VOLTAR

Coleções com Mapas e Listas em DART

Um recurso que utilizamos muito no Flutter são as famosas listas, especialmente quando estamos consumindo dados de um banco de dados ou de uma aplicação externa. Para isso, veremos nesta aula como trabalhar com listas e mapas em Dart.

NESTE TÓPICO

Conceito de listas

Trabalhar com listas é algo muito importante em qualquer linguagem de programação, pois permitir gerir, de uma única vez, ou seja, com um único código, uma série de dados que podem ser necessários para o contexto da aplicação. Mas o que é uma lista?

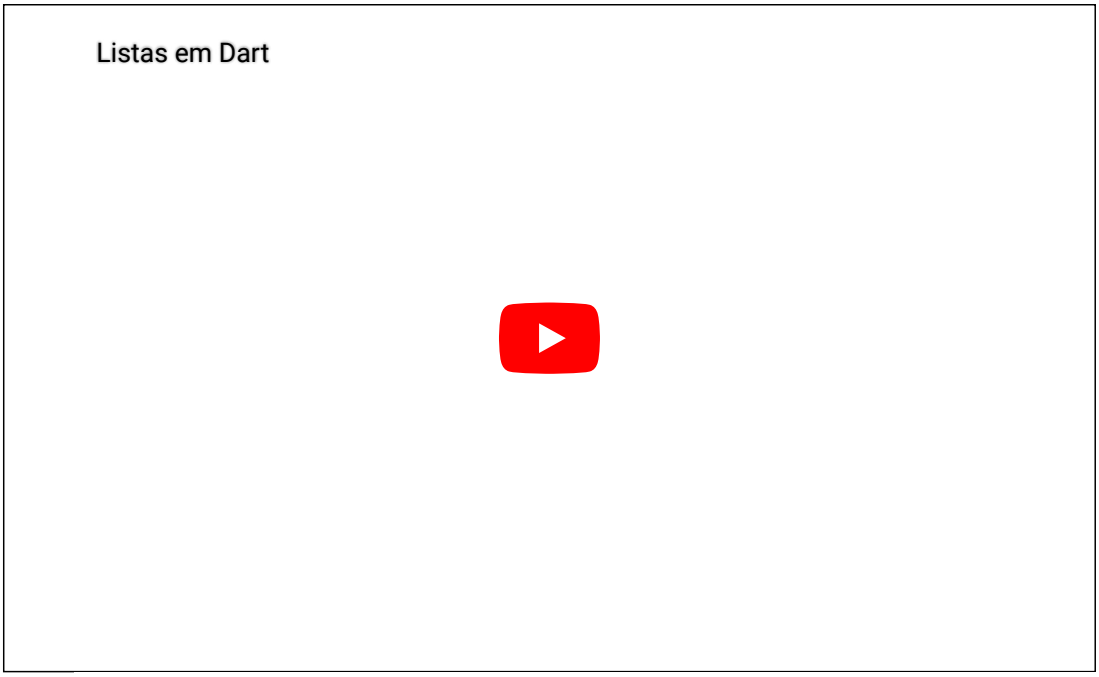
Bom, uma lista é nada mais do que uma estrutura que armazena dados na memória do dispositivo. Estes dados são normalmente armazenados de forma sequencial, ou seja, uma lista comum é uma estrutura que armazena vários dados um após o outro, na mesma variável.
As listas em Dart são baseadas em vetores, da estrutura de dados básica, mas não se preocupe se você não sabe ou não lembra muito bem o que é um vetor. O vídeo abaixo mostra cuidadosamente o conceito da estrutura de dados do tipo "vetor" e como são endereçadas suas posições.

Listas em Dart

Uma vez entendido o conceito base de vetores e como funciona seu endereçamento, vamos entender como aplicar isso em Dart agora.

Em Dart a forma mais simples e usual de utilizar um vetor é com o objeto do tipo "List", que representa uma lista no sistema.

O vídeo abaixo mostra como podem ser implementadas as listas em Dart, utilizando o site DartPad.dev. Não deixe de tentar criar suas próprias listas.



O código descrito abaixo representa o mesmo utilizado no vídeo anterior, devidamente comentado para melhor entendimento:

```
class Aluno {

  //Atributos
  int ra;
  String nome;

  //Construtor
  Aluno(this.ra, this.nome);
}

void main() {

  List<String> compras = ["Cenoura", "Banana", "Brócolis", "Refri"];

  //Imprimindo o primeiro item da lista:
  print(compras[0]);

  //Adicionando mais um item na lista:
  compras.add("Suco");

  //Imprimindo a lista inteira:
  print(compras);

  //Para pegar o tamanho da lista:
  print(compras.length);

  //verific ar se a lista contém um determinado item:
  print(compras.contains("Refri"));

  //Lista de objetos do tipo aluno
  List<Aluno> alunos = List();

  //Acrescentando alunos na lista, através do
  //construtor do aluno diretamente na ação de
  //acrescentar na lista:
  alunos.add(Aluno(123, "Josefina"));
  alunos.add(Aluno(321, "Josefino"));
  alunos.add(Aluno(345, "Thiago"));

  //Laço de repetiçãso do tipo foreach (para cada)
  for(Aluno a in alunos) {
    print(a.nome);
  }
}
```

Muito interessante isso, não é mesmo? Bom, no Flutter, quando começarmos a desenvolver aplicações realmente funcionais, o uso das listas será muito importante para que possamos armazenar e manipular dados de origens externas, como bancos de dados, arquivos recebidos, informações para o usuário etc. Você vai gostar muito de desenvolver com Flutter.

