**PHP设计模式之简单工厂和工厂模式实例分析**

 更新时间：2019年03月25日 10:48:58   作者：12345987

这篇文章主要介绍了PHP设计模式之简单工厂和工厂模式,结合实例形式分析了php设计模式中工厂模式的实现方法及相关操作注意事项,需要的朋友可以参考下

本文实例讲述了PHP设计模式之简单工厂和工厂模式。分享给大家供大家参考，具体如下：

**工厂模式是创建型模式的一种，分为简单工厂模式，工厂模式，抽象工厂模式，简单工厂可以称之为工厂模式的一个特例**

先用一段简单的代码说明一下，现在假设我们是顾客，需要苹果味饮料和香蕉味饮料

[?](https://www.jb51.net/article/158403.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | <?php  class AppleDrink{    function getDrinkName()    {      echo '苹果饮料';    }  }  class BananaDrink{    function getDrinkName()    {      echo '香蕉饮料';    }  }  //顾客1  $apple = new AppleDrink();  $apple->getDrinkName();  echo "<br/>";  $banana = new BananaDrink();  $banana->getDrinkName();  echo "<br/>";  //顾客2  $apple1 = new AppleDrink();  $apple1->getDrinkName();  echo "<br/>";  $banana1 = new BananaDrink();  $banana1->getDrinkName(); |



运行结果：

苹果饮料  
香蕉饮料  
苹果饮料  
香蕉饮料

这是最基本的写法，顾客和饮料是强耦合关系，也就是咱们平常说的硬编码

有一天，老板觉得AppleDrink这个名字在国内不好卖，想要改成Xingmu（醒目）这种接中国地气的名字（麦当劳改成金拱门）

你会发现，需要去代码中找到每一处NEW的地方，全部改成new Xingmu()

如果它还有其他的初始化步骤，就更是噩梦，很可能会引起一些不必要的麻烦

下面咱们用简单工厂模式，更改一下这个文件

[?](https://www.jb51.net/article/158403.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  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 | <?php  class AppleDrink{    function getDrinkName()    {      echo '苹果饮料';    }  }  class BananaDrink{    function getDrinkName()    {      echo '香蕉饮料';    }  }  class FruitFactory{    function makeDrink($fruit){      if ($fruit == 'apple'){        return new AppleDrink();      }elseif ($fruit == 'banana'){        return new BananaDrink();      }    }  }  $factory = new FruitFactory();  $apple = $factory->makeDrink('apple');  $apple->getDrinkName();  echo "<br/>";  $banana = $factory->makeDrink('banana');  $banana->getDrinkName();  echo "<br/>";  $apple1 = $factory->makeDrink('apple');  $apple1->getDrinkName();  echo "<br/>";  $banana1 = $factory->makeDrink('banana');  $banana1->getDrinkName(); |



运行结果：

苹果饮料  
香蕉饮料  
苹果饮料  
香蕉饮料

现在我们发现，假如说老板想改名字，我只需要把FruitFactory里的new AppleDrink改为new Xingmu()即可，其他地方都不需要改动，也不用翻代码找哪些地方用new了，实现了顾客和饮料之间的解耦，也符合面向对象的设计思想，我只是想要一瓶饮料，我并不需要了解这个饮料是怎么制造出来的

**这就是简单工厂模式，用户在使用时可以直接根据工厂类去创建所需的实例，而无需了解这些对象是如何创建以及如何组织的，外界与具体类隔离开来，耦合性低，有利于整个软件体系结构的优化，适用于工厂类负责创建的对象比较少，客户只知道传入了工厂类的参数，对于如何创建对象（逻辑）不关心，简单工厂模式又叫静态工厂模式 可以把工厂类的方法写成静态方法 在不需要实例化工厂的前提下 直接调用静态方法 返回所需实例**

好的，那么接下来，问题又来了，公司准备多元化产品，增加桔子味饮料，那么我们需要怎么做呢，首先要增加桔子味饮料类，然后需要在工厂里增加判断，当水果标识是orange的时候，返回桔子味饮料

[?](https://www.jb51.net/article/158403.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | class OrangeDrink{    function getDrinkName()    {      echo '桔子味饮料';    }  }  class FruitFactory{    function makeDrink($fruit){      if ($fruit == 'apple'){        return new AppleDrink();      }elseif ($fruit == 'banana'){        return new BananaDrink();      }elseif ($fruit == 'orange'){        return new OrangeDrink();      }    }  } |



然后当以后每次需要增加新的产品时，我们都需要更改工厂文件，当对象生成复杂的时候，这个工厂文件会越来越大，更改或许会引起一些意想不到的问题

面向对象的设计原则，对扩展开放，对更改关闭，那么有没有办法，在不更改原有代码的基础上，增加产品呢

答案是有的，咱们再改写一下这个方法

[?](https://www.jb51.net/article/158403.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  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 | <?php  interface Drink{    function getDrinkName();  }  class AppleDrink implements Drink{    function getDrinkName()    {      echo '苹果味饮料';    }  }  class BananaDrink implements Drink{    function getDrinkName()    {      echo '香蕉味饮料';    }  }  interface FruitFactory{    function makeDrink();  }  class AppleFactory implements FruitFactory{    function makeDrink()    {      return new AppleDrink();    }  }  class BananaFactory implements FruitFactory{    function makeDrink()    {      return new BananaDrink();    }  }  $appleFactory = new AppleFactory();  $apple = $appleFactory->makeDrink();  $apple->getDrinkName();  echo "<br/>";  $bananaFactory = new BananaFactory();  $banana = $bananaFactory->makeDrink();  $banana->getDrinkName(); |



运行结果：

苹果味饮料  
香蕉味饮料

现在当再次需要增加桔子味饮料时，只需要增加桔子味饮料产品和桔子味饮料工厂即可，不需要改动原来的代码

[?](https://www.jb51.net/article/158403.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class OrangeDrink implements Drink{    function getDrinkName()    {      echo '桔子味饮料';    }  }  class OrangeFactory implements FruitFactory{    function makeDrink()    {      return new OrangeDrink();    }  } |



**这就是工厂模式，它是简单工厂模式的衍生，解决了许多简单工厂模式的问题。首先完全实现开闭原则，实现了对扩展开放，对更改关闭。其次实现更复杂的层次结构，可以应用于产品结果复杂的场合。工厂方法模式是对简单工厂模式进行了抽象。有一个抽象的Factory类（可以是抽象类和接口），这个类将不在负责具体的产品生产，而是只制定一些规范，具体的生产工作由其子类去完成。在这个模式中，工厂类和产品类往往可以依次对应。即一个抽象工厂对应一个抽象产品，一个具体工厂对应一个具体产品，这个具体的工厂就负责生产对应的产品。**

**总结：**

无论是简单工厂模式，工厂方法模式，还是抽象工厂模式，他们都属于工厂模式，在形式和特点上也是极为相似的，他们的最终目的都是为了解耦。在使用时，我们不必去在意这个模式到底工厂方法模式还是抽象工厂模式，因为他们之间的演变常常是令人琢磨不透的。经常你会发现，明明使用的工厂方法模式，当新需求来临，稍加修改，加入了一个新方法后，由于类中的产品构成了不同等级结构中的产品族，它就变成抽象工厂模式了；而对于抽象工厂模式，当减少一个方法使的提供的产品不再构成产品族之后，它就演变成了工厂方法模式。所以，在使用工厂模式时，只需要关心降低耦合度的目的是否达到了