Dart

TD (Treball Dirigit) de LP (Llenguatges de Programació)

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Curs 2021-2022 Primavera

Hash: 11728

Índex

1. Introducció

Dart és un llenguatge de programació *open source*, desenvolupat per Google, presentat en la conferència GOTO a Dinamarca l'octubre de l'any 2011, però no va ser fins al novembre del 2013 que va ser lliberat. El projecte va ser fundat per Lars Bak i Kasper Lund.

Dart és un llenguatge orientat a objectes (*object-oriented*)[1], basada en classes (*class-based*)[2], recollida d'escombriaires (*garbage-collected*)[3] amb una sintaxi estil C (*C-style sintax*).

Dart és un llenguatge que admet múltiples **paradigmes**, els quals:

- Orientat a objectes, com hem dit abans
- Funcional: els programes es construeixen aplicant i composant funcions, on tenim llenguatges com: Haskell, JavaScript, Scala, Elrang...
- Imperatiu: s'utilitzen declaracions per canviar l'estat del programa, on tenim llenguatges com: Fortran, Java, C#, C++, COBOL, Assembler, Python, Ruby, ALGOL, Pascal...
- Reflexiu: és la capacitat d'un programa d'examinar, i eventualment modificar, les seves estructures internes d'alt nivell durant la seva execució. Llenguatges que també suporten aquest paradigma: Python, Ruby, Java, Objective-C...

2. Historia, propòsit i relació amb llenguatges similars

Inicialment, Dart es va presentar com una alternativa a *JavaScript* en termes de programació web, ja que els seus creadors creien que *JavaScript* no era adequat per dur a terme aplicacions web a gran escala. Molts van criticar la iniciativa de Dart per fragmentar la web, donat que la idea original era fer que el navegador Google Chrome pogués interpretar-lo nativament. Però l'any 2015, amb la versió 1.9, Dart va abandonar aquesta idea i es va focalitzar en compilar Dart en *JavaScript*.

Al juny de l'any 2014, Dart esdevé un estàndard d'Ecma[4], fent que Dart pugui ser compilat segons l'estàndard *JavaScript*, de manera que esdevé compatible en qualsevol navegador web modern.

Finalment, amb la versió 2.6, Dart **permet compilar nativament** en les plataformes d'escriptori Linux, MacOs i Windows, abans només era possible crear eines a dispositius Android o iOS.

---

[1] Orientat a objectes (*object-oriented*): és un paradigma de programació en el que el programa es divideix en estructures abstractes anomenades classes que proporcionen dades i funcions.
[2] Basada en classes (*class-base*): és un estil de programació orientat a objectes en el que l'herència es produeix al extendre les característiques (com tipus, funcions i variables) d'una classe a una altra.
[3] Recollida d'escombriaires (*garbage-collected*): és un mecanisme implícit de gestió de memòria que utiliten alguns compiladors i intèrprets de llenguatge de programació, s'utilitza per evitar la gestió manual de memòria en el llenguatge.
[4] Ecma: és una organització dedicada a l'estandardització de sistemes d'informació fundada el 1961.

Dart és un llenguatge compilat, com C# i Java, però també és un llenguatge script com Python i JavaScript, per tant, tenim les característiques d'ambdues parts. A més que Dart és transplia[5] a JavaScript.

3. Principals aplicacions

Dart és un llenguatge de propòsit general, s'utilitza per:

- Aplicacions web
- Servidors
- Aplicacions de consola
- Aplicacions mòbils.

4. Frameworks

Flutter és un Framework de Dart que permet desenvolupar aplicacions multiplataforma per Android iOs, Linux, MacOS, Windows, Google Fuchsia i web emprant un únic codi base amb rendiment natiu i amb una sèrie de característiques avançades com:

- Stateful Hot Reload: que ens permet realitzar canvis en el nostre codi i que aquests es mostrin instantàniament a la vista prèvia de l'aplicació
- Compilació JIT (Just In Time) i OAT (Operational Acceptance Testing): JIT és una manera d'executar que permet compilar codi en durant l'execució i no prèviament l'execució, mitjançant DART VM, una màquina virtual. Però quan el programa ja està desenvolupat i volem preparar el programa per fer el *release* el codi es compila amb anticipació (OAT), convertint el codi Dart en natiu.
- Skia: és una biblioteca gràfica de renderitzat de gràfics en 2D que Flutter utilitza.

