Lớp trong Dart khai báo và sử dụng class

Cách tạo ra một lớp class, các loại phương thức khởi tạo, hàm setter getter. Tìm hiểu tính kể thừa, interface, mixin trong ngôn ngữ lập trình Dart

Class trong Dart

Class để tạo ra các đối tượng, với Dart mọi thứ kể cả số đều là đối tượng, các đối tượng đề kế thừa từ class Object

Trong một class nó có thể có các thành phần:

* Các phương thức khởi tạo - Hàm được gọi khi tạo ra một đối tượng mới từ class
* Các biến lưu dữ liệu của đối tượng - gọi là các trường - các thuộc tính
* Các hàm - gọi là các thành viên hàm - các phương thức
* Các hàm đặc biệt gọi khi thực hiện gán thuộc tính / truy cập thuộc tính - hầm setter/getter

Từ lớp đã có khởi tạo đối tượng bằng cách gọi hàm khởi tạo của nó với toán tử new, sau khi có đối tượng thì truy cập vào các thành viên (phương thức, thuộc tính) bằng ký hiệu chấm . như object.phuongthuc();

Khai báo một lớp thì dùng từ khóa class, ví dụ sau khai báo một lớp:

class Product {

//Khai báo các thuộc tính

String manufacture = '';

String name = '';

var price;

int quantity;

//Khai báo hàm khởi tạo

Product(var price, {int quantity:0}) {

this.price = price;

this.quantity = quantity;

}

//Khai báo các phương thức

calulateTotal() {

return this.price \* this.quantity;

}

showTotal() {

var tong = this.calulateTotal();

print("Tổng số tiền là: $tong");

}

}

Sử dụng

var product = new Product(600, quantity: 1);

product.showTotal();

product.quantity = 2;

product.showTotal();

//KẾT QUẢ CHẠY

Tổng số tiền là: 600

Tổng số tiền là: 1200

Như vậy ta thấy **khai báo thuộc tính, phương thức** trong lớp tương tự như khai báo biến và hàm thông thường chỉ có điều nó nằm trong class

Để truy cập vào một phương thức, thuộc tính dùng ký kiệu chấm . ví dụ product.quantity = 2, product.showTotal()

Trong phương thức của lớp, để tham khảo đến đối tượng của lớp dùng từ khóa this, ví dụ trong hàm calulateTotal() có đoạn return this.price \* this.quantity

Khi đã có lớp, việc tạo ra đối tượng lớp thì dùng toán tử new, var product = new Product(600, quantity: 1) hoặc không cần toán tử new vẫn được chấp nhận product = Product(600, quantity: 1)

Khi khởi tạo như vậy, nó sẽ gọi đến hàm có cùng tên với lớp, gọi là hàm khởi tạo - để thiết lập các thông tin cho lớp, ở ví dụ trên có một hàm khởi tạo Product(var price, {int quantity:0}), tuy nhiên bạn có thể tạo ra nhiều hàm tạo có tên gọi theo nguyên tắc như sau:

Hàm khởi tạo có tên

Giả sử sẽ tạo ra hàm tạo tên iphone để khởi tạo một loại sản phẩm cụ thể, thì khai báo trong lớp như sau:

class Product {

// ...

Product.iphone(var price, {int quantity:0}) {

this.price = price;

this.quantity = quantity;

this.manufacture = 'Apple';

}

// ...

}

Nếu vậy bạn có thể khởi tạo bằng hàm tạo này

var product = Product.iphone(700, quantity: 2);

Phương thức tĩnh

Các phương thức (hàm) trong lớp chỉ truy cập được trên một đối tượng cụ thể triển khai từ lớp (biến product), nhưng bạn có thể chỉ định phương thức là tĩnh bằng từ khóa static, thì hàm không cần đối tượng triển khai từ lớp để hoạt động mà có thể gọi hàm đó thông qua tên lớp. Ví dụ khai báo phương thức có tên showListStore() là phương thức tĩnh.

class Product {

// ...

static showListStore() {

print('Store 1 ...');

print('Store 2 ...');

}

// ...

