EBook Gratis

APRENDIZAJE dart

Free unaffiliated eBook created from
Stack Overflow contributors.

#dart

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: dart

It is an unofficial and free dart ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official dart.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con el dardo

Observaciones

Dart es un lenguaje de programación de código abierto, basado en clases y opcionalmente escrito para crear aplicaciones web, tanto en el cliente como en el servidor, creado por Google. Los objetivos de diseño de Dart son:

- Crea un lenguaje estructurado pero flexible para la programación web.
- Haga que Dart se sienta familiar y natural para los programadores y, por lo tanto, fácil de aprender.
- Asegúrese de que Dart ofrezca un alto rendimiento en todos los navegadores web y entornos modernos, desde pequeños dispositivos de mano hasta la ejecución del lado del servidor.

Dart se dirige a una amplia gama de escenarios de desarrollo, desde un proyecto de una sola persona sin mucha estructura hasta un proyecto a gran escala que requiere tipos formales en el código para indicar la intención del programador.

Para dar soporte a esta amplia gama de proyectos, Dart proporciona las siguientes funciones y herramientas:

- **Tipos opcionales:** esto significa que puede comenzar a codificar sin tipos y agregarlos más tarde según sea necesario.
- **Aísla:** programación concurrente en servidor y cliente.
- **Fácil acceso a DOM:** usando selectores de CSS (de la misma manera que lo hace jQuery)
- **Herramientas de IDE de Dart:** Existen complementos de Dart para muchos IDE de uso común, por ejemplo , WebStorm .
- **Dartium:** una compilación del navegador web Chromium con una máquina virtual Dart incorporada

Campo de golf

- La página de Dart
- Noticias y actualizaciones oficiales de Dart
- The Dartosphere - Una colección de publicaciones recientes del blog Dart.
- Dartisans Dartisans community en Google+
- Dart Web Development - Página de Grupos de Google
- Dart Language Misc - Página de Grupos de Google
- DartLang sub-Reddit

Documentación

- Recorrido por el lenguaje del dardo
- Recorrido por las bibliotecas de dardos
- Dart Code samples
- Referencia API de Dart

Preguntas más frecuentes

- Preguntas frecuentes

Versiones

Versión	Fecha de lanzamiento
1.22.1	2017-02-22
1.22.0	2017-02-14
1.21.1	2016-01-13
1.21.0	2016-12-07
1.20.1	2016-10-13
1.20.0	2016-10-11
1.19.1	2016-09-07
1.19.0	2016-08-26
1.18.1	2016-08-02
1.18.0	2016-07-27
1.17.1	2016-06-10
1.17.0	2016-06-06
1.16.1	2016-05-23
1.16.0	2016-04-26
1.15.0	2016-03-09
1.14.2	2016-02-09
1.14.1	2016-02-03

Versión	Fecha de lanzamiento
1.14.0	2016-01-28
1.13.2	2016-01-05
1.13.1	2015-12-17
1.13.0	2015-11-18
1.12.2	2015-10-21
1.12.1	2015-09-08
1.12.0	2015-08-31
1.11.3	2015-08-03
1.11.1	2015-07-02
1.11.0	2015-06-24
1.10.1	2015-05-11
1.10.0	2015-04-24
1.9.3	2015-04-13
1.9.1	2015-03-25
1.8.5	2015-01-13
1.8.3	2014-12-01
1.8.0	2014-11-27
1.7.2	2014-10-14
1.6.0	2014-08-27
1.5.8	2014-07-29
1.5.3	2014-07-03
1.5.2	2014-07-02
1.5.1	2014-06-24
1.4.3	2014-06-16
1.4.2	2014-05-27

Versión	Fecha de lanzamiento
1.4.0	2014-05-20
1.3.6	2014-04-30
1.3.3	2014-04-16
1.3.0	2014-04-08
1.2.0	2014-02-25
1.1.3	2014-02-06
1.1.1	2014-01-15
1.0.0.10_r30798	2013-12-02
1.0.0.3_r30188	2013-11-12
0.8.10.10_r30107	2013-11-08
0.8.10.6_r30036	2013-11-07
0.8.10.3_r29803	2013-11-04

Examples

Instalación o configuración

Dart SDK incluye todo lo que necesita para escribir y ejecutar el código Dart: VM, bibliotecas, analizador, gestor de paquetes, generador de documentos, formateador, depurador y más. Si está haciendo desarrollo web, también necesitará Dartium.

Instalación automatizada y actualizaciones

- Instalación de Dart en Windows
- Instalación de Dart en Mac
- Instalación de Dart en Linux

Manual de instalación

También puede instalar manualmente cualquier versión del SDK .

¡Hola Mundo!

