Linguagem Dart - Overview

Dart Básico

Blocos de construção fundamentais da linguagem Dart

O que é Dart?

Dart é uma linguagem de programação orientada a objetos desenvolvida pelo Google. Embora não seja tecnicamente restrito, ele é usado principalmente para criar interfaces de usuário front-end para a web (com AngularDart ou Flutter para Web) e aplicativos móveis (Flutter).

Ele está em desenvolvimento ativo, compilado para código de máquina nativo (quando usado para construir aplicativos móveis), inspirado por recursos modernos de outras linguagens de programação (principalmente Java, JavaScript, C #) e fortemente tipado.

Como já foi mencionado, o Dart é uma linguagem compilada. Isso significa que seu código não é executado como você o escreveu, mas em vez disso, um compilador o analisa e transforma (em código de máquina).

O que são "Tipos"?

Nem todas as linguagens de programação são fortemente tipadas - mas Dart é. O que isso significa?

Cada valor que você usa em seu programa (por exemplo, alguma entrada do usuário que você está salvando) tem um tipo - pode ser "texto", por exemplo. No Dart (e em praticamente todas as outras linguagens de programação), isso não seria chamado de "texto", mas de "string".

Por outro lado, se você está trabalhando com a idade de um usuário, pode estar usando um número sem casas decimais - um valor denominado "inteiro". Números com casas decimais são chamados de "duplos" (ou "float" - em outras linguagens de programação).

Esses são alguns tipos básicos - o Dart tem muito mais do que esses tipos básicos. Você aprenderá muito mais sobre esses (e todos os outros tipos importantes) ao longo do curso.

No momento, esses são os tipos que você deve ter em mente:

Linguagem Dart - Overview

Tipo	Exemplo	Mais informações
String	String meuNome = “Máx”	Contém texto. Você pode usar aspas simples ou duplas - apenas seja consistente depois de fazer sua escolha.
num, int, duplo	idade int = 30 preço duplo = 9,99	Num refere-se a “número” - existem dois tipos de números no Dart: Integer (números sem casa decimal) e double (números com casas decimais)
Objeto	Person = Person ()	Geralmente, tudo no Dart é um objeto - até mesmo um número inteiro. Mas os objetos também podem ser mais complexos, veja abaixo.

Também é importante ter em mente que CADA valor no Dart é um objeto. Mais sobre isso pode ser encontrado abaixo

Linguagem Dart - Overview

Variáveis & Funções

Em seus programas, você normalmente precisa armazenar alguns valores. Não necessariamente em um banco de dados ou arquivo, mas na memória. Pode ser necessário armazenar algum resultado intermediário, a entrada de um usuário antes de processá-lo ou alguma informação sobre o widget Flutter (por exemplo, "Ele deve ser exibido na tela?").

Você armazena essas informações (= valores) nas chamadas variáveis.

Variáveis são contêineres de dados - eles têm nomes e armazenam valores de qualquer tipo. Por exemplo:

```
var minhaIdade = 30;
```

é uma variável que armazena um número inteiro (**int**) de valor **30**.

A palavra-chave var informa ao Dart que minhaIdade é uma variável. Como alternativa a var, você também pode usar o nome do tipo - além de informar ao Dart sobre a variável, você também "informaria ao Dart" sobre o tipo de dados armazenados na variável:

```
int minhaIdade = 30;
```

No entanto, o Dart tem um recurso chamado "inferência de tipo".

Isso significa que Dart é muito inteligente para inferir tipos de valores. Se você criou a variável com var, o Dart ainda é capaz de inferir que myAge é do tipo int porque você inicializa a variável (ou seja, você atribui um valor desde o início) com um valor inteiro. Por causa dessa inferência de tipo embutida, é considerado uma boa prática NÃO definir explicitamente o tipo, mas, em vez disso, usar var ao definir variáveis. Portanto, este snippet seria preferido:

```
var minhaIdade = 30;
```

Isso muda se você criar uma variável sem um valor inicial - então, você deve informar ao Dart sobre que tipo de dados você planeja salvar lá:

```
int minhaIdade;
```

```
minhaIdade = 30;
```