Com a curiositat, cada vegada més companyies utilitzen Flutter, tenim grans companyies com BMW, Ebay, Alibaba, Google, entre d'altres.

Tenim Frameworks per tota classe de projectes:

- Server Frameworks: Jaguar, Start, Shelf, Vane...
- Client Web App Frameworks: AngularDart, MDL/Dart, OverReact...
- Videojocs: Flame, StageCL, DartRocker, Pixi Dart...
- Animacions: Universal Tween Engine i Spine Dart
- Bases de dades: Postgres, PostresSQL...
- ...

5. Característiques a mencionar

- Tipus: són opcionals i no fa falta definir-los
- Genèrics: permet no definir el tipus d'objecte dins un contenidor (`List()` en lloc de `List<String>()`).

[5] Transpliador: és un tipus especial de compilador que tradueix d'un llenguatge font a un altre amb un nivell d'abstracció similar.

- Manipular el DOM (Document Object Model): permet elements i nodes com a objectes.
- Isolates: permet executar codi en threads o processos separats, utilitzant Dart VM.
- Dart VM: és una col·lecció de components per executar Dart de forma nativa.

6. Exemples de codi

A continuació, trobem una breu introducció al llenguatge Dart.

6.1. Declaració de variables

A continuació podem veure un exemple de creació i inicialització d'una variable:

```
var name = 'Bob';
```

Les variables emmagatzemem referències, és a dir, la variable "name" conté una referència a un Objecte String amb valor "Bob". En el cas anterior, queda clar que la variable es un String, però el cas de voler canviar el tipus de variable, és pot especificar el tipus d'objecte que volem, o simplement, podem declarar que és un objecte.

```
String name1 = 'Bob';

Object name2 = 'Bob';
```

Totes les variables, incloent-hi les numèriques, si no s'inicialitzen tenen valor inicial *null*, això és pel fet que totes les variables es tracten com a objectes. Això s'aplica sempre que no anul·lem el *null safety* nosaltres, ho podem fer marcant la variable com *late*, però això només ho podem fer sempre que estem segurs que la variable la inicialitzem en algun moment abans d'utilitzar-la.

```
late String description;

void main() {
  description = 'Feijoada!';
  print(description);
}
```

Sempre que no tinguem intenció de canviar de valor una variable, hem d'usar *final i const*, en lloc de *var*. Una variable final només pot ser inicialitzada una vegada; una variable *const* és una constant en temps de compilació.

```
final name = 'Bob'; // Without a type annotation
```

```
  name = 'Alice'; // Error: a final variable can only be
set once.

  const bar = 1000000; // Unit of pressure (dynes/cm2)
  const double atm = 1.01325 * bar; // Standard atmosphere
```

Tipus incorporats:

- Números (int, double)
- Strings (String)
- Booleans (bool)
- Llistes (List)
- Sets (Set)
- Maps (Map)
- Runes o caràcters (Runes)
- Símbols (Symbol)
- Valor nul (Null)

6.2. Funcions

Com sol ser costum tenim la funció `main()` que serveix com a punt d'entrada a l'aplicació.

```
void main() {
  print('Hello, World!');
}
```

Exemple d'implementació d'una funció:

```
bool nom_funcio(int param) {
    return param != null;
}
```

També podem ometre el tipus de valor que ha de retornar.

```
nom_funcio(int param) {
    return param != null;
}
```

Per les funcions que contenen només una expressió, podem utilitzar una notació més breu.

```
bool nom_funcio(int param) => return param != null;
```