Cree un nuevo archivo llamado `hello_world.dart` con el siguiente contenido:

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

En el terminal, navegue hasta el directorio que contiene el archivo `hello_world.dart` y escriba lo siguiente:

```
dart hello_world.dart
```

Presiona `enter` para mostrar `Hello, World!` En la ventana del terminal.

Solicitud de http

Html

```
<img id="cats"></img>
```

Dardo

```
import 'dart:html';

/// Stores the image in [blob] in the [ImageElement] of the given [selector].
void setImage(selector, blob) {
  FileReader reader = new FileReader();
  reader.onLoad.listen((fe) {
    ImageElement image = document.querySelector(selector);
    image.src = reader.result;
  });
  reader.readAsDataUrl(blob);
}

main() async {
  var url = "https://upload.wikimedia.org/wikipedia/commons/2/28/Tortoiseshell_she-cat.JPG";

  // Initiates a request and asynchronously waits for the result.
  var request = await HttpRequest.request(url, responseType: 'blob');
  var blob = request.response;
  setImage("#cats", blob);
}
```

Ejemplo

vea el Ejemplo en https://dartpad.dartlang.org/a0e092983f63a40b0b716989cac6969a

Hechiceros y Setters

```
void main() {
  var cat = new Cat();

  print("Is cat hungry? ${cat.isHungry}");  // Is cat hungry? true
```

```
  print("Is cat cuddly? ${cat.isCuddly}");  // Is cat cuddly? false
  print("Feed cat.");
  cat.isHungry = false;
  print("Is cat hungry? ${cat.isHungry}");  // Is cat hungry? false
  print("Is cat cuddly? ${cat.isCuddly}");  // Is cat cuddly? true
}

class Cat {
  bool _isHungry = true;

  bool get isCuddly => !_isHungry;

  bool get isHungry => _isHungry;
  bool set isHungry(bool hungry) => this._isHungry = hungry;
}
```

Los captadores y definidores de clases de Dart permiten que las API encapsulen cambios de estado de objetos.

Vea el ejemplo de dartpad aquí:
https://dartpad.dartlang.org/c25af60ca18a192b84af6990f3313233

Lea Empezando con el dardo en línea: https://riptutorial.com/es/dart/topic/843/empezando-con-el-dardo

Capítulo 2: Bibliotecas

Observaciones

Las directivas de `import` y `library` pueden ayudarlo a crear una base de código modular y compartible. Cada aplicación Dart es una `library` , incluso si no utiliza una directiva de biblioteca. Las bibliotecas se pueden distribuir usando paquetes. Consulte Pub Package y Asset Manager para obtener información sobre pub, un administrador de paquetes incluido en el SDK.

Examples

Utilizando bibliotecas

Use `import` para especificar cómo se usa un espacio de nombres de una biblioteca en el alcance de otra biblioteca.

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

El único argumento necesario para `import` es un URI que especifique la biblioteca. Para las bibliotecas integradas, el URI tiene el esquema especial de `dart:` Para otras bibliotecas, puede utilizar una ruta del sistema de archivos o el `package:` esquema. El `package:` esquema especifica las bibliotecas proporcionadas por un administrador de paquetes como la herramienta de publicación. Por ejemplo:

```
import 'dart:io';
import 'package:mylib/mylib.dart';
import 'package:utils/utils.dart';
```

Bibliotecas y visibilidad.

A diferencia de Java, Dart no tiene las palabras clave `public` , `protected` y privada. Si un identificador comienza con un guión bajo `_` , es privado a su biblioteca.

Si, por ejemplo, tiene la clase A en un archivo de biblioteca separado (por ejemplo, `other.dart`), como:

```
library other;

class A {
  int _private = 0;

  testA() {
    print('int value: $_private');  // 0
    _private = 5;
    print('int value: $_private'); // 5
  }
}
```

y luego impórtelo en su aplicación principal, como:

```
import 'other.dart';

void main() {
  var b = new B();
  b.testB();
}

class B extends A {
  String _private;

  testB() {
    _private = 'Hello';
    print('String value: $_private'); // Hello
    testA();
    print('String value: $_private'); // Hello
  }
}
```

Obtienes el resultado esperado:

```
String value: Hello
int value: 0
int value: 5
String value: Hello
```

Especificando un prefijo de biblioteca

Si importa dos bibliotecas que tienen identificadores en conflicto, puede especificar un prefijo para una o ambas bibliotecas. Por ejemplo, si library1 y library2 tienen una clase Element, entonces podría tener un código como este:

```
import 'package:lib1/lib1.dart';
import 'package:lib2/lib2.dart' as lib2;
// ...
var element1 = new Element(); // Uses Element from lib1.
var element2 =
    new lib2.Element();       // Uses Element from lib2.
```

Importando solo una parte de una biblioteca

Si desea utilizar solo una parte de una biblioteca, puede importar selectivamente la biblioteca. Por ejemplo:

```
// Import only foo and bar.
import 'package:lib1/lib1.dart' show foo, bar;

// Import all names EXCEPT foo.
import 'package:lib2/lib2.dart' hide foo;
```

