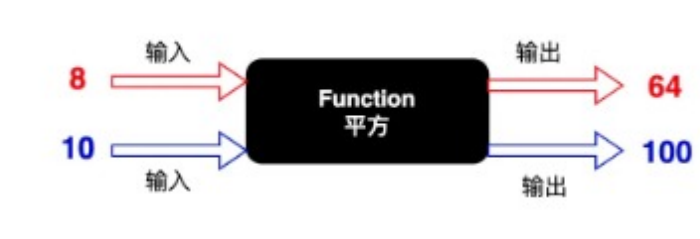
> 本章视频版见: 《Flutter 梦始之地 - 函数方法，程序逻辑的基石》

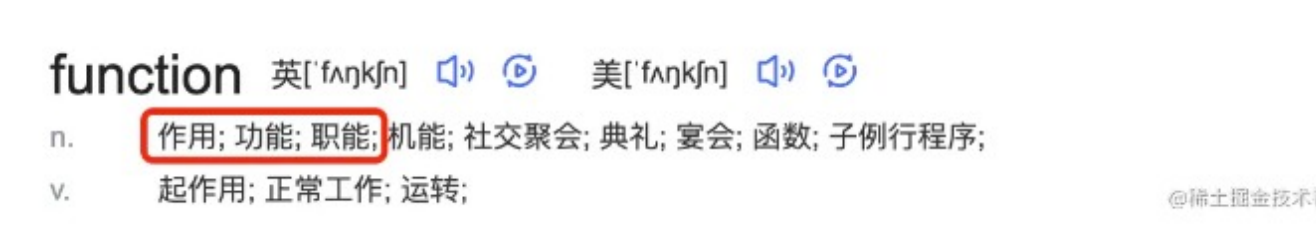
一、函数是什么

我们可以把函数理解成一个黑箱，不同的输入，会通过黑箱中的逻辑处理产生一个对应的输出。在数学意义上，这样的输入是 `自变量 x` ， 输出是 `因变量 y` ，其中黑箱的处理逻辑称为对应法则 `f` 。三者的关系可以通过 `y = f(x)` 来表示。



1. 编程中的函数与方法

数学中的 `函数` 是强调两个 `集合` 间的映射关系，这对于编程中的函数而言，还是有区别的。编程中并不强制输入和输出的数量， `function` 更强调的是 `作用` 和 `功能` ，其目的是完成特定的 `任务` ，而非仅是维护映射关系。



function 英[ˈfʌŋkʃn] 美[ˈfʌŋkʃn]
n. 作用; 功能; 职能; 机能; 社交聚会; 典礼; 宴会; 函数; 子例行程序;
v. 起作用; 正常工作; 运转;

在 `C` 语言或 `C++` 中，一般习惯称一个功能团为 `函数 function` ；在 `Java` 语言中，对于 `类中` 的功能团，习惯称之为 `方法 method` 。函数也好，方法也好，都是用于完成特定任务的功能团，本质上并没有太大的区别，根据个人的喜好即可。

就像对于饺子，由于南北方的地域差异，有着不同的做法，但本质上而言没有什么太大的区别，都是 `面皮` 包 `馅料` 而已，我们可以将其都是为 `广义的饺子` 。这种文化上的差异是不能强行统一的，所以有入乡随俗的说法。这种并不影响社会(程序)运行的差异性，我们应抱有包容和尊重。又不是在做学术研究，没必要那么较真。

北方饺子	南方饺子
	

2. 函数的定义

在 `dart` 中，定义函数并不需要关键字，如下是一个简单的函数，用于通过输入的值，返回其平方。

```dart
---->[grammar/function/01.dart]----
int square(int x) {
  return x * x;
}
```

下面是一个函数的基本结构，如果方法没有返回值， `返回值类型` 则为 `void` 。

```ini
返回值类型 方法名(参数类型 参数名){
   方法体;
}
```

另外 `返回值类型` 和 `参数类型` 是可以省略的，但在日常开发中这是 `不推荐` 的。还是那句话， `明确类型` 无论是对编码者还是阅读者都非常重要，没有必要偷工减料，而额外增加阅读的负担。

```
square(x) {
  return x * x;
}
```

3. 函数的简写

当函数体中只要一行表达式时，可以通过 `=>` 进行简写，返回表达式的值。这样可以省去 `{}` 和 `return` 关键字，从而让函数看起来更简洁。

```dart
---->[grammar/function/02.dart]----
int square(int x) => x * x;
```

