**设计模式之Flyweight(享元) FlyWeight模式**

[**板桥里人**](http://www.jdon.com/aboutme.htm) http://www.jdon.com 2002/05/02

**Flyweight模式定义:**  
避免大量拥有相同内容的小类的开销(如耗费内存),使大家共享一个类(元类).

**为什么使用?**  
面向对象语言的原则就是一切都是对象,但是如果真正使用起来,有时对象数可能显得很庞大,比如,字处理软件,如果以每个文字都作为一个对象,几千个字,对象数就是几千,无疑耗费内存,那么我们还是要"求同存异",找出这些对象群的共同点,设计一个元类,封装可以被共享的类,另外,还有一些特性是取决于应用(context),是不可共享的,这也Flyweight中两个重要概念内部状态intrinsic和外部状态extrinsic之分.

说白点,就是先捏一个的原始模型,然后随着不同场合和环境,再产生各具特征的具体模型,很显然,在这里需要产生不同的新对象,所以Flyweight模式中常出现Factory模式.Flyweight的内部状态是用来共享的,Flyweight factory负责维护一个Flyweight pool(模式池)来存放内部状态的对象.

Flyweight模式是一个提高程序效率和性能的模式,会大大加快程序的运行速度.应用场合很多:比如你要从一个数据库中读取一系列字符串,这些字符串中有许多是重复的,那么我们可以将这些字符串储存在Flyweight池(pool)中.

**如何使用?**

我们先从Flyweight抽象接口开始:

|  |
| --- |
| public interface Flyweight  { 　　public void operation( ExtrinsicState state ); }  //用于本模式的抽象数据类型(自行设计) public interface ExtrinsicState { } |

下面是接口的具体实现(ConcreteFlyweight) ,并为内部状态增加内存空间, ConcreteFlyweight必须是可共享的,它保存的任何状态都必须是内部(intrinsic),也就是说,ConcreteFlyweight必须和它的应用环境场合无关.

|  |
| --- |
| public class ConcreteFlyweight implements Flyweight { 　　private IntrinsicState state;  　　 　　public void operation( ExtrinsicState state )  　　{  　　　　　　//具体操作 　　}  } |

当然,并不是所有的Flyweight具体实现子类都需要被共享的,所以还有另外一种不共享的ConcreteFlyweight:

|  |
| --- |
| public class UnsharedConcreteFlyweight implements Flyweight {  　　public void operation( ExtrinsicState state ) { }  } |

Flyweight factory负责维护一个Flyweight池(存放内部状态),当客户端请求一个共享Flyweight时,这个factory首先搜索池中是否已经有可适用的,如果有,factory只是简单返回送出这个对象,否则,创建一个新的对象,加入到池中,再返回送出这个对象.池

|  |
| --- |
| public class FlyweightFactory {  　　//Flyweight pool 　　private Hashtable flyweights = new Hashtable();   　　public Flyweight getFlyweight( Object key ) {  　　　　Flyweight flyweight = (Flyweight) flyweights.get(key);  　　　　if( flyweight == null ) { 　　　　　　//产生新的ConcreteFlyweight 　　　　　　flyweight = new ConcreteFlyweight();  　　　　　　flyweights.put( key, flyweight );  　　　　}  　　　　return flyweight;  　　}  } |

至此,Flyweight模式的基本框架已经就绪,我们看看如何调用:

FlyweightFactory factory = new FlyweightFactory();   
Flyweight fly1 = factory.getFlyweight( "Fred" );   
Flyweight fly2 = factory.getFlyweight( "Wilma" );  
......

从调用上看,好象是个纯粹的Factory使用,但奥妙就在于Factory的内部设计上.