Lazily cargando una biblioteca

La carga diferida (también llamada carga diferida) permite que una aplicación cargue una biblioteca a pedido, cuando sea necesario. Para cargar una biblioteca de forma perezosa, primero debe importarla usando diferido como.

```
import 'package:deferred/hello.dart' deferred as hello;
```

Cuando necesite la biblioteca, invoque loadLibrary () utilizando el identificador de la biblioteca.

```
greet() async {
  await hello.loadLibrary();
  hello.printGreeting();
}
```

En el código anterior, la `await` palabra clave se detiene la ejecución hasta que se carga la biblioteca. Para obtener más información sobre `async` y `await` , vea más ejemplos aquí sobre asistencia de asincronía o visite la parte de asistencia de asincronía de la gira de idiomas.

Lea Bibliotecas en línea: https://riptutorial.com/es/dart/topic/3332/bibliotecas

Capítulo 3: Colecciones

Examples

Creando una nueva lista

Las listas se pueden crear de múltiples maneras.

La forma recomendada es usar un literal de `List` :

```
var vegetables = ['broccoli', 'cabbage'];
```

El constructor de `List` se puede utilizar:

```
var fruits = new List();
```

Si prefiere una escritura más fuerte, también puede proporcionar un parámetro de tipo de una de las siguientes maneras:

```
var fruits = <String>['apples', 'oranges'];
var fruits = new List<String>();
```

Para crear una pequeña lista ampliable, ya sea vacía o que contenga algunos valores iniciales conocidos, se prefiere la forma literal. Existen constructores especializados para otros tipos de listas:

```
var fixedLengthList1 = new List(8);
var fixedLengthList2 = new List.filled(8, "initial text");
var computedValues = new List.generate(8, (n) => "x" * n);
var fromIterable = new List<String>.from(computedValues.getRange(2, 5));
```

Vea también la guía de estilo de Dardos efectivos sobre colecciones .

Creando un nuevo conjunto

Los conjuntos se pueden crear a través del constructor:

```
var ingredients = new Set();
ingredients.addAll(['gold', 'titanium', 'xenon']);
```

Creando un nuevo mapa

Los mapas se pueden crear de múltiples maneras.

Usando el constructor, puede crear un nuevo mapa de la siguiente manera:

```
var searchTerms = new Map();
```

Los tipos para la clave y el valor también se pueden definir usando genéricos:

```
var nobleGases = new Map<int, String>();
var nobleGases = <int, String>{};
```

De lo contrario, los mapas se pueden crear utilizando el literal del mapa:

```
var map = {
    "key1": "value1",
    "key2": "value2"
};
```

Mapea cada elemento de la colección.

Todos los objetos de colección contienen un método de `map` que toma una `Function` como un argumento, que debe tomar un solo argumento. Esto devuelve un `Iterable` respaldado por la colección. Cuando se itera el `Iterable` , cada paso llama a la función con un nuevo elemento de la colección, y el resultado de la llamada se convierte en el siguiente elemento de la iteración.

Puede convertir un `Iterable` en una colección nuevamente usando los métodos `Iterable.toSet()` o `Iterable.toList()` , o usando un constructor de colección que toma un iterable como `Queue.from` o `List.from` .

Ejemplo:

```
main() {
  var cats = [
    'Abyssinian',
    'Scottish Fold',
    'Domestic Shorthair'
  ];

  print(cats); // [Abyssinian, Scottish Fold, Domestic Shorthair]

  var catsInReverse =
  cats.map((String cat) {
    return new String.fromCharCodes(cat.codeUnits.reversed);
  })
  .toList(); // [nainissybA, dloF hsittocS, riahtrohS citsemoD]

  print(catsInReverse);
}
```

Vea el ejemplo de dartpad aquí: https://dartpad.dartlang.org/a18367ff767f172b34ff03c7008a6fa1

Filtrar una lista

Dart permite filtrar fácilmente una lista usando `where` .

```
var fruits = ['apples', 'oranges', 'bananas'];
```

```
fruits.where((f) => f.startsWith('a')).toList(); //apples
```

Por supuesto, puede usar algunos operadores AND u OR en su cláusula where.

Lea Colecciones en línea: https://riptutorial.com/es/dart/topic/859/colecciones

Capítulo 4: Comentarios

Sintaxis

- // Comentario de una sola línea
- / * Multi-línea / Comentario en línea * /
- /// comentario Dartdoc

Observaciones

Es una buena práctica agregar comentarios a su código para explicar por qué se hace algo o para explicar qué hace algo. Esto ayuda a los futuros lectores de su código a comprender más fácilmente su código.

