

# 设计模式 --- 装饰模型 (转)

作者: [ironMan](#)

原文链接: <https://ld246.com/article/1605714211495>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)



## 前言

Step y Step

前一篇已经讲解了代理模式了，今天要讲解的就是**装饰模式**啦~

在看到**FilterInputStream**和**FilterOutputStream**时看到了之前常听见的**装饰模式**(对IO一定了解的同可能都会知道那么一句话：在IO用得最多的就是装饰模式了)!

其实无论是代理模式还是装饰模式。本质上我认为就是**对原有对象增强的方式**~

那么接下来就开始吧，如果文章有错误的地方请大家多多包涵，不吝在评论区指正哦~

声明：本文使用JDK1.8

## 一、对象增强的常用方式

很多时候我们可能对**Java提供给我们对象不满意**，不能满足我们的功能。此时我们就想对Java原对进行增强，能够实现我们想要的功能就好~

一般来说，实现对象增强有三种方式：

- **继承**
  - 继承父类，子类扩展
- **装饰器模式**
  - 使用“包装”的方式来增强对象
- **代理模式**

# 1.1 继承

最简单的方式就是继承父类，子类扩展来达到目的。虽然简单，但是这种方式的**缺陷非常大**：

- 一、如果 **父类是带有数据、信息、属性的话，那么子类无法增强。**
- 二、子类实现了之后 **需求无法变更**，增强的内容是**固定**的。

## 1.1.1 第一点

**第一点**就拿以前在学JDBC的时候来说：

- 当时想要自己写一个简易的JDBC连接池，连接池由 `List<Connection>`来管理。显然我们的对象Connection，当写到`close()`方法的时候卡住了。
- 因为我们想要的功能是：调用 `close()`是让我们的Connection返回到“连接池”（集合）中，而不是闭掉。
- 此时我们 **不能使用继承父类的方式来实现增强**。因为Connection对象是由数据库厂商来实现的，在得到Connection对象的时候绑定了各种信息(数据库的username、password、具体的数据库是啥等)。我们**子类继承Connection是无法得到对应的数据的！**就更别说调用`close()`方法了。

## 1.1.2 第二点

**第二点**我也举个例子：

现在我设计一个电话类：

```
public class Phone {
    // 可以打电话
    public void call() {
        System.out.println("打电话给周围的人关注公众号Java3y");
    }
}
```

此时，我想**打电话之前能听彩铃**，于是我继承Phone类，实现我想要的功能。

```
public class MusicPhone extends Phone {

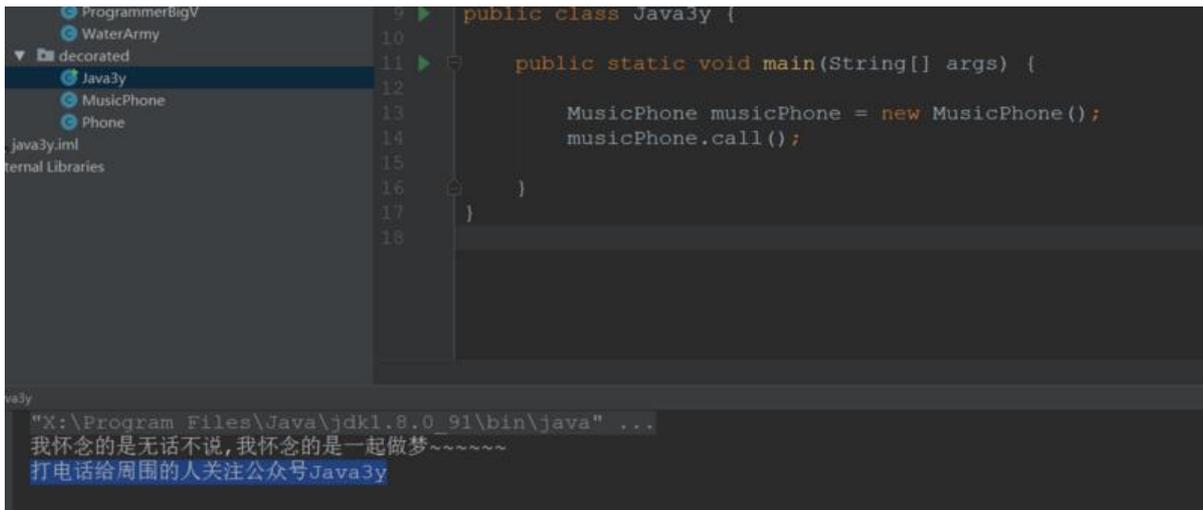
    // 听彩铃
    public void listenMusic() {
        System.out.println("我怀念的是无话不说,我怀念的是一起做梦~~~~~");
    }

    @Override
    public void call() {

        // 在打电话之前听彩铃
        listenMusic();

        super.call();
    }
}
```