**Flyweight模式在XML等数据源中应用**  
我们上面已经提到,当大量从数据源中读取字符串,其中肯定有重复的,那么我们使用Flyweight模式可以提高效率,以唱片CD为例,在一个XML文件中,存放了多个CD的资料.

每个CD有三个字段:  
1.出片日期(year)  
2.歌唱者姓名等信息(artist)  
3.唱片曲目 (title)

其中,歌唱者姓名有可能重复,也就是说,可能有同一个演唱者的多个不同时期 不同曲目的CD.我们将"歌唱者姓名"作为可共享的ConcreteFlyweight.其他两个字段作为UnsharedConcreteFlyweight.

首先看看数据源XML文件的内容:

|  |
| --- |
| <?xml version="1.0"?> <collection>  <cd> <title>Another Green World</title> <year>1978</year> <artist>Eno, Brian</artist> </cd>  <cd> <title>Greatest Hits</title> <year>1950</year> <artist>Holiday, Billie</artist> </cd>  <cd> <title>Taking Tiger Mountain (by strategy)</title> <year>1977</year> <artist>Eno, Brian</artist> </cd>  .......  </collection> |

虽然上面举例CD只有3张,CD可看成是大量重复的小类,因为其中成分只有三个字段,而且有重复的(歌唱者姓名).

CD就是类似上面接口 Flyweight:

|  |
| --- |
| public class CD {  　　private String title; 　　private int year; 　　private Artist artist;  　　public String getTitle() {　　return title;　} 　　public int getYear() {　　　　return year;　　} 　　public Artist getArtist() {　　　　return artist;　　}  　　public void setTitle(String t){　　　　title = t;} 　　public void setYear(int y){year = y;} 　　public void setArtist(Artist a){artist = a;}  } |

将"歌唱者姓名"作为可共享的ConcreteFlyweight:

|  |
| --- |
| public class Artist {  　　//内部状态 　　private String name;  　　// note that Artist is immutable. 　　String getName(){return name;}  　　Artist(String n){ 　　　　name = n; 　　}   } |

再看看Flyweight factory,专门用来制造上面的可共享的ConcreteFlyweight:Artist

|  |
| --- |
| public class ArtistFactory {  　　Hashtable pool = new Hashtable();  　　Artist getArtist(String key){  　　　　Artist result; 　　　　result = (Artist)pool.get(key); 　　　　////产生新的Artist 　　　　if(result == null) { 　　　　　　result = new Artist(key); 　　　　　　pool.put(key,result); 　　　　　　 　　　　} 　　　 return result; 　 }  } |

当你有几千张甚至更多CD时,Flyweight模式将节省更多空间,共享的flyweight越多,空间节省也就越大.

## 一、 享元（Flyweight）模式

Flyweight在拳击比赛中指最轻量级，即"蝇量级"，有些作者翻译为"羽量级"。这里使用"享元模式"更能反映模式的用意。

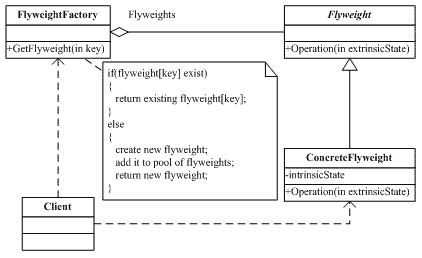
享元模式以共享的方式高效地支持大量的细粒度对象。享元对象能做到共享的关键是区分内蕴状态（Internal State）和外蕴状态（External State）。内蕴状态是存储在享元对象内部并且不会随环境改变而改变。因此内蕴状态并可以共享。

外蕴状态是随环境改变而改变的、不可以共享的状态。享元对象的外蕴状态必须由客户端保存，并在享元对象被创建之后，在需要使用的时候再传入到享元对象内部。外蕴状态与内蕴状态是相互独立的。

**享元模式的应用**

享元模式在编辑器系统中大量使用。一个文本编辑器往往会提供很多种字体，而通常的做法就是将每一个字母做成一个享元对象。享元对象的内蕴状态就是这个字母，而字母在文本中的位置和字模风格等其他信息则是外蕴状态。比如，字母a可能出现在文本的很多地方，虽然这些字母a的位置和字模风格不同，但是所有这些地方使用的都是同一个字母对象。这样一来，字母对象就可以在整个系统中共享。

