Análise das EDs built-in em Dart

Tudo certo?

Vamos mergulhar nas nuances das estruturas de dados em Dart.

Alunos
LAÍS E SERGIO

Data
04/10/2023

Agenda

- O que é Dart?
- Importância das Estruturas de Dados
- Comportamento das EDs built-in de Dart
- Aplicações
- Referências

O que é Dart?

- Desenvolvida pela Google e anunciada em 2011;
- Foi criada para ser uma alternativa ao JavaScript;
- Utilizada para aplicações web, mobile e de servidor;
- Usada principalmente pelo Flutter;

Dart

Legibilidade, eficiência e escalabilidade

foi projetada com foco nessas características

Tipagem estática opcional

determina o tipo caso não seja declarado

Possui um sistema de pacotes robusto

para gerenciamento de dependências

Estruturas de Dados são importantes?

MOSTRAREMOS AGORA A IMPORTÂNCIA DAS ESTRUTURAS DE DADOS

UMA ANALOGIA:
ONDE É MAIS FÁCIL ENCONTRAR RAPIDAMENTE O QUE PRECISA?

Armário bagunçado

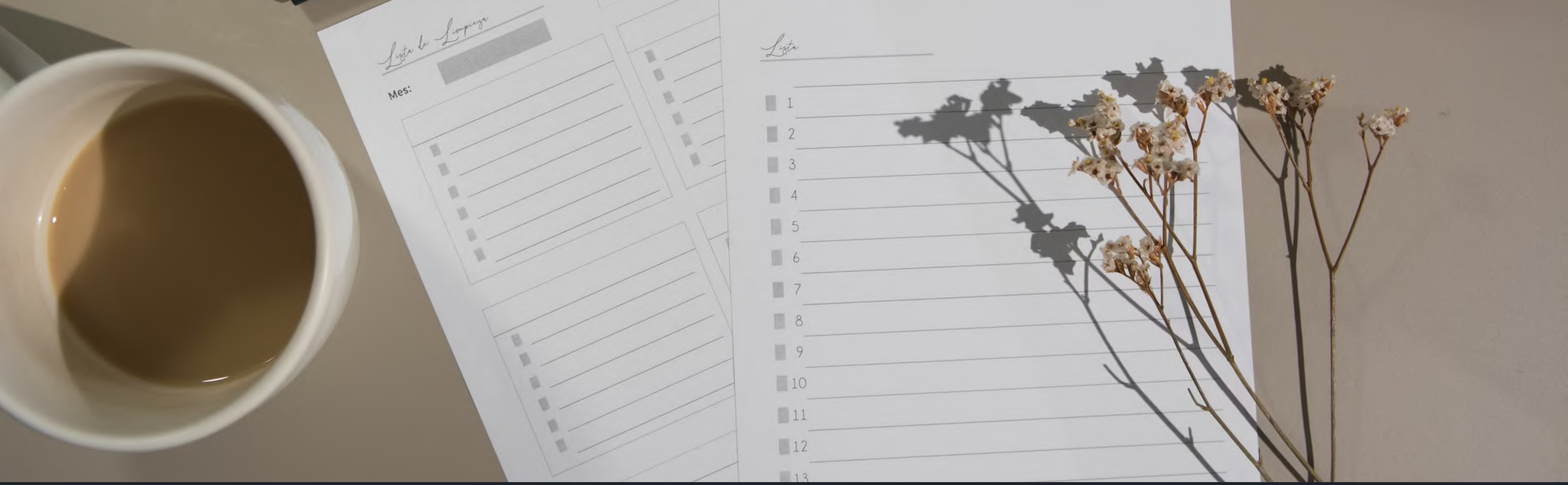
Com o uso incorreto das EDs

Armário organizado

Com o uso correto das EDs

Comportamento das EDs built-in de Dart

VAMOS EXPLORAR O COMPORTAMENTO DAS PRINCIPAIS ESTRUTURAS DE DART

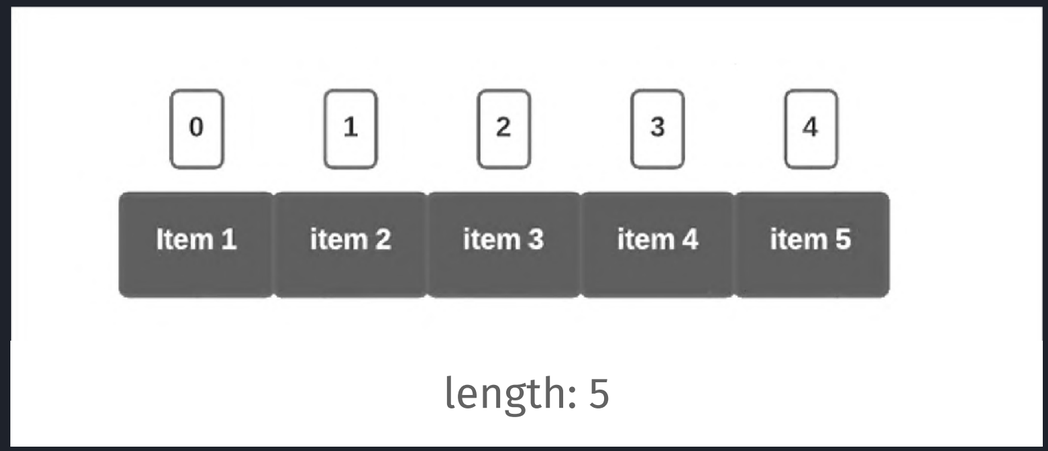


List

É uma sequência ordenada de elementos, permitindo armazenar valores de maneira indexada. Possuem dois tipos: Fixed-length list e Growable list.

List

Dart define List como uma classe abstrata com métodos para acessar e modificar os elementos da coleção por índice. O List também é iterável, isso significa que podemos percorrer os elementos sequencialmente. Além de retornar o número de elementos da coleção através do "length"