Além das variáveis, outro recurso central de QUALQUER linguagem de programação são as "funções". As funções permitem que você "terceirize" o código em "fragmentos de código reutilizáveis". Aqui está um exemplo:

```
var preco1 = 9.99;

var preco2 = 10.99;

var total = preco1 + preco2;

var num1 = 10;

var num2 = 45;

var totalNum = num1 + num2;

minhaIdade = 30;
```

Neste exemplo, temos a mesma lógica de adicionar dois números em dois locais diferentes de nosso código. Em vez de nos repetir, seria ótimo colocar essa lógica em uma função que podemos chamar sempre que quisermos. Aqui está o snippet alterado, usando uma função:

```
num adicionarNum(num n1, num n2) {
// use num como um tipo porque ele trabalharã com  int e double
      return n1 + n2;
}
var preco1 = 9.99;
var preco2 = 10.99;
var total = adicionarNum(preco1, preco2);
var num1 = 10;
var num2 = 45;
var totalNums = adicionarNum(num1, num2);
```

Linguagem Dart - Overview

Usamos a linguagem Dart ao escrever Flutter, mas Dart não é muito popular (ainda). A maioria dos desenvolvedores salta direto para o Flutter sem nenhum conhecimento prévio da linguagem. Caso seja você, gostaríamos de obter uma ajudinha.

Nestes documentos, não estamos fazendo nenhuma tentativa de ensinar a você tudo sobre o Dart. Nosso objetivo aqui é dar a você apenas o suficiente para ser eficaz ao escrever Flutter. Portanto, este apêndice é breve e direto ao ponto. Estamos lidando apenas com coisas que, de outra forma, teriam deixado você mais lento enquanto escrevia Flutter.

Um exemplo disso é o tipo de dado rúnico. Recurso Dart super legal e inovador, mas raramente usado com Flutter, então o omitimos. Por favor, tente ser tolerante conosco se omitimos seu recurso favorito. Nós não esquecemos. Nós apenas decidimos que não era tão importante quanto você achou que deveria ser. Por favor, nos perdoe.

O que é Dart?

Dart é uma linguagem de programação procedural compilada, estaticamente tipada e orientada a objetos. Ele tem uma estrutura muito comum, muito parecida com outras linguagens OO, tornando-o extremamente fácil de aprender para pessoas que têm experiência com Java, C #, C ++ ou outras linguagens OO, semelhantes a C. E adiciona alguns recursos que os desenvolvedores dessas outras linguagens não esperariam, mas são muito interessantes e tornam a linguagem mais do que elegante.

Visão Geral da Linguagem Dart

Diante de tudo isso, este documento está organizado em duas seções:

• Recursos esperados - uma referência rápida (também conhecida como "cheatsheet") dos recursos principais, o mínimo do que você precisa saber sobre o Flutter. Você deve percorrer esta seção na velocidade da luz.

• Recursos inesperados - são coisas que podem ser uma surpresa para os desenvolvedores que trabalham em linguagens OO tradicionais. Como o Dart se afasta da tradição nessas áreas, achamos melhor explicá-las resumidamente - muito brevemente.

Linguagem Dart - Overview

Expected features – Dart Cheatsheet

Esta referência rápida pressupõe que você é um desenvolvedor OO experiente e ignora as coisas que seriam dolorosamente óbvias para você. Para uma visão mais aprofundada e detalhada do Dart, visite https://dart.dev/guides/language/language-tour.

Data types

```
int x = 10;         // Integers
double y = 2.0;     // IEEE754 floating point numbers
bool z = true;      // Booleans
String s = "hello"; // Strings
dynamic d;          // Dynamic variables can change types at any time. Use sparingly!
d = x;              //
d = y;
d = z;
```

Arrays/lists

```
// Square brackets means a list/array

// In Dart, arrays and lists are the same thing.

List<dynamic> list = [1, "two", 3];

// Optional angle brackets show the type - Dart supports Generics

// How to iterate a list

for (var d in list) {

  print(d);

}

// Another way to iterate a list

list.forEach((d) => print(d));

// Both of these would print "1", then "two", then "3"
```