Tema (s) relacionado (s) que no están en StackOverflow:

- Dardo efectivo: Documentación

Examples

Comentario de fin de línea

Todo a la derecha de `//` en la misma línea está comentado.

```
int i = 0; // Commented out text
```

Comentario multilínea

Todo entre `/*` y `*/` está comentado.

```
void main() {
  for (int i = 0; i < 5; i++) {
    /* This is commented, and
    will not affect code */
    print('hello ${i + 1}');
  }
}
```

Documentación utilizando Dartdoc.

El uso de un comentario doc en lugar de un comentario regular permite a dartdoc encontrarlo y generar documentación para él.

```
/// The number of characters in this chunk when unsplit.
int get length => ...
```

Se le permite utilizar la mayoría de rebajas formato en sus comentarios doc y dartdoc procesará en consecuencia utilizando el paquete de reducción del precio .

```
/// This is a paragraph of regular text.
///
/// This sentence has *two* _emphasized_ words (i.e. italics) and **two**
/// __strong__ ones (bold).
///
/// A blank line creates another separate paragraph. It has some `inline code`
/// delimited using backticks.
///
/// * Unordered lists.
/// * Look like ASCII bullet lists.
/// * You can also use `-` or `+`.
///
/// Links can be:
///
/// * http://www.just-a-bare-url.com
/// * [with the URL inline](http://google.com)
/// * [or separated out][ref link]
///
/// [ref link]: http://google.com
///
/// # A Header
///
/// ## A subheader
```

Lea Comentarios en línea: https://riptutorial.com/es/dart/topic/2436/comentarios

Capítulo 5: Convertir datos

Examples

JSON

```
import 'dart:convert';

void main() {
    var jsonString = """
      {
        "cats": {
                "abysinnian": {
                    "origin": "Burma",
                    "behavior": "playful"
                }
            }
    }
  """;

  var obj = JSON.decode(jsonString);

  print(obj['cats']['abysinnian']['behavior']); // playful
}
```

Vea el ejemplo en dartpad: https://dartpad.dartlang.org/7d5958cf10e611b36326f27b062108fe

Lea Convertir datos en línea: https://riptutorial.com/es/dart/topic/2778/convertir-datos

Capítulo 6: Enums

Examples

Uso básico

```
enum Fruit {
  apple, banana
}

main() {
  var a = Fruit.apple;
  switch (a) {
    case Fruit.apple:
      print('it is an apple');
      break;
  }

  // get all the values of the enums
  for (List<Fruit> value in Fruit.values) {
    print(value);
  }

  // get the second value
  print(Fruit.values[1]);
}
```

Lea Enums en línea: https://riptutorial.com/es/dart/topic/5107/enums

Capítulo 7: Excepciones

Observaciones

El código de Dart puede lanzar y atrapar excepciones. Las excepciones son errores que indican que algo inesperado sucedió. Si la excepción no se detecta, el aislamiento que generó la excepción se suspende y, por lo general, el aislamiento y su programa se terminan.

A diferencia de Java, todas las excepciones de Dart son excepciones sin marcar. Los métodos no declaran qué excepciones pueden lanzar, y no se requiere que atrapes ninguna excepción.

Dart proporciona tipos de Excepción y Error , así como numerosos subtipos predefinidos. Por supuesto, puedes definir tus propias excepciones. Sin embargo, los programas de Dart pueden lanzar cualquier objeto que no sea nulo, no solo los objetos de Excepción y Error, como excepción.

Examples

Excepción personalizada

```
class CustomException implements Exception {
  String cause;
  CustomException(this.cause);
}

void main() {
  try {
    throwException();
  } on CustomException {
    print("custom exception is been obtained");
  }
}

throwException() {
  throw new CustomException('This is my first custom exception');
}
```

Lea Excepciones en línea: https://riptutorial.com/es/dart/topic/3334/excepciones

Capítulo 8: Expresiones regulares

Sintaxis

- var regExp = RegExp (r '^ (. *) $', multiLine: true, caseSensitive: false);

Parámetros

Parámetro	Detalles
`String source`	La expresión regular como una `String`
`{bool multiline}`	Si esta es una expresión regular multilínea. (coincide con `^` y `$` al principio y al final de cada línea individualmente no toda la Cadena)
`{bool caseSensitive}`	Si la expresión distingue entre mayúsculas y minúsculas

Observaciones

Las expresiones regulares de Dart tienen la misma sintaxis y semántica que las expresiones regulares de JavaScript. Consulte http://ecma-international.org/ecma-262/5.1/#sec-15.10 para la especificación de las expresiones regulares de JavaScript.

Esto significa que cualquier recurso de JavaScript que encuentre sobre Expresiones regulares en línea se aplica al dardo.

Examples

Crea y usa una expresión regular

```
var regExp = new RegExp(r"(\w+)");
var str = "Parse my string";
Iterable<Match> matches = regExp.allMatches(str);
```

Es una buena idea usar "cadenas sin formato" (prefijo con `r`) al escribir expresiones regulares para que pueda usar barras diagonales sin escape en su expresión.