}

Như vậy bất kỳ đâu cũng có thể gọi đến phương thức này mà không cần khởi tạo đối tượng. Chỉ cần tên lớp để gọi (cần nhớ là hàm này thuộc về lớp chứ không thuộc về đối tượng triển khai từ lớp)

Product.showListStore();

Hàm Setter/Getter

Ta có thể xây dựng hàm đặc biệt gọi mà có thể truy cập nó giống phương thức thì nó thi hành (Setter gọi khi thực hiện gán, Getter gọi khi truy cập). Sử dụng từ khóa get trước một hàm không có tham số thì hàm đó trở thành Getter, sử dụng từ khóa get trước hàm 1 tham số thì đó là hàm Getter

Ví dụ có thêm vào lớp Product hàm Getter và Setter đều có tên là nameProduct

class Product {

// ...

//getter

get nameProduct {

return this.name;

}

//Hàm setter

set nameProduct(name) {

this.name = name;

switch (this.name) {

case 'Iphone 6':

this.manufacture = "Apple";

break;

case 'Galaxy S6':

this.manufacture = 'Samsung';

break;

default: this.manufacture = '';

}

}

// ...

}

Sau khi có Getter/Setter thì truy cập giống như thuộc tính

product.nameProduct = "Galaxy S6"; //Gọi đến hàm Setter - nameProduct

var info = product.nameProduct; //Gọi đến Getter

Tính kế thừa trong lớp

Từ một lớp đã có, bạn có thể tạo ra một định nghĩa lớp mới, lớp mới đó gọi là lớp kế thừa - lớp con có luôn các thuộc tính, phương thức từ lớp mà nó kế thừa (gọi là lớp cha).

Để xây dựng một lớp mới kế thừa lớp đã có dùng tới từ khóa extends, ví dụ từ lớp Product xây dựng thêm lớp Table có thêm thuộc tính mô tả chiều dài, chiều rộng của sản phẩm.

class Table extends Product {

double length = 0;

double width = 0;

Table(var giatien) : super(giatien, quantity:1) {

//Mã khởi tạo tại lớp con, sau khi hàm tạo lớp cha chạy xong

this.name = "Bàn Ăn";

}

}

Khởi tạo tại lớp con và sự truy cập đến lớp cha

Lớp con nói chung sẽ có những thuộc tính và phương thức kế thừa từ lớp cha, nên từ lớp con bằng từ khóa this có thể truy cập đến những thành phần này. Tuy nhiên có những phương thức mà lớp con sẽ định nghĩa lại mà vẫn giữ tên cũ (quá tải) lúc này this sẽ sử dụng phương thức định nghĩa lại, tuy nhiên phiên bản ở lớp cha vẫn còn đó, lúc này nếu muốn truy cập đến phiên bản định nghĩa bởi lớp cha sẽ dùng ký hiệu super thay cho this (xem thêm phần quá tải hàm ở dưới).

Hàm khởi tạo ở lớp con, nói chung bắt buộc cũng phải gọi một hàm khởi tạo nào đó của lớp cha. Để làm được điều đó sau hàm khởi tạo của lớp con chỉ rõ hàm tạo nào cả lớp cha sẽ gọi sau dấu :

Ở ví dụ trên chính là đoạn supper(giatien, quantity:1) tương đương với hàm khởi tạo Product(giatien, quantity:1)

Trở lại lớp Table trên, khi tạo đối tượng từ lớp

var table = new Table(500);

Hàm tạo thi hành nó đã gọi đến hàm tạo của lớp cha Product rồi đến các code của chính hàm tạo Table

Nạp chồng phương thức / toán tử

Bạn có thể tạo ra phiên bản mới của một phương thức đã có trên lớp cha, và từ đây đối tượng sẽ sử dụng phương thức mới được định nghĩa, để làm điều đó ở lớp con tạo lại phương thức với chỉ thị @override - nạp chồng phương thức

Ví dụ, ta sẽ nạp chồng phương thức showTotal()

class Table extends Product {

// ...

@override

showTotal() {

print('Sản phẩm:' + this.name);

//gọi đến phương thức ở lớp cha

super.showTotal();

}

// ...

