

TypeScriptで使いやすいOpenAPIの書き方

株式会社ドワンゴ 教育事業本部

上坂 直輝

2024.05.11 TSKaigi 2024 スポンサーLT

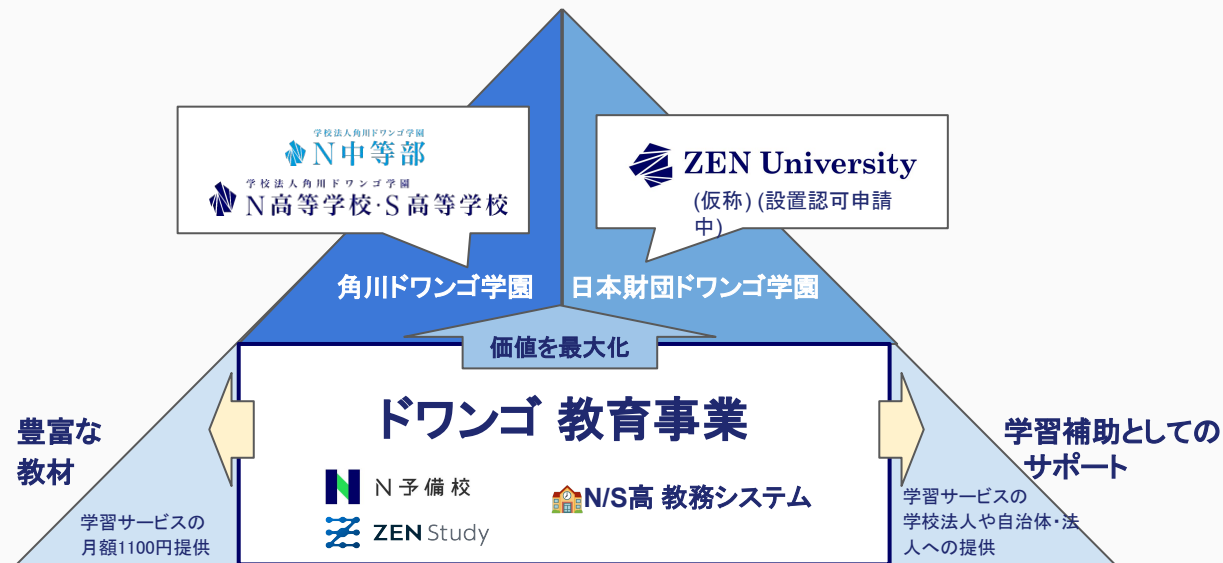
未来の「当たり前」の教育をつくる

- 学習プラットフォーム

N予備校 (ZEN Study)

- 学園教務システム

N/S高 教務システム



N予備校のバックエンド・マイクロサービス

N予備校のバックエンドでは、マイクロサービス構成を採用

当初

各マイクロサービスがほぼ Ruby on Rails によって実装



現在

新設のマイクロサービスを中心に適材適所の技術選定を実施
TypeScript (NestJS) 含めさまざまな言語・フレームワークを利用

N予備校の教材の特性と忠実な型定義の必要性

似た構造を持ちながら部分的に異なるたくさんの教材種別が存在します。
これらを安心して扱えるよう忠実な型定義の管理・運用が求められています。

共通部分	参考書	動画	テキスト付動画	生授業
・タイトル ・権限	共通部分＋ ・テキスト情報	共通部分＋ ・ビデオ情報	共通部分＋ ・ビデオ情報 ・テキスト情報	共通部分＋ ・配信用情報

...

※ 図は一例で、実際の構造とは異なります

マイクロサービス構成における機械可読な API 定義の魅力

マイクロサービス間通信に REST API と gRPC を採用

OpenAPI や proto をアプリケーションとは別のリポジトリで管理・運用

- GitHub の PRベースで具体的な API の検討などが可能
- サーバ・クライアントの同時開発が容易になる
 - OpenAPI を手書きするスキーマセントリックを採用

