

# 设计模式 工厂模式 从卖肉夹馍说起

原创

鸿洋\_

于 2014-04-25 13:00:38 发布

37125

收藏 34

版权

分类专栏：


【Java 设计模式】

设计模式融入生活

文章标签：

设计模式

工厂模式



【Java 设计模式】 同时关注

67 订阅 10 篇文章

订阅专栏

转载请标明出处：<http://blog.csdn.net/Imj623565791/article/details/24460585>

今天继续 设计模式 之旅，给大家带来工厂模式，简单列一下这个模式的家族：

- 1、静态工厂模式
- 2、简单工厂模式
- 3、工厂方法模式
- 4、抽象工厂模式

是不是觉得，我勒个去，这工厂还能列出这么多分类，哈哈，下面开始各个击破。

## 1、静态工厂模式

这个最常见了，项目中的辅助类，TextUtil.isEmpty等，类+静态方法。下面开始详细介绍：略。

## 2、简单工厂模式

下面开始谈谈卖肉夹馍，最近程序员卖肉夹馍很火，啥时候大牛们都去卖了，我等就崛起了，哈哈。

首先你得有个店：RoujiaMoStore

```
1 package com.zhy.pattern.factory.a;
2
3 public class RoujiaMoStore
4 {
5
6     /**
7      * 根据传入类型卖不同的肉夹馍
8      *
9      * @param type
10     * @return
11     */
12     public RoujiaMo sellRouJiaMo(String type)
13     {
14         RoujiaMo rouJiaMo = null;
15
16         if (type.equals("Suan"))
17         {
18             rouJiaMo = new SuanRouJiaMo();
19
20         } else if (type.equals("Tian"))
21         {
22             rouJiaMo = new TianRouJiaMo();
23         } else if (type.equals("La"))
24         {
25             rouJiaMo = new LaRouJiaMo();
26         }
27
28         rouJiaMo.prepare();
29         rouJiaMo.fire();
30         rouJiaMo.pack();
31         return rouJiaMo;
32     }
33 }
34 }
```

然后你得有各种风味的馍馍：

```
1 package com.zhy.pattern.factory.a;
2
3 public abstract class RouJiaMo
4 {
5     protected String name;
6
7     /**
8      * 准备工作
9      */
10    public void prepare()
11    {
12        System.out.println("揉面-剁肉-完成准备工作");
13    }
14
15    /**
16     * 使用你们的专用袋-包装
17     */
18    public void pack()
19    {
20        System.out.println("肉夹馍-专用袋-包装");
21    }
22    /**
23     * 秘制设备-烘烤2分钟
24     */
25    public void fire()
26    {
27        System.out.println("肉夹馍-专用设备-烘烤");
28    }
29 }
```

```
1 package com.zhy.pattern.factory.a;
2
3 import com.zhy.pattern.factory.a.RouJiaMo;
4
5 /**
6  * 辣味肉夹馍
7  *
8  * @author zhy
9  *
10 */
11 public class LaRouJiaMo extends RouJiaMo
12 {
13     public LaRouJiaMo()
14     {
15         this.name = "辣味肉夹馍";
16     }
17 }
```

```
1 package com.zhy.pattern.factory.a;
2
3
4
5 /**
6  * 酸味肉夹馍
7  *
8  * @author zhy
9  *
10 */
11 public class SuanRouJiaMo extends RouJiaMo
12 {
13     public SuanRouJiaMo()
```

```

14 |     {
15 |         this.name = "酸味肉夹馍";
16 |     }
17 | }

```

```

1 | package com.zhy.pattern.factory.a;
2 |
3 |
4 |
5 | /**
6 |  * 酸味肉夹馍
7 |  *
8 |  * @author zhy
9 |  *
10 |  */
11 | public class SuanRouJiaMo extends RouJiaMo
12 | {
13 |     public SuanRouJiaMo()
14 |     {
15 |         this.name = "酸味肉夹馍";
16 |     }
17 | }

```

现在这样的设计，虽然可以支持你卖肉夹馍了，但是有点问题，生产馍的种类和你的RouJiaMoStore耦合度太高了，如果增加几种风味，删除几种风味，你得一直修改sellRouJiaMo中的方法，所以我们需要做一定的修改，此时简单工厂模式就能派上用场了。

