記事
Toshihiko Minamoto · 2022年11月29日 11m read

Angular: 一般的なヒント

中・上級トピックに進む前に、より一般的なポイントについてまとめておきたいと思います。 これはもちろん主観的な内容であるため、他の意見やさらに良い根拠をお持ちであれば、それについて喜んでお聞きします。

このリストは包括的ではありません。一部のトピックは今後の記事で対応するつもりなので、意図的にそうしています。

ヒント 1. 公式スタイルガイドに従う

Angular は、アプリケーションに使用可能なアーキテクチャを制限するという点で非常に厳格ですが、それでも独自の方法で行えることはたくさんあります。 開発者の想像力は無限ではありますが、そのために、あなたと、またはあなたを引き継いでプロジェクトに携わる人の作業が困難になってしまうことがあります。

Angular チームは、Angular アーキテクチャ自体とそのライブラリをうまく管理しているため、安定した対応可能なコードベースを作成する方法をよく理解しています。

したがって、公式スタイルガイドに従い、そのとおりに動作しない場合にのみ他の方法を取るようにすることをお勧めします。 こうすることで、新しいプロジェクトに参加する際や、他の人がプロジェクトに参加する際に、事がより簡単に進められるようになります。

コードとアプリアーキテクチャが対応可能で安定しており、理解しやすい方が、賢明でありながらも暗号的なソリューションを作るよりも重要です。誰も追いつけなくなってしまいます（開発者自身が後に追いつけなくなる可能性もあります）。

Angular 公式スタイルガイド: https://angular.io/guide/styleguide

ヒント 2. Angular の Ninja 本を購入する

宣伝と思われるかもしれませんが、つまりこういうことです。Angular
のすべての主要概念がカバーされた非常に良い本で、価格はあなたが決められます。
また、価格のどれくらいが作者に支払われ、どれくらいが慈善活動に募金されるかを決めることもできます。

この本の作者の 1 人は Angular チームのメンバーであるため、公式ドキュメントの次に最も信頼できる Angular の情報源であることは間違いありません。 本の章構成は、本のページで確認できるようになっているため、サンプルページを読んでから購入する価値があるかを決めることができます。

さらに、この本は Angular
の新しいバージョンのリリースで更新されており、本のすべての更新内容を無料で得られます。

Ninja Squad ブログ自体も Angular に関する非常に優れた情報源で、新しいバージョンに関する記事や最新情報、ベストプラクティス、実験的機能などが掲載されています。

Angular の Ninja 本: https://books.ninja-squad.com/angular

ヒント 3. 公式ドキュメントを読む

アプリのコードを書き始める前、特に、以前に使用したことのないバージョンの Angular でプロジェクトに取り組む場合には、公式ドキュメントとガイドに目を通すことをお勧めします。 廃止機能は常に増えており、インターネット上のチュートリアルとガイドが古くなっている可能性があります。そのため、新しいベストプラクティスや機能を使用する代わりに、技術的負債が増え続けてしまうことになりかねません。

ガイドがどのバージョンを対象にして書かれているのかを確認するのも良い習慣です。 使用しているのが 2 つ以上前のバージョンであれば、それが書かれた当時から変更されたことがないかを確認することをお勧めします。

公式ドキュメント: https://angular.io

ヒント 4. Angular Material を使用しなくても、Angular CDK の使用を検討する

これについては今後の記事に取り上げるのが良いと思いますが、私は、多くの Angular 開発者は Angular CDK の存在すらも知らないことを知っています。

Angular CDK は、より優れたアプリケーションの開発を支援できる便利なディレクティブと基底クラスが集められたライブラリです。 たとえば、FocusTrap、Drag & Drop、VirtualScroll などが含まれており、コンポーネントに簡単に追加することができます。

