string. Result might be:
'{"a":5,"b":"text"}'. -->
         function-item="{{parentScopeFunction}}"> <!-- Will be an empty string. -->
</one-way>
```

= liaison bidirectionnelle.

En passant une valeur par référence, vous souhaitez partager la valeur entre les deux étendues et les manipuler à partir des deux portées. Vous ne devez pas utiliser {{...}} pour l'interpolation.

```
<two-way text="'Simple text.'" <!-- 'Simple text.' -->
         simple-value="123" <!-- 123 Note, is actually a number now. -->
         interpolated-value="parentScopeValue" <!-- Some value from parent scope. You may
change it in one scope and have updated value in another. -->
         object-item="objectItem" <!-- Some object from parent scope. You may change object
properties in one scope and have updated properties in another. -->

         <!-- Unexpected usage. -->
         interpolated-function-value="parentScopeFunction()" <!-- Will raise an error. -->
         function-item="incrementInterpolated"> <!-- Pass the function by reference and you
may use it in child scope. -->
```

```
</two-way>
```

Passer fonction par référence est une mauvaise idée: pour permettre à la portée de modifier la définition d'une fonction, et deux observateurs inutiles seront créés, vous devez minimiser le nombre d'observateurs.

& liaison de fonction, liaison d'expression.

Transmettre une méthode dans une directive. Il permet d'exécuter une expression dans le contexte de l'étendue parent. La méthode sera exécutée dans la portée du parent, vous pouvez y transmettre certains paramètres de la portée de l'enfant. Vous ne devez pas utiliser {{...}} pour l'interpolation. Lorsque vous utilisez & dans une directive, il génère une fonction qui renvoie la valeur de l'expression évaluée par rapport à la portée parente (différente de = où vous ne faites que passer une référence).

```
<expression-binding interpolated-function-value="incrementInterpolated(param)" <!--
interpolatedFunctionValue({param: 'Hey'}) will call passed function with an argument. -->
                    function-item="incrementInterpolated" <!-- functionItem({param: 'Hey'})()
will call passed function, but with no possibility set up a parameter. -->
                    text="'Simple text.'"  <!-- text() == 'Simple text.'-->
                    simple-value="123" <!-- simpleValue() == 123 -->
                    interpolated-value="parentScopeValue" <!-- interpolatedValue() == Some
value from parent scope. -->
                    object-item="objectItem"> <!-- objectItem() == Object item from parent
scope. -->
</expression-binding>
```

Tous les paramètres seront intégrés dans des fonctions.

Liaison disponible via un échantillon simple

```
angular.component("SampleComponent", {
  bindings: {
     title: '@',
     movies: '<',
     reservation: "=",
     processReservation: "&"
  }
});
```

Ici, nous avons tous les éléments de liaison.

@ indique que nous avons besoin d'une **liaison** très **simple** , de l'étendue parent à l'étendue enfants, sans aucun observateur, de quelque manière que ce soit. Chaque mise à jour de l'étendue parent resterait dans l'étendue parent et toute mise à jour de l'étendue enfant ne serait pas communiquée à l'étendue parent.

< indique une **liaison à sens unique** . Les mises à jour dans l'étendue parent seraient propagées à l'étendue enfants, mais toute mise à jour dans l'étendue enfants ne serait pas appliquée à l'étendue parent.

= est déjà connu comme une liaison bidirectionnelle. Chaque mise à jour de l'étendue parent serait appliquée aux enfants et chaque mise à jour enfant serait appliquée à l'étendue parent.

& est maintenant utilisé pour une liaison de sortie. Selon la documentation du composant, il doit être utilisé pour référencer la méthode de la portée parent. Au lieu de manipuler la portée des enfants, appelez simplement la méthode parente avec les données mises à jour!

Attribut facultatif de liaison

```
bindings: {
   mandatory: '='
   optional: '=?',
   foo: '=?bar'
}
```

Les attributs facultatifs doivent être marqués d'un point d'interrogation: `=?` ou `=?bar` . C'est une protection pour l'exception `($compile:nonassign)` .

Lire Options de liaisons AngularJS (`=`, `@`, `&` etc.) en ligne: https://riptutorial.com/fr/angularjs/topic/6149/options-de-liaisons-angularjs--------------amp---etc--

Chapitre 37: Partage de données

Remarques

Une question très courante lorsque vous travaillez avec Angular est de partager des données entre contrôleurs. L'utilisation d'un service est la réponse la plus fréquente. Il s'agit d'un exemple simple démontrant un modèle d' usine pour partager tout type d'objet de données entre deux ou plusieurs contrôleurs. Comme il s'agit d'une référence d'objet partagé, une mise à jour dans un contrôleur sera immédiatement disponible dans tous les autres contrôleurs utilisant le service. Notez que le service et l'usine et les deux fournisseurs .

Examples

Utiliser ngStorage pour partager des données

Tout d'abord, incluez la source **ngStorage** dans votre index.html.

Un exemple d'injection de `ngStorage` src serait:

```
<head>
    <title>Angular JS ngStorage</title>
    <script src =
"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
    <script src="https://rawgithub.com/gsklee/ngStorage/master/ngStorage.js"></script>
</head>
```

`ngStorage` vous donne 2 stockage à savoir: `$localStorage` et `$sessionStorage` . Vous devez exiger ngStorage et Inject les services.

Supposons que si `ng-app="myApp"` , alors vous injecteriez `ngStorage` comme suit:

```
var app = angular.module('myApp', ['ngStorage']);
        app.controller('controllerOne', function($localStorage,$sessionStorage) {
        // an object to share
        var sampleObject = {
            name: 'angularjs',
            value: 1
        };
       $localStorage.valueToShare = sampleObject;
       $sessionStorage.valueToShare = sampleObject;
    })
.controller('controllerTwo', function($localStorage,$sessionStorage) {
    console.log('localStorage: '+ $localStorage +'sessionStorage: '+$sessionStorage);
})
```

`$localStorage` et `$sessionStorage` sont globalement accessibles via tous les contrôleurs tant que vous injectez ces services dans les contrôleurs.

Vous pouvez également utiliser `localStorage` et `sessionStorage` de `HTML5` . Cependant, l'utilisation de

`HTML5 localStorage` nécessiterait que vous sérialisiez et désérialisiez vos objets avant de les utiliser ou de les enregistrer.

Par exemple:

```
var myObj = {
    firstname: "Nic",
    lastname: "Raboy",
    website: "https://www.google.com"
}
//if you wanted to save into localStorage, serialize it
window.localStorage.set("saved", JSON.stringify(myObj));

//unserialize to get object
var myObj = JSON.parse(window.localStorage.get("saved"));
```

Partage de données d'un contrôleur à un autre en utilisant le service

Nous pouvons créer un `service` pour `set` et `get` les données entre les `controllers` , puis injecter ce service dans la fonction de contrôleur où nous voulons l'utiliser.

Un service :

```
app.service('setGetData', function() {
  var data = '';
    getData: function() { return data; },
    setData: function(requestData) { data = requestData; }
});
```

Contrôleurs:

```
app.controller('myCtrl1', ['setGetData',function(setGetData) {

  // To set the data from the one controller
  var data = 'Hello World !!';
  setGetData.setData(data);

}]);

app.controller('myCtrl2', ['setGetData',function(setGetData) {

  // To get the data from the another controller
  var res = setGetData.getData();
  console.log(res); // Hello World !!

}]);
```

Ici, nous pouvons voir que `myCtrl1` est utilisé pour `setting` les données et `myCtrl2` est utilisé pour `getting` les données. Nous pouvons donc partager les données d'un contrôleur avec un autre comme celui-ci.

Lire Partage de données en ligne: https://riptutorial.com/fr/angularjs/topic/1923/partage-de-donnees

Chapitre 38: Portées angulaires

Remarques

Angular utilise une **arborescence** de portées pour lier la logique (depuis les contrôleurs, les directives, etc.) à la vue et constitue le principal mécanisme de détection des modifications dans AngularJS. Une référence plus détaillée pour les étendues peut être trouvée à docs.angularjs.org

La racine de l'arbre est accessible via un service **injectable $ rootScope** . Tous les enfants $ scope héritent des méthodes et des propriétés de leur étendue parent $, permettant aux enfants d'accéder aux méthodes sans utiliser les services angulaires.

Examples

Exemple de base de l'héritage $ scope

```
angular.module('app', [])
.controller('myController', ['$scope', function($scope){
    $scope.person = { name: 'John Doe' };
}]);

<div ng-app="app" ng-conroller="myController">
    <input ng-model="person.name" />
    <div ng-repeat="number in [0,1,2,3]">
        {{person.name}} {{number}}
    </div>
</div>
```

Dans cet exemple, la directive ng-repeat crée une nouvelle portée pour chacun des enfants nouvellement créés.

Ces étendues créées sont des enfants de leur étendue parente (dans ce cas, la portée créée par myController) et, par conséquent, elles héritent de toutes ses propriétés, telles que personne.

Éviter d'hériter des valeurs primitives

En javascript, l'attribution d'une valeur non primitive (telle que Object, Array, Function et bien d' autres) permet de conserver une référence (une adresse dans la mémoire) à la valeur attribuée.

L'affectation d'une valeur primitive (String, Number, Boolean ou Symbol) à deux variables différentes et à leur modification ne changera pas les deux:

```
var x = 5;
var y = x;
y = 6;
console.log(y === x, x, y); //false, 5, 6
```

Mais avec une valeur non primitive, puisque les deux variables sont simplement les références à

gardent le même objet, en changeant une variable **va** changer l'autre:

```
var x = { name : 'John Doe' };
var y = x;
y.name = 'Jhon';
console.log(x.name === y.name, x.name, y.name); //true, John, John
```

En mode angulaire, lorsqu'une étendue est créée, toutes les propriétés de son parent lui sont affectées. Cependant, modifier les propriétés par la suite n'affectera la portée parent que s'il s'agit d'une valeur non primitive:

```
angular.module('app', [])
.controller('myController', ['$scope', function($scope){
    $scope.person = { name: 'John Doe' }; //non-primitive
    $scope.name = 'Jhon Doe'; //primitive
}])
.controller('myController1', ['$scope', function($scope){}]);

<div ng-app="app" ng-controller="myController">
    binding to input works: {{person.name}}<br/>
    binding to input does not work: {{name}}<br/>
    <div ng-controller="myController1">
        <input ng-model="person.name" />
        <input ng-model="name" />
    </div>
</div>
```

Rappelez-vous: dans les étendues angulaires, les portées peuvent être créées de plusieurs manières (telles que les directives intégrées ou personnalisées, la fonction `$scope.$new()`) et le suivi de l'arborescence est probablement impossible.

Utiliser uniquement des valeurs non primitives comme propriétés de portée vous permettra de rester en sécurité (sauf si vous avez besoin d'une propriété pour ne pas hériter ou d'autres cas où vous connaissez l'héritage de la portée).

Une fonction disponible dans toute l'application

Attention, cette approche peut être considérée comme une mauvaise conception pour les applications angulaires, car elle nécessite que les programmeurs se souviennent à la fois de l'emplacement des fonctions dans l'arborescence et de la connaissance de l'héritage de la portée. Dans de nombreux cas, il serait préférable d'injecter un service (pratique angulaire - utilisation de l'héritage de la portée par rapport à l'injection) .

Cet exemple montre seulement comment l'héritage de la portée peut être utilisé pour nos besoins et comment vous pouvez en tirer parti, et non les meilleures pratiques de conception d'une application complète.

Dans certains cas, nous pourrions tirer parti de l'héritage de la portée et définir une fonction en tant que propriété du rootScope. De cette façon, toutes les étendues de l'application (à l'exception des étendues isolées) hériteront de cette fonction et pourront être appelées de n'importe où dans l'application.

```
angular.module('app', [])
.run(['$rootScope', function($rootScope){
    var messages = []
    $rootScope.addMessage = function(msg){
        messages.push(msg);
    }
}]);

<div ng-app="app">
    <a ng-click="addMessage('hello world!')">it could be accsessed from here</a>
    <div ng-include="inner.html"></div>
</div>
```

inner.html:

```
<div>
    <button ng-click="addMessage('page')">and from here to!</button>
</div>
```

Créer des événements $ scope personnalisés

Comme pour les éléments HTML normaux, $ scopes peut avoir ses propres événements. Les événements $ scope peuvent être abonnés de la manière suivante:

```
 $scope.$on('my-event', function(event, args) {
    console.log(args); // { custom: 'data' }
});
```

Si vous devez annuler l'enregistrement d'un écouteur d'événement, la fonction **$ on** renverra une fonction de dissociation. Pour continuer avec l'exemple ci-dessus:

```
var unregisterMyEvent = $scope.$on('my-event', function(event, args) {
    console.log(args); // { custom: 'data' }
    unregisterMyEvent();
});
```

Il existe deux manières de déclencher votre propre événement **$** scope personnalisé **$ broadcast** et **$ emit** . Pour notifier le (s) parent (s) d'une portée d'un événement spécifique, utilisez **$ emit**

```
$scope.$emit('my-event', { custom: 'data' });
```

L'exemple ci-dessus déclenchera tous les écouteurs d'événement pour `my-event` sur la portée parent et continuera jusqu'à l'arborescence des **étendues** vers **$ rootScope,** sauf si un écouteur appelle `stopPropagation` sur l'événement. Seuls les événements déclenchés avec **$** `stopPropagation` peuvent appeler `stopPropagation`

L'inverse de **$ emit** est **$ broadcast** , ce qui déclenchera tous les écouteurs d'événement sur toutes les étendues enfants de l'arborescence qui sont des enfants de la portée appelée **$ broadcast** .

