1. Dart语法

Dart官网：<http://dart.goodev.org/guides/platforms>

一 、重要概念

1. 一切变量都是对象，包括数字，函数和null
2. 对象是累的实例，所有类都继承Object
3. 强类型支持类型推断
4. 不想指定具体类型，使用dynamic
5. 支持泛型，比如List<Int>和list<dynamic>
6. 支持顶层函数，类方法，实例方法，嵌套函数
7. 支持顶层变量，类变量，实例变量，局部变量
8. 以下划线(\_)开头变量是私有的，只能在库内部使用
9. 标识符只能以字母或下划线打头，后面跟任意多个字母，数字，下划线字符
10. 表达式在运行时可以求值，而语句不行
11. 代码问题分为warning和error两种，waring不会阻止程序运行，但error会

二 、关键字



三 、变量

1. 变量类型可以自动推断或者显示指定
2. 未初始化的变量都是null，包括数字
3. 如果某个变量的值在定义之后不会改变，就申明final或者const
4. final或者const变量只能赋值一次，但是const变量的值需要在编译的时候就能确定
5. final和const必须在定义的时候就完成初始化
6. const变量的值是编译时常量，包括数字，字符或者字符串常量，其他常量或算术运算结果
7. cosnt除了申明变量，也可用来申明值，也就是常量
8. 初始化cosnt变量时可以省去值前面的const，因为const变量的值肯定是一个常值
9. 常值用来防止内容变化，常值用来避免变量寻址，final变量用来提升代码安全性

注意：

1. 实例变量可以为 final 但是不能是 const 。
2. 被final，const 修饰的值是常值，不可被修改
3. 在生产模式 assert() 语句被忽略了。在检查模式 assert(*condition*) 会执行，如果条件不为 true 则会抛出一个异常

四 、内置类型

1. 数字（num）

1） num是int和double的基类

2） int是64位整数，double是64位整数

2. 字符串（String）

1） 字符串是utf-16码元（code unit）序列，每个unicode字符映射为1个或2个码元

2） 字符串字面量可以使用单引号或双引号

3） 使用${expression}来插入变量值，如果expression是个合法的标识符，则可以省去{}

4） 使用毗连字符串常量或+操作符来拼接字符

5） 使用三重引号来定义多行字符

6） 使用r前缀来声明原始字符串，其内不做字符转义

7） 只要内部插入的表达式为编译时常量，则字符串字面量为编译时常量

3. 布尔（bool）

1） 只有2个值，true和false，都是编译时常量

2） 在需要布尔类型的地方必须使用布尔值

4. 列表（list）

1） 列表就是其他语言里面的数组

2） 列表索引序号从0开始，直到list.length-1

3） 在列表字面量前使用const来声明常值

4） 使用[]操作符来操作列表元素

5. 哈希表（Map）

1） 哈希表用来关联键和值，键和值可以是任意类型的对象，但键必须唯一

2） 使用[]操作符来操作哈希表的值

3） 在哈希表字面量前使用const来声明常值

6. 符文（Rune）

1) Rune是由utf-32码点（code point）组成的字符串，一个Rune字符对应一个Unicode字符

2） 码点一般使用4个十六进制字符\uXXXX来表示，如果超过4个，则需要使用{}将十六进制字符括起来

3） 字符串和Rune之间可以互相转换

注意：

1. String->int，int.parse(‘xxx’)

String->double，double.parse(‘xxx’)

Int->string，xxx.toString

Double->string，xxx.toStringAsFixed(2),保留2位小数

1. << 是前面数字二进制左移 \*2 ，>>是前面数字右移 /2
2. 编译时常量，在编译时可以互相赋值
3. Xxx.runtimeType 返回变量的类型

7. symbols

其字面量定义是编译时常量

使用 Symbol 字面量来获取标识符的 symbol 对象，也就是在标识符 前面添加一个 # 符号

五、 函数

1. 函数也是一种对象，类型为Function

2. 函数参数和返回值类型可以省略，以支持动态类型

3. 如果函数体只包含一个表达式，可使用箭头语法来定义

4. 可选参数：

可选命名参数使用{}来指定，并且可使用注解@required标注为必须

可选位置参数使用[]来指定

可选参数默认值使用=来指定，如未指定则默认值为null

5. 每个程序都应该有个位于顶层的main函数，它是程序的入口

6. 函数可作为函数参数值，也可赋值给变量(一等方法对象)

