

08 | 综合案例：掌握Dart核心特性

2019-07-16 陈航

Flutter核心技术与实战

[进入课程 >](#)



讲述：陈航

时长 10:45 大小 9.85M



你好，我是陈航。

在前两篇文章中，我首先与你一起学习了 Dart 程序的基本结构和语法，认识了 Dart 语言世界的基本构成要素，也就是类型系统，以及它们是怎么表示信息的。然后，我带你学习了 Dart 面向对象设计的基本思路，知道了函数、类与运算符这些其他编程语言中常见的概念，在 Dart 中的差异及典型用法，理解了 Dart 是怎么处理信息的。

可以看到，Dart 吸纳了其他编程语言的优点，在关于如何表达以及处理信息上，既简单又简洁，而且又不失强大。俗话说，纸上得来终觉浅，绝知此事要躬行。那么今天，我就用一个综合案例，把前面学习的关于 Dart 的零散知识串起来，希望你可以动手试验一下这个案例，借此掌握如何用 Dart 编程。


有了前面学习的知识点，再加上今天的综合案例练习，我认为你已经掌握了 Dart 最常用的 80% 的特性，可以在基本没有语言障碍的情况下去使用 Flutter 了。至于剩下的那 20% 的特性，因为使用较少，所以我不会在本专栏做重点讲解。如果你对这部分内容感兴趣的话，可以访问[官方文档](#)去做进一步了解。

此外，关于 Dart 中的异步和并发，我会在后面的第 23 篇文章“单线程模型怎么保证 UI 运行流畅？”中进行深入介绍。

案例介绍

今天，我选择的案例是，先用 Dart 写一段购物车程序，但先不使用 Dart 独有的特性。然后，我们再以这段程序为起点，逐步加入 Dart 语言特性，将其改造为一个符合 Dart 设计思想的程序。你可以在这个改造过程中，进一步体会到 Dart 的魅力所在。

首先，我们来看看在不使用任何 Dart 语法特性的情况下，一个有着基本功能的购物车程序长什么样子。

 复制代码

```
1 // 定义商品 Item 类
2 class Item {
3     double price;
4     String name;
5     Item(name, price) {
6         this.name = name;
7         this.price = price;
8     }
9 }
10
11 // 定义购物车类
12 class ShoppingCart {
13     String name;
14     DateTime date;
15     String code;
16     List<Item> bookings;
17
18     price() {
19         double sum = 0.0;
20         for(var i in bookings) {
21             sum += i.price;
22         }
23         return sum;
24     }
25
26     ShoppingCart(name, code) {
```


```

27     this.name = name;
28     this.code = code;
29     this.date = DateTime.now();
30 }
31
32 getInfo() {
33     return '购物车信息:' +
34         '\n-----' +
35         '\n 用户名: ' + name +
36         '\n 优惠码: ' + code +
37         '\n 总价: ' + price().toString() +
38         '\n 日期: ' + date.toString() +
39         '\n-----';
40 }
41 }
42
43 void main() {
44     ShoppingCart sc = ShoppingCart('张三', '123456');
45     sc.bookings = [Item('苹果',10.0), Item('鸭梨',20.0)];
46     print(sc.getInfo());
47 }

```

在这段程序中，我定义了商品 Item 类，以及购物车 ShoppingCart 类。它们分别包含了一个初始化构造方法，将 main 函数传入的参数信息赋值给对象内部属性。而购物车的基本信息，则通过 ShoppingCart 类中的 getInfo 方法输出。在这个方法中，我采用字符串拼接的方式，将各类信息进行格式化组合后，返回给调用者。

运行这段程序，不出意外，购物车对象 sc 包括的用户名、优惠码、总价与日期在内的基本信息都会被打印到命令行中。

 复制代码

```

1 购物车信息:
2  -----
3  用户名: 张三
4  优惠码: 123456
5  总价: 30.0
6  日期: 2019-06-01 17:17:57.004645
7  -----

```

这段程序的功能非常简单：我们初始化了一个购物车对象，然后给购物车对象进行加购操作，最后打印出基本信息。可以看到，在不使用 Dart 语法任何特性的情况下，这段代码与


