

Ngôn ngữ lập trình Dart

Bùi Võ Quốc Bảo

Khoa CNTT | Trường CNTT-TT | Đại học Cần Thơ

Tài liệu tham khảo

- Dart Apprentice by Jonathan Sande & Matt Galloway



Nội dung

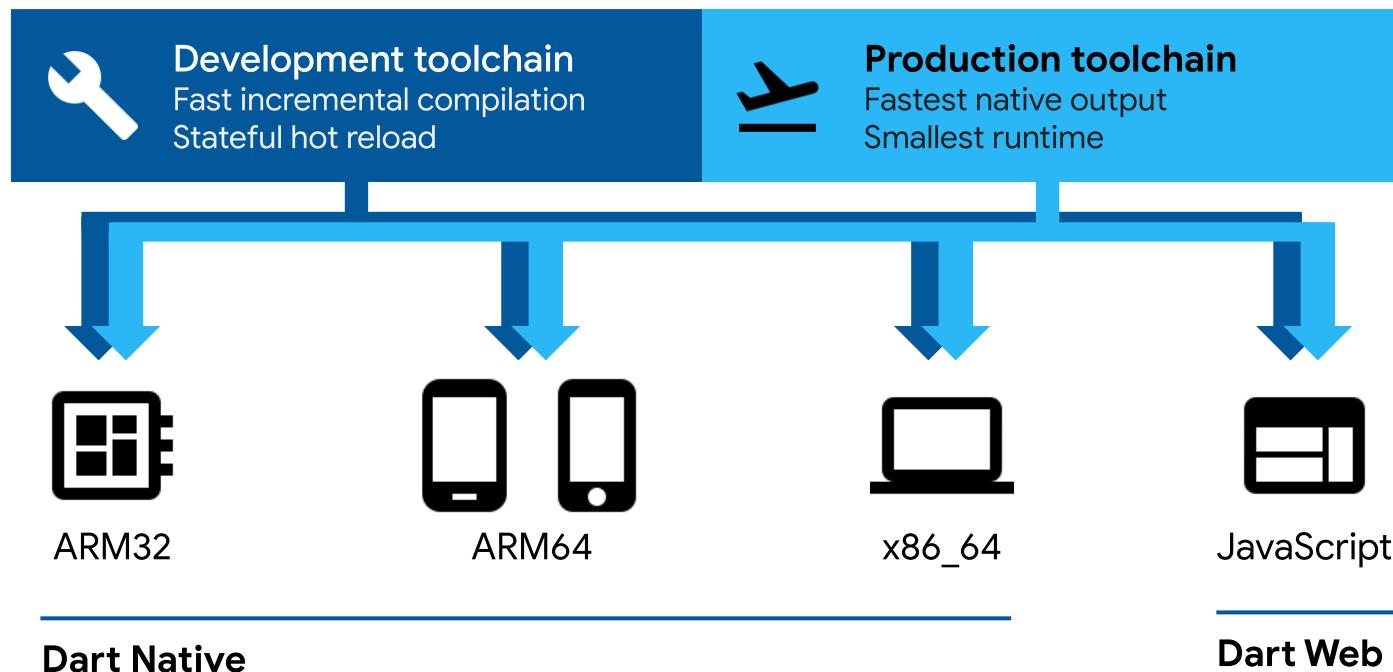
- Giới thiệu
- Hello, Dart!
- Cấu trúc điều khiển
- Hàm
- Lớp cơ bản
- Nullability
- Collection
- Lớp nâng cao
- Lập trình bất đồng bộ

Giới thiệu về Dart

- Một ngôn ngữ lập trình *hướng đối tượng* có cú pháp tương tự C (C-like) được phát triển bởi Google
- Dart có thể được dùng để viết các ứng dụng web, máy chủ, desktop, và di động
 - Có thể được biên dịch ra JavaScript (Dart Web) hoặc mã máy (machine code) cho desktop, thiết bị di động và thiết bị nhúng (Dart Native)

Giới thiệu về Dart

- Dart Native hỗ trợ cả *trình biên dịch chỉ trong thời gian* (*Just-in-Time, JIT*) dùng cho quá trình phát triển và *trình biên dịch trước thời gian* (*Ahead-Of-Time, AOT*) cho quá trình triển khai



Bạn cần những gì?

- DartPad (online): <https://dartpad.dev/>, hoặc cài đặt môi trường làm việc cục bộ
- Cài đặt Dart SDK (Software Development Kit):
<https://dart.dev/get-dart>
 - Nếu đã cài Flutter SDK thì **KHÔNG** cần cài đặt Dart SDK
- Trình soạn thảo code: dùng VS Code
(<https://code.visualstudio.com/download>)
 - Cài thêm phần mở rộng Dart cho VS Code

Dart SDK

```
PowerShell 7 + ^

slides> dart help
A command-line utility for Dart development.

Usage: dart <command|dart-file> [arguments]

Global options:
-h, --help                  Print this usage information.
-v, --verbose                Show additional command output.
--version                   Print the Dart SDK version.
--enable-analytics          Enable analytics.
--disable-analytics         Disable analytics.

Available commands:
analyze      Analyze Dart code in a directory.
compile      Compile Dart to various formats.
create       Create a new Dart project.
devtools     Open DevTools (optionally connecting to an existing application).
doc          Generate HTML API documentation from Dart documentation comments.
fix          Apply automated fixes to Dart source code.
format       Idiomatically format Dart source code.
migrate     Perform null safety migration on a project.
pub          Work with packages.
run          Run a Dart program.
test         Run tests for a project.
```

Phần mở rộng Dart trên VS Code

The screenshot shows the Visual Studio Code interface with the Marketplace extension search results for "dart". The main focus is the "Dart" extension by Dart Code, version v3.38.2, which has 4,538,999 installs and a 5-star rating from 58 reviews. The extension provides language support and a debugger for Dart in VS Code. It is currently enabled globally. Below the extension details, there are tabs for "Details", "Feature Contributions", "Changelog", and "Runtime Status". A sidebar on the right lists categories such as Programming Languages, Snippets, Linters, Formatters, and Debuggers. The "Introduction" section describes how Dart Code extends VS Code with support for the Dart programming language and its tools for editing, refactoring, running, and reloading Flutter mobile apps.

File Edit Selection View Go Run Terminal Help

EXTENSIONS: MARKETPLACE

Extension: Dart - Visual Studio Code

dart

Dart Dart language support and debugger for ... Dart Code

dart-import Fix Dart/Flutter's imports Luan Install

Dart (Syntax Highlighting ... Syntax highlighting for Dart and nothing e... oscarcs Install

Json to Dart Model Extension convert Json to Dart Model class hirantha Install

Flutter & Dart Utilities Official package of Academia do Flutter (B... Rodrigo Rahman Install

Dart Getters And Setters Dart Generate Getters And Setters Peter Haddad Install

Dart v3.38.2

Dart language support and debugger for Visual Studio Code.

Disable Uninstall Switch to Pre-Release Version

This extension is enabled globally.

Details Feature Contributions Changelog Runtime Status

chat discord chat gitter twitter dartcode help contribute

Categories

Programming Languages

Snippets Linters

Formatters

Debuggers

Introduction

Dart Code extends VS Code with support for the Dart programming language, and provides tools for effectively editing, refactoring, running, and reloading Flutter mobile apps.

Nội dung

- Giới thiệu
- Hello, Dart!
- Cấu trúc điều khiển
- Hàm
- Lớp cơ bản
- Nullability
- Collection
- Lớp nâng cao
- Lập trình bất đồng bộ

Hello, Dart!

```
Run | Debug
1 < void main() {
2   | print('Hello, Dart!');
3 }
```

[TERMINAL](#)[PROBLEMS](#)[OUTPUT](#)[DEBUG CONSOLE](#)

```
slides> dart hello.dart
Hello, Dart!
slides>
```

Hello, Dart!

