**大话设计模式（PHP版）**

**UML 类图图示样例**

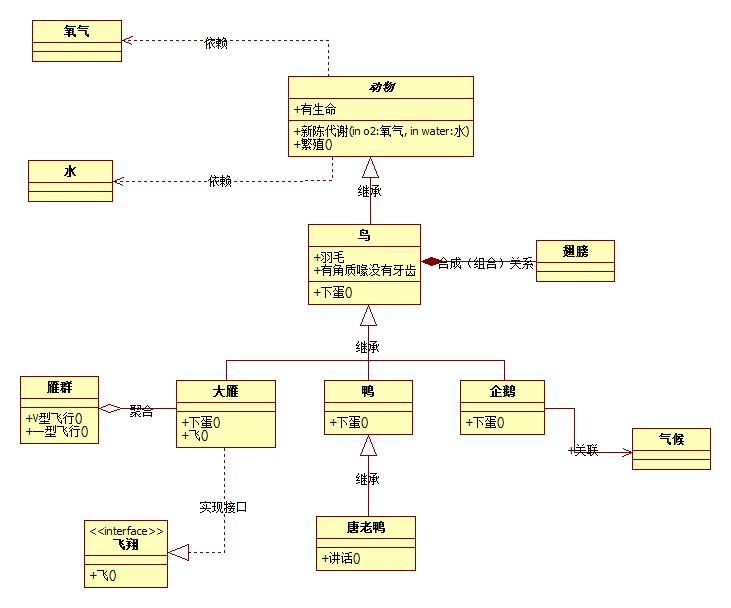
    单一职责原则（SRP）：就一个类而言，应该仅有一个引起他变化的原因；

    开放-封闭原则：软件实体（类，模块，函数等）应该可以扩展，但是不可修改；对于扩展是开放的，对于更改是封闭的；

    依赖倒转原则：抽象不应该依赖细节，细节应该依赖于抽象；高层模块不应该依赖低层模块，两个都应该依赖抽象。抽象不应该依赖细节，细节应该依赖抽象；

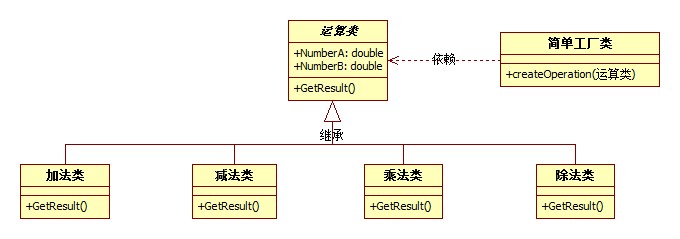
    里氏代换原则：子类型必须能够替换它们的父类型。一个软件实体如果使用的是一个父类的话，那么一定适用于其子类，而且它察觉不出父类对象和子类对象的区别。也就是说，在软件里面，把父类都替换成子类，程序的行为没有变化。只有当子类可以替换父类，软件单位的功能不受影响时，服了才能真正被复用，而子类也能够在父类的基础上增加新的行为。

    迪米特法则：强调类的松耦合，在类的结构设计上，没一个类都应当尽量降低成员的访问权限。类之间的耦合越弱，越有利于复用，一个处于弱耦合的类被修改，不会对有关系的类造成波及；



**1简单工厂模式**

    类结构图

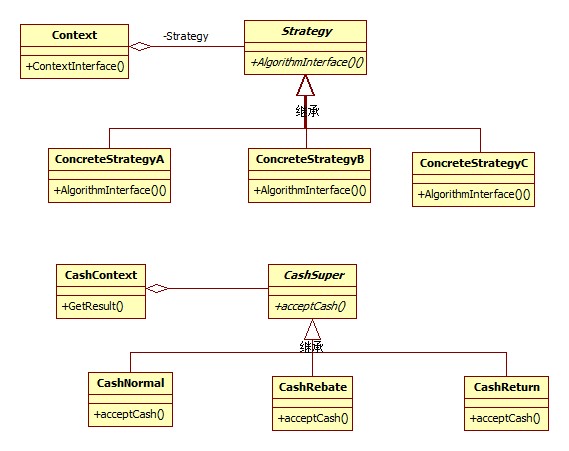


    PHP代码示例：

1. <?php
2. //简单工厂模式PHP代码
3. //抽象运算类
4. abstract class Operation
5. {
6. public $A;          //参数A
7. public $B;          //参数B
8. public $result;     //运算结果
9. //抽象方法获取结果方法
10. abstract protected  function getResult();
11. }
12. /\*\*
13. \* Class OperationAdd
14. \* 运算 和 类
15. \*/
16. class OperationAdd extends  Operation
17. {
18. //具体实现抽象类中的getResult方法
19. public function getResult()
20. {
21. $this->result = $this->A + $this->B;
22. return $this->result;
23. }
24. }
25. /\*\*
26. \* Class OperationSub
27. \* 运算 差 类
28. \*/
29. class OperationSub extends  Operation
30. {
31. //具体实现抽象类中的getResult方法
32. public function getResult()
33. {
34. $this->result = $this->A - $this->B;
35. return $this->result;
36. }
37. }
38. /\*\*
39. \* Class OperationMul
40. \* 运算 乘积 类
41. \*/
42. class OperationMul extends  Operation
43. {
44. //具体实现抽象类中的getResult方法
45. public function getResult()
46. {
47. $this->result = $this->A \* $this->B;
48. return $this->result;
49. }
50. }
51. /\*\*
52. \* Class OperationDiv
53. \* 运算 除 类
54. \*/
55. class OperationDiv extends  Operation
56. {
57. //具体实现抽象类中的getResult方法
58. public function getResult()
59. {
60. if($this->B == 0){
61. exit( '除数不能为0');
62. }
63. $this->result = $this->A / $this->B;
64. return $this->result;
65. }
66. }
67. /\*\*
68. \* Class OperationFactory
69. \* 简单工厂类
70. \*/
71. class OperationFactory
72. {
73. public static function creatOperation($operation)
74. {
75. $operationClass = null;
76. switch($operation)
77. {
78. case '+':
79. $operationClass = new OperationAdd();
80. break;
81. case '-':
82. $operationClass = new OperationSub();
83. break;
84. case '\*':
85. $operationClass = new OperationMul();
86. break;
87. case '/':
88. $operationClass = new OperationDiv();
89. break;
90. }
91. return $operationClass;
92. }
93. }
94. $operationClass = OperationFactory::creatOperation('/');    //通过简单工厂类获取类
95. $operationClass->A = 8;                     //初始化参数
96. $operationClass->B = 4;                     //初始化参数
97. $result = $operationClass->getResult();     //获得运算结果
98. echo $result;

**2策略模式**

    类结构图



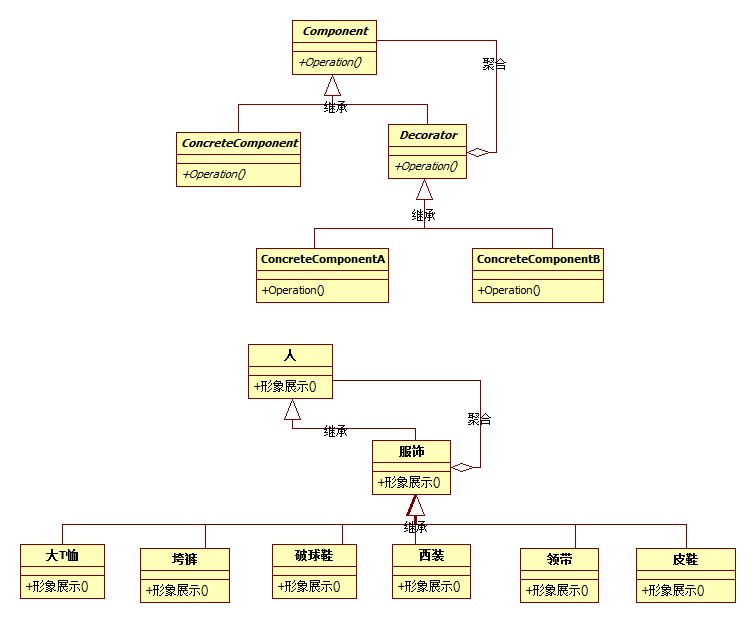
        PHP代码示例

1. <?php
2. //策略模式PHP代码
3. //现金收费抽象类
4. abstract class CashSuper
5. {
6. //抽象方法获取结果方法
7. abstract public  function acceptCash($money);
8. }
9. //正常收费子类
10. class CashNormal extends CashSuper
11. {
12. public function acceptCash($money)
13. {
14. return $money;
15. }
16. }
17. //打折收费子类
18. class CashRebate extends CashSuper
19. {
20. private $moneyRebate = 1;
21. public function \_\_construct($moneyRebate)
22. {
23. $this->moneyRebate = $moneyRebate;
24. }
25. public function acceptCash($money)
26. {
27. return $this->moneyRebate \* $money;
28. }
29. }
30. //返利收费子类
31. class CashReturn extends CashSuper
32. {
33. private $moneyConditon;
34. private $moneyReturn;
35. public function \_\_construct($moneyCondition,$moneyReturn)
36. {
37. $this->moneyConditon = $moneyCondition;
38. $this->moneyReturn = $moneyReturn;
39. }
40. public function acceptCash($money)
41. {
42. $result = $money;
43. if($money >= $this->moneyConditon)
44. {
45. $result = $money - floor($money/$this->moneyConditon)\*$this->moneyReturn;
46. }
47. return $result;
48. }
49. }
50. class CashContext
51. {
52. private $cs;
53. public function \_\_construct(CashSuper $csuper)
54. {
55. $this->cs = $csuper;
56. }
57. public function GetResult($money)
58. {
59. return $this->cs->acceptCash($money);
60. }
61. }
62. function getCash($money,$cashType)
63. {
64. $cash = 0;
65. switch($cashType)
66. {
67. case '正常收费':
68. $cashContext = new CashContext(new CashNormal());
69. break;
70. case '满300返100':
71. $cashContext = new CashContext(new CashReturn(300,100));
72. break;
73. case '打八折':
74. $cashContext = new CashContext(new CashRebate(0.8));
75. break;
76. }
77. $cash = $cashContext->GetResult($money);
78. echo $cash."<br/>";
79. }
80. getCash(400,'正常收费');
81. getCash(400,'满300返100');
82. getCash(400,'打八折');
83. /\*
84. //策略模式与简单工厂模式结合
85. class CashContext
86. {
87. private $cs;
88. public function \_\_construct($cashType)
89. {
90. switch($cashType)
91. {
92. case '正常收费':
93. $this->cs = new CashNormal();
94. break;
95. case '满300返100':
96. $this->cs = new CashReturn(300,100);
97. break;
98. case '打八折':
99. $this->cs = new CashRebate(0.8);
100. break;
101. }
102. }
103. public function GetResult($money)
104. {
105. return $this->cs->acceptCash($money);
106. }
107. }
108. function getCash($money,$cashType)
109. {
110. $cash = 0;
111. $cashContext = new CashContext($cashType);
112. $cash = $cashContext->GetResult($money);
113. echo $cash."<br/>";
114. }
115. getCash(400,'正常收费');
116. getCash(400,'满300返100');
117. getCash(400,'打八折');\*/