Lea Expresiones regulares en línea: https://riptutorial.com/es/dart/topic/3624/expresiones-regulares

Capítulo 9: Fecha y hora

Examples

Uso básico de DateTime

```
DateTime now = new DateTime.now();
DateTime berlinWallFell = new DateTime(1989, 11, 9);
DateTime moonLanding = DateTime.parse("1969-07-20 20:18:00");  // 8:18pm
```

Puedes encontrar más información en profundidad aquí .

Lea Fecha y hora en línea: https://riptutorial.com/es/dart/topic/3322/fecha-y-hora

Capítulo 10: Flujo de control

Examples

Si mas

Dardo tiene si otra cosa:

```
if (year >= 2001) {
  print('21st century');
} else if (year >= 1901) {
  print('20th century');
} else {
  print('We Must Go Back!');
}
```

El dardo también tiene un operador ternario `if` :

```
var foo = true;
print(foo ? 'Foo' : 'Bar'); // Displays "Foo".
```

Mientras bucle

Mientras que los bucles y hacer mientras que los bucles están permitidos en Dart:

```
while(peopleAreClapping()) {
  playSongs();
}
```

y:

```
do {
  processRequest();
} while(stillRunning());
```

Los bucles se pueden terminar con una ruptura:

```
while (true) {
  if (shutDownRequested()) break;
  processIncomingRequests();
}
```

Puede omitir iteraciones en un bucle usando continue:

```
for (var i = 0; i < bigNumber; i++) {
  if (i.isEven){
    continue;
  }
  doSomething();
```

```
}
```

En bucle

Se permiten dos tipos de bucles for:

```
for (int month = 1; month <= 12; month++) {
  print(month);
}
```

y:

```
for (var object in flybyObjects) {
  print(object);
}
```

El bucle `for-in` es conveniente cuando simplemente se itera sobre una colección `Iterable` . También hay un método `forEach` que puedes invocar objetos `Iterable` que se comportan como `for-in` :

```
flybyObjects.forEach((object) => print(object));
```

o, más concisamente:

```
flybyObjects.forEach(print);
```

Caja de interruptores

Dart tiene un caso de conmutador que se puede usar en lugar de largas declaraciones if-else:

```
var command = 'OPEN';

switch (command) {
  case 'CLOSED':
    executeClosed();
    break;
  case 'OPEN':
    executeOpen();
    break;
  case 'APPROVED':
    executeApproved();
    break;
  case 'UNSURE':
    // missing break statement means this case will fall through
    // to the next statement, in this case the default case
  default:
    executeUnknown();
}
```

Solo puede comparar constantes enteras, de cadena o de tiempo de compilación. Los objetos comparados deben ser instancias de la misma clase (y no de ninguno de sus subtipos), y la clase

no debe anular ==.

Un aspecto sorprendente del cambio en Dart es que las cláusulas de casos no vacíos deben terminar con un descanso, o con menos frecuencia, continuar, lanzar o devolver. Es decir, las cláusulas de casos no vacíos no pueden fallar. Debe finalizar explícitamente una cláusula de caso no vacía, generalmente con una ruptura. Recibirá una advertencia estática si omite el descanso, la continuación, el lanzamiento o la devolución, y el código generará un error en esa ubicación en el tiempo de ejecución.

```
var command = 'OPEN';
switch (command) {
  case 'OPEN':
    executeOpen();
    // ERROR: Missing break causes an exception to be thrown!!

  case 'CLOSED': // Empty case falls through
  case 'LOCKED':
    executeClosed();
    break;
}
```

Si desea obtener información detallada en un `case` que no esté vacío, puede usar `continue` y una etiqueta:

```
      var command = 'OPEN';
      switch (command) {
        case 'OPEN':
          executeOpen();
          continue locked;
locked: case 'LOCKED':
          executeClosed();
          break;
      }
```

Lea Flujo de control en línea: https://riptutorial.com/es/dart/topic/923/flujo-de-control

Capítulo 11: Funciones

Observaciones

Dart es un verdadero lenguaje orientado a objetos, por lo que incluso las funciones son objetos y tienen un tipo, Función. Esto significa que las funciones pueden asignarse a variables o pasarse como argumentos a otras funciones. También puede llamar a una instancia de una clase Dart como si fuera una función.

Examples

Funciones con parámetros nombrados.

Al definir una función, use {param1, param2,…} para especificar parámetros nombrados:

```
void enableFlags({bool bold, bool hidden}) {
  // ...
}
```

Al llamar a una función, puede especificar parámetros con nombre usando paramName: valor

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

Función de alcance

Las funciones de dardo también se pueden declarar de forma anónima o anónima. Por ejemplo, para crear una función anidada, simplemente abra un nuevo bloque de funciones dentro de un bloque de funciones existente

```
void outerFunction() {

    bool innerFunction() {
        /// Does stuff
    }
}
```

La función `innerFunction` ahora se puede usar dentro, y solo dentro, `outerFunction` . Ninguna otra función tiene acceso a ella.

