Article
Sergei Sarkisian · Juil 5, 2022 10m de lecture

Angular en Profondeur. Conseils généraux

Avant de commencer à aborder des sujets intermédiaires et avancés, je voudrais résumer quelques points plus généraux. Ils sont subjectifs, bien sûr, et je serai heureux d'en discuter si vous avez une autre opinion ou de meilleurs arguments pour l'un d'entre eux.

La liste n'est pas exhaustive et c'est voulu, car je couvrirai certains sujets dans de futurs articles.

Conseil 1. Suivez le guide de style officiel

Angular est assez strict en termes de limitation de l'architecture possible d'une application, mais il y a encore de nombreux endroits qui vous permettent de faire les choses à votre façon. L'imagination des développeurs est sans limite, mais elle rend parfois le travail difficile pour ceux qui travaillent sur le projet avec vous ou après vous.

L'équipe Angular fait un bon travail en maintenant l'architecture Angular elle-même et ses bibliothèques, donc ils savent certainement comment créer une base de code stable et supportable.

Je recommande de suivre leur guide de style officiel et de ne s'en écarter que si les choses ne fonctionnent pas de cette façon. Cela rendra les choses plus faciles quand vous arriverez à un nouveau projet ou si quelqu'un arrive dans votre projet.

Rappelez-vous, un code et une architecture d'application supportables, stables et faciles à comprendre sont plus importants que des solutions intelligentes mais cryptiques que personne ne pourra rattraper (peut-être même vous à l'avenir).

Guide de style officiel d'Angular : https://angular.io/guide/styleguide

Conseil 2. Envisagez d'acheter le livre Angular Ninja

Vous pouvez penser que c'est de la pub, mais voilà : c'est un très bon livre avec tous les principaux concepts d'Angular et le prix est à votre convenance. Vous pouvez également choisir combien d'argent ira aux auteurs et combien ira à une œuvre de charité.

L'un des auteurs de ce livre est membre d'Angular teem, donc c'est certainement la source d'information la plus fiable sur Angular après la documentation officielle. Vous pouvez voir les chapitres du livre sur la page du livre et lire des exemples de chapitres pour décider si le livre en vaut la peine.

De plus, le livre est mis à jour avec la sortie d'une nouvelle version d'Angular et vous obtenez toutes les mises à jour du livre gratuitement.

Le blog Ninja Squad lui-même est une très bonne source d'informations sur Angular, avec des articles et des nouvelles sur les nouvelles versions, les meilleures pratiques, les fonctionnalités expérimentales et plus encore.

Le livre Angular Ninja : https://books.ninja-squad.com/angular

Conseil 3. Lisez la documentation officielle

Avant de vous plonger dans l'écriture du code de votre application, il est bon de lire la documentation et les guides

officiels, surtout si vous démarrez votre projet sur une version d'Angular que vous n'avez jamais utilisée auparavant. Les choses sont dépréciées tout le temps, les tutoriels et les guides sur Internet peuvent être dépassés et vous pouvez finir par augmenter votre dette technique au lieu d'utiliser les nouvelles meilleures pratiques et fonctionnalités.

C'est aussi une bonne habitude de vérifier pour quelle version le guide a été écrit. S'il s'agit d'une version antérieure de plus de deux ans à celle que vous utilisez, il est préférable de vérifier si les choses ont changé depuis.

Documentation officielle : https://angular.io

Conseil 4. Envisagez d'utiliser Angular CDK même si vous n'utilisez pas Angular Material.

Je l'aborderai plus en détail dans de futurs articles, mais je sais que de nombreux développeurs Angular ne connaissent même pas Angular CDK.

Angular CDK est une bibliothèque de directives et de classes de base utiles qui peuvent vous aider à développer de meilleures applications. Par exemple, elle contient des éléments tels que FocusTrap, Drag & Drop, VirtualScroll et d'autres qui peuvent être facilement ajoutés à vos composants.

