一、开篇

dart 语言具有如下特性

- 一切变量皆是对象，每个对象都是类的实例。int、double、null、函数等都是对象，所有对象都继承自 Object 类
- dart 是强类型语言，但由于具备类型推导功能所以类型声明是可选的
- dart 支持顶级函数、静态函数、实例函数，也允许在函数中嵌套函数，即局部函数。类似的，dart 也支持顶级变量、静态变量和实例变量
- dart 没有关于 public、protected、private 的关键字。通过为变量标识符添加下划线前缀，表明该对象是私有的
- 标识符以字母或下划线开头，后跟任意字母和数字组合

看个小例子

```
/**
 * 多行注释
 */
void printString(String msg) {
  print('msg value: $msg');
}

//单行注释
void main() {
  var msg = 'Hello, World!';
  printString(msg); //msg value: Hello, World!
  printString(null); //msg value: null
}
```

如上代码包含了 dart 语言（也是基本所有的编程语言都具备的）的一些基本元素

- 多行注释和单行注释
- 以分号结尾且是必需的
- 允许定义顶层函数
- 最基础的数据类型之一：String，以单引号或双引号包裹起来。其它的内置数据类型还有 int 、double、list、map 等
- 类型推导。通过关键字 var 来声明变量而无需指明变量类型
- 一种方便的插入变量值的方式，字符串字面值：$msg
- 应用程序的入口：main 函数

二、变量

2.1、变量声明

与 Java 语言相比，dart 语言包含的类似的基本数据类型只有 `int` 和 `double` 两种，且这两种类型的变量均是对象，其默认值均为 null

本教程遵循官方风格指南建议，大部分例子都是使用 var 来声明变量而不指明对象类型

```
void main() {
  int intValue;
  print(intValue); //null

  intValue = 10;
  print(intValue); //10

  var varIntValue = 20;
  print(varIntValue); //20
}
```

dart 是强类型语言，无法将一个已声明具体变量类型的变量赋值为另一个无继承关系的变量

例如，以下代码会导致报错，因为无法将一个 double 值赋值到一个 int 类型变量上

```
  int intValue = 20;
  intValue = 20.0; //error
```

但由于 `int` 和 `double` 类都是 **num 类的子类**，所以以下操作是合法的

```
  num numValue = 10;
  print(numValue.runtimeType); //int
  numValue = 10.22;
  print(numValue.runtimeType); //double
```

2.2、dynamic

`dynamic` 类似于 Java 中的 `Object`， `dynamic` 对象可以用来指向任意类型变量，非 null 的 dynamic 变量会有具体的运行时类型

```
  dynamic value = "leavesC";
  print(value.runtimeType); //String
  value = 12121;
  print(value.runtimeType); //int
```

2.3、final 和 const

如果你希望一个变量在赋值后其引用不能再改变，可以通过 **final** 或 **const** 这两个关键字来实现

const 变量代表的是编译时常量，例如字面量就是一种编译时常量，在编译期，程序运行前就有确定值了，因此实例变量不能使用 const 修饰。如果 const 变量是类级别的，需要标记为 `static const`

而 final 修饰的变量是运行时常量，赋值后不能再改变引用，可以在运行时再赋予变量值，因此实例变量能使用 final 修饰

```
void main() {
  const URL = "https://github.com/leavesC/AndroidAllGuide";
  var booleValue = true;
  final name = getName(booleValue);
  print(name);
}

String getName(boolValue) {
  if (boolValue) {
    return "leavesC";
  } else {
    return "leavesC =_=";
  }
}
```

三、内建类型

3.1、num

dart 的数字类型有 int 和 double 两种，这两种都是 num 类的子类。int 类型根据平台的不同，整数值不大于64位。在 Dart VM 上，值可以从 **-2^{63} 到 $2^{63}-1$**，编译成 JavaScript 的 Dart 使用JavaScript 代码，允许值从 **-2^{53} 到 $2^{53}-1$**。double 类型即**64位双精度浮点数**，由 IEEE 754 标准指定

```
void main() {
  var intValue = 100;
  print(intValue.runtimeType); //int

  var doubleValue = 100.0;
  print(doubleValue.runtimeType); //double

  num numValue = 100;
  print(numValue.runtimeType); //int
  numValue = 100.0;
  print(numValue.runtimeType); //double
}
```