我们开始写个简单工厂，把产生馍的过程拿出来：

```

1 | package com.zhy.pattern.factory.a;
2 |
3 | public class SimpleRouJiaMoFactroy
4 | {
5 |     public RouJiaMo createRouJiaMo(String type)
6 |     {
7 |         RouJiaMo rouJiaMo = null;
8 |         if (type.equals("Suan"))
9 |         {
10 |             rouJiaMo = new SuanRouJiaMo();
11 |
12 |         } else if (type.equals("Tian"))
13 |         {
14 |             rouJiaMo = new TianRouJiaMo();
15 |         } else if (type.equals("La"))
16 |         {
17 |             rouJiaMo = new LaRouJiaMo();
18 |         }
19 |         return rouJiaMo;
20 |     }
21 |
22 | }

```

然后以组合的方式，让Store来使用：

```

1 | package com.zhy.pattern.factory.a;
2 |
3 | public class RouJiaMoStore
4 | {
5 |     private SimpleRouJiaMoFactroy factroy;
6 |
7 |     public RouJiaMoStore(SimpleRouJiaMoFactroy factroy)

```

```
8 | {
9 |     this.factory = factory;
10 | }
11 |
12 | /**
13 |  * 根据传入类型卖不同的肉夹馍
14 |  *
15 |  * @param type
16 |  * @return
17 |  */
18 | public RouJiaMo sellRouJiaMo(String type)
19 | {
20 |     RouJiaMo rouJiaMo = factory.createRouJiaMo(type);
21 |     rouJiaMo.prepare();
22 |     rouJiaMo.fire();
23 |     rouJiaMo.pack();
24 |     return rouJiaMo;
25 | }
26 |
27 | }
```

好了，现在你随便添加什么种类的馍，删除什么种类的馍就和Store无关了，就是么~人家只负责卖馍么~这就是简单工厂模式，当然了，大家也都比较熟悉。

### 3、工厂方法模式

**定义：**定义一个创建对象的接口，但由子类决定要实例化的类是哪一个。工厂方法模式把类实例化的过程推迟到子类。

好了，看完定义，下面我们用例子来展示。继续肉夹馍，由于使用了简单工厂模式，肉夹馍生意那个好啊，所以下载决定去西安开个分店，去北京开个分店。既然有分店了，那总店就是抽象的了：

```
1 | package com.zhy.pattern.factory.b;
2 |
3 | public abstract class RouJiaMoStore
4 | {
5 |
6 |     public abstract RouJiaMo createRouJiaMo(String type);
7 |
8 |     /**
9 |      * 根据传入类型卖不同的肉夹馍
10 |      *
11 |      * @param type
12 |      * @return
13 |      */
14 |     public RouJiaMo sellRouJiaMo(String type)
15 |     {
16 |         RouJiaMo rouJiaMo = createRouJiaMo(type);
17 |         rouJiaMo.prepare();
18 |         rouJiaMo.fire();
19 |         rouJiaMo.pack();
20 |         return rouJiaMo;
21 |     }
22 |
23 | }
```

然后在开两个分店，这里拿一个代码做演示，其他都一样：

```
1 | package com.zhy.pattern.factory.b;
2 |
3 |
4 |
5 | /**
6 |  * 西安肉夹馍店
```

```

7 | *
8 | * @author zhy
9 | *
10 | */
11 | public class XianRouJiaMoStore extends RouJiaMoStore
12 | {
13 |
14 |     @Override
15 |     public RouJiaMo createRouJiaMo(String type)
16 |     {
17 |         RouJiaMo rouJiaMo = null;
18 |         if (type.equals("Suan"))
19 |         {
20 |             rouJiaMo = new XianSuanRouJiaMo();
21 |
22 |         } else if (type.equals("Tian"))
23 |         {
24 |             rouJiaMo = new XianTianRouJiaMo();
25 |         } else if (type.equals("La"))
26 |         {
27 |             rouJiaMo = new XianLaRouJiaMo();
28 |         }
29 |         return rouJiaMo;
30 |
31 |     }
32 |
33 | }

```

然后就是各个西安口味的肉夹馍了，这代码就不贴了。可以看出我们把制作肉夹馍的过程以抽象方法的形式让子类去决定了，对照定义：

1、定义了创建对象的一个接口：public abstract RouJiaMo createRouJiaMo(String type);