Las funciones en Dart también se pueden declarar de forma anónima, que se utiliza comúnmente como argumentos de función. Un ejemplo común es el método de `sort` del objeto `List` . Este método toma un argumento opcional con la siguiente firma:

```
int compare(E a, E b)
```

La documentación indica que la función debe devolver `0` si `a` y `b` son iguales. Devuelve `-1` si `a < b`

y `1` si `a > b` .

Sabiendo esto, podemos ordenar una lista de enteros usando una función anónima.

```
List<int> numbers = [4,1,3,5,7];

numbers.sort((int a, int b) {
   if(a == b) {
      return 0;
   } else if (a < b) {
      return -1;
   } else {
      return 1;
   }
});
```

La función anónima también puede estar vinculada a identificadores como tal:

```
Function intSorter = (int a, int b) {
   if(a == b) {
      return 0;
   } else if (a < b) {
      return -1;
   } else {
      return 1;
   }
}
```

y usado como una variable ordinaria.

```
numbers.sort(intSorter);
```

Lea Funciones en línea: https://riptutorial.com/es/dart/topic/2965/funciones

Capítulo 12: Instrumentos de cuerda

Examples

Concatenación e interpolación.

Puede utilizar el operador más (`+`) para concatenar cadenas:

```
'Dart ' + 'is ' + 'fun!'; // 'Dart is fun!'
```

También puede utilizar literales de cadena adyacentes para la concatenación:

```
'Dart ' 'is ' 'fun!';    // 'Dart is fun!'
```

Puede usar `${}` para interpolar el valor de las expresiones de Dart dentro de las cadenas. Las llaves se pueden omitir al evaluar identificadores:

```
var text = 'dartlang';
'$text has ${text.length} letters'; // 'dartlang has 8 letters'
```

Cadenas validas

Una cadena puede ser simple o multilínea. Las cadenas de una sola línea se escriben con comillas simples o dobles coincidentes, y las cadenas de varias líneas se escriben con comillas triples. Las siguientes son todas las cadenas de Dart válidas:

```
'Single quotes';
"Double quotes";
'Double quotes in "single" quotes';
"Single quotes in 'double' quotes";

'''A
multiline
string''';

"""
Another
multiline
string""";
```

Construyendo desde partes

La generación programada de un String se realiza mejor con un StringBuffer . Un StringBuffer no genera un nuevo objeto String hasta que se llama a `toString()` .

```
var sb = new StringBuffer();

sb.write("Use a StringBuffer");
```

```
sb.writeAll(["for ", "efficient ", "string ", "creation "]);
sb.write("if you are ")
sb.write("building lots of strings");

// or you can use method cascades:

sb
  ..write("Use a StringBuffer")
  ..writeAll(["for ", "efficient ", "string ", "creation "])
  ..write("if you are ")
  ..write("building lots of strings");

var fullString = sb.toString();

print(fullString);
// Use a StringBufferfor efficient string creation if you are building lots of strings

sb.clear(); // all gone!
```

Lea Instrumentos de cuerda en línea: https://riptutorial.com/es/dart/topic/5003/instrumentos-de-cuerda

Capítulo 13: Interoperabilidad Dart-JavaScript

Introducción

La interoperabilidad Dart-JavaScript nos permite ejecutar código JavaScript desde nuestros programas Dart.

La interoperabilidad se logra utilizando la biblioteca `js` para crear apéndices de Dart. Estos apéndices describen la interfaz que nos gustaría tener con el código JavaScript subyacente. En el tiempo de ejecución, llamar al código auxiliar de Dart invocará el código JavaScript.

Examples

Llamando a una función global

Supongamos que nos gustaría invocar la función de JavaScript `JSON.stringify` que recibe un objeto, lo codifica en una cadena JSON y lo devuelve.

Todo lo que tendríamos que hacer es escribir la firma de la función, marcarla como externa y anotarla con la anotación `@JS` :

```
 @JS("JSON.stringify")
external String stringify(obj);
```

La anotación `@JS` se usará de aquí en adelante para marcar las clases de Dart que también nos gustaría usar en JavaScript.

Envolviendo clases de JavaScript / espacios de nombres

Supongamos que nos gustaría envolver las API de Google Maps JavaScript `google.maps` :

```
@JS('google.maps')
library maps;

import "package:js/js.dart";

@JS()
class Map {
  external Map(Location location);
  external Location getLocation();
}
```

Ahora tenemos la clase Map Dart que corresponde a la clase `google.maps.Map` JavaScript.