        策略模式是一种定义一系列算法的方法，从概念上来看，所有这些算法完成的都是相同的工作，只是实现的不同，它可以相同的方式调用所有的算法，减少各种算法与使用算法类之间的耦合。策略模式的Stategy类层次为Context定义了一系列的可供重用的算法或行为，继承有助于析取出这些算法的公共功能。策略模式的优点是简化了单元测试，因为每个算法都有自己的类，可以通过自己的接口单独测试。

**3装饰模式**

    类结构图

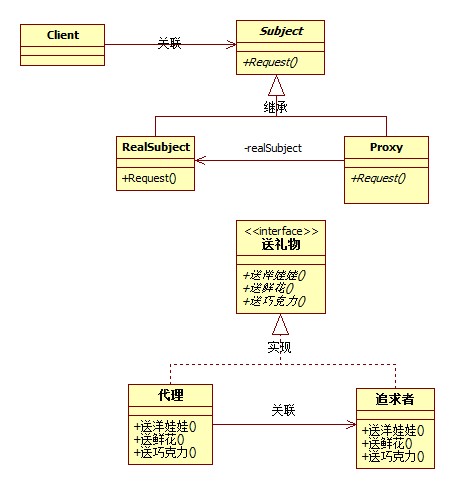


    PHP代码示例：

1. <?php
2. //装饰模式PHP代码
3. class Person
4. {
5. private $name;
6. public function SetName($name)
7. {
8. $this->name = $name;
9. }
10. public function Show()
11. {
12. echo '装扮的'.$this->name."<br />";
13. }
14. }
15. //服饰类
16. class Finery extends Person
17. {
18. protected $compoent;
19. //打扮
20. public function Decorate(Person $component)
21. {
22. $this->compoent = $component;
23. }
24. public function Show()
25. {
26. if($this->compoent != null){
27. $this->compoent->Show();
28. }
29. }
30. }
31. class TShits extends Finery
32. {
33. public function Show()
34. {
35. echo '大T恤 ';
36. $this->compoent->Show();
37. }
38. }
39. class BigTrouser extends Finery
40. {
41. public function Show()
42. {
43. echo '垮裤 ';
44. $this->compoent->Show();
45. }
46. }
47. class Sneakers extends Finery
48. {
49. public function Show()
50. {
51. echo '破球鞋 ';
52. $this->compoent->Show();
53. }
54. }
55. class LeatherShoes extends Finery
56. {
57. public function \_\_construct()
58. {
59. }
60. public function Show()
61. {
62. echo '皮鞋 ';
63. $this->compoent->Show();
64. }
65. }
66. class Tie extends Finery
67. {
68. public function Show()
69. {
70. echo '领带 ';
71. $this->compoent->Show();
72. }
73. }
74. class Suit extends Finery
75. {
76. public function Show()
77. {
78. echo '西装 ';
79. $this->compoent->Show();
80. }
81. }
82. $sneaker = new Sneakers();
83. $bigTrouser = new BigTrouser();
84. $TShirts = new TShits();
85. $person = new Person();
86. $person->SetName('小菜');
87. $sneaker->Decorate($person);
88. $bigTrouser->Decorate($sneaker);
89. $TShirts->Decorate($bigTrouser);
90. $TShirts->Show();
91. $leather = new LeatherShoes();
92. $tie = new Tie();
93. $suit = new Suit();
94. $leather->Decorate($person);
95. $tie->Decorate($leather);
96. $suit->Decorate($tie);
97. $suit->Show();

**4代理模式**

    类结构

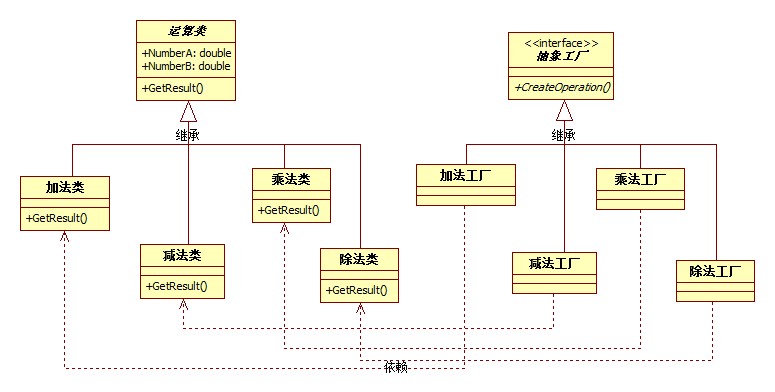


    PHP代码示例

1. <?php
2. //代理模式PHP代码
3. //被追求者类
4. class SchoolGirl
5. {
6. public $name;
7. }
8. //代理接口
9. interface IGiveGift
10. {
11. public function GiveDolls();
12. public function GiveFlowers();
13. public function GiveChocolate();
14. }
15. //追求者类
16. class Pursuit implements IGiveGift
17. {
18. private $mm;
19. public function \_\_construct(SchoolGirl $mm)
20. {
21. $this->mm = $mm;
22. }
23. public function GiveDolls()
24. {
25. echo $this->mm->name.' 送你洋娃娃。'."<br />";
26. }
27. public function GiveFlowers()
28. {
29. echo $this->mm->name.' 送你鲜花。'."<br />";
30. }
31. public function GiveChocolate()
32. {
33. echo $this->mm->name.' 送你巧克力。'."<br />";
34. }
35. }
36. //代理类
37. class Proxy implements IGiveGift
38. {
39. public $gg;
40. public function \_\_construct(SchoolGirl $mm)
41. {
42. $this->gg = new Pursuit($mm);
43. }
44. public function GiveDolls()
45. {
46. $this->gg->GiveDolls();
47. }
48. public function GiveFlowers()
49. {
50. $this->gg->GiveFlowers();
51. }
52. public function GiveChocolate()
53. {
54. $this->gg->GiveChocolate();
55. }
56. }
57. $schoolGirl = new SchoolGirl();
58. $schoolGirl->name = '邱夏';
59. $proxy = new Proxy($schoolGirl);
60. $proxy->GiveDolls();
61. $proxy->GiveFlowers();
62. $proxy->GiveChocolate();

**5工厂方法模式**

    类结构图

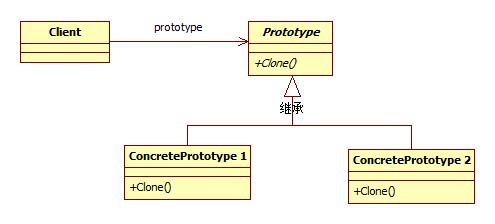


    PHP代码示例

1. <?php
2. //工厂方法PHP代码
3. //抽象运算类
4. abstract class Operation
5. {
6. public $A;          //参数A
7. public $B;          //参数B
8. public $result;     //运算结果
9. //抽象方法获取结果方法
10. abstract protected  function getResult();
11. }
12. /\*\*
13. \* Class OperationAdd
14. \* 运算 和 类
15. \*/
16. class OperationAdd extends  Operation
17. {
18. //具体实现抽象类中的getResult方法
19. public function getResult()
20. {
21. $this->result = $this->A + $this->B;
22. return $this->result;
23. }
24. }
25. /\*\*
26. \* Class OperationSub
27. \* 运算 差 类
28. \*/
29. class OperationSub extends  Operation
30. {
31. //具体实现抽象类中的getResult方法
32. public function getResult()
33. {
34. $this->result = $this->A - $this->B;
35. return $this->result;
36. }
37. }
38. /\*\*
39. \* Class OperationMul
40. \* 运算 乘积 类
41. \*/
42. class OperationMul extends  Operation
43. {
44. //具体实现抽象类中的getResult方法
45. public function getResult()
46. {
47. $this->result = $this->A \* $this->B;
48. return $this->result;
49. }
50. }
51. /\*\*
52. \* Class OperationDiv
53. \* 运算 除 类
54. \*/
55. class OperationDiv extends  Operation
56. {
57. //具体实现抽象类中的getResult方法
58. public function getResult()
59. {
60. if($this->B == 0){
61. exit( '除数不能为0');
62. }
63. $this->result = $this->A / $this->B;
64. return $this->result;
65. }
66. }
67. //工厂接口
68. interface IFactory
69. {
70. public function CreateOperation();
71. }
72. //加法工厂类
73. class AddFactory implements IFactory
74. {
75. public function CreateOperation()
76. {
77. return new OperationAdd();
78. }
79. }
80. //减法工厂类
81. class SubFactory implements IFactory
82. {
83. public function CreateOperation()
84. {
85. return new OperationSub();
86. }
87. }
88. //乘法工厂
89. class MulFactory implements IFactory
90. {
91. public function CreateOperation()
92. {
93. return new OperationMul();
94. }
95. }
96. //除法工厂
97. class DivFactory implements IFactory
98. {
99. public function CreateOperation()
100. {
101. return new OperationDiv();
102. }
103. }
104. $operFactory = new AddFactory();    //通过简单工厂类获取类
105. $operationClass = $operFactory->CreateOperation();
106. $operationClass->A = 8;                     //初始化参数
107. $operationClass->B = 4;                     //初始化参数
108. $result = $operationClass->getResult();     //获得运算结果
109. echo $result;
110. /\*
111. //雷锋类
112. class LeiFeng
113. {
114. public function Sweep()
115. {
116. echo "扫地<br />";
117. }
118. public function Wash()
119. {
120. echo "洗衣<br />";
121. }
122. public function BuyRice()
123. {
124. echo "买米<br />";
125. }
126. }
127. //学雷锋的大学生
128. class Undergraduate extends LeiFeng
129. {}
130. //学雷锋的志愿者
131. class Volunteer extends LeiFeng
132. {}
133. //雷锋工厂
134. interface IFactory
135. {
136. public function CreateLeiFeng();
137. }
138. class UndergraduateFactory implements IFactory
139. {
140. public function CreateLeiFeng()
141. {
142. return new Undergraduate();
143. }
144. }
145. class VolunteerFactory implements IFactory
146. {
147. public function CreateLeiFeng()
148. {
149. return new Volunteer();
150. }
151. }
152. $factory = new UndergraduateFactory();
153. $stuent = $factory->CreateLeiFeng();
154. $stuent->BuyRice();
155. $stuent->Sweep();
156. $stuent->Wash();\*/