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

ヒント 5. package.json 内の依存関係を修正する

特に Angular に関連することではありませんし、どんなプロジェクトでも重要なことかもしれません。 npm install --save <something> を実行すると、パッケージバージョンの先頭に ^ または ~付きで package.json に追加されます。 つまり、プロジェクトはその依存関係と同じメジャーバージョンのあらゆるマイナー/パッチバージョンを使用できるということになります。 この場合、将来的にほぼ 100% の確率で問題となります。
私はさまざまなプロジェクトにおいて何度もこの問題に直面しました。
時間が経過し、ある依存関係の新しいマイナーバージョンが作成されれば、アプリはビルドできなくなります。
この依存関係の作成者がそれをマイナー（またはパッチ）バージョンで更新したために、ビルドと競合するようになってしまうこともあるでしょう。 また、依存関係の新しいマイナーバージョンで新しいバグが見つかることもあります。自分のコードでは問題がなくても（新しいバージョンがリリースされる前に依存関係をインストールしたため）、リポジトリから自分のプロジェクトをインストールしてビルドしようとしている人たちが、自分が気付いていないバグに直面することもあります（また、自分のマシンでそのバグを再現することもできません）。

package-lock.json は、この問題を解決し、複数の開発者がプロジェクト全体で同じ依存関係セットを使用できるようにするために存在します。 リポジトリにコミットすることが理想的ですが、このファイルを .gitignore に追加している開発者がたくさんいます。 それでは意味がなく、結局、上記と同じ問題が起きてしまいます。 そのため、依存関係の解決を盲目的に信頼して特定のバージョンに固定するのではなく、定期的に脆弱性をスキャンして（npm audit を使用）、手動で更新してテストすることをお勧めします。

ヒント 6. できるだけ早期にアプリを新しい Angular バージョンにアップグレードするようにする

Angular は絶えず進化しており、約半年ごとに新しいバージョンがリリースされています。 アプリを最新状態に維持することで、ビルドの高速化やその他の最適化を含むすべての新機能を使用することができます。
この場合には、Angular の次期メジャーバージョンへのアップグレードは大した問題ではないでしょう。
ただし、Angular には、Angular の中間バージョンをスキップしてアプリをアップグレードするためのスケマティックがないため、5 バージョン前の Angular を使用しており、アプリケーションの規模が中〜大である場合には、最新バージョンへの更新プロセスが困難になる可能性があります。
各バージョンでアプリケーションが動作することを確認しながら、すべての中間バージョンを 1

つずつアップグレードすることが必要になるでしょう。 Angular CLI のスケマティックを使わずに直接アップデートすることは可能ですが、コツが必要となるため、常にアプリを最新状態に維持しておくことをお勧めします。

ヒント 7. 依存関係リストをできるだけ短くする

新機能が必要となるたびに新しい依存関係をアプリに採り入れるという習慣に陥りやすいものですが、
依存関係ごとに、アプリの複雑性が増し、技術的負債が突然雪崩れてしまうリスクが高まります。

アプリに新しい依存関係を追加すると決めた場合は、以下のことを検討してください。

- どれほど十分に依存関係がサポートされているか？
- どれくらいの人が使用しているか？
- 開発チームはどれくらいの規模か？
- どれくらい迅速に GitHub 課題をクローズしているか？
- どれくらい十分に依存関係のドキュメントが書かれているか？
- Angular の新しいメジャーバージョンがリリースされてからどれくらい迅速に更新されているか？
- この依存関係によるアプリケーションのパフォーマンスとバンドルサイズへのインパクトはどれくらいか？
- 将来的に放置または廃止された場合に、この依存関係を入れ替えるのがどれくらい困難になるか？

この質問のいくつかに否定的な回答がある場合、その依存関係を排除するか、少なくともっと成熟して十分にサポートされているものに交換することを検討してください。

ヒント 8. 「暫定的な」型として <any> を使用しない

繰り返しますが、適切な型を書くには時間がかかるものであり、このスプリントでタスクを完了させる必要があるため、ビジネスロジックを書く際に簡単に any を使いがちです。 もちろん、型は後で追加できますが、
技術的負債がすぐに積もる危険があります。

アプリと型のアーキテクチャは、ビジネスロジックを書く前に定義しておく必要があります。
どのようなオブジェクトがあり、どこで使用されるかを明確に理解しておくかなければなりません。
コードの前に仕様書、ですよね？（Tom Demarco
は、この考えが主流になる前にこれについての本を書いています: https://www.amazon.com/Deadline-Novel-About-Project-Management/dp/0932633390）

