

The background features a collection of abstract geometric shapes in four colors: yellow, blue, red, and pink. These shapes include various polygons, circles, and semi-circles, some of which are partially obscured by the text. The shapes are scattered across the left and center of the image, creating a playful, geometric pattern.

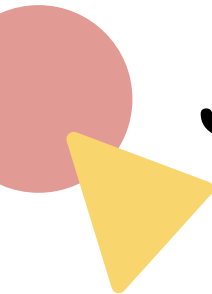

TypeSpec を使い倒してる

# 自己紹介





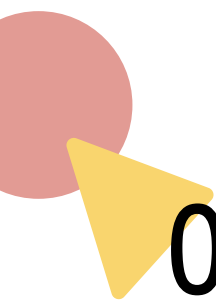
ユーン (@euxn23)

- ドワンゴで教育事業をやっています
- いまは何も聞かないでくれ

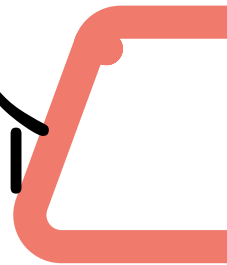


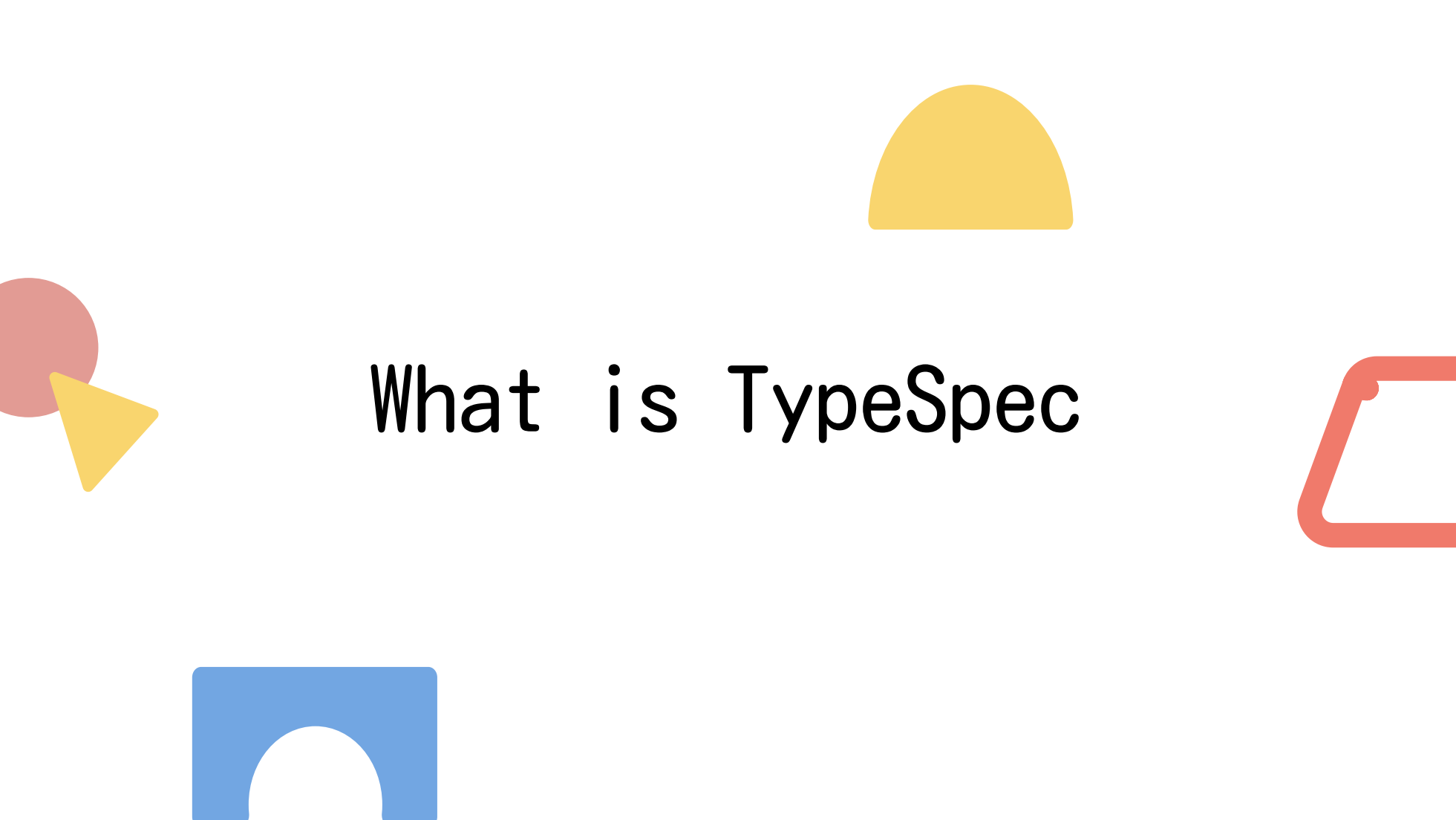
JS と言ったら OpenAPI Spec  
(誇大表現)





TypeSpec を使い倒して  
OpenAPI Spec を書く例を紹介







What is TypeScript



<https://typespec.io/>

- TypeScript / C# っぽい風味に API Spec を記述できる DSL/コンパイラ実装
  - プログラミング言語的な作法でコード分割や再利用が可能
  - OpenAPI Spec だけでなく JSON Schema としてや protobuf としての出力も可能
- 
- 



```
@route("/pets")
namespace Pets {
  op list(@query skip: int32, @query top: int32): {
    @body pets: Pet[];
  };
  op read(@path petId: int32): {
    @body pet: Pet;
  };
  @post
  op create(@body pet: Pet): {};
}
```