**6原型模式**

    类结构图：



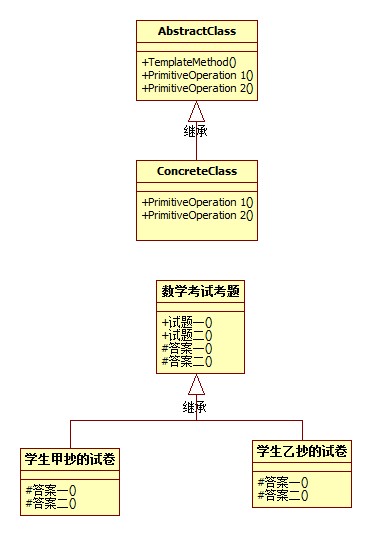
    PHP代码示例：

1. <?php
2. //原型模式PHP代码示例
3. //浅复制 start
4. //简历类
5. class Resume
6. {
7. private $name;
8. private $sex;
9. private $age;
10. private $timeArea;
11. private $company;
12. public function \_\_construct($name)
13. {
14. $this->name = $name;
15. }
16. public function SetPersonalInfo($sex , $age)
17. {
18. $this->sex = $sex;
19. $this->age = $age;
20. }
21. public function SetWorkExperience($timeArea , $company)
22. {
23. $this->timeArea = $timeArea;
24. $this->company = $company;
25. }
26. public function Display()
27. {
28. echo $this->name.' '.$this->sex.' '.$this->age."<br />";
29. echo '工作经历: '.$this->timeArea.' '.$this->company."<br />";
30. }
31. }
32. $Resume = new Resume('文道蛟');
33. $Resume->SetPersonalInfo('男', 29);
34. $Resume->SetWorkExperience('2011 - 2012', '深圳中联通电子有限公司');
35. $Resume\_a = clone $Resume;
36. $Resume\_a->SetWorkExperience('2012 - 2014', '北京因动思网络技术有限公司');
37. $Resume\_b = clone $Resume;
38. $Resume\_b->SetWorkExperience('2014 - 2015', '软通动力深圳分公司');
39. $Resume->Display();
40. $Resume\_a->Display();
41. $Resume\_b->Display();
42. //浅复制 end
43. /\*//深复制 start
44. //工作经历类
45. class WorkExperience
46. {
47. public $workDate;
48. public $company;
49. }
50. //简历类
51. class Resume
52. {
53. private $name;
54. private $sex;
55. private $age;
56. private $workExperience;
57. public function \_\_construct($name, WorkExperience $workExperience)
58. {
59. $this->name = $name;
60. $this->workExperience = $workExperience;
61. }
62. public function SetPersonalInfo($sex , $age)
63. {
64. $this->sex = $sex;
65. $this->age = $age;
66. }
67. public function SetWorkExperience($timeArea , $company)
68. {
69. $this->workExperience->workDate = $timeArea;
70. $this->workExperience->company = $company;
71. }
72. public function Display()
73. {
74. echo $this->name.' '.$this->sex.' '.$this->age."<br />";
75. echo '工作经历: '.$this->workExperience->workDate.' '.$this->workExperience->company."<br />";
76. }
77. public function \_\_clone()
78. {
79. $this->workExperience = new WorkExperience();
80. }
81. }
82. $workExperience = new WorkExperience();
83. $Resume = new Resume('文道蛟',$workExperience);
84. $Resume->SetPersonalInfo('男', 29);
85. $Resume->SetWorkExperience('2011 - 2012', '深圳中联通电子有限公司');
86. $Resume\_a = clone $Resume;
87. $Resume\_a->SetWorkExperience('2012 - 2014', '北京因动思网络技术有限公司');
88. $Resume\_b = clone $Resume;
89. $Resume\_b->SetWorkExperience('2014 - 2015', '软通动力深圳分公司');
90. $Resume->Display();
91. $Resume\_a->Display();
92. $Resume\_b->Display();\*/
93. //深复制 end

    一般在初始化的信息不发生变化的情况下，克隆是最好的办法，这既隐藏了对象创建的细节，又对性能是大大的提高。如果字段是值类型，则对该字段执行逐位复制，如果字段是引用类型，则复制引用不复制引用的对象，因此，原始对象及其复本引用同一对象；

**7模板方法模式**

    类结构图



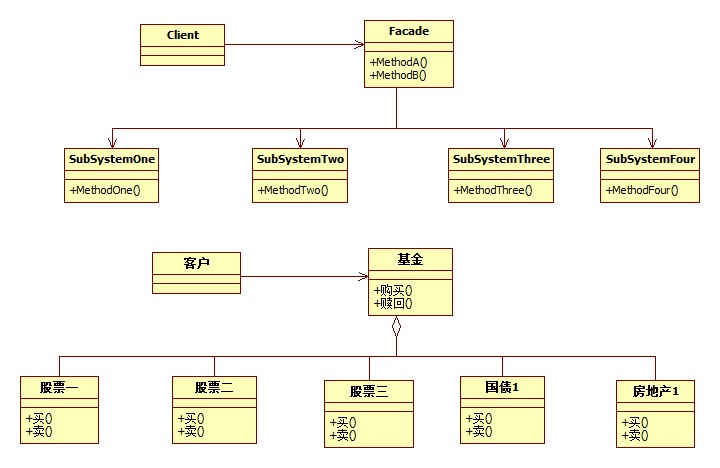
    PHP代码示例:

1. <?php
2. //模板方法模式PHP代码示例
3. //试卷父类
4. class TestPaper
5. {
6. public function TestQuestion1()
7. {
8. echo '3 + 7 = '.$this->Answer1()."<br />";
9. }
10. public function TestQuestion2()
11. {
12. echo '5 \* 9 = '.$this->Answer2()."<br />";
13. }
14. protected  function Answer1()
15. {
16. return '';
17. }
18. protected  function Answer2()
19. {
20. return '';
21. }
22. }
23. //学生甲抄的试卷
24. class TestPaperA extends TestPaper
25. {
26. protected function Answer1()
27. {
28. return 10;
29. }
30. protected  function Answer2()
31. {
32. return 44;
33. }
34. }
35. //学生乙抄的试卷
36. class TestPaperB extends TestPaper
37. {
38. protected function Answer1()
39. {
40. return 9;
41. }
42. protected function Answer2()
43. {
44. return 45;
45. }
46. }
47. $stuentA = new TestPaperA();
48. $stuentA->TestQuestion1();
49. $stuentA->TestQuestion2();
50. $stuentB = new TestPaperB();
51. $stuentB->TestQuestion1();
52. $stuentB->TestQuestion2();

        模板方法模式是通过把不变的行为搬移到超类，去除子类中的重复代码去体现它的优势。当不变的和可变的行为在方法的子类中混合在一起的时候不变的行为就会在子类中重复出现，我们通过模板方法模式把这些行为搬移到单一的地方，这样就帮助子类摆脱重复的不变行为的纠缠。

**8外观模式**

    类结构图

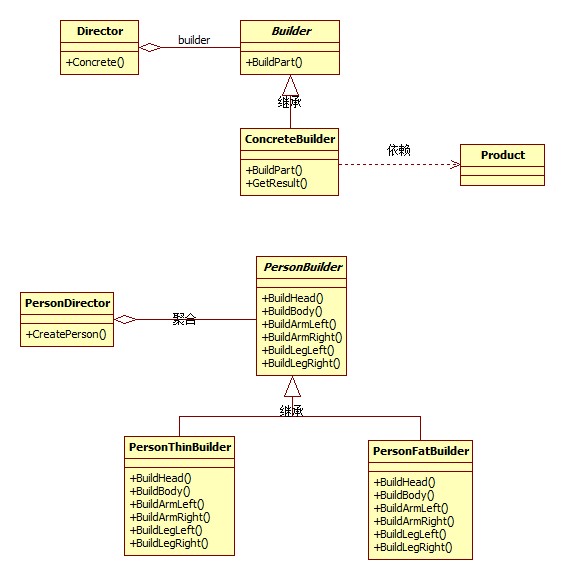


    PHP代码示例:

1. <?php
2. //外观模式PHP代码示例
3. //股票1
4. class Stock1
5. {
6. public function sell()
7. {
8. echo "股票1卖出。<br />";
9. }
10. public function buy()
11. {
12. echo "股票1买入。<br />";
13. }
14. }
15. //股票2
16. class Stock2
17. {
18. public function sell()
19. {
20. echo "股票2卖出。<br />";
21. }
22. public function buy()
23. {
24. echo "股票2买入。<br />";
25. }
26. }
27. //股票3
28. class Stock3
29. {
30. public function sell()
31. {
32. echo "股票3卖出。<br />";
33. }
34. public function buy()
35. {
36. echo "股票3买入。<br />";
37. }
38. }
39. //国债1
40. class NationalDebt1
41. {
42. public function sell()
43. {
44. echo "国债1卖出。<br />";
45. }
46. public function buy()
47. {
48. echo "国债1买入。<br />";
49. }
50. }
51. //房地产1
52. class Realty1
53. {
54. public function sell()
55. {
56. echo "房地产1卖出。<br />";
57. }
58. public function buy()
59. {
60. echo "房地产1买入。<br />";
61. }
62. }
63. class Fund
64. {
65. private $gu1;
66. private $gu2;
67. private $gu3;
68. private $nd1;
69. private $rt1;
70. public function \_\_construct()
71. {
72. $this->gu1 = new Stock1();
73. $this->gu2 = new Stock2();
74. $this->gu3 = new Stock3();
75. $this->nd1 = new NationalDebt1();
76. $this->rt1 = new Realty1();
77. }
78. public function BuyFund()
79. {
80. $this->gu1->buy();
81. $this->gu2->buy();
82. $this->gu3->buy();
83. $this->nd1->buy();
84. $this->rt1->buy();
85. }
86. public function SellFund()
87. {
88. $this->gu1->sell();
89. $this->gu2->sell();
90. $this->gu3->sell();
91. $this->nd1->sell();
92. $this->rt1->sell();
93. }
94. }
95. $jijin = new Fund();
96. $jijin->BuyFund();
97. $jijin->SellFund();