Ejecutar `new Map(someLocation)` en Dart invocará `new google.maps.Map(location)` en JavaScript.

Tenga en cuenta que no tiene que nombrar su clase de Dart igual que la clase de JavaScript:

```
@JS("LatLng")
class Location {
  external Location(num lat, num lng);
}
```

La clase `Location` Dart corresponde a la clase `google.maps.LatLng` .

Se desaconseja el uso de nombres inconsistentes ya que pueden crear confusión.

Paso de literales de objeto.

Es una práctica común en JavaScript pasar objetos literales a funciones:

```
// JavaScript
printOptions({responsive: true});
Unfortunately we cannot pass Dart Map objects to JavaScript in these cases.
```

Lo que tenemos que hacer es crear un objeto Dart que represente el objeto literal y contenga todos sus campos:

```
// Dart
@JS()
@anonymous
class Options {
  external bool get responsive;

  external factory Options({bool responsive});
}
```

Tenga en cuenta que la clase Dart de `Options` no corresponde a ninguna clase de JavaScript. Como tal debemos marcarlo con la anotación `@anonymous` .

Ahora podemos crear un código auxiliar para la función de impresión original y llamarlo con un nuevo objeto de Opciones:

```
// Dart
@JS()
external printOptions(Options options);

printOptions(new Options(responsive: true));
```

Lea Interoperabilidad Dart-JavaScript en línea:
https://riptutorial.com/es/dart/topic/9240/interoperabilidad-dart-javascript

Capítulo 14: Las clases

Examples

Creando una clase

Las clases se pueden crear de la siguiente manera:

```
class InputField {
  int maxLength;
  String name;
}
```

Se puede crear una instancia de la clase utilizando la `new` palabra clave, después de lo cual los valores de campo serán nulos de forma predeterminada.

```
var field = new InputField();
```

Se puede acceder a los valores de campo:

```
// this will trigger the setter
field.name = "fieldname";

// this will trigger the getter
print(field.name);
```

Miembros

Una clase puede tener miembros.

Las variables de instancia se pueden declarar con / sin anotaciones de tipo y, opcionalmente, se pueden inicializar. Los miembros sin inicializar tienen el valor de `null` , a menos que el constructor los establezca en otro valor.

```
class Foo {
  var member1;
  int member2;
  String member3 = "Hello world!";
}
```

Las variables de clase se declaran utilizando la palabra clave `static` .

```
class Bar {
  static var member4;
  static String member5;
  static int member6 = 42;
}
```

Si un método no toma argumentos, es rápido, devuelve un valor y no tiene efectos secundarios visibles, entonces se puede usar un método getter:

```
class Foo {
  String get bar {
    var result;
    // ...
    return result;
  }
}
```

Los getters nunca toman argumentos, por lo que los paréntesis para la lista de parámetros (vacía) se omiten tanto para declarar a los getters, como los anteriores, y para llamarlos, de esta manera:

```
main() {
  var foo = new Foo();
  print(foo.bar); // prints "bar"
}
```

También hay métodos de establecimiento, que deben tomar exactamente un argumento:

```
class Foo {
  String _bar;

  String get bar => _bar;

  void set bar(String value) {
    _bar = value;
  }
}
```

La sintaxis para llamar a un definidor es la misma que la asignación de variables:

```
main() {
  var foo = new Foo();
  foo.bar = "this is calling a setter method";
}
```

Constructores

Un constructor de clase debe tener el mismo nombre que su clase.

Vamos a crear un constructor para una persona de clase:

```
class Person {
  String name;
  String gender;
  int age;

  Person(this.name, this.gender, this.age);
}
```

El ejemplo anterior es una forma más simple y mejor de definir el constructor que la siguiente, que

también es posible:

```
class Person {
  String name;
  String gender;
  int age;

  Person(String name, String gender, int age) {
    this.name = name;
    this.gender = gender;
    this.age = age;
  }
}
```

Ahora puedes crear una instancia de Persona como esta:

```
var alice = new Person('Alice', 'female', 21);
```

Lea Las clases en línea: https://riptutorial.com/es/dart/topic/1511/las-clases

Capítulo 15: Lista de filtros

Introducción

Dart filtra las listas a través de los métodos `List.where` y `List.retainWhere` . La función `where` toma un argumento: una función booleana que se aplica a cada elemento de la lista. Si la función se evalúa como `true` entonces se retiene el elemento de lista; Si la función se evalúa como `false` , el elemento se elimina.

Llamar a `theList.retainWhere(foo)` es prácticamente equivalente a configurar `theList = theList.where(foo)` .