- `dart compile exe hello.dart -o hello.exe`: biên dịch thành một tập tin thực thi độc lập
- `dart compile aot-snapshot hello.dart`: biên dịch thành môđun AOT không chứa dart runtime (dùng `dart aot runtime` để thực thi)
- `dart compile jit-snapshot hello.dart`: biên dịch thành môđun JIT (dùng `dart run` để thực thi)
- `dart compile js hello.dart`: biên dịch sang JavaScript

Hello, Dart!

- Cú pháp tương tự C
 - Hỗ trợ ghi chú trên một dòng với //, nhiều dòng với /* */
 - Câu lệnh kết thúc bằng dấu ;
 - Quy tắc định danh, các biểu thức, toán tử,...
 - Bắt buộc có một hàm **main** cho mỗi chương trình Dart
- Kiểu dữ liệu cơ bản: **bool, num (int, double), String**
 - Chuỗi được định nghĩa bằng dấu nháy đơn (‘’) hoặc dấu nháy kép (“”)
- Dart hỗ trợ tính năng suy luận kiểu (type inference)
 - Dùng từ khóa **var** để khai báo biến mà không cần chỉ định kiểu tường minh (được suy ra khi gán giá trị)

Hello, Dart!

- Định danh phân biệt hoa thường. Khuyến nghị dùng *lowerCamelCase* để đặt tên biến
- Hằng số:
 - Hằng số tại thời điểm biên dịch: dùng từ khóa **const**
 - Hằng số tại thời điểm thực thi: dùng từ khóa **final**



```
const myConstant = 10;  
myConstant = 0; // Not allowed.
```

```
final hoursSinceMidnight = DateTime.now().hour;
```

Kiểu dữ liệu

- Dart là ngôn ngữ hỗ trợ cả kiểu tĩnh và kiểu động
 - Không thể thay đổi kiểu dữ liệu cho biến khi đã xác định kiểu
 - Dùng từ khóa **dynamic** cho biến kiểu động



```
int myInteger = 10;  
myInteger = 3.14159;      // No, no, no.  
  
var someNumber = 10;  
someNumber = 15;          // OK  
someNumber = 3.14159;     // No, no, no.  
  
dynamic myVariable;  
myVariable = 10;           // OK  
myVariable = 3.14159;     // OK  
myVariable = 'ten';       // OK
```

Kiểu dữ liệu

- Chuyển đổi kiểu không tường minh

```
const hourlyRate = 19.5;
const hoursWorked = 10;
const totalCost = hourlyRate * hoursWorked; // double
```

- Chuyển đổi kiểu tường minh

```
// supertype as subtype
num someNumber = 3;
final someInt = someNumber as int;
print(someInt.isEven);

final someDouble = someNumber as double; // Error
final someDouble = someNumber.toDouble(); // OK
```

Kiểu dữ liệu

- Để đảm bảo chắc chắn kiểu dữ liệu của biến: chỉ định tường minh thông qua tên kiểu hoặc giá trị khởi tạo



```
const wantADouble = 3; // => int  
  
final actuallyDouble = 3.toDouble();  
const double actuallyDouble = 3;  
const wantADouble = 3.0;
```

Làm việc với chuỗi

```
● ● ●

String str1 = 'I like cats';
String str2 = "I like cats";
String str3 = "my cat's food";
str3 = 'my cat\'s food';

var message = 'Hello' + ' my name is ';
const name = 'Ray';
message += name;
// 'Hello my name is Ray'

final message = StringBuffer();
message.write('Hello');
message.write(' my name is ');
message.write('Ray');
message.toString();
// "Hello my name is Ray"
```

Làm việc với chuỗi



```
const name = 'Ray';
const introduction = 'Hello my name is $name';
// 'Hello my name is Ray'

const oneThird = 1 / 3;
final sentence = 'One third is ${oneThird.toStringAsFixed(3)}.';

const bigString = '''
You can have a string
that contains multiple
lines
by
doing this.'''
print(bigString);
```

Làm việc với chuỗi

```
● ● ●

const oneLine = 'This is only '
  'a single '
  'line '
  'at runtime.';
// The same as
const oneLine = 'This is only ' +
  'a single ' +
  'line ' +
  'at runtime.';

const twoLines = 'This is\ntwo lines.';
const rawString = r'My name \n is $name.';
```

Nội dung

- Giới thiệu
- Hello, Dart!
- Cấu trúc điều khiển
- Hàm
- Lớp cơ bản
- Nullability
- Collection
- Lớp nâng cao
- Lập trình bất đồng bộ

Cấu trúc điều khiển

- Lệnh if

```
● ● ●  
if (2 > 1) {  
    print('Yes, 2 is greater than 1.');//  
}  
  
const animal = 'Fox';  
if (animal == 'Cat' || animal == 'Dog') {  
    print('Animal is a house pet.');//  
} else {  
    print('Animal is not a house pet.');//  
}
```

```
● ● ●  
const trafficLight = 'yellow';  
var command = '';  
if (trafficLight == 'red') {  
    command = 'Stop';  
} else if (trafficLight == 'yellow') {  
    command = 'Slow down';  
} else if (trafficLight == 'green') {  
    command = 'Go';  
} else {  
    command = 'INVALID COLOR!';  
}  
print(command);
```

Cấu trúc điều khiển

- Toán tử điều kiện ba ngôi:
(condition) ? valueIfTrue : valueIfFalse;



```
const score = 83;
String message;
if (score >= 60) {
    message = 'You passed';
} else {
    message = 'You failed';
}

const score = 83;
const message = (score >= 60) ? 'You passed' : 'You failed';
```

Cấu trúc điều khiển

- Lệnh switch

```
● ● ●

const weather = 'cloudy';
switch (weather) {
  case 'sunny':
    print('Put on sunscreen.');
    break;
  case 'snowy':
    print('Get your skis.');
    break;
  case 'cloudy':
  case 'rainy':
    print('Bring an umbrella.');
    break;
  default:
    print("I'm not familiar with that weather.");
}
```

Cấu trúc điều khiển

- Lệnh switch

```
● ● ●  
enum Weather { sunny, snowy, cloudy, rainy }  
  
const weatherToday = Weather.cloudy;  
switch (weatherToday) {  
    case Weather.sunny:  
        print('Put on sunscreen.');//  
        break;  
    case Weather.snowy:  
        print('Get your skis.');//  
        break;  
    case Weather.cloudy:  
    case Weather.rainy:  
        print('Bring an umbrella.');//  
        break;  
}
```

Cấu trúc điều khiển

- Vòng lặp: for, while, do-while tương tự như C
- Ngoài ra Dart còn hỗ trợ vòng lặp for-in

```
● ● ●

const myNumbers = [1, 2, 3];
for (var myNumber in myNumbers) {
    print(myNumber);
}

myNumbers.forEach((number) => print(number));

myNumbers.forEach((number) {
    print(number);
});
```

Nội dung

- Giới thiệu
- Hello, Dart!
- Cấu trúc điều khiển
- Hàm
- Lớp cơ bản
- Nullability
- Collection
- Lớp nâng cao
- Lập trình bất đồng bộ

Hàm

- Hàm trong Dart

```
Return type      Function name      Parameter type      Parameter name  
String          compliment(int number) {  
    return '$number is a very nice number.';  
}  
  
                                         ↑  
                                         Return value
```

Hàm

- Tham số tùy chọn: đặt tham số trong cặp ngoặc []

```
● ● ●

String fullName(String first, String last, [String? title]) {
    if (title != null) {
        return '$title $first $last';
    } else {
        return '$first $last';
    }
}

print(fullName('Ray', 'Wenderlich'));
// Ray Wenderlich
print(fullName('Albert', 'Einstein', 'Professor'));
// Professor Albert Einstein
```

Hàm

- Giá trị mặc định cho tham số



```
bool withinTolerance(int value, [int min = 0, int max = 10]) {  
    return min <= value && value <= max;  
}  
  
withinTolerance(5) // true  
withinTolerance(15) // false  
withinTolerance(9, 7, 11) // true
```