```
$scope.$broadcast('my-event', { custom: 'data' });
```

Les événements déclenchés avec **$ broadcast** ne peuvent pas être annulés.

Utiliser les fonctions $ scope

Bien que la déclaration d'une fonction dans $ rootscope présente des avantages, nous pouvons également déclarer une fonction $ scope toute partie du code injectée par le service $ scope. Contrôleur, par exemple.

Manette

```
myApp.controller('myController', ['$scope', function($scope){
    $scope.myFunction = function () {
        alert("You are in myFunction!");
    };
}]);
```

Vous pouvez maintenant appeler votre fonction depuis le contrôleur en utilisant:

```
$scope.myfunction();
```

Ou via HTML qui se trouve sous ce contrôleur spécifique:

```
<div ng-controller="myController">
    <button ng-click="myFunction()"> Click me! </button>
</div>
```

Directif

Une directive angulaire est un autre endroit où vous pouvez utiliser votre champ d'application:

```
myApp.directive('triggerFunction', function() {
    return {
        scope: {
            triggerFunction: '&'
        },
        link: function(scope, element) {
            element.bind('mouseover', function() {
                scope.triggerFunction();
            });
        }
    };
});
```

Et dans votre code HTML sous le même contrôleur:

```
<div ng-controller="myController">
    <button trigger-function="myFunction()"> Hover over me! </button>
</div>
```

Bien sûr, vous pouvez utiliser ngMouseover pour la même chose, mais ce qui est spécial avec les directives, c'est que vous pouvez les personnaliser comme vous le souhaitez. Et maintenant, vous savez comment utiliser vos fonctions $ scope en leur sein, soyez créatif!

Comment pouvez-vous limiter la portée d'une directive et pourquoi le feriez-vous?

Scope est utilisé comme "colle" pour communiquer entre le contrôleur parent, la directive et le modèle de directive. Chaque fois que l'application AngularJS est amorcée, un objet rootScope est créé. Chaque portée créée par des contrôleurs, des directives et des services est héritée de manière systématique de rootScope.

Oui, nous pouvons limiter la portée d'une directive. Nous pouvons le faire en créant un champ d'intervention isolé.

Il existe 3 types de champs d'application:

1. Portée: Faux (la directive utilise sa portée parente)
2. Portée: True (la directive a une nouvelle portée)
3. Portée: {} (la directive obtient une nouvelle portée isolée)

Directives avec la nouvelle étendue isolée: Lorsque nous créons une nouvelle étendue isolée, elle ne sera pas héritée de la portée parente. Cette nouvelle étendue est appelée étendue isolée car elle est complètement détachée de sa portée parente. Pourquoi? devrions-nous utiliser une portée isolée: nous devrions utiliser une portée isolée lorsque nous souhaitons créer une directive personnalisée car elle s'assurera que notre directive est générique et placée n'importe où dans l'application. La portée parent ne va pas interférer avec la portée de la directive.

Exemple de portée isolée:

```
var app = angular.module("test",[]);

app.controller("Ctrl1",function($scope){
    $scope.name = "Prateek";
    $scope.reverseName = function(){
        $scope.name = $scope.name.split('').reverse().join('');
    };
});
app.directive("myDirective", function(){
    return {
        restrict: "EA",
        scope: {},
        template: "<div>Your name is : {{name}}</div>"+
        "Change your name : <input type='text' ng-model='name'/>"
    };
});
```

Il existe 3 types de préfixes que AngularJS fournit pour la portée isolée:

1. "@" (Liaison de texte / liaison unidirectionnelle)
2. "=" (Liaison directe du modèle / liaison bidirectionnelle)
3. "&" (Liaison de comportement / liaison de méthode)

Tous ces préfixes reçoivent des données des attributs de l'élément directive comme:

```
<div my-directive
  class="directive"
  name="{{name}}"
  reverse="reverseName()"
  color="color" >
</div>
```

Lire Portées angulaires en ligne: https://riptutorial.com/fr/angularjs/topic/3157/portees-angulaires

Chapitre 39: Préparez-vous à la production - Grunt

Examples

Afficher le préchargement

Lorsque la première vue temporelle est demandée, Angular effectue `XHR` requête `XHR` pour obtenir cette vue. Pour les projets de taille moyenne, le nombre de vues peut être important et ralentir la réactivité des applications.

La **bonne pratique consiste à pré-charger** toutes les vues en même temps pour les projets de petite et moyenne taille. Pour les projets plus importants, il est également bon de les regrouper dans des volumes importants, mais d'autres méthodes peuvent être utiles pour diviser la charge. Pour automatiser cette tâche, il est utile d'utiliser les tâches Grunt ou Gulp.

Pour pré-charger les vues, nous pouvons utiliser l'objet `$templateCache` . C'est un objet, où angulaire stocke chaque vue reçue du serveur.

Il est possible d'utiliser le module `html2js` , qui convertira toutes nos vues en un seul fichier module-js. Ensuite, nous devrons injecter ce module dans notre application et c'est tout.

Pour créer un fichier concaténé de toutes les vues, nous pouvons utiliser cette tâche

```
module.exports = function (grunt) {
 //set up the location of your views here
 var viewLocation = ['app/views/**.html'];

 grunt.initConfig({
        pkg: require('./package.json'),
            //section that sets up the settings for concatenation of the html files into one
file
            html2js: {
                        options: {
                            base: '',
                            module: 'app.templates', //new module name
                            singleModule: true,
                            useStrict: true,
                            htmlmin: {
                                collapseBooleanAttributes: true,
                                collapseWhitespace: true
                            }
                        },
                        main: {
                            src: viewLocation,
                            dest: 'build/app.templates.js'
                        }
                    },
            //this section is watching for changes in view files, and if there was a change,
it will regenerate the production file. This task can be handy during development.
            watch: {
```

```
                views:{
                    files: viewLocation,
                    tasks: ['buildHTML']
                },
            }
        });

        //to automatically generate one view file
        grunt.loadNpmTasks('grunt-html2js');

        //to watch for changes and if the file has been changed, regenerate the file
        grunt.loadNpmTasks('grunt-contrib-watch');

        //just a task with friendly name to reference in watch
        grunt.registerTask('buildHTML', ['html2js']);
};
```

Pour utiliser cette méthode de concaténation, vous devez apporter 2 modifications: Dans votre fichier `index.html` , vous devez référencer le fichier de vue concaténé.

```
<script src="build/app.templates.js"></script>
```

Dans le fichier où vous déclarez votre application, vous devez injecter la dépendance

```
angular.module('app', ['app.templates'])
```

Si vous utilisez des routeurs populaires comme `ui-router` , il n'y a aucun changement dans la façon dont vous référencez les modèles

```
    .state('home', {
        url: '/home',
        views: {
            "@": {
                controller: 'homeController',
                //this will be picked up from $templateCache
                templateUrl: 'app/views/home.html'
            },
        }

    })
```

Optimisation de script

Il est recommandé de **regrouper les fichiers JS** et de les réduire. Pour les projets plus importants, il pourrait y avoir des centaines de fichiers JS et une latence inutile pour charger chaque fichier séparément du serveur.

Pour une minification angulaire, il est nécessaire de faire annoter toutes les fonctions. Cela nécessaire pour l'injection angulaire de dépendance propre minificaiton. (Pendant la minification, les noms de fonctions et les variables seront renommés et cela interrompra l'injection de dépendances si aucune action supplémentaire ne sera effectuée.)

Au cours de minificaiton `$scope` et `myService` variables seront remplacées par d'autres valeurs.

L'injection de dépendance angulaire fonctionne en fonction du nom, par conséquent, ces noms ne doivent pas changer

```
.controller('myController', function($scope, myService){
})
```

Angular comprendra la notation du tableau, car la minification ne remplacera pas les littéraux de chaîne.

```
.controller('myController', ['$scope','myService', function($scope, myService){
}])
```

- Nous allons tout d'abord concilier tous les fichiers.
- Deuxièmement, nous utiliserons le module `ng-annotate` , qui préparera le code pour la minification
- Enfin, nous appliquerons le module `uglify` .

module.exports = function (grunt) {// configure l'emplacement de vos scripts ici pour le réutiliser dans le code var scriptLocation = ['app / scripts / *. js'];

```
 grunt.initConfig({
        pkg: require('./package.json'),
            //add necessary annotations for safe minification
         ngAnnotate: {
            angular: {
                src: ['staging/concatenated.js'],
                dest: 'staging/anotated.js'
            }
        },
        //combines all the files into one file
        concat: {
                js: {
                    src: scriptLocation,
                    dest: 'staging/concatenated.js'
                }
            },
        //final uglifying
        uglify: {
            options: {
                report: 'min',
                mangle: false,
                sourceMap:true
            },
            my_target: {
                files: {
                    'build/app.min.js': ['staging/anotated.js']
                }
            }
        },

        //this section is watching for changes in JS files, and if there was a change, it will
regenerate the production file. You can choose not to do it, but I like to keep concatenated
version up to date
        watch: {
            scripts: {
                files: scriptLocation,
```

```
                tasks: ['buildJS']
            }
        }

});

    //module to make files less readable
    grunt.loadNpmTasks('grunt-contrib-uglify');

    //mdule to concatenate files together
    grunt.loadNpmTasks('grunt-contrib-concat');

    //module to make angularJS files ready for minification
    grunt.loadNpmTasks('grunt-ng-annotate');

    //to watch for changes and if the file has been changed, regenerate the file
    grunt.loadNpmTasks('grunt-contrib-watch');

    //task that sequentially executes all steps to prepare JS file for production
    //concatinate all JS files
    //annotate JS file (prepare for minification
    //uglify file
     grunt.registerTask('buildJS', ['concat:js', 'ngAnnotate', 'uglify']);
};
```

Lire Préparez-vous à la production - Grunt en ligne:
https://riptutorial.com/fr/angularjs/topic/4434/preparez-vous-a-la-production---grunt

Chapitre 40: Prestations de service

Examples

Comment créer un service

```
angular.module("app")
  .service("counterService", function(){

    var service = {
        number: 0
    };

    return service;
   });
```

Comment utiliser un service

```
    angular.module("app")

        // Custom services are injected just like Angular's built-in services
        .controller("step1Controller", ['counterService', '$scope', function(counterService,
$scope) {
            counterService.number++;
            // bind to object (by reference), not to value, for automatic sync
            $scope.counter = counterService;
        })
```

Dans le modèle utilisant ce contrôleur, vous écrivez alors:

```
// editable
<input ng-model="counter.number" />
```

ou

```
// read-only
<span ng-bind="counter.number"></span>
```

Bien sûr, en code réel, vous interagissez avec le service en utilisant des méthodes sur le contrôleur, qui à son tour délègue au service. L'exemple ci-dessus incrémente simplement la valeur du compteur chaque fois que le contrôleur est utilisé dans un modèle.

Les services à Angularjs sont singletons:

Les services sont des objets singleton qui ne sont instanciés qu'une fois par application (par l'injecteur $) et chargés paresseux (créés uniquement si nécessaire).

> Un singleton est une classe qui ne permet de créer qu'une seule instance - et donne

un accès simple et facile à cette instance. Comme indiqué ici

Créer un service avec angular.factory

Définissez d'abord le service (dans ce cas, il utilise le modèle d'usine):

```
.factory('dataService', function() {
    var dataObject = {};
    var service = {
        // define the getter method
        get data() {
            return dataObject;
        },
        // define the setter method
        set data(value) {
            dataObject = value || {};
        }
    };
    // return the "service" object to expose the getter/setter
    return service;
})
```

Vous pouvez maintenant utiliser le service pour partager des données entre les contrôleurs:

```
.controller('controllerOne', function(dataService) {
    // create a local reference to the dataService
    this.dataService = dataService;
    // create an object to store
    var someObject = {
        name: 'SomeObject',
        value: 1
    };
    // store the object
    this.dataService.data = someObject;
})

.controller('controllerTwo', function(dataService) {
    // create a local reference to the dataService
    this.dataService = dataService;
    // this will automatically update with any changes to the shared data object
    this.objectFromControllerOne = this.dataService.data;
})
```

$ sce - assainit et rend le contenu et les ressources dans des modèles

> $ sce ("Strict Contextual Escaping") est un service angulaire intégré qui nettoie automatiquement le contenu et les sources internes dans les modèles.

L'injection **de** sources **externes** et **de HTML brut** dans le modèle nécessite l' `$sce` manuelle de `$sce` .

Dans cet exemple, nous allons créer un simple filtre d'assainissement $ sce: `.

Démo

```
.filter('sanitizer', ['$sce', [function($sce) {
     return function(content) {
          return $sce.trustAsResourceUrl(content);
      };
}]);
```

Utilisation dans le modèle

```
<div ng-repeat="item in items">

    // Sanitize external sources
    <ifrmae ng-src="{{item.youtube_url | sanitizer}}">

    // Sanitaize and render HTML
    <div ng-bind-html="{{item.raw_html_content| sanitizer}}"></div>

</div>
```

Comment créer un service avec des dépendances en utilisant la "syntaxe de tableau"

```
angular.module("app")
  .service("counterService", ["fooService", "barService", function(anotherService,
barService){

    var service = {
        number: 0,
        foo: function () {
            return fooService.bazMethod(); // Use of 'fooService'
        },
        bar: function () {
            return barService.bazMethod(); // Use of 'barService'
        }
    };

    return service;
   }]);
```

Enregistrement d'un service

La méthode la plus courante et la plus flexible pour créer un service utilise la fabrique angular.module API:

```
angular.module('myApp.services', []).factory('githubService', function() {
   var serviceInstance = {};
   // Our first service
   return serviceInstance;
});
```

La fonction fabrique de services peut être une fonction ou un tableau, tout comme la façon dont nous créons les contrôleurs:

```
// Creating the factory through using the
```

```
// bracket notation
angular.module('myApp.services', [])
.factory('githubService', [function($http) {
}]);
```

Pour exposer une méthode sur notre service, nous pouvons la placer en tant qu'attribut sur l'objet de service.

