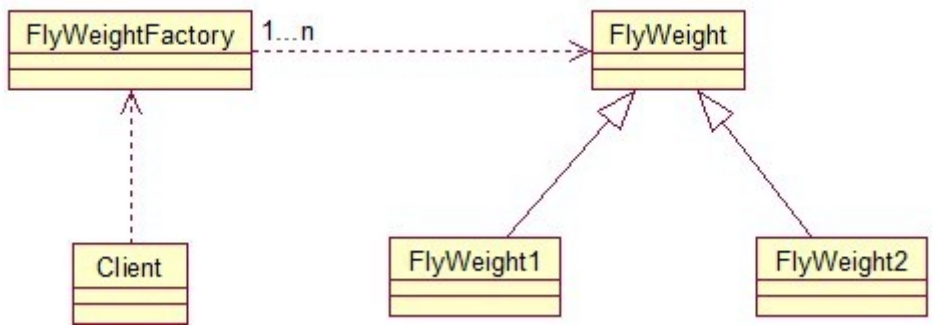
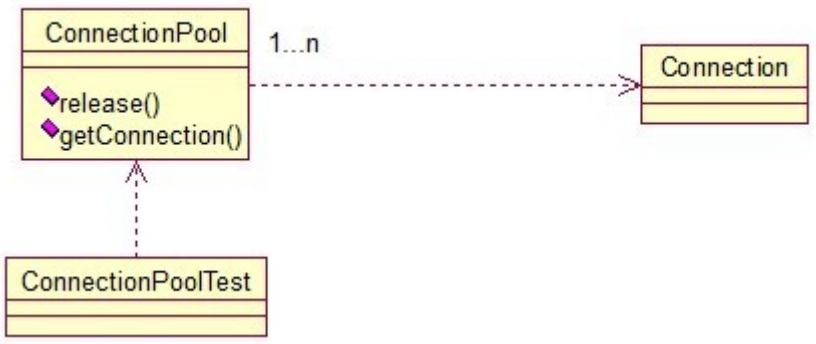
享元模式（Flyweight）

享元模式的主要目的是实现对象的共享，即共享池，当系统中对象多的时候可以减少内存的开销，通常与工厂模式一起使用。



FlyWeightFactory负责创建和管理享元单元，当一个客户端请求时，工厂需要检查当前对象池中是否有符合条件的对象，如果有，就返回已经存在的对象，如果没有，则创建一个新对象，FlyWeight是超类。一提到共享池，我们很容易联想到Java里面的JDBC连接池，想想每个连接的特点，我们不难总结出：适用于作共享的一些个对象，他们有一些共有的属性，就拿数据库连接池来说，url、driverClassName、username、password及dbname，这些属性对于每个连接来说都是一样的，所以就适合用享元模式来处理，建一个工厂类，将上述类似属性作为内部数据，其它的作为外部数据，在方法调用时，当做参数传进来，这样就节省了空间，减少了实例的数量。

看个例子：



看下数据库连接池的代码：

[java] view plaincopy

```
1. public class ConnectionPool {
2.
3.     private Vector<Connection> pool;
4.
5.     /*公有属性*/
6.     private String url = "jdbc:mysql://localhost:3306/test";
7.     private String username = "root";
8.     private String password = "root";
9.     private String driverClassName = "com.mysql.jdbc.Driver";
```

```
10.
11.     private int poolSize = 100;
12.     private static ConnectionPool instance = null;
13.     Connection conn = null;
14.
15.     /*构造方法，做一些初始化工作*/
16.     private ConnectionPool() {
17.         pool = new Vector<Connection>(poolSize);
18.
19.         for (int i = 0; i < poolSize; i++) {
20.             try {
21.                 Class.forName(driverClassName);
22.
                conn = DriverManager.getConnection(url, username, password
);
23.                 pool.add(conn);
24.             } catch (ClassNotFoundException e) {
25.                 e.printStackTrace();
26.             } catch (SQLException e) {
27.                 e.printStackTrace();
28.             }
29.         }
30.     }
31.
32.     /* 返回连接到连接池 */
33.     public synchronized void release() {
34.         pool.add(conn);
35.     }
36.
37.     /* 返回连接池中的一个数据库连接 */
38.     public synchronized Connection getConnection() {
39.         if (pool.size() > 0) {
40.             Connection conn = pool.get(0);
41.             pool.remove(conn);
42.             return conn;
43.         } else {
44.             return null;
45.         }
46.     }
47. }
```

通过连接池的管理，实现了数据库连接的共享，不需要每一次都重新创建连接，节省了数据库重新创建的开销，提升了系统的性能！