今回の前提条件

TypeScript 対応の型定義・クライアント生成

OpenAPI Generator の typescript-axios を利用

用途

- クライアントの生成
- **リクエスト・応答の型定義の生成**
 - 今回は型定義の方法について取り上げます

記法

- YAML を用いる

1/4. object のプロパティを切り出して interface 名を制御する

```
Item:
  type: object
  properties:
    item_id:
      type: integer
    owner:
      type: object
      required:
        - user_id
        - name
      properties:
        user_id:
          type: integer
        name:
          type: string
    author:
      $ref: '#/User'
```

```
User:
  type: object
  required:
    - user_id
    - name
  properties:
    user_id:
      type: integer
    name:
      type: string
```

1/4. object のプロパティを切り出して interface 名を制御する

```
Item:
  type: object
  properties:
    item_id:
      type: integer
    owner:
      type: object
      required:
        - user_id
        - name
      properties:
        user_id:
          type: integer
        name:
          type: string
    author:
      $ref: '#/User'
```

```
User:
  type: object
  required:
    - user_id
    - name
  properties:
    user_id:
      type: integer
    name:
      type: string
  owner:
    type: object
    required:
      - user_id
      - name
    properties:
      user_id:
        type: integer
      name:
        type: string
```

```
/**
 *
 * @export
 * @interface ItemOwner
 */
export interface ItemOwner {
  /**
   *
   * @type {number}
   * @memberof ItemOwner
   */
  'user_id': number;
  /**
   *
   * @type {string}
   * @memberof ItemOwner
   */
  'name': string;
}
```

1/4. object のプロパティを切り出して interface 名を制御する

```
Item:
  type: object
  properties:
    item_id:
      type: integer
    owner:
      type: object
      required:
        - user_id
        - name
      properties:
        user_id:
          type: integer
        name:
          type: string
    author:
      $ref: '#/User'
```

```
User:
  type: object
  required:
    - user_id
    - name
  properties:
    user_id:
      type: integer
    name:
      type: string
```

```
author:
  $ref: '#/User'
```

```
/**
 *
 * @export
 * @interface User
 */
export interface User {
  /**
   *
   * @type {number}
   * @memberof User
   */
  'user_id': number;
  /**
   *
   * @type {string}
   * @memberof User
   */
  'name': string;
}
```

2/4. oneOf のふるまいを理解して、ユニオン型を生成させる

```
Item:
  type: object
  properties:
    item_id:
      type: integer
  owner:
    oneOf:
      - $ref: '#/User'
      - $ref: '#/Student'

User:
  type: object
  required:
    - user_id
    - name
  properties:
    user_id:
      type: integer
    name:
      type: string

Student:
  allOf:
    - $ref: '#/User'
    - type: object
      properties:
        class:
          type: string
```

- oneOf は Union 型の別の type を生成する
- allOf は展開される

2/4. oneOf のふるまいを理解して、ユニオン型を生成させる

```
Item:
  type: object
  properties:
    item_id:
      type: integer
    owner:
      oneOf:
        - $ref: '#/User'
        - $ref: '#/Student'
```

```
User:
  type: object
```

```
    owner:
      oneOf:
        - $ref: '#/User'
        - $ref: '#/Student'
```

```
Student:
  allOf:
    - $ref: '#/User'
    - type: object
      properties:
        class:
          type: string
```

```
/**
 * @type ItemOwner
 * @export
 */
export type ItemOwner = Student | User;
```

2/4. oneOf のふるまいを理解して、ユニオン型を生成させる

```
Item:
  type: object
  properties:
    item_id:
      type: integer
    owner:
      oneOf:
        - $ref: '#/User'
```

```
Student:
  allOf:
    - $ref: '#/User'
```

```
      type: string
```

```
Student:
  allOf:
    - $ref: '#/User'
```

```
/**
 *
 * @export
 * @interface Student
 */
export interface Student {
  'user_id': number;
  'name': string;
  'class'?: string;
}
```