## 二、 单纯享元模式的结构



在单纯享元模式中，所有的享元对象都是可以共享的。单纯享元模式所涉及的角色如下：

抽象享元(Flyweight)角色：此角色是所有的具体享元类的超类，为这些类规定出需要实现的公共接口。那些需要外蕴状态(External State)的操作可以通过调用商业方法以参数形式传入。

具体享元(ConcreteFlyweight)角色：实现抽象享元角色所规定的接口。如果有内蕴状态的话，必须负责为内蕴状态提供存储空间。享元对象的内蕴状态必须与对象所处的周围环境无关，从而使得享元对象可以在系统内共享的。

享元工厂(FlyweightFactory)角色：本角色负责创建和管理享元角色。本角色必须保证享元对象可以被系统适当地共享。当一个客户端对象调用一个享元对象的时候，享元工厂角色会检查系统中是否已经有一个复合要求的享元对象。如果已经有了，享元工厂角色就应当提供这个已有的享元对象；如果系统中没有一个适当的享元对象的话，享元工厂角色就应当创建一个合适的享元对象。

客户端(Client)角色：本角色需要维护一个对所有享元对象的引用。本角色需要自行存储所有享元对象的外蕴状态。

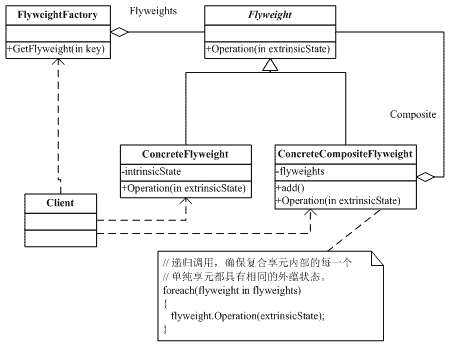
## 三、 单纯享元模式的示意性源代码

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// Flyweight pattern -- Structural example    
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System.Collections;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "FlyweightFactory"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass FlyweightFactory  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Fields  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private Hashtable flyweights = new Hashtable();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Constructors  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public FlyweightFactory()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    flyweights.Add("X", new ConcreteFlyweight());  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    flyweights.Add("Y", new ConcreteFlyweight());  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    flyweights.Add("Z", new ConcreteFlyweight());  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public Flyweight GetFlyweight(string key)  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    return((Flyweight)flyweights[ key ]);  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "Flyweight"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifabstract class Flyweight  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  abstract public void Operation( int extrinsicstate );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteFlyweight"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass ConcreteFlyweight : Flyweight  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private string intrinsicstate = "A";  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  override public void Operation( int extrinsicstate )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("ConcreteFlyweight: intrinsicstate {0}, extrinsicstate {1}",   
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif      intrinsicstate, extrinsicstate );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// Client test  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class Client  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public static void Main( string[] args )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // Arbitrary extrisic state  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    int extrinsicstate = 22;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif       
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    FlyweightFactory f = new FlyweightFactory();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // Work with different flyweight instances  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Flyweight fx = f.GetFlyweight("X");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    fx.Operation( --extrinsicstate );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Flyweight fy = f.GetFlyweight("Y");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    fy.Operation( --extrinsicstate );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Flyweight fz = f.GetFlyweight("Z");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    fz.Operation( --extrinsicstate );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}

## 四、 复合享元模式的结构

单纯享元模式中，所有的享元对象都可以直接共享。下面考虑一个较为复杂的情况，即将一些单纯享元使用合成模式加以复合，形成复合享元对象。这样的复合享元对象本身不能共享，但是它们可以分解成单纯享元对象，而后者则可以共享。

复合享元模式的类图如下图所示：



享元模式所涉及的角色有抽象享元角色、具体享元角色、复合享元角色、享员工厂角色，以及客户端角色等。

抽象享元角色：此角色是所有的具体享元类的超类，为这些类规定出需要实现的公共接口。那些需要外蕴状态(External State)的操作可以通过方法的参数传入。抽象享元的接口使得享元变得可能，但是并不强制子类实行共享，因此并非所有的享元对象都是可以共享的。