Angular CDK : https://material.angular.io/cdk/categories

Conseil 5. Fixez vos dépendances dans package.json

Il n'est pas particulièrement lié à Angular et il peut être important dans tout projet. Lorsque vous faites npm install --save <quelque chose> , il sera ajouté à votre package.json avec ^ ou ~au début de la version du package. Cela signifie que votre projet sera capable d'utiliser n'importe quelle version mineure/patch de la même version majeure de la dépendance. Cela se passera mal à l'avenir avec une probabilité de presque 100%. J'ai été confronté à ce problème dans différents projets à de nombreuses reprises. Le temps passe et une nouvelle version mineure d'une dépendance sort et votre application ne se construit plus. Parce qu'un auteur de cette dépendance a mis à jour ses dépendances en version mineure (ou même en patch) et maintenant elles sont en conflit avec votre build. Ou peut-être qu'il y a un nouveau bug dans la nouvelle version mineure de votre dépendance et vous n'avez aucun problème dans votre code (parce que vous avez installé vos dépendances avant la nouvelle version), mais toute autre personne essayant d'installer et de construire votre projet à partir du référentiel sera confrontée au bug dont vous n'avez même pas connaissance (et vous ne serez pas en mesure de le reproduire sur votre machine).

package-lock.json existe pour résoudre ce problème et pour aider les développeurs à avoir le même ensemble de dépendances à travers le projet. Idéalement, il devrait être livré au dépôt, mais de nombreux développeurs décident d'ajouter ce fichier à .gitignore. Et s'il n'est plus là, vous pouvez vous retrouver avec les problèmes ci-dessus. Il est donc préférable de ne pas faire aveuglément confiance à la résolution de vos dépendances et de la fixer sur la version spécifique, de la scanner régulièrement pour détecter les vulnérabilités (avec npm audit), puis de la mettre à jour et de la tester manuellement.

Conseil 6. Essayez de mettre votre application à niveau vers la nouvelle version d'Angular dès que vous le pouvez.

Angular évolue en permanence et de nouvelles versions sortent tous les 6 mois environ. En gardant votre application à jour, vous pourrez utiliser toutes les nouvelles fonctionnalités, y compris des constructions plus rapides et d'autres optimisations. En outre, la mise à niveau vers la prochaine version majeure d'Angular ne devrait pas poser de gros problèmes dans ce cas. Mais si vous avez 5 versions de retard et que votre application est de taille moyenne ou grande, le processus de mise à jour vers la dernière version peut être difficile car Angular n'a pas de schéma pour mettre à niveau les applications sautant des versions intermédiaires d'Angular. Vous devrez mettre à jour toutes les versions intermédiaires une par une, en vérifiant que l'application fonctionne sur chaque version. La mise à jour directe sans schéma Angular CLI est possible, mais peut aussi être délicate. Je vous

suggère donc de garder votre application fraîche et à jour.

Conseil 7. Essayez de réduire votre liste de dépendances

Il est facile de prendre l'habitude d'introduire de nouvelles dépendances dans votre application lorsque vous avez besoin de nouvelles fonctionnalités. Mais avec chaque dépendance, la complexité de votre application augmente et le risque d'une soudaine avalanche de dettes techniques s'accroît.

Si vous décidez d'ajouter une nouvelle dépendance dans votre application, pensez à ceci :

- La dépendance est-elle bien supportée ?
- Combien de personnes l'utilisent ?
- Quelle est la taille de l'équipe de développement ?
- A quelle vitesse ils ferment les problèmes sur GitHub ?
- La documentation de la dépendance est-elle bien rédigée ?
- A quelle vitesse est-elle mise à jour après la sortie d'une nouvelle version d'Angular ?
- Quel est l'impact de cette dépendance sur les performances et la taille du bundle de votre application ?
- Quelle sera la difficulté de remplacer cette dépendance si elle est abandonnée ou dépréciée à l'avenir ?

Si la réponse à certaines de ces questions est négative, envisagez de vous en débarrasser ou au moins de la remplacer par quelque chose de plus mature et de mieux supporté.

Conseil 8. N'utilisez pas <any> comme type "temporaire".

Encore une fois, il est facile de commencer à utiliser n'importe quel type au fur et à mesure que vous écrivez votre logique d'entreprise, parce que l'écriture de types appropriés pourrait prendre du temps, et vous devez terminer votre tâche dans ce sprint. Les types peuvent être ajoutés plus tard, bien sûr. Mais c'est une pente glissante pour augmenter la dette technique.

