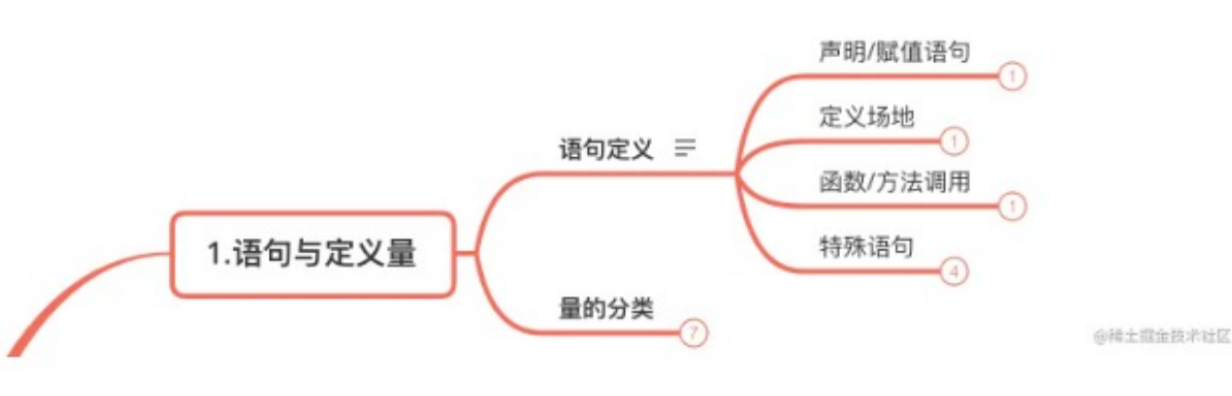
一、认识语句定义

> 视频版见:《Flutter梦始之地 - 了解语句定义，开始牙牙学语》

编程语言比之于社会语言，则代码比之于文字，语句比之为言辞。所以学习编程中的语句定义，就相当于了解如何通过代码来说话。语句定义中，最基础的是 `声明/赋值语句` 和 `函数/方法调用语句` 。



1. 宇宙的终极三问

```dart
我是谁?
我从哪里来?
我要到哪里去?
```

语句的表达就是为了解决 `量` 的这三个问题。 比如下面：

- 【1】 通过 `str` 指代 `hello, world!` 字符串对象，解决了 `我是谁` 的问题。
- 【2】 `str` 在 `foo` 函数中，解决了 `我从哪里来` 的问题。
- 【3】在 `print` 函数中使用了 `str` ，解决了 `我要到哪里去` 的问题。

```dart
---->[grammar/statement/01.dart]----
// 我从哪里来? (在哪里)
void foo(){
  // 我是谁? (是什么)
  String str = 'hello, world!';
  // 我要到哪里去? (做什么)
  print(str);
}
```

把握着三个问题，就不会在编码中迷失方向。

2. 认识语句的价值

代码中的一切逻辑都是对 `量` 的维护和使用，而 `量` 通过语句进行声明和赋值，通过 `函数/方法` 的调用实现功能。所以可见语句的重要性，它是一切逻辑表述的单元。

一个 `函数/方法` 可以包含若干个 `语句` ，就像是由若干句子形成的 `段落` 。一个 `函数/方法` 的调用本身也算是一条语句，若干个方法，就构成了源码的 `篇章` 。其实 `类` 的本身，就是通过语句来维护 `量` 和 `方法` ；可就是常说的 `成员属性` 和 `成员方法` 。
`*注*`：如果你还不知道 `类` 是什么，没有关系，这里简单了解一下，后面会进行介绍。

```dart
// 我从哪里来? (在哪里)
void foo() {
  // 我是谁?(是什么)
  Person toly = Person('toly'); // 赋值语句
  // 我要到哪里去? (做什么)
  toly.say(); // 方法语句
}

class Person {
  final String name; // 声明语句

  Person(this.name); // 方法语句

  void say(){
    print('我是 $name'); // 方法语句
  }
}
```

从这里可以看出语句是非常重要的，其中的量 `是什么` 、 `在哪里` 、 `做什么` ，这三点应该成为你的指路明灯，让你不会在各种复杂的逻辑中被绕的晕头转向。特别是当你在探索源码，或分析别的代码时，在深邃的源码之海中，很容易迷失方向。

3.语句定义的语法

赋值语句的语法是 `类型 变量名 = 对象` 。这就相当于为对象取个 `名字` ，进行指代，方便后面的使用。

```dart
---->[grammar/statement/03.dart]----
//类型 变量名 = 对象
String toly = 'toly'; // 赋值语句
```