事前に定義された型を使わずにコードを書いている場合、非常に似ているようで異なるオブジェクトを使用する、質の悪いアプリアーキテクチャと関数が出来上がってしまう可能性があります。
そのため、関数ごと
に異なる型を作成する（余計に悪くなっ
てしまいます）か、仕様書、型、**および**
リファクタリングを書くことに時間を費やす必要があるでしょう。後者は、前もってやっていれば時間の無駄になりません。

ヒント 9. ビルドプロセスを理解するのに時間をかける

Angular は、プロジェクトの構築に関する開発者の作業をうまく容易にしていますが、
デフォルトのオプションは、すべての場合に必ずしも最適ではありません。

Angular のビルドプロセスの仕組み、開発ビルドと本番ビルドの違い、Angular で提供されているビルドのオプション（最適化、ソースマップ、バンドル化など）を理解する時間を取りましょう。

ヒント 10. バンドル内のものを調べる

すべてのライブラリがツリーシェイキングを提供しているわけでも、必ずしもすぐにインポートするわけでもありません。そのため、冗長するものがアプリにバンドルされる可能性が必ずあります。

したがって、たまにバンドルの中身を調べる習慣をもつことをお勧めします。

webpack-bundle-analyzer
を使ったプロセスをうまく説明した記事がいくつかあるため、ここでは説明しませんが、その中の 1
つのリンクをご覧ください:
https://www.digitalocean.com/community/tutorials/angular-angular-webpack-bundle-analyzer

このトピックについては、連載の後の方でより詳しく説明します。

ヒント 11.
モジュールにサービスをインポートする代わりに、サービスの providedIn プロパティを使用する

インターネットに公開されている Angular
コードサンプルの多くは、モジュールへのサービスのインポートを使用しています。 ただし、Angular 6 または 7 以降では、これはサービスを宣言する方法として推奨されていません。
ツリーシェイキングを有効にし、アプリケーションのバンドル化を改善するには、@Injectable
デコレータのプロパティ providedIn を使用する必要があります。 Angular は、どのバンドルにサービスを含める必要があるのか、いつ初期化すべきなのか、サービスのインスタンスをいくつ作成する必要があるのかを理解できるほどスマートなフレームワークです。

providedIn が受け入れる値には 3 つあります。 ほとんどの場合は root で十分ですが、そのほかに 2 つあります。

- root: サービスアプリケーションの範囲でシングルトンになります。
- any: すべての Eager ロードされたモジュールにサービスのインスタンスが 1
  つ作成され、遅延モジュールごとに別のインスタンスが作成されます。
- platform: 同じページで実行するすべてのアプリケーションに対し、サービスはシングルトンになります。

ヒント 12. 主要なパフォーマンスルールを忘れないこと

- 再描画操作と JavaScript 実行を減らすには、コレクションに trackBy を使用する
- 使用できるすべての場所に onPush
  変更検出ストラテジーを使用する（これについては、専用の記事で説明します）
- 高い処理能力を必要とする計算は、ngZone の外で実行する
- イベントにスロットル・デバウンスパターンを使用して、不要なサーバー呼び出しとイベントの大量送信を防止する
- ビッグデータセットの表示には仮想スクロールを使用する
- テンプレートにデータ変換用の純粋なパイプを使用する
- AoT コンパイルを使用する
- アプリケーションの起動には必要ないアプリの部分を遅延読み込みする
- テンプレートでは計算と条件付き関数呼び出しを使わない（関数呼び出しはイベントのみに使用します）

#
お読みいただきありがとうございました！ いくつかのヒントがお役に立ちますように。
コメントやご意見がございましたら、コメント欄でお知らせください。喜んでお伺いします

それではまた！

#Angular #Angular2 #UI 開発 #コーディングのガイドライン #フロントエンド #その他

ソースURL:

https://jp.community.intersystems.com/post/angular-%E4%B8%80%E8%88%AC%E7%9A%84%E3%81%AA%E3%83%92%E3%83%B3%E3%83%88