**JAVA设计模式之享元模式**

原创 2014年04月04日 07:34:52

* 标签：
* [java设计模式](http://so.csdn.net/so/search/s.do?q=java%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F&t=blog)
* 28429

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](http://dev.yesky.com/)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：

**[java]** [view plain](http://blog.csdn.net/jason0539/article/details/22908915) [copy](http://blog.csdn.net/jason0539/article/details/22908915)

1. **package** Flyweight;
2. **public** **abstract** **class** Flyweight{
3. **public** **abstract** **void** operation();
4. }

实现一个具体类：

**[java]** [view plain](http://blog.csdn.net/jason0539/article/details/22908915) [copy](http://blog.csdn.net/jason0539/article/details/22908915)

1. **package** Flyweight;
2. **public** **class** ConcreteFlyweight **extends** Flyweight{
3. **private** String string;
4. **public** ConcreteFlyweight(String str){
5. string = str;
6. }
7. **public** **void** operation()
8. {
9. System.out.println("Concrete---Flyweight : " + string);
10. }
11. }

实现一个工厂方法类：

**[java]** [view plain](http://blog.csdn.net/jason0539/article/details/22908915) [copy](http://blog.csdn.net/jason0539/article/details/22908915)

1. **package** Flyweight;
2. **import** java.util.Hashtable;
3. **public** **class** FlyweightFactory{
4. **private** Hashtable flyweights = **new** Hashtable();//----------------------------1
5. **public** FlyweightFactory(){}
6. **public** Flyweight getFlyWeight(Object obj){
7. Flyweight flyweight = (Flyweight) flyweights.get(obj);//----------------2
8. **if**(flyweight == **null**){//---------------------------------------------------3
9. //产生新的ConcreteFlyweight
10. flyweight = **new** ConcreteFlyweight((String)obj);
11. flyweights.put(obj, flyweight);//--------------------------------------5
12. }
13. **return** flyweight;//---------------------------------------------------------6
14. }
15. **public** **int** getFlyweightSize(){
16. **return** flyweights.size();
17. }
18. }

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。  
　　最后看看Flyweight的调用：

**[java]** [view plain](http://blog.csdn.net/jason0539/article/details/22908915) [copy](http://blog.csdn.net/jason0539/article/details/22908915)

1. **package** Flyweight;
2. **import** java.util.Hashtable;
3. **public** **class** FlyweightPattern{
4. FlyweightFactory factory = **new** FlyweightFactory();
5. Flyweight fly1;
6. Flyweight fly2;
7. Flyweight fly3;
8. Flyweight fly4;
9. Flyweight fly5;
10. Flyweight fly6;
11. /\*\* \*//\*\* Creates a new instance of FlyweightPattern \*/
12. **public** FlyweightPattern(){
13. fly1 = factory.getFlyWeight("Google");
14. fly2 = factory.getFlyWeight("Qutr");
15. fly3 = factory.getFlyWeight("Google");
16. fly4 = factory.getFlyWeight("Google");
17. fly5 = factory.getFlyWeight("Google");
18. fly6 = factory.getFlyWeight("Google");
19. }
20. **public** **void** showFlyweight(){
21. fly1.operation();
22. fly2.operation();
23. fly3.operation();
24. fly4.operation();
25. fly5.operation();
26. fly6.operation();
27. **int** objSize = factory.getFlyweightSize();
28. System.out.println("objSize = " + objSize);
29. }
30. **public** **static** **void** main(String[] args){
31. System.out.println("The FlyWeight Pattern!");
32. FlyweightPattern fp = **new** FlyweightPattern();
33. fp.showFlyweight();
34. }
35. }

下面是运行结果：

**[java]** [view plain](http://blog.csdn.net/jason0539/article/details/22908915) [copy](http://blog.csdn.net/jason0539/article/details/22908915)

1. Concrete---Flyweight : Google
2. Concrete---Flyweight : Qutr
3. Concrete---Flyweight : Google
4. Concrete---Flyweight : Google
5. Concrete---Flyweight : Google
6. Concrete---Flyweight : Google
7. objSize = 2

　我们定义了6个对象，其中有5个是相同的，按照Flyweight模式的定义“Google”应该共享一个对象，在实际的对象数中我们可以看出实际的对象却是只有2个。

总结：  
　　Flyweight(享元)模式是如此的重要，因为它能帮你在一个复杂的系统中大量的节省内存空间。在JAVA语言中，String类型就是使用了享元模式。String对象是final类型，对象一旦创建就不可改变。在JAVA中字符串常量都是存在常量池中的，JAVA会确保一个字符串常量在常量池中只有一个拷贝。String a="abc"，其中"abc"就是一个字符串常量。

熟悉java的应该知道下面这个例子：

**[java]** [view plain](http://blog.csdn.net/jason0539/article/details/22908915) [copy](http://blog.csdn.net/jason0539/article/details/22908915)

1. String a = "hello";
2. String b = "hello";
3. **if**(a == b)
4. System.out.println("OK");
5. **else**
6. System.out.println("Error");

输出结果是：OK。可以看出if条件比较的是两a和b的地址，也可以说是内存空间  
核心总结，可以共享的对象，也就是说返回的同一类型的对象其实是同一实例，当客户端要求生成一个对象时，工厂会检测是否存在此对象的实例，如果存在那么直接返回此对象实例，如果不存在就创建一个并保存起来，这点有些单例模式的意思。通常工厂类会有一个集合类型的成员变量来用以保存对象，如hashtable,vector等。在java中，数据库连接池，线程池等即是用享元模式的应用