**9建造者模式**

    类结构图

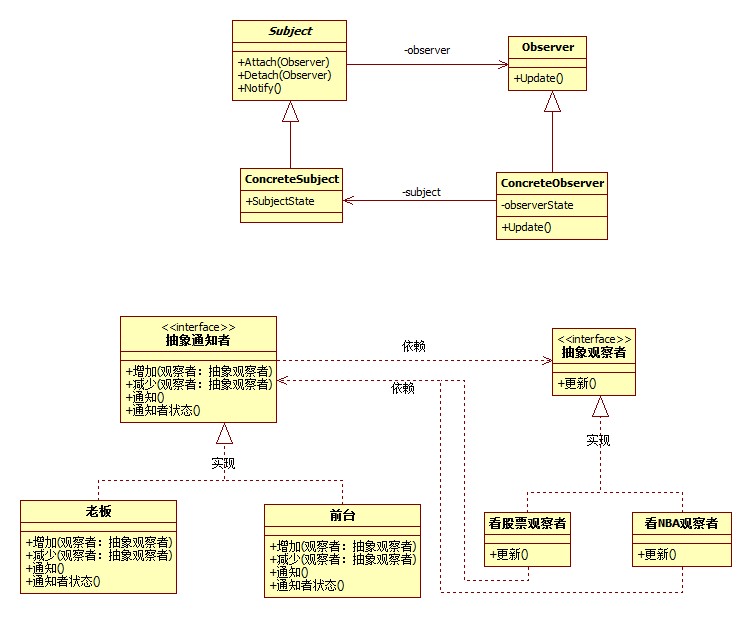


    PHP代码示例:

1. <?php
2. //建造者（Builder）模式PHP代码示例
3. //产品类
4. class Product
5. {
6. private $parts = array();
7. public function Add($part)
8. {
9. $this->parts[] = $part;
10. }
11. public function Show()
12. {
13. foreach($this->parts as $val)
14. {
15. echo $val."<br />";
16. }
17. }
18. }
19. abstract class Builder
20. {
21. abstract public function BuildPartA();
22. abstract public function BuildPartB();
23. abstract public function GetResult();
24. }
25. class ConcreteBuilder1 extends Builder
26. {
27. private $product;
28. public function \_\_construct(Product $product)
29. {
30. $this->product = $product;
31. }
32. public function BuildPartA()
33. {
34. $this->product->Add('部件A');
35. }
36. public function BuildPartB()
37. {
38. $this->product->Add('部件B');
39. }
40. public function GetResult()
41. {
42. return $this->product;
43. }
44. }
45. class ConcreteBuilder2 extends Builder
46. {
47. private $product;
48. public function \_\_construct(Product $product)
49. {
50. $this->product = $product;
51. }
52. public function BuildPartA()
53. {
54. $this->product->Add('部件X');
55. }
56. public function BuildPartB()
57. {
58. $this->product->Add('部件Y');
59. }
60. public function GetResult()
61. {
62. return $this->product;
63. }
64. }
65. class Director
66. {
67. public function Concrete(Builder $builder)
68. {
69. $builder->BuildPartA();
70. $builder->BuildPartB();
71. }
72. }
73. $product1 = new Product();
74. $b1 = new ConcreteBuilder1($product1);
75. $product2 = new Product();
76. $b2 = new ConcreteBuilder2($product2);
77. $director = new Director();
78. $director->Concrete($b1);
79. $product = $b1->GetResult();
80. $product->Show();
81. $director->Concrete($b2);
82. $product = $b2->GetResult();
83. $product->Show();

**10观察者模式**

    类结构图：

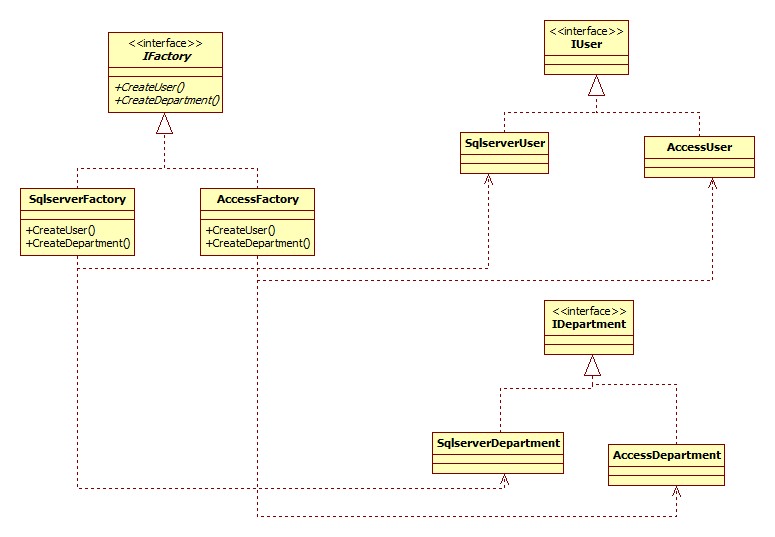


    PHP代码示例：

1. <?php
2. //观察者（Observer）模式PHP代码示例
3. //通知者接口
4. interface Subject
5. {
6. public function Attach(Observer $observer);
7. public function Detach(Observer $observer);
8. public function Notify();
9. }
10. class Boss implements Subject
11. {
12. private $observerArray = array();
13. public $subjectState;
14. public function Attach(Observer $observer)
15. {
16. $this->observerArray[] = $observer;
17. }
18. public function Detach(Observer $observer)
19. {
20. $key = array\_search($observer,$this->observerArray);
21. if($key)
22. {
23. unset($this->observerArray[$key]);
24. }
25. }
26. public function Notify()
27. {
28. foreach($this->observerArray as $observer)
29. {
30. $observer->Update();
31. }
32. }
33. }
34. abstract class Observer
35. {
36. protected  $name;
37. protected  $subject;
38. public function \_\_construct($name,Subject $subject)
39. {
40. $this->name = $name;
41. $this->subject = $subject;
42. }
43. abstract  public function Update();
44. }
45. class StockObserver extends Observer
46. {
47. public function Update()
48. {
49. echo $this->subject->subjectState.' '.$this->name."关闭股票行情，继续工作。<br />";
50. }
51. }
52. class NBAObserver extends Observer
53. {
54. public function Update()
55. {
56. echo $this->subject->subjectState.' '.$this->name."关闭NBA直播，继续工作。<br />";
57. }
58. }
59. $huhansan = new Boss();
60. $tongshi1 = new StockObserver('炒股同事', $huhansan);
61. $tongshi2 = new StockObserver('篮球迷同事', $huhansan);
62. $huhansan->Attach($tongshi1);
63. $huhansan->Attach($tongshi2);
64. $huhansan->subjectState = '我胡汉三回来了！';
65. $huhansan->Detach($tongshi2);
66. $huhansan->Notify();

**11抽象工厂模式**

    类结构图:

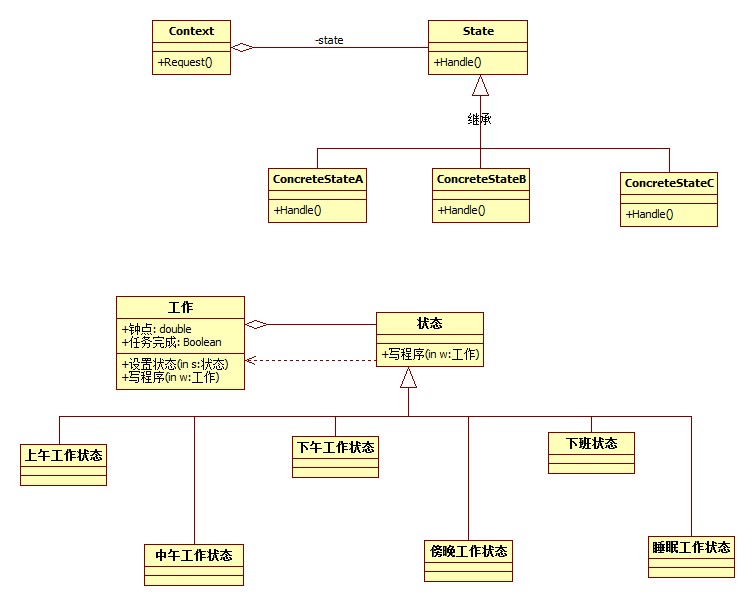


    PHP代码示例

1. <?php
2. //抽象工厂模式PHP代码示例
3. class User{}
4. class Department{}
5. interface IUser
6. {
7. public function Insert(User $user);
8. public function GetUser($id);
9. }
10. class SqlserverUser implements IUser
11. {
12. public function Insert(User $user)
13. {
14. echo "在SQL Server 中给User表中增加一条记录。<br />";
15. }
16. public function GetUser($id)
17. {
18. echo "在SQL Server 中更加ID得到User表中的一条记录。<br />";
19. }
20. }
21. class AccessUser implements IUser
22. {
23. public function Insert(User $user)
24. {
25. echo "在Access Server 中给User表中增加一条记录。<br />";
26. }
27. public function GetUser($id)
28. {
29. echo "在Access Server 中更加ID得到User表中的一条记录。<br />";
30. }
31. }
32. interface IDepartment
33. {
34. public function Insert(Department $department);
35. public function GetDepartment($id);
36. }
37. class SqlserverDepartment implements IDepartment
38. {
39. public function Insert(Department $department)
40. {
41. echo "在SQL Server 中给Department表中增加一条记录。<br />";
42. }
43. public function GetDepartment($id)
44. {
45. echo "在SQL Server 中更加ID得到Department表中的一条记录。<br />";
46. }
47. }
48. class AccessDepartment implements IDepartment
49. {
50. public function Insert(Department $department)
51. {
52. echo "在Access Server 中给Department表中增加一条记录。<br />";
53. }
54. public function GetDepartment($id)
55. {
56. echo "在Access Server 中更加ID得到Department表中的一条记录。<br />";
57. }
58. }
59. interface IFactory
60. {
61. public function CreateUser();
62. public function CreateDepartment();
63. }
64. class SqlserverFactory implements  IFactory
65. {
66. public function CreateUser()
67. {
68. return new SqlserverUser();
69. }
70. public function CreateDepartment()
71. {
72. return new SqlserverDepartment();
73. }
74. }
75. class AccessFactory implements  IFactory
76. {
77. public function CreateUser()
78. {
79. return new AccessUser();
80. }
81. public function CreateDepartment()
82. {
83. return new AccessDepartment();
84. }
85. }
86. $user = new User();
87. $department = new Department();
88. $factory = new SqlserverFactory();
89. $iu = $factory->CreateUser();
90. $iu->Insert($user);
91. $iu->GetUser(1);
92. $id = $factory->CreateDepartment();
93. $id->Insert($department);
94. $id->GetDepartment(1);