Conditional expressions

```
// Traditional if/else statement

int x = 10;
```

Linguagem Dart - Overview

```
if (x < 100) {
  print('Yes');
} else {
  print('No');
}
// Would print "Yes"
// Dart also supports ternaries
String response = (x < 100) ? 'Yes' : 'No';
// If name is set, use it. Otherwise use 'No name given'
String name;
String res = name ?? 'No name given';
//the "Elvis" operator. If the object is non-null, evaluate
//the property. Prevents null exceptions from throwing.
print(name?.length);
```

Looping

```
// A for loop
for (int i=1 ; i<10 ; i++) {
  print(i);
}
// Would print 1 thru 9
// A while loop
int i=1;
while(i<10) {
  print(i++);
}
// Would print 1 thru 9
```

Linguagem Dart - Overview

Classes

```
class Name {
  String first;
  String last;
  String suffix;
}
class Person {
  // Classes have properties
  int id;
  Name name;    // Another class can be used as a type
  String email;
  String phone;
  // Classes have methods
  void save() {
    // Write to a database somehow.
  }
}
```

Class constructors

```
class Person {
  Name name;
  // Typical constructor
  Person() {
    name = Name(); name.first = ""; name.last = "";
  }
}
```

Linguagem Dart - Overview

Coisas inesperadas sobre Dart

Os recursos anteriores do Dart não surpreenderam nenhum desenvolvedor OO experiente, mas o Dart tem alguns recursos muito legais que são únicos. Vamos cobrir isso a seguir, mas como eles são menos familiares, vamos pegar apenas uma ou duas frases para cada um e explicá-los brevemente antes de dar a você um exemplo de código.

Type inference

Se eu dissesse “x = 10,0”, que tipo de dados você acha que x é? Duplo? E como você soube? Porque você olhou à direita do sinal de igual e inferiu seu tipo com base no valor atribuído a ele. Dart também pode fazer isso. Se você usar a palavra-chave var em vez de um tipo de dados, o Dart inferirá que tipo é e atribuirá esse tipo:

```
var  i = 10;        // i is now defined as an int.

i = 12;             // Works, because 12 is an int.

i = "twelve";       // No! "twelve" is a String and not an int.

var str = "ten";    // str is now defined as a String.

str = "a million";  // Yep, works great.

Str = 1000000.0;    // Nope! 1000000.0 is a double, not a string.
```

Isso geralmente é confundido com dinâmico. Dinâmico pode conter qualquer tipo de dados e pode mudar em tempo de execução. Var é fortemente tipado estaticamente. final e const final e const são modificadores de variáveis Dart:

```
final int x = 10;

const double y = 2.0;
```

Ambos significam que, uma vez atribuído, o valor não pode mudar. Mas const vai um pouco além - o valor é definido no momento da compilação e, portanto, incorporado ao pacote de instalação. Final significa que a variável não pode ser reatribuída. Isso não significa que não pode mudar. Por exemplo, isso é permitido:

```
final Employee e = Employee();

e.employer = "The Bluth Company";
```

e mudou, mas não foi reatribuído, então está tudo bem. Isso, no entanto, não é permitido:

```
const Employee e = Employee();
```

const não é permitido porque esta classe particular tem propriedades que podem mudar potencialmente em tempo de execução. final marca uma variável como imutável, mas const marca um valor como imutável.

Então, em resumo

• dinâmico - pode armazenar qualquer tipo de dados. O tipo de dados pode mudar a qualquer momento.

• var - o tipo de dados é inferido do valor no lado direito de "=". O tipo de dados não muda.

• final - A variável, uma vez definida, não pode ser reatribuída.

• const - O valor é definido em tempo de compilação, não em tempo de execução.

Variables are initialized to null

O tipo de dados padrão para a maioria das variáveis é nulo. O valor de retorno padrão de uma função é nulo:

int x;

double y;

bool z;

String s;

dynamic d;

Todos os dados anteriores são nulos, pois ainda não foram atribuídos um valor.

