



[🏠](#) / [设计模式](#) / [创建型模式](#)

工厂模式比较

本文将对下列概念之间的差异进行说明：

1. 工厂
2. 构建方法
3. 静态构建（或工厂）方法
4. 简单工厂
5. 工厂方法
6. 抽象工厂

你可以在网上找到这些术语的参考信息。尽管它们看上去相似，但其含义都不一样。许多人没有意识到这一点，从而出现了混淆和误解。

因此让我们搞清楚其中的不同之处，一劳永逸地解决这个问题。

1. 工厂

工厂是一个含义模糊的术语，表示可以创建一些东西的函数、方法或类。最常见的情况下，工厂创建的是对象。但是它们也可以创建文件和数据库记录等其他东西。

例如，下面这些东西都可以非正式地被称为“工厂”：

- 创建程序 GUI 的函数或方法；
- 创建用户的类；
- 以特定方式调用类构造函数的静态方法。
- 一种创建型设计模式。

当某人说到“工厂”这个词时，其具体含义通常可以根据上下文来确定。但如果你有疑问，可以直接提问。毕竟作者本人有时候也没有搞清楚。

2. 构建方法

构建方法在《[重构与模式](#)》中被定义为“创建对象的方法”。这意味着每个工厂方法模式的结果都是“构建方法”，但反过来则并非如此。这也意味着你可以用“构建方法”来替代马丁·福勒在[重构](#)中使用的“工厂方法”和乔斯华·布洛克在《[Effective Java](#)》中使用的“静态工厂方法”。

✓ 在实际中，构建方法只是构造函数调用的封装器。它可能只是一个能更好地表达意图的名称。此外，它可以让你的代码独立于构造函数的改动，甚至还可以包含一些特殊的逻辑，返回已有对象而不是创建新对象。

许多人会仅仅因为这些方法创建了新对象而称之为“工厂方法”。其中的逻辑很直接：所有的工厂都会创建对象，而该方法会创建对象，所以显然它是一个工厂方法。当遇到真正的[工厂方法](#)时，这自然会造成许多混淆。

在下面的示例中，`next` 是一个构建方法：

```
class Number {
    private $value;

    public function __construct($value) {
        $this->value = $value;
    }

    public function next() {
        return new Number ($this->value + 1);
    }
}
```

3. 静态构建方法

静态构建方法是被声明为 `static` 的构建方法。换句话说，你无需创建对象就能在某个类上调用该方法。

不要因为某些人将这些方法称为“静态工厂方法”而被其迷惑。这种称呼只是一个坏习惯。[工厂方法](#)是一种依赖于继承的设计模式。如果将它设置为 `static`，你就不能在子类中对其进行扩展，这就破坏了该模式的目的。

当静态构建方法返回一个新对象时，它就成为了构造函数的替代品。


在下列情况中，这可能会非常实用：

- 你必须针对不同的目的提供多个不同的构造函数，但是其签名相同时。例如，在 Java、C++、C# 以及其他许多语言中不可能同时存在 `Random(int max)` 和 `Random(int min)` 函数。最常用的变通方式是创建多个调用默认构造函数的静态方法，并于稍后再设置适当的数值。
- 你希望复用已有对象而不是初始化新对象时（参考单例模式）。绝大多数编程语言的构造函数都必须都返回一个新的类实例。静态构建方法是应对该限制的变通方法。在静态方法内部，你的代码会决定是调用构造函数创建一个全新实例，还是返回一个在缓存中已有的对象。

在下面的例子中，`load` 方法是一个静态构建方法。它提供了一种从数据库中获取用户的灵活方式。

```
class User {  
    private $id, $name, $email, $phone;  
  
    public function __construct($id, $name, $email, $phone) {  
        $this->id = $id;  
        $this->name = $name;  
        $this->email = $email;  
        $this->phone = $phone;  
    }  
  
    public static function load($id) {  
        list($id, $name, $email, $phone) = DB::load_data('users', 'id', 'name', 'email', 'phone');  
        $user = new User($id, $name, $email, $phone);  
        return $user;  
    }  
}
```

4. 简单工厂模式

简单工厂模式  描述了一个类，它拥有一个包含大量条件语句的构建方法，可根据方法的参数来选择对何种产品进行初始化并将其返回。

人们通常会将简单工厂与普通的工厂或其它创建型设计模式混淆。在绝大多数情况下，简单工厂是引入工厂方法或抽象工厂模式时的一个中间步骤。

简单工厂通常没有子类。但当从一个简单工厂中抽取出子类后，它看上去就会更像经典的工厂方法模式了。

顺便提一句，如果你将一个简单工厂声明为 `abstract` 类型，它并不会神奇地变成抽象工厂模式。

这里是一个简单工厂的例子：

```
class UserFactory {
    public static function create($type) {
        switch ($type) {
            case 'user': return new User();
            case 'customer': return new Customer();
            case 'admin': return new Admin();
            default:
                throw new Exception('传递的用户类型错误。');
        }
    }
}
```

5. 工厂方法模式

工厂方法 ①是一种创建型设计模式，其在父类中提供一个创建对象的方法，允许子类决定实例化对象的类型。

如果在基类及其扩展的子类中都有一个构建方法的话，那它可能就是工厂方法。


```
abstract class Department {
    public abstract function createEmployee($id);

    public function fire($id) {
        $employee = $this->createEmployee($id);
        $employee->paySalary();
        $employee->dismiss();
    }
}

class ITDepartment extends Department {
    public function createEmployee($id) {
        return new Programmer($id);
    }
}

class AccountingDepartment extends Department {
    public function createEmployee($id) {
        return new Accountant($id);
    }
}
```

6. 抽象工厂模式

抽象工厂  是一种创建型设计模式，它能创建一系列相关或相互依赖的对象，而无需指定其具体类。

什么是“系列对象”？例如有这样一组对象： 运输工具 + 引擎 + 控制器 。它可能会有几个变体：

1. 汽车 + 内燃机 + 方向盘
2. 飞机 + 喷气式发动机 + 操纵杆

如果你的程序中并不涉及产品系列的话，那就不需要抽象工厂。

再次重申，许多人分不清抽象工厂模式和声明为 `abstract` 的简单工厂。不要犯这个错误！

后记

现在你知道了它们之间的区别，试试用全新眼光看待下面的设计模式吧：

- 工厂方法
- 抽象工厂

主页



重构



设计模式

会员专属内容

论坛

联系我们

© 2014-2020 Refactoring.Guru. 版权所有

📍 Khmelnytske shosse 19 / 27, Kamianets-Podilskyi, 乌克兰, 32305

✉ Email: support@refactoring.guru

📷 图片作者：Dmitry Zhart

条款与政策

隐私政策

内容使用政策