我们的功能就做好了：



```
public class Java3y {
    10
    11     public static void main(String[] args) {
    12
    13         MusicPhone musicPhone = new MusicPhone();
    14         musicPhone.call();
    15
    16     }
    17
    18
}
```

"X:\Program Files\Java\jdk1.8.0\_91\bin\java" ...  
我怀念的是无话不说,我怀念的是一起做梦~~~~~  
打电话给周围的人关注公众号Java3y

• 此时，我又突然想实现多一个需求了，我想要听完电话之后告诉我一下当前的时间是多少。没事，我们又继承来增强一下：

// 这里继承的是MusicPhone类

```
public class GiveCurrentTimePhone extends MusicPhone {
```

```
    // 给出当前的时间
```

```
    public void currentTime() {
```

```
        System.out.println("当前的时间是: " + System.currentTimeMillis());
```

```
    }
```

```
    @Override
```

```
    public void call() {
```

```
        super.call();
```

```
        // 打完电话提示现在的时间是多少啦
```

```
        currentTime();
```

```
    }
```

```
}
```

所以我们还是可以完成任务滴：



```
public static void main(String[] args) {
    12
    13     GiveCurrentTimePhone currentTimePhone = new GiveCurrentTimePhone();
    14     currentTimePhone.call();
    15
    16 }
    17
    18
}
```

Run java3y  
"X:\Program Files\Java\jdk1.8.0\_91\bin\java" ...  
我怀念的是无话不说,我怀念的是一起做梦~~~~~  
打电话给周围的人关注公众号Java3y  
当前的时间是: 1525699872270

可是我需求现在又想变了：

• 我不想听彩铃了，只想听完电话通知一下时间就好了.....(可是我们的通知时间电话类是继承在听彩的电话类基础之上的),,,

- 我又有可能：我想在听电话之前报告一下时间，听完电话听音乐！ ...
- 如果 **需求变动很大的情况下**，而我们又用**继承的方式来实现**这样会导致一种现象：**类爆炸**(类数量增)！并且**继承的层次可能会比较多**~

所以，我们可以看到子类继承父类这种方式来扩展是**十分局限的，不灵活的**~

因此我们就有了**装饰模式**！

## 1.2装饰模式

首先我们来看看装饰模式是怎么用的吧。

### 1.2.1前提代码

电话接口：

```
// 一个好的设计是抽取成接口或者抽象类的
public interface Phone {

    // 可以打电话
    void call();
}
```

具体的实现：

```
public class IphoneX implements Phone {

    @Override
    public void call() {
        System.out.println("打电话给周围的人关注公众号Java3y");
    }
}
```

### 1.2.2包装模式实现

上面我们已经**拥有了一个接口还有一个默认实现**。包装模式是这样干的：

首先我们弄一个**装饰器**，它实现了接口，以**组合**的方式接收我们的**默认实现类**。

```
// 装饰器，实现接口
public abstract class PhoneDecorate implements Phone {

    // 以组合的方式来获取默认实现类
    private Phone phone;
    public PhoneDecorate(Phone phone) {
        this.phone = phone;
    }

    @Override
    public void call() {
        phone.call();
    }
}
```

```
}  
}
```

有了装饰器以后，我们的扩展都可以**以装饰器为基础进行扩展**，继承装饰器来扩展就好了！

我们想要在**打电话之前听音乐**：

// 继承着装饰器来扩展

```
public class MusicPhone extends PhoneDecorate {
```

```
    public MusicPhone(Phone phone) {  
        super(phone);  
    }
```

// 定义想要扩展的功能

```
    public void listenMusic() {
```

```
        System.out.println("继续跑 带着赤子的骄傲，生命的闪耀不坚持到底怎能看到，与其苟延残喘  
        如纵情燃烧");
```

```
    }
```

