気軽な Node.js バックエンド開発には TypeORM がちょうどいい

2019.08.02 #kng7 / LINE株式会社 UIT室 花谷拓磨(@potato4d)

自己紹介

- 花谷拓磨(@potato4d)
- Working at...
  - LINE株式会社 UIT室 / Developer Relations室
- 今日は東京から来ました
  - 昨日は Serverless Meetup Osaka #5 で
    フル Firebase Functions アプリを Firestore へと移行した記録の話をしてきました

Agenda

- 結局俺たちは O/R Mapper に何を求めているのか
- TypeScript ベースで「ちょうどいい」 TypeORM の紹介
- TypeORM の活用シチュエーション

みなさん ORM は好きですか？

何が便利で何が不便ですか？

実はみなさんの言う問題点って
「ActiveRecord かどうかが排他なこと」
に起因していませんか？

結局俺たちは O/R Mapper に何を求めているのか

1. ORM を使うなら ActiveRecord で雑に作りたい

- 開発初期はデータ構造やリレーションにドラスティックな変更が入りやすい
  - できれば高速に対処したい
- 初期の CRUD くらいは爆速で作ってしまいたい
  - プロトタイピングの速度は落としたくない

結局俺たちは O/R Mapper に何を求めているのか

2. ActiveRecord で作ったものをメンテし続けたくない

- とはいえ継続的に Model = DB みたいな状態で運用したくない
  - この辺りが年季の入った Rails が嫌われやすい点な気がする
- 段階的に Repository パターンを適用できるような世界観が欲しい
  - DAO を DAO として用意した上でより抽象度の高い実装に落とし込みたい
  - はじめからやると早すぎる最適化感があるので折を見て調整したい

結局俺たちは O/R Mapper に何を求めているのか

3. 可能な限り力の入れ加減をコントロールしたい

- ゼロベースで何かを作っていく段階
  - 多少密結合でも生産性を高めたい
  - アプリケーションコードから柔軟に DB の設計に手を入れられると良い
- 継続的にメンテナンスしていく段階
  - アプリケーション側のエンティティのスキーマ定義もかっちりしたい
  - ドメイン上の概念とデータベースアクセスはある程度疎結合にしたい
  - Node.js の ORM はマイグレーション弱めだけどしっかり管理したい
  - etc..

そんなものなくない？

TYPEORM

TypeScript ベースで「ちょうどいい」 TypeORM の紹介

TypeORM の特徴

- TypeScript 向けの ORM (JavaScript 対応)
  - MySQL, PostgreSQL, SQLite から MongoDB まで対応
- GitHub Star も 15k+ と注目度の高い ORM
  - https://github.com/typeorm/typeorm
  - ちなみに NPM の weekly downloads は 100k+

TypeORM の特徴

1. デコレータベースの簡潔かつ TypeSafe なエンティティ定義
2. 柔軟かつ便利な Migration
3. ActiveRecord と Repository にどちらも対応

TypeORM のうれしいところ

- デコレータベースの簡潔かつ TypeSafe なエンティティ定義
  - `@Entity` から始まってクラスに対してデコレータで情報を付与していくだけ
  - 勿論 TypeScript 自体の型の定義と相互作用があり、メタデータによって型定義ベースでの動作指定が可能
  - `@Column` のオプションも全て型がついている
    - それもそのはず TypeORM は全て TypeScript で記述されている

TypeORM のうれしいところ

```
import {Entity, PrimaryGeneratedColumn, Column, BaseEntity} from "typeorm";

@Entity()
export class User extends BaseEntity {

    @PrimaryGeneratedColumn()
    id: number;

    @Column()
    firstName: string;

    @Column()
    lastName: string;

    @Column()
    age: number;

}
```

TypeORM のうれしいところ

- 柔軟かつ便利な Migration
  - 自動マイグレーションと手動マイグレーションの 2 つのモードが有る
  - 自動マイグレーションは開発時に便利
    - `@Entity` の定義と DB の差分をみて自動でマイグレーション
    - 設計と実装にブレがあった場合に実装側で試行錯誤しやすい
    - 初期の開発において悩みどころが少ない、多少古い情報を共有されても実行時でなんとかなる
  - 手動マイグレーションは完成品のコードとして便利
    - TypeORM の CLI にはマイグレーションのコマンドが一通り揃っている
    - マージする時は必ずマイグレーションファイルに落とし込んで適用みたいに柔軟な形に落とし込みやすい

Use manual migration

```
$ typeorm migration:generate -n PostRefactoring # Create migration file
$ typeorm migration:run # Execute migration
$ typeorm migration:revert # Revert migration
```