3/4. oneOf であるキーを識別子にする複数の型を持つプロパティを表現する

```
UserType:
  enum:
    - "USER"
    - "STUDENT"

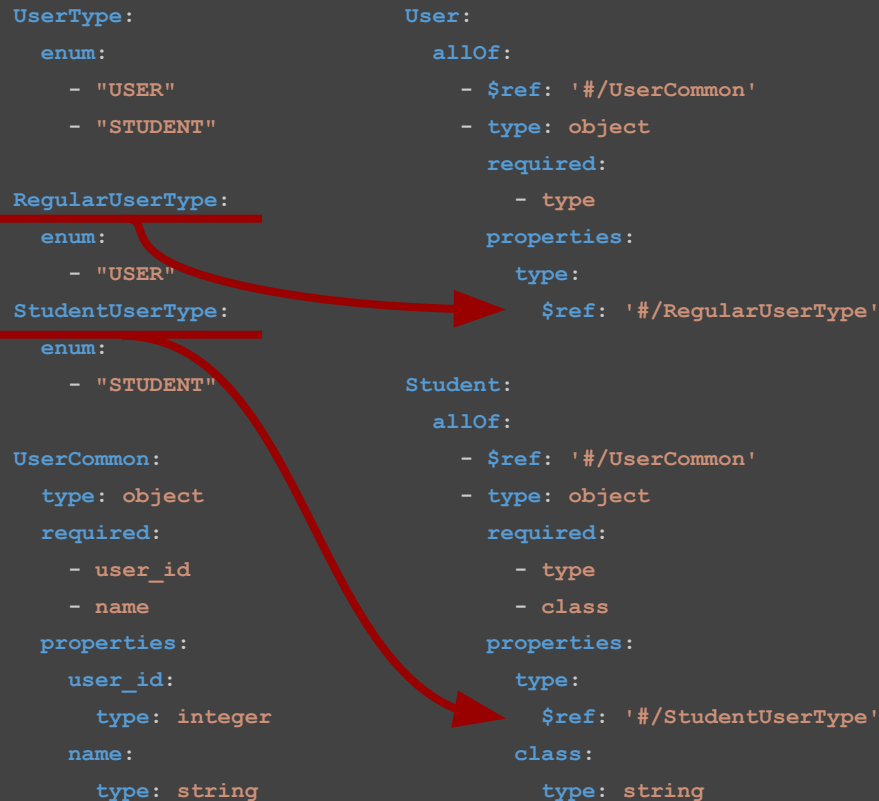
RegularUserType:
  enum:
    - "USER"

StudentUserType:
  enum:
    - "STUDENT"

UserCommon:
  type: object
  required:
    - user_id
    - name
  properties:
    user_id:
      type: integer
    name:
      type: string

User:
  allOf:
    - $ref: '#/UserCommon'
    - type: object
      required:
        - type
      properties:
        type:
          $ref: '#/RegularUserType'

Student:
  allOf:
    - $ref: '#/UserCommon'
    - type: object
      required:
        - type
        - class
      properties:
        type:
          $ref: '#/StudentUserType'
        class:
          type: string
```



1. 取りうる値が1つのみで構成される列挙型を定義する
2. 対応する object の識別子の型として該当する列挙型を指定する

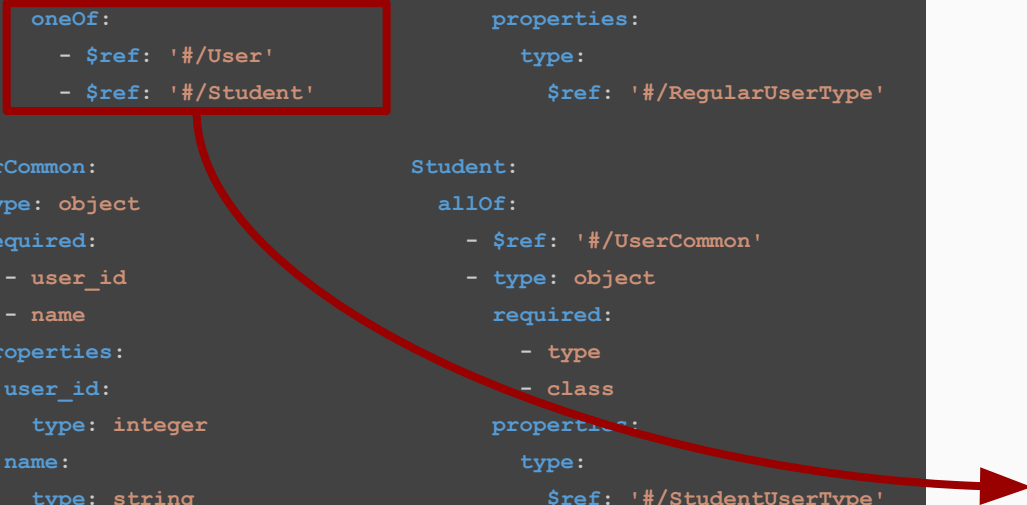
3/4. oneOf でプロパティを識別子にする複数の型を持つプロパティを表現する

```
Item:
  type: object
  properties:
    item_id:
      type: integer
    owner:
      oneOf:
        - $ref: '#/User'
        - $ref: '#/Student'

User:
  allOf:
    - $ref: '#/UserCommon'
    - type: object
      required:
        - type
      properties:
        type:
          $ref: '#/RegularUserType'

UserCommon:
  type: object
  required:
    - user_id
    - name
  properties:
    user_id:
      type: integer
    name:
      type: string

Student:
  allOf:
    - $ref: '#/UserCommon'
    - type: object
      required:
        - type
        - class
      properties:
        type:
          $ref: '#/StudentUserType'
        class:
          type: string
```