```
angular.module('myApp.services', [])
    .factory('githubService', function($http) {
        var githubUrl = 'https://api.github.com';
        var runUserRequest = function(username, path) {
        // Return the promise from the $http service
        // that calls the Github API using JSONP
        return $http({
            method: 'JSONP',
            url: githubUrl + '/users/' +
            username + '/' +
            path + '?callback=JSON_CALLBACK'
        });
    }
// Return the service object with a single function
// events
return {
    events: function(username) {
    return runUserRequest(username, 'events');
    }
};
```

Différence entre service et usine

1) Services

Un service est une fonction `constructor` qui est appelée une fois lors de l'exécution avec `new` , tout comme ce que nous ferions avec un javascript simple, à la différence `AngularJs` appelle le `new` dans les coulisses.

Il y a une règle de pouce à retenir en cas de services

1. Les services sont des constructeurs appelés avec de `new`

Voyons un exemple simple où nous enregistrerions un service qui utilise le service `$http` pour récupérer les détails de l'étudiant, et l'utiliser dans le contrôleur

```
function StudentDetailsService($http) {
  this.getStudentDetails = function getStudentDetails() {
    return $http.get('/details');
  };
}

angular.module('myapp').service('StudentDetailsService', StudentDetailsService);
```

Nous venons d'injecter ce service dans le contrôleur

```
function StudentController(StudentDetailsService) {
  StudentDetailsService.getStudentDetails().then(function (response) {
      // handle response
    });
}
angular.module('app').controller('StudentController', StudentController);
```

Quand utiliser?

Utilisez `.service()` où que vous souhaitiez utiliser un constructeur. Il est généralement utilisé pour créer des API publiques comme `getStudentDetails()` . Mais si vous ne voulez pas utiliser un constructeur et que vous souhaitez utiliser un modèle d'API simple, il n'y a pas beaucoup de flexibilité dans `.service()` .

2) usine

Même si nous pouvons réaliser toutes les choses en utilisant `.factory()` , en utilisant `.services()` , cela ne rend pas " `.factory()` identique à" `.service()` . Il est beaucoup plus puissant et flexible que `.service()`

Un `.factory()` est un modèle de conception utilisé pour renvoyer une valeur.

Il y a deux règles de pouce à retenir dans le cas des usines

1. Les usines renvoient des valeurs
2. Les usines (peuvent) créer des objets (n'importe quel objet)

Voyons quelques exemples de ce que nous pouvons faire en utilisant `.factory()`

Retour des objets littéraux

Permet de voir un exemple où factory est utilisé pour renvoyer un objet en utilisant un modèle de module de base Revealing

```
function StudentDetailsService($http) {
  function getStudentDetails() {
    return $http.get('/details');
  }
  return {
    getStudentDetails: getStudentDetails
  };
}

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

Utilisation dans un contrôleur

```
function StudentController(StudentDetailsService) {
  StudentDetailsService.getStudentDetails().then(function (response) {
      // handle response
    });
}
angular.module('app').controller('StudentController', StudentController);
```

Fermetures de retour

Qu'est-ce qu'une fermeture?

Les fermetures sont des fonctions qui font référence à des variables utilisées localement, MAIS définies dans une portée englobante.

Voici un exemple de fermeture

```
function closureFunction(name) {
  function innerClosureFunction(age) { // innerClosureFunction() is the inner function, a
closure
    // Here you can manipulate 'age' AND 'name' variables both
  };
};
```

La partie *"merveilleuse"* est qu'elle peut accéder au `name` qui se trouve dans la portée parent.

`.factory()` exemple de fermeture ci-dessus à l'intérieur de `.factory()`

```
function StudentDetailsService($http) {
  function closureFunction(name) {
  function innerClosureFunction(age) {
    // Here you can manipulate 'age' AND 'name' variables
    };
  };
};

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

Utilisation dans un contrôleur

```
function StudentController(StudentDetailsService) {
  var myClosure = StudentDetailsService('Student Name'); // This now HAS the
innerClosureFunction()
  var callMyClosure = myClosure(24); // This calls the innerClosureFunction()
};

angular.module('app').controller('StudentController', StudentController);
```

Création de constructeurs / instances

`.service()` crée des constructeurs avec un appel à `new` comme vu ci-dessus. `.factory()` peut également créer des constructeurs avec un appel à `new`

Voyons un exemple sur la façon d'y parvenir

```
function StudentDetailsService($http) {
  function Student() {
    this.age = function () {
        return 'This is my age';
    };
  }
  Student.prototype.address = function () {
```

```
        return 'This is my address';
  };
  return Student;
};

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

Utilisation dans un contrôleur

```
function StudentController(StudentDetailsService) {
  var newStudent = new StudentDetailsService();

  //Now the instance has been created. Its properties can be accessed.

 newStudent.age();
 newStudent.address();

};

angular.module('app').controller('StudentController', StudentController);
```

Lire Prestations de service en ligne: https://riptutorial.com/fr/angularjs/topic/1486/prestations-de-service

Chapitre 41: Profilage de performance

Examples

Tout sur le profilage

Qu'est-ce que le profilage?

Par définition, le profilage est une forme d'analyse de programme dynamique qui mesure, par exemple, l'espace (mémoire) ou la complexité temporelle d'un programme, l'utilisation d'instructions particulières ou la fréquence et la durée des appels de fonction.

Pourquoi est-ce nécessaire?

Le profilage est important car vous ne pouvez pas optimiser efficacement tant que vous ne savez pas ce que votre programme passe le plus de temps à faire. Sans mesurer le temps d'exécution de votre programme (profilage), vous ne saurez pas si vous l'avez réellement amélioré.

Outils et techniques:

1. Outils de développement intégrés à Chrome

   Cela inclut un ensemble complet d’outils à utiliser pour le profilage. Vous pouvez aller plus loin pour trouver des goulots d’étranglement dans votre fichier javascript, vos fichiers CSS, vos animations, votre consommation de processeur, les fuites de mémoire, le réseau, la sécurité, etc.

   Effectuez un enregistrement sur la timeline et recherchez les événements d'évaluation du script d'une durée anormalement longue. Si vous en trouvez, vous pouvez activer JS Profiler et refaire votre enregistrement pour obtenir des informations plus détaillées sur les fonctions JS qui ont été appelées et leur durée. Lire la suite...

2. FireBug (utilisation avec Firefox)

3. Dynatrace (utiliser avec IE)

4. Batarang (utiliser avec Chrome)

   C'est un add-on obsolète pour le navigateur Chrome, même s'il est stable et peut être utilisé pour surveiller les modèles, les performances, les dépendances pour une application angulaire. Cela fonctionne très bien pour les applications à petite échelle et peut vous donner un aperçu de ce que la variable de portée retient à différents niveaux. Il vous informe sur les observateurs actifs, les expressions de surveillance et les collections de montres dans l'application.

5. Watcher (utiliser avec Chrome)

   Interface utilisateur agréable et simpliste pour compter le nombre d'observateurs dans une

application angulaire.

6. Utilisez le code suivant pour rechercher manuellement le nombre d'observateurs dans votre application angulaire (crédit à @Words Like Jared Nombre d'observateurs)

```
(function() {
    var root = angular.element(document.getElementsByTagName('body')),
        watchers = [],
        f = function(element) {
        angular.forEach(['$scope', '$isolateScope'], function(scopeProperty) {
            if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
                angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
                watchers.push(watcher);
                });
            }
        });

        angular.forEach(element.children(), function(childElement) {
            f(angular.element(childElement));
        });
    };

    f(root);

    // Remove duplicate watchers
    var watchersWithoutDuplicates = [];
    angular.forEach(watchers, function(item) {
        if(watchersWithoutDuplicates.indexOf(item) < 0) {
            watchersWithoutDuplicates.push(item);
        }
    });
    console.log(watchersWithoutDuplicates.length);
})();
```

7. Plusieurs outils / sites Web en ligne sont disponibles pour faciliter un large éventail de fonctionnalités afin de créer un profil de votre application.

   Un de ces sites est: https://www.webpagetest.org/

   Avec cela, vous pouvez exécuter un test de vitesse de site Web gratuit à partir de plusieurs endroits dans le monde en utilisant de vrais navigateurs (IE et Chrome) et à des vitesses de connexion réelles. Vous pouvez exécuter des tests simples ou effectuer des tests avancés, notamment des transactions en plusieurs étapes, la capture vidéo, le blocage de contenu et bien plus encore.

Prochaines étapes:

Fait avec profilage. Il ne vous amène qu'à mi-chemin. La tâche suivante consiste à transformer vos résultats en éléments d'action pour optimiser votre application. Consultez cette documentation pour savoir comment améliorer les performances de votre application angulaire avec des astuces simples.

Bonne codage :)

Lire Profilage de performance en ligne: https://riptutorial.com/fr/angularjs/topic/7033/profilage-de-

performance

Chapitre 42: Profilage et performance

Examples

7 améliorations simples de la performance

1) Utilisez ng-repeat avec parcimonie

L'utilisation de `ng-repeat` dans les vues entraîne généralement de mauvaises performances, en particulier lorsque des `ng-repeat` sont imbriqués.

C'est super lent!

```
<div ng-repeat="user in userCollection">
  <div ng-repeat="details in user">
    {{details}}
  </div>
</div>
```

Essayez d'éviter les répétitions imbriquées autant que possible. Une façon d'améliorer les performances de `ng-repeat` est d'utiliser `track by $index` (ou un autre champ id). Par défaut, `ng-repeat` suit l'objet entier. Avec `track by` , Angular ne regarde l'objet que par l' `$index` ou object.

```
<div ng-repeat="user in userCollection track by $index">
  {{user.data}}
</div>
```

Utilisez d'autres approches telles que la pagination , les parchemins virtuels , les parchemins infinis ou les limites .

---

2) Lier une fois

Angular a une liaison de données bidirectionnelle. Cela a un coût pour être lent s'il est trop utilisé.

Performance plus lente

```
<!-- Default data binding has a performance cost -->
<div>{{ my.data }}</div>
```

Performances plus rapides (AngularJS> = 1.3)

```
<!-- Bind once is much faster -->
<div>{{ ::my.data }}</div>

<div ng-bind="::my.data"></div>
```

```
<!-- Use single binding notation in ng-repeat where only list display is needed  -->
<div ng-repeat="user in ::userCollection">
  {{::user.data}}
</div>
```

L'utilisation de la notation "Lier une fois" indique à Angular d'attendre que la valeur se stabilise après la première série de cycles de digestion. Angular utilisera cette valeur dans le DOM, puis supprimera tous les observateurs de sorte qu'il devienne une valeur statique et ne soit plus lié au modèle.

Le `{{}}` est beaucoup plus lent.

Ce `ng-bind` est une directive et placera un observateur sur la variable transmise. Ainsi, `ng-bind` ne s'appliquera que lorsque la valeur passée changera réellement.

Par contre, les crochets seront vérifiés et rafraîchis dans chaque `$digest` , même si ce n'est pas nécessaire.

---

3) Les fonctions de portée et les filtres prennent du temps

AngularJS a une boucle de résumé. Toutes vos fonctions sont dans une vue et les filtres sont exécutés à chaque exécution du cycle de digestion. La boucle de résumé sera exécutée chaque fois que le modèle est mis à jour et cela peut ralentir votre application (le filtre peut être atteint plusieurs fois avant le chargement de la page).

Éviter ceci:

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.calculateMe()}}</p>
  <p>{{calc.data | heavyFilter}}</p>
</div>
```

Meilleure approche

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.preCalculatedValue}}</p>
  <p>{{calc.data | lightFilter}}</p>
</div>
```

Où le contrôleur peut être:

```
app.controller('bigCalulations', function(valueService) {
    // bad, because this is called in every digest loop
    this.calculateMe = function() {
        var t = 0;
        for(i = 0; i < 1000; i++) {
            t += i;
        }
        return t;
    }
```

```
    // good, because this is executed just once and logic is separated in service to    keep
the controller light
    this.preCalulatedValue = valueService.valueCalculation(); // returns 499500
});
```

---

4 spectateurs

Les observateurs abandonnent énormément la performance. Avec plus d'observateurs, la boucle de résumé prendra plus de temps et l'interface utilisateur ralentira. Si l'observateur détecte un changement, il lancera la boucle de résumé et restituera la vue.

Il existe trois façons de surveiller manuellement les changements de variables dans Angular.

`$watch()` - surveille les changements de valeur

`$watchCollection()` - surveille les changements dans la collection (regarde plus que `$watch` habituel)

`$watch(..., true)` - **Évitez cela** autant que possible, il effectuera une "montre profonde" et diminuera la performance (regarde plus que `watchCollection`)

Notez que si vous liez des variables dans la vue, vous créez de nouvelles montres - utilisez `{{::variable}}` pour éviter de créer une montre, en particulier dans les boucles.

En conséquence, vous devez suivre le nombre de visiteurs que vous utilisez. Vous pouvez compter les observateurs avec ce script (crédit à @Words Like Jared Nombre de spectateurs)

```
(function() {
    var root = angular.element(document.getElementsByTagName('body')),
        watchers = [],
        f = function(element) {
        angular.forEach(['$scope', '$isolateScope'], function(scopeProperty) {
            if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
                angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
                watchers.push(watcher);
                });
            }
        });

        angular.forEach(element.children(), function(childElement) {
            f(angular.element(childElement));
        });
    };

    f(root);

    // Remove duplicate watchers
    var watchersWithoutDuplicates = [];
    angular.forEach(watchers, function(item) {
        if(watchersWithoutDuplicates.indexOf(item) < 0) {
            watchersWithoutDuplicates.push(item);
        }
    });
    console.log(watchersWithoutDuplicates.length);
```

```
})();
```

---

5) ng-if / ng-show

Ces fonctions ont un comportement très similaire. `ng-if` supprime les éléments du DOM alors que `ng-show` cache uniquement les éléments mais garde tous les gestionnaires. Si vous ne souhaitez pas afficher certaines parties du code, utilisez `ng-if` .

Cela dépend du type d'utilisation, mais souvent l'un convient mieux que l'autre.

- Si l'élément n'est pas nécessaire, utilisez `ng-if`
- Pour activer / désactiver rapidement, utilisez `ng-show/ng-hide`

```
<div ng-repeat="user in userCollection">
  <p ng-if="user.hasTreeLegs">I am special<!-- some complicated DOM --></p>
  <p ng-show="user.hasSubscribed">I am awesome<!-- switch this setting on and off --></p>
</div>
```

En cas de doute - utilisez `ng-if` et testez!

---

6) Désactiver le débogage

Par défaut, les directives de liaison et les étendues laissent des classes et des balises supplémentaires dans le code pour aider à divers outils de débogage. La désactivation de cette option signifie que vous ne restituez plus ces différents éléments pendant le cycle de digestion.

```
angular.module('exampleApp', []).config(['$compileProvider', function ($compileProvider) {
    $compileProvider.debugInfoEnabled(false);
}]);
```

---

7) Utiliser l'injection de dépendance pour exposer vos ressources

Dependency Injection est un modèle de conception logicielle dans lequel un objet reçoit ses dépendances, plutôt que l'objet qui les crée lui-même. Il s'agit de supprimer les dépendances codées en dur et de les modifier chaque fois que nécessaire.

Vous pourriez vous interroger sur le coût des performances associé à une telle analyse syntaxique de toutes les fonctions injectables. Angular s'en charge en mettant en cache la propriété $ inject après la première fois. Cela ne se produit donc pas chaque fois qu'une fonction doit être invoquée.

CONSEIL PRO: Si vous recherchez une approche offrant les meilleures performances, utilisez

l'approche d'annotation de la propriété $ inject. Cette approche évite entièrement l'analyse de définition de fonction car cette logique est incluse dans la vérification suivante dans la fonction d'annotation: if (! ($ Inject = fn. $ Inject)). Si $ inject est déjà disponible, aucune analyse n'est requise!