Java、C++ 甚至 JavaScript 没有明显的语法差异。

在关于如何表达以及处理信息上，Dart 保持了既简单又简洁的风格。那接下来，**我们就先从表达信息入手，看看 Dart 是如何优化这段代码的。**

类抽象改造

我们先来看看 Item 类与 ShoppingCart 类的初始化部分。它们在构造函数中的初始化工作，仅仅是将 main 函数传入的参数进行属性赋值。

在其他编程语言中，在构造函数的函数体内，将初始化参数赋值给实例变量的方式非常常见。而在 Dart 里，我们可以利用语法糖以及初始化列表，来简化这样的赋值过程，从而直接省去构造函数的函数体：

 复制代码

```
1 class Item {
2   double price;
3   String name;
4   Item(this.name, this.price);
5 }
6
7 class ShoppingCart {
8   String name;
9   DateTime date;
10  String code;
11  List<Item> bookings;
12  price() {...}
13  // 删掉了构造函数函数体
14  ShoppingCart(this.name, this.code) : date = DateTime.now();
15  ...
16 }
```


这一下就省去了 7 行代码！通过这次改造，我们有两个新的发现：

首先，Item 类与 ShoppingCart 类中都有一个 name 属性，在 Item 中表示商品名称，在 ShoppingCart 中则表示用户名；

然后，Item 类中有一个 price 属性，ShoppingCart 中有一个 price 方法，它们都表示当前的价格。

考虑到 name 属性与 price 属性（方法）的名称与类型完全一致，在信息表达上的作用也几乎一致，因此我可以在这两个类的基础上，再抽象出一个新的基类 Meta，用于存放 price 属性与 name 属性。

同时，考虑到在 ShoppingCart 类中，price 属性仅用做计算购物车中商品的价格（而不是像 Item 类那样用于数据存取），因此在继承了 Meta 类后，我改写了 ShoppingCart 类中 price 属性的 get 方法：


 复制代码

```
1 class Meta {
2     double price;
3     String name;
4     Meta(this.name, this.price);
5 }
6 class Item extends Meta{
7     Item(name, price) : super(name, price);
8 }
9
10 class ShoppingCart extends Meta{
11     DateTime date;
12     String code;
13     List<Item> bookings;
14
15     double get price {...}
16     ShoppingCart(name, this.code) : date = DateTime.now(),super(name,0);
17     getInfo() {...}
18 }
```

通过这次类抽象改造，程序中各个类的依赖关系变得更加清晰了。不过，目前这段程序中还有两个冗长的方法显得格格不入，即 ShoppingCart 类中计算价格的 price 属性 get 方法，以及提供购物车基本信息的 getInfo 方法。接下来，我们分别来改造这两个方法。

方法改造

我们先看看 price 属性的 get 方法：

 复制代码


```
1 double get price {
2     double sum = 0.0;
3     for(var i in bookings) {
4         sum += i.price;
```

```
5    }  
6    return sum;  
7 }
```

在这个方法里，我采用了其他语言常见的求和算法，依次遍历 bookings 列表中的 Item 对象，累积相加求和。

而在 Dart 中，这样的求和运算我们只需重载 Item 类的 “+” 运算符，并通过对列表对象进行归纳合并操作即可实现（你可以想象成，把购物车中的所有商品都合并成了一个商品套餐对象）。

另外，由于函数体只有一行，所以我们可以使用 Dart 的箭头函数来进一步简化实现函数：

 复制代码

```
1 class Item extends Meta{  
2     ...  
3     // 重载了 + 运算符，合并商品为套餐商品  
4     Item operator+(Item item) => Item(name + item.name, price + item.price);  
5 }  
6  
7 class ShoppingCart extends Meta{  
8     ...  
9     // 把迭代求和改写为归纳合并  
10    double get price => bookings.reduce((value, element) => value + element).price;  
11    ...  
12    getInfo() {...}  
13 }
```

可以看到，这段代码又简洁了很多！接下来，我们再看看 getInfo 方法如何优化。

在 getInfo 方法中，我们将 ShoppingCart 类的基本信息通过字符串拼接的方式，进行格式化组合，这在其他编程语言中非常常见。而在 Dart 中，我们可以通过对字符串插入变量或表达式，并使用多行字符串声明的方式，来完全抛弃不优雅的字符串拼接，实现字符串格式化组合。