// 重写打电话的方法

@Override

```
    public void call() {
```

// 在打电话之前听音乐

```
        listenMusic();
```

```
        super.call();
```

```
    }
```

```
}
```



The screenshot shows an IDE with a project named 'decorated'. The package structure includes 'GiveCurrentTimePhone', 'IphoneX', 'Java3y', 'MusicPhone', 'Phone', and 'PhoneDecorate'. The 'main' method in 'Java3y' is shown, which creates an 'IphoneX' object, wraps it in a 'MusicPhone' object, and calls 'call()' on it. The output window shows the execution of 'main()' with the following text: '继续跑 带着赤子的骄傲，生命的闪耀不坚持到底怎能看到，与其苟延残喘 打电话给周围的人关注公众号Java3y'.

现在我也想在**打完电话后通知当前的时间**，于是我们也**继承装饰类来扩展**：

// 这里继承的是MusicPhone装饰器类

```
public class GiveCurrentTimePhone extends PhoneDecorate {
```

```
    public GiveCurrentTimePhone(Phone phone) {  
        super(phone);  
    }
```

```

// 自定义要实现的功能：给出当前的时间
public void currentTime() {
    System.out.println("当前的时间是： " + System.currentTimeMillis());
}

// 重写要增强的方法
@Override
public void call() {
    super.call();
    // 打完电话后通知一下当前时间
    currentTime();
}
}
}

```

可以完成任务：

```

// 创建出最原始的实现类
Phone phone = new IphoneX();

// 装饰成打电话之前可以听音乐的功能
phone = new MusicPhone(phone);

// 装饰成打电话之后可以通知当前时间的功能
phone = new GiveCurrentTimePhone(phone);

phone.call();
}
}

```

Java3y  
 "X:\Program Files\Java\jdk1.8.0\_91\bin\java" ...  
 继续跑，带着赤子的骄傲，生命的闪耀不坚持到底怎能看到，与其苟延残喘不如纵情燃烧  
 打电话给周围的人关注公众号Java3y  
 当前的时间是：1525742740058

就目前这样看起来，比我直接继承父类要麻烦，而功能效果是一样的...我们继续往下看~~

此时，我不想在打电话之前听到彩铃了，很简单：我们**不装饰**它就好了！

```

// 创建出最原始的实现类
Phone phone = new IphoneX();

// 装饰成打电话之前可以听音乐的功能
//phone = new MusicPhone(phone);

// 装饰成打电话之后可以通知当前时间的功能
phone = new GiveCurrentTimePhone(phone);

phone.call();
}
}

```

Java3y main()  
 "X:\Program Files\Java\jdk1.8.0\_91\bin\java" ...  
 打电话给周围的人关注公众号Java3y  
 当前的时间是：1525744185063

此时，我想在打电话前报告一下时间，在打完电话之后听彩铃。

- 注意：虽然说要改动类中的代码，**但是**这种改动是合理的。因为我定义出的GiveCurrentTimePhone类和MusicPhone类本身从语义上就**没有规定**扩展功能的执行顺序
- 而继承不一样：先继承Phone->实现MusicPhone->再继承MusicPhone实现GiveCurrentTimePhone。这是**固定的**，从继承的逻辑上已经**写死了**具体的代码，是**难以改变**的。

```
// 这里继承的是MusicPhone装饰器类
public class GiveCurrentTimePhone extends PhoneDecorate {

    public GiveCurrentTimePhone(Phone phone) {
        super(phone);
    }

    // 自定义想要实现的功能：给出当前的时间
    public void currentTime() {
        System.out.println("当前的时间是：" + System.currentTimeMillis());
    }

    // 重写要增强的方法
    @Override
    public void call() {

        currentTime();|
        super.call();
        // 打完电话后通知一下当前时间
        //currentTime();
    }
}
```

```
// 继承着装饰器来扩展
public class MusicPhone extends PhoneDecorate {

    public MusicPhone(Phone phone) {
        super(phone);
    }

    // 定义想要扩展的功能
    public void listenMusic() {

        System.out.println("继续跑 带着赤子的骄傲，生命的闪耀不坚持到底的
        燃烧");
    }

    // 重写打电话的方法
    @Override
    public void call() {

        // 在打电话之前听音乐
        //listenMusic();
        super.call();

        listenMusic();
    }
}
```

