LeanCode

Programming Mobile Applications in Flutter

Code Generation

Code Generation

- Why? Generation of boilerplate code to satisfy language's needs
- Improve code quality:
  - Reduce boilerplate
  - Make the code more readable
  - Reduce the number of possible bugs
- When?
  - Data classes - classes whose main purpose is to hold data
  - Sealed classes - used to restrict the users from inheriting the class
  - Architecture boilerplate - *MobX, Hive*
  - Common features/functions - *fromJson(), toJson(), copyWith()*

Code Generation

- Why do we need it?
  - Dart contains little magic, code is explicit
  - Things like json serialization are a manual task
  - `dart:mirrors` is forbidden in Flutter: kills tree shaking capabilities
    - `package:reflectable` provides mirrors using code generation

Annotations

Annotations

- Used to annotate classes, fields, functions, etc.
- Has **no** direct effect on code, it is just an annotation
  - Unlike decorators in Python/Typescript which change the behavior
- Have to be constant objects in order to be analyzable at compile time
- Analyzed at the semantic stage of the compiler
- Any constant class can be an annotation, meaning is given during analysis

```
@Deprecated('Use `NewApi` instead.')
class OldApi {}

final use = OldApi();
```

`package:meta`

demo

Part files

Part files

- Used to split a file into multiple ones
- Private identifiers are visible across part files
- `part of` directive has to be matched with a `part`: it is not dynamic
- Useful for separating generated files from our files

```dart
part 'custom.g.dart';
```

In `custom.dart`

```dart
part of 'custom.dart';
```

In `custom.g.dart`

`package:json_serializable`

Json serialization mappers

Code Generation - `json_serializable`

```dart
import 'package:freezed_annotation/freezed_annotation.dart';

// A from and to json function will be generated by `build_runner`.
// Generated files land in `*.g.dart` where `g` is for "generated".
part 'user.g.dart';

// An annotation for the code generator to know that this class needs the
// JSON serialization logic to be generated.
// The annotation accepts many configuration options, for instance changing
// the casing of fields (here changed to PascalCase).
@JsonSerializable(fieldRename: FieldRename.pascal)
class User {
 const User(this.name, this.email, this.age, this.isAdmin);

 // A convention factory constructor for creating a new User instance
 // from a map. Pass the map to the generated `_$UserFromJson()` function.
 // The constructor is named after the source class, in this case, User.
 factory User.fromJson(Map<String, dynamic> json) => _$UserFromJson(json);

 final String name;
 final String email;
 final int age;
 final bool isAdmin;

 // A convention method `toJson` The implementation simply calls
 // the private, generated helper function `_$UserToJson`.
 Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

Code Generation - `json_serializable`

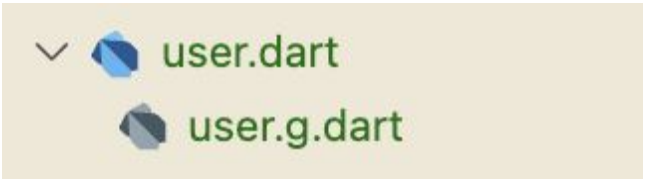
```dart
// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'user.dart';

// **************************************************************************
// JsonSerializableGenerator
// **************************************************************************

User _$UserFromJson(Map<String, dynamic> json) => User(
     json['Name'] as String,
     json['Email'] as String,
     json['Age'] as int,
     json['IsAdmin'] as bool,
   );

Map<String, dynamic> _$UserToJson(User instance) => <String, dynamic>{
     'Name': instance.name,
     'Email': instance.email,
     'Age': instance.age,
     'IsAdmin': instance.isAdmin,
   };
```



`package:freezed`

Immutable data classes with unions

Freezed

Using **Freezed**, you will get:

- A simple and concise syntax for defining models, where we don't need to define both a constructor and a property. Instead, we only need to define the constructor, removing unnecessary duplication
- A *copyWith* method, for cloning objects with different values
- **union-types/pattern matching,** for making impossible states impossible
- An automatic serialization/deserialization of your objects (including union types)
- A default **==/toString** implementation which respectively compares/shows all properties of the object

Source: https://pub.dev/packages/freezed#motivation

Freezed

```dart
import 'package:freezed_annotation/freezed_annotation.dart';

part 'data_class.freezed.dart';

// freezed classes are "sealed": you shouldn't extend them
@freezed
class User with _$User {
 // Freezed will read all constructor parameters and turn them into fields of `User`
 // assertions need a special annotation
 @Assert('age ⩾ 18')
 const factory User({
   required String name,
   required String? email,
   required int age,
   // default value needs a special annotation
   @Default(false) bool isAdmin,
   required Map<String, String> attributes,
 }) = _User;
}
```

freezed + json_serializable

```dart
import 'package:freezed_annotation/freezed_annotation.dart';

part 'person.freezed.dart';
part 'person.g.dart';

@freezed
class Person with _$Person {
  @Assert('age >= 0')
  factory Person({ required String name, @Default(18) int age }) = _Person;

  factory Person.fromJson(Map<String, dynamic> json) => _$PersonFromJson(json);
}
```

Freezed - Unions/Sealed classes

```dart
import 'package:freezed_annotation/freezed_annotation.dart';
import 'package:week11_lecture/code_generation/freezed/person.dart';

part 'person_state.freezed.dart';

@freezed
class PersonState with _$PersonState {
  const factory PersonState.success(Person person) = _Success;

  const factory PersonState.error(String errorText) = _Error;

  const factory PersonState.loading() = _Loading;
}
```

`package:build_runner`

- A *dev_dependency* that manages code generation
  - Provides a protocol for code generators
  - Generators are usually split into two packages: `<package>_annotations` and `<package>`
    - `<package>` is executed by `build_runner`
    - `<package>` receives the semantic AST
  - Not always annotation-based: `package:flutter_gen`
- Manages file changes and dependencies between
- ***flutter pub run build_runner build*** - Runs a single build and exits
- Annotations
- More resources about code generation:
  - Code generation in Dart: the basics
  - source_gen

flutter_localizations

Questions?