<https://typespec.io/docs/getting-started/getting-started-http#request--response-bodies>





いつもの PetStore を書いてみる





```
enum Versions {  
    v1,  
}  
  
model Pet = {  
    id: int32;  
    category: {  
        id: int32;  
        name: string;  
    };  
    name: string;  
    photoUrls: string[];  
    tags: {  
        id: int32;  
        name: string;  
    }[];  
    status: "available" | "pending" | "sold";  
};  
  
@service({  
    title: "PetStore",  
})  
@versioned(Versions)  
namespace PetStore {  
    @operationId("findPetById")  
    @summary("find-pet-by-id")  
    @get findPetById(@path id: int32): {  
        @body pet: Pet;  
    };  
}
```



ファイル分割してみる


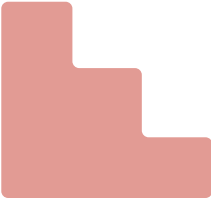
## こういうディレクトリ構造を考える


```
spec/  
├─ main.tsp  
├─ namespace.tsp  
├─ routes/  
│   ├── pet.tsp  
│   ├── store.tsp  
│   ├── user.tsp  
│   └─ main.tsp  
└─ models/  
    ├── api-response.tsp  
    ├── category.tsp  
    ├── pet.tsp  
    ├── tag.tsp  
    ├── order.tsp  
    ├── user.tsp  
    └─ main.tsp
```



main. tsp

```
import "./routes";  
import "./namespace";
```





namespace を空で宣言する namespace.tsp を作る

(各 route が namespace に子を生やしていく)



```
import "@typespec/versioning";

using TypeSpec.Versioning;

enum Versions {
  v1,
}

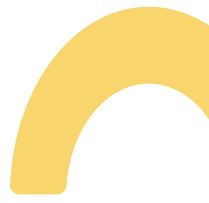
@service({
  title: "PetStore",
})
@versioned(Versions)
namespace PetStore {}
```





main.tsp がディレクトリのエントリーポイントとして解決されるので、  
models/main.tsp に配下の import を書く

```
import "./api-response.tsp";  
import "./category.tsp";  
import "./pet.tsp";  
import "./tag.tsp";  
import "./order.tsp";  
import "./user.tsp";
```



## models/pet.tsp

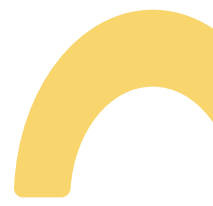
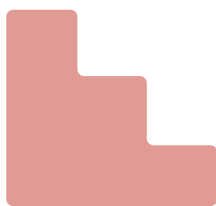
```
// import されたファイルに記載されている model は
// global を汚染するので具体的な名前をつける
model PetStatus = "available" | "pending" | "sold";

model Pet = {
  id: int32;
  category: {
    id: int32;
    name: string;
  };
  name: string;
  photoUrls: string[];
  tags: {
    id: int32;
    name: string;
  }[];
  status: PetStatus;
};
```



同様に routes/main.tsp に配下の import を書く

```
import "./pet.tsp";  
import "./store.tsp";  
import "./user.tsp";
```





## routes/pet.tsp

```
@route("/pet")
namespace PetStore.Pet {
  @route("/")
  interface Root {
    @operationId("post-pet")
    @summary("Add a new pet to the store")
    @post
    post(
      @body pet: Pet
    ): {
      @statusCode
      statusCode: 200 | 405;
      @body pet: Pet;
    };

    @operationId("put-pet")
    @summary("Update an existing pet")
    @put
    put(
      @body pet: Pet
    ): {
      @statusCode
      statusCode: 200 | 400 | 404 | 405;
      @body pet: Pet;
    };
  }
}
```

```

@route("/{id}")
namespace Id {
  @route("/")
  interface Root {
    @operationId("get-pet-by-id")
    @summary("Find pet by ID")
    @get
    get(
      @doc("ID of pet to return")
      @path
      id: string
    ): {
      @statusCode
      statusCode: 200 | 400 | 404;

      @body pet: Pet;
    };

    @operationId("post-pet-by-id")
    @summary("Update a pet in the store with form data")
    @post
    post(
      @header
      `content-type`: "multipart/form-data",
      @path
      id: string
      @multipartBody body: {
        name: HttpPart<string>;
        status: HttpPart<string>;
      }
    );
  }
}

```

...

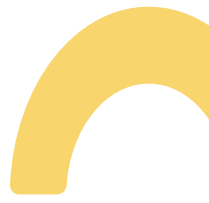
いいかんじ




こんなヘルパーを用意してもいいかもしれない


```
alias OmitID<T> = OmitProperties<T, "id">;
```



```
model EphemeralPet = OmitID<Pet>
```





## 現代の当たり前を、OpenAPI Spec でも

- 
- 冗長な重複記述は再利用で DRY に
  - VSCode のプラグインがあるので補完が効く
  - 構文エラーはコンパイラが落としてくれる



話し切れなかったことや具体的なテクは  
後日ドワンゴ教育サービス開発者ブログで！！

[blog.nnn.dev](http://blog.nnn.dev)





ありがとうございました

