Programação Assíncrona e Armazenamento Offline

Computação para Dispositivos Móveis

Prof. Gustavo Custodio
gustavo.custodio@anhembi.br

Programação Assíncrona

Programacao Assncrona e Armazenamento Offline

Programação Assíncrona

- Quando você escreve um código, você espera que as instruções rodem de forma sequencial.

```
1    int x = 5;
2    int y = x * 2;
```

- O esperado é que o valor de y será 10 porque a segunda linha **espera** a primeira linha terminar.

Programação Assíncrona

- Se uma linha demora para ser executada, o aplicativo “trava”, não permitindo input do usuário.
  - Por isso, na maioria das linguagens de programação modernas, incluindo Dart, você pode realizar **operações assíncronas**.
    - Dessa forma, o usuário pode continuar interagindo com o app enquanto ele realiza uma operação complicada.
- **Operações assíncronas não afetam a linha de execução principal**.

Futures

- Em Dart e Flutter, é possível escrever código assíncrono utilizando **`Futures`**.
- Um `Future` representa o resultado de uma operação assíncrona e possui dois estados: completo e incompleto.
- Um `Future` possui a sintaxe: `Future<Tipo>`
  - onde o `Tipo` corresponde ao retorno esperado da operação assíncrona.

Futures

- Vamos utilizar um exemplo onde adquirimos informação da API do *Google Books*.
- Para fazer isso, utilizamos a biblioteca `http` do Dart.
  - Mas antes, ela precisa ser adicionada em nosso projeto.
- Primeiro, crie um projeto no Flutter.

Futures

- Abra o arquivo `pubspec.yaml`.
- Procure a linha `dependencies`.
- E adicione o `http` versão 0.13.0 conforme mostrado abaixo:

```
1 dependencies:
2   flutter:
3     sdk: flutter
4   http: ^0.13.0
```

  - Em seguida, salve as alterações.

Futures

- Vamos criar nossa classe principal no arquivo `main`:

```dart
import 'package:flutter/material.dart';
import 'tela_google_books.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.green,
      ),
      home: PaginaFutura(),
    );
  }
}
```

Futures

- Em seguida crie um arquivo chamado `tela_google_books.dart`.
  - Realize os imports necessários:

```dart
import 'package:flutter/material.dart';
import 'dart:async';
import 'package:http/http.dart';
import 'package:http/http.dart' as http;
```

Futures

- Crie um `StatefulWidget` chamado `PaginaFutura`:

```dart
class PaginaFutura extends StatefulWidget {
  const PaginaFutura({Key? key}) : super(key: key);

  @override
  State<PaginaFutura> createState() => _PaginaFuturaState();
}

class _PaginaFuturaState extends State<PaginaFutura> {
  // Retorna o resultado do conteúdo procurado na web
  String resultado = "";

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Testando Futures"),
      ),
      body: Center(
        child: Column(children: [
          Spacer(), // adiciona um espaço entre elementos na coluna
          //_criarBotaoDeBusca(context),
          Spacer(),
          //Text(resultado.toString()),
        ]),
      ),
    );
  }
```

Futures

- Agora criaremos uma função chamada `getDadosAPI()`.
  - Ela será responsável por adquirir os dados em `json` da API do Google Books.

```dart
Future<Response> getDadosAPI() async {
  final String dominio = 'www.googleapis.com';
  final String caminho = '/books/v1/volumes/junbDwAAQBAJ';
  Uri url = Uri.https(dominio, caminho);
  return http.get(url);
}
```

- A função retorna uma `Future` que quando completa, "se transforma" em uma `Response`.

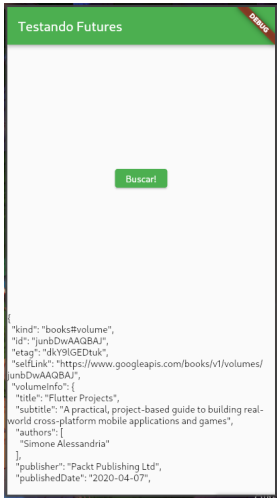
Futures

- Por último, criamos um botão que, quando pressionado, mostrará o texto recuperado da API em um widget de texto.

```
Widget _criarBotaoDeBusca(BuildContext context) {
  return ElevatedButton(
    child: Text('Buscar!'),
    onPressed: (){
      resultado = '';
      setState(() {
        resultado = resultado;
      });
      /* Chama getDadosAPI e quando a future for completada,
         rode o codigo dentro do then */
      getDadosAPI().then((valorRetorno) {
        resultado = valorRetorno.body.toString().substring(0, 450);
        setState(() {
          resultado = resultado;
        });
      });
    },
  );
}
```

Futures

- Produzimos o seguinte resultado:
- No futuro aprenderemos como extrair os dados formatados de um `json`.



Futures

- O que aconteceu?
  - Quando clicamos no botão, chamamos a **função assíncrona** chamada `getDadosAPI`.
  - Quando o `Future` é completada, a função no `then` é invocada.

```
getDadosAPI().then((valorRetorno) {
  resultado = valorRetorno.body.toString().substring(0, 450);
  setState(() {
    resultado = resultado;
  });
});
```

Futures

- O `then` não facilita a visibilidade do código.
  - Cria muitas estruturas aninhadas.
  - Uma alternativa ao `then` são as palavras chave `async` e `await`.