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

var DemoController = function (s, h) {
    h.get('https://api.github.com/users/angular/repos').success(function (repos) {
        s.repos = repos;
    });
}
// $inject property annotation
DemoController['$inject'] = ['$scope', '$http'];

app.controller('DemoController', DemoController);
```

CONSEIL PRO 2: Vous pouvez ajouter une directive `ng-strict-di` sur le même élément que `ng-app` pour opter en mode DI strict, ce qui provoquera une erreur à chaque fois qu'un service essaiera d'utiliser des annotations implicites. Exemple:

```
<html ng-app="DemoApp" ng-strict-di>
```

Ou si vous utilisez un amorçage manuel:

```
angular.bootstrap(document, ['DemoApp'], {
    strictDi: true
});
```

Lier une fois

Angular a la réputation d'avoir une liaison de données bidirectionnelle impressionnante. Par défaut, Angular synchronise en permanence les valeurs liées aux composants du modèle et de la vue à tout moment les modifications de données dans le composant de modèle ou de vue.

Cela a un coût pour être un peu lent s'il est trop utilisé. Cela aura un impact plus important sur la performance:

Mauvaises performances: `{{my.data}}`

Ajoutez deux colons `::` avant le nom de la variable pour utiliser la liaison unique. Dans ce cas, la valeur est mise à jour uniquement lorsque my.data est défini. Vous pointez explicitement pour ne pas surveiller les modifications de données. Angular n'effectue aucune vérification de valeur, ce qui réduit le nombre d'expressions évaluées à chaque cycle de digestion.

Bons exemples de performances utilisant une liaison unique

```
{{::my.data}}
<span ng-bind="::my.data"></span>
<span ng-if="::my.data"></span>
<span ng-repeat="item in ::my.data">{{item}}</span>
```

```
<span ng-class="::{ 'my-class': my.data }"></div>
```

Remarque: Ceci supprime toutefois la liaison de données bidirectionnelle pour `my.data` , de sorte que chaque fois que ce champ change dans votre application, la même chose ne sera pas automatiquement reflétée dans la vue. Donc, **utilisez-le uniquement pour les valeurs qui ne changeront pas pendant toute la durée de vie de votre application** .

Fonctions et filtres de portée

AngularJS a la boucle de résumé et toutes vos fonctions dans une vue et les filtres sont exécutés chaque fois que le cycle de digestion est exécuté. La boucle de résumé sera exécutée chaque fois que le modèle est mis à jour et cela peut ralentir votre application (le filtre peut être atteint plusieurs fois avant le chargement de la page).

Vous devriez éviter ceci:

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.calculateMe()}}</p>
  <p>{{calc.data | heavyFilter}}</p>
</div>
```

Meilleure approche

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.preCalculatedValue}}</p>
  <p>{{calc.data | lightFilter}}</p>
</div>
```

Où échantillon de contrôleur est:

```
.controller("bigCalulations", function(valueService) {
  // bad, because this is called in every digest loop
  this.calculateMe = function() {
    var t = 0;
    for(i = 0; i < 1000; i++) {
      t = t + i;
    }
    return t;
  }
  //good, because it is executed just once and logic is separated in service to    keep the
controller light
  this.preCalulatedValue = valueService.caluclateSumm(); // returns 499500
});
```

Observateurs

Les observateurs nécessaires pour surveiller certaines valeurs et détecter que cette valeur est modifiée.

Après l'appel de `$watch()` ou `$watchCollection` nouvel observateur ajoute à la collection d'observateurs internes dans la portée actuelle.

Alors, qu'est-ce que l'observateur?

Watcher est une fonction simple, appelée à chaque cycle de digestion et qui renvoie une valeur. Angular vérifie la valeur renvoyée, si ce n'est pas la même chose que lors de l'appel précédent - un rappel transmis en second paramètre à la fonction `$watch()` ou `$watchCollection` sera exécuté.

```
(function() {
  angular.module("app", []).controller("ctrl", function($scope) {
    $scope.value = 10;
    $scope.$watch(
      function() { return $scope.value; },
      function() { console.log("value changed"); }
    );
  }
})();
```

Les observateurs sont des tueurs de performance. Plus vous avez de observateurs, plus ils prennent de temps pour faire une boucle de lecture, l'interface utilisateur la plus lente. Si un observateur détecte des changements, il lancera la boucle de résumé (recalcul sur tous les écrans)

Il existe trois façons de faire une surveillance manuelle pour les changements de variables dans Angular.

`$watch()` - surveille simplement les changements de valeur

`$watchCollection()` - surveille les changements dans la collection (regarde plus que $ watch habituel)

`$watch(..., true)` - **Évitez cela** autant que possible, il effectuera "deep watch" et tuera la performance (regarde plus que watchCollection)

Notez que si vous liez des variables dans la vue, vous créez de nouveaux observateurs - utilisez `{{::variable}}` pour ne pas créer d'observateur, en particulier dans les boucles

En conséquence, vous devez suivre le nombre de visiteurs que vous utilisez. Vous pouvez compter les observateurs avec ce script (crédit à @Words Like Jared - Comment compter le nombre total de montres sur une page?

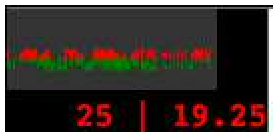
```
(function() {
  var root = angular.element(document.getElementsByTagName("body")),
      watchers = [];

  var f = function(element) {

    angular.forEach(["$scope", "$isolateScope"], function(scopeProperty) {
      if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
        angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
          watchers.push(watcher);
        });
      }
    });
```

```
    angular.forEach(element.children(), function(childElement) {
      f(angular.element(childElement));
    });

  };

  f(root);

  // Remove duplicate watchers
  var watchersWithoutDuplicates = [];
  angular.forEach(watchers, function(item) {
    if(watchersWithoutDuplicates.indexOf(item) < 0) {
      watchersWithoutDuplicates.push(item);
    }
  });

  console.log(watchersWithoutDuplicates.length);

})();
```

Si vous ne voulez pas créer votre propre script, il existe un utilitaire open source appelé ng-stats qui utilise un graphique en temps réel incorporé dans la page pour vous donner un aperçu du nombre de montres gérées par Angular, ainsi fréquence et durée des cycles de digestion au fil du temps. L'utilitaire expose une fonction globale nommée `showAngularStats` que vous pouvez appeler pour configurer le fonctionnement du graphique.

```
showAngularStats({
  "position": "topleft",
  "digestTimeThreshold": 16,
  "autoload": true,
  "logDigest": true,
  "logWatches": true
});
```

L'exemple de code ci-dessus affiche automatiquement le graphique suivant sur la page (démonstration interactive).



ng-if vs ng-show

Ces fonctions ont un comportement très similaire. La différence est que `ng-if` supprime les éléments du DOM. S'il y a de grandes parties du code qui ne seront pas affichées, alors `ng-if` est la voie à suivre. `ng-show` ne fera que cacher les éléments mais gardera tous les gestionnaires.

ng-if

La directive ngIf supprime ou recrée une partie de l'arborescence DOM basée sur une expression.

Si l'expression affectée à ngIf est évaluée à une valeur fausse, l'élément est supprimé du DOM, sinon un clone de l'élément est réinséré dans le DOM.

ng-show

La directive ngShow affiche ou masque l'élément HTML donné en fonction de l'expression fournie à l'attribut ngShow. L'élément est affiché ou masqué en supprimant ou en ajoutant la classe CSS ng-hide à l'élément.

Exemple

```
<div ng-repeat="user in userCollection">
  <p ng-if="user.hasTreeLegs">I am special
    <!-- some complicated DOM -->
  </p>
  <p ng-show="user.hasSubscribed">I am aweosme
    <!-- switch this setting on and off -->
  </p>
</div>
```

Conclusion

Cela dépend du type d'utilisation, mais souvent l'un est plus adapté que l'autre (par exemple, si 95% du temps, l'élément n'est pas nécessaire, utilisez `ng-if` ; si vous devez basculer la visibilité de l'élément DOM, utilisez `ng-show`).

En cas de doute, utilisez `ng-if` et testez!

Remarque : `ng-if` crée une nouvelle étendue isolée, alors que `ng-show` et `ng-hide` ne le font pas. Utilisez `$parent.property` si la propriété de la portée parent n'y est pas directement accessible.

Debounce votre modèle

```
<div ng-controller="ExampleController">
    <form name="userForm">
        Name:
        <input type="text" name="userName"
           ng-model="user.name"
           ng-model-options="{ debounce: 1000 }" />
            <button ng-click="userForm.userName.$rollbackViewValue();
user.name=''">Clear</button><br />
    </form>
    <pre>user.name = </pre>
</div>
```

L'exemple ci-dessus définit une valeur de rebond de 1 000 millisecondes, soit 1 seconde. Il s'agit d'un retard considérable, mais empêchera l'entrée de supprimer à plusieurs reprises le `ng-model`

avec plusieurs cycles `$digest` .

En utilisant l'anti-rebond sur vos champs de saisie et partout où une mise à jour instantanée n'est pas requise, vous pouvez augmenter considérablement les performances de vos applications angulaires. Non seulement vous pouvez retarder le temps, mais vous pouvez également retarder le déclenchement de l’action. Si vous ne souhaitez pas mettre à jour votre modèle ng à chaque frappe, vous pouvez également mettre à jour le flou.

Toujours désinscrire les auditeurs inscrits sur d'autres portées que l'étendue actuelle

Vous devez toujours désenregistrer les portées autres que votre portée actuelle, comme indiqué ci-dessous:

```
//always deregister these
$rootScope.$on(...);
$scope.$parent.$on(...);
```

Vous n'êtes pas obligé d'annuler l'enregistrement des auditeurs sur la portée actuelle car anguleux s'en chargerait:

```
//no need to deregister this
$scope.$on(...);
```

`$rootScope.$on` écouteurs restera en mémoire si vous naviguez vers un autre contrôleur. Cela créera une fuite de mémoire si le contrôleur est hors de portée.

Ne pas

```
angular.module('app').controller('badExampleController', badExample);
badExample.$inject = ['$scope', '$rootScope'];

function badExample($scope, $rootScope) {
    $rootScope.$on('post:created', function postCreated(event, data) {});
}
```

Faire

```
angular.module('app').controller('goodExampleController', goodExample);
goodExample.$inject = ['$scope', '$rootScope'];

function goodExample($scope, $rootScope) {
    var deregister = $rootScope.$on('post:created', function postCreated(event, data) {});