Examples

Filtrar una lista de enteros

```
[-1, 0, 2, 4, 7, 9].where((x) => x > 2) --> [4, 7, 9]
```

Lea Lista de filtros en línea: https://riptutorial.com/es/dart/topic/10948/lista-de-filtros

Capítulo 16: Programación Asíncrona

Examples

Devolviendo un futuro usando un Completer

```
Future<Results> costlyQuery() {
  var completer = new Completer();

  database.query("SELECT * FROM giant_table", (results) {
    // when complete
    completer.complete(results);
  }, (error) {
    completer.completeException(error);
  });

  // this returns essentially immediately,
  // before query is finished
  return completer.future;
}
```

Async y espera

```
import 'dart:async';

Future main() async {
  var value = await _waitForValue();
  print("Here is the value: $value");
  //since _waitForValue() returns immediately if you un it without await you won't get the
result
  var errorValue = "not finished yet";
  _waitForValue();
  print("Here is the error value: $value");// not finished yet
}

Future<int> _waitForValue() => new Future((){

  var n = 100000000;

  // Do some long process
  for (var i = 1; i <= n; i++) {
    // Print out progress:
    if ([n / 2, n / 4, n / 10, n / 20].contains(i)) {
      print("Not done yet...");
    }

    // Return value when done.
    if (i == n) {
      print("Done.");
      return i;
    }
  }
});
```

Vea el ejemplo en Dartpad: https://dartpad.dartlang.org/11d189b51e0f2680793ab3e16e53613c

Convertir devoluciones de llamada a futuros

Dart tiene una robusta biblioteca asíncrona, con Future , Stream y más. Sin embargo, a veces puede encontrarse con una API asíncrona que utiliza *devoluciones* de *llamada en* lugar de *futuros* . Para cerrar la brecha entre las devoluciones de llamada y los futuros, Dart ofrece la clase *Completer* . Puede utilizar un Completer para convertir una devolución de llamada en un futuro.

Los completadores son excelentes para enlazar una API basada en devolución de llamada con una API basada en futuro. Por ejemplo, suponga que el controlador de su base de datos no usa futuros, pero necesita devolver un futuro. Prueba este código:

```
// A good use of a Completer.

Future doStuff() {
  Completer completer = new Completer();
  runDatabaseQuery(sql, (results) {
    completer.complete(results);
  });
  return completer.future;
}
```

Si está utilizando una API que ya devuelve un Futuro, no necesita usar un Completador.

Lea Programación Asíncrona en línea: https://riptutorial.com/es/dart/topic/2520/programacion-asincrona

Capítulo 17: pub

Observaciones

Cuando instala el Dart SDK, una de las herramientas que obtiene es pub. La herramienta de publicación proporciona comandos para una variedad de propósitos. Un comando instala paquetes, otro inicia un servidor HTTP para la prueba, otro prepara su aplicación para la implementación y otro publica su paquete en pub.dartlang.org . Puede acceder a los comandos del pub a través de un IDE, como WebStorm, o en la línea de comandos.

Para obtener una descripción general de estos comandos, consulte Comandos de publicación .

Examples

construcción de pub

Use pub build cuando esté listo para implementar su aplicación web. Cuando ejecuta pub build, genera los activos para el paquete actual y todas sus dependencias, colocándolos en el nuevo directorio llamado build.

Para usar `pub build` , simplemente ejecútelo en el directorio raíz de su paquete. Por ejemplo:

```
$ cd ~/dart/helloworld
$ pub build
Building helloworld......
Built 5 files!
```

servicio de pub

Este comando inicia un servidor de desarrollo o servidor dev para su aplicación web Dart. El servidor dev es un servidor HTTP en localhost que sirve los activos de su aplicación web.

Inicie el servidor dev desde el directorio que contiene el archivo `pubspec.yaml` su aplicación web:

```
$ cd ~/dart/helloworld
$ pub serve
Serving helloworld on http://localhost:8080
```

Lea pub en línea: https://riptutorial.com/es/dart/topic/3335/pub

Creditos

S. No	Capítulos	Contributors
1	Empezando con el dardo	4444, Challe, Community, Damon, Florian Loitsch, Gomiero, Kleak, losnake, martin, Raph, Timothy C. Quinn
2	Bibliotecas	Challe, Ganymede
3	Colecciones	Alexi Coard, Damon, Jan Vladimir Mostert, Kleak, lrn, Pacane, Raph
4	Comentarios	Challe
5	Convertir datos	Damon
6	Enums	Challe
7	Excepciones	Challe
8	Expresiones regulares	enyo
9	Fecha y hora	Challe
10	Flujo de control	Ganymede, Jan Vladimir Mostert, Pacane, Raph
11	Funciones	Jan Vladimir Mostert, Kim Rostgaard Christensen
12	Instrumentos de cuerda	Challe
13	Interoperabilidad Dart-JavaScript	Meshulam Silk
14	Las clases	Ganymede, Hoylen, Jan Vladimir Mostert, Raph
15	Lista de filtros	jxmorris12
16	Programación Asíncrona	Challe, Damon, Ray Hulha, Zied Hamdi
17	pub	Challe