一些常见的数字类型转换方法

```
print(num.parse("2000"));
print(int.parse("200"));
print(double.parse("121"));
```

3.2、string

除了可以通过单引号或者双引号来声明一个字符串外，也可以通过相邻的字符串字面量来声明一个组合字符串（相当于使用 + 把字符串相加为一个整体），此时可以混用**单引号**和**双引号**，但建议只用单引号

```
var stringValue = "leavesC";
var stringValue2 = 'leavesC =_=';
var stringValue3 = "分段 "
    "换行了也可以"
    '又换了一行';
print(stringValue3); //分段 换行了也可以又换了一行
```

此外，也可以使用带有单引号或双引号的三重引号，此时包含的转义字符是有效的

```
var stringValue='''\n 换行符
  \t 制表符
  ''';
```

也可以用 **r前缀** 创建一个原始字符串，包含的转义字符将会失效

```
var stringValue=r'''
\n 换行符
\t 制表符
''';
```

3.3、bool

dart 语言用 bool 关键字来表示真假两面，bool 类型只有两个实例：true 和 false，它们都是编译时常量，且只有 bool 类型的值是 true 才被认为是 true

3.4、list

list 也是最常见的数据类型之一，dart 通过方括号来声明 list 变量

由于 dart 具有类型推导功能，因此 listValue 自动被赋予为 **List< int >** 类型，因此在声明 listValue 后就无法为其添加其他类型变量的值了

```
var listValue = [1, 2, 3];
// listValue.add("4"); error
print(listValue.runtimeType); //List<int>
```

如果想要为 List 添加不同数据类型的变量，则需要直接指明数据类型为 Object

```
var listValue = <Object>[1, 2, 3];
listValue.add("4");
print(listValue.runtimeType); //List<Object>
```

大多数时候为了限制 List 的可存储数据类型，在使用时就直接指明数据类型

```
var intList = <int>[1, 2, 3, 4];
```

如果在声明 List 时调用了其指定集合大小的构造函数，则集合大小就是固定的了，列表的长度不能在运行时更改

```
var list = List<int>(2);
list.add(2); //error, 会导致抛出 Unsupported operation: Cannot add to a fixed-length lis
```

要创建一个编译时常量列表，则在列表字面量之前添加 const 关键字

```
var constantList = const [1, 2, 3];
//error, 可正常编译，但会导致运行时抛出异常
//constantList[0] = 2;
//constantList.add(2);
```

3.5、set

Set 是一种不包含重复数据的数据集合，使用方式上和 List 基本类似

```
void main() {
  var list = [1, 2, 2, 3, 4, 5, 5];
  var set = Set.from(list);
  print(set); //{1, 2, 3, 4, 5}
}
```

3.6、map

map 是一个关联键和值的数据类型，键和值可以是任何类型的对象

```
void main() {
  var mapValue = {"name": "leavesC", "age": 24};
  mapValue["url"] = "https://github.com/leavesC";
  print(mapValue); //{name: leavesC, age: 24, url: https://github.com/leavesC}
  print(mapValue.length); //3
  print(mapValue.runtimeType); //_InternalLinkedHashMap<String, Object>
}
```

也可以限定 map 可以存储的数据类型

```
var mapValue = <String, String>{"name": "leavesC"};
```

与 list 类似，要创建一个编译时常量的 map 需要在 map 的字面量前加上 const 关键字

```
  var mapValue = const {"name": "leavesC", "age": 24};
  //error，可正常编译，但会导致运行时抛出异常
  mapValue["name"] = "hi";
```

四、函数

dart 是一种真正的面向对象语言，所以即使函数也是对象，即变量可以指向函数，也可以将函数作为参数传递给其他函数

4.1、一般概念

一般，为了方便调用者理解，函数需要指明其接受的参数类型，但也允许不指明参数类型，此时函数依然可以正常调用

```
void main() {
  printMsg("leavesC");

  printMsg2(100);
  printMsg2("leavesC");
}

void printMsg(String msg) {
  print(msg);
}

void printMsg2(msg) {
  print(msg);
}
```

如果函数只包含一个表达式，则可以使用简写语法

```
void printMsg3(msg) => print(msg);
```

所有函数均有返回值，如果没有指明函数的返回值类型，则函数默认返回 null

```
void main() {
  print(printValue(121) == null); //true
}

printValue(value) {
  print("value is: $value");
}
```