**12状态模式**

    类结构图

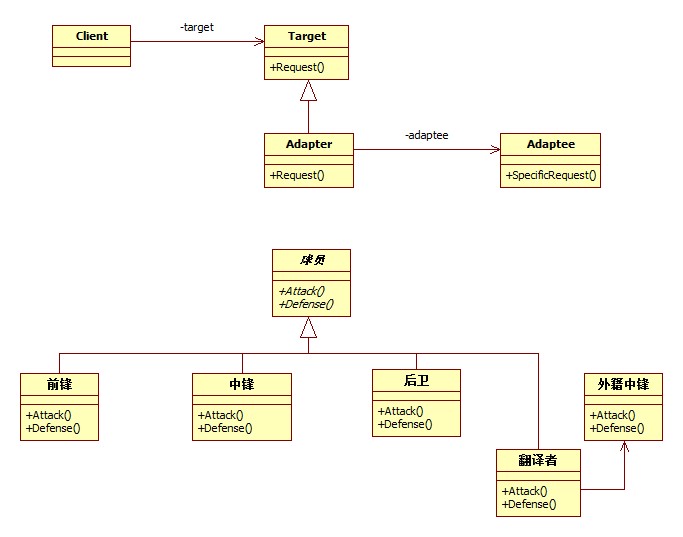


    PHP代码示例

1. <?php
2. //状态模式PHP代码示例
3. abstract class State
4. {
5. abstract public function WriteProgram(Work $work);
6. }
7. //上午工作状态
8. class ForenoonState extends State
9. {
10. public function WriteProgram(Work $work)
11. {
12. if($work->hour < 12)
13. {
14. echo "当前时间:{$work->hour}点 上午工作 精神百倍<br />";
15. }
16. else
17. {
18. $work->SetState(new NoonState());
19. $work->WriteProgram();
20. }
21. }
22. }
23. //中午工作状态
24. class NoonState extends State
25. {
26. public function WriteProgram(Work $work)
27. {
28. if($work->hour < 13)
29. {
30. echo "当前时间:{$work->hour}点 饿了, 午饭; 犯困，午休<br />";
31. }
32. else
33. {
34. $work->SetState(new AfternoonState());
35. $work->WriteProgram();
36. }
37. }
38. }
39. //下午工作状态
40. class AfternoonState extends State
41. {
42. public function WriteProgram(Work $work)
43. {
44. if($work->hour < 17)
45. {
46. echo "当前时间:{$work->hour}点 下午状态还不错，继续努力<br />";
47. }
48. else
49. {
50. $work->SetState(new EveningState());
51. $work->WriteProgram();
52. }
53. }
54. }
55. //晚上工作状态
56. class EveningState extends State
57. {
58. public function WriteProgram(Work $work)
59. {
60. if($work->finish)
61. {
62. $work->SetState(new RestState());
63. $work->WriteProgram();
64. }
65. else
66. {
67. if($work->hour <21)
68. {
69. echo "当前时间:{$work->hour}点 加班哦, 疲累之极<br />";
70. }
71. else
72. {
73. $work->SetState(new SleepState);
74. $work->WriteProgram();
75. }
76. }
77. }
78. }
79. //睡眠状态
80. class SleepState extends State
81. {
82. public function WriteProgram(Work $work)
83. {
84. echo "当前时间:{$work->hour}点不行了,睡着了<br />";
85. }
86. }
87. //休息状态
88. class RestState extends State
89. {
90. public function WriteProgram(Work $work)
91. {
92. echo "当前时间:{$work->hour}点下班回家了<br />";
93. }
94. }
95. //工作
96. class Work
97. {
98. private $current;
99. public $hour;
100. public $finish = false;
101. public function \_\_construct(){
102. $this->current = new ForenoonState();
103. }
104. public function SetState(State $state)
105. {
106. $this->current = $state;
107. }
108. public function WriteProgram()
109. {
110. $this->current->WriteProgram($this);
111. }
112. }
113. $work = new work();
114. $work->hour = 9;
115. $work->WriteProgram();
116. $work->hour = 10;
117. $work->WriteProgram();
118. $work->hour = 12;
119. $work->WriteProgram();
120. $work->hour = 13;
121. $work->WriteProgram();
122. $work->hour = 14;
123. $work->WriteProgram();
124. $work->hour = 17;
125. $work->WriteProgram();
126. //$work->finish = true;
127. $work->hour = 19;
128. $work->WriteProgram();
129. $work->hour = 22;
130. $work->WriteProgram();

**13适配器模式**

    类结构图

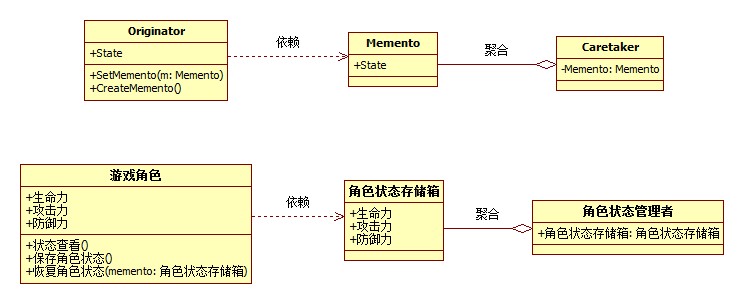


    PHP代码示例

1. <?php
2. //适配器模式PHP代码示例
3. //球员
4. abstract class Player
5. {
6. public $name;
7. public function \_\_construct($name)
8. {
9. $this->name = $name;
10. }
11. abstract public function Attack();
12. abstract public function Defense();
13. }
14. //前锋
15. class Forwards extends Player
16. {
17. public $name;
18. public function \_\_construct($name)
19. {
20. $this->name = $name;
21. }
22. public function Attack()
23. {
24. echo "前锋{$this->name}进攻<br />";
25. }
26. public function Defense()
27. {
28. echo "前锋{$this->name}防守<br />";
29. }
30. }
31. //中锋
32. class Center extends Player
33. {
34. public $name;
35. public function \_\_construct($name)
36. {
37. $this->name = $name;
38. }
39. public function Attack()
40. {
41. echo "中锋{$this->name}进攻<br />";
42. }
43. public function Defense()
44. {
45. echo "中锋{$this->name}防守<br />";
46. }
47. }
48. //后卫
49. class Guards extends Player
50. {
51. public $name;
52. public function \_\_construct($name)
53. {
54. $this->name = $name;
55. }
56. public function Attack()
57. {
58. echo "后卫{$this->name}进攻<br />";
59. }
60. public function Defense()
61. {
62. echo "后卫{$this->name}防守<br />";
63. }
64. }
65. //中锋
66. class ForeignCenter extends Player
67. {
68. public $name;
69. public function \_\_construct($name)
70. {
71. $this->name = $name;
72. }
73. public function Attack()
74. {
75. echo "外籍中锋{$this->name}进攻<br />";
76. }
77. public function Defense()
78. {
79. echo "外籍中锋{$this->name}防守<br />";
80. }
81. }
82. //翻译者类
83. class Translator extends Player
84. {
85. private  $ForeignCenter;
86. public  $name;
87. public function \_\_construct(ForeignCenter $ForeignCenter)
88. {
89. $this->ForeignCenter = $ForeignCenter;
90. $this->name = $ForeignCenter->name;
91. }
92. public function Attack()
93. {
94. $this->ForeignCenter->Attack();
95. }
96. public function Defense()
97. {
98. $this->ForeignCenter->Defense();
99. }
100. }
101. $b = new Forwards("巴蒂尔");
102. $b->Attack();
103. $m = new Guards("麦格雷迪");
104. $m->Attack();
105. $ym = new ForeignCenter("姚明");
106. $tr = new Translator($ym);
107. $tr->Attack();
108. $tr->Defense();

**14备忘录模式**

    类结构图

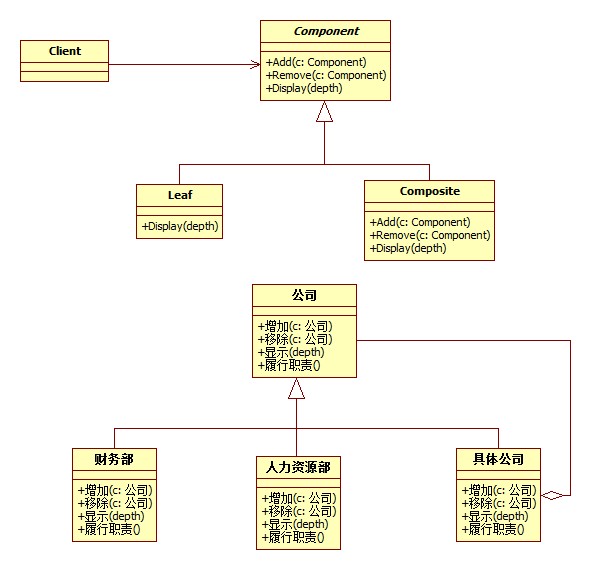


    PHP代码示例:

1. <?php
2. //备忘录模式PHP代码示例
3. //游戏角色类
4. class GameRole
5. {
6. private $vit;
7. private $atk;
8. private $def;
9. public function GetInitState()
10. {
11. $this->vit = 100;
12. $this->atk = 100;
13. $this->def = 100;
14. }
15. public function Fight()
16. {
17. $this->vit = 0;
18. $this->atk = 0;
19. $this->def = 0;
20. }
21. public function StateDisplay()
22. {
23. echo "角色状态:<br />";
24. echo "体力: {$this->vit}<br />";
25. echo "攻击: {$this->atk}<br />";
26. echo "防御: {$this->def}<br />";
27. }
28. public function SaveState()
29. {
30. return new RoleStateMemento($this->vit,$this->atk,$this->def);
31. }
32. public function RecoveryState(RoleStateMemento $memento)
33. {
34. $this->vit = $memento->GetVit();
35. $this->atk = $memento->GetAtk();
36. $this->def = $memento->GetDef();
37. }
38. }
39. //角色状态存储箱
40. class RoleStateMemento
41. {
42. private $vit;
43. private $atk;
44. private $def;
45. public function \_\_construct($vit,$atk,$def)
46. {
47. $this->vit = $vit;
48. $this->atk = $atk;
49. $this->def = $def;
50. }
51. public function GetVit()
52. {
53. return $this->vit;
54. }
55. public function GetAtk()
56. {
57. return $this->atk;
58. }
59. public function GetDef()
60. {
61. return $this->def;
62. }
63. }
64. //管理者类
65. class RoleStateCaretaker
66. {
67. private $memento;
68. public function  SetMemento(RoleStateMemento $memento)
69. {
70. $this->memento = $memento;
71. }
72. public function GetMemento()
73. {
74. return $this->memento;
75. }
76. }
77. $role = new GameRole();
78. $role->GetInitState();
79. $role->StateDisplay();
80. //保存进度
81. $stateAdmin = new RoleStateCaretaker();
82. $stateAdmin->SetMemento($role->SaveState());
83. $role->Fight();
84. $role->StateDisplay();
85. $role->RecoveryState($stateAdmin->GetMemento());
86. $role->StateDisplay();