2、由子类决定实例化的类，可以看到我们的馍是子类生成的。

可能有人会说，我用简单工厂模式也行啊，但是如果10来个城市\*5种风味/城市，那么岂不是简单工厂里面需要50多个if，再说人家西安肉夹馍分店就不能有点自己的秘诀，当然由它自己定最好。

好了，方法工厂模式介绍完毕。

#### 4、抽象工厂模式

**定义：**提供一个接口，用于创建相关的或依赖对象的家族，而不需要明确指定具体类。

这定义有点绕口，算了，还是拿例子来说。继续卖肉夹馍，咱们生意这么好，难免有些分店开始动歪脑子，开始使用劣质肉等，砸我们的品牌。所以我们要拿钱在每个城市建立自己的原料场，保证高质量原料的供应。

于是我们新建一个提供原料的接口：

```

1 | package com.zhy.pattern.factory.b;
2 |
3 | /**
4 |  * 提供肉夹馍的原料
5 |  * @author zhy
6 |  *
7 |  */
8 | public interface RouJiaMoYLFactroy
9 | {
10 |     /**
11 |      * 生产肉
12 |      * @return
13 |      */
14 |     public Meat createMeat();
15 | }

```

```

16 | /**17 |      * 生产调料神马的
18 |      * @return
19 |      */
20 | public YuanLiao createYuanliao();
21 |
22 | }

```

```

1 | package com.zhy.pattern.factory.b;
2 |
3 | /**
4 |  * 根据西安当地特色，提供这两种材料
5 |  * @author zhy
6 |  *
7 |  */
8 | public class XianRouJiaMoYLFactroy implements RouJiaMoYLFactroy
9 | {
10 |
11 |     @Override
12 |     public Meat createMeat()
13 |     {
14 |         return new FreshMest();
15 |     }
16 |
17 |     @Override
18 |     public YuanLiao createYuanliao()
19 |     {
20 |         return new XianTeSeYuanliao();
21 |     }
22 |
23 | }

```

有了原理工厂，那我们稍微修改下RouJiaMo的prepare方法：

```

1 | package com.zhy.pattern.factory.b;
2 |
3 | public abstract class RouJiaMo
4 | {
5 |     protected String name;
6 |
7 |     /**
8 |      * 准备工作
9 |      */
10 |     public final void prepare(RouJiaMoYLFactroy ylFactroy)
11 |     {
12 |         Meat meat = ylFactroy.createMeat();
13 |         YuanLiao yuanliao = ylFactroy.createYuanliao();
14 |         System.out.println("使用官方的原料" + meat + " , " + yuanli
15 |     }
16 |
17 |     /**
18 |      * 使用你们的专用袋-包装
19 |      */
20 |     public final void pack()
21 |     {
22 |         System.out.println("肉夹馍-专用袋-包装");
23 |     }
24 |
25 |     /**
26 |      * 秘制设备-烘烤2分钟
27 |      */
28 |     public final void fire()
29 |     {
30 |         System.out.println("肉夹馍-专用设备-烘烤");
31 |     }

```

好了，现在必须用我们官方原料做为原材料了。

现在对比定义：

- 1、提供一个接口：public interface RouJiaMoYLFactroy
- 2、用于创建相关的或依赖对象的家族 public Meat createMeat();public YuanLiao createYuanliao();我们接口用于创建一系列的原材料。

好了，最后测试下，我要在西安馍店，买个酸味的尝尝：

```
1 package com.zhy.pattern.factory.b;
2
3
4 public class Test
5 {
6     public static void main(String[] args)
7     {
8
9         RoujiaMoStore roujiaMoStore = new XianRouJiaMoStore();
10
11         RouJiaMo suanRoujiaMo = roujiaMoStore.sellRouJiaMo("Suan")
12         System.out.println(suanRoujiaMo.name);12
13     }
```

- 1 使用官方的原料com.zhy.pattern.factory.b.FreshMest@e53108 ， com.zhy.
- 2 肉夹馍-专用设备-烘烤
- 3 肉夹馍-专用袋-包装
- 4 酸味肉夹馍

哈哈~肉夹馍店的已经建立起来了，兄弟们卖馍去把~记得留个言，给个赞~

显示推荐内容

觉得还不错? 一键收藏

鸿洋\_ 关注

179

34

45

专栏目录