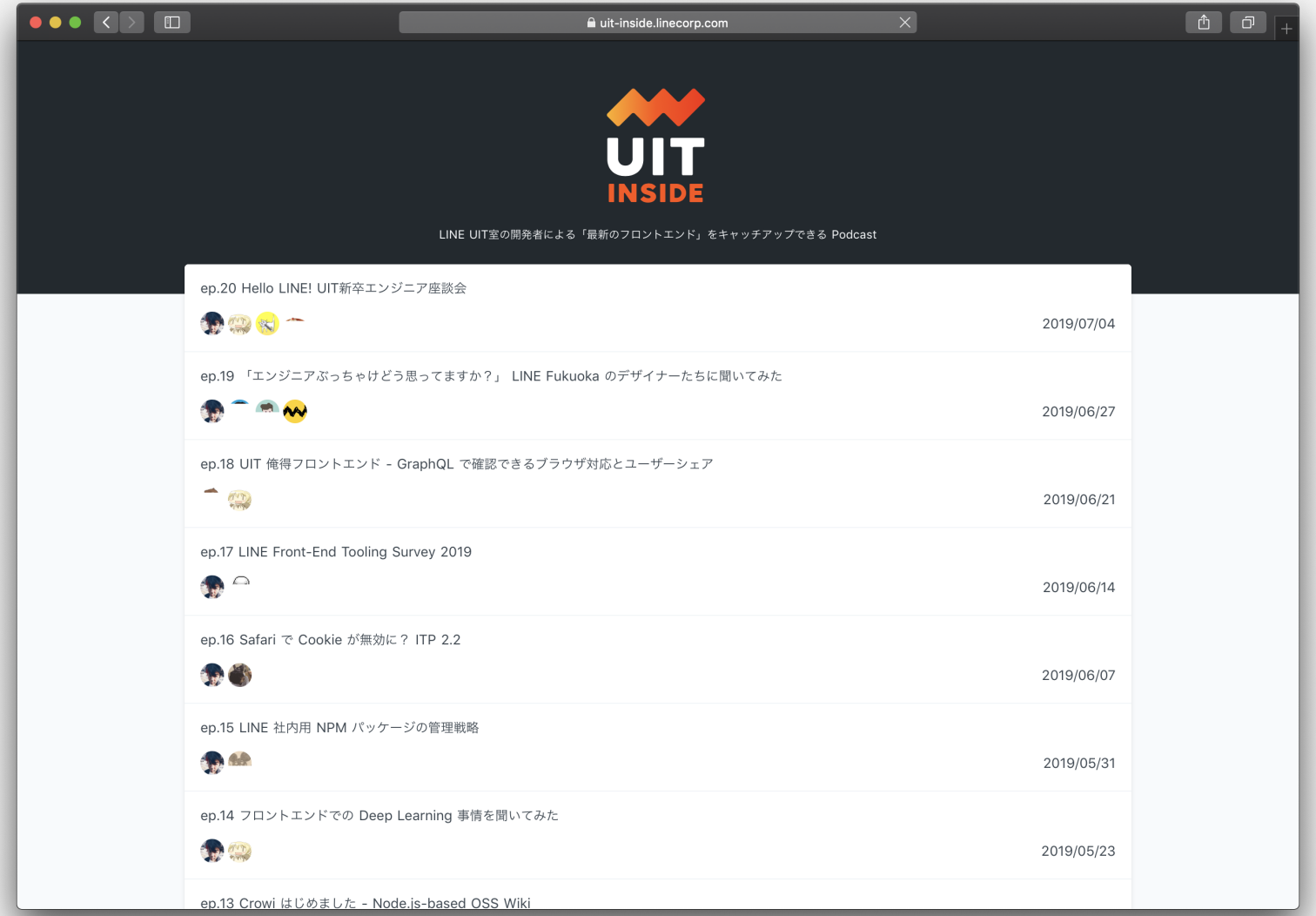
Use automatic migration

```
$ yarn dev
yarn run v1.15.2
warning package.json: No license field
$ ts-node src/server.ts
Listen on http://0.0.0.0:8000
query: START TRANSACTION
query: SELECT DATABASE() AS `db_name`
query: SELECT * FROM `INFORMATION_SCHEMA`.`TABLES` WHERE (`TABLE_SCHEMA` = 'podcast' OR (`T
query: SELECT * FROM `INFORMATION_SCHEMA`.`COLUMNS` WHERE (`TABLE_SCHEMA` = 'podcast' OR (`
```

TypeORM のうれしいところ

- ActiveRecord と Repository にどちらも対応
  - 先程述べた「ActiveRecordメンテしたくない問題」を解消できる
  - 一方で早すぎる最適化が起こるわけでもなく、柔軟に対応しやすい

TypeORM の活用シチュエーション



3 月に立ち上げた Podcast サイトで使ってます

TypeORM の活用シチュエーション

- すみません！あんまりUIT室内では使ってません。
  - Node.js バックエンドの仕事がある現場だと重宝します

TypeORM の活用シチュエーション

- LINE UIT室の場合
  - Podcast UIT INSIDE の API サーバーで使っています
- UIT INSIDE の立ち上げ時の場合
  - とりあえずクリティカルではないのでサクッと立ち上げたい
    - とはいえ CMS を組みたいので静的サイトだとちょっと具合が悪い
  - Express + TypeORM で TypeScript な Node.js サーバーを運用
  - 初期リリースまでは高生産性の恩恵を存分に受ける
    - 公開前までは自動マイグレーションで開発
    - データモデルは極力 ActiveRecord で実装

TypeORM の活用シチュエーション

- 立ち上げ後機能改修を加えていく場合
  - ここからは非 LINE プロジェクトでの体験の話
  - 直で ActiveRecord を使わずとも Repository に切り替えて queryBuilder も使いつつ運用できることに気づいていく
    - 徐々に ActiveRecord による実装が辛くなっていくので移行していく
    - もちろん全部ができるわけではないが、割れ窓となりうるコードを排除しつつ前進
      - これまでの ORM ではできなかった戦略のとり方で嬉しい！

おわりに

おわりに

- TypeORM はプロダクトと共に成長していける手軽な ORM
  - Rapid Development のための土壌と堅牢な設計のための布石がどちらも揃っている
  - はじめから厳格に実装するもよし、あとから切り出すもよしな柔軟な仕様
  - 決して全ての機能が Simple とは言えないが、 Simple / Easy どちらにも寄せられる

おわりに

- ORM 好き派嫌い派みたいな概念は必要ない
  - 好き嫌いじゃなくているか要らないかで選ぶんだよ
  - ORM というざっくりしたものではなくて何が不満で何が便利化を言語化してみる

Thank you!