 复制代码

```
1 getInfo () => '''
```


```
2 购物车信息：
3 -----
4   用户名: $name
5   优惠码: $code
6   总价: $price
7   Date: $date
8 -----
9 ''';
```

在去掉了多余的字符串转义和拼接代码后，getInfo 方法看着就清晰多了。

在优化完了 ShoppingCart 类与 Item 类的内部实现后，我们再来看看 main 函数，从调用方的角度去分析程序还能在哪些方面做优化。

对象初始化方式的优化

在 main 函数中，我们使用

 复制代码

```
1 ShoppingCart sc = ShoppingCart('张三', '123456');
```


初始化了一个使用 ‘123456’ 优惠码、名为 ‘张三’ 的用户所使用的购物车对象。而这段初始化方法的调用，我们可以从两个方面优化：

首先，在对 ShoppingCart 的构造函数进行了大量简写后，我们希望能够提供给调用者更明确的初始化方法调用方式，让调用者以“参数名: 参数键值对”的方式指定调用参数，让调用者明确传递的初始化参数的意义。在 Dart 中，这样的需求，我们在声明函数时，可以通过给参数增加{}实现。

其次，对一个购物车对象来说，一定会有一个有用户名，但不一定有优惠码的用户。因此，对于购物车对象的初始化，我们还需要提供一个不含优惠码的初始化方法，并且需要确定多个初始化方法与父类的初始化方法之间的正确调用顺序。


按照这样的思路，我们开始对 ShoppingCart 进行改造。

需要注意的是，由于优惠码可以为空，我们还需要对 `getInfo` 方法进行兼容处理。在这里，我用到了 `a??b` 运算符，这个运算符能够大量简化在其他语言中三元表达式 `(a != null)? a : b` 的写法：

 复制代码

```
1 class ShoppingCart extends Meta{
2     ...
3     // 默认初始化方法，转发到 withCode 里
4     ShoppingCart({name}) : this.withCode(name:name, code:null);
5     //withCode 初始化方法，使用语法糖和初始化列表进行赋值，并调用父类初始化方法
6     ShoppingCart.withCode({name, this.code}) : date = DateTime.now(), super(name,0);
7
8     //?? 运算符表示为 code 不为 null，则用原值，否则使用默认值 " 没有 "
9     getInfo () => '''
10 购物车信息：
11 -----
12 用户名: $name
13 优惠码: ${code??" 没有 "}
14 总价: $price
15 Date: $date
16 -----
17 ''';
18 }
19
20 void main() {
21     ShoppingCart sc = ShoppingCart.withCode(name:'张三', code:'123456');
22     sc.bookings = [Item('苹果',10.0), Item('鸭梨',20.0)];
23     print(sc.getInfo());
24
25     ShoppingCart sc2 = ShoppingCart(name:'李四');
26     sc2.bookings = [Item('香蕉',15.0), Item('西瓜',40.0)];
27     print(sc2.getInfo());
28 }
```

运行这段程序，张三和李四的购物车信息都会被打印到命令行中：

 复制代码

```
1 购物车信息：
2 -----
3 用户名: 张三
4 优惠码: 123456
5 总价: 30.0
6 Date: 2019-06-01 19:59:30.443817
7 -----
8
```




```
9 购物车信息：
10 -----
11  用户名：李四
12  优惠码：没有
13  总价：55.0
14  Date: 2019-06-01 19:59:30.451747
15 -----
```

关于购物车信息的打印，我们是通过在 main 函数中获取到购物车对象的信息后，使用全局的 print 函数打印的，我们希望把打印信息的行为封装到 ShoppingCart 类中。而对于打印信息的行为而言，这是一个非常通用的功能，不止 ShoppingCart 类需要，Item 对象也可能需要。

因此，我们需要把打印信息的能力单独封装成一个单独的类 PrintHelper。但，ShoppingCart 类本身已经继承自 Meta 类，考虑到 Dart 并不支持多继承，我们怎样才能实现 PrintHelper 类的复用呢？