4.2、函数也是对象

在 dart 中，可以用变量来引用函数对象、向函数传递函数参数、创建函数对象

```
void main() {
  var printUserFun = printName;
  printUserFun("leavesC"); //name: leavesC

  var list = ["leavesC", "叶"];
  list.forEach(printName); //name: leavesC   name: 叶

  var sayHelloFun = (String name) => print("$name , hello");
  sayHelloFun("leavesC"); //leavesC , hello
}

void printName(String name) {
  print("name: $name");
}
```

4.3、位置参数

位置参数即该参数可传也可不传，当不传时该参数值默认为 null，位置参数用中括号包裹起来

```
void main() {
  printUser("leavesC"); //name: leavesC, age: null
  printUser("leavesC", 25); //name: leavesC, age: 25
}

void printUser(String name, [int age]) {
  print("name: $name, age: $age");
}
```

4.4、命名参数

命名参数，即在调用该函数时需同时标明该参数的参数名，命名参数用花括号包裹起来，以 `{type paramName}` 或者 `{paramName: type}` 两种方式声明参数，调用命名参数时只能以 `funcName(paramName: paramValue)` 的形式来调用。且命名参数可传也可不传值，当不传指时该参数值为 null

```
void main() {
  printUser("leavesC"); //name: leavesC, age: null
  printUser("leavesC", age: 25); //name: leavesC, age: 25
  printUser2("leavesC", age: 25); //name: leavesC, age: 25
}

void printUser(String name, {int age}) {
  print("name: $name, age: $age");
}

void printUser2(String name, {age: int}) {
  print("name: $name, age: $age");
}
```

4.5、默认参数值

和 kotlin 类似，dart 语言也支持为位置函数和命名参数设置默认值，默认值必须是编译时常量，如果没有提供默认值，则默认值为 null

```
void main() {
  printUser("leavesC"); //name: leavesC, age: 30
  printUser("leavesC", 25); //name: leavesC, age: 25

  printUser2("leavesC"); //name: leavesC, age: 30
  printUser2("leavesC", age: 25); //name: leavesC, age: 25
}

void printUser(String name, [int age = 30]) {
  print("name: $name, age: $age");
}

void printUser2(String name, {int age = 30}) {
  print("name: $name, age: $age");
}
```

4.6、函数变量

前面说了，dart 是一种真正的面向对象语言，即使函数也是对象，即变量可以赋予函数类型

```
void main() {
  var printFunction = printUser;
  printFunction("leavesC");
  print(printFunction); //Closure: (String, [int]) => void from Function 'printUser': st
}

void printUser(String name, [int age = 30]) {
  print("name: $name, age: $age");
}
```

也可以将函数作为参数传递给另外一个函数

```
void main() {
  var list = {1, 2, 3};
//  value is: 1
//  value is: 2
//  value is: 3
  list.forEach(printValue);
}

void printValue(value) {
  print("value is: $value");
}
```

4.7、匿名函数

匿名函数即不具备函数名称的函数，在函数只使用一次会就不再调用时使用匿名函数会比较方便

```
void main() {
  var list = {1, 2, 3};
//  value is: 1
//  value is: 2
//  value is: 3
  list.forEach((element) {
    print("value is: $element");
  });
  list.forEach((element) => print("value is: $element"));
}
```

4.8、局部函数

局部函数即嵌套于其他函数内部的函数

```
void main() {
  var list = {1, 2, 3};
//  value is: 2
//  value is: 3
//  value is: 4
  list.forEach((element) {
    int add(int value1, int value2) {
      return value1 + value2;
    }
    print("value is: ${add(element, 1)}");
  });
}
```