List

Podemos criar uma lista usando uma **lista literal** , que é uma lista de valores separados por vírgula entre colchetes. Por exemplo:

```
final people = ['Pablo', 'Manda', 'Megan']; // Cria uma growable list.
```

Ou através da chamada aos métodos da classe List, alguns exemplos:

```
final fixedLengthList = List<int>.filled(5, 0); // Cria uma fixed-length list.

final growableList = List<int>.of([0, 0, 0, 0, 0]); // Cria uma growable list.
```

Capacidade da Lista

As listas do Dart são alocadas com uma **quantidade predeterminada de espaço para seus elementos**.

No caso de uma lista de tamanho fixo - **fixed-length list - essa quantidade não pode ser modificada**, já para as growableLists ao tentar inserir uma quantidade maior que a capacidade a lista irá se reestruturar **copiando todos os elementos** atuais da lista para um espaço novo com o dobro da capacidade atual.

Capacidade da Lista

As listas do Dart são alocadas com uma **quantidade predeterminada de espaço para seus elementos**.

No caso de uma lista de tamanho fixo - **fixed-length list - essa quantidade não pode ser modificada**, já para as growableLists ao tentar inserir uma quantidade maior que a capacidade a lista irá se reestruturar **copiando todos os elementos** atuais da lista para um espaço novo com o dobro da capacidade atual.

Observação: Caso a capacidade inicial seja 0 aumenta para 3

```
int _nextCapacity(int old_capacity) => (old_capacity * 2) | 3;
```

