Sintaxe e Semântica em Dart
Linguagens de Programação

Lucas Carvalho da Luz, João Pedro Barroso Da Silva Neto, Lucas Vinícius S. G. Coelho

[1]ICEI – Pontifícia Universidade Católica de Minas Gerais (PUC)
Rua Claudio Manuel, 1.162 – Funcionarios – Belo Horizonte – MG – Brazil

{lcluz}@sga.pucminas.br, {lucas.coelho.1324003}@sga.pucminas.br,

{joao.neto.1313048}@sga.pucminas.br

Resumo. *Esse texto e um guia introdutório da linguagem Dart, com o intuito de brevemente explicar os conteudos de aplicações, estrutura de dados, tipos, paradigmas, estruturas de controle, funções e topicos avançados, como concorrência e paralelismo, presentes na linguagem. O repositorio da aplicação pratica esta localizado no Github, no qual o arquivo contendo o codigo: main.dart, e o vıdeo final no YouTube.*

1. Domínios de Aplicação

Dart é uma linguágem primariamente, desenvolvimento de aplicações com foco client-side devido à sua adoção pelo framework de desenvolvimento front-end Flutter. Porém, serve também para o desenvolvimento de aplicações server-side.

2. Tipos de Dados e Sistemas de Tipos

Um benefício dessa checagem estática é a capacidade de encontrar problemas em tempo de compilação, e é possível consertar a maior parte dos erros capturados por ela adicionando anotações de tipo à classes genéricas, como listas (List $< T >$) e mapas (Map $< K, V >$).

2.1. built in types

Dart já vem vários tipos integrados, como:

- int, double
- String
- bool
- List
- Set
- Map
- Runes
- Symbol
- null
- Future, Stream
- Never
- dynamic
- void

Todos eles exceto o Null, são subclasses da classe object

3. Alocação de Memória

Em Dart é possível alocar memória estaticamente com a palavra chave static (só são inicializadas quando são utilizadas), tanto de variáveis como de métodos.

Os métodos estáticos não operam em uma instância, no entanto, eles têm acesso a variáveis estáticas e podem ser passados como parâmetro.

Métodos de instâncias funcionam de forma similar a de métodos estáticos, tirando a parte de serem estáticos, e pertencem a um objeto. Ambos são métodos normais, possuem tipos, podem ser chamados, e podem ser passados como parâmetro como qualquer outro valor.

Objetos instanciados vivem na heap, que é gerenciada pela Dart VM, que é a máquina virtual do Dart que gerencia a execução e a memória do programa.

O coletor de lixo que é utilizado pela Dart VM para fazer a limpeza da memória trata a heap em busca de regiões de memória não mais utilizadas, permitindo a melhor reutilização da memória do programa e reduzindo problemas relacionados ao uso de memória. Esse processo é feito automaticamente pela VM.

4. Principais instruções de controle

Dart possui as seguintes instruções de controle:

- if e else
- for, while e do while
- break e continue
- switch e case
- assert
- try, catch, throw e finally

O funcionamento delas não difere muito do que é visto em C ou C++, com exceção do assert, que é um mecanismo de checagem de consistência usado na fase de desenvolvimentopara notificar a utilização incorreta de alguma classe.

```
// Make sure the variable has a non-null value.
assert(text != null);

// Make sure the value is less than 100.
assert(number < 100);

// Make sure this is an https URL.
assert(urlString.startsWith('https'));
```

5. Sinxtaxe e expressividade

Dart apresenta uma sintaxe semelhante à de Java, JavaScript e C devido a similaridades como a forma de fazer anotações e nomenclatura e estrutura de classes e tipos, nomes de métodos existentes em tipos e classes embutidos na linguagem e conceitos que a linguagem aplica.

Um exemplo disso são métodos utilitários presentes em classes em geral, como map, a forma que é possível declarar valores de um certo tipo utilizando literals bem similares à forma como é feito em JavaScript, e também a sintaxe de declaração de lambdas.

6. Recursividade

Dart suporta recursividade como qualquer outra linguagem que também suporta.

```dart
void main() {
  func(10);
}

void func(int n) {
  if (n <= 0) return;

  return func(--n);
}
```

```dart
class Coisa {
  final Coisa? coisa;

  const Coisa(this.coisa);
}
```

7. Concorrência

As palavras chave async e await têm significado especial na linguagem Dart, e permitem uma maneira declarativa de definir funções assíncronas.

Em termos práticos, é possível dar await em uma função que retorne uma Future, mas somente dentro de uma função que seja async. Para definir uma função assíncrona, adicione a palavra chave async antes da declaração do corpo da função.

```dart
String createOrderMessage() {
  var order = fetchUserOrder();
  return 'Your order is: $order';
}

Future<String> fetchUserOrder() =>
    // Imagine that this function is
    // more complex and slow.
    Future.delayed(
      const Duration(seconds: 2),
      () => 'Large Latte',
    );

void main() {
  print('Fetching user order...');
  print(createOrderMessage());
}
```

Quanto à concorrência, muitos programas desenvolvidos em Dart utilizam de somente um isolate, que atua como o principal processo que roda a aplicação, mas é possível criar vários isolates, permitindo a execução de código em paralelo utilizando múltiplos cores.