7. 可定义匿名函数，一般作为函数参数值或赋值给变量（匿名方法）

8. 变量作用域静态确定，也就是同代码布局，每对大括号定义一个作用域，嵌套大括号定义嵌套作用域

9. 闭包是一个能访问其外层作用域变量的函数，即便该函数在其他地方被调用（一般匿名函数，有闭包之称）

10. 如果函数没有指定函数值，则默认返回null，如果确实不想返回任何值，则可指定返回类型为Void

六、 操作符

1. 操作符有优先级，从高到低为一元后缀，一元前缀，乘除，加减，移位，位运算，关系运算和类型测试，相等性运算，逻辑运算，null判断，三元表达式，级联调用和赋值

2. 在优先级判断比较模糊的地方，使用（）来提升可读性

3. 算术运算

+、-、\*、/、-expr(负号)、~/(整除)、%(求余)

++var 、var++、--var、var—

相等性和关系运算

==、！=、>、<、>=、<=

==判断值是否相等，如果要判断是否为同一个对象，使用identical()

4. 类型测试

As、is、is!

5. 赋值

=、？？=（null赋值）

-=、/=、%=、>>=、^=、+=、\*=、~/=、<<=、&=、|=

6. 逻辑运算

!expr（取反）、||、&&

7. 位运算

&、|、^、~expr(按位取反)、<<、>>

8. 条件表达式

三元表达式condition?expr1:expr2

Null判断expr1??expr2

9. 级联调用..，严格来说只是个语法糖（如果调用的返回是Void就不能使用级联操作符号）

10. 函数调用（）

11. 小标访问[]

12. 成员访问.

13. 条件式成员访问?.

七、 控制流程语句

1. if else 条件必须为布尔值

2. for loop

1>循环内的闭包会捕获循环变量的当前值

2>迭代对象可使用for in 来迭代其内部元素

1. While , do while

While 先判断条件，满足后再执行循环体，do while 先执行再判断

1. Break ,continue

break终止循环，continue 跳过本次循环剩余代码，直接进入下一次循环

1. Switch case

1>使用== 操作符来比较整数，字符串或编译时常量，包括枚举类型

2>非空case语句正常情况下应该以break结束，也可使用continue,throw和return来结束

3>使用default语句来匹配其他情况

4>空case语句会落入下方case语句

5>可结合使用continue和label来跳转到其他case语句

6>一般用于解释器和扫描器，编写应用时，尽量少看

1. Assert

1>只再开发模式下有效，生产模式下呗忽略

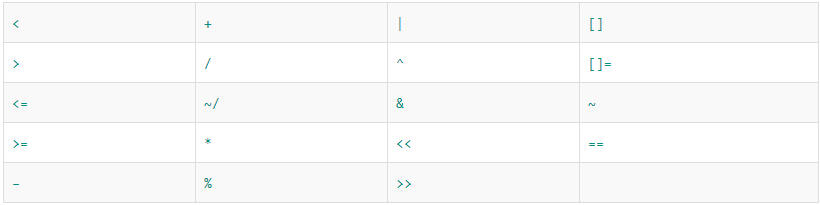
2>条件判断识别时会抛出assertionError异常，可以通过第二个参数指定错误消息