这就用到了我在上一篇文章中提到的“混入”（Mixin），相信你还记得只要在使用时加上 with 关键字即可。

我们来试着增加 PrintHelper 类，并调整 ShoppingCart 的声明：

 复制代码

```
1 abstract class PrintHelper {
2   printInfo() => print(getInfo());
3   getInfo();
4 }
5
6 class ShoppingCart extends Meta with PrintHelper{
7   ...
8 }
```

经过 Mixin 的改造，我们终于把所有购物车的行为都封装到 ShoppingCart 内部了。而对于调用方而言，还可以使用级联运算符“..”，在同一个对象上连续调用多个函数以及访问成员变量。使用级联操作符可以避免创建临时变量，让代码看起来更流畅：

 复制代码

```

1 void main() {
2   ShoppingCart.withCode(name: '张三', code: '123456')
3   ..bookings = [Item('苹果', 10.0), Item('鸭梨', 20.0)]
4   ..printInfo();
5
6   ShoppingCart(name: '李四')
7   ..bookings = [Item('香蕉', 15.0), Item('西瓜', 40.0)]
8   ..printInfo();
9 }

```


很好！通过 Dart 独有的语法特性，我们终于把这段购物车代码改造成了简洁、直接而又强大的 Dart 风格程序。

总结

这就是今天分享的全部内容了。在今天，我们以一个与 Java、C++ 甚至 JavaScript 没有明显语法差异的购物车雏形为起步，逐步将它改造成了一个符合 Dart 设计思想的程序。

首先，我们使用构造函数语法糖及初始化列表，简化了成员变量的赋值过程。然后，我们重载了 “+” 运算符，并采用归纳合并的方式实现了价格计算，并且使用多行字符串和内嵌表达式的方式，省去了无谓的字符串拼接。最后，我们重新梳理了类之间的继承关系，通过 mixin、多构造函数，可选命名参数等手段，优化了对象初始化调用方式。

下面是今天购物车综合案例的完整代码，希望你在 IDE 中多多练习，体会这次的改造过程，从而对 Dart 那些使代码变得更简洁、直接而强大的关键语法特性产生更深刻的印象。同时，改造前后的代码，你也可以在 GitHub 的 [Dart Sample](#) 中找到：

 复制代码

```

1 class Meta {
2   double price;
3   String name;
4   // 成员变量初始化语法糖
5   Meta(this.name, this.price);
6 }
7
8 class Item extends Meta{
9   Item(name, price) : super(name, price);
10  // 重载 + 运算符，将商品对象合并为套餐商品
11  Item operator+(Item item) => Item(name + item.name, price + item.price);
12 }
13

```

```

14 abstract class PrintHelper {
15     printInfo() => print(getInfo());
16     getInfo();
17 }
18
19 //with 表示以非继承的方式复用了另一个类的成员变量及函数
20 class ShoppingCart extends Meta with PrintHelper{
21     DateTime date;
22     String code;
23     List<Item> bookings;
24     // 以归纳合并方式求和
25     double get price => bookings.reduce((value, element) => value + element).price;
26     // 默认初始化函数，转发至 withCode 函数
27     ShoppingCart({name}) : this.withCode(name:name, code:null);
28     //withCode 初始化方法，使用语法糖和初始化列表进行赋值，并调用父类初始化方法
29     ShoppingCart.withCode({name, this.code}) : date = DateTime.now(), super(name,0);
30
31     //?? 运算符表示为 code 不为 null，则用原值，否则使用默认值 " 没有 "
32     @override
33     getInfo() => '''
34 购物车信息：
35 -----
36 用户名: $name
37 优惠码: ${code??" 没有 "}
38 总价: $price
39 Date: $date
40 -----
41 ''';
42 }
43
44 void main() {
45     ShoppingCart.withCode(name:'张三', code:'123456')
46     ..bookings = [Item('苹果',10.0), Item('鸭梨',20.0)]
47     ..printInfo();
48
49     ShoppingCart(name:'李四')
50     ..bookings = [Item('香蕉',15.0), Item('西瓜',40.0)]
51     ..printInfo();
52 }