Capacidade da Lista

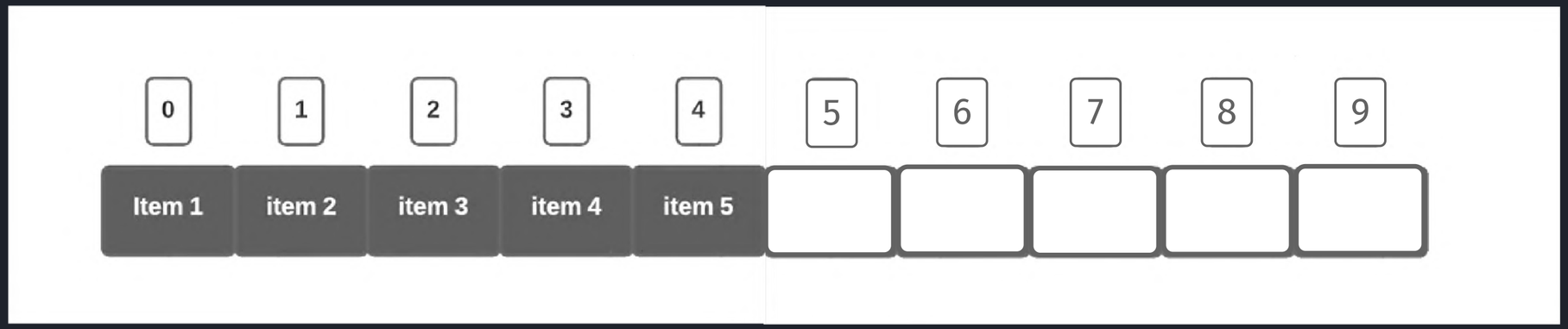
EXEMPLO:

Ao tentar inserir o item 6 nota-se que a lista abaixo já está cheia:

Capacidade da Lista

EXEMPLO:

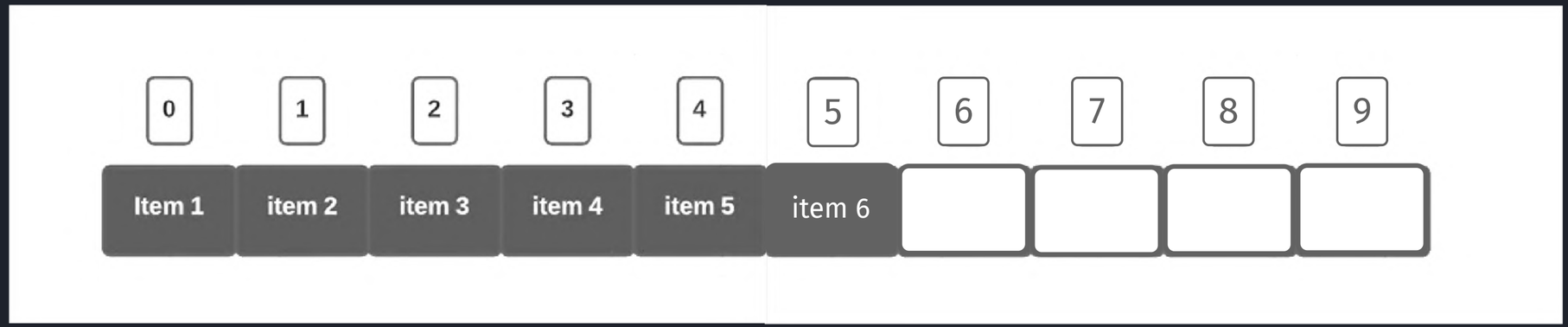
Desse modo, será realizada a cópia de todos os itens da lista para uma outra lista com o dobro da capacidade:



Capacidade da Lista

EXEMPLO:

Só então é possível inserir o item 6 na lista



Implementação de lista de tamanho fixo

```dart
void main() {
  List<int> minhaLista = List.filled(10, 0);

    minhaLista.setAll(0,[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);

  // Você pode acessar e manipular os itens da lista da seguinte maneira:
  print(minhaLista);  // Isso imprimirá a lista inteira.

  // Acessando um item específico pelo índice:
  int item = minhaLista[0];  // Isso obtém o primeiro item da lista (índice
  print(item);

  // Alterando um item específico pelo índice:
  minhaLista[0] = 11;  // Isso substituirá o primeiro item da lista por 11.
  print(minhaLista);

  // Verificando o tamanho da lista:
  int tamanho = minhaLista.length;
  print("Tamanho da lista: $tamanho");

  // Tentativa de exceder tamanho da lista - erro:
  minhaLista.add(12);
  print(minhaLista);
}

```

Run

Console

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
1
[11, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Tamanho da lista: 10
Uncaught Error: Unsupported operation: add
```

Documentation

Implementação de lista de tamanho dinâmico

```dart
void main() {
  // Crie uma lista vazia
  List<String> minhaLista = [];

  // Adicione 5 itens à lista inicialmente
  for (int i = 1; i <= 5; i++) {
    minhaLista.add('Item $i');
  }

  // A partir daqui, você pode aumentar a lista sempre que quiser
  minhaLista.add('Novo Item 6');
  minhaLista.add('Novo Item 7');
  // E assim por diante...