També podem fer ús dels "named parametres", són paràmetres opcionals, a no ser que es marquin com "required".

```
enableFlags(bold: true, hidden: false);

/// Sets the [bold] and [hidden] flags ...
void enableFlags({bool? bold, bool? hidden}) {...}

/// Uses required to indicate that the parameter is mandatory
void func({bool? bold, required bool hidden}) {...}
```

Una altra classe de paràmetre que podem usar són els posicionals opcionals, com el seu nom indica son opcionals i venen marcats amb [], de la següent forma:

```
String say(String from, String msg, [String? device]) {
    ...
}

/// Without using the third parameter
assert(say('Bob', 'Howdy') == 'Bob says Howdy');

/// Using the third parameter
assert(say('Bob', 'Howdy', 'smoke signal') ==
      'Bob says Howdy with a smoke signal');
```

La diferència entre els dos tipus de paràmetres és que aquest últim permet afegir un valor per defecte, aquests valors es compilen en temps constant, i en el cas de no proporcionar-se cap valor per defecte, aquest és nul.

```
/// Sets the [bold] and [hidden] flags ...
void enableFlags({bool bold = false, bool hidden =
                                              false}) {...}

// bold will be true; hidden will be false.
enableFlags(bold: true);
```

Les funcions com objectes de primera classe ens permeten passar com a paràmetre una funció a una altra funció.

```
void printElement(int element) {
    print(element);
}

var list = [1, 2, 3];

// Pass printElement as a parameter.
list.forEach(printElement);
```

Dart també permet l'ús de funcions anònimes, aquestes segueixen la següent estructura:

```
([[Type] param1[, …]]) {
  codeBlock;
};
```

Un exemple a continuació:

```
const list = ['apples', 'bananas', 'oranges'];
list.forEach((item) {
    print('${list.indexOf(item)}: $item');
});
```

La següent taula mostra els operadors acceptats per Dart:

Descripció	*Operador*
Sufix unari	`expr++ expr-- () [] ?[] . ?. !`
Prefix unari	`-expr !expr ~expr ++expr --expr await expr`
Multiplicatiu	`* / % ~/`
aditiu	`+ -`
Desplaçament	`<< >> >>>`
AND bit a bit	`&`
XOR bit a bit	`^`
OR bit a bit	`|`
Proba relacional de tipus	`>= > <= < as is is!`
igualtat	`== !=`
AND lògica	`&&`
OR lógica	`||`
if null	`??`
condicional	`expr1 ? expr2 : expr3`
cascade	`.. ?..`
assignació	`= *= /= += -= &= ^=` *etc.*

Declaracions de flux de control que admet:

- `if` and `else`
- `for` loops
- `while` and `do-while` loops
- `break` and `continue`
- `switch` and `case`
- `assert`

Control d'excepcions que admet:

- `throw`
- `try` and `catch`

- `try` and `finally`

6.3. Classes

Exemple de com definim una classe en Dart:

```
class Point {
  double x = 0;
  double y = 0;

  // Constructor
  Point(double x, double y) {
    this.x = x;
    this.y = y;
  }

  // Instance method
  double distanceTo(Point other) {
    var dx = x - other.x;
    var dy = y - other.y;
    return sqrt(dx * dx + dy * dy);
  }
}
```

També podem fer ús de les classes abstractes.

```
// This class is declared abstract and thus
// can't be instantiated.
abstract class AbstractContainer {
  // Define constructors, fields, methods...

  void updateChildren(); // Abstract method.
}
```

Podem fer ús d'interfícies implícites, fent ús de `implements`.

```
// A person. The implicit interface contains greet().
class Person {
  // In the interface, but visible only in this library.
  final String _name;

  // Not in the interface, since this is a constructor.
  Person(this._name);

  // In the interface.
  String greet(String who) => 'Hello, $who. I am
                                        $_name.';
}
```

```
// An implementation of the Person interface.
class Impostor implements Person {
  String get _name => '';

  String greet(String who) => 'Hi $who. Do you know
                                          who I am?';
}
```