Hàm

- Tham số có tên (named parameter) {}, mặc định sẽ là tham số tùy chọn



```
bool withinTolerance(int value, {int min = 0, int max = 10}) {
    return min <= value && value <= max;
}

withinTolerance(9, min: 7, max: 11) // true
withinTolerance(9, max: 11, min: 7) // true
withinTolerance(5) // true
withinTolerance(15) // false
withinTolerance(5, min: 7) // false
withinTolerance(15, max: 20) // true
```

Hàm

- Dùng từ khóa **required** để chỉ định tham số có tên là bắt buộc



```
bool withinTolerance({  
    required int value,  
    int min = 0,  
    int max = 10,  
}) {  
    return min <= value && value <= max;  
}
```

Hàm

- Hàm trong Dart là công dân hạng nhất
 - Có thể xem hàm như kiểu dữ liệu
 - Gán hàm vào biến
 - Truyền hàm như tham số cho hàm khác
 - Trả về hàm từ hàm khác



```
int number = 4;
String greeting = 'hello';
bool isHungry = true; // anonymous function
Function multiply = (int a, int b) {
    return a * b;
};

void namedFunction(Function anonymousFunction) { // function body
}

Function namedFunction() {
    return () {
        print('hello');
    };
}
```

Hàm

- Bao đóng (closure): hàm cùng với giá trị các biến trong phạm vi mà nó được định nghĩa



```
Function countingFunction() {  
    var counter = 0;  
    int incrementCounter() {  
        counter += 1;  
        return counter;  
    }  
  
    return incrementCounter;  
}  
  
final counter1 = countingFunction();  
final counter2 = countingFunction();  
  
print(counter1()); // 1  
print(counter2()); // 1  
print(counter1()); // 2  
print(counter1()); // 3  
print(counter2()); // 2
```

Hàm

- Nếu thân hàm chỉ có một dòng lệnh, có thể dùng ký pháp mũi tên (arrow syntax): (parameters) => expression;

```
final multiply = (int a, int b) {  
    return a * b;  
};  
// the same as  
final multiply = (int a, int b) => a * b;  
  
print(multiply(2, 3)); // 6
```

Nội dung

- Giới thiệu
- Hello, Dart!
- Cấu trúc điều khiển
- Hàm
- Lớp cơ bản
- Nullability
- Collection
- Lớp nâng cao
- Lập trình bất đồng bộ

Lớp

- Lớp kết hợp dữ liệu và hàm (trong ngữ cảnh của lớp gọi là phương thức) thành một cấu trúc đơn nhất

```
class MyClass {  
    var myProperty = 'Hello, Dart!'; ← data  
  
    // constructor  
    MyClass(); ← functions  
  
    void myMethod() {  
        print(myProperty);  
    }  
}
```

Lớp

- Định nghĩa lớp, tạo đối tượng và truy xuất các thuộc tính

```
void main() {
    // Creating objects,
    //   the new keyword is optional
    final User user1 = new User();
    user1.name = "Ray";
    user1.id = 42;

    // Cascade notation
    final User user2 = new User()
        ..name = "Peter"
        ..id = 22;
}

class User {
    int id = 0;
    String name = "";
}
```

Lớp

- Biểu diễn dạng chuỗi của đối tượng

```
● ● ●

class User {
    int id = 0;
    String name = '';

    @override
    String toString() {
        return 'User(id: $id, name: $name)';
    }
}
```

Lớp

- Biểu diễn dạng chuỗi của đối tượng: dùng định dạng JSON

```
● ● ●

class User {
    int id = 0;
    String name = '';

    String toJson() {
        return '{"id":$id,"name":"$name"}';
    }

    @override
    String toString() {
        return 'User(id: $id, name: $name)';
    }
}
```

Lớp

- Hàm xây dựng (constructor)
 - Hàm xây dựng mặc định: nếu lớp không có định nghĩa hàm xây dựng một cách tường minh, Dart cung cấp một hàm xây dựng mặc định
 - Hàm xây dựng tùy biến: tên trùng với tên lớp



```
class User {  
    // long-form constructor  
    User(int id, String name) {  
        this.id = id;  
        this.name = name;  
    }  
  
    int id = 0;  
    String name = '';  
  
    // ...  
}
```

Lớp

- Hàm xây dựng (constructor)
 - Hàm xây dựng mặc định: nếu lớp không có định nghĩa hàm xây dựng một cách tường minh, Dart cung cấp một hàm xây dựng mặc định
 - Hàm xây dựng tùy biến: tên trùng với tên lớp



```
class User {  
    // short-form constructor  
    User(this.id, this.name);  
  
    int id = 0;  
    String name = '';  
  
    // ...  
}
```

Lớp

- Hàm xây dựng có tên
(named constructor)

```
class User {  
    User(this.id, this.name);  
  
    User.anonymous() {  
        id = 0;  
        name = 'anonymous';  
    }  
  
    int id = 0;  
    String name = '';  
  
    // ...  
}  
  
void main() {  
    final anon = User.anonymous();  
    print(anon);  
}
```

Lớp

- Hàm xây dựng chuyển tiếp
(forwarding constructor)



```
class User {  
    User(this.id, this.name);  
  
    User.anonymous(): this(0, 'anonymous');  
  
    int id;  
    String name;  
  
    // ...  
}  
  
void main() {  
    final anon = User.anonymous();  
    print(anon);  
}
```

Lớp

- Hàm xây dựng: tham số tùy chọn, tham số có tên

```
● ● ●

class User {
    User({this.id = 0, this.name = 'anonymous'});

    User.anonymous(): this();

    int id;
    String name;

    // ...
}

void main() {
    final anon = User.anonymous();
    print(anon);
}
```

Lớp

- Các thuộc tính private: có tên bắt đầu bằng dấu gạch dưới _ (underscore)
 - Tính private ở đây sẽ được áp dụng trong phạm vi thư viện (tập tin) chứ không phải ở phạm vi lớp
- Danh sách khởi tạo (initializer list): được thực thi trước thân hàm xây dựng



```
class User {  
    User({int id = 0, String name = 'anonymous'})  
        : _id = id,  
          _name = name;  
  
    User.anonymous(): this();  
  
    int _id;  
    String _name;  
  
    // ...  
}
```

Lớp

- Thuộc tính bất biến (immutable): dùng từ khóa **final** khi định nghĩa thuộc tính
- Hàm xây dựng hằng (constant constructor): nếu các thuộc tính là bất biến có thể thêm từ khóa **const** vào trước hàm xây dựng
 - Các hằng đối tượng có lợi về hiệu năng

```
● ● ●

class User {
  const User({int id = 0, String name = 'anonymous'})
    : _id = id,
      _name = name;

  const User.anonymous(): this();

  final int _id;
  final String _name;

  // ...
}

void main() {
  const user = User(id: 42, name: 'Ray');
  const anonymousUser = User.anonymous();
}
```

Lớp

- Hàm xây dựng sinh (generative constructor): chỉ có thể trả về một thể hiện mới thuộc lớp
 - Các hàm xây dựng đã xét
- Hàm xây dựng factory (factory constructor): có thể trả về các đối tượng đang tồn tại của lớp hoặc của lớp con
 - Định nghĩa dùng từ khóa **factory**



```
class User {  
    // ...  
    factory User.fromJson(Map<String, Object> json)  
    {  
        final userId = json['id'] as int;  
        final userName = json['name'] as String;  
        return User(id: userId, name: userName);  
    }  
}  
  
void main() {  
    final map = {'id': 10, 'name': 'Manda'};  
    final manda = User.fromJson(map);  
}
```

Lớp

- Getter: phương thức đặc biệt dùng từ khóa **get** trước tên thuộc tính và trả về một giá trị

```
● ● ●  
  
class User {  
    // ...  
    final int _id;  
    final String _name;  
    // ...  
    int get id => _id;  
    String get name => _name;  
}  
  
void main() {  
    const ray = User(id: 42, name: 'Ray');  
    print(ray.id); // 42  
    print(ray.name); // Ray  
}
```