  // Imprima a lista para verificar seu conteúdo
  print(minhaLista);
}
```

Run

Console

```
[Item 1, Item 2, Item 3, Item 4, Item 5, Novo Item 6, Novo Item 7]
```

Documentation

Lista de construtores do List

- List.empty({bool growable = false})

Cria uma nova lista vazia.

- **List.filled**(int length, E fill, {bool growable = false})

Cria uma lista do comprimento fornecido com preenchimento em cada posição.

- **List.from**(Iterable elements, {bool growable = true})

Cria uma lista contendo todos os elementos.

- **List.generate**(int length, E generator(int index), {bool growable = true})

Gera uma lista de valores.

- **List.of**(Iterable<E> elements, {bool growable = true})

Cria uma lista de elementos.

- List.unmodifiable(Iterable elements)

Cria uma lista não modificável contendo todos os elementos.

Lista de métodos do List

- add
- addAll
- any
- asMap
- cast
- clear
- contains
- elementAt
- every
- expand
- fillRange
- firstWhere
- fold
- followedBy
- forEach
- getRange
- indexOf
- indexWhere
- insert
- insertAll
- join
- lastIndexOf
- lastIndexWhere
- lastWhere
- map
- noSuchMethod
- reduce
- remove
- removeAt
- removeLast
- removeRange
- removeWhere
- replaceRange
- retainWhere
- setAll
- setRange
- shuffle
- singleWhere
- skip
- skipWhile
- sort
- sublist
- take
- takeWhile
- toList
- toSet
- toString
- where
- whereType

Complexidade da Lista

Um ponto determinante para a análise de complexidade da lista é onde o item será inserido. Sendo o melhor caso a inserção no final:

```
people.add('Edith'); // O(1)
print(people);
// [Pablo, Manda, Megan, Edith];
```

E o pior caso a inserção no início, com complexidade O(n), pois é necessário realocar os demais itens:

```

people.insert(0, 'Ray'); // [Ray, Pablo, Manda, Megan, Edith]

```

Observação: Complexidade

No entanto, **qualquer inserção, mesmo no final, poderá levar n etapas para ser concluída se for realizada uma cópia da lista, como vimos anteriormente no caso em que o tamanho corresponde a capacidade máxima da lista.

Set

O Set é utilizado para armazenar uma coleção de elementos únicos, sem ordem definida.

Set

Uma coleção de objetos em que cada objeto pode existir apenas uma vez. Por exemplo:

```
var bag = {'Candy', 'Juice', 'Gummy'};
bag.add('Candy'); // tentativa de inserção de 'Candy'
print(bag); // resultado: {Candy, Juice, Gummy}

// observe que o 'Candy' continuou aparecendo apenas uma vez
```

Set

Existem 3 tipos de implementação: HashSet, LinkedHashSet e SplayTreeSet. Cada tipo de implementação possui uma ordenação diferente:

- LinkedHashSet: e mantém a ordem de inserção.
- HashSet: a ordem de iteração depende dos códigos hash dos elementos fornecidos.
- SplayTreeSet: compara os itens entre si e os ordena.