**15组合模式**

    类结构图

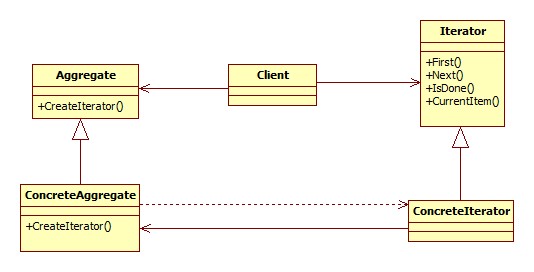


    PHP代码示例:

1. <?php
2. //组合模式PHP代码示例
3. //抽象公司类
4. abstract class Company
5. {
6. protected $name;
7. public function \_\_construct($name)
8. {
9. $this->name = $name;
10. }
11. abstract public function Add(Company $c);
12. abstract public function Remove(Company $c);
13. abstract public function Display($depth);
14. abstract public function LineOfDuty();
15. }
16. //具体公司类
17. class ConcreteCompany extends Company
18. {
19. private $list = array();
20. public function Add(Company $c)
21. {
22. $this->list[] = $c;
23. }
24. public function Remove(Company $c)
25. {
26. $key = array\_search($c,$this->list);
27. unset($this->list[$key]);
28. }
29. public function Display($depth)
30. {
31. echo str\_repeat('-',$depth).$this->name."<br >";
32. if($this->list){
33. foreach($this->list as $value){
34. $value->Display($depth+2);
35. }
36. }
37. }
38. public function LineOfDuty()
39. {
40. if($this->list){
41. foreach($this->list as $value){
42. $value->LineOfDuty();
43. }
44. }
45. }
46. }
47. //人力资源部
48. class HRDepartment extends Company
49. {
50. public function Add(Company $c){}
51. public function Remove(Company $c){}
52. public function Display($depth)
53. {
54. echo str\_repeat('-',$depth).$this->name."<br />";
55. }
56. public function LineOfDuty()
57. {
58. echo $this->name."员工招聘管理<br />";
59. }
60. }
61. //人力资源部
62. class FinanceDepartment extends Company
63. {
64. public function Add(Company $c){}
65. public function Remove(Company $c){}
66. public function Display($depth)
67. {
68. echo str\_repeat('-',$depth).$this->name."<br />";
69. }
70. public function LineOfDuty()
71. {
72. echo $this->name."公司财务收支管理<br />";
73. }
74. }
75. $root = new ConcreteCompany('北京总公司');
76. $root->Add(new HRDepartment('总公司人力资源部'));
77. $root->Add(new FinanceDepartment('总公司财务部'));
78. $comp = new ConcreteCompany('上海华东分公司');
79. $comp->Add(new HRDepartment('上海华东分公司人力资源部'));
80. $comp->Add(new FinanceDepartment('上海华东分公司财务部'));
81. $root->Add($comp);
82. $comp1 = new ConcreteCompany('南京办事处');
83. $comp1->Add(new HRDepartment('南京办事处人力资源部'));
84. $comp1->Add(new FinanceDepartment('南京办事处公司财务部'));
85. $comp->Add($comp1);
86. $comp2 = new ConcreteCompany('杭州办事处');
87. $comp2->Add(new HRDepartment('杭州办事处人力资源部'));
88. $comp2->Add(new FinanceDepartment('杭州办事处公司财务部'));
89. $comp->Add($comp2);
90. //$comp->Remove($comp1);
91. $root->Display(1);
92. $root->LineOfDuty();

**16迭代器模式**

    类结构图

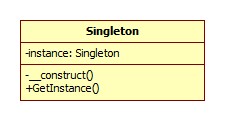


    PHP代码示例：

1. <?php
2. //迭代器模式PHP代码示例
3. //Iterator 迭代器抽象类
4. abstract class Iteratorclass
5. {
6. abstract public function First();
7. abstract public function Next();
8. abstract public function IsDone();
9. abstract public function CurrentItem();
10. }
11. //Aggregate 聚集抽象类
12. abstract class Aggregate
13. {
14. abstract public function CreateIterator();
15. }
16. //ConcreteIterator 具体迭代器类
17. class ConcreteIterator extends Iteratorclass
18. {
19. private $aggregate;
20. private $data;
21. public function \_\_construct(ConcreteAggregate $aggregate)
22. {
23. $this->aggregate = $aggregate;
24. $this->data = $aggregate->GetData();
25. }
26. public function First()
27. {
28. return $this->data[0];
29. }
30. public function Next()
31. {
32. return next($this->data);
33. }
34. public function IsDone()
35. {
36. return ( (key($this->data)+1) >= count($this->data))?TRUE:FALSE;
37. }
38. public function CurrentItem()
39. {
40. return current($this->data);
41. }
42. }
43. //ConcreteAggregate 具体聚集类
44. class ConcreteAggregate extends Aggregate
45. {
46. private  $data;
47. public function \_\_construct(Array $data)
48. {
49. $this->data = $data;
50. }
51. public function CreateIterator()
52. {
53. return new ConcreteIterator($this);
54. }
55. public function Count()
56. {
57. return count($this->data);
58. }
59. public function GetData()
60. {
61. return $this->data;
62. }
63. }
64. $data = array('大鸟','小菜','行李','老外','公交内部员工','小偷');
65. $aggregate = new ConcreteAggregate($data);
66. $iterator = new ConcreteIterator($aggregate);
67. $item = $iterator->First();
68. echo $iterator->CurrentItem()."请买车票<br />";
69. do{
70. $iterator->Next();
71. echo $iterator->CurrentItem()."请买车票<br />";
72. }while(!$iterator->IsDone());

**17单例模式**

    类结构图

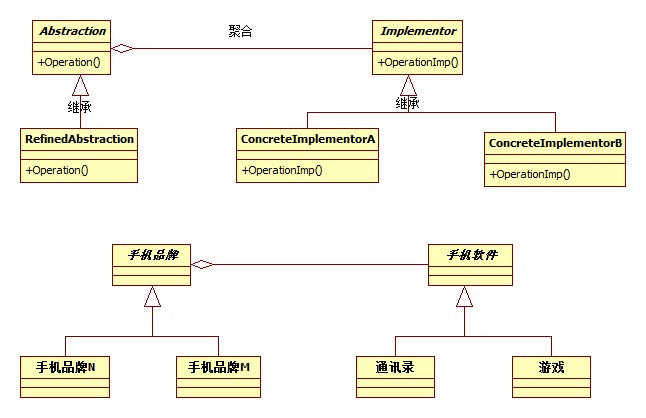


    PHP代码示例:

1. <?php
2. //单例器模式PHP代码示例
3. class Singleton
4. {
5. private static  $\_instance;
6. private function \_\_construct(){}
7. static public function GetInstance()
8. {
9. if (!(self::$\_instance instanceof self)) {
10. self::$\_instance = new self ();
11. }
12. return self::$\_instance;
13. }
14. private function \_\_clone(){}
15. }
16. $s1 = Singleton::GetInstance();
17. $s2 = Singleton::GetInstance();
18. if($s1 == $s2){
19. echo '两个对象是同一个实例！';
20. }

**18桥接模式**

    类结构图

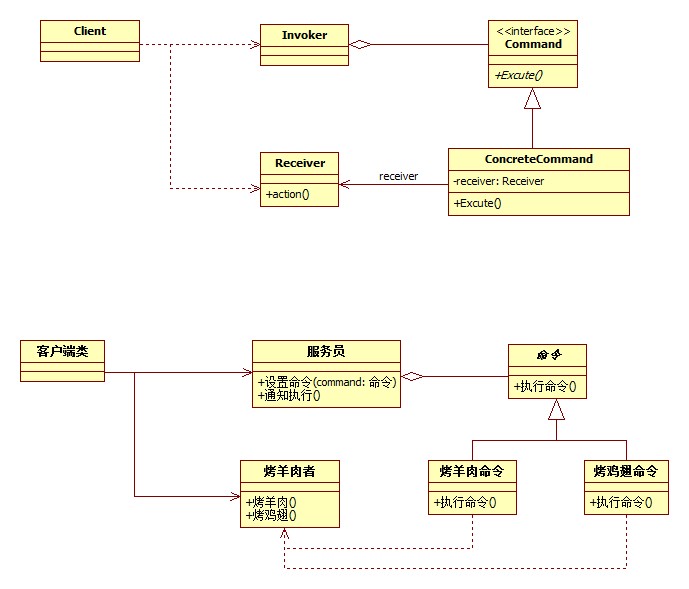


    PHP代码示例

1. <?php
2. //桥接模式PHP代码示例
3. //手机软件
4. abstract class HandsetSoft
5. {
6. abstract public function Run();
7. }
8. //手机游戏
9. class HandsetGame extends HandsetSoft
10. {
11. public function Run()
12. {
13. return "运行手机游戏。<br />";
14. }
15. }
16. //手机通讯录
17. class HandsetAddressList extends HandsetSoft
18. {
19. public function Run()
20. {
21. return "运行手机通讯录。<br />";
22. }
23. }
24. //手机品牌类
25. abstract class HandsetBrand
26. {
27. protected $soft;
28. public function SetHandsetSoft(HandsetSoft $soft)
29. {
30. $this->soft = $soft;
31. }
32. abstract public function Run();
33. }
34. class HandsetBrandN extends HandsetBrand
35. {
36. public function Run()
37. {
38. echo '品牌N'.$this->soft->Run();
39. }
40. }
41. class HandsetBrandM extends HandsetBrand
42. {
43. public function Run()
44. {
45. echo '品牌M'.$this->soft->Run();
46. }
47. }
48. $n = new HandsetBrandN();
49. $n->SetHandsetSoft(new HandsetGame());
50. $n->Run();
51. $n->SetHandsetSoft(new HandsetAddressList());
52. $n->Run();
53. $m = new HandsetBrandM();
54. $m->SetHandsetSoft(new HandsetGame());
55. $m->Run();
56. $m->SetHandsetSoft(new HandsetAddressList());
57. $m->Run();

**19命令模式**

    类结构图

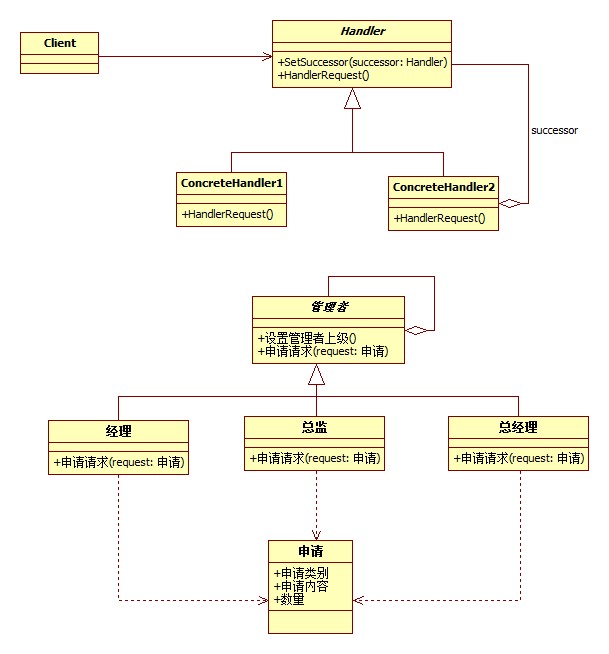


    PHP代码示例:

