# 第 9 章

# 工 厂 模 式

在面向对象的程序中，我曾经见过的最大的 switch/case 语句所具有的条件超过了 20 个。在运行期间，这个代码模块每次都要执行所有条件。每个条件都负责创建一个新的对象，这个对象被用于与应用程序 API 的外部使用者进行通信。对其中一个执行某些常规的故障处理之后，我开始研究应用程序所实现的接口。令我吃惊的是，switch/case 语句中引用的每个类都在实现相同的接口！随后要在 switch/case 语句中添加新的条件语句时，我建议转而使用工厂模式。正如前面所介绍的，设计模式的名称非常重要，它们不仅提供了引用每种设计模式的一致性，而且是模式实际作用的关键所在。在上面所介绍的代码中，工厂设计模式是最适用的。

**名称：工厂**

工厂设计模式提供获取某个对象的新实例的一个接口，同时使调用代码避免确定实际实例化基类的步骤。

## 9.1 问题与解决方案

随着 PHP 发展并演变为一种语言，它的功能不断地通过应用经过验证的设计模式来提供更简单的开发手段。PHP 中一个特别有用的功能是能够创建基于变量内容的类的新实例。这种对象实例化的动态方法是 PHP 中实现工厂设计模式的一种代码块构建途径。

基于工厂设计模式的类有助于减少主代码流中基于条件的复杂性。在整个应用程序中，调用对象的方式众多而且不同。与某个对象创建相关的任何修改都会影响到应用程序的其余部分。假设实例化 5 个对象中的其中一个，从而完成某种功能性。创建条件语句来判断实例化的对象是一种方法。条件语句可能是复杂的 if/else 语句或 switch/case 语句。这种功能性可以在应用程序中的许多地方使用，但是会导致重复的代码。接下来，如果添加

第六个对象或修改现有 5 个对象其中一个的名称，那么程序代码的所有实例都需要被修改和再次测试。通过提供一个创建上述对象的简单接口，工厂设计模式有助于避免这个令人头痛的问题。无论对象被修改或者添加了其他对象，Factory 对象的调用方式仍然是相同的。

显示博客条目就是现实中的一个实例。个性化的博客非常流行，并且具有向使用者提供其内容的多种方式，包括标准的 Web 浏览器、RSS feed、移动传输以及 REST API。实际检索适当博客项的代码流或控制器并不需要关心具体使用的视图，而只是从视图创建工厂中请求新的视图对象。一旦具有该视图对象的实例，文章对象就会被传递至视图内。最后，代码流会调用执行视图对象的渲染对象。在上述整个过程中，由于使用了工厂对象，所以主代码流不必处理找出待创建视图对象的问题，而是笼统地调用工厂对象并提供正确的对象进行处理。

然而，不同对象的创建并不是使用工厂设计模式的唯一目的。使用基于工厂设计模式的类的另一个场合是处理若干项的集合。在这种情况下，对象集合包含相同的基对象，但是每个对象都具有不同的特征。

库存系统是使用工厂类管理对象集合的一个优秀实例。某个音乐店可能具有显示其吉他库存的应用系统。具体的视图最初被创建用于处理单个吉他对象，以便确定其品牌、型号、颜色和字符弦数。为了显示库存中的多个结果，我们可以使用一个 Guitars 工厂对象，该对象接受来自数据库的实例化 ID 集合，随后使用名为 getGuitar( )的公共方法返回根据单个 ID 创建的吉他对象。此时，工厂对象能够根据 ID 集合不断地创建新的吉他对象，并且使用一个公共方法统一地返回这些吉他对象。