不过，有时候并不能直接给出 `量` 的值，可以先进行声明，在后续的逻辑中进行赋值。

```dart
// 等价于
String toly2; // 声明语句
toly2 = 'toly2'; // 赋值语句
toly2 = toly; // 赋值语句
```

另外，在声明和赋值时，可以通过某些 `关键字` 进行修饰。比如通过 `const` 修饰编译期常量、通过 `final` 修饰运行期不可变量。如 `t2` 所示，被关键字修饰的量可以省略 `类型` 名，但一般并不推荐这么做，这只会增加阅读的压力。

```dart
// 修饰符 类型 变量名 = 对象;
const String t1 = 't1'; // 赋值语句

// 修饰符 变量名 = 对象; (不推荐)
final t2 = t1*2; // 赋值语句
```

通过修饰，就是对量进行 `限制` ，比如通过 `const` 修饰的量不能再被赋值：

```dart
  // 修饰符 类型 变量名 = 对象;
  const String t1 = 't1'; // 赋值语句

  // 修饰符 变量名 = 对象; (不推荐)
  final t2 = t1*2; // 赋值语句

  t1 = t2;
}
```

Constant variables can't be assigned a value. (Documentation)

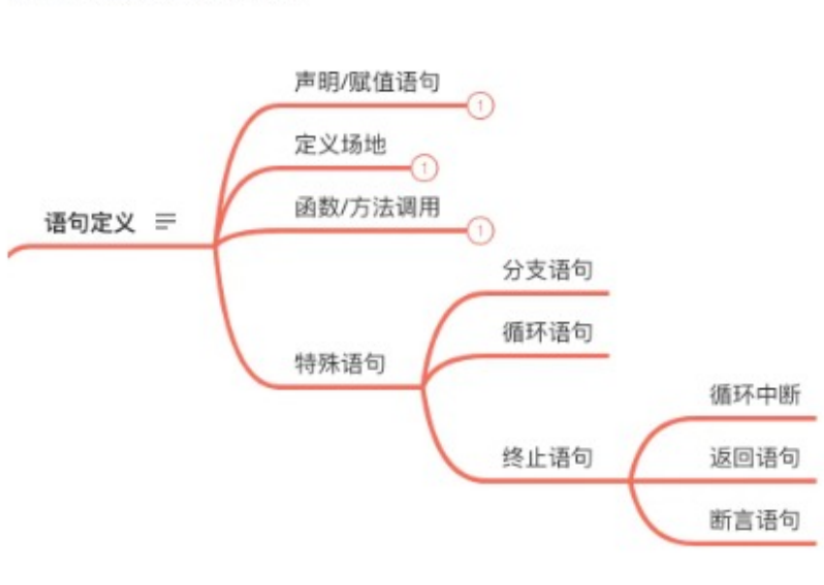
Try removing the assignment, or remove the modifier 'const' from the variable.

4. 函数/方法调用语句

在函数/方法中可以指定若干个参数，并通过返回语句来提供结果，所以有时 `方法语句` 和 `赋值语句` 会结合。我们一般把 `函数/方法调用` 和 `声明/赋值` 这两个基础语句，统称为 `表达式` 。也就是说，通过它们，你可以向计算机表达最基础的诉求。

```dart
---->[grammar/statement/04.dart]----
main() {
  int result = add(1,2); // 方法语句 + 赋值语句
}

int add(int a ,int b ) {
  int c = a + b; // 赋值语句
  return c; // 返回语句
}
```

另外，还有一些在特殊场合使用的 `特殊语句` ，这些在后续的学习中会逐步介绍。现在才刚开始牙牙学语，了解最基础的话这么说就行了。知识的累积是一个过程，每个阶段都有核心的要务，不要想着一蹴而就。人对知识的接受能力是受到自身知识储备影响的，就像你不能一开始就教小学生微积分或文言文。



二、量的分类

就像人类社会由各不相同的人，实现自己的职能，维护社会运转一样。对于一个应用来说，也是各种不同的量，发挥自己的作用，维护着应用的正确运转。 `量` 是一种指代关系，通过一个名称指代真实的对象，从而建立对应关系。

> 视频版见:《Flutter 梦始之地 - 量的定义，逻辑运算的根基》

1. 生活中的量

日常中我们所说的 `重量` 、 `降雨量` 、 `产量` 等，本质上来说就是通过 `数字` 表示某种属性的 `多少` 。有了指代关系，可以便于分析，比如鸡兔同笼的问题：

```lua
今有雉兔同笼，上有三十五头，下有九十四足，问雉兔各几何?
注: 【雉】 文言中的野鸡。
                    ----《孙子算经》
```

