链滴

# GOF 设计模式小白教程享元模式

作者：valarchie
原文链接：https://ld246.com/article/1567140862938
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

# 享元模式（Flyweight）

## 定义：

运用共享技术来有效地支持大量细粒度对象的复用。它通过共享已经存在的对象来大幅度减少需要创的对象数量、避免大量相似类的开销，从而提高系统资源的利用率。

享元模式能做到共享的关键是区分内蕴状态和外蕴状态。内蕴状态存储在享元内部，不会随环境的改而有所不同。外蕴状态是随环境的改变而改变的。

## 通俗解释：

设计一个围棋游戏程序，每下一个棋子我们就新建一个棋子的对象。棋盘大小的是19*19=361的位置下满棋盘的话就需要创建361个对象。而分析一下，其实棋子在表现模型上是一样的，只分为黑棋和棋。我们只需要共享一个黑棋一个白棋，在下棋的时候通过传入棋盘的坐标即可。例如JDK当中的Strig类型，不同地方相同的字符串会引用缓存池当中的同一个字符串，这也是享元模式的一种体现。

## 代码：

抽象棋子即抽象享元类

```
public interface ChessPiece {
    // 下棋的动作
    void placePiece(int x, int y);
}
```

黑色棋子、白色棋子即具体享元类

```
public class BlackChessPiece implements ChessPiece {

    private String color = "黑子";

    @Override
    public void placePiece(int x, int y) {
        System.out.println(color + "下在棋盘下标为" + x + "," + y + "的位置！");
    }

}

public class WhiteChessPiece implements ChessPiece {

    private String color = "白子";

    @Override
    public void placePiece(int x, int y) {
        System.out.println(color + "下在棋盘下标为" + x + "," + y + "的位置！");
    }

}
```

享元工厂类，保存共享的享元对象，棋子工厂仅保存一个黑色棋子和一个白色棋子

```
public class ChessPieceFactory {

    Map<String, ChessPiece> chessPieceMap = new HashMap<>();

    public ChessPieceFactory() {

        chessPieceMap.put("黑", new BlackChessPiece());
        chessPieceMap.put("白", new WhiteChessPiece());

    }

    public ChessPiece getChessPiece(String color) {
        return chessPieceMap.get(color);
    }

}
```

测试享元模式，通过共享唯一的白棋和黑棋进行下棋。

```
public class TestFlyweight {

    public static void main(String[] args) {

        ChessPieceFactory chessPieceFactory = new ChessPieceFactory();

        ChessPiece black1 = chessPieceFactory.getChessPiece("黑");
        black1.placePiece(1, 1);

        ChessPiece white1 = chessPieceFactory.getChessPiece("白");
        white1.placePiece(2,2);

        ChessPiece black2 = chessPieceFactory.getChessPiece("黑");
        black1.placePiece(3, 3);

        ChessPiece white2 = chessPieceFactory.getChessPiece("白");
        white1.placePiece(4,4);


    }


}
```

运行结果：

```
黑子下在棋盘下标为1,1的位置！
白子下在棋盘下标为2,2的位置！
黑子下在棋盘下标为3,3的位置！
白子下在棋盘下标为4,4的位置！
```

## 解析：

1. 相同对象只要保存一份，这降低了系统中对象的数量，从而降低了系统中细粒度对象给内存带来的力。

2. 适用于系统有大量相似对象或需要缓冲池的场景。