八、 异常

1. 异常表示有一些预料之外错误发生
2. 如果异常未被捕捉，则程序徐将终止运行
3. Dart内置了exception和error两种类型，Exception用于核心库异常，Error用于应用代码异常
4. 可使用throw抛出任何非null对象作为异常
5. Throw语句是一个表达式
6. 使用thy catch 语句来捕捉异常，以防止异常扩散，同时可使用on语句来处理特定类型的异常
7. 使用rethrow来再次抛出异常
8. 使用finally语句来执行，无论是否出现异常都要运行的代码
9. Dart语言进阶
10. 类
11. 使用类成员
    * + 1. 对象的成员包括函数和数据，分别称为方法和实列变量
        2. 方法在某个对象上被调用，可以访问该对象的实列变量
12. 使用构造函数
    * + 1. 构造函数可以是className或className.identifier,关键字new可选
        2. 如果类提供了常量构造函数，可使用const来创建编译时常量，创建2个完全一样的编译时常量将产生同一个实例
        3. 在一个常量上下文里，可以省去构造函数前面的const
        4. 在一个非常量上下文里，如果省去构造函数前面的const，则会创建一个非常量对象
13. 获取对象类型
    * + 1. 运行时使用对象的runtimeType属性来获得对象类型，一个Type类型的对象
        2. 实列变量
        3. 未初始化的实列变量值为null
        4. 每个实列变量都会自动生成一个隐式的getter方法，非final实列变量还会生成setter方法
        5. 在声明实列变量的同时进行的初始化，将先于构造函数和初始化器运行
14. 构造函数
    * + 1. 引用实列变量一般不需要使用this,但如果出现变量名冲突，则可以使用this区分
        2. 在构造函数参数里使用this来简化赋值
        3. 如果不声明构造函数，则构造函数会使用默认构造函数，它会调用父类的无参数非命名构造函数
        4. 构造函数不会继承于父类
        5. 如果需要多个构造函数，可以使用命名构造函数
        6. 默认会调用父类的无参数非命名构造函数，如果没有则需显示调用某个构造函数

构造函数总结：

1. 构造时语句执行顺序依次是初始化器，父类构造函数，子类构造函数
2. 传递给父类构造函数的参数不能访问任何实例变量
3. 在执行构造函数体之前可以先执行初始化器，用来初始化实例变量或者检测参数合法性
4. 在初始化器里可以赋值给final实例变量
5. 可以将某个构造函数重定向到其他构造函数
6. 如果类的实例一旦创建后就不再修改，可使用常量构造函数，要求类的所有实例变量都为final
7. 使用工厂构造函数来返回并不总是全新构造的对象，比如来自缓存的或者子类对象
8. 工厂构造函数里不能使用this
9. 超类构造函数
10. 构造函数执行顺序：初始化参数列表->超类的无名构造函数->主类无名构造函数
11. 初始化列表
12. 在构造函数体执行之前除了可以调用超类构造函数之外，还可以 初始化实例参数。 使用逗号分隔初始化表达式（初始化表达式等号右边的部分不能访问 this）
13. 初始化列表非常适合用来设置 final 变量的值
14. 重定向构造函数
15. 有时候一个构造函数会调动类中的其他构造函数。 一个重定向构造函数是没有代码的，在构造函数声明后，使用 冒号调用其他构造函数
16. 常量构造函数
17. 如果你的类提供一个状态不变的对象，你可以把这些对象 定义为编译时常量。要实现这个功能，需要定义一个 const构造函数， 并且声明所有类的变量为 final
18. 工厂方法构造函数
19. 一个构造函数并不总是返回一个新的对象，则使用 factory 来定义 这个构造函数。例如，一个工厂构造函数 可能从缓存中获取一个实例并返回，或者 返回一个子类型的实例
20. 方法
    * + 1. 实例方法可以访问实例变量
        2. 可以使用getter和setter来创建新属性
        3. 如果不定义函数体，则方法为抽象方法，需要在子类里实现该方法(为;结尾的，并不具体实现的方法)
21. 操作符(operators)



Operator +，operator - .........