Laços de repetição

Quando começamos a falar de listas o conceito de repetição, para acessar os dados dessa lista, fora introduzido. Embora os laços de repetição em Dart sejam muito parecidos com qualquer outra linguagem, vamos ver nesta sessão como eles funcionam.

O laço de repetição, apenas "loop" é uma estrutura de código que permite que um certo trecho de código (interno ao ou laço) seja repetido, por diversos motivos, várias vezes, até um determinado critério de parada / saída.

Na maior parte das linguagens de programação existem quatro tipos de laços de repetição (ou loops), que são o "for", o "while", o "do..while" e o "foreach:". Para entendermos qual é a estrutura de cada um deles, não deixe de ver o vídeo abaixo.

E o código do vídeo anterior por ser visto no bloco a seguir:

```
1.  void main() {
2.
3.    //For
4.    for(int i = 0; i <= 20; i+=2) {
5.      print(i);
6.    }
7.
8.    //Whilw (enquanto)
9.    int j = 0;
10.   while (j <= 20) {
11.     print(j);
12.     j++;
13.   }
14.
15.   //do.. while (faça, enquanto)
16.   int k = -1;
17.   do {
18.     print(k);
19.     k+=3;
20.   } while (k <= 21);
21. }
```

Como você pode observar, se você conhece outras linguagens de programação deve ter notado que a sintaxe destes loops é muito parecida com as linguagens modernas. Isso ocorre pela própria característica do Dart, que é possui uma sintaxe baseada no que cada linguagem de programação moderna trás de melhor. Sensacional isso.

Loops para iterar listas de objetos

Muitas vezes temos listas de objetos que nós mesmos definimos e, para acessar os itens internos dessas listas, temos que utilizar os laços de repetição.

Para praticarmos um pouquinho mais sobre estes laços de repetição, listas e classes, vamos assistir cuidadosamente o vídeo abaixo, que mostra a criação de uma classe, a criação de uma lista de objetos dessa classe e os diferentes tipos de loops para acessar cada um dos itens da lista em questão.

E o código completo e devidamente comentado, do vídeo anterior pode ser visto em:

```
class Fruta {
  String nome;
  bool madura;

  Fruta(this.nome, this.madura);
}

void main() {
  List<Fruta> frutas = List();

  frutas.add(Fruta("Banana", true));
  frutas.add(Fruta("Mamão", false));
  frutas.add(Fruta("Abacate", true));
  frutas.add(Fruta("Abacaxi", false));
  frutas.add(Fruta("Carambola", true));

  //foreach
  print("Usando o foreach: ");
  for(Fruta f in frutas){
    print(f.madura
        ? "Já pode comer ${f.nome} "
        : "Não pode comer ainda ${f.nome}");
  }

  //usando o for...
  print("Usando o for: ");
  for (int i = 0; i < frutas.length; i++) {
    //print(frutas[i].nome);
    print(frutas[i].madura
        ? "Já pode comer ${frutas[i].nome} "
        : "Não pode comer ainda ${frutas[i].nome}");
  }

  //usando o "while"...
  print("Usando o while: ");
  int j = 0;
  while (j < frutas.length) {
    print(frutas[j].madura
        ? "Já pode comer ${frutas[j].nome} "
        : "Não pode comer ainda ${frutas[j].nome}");
    j++;;
  }

  //do.. while (faça, enquanto)
  print("Usando o do..while, impriminto a lista ao contrário: ");
  int k = frutas.length -1;
  do {
    print(frutas[k].madura
        ? "Já pode comer ${frutas[k].nome} "
        : "Não pode comer ainda ${frutas[k].nome}");
    k--;
  } while (k >= 0);
}
```

Não deixe de praticar bastante criando outras listas com diferentes objetos e realizando a interação de seus itens, imprimindo de vária formas cada um dos elementos. Você verá que a prática é essencial para entender muito bem este conceito.

Mapas em Dart

Os mapas representam um recurso extremamente importante pois utilizamos consideravelmente bastante no desenvolvimento de aplicativos com Flutter, especialmente quando estamos consumindo dados externos à nossa aplicação. Mas o que é um Mapa? Não, não se confunda com mapas geográficos, não possuem nem mesmo relação com isso.

Em programação os mapas são listas especiais, que associam um dado sempre à uma chave (relação chave-valor, *key-value*).

Para entendermos bem o conceito de mapa, pense em uma tabela de duas colunas. A primeira coluna é a chave que é única para cada linha e a segunda coluna é o valor. Então os mapas são uma representação exata deste conceito.