String interpolation with $

A interpolação evita que os desenvolvedores gravem concatenações de strings. Esta ...

String fullName = '$first $last, $suffix';

... é efetivamente a mesma coisa que isso ...

String fullName = first + ' ' + last + ', ' + suffix;

Quando a variável faz parte de um mapa ou objeto, o compilador pode ficar confuso, então você deve envolver a interpolação entre chaves.

String fullName = '${name['first']} ${name['last']}';

Linguagem Dart - Overview

Multiline strings

Você pode criar strings de várias linhas com três aspas simples ou duplas:

String introduction = """

Agora a história de uma família rica que perdeu tudo E o filho que não teve escolha a não ser mantê-los todos juntos.

""";

Spread operator

O operador "..." espalhará os elementos de uma matriz, achatando-os. Isso será muito familiar para os desenvolvedores de JavaScript:

List fiveTo10 = [5, 6, 7, 8, 9, 10,];

// Spreading the inner array with "...":

List numbers = [1, 2, 3, 4, ...fiveTo10, 11, 12];

// numbers now has [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

Map<foo, bar>

Os mapas são como um hash ou dicionário. Eles são apenas um objeto com um conjunto de pares de valores-chave. As chaves e valores podem ser de qualquer tipo:

// Você define o valor de um mapa com chaves:

Map<String, dynamic> person = {

  "first": "George",

  "last": "Bluth",

  "dob": DateTime.parse("1972-07-16"),

  "email": "amazingGob@gmail.com",

};

Os colchetes angulares em um mapa definem os tipos de dados das chaves e valores. Eles não são obrigatórios, mas são uma boa prática. Você faz referência a um membro do mapa com colchetes:

String introduction = person['first'] + " was born "+

person['dob'].toString();

Linguagem Dart - Overview

Functions are objects

Assim como no JavaScript, as funções são objetos de primeira classe. Eles podem ser passados como dados, retornados de uma função, passados para uma função como um parâmetro ou configurados como iguais a uma variável. Você pode fazer quase tudo com uma função que você pode fazer com um objeto em Java ou C #:

```
Function sayHi = (String name) => print('Hello, ' + name);

// Você pode passar sayHi por aí como dados; é um objeto!

Function meToo = sayHi;

meToo("Tobias");
```

Big arrow/Fat arrow

No exemplo anterior, também vimos a sintaxe da seta grande. Quando você tem uma função que retorna um valor em uma linha de código, pode colocar esse valor retornado no lado direito de um “=>” e a lista de argumentos no lado esquerdo. São todos iguais:

```
int triple(int val) {
  return val * 3;
}

Function triple = (int val) {
  return val * 3;
};

Function triple = (int val) => val * 3;
```

A seta gorda é apenas açúcar sintático, permitindo que os desenvolvedores sejam mais expressivos com menos código.

Named function parameters

Parâmetros posicionais são ótimos, mas podem ser menos propensos a erros (embora mais digitação) ter parâmetros nomeados. Em vez de chamar uma função como esta:

```
sendEmail('ceo@bluthcompany.com','Popcorn in the breakroom');
```

Você pode chamá-lo assim:

```
sendEmail(subject:'Popcorn in the breakroom',  toAddress:'ceo@bluthcompany.com');
```

Agora, a ordem dos parâmetros não é importante. Aqui está como você escreveria a função para usar parâmetros nomeados. Observe as chaves:

```
void sendEmail({String toAddress, String subject}) {
  // send the email here
}
```

Parâmetros nomeados também funcionam bem com construtores de classe, onde são muito comumente usados no Flutter:

```
class Person {
  Name name;
  // Named parameters
  Person({String firstName, String lastName}) {
    name = Name()..first=firstName..last=lastName;
  }
}
```

Omitting “new” and “this.”