```
final emptySet = <String>{}; // LinkedHashSet - implementação padrão
final letters = HashSet<String>(); // HashSet
final planets = SplayTreeSet<String>((a, b) ⇒ a.compareTo(b)); // SplayTreeSet
```

HashSet x LinkedHashSet x SplayTreeSet

Baseado em tabela Hash e itera na ordem do código hash dos elementos fornecidos.

Baseado em tabela Hash e itera na ordem de inserção independente do código hash dos elementos fornecidos.

Baseado em uma árvore binária de busca balanceada e não aceita itens que não são comparáveis entre si.

Implementação do Set - LinkedHashSet

```dart
void main() {
  // Criando um conjunto vazio
  Set<String> frutas = Set();
  // Adicionando elementos ao conjunto
  frutas.add("maçã");
  frutas.add("banana");
  frutas.add("laranja");
  frutas.add("uva");

  // Exibindo o conjunto
  print("Conjunto de Frutas: $frutas");

  // Verificando se um elemento está no conjunto
  if (frutas.contains("maçã")) {
    print("A maçã está no conjunto de frutas.");
  } else {
    print("A maçã não está no conjunto de frutas.");
  }

  // Removendo um elemento do conjunto
  frutas.remove("banana");
  print("Após remover a banana: $frutas");

  // Iterando pelos elementos do conjunto
  for (var fruta in frutas) {
    print("Uma fruta: $fruta");
  }

  // Tamanho do conjunto
  print("Tamanho do conjunto de frutas: ${frutas.length}");

  // Limpando o conjunto
  frutas.clear();
  print("Conjunto de frutas após limpar: $frutas");
}
```

Run

Console

```
Conjunto de Frutas: {maçã, banana, laranja, uva}
A maçã está no conjunto de frutas.
Após remover a banana: {maçã, laranja, uva}
Uma fruta: maçã
Uma fruta: laranja
Uma fruta: uva
Tamanho do conjunto de frutas: 3
Conjunto de frutas após limpar: {}
```

Documentation

info

Try Pitch

Implementação do Set - Hashset

```dart
import 'dart:collection';
void main() {
  // Criando um HashSet
  HashSet<int> numeros = HashSet<int>();

  // Adicionando elementos ao HashSet
  numeros.add(10);
  numeros.add(5);
  numeros.add(15);
  numeros.add(7);

  // Exibindo os elementos na ordem dos codigos hash
  print("Elementos na ordem hash:");
  for (var numero in numeros) {
    print(numero);
  }

  // Removendo um elemento
  numeros.remove(5);

  // Exibindo os elementos após a remoção
  print("\nElementos após a remoção do número 5:");
  for (var numero in numeros) {
    print(numero);
  }
}
```

Run

Console

```
Elementos na ordem hash:
5
7
10
15

Elementos após a remoção do número 5:
7
10
15
```

Documentation

Comparação: HashSet x LinkedHashset

```dart
import 'dart:collection';
void main() {
  // Criando um LinkedHashSet
  LinkedHashSet<int> numeros = LinkedHashSet<int>();

  // Adicionando elementos ao LinkedHashSet
  numeros.add(10);
  numeros.add(5);
  numeros.add(15);
  numeros.add(7);

  // Exibindo os elementos na ordem em que foram inseridos
  print("Elementos na ordem de inserção:");
  for (var numero in numeros) {
    print(numero);
  }

  // Removendo um elemento
  numeros.remove(5);

  // Exibindo os elementos após a remoção
  print("\nElementos após a remoção do número 5:");
  for (var numero in numeros) {
    print(numero);
  }
}

```

Run

Console

```
Elementos na ordem de inserção:
10
5
15
7

Elementos após a remoção do número 5:
10
15
7
```

Documentation

Implementação do Set - SplayTreeSet

```dart
import 'dart:collection';
void main() {
  // Criando um SplayTreeSet de números inteiros
  SplayTreeSet<int> splayTreeSet = SplayTreeSet<int>();
  // Adicionando elementos ao conjunto
  splayTreeSet.add(5);
  splayTreeSet.add(2);
  splayTreeSet.add(8);
  splayTreeSet.add(1);
  splayTreeSet.add(10);

  // Exibindo os elementos na ordem em que serão acessados (splay)
  print("Elementos na ordem Splay:");
  for (var elemento in splayTreeSet) {
    print(elemento);
  }

