# 享元模式

享元模式（Flyweight Pattern）主要用于减少创建对象的数量，以减少内存占用和提高性能。这种类型的设计模式属于结构型模式，它提供了减少对象数量从而改善应用所需的对象结构的方式。

享元模式尝试重用现有的同类对象，如果未找到匹配的对象，则创建新对象。我们将通过创建 5 个对象来画出 20 个分布于不同位置的圆来演示这种模式。由于只有 5 种可用的颜色，所以 color 属性被用来检查现有的 *Circle* 对象。

## 介绍

**意图：**运用共享技术有效地支持大量细粒度的对象。

**主要解决：**在有大量对象时，有可能会造成内存溢出，我们把其中共同的部分抽象出来，如果有相同的业务请求，直接返回在内存中已有的对象，避免重新创建。

**何时使用：**

1. 系统中有大量对象。
2. 这些对象消耗大量内存。

3、这些对象的状态大部分可以外部化。

4、这些对象可以按照内蕴状态分为很多组，当把外蕴对象从对象中剔除出来时，每一组对象都可以用一个对象来代替。

5、系统不依赖于这些对象身份，这些对象是不可分辨的。

**如何解决：**用唯一标识码判断，如果在内存中有，则返回这个唯一标识码所标识的对象。

**关键代码：**用 HashMap 存储这些对象。

**应用实例：**

1. JAVA 中的 String，如果有则返回，如果没有则创建一个字符串保存在字符串缓存池里面。

2、数据库的数据池。

**优点：**大大减少对象的创建，降低系统的内存，使效率提高。

**缺点：**提高了系统的复杂度，需要分离出外部状态和内部状态，而且外部状态具有固有化的性质，不应该随着内部状态的变化而变化，否则会造成系统的混乱。

**使用场景：** 1、系统有大量相似对象。 2、需要缓冲池的场景。

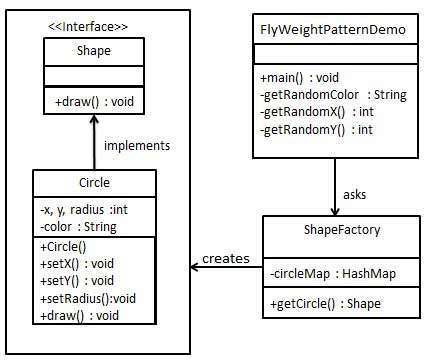
**注意事项：** 1、注意划分外部状态和内部状态，否则可能会引起线程安全问题。 2、这些类必须有一个工厂对象加以控制。

## 实现

我们将创建一个 *Shape* 接口和实现了 *Shape* 接口的实体类 *Circle*。下一步是定义工厂类 *ShapeFactory*。

*ShapeFactory* 有一个 *Circle* 的 *HashMap*，其中键名为 *Circle* 对象的颜色。无论何时接收到请求，都会创建一个特定颜色的圆。*ShapeFactory* 检查它的 *HashMap* 中的 circle 对象，如果找到 *Circle* 对象，则返回该对象，否则将创建一个存储在 hashmap 中以备后续使用的新对象，并把该对象返回到客户端。

*FlyWeightPatternDemo*，我们的演示类使用 *ShapeFactory* 来获取 *Shape* 对象。它将向 *ShapeFactory* 传递信息（*red / green / blue/ black / white*），以便获取它所需对象的颜色。



**优缺点**

享元模式优点就在于它能够大幅度的降低内存中对象的数量；而为了做到这一步也带来了它的缺点：它使得系统逻辑复杂化，而且在一定程度上外蕴状态影响了系统的速度。

所以一定要切记使用享元模式的条件：

1)        系统中有大量的对象，他们使系统的效率降低。

2)        这些对象的状态可以分离出所需要的内外两部分。

外蕴状态和内蕴状态的划分以及两者关系的对应也是非常值得重视的。只有将内外划分妥当才能使内蕴状态发挥它应有的作用；如果划分失误，在最糟糕的情况下系统中的对象是一个也不会减少的！两者的对应关系的维护和查找也是要花费一定的空间（当然这个比起不使用共享对象要小得多）和时间的，可以说享元模式就是使用时间来换取空间的。在Gof的书中是使用了B树来进行对应关系查找优化。

**总结**  
      也许你要长叹一声：这个享元模式未必太复杂了吧！这点是不得不承认的，也许由于它的复杂，实际应用也不是很多，这是我们更加无法看清他的真面目了。不过享元模式并不是鸡肋，它的精髓——共享是对我们系统优化非常有好处的，而且这种思想已经别越来越多的应用，这应该就算是享元模式的应用了吧。如果你已经领会到了享元模式的精髓，那么也就是掌握了享元模式了!