Generic Type in API Models

Rajendra Shrestha
Mobile Developer

What is Generic Type?

- A Generic Type is a class or interface that is parameterized over types.
- Types are the particular type of data item, defined by the values it can take, the programming language used, the operations that can be performed on it.
- For a simpler example:
  List<int> listOfIntegers;
  Here, is the example of generic type *int* (data type);
- Simply, it can be denoted by *<T>*, where *T* is the type.

What is API Models?

- In plain words, API Models can be defined as the class that helps to deserialize the json from the API and set the values for each data types of the required keys.
- For example for json: *API_RESPONSE= { 'name':**'Rajendra'** }*

```
class NameDetail  {
String? name;

NameDetail.fromJson(Map<String, dynamic> json) {
        name = json['name'];
        }
}
```

- Here , *'NameDetail'* is the class that helps decodes the *API_RESPONSE* json to the nullable data type ***'String**?'* and sets the value ***'Rajendra'*** by calling the method

```
NameDetail.fromJson(API_RESPONSE);
```

So Why Generic Type<T>?

- In larger projects, we have multiple api calls and responses that helps to properly execute the software.
- Hence multiple api call has multiple API model class that helps to decode the data provided through the api.
- Here a common *wrapper* class is used to wrap the data in the api responses
- So for each api response there will be api model class
- This makes the api model class heavier and more duplicate models are present.
- So for this we use a common api model for the data wrapper class.

So Why Generic Type<T>?

For example we have two json api response such as:

// single json data

```
{
    "statusCode": 200,
    "message": "Success",
    "data": {
        "name": "Rajendra",
        "id": 1
    }
}
```

// Multiple list json data

```
{
  "statusCode": 200,
  "message": "Success",
  "data": [
    {
      "name": "Rajendra",
      "id": 1
    },
    {
      "name": "Praveen",
      "id": 2
    }
  ]
}
```

So Why Generic Type<T>?

The api model class for these two would look something like:

```dart
class SingleResponse {
  int? statusCode;
  String? message;
  Data? data;

  SingleResponse({this.statusCode, this.message, this.data});

  SingleResponse.fromJson(Map<String, dynamic> json) {
    statusCode = json['statusCode'];
    message = json['message'];
    data = json['data'] != null ? Data.fromJson(json['data']) : null;
  }

  Map<String, dynamic> toJson() {
    final Map<String, dynamic> data = <String, dynamic>{};
    data['statusCode'] = statusCode;
    data['message'] = message;
    if (this.data != null) {
      data['data'] = this.data!.toJson();
    }
    return data;
  }
}
```

```dart
class ListResponse {
  int? statusCode;
  String? message;
  List<Data>? data;

  ListResponse({this.statusCode, this.message, this.data});

  ListResponse.fromJson(Map<String, dynamic> json) {
    statusCode = json['statusCode'];
    message = json['message'];
    if (json['data'] != null) {
      data = <Data>[];
      json['data'].forEach((v) {
        data!.add(Data.fromJson(v));
      });
    }
  }

  Map<String, dynamic> toJson() {
    final Map<String, dynamic> data = <String, dynamic>{};
    data['statusCode'] = statusCode;
    data['message'] = message;
    if (this.data != null) {
      data['data'] = this.data!.map((v) => v.toJson()).toList();
    }
    return data;
  }
}
```

Conclusion

So we saw how we could get data from api creating a generic data class that works on both list and single data types.

If you have any questions please let me know.

So Why Generic Type<T>?

Here we can see that same data type and decoding is done for two different class

```
SingleResponse.fromJson(Map<String, dynamic> json) {
  statusCode = json['statusCode'];
  message = json['message'];
  data = json['data'] != null ? Data.fromJson(json['data']) : null;
}
```

```
ListResponse.fromJson(Map<String, dynamic> json) {
  statusCode = json['statusCode'];
  message = json['message'];
  if (json['data'] != null) {
    data = <Data>[];
    json['data'].forEach((v) {
      data!.add(Data.fromJson(v));
    });
  }
}
```

To remove this redundancy we create a common response class that helps us to decode these data once and also have different data models in '*data*'

For this we create a generic class that helps us to have multiple data classes

So Why Generic Type<T>?

Here we create a common api response for single class and data class

```dart
class Data {
  String? name;
  int? id;

  Data({this.name, this.id});

  Data.fromJson(Map<String, dynamic> json) {
    name = json['name'];
    id = json['id'];
  }
```

Here *'Data'* is the class that has data, so we pass this class as a generic class to the *'CommonSingleApiResponse<Data>'*.

```dart
CommonSingleApiResponse<Data> anyData = CommonSingleApiResponse.fromJson(
  singleData,
  (json) => Data.fromJson(json),
);  // CommonSingleApiResponse.fromJson

var apiData = anyData.data as Data;
```

```dart
class CommonSingleApiResponse<T> {

  int? statusCode;
  String? message;
  dynamic data;

  CommonSingleApiResponse(
      {this.statusCode,
        this.message,
      this.data});

  factory CommonSingleApiResponse.fromJson(
      Map<String, dynamic> json,
      Function(Map<String, dynamic>) create,
      ) {
    return CommonSingleApiResponse<T>(
      statusCode: json['statusCode'],
      message: json['message'],
      data: create(json["Data"]),
    );
  }
}
```

So Why Generic Type<T>?

For list data we have a different way for decoding the list, so we have to create another class for generic data type

Here '*Data*' is the class that has data, here we have list of data that is to be fetched from api so we have a bit different approach.

```dart
/// This is how we get list of generic data in the apiData
CommonListApiResponse<Data> genericListData = CommonListApiResponse.fromJson(
  listData,
      (json) => Data.fromJson(json),
);  // CommonListApiResponse.fromJson

var apiListData = genericListData.data as List<Data>;
```

```dart
class CommonListApiResponse<T> {
  int? statusCode;
  String? message;
  List<T>? data;

  CommonListApiResponse({this.statusCode, this.message, this.data});

  factory CommonListApiResponse.fromJson(
    Map<String, dynamic> json,
    Function(Map<String, dynamic>) create,
  ) {
    var data = <T>[];

    json['data'].forEach((v) {
      data.add(create(v));
    });
    return CommonListApiResponse<T>(
      statusCode: json['statusCode'],
      message: json['message'],
      data: data,
    );
  }
}
```