具体享元(ConcreteFlyweight)角色：实现抽象享元角色所规定的接口。如果有内蕴状态的话，必须负责为内蕴状态提供存储空间。享元对象的内蕴状态必须与对象所处的周围环境无关，从而使得享元对象可以在系统内共享。有时候具体享元角色又叫做单纯具体享元角色，因为复合享元角色是由单纯具体享元角色通过复合而成的。

复合享元(UnsharableFlyweight)角色：复合享元角色所代表的对象是不可以共享的，但是一个复合享元对象可以分解成为多个本身是单纯享元对象的组合。复合享元角色又称做不可共享的享元对象。

享元工厂(FlyweightFactoiy)角色：本角色负责创建和管理享元角色。本角色必须保证享元对象可以被系统适当地共享。当一个客户端对象请求一个享元对象的时候，享元工厂角色需要检查系统中是否已经有一个符合要求的享元对象，如果已经有了，享元工厂角色就应当提供这个已有的享元对象；如果系统中没有一个适当的享元对象的话，享元工厂角色就应当创建一个新的合适的享元对象。

客户端(Client)角色：本角色还需要自行存储所有享元对象的外蕴状态。

注：由于复合享元模式比较复杂，这里就不再给出示意性代码。通过将享元模式与合成模式组合在一起，可以确保复合享元中所包含的每个单纯享元都具有相同的外蕴状态，而这些单纯享元的内蕴状态往往不同。该部分内容可以参考《Java与模式》第31章内容。

## 五、 一个咖啡摊的例子

在这个咖啡摊(Coffee Stall)所使用的系统里，有一系列的咖啡"风味(Flavor)"。客人到摊位上购买咖啡，所有的咖啡均放在台子上，客人自己拿到咖啡后就离开摊位。咖啡有内蕴状态，也就是咖啡的风味；咖啡没有环境因素，也就是说没有外蕴状态。如果系统为每一杯咖啡都创建一个独立的对象的话，那么就需要创建出很多的细小对象来。这样就不如把咖啡按照种类(即"风味")划分，每一种风味的咖啡只创建一个对象，并实行共享。

使用咖啡摊主的语言来讲，所有的咖啡都可按"风味"划分成如Capucino、Espresso等，每一种风味的咖啡不论卖出多少杯，都是全同、不可分辨的。所谓共享，就是咖啡风味的共享，制造方法的共享等。因此，享元模式对咖啡摊来说，就意味着不需要为每一份单独调制。摊主可以在需要时，一次性地调制出足够一天出售的某一种风味的咖啡。

很显然，这里适合使用单纯享元模式。系统的设计如下：

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System.Collections;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic abstract class Order  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // 将咖啡卖给客人  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public abstract void Serve();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // 返回咖啡的名字  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public abstract string GetFlavor();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class Flavor : Order  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private string flavor;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // 构造函数，内蕴状态以参数方式传入  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public Flavor(string flavor)  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.flavor = flavor;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // 返回咖啡的名字  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override string GetFlavor()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    return this.flavor;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // 将咖啡卖给客人  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Serve()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("Serving flavor " + flavor);  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class FlavorFactory  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private Hashtable flavors = new Hashtable();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public Order GetOrder(string key)  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    if(! flavors.ContainsKey(key))  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif      flavors.Add(key, new Flavor(key));  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif        return ((Order)flavors[key]);  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public int GetTotalFlavorsMade()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    return flavors.Count;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class Client  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private static FlavorFactory flavorFactory;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private static int ordersMade = 0;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public static void Main( string[] args )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    flavorFactory = new FlavorFactory();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Black Coffee");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Capucino");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Espresso");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Capucino");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Espresso");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Black Coffee");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Espresso");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Espresso");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Black Coffee");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Capucino");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Capucino");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Black Coffee");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("\nTotal Orders made: " + ordersMade);  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("\nTotal Flavor objects made: " +   
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif      flavorFactory.GetTotalFlavorsMade());  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private static void TakeOrder(string aFlavor)  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Order o = flavorFactory.GetOrder(aFlavor);  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // 将咖啡卖给客人  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    o.Serve();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    ordersMade++;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}