通过量进行指代，可以简化问题模型，让实际的问题仅暴露出它内在的关系。其实这个问题，并不关心是 `兔` 、还是 `牛` 、 `羊` 、 `马` ，只关心这种动物有几个头，几只脚。

```dart
设 雉数 = x ;  兔数 = y;
根据题目已知条件可得:
x + y = 35;
2x + 4y = 94;
```

通过 `x` 指代鸡的数量，也就是说知道了 `x` 是多少，就等价于知道了 `鸡有多少个` 。这就是通过 `量` 来构建 `对应关系` 的好处。从实际问题中简化模型，寻求算法，解决问题。

```dart
y = (94 - 35*2)/2;
x = 35 - y;
```

2.编程中的量

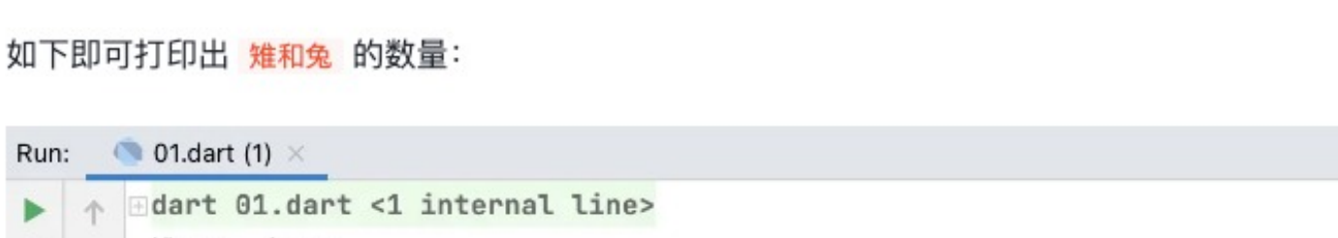
编程中我们用量来指代对象，就像现实中给人起个名字，方便喊他做事。

```dart
---->[grammar/variable/01.dart]----
main(){
  int n = 35; // 头数
  int m = 94; // 足数
}
```

同样，我们通过 `n` 和 `m` 分别对 `头数` 和 `足数` 进行指定，在算法中就可以通过量的名称进行运算。

```dart
main(){
  int n = 35; // 头数
  int m = 94; // 足数

  int y = (m - n*2) ~/ 2;
  int x = n - y;
  print("雉:$x  兔:$y");
}
```

如下即可打印出 `雉和兔` 的数量：



```
dart 01.dart <1 internal line>
雉:23  兔:12
```

3.量的分类

在编程中，量主要分为如下四种，通过不同的关键字对量进行修饰：

默认情况下，声明的量都是 `可变量` 。

```dart
int a = 10;
a = a*2; // 允许修改量的值
```

另外有一个 `var` 关键词可以修饰可变量，此时不需要指定 `量` 的类型。编译器会自己推导，但这种方式 `不推荐` 使用。至少在 `Flutter` 框架的源码中，没有任何一处使用 `var` 关键字声明变量的。因为对于编写者和阅读者来说，明确变量类型非常重要。

```dart
var a = 10;
a = a*2; // 允许修改量的值
var b = 'hello';
```