Lớp

- Setter: phương thức đặc biệt định nghĩa dùng từ khóa **set**, đặt giá trị cho thuộc tính



```
class Email {  
    var _address = '';  
    String get value => _address;  
    set value(String address) => _address = address;  
}  
  
void main() {  
    final email = Email();  
    email.value = 'ray@example.com';  
    final emailString = email.value;  
}
```

Lớp

- Thành viên tĩnh
 - Định nghĩa dùng từ khóa **static**
 - Thuộc về lớp thay vì đối tượng



```
class SomeClass {  
    static int myProperty = 0;  
    static void myMethod() {  
        print('Hello, Dart!');  
    }  
}  
  
void main() {  
    final value = SomeClass.myProperty;  
    SomeClass.myMethod();  
}
```

Nội dung

- Giới thiệu
- Hello, Dart!
- Cấu trúc điều khiển
- Hàm
- Lớp cơ bản
- Nullability
- Collection
- Lớp nâng cao
- Lập trình bất đồng bộ

Nullability

- Giá trị **null** có ý nghĩa gì?
 - Không có giá trị, vắng mặt của giá trị
- Từ phiên bản 2.12, Dart giới thiệu tính năng “sound null safety”: các kiểu dữ liệu mặc định không chứa giá trị null
 - Giúp phòng tránh một số lỗi xảy ra ngay tại thời điểm biên dịch

```
int myNumber = null; // Error
```

Nullability

- Kiểu dữ liệu có chứa giá trị null (nullable type): thêm dấu ? vào cuối tên kiểu

```
int? myInt = null;  
double? myDouble = null;  
bool? myBool = null;  
String? myString = null;  
User? myUser = null;
```

Nullability

- Làm việc với các kiểu dữ liệu nullable
 - Trả về giá trị thay thế nếu null: ??
 - Gán giá trị nếu null: ??=
 - Truy xuất thuộc tính, gọi phương thức nếu không null: ?.
 - Xác nhận không null: !
 - Truy xuất phần tử trong mảng nếu không null: ?[]

Nullability

- Làm việc với các kiểu dữ liệu nullable



```
String? message; // message = null
final text = message ?? 'Error'; // text = 'Error'

double? fontSize;
fontSize ??= 20.0; // fontSize = 20.0
fontSize ??= 30.0; // fontSize = 20.0

int? age;
print(age?.isNegative); // null

bool? isBeautiful(String? item) {
    if (item == 'flower') {
        return true;
    } else if (item == 'garbage') {
        return false;
    }
    return null;
}
bool flowerIsBeautiful = isBeautiful('flower')!;
// As bool is a subtype of bool?
bool flowerIsBeautiful = isBeautiful('flower') as bool;
```

Nullability

- Làm việc với các kiểu dữ liệu nullable
 - Khi tạo đối tượng từ lớp, Dart yêu cầu tất cả các biến thành viên không thể nhận giá trị null (non-nullable) phải được khởi tạo
 - Dùng từ khóa **late** để trì hoãn quá trình khởi tạo biến thành viên (lazy initialization) cho đến khi biến được truy xuất lần đầu tiên



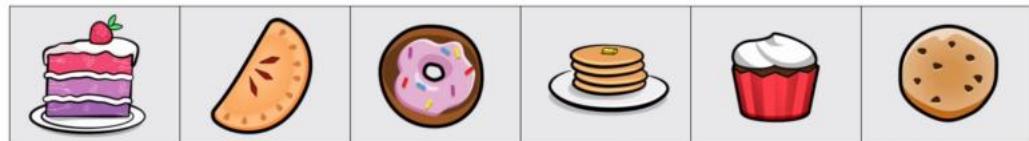
```
class User {  
    User(this.name);  
  
    final String name;  
    late final int _secretNumber = _calculateSecret();  
  
    int _calculateSecret() {  
        return name.length + 42;  
    }  
}
```

Nội dung

- Giới thiệu
- Hello, Dart!
- Cấu trúc điều khiển
- Hàm
- Lớp cơ bản
- Nullability
- **Collection**
- Lớp nâng cao
- Lập trình bất đồng bộ

Các cấu trúc collection

- Các cấu trúc collection chính: List/Array, Set và Map



0 1 2 3 4 5

List/Array



Set

{: 500 calories,
: 150 calories, ...}

Map<Food, Calories>
Key type Value type

Map

Các cấu trúc collection

- Các thao tác cơ bản với List
 - Tạo một list các String



```
var desserts = ['cookies', 'cupcakes', 'donuts', 'pie'];
List<String> snacks = [];
var snacks = <String>[];
```

Các cấu trúc collection

- Các thao tác cơ bản với List
 - Truy xuất, hiệu chỉnh các phần tử



```
final index = desserts.indexOf('pie');
final value = desserts[index];

desserts[1] = 'cake';

desserts.add('brownies');
desserts.remove('cake');
```

Các cấu trúc collection

- Các thao tác cơ bản với List
 - List bất biến (immutable): dùng từ khóa **final**



```
final desserts = ['cookies', 'cupcakes', 'donuts', 'pie'];
desserts = [];// not allowed
desserts = ['cake', 'ice cream'];// not allowed
desserts = someOtherList;// not allowed

desserts.remove('cookies');// OK
desserts.remove('cupcakes');// OK
desserts.add('ice cream');// OK
```

Các cấu trúc collection

- Các thao tác cơ bản với List
 - List bất biến sâu (deeply immutable): dùng từ khóa **const** hoặc hàm xây dựng **List.unmodifiable()**



```
// Compile-time constant
const desserts = ['cookies', 'cupcakes', 'donuts', 'pie'];
// Or
final desserts = const ['cookies', 'cupcakes', 'donuts', 'pie'];

desserts.add('brownie');      // not allowed
desserts.remove('pie');       // not allowed
desserts[0] = 'fudge';        // not allowed

// Runtime constant
final modifiableList = [DateTime.now(), DateTime.now()];
final unmodifiableList = List.unmodifiable(modifiableList);
```

Các cấu trúc collection

- Các thao tác cơ bản với List
 - Các thuộc tính của List



```
const drinks = ['water', 'milk', 'juice', 'soda'];

drinks.first          // water
drinks.last           // soda

drinks.isEmpty        // false
drinks.isNotEmpty     // true

drinks.length == 0    // false
drinks.length > 0     // true
```

Các cấu trúc collection

- Các thao tác cơ bản với List
 - Lặp qua các phần tử trong List



```
const desserts = ['cookies', 'cupcakes', 'donuts', 'pie'];

for (var dessert in desserts) {
    print(dessert);
}

desserts.forEach((dessert) => print(dessert));
desserts.forEach(print);    // a tear-off
```

Các cấu trúc collection

- Các thao tác cơ bản với List
 - Toán tử làm phẳng (spread operator): ...



```
const pastries = ['cookies', 'croissants'];
const candy = ['Junior Mints', 'Twizzlers', 'M&Ms'];

const desserts = ['donuts', ...pastries, ...candy];

List<String>? coffees;
final hotDrinks = ['milk tea', ...?coffees];
```

Các cấu trúc collection

- Các thao tác cơ bản với List
 - Chuyển đổi kiểu List<dynamic>

```
final List<dynamic> list1 = ['cookies', 'donuts', 'pie'];

// Use the spread operator
final List<String> list2 = [...list1];

// Use cast<T>()
final List<String> list3 = list1.cast<String>();

// Use map() to convert each element
final List<String> list4 = list1.map((e) => e as String).toList();
```

Các cấu trúc collection

- Các thao tác cơ bản với List
 - Collection if, collection for



```
const peanutAllergy = true;
const candy = [
  'Junior Mints',
  'Twizzlers',
  if (!peanutAllergy) 'Reeses',
];

const deserts = ['gobi', 'sahara', 'arctic'];
var bigDeserts = [
  'ARABIAN',
  for (var desert in deserts) desert.toUpperCase(),
];
```

