# 针对PHP中大量的if判断的重构

## 案例一

一个if嵌套

<?php

$table = 'test\_order';

if ($table === 'test\_order' OR

$table === 'test\_client' OR

$table === 'test\_area'

) {

echo $table;

} else {

die('err');

}

用数组解决解决

<?php

$table = 'test\_order';

$arr = array(

'test\_order' => '',

'test\_area' => '',

'test\_client' => '',);

if (isset($arr[$table])) {

echo $table;

} else {

die('err');

}

很明显用数组代码更少，更重要的是，数据可以作为值来传递，而在php中，代码是不能作为数据结构来传递的。当然，有时候用数组会损失一点点效率，但不是系统瓶颈的话，是不值得去优化的——过早优化之所以不好，是因为很多时候优化 约等于 “写死”， 也就失去了灵活性。

## 案例二

大段的要死人的if语句

<?php

$int\_pattern = '3';

if ($inkai\_pattern == "1") {

echo "";

} elseif ($inkai\_pattern == "2") {

echo "";

} elseif ($inkai\_pattern == "3") {

echo "";

} elseif ($inkai\_pattern == "4") {

echo "";

} elseif ($inkai\_pattern == "5") {

echo "";

} elseif ($inkai\_pattern == "6") {

echo "";

}

用switch来解决问题

<?php

$int\_pattern = '3';

switch ($\_pattern)

{

case 1:

echo "";

break ;

case 2:

echo "";

break ;

case 3:

echo "";

break;

case 4:

echo "";

break;

case 5:

echo "";

break;

case 6:

echo "";

break;

case 101:

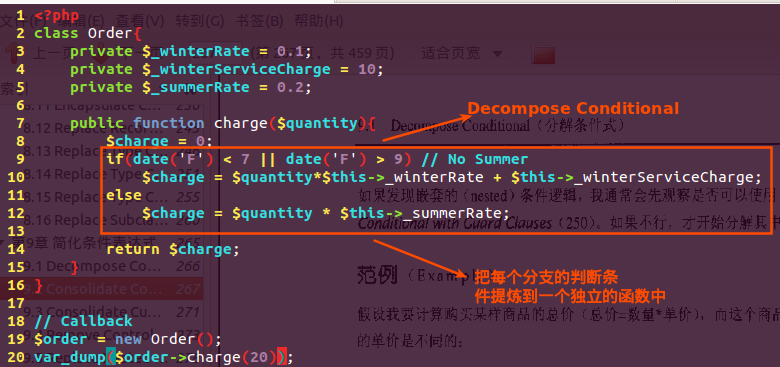
echo "";

break;

}

## 案例三

**状况：**你有一个复杂的条件（if-else if-else）语句，那么**从if、else if、else三个段落中分别提炼出函数。**



http://pic002.cnblogs.com/images/2012/152332/2012033017551723.png

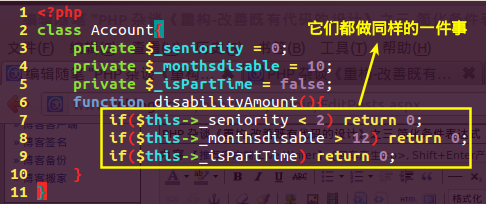


## 案例三

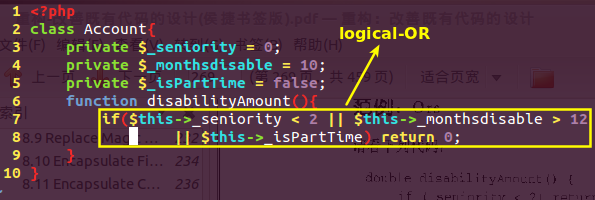
状况：你有一些条件测试，都得到相同的结果，那么将这些测试合并为一个条件式，并将这个条件提炼称为一个独立的函数。

动机： 1、合并后的条件代码会告诉你“实际上只有一次条件检查，只不过有数个并列条件需要检查而已“，——使检查的用意更清晰。

　　　 2、为Extract Method做好准备。——将检查条件提炼成一个独立函数，对于理清代码意义非常有用。它把描述“做什么”的语句换成了“为什么这样做”。



http://pic002.cnblogs.com/images/2012/152332/2012033017551723.png



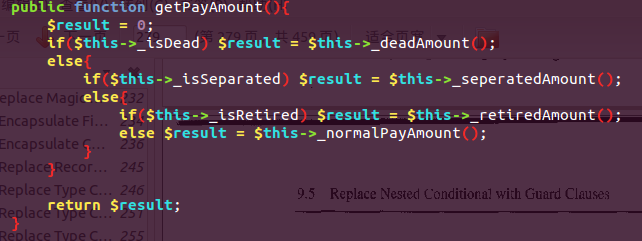
http://pic002.cnblogs.com/images/2012/152332/2012033017551723.png