所以我们还是可以很简单地完成功能：

```
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

## 二、装饰模式讲解

可能的同学在看完上面的代码之后，还是迷迷糊糊地不知道装饰模式是怎么实现“装饰”的。下面就再来解析一下：

- 第一步：我们有一个Phone接口，该接口定义了Phone的功能
- 第二步：我们有一个最简单的实现类iPhoneX
- 第三步：写一个装饰器抽象类PhoneDecorate，以 **组合**(构造函数传递)的方式接收我们最简单的现类iPhoneX。其实装饰器抽象类的作用就是**代理**(核心的功能还是由最简单的实现类iPhoneX来做，不过在**扩展**的时候可以**添加一些没有的功能**而已)。
- 第四步：想要扩展什么功能，就继承PhoneDecorate装饰器抽象类，将想要增强的对象(最简单的现类iPhoneX或者已经被增强过的对象)传进去，完成我们的扩展！

再来看看下面的图，就懂了！

```
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

往往我们的代码可以省略起来，成了这个样子(是不是和IO的非常像！)

// 先增强听音乐的功能，再增强通知时间的功能

```
Phone phone = new GiveCurrentTimePhone(new MusicPhone(new IphoneX()));
```

结果是一样的：



```
23 // 先增强听音乐的功能，再增强通知时间的功能
24 Phone phone = new GiveCurrentTimePhone(new MusicPhone(new IphoneX()));
25
26 phone.call();
27 }
28 }
29
```

Java3y / main0

当前的时间是: 1525746722856  
打电话给周围的人关注公众号Java3y  
继续跑 带着赤子的骄傲，生命的闪耀不坚持到底怎能看到，与其苟延残喘不如纵情燃烧

## 2.1 装饰模式的优缺点

优点：

- 装饰类和被装饰类是可以 **独立的**，低耦合的。互相都不用知道对方的存在
- 装饰模式是继承的一种 **替代方案**，**无论包装多少层，返回的对象都是is-a的关系**(上面的例子：包完还是Phone类型)。
- 实现动态扩展，只要 **继承了装饰器就可以动态扩展想要的功能了**。

缺点：

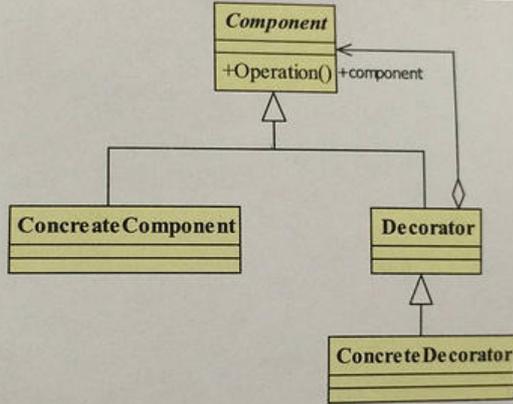
- 多层装饰是比较复杂的，提高了系统的复杂度。不利于我们调试~

## 三、总结

最后来补充一下包装模式和代理模式的类图：

模式名称	装饰模式	类型	结构类
	Decorator Pattern		

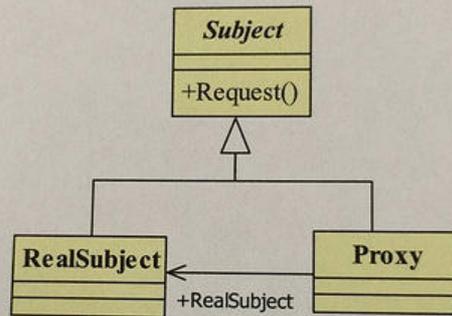
21



**描述：**动态地给一个对象添加一些额外的职责。就增加功能来说，它相比生成子类更为灵活。

模式名称	代理模式	类型	结构类
	Proxy Pattern		

19



**描述：**为其他对象提供一种代理以控制对这个对象的访问。

对象增强的三种方式：

- 继承
- 包装模式
- 代理模式

那么只要遇到Java提供给我们的API不够用，我们增强一下就行了。在写代码时，某个类被写死了，不够用，增强一下就可以了！

原文地址：<https://segmentfault.com/a/1190000014771830>