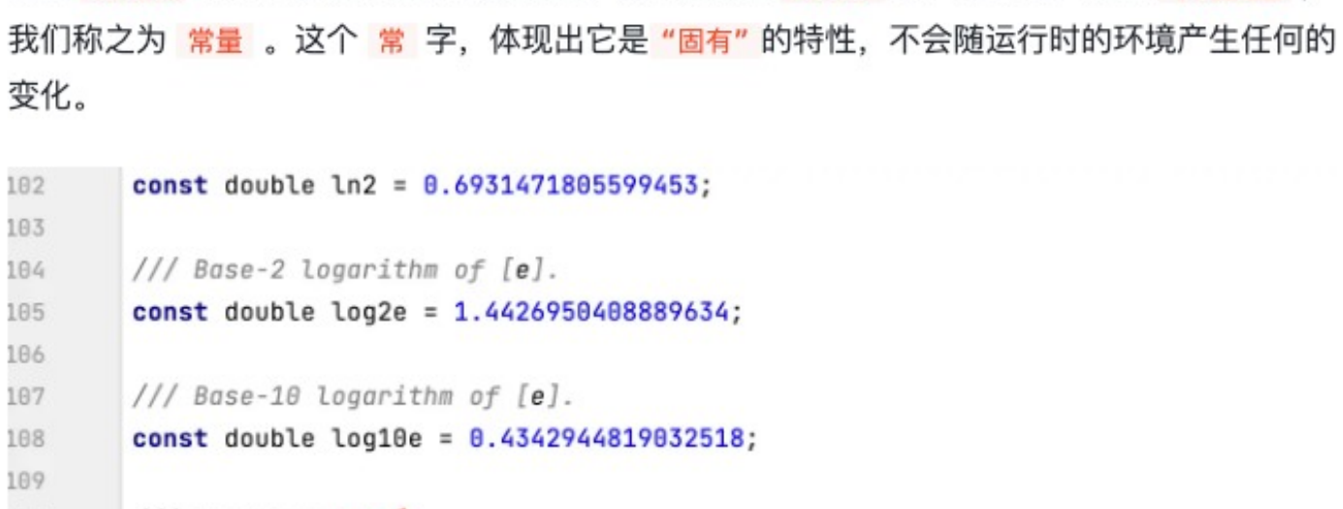
与 `可变量` 相对应的是 `不可变量` ，通过修饰符来进行限制，其中包括 `final` 和 `const` 关键字。打个比方，一个学生入学之后才会分配学号，而且学号在入学期间始终不变，这样的量就是 `final` 所修饰的不可变量，你可以在 `运行期间` 可以为其赋值一次，之后该量不能再指代其他对象。 如下代码所示，被 `final` 和 `const` 修饰的量不可以再被赋值。

```dart
f0(){
  final int a = 10;
  a = a*2; // 不允许修改量的值
}

f1(){
  const int a = 10; // 不推荐
  a = a*2; // 不允许修改量的值
}
```

源码中倾向于对一切没有修改需求的量使用 `final` 修饰，我也是这么推荐。因为这样可以从根本上保证：没有修改需求的量不会被修改，从而消除潜在的隐患，这也是一个很好的编程习惯。另外说句题外话：有些编程语言，比如 `rust` ，默认定义的是 `不可修改` 量。

```dart
void attachRootWidget(Widget rootWidget) {
  final bool isBootstrapFrame = renderViewElement == null;
  _readyToProduceFrames = true;
  _renderViewElement = RenderObjectToWidgetAdapter<RenderBox>(
    container: renderView,
    debugShortDescription: '[root]',
    child: rootWidget,
  ).attachToRenderTree(buildOwner!, renderViewElement as RenderObjectToWidgetElement<RenderBox
  if (isBootstrapFrame) {
    SchedulerBinding.instance!.ensureVisualUpdate();
  }
}
```

通过 `const` 来修饰那些在程序运行前就已经确定的量，比如一些永恒不变的数字，如圆周率。通过 `const` 修饰的量也不允许再被修改。可以感觉出 `const` 是一种更为严格的 `不可变量` ，我们称之为 `常量` 。这个 `常` 字，体现出它是 `“固有”` 的特性，不会随运行时的环境产生任何的变化。

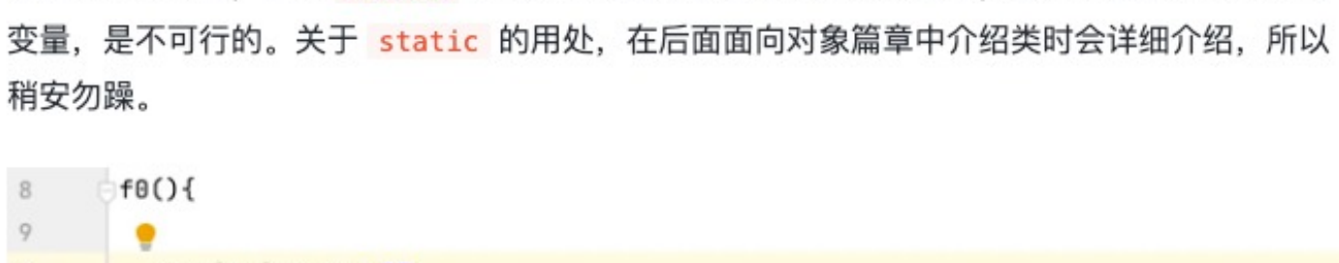


```dart
const double ln2 = 0.6931471805599453;

/// Base-2 logarithm of [e].
const double log2e = 1.4426950408889634;

/// Base-10 logarithm of [e].
const double log10e = 0.4342944819032518;

/// The PI constant.
const double pi = 3.1415926535897932;
```