1. <?php
2. //命令模式PHP代码示例
3. //烤肉串者类
4. class BarBecuer
5. {
6. //烤羊肉
7. public function BakeMutton()
8. {
9. echo "烤羊肉串。<br />";
10. }
11. //烤鸡翅
12. public function BakeChickenWing()
13. {
14. echo "烤鸡翅。<br />";
15. }
16. }
17. abstract class Command
18. {
19. protected $receiver;
20. public function \_\_construct(BarBecuer $receiver)
21. {
22. $this->receiver = $receiver;
23. }
24. abstract public function GetCommandName();
25. abstract public function ExcuteCommand();
26. }
27. class BakeMuttonCommand extends Command
28. {
29. public function ExcuteCommand(){
30. $this->receiver->BakeMutton();
31. }
32. public function GetCommandName()
33. {
34. return "烤羊肉";
35. }
36. }
37. class BakeChickenWingCommand extends Command
38. {
39. public function ExcuteCommand()
40. {
41. $this->receiver->BakeChickenWing();
42. }
43. public function GetCommandName()
44. {
45. return "烤鸡翅";
46. }
47. }
48. class Waiter
49. {
50. private $list;
51. public function SetOrder(Command $command)
52. {
53. if($command->GetCommandName() == '烤鸡翅')
54. {
55. echo "鸡翅没有了。<br />";
56. }
57. else
58. {
59. $this->list[] = $command;
60. }
61. }
62. public function ConcelOrder(Command $command)
63. {
64. $key = array\_search($command,$this->list);
65. unset($this->list[$key]);
66. echo '取消订单'.$command->GetCommandName().date('Y-m-d H:i:s')."<br >";
67. }
68. public function Notify()
69. {
70. if($this->list)
71. {
72. foreach($this->list as $command)
73. {
74. $command->ExcuteCommand();
75. }
76. }
77. }
78. }
79. $boy = new BarBecuer();
80. $command1 = new BakeMuttonCommand($boy);
81. $command2 = new BakeMuttonCommand($boy);
82. $command3 = new BakeChickenWingCommand($boy);
83. $waiter = new Waiter();
84. $waiter->SetOrder($command1);
85. $waiter->SetOrder($command2);
86. $waiter->ConcelOrder($command2);
87. $waiter->SetOrder($command3);
88. $waiter->Notify();

**20职责链模式**

    类结构图

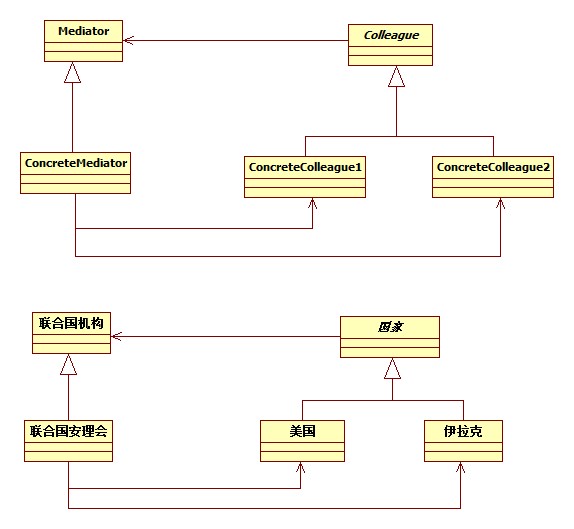


    PHP代码示例：

1. <?php
2. //职责链模式PHP代码示例
3. //申请类
4. class Request
5. {
6. public $requestType;
7. public $requestContent;
8. public $number;
9. }
10. //管理者类
11. abstract class Manager
12. {
13. protected $name;
14. protected $superior; //管理者的上级
15. public function \_\_construct($name)
16. {
17. $this->name = $name;
18. }
19. public function SetSuperior(Manager $superior)
20. {
21. $this->superior = $superior;
22. }
23. abstract public function RequestApplications(Request $request);
24. }
25. //经理
26. class CommonManager extends Manager
27. {
28. public function RequestApplications(Request $request)
29. {
30. if(($request->requestType == '请假') && ($request->number))
31. {
32. echo $this->name.':'.$request->requestContent.'数量'.$request->number." 被批准。<br />";
33. }
34. else
35. {
36. if($this->superior != null){
37. $this->superior->RequestApplications($request);
38. }
39. }
40. }
41. }
42. //总监
43. class Majordomo extends Manager
44. {
45. public function RequestApplications(Request $request)
46. {
47. if(($request->requestType == '请假') && ($request->number<=5))
48. {
49. echo $this->name.':'.$request->requestContent.'数量'.$request->number." 被批准。<br />";
50. }
51. else
52. {
53. if($this->superior != null){
54. $this->superior->RequestApplications($request);
55. }
56. }
57. }
58. }
59. //总经理
60. class GeneralManager extends Manager
61. {
62. public function RequestApplications(Request $request)
63. {
64. if($request->requestType == '请假')
65. {
66. echo $this->name.':'.$request->requestContent.'数量'.$request->number." 被批准。<br />";
67. }
68. elseif(($request->requestType == '加薪') && ($request->number <=500))
69. {
70. echo $this->name.':'.$request->requestContent.'数量'.$request->number." 被批准。<br />";
71. }
72. elseif(($request->requestType == '加薪') && ($request->number >500))
73. {
74. echo $this->name.':'.$request->requestContent.'数量'.$request->number." 再说吧。<br />";
75. }
76. }
77. }
78. $jinli = new CommonManager('秦');
79. $zongjian = new Majordomo('荣');
80. $zongjinli = new GeneralManager('王');
81. $jinli->SetSuperior($zongjian);
82. $zongjian->SetSuperior($zongjinli);
83. $request = new Request();
84. $request->requestType = '请假';
85. $request->requestContent = '小菜请假';
86. $request->number = 1;
87. $jinli->RequestApplications($request);
88. $request2 = new Request();
89. $request2->requestType = '请假';
90. $request2->requestContent = '小菜请假';
91. $request2->number = 4;
92. $jinli->RequestApplications($request2);
93. $request3 = new Request();
94. $request3->requestType = '加薪';
95. $request3->requestContent = '小菜请求加薪';
96. $request3->number = 500;
97. $jinli->RequestApplications($request3);
98. $request4 = new Request();
99. $request4->requestType = '加薪';
100. $request4->requestContent = '小菜请求加薪';
101. $request4->number = 1000;
102. $jinli->RequestApplications($request4);

**21中介者模式**

    类结构图

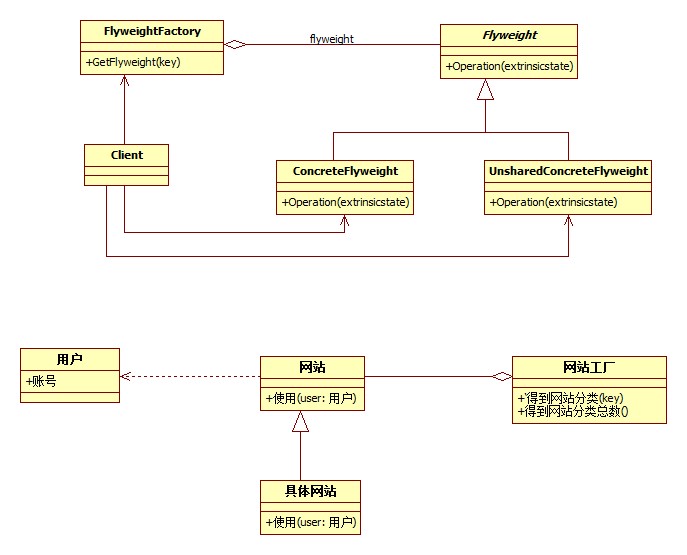


    PHP代码示例：

1. <?php
2. //中介者(Mediator)模式PHP代码示例
3. //联合国机构
4. abstract class UnitedNations
5. {
6. abstract function Declar($message,Country $colleague);
7. }
8. class Country
9. {
10. protected $mediator;
11. public function \_\_construct(UnitedNations $mediator)
12. {
13. $this->mediator = $mediator;
14. }
15. }
16. class USA extends Country
17. {
18. public function Declar($message)
19. {
20. $this->mediator->Declar($message,$this);
21. }
22. public function GetMessage($message)
23. {
24. echo '美国获得对方信息'.$message."<br />";
25. }
26. }
27. class Iraq extends Country
28. {
29. public function Declar($message)
30. {
31. $this->mediator->Declar($message,$this);
32. }
33. public function GetMessage($message)
34. {
35. echo '伊拉克获得对方信息'.$message."<br />";
36. }
37. }
38. //联合国安理会
39. class UnitedNationSecurityCouncil extends UnitedNations
40. {
41. private $colleague1;
42. private $colleague2;
43. public function SetColleague1(Country $colleague1)
44. {
45. $this->colleague1 = $colleague1;
46. }
47. public function SetColleague2(Country $colleague2)
48. {
49. $this->colleague2 = $colleague2;
50. }
51. public function Declar($message,Country $colleague)
52. {
53. if($colleague == $this->colleague1)
54. {
55. $this->colleague2-> GetMessage($message);
56. }
57. else
58. {
59. $this->colleague1->GetMessage($message);
60. }
61. }
62. }
63. $UNSC = new UnitedNationSecurityCouncil();
64. $usa = new USA($UNSC);
65. $iraq = new Iraq($UNSC);
66. $UNSC->SetColleague1($usa);
67. $UNSC->SetColleague2($iraq);
68. $usa->Declar('不准研制核武器，否则要发动战争!');
69. $iraq->Declar('我们没有核武器，也不怕侵略。');