    $scope.$on('$destroy', function destroyScope() {
        deregister();
    });
}
```

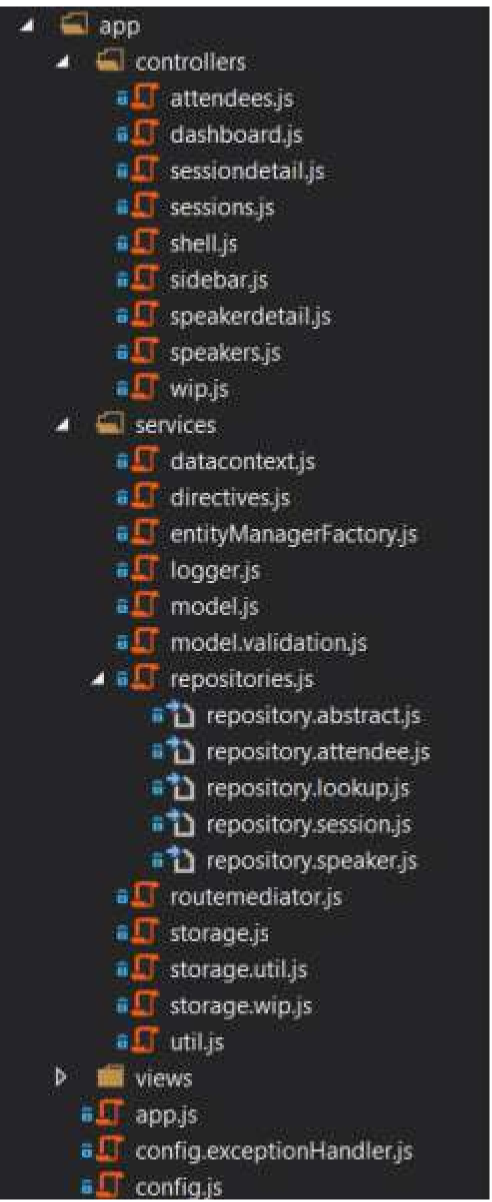
Lire Profilage et performance en ligne: https://riptutorial.com/fr/angularjs/topic/1921/profilage-et-performance

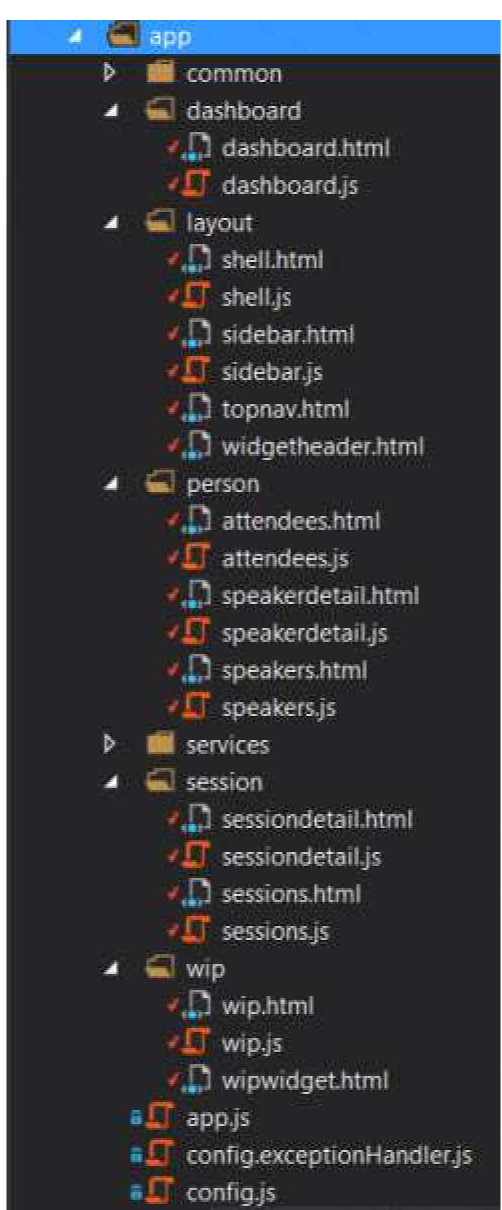
Chapitre 43: Projet angulaire - Structure des répertoires

Examples

Structure du répertoire

Une question commune parmi les nouveaux programmeurs angulaires - "Quelle devrait être la structure du projet?". Une bonne structure contribue à un développement d'applications évolutif. Lorsque nous démarrons un projet, nous avons deux choix: **Trier par type** (à gauche) et **Trier par entité** (à droite). La seconde est meilleure, en particulier dans les grandes applications, le projet devient beaucoup plus facile à gérer.





Trier par type (à gauche)

L'application est organisée par type de fichier.

- **Avantage** - Bon pour les petites applications, pour les programmeurs qui commencent seulement à utiliser Angular, et facile à convertir à la seconde méthode.
- **Inconvénient** - Même pour les petites applications, il devient plus difficile de trouver un fichier spécifique. Par exemple, une vue et son contrôleur sont dans deux dossiers distincts.

Trier par élément (à droite)

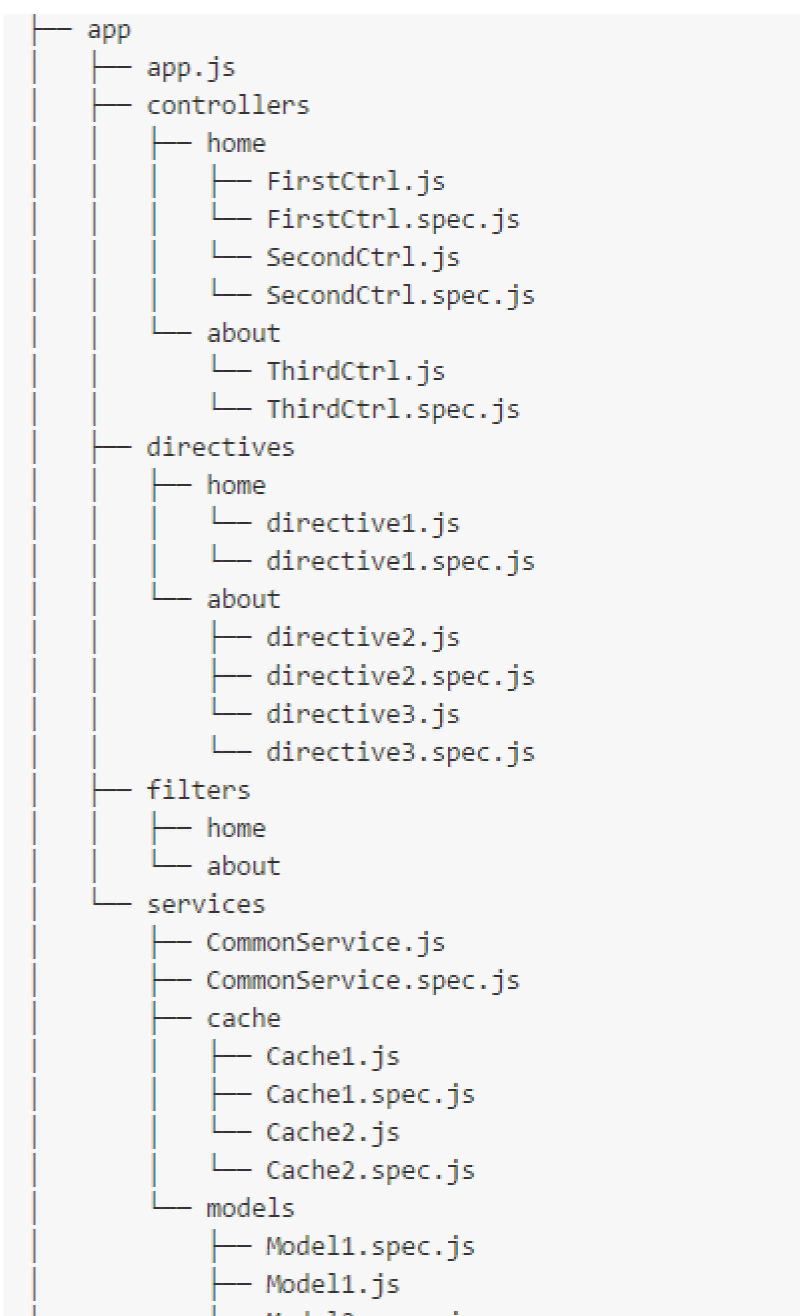
La méthode d'organisation suggérée où les fichiers sont classés par type d'entités.

Toutes les vues et tous les contrôleurs de la mise en page vont dans le dossier de mise en page, le contenu de l'administrateur va dans le dossier admin, etc.

- **Avantage** - Lorsque vous recherchez une section de code déterminant une fonctionnalité donnée, tout se trouve dans un seul dossier.
- **Inconvénient** - Les services sont un peu différents car ils «desservent» de nombreuses fonctionnalités.

Vous pouvez en savoir plus sur la structure angulaire: Refactoring for Growth

La structure de fichier suggérée combinant les deux méthodes susmentionnées:



Crédit à: Guide de style angulaire

Lire Projet angulaire - Structure des répertoires en ligne: https://riptutorial.com/fr/angularjs/topic/6148/projet-angulaire---structure-des-repertoires

Chapitre 44: Routage à l'aide de ngRoute

Remarques

Le `ngRoute` est un module intégré fournissant des services de routage et de deeplinking et des directives pour les applications angulaires.

Une documentation complète sur `ngRoute` est disponible sur https://docs.angularjs.org/api/ngRoute

Examples

Exemple de base

Cet exemple montre comment configurer une petite application avec 3 itinéraires, chacun avec sa propre vue et son propre contrôleur, en utilisant la syntaxe `controllerAs` .

Nous configurons notre routeur à la fonction angulaire `.config`

1. Nous injectons `$routeProvider` dans `.config`
2. Nous définissons nos noms de route à la méthode `.when` avec un objet de définition de route.
3. Nous fournissons la méthode `.when` avec un objet spécifiant notre `template` ou `templateUrl` , notre `controller` et nos `controllerAs`

app.js

```
angular.module('myApp', ['ngRoute'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .config(function($routeProvider) {
    $routeProvider
    .when('/one', {
      templateUrl: 'view-one.html',
      controller: 'controllerOne',
      controllerAs: 'ctrlOne'
    })
    .when('/two', {
      templateUrl: 'view-two.html',
      controller: 'controllerTwo',
      controllerAs: 'ctrlTwo'
    })
    .when('/three', {
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
```

```
    })
    // redirect to here if no other routes match
    .otherwise({
      redirectTo: '/one'
    });
  });
```

Ensuite, dans notre HTML, nous définissons notre navigation à l'aide des éléments `<a>` avec `href` , pour un nom de route `helloRoute` nous acheminerons comme `<a href="#/helloRoute">My route</a>`

Nous fournissons également notre vue avec un conteneur et la directive `ng-view` pour injecter nos routes.

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a href="#/one">View One</a>
    <a href="#/two">View Two</a>
    <a href="#/three">View Three</a>
  </nav>
  <!-- views will be injected here -->
  <div ng-view></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

Exemple de paramètres de route

Cet exemple étend l'exemple de base en passant des paramètres dans la route afin de les utiliser dans le contrôleur

Pour ce faire, nous devons:

1. Configurez la position et le nom du paramètre dans le nom de la route
2. Injecter le service `$routeParams` dans notre contrôleur

app.js

```
angular.module('myApp', ['ngRoute'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
```

```
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', ['$routeParams', function($routeParams) {
    var routeParam = $routeParams.paramName

    if ($routeParams.message) {
        // If a param called 'message' exists, we show it's value as the message
        this.message = $routeParams.message;
    } else {
        // If it doesn't exist, we show a default message
        this.message = 'Hello world from Controller Three!';
    }
  }])
  .config(function($routeProvider) {
    $routeProvider
    .when('/one', {
      templateUrl: 'view-one.html',
      controller: 'controllerOne',
      controllerAs: 'ctrlOne'
    })
    .when('/two', {
      templateUrl: 'view-two.html',
      controller: 'controllerTwo',
      controllerAs: 'ctrlTwo'
    })
    .when('/three', {
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
    })
    .when('/three/:message', { // We will pass a param called 'message' with this route
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
    })
    // redirect to here if no other routes match
    .otherwise({
      redirectTo: '/one'
    });
  });
```

Alors, sans apporter de modifications à nos modèles, en ajoutant uniquement un nouveau lien avec un message personnalisé, nous pouvons voir le nouveau message personnalisé à notre avis.

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a href="#/one">View One</a>
    <a href="#/two">View Two</a>
    <a href="#/three">View Three</a>
    <!-- New link with custom message -->
    <a href="#/three/This-is-a-message">View Three with "This-is-a-message" custom message</a>
  </nav>
  <!-- views will be injected here -->
  <div ng-view></div>
  <!-- templates can live in normal html files -->
```

```
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

Définir un comportement personnalisé pour des itinéraires individuels

La manière la plus simple de définir un comportement personnalisé pour des itinéraires individuels serait assez facile.

Dans cet exemple, nous l'utilisons pour authentifier un utilisateur:

1) `routes.js` : créer une nouvelle propriété (comme `requireAuth`) pour n'importe quelle route désirée

```
angular.module('yourApp').config(['$routeProvider', function($routeProvider) {
    $routeProvider
        .when('/home', {
            templateUrl: 'templates/home.html',
            requireAuth: true
        })
        .when('/login', {
            templateUrl: 'templates/login.html',
        })
        .otherwise({
            redirectTo: '/home'
        });
}])
```

2) Dans un contrôleur de premier niveau qui n'est pas lié à un élément dans la `ng-view` (pour éviter les conflits avec l'angle angulaire `$routeProvider`), vérifiez si la nouvelle `newUrl` possède la propriété `requireAuth` et agissez en conséquence

```
angular.module('YourApp').controller('YourController', ['$scope', 'session', '$location',
    function($scope, session, $location) {

        $scope.$on('$routeChangeStart', function(angularEvent, newUrl) {

            if (newUrl.requireAuth && !session.user) {
                // User isn't authenticated
                $location.path("/login");
            }

        });
    }
]);
```

Lire Routage à l'aide de ngRoute en ligne: https://riptutorial.com/fr/angularjs/topic/2391/routage-a-l-aide-de-ngroute

Chapitre 45: SignalR avec AngularJs

Introduction

Dans cet article, nous nous concentrons sur "Comment créer un projet simple en utilisant AngularJs et SignalR", dans cette formation, vous devez connaître "comment créer une application avec angularjs", "créer / utiliser un service sur angulaire" et des connaissances de base sur SignalR "pour cela nous recommandons https://www.codeproject.com/Tips/590660/Introduction-to-SignalR) .

Examples

SignalR et AngularJs [ChatProject]

étape 1: créer un projet

```
- Application
    - app.js
    - Controllers
        - appController.js
    - Factories
        - SignalR-factory.js
- index.html
- Scripts
    - angular.js
    - jquery.js
    - jquery.signalR.min.js
- Hubs
```

> Version de signal signal: signalR-2.2.1

Étape 2: Startup.cs et ChatHub.cs

Allez dans votre répertoire *"/ Hubs"* et ajoutez 2 fichiers *[Startup.cs, ChatHub.cs]*

Startup.cs

```
using Microsoft.Owin;
using Owin;
[assembly: OwinStartup(typeof(SignalR.Hubs.Startup))]

namespace SignalR.Hubs
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

ChatHub.cs

```
using Microsoft.AspNet.SignalR;

namespace SignalR.Hubs
{
    public class ChatHub : Hub
    {
        public void Send(string name, string message, string time)
        {
            Clients.All.broadcastMessage(name, message, time);
        }
    }
}
```

étape 3: créer une application angulaire

Accédez à votre répertoire *"/ Application"* et ajoutez le fichier *[app.js]*

app.js

```
var app = angular.module("app", []);
```

étape 4: créer SignalR Factory

Allez dans votre répertoire *"/ Application / Factories"* et ajoutez le fichier *[SignalR-factory.js]*

SignalR-factory.js

```
app.factory("signalR", function () {
    var factory = {};

    factory.url = function (url) {
        $.connection.hub.url = url;
    }

    factory.setHubName = function (hubName) {
        factory.hub = hubName;
    }

    factory.connectToHub = function () {
        return $.connection[factory.hub];
    }

    factory.client = function () {
        var hub = factory.connectToHub();
        return hub.client;
    }

    factory.server = function () {
        var hub = factory.connectToHub();
        return hub.server;
    }

    factory.start = function (fn) {
        return $.connection.hub.start().done(fn);
    }
```

```
    return factory;
});
```

étape 5: mettre à jour app.js

```
var app = angular.module("app", []);

app.run(function(signalR) {
    signalR.url("http://localhost:21991/signalr");
});
```

localhost: 21991 / signalr | **c'est votre Url SignalR Hubs**

étape 6: ajouter un contrôleur

Accédez au répertoire *"/ Application / Controllers"* et ajoutez le fichier *[appController.js]*

```
app.controller("ctrl", function ($scope, signalR) {
    $scope.messages = [];
    $scope.user = {};

    signalR.setHubName("chatHub");

    signalR.client().broadcastMessage = function (name, message, time) {
        var newChat = { name: name, message: message, time: time };

        $scope.$apply(function() {
            $scope.messages.push(newChat);
        });
    };

    signalR.start(function () {
        $scope.send = function () {
            var dt = new Date();
            var time = dt.getHours() + ":" + dt.getMinutes() + ":" + dt.getSeconds();

            signalR.server().send($scope.user.name, $scope.user.message, time);
        }
    });
});
```

signalR.setHubName ("chatHub") | **[ChatHub] (classe publique)>** *ChatHub.cs*

Remarque: *n'insérez* pas *HubName* avec les majuscules, la **première lettre** est la casse inférieure.

signalR.client () | Cette méthode tente de se connecter à vos concentrateurs et d'obtenir toutes les fonctions dans les concentrateurs. Dans cet exemple, nous avons "chatHub" pour obtenir la fonction "broadcastMessage ()";

étape 7: ajouter index.html dans la route du répertoire

index.html

```
<!DOCTYPE html>
<html ng-app="app" ng-controller="ctrl">
<head>
    <meta charset="utf-8" />
    <title>SignalR Simple Chat</title>
</head>
<body>
    <form>
        <input type="text" placeholder="name" ng-model="user.name" />
        <input type="text" placeholder="message" ng-model="user.message" />
        <button ng-click="send()">send</button>

        <ul>
            <li ng-repeat="item in messages">
                <b ng-bind="item.name"></b> <small ng-bind="item.time"></small> :
{{item.message}}
            </li>
        </ul>
    </form>

    <script src="Scripts/angular.min.js"></script>
    <script src="Scripts/jquery-1.6.4.min.js"></script>
    <script src="Scripts/jquery.signalR-2.2.1.min.js"></script>
    <script src="signalr/hubs"></script>
    <script src="app.js"></script>
    <script src="SignalR-factory.js"></script>
</body>
</html
```

Résultat avec image

Utilisateur 1 (envoyer et recevoir)

Utilisateur 2 (envoyer et recevoir)

Lire SignalR avec AngularJs en ligne: https://riptutorial.com/fr/angularjs/topic/9964/signalr-avec-angularjs

Chapitre 46: Stockage de session

Examples

Gestion du stockage de session via le service à l'aide d'angularjs

Service de stockage de session:

Service d'usine commun qui sauvegarde et renvoie les données de session enregistrées en fonction de la clé.

```
 'use strict';

/**
 * @ngdoc factory
 * @name app.factory:storageService
 * @description This function will communicate with HTML5 sessionStorage via Factory Service.
 */

app.factory('storageService', ['$rootScope', function($rootScope) {

    return {
        get: function(key) {
            return sessionStorage.getItem(key);
        },
        save: function(key, data) {
            sessionStorage.setItem(key, data);
        }
    };
}]);
```

Dans le contrôleur:

Injectez la dépendance storageService dans le contrôleur pour définir et obtenir les données du stockage de session.

```
app.controller('myCtrl',['storageService',function(storageService) {

  // Save session data to storageService
  storageService.save('key', 'value');

  // Get saved session data from storageService
  var sessionData = storageService.get('key');

});
```

Lire Stockage de session en ligne: https://riptutorial.com/fr/angularjs/topic/8201/stockage-de-session

Chapitre 47: Tâches Grunt

Examples

Exécutez l'application localement

> L'exemple suivant requiert l'installation de node.js et la disponibilité de npm .
> Un code de travail complet peut être généré à partir de GitHub @
> https://github.com/mikkoviitala/angular-grunt-run-local

Généralement, l'une des premières choses à faire lors du développement d'une nouvelle application Web consiste à le faire fonctionner localement.