Các cấu trúc collection

- Set: tập các phần tử duy nhất (dựa trên Object.== và Object.hashCode)
- Các thao tác cơ bản với set
 - Tạo một set các giá trị int



```
final Set<int> someSet = {};
final someSet = <int>{};
final anotherSet = {1, 2, 3, 1};
print(anotherSet); // {1, 2, 3}
```

Các cấu trúc collection

- Các thao tác cơ bản với set
 - Kiểm tra tồn tại, thêm, xóa phần tử



```
final anotherSet = {1, 2, 3, 1};  
print(anotherSet.contains(1)); // true  
print(anotherSet.contains(99)); // false  
  
final someSet = <int>{};  
someSet.add(42);  
someSet.add(2112);  
someSet.add(42);  
  
someSet.remove(2112);  
  
someSet.addAll([1, 2, 3, 4]);  
// {42, 1, 2, 3, 4}
```

Các cấu trúc collection

- Các thao tác cơ bản với set
 - Giao và hợp



```
final setA = {8, 2, 3, 1, 4};  
final setB = {1, 6, 5, 4};  
  
final intersection = setA.intersection(setB);  
// {1, 4}  
final union = setA.union(setB);  
// {8, 2, 3, 1, 4, 6, 5}
```

Các cấu trúc collection

- Các thao tác cơ bản với set
 - collection if
 - collection for
 - for-in
 - forEach
 - spread operator

Các cấu trúc collection

- Map: các cặp key-value (có thể hiểu như tập các biến)
- Các thao tác cơ bản với map
 - Tạo một map rỗng có key là String, value là int



```
final Map<String, int> emptyMap = {};  
final emptyMap = <String, int>{};  
  
final emptySomething = {};// Map<dynamic, dynamic>
```

Các cấu trúc collection

- Các thao tác cơ bản với map
 - Khởi tạo map với các giá trị



```
final inventory = {  
    'cakes': 20,  
    'pies': 14,  
    'donuts': 37,  
    'cookies': 141,  
};  
  
final digitToWord = {  
    1: 'one',  
    2: 'two',  
    3: 'three',  
    4: 'four',  
};
```

Các cấu trúc collection

- Các thao tác cơ bản với map
 - Truy xuất phần tử: map trả về null nếu key không tồn tại
 - Thêm, cập nhật, xóa phần tử



```
final numberOfCakes = inventory['cakes']; // int?  
print(numberOfCakes?.isEven);  
  
inventory['brownies'] = 3;  
inventory['cakes'] = 1;  
inventory.remove('cookies');
```

Các cấu trúc collection

- Các thao tác cơ bản với map
 - Các thuộc tính của map
 - Kiểm tra key hoặc value có tồn tại



```
inventory.isEmpty          // false
inventory.isNotEmpty       // true
inventory.length           // 4

print(inventory.keys);     // (cakes, pies, donuts, brownies)
print(inventory.values);   // (1, 14, 37, 3)

print(inventory.containsKey('pies')); // true
print(inventory.containsValue(42));    // false
```

Các cấu trúc collection

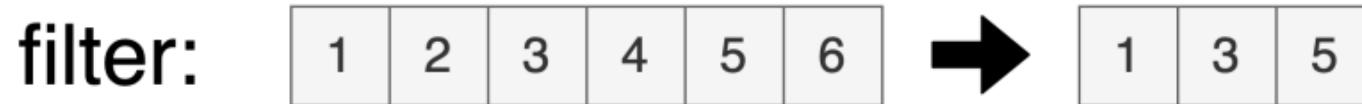
- Các thao tác cơ bản với map
 - Lặp qua các phần tử trong map



```
for (var item in inventory.keys) {  
    print(inventory[item]);  
}  
  
inventory.forEach((key, value) => print('$key -> $value'));  
  
for (final entry in inventory.entries) {  
    print('${entry.key} -> ${entry.value}');  
}
```

Các cấu trúc collection

- Các phương thức bậc cao trên collection
 - Phương thức bậc cao (higher order method): phương thức nhận vào hàm làm tham số và/hoặc trả về một hàm khác



Các cấu trúc collection

- Các phương thức bậc cao trên collection
 - Phương thức map (List, Set, Map): nhận vào một hàm vô danh làm tham số và trả về một tập các phần tử (kiểu Iterable) là kết quả xử lý của hàm trên từng phần tử trong tập ban đầu



```
const numbers = [1, 2, 3, 4];
final squares = numbers.map((number) => number * number);
print(squares.toList()); // [1, 4, 9, 16]
```

Các cấu trúc collection

- Các phương thức bậc cao trên collection
 - Phương thức where (List, Set): nhận vào một hàm vô danh và trả về một tập các phần tử là các phần tử trong tập ban đầu mà tại đó hàm nhận vào cho kết quả là true



```
const numbers = [1, 2, 3, 4];
final squares = numbers.map((number) => number * number);
print(squares.toList()); // [1, 4, 9, 16]

final evens = squares.where((square) => square.isEven);
print(evens.toList()); // [4, 16]
```

Các cấu trúc collection

- Các phương thức bậc cao trên collection

- Phương thức reduce/fold (List, Set): kết hợp các phần tử thành một phần tử đơn (dựa trên hàm được cung cấp). fold có độ tin cậy cao hơn (không gặp lỗi khi tập phần tử là rỗng)



```
const amounts = [199, 299, 299, 199, 499];
final total = amounts.reduce((sum, element) => sum + element);

const amounts = [199, 299, 299, 199, 499];
final total = amounts.fold(
  0,    // init value
  (int sum, element) => sum + element,
);
```

Các cấu trúc collection

- Các phương thức bậc cao trên collection
 - Sắp xếp list



```
final desserts = ['cookies', 'pie', 'donuts', 'brownies'];
desserts.sort(); // [brownies, cookies, donuts, pie]

var dessertsReversed = desserts.reversed;
// (pie, donuts, cookies, brownies)

desserts.sort((d1, d2) => d1.length.compareTo(d2.length));
// [pie, donuts, cookies, brownies]
```

Các cấu trúc collection

- Các phương thức bậc cao trên collection
 - Kết hợp các phương thức bậc cao

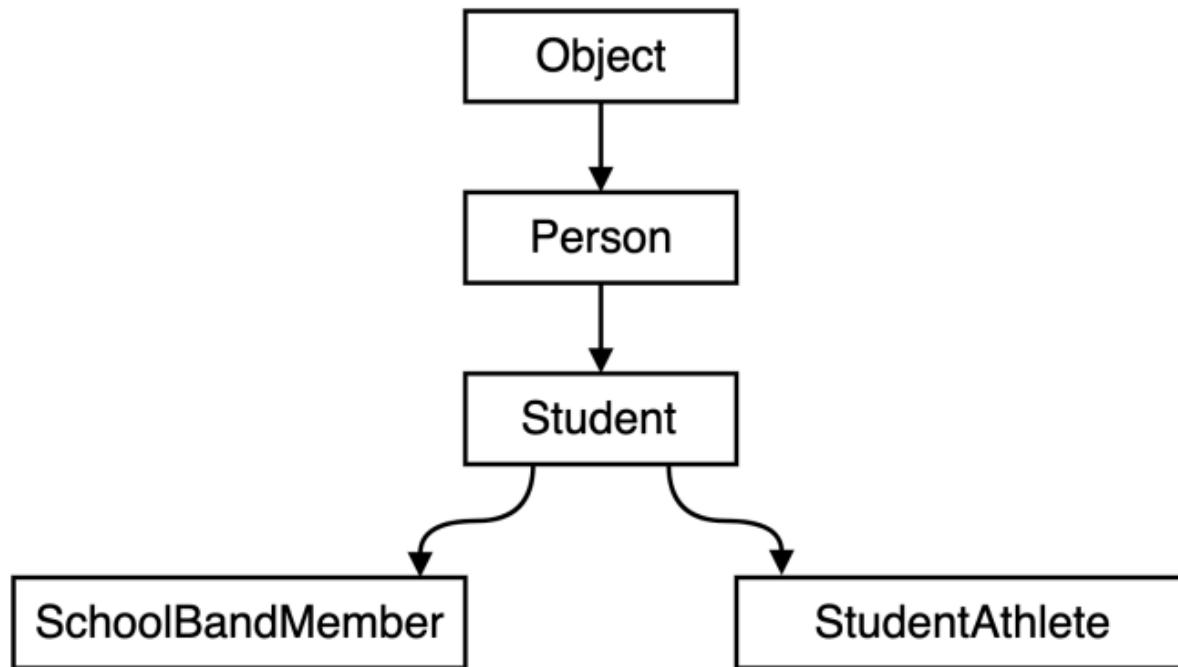


```
const desserts = ['cake', 'pie', 'donuts', 'brownies'];
final bigTallDesserts = desserts
    .where((dessert) => dessert.length > 5)
    .map((dessert) => dessert.toUpperCase());
// (DONUTS, BROWNIES)
```