五、运算符

dart 提供了一些比较简便的运算符来简化操作，大部分和 Java 相同，以下介绍下几个不同的运算符

```dart
void main() {
  //is 用于判断变量是否是指定的数据类型
  //is! 含义是 is 取反
  var strValue = "leavesC";
  print(strValue is String); //true
  print(strValue is int); //false
  print(strValue is! String); //false

  // ~/ 用于除法运算时取整，/ 则不取整
  print(10 / 3); //3.3333333333333335
  print(10 ~/ 3); //3

  //as 用于强制类型转换
  num numValue = 10;
  int intValue = numValue as int;

  //如果 ??= 左边的变量值为 null ,则将其赋值为右边的值
  var name = null;
  var age = 25;
  name ??= "leavesC";
  age ??= 30;
  print("name：$name"); //name：leavesC
  print("age: $age"); //age: 25

  //如果 ?. 左边的变量值不为 null，则右边的操作生效
  //用于避免发生空指针异常
  var area = null;
  print(area?.runtimeType); //null

  //级联运算符 .. 用于连续操作某个对象，而无需每次操作时都调用该对象
  var list = [1, 2, 3, 4, 5];
  list
    ..insert(0, 6)
    ..removeAt(4)
    ..add(7);
  print(list); //[6, 1, 2, 3, 5, 7]

  //扩展运算符 ... 和 空值感知扩展运算符 ...?
  //提供了一种将多个元素插入集合的简洁方法
  var list1 = [1, 2, 3];
  var list2 = [0, ...list1];
  print(list2); //[0, 1, 2, 3]
  //如果 list3 可能为 null，此时则需要使用空值感知扩展运算符，否则会抛出异常
  //空值感知扩展运算符只有当 list3 不为 null 时才会执行插值操作
  var list3 = null;
  var list4 = [0, ...?list3];
  print(list4); //[0]
}
```

六、流程控制

dart 的流程控制的语义和逻辑大多和 Java 相同

```
void main() {
  //if
  int value = 20;
  if (value < 10) {
    print("value < 10");
  } else if (value < 30) {
    print("value < 30");
  } else {
    print("value unknown");
  }

  //while
  int score = 10;
  while (score < 100) {
    score++;
  }

  //switch
  String strValue = "leavesC";
  switch (strValue) {
    case "ye":
      break;
    case "leavesC":
      print(strValue);
      break;
    default:
      print("default");
      break;
  }

  //for
  var list = [];
  for (int index = 1; index < 10; index++) {
    list.add(index.toString());
  }
  for (var item in list) {
    print("循环遍历：$item");
  }
}
```

此外也有一些比较奇特的操作，可以通过**条件 if** 和**循环 for** 来构建集合

```
  var hasName = true;
  var info = {if (hasName) "name": "leavesC", "age": 24};
  print(info); //{name: leavesC, age: 24}


  var list = {1, 2, 3};
  var info = {for (var item in list) "$item", 4};
  print(info); //{1, 2, 3, 4}
```

七、枚举

枚举用于定义命名常量值，使用 enum 关键字来声明枚举类型

```
enum State { RESUME, STOP, PAUSE }

void main() {
  var state = State.PAUSE;
  print(state);
  State.values.forEach((state) {
    print(state);
  });
}
```

八、异常处理

dart 语言可以抛出和捕获异，捕获异常可以避免程序运行崩溃，与 Java 不同，dart 的所有异常均是未检查异常，方法不必声明本身可能抛出的异常，也不要求调用方捕获任何异常。dart 提供了 Exception 和 Error 类型，以及许多预定义的子类型，开发者也可以自己定义异常。而且，dart 可以抛出任何对象，不仅仅是 Exception 和 Error 两类

例如，如下代码就表示 `throwException` 方法内部捕获了 RangeError ，但对其他异常不进行处理

```
void throwException() {
  try {
    List<String> stringList = new List();
    stringList.add("value");
    print('${stringList[1]}');
  } on RangeError {
    print("抛出了异常...");
  }
}

void main() {
  throwException();
}
```

如果在抛出异常时需要异常对象，则需要用到 `catch`

```
void throwException() {
  try {
    List<String> stringList = new List();
    stringList.add("value");
    print('${stringList[1]}');
  } on RangeError catch (e) {
    print("${e.message}"); //Invalid value
    print("${e.runtimeType}"); //RangeError
  } catch (e) {
    print("如果异常没有被上方捕获，则会统一被此处捕获");
  }
}
```

也可以使用两个参数的写法，第二个参数代表堆栈跟踪(StackTrace对象)

```
void throwException() {
  try {
    List<String> stringList = new List();
    stringList.add("value");
    print('${stringList[1]}');
  } on RangeError catch (e, s) {
    print("${s.toString()}");
  }
}
```

也可以在捕获异常后将异常再次抛出

```
void throwException() {
  try {
    List<String> stringList = new List();
    stringList.add("value");
    print('${stringList[1]}');
  } on RangeError catch (e, s) {
    print("${s.toString()}");
    rethrow;
  }
}
```