Vous trouverez ci-dessous un exemple complet de réalisation, en utilisant grunt (coureur de tâches javascript), npm (gestionnaire de paquets de nœuds) et bower (encore un autre gestionnaire de paquets).

Outre vos fichiers d'application réels, vous devrez installer quelques dépendances tierces à l'aide des outils mentionnés ci-dessus. Dans votre répertoire de projet, de **préférence root** , vous aurez besoin de trois (3) fichiers.

- package.json (dépendances gérées par npm)
- bower.json (dépendances gérées par bower)
- gruntfile.js (tâches grognantes)

Ainsi, votre répertoire de projet ressemble à ceci:

---

package.json

Nous allons installer **grunt** lui-même, **matchdep** pour rendre notre vie plus facile nous permettant de filtrer les dépendances par nom, **grunt-express** utilisé pour démarrer express web server via grunt et **grunt-open** pour ouvrir des URL / fichiers à partir d'une tâche grunt.

Donc, ces paquets sont tout au sujet de "infrastructure" et des aides sur lesquelles nous allons construire notre application.

```
{
  "name": "app",
  "version": "1.0.0",
  "dependencies": {},
  "devDependencies": {
```

```
    "grunt": "~0.4.1",
    "matchdep": "~0.1.2",
    "grunt-express": "~1.0.0-beta2",
    "grunt-open": "~0.2.1"
  },
  "scripts": {
    "postinstall": "bower install"
  }
}
```

bower.json

Bower est (ou du moins devrait être) tout au sujet du frontal et nous l'utiliserons pour installer **angular** .

```
{
  "name": "app",
  "version": "1.0.0",
  "dependencies": {
    "angular": "~1.3.x"
  },
  "devDependencies": {}
}
```

gruntfile.js

A l'intérieur de gruntfile.js, nous aurons la magie "application en cours d'exécution", qui ouvre notre application dans une nouvelle fenêtre du navigateur, exécutée sur http: // localhost: 9000 /

```
'use strict';

// see http://rhumaric.com/2013/07/renewing-the-grunt-livereload-magic/

module.exports = function(grunt) {
  require('matchdep').filterDev('grunt-*').forEach(grunt.loadNpmTasks);

  grunt.initConfig({
    express: {
      all: {
        options: {
          port: 9000,
          hostname: 'localhost',
          bases: [__dirname]
        }
      }
    },

    open: {
      all: {
        path: 'http://localhost:<%= express.all.options.port%>'
      }
    }
  });

  grunt.registerTask('app', [
    'express',
    'open',
```

```
    'express-keepalive'
  ]);
};
```

Usage

Pour faire fonctionner votre application à partir de zéro, enregistrez les fichiers ci-dessus dans le répertoire racine de votre projet (tout dossier vide fera l'affaire). Ensuite, lancez console / ligne de commande et tapez ce qui suit pour installer toutes les dépendances requises.

```
npm install -g grunt-cli bower
npm install
```

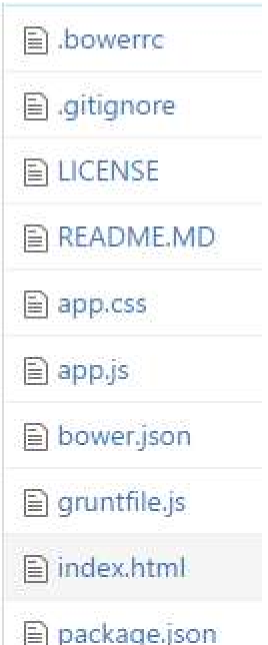
Et puis exécutez votre application en utilisant

```
grunt app
```

Notez que oui, vous aurez également besoin de vos fichiers d'application réels.
Pour un exemple presque minimal, parcourez le référentiel GitHub mentionné au début de cet exemple.

La structure n'est pas si différente. Il y a juste un template `index.html` , un code angulaire dans `app.js` et quelques styles dans `app.css` . D'autres fichiers sont destinés à la configuration de Git et de l'éditeur, ainsi qu'à des éléments génériques. Essaie!

AngularJS application
Hello Stack Overflow Documentation (beta)



Lire Tâches Grunt en ligne: https://riptutorial.com/fr/angularjs/topic/6077/taches-grunt

Chapitre 48: Tests unitaires

Remarques

Cette rubrique fournit des exemples de tests unitaires des différentes constructions dans AngularJS. Les tests unitaires sont souvent écrits à l'aide de Jasmine , un framework de test basé sur le comportement populaire. Lorsque vous testez des constructions angulaires, vous devez inclure ngMock comme dépendance lors de l'exécution des tests unitaires.

Examples

Unité teste un filtre

Code de filtre:

```
angular.module('myModule', []).filter('multiplier', function() {
  return function(number, multiplier) {
    if (!angular.isNumber(number)) {
      throw new Error(number + " is not a number!");
    }
    if (!multiplier) {
      multiplier = 2;
    }
    return number * multiplier;
  }
});
```

Le test:

```
describe('multiplierFilter', function() {
  var filter;

  beforeEach(function() {
    module('myModule');
    inject(function(multiplierFilter) {
      filter = multiplierFilter;
    });
  });

  it('multiply by 2 by default', function() {
    expect(filter(2)).toBe(4);
    expect(filter(3)).toBe(6);
  });

  it('allow to specify custom multiplier', function() {
    expect(filter(2, 4)).toBe(8);
  });

  it('throws error on invalid input', function() {
    expect(function() {
      filter(null);
    }).toThrow();
```

```
  });
});
```

Courir!

Remarque: Dans l'appel d' `inject` dans le test, votre filtre doit être spécifié par son nom + *Filtre* . La raison en est que chaque fois que vous enregistrez un filtre pour votre module, Angular l'enregistre avec un `Filter` ajouté à son nom.

Unité teste un composant (1.5+)

Code du composant:

```
angular.module('myModule', []).component('myComponent', {
  bindings: {
    myValue: '<'
  },
  controller: function(MyService) {
    this.service = MyService;
    this.componentMethod = function() {
      return 2;
    };
  }
});
```

Le test:

```
describe('myComponent', function() {
  var component;

  var MyServiceFake = jasmine.createSpyObj(['serviceMethod']);

  beforeEach(function() {
    module('myModule');
    inject(function($componentController) {
      // 1st - component name, 2nd - controller injections, 3rd - bindings
      component = $componentController('myComponent', {
        MyService: MyServiceFake
      }, {
        myValue: 3
      });
    });
  });

  /** Here you test the injector. Useless. */

  it('injects the binding', function() {
    expect(component.myValue).toBe(3);
  });

  it('has some cool behavior', function() {
    expect(component.componentMethod()).toBe(2);
  });
});
```

Courir!

Unité teste un contrôleur

Code du contrôleur:

```
angular.module('myModule', [])
  .controller('myController', function($scope) {
    $scope.num = 2;
    $scope.doSomething = function() {
      $scope.num += 2;
    }
  });
```

Le test:

```
describe('myController', function() {
  var $scope;
  beforeEach(function() {
    module('myModule');
    inject(function($controller, $rootScope) {
      $scope = $rootScope.$new();
      $controller('myController', {
        '$scope': $scope
      })
    });
  });
  it('should increment `num` by 2', function() {
    expect($scope.num).toEqual(2);
    $scope.doSomething();
    expect($scope.num).toEqual(4);
  });
});
```

Courir!

Unité teste un service

Code de service

```
angular.module('myModule', [])
  .service('myService', function() {
    this.doSomething = function(someNumber) {
      return someNumber + 2;
    }
  });
```

Le test

```
describe('myService', function() {
  var myService;
  beforeEach(function() {
    module('myModule');
    inject(function(_myService_) {
      myService = _myService_;
    });
  });
```

```
  it('should increment `num` by 2', function() {
    var result = myService.doSomething(4);
    expect(result).toEqual(6);
  });
});
```

Courir!

Unité teste une directive

Code de la directive

```
angular.module('myModule', [])
  .directive('myDirective', function() {
    return {
      template: '<div>{{greeting}} {{name}}!</div>',
      scope: {
        name: '=',
        greeting: '@'
      }
    };
  });
```

Le test

```
describe('myDirective', function() {
  var element, scope;
  beforeEach(function() {
    module('myModule');
    inject(function($compile, $rootScope) {
      scope = $rootScope.$new();
      element = angular.element("<my-directive name='name' greeting='Hello'></my-directive>");
      $compile(element)(scope);
      /* PLEASE NEVER USE scope.$digest(). scope.$apply use a protection to avoid to run a
digest loop when there is already one, so, use scope.$apply() instead. */
      scope.$apply();
    })
  });

  it('has the text attribute injected', function() {
    expect(element.html()).toContain('Hello');
  });

  it('should have proper message after scope change', function() {
    scope.name = 'John';
    scope.$apply();
    expect(element.html()).toContain("John");
    scope.name = 'Alice';
    expect(element.html()).toContain("John");
    scope.$apply();
    expect(element.html()).toContain("Alice");
  });
});
```

Courir!

Lire Tests unitaires en ligne: https://riptutorial.com/fr/angularjs/topic/1689/tests-unitaires

Chapitre 49: ui-routeur

Remarques

Qu'est-ce `ui-router` ?

> Angular UI-Router est une structure de routage d'application de page unique côté client pour AngularJS.
>
> Les structures de routage pour les SPA mettent à jour l'URL du navigateur lorsque l'utilisateur navigue dans l'application. Inversement, cela permet de modifier l'URL du navigateur pour naviguer dans l'application, permettant ainsi à l'utilisateur de créer un signet dans un emplacement situé au cœur du SPA.
>
> Les applications UI-Router sont modélisées sous la forme d’une arborescence hiérarchique. UI-Router fournit une machine d'état pour gérer les transitions entre ces états d'application de manière transactionnelle.

Liens utiles

Vous pouvez trouver la documentation API officielle ici . Si vous avez des questions sur `ui-router` VS `ng-router` , vous pouvez trouver une réponse assez détaillée ici . Gardez à l'esprit que `ng-router` a déjà publié une nouvelle mise à jour de ng-routeur appelée `ngNewRouter` (compatible avec Angular 1.5 + / 2.0) qui prend en charge les états comme ui-router. Vous pouvez en savoir plus sur `ngNewRouter` ici .

Examples

Exemple de base

app.js

```
angular.module('myApp', ['ui.router'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
      .state('one', {
        url: "/one",
        templateUrl: "view-one.html",
        controller: 'controllerOne',
        controllerAs: 'ctrlOne'
```

```
      })
      .state('two', {
        url: "/two",
        templateUrl: "view-two.html",
        controller: 'controllerTwo',
        controllerAs: 'ctrlTwo'
      })
      .state('three', {
        url: "/three",
        templateUrl: "view-three.html",
        controller: 'controllerThree',
        controllerAs: 'ctrlThree'
      });

      $urlRouterProvider.otherwise('/one');
  });
```

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a ui-sref="one">View One</a>
    <a ui-sref="two">View Two</a>
    <a ui-sref="three">View Three</a>
  </nav>
  <!-- views will be injected here -->
  <div ui-view></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

Vues multiples

app.js

```
angular.module('myApp', ['ui.router'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .controller('controllerFour', function() {
    this.message = 'Hello world from Controller Four!';
```

```
  })
  .config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
      .state('one', {
        url: "/one",
        views: {
          "viewA": {
            templateUrl: "view-one.html",
            controller: 'controllerOne',
            controllerAs: 'ctrlOne'
          },
          "viewB": {
            templateUrl: "view-two.html",
            controller: 'controllerTwo',
            controllerAs: 'ctrlTwo'
          }
        }
      })
      .state('two', {
        url: "/two",
        views: {
          "viewA": {
            templateUrl: "view-three.html",
            controller: 'controllerThree',
            controllerAs: 'ctrlThree'
          },
          "viewB": {
            templateUrl: "view-four.html",
            controller: 'controllerFour',
            controllerAs: 'ctrlFour'
          }
        }
      });

    $urlRouterProvider.otherwise('/one');
  });
```

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a ui-sref="one">Route One</a>
    <a ui-sref="two">Route Two</a>
  </nav>
  <!-- views will be injected here -->
  <div ui-view="viewA"></div>
  <div ui-view="viewB"></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
```

```
  <script type="text/ng-template" id="view-four.html">
    <h1>{{ctrlFour.message}}</h1>
  </script>
</div>
```

Utilisation des fonctions de résolution pour charger des données

app.js

