Java设计模式之Flyweight模式Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022_Java_E8_AE_BE_E8_AE_A1_c104_644657.htm GOF：运用共享技术有效地支持大量细粒度的对象。 解释一下概念：也就是说在一个系统中如果有多个相同的对象，那么只共享一份就可以了，不必每个都去实例化一个对象。比如说（这里引用GOF书中的例子）一个文本系统，每个字母定一个对象，那么大小写字母一共就是52 个，那么就要定义 52个对象。如果有一个1M的文本，那么字母是何其的多，如果每个字母都定义一个对象那么内存早就爆了。那么如果要是每个字母都共享一个对象，那么就大大节约了资源。 在Flyweight模式中，由于要产生各种各样的对象，所以在Flyweight(享元)模式中常出现Factory模式。Flyweight的内部状态是用来共享的,Flyweight factory负责维护一个对象存储池（Flyweight Pool）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， 下面举个例子： 先定义一个抽象的Flyweight类： package Flyweight.public abstract class Flyweight...{ public abstract void operation().}//end abstract class Flyweight 在实现一个具体类： package Flyweight.public class ConcreteFlyweight extends Flyweight...{ private String string. public ConcreteFlyweight(String str) ... { string = str. }//end ConcreteFlyweight(...) public void operation() ... { System.out.println("Concrete---Flyweight : " string). }//end operation()}//end class ConcreteFlyweight 实现一个工厂方法类： package Flyweight.import java.util.Hashtable.public class

FlyweightFactory...{ private Hashtable flyweights = new Hashtable().//----------1 public FlyweightFactory() ...{} public Flyweight getFlyWeight(Object obj) ... { Flyweight flyweight = (Flyweight) flyweights.get(obj).//----------2 if(flyweight == null) ...{//----------3 //产生新的ConcreteFlyweight flyweight = new ConcreteFlyweight((String)obj). flyweights.put(obj, flyweight).//----------5 } return flyweight.//----------6 }//end GetFlyWeight(...) public int getFlyweightSize() ... { return flyweights.size(). }}//end class FlyweightFactory 这个工厂方法类非常关键，这里详细解释一下： 在1处定义了一个Hashtable用来存储各个对象；在2处选出要实例化的对象，在6处将该对象返回，如果在Hashtable中没有要选择的对象，此时变量flyweight为null，产生一个新的flyweight存储在Hashtable中，并将该对象返回。 