Async e Await

- Ao invés de utilizar, o `then`, vamos reescrever o trecho de código onde o `Future` é executado utilizando o `await`.

```
onPressed: ()async {
  resultado = '';
  setState(() {
    resultado = resultado;
  });
  // Métodos assíncronas devem ter um await antes de serem chamados.
  var valorRetorno = await getDadosAPI();
  resultado = valorRetorno.body.toString().substring(0, 450);
  setState(() {
    resultado = resultado;
  });
},
```

Async e Await

- Observe que a função anônima chamada pelo `onPressed` agora possui um `async` em sua assinatura.
  - Todos os métodos que utilizam a palavra chave `await` devem ser assíncronos (`async`).

Async e Await

- Lado a lado a diferença entre o then e o async / await:

```
Future<Response> getData() {
    String url = 'https//url.com';
    return http.get(url);
}

void algumMetodo() {
    getData()
        .then((valor) {
            // Faça algo com o valor retornado.
        }
    )};
}
```

```
Future<Response> getData() {
    String url = 'https//url.com';
    return http.get(url);
}

Future algumMetodo() async {
    var valor = await.getData();
    // Faça algo com o valor retornado.
}
```

Programação Assíncrona

- O armazenamento offline utiliza programação assíncrona para armaazenar conteúdo.
- Isso será feito utilizando o recurso `SharedPreferences`.

Programacao Assncrona e Armazenamento Offline

Armazenamento Offline

Shared Preferences

- O primeiro passo é realizar a importação do SharedPreferences.
  - Abra o arquivo pubspec.yaml e adicione o seguinte conteúdo em dependencies:

```
dependencies:
  flutter:
    sdk: flutter
  shared_preferences: ^2.0.15
```

Shared Preferences

- Adicione o import em seu projeto:

```
1    import 'package:shared_preferences/shared_preferences.dart';
```

- Crie uma classe que herda de `StatefulWidget` chamada `MinhaPaginaPrincipal`.
  - Adicione o atributo em `_MinhaPaginaPrincipalState`:
    - `int contadorApp;`

Armazenameno Offline

- Crie um método chamado `lerEscreverPreferencias`:
  - `Future lerEscreverPreferencias()async.`
- Dentro desse método, crie uma instância de `SharedPreferences`:
  - `SharedPreferences prefs = await SharedPreferences.getInstance();`

Armazenamento

- Ainda no mesmo método, tente ler o valor armazenado de contadorApp.
  - Se o valor é nulo, atribua 1.
  - Caso contrário, acrescente 1 no contador.

```
1    int? valorSalvo = prefs.getInt('contadorApp');
2    // Verifica se o contador salvo énulo
3    if (valorSalvo == null) {
4      contadorApp = 1;
5    }else {
6      contadorApp = valorSalvo;
7      contadorApp++;
8    }
```

Armazenamento

- Atualize o valor de contadorApp armazenado:
  - `await prefs.setInt('contadorApp', contadorApp);`
- E atualize o valor na tela:

```
1    setState(() {
2        contadorApp = contadorApp;
3    }
```

Armazenamento

- Ainda na classe `_MinhaPaginaPrincipalState`, adicione o método `initState`:

```
1    @override
2    void initState() {
3        lerEscreverPreferencias();
4        super.initState();
5    }
```

Armazenamento

- No método build, adicione o seguinte código

```
1       body: Center(
2         child: Column(
3           mainAxisAlignment: MainAxisAlignment.spaceEvenly,
4           children: [
5             Text('Você abriu o aplicativo ' +
6                 contadorApp.toString() + ' vezes.'),
7             ElevatedButton(
8               onPressed: (){},
9               child: Text('Reiniciar Contador'),
10            )
11          ],
12        ),
13      ),
```

Armazenamento

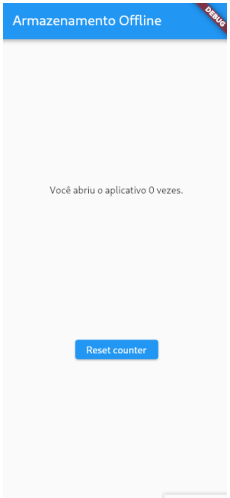
- Por último, adicione um método para apagar os dados armazenados:
  - `Future deletarPreferencias()async`

```
Future deletarPreferencias() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.clear();
  setState(() {
    contadorApp = 0;
  });
}
```

- Adicione o método ao botão:
  - `onPressed: deletarPreferencias,`

Armazenamento

- O resultado é um aplicativo que conta quantas vezes ele foi aberto.



Referências

Simone Alessandria and Brian Kayfitz.
Flutter Cookbook: Over 100 proven techniques and solutions for app development with Flutter 2.2 and Dart.
Packt Publishing Ltd, 2021.

Conteúdo

https://gustavotcustodio.github.io/dispomoveis.html

Obrigado

gustavo.custodio@anhembi.br