L'architecture de votre application et les types doivent être définis avant d'écrire toute logique métier. Vous devez clairement comprendre quels objets vous aurez et où ils seront utilisés. La spécification avant le code, n'est-ce pas ? (Tom Demarco a écrit un livre à ce sujet avant qu'il ne devienne courant : https://www.amazon.com/Deadline-Novel-About-Project-Management/dp/0932633390).

Si vous écrivez le code sans types prédéfinis, vous risquez de vous retrouver avec une architecture d'application moins bonne et des fonctions qui utilisent des objets très similaires mais différents. Vous devrez donc soit créer des types différents pour chaque fonction (ce qui aggravera encore les choses), soit passer votre temps à écrire des spécifications, des types **ET** du refactoring, ce qui est une perte de temps par rapport à ce qui aurait été fait auparavant.

Conseil 9. Prenez le temps de comprendre le processus de construction

Angular fait un excellent travail pour faciliter le travail du développeur en ce qui concerne la construction du projet. Mais les options par défaut ne sont pas toujours les meilleures dans tous les cas.

Passez votre temps à comprendre comment le processus de construction fonctionne dans Angular, quelles sont les différences entre les constructions de développement et de production, quelles options Angular a pour les constructions (comme les optimisations, les cartes de source, le regroupement et plus).

Conseil 10. Examinez le contenu de votre liasse.

Toutes les bibliothèques ne fournissent pas de tree-shaking et nous ne faisons pas toujours les importations de la bonne manière, il y a donc toujours une chance que quelque chose de redondant soit empaqueté avec notre application.

C'est donc une bonne habitude d'examiner le contenu de votre bundle de temps en temps.

Il y a de bons articles décrivant le processus en utilisant webpack-bundle-analyzer , donc je ne le couvrirai pas ici, mais voici un lien pour l'un d'entre eux : https://www.digitalocean.com/community/tutorials/angular-angular-webpack-bundle-analyzer

Je couvrirai ce sujet plus en détail plus tard dans la série.

Conseil 11. Utilisez la propriété providedIn des services au lieu d'importer le service dans le module.

De nombreux exemples de code Angular sur Internet utilisent l'importation des services dans les modules. Mais ce n'est pas la façon préférée de déclarer les services depuis Angular 6 ou 7. Nous devrions utiliser la propriété providedIn du décorateur @Injectable pour permettre le tree-shaking et un meilleur bundle de nos applications. Angular est suffisamment intelligent pour comprendre dans quel bundle le service doit être inclus, quand il doit être initialisé et combien d'instances du service doivent être créées.

Il y a trois valeurs que providedIn accepte. La plupart du temps, root est suffisant, mais il en existe deux autres :

- root : le service sera singleton dans la portée de l'application
- any : une instance du service sera créée pour tous les modules chargés avec empressement, avec une instance différente créée pour chaque module paresseux.
- platform : le service sera singleton pour toutes les applications tournant sur la même page.

Conseil 12. N'oubliez pas les principales règles d'exécution

- Utilisez trackBy pour les collections afin de réduire les opérations de redécoupage et l'exécution de JavaScript.
- Utilisez la stratégie de détection des changements onPush partout où vous le pouvez (je l'aborderai dans l'article dédié).
- Effectuez les calculs lourds en dehors de ngZone
- Utilisez des modèles d'accélération et de ralentissement avec vos événements pour éviter les appels inutiles au serveur et l'inondation d'événements.
- Utilisez le défilement virtuel pour afficher de grands ensembles de données.
- Utiliser des pure pipes pour la transformation des données dans vos templates
- Utilisez la compilation AoT
- Chargez paresseusement les parties de votre application qui ne sont pas nécessaires à son démarrage.
- Évitez les calculs et les appels de fonction dans vos modèles.

Merci de votre lecture ! J'espère que certains de ces conseils vous ont été utiles. Si vous avez des commentaires et des remarques, s'il vous plaît, faites-le moi savoir dans les commentaires, je serai heureux de discuter .
A bientôt !

#Angular #Angular2 #Développement de l'interface utilisateur #FrontEnd #Instructions de codage #Autre