```

思考题

请你扩展购物车程序的实现，使得我们的购物车可以支持：

1. 商品数量属性；

2. 购物车信息增加商品列表信息（包括商品名称，数量及单价）输出，实现小票的基本功能。

欢迎你在评论区给我留言分享你的观点，我会在下一篇文章中等待你！感谢你的收听，也欢迎你把这篇文章分享给更多的朋友一起阅读。



Flutter 核心技术与实战

来自 Google 的高性能跨平台开发框架

陈航

美团点评高级技术专家



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 07 | 函数、类与运算符：Dart是如何处理信息的？

下一篇 09 | Widget，构建Flutter界面的基石

精选留言 (12)

写留言



bytecode

2019-07-16

```
double get price() {  
    double sum = 0.0;  
    for(var i in bookings) {  
        sum += i.price;  
    }  
}
```

}...

展开 ▾

作者回复: 感谢提醒



👍 7



Ω

2019-07-16

Item operator+(Item item) => l...
double get price => bookings...

这个改造+号的代码还是无法看懂

作者回复: operator+ : 把两个Item对象合并为一个。NewItem对象的name为这两个对象的name拼接而成, price为他们的price相加而成。

get price : 对列表数据采用累加的方式进行求和。这里用到了reduce方法。reduce是函数迭代方法, 需要传递一个二元函数进行列表的合并工作。list[0...n].reduce(f)等于:

a0 = list[0]

a1 = f(a0,list[1])

a2 = f(a1,list[2])

an = f(an-1,list[n])

在这里我们的f是求和函数f(x,y)=x+y, 可以理解成an=list[0]+list[1]+list[n-1]+list[n]



💬 1

👍 6



Paradise

2019-07-16

这种结合代码的方式很友好, 可以一边听一边上手, 更容易理解... 😊



👍 3



徐西文

2019-07-19

// 思考题

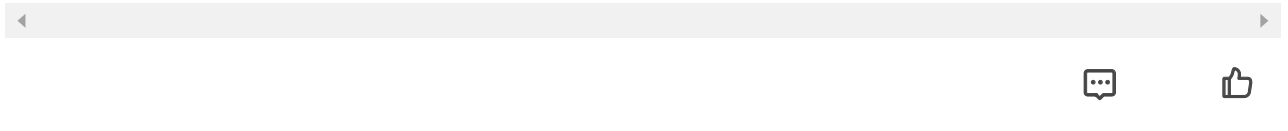
class Meta {

```
String name;  
double price;  
double count;...
```

展开 ▾

作者回复: 这里有个问题：在合并成新的套餐对象时，数量应该变成1。比如三斤苹果和四斤梨合并，应该是一份苹果梨套餐；而不是7份苹果梨，否则价格就算错啦。

另外你这段代码好像没法编译通过吧



江宁彭于晏

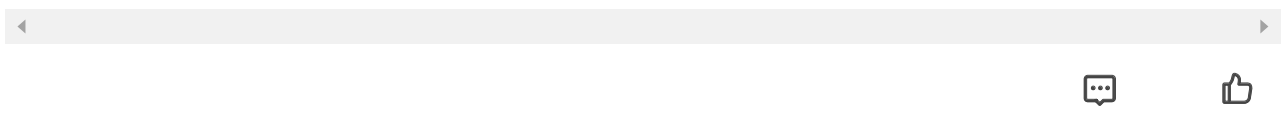
2019-07-19

```
class Meta {  
    double price;  
    String name;  
    // 成员变量初始化语法糖  
    Meta(this.name, this.price);...
```

展开 ▾

作者回复: 赞👍

不过数量作为Item的属性会更好一点。



Geek_388dc2

2019-07-18

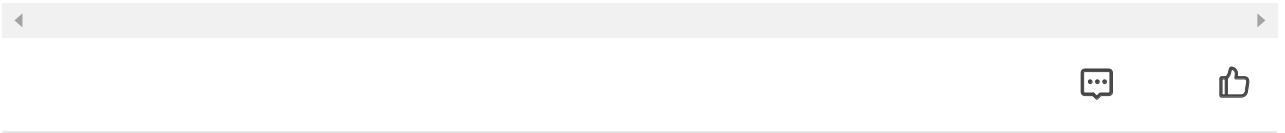
我想用泛型来取后台 data 中的数据。我知道限制继承了某个类的是可以的。我想传入的类限制为实现了fromJsonMap这个工厂方法的可以吗。