**22享元模式**

    类结构图

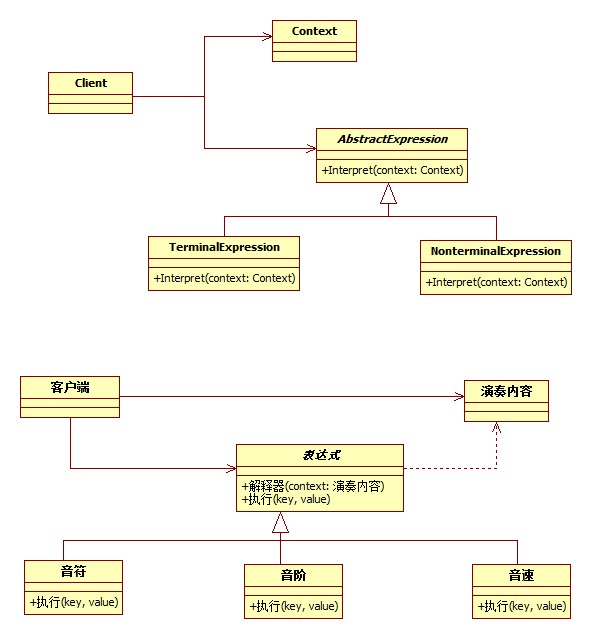


    PHP代码示例:

1. <?php
2. //享元(Flyweight)模式PHP代码示例
3. //用户类
4. class User
5. {
6. private $name;
7. public function \_\_construct($name)
8. {
9. $this->name = $name;
10. }
11. public function GetName()
12. {
13. return $this->name;
14. }
15. }
16. //网站抽象类
17. abstract class WebSite
18. {
19. abstract public function Create(User $user);
20. }
21. //具体网站类
22. class ConcreteWebSite extends WebSite
23. {
24. private $name;
25. public function \_\_construct($name)
26. {
27. $this->name = $name;
28. }
29. public function Create(User $user)
30. {
31. echo '网站分类'.$this->name.' 用户:'.$user->GetName()."<br />";
32. }
33. }
34. //网站工厂类
35. class WebSiteFactory
36. {
37. private $flyweight = array();
38. public function GetWebSiteCategory($key)
39. {
40. if(!key\_exists($key,$this->flyweight))
41. {
42. $this->flyweight[$key] = new ConcreteWebSite($key);
43. }
44. return $this->flyweight[$key];
45. }
46. public function GetWebSiteCount()
47. {
48. return count($this->flyweight);
49. }
50. }
51. $f = new WebSiteFactory();
52. $fx = $f->GetWebSiteCategory('产品展示');
53. $fx->Create(new User("小菜"));
54. $fy = $f->GetWebSiteCategory('产品展示');
55. $fy->Create(new User("大鸟"));
56. $fz = $f->GetWebSiteCategory('产品展示');
57. $fz->Create(new User("娇娇"));
58. $fl = $f->GetWebSiteCategory('博客');
59. $fl->Create(new User("老顽童"));
60. $fm = $f->GetWebSiteCategory('博客');
61. $fm->Create(new User("桃谷六仙"));
62. $fn = $f->GetWebSiteCategory('博客');
63. $fn->Create(new User("南海鳄神"));
64. echo '网站分类总数：'.$f->GetWebSiteCount();

**23解释器模式**

    类结构图

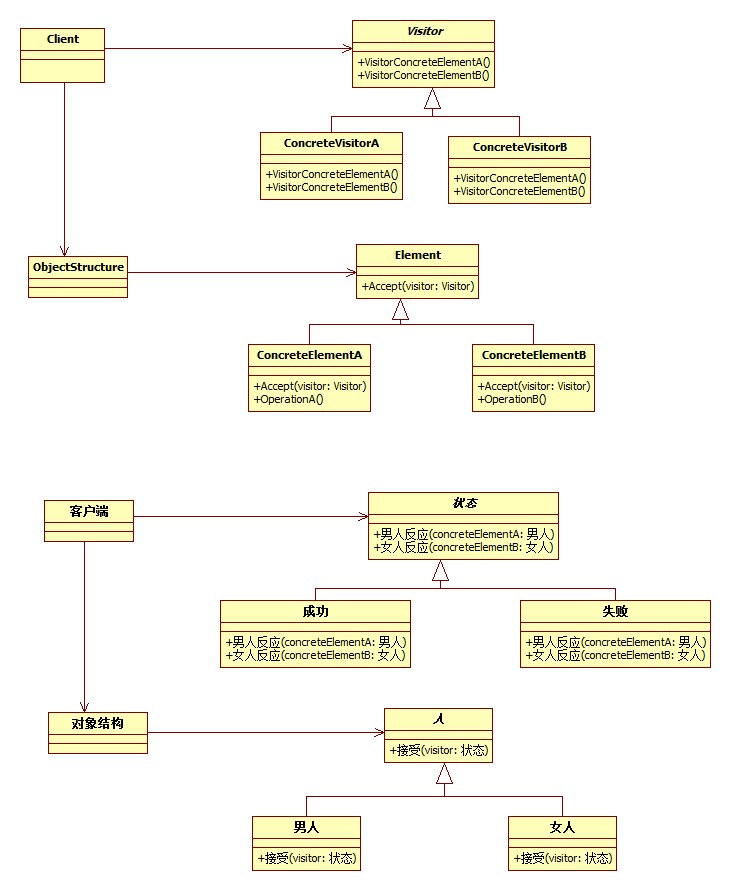


    PHP代码示例:

1. <?php
2. //解释器（interpreter）模式PHP代码示例
3. //演奏内容
4. class PlayContext
5. {
6. private $text;
7. public function SetText($text)
8. {
9. $this->text = $text;
10. }
11. public function GetText()
12. {
13. return $this->text;
14. }
15. }
16. //表达式类
17. abstract class Expression
18. {
19. public function Interpret(PlayContext $context)
20. {
21. if(strlen($context->GetText()) == 0){
22. return;
23. }
24. $playKey = substr($context->GetText(),0,1);
25. $context->SetText(substr($context->GetText(),1));
26. $playValue = (substr($context->GetText(),0,strpos($context->GetText(),' ')));
27. $context->SetText(substr($context->GetText(),strpos($context->GetText(),' ')+1));
28. $this->Excute($playKey,$playValue);
29. }
30. abstract public function Excute($key,$value);
31. }
32. //音符类
33. class Node extends Expression
34. {
35. public function Excute($key,$value)
36. {
37. $note = '';
38. switch($key)
39. {
40. case 'C':
41. $note = 1;
42. break;
43. case 'D':
44. $note = 2;
45. break;
46. case 'E':
47. $note = 3;
48. break;
49. case 'F':
50. $note = 4;
51. break;
52. case 'G':
53. $note = 5;
54. break;
55. case 'A':
56. $note = 6;
57. break;
58. case 'B':
59. $note = 7;
60. break;
61. }
62. echo $note;
63. }
64. }
65. //音阶类
66. class Scale extends Expression
67. {
68. public function Excute($key,$value)
69. {
70. $scale = '';
71. switch(intval($value))
72. {
73. case 1:
74. $scale = '低音';
75. break;
76. case 2:
77. $scale = '中音';
78. break;
79. case 3:
80. $scale = '高音';
81. break;
82. }
83. echo $scale;
84. }
85. }
86. $context = new PlayContext();
87. echo "上海滩:<br />";
88. $str = "O2 E0.5 G0.5 A3 E0.5 G0.5 D3 E0.5 G0.5 A0.5";
89. $context->SetText($str);
90. $expression = null;
91. try{
92. while(strlen($context->GetText()) > 0)
93. {
94. $str = substr($context->GetText(),0,1);
95. switch($str)
96. {
97. case 'O':
98. $expression = new Scale();
99. break;
100. case 'A':
101. case 'B':
102. case 'C':
103. case 'D':
104. case 'E':
105. case 'G':
106. case 'P':
107. $expression = new Node();
108. break;
109. }
110. $expression->Interpret($context);
111. }
112. }catch(Exception $e){
113. echo $e->getMessage();
114. }

**24访问者模式**

    类结构图



    PHP代码示例

1. <?php
2. //访问者模式PHP代码示例
3. //状态抽象类
4. abstract class Action
5. {
6. abstract public function GetManConclusion(Man $concreteElementA);
7. abstract public function GetWomanConclusion(Woman $concreteElementB);
8. }
9. //抽象人类
10. abstract class Person
11. {
12. abstract public function Accept(Action $visitor);
13. }
14. //成功
15. class Success extends Action
16. {
17. public function GetManConclusion(Man $concreteElementA)
18. {
19. echo $concreteElementA->getName().' '.\_\_CLASS\_\_."时，背后多半有一个伟大的女人。<br />";
20. }
21. public function GetWomanConclusion(Woman $concreteElementB)
22. {
23. echo $concreteElementB->getName().' '.\_\_CLASS\_\_."时，背后多半有一个不成功的男人。<br />";
24. }
25. }
26. //失败
27. class Failing extends Action
28. {
29. public function GetManConclusion(Man $concreteElementA)
30. {
31. echo $concreteElementA->getName().' '.\_\_CLASS\_\_."时，闷头喝酒谁也不用劝。<br />";
32. }
33. public function GetWomanConclusion(Woman $concreteElementB)
34. {
35. echo $concreteElementB->getName().' '.\_\_CLASS\_\_."时，眼泪汪汪，谁也劝不了。<br />";
36. }
37. }
38. //男人类
39. class Man extends Person
40. {
41. public function getName()
42. {
43. return \_\_CLASS\_\_;
44. }
45. public function Accept(Action $visitor)
46. {
47. $visitor->GetManConclusion($this);
48. }
49. }
50. class Woman extends Person
51. {
52. public function getName()
53. {
54. return \_\_CLASS\_\_;
55. }
56. public function Accept(Action $visitor)
57. {
58. $visitor->GetWomanConclusion($this);
59. }
60. }
61. class ObjectStructure
62. {
63. private $list = array();
64. public function Add(Person $element)
65. {
66. $this->list[] = $element;
67. }
68. public function Detach(Person $element)
69. {
70. $key = array\_search($element,$this->list);
71. unset($this->list[$key]);
72. }
73. public function Display(Action $visitor)
74. {
75. if($this->list)
76. {
77. foreach($this->list as $person)
78. {
79. $person->Accept($visitor);
80. }
81. }
82. }
83. }
84. $o = new ObjectStructure();
85. $o->Add(new Man());
86. $o->Add(new Woman());
87. $v1 = new Success();
88. $o->Display($v1);
89. $v2 = new Failing();
90. $o->Display($v2);

**25超级模式大赛（模式总结）**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 单例模式 | 创造型模式  （工厂方法模式） | 适配器模式 | 行为型模式一组  （观察者模式） | 观察者模式 |
| 工厂方法模式 | 模板方法模式 |
| 抽象工厂模式 | 命令模式 |
| 建造者模式 | 状态模式 |
| 原型模式 | 职责链模式 |
| 适配器模式 | 结构型模式  （外观模式） | 行为型模式二组  （策略模式） | 解释器模式 |
| 装饰模式 | 中介者模式 |
| 桥接模式 | 访问者模式 |
| 组合模式 | 策略模式 |
| 享元模式 | 备忘录模式 |
| 代理模式 | 迭代器模式 |
| 外观模式 |