二、函数的参数与调用

不同的编程语言，有不同的参数特性。对于 `Dart` 而言，支持 `命名参数` 、 `位置参数` 、 `默认参数` ，总的来说还是比较丰富的。

1. 多个参数

函数中可以传入多个参数，在 `()` 中依次列出。拿上一篇中 `雉兔同笼` 的解决算法来看，可以把核心逻辑提取成 `solution` 丰富，其中入参 `n` 和 `m` 分别表示 `头数` 和 `足数` 。

```dart
---->[grammar/function/02.dart]----
void solution(int n, int m) {
  print("头数:$n  足数:$m");
  int y = (m - n * 2) ~/ 2;
  int x = n - y;
  print("雉数:$x  兔数:$y");
}
```

方法的调用时，需要依次传入参数值，如下所示：

```dart
solution(85, 194);
```

```
头数:85  足数:194
雉数:73  兔数:12
```

2.命名参数的作用

命名参数有很多好处，其一是：它可以使 `语义更加明确` ，比如上面的 `solution(85, 194)` ，如果直接给别人看，需要看注释，或理清函数逻辑才能知道那个参数是 `头数` ，哪个参数是 `足数` 。 在函数定义时，通过 `{}` 可以设为命名参数。

```dart
---->[grammar/function/04.dart]----
void solution({
  required int head,
  required int foot,
}) {
  print("头数:$head  足数:$foot");
  int y = (foot - head * 2) ~/ 2;
  int x = head - y;
  print("雉数:$x  兔数:$y");
}

```

在调用，通过 `名称: 入参` 的形式进行传参，这样可以明确参数的名称，便于理解。

```dart
solution(head: 85, foot: 194);
```

其二，命名参数无需按照参数顺序进行传递，比如下面的写法和上面是一样等效的。这就摆脱了必须按照参数顺序传参的魔咒，特别是入参非常多的时候，如果不用命名参数，调用时还得记着每个参数的顺序。

```dart
solution(foot: 194, head: 85);
```

3.默认参数

其三，在 `命名参数` 中可以设置默认值，语法是 `{类型 参数名 = 默认值}` 。下面是让字符串 `src` 重复 `count` 次的测试函数：

```dart
---->[grammar/function/05.dart]----
String repeat({
  String src = '张风捷特烈',
  int count = 2,
}) {
  return src * count;
}
```

`默认参数` 顾名思义，就是未指定参数时，使用函数中指定的默认值。比如下面调用 `repeat` 方法，未传入参数，返回值就是 `张风捷特烈` 重复两次。

```dart
String s1 = repeat();
---->[输出]----
张风捷特烈张风捷特烈
```

再比如下面传入 `src` 和 `count` ，那么函数的默认值就无效了，返回值是 `toly` 重复 `5` 次。

```dart
String s1 = repeat(src: 'toly',count: 5);
---->[输出]----
tolytolytolytolytoly
```

可以看出 `默认参数` 是非常方便的功能，它可以让函数的使用更加便捷。

4. 位置参数

位置参数通过 `[参数1, 参数2, ...]` 来指定，其特点是必须 `按顺序` 依次进行指定若干入参。正是因为这种特性，一般来说用于参数有前后顺序的场景。最典型的就是 `DateTime` 的构造方法，其中必须传入 `year` 参数。之后的参数为 `位置参数` ，是选填的，但必须按照顺序进行填写。注意一点 `位置参数` 也允许提供 `默认值` 。

```dart
DateTime(int year,
      [int month = 1,
      int day = 1,
      int hour = 0,
      int minute = 0,
      int second = 0,
      int millisecond = 0,
      int microsecond = 0])
```

比如下面就表示 `1994 年 3 月 28 日` ，只传了 `[month, day]` ，后面的参数使用默认值。

```dart
---->[grammar/function/06.dart]----
DateTime(1994,3,28);
```

比如下面表示 `1994 年 3 月 28 日 4 时 35 分`，这时应该能体会出 `位置参数` 的用意了吧。

```dart
DateTime(1994,3,28,4,35);
```

如果说 `命名参数` 是为了方便使用而允许乱序，提供名称传参；那么 `位置参数` 就是不允许乱序，依次传参。两者各有其优劣，如果入参是由严格顺序，并且想支持默认参数的，可以考虑 `位置参数` 。

三、函数类型

函数可以通过 `typedef` 定义类型，如下定义了一个 `Operation` 类型，表示一种入参是 `double` ，返回值是 `double` 的函数。

```dart
---->[grammar/function/07.dart]----
typedef Operation = double Function(double);
```

1.函数对象的声明和使用

如下，可以声明一个 `Operation` 类型的 `op` 变量来指代 `square` 函数。

```dart
main(){
   Operation op = square;
}

double square(double a) {
  return a * a;
}
```