类似 Java， `finally` 用于确保在抛出异常时某些代码也可以被执行

```
void throwException() {
  try {
    List<String> stringList = new List();
    stringList.add("value");
    print('${stringList[1]}');
  } on RangeError catch (e, s) {
    print("throwException ${e.message}");
    rethrow;
  } finally {
    print("finally");
  }
}

void main() {
  try {
    throwException();
  } catch (e) {
    print("main ${e.message}");
  }
//  throwException Invalid value
//  finally
//  main Invalid value
}
```

此外，dart 也允许 throw 任何对象，包括 `null`

```
void throwException() {
  try {
    List<String> stringList = new List();
    stringList.add("value");
    print('${stringList[1]}');
  } on RangeError catch (e, s) {
    throw null;
    //or throw "发生了异常";
  }
}

void main() {
  try {
    throwException();
  } catch (e) {
    print("main ${e}"); //main Throw of null.
    print("${e.runtimeType}"); //NullThrownError
  }
}
```

此外，由于抛出异常是一个表达式，所以以下写法是合乎语法的

```
int throwException() => throw "";
```

九、类

9.1、类声明

dart 是一门完全面向对象的语言，其关于类的内容和 Java 较为类似

```
class Person {

  String name;

  int age;

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

}
```

dart 会自动为 `name` 和 `age` 提供隐式的 `getter` 和 `setter` 方法，且未经初始化的实例变量均为 null

在 dart 2 中 **new 关键字**成为了可选关键字，因此可以选择省略 **new** 声明，这一点和 kotlin 相同

```
void main() {
  var person = Person("leavesC", 25);
  print('${person.name} ${person.age}'); //leavesC 25
  print('${person.runtimeType}'); //Person
}
```

9.2、构造函数

dart 为用于赋予初始值的构造函数提供了简便写法，以下两种写法的语义是一致的

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

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

此外，dart 也提供了命名构造函数，用于方便调用者生成不同用途或含义的变量

```
  Person.getDefault() {
    this.name = "leavesC";
    this.age = 25;
  }
```

也可以在构造函数主体运行之前初始化实例变量

```
  Person.getDefault()
      : name = "leavesC",
        age = 25 {}
```

因此当想要获取一个默认的 `Person` 实例时就如同在调用 Person 的一个静态方法

```
void main() {
  var defaultPerson = Person.getDefault();
  print('${defaultPerson.name} ${defaultPerson.age}'); //leavesC 25
  print('${defaultPerson.runtimeType}'); //Person
}
```

9.3、继承

默认情况下，子类中的构造函数会隐式调用父类的未命名的无参数构造函数，父类的构造函数在子类构造函数体的开始处被调用。如果父类没有未命名的无参数构造函数，则必须手动调用父类中的构造函数

此外，构造函数不能被子类继承，父类中的命名构造函数不会被子类继承，所以如果子类也想要拥有一个和父类一样名字的构造函数，则必须子类自己实现这个构造函数

```
class Man extends Person {

  Man(String name, int age) : super(name, age);

  Man.getDefault() : super.getDefault();

}
```

9.4、抽象类

dart 语言的抽象类和 Java 基本一致

```
abstract class Printer {
  void print(String msg);
}

class HpPrinter extends Printer {
  @override
  void print(String msg) {
    // TODO: implement print
  }
}
```

9.5、接口

dart 没有用来专门声明接口的语法，类声明本身就是 dart 中的接口，实现类使用 `implements` 关键字来实现接口，实现类必须提供目标接口的所有功能的具体实现，即类必须重新定义它希望实现的接口中的每一个函数

```
void main() {
  var human = Human();
  human.eat();
  human.speak();
}

class Human implements Eat, Speak {
  void funA() {}

  @override
  void eat() {
    // TODO: implement funB
  }

  @override
  void speak() {
    // TODO: implement funC
  }
}

class Eat {
  void eat() {}
}

class Speak {
  void speak() {}
}
```

9.6、mixins

mixins 是一个重复使用类中代码的方式

```
void main() {
  var c = C();
  c.funA();
  c.funB();
}

class A {
  void funA() {}
}

class B {
  void funB() {}
}

//使用 with 关键字表示类C是由类A和类B混合构成的
class C = A with B;
```