  // Verificando se um elemento existe no conjunto
  var elementoProcurado = 8;
  if (splayTreeSet.contains(elementoProcurado)) {
    print("$elementoProcurado está presente no conjunto.");
  } else {
    print("$elementoProcurado não está presente no conjunto.");
  }

  // Removendo um elemento do conjunto
  var elementoRemovido = 2;
  splayTreeSet.remove(elementoRemovido);
  print("Elemento $elementoRemovido removido.");

  // Exibindo os elementos restantes na ordem em que serão acessados (splay)
  print("Elementos restantes na ordem Splay:");
  for (var elemento in splayTreeSet) {
    print(elemento);
  }
}
```

Run

Console

```
Elementos na ordem Splay:
1
2
5
8
10
8 está presente no conjunto.
Elemento 2 removido.
Elementos restantes na ordem Splay:
1
5
8
10
```

Documentation

Lista de construtores do Set

- **Set**()

Cria um Set vazio .

- Set.from(Iterable elements)

Cria um conjunto que contém todos os arquivos elementos.

- Set.identity()

Cria um conjunto de identidade vazio

- Set.of(Iterable<E> elements)

Cria um conjunto de elementos.

- Set.unmodifiable(Iterable<E> elements)

Cria um conjunto não modificável de elementos.

Lista de métodos do Set

- add
- addAll
- any
- cast
- clear
- contains
- containsAll
- difference
- elementAt
- every
- expand
- firstWhere
- fold
- followedBy
- forEach
- intersection
- join
- lastWhere
- lookup
- map
- noSuchMethod
- reduce
- remove
- removeAll
- removeWhere
- retainAll
- retainWhere
- singleWhere
- skip
- skipWhile
- take
- takeWhile
- toList
- toSet
- toString
- union
- where
- whereType

Complexidade do Set

A maioria das operações simples HashSet são feitas em tempo constante (potencialmente amortizado): add , contains , remove e length , desde que os códigos hash dos objetos estejam bem distribuídos.

Por este motivo é importante não modificar o conjunto (adicionar ou remover elementos) enquanto uma operação no conjunto está sendo executada, por exemplo, durante uma chamada para forEach ou containsAll .

COMPARAÇÃO DE CADA TIPO DE IMPLEMENTAÇÃO

HashSet	LinkedHashSet	SplayTreeSet
Possui uma complexidade de O(1) ao inserir, recuperar e remover dados.	Possui uma complexidade de O(1) ao inserir, recuperar e remover dados.	Possui uma complexidade de O(log(n)) ao inserir, recuperar e remover dados.
Permite valor nulo.	Permite valor nulo.	Não permite valor nulo.
Ele usa o código hash e métodos para comparar itens. `equalTo()`	Ele usa o código hash e métodos para comparar itens. `equalTo()`	Ele usa o método para comparar itens. `Comparable.compareTo()`

Map

O Map é uma estrutura chave-valor, que associa chaves a valores correspondentes.

Map

É uma coleção dinâmica e genérica de itens armazenados como um par de chave-valor, o qual você recupera um valor usando sua chave associada.

```
var map = {
    "nome": "Marcos",
    "idade": "10"
};
print(map); // {"nome": "Marcos", "idade": "10"}
```

Map

Existem 3 tipos de implementação: HashMap, LinkedHashMap e SplayTreeMap. Cada tipo de implementação possui uma ordem de iteração diferente:

- HashMap: não está ordenado (a ordem de iteração não é garantida).
- LinkedHashMap: itera na ordem de inserção da chave.
- SplayTreeMap: itera as chaves na ordem da ordenação pré-definida.