1. 抽象类(abstract)
   * + 1. 抽象类不能被实例化，通常用来定义接口
       2. 抽象类通常包含抽象方法，也可包含部分实现
       3. 不是抽象类，但是里面有抽象方法，这样的类是可以被实例化的
2. 隐式接口
   * + 1. 如果不想继承父类方法，可以使用implements来实现父类接口，此时父类可以有多个
       2. 每个类都隐式定义了一个包含其所有成员和实现接口的接口
3. 扩展类
   * + 1. 使用extends来扩展父类，子类中可使用super引用父类
       2. 使用@override 来表示有意地覆盖父类方法
       3. 使用operator来覆盖算术运算，相等运算，比较运算，位运算，下标访问等操作符
       4. 如果覆盖==操作符，需要同时覆盖hashCode
       5. 可以覆盖noSuchMethod来拦截对未定义方法或实例变量的访问
       6. 可以使用 @proxy 注解来避免警告信息,如果你知道编译时的具体类型，则可以 实现这些类来避免警告
4. 枚举类型
   * + 1. 枚举类型是特殊的类，用来表示固定数量的常值
       2. 枚举类型可用于switch语句，case语句需要处理所有可能情况
       3. 枚举类型不能被继承，混入或实现，也不能实例化
5. 混入(mixins)
   * + 1. 混入用来给类增加功能，是多继承模式下的一种代码复用方式
       2. 使用with来混入一个或多个mixin
       3. 混入类需直接继承于Object,并且没有构造函数，如果混入类不需要用作普通类，使用mixin来定义，否则使用class
       4. 使用on来限制可以使用混入的类
6. 类变量和方法
   * + 1. 使用static来定义类级别的变量和方法
       2. 类变量直到类被使用时才进行初始化
       3. 类方法里不能引用this
       4. 类方法是编译时常量，比如可以用作const构造函数的参数
7. 泛型
8. 正确指定泛型类型有助于产生更好的代码
9. 泛型可减少代码重复
10. 列表和哈希表构造函数和字面量均支持类型参数
11. 泛型集合的类型信息会保留到运行时
12. 可使用extends来限制类型参数的可选范围
13. 泛型不仅支持类，还支持方法和函数

注意事项：

* 1. Java 中的泛型信息是编译时的，泛型信息在运行时是不纯在的
  2. 函数中的泛型有几种可能：
     1. 函数的返回值类型 (T)
     2. 参数的类型 (List<T>)
     3. 局部变量的类型 (T tmp)

1. 库
2. 库是模块化的，可分享的代码包
3. 库是一个私有单元，以下划线打头的标识符只能在库内部使用
4. 每个Dart应用都自动成为一个库
5. 使用import来导入另一个库，内置库的URL前缀为Dart,由包管理工具提供的库前缀为package
6. 如果导入的多个库之间出现标识符冲突，可使用as指定引用前缀来避免
7. 使用show和hide来导入或隐藏指定的标识符
8. 使用deferred as 来延迟加载库

注意事项：

* 1. 延迟加载库的常量在导入的时候是不可用的。 只有当库加载完毕的时候，库中常量才可以使用。
  2. 在导入文件的时候无法使用延迟库中的类型。 如果你需要使用类型，则考虑把接口类型移动到另外一个库中， 让两个库都分别导入这个接口库。
  3. Dart 隐含的把 loadLibrary() 函数导入到使用 deferred as 的命名空间 中。 loadLibrary() 方法返回一个 [Future](http://dart.goodev.org/guides/libraries/library-tour" \l "future)

1. 异步
2. Async/await 使得可以像编写同步代码那样编写异步代码
3. 只有在标记为async的函数里可以使用await，该函数在执行到await语句时将暂停，直到await后的表达式运行完成
4. 使用try-catch来捕捉await语句的错误
5. Async函数里可以多次使用await
6. Async函数返回Future对象，普通返回值将自动包装为Future对象
7. 使用await for 来从Stream里读取序列值，如果想中止，可以使用break或return
8. 生成器
9. 生成器惰性地产生序列值
10. 生成器分为同步和异步两种，分别返回Iterable和Stream对象
11. 使用sync/yield来定义同步生成器
12. 使用async/yield来定义异步生成器
13. 使用yield来优化从嵌套的生成器里读取序列值
14. 可调用类
15. 如果想让类可以像函数一样执行，可以给类添加一个Call方法
16. 类型定义(typedef)
17. 类型定义用来命名已有类型
18. 目前只能用来给函数类型命名，以后有望支持更多类型
19. 使用 typedef, 或者 function-type alias 来为方法类型命名， 然后可以使用命名的方法。 当把方法类型赋值给一个变量的时候，typedef 保留类型信息；
20. typedefs 只能使用在 function 类型上，但是将来 可能会有变化
21. 元数据
22. 元数据用来给代码添加额外信息，这些信息在运行时可以通过反射获取到
23. 元数据注解以@打头，其后跟运行时常量或对常量构造函数的调用
24. 内助的元数据注解有@deprecated，@override，@proxy
25. 元数据可出现在这些语句之前，包括库，类，类型定义，类型参数，构造函数，工厂构造函数，函数，类成员，函数参数，变量声明和导入导出
26. 注释
27. 单行代码注释使用//
28. 多行代码注释使用/\*\*/
29. 单行文档注释使用///
30. 多行文档注释使用/\*\* \*/，也可以使用多行连续的///，推荐后者
31. 代码注释在编译后会被丢弃，而文档注释会保留
32. 文档注释里可使用[]来引用当前作用域内可见的函数、类、方法等，一旦注释被转换为文档，引用文字会被替换为文档链接
33. Flutter框架基础