这样就可以通过 `op` 变量触发该函数，这本质上和直接使用 `square` 没有区别。

```dart
op(10);
op.call(10);
```

那可能会有人问，那这有什么意义呢？通过函数类型，可以让我们把函数当做 `对象` 来看待，这就表示 `op` 变量可以在编程中指代其他符合方法签名的其他函数。比如 `tag1` 处，将 `op` 赋值为 `cube` 函数，计算入参的三次方。

```dart
Operation op = square;
print(op(10)); // 100
print(op.call(10)); // 100

op = cube; // tag1
print(op(10)); // 1000

double cube(double a) => a * a * a;
```

2.函数类型方法入参

函数既然可以当做 `对象` 看待，那自然就可以作为函数的入参，这也是函数类型的价值体现。如下在 `add` 方法中传入了类型为 `Operation?` 的命名参数 `op` ，其中 `?` 表示该入参可为空。

```dart
---->[grammar/function/08.dart]----
double add(double a, double b, {Operation? op}) {
  if (op == null) return a + b;
  return op(a) + op(b);
}
```

其中的逻辑是：先通过 `op` 函数对 `a` 、 `b` 进行操作，再累加。比如下面把 `square` 作为 `op` 的参数，则表示计算 `3` 、 `4` 的平方和。

```dart
double result = add(3, 4, op: square);
print(result); // 25
```

你可以直接为 `op` 提供一个 `匿名函数` ，如下是计算 `3` 、 `4` 的立方和。编程中一般称 `匿名函数` 为 `lambda表达式` 。在后面我们会接触 `回调函数` ，其本质就是利用函数类型作为方法入参，在何时的时机触发方法而已。

```dart
double result = add(3, 4, op: (double e) => e * e * e);
print(result);
```

通过 `函数类型` 把函数对象化，可以更方便指代和操作。

3.函数的作用

最后来思考一下，函数的作用是什么？可以看出，函数中可以集成若干语句，通过逻辑算法完成某项任务，这就是最基础的封装概念。当别人通过函数，封装了一个非常实用的功能，当这段代码被共享。你只需要调用函数即可，不必了解其中具体的实现细节，该函数对你来说就是一个实现某种功能的 `黑盒` 。

这就导致，普通的开发者，也能通过 `顶尖开发者` 分享的代码实现顶尖的功能。这其实有利也有弊，利在于：普通的开发者即使能力不强，也可以通过引入三方库，或者复制粘贴来完成功能需求。这就大大降低了编程中实现功能的门槛，也有利于集中人类智慧去共同解决一些问题。

弊在于，理想很丰满，现实很骨感。就像吃大锅饭一样，这会导致很大一批人，过于依赖别人，缺乏自主思考的能力，当出现问题或找不到库时，就举步维艰。就像一个人平时只会抄别人的作业，自我感觉良好，作业都能及时上交。一旦在某些特殊场景下，没得抄的时候，才会发现自己其实什么都不会。

留言

输入评论（Enter换行，⌘ + Enter发送）

发表评论

全部评论（14）

aYuan LV.2 JY.4 | 前端攻城狮 @ 知交科技 | 3月前
老师讲得非常详细、"函数方法入参"这里有个语义错误、"编程中一般成 匿名函数 为 lambda表达式"、成、是不是想说"称"呢？😍
点赞 1

> **张风捷特烈** （作者） | 3月前
> 已更正😁
> 点赞 回复

苏灿烤鱼 LV.4 JY.5 | 安卓 | 4月前
老师，最后一节说得非常好👍，最后一段有一个错别字，"显示很骨感"应该是"现实很骨感"。
点赞 1

> **张风捷特烈** （作者） | 4月前
> 已更正
> 点赞 回复

海明没有威 JY.4 | 4月前
函数作为参数，一定要区分：func 和 func() 的区别，前者是作为参数，后者是返回值作为参数，而且后者会在传参的时候调用这个函数😭
1 回复

大大大大超 JY.4 | 高级前端开发工程师 | 5月前
写的真好，爱了。
点赞 回复

听蝉 LV.3 JY.5 | Flutter @ 编程黑板报 | 5月前
看了饺子，肚子好饿，就没劲看下面文章了，建议饺子图放文末。
1 回复

禅是禅非 LV.1 JY.4 | ios开发 @ 失业人员 | 5月前
安徽人表示 县南是北方饺子 县北是南方饺子
点赞 回复

故乡的原风景 LV.1 JY.3 | 5月前
写的好
点赞 回复

月饼393 JY.3 | 6月前
北方的饺子和南方的馄饨长一个样😂
点赞 回复

Mr_boya JY.4 | 移动开发 | 6月前
北方饺子和南方饺子图没反吗?🙀
点赞 3

> **张风捷特烈** （作者） | 6月前
> 无论怎么放，总会有人说放反😏
> 点赞 回复
>
> **用户149531...** 回复 **张风捷特烈** | 6月前
> 🤔
> "无论怎么放，总会有人说放反😏"
> 点赞 回复
>
> 查看更多回复