Nội dung

- Giới thiệu
- Hello, Dart!
- Cấu trúc điều khiển
- Hàm
- Lớp cơ bản
- Nullability
- Collection
- Lớp nâng cao
- Lập trình bất đồng bộ

Thừa kế



Thừa kế



```
enum Grade { A, B, C, D, F }

class Person {
    Person(this.givenName, this.surname);

    String givenName;
    String surname;
    String get fullName => '$givenName $surname';

    @override
    String toString() => fullName;
}
```



```
class Student extends Person {
    Student(String givenName, String surname)
        : super(givenName, surname);

    var grades = <Grade>[];

    @override
    String get fullName => '$surname, $givenName';
}
```

Thừa kế



```
enum Grade { A, B, C, D, F }

class Person {
    Person(this.givenName, this.surname);

    String givenName;
    String surname;
    String get fullName => '$givenName $surname';

    @override
    String toString() => fullName;
}
```



```
class Student extends Person {
    Student(super.givenName, super.surname);

    var grades = <Grade>[];

    @override
    String get fullName => '$surname, $givenName';
}
```

Dart 2.17: hỗ trợ các tham số super

Thừa kế



```
class SchoolBandMember extends Student {  
    SchoolBandMember(String givenName, String surname)  
        : super(givenName, surname);  
    static const minimumPracticeTime = 2;  
}  
  
class StudentAthlete extends Student {  
    StudentAthlete(String givenName, String surname)  
        : super(givenName, surname);  
    bool get isEligible => grades.every((grade) => grade != Grade.F);  
}
```

Thừa kế

```
● ● ●  
  
final jane = Student('Jane', 'Snow');  
final jessie = SchoolBandMember('Jessie', 'Jones');  
final marty = StudentAthlete('Marty', 'McFly');  
  
final students = [jane, jessie, marty];  
  
print(jessie is Object);           // true  
print(jessie is Person);          // true  
print(jessie is Student);         // true  
print(jessie is SchoolBandMember); // true  
print(jessie is! StudentAthlete); // true
```

Thừa kế

- Nên sử dụng quan hệ hợp thành (composition) thay vì quan hệ thừa kế (inheritance)
 - Các lớp trong thừa kế có mối quan hệ chặt chẽ với nhau gây khó khăn cho việc thay đổi
 - Trường hợp Jessie vừa là thành viên ban nhạc, vừa là vận động viên thì làm thế nào?
 - Thay vì tạo lớp SchoolBandMember và StudentAthlete, có thể định nghĩa một list các role cho Student



```
class Student {  
    List<Role>? roles;  
}
```

Lớp trừu tượng



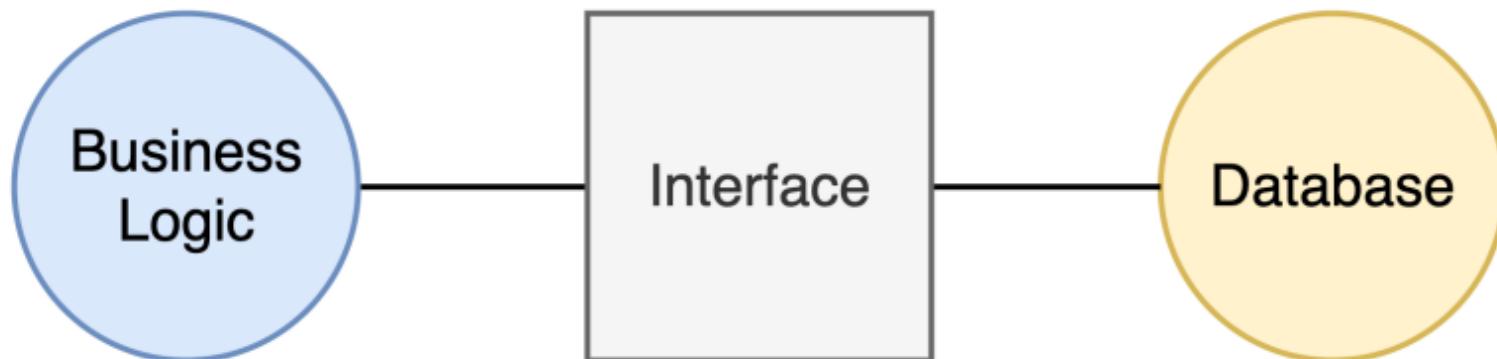
```
abstract class Animal {  
    bool isAlive = true;  
    void eat();  
    void move();  
  
    @override  
    String toString() {  
        return "I'm a ${runtimeType}";  
    }  
}  
  
final animal = Animal(); // Error
```



```
class Platypus extends Animal {  
    @override  
    void eat() {  
        print('Munch munch');  
    }  
    @override  
    void move() {  
        print('Glide glide');  
    }  
    void layEggs() {  
        print('Plop plop');  
    }  
}
```

Giao diện

- Giao diện (interface) là một miêu tả về cách hai bên giao tiếp với nhau
 - Ví dụ số điện thoại là một loại giao diện
 - Một từ khác tương đồng với giao diện là giao thức (protocol)
 - Dart dùng abstract class để cài đặt giao diện



Giao diện

- Tạo, cài đặt và sử dụng interface

```
abstract class DataRepository {  
    double? fetchTemperature(String city);  
}  
  
class FakeWebServer implements DataRepository {  
    @override  
    double? fetchTemperature(String city) {  
        return 42.0;  
    }  
}  
  
final DataRepository repository = FakeWebServer();  
final temperature = repository.fetchTemperature('Berlin');
```

Giao diện

- Tạo, cài đặt và sử dụng interface



```
abstract class DataRepository {  
    factory DataRepository() => FakeWebServer();  
    double? fetchTemperature(String city);  
}  
  
final repository = DataRepository();  
final temperature = repository.fetchTemperature('Manila');
```

Giao diện

- **extends vs implements**

class SomeClass **extends** AnotherClass {}

SomeClass có thể truy xuất các logic, các biến trong AnotherClass

class SomeClass **implements** AnotherClass {}

SomeClass phải cài đặt tất cả phương thức và các biến trong AnotherClass

Giao diện

- extends vs implements



```
class AnotherClass {  
    int myField = 42;  
    void myMethod() => print(myField);  
}  
  
class SomeClass extends AnotherClass {}  
  
final someClass = SomeClass();  
print(someClass.myField); // 42  
someClass.myMethod(); // 42
```



```
class SomeClass implements AnotherClass  
{  
    @override  
    int myField = 0;  
  
    @override  
    void myMethod() => print('Hello');  
}  
  
final someClass = SomeClass();  
print(someClass.myField); // 0  
someClass.myMethod(); // Hello
```

Mixin

- Ngoài thừa kế, Dart còn hỗ trợ một tính năng khác giúp tái sử dụng code là mixin
 - Định nghĩa mixin với từ khóa **mixin**
 - *Các lớp có thể được sử dụng như một mixin*
 - Sử dụng mixin với từ khóa **with**