3. 含む object 側でそれぞれを
oneOf で列挙する

```
/**
 * @type ItemOwner
 * @export
 */
export type ItemOwner = User | Student;
```

4/4. allOf で定義された object を参照しながら nullable を表現する

```
Item:
  type: object
  properties:
    item_id:
      type: integer
    owner:
      oneOf:
        - $ref: '#/User'
        - $ref: '#/Student'
    author:
      $ref: '#/Student'
```



```
Item:
  type: object
  properties:
    item_id:
      type: integer
    owner:
      nullable: true
      oneOf:
        - $ref: '#/User'
        - $ref: '#/Student'
    author:
      nullable: true
      allOf:
        - $ref: '#/Student'
```

\$ref で参照している object に nullable を設定することはできない

allOf に参照する object のみを記し nullable を設定することで表現できる

4/4. allOf で定義された object を参照しながら nullable を表現する

```
Item:
  type: object
  properties:
    item_id:
      type: integer
  owner:
    oneOf:
      - $ref: '#/User'
      - $ref: '#/Student'
  author:
    $ref: '#/Student'
```



```
Item:
  type: object
  properties:
    item_id:
      type: integer
  owner:
    nullable: true
    oneOf:
      - $ref: '#/User'
      - $ref: '#/Student'
  author:
    nullable: true
    allOf:
      - $ref: '#/Student'
```

```
/**
 *
 * @export
 * @interface Item
 */
export interface Item {
  'item_id'?: number;
  'owner'?: ItemOwner | null;
  'author'?: Student | null;
}
```

まとめ

1. object のプロパティを切り出して interface 名を制御する
2. oneOf と allOf のふるまいを理解して、ユニオン型を生成させる
3. oneOf でプロパティを識別子にする複数の型を持つオブジェクトを表現する
4. allOf で定義された object を参照しながら nullable を表現する

gRPC クライアントの運用戦略についても技術ブログで公開しています。

<https://blog.nnn.dev/entry/2023/10/13/110000>

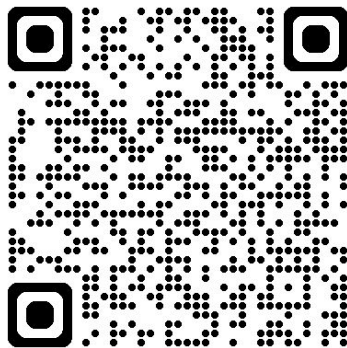


ご清聴いただきありがとうございました



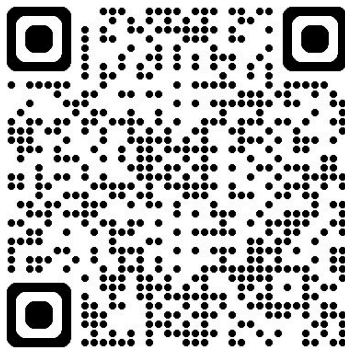
N 予 備 校

<https://www.nnn.ed.nico>



ドワンゴ教育サービス
開発者ブログ

<https://blog.nnn.dev>



教育プロジェクト
採用サイト

<https://www.nnn.ed.nico/recruit/>