No Dart, tanto as chaves quando os valores dos mapas, podem ser de qualquer tipo de dados e, por isso, utilizamos muito o tipo "dynamic" que permite o uso de qualquer tipo de dado.

Mas antes de ver o vídeo dos mapas em Dart, não deixe de assistir o vídeo abaixo que explica melhor um conceito muito importante, usado no vídeo dos mapas, sobre "arrow functions" em Dart.

O código utilizado no vídeo anterior pode ser visto em:

```dart
void main() {

  print(ePar(28));

  print(ePar2(2316201));

  print(calculaArea(3, 9));

  imprimeNome("Josefina");
}

bool ePar(int numero) {
//   if (numero % 2 == 0) {
//     return true;
//   }
//   return false;
  return (numero % 2 == 0) ? true : false;
}

bool ePar2(int numero) => (numero % 2 == 0);

int calculaArea(int base, int altura) => base * altura;

void imprimeNome(String nome) => print("Olá $nome");
```

Agora sim, já conhecemos um pouco melhor as funções "arrow" do Dart e podemos assistir o vídeo sobre os mapas em Dart, conforme abaixo.

E o bloco de código abaixo representa o código desenvolvido no vídeo anterior:

```
class DadosVeiculo {
  String fabricante;
  String modelo;
  int ano;
  DadosVeiculo(this.fabricante, this.modelo, this.ano);
}

void main() {

  Map<String, int> objetos = Map();
  objetos["Celular"] = 2;
  objetos["Canetas"] = 20;
  objetos["Cadeiras"] = 4;

  print(objetos);

  Map<String, dynamic> func = Map();
  func["nome"] = "Josefina";
  func["salario"] = 20000.50;
  func["dependentes"] = 1;
  func["emHomeOffice"] = true;

  print(func["emHomeOffice"]);

  Map<String, DadosVeiculo> carros = Map();

  carros["ABC-1234"] = DadosVeiculo("Ferrari", "F-250", 2021);
  carros["ABC-4321"] = DadosVeiculo("Bugatti", "Veyron", 2022);

   carros.forEach((k, v) => print("Placa: ${k}: ${v.fabricante}, ${v.modelo} (${v.ano})"));

}
```

Fique tranquilo(a) se os conceitos de listas e mapas não ficarem tão claros neste momento, pois eles realmente possuem uma certa complexidade. Quando começarmos a desenvolver aplicativos realmente funcionais, para um determinado propósito, a aplicação prática destes conceitos os deixará muito mais simples e claros de entender. Então, vamos em frente que ainda temos bastante código para desenvolver. Usaremos muito o conceito de mapas e listas quando estivermos desenvolvendo com Flutter e os conceitos novos e subjacentes serão apresentados conforme a necessidade e uso dos recursos. Você vai gostar bastante, pois isso deixará a programação ainda mais fácil e rápida.

Quiz

Exercício Final

Coleções com Mapas e Listas em DART

INICIAR

Referências

BRACH, Gilard, LARS, Bak. Dart: a new programming language for structured web programming. **GOTO Conference**, 10 out. 2011. Disponível em: https://gotocon.com/dl/goto-aarhus-2011/slides/GiladBracha_and_LarsBak_OpeningKeynoteDartANewProgrammingLanguageForStructuredWebProgramming.pdf. Acesso em: 12 nov. 2020.

DART. **Dart documentation**. *Site*. Disponível em: https://dart.dev/. Acesso em: 12 nov. 2020.

FELIX, Rafael. **Programação orientada a objetos.** São Paulo: Pearson, Ed. Pearson, 2017. *E-book*. Disponível em: https://plataforma.bvirtual.com.br/Leitor/Loader/128217/pdf. Acesso em 19 nov. 2020.

OKEDIRAN, O. O. *et al*. Mobile operating systems and application development platforms: a survey. **Int. J. Advanced Networking and Applications**. v.6, n.1, p. 2195-2201, july-aug. 2014. Disponível em: https://www.ijana.in/download%206-1-9.php?file=V6I1-9.pdf. Acesso em: 12 nov. 2020.

SCHWARZMÜLLER, Maximilian. **Learn Flutter and Dart to Build iOS and Android Apps 2020.** Oreilly, Packt Publishing, 2020. *Vídeo*. Disponível em: https://learning.oreilly.com/videos/learn-flutter-and/9781789951998/. Acesso em 19 nov. 2020.

SINHA, Sanjib . **Quick start guide to Dart programming:** create high performance applications for the web and mobile. Lompoc, CA, EUA: Apress, 2019. *E-book*. Disponível em: https://learning.oreilly.com/library/view/quick-start-guide/9781484255629/. Acesso em: 12 nov. 2020.

WINDMILL, Eric. **Flutter in action.** Nova Iorque: Manning publications, 2020. *E-book*. Disponível em: https://learning.oreilly.com/library/view/flutter-in-action/9781617296147/, Acesso em: 12 nov. 2020.

Avalie este tópico

Ajuda? (https://ava.uninove.br/seu/AVA/topico/topico.php?idCurso=)

Portal Uninove
(http://www.uninove.br)
Mapa do Site