在需要若干步骤才能确定要创建的对象类型时，我们最好使用基于工厂设计模式的类来检索新的实例。

## 9.2 UML

如图 9-1 所示，该 UML 图详细说明了一个使用工厂设计模式的类设计。

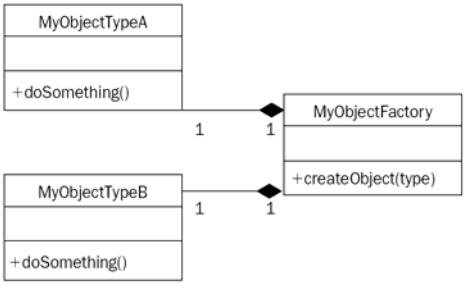


图 9-1

下面是对图 9-1 的说明：

- 现存的两个基类是 MyObjectTypeA 和 MyObjectTypeB。这两个类都具有名为 doSomething( )的公共方法，该方法采用自己独特的方式执行具体对象的逻辑。两个基类的公共接口和返回类型是完全相同的。
- MyObjectFactory 类用于创建上述任意一个基类的实例并将其返回至代码流。它具有一个名为 createObject( )的公共方法，该方法接受参数 type，这有助于判断应当创建哪一个基类的实例。随后，createObject( )方法会返回被请求类型类的一个实例。

## 9.3　代码示例

为了管理控制 CD，应用程序需要将必要的信息编辑入 CD 对象。将 CD 对象传递给外部供应商，这些人会完成实际的 CD 创建工作。CD 对象需要包含标题、乐队名称以及曲目列表。

如下所示，简单的 CD 类包含添加标题、乐队名称和曲目列表的方法。

```
class CD
{
    public $title = '';
    public $band = '';
    public $tracks = array();

    public function __construct()
    {}

    public function setTitle($title)
    {
        $this->title = $title;
    }

    public function setBand($band)
    {
        $this->band = $band;
    }

    public function addTrack($track)
    {
        $this->tracks[] = $track;
    }
}
```

为了创建完整的 CD 对象，处理过程总是相同的：首先创建 CD 类的一个实例，然后添加标题、乐队名称和曲目列表。

```
$title = 'Waste of a Rib';
$band = 'Never Again';
$tracksFromExternalSource = array('What It Means', 'Brrr', 'Goodbye');

$cd = new CD();
$cd->setTitle($title);
$cd->setBand($band);
foreach ($tracksFromExternalSource as $track) {
    $cd->addTrack($track);
}
```

如今，某些艺术家在他们的 CD 上发布了在计算机中能够使用的其他内容。这些 CD 称为增强型 CD。写至光盘的第一个音轨是数据音轨。管理控制软件通过其标签 DATA TRACK 识别数据音轨，并且创建相应的 CD 对象。

```
class enhancedCD
{
    public $title = '';
    public $band = '';
    public $tracks = array();

    public function __construct()
    {
        $this->tracks[] = 'DATA TRACK';
    }

    public function setTitle($title)
    {
        $this->title = $title;
    }

    public function setBand($band)
    {
        $this->band = $band;
    }

    public function addTrack($track)
    {
        $this->tracks[] = $track;
    }
}
```

查看上述共性和认识到只可能存在两种 CD 类型之后，似乎我们只需要创建条件语句。如果 CD 类型是增强型 CD，那么就创建 enhancedCD 类的新实例；否则，就应创建通用的 CD 类。然而，还存在更好的解决方案：使用工厂设计模式。

CDFactory 类使用了 PHP 根据变量动态实例化一个类的能力。create( )方法接受被请求类的类型并返回该类的一个实例：

```
class CDFactory
{
    public static function create($type)
    {
        $class = strtolower($type) . "CD";

        return new $class;
    }
}
```

现在，类的创建和执行的变化反映了 Factory 类的用法：

```
$type = 'enhanced';

$cd = CDFactory::create($type);
$cd->setBand($band);
$cd->setTitle($title);
foreach ($tracksFromExternalSource as $track) {
    $cd->addTrack($track);
}
```

最后需要考虑的可能是已有 CD 类的名称。为了使其统一，将类名改变为 standardCD 也许更有实际意义。确认这样不会破坏代码中其他位置的其他功能性。此时，我们最好将 CD 的新实例修改为使用 CDFactory 类。

请求需要某些逻辑和步骤才能确定基对象的类实例时，最佳的做法是使用一个基于工厂设计模式的类。