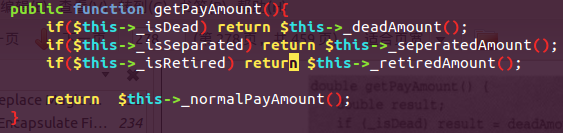
 条件语句的“合并理由”也同时指出了“不要合并”的理由：如果你认为你的这些检查的确彼此独立，的确不应该被视为同一次检查，那么就不要使用本项重构。因为在这种情况下，你的代码已经清楚表达出自己的意义。

## 案例五

**状况：**函数中的条件逻辑使人很难看清正常的执行路径，那么**使用卫语句（Guard Clauses）表现所有特殊情况。**



http://pic002.cnblogs.com/images/2012/152332/2012033017551723.png



条件式的两种形式：

　　1、所有分支都属于正常行为：使用[if ... else..]

　　2、条件式极其罕见：应该单独检查该条件，并在该条件为真时，立刻从函数中返回。——这样的单独检查常常被称为”卫语句“

Replace Nested Conditional with Guard Clauses精髓：给某一分支以特别重视。

## 案例六

你手上有个表达式，它根据对象型别的不同而选择不同的行为，那么**将这个条件式的每个分支放进一个subclass内的覆写函数中，然后将原始函数声明为抽象函数。**



**此代码的坏味道：**

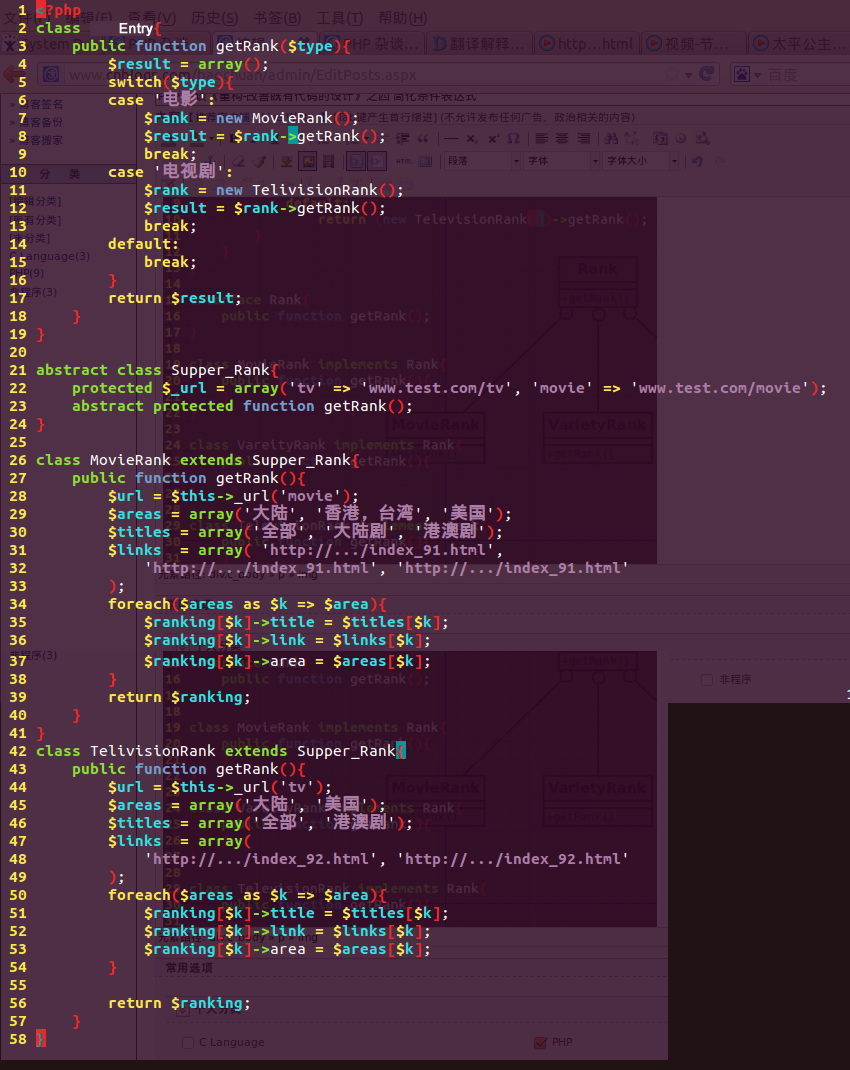
　　1、它太长，当视频有新类型的时候，它会变得更长。

　　2、它明显做了不止一件事。