```
var map = Map(); // ou var map = {};

print(map); // {}
```

HashMap x LinkedHashMap x SplayTreeMap

Baseado em tabela Hash e a ordem de iteração não é garantida

Baseado em tabela Hash e itera na ordem de inserção.

Baseado em uma árvore binária de busca balanceada e não aceita valores nulos.

Implementação do Map - LinkedHashMap

```dart
void main() {
  // Criando um mapa de frutas e suas quantidades
  Map<String, int> frutas = {
    'maçã': 3,
    'banana': 2,
    'laranja': 5,
  };

  // Acessando valores no mapa
  print('Quantidade de maçãs: ${frutas['maçã']}');
  print('Quantidade de bananas: ${frutas['banana']}');
  print('Quantidade de laranjas: ${frutas['laranja']}');

  // Adicionando uma nova fruta ao mapa
  frutas['uva'] = 4;

  // Atualizando o valor de uma fruta no mapa
  frutas['laranja'] = 6;

  // Removendo uma fruta do mapa
  frutas.remove('banana');

  // Iterando pelo mapa e imprimindo as frutas e suas quantidades
  frutas.forEach((fruta, quantidade) {
    print('$fruta: $quantidade');
  });
}

```

Run

Console

```
Quantidade de maçãs: 3
Quantidade de bananas: 2
Quantidade de laranjas: 5
maçã: 3
laranja: 6
uva: 4
```

Documentation

Implementação do Map - HashMap

DartPad

New Pad | Reset | Format | Install SDK

windy-patter-2973 local edits

Run

```dart
import 'dart:collection';
void main() {
  // Criando um HashMap com chaves do tipo String e valores do tipo int
  HashMap<String, int> hashMap = HashMap();

  hashMap.addAll({
    'Bob': 30,
    'Carol': 28,
    'Alice': 25,
  });

  // Adicionando um novo par chave-valor ao HashMap
  hashMap['David'] = 35;

  // Acessando valores do HashMap
  print('Idade de Alice: ${hashMap['Alice']}');
  print('Idade de David: ${hashMap['David']}');

  // Verificando se uma chave existe no HashMap
  if (hashMap.containsKey('Eve')) {
    print('Eve está no HashMap');
  } else {
    print('Eve não está no HashMap');
  }

  // Removendo um par chave-valor do HashMap
  hashMap.remove('Bob');

  // Iterando sobre as chaves e valores do HashMap
  hashMap.forEach((chave, valor) {
    print('$chave: $valor anos de idade');
  });
}

```

Console

```
Idade de Alice: 25
Idade de David: 35
Eve não está no HashMap
Carol: 28 anos de idade
Alice: 25 anos de idade
David: 35 anos de idade
```

Documentation

Implementação do Map - SplayTreeMap

```dart
import 'dart:collection';

void main() {
  // Criando um SplayTreeMap de inteiros
  SplayTreeMap<int, String> splayTreeMap = SplayTreeMap<int, String>();

  // Adicionando elementos ao mapa
  splayTreeMap[3] = "Três";
  splayTreeMap[1] = "Um";
  splayTreeMap[4] = "Quatro";
  splayTreeMap[2] = "Dois";

  // Imprimindo o mapa após a inserção
  print("SplayTreeMap após a inserção:");
  splayTreeMap.forEach((chave, valor) {
    print("$chave: $valor");
  });

  // Acessando um elemento pelo índice
  int chave = 2;
  String? valor = splayTreeMap[chave];
  print("\nValor associado à chave $chave: $valor");

  // Removendo um elemento do mapa
  chave = 1;
  splayTreeMap.remove(chave);
  print("\nSplayTreeMap após a remoção da chave $chave:");
  splayTreeMap.forEach((chave, valor) {
    print("$chave: $valor");
  });
}
```

Run

Console