展开 ▾

作者回复: 可以使用类型约束，让T必须继承自某个类

```
class Foo<T extends SomeBaseClass> {}
```

```
T foo<T extends SomeBaseClass>(T ts) {}
```



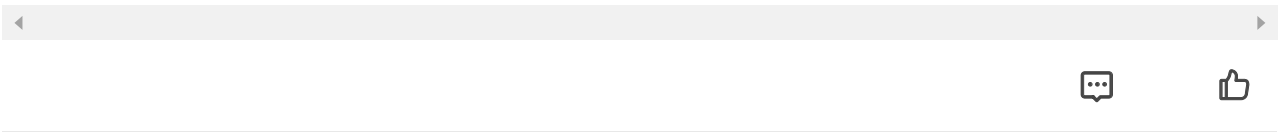
allen

2019-07-17

ShoppingCart.withCode这个withCode方法是自己定义的么，还是系统自带的，我没见到这个方法，还是说直接能类名加方法名定义的

展开 ▾

作者回复: 上一节讲过这个知识点哈，命名构造函数



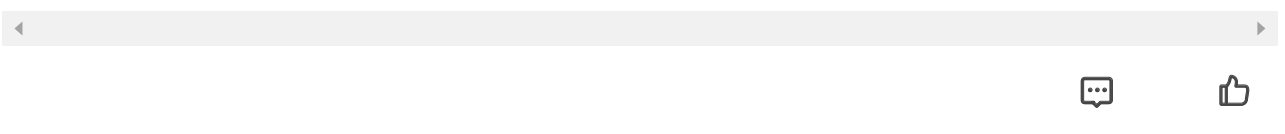
大米不剩了

2019-07-17

ShoppingCart.withCode({name, this.code}):date = DateTime.now(),super(name,0);
疑惑了好久，后来发现如果在构造函数的初始化列表中使用super()，需要把它放到最后。
参考<http://dart.goodev.org/guides/language/effective-dart/usage#do-place-the-super-call-last-in-a-constructor-initialization-list>

展开 ▾

作者回复: 赞



Geek_rvg

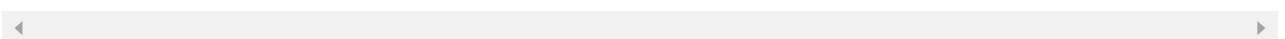
2019-07-17

作业：

1. 抽象类PrintHelper添加printGoodsInfo方法定义，然后在ShoppingCart类中实现getGoodsInfo方法（用户获取小票信息）；
2. 由于需要获取各个商品的数量，Item类添加count属性（用于表示每个商品数量），ShoppingCart类的bookings变更为私有属性，同时添加add方法用于向Shopping类中...

展开 ▾

作者回复: 你的price没算对哦，总price=求和（单价*count）





Geek_759ee2c98fce

2019-07-17

我感觉这个例子中 用 reduce+ item 重载运算符似乎不太好，毕竟item相加是说不出来业务含义的。

price 求和我觉得用 fold 比较合理，这里也是dart和一般其他语言reduce 的一个差异点。

作者回复: 业务含义理解成是把Item商品打包成一个大商品套餐就可以了。



小师弟

2019-07-16

思考题：

```
// Meta类加入变量count
```

```
class Meta {
```

```
    double price;...
```

展开 ∨

作者回复: 这里有个问题：在合并成新的套餐对象时，数量应该变成1。比如三斤苹果和四斤梨合并，应该是一份苹果梨套餐；而不是7份苹果梨



文心雕龙

2019-07-16

思考题：

```
// 定义商品 Item 类
```

```
class Item extends Meta {
```

```
    double quantity;
```

```
    Item({name, price, this.quantity = 1.0}) : super(name, price);...
```

展开 ∨

作者回复: 这里有个问题：在合并成套餐时，数量应该变成1，而不能直接相加。比如三斤苹果和五盒巧克力，合并完应该是一份苹果巧克力，而不是八份苹果巧克力。