Mixin



```
 mixin EggLayer {  
     void layEggs() {  
         print('Plop plop');  
     }  
 }  
  
 mixin Flyer {  
     void fly() {  
         print('Swoosh swoosh');  
     }  
 }
```



```
 class Robin extends Bird with EggLayer, Flyer {}  
  
 class Platypus extends Animal with EggLayer {  
     @override  
     void eat() {  
         print('Munch munch');  
     }  
     @override  
     void move() {  
         print('Glide glide');  
     }  
 }  
  
 final platypus = Platypus();  
 final robin = Robin();  
 platypus.layEggs();  
 robin.layEggs();
```

Các phương thức mở rộng

- Dart hỗ trợ tính năng phương thức mở rộng (extension method) cho phép thêm chức năng vào các lớp sẵn có

```
● ● ●

String encode(String input) {
    final output = StringBuffer();
    for (final codePoint in input.runes) {
        output.writeCharCode(codePoint + 1);
    }
    return output.toString();
}

final original = 'abc';
final secret = encode(original);
print(secret); // bcd
```

Các phương thức mở rộng

- Dart hỗ trợ tính năng phương thức mở rộng (extension method) cho phép thêm chức năng vào các lớp sẵn có

```
extension on String {  
    String get encoded {  
        final output = StringBuffer();  
        for (final codePoint in runes) {  
            output.writeCharCode(codePoint + 1);  
        }  
        return output.toString();  
    }  
}  
  
final secret = 'abc'.encoded;  
print(secret); // bcd
```

Các phương thức mở rộng

```
extension on String {  
    String get encoded => _code( 1 );  
    String get decoded => _code( -1 );  
  
    String _code( int step ) {  
        final output = StringBuffer( );  
        for ( final codePoint in runes ) {  
            output.writeCharCode( codePoint + step );  
        }  
        return output.toString( );  
    }  
}  
  
final original = 'I like extensions!';  
final secret = original.encoded;  
final revealed = secret.decoded;  
print(secret);      // J!mjlf!fyufotjpot"  
print(revealed);    // I like extensions!
```

Nội dung

- Giới thiệu
- Hello, Dart!
- Cấu trúc điều khiển
- Hàm
- Lớp cơ bản
- Nullability
- Collection
- Lớp nâng cao
- Lập trình bất đồng bộ

Lập trình bất đồng bộ

- Song song (parallelism) vs. cạnh tranh (concurrency)
 - Song song (parallelism): nhiều tác vụ chạy tại cùng một thời điểm trên nhiều bộ nhớ lý/core CPU
 - Cạnh tranh (concurrency): nhiều tác vụ chạy *luân phiên* trên cùng một CPU đơn



Lập trình bất đồng bộ

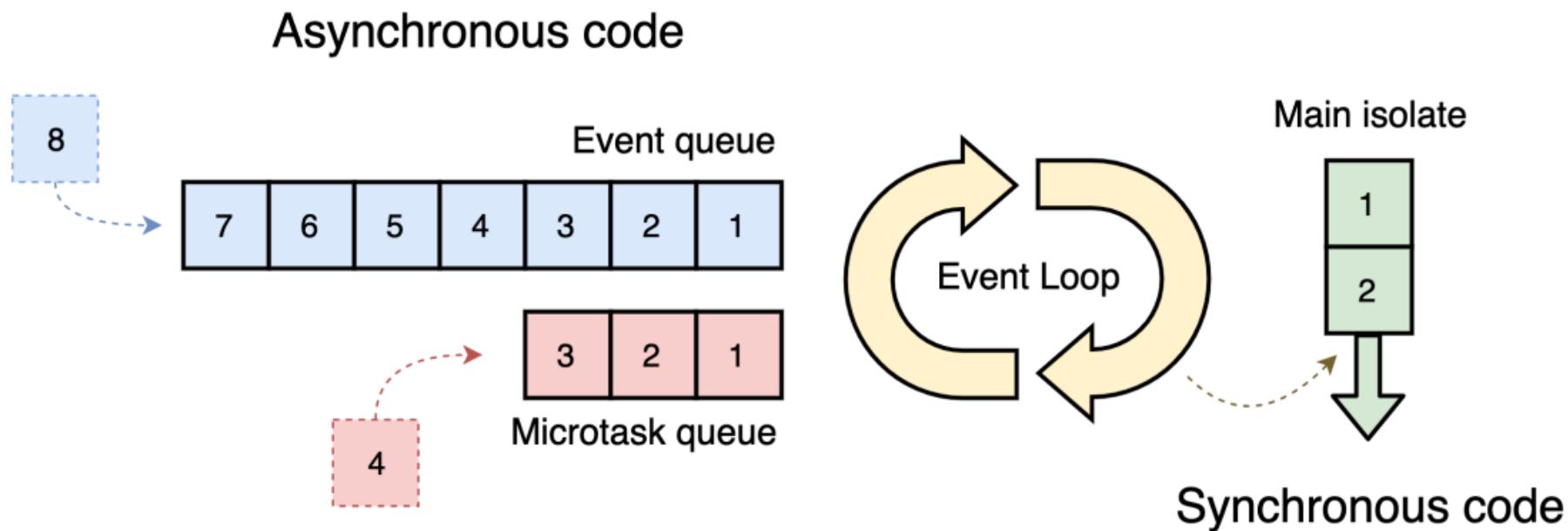
- Dart là một ngôn ngữ *đơn luồng* (single-threaded). Luồng thực thi của Dart chạy trong một **isolate**
 - Mỗi isolate có vùng nhớ được cấp phát riêng
- Flutter cần cập nhật UI 60 lần/s. Mỗi khoảng thời gian cập nhật gọi là một khung (frame) => Thời gian cập nhật UI cho mỗi khung là 16ms
 - Việc cập nhật UI thường không tốn nhiều thời gian như vậy
 - Lập lịch thực thi các tác vụ khác trong thời gian khác

Lập trình bất đồng bộ

- Mã lệnh đồng bộ (synchronous) vs. bất đồng bộ (asynchronous)
 - Mã lệnh đồng bộ (synchronous): mỗi chỉ thị được thực thi theo thứ tự, một dòng mã lệnh theo ngay sau dòng mã lệnh trước đó
 - Mã lệnh bất đồng bộ (asynchronous): một vài tác vụ được lập lịch biểu lại để được *thực thi trong tương lai*

Lập trình bất đồng bộ

- Dart sử dụng một vòng lặp sự kiện (event loop) để thực thi các tác vụ được trì hoãn trước đó



Lập trình bất đồng bộ

- Kiểu dữ liệu **Future<T>**
 - Tương tự như kiểu Promise trong JavaScript
 - Đại diện kết quả của một tác vụ bất đồng bộ



```
final myFuture = Future<int>.delayed(  
    Duration(seconds: 1),  
    () => 42,  
);
```

Lập trình bất đồng bộ

- Kiểu dữ liệu **Future<T>**
 - Ba trạng thái của một đối tượng kiểu Future:
 - + Chưa hoàn thành
 - + Hoàn thành với một giá trị
 - + Hoàn thành với một lỗi
 - Lấy kết quả từ một đối tượng kiểu Future đã hoàn thành:
 - + Dùng callback
 - + Dùng async-await

Lập trình bất đồng bộ



```
print('Before the future');

final myFuture = Future<int>.delayed(
    Duration(seconds: 1),
    () => 42,
)
    .then(
        (value) => print('Value: $value'),
    )
    .catchError(
        (error) => print('Error: $error'),
    )
    .whenComplete(
        () => print('Future is complete'),
    );

print('After the future');
```

- Kiểu dữ liệu **Future<T>**
 - Lấy kết quả dùng callback

```
Before the future
After the future
Value: 42
Future is complete.
```

Lập trình bất đồng bộ



```
Future<void> main() async {
    print('Before the future');

    final value = await Future<int>.delayed(
        Duration(seconds: 1),
        () => 42,
    );
    print('Value: $value');

    print('After the future');
}
```

- Kiểu dữ liệu **Future<T>**
 - Lấy kết quả dùng `async-await`

Before the future
Value: 42
After the future

Lập trình bất đồng bộ



```
print('Before the future');

try {
    final value = await Future<int>.delayed(
        Duration(seconds: 1),
        () => 42,
    );
    throw Exception('There was an error');
    print('Value: $value');
} catch (error) {
    print(error);
} finally {
    print('Future is complete');
}

print('After the future');
```