```
angular.module('myApp', ['ui.router'])
  .service('User', ['$http', function User ($http) {
    this.getProfile = function (id) {
      return $http.get(...) // method to load data from API
    };
  }])
  .controller('profileCtrl', ['profile', function profileCtrl (profile) {
    // inject resolved data under the name of the resolve function
    // data will already be returned and processed
    this.profile = profile;
  }])
  .config(['$stateProvider', '$urlRouterProvider', function ($stateProvider,
$urlRouterProvider) {
    $stateProvider
      .state('profile', {
        url: "/profile/:userId",
        templateUrl: "profile.html",
        controller: 'profileCtrl',
        controllerAs: 'vm',
        resolve: {
          profile: ['$stateParams', 'User', function ($stateParams, User) {
            // $stateParams will contain any parameter defined in your url
            return User.getProfile($stateParams.userId)
              // .then is only necessary if you need to process returned data
              .then(function (data) {
                return doSomeProcessing(data);
              });
          }]
        }
      }]);

      $urlRouterProvider.otherwise('/');
  });
```

profile.html

```
<ul>
  <li>Name: {{vm.profile.name}}</li>
  <li>Age: {{vm.profile.age}}</li>
  <li>Sex: {{vm.profile.sex}}</li>
</ul>
```

Voir l' entrée Wiki UI-Routeur sur resolves ici .

Les fonctions de résolution doivent être résolues avant que l'événement `$stateChangeSuccess` soit déclenché, ce qui signifie que l'interface utilisateur ne se charge pas tant que *toutes* les fonctions

de résolution de l'état ne sont pas terminées. C'est un excellent moyen de vous assurer que les données seront disponibles pour votre contrôleur et votre interface utilisateur. Cependant, vous pouvez voir qu'une fonction de résolution doit être rapide afin d'éviter de suspendre l'interface utilisateur.

Vues imbriquées / États

app.js

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

app.config(function($stateProvider,$urlRouterProvider) {

    $stateProvider

    .state('home', {
        url: '/home',
        templateUrl: 'home.html',
        controller: function($scope){
            $scope.text = 'This is the Home'
        }
    })

    .state('home.nested1',{
        url: '/nested1',
        templateUrl:'nested1.html',
        controller: function($scope){
            $scope.text1 = 'This is the nested view 1'
        }
    })

    .state('home.nested2',{
        url: '/nested2',
        templateUrl:'nested2.html',
        controller: function($scope){
            $scope.fruits = ['apple','mango','oranges'];
        }
    });

    $urlRouterProvider.otherwise('/home');

});
```

index.html

```
  <div ui-view></div>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
   <script src="angular-ui-router.min.js"></script>
   <script src="app.js"></script>
```

home.html

```
<div>
<h1> {{text}} </h1>
<br>
```

```
    <a ui-sref="home.nested1">Show nested1</a>
    <br>
    <a ui-sref="home.nested2">Show nested2</a>
    <br>

    <div ui-view></div>
</div>
```

nested1.html

```
<div>
<h1> {{text1}} </h1>
</div>
```

nested2.html

```
<div>
    <ul>
      <li ng-repeat="fruit in fruits">{{ fruit }}</li>
    </ul>
</div>
```

Lire ui-routeur en ligne: https://riptutorial.com/fr/angularjs/topic/2545/ui-routeur

Chapitre 50: Utilisation de directives intégrées

Examples

Masquer / Afficher les éléments HTML

Cet exemple cache des éléments HTML.

```
<!DOCTYPE html>
<html ng-app="myDemoApp">
  <head>
    <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
    <script>

      function HideShowController() {
        var vm = this;
        vm.show=false;
        vm.toggle= function() {
          vm.show=!vm.show;
        }
      }

      angular.module("myDemoApp", [/* module dependencies go here */])
        .controller("hideShowController", [HideShowController]);
    </script>
  </head>
  <body ng-cloak>
    <div ng-controller="hideShowController as vm">
      <a style="cursor: pointer;" ng-show="vm.show" ng-click="vm.toggle()">Show Me!</a>
      <a style="cursor: pointer;" ng-hide="vm.show" ng-click="vm.toggle()">Hide Me!</a>
    </div>
  </body>
</html>
```

Démo en direct

Explication étape par étape:

1. `ng-app="myDemoApp"` , la directive ngApp indique angulairement qu'un élément DOM est contrôlé par un module angulaire spécifique nommé "myDemoApp".
2. `<script src="[//angular include]">` inclut les js angulaires.
3. `HideShowController` fonction `HideShowController` est définie et contient une autre fonction nommée `toggle` qui aide à masquer l' `HideShowController` l'élément.
4. `angular.module(...)` crée un nouveau module.
5. `.controller(...)` Angular Controller et renvoie le module pour le chaînage;
6. `ng-controller` directive `ng-controller` est un aspect clé de la manière dont angular prend en charge les principes à la base du modèle de conception Model-View-Controller.
7. `ng-show` directive `ng-show` montre l'élément HTML donné si l'expression fournie est vraie.
8. `ng-hide` directive `ng-hide` masque l'élément HTML donné si l'expression fournie est vraie.

9. `ng-click` directive `ng-click` déclenche une fonction de basculement dans le contrôleur

Lire Utilisation de directives intégrées en ligne:
https://riptutorial.com/fr/angularjs/topic/7644/utilisation-de-directives-integrees

Chapitre 51: Utiliser AngularJS avec TypeScript

Syntaxe

- $ scope: ng.IScope - c'est une façon de dactylographier un type pour une variable particulière.

Examples

Contrôleurs angulaires en caractères dactylographiés

Comme défini dans la documentation AngularJS

> Lorsqu'un contrôleur est connecté au DOM via la directive ng-controller, Angular instanciera un nouvel objet Controller, en utilisant la fonction constructeur du contrôleur spécifiée. Une nouvelle étendue enfant sera créée et mise à disposition en tant que paramètre injectable à la fonction constructeur du contrôleur en tant que $ scope.

Les contrôleurs peuvent être réalisés très facilement en utilisant les classes dactylographiées.

```
module App.Controllers {
    class Address {
        line1: string;
        line2: string;
        city: string;
        state: string;
    }
    export class SampleController {
        firstName: string;
        lastName: string;
        age: number;
        address: Address;
        setUpWatches($scope: ng.IScope): void {
            $scope.$watch(() => this.firstName, (n, o) => {
                //n is string and so is o
            });
        };
        constructor($scope: ng.IScope) {
            this.setUpWatches($scope);
        }
    }
}
```

Le Javascript résultant est

```
var App;
(function (App) {
    var Controllers;
```

```
    (function (Controllers) {
        var Address = (function () {
            function Address() {
            }
            return Address;
        }());
        var SampleController = (function () {
            function SampleController($scope) {
                this.setUpWatches($scope);
            }
            SampleController.prototype.setUpWatches = function ($scope) {
                var _this = this;
                $scope.$watch(function () { return _this.firstName; }, function (n, o) {
                    //n is string and so is o
                });
            };
            ;
            return SampleController;
        }());
        Controllers.SampleController = SampleController;
    })(Controllers = App.Controllers || (App.Controllers = {}));
})(App || (App = {}));
//# sourceMappingURL=ExampleController.js.map
```

Après avoir fait la classe de contrôleur laisser le module angulaire js autour du contrôleur peut être fait simple en utilisant la classe

```
app
 .module('app')
 .controller('exampleController', App.Controller.SampleController)
```

Utilisation du contrôleur avec la syntaxe ControllerAs

Le contrôleur que nous avons créé peut être instancié et utilisé en utilisant le `controller as` syntaxe. C'est parce que nous avons placé la variable directement sur la classe du contrôleur et non sur la `$scope` .

Utiliser `controller as someName` consiste à `controller as someName` le contrôleur de `$scope` même. Il n'est donc pas nécessaire d'injecter $ scope en tant que dépendance dans le contrôleur.

Façon traditionnelle :

```
// we are using $scope object.
app.controller('MyCtrl', function ($scope) {
  $scope.name = 'John';
});

<div ng-controller="MyCtrl">
  {{name}}
</div>
```

Maintenant, avec le `controller as` syntaxe :

```
// we are using the "this" Object instead of "$scope"
```

```
app.controller('MyCtrl', function() {
  this.name = 'John';
});

<div ng-controller="MyCtrl as info">
  {{info.name}}
</div>
```

Si vous instanciez une "classe" en JavaScript, vous pouvez le faire:

```
var jsClass = function () {
  this.name = 'John';
}
var jsObj = new jsClass();
```

Donc, maintenant, nous pouvons utiliser l'instance `jsObj` pour accéder à toute méthode ou propriété de `jsClass` .

En angulaire, nous faisons le même type de chose. Nous utilisons le contrôleur comme syntaxe pour l'instanciation.

Utilisation de groupage / minification

La manière dont $ scope est injecté dans les fonctions de constructeur du contrôleur est une façon de démontrer et d'utiliser l'option de base de l' injection de dépendance angulaire, mais n'est pas prête pour la production car elle ne peut pas être réduite. C'est parce que le système de minification modifie les noms de variables et l'injection de dépendance d'anguar utilise les noms de paramètres pour savoir ce qui doit être injecté. Ainsi, pour un exemple, la fonction constructeur de ExampleController est réduite au code suivant.

```
function n(n){this.setUpWatches(n)
```

et `$scope` est changé en `n` !
pour surmonter cela, nous pouvons ajouter un tableau $ inject (`string[]`). Ainsi, l'ID angulaire sait quoi injecter à quelle position se trouve la fonction constructeur du contrôleur.
Donc, le texte ci-dessus change pour

```
module App.Controllers {
    class Address {
        line1: string;
        line2: string;
        city: string;
        state: string;
    }
    export class SampleController {
        firstName: string;
        lastName: string;
        age: number;
        address: Address;
        setUpWatches($scope: ng.IScope): void {
            $scope.$watch(() => this.firstName, (n, o) => {
                //n is string and so is o
```

```
            });
        };
        static $inject : string[] = ['$scope'];
        constructor($scope: ng.IScope) {
            this.setUpWatches($scope);
        }
    }
}
```

Pourquoi la syntaxe ControllerAs?

Fonction du contrôleur

La fonction de contrôleur n'est rien d'autre qu'une fonction de constructeur JavaScript. Par conséquent, lorsqu'une vue se charge, le `function context` (**`this`**) est défini sur l'objet contrôleur.

Cas 1 :

```
this.constFunction = function() { ... }
```

Il est créé dans l' `controller object` , pas sur `$scope` . les vues ne peuvent pas accéder aux fonctions définies sur l'objet contrôleur.

Exemple :

```
<a href="#123" ng-click="constFunction()"></a> // It will not work
```

Cas 2:

```
$scope.scopeFunction = function() { ... }
```

Il est créé dans l' `$scope object` , pas sur `controller object` . les vues ne peuvent accéder qu'aux fonctions définies sur l'objet `$scope` .

Exemple :

```
<a href="#123" ng-click="scopeFunction()"></a> // It will work
```

Pourquoi les contrôleurs?

- **`ControllerAs`** syntaxe rend beaucoup plus claire où les objets sont en cours manipulated.Having `oneCtrl.name` et `anotherCtrl.name` rend beaucoup plus facile d'identifier que vous avez un `name` attribué par plusieurs contrôleurs différents à des fins différentes , mais si les deux mêmes que celles utilisées `$scope.name` et ayant deux éléments HTML différents sur une page, tous deux liés à **`{{name}}`** il est alors difficile d'identifier celui qui provient de quel contrôleur.

- Masquer la `$scope` et exposer les membres du contrôleur à la vue via un `intermediary object` . En définissant `this.*` , Nous pouvons exposer exactement ce que nous voulons exposer du contrôleur à la vue.

  ```
  <div ng-controller="FirstCtrl">
      {{ name }}
      <div ng-controller="SecondCtrl">
          {{ name }}
          <div ng-controller="ThirdCtrl">
              {{ name }}
          </div>
      </div>
  </div>
  ```

Ici, dans le cas ci-dessus, `{{ name }}` sera très déroutant à utiliser et nous ne savons pas non plus lequel est lié à quel contrôleur.

```
<div ng-controller="FirstCtrl as first">
    {{ first.name }}
    <div ng-controller="SecondCtrl as second">
        {{ second.name }}
        <div ng-controller="ThirdCtrl as third">
            {{ third.name }}
        </div>
    </div>
</div>
```

Pourquoi $ scope

- Utilisez `$scope` lorsque vous avez besoin d'accéder à une ou plusieurs méthodes de $ scope telles que `$watch` , `$digest` , `$emit` , `$http` etc.
- limiter les propriétés et / ou méthodes exposées à `$scope` , puis les transmettre explicitement à `$scope` si nécessaire.

Lire Utiliser AngularJS avec TypeScript en ligne: https://riptutorial.com/fr/angularjs/topic/3477/utiliser-angularjs-avec-typescript

Chapitre 52: Validation de formulaire

Examples

Validation de formulaire de base

L'un des atouts d'Angular est la validation du formulaire côté client.

Traiter avec les entrées de formulaire traditionnelles et devoir utiliser un traitement interrogatif de style jQuery peut être long et complexe. Angular vous permet de produire *des* formulaires *interactifs* professionnels relativement facilement.

La directive **ng-model** fournit une liaison bidirectionnelle avec des champs d'entrée et généralement l'attribut **novalidate** est également placé sur l'élément de formulaire pour empêcher le navigateur d'effectuer une validation native.

Ainsi, une forme simple ressemblerait à:

```
<form name="form" novalidate>
  <label name="email"> Your email </label>
  <input type="email" name="email" ng-model="email" />
</form>
```

Pour que les entrées angulaires soient valides, utilisez exactement la même syntaxe qu'un élément d' *entrée* standard, à l'exception de l'ajout de l'attribut **ng-model** pour spécifier la variable à associer à l'étendue. Le courrier électronique est montré dans l'exemple précédent. Pour valider un nombre, la syntaxe serait:

```
<input type="number" name="postalcode" ng-model="zipcode" />
```

Les dernières étapes de la validation de base du formulaire sont la connexion à une fonction d'envoi de formulaire sur le contrôleur à l'aide de **ng-submit** , plutôt que d'autoriser la soumission du formulaire par défaut. Ce n'est pas obligatoire, mais il est généralement utilisé, car les variables d'entrée sont déjà disponibles sur la portée et sont donc disponibles pour votre fonction d'envoi. Il est également bon de donner un nom au formulaire. Ces modifications entraîneraient la syntaxe suivante:

```
<form name="signup_form" ng-submit="submitFunc()" novalidate>
  <label name="email"> Your email </label>
  <input type="email" name="email" ng-model="email" />
  <button type="submit">Signup</button>
</form>
```

Ce code ci-dessus est fonctionnel mais il existe d'autres fonctionnalités fournies par Angular.