Podem ampliar classes fent servir `extends`.

```
class Television {
  void turnOn() {
      _illuminateDisplay();
      _activateIrSensor();
  }
  // ...
}

class SmartTelevision extends Television {
  void turnOn() {
    super.turnOn();
    _bootNetworkInterface();
    _initializeMemory();
    _upgradeApps();
  }
  // ...
}
```

També disposem d'enumeradors, usant `enum`.

```
enum Color { red, green, blue }

assert(Color.red.index == 0);
assert(Color.green.index == 1);
assert(Color.blue.index == 2);
```

Podem reutilitzar codi d'una classe en vàries jerarquies de classe utilitzant `with`, de la següent manera:

```
class Maestro extends Person with Musical, Aggressive {
    Maestro(String maestroName) {
    name = maestroName;
    canConduct = true;
  }
}
```

6.4. Llibreries

S'utilitza `import` per importar una llibreria.

```
import 'dart:html';
```

6.5. Suport de asincronia

A causa que existeixen llibreries que utilitzen funcions que són asíncrones: retornen després d'establir una operació que possiblement requereix temps (com ara I/O), sense esperar que aquesta operació es completi. Fem servir `async` i `await` que ens permeten executar aquest tipus de funcions.

6.6. Altres

Els àlies `typedef` que ens deixen referir a un tipus, com ara:

```
typedef IntList = List<int>;
IntList il = [1, 2, 3];
```

L'ús de metadades que ens permet aportar informació addicional, tenim tres anotacions disponibles `@Deprecated`, `@deprecated` i `@override`. Un exemple de com s'utilitza:

```
class Television {
  /// Use [turnOn] to turn the power on instead.
  @Deprecated('Use turnOn instead')
  void activate() {
    turnOn();
  }

  /// Turns the TV's power on.
  void turnOn() {...}
  // ···
}
```

Per comentar una línia fem servir `//`, en canvi, per comentar més d'una línia `/*` i `*/`.

```
// Single-line comments

/*
 * Multi-line comments
 */
```

7. Bibliografia

A continuació es mostren les fonts d'informació utilitzades i una breu descripció del contingut:

Walrath, K., & Ladd, S. (2012). Dart: Up and Running (1a ed.) O'Reilly

És un llibre que ens parla una mica sobre Dart, els seus orígens i motivacions i posteriorment ens fa un *tour* del llenguatge, les llibreries i les diferents eines.

Walrath, K., & Ladd, S. (2012). What is Dart? (1a ed.) O'Reilly

És un llibre de 28 pàgines que es dedica a contestar les principals preguntes sobre Dart, com que és, perquè es va crear, amb quines novetats ve...

Bracha G. (2015). The Dart Programming Language (1a ed.) Addison Wesley

És un llibre que ens mostra les construccions clau de Dart amb diferents exemples, centrant-se en els principis del llenguatge.

Sade, J., & Galloway M. (2021) Dart Apprentice (1a ed.) Bowker

És un llibre que està pensat per anar llegint-lo mentre es fan exercicis de programació que els autors proposen al final de cada capítol.

https://dart.dev/guides/language/language-tour

Aquest enllaç ens porta a la pàgina oficial del llenguatge on podem trobar tant informació sobre el llenguatge com exemples i explicacions de les diferents funcionalitats que ofereix.

https://medium.com/comunidad-flutter/introducción-a-dart-vm-889aedcd9c00

És un article que explica el funcionament de Dart VM.

https://profile.es/blog/que-es-flutter-sdk/

És un article que explica què és Flutter i les seves característiques principals.

https://github.com/yissachar/awesome-dart

És un llistat dels frameworks, llibreries i software amb els que podem usar Dart.

https://inlab.fib.upc.edu/ca/blog/que-es-el-llenguatge-de-programacio-dart

És un article del inLab de la FIB que explica breument què és Dart i perquè serveix.

https://en.wikipedia.org/wiki/Dart_(programming_language)

A Wikipedia podem trobar informació sobre els orígens de Dart, els usos i característiques principals.

https://www.pcmag.com/encyclopedia

És un diccionari web en el qual he cercat definicions