## 六、 咖啡屋的例子

在前面的咖啡摊项目里，由于没有供客人坐的桌子，所有的咖啡均没有环境的影响。换言之，咖啡仅有内蕴状态，也就是咖啡的种类，而没有外蕴状态。

下面考虑一个规模稍稍大一点的咖啡屋(Coffee Shop)项目。屋子里有很多的桌子供客人坐，系统除了需要提供咖啡的"风味"之外，还需要跟踪咖啡被送到哪一个桌位上，因此，咖啡就有了桌子作为外蕴状态。

由于外蕴状态的存在，没有外蕴状态的单纯享元模式不再符合要求。系统的设计可以利用有外蕴状态的单纯享元模式。系统的代码如下：

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System.Collections;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic abstract class Order  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // 将咖啡卖给客人  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public abstract void Serve(Table table);  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // 返回咖啡的名字  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public abstract string GetFlavor();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class Flavor : Order  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private string flavor;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // 构造函数，内蕴状态以参数方式传入  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public Flavor(string flavor)  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.flavor = flavor;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // 返回咖啡的名字  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override string GetFlavor()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    return this.flavor;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // 将咖啡卖给客人  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Serve(Table table)  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("Serving table {0} with flavor {1}", table.Number, flavor);  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class FlavorFactory  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private Hashtable flavors = new Hashtable();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public Order GetOrder(string key)  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    if(! flavors.ContainsKey(key))  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif      flavors.Add(key, new Flavor(key));  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif        return ((Order)flavors[key]);  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public int GetTotalFlavorsMade()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    return flavors.Count;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class Table  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private int number;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public Table(int number)  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    this.number = number;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public int Number  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    get { return number; }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class Client  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private static FlavorFactory flavorFactory;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private static int ordersMade = 0;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public static void Main( string[] args )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    flavorFactory = new FlavorFactory();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Black Coffee");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Capucino");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Espresso");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Capucino");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Espresso");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Black Coffee");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Espresso");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Espresso");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Black Coffee");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Capucino");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Capucino");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    TakeOrder("Black Coffee");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("\nTotal Orders made: " + ordersMade);  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("\nTotal Flavor objects made: " +   
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif      flavorFactory.GetTotalFlavorsMade());  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private static void TakeOrder(string aFlavor)  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Order o = flavorFactory.GetOrder(aFlavor);  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif      
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // 将咖啡卖给客人  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    o.Serve(new Table(++ordersMade));  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}

## 七、 享元模式应当在什么情况下使用

当以下所有的条件都满足时，可以考虑使用享元模式：

* 一个系统有大量的对象。
* 这些对象耗费大量的内存。
* 这些对象的状态中的大部分都可以外部化。
* 这些对象可以按照内蕴状态分成很多的组，当把外蕴对象从对象中剔除时，每一个组都可以仅用一个对象代替。
* 软件系统不依赖于这些对象的身份，换言之，这些对象可以是不可分辨的。

满足以上的这些条件的系统可以使用享元对象。

最后，使用享元模式需要维护一个记录了系统已有的所有享元的表，而这需要耗费资源。因此，应当在有足够多的享元实例可供共享时才值得使用享元模式。

## 八、 享元模式的优点和缺点

享元模式的优点在于它大幅度地降低内存中对象的数量。但是，它做到这一点所付出的代价也是很高的：

* 享元模式使得系统更加复杂。为了使对象可以共享，需要将一些状态外部化，这使得程序的逻辑复杂化。
* 享元模式将享元对象的状态外部化，而读取外部状态使得运行时间稍微变长。