L'étape suivante consiste à comprendre que Angular associe les attributs de classe à l'aide de **ng-pristine** , **ng-dirty** , **ng-valid** et **ng-invalid** pour le traitement de formulaire. L'utilisation de ces

classes dans votre css vous permettra de **définir** les champs de saisie **valides / non valides / vierges / sales** et de modifier ainsi la présentation lorsque l'utilisateur entre des données dans le formulaire.

États de formulaire et d'entrée

Les formes angulaires et les entrées ont différents états utiles lors de la validation du contenu

États d'entrée

Etat	La description
`$touched`	Le terrain a été touché
`$untouched`	Le champ n'a pas été touché
`$pristine`	Le champ n'a pas été modifié
`$dirty`	Le champ a été modifié
`$valid`	Le contenu du champ est valide
`$invalid`	Le contenu du champ est invalide

Tous les états ci-dessus sont des propriétés booléennes et peuvent être vrais ou faux.

Avec ceux-ci, il est très facile d'afficher des messages à un utilisateur.

```
<form name="myForm" novalidate>
    <input name="myName" ng-model="myName" required>
    <span ng-show="myForm.myName.$touched && myForm.myName.$invalid">This name is
invalid</span>
</form>
```

Ici, nous utilisons la directive `ng-show` pour afficher un message à un utilisateur qui a modifié un formulaire mais qui est invalide.

Classes CSS

Angular fournit également des classes CSS pour les formulaires et les entrées en fonction de leur état

Classe	La description
`ng-touched`	Le terrain a été touché
`ng-untouched`	Le champ n'a pas été touché
`ng-pristine`	Le champ n'a pas été modifié

Classe	La description
`ng-dirty`	Le champ a été modifié
`ng-valid`	Le champ est valide
`ng-invalid`	Le champ est invalide

Vous pouvez utiliser ces classes pour ajouter des styles à vos formulaires

```
input.ng-invalid {
  background-color: crimson;
}
input.ng-valid {
  background-color: green;
}
```

ngMessages

`ngMessages` est utilisé pour améliorer le style d'affichage des messages de validation dans la vue.

Approche traditionnelle

Avant `ngMessages` , nous `ngMessages` normalement les messages de validation en utilisant des directives prédéfinies angulaires `ng-class` . Cette approche était un déchet et une tâche `repetitive` .

Maintenant, en utilisant `ngMessages` nous pouvons créer nos propres messages personnalisés.

Exemple

Html:

```
<form name="ngMessagesDemo">
  <input name="firstname" type="text" ng-model="firstname" required>
  <div ng-messages="ngMessagesDemo.firstname.$error">
    <div ng-message="required">Firstname is required.</div>
  </div>
</form>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.16/angular.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.16/angular-
messages.min.js"></script>
```

JS:

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

app.controller('mainCtrl', function ($scope) {
```

```
    $scope.firstname = "Rohit";
});
```

Validation de formulaire personnalisé

Dans certains cas, la validation de base ne suffit pas. Prise en charge angulaire de la validation personnalisée ajoutant des fonctions de validation à l'objet `$validators` sur le `ngModelController` :

```
angular.module('app', [])
  .directive('myValidator', function() {
    return {
      // element must have ng-model attribute
      // or $validators does not work
      require: 'ngModel',
      link: function(scope, elm, attrs, ctrl) {
        ctrl.$validators.myValidator = function(modelValue, viewValue) {
            // validate viewValue with your custom logic
            var valid = (viewValue && viewValue.length > 0) || false;
            return valid;
        };
      }
    };
```

Le validateur est défini comme une directive nécessitant `ngModel` . Par conséquent, pour appliquer le validateur, ajoutez simplement la directive personnalisée au contrôle de formulaire en entrée.

```
<form name="form">
  <input type="text"
         ng-model="model"
         name="model"
         my-validator>
  <pre ng-bind="'my-validator returned: ' + form.model.$valid"></pre>
</form>
```

Et `my-validator` ne doit pas nécessairement être appliqué au contrôle de formulaire natif. Il peut s'agir de n'importe quel élément, tant que `ng-model` dans ses attributs. Ceci est utile lorsque vous avez un composant d'interface utilisateur personnalisé.

> some text
>
> my-validator returned: true

Formulaires imbriqués

Parfois, il est souhaitable d'imbriquer des formulaires dans le but de regrouper logiquement les contrôles et les entrées sur la page. Toutefois, les formulaires HTML5 ne doivent pas être imbriqués. Fournitures angulaires `ng-form` place.

```
<form name="myForm" noValidate>
  <!-- nested form can be referenced via 'myForm.myNestedForm' -->
  <ng-form name="myNestedForm" noValidate>
```

```
    <input name="myInput1" ng-minlength="1" ng-model="input1" required />
    <input name="myInput2" ng-minlength="1" ng-model="input2" required />
  </ng-form>

  <!-- show errors for the nested subform here -->
  <div ng-messages="myForm.myNestedForm.$error">
    <!-- note that this will show if either input does not meet the minimum -->
    <div ng-message="minlength">Length is not at least 1</div>
  </div>
</form>

<!-- status of the form -->
<p>Has any field on my form been edited? {{myForm.$dirty}}</p>
<p>Is my nested form valid? {{myForm.myNestedForm.$valid}}</p>
<p>Is myInput1 valid? {{myForm.myNestedForm.myInput1.$valid}}</p>
```

Chaque partie de la forme contribue à l'état général de la forme. Par conséquent, si l'une des entrées `myInput1` a été modifiée et est `$dirty` , sa forme contenant sera également `$dirty` . Cela cascade à chaque formulaire contenant, donc `myNestedForm` et `myForm` seront `$dirty` .

Validateurs asynchrones

Les validateurs asynchrones vous permettent de valider les informations de formulaire par rapport à votre backend (en utilisant $ http).

Ces types de validateurs sont nécessaires lorsque vous devez accéder aux informations stockées sur le serveur que vous ne pouvez pas avoir sur votre client pour diverses raisons, telles que le tableau des utilisateurs et d'autres informations de base de données.

Pour utiliser des validateurs asynchrones, vous accédez au `ng-model` de votre `input` et définissez des fonctions de rappel pour la propriété `$asyncValidators` .

Exemple:

L'exemple suivant vérifie si un nom fourni existe déjà, le serveur retourne un statut qui rejette la promesse si le nom existe déjà ou s'il n'a pas été fourni. Si le nom n'existe pas, il renverra une promesse résolue.

```
ngModel.$asyncValidators.usernameValidate = function (name) {
     if (name) {
          return AuthenticationService.checkIfNameExists(name); // returns a promise
     } else {
          return $q.reject("This username is already taken!"); // rejected promise
     }
};
```

Maintenant, chaque fois que le `ng-model` de l'entrée est modifié, cette fonction s'exécute et renvoie une promesse avec le résultat.

Lire Validation de formulaire en ligne: https://riptutorial.com/fr/angularjs/topic/3979/validation-de-formulaire

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec AngularJS	Abhishek Pandey, After Class, Andrés Encarnación, AnonymousNotReally, badzilla, Charlie H, Chirag Bhatia - chirag64, Community, daniellmb, David G., Devid Farinelli, Eugene, fracz, Franck Dernoncourt, Gabriel Pires, Gourav Garg , H. Pauwelyn, Igor Raush, jengeb, Jeroen, John F., Léo Martin, Lotus91, LucyMarieJ, M. Junaid Salaat, Maaz.Musa, Matt, Mikko Viitala, Mistalis, Nemanja Trifunovic, Nhan, Nico, pathe.kiran, Patrick, Pushpendra, Richard Hamilton, Stepan Suvorov, Stephen Leppik, Sunil Lama, superluminary, Syed Priom, timbo, Ven, vincentvanjoe, Yasin Patel, Ze Rubeus, Артем Комаров
2	Angulaire MVC	Ashok choudhary, Gavishiddappa Gadagi, Jim
3	angularjs avec filtre de données, pagination, etc.	Paresh Maghodiya
4	AngularJS des pièges et des pièges	Alon Eitan, Cosmin Ababei, doctorsherlock, Faruk Yazıcı, ngLover, Phil
5	Chargement paresseux	Muli Yulzary
6	Comment fonctionne la liaison de données	Lucas L, Sasank Sunkavalli, theblindprophet
7	Composants	Alon Eitan, Artem K., badzilla, BarakD, Hubert Grzeskowiak, John F., Juri, M. Junaid Salaat, Mansouri, Pankaj Parkar, Ravi Singh, sgarcia.dev, Syed Priom, Yogesh Mangaj
8	Contrôleurs	Adam Harrison, Aeolingamenfel, Alon Eitan, badzilla, Bon Macalindong, Braiam, chatuur, DerekMT12, Dr. Cool, Florian, George Kagan, Grundy, Jared Hooper, Liron Ilayev, M. Junaid Salaat, Mark Cidade, Matthew Green, Mike, Nad Flores, Praveen Poonia, RamenChef, Sébastien Deprez, sgarcia.dev, thegreenpizza, timbo, Und3rTow, WMios
9	Contrôleurs avec ES6	Bouraoui KACEM
10	Décorateurs	Mikko Viitala

11	Demande $ http	CENT1PEDE, jaredsk, Liron Ilayev
12	Des filtres	Aeolingamenfel, developer033, Ed Hinchliffe, fracz, gustavohenke, Matthew Green, Nico
13	Des promesses angulaires avec le service $ q	Alon Eitan, caiocpricci2, ganqqwerty, georgeawg, John F., Muli Yulzary, Praveen Poonia, Richard Hamilton, Rohit Jindal, svarog
14	digestion en boucle	Alon Eitan, chris, MoLow, prit4fun
15	directive classe ng	Dr. Cool
16	Directives intégrées	Adam Harrison, Alon Eitan, Aron, AWolf, Ayan, Bon Macalindong, CENT1PEDE, Devid Farinelli, DillonChanis, Divya Jain, Dr. Cool, Eric Siebeneich, George Kagan, Grinn, gustavohenke, IncrediApp, kelvinelove, Krupesh Kotecha, Liron Ilayev, m.e.conroy, Maciej Gurban, Mansouri, Mikko Viitala, Mistalis, Mitul, MoLow, Naga2Raja, ngLover, Nishant123, Piet, redunderthebed, Richard Hamilton, svarog, tanmay, theblindprophet, timbo, Tomislav Stankovic, vincentvanjoe, Vishal Singh
17	Directives sur mesure	Alon Eitan, br3w5, casraf, Cody Stott, Daniel, Everettss, Filipe Amaral, Gaara, Gavishiddappa Gadagi, Jinw, jkris, mnoronha, Pushpendra, Rahul Bhooteshwar, Sajal, sgarcia.dev, Stephan, theblindprophet, TheChetan, Yuri Blanc
18	Directives utilisant ngModelController	Nikos Paraskevopoulos
19	Distinguer Service vs Usine	Deepak Bansal
20	Événements	CodeWarrior, Nguyen Tran, Rohit Jindal, RyanDawkins, sgarcia.dev, shaN, Shashank Vivek
21	Filtres personnalisés	doodhwala, Pat, Sylvain
22	Filtres personnalisés avec ES6	Bouraoui KACEM
23	Fonctions d'assistance intégrées	MoLow, Pranav C Balan, svarog
24	Fournisseurs	Mikko Viitala
25	Impression	ziaulain

26	Injection de dépendance	Andrea, badzilla, Gavishiddappa Gadagi, George Kagan, MoLow, Omri Aharon
27	Intercepteur HTTP	G Akshay, Istvan Reiter, MeanMan, Mistalis, mnoronha
28	Le débogage	Aman, AWolf, Vinay K
29	Le moi ou cette variable dans un contrôleur	It-Z, Jim
30	Les constantes	Sylvain
31	Migration vers Angular 2+	ShinDarth
32	Modules	Alon Eitan, Ankit, badzilla, Bon Macalindong, Matthew Green, Nad Flores, ojus kulkarni, sgarcia.dev, thegreenpizza
33	ng-repeat	Divya Jain, Jim, Sender, zucker
34	ng-style	Divya Jain, Jim
35	ng-view	Aayushi Jain, Manikandan Velayutham, Umesh Shende
36	Options de liaisons AngularJS (`=`, `@`, `&` etc.)	Alon Eitan, Lucas L, Makarov Sergey, Nico, zucker
37	Partage de données	elliot-j, Grundy, Lex, Mikko Viitala, Mistalis, Nix, prit4fun, Rohit Jindal, sgarcia.dev, Sunil Lama
38	Portées angulaires	Abhishek Maurya, elliot-j, Eugene, jaredsk, Liron Ilayev, MoLow, Prateek Gupta, RamenChef, ryansstack, Tony
39	Préparez-vous à la production - Grunt	JanisP
40	Prestations de service	Abdellah Alaoui, Alvaro Vazquez, AnonDCX, DotBot, elliot-j, Flash, Gavishiddappa Gadagi, Hubert Grzeskowiak, Lex, Nishant123
41	Profilage de performance	Deepak Bansal
42	Profilage et performance	Ajeet Lakhani, Alon Eitan, Andrew Piliser, Anfelipe, Anirudha, Ashwin Ramaswami, atul mishra, Braiam, bwoebi, chris, Dania, Daniel Molin, daniellmb, Deepak Bansal, Divya Jain, DotBot, Dr. Cool, Durgpal Singh, fracz, Gabriel Pires, George Kagan, Grundy, JanisP, Jared Hooper, jhampton, John Slegers, jusopi,

		M22an, Matthew Green, Mistalis, Mudassir Ali, Nhan, Psaniko, Richard Hamilton, RyanDawkins, sgarcia.dev, theblindprophet, user3632710, vincentvanjoe, Yasin Patel, Ze Rubeus
43	Projet angulaire - Structure des répertoires	jitender, Liron Ilayev
44	Routage à l'aide de ngRoute	Alon Eitan, Alvaro Vazquez, camabeh, DotBot, sgarcia.dev, svarog
45	SignalR avec AngularJs	Maher
46	Stockage de session	Rohit Jindal
47	Tâches Grunt	Mikko Viitala
48	Tests unitaires	daniellmb, elliot-j, fracz, Gabriel Pires, Nico, ronapelbaum
49	ui-routeur	George Kagan, H.T, Michael P. Bazos, Ryan Hamley, sgarcia.dev
50	Utilisation de directives intégrées	Gourav Garg
51	Utiliser AngularJS avec TypeScript	Parv Sharma, Rohit Jindal
52	Validation de formulaire	Alon Eitan, fantarama, garyx, Mikko Viitala, Richard Hamilton, Rohit Jindal, shane, svarog, timbo