关于常量，先认识到这种程度就行了。另外关于 `const` 的用法，有一个专属视频，新手的话不必急着看：

> 《Flutter 梦始之地 - const关键字的一点点小细节》

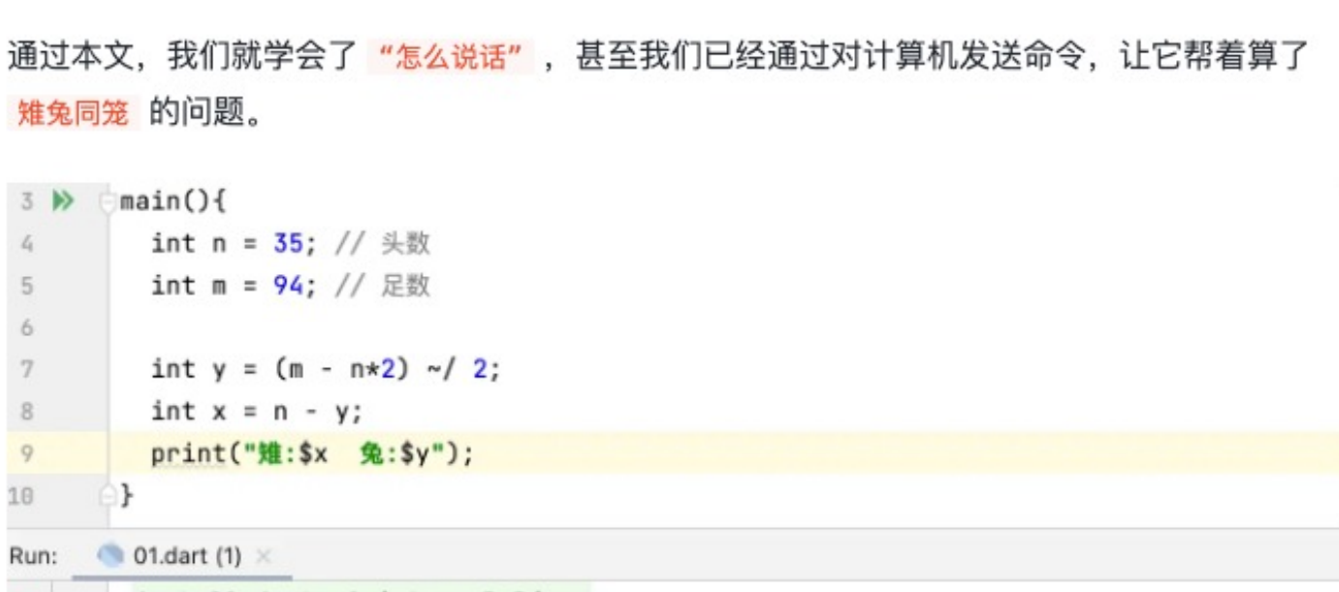
最后关于静态量，通过 `static` 修饰。不过这个关键字只能用于类中，比如下面在方法中修饰变量，是不可行的。关于 `static` 的用处，在后面面向对象篇章中介绍类时会详细介绍，所以稍安勿躁。



```dart
f0(){

  static int a = 10;
  a = a
}
```

Can't have modifier 'static' here.

通过本文，我们就学会了 `“怎么说话”` ，甚至我们已经通过对计算机发送命令，让它帮着算了 `雉兔同笼` 的问题。



```dart
main(){
  int n = 35; // 头数
  int m = 94; // 足数

  int y = (m - n*2) ~/ 2;
  int x = n - y;
  print("雉:$x  兔:$y");
}
```

```
dart 01.dart <1 internal line>
雉:23  兔:12
```

了解如何通过语句来 `定义量` 、 `操作量` ，就可以和计算机进行基础的 `“交流”` 了，也就已经一只脚跨入了大门。再次强调，计算机和语言只是工具，不是目的，在解决问题时，人仍然需要发挥主观能动性。

留言

输入评论（Enter换行，⌘ + Enter发送）

发表评论

全部评论（6）

juedui0769 JY.4 | 3月前
? 和 ! 应该重点讲讲的，写代码时很容易报错，要么是没加?，再就是调用时没加 ! 。
点赞 1

> **张风捷特烈** （作者） | 3月前
> 13 章：3. 空安全机制 中有介绍
> 2 回复

萌果爱吃柠檬 LV.3 JY.4 | Android开发 @ zjgsu | 5月前
“另外，在声明和赋值时，可以通过某些 关键字 进行修饰。比如通过 const 修饰编译器常量”编译器 -> 编译期
点赞 1

> **张风捷特烈** （作者） | 5月前
> 已更正
> 点赞 回复

codehelen JY.3 | 工程 | 5月前
“人对知识的接受能力是收到自身知识储备影响的” 收到 -> 受到
点赞 1

> **张风捷特烈** （作者） | 5月前
> 已更正
> 点赞 回复