}

Ở phiên bản ở lớp con, do có nhu cầu sử dụng lại phương thức của lớp cha nên nó có gọi đến phương thức cũ bằng super.showTotal()

Như vậy các đối tượng triển khai từ lớp Table đã có một phiên bản riêng của phương thức showTable, mặc định nó sẽ gọi đến phương thức mới này (Kể các các phương thức lớp cha cũng sẽ tự động gọi đến phương thức mới định nghĩa này).

var table = new Table(600);

table.showTotal();

//Kết quả chạy

var table = new Table(600);

table.showTotal();

Đối với các toán tử như +, -, \*, [] ... cũng có thể nạp chồng như trên.

Ví dụ định nghĩa toán tử +

Product operator + (Product p) => new Product(this.quantity + p.quantity);

Lớp trừu tượng

Lớp trừu tượng là lớp không dùng trực tiếp để tạo ra đối tượng được, nó chỉ được kế thừa từ lớp khác. Phương thức nào trong lớp trừu tượng chỉ khai báo tên hàm, thì phương thức đó gọi là phương thức trừu tượng, lớp kế thừa bắt buộc phải định nghĩa nội dung hàm này. Sau đây là tạo ra lớp tượng A với từ khóa abstract

abstract class A {

//Khai báo các thuộc tính

var name = 'My Abstract Class';

//Khai báo các phương thức nếu cần

//Khai báo phương thức trừu tượng (chỉ có tên)

void displayInfomation();

}

Lớp này không thể dùng để tạo ra đối tượng, nhưng nó được kế thừa bởi lớp khác. Lớp con kế thừa bắt buộc phải định nghĩa nội dung cho phương thức trừu tượng bằng cách nạp chồng (@override). Ví dụ khai báo lớp B kế thừa lớp trừu tượng A

class B extends A {

@override

void displayInfomation() {

print(this.name);

}

}

//Áp dụng

var i = new B();

i.displayInfomation();

//Kết quả chạy

My Abstract Class

Tìm hiểu về interface

Interface - giao diện - là khái niệm quen thuộc trong các ngôn ngữ lập trình hướng tối tượng, với Dart mặc định mọi lớp đều là interface lớp lớp đó được triển khai bởi lớp khác bằng từ khóa implements

Khi một lớp được coi là giao diện thì lớp triển khai nó phải định nghĩa lại mọi phương thức, thuộc tính có trong giao diện

Ví dụ xây dựng lớp C triển khai từ lớp B bằng implements vậy B sẽ là interface và trong C bắt buộc phải định nghĩa lại mọi thứ trong B

class C implements B {

@override

String name;

@override

void displayInfomation() {

// ...

}

}

Mục đích sử dụng giao diện là để đảm báo các lớp có cùng giao diện sẽ có các API giống nhau.

Sử dụng Mixin

Với Dart thì Mixin là một lớp, nó không được sử dụng trực tiếp để tạo ra đối tượng, một Mixin chứa các phương thức, thuộc tính dùng để gộp vào một lớp khác.

Ví dụ ta có một Mixin tên là M thì khi khai báo lớp C ở trên muốn gộp những gì có ở M vào dùng từ khóa with (mang ý nghĩa gộp code chứ không mang ý nghĩa kế thừa) với cú pháp như sau

mixin M {

var var1 = null;

showSomething()

{

print('Print message ...');

}

}

class C extends B with M {

@override

String name;

@override

void displayInfomation() {

}

}

Một số lưu ý bổ sung về Class

Lớp Object

Đây là lớp cơ sở của Dart, mặc định mọi lớp, mọi hàm ... kể cả lớp, hàm do bạn định nghĩa đều mở rộng từ lớp này. Như vậy mọi lớp đều có một số phương thức, thuộc tính chung là:

* hashCode thuộc tính chứa mã hash của đối tượng
* toString() trả về chuỗi mô tả đối tượng
* == toán tử so sánh theo hashCode của hai đối tượng

Toán tử cascade ..

Khi bạn có một chuỗi tác vụ trên đối tượng (gọi phương thức, thiết lập thuộc tính) thay vì phải viết đầy đủ đối tượng thì bạn chỉ cần viết nó một lần, các tương tác tiếp theo thay thế bằng ..

var table = new Table(1);

table

..calulateTotal() //Thay cho table.calulateTotal();

..length=100 //Thay cho table.length=100;

..name='Abc'

..quantity=100

..showTotal();

Nếu bạn lớp của bạn sinh ra các đối tượng không thay đổi, hãy thêm từ khóa const vào trước hàm tạo.