　　3、它违反了单一权责原则，因为它有好几个修改它的理由。

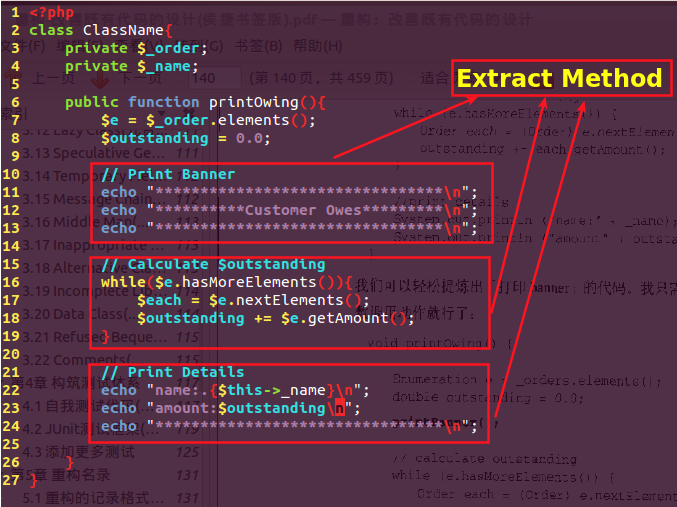
4、它违反了开放闭合原则，因为每当添加新类型时，必须修改它。不过最麻烦的可能是到处皆有类似结构(\_get类型名Rank())的函数。

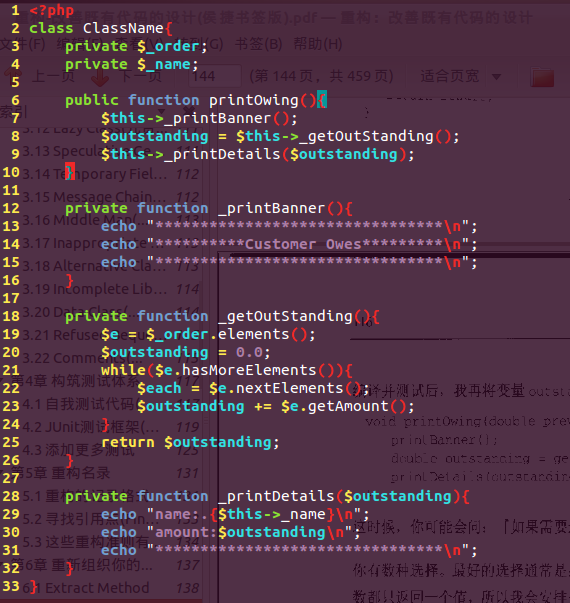
http://pic002.cnblogs.com/images/2012/152332/2012033017551723.png



# 提取函数

状况：我看见一个过长的函数或者需要一段注释才能让人理解用途的代码，那么将这段代码放进一个独立函数中，并让函数名称解释改函数的用途。





动机:

简短而有良好命名的函数：——finely grained

　　1、复用机会大。

　　2、函数读起来像读一系列comments。

　　3、函数覆写容易。

重点：函数长度关键在于函数名称和函数本体之间的语义距离。如果提炼动作可以强化代码的清晰度，那么就去做。

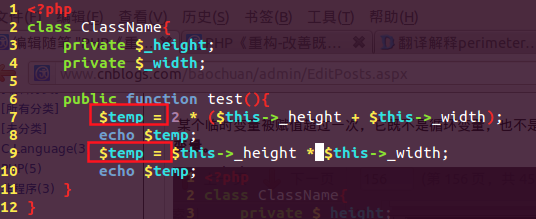
作法：

　　1、创建新函数，根据函数的意图命名——以它“做什么”命名，而不是以它“怎样做”命名。=》 即使Extract Function 非常简单，例如只是消息或函数调用，只要新Function能够以更好方式昭示代码意图，你也应该提炼它。但如果你想不出更有意义的名称，就别动它。