Flutter官网：https://flutter.dev/docs/get-started/install

1. 安装
2. window:
   1. 下载Flutter windows SDK压缩包，解压flutter\_console.bat脚本打开Flutter,执行flutter命令
   2. 或者解压SDK，bin目录的完整路径添加path,直接在普通的windows控制台执行命令
   3. Android -> 安装AS，下载插件Flutter,执行flutter doctor 验证是否安装成功
3. macOS
   1. 下载Flutter windows SDK压缩包，解压bin目录添加path，执行命令
   2. Ios ->
      * 1. 安装xCode9.0及以上
        2. Sudo xcode-select --switch/Applications/Xcode.app/Contents/Developer命令配置Xcode命令行工具使用上一步安装的版本
        3. 打开Xcode或执行sudo xcodebuild -licence命令来接收Xcode许可
4. 布局机制
5. 核心是组件
6. 一切皆是组件，除了可见的图片，图标和文字等内容，还有用来排列，限制，对齐这些可见组件的行，列，网格等
7. 简单的组件来构建复杂组件
8. 布局步骤
9. 选择用来容纳组件的布局组件，Center，Container，Row，Column
10. 创建用来容纳可见内容的组件，Text，Image，Icon
11. 将可见组件添加布局组件，通过传递其属性Child或Children
12. 布局组件添加到页面组件，一般在其build方法里完成
13. 水平和垂直布局
14. mainAxisAlignment 主轴对齐
15. mainAxisSize 主轴尺寸
16. crossAxisAlignment 十字轴对齐
17. Expanded 可以修改子控件的大小
18. 常用布局组件
19. 标准布局组件
    1. Container给组件添加填充，边距，边框，背景色等装饰
    2. GradView可滚动的网格
    3. ListView可滚动的列表
    4. Stack重叠组件
20. Material组件
    1. Card讲相关的组件放到一个卡片里，该卡片带有圆角和阴影
    2. ListTile带有标题和副标题的行，首尾可以添加图标
21. Container
    1. 添加填充，边距和边框
    2. 设置背景颜色或图片
    3. 只能包含一个组件
22. GridView
    1. 用于二维列表
    2. 当内容超过渲染区域时，讲自动滚动
    3. 如果知道单元所处的行列，使用table或dataTable
23. ListView
    1. 特定化的Column,不如Column可定制性强，但更容易移动和支持滚动
    2. 支持垂直或水平滚动
24. Stack
    1. 用来重叠多个组件
    2. 子组件列表里后面的重叠在前面之上
    3. 里面的内容无法移动
    4. 超过渲染区域的内容可剪切掉
25. Card
    1. 用来呈现一组相关性的信息
    2. 只接受一个子组件，但子组件可以是row,column这样的接受多个子组件的组件
    3. 带圆角和阴影
    4. 内容不能移动
    5. 需在materialApp内使用
26. ListTile/ListTileBar
    1. 特定化的Row,包含最多3行文字，以及可选的图标
    2. 可定制性不如row,但更易使用
    3. 需在materialApp内使用
27. 布局实例
28. 导航器
29. 导航器（navigator）是一个按照堆栈方式管理其子组件（页面）的组件
30. 内部使用overlay组件来管理多个页面，位于最上层的页面可见
31. 通过调整overlay里的页面顺序来切换页面
32. 页面（page）也称为screen或route
33. widgetApp和MaterialApp会自动创建一个导航器，组件里可使用Navigator.of来获取其祖先组件里的导航器
34. 进入新页和返回
35. 使用home属性来指定MaterialApp的首页，该页位于导航器堆栈的最底层
36. 使用Navigator.push方法来进入某一个页面，需传递一个MaterialPageRoute对象