```
SplayTreeMap após a inserção:
1: Um
2: Dois
3: Três
4: Quatro

Valor associado à chave 2: Dois

SplayTreeMap após a remoção da chave 1:
2: Dois
3: Três
4: Quatro
```

Documentation

String? [](Object? key)

Lista de construtores do Map

- **Map**()

Cria um LinkedHashMap vazio.

- **Map.from**(Map other)

Cria um LinkedHashMap com as mesmas chaves e valores de outro.

- **Map.fromEntries**(Iterable<MapEntry<K, V>> entries)Set.identity()

Cria um novo mapa e adiciona todas as entradas.

- **Map.fromIterable**(Iterable iterable, {K key(dynamic element)?, V value(dynamic element)?})

Cria uma instância de Map na qual as chaves e os valores são calculados a partir do iterável.

- **Map.fromIterables**(Iterable<K> keys, Iterable<V> values)

Cria um mapa associando as chaves fornecidas aos valores fornecidos.

- Map.identity()

Cria um mapa de identidade com a implementação padrão, LinkedHashMap.

- **Map.of**(Map<K, V> other)

Cria um LinkedHashMap com as mesmas chaves e valores de outro.

- **Map.unmodifiable**(Map other)

Cria um mapa baseado em hash não modificável contendo as entradas de other.

Lista de métodos do Map

- addAll
- addEntries
- cast
- clear
- containsKey
- containsValue
- forEach
- map
- noSuchMethod
- putIfAbsent
- remove
- removeWhere
- toString
- update
- updateAll

Complexidade do Map

Ao contrário das listas, os mapas não precisam se preocupar com a mudança de elementos. A inserção em um mapa sempre leva um **tempo constante O(1)**.

As operações de pesquisa também costumam ser em **tempo constante**, com exceção da implementação com SplayTreeMap que é baseado em uma árvore binária de busca balanceada permite a maioria das operações de entrada única em **tempo logarítmico amortizado O(log(n))**.

Geralmente não é permitido modificar o Map (adicionar ou remover chaves) enquanto uma operação está sendo executada no map, por exemplo, em funções chamadas durante uma chamada forEach ou putIfAbsent.

COMPARAÇÃO DE CADA TIPO DE IMPLEMENTAÇÃO

HashMap	LikedHashMap	SplayTreeMap
Possui uma complexidade de O(1) ao inserir, recuperar e remover dados.	Possui uma complexidade de O(1) ao inserir, recuperar e remover dados.	Possui uma complexidade de O(log(n)) ao inserir, recuperar e remover dados.
Permite valor nulo.	Permite valor nulo.	Não permite valor nulo.

Algumas considerações de Performance

- Caso seja necessário muitas inserções no início o List não é a estrutura de dados mais indicada
- O Map é significativamente mais rápido do que procurar um elemento específico em uma lista, que requer uma caminhada desde o início da lista até o ponto de inserção.
- Os Set são mais adequados para armazenar valores únicos cuja ordem não é essencial.

Aplicações

DAS ESTRUTURAS DE DADOS LIST, SET E MAP EM DART

APLICAÇÕES DE CADA TIPO DE ED

List

- Lista de afazeres;
- Nomes de pessoas em uma fila de atendimentos;
- Armazenamento do histórico de navegação;
- Armazenar músicas em um reprodutor de música;

Set

- Palavras únicas em um texto;
- Categorias dos filmes em um sistema de streaming;
- Status possíveis de pedidos em lojas virtuais;

Map

- Dicionário – a palavra é a chave e o significado é o valor;
- Dados de contatos — o nome do contato pode ser considerado a chave e as informações completas do contato, o valor;
- Sistema bancário - o número da conta é a chave e a entidade ou detalhes da conta é o valor.

Conclusão

Exploramos com detalhes as diferentes estruturas de dados built-in disponíveis em Dart e como elas podem ser aplicadas para otimizar nossos códigos. Ao dominar essas estruturas, você estará bem equipado para enfrentar desafios de programação de maneira mais eficiente e eficaz.

A jornada que percorremos nos levou a compreender a importância de escolher a estrutura de dados certa para cada situação. Através de exemplos práticos, vimos como listas, mapas, conjuntos e outras estruturas podem ser utilizadas para resolver uma variedade de problemas de programação. Ao dominar essas ferramentas, você não apenas economiza tempo, mas também melhora a legibilidade e a manutenção de seu código.

Referências:

- Dart Programming Language - Site oficial
- Data Structures & Algorithms in Dart - Kodeco
- GitHub Repository
- Documentação do Dart
- Dart and Flutter data structures: A comprehensive guide - LogRocket
- Data Structures and Algorithms in Dart - Youtube