　　2、将Extract的代码从Source Function 中Move到New Function中。

# 分离临时变量

某个临时变量被赋值超过一次，它既不是循环变量，也不是集合变量。那么针对每次赋值，创造一个独立的，对应的临时变量。



http://pic002.cnblogs.com/images/2012/152332/2012033017551723.png



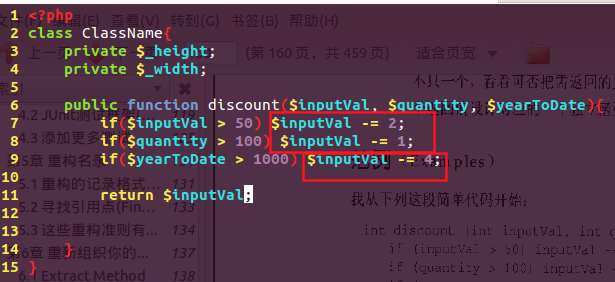
 动机：

　　1、如果临时变量承担多个责任，它就应该被替换为多个临时变量。每个变量只承担一个责任。

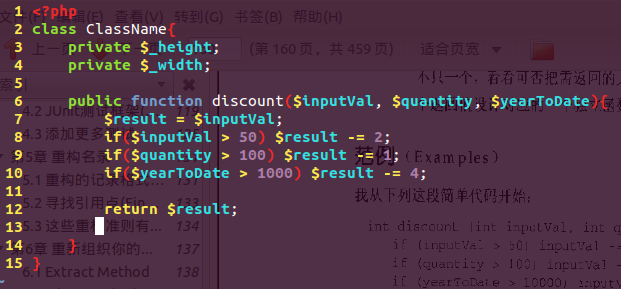
　　2、同一个临时变量承担两件不同的事情，会令review变得糊涂。

# 移除参数变量

如果你的代码对参数进行赋值，那么以一个临时变量取代该参数的位置。

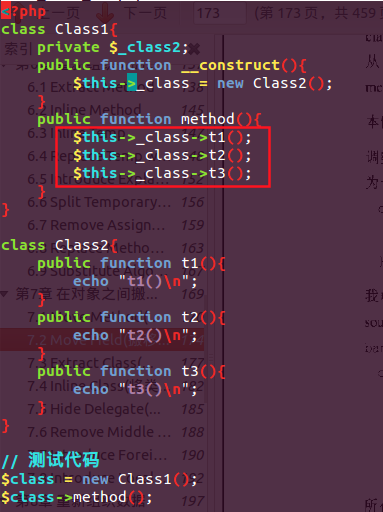


http://pic002.cnblogs.com/images/2012/152332/2012033017551723.png

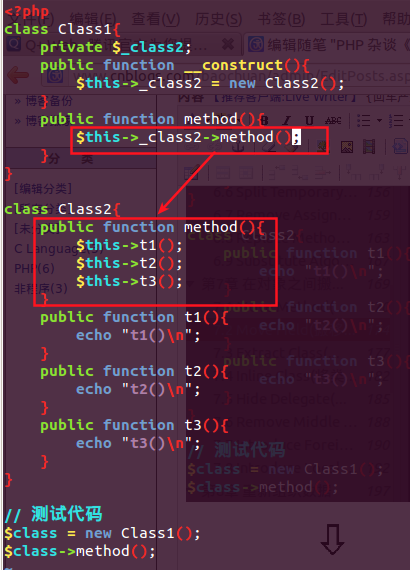


# 移动方法

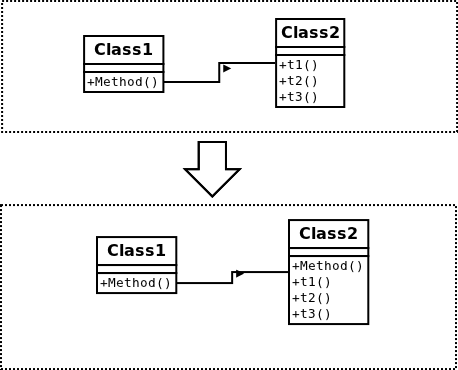
如果一个类中的方法与另一个类有很多的交流，那么我们就**在另一个类中建立一个有类似功能的新函数，将旧函数变成一个单纯的Delegating Method， 或是将旧函数移除。**



http://pic002.cnblogs.com/images/2012/152332/2012033017551723.png



 类图:



动机：

　　1、如果一个类与另一个类有高度耦合，我就会Move Method。——class更简单，更干净利落的实现系统交付的任务。

　　2、移动一些值域，就要检查是否使用另一个类的次数必使用所驻对象的次数还多。