37. 使用Navigator.pop方法来返回上一页
38. 如果页面使用了Scaffold，那么导航条上会自动出现返回按钮，android上可用返回实体按键返回
39. 跨页面传递数据
40. 使用命名路由
41. 命名路由用来简化导航代码，减少代码重复
42. 先创建命名路由，然后使用Navigator.pushNamed来进入指定名字的路由
43. 路由创建是在materialApp中创建
44. 跨页面传递组件
45. 把一个组件以动画的方式从一个页面传递到下一个页面，能够在两个页面之间建立视觉瞄点链接，从而起到引导用户的作用
46. 使用hero组件来跨页面传递组件
47. 导航器嵌套
48. 导航器嵌套可用来创建应用模块内导航，比如常见的tabBar导航，每个tab内部都有一个独立的导航器
49. 应用内所有导航器为树状结构，有一个根root导航器，同一节点下各兄弟导航器并行导航
50. 使用willpopScope来使内层导航器响应android实体返回键
51. 无状态和有状态组件
52. 组件分为无状态（icon,text）和有状态(textField,checkbbox)2种
53. 组件会变化（用户交互或时间）则为有状态，否则无状态
54. 组件的状态由许多值组成，这些值决定了组件外观的可变部分
55. 组件状态存放在其state对象里，以便分离外观和状态
56. 如果状态变化，需调用setState通知框架重绘组件
57. 创建有状态组件
58. 创建有状态组件需创建2个类，一个用来定义外观（继承statefulWidget）,另一个包含状态和定义build方法（继承state）
    1. 确定组件状态放置在哪里
    2. 创建继承于statefulWidget组件类
    3. 创建包含业务逻辑和内部状态的继承于state的状态类
    4. 讲组件插入组件树里
59. 管理状态
60. 有状态组件可以放在其内部或其父组件里，又或者2者都有
61. 如果是用户输入数据（比如复选框是否勾选），最好放在父组件里
62. 如果是有关界面美观度的（比如动画），则最好放在组件内
63. 如果有疑问，可放在父组件里
64. 手势
65. Flutter手势系统分2层，下层识别原始指针事件，描述指针位置和移动，上层识别手势，语义动作，由一个或多个事件组成
66. 共4种指针事件，分别是pointDownEvent,pointMoveEvent,pointUpEvent,pointCancleEvent
67. 指针按下，事件派送给最内层组件，从这开始所有后续指针事件以冒泡方式向上传递给到根组件的所有途径组件
68. 没办法取消或停止指针事件冒泡传递
69. 可使用listener来监听指针事件，不过应尽量使用更高层的手势
70. 手势表示点击，拖动，缩放这样的语义动作，通过多个独立的指针事件识别出，甚至是多个独立的指针
71. 手势的生命周期内会发送一个或多个事件，使用gestureDetector来监听手势事件
72. 点击（tap）onTapDown,onTapUp,onTap,onTapCancel
73. 双击（double tap）onDoubleTap
74. 长按（long press）onLongpress
75. 垂直拖动（vertical drag）onveriticalDragStart,onVerticalDragUpdate,onVeriticalDragEnd
76. 水平拖动（horizontal drag）onHorizontalDragStart,onHorizontalDragUpdate,onHorizontalDragEnd
77. 平移（pan）onPanStart,onPanUpdate,onPanEnd
78. GestureDetector通过检查哪些手势事件的处理器不为null来获悉需要识别哪些手势
79. 当有多种手势需要识别时按以下规则消除歧义
    1. 任何时候，某种手势的识别器都可以宣告失败，直到只剩下一个识别器，他就是胜利者
    2. 任何时候，某种手势的识别器都可以宣告胜利，其他识别讲自动失败
    3. 当只有一种手势需要识别，识别比较快，因为不需要消除歧义
80. 交互实例
81. 使用2个状态值来分别保存是否收藏和被收藏次数
82. 通过状态值是否被收藏显示实心还是空心图标
83. 使用iconbutton创建可点击的图标
84. 更多widget可参考flutter方法文档
85. App开发实战