# 享元模式

## 问题引入

在面向对象程序设计过程中，有时会面临要创建大量相同或相似对象实例的问题。创建那么多的对象将会耗费很多的系统资源，它是系统性能提升的一个瓶颈。这些对象有很多相似的地方，如果能把它们相同的部分抽取出来共享，那将大量节省系统资源，这就是享元模式产生的背景。

享元，即共享对象。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的对象。

说到享元模式，第一个想到的应该就是池技术了，String常量池、数据库连接池、缓冲池等等都是享元模式的应用，所以说享元模式是池技术的重要实现方式。

比如我们每次创建字符串对象时，都需要创建一个新的字符串对象的话，内存开销会很大，所以如果第一次创建了字符串对象“adam“，下次再创建相同的字符串”adam“时，只是把它的引用指向”adam“，这样就实现了”adam“字符串再内存中的共享。

## 实例

## 定义

利用共享技术，有效地支持大量细粒度对象的复用。通过共享已经存在的对象来大幅减少需要创建的对象数量，避免创建大量相似类带来的开销，从而提高系统资源利用率。

## 优点

享元模式的优点：

1. 大大减少同类对象的创建，减少系统中对象的数量，降低对象创建带来的系统开销以及内存压力。

## 缺点

享元模式的缺点：

1. 系统复杂度提高，维护难度提高。（为了使对象可以共享，需要将一些不能共享的状态外部化，从而造成系统复杂度提高）
2. 读取外部状态会使得运行时间稍微变长。

## 结构

享元模式存在以下两种状态：

1. 内部状态。内部状态即不会随环境改变的可共享部分。
2. 外部状态。外部状态指随环境改变的不可共享的部分。

我们举一个最简单的例子，棋牌类游戏大家都有玩过吧，比如说说围棋和跳棋，它们都有大量的棋子对象，围棋和五子棋只有黑白两色，跳棋颜色略多一点，但也是不太变化的，所以棋子颜色就是棋子的内部状态；而各个棋子之间的差别就是位置的不同，我们落子嘛，落子颜色是定的，但位置是变化的，所以方位坐标就是棋子的外部状态。那么为什么这里要用享元模式呢？可以想象一下，上面提到的棋类游戏的例子，比如围棋，理论上有361个空位可以放棋子，常规情况下每盘棋都有可能有两三百个棋子对象产生，因为内存空间有限，一台服务器很难支持更多的玩家玩围棋游戏，如果用享元模式来处理棋子，那么棋子对象就可以减少到只有两个实例，这样就很好的解决了对象的开销问题。

享元模式存在以下角色划分：

1. 抽象享元角色：所有具体享元类的基类，为具体享元规范要实现的公共接口，非共享的外部状态以参数的形式通过方法传入。
2. 具体享元角色：实现抽象享元角色所规定的所有接口。
3. 非享元角色：即不共享的部分，以参数的形式通过方法传入。
4. 享元工厂角色：负责创建和管理享元角色。当客户端请求一个享元对象时，享元工厂先检查工厂中是否已经存在符合要求的享元对象，如果有则直接提供给客户端，如果没有则先创建一个对象，然后返回给客户端。

## 实现要领

享元模式的实现要领就是区分应用中的内部状态和外部状态，并将外部状态外部化（作为参数传入）。

## 应用场景

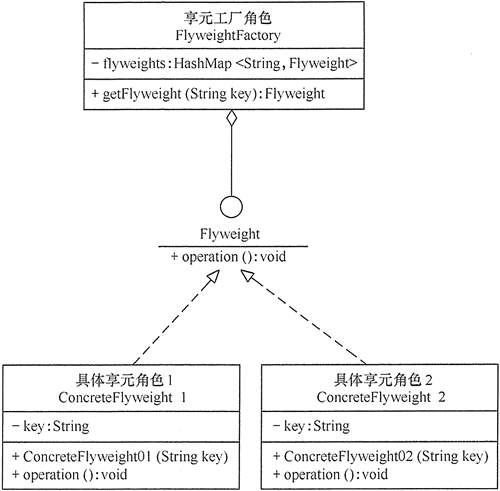
享元模式通过减少内存中对象的数量来节省内存空间的消耗，所以以下几种情形适合采用享元模式：

1. 系统中存在大量相同或相似的对象，这些对象消耗大量的内存资源。
2. 大部分的对象可以按照内部状态进行分组，且可将不同部分外部化，这样每一个组只需保存一个内部状态。
3. 由于享元模式需要额外维护一个保存享元的数据结构，所以应当在有足够多的享元实例时才值得使用享元模式。

## 拓展

1. 单纯享元模式

这种享元模式中的所有的具体享元类都是可以共享的，不存在非共享的具体享元类，其结构图如图



1. 复合享元模式

这种享元模式中的有些享元对象是由一些单纯享元对象组合而成的，它们就是复合享元对象。虽然复合享元对象本身不能共享，但它们可以分解成单纯享元对象再被共享，其结构图如图