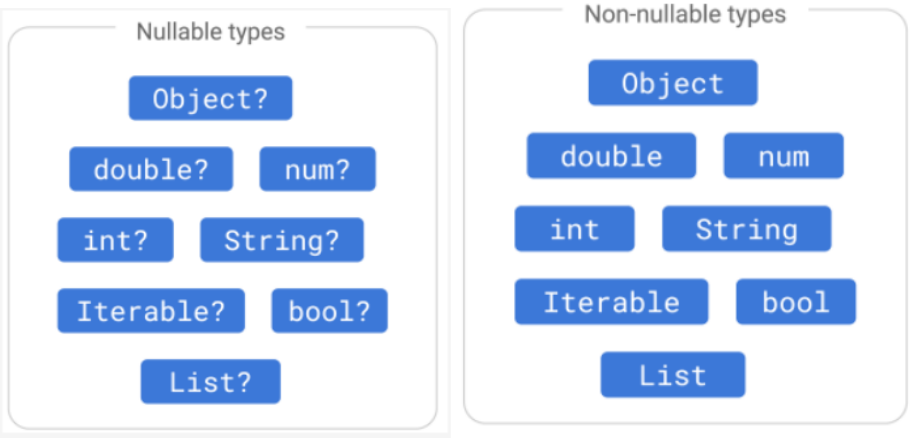
8. Paradigmas de programação presentes

Dart é uma linguagem orientada a objetos e apresenta aspectos dos paradigmas funcional, orientado a objetos e imperativo. Apresenta conceitos de orientação a objeto e mixins. Cada objeto é uma instância de uma classe, e todas as classes, exceto Null, descendem de Object. Quando você chama um método, você o invoca em um objeto: o método tem acesso às funções e dados desse objeto

9. Null safety

Em Dart as variáveis são não anuláveis, entretanto é possível adicionar ? para aceitar valores nulos.

```
var i = 42;
String name = getFileName();
final b = Foo();
ou
int? nullInt = null;
```



10. Parâmetros

Dart tem 2 tipos de parâmetros: posicionais e nomeados. Parâmetros posicionais são o tipo de parâmetro que são comumente vistos em outras linguagens:

```
int sumUp(int a, int b, int c) {
  return a + b + c;
}
//
  int total = sumUp(1, 2, 3);
```

É possível fazer com que parâmetros sejam opcionais colocando-os entre colchetes, e eles têm sempre que vir por último na lista de parâmetros:

```
int sumUpToFive(int a, [int? b, int? c, int? d, int? e]) {
  int sum = a;
  if (b != null) sum += b;
  if (c != null) sum += c;
```

```
  if (d != null) sum += d;
  if (e != null) sum += e;
  return sum;
}
//
  int total = sumUpToFive(1, 2);
  int otherTotal = sumUpToFive(1, 2, 3, 4, 5);
```

Utilizando chaves, é possível definir parâmetros que funcionam a partir de nomes.

```
void printName(String firstName, String lastName, {required
   String suffix}) {
  print('$firstName $lastName $suffix');
}
//
  printName('Avinash', 'Gupta');
  printName('Poshmeister', 'Moneybuckets', suffix: 'IV');
```

É possível também fazer estes parâmetros serem obrigatórios utilizando a palavra chave required.

```
void printName(String firstName, String lastName, {String?
   suffix}) {
  print('$firstName $lastName ${suffix ??       }');
}
//
  printName('Avinash', 'Gupta');
  printName('Poshmeister', 'Moneybuckets', suffix: 'IV');
```

11. Modificadores de acesso

Em Dart, não existem identificadores de acesso como public ou private. Em vez de utilizar essas palavras chave, a linguagem utiliza de uma outra lógica, que envolve a utilização do caractere (_) antes do nome da variável, o que torna ela private - caso contrário ela é public.

Essa lógica também vale para a exportação de membros entre arquivos: um identificador iniciado com (_) não poderá ser importado por outros arquivos.

12. Construtores

Em Dart é possível usar atalhos para criação de construtores, como o this.propertyName para declarar o construtor, e criação de múltiplos construtores

```
class MyColor {
  int red;
  int green;
  int blue;

  MyColor(this.red, this.green, this.blue);
  MyColor.black()
      : red = 0,
```

```
        green = 0,
        blue= 0;
}
final color = MyColor(80, 80, 128);
final blackColor = MyColor.black();
```

Em alguns casos é possível também utilizar o required para impedir valores nulos ou colocar um valor default

```
 MyColor({required this.red, required this.green, required this.
    blue});
ou
  MyColor([this.red = 0, this.green = 0, this.blue = 0]);
```

13. Plataformas

Flutter tem suporte a várias plataformas como a Web, Android, iOS, Windows e Linux. O que isso significa é que com ele é possível fazer aplicativos que rodam no navegador, em celulares, e como aplicações desktop também.

References

https://dart.dev/;

https://youtu.be/GA9tNAp1itE/;

https://github.com/LucasVinicius314/lp-linguagem;