- Kiểu dữ liệu **Future<T>**
 - Xử lý lỗi với async-await

```
Before the future
Exception: There was an error
Future is complete
After the future
```

Lập trình bất đồng bộ

pubspec.yaml

```
name: 'http_example'

environment:
  sdk: '≥2.12.0 <3.0.0'

dependencies:
  http: ^0.13.4
```

Tải các gói phụ thuộc:
dart pub get

```
1 import 'dart:convert'; // for jsonDecode()
2 import 'dart:io'; // for HttpException, SocketException
3 import 'package:http/http.dart' as http;
4
5 Run | Debug
6 Future<void> main() async {
7   try {
8     final url = 'https://jsonplaceholder.typicode.com/todos/1';
9     final parsedUrl = Uri.parse(url);
10    final response = await http.get(parsedUrl);
11    final statusCode = response.statusCode;
12    if (statusCode == 200) {
13      final rawJsonString = response.body;
14      final jsonMap = jsonDecode(rawJsonString);
15      final todo = Todo.fromJson(jsonMap);
16      print(todo);
17    } else {
18      throw HttpException('$statusCode');
19    }
20  } on SocketException catch (error) {
21    print(error);
22  } on HttpException catch (error) {
23    print(error);
24  } on FormatException catch (error) {
25    print(error);
26 }
```

Lập trình bất đồng bộ

- Kiểu dữ liệu **Stream<T>**

- Một stream đại diện *nhiều* giá trị sẽ đến trong tương lai



```
import 'dart:io';

Future<void> main() async {
    final file = File('assets/text.txt');
    final contents = await file.readAsString();
    print(contents);
}
```

Trường hợp tập
tin cần đọc lớn?

Lập trình bất đồng bộ

- Kiểu dữ liệu **Stream<T>**

- Một stream đại diện *nhiều* giá trị sẽ đến trong tương lai



```
import 'dart:io';

void main() {
    final file = File('assets/text_long.txt');
    final stream = file.openRead(); // Stream<List<int>>
    stream.listen(
        (data) {
            print(data.length);
        },
    );
}
```

openRead() đọc tập tin theo từng phân khúc (chunk)

Lập trình bất đồng bộ

- Kiểu dữ liệu **Stream<T>**
 - Một stream đại diện *nhiều* giá trị sẽ đến trong tương lai



```
import 'dart:io';

Future<void> main() async {
    final file = File('assets/text_long.txt');
    final stream = file.openRead();
    await for (var data in stream) {
        print(data.length);
    }
}
```

Lập trình bất đồng bộ

- Kiểu dữ liệu **Stream<T>**
 - Xử lý lỗi dùng callback

```
import 'dart:io';

void main() {
    final file = File('assets/text_long.txt');
    final stream = file.openRead();
    stream.listen(
        (data) {
            print(data.length);
        },
        onError: (error) {
            print(error);
        },
        onDone: () {
            print('All finished');
        },
    );
}
```

Lập trình bất đồng bộ

- Kiểu dữ liệu **Stream<T>**
 - Xử lý lỗi dùng try-catch

```
import 'dart:io';

Future<void> main() async {
    try {
        final file = File('assets/text_long.txt');
        final stream = file.openRead();
        await for (var data in stream) {
            print(data.length);
        }
    } on Exception catch (error) {
        print(error);
    } finally {
        print('All finished');
    }
}
```

Lập trình bất đồng bộ

- Kiểu dữ liệu **Stream<T>**
 - Hủy lắng nghe stream
 - + Nên hủy lắng nghe khi không cần nữa

```
import 'dart:async';
import 'dart:io';

Future<void> main() async {
  final file = File('assets/text_long.txt');
  final stream = file.openRead();
  StreamSubscription<List<int>>? subscription;
  subscription = stream.listen(
    (data) {
      print(data.length);
      subscription?.cancel();
    },
    cancelOnError: true,
    onDone: () {
      print('All finished');
    },
  );
}
```

Lập trình bất đồng bộ

- Kiểu dữ liệu **Stream<T>**
 - Biến đổi stream



```
import 'dart:convert';
import 'dart:io';

Future<void> main() async {
    final file = File('assets/text.txt');
    final stream = file.openRead();
    await for (var data in stream.transform(utf8.decoder)) {
        print(data);
    }
}
```

Lập trình bất đồng bộ

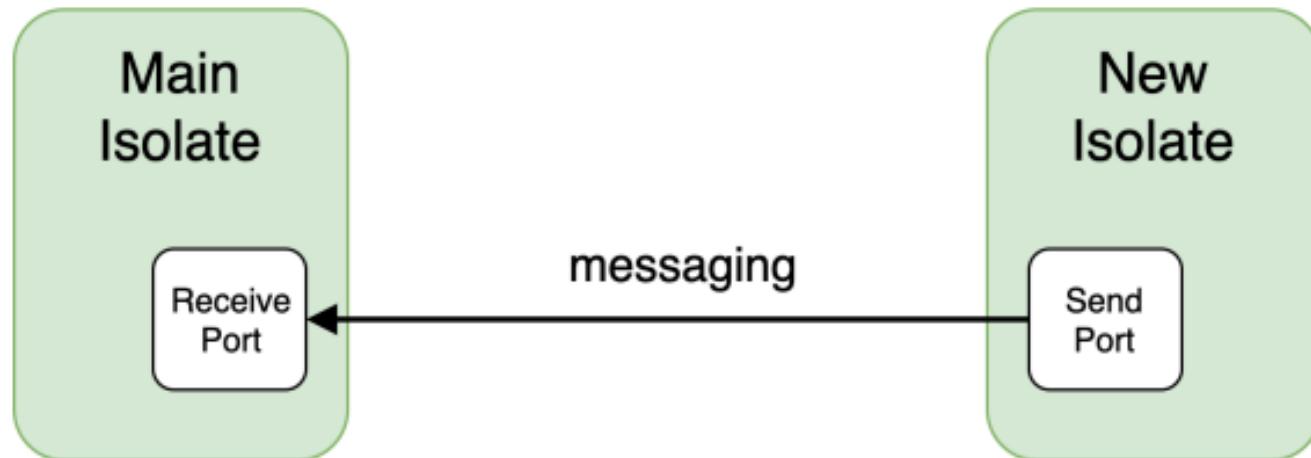
- Kiểu dữ liệu **Stream<T>**
 - Tạo một stream phát sinh sự kiện theo thời gian



```
final myStream = Stream<int>.periodic(  
    Duration(seconds: 1),  
    (i) => i,  
).take(10);  
  
await for (var i in myStream) {  
    print(i);  
}
```

Lập trình bất đồng bộ

- Thực thi các mã lệnh song song trong các isolate
 - Khác với thread thông thường, các isolate không chia sẻ bộ nhớ
 - Các isolate giao tiếp với nhau thông qua việc truyền thông điệp
 - Truyền thông điệp diễn ra giữa cổng gửi và cổng nhận



Lập trình bất đồng bộ

- Thực thi các mã lệnh song song trong các isolate
 - Dùng một cổng gửi để trả về kết quả



```
import 'dart:isolate';

void playHideAndSeekTheLongVersion(SendPort sendPort) {
    var counting = 0;
    for (var i = 1; i <= 1000000000; i++) {
        counting = i;
    }
    sendPort.send('$counting! Ready or not, here I come!');
}
```

Lập trình bất đồng bộ

- Thực thi các mã lệnh song song trong các isolate
 - Tạo isolate và lắng nghe thông điệp



```
Future<void> main() async {
    final receivePort = ReceivePort();
    final isolate = await Isolate.spawn(
        playHideAndSeekTheLongVersion,
        receivePort.sendPort,
    );
    receivePort.listen((message) {
        print(message);
        receivePort.close();
        isolate.kill();
    });
}
```

Câu hỏi?