No Dart, é possível - e encorajado - evitar o uso da nova palavra-chave ao instanciar uma classe:

```
// No. Avoid.
Person p = new Person();
// Yes
Person p = Person();
```

Da mesma forma, dentro de uma aula, o uso de “isso”. referir-se aos membros da classe não só é desnecessário porque é presumido, mas também é desencorajado. O código é mais curto e mais limpo:

```
class Name {
  String first;
  String last;
```

```
  String suffix;
  String getFullName() {
    // No. Avoid "this.":
    String full=this.first+" "+this.last+", "+this.suffix;
    // Better.
    String full=first+" "+last+", "+suffix;
    return full;
  }
}
```

Class constructor parameter shorthand

Apenas uma maneira mais curta de escrever suas classes Dart que recebem parâmetros. Quando você escreve o construtor para receber “this.something” e tem uma propriedade com escopo de classe com o mesmo nome, o compilador escreve as atribuições para que você não precise:

```
class Person {
      String e-mail, phone;
      // Os parâmetros são atribuídos às propriedades automaticamente
      // porque os parâmetros dizem "isso".
      Person(this.email, this.phone) {}
}
```

O código anterior é equivalente a

```
class Person {
  String email;
  String phone;
  Person(String email, String phone) {
    this.email = email;
    this.phone = phone;
  }
```

```
}
```

Private class members

O Dart não usa modificadores de visibilidade de classe como público, privado, protegido, pacote ou amigo como outras linguagens OO. Todos os membros são públicos por padrão. Para tornar um membro da turma privado, coloque um sublinhado antes do nome:

```
class Person {
  int id;
  String email;
  String phone;
  String _password;
  set password(String value) {
    _password = value;
  }
  String get hashedPassword {
    return sha512.convert(utf8.encode(_password)).toString();
  }
}
```

Nesse exemplo, id, email e telefone são públicos. _password é privado porque o primeiro caractere do nome é "_", o caractere de sublinhado.

Mixins

Mixins são cestas de propriedades e métodos que podem ser adicionados a qualquer classe. Eles se parecem com classes, mas não podem ser instanciados:

```
mixin Employment {
  String employer;
  String businessPhone;
  void callBoss() {
    print('Calling my boss');
```

```
  }
}
```

Um mixin é adicionado a uma classe quando usa a palavra-chave “with”:

```
class Employee extends Person with Employment {
  String position;
}
```

Esta classe Employee agora tem as propriedades Employee e businessPhone e um método callBoss ():

```
Employee e = Employee();
e.employer = "The Bluth Company";
e.callBoss();        // An employee can call its boss.
```

Dart, como Java e C #, suporta apenas herança única. Uma classe só pode estender uma coisa. Mas os membros do mixin são adicionados a uma classe, então qualquer classe pode implementar vários mixins e um mixin pode ser usado em várias outras classes.

The cascade operator (..)

Quando você vir dois pontos, significa "retornar esta classe, mas antes de fazer algo com uma propriedade ou método". Podemos fazer isso

```
Person p = Person()..id=100..email='gob@bluth.com'..save();
```

que seria uma forma mais concisa de escrever

```
Person p = Person();
p.id=100;
p.email='gob@bluth.com';
p.save();
```

Linguagem Dart - Overview

Sem sobrecarga de Métodos

O Dart não suporta métodos de sobrecarga. Isso inclui construtores.

Named constructors

Visto que não podemos ter construtores sobrecarregados, o Dart oferece uma maneira diferente de fazer essencialmente a mesma coisa. Eles são chamados de construtores nomeados e acontecem quando você escreve um construtor típico, mas você adiciona um ponto e outro nome:

```
class Person {
  // Typical constructor
  Person() {
    name = Name()..first=""..last="";
  }
// Um construtor nomeado
  Person.withName({String firstName, String lastName}) {
    name = Name()
      ..first = firstName
      ..last = lastName;
  }
// Outro construtor nomeado
  Person.byId(int id) {
      // Talvez vá buscar em um serviço pelo id fornecido
  }
}
```

E para usar esses construtores nomeados, faça o seguinte:

```
Person p = Person();  // p seria uma pessoa com nome e sobrenome em branco
Person p1 = Person.withName(firstName:"Lindsay",lastName:"Fünke");
// p1 tem o primeiro nome "Lindsay" e o sobrenome "Funke"
Person p3 = Person.byId(100); // p3 seria obtido com base na id de 100
```