Android设计模式源码解析之工厂方法模式

> 本文为 Android 设计模式源码解析 中工厂方法模式分析
> Android系统版本：4.4.4
> 分析者：Aige，分析状态：未完成，校对者：Mr.Simple，校对状态：未开始

1. 模式介绍

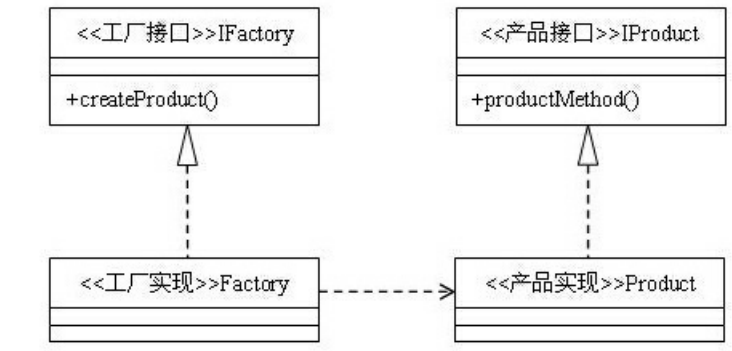
模式的定义

你要什么工厂造给你就是了，你不用管我是怎么造的，造好你拿去用就是了，奏是介么任性。

模式的使用场景

任何需要生成对象的情况都可使用工厂方法替代生成。

2. UML类图



角色介绍

如图

3. 模式的简单实现

简单实现的介绍

工厂方法相对来说比较简单也很常用，如上所说，任何需要生成对象的情况都可使用工厂方法替代生成。我们知道在Java中生成对象一般会使用关键字new：

```
Client client = new Client();
```

如果使用工厂方法来替代，我们则可以先声明一个工厂类：

```
/**
 * 工厂类
 *
 * @author Aige{@link https://github.com/AigeStudio}
 *
 */
public class Factory {
    public static Client getClient() {
        return new Client();
    }
}
```

然后呢就可以使用这个工厂类来生成Client对象：

```
Client client = Factory.getClient();
```

但是，相信即便是新手也会觉得这套路感觉不对啊是吧，生成个对象居然这么麻烦我也是醉了。So，我们极少这么使用，生成某个具体的对象直接new就是了对吧。那好，假定我们这有三个类：香蕉类、黄瓜类、甘蔗类，我们将其统称为水果类，额某种意义上来说黄瓜也算是水果，为其定义一个抽象父类：

不说黄瓜也算是水果，为其定义一个抽象父类：

```
/**
 * 抽象水果类
 *
 * @author Aige{@link https://github.com/AigeStudio}
 *
 */
public abstract class Fruits {
    /**
     * 水果的颜色
     */
    public abstract void color();

    /**
     * 水果的重量
     */
    public abstract void weight();
}
```

然后呢则是各个具体的水果类：

```
/**
 * 香蕉类
 *
 * @author Aige{@link https://github.com/AigeStudio}
 *
 */
public class Banana extends Fruits {
    @Override
    public void color() {
        System.out.println("Banana is red");
    }

    @Override
    public void weight() {
        System.out.println("Weight 0.3kg");
    }
}
```

```
/**
 * 黄瓜类
 *
 * @author Aige{@link https://github.com/AigeStudio}
 *
 */
public class Cucumber extends Fruits {
    @Override
    public void color() {
        System.out.println("Cucumber is green");
    }

    @Override
    public void weight() {
        System.out.println("Weight 0.5kg");
    }
}
```

```
/**
 * 甘蔗类
 *
 * @author Aige{@link https://github.com/AigeStudio}
 *
 */
public class Sugarcane extends Fruits {
    @Override
    public void color() {
        System.out.println("Sugarcane is purple");
    }

    @Override
```

```
    public void weight() {
        System.out.println("Weight 1.3kg");
    }
}
```

最后，上场景：

```
/**
 * 场景模拟类
 *
 * @author Aige{@link https://github.com/AigeStudio}
 *
 */
public class Client {
    public static void main(String[] args) {
        Fruits banana = new Banana();
        banana.color();
        banana.weight();

        Fruits cucumber = new Cucumber();
        cucumber.color();
        cucumber.weight();

        Fruits sugarcane = new Sugarcane();
        sugarcane.color();
        sugarcane.weight();
    }
}
```

具体的运行结果就不贴了，不用运行你也该知道是啥结果 = = 。这样一段代码其实已经算过得去了，抽象有了具现也有，也确实没啥问题对吧，可是仔细想想每个类我们都要通过具体的类去new生成，能不能进一步解耦将生成具体对象的工作交由第三方去做呢？答案是肯定的，我们来定义一个果农类，你要啥水果给果农说一声，让它给你就是了：

```
/**
 * 果农类
 *
 * @author Aige{@link https://github.com/AigeStudio}
 *
 */
public class Grower {
    public <T extends Fruits> T getFruits(Class<T> clz) {
        try {
            Fruits fruits = (Fruits) Class.forName(clz.getName()).newInstance();
            return (T) fruits;
        } catch (Exception e) {
            return null;
        }
    }
}
```

这样一来，我们要什么水果直接跟果农报个名（Class clz），然后果农给你就是了，OK，修改下场景类：

```
/**
 * 场景模拟类
 *
 * @author Aige{@link https://github.com/AigeStudio}
 *
 */
public class Client {
    public static void main(String[] args) {
        Grower grower = new Grower();

        Fruits banana = grower.getFruits(Banana.class);
        banana.color();
        banana.weight();

        Fruits cucumber = grower.getFruits(Cucumber.class);
```

```
        cucumber.color();
        cucumber.weight();

        Fruits sugarcane = grower.getFruits(Sugarcane.class);
        sugarcane.color();
        sugarcane.weight();
    }
}
```

具体的水果类不用变，运行结果什么的就不扯了，自己去试。恩，这样就差不多，更进一步地，我们可以考虑将果农类进一步抽象，在抽象类中定义方法：

```
/**
 * 抽象果农类
 *
 * @author Aige{@link https://github.com/AigeStudio}
 *
 */
public abstract class AGrower {
    /**
     * 获取水果
     *
     * @param clz
     *            具体水果类型
     * @return 具体水果的对象
     */
    public abstract <T extends Fruits> T getFruits(Class<T> clz);
}
```

然后让Grower extends AGrower就行了其他不变，最终的结构就与上面我们给出的UML类图一致了，香蕉、甘蔗、黄瓜什么的为具体的产品类而水果则作为产品类的抽象，Grower和AGrower的关系亦如此。然而很多时候其实我们没必要多个工厂类，一个足以：

```
/**
 * 果农类
 *
 * @author Aige{@link https://github.com/AigeStudio}
 *
 */
public class Grower {
    public static <T extends Fruits> T getFruits(Class<T> clz) {
        try {
            Fruits fruits = (Fruits) Class.forName(clz.getName()).newInstance();
            return (T) fruits;
        } catch (Exception e) {
            return null;
        }
    }
}
```

如代码所示，去掉了抽象父类的继承使getFruits变为静态方法在Client中直接调用不再生成类了。具体代码就不贴了。

实现源码

上述案例的源码实现

总结

对上述的简单示例进行总结说明

Android源码中的模式实现

分析源码中的模式实现，列出相关源码，以及使用该模式原因等

4. 杂谈

该模式的优缺点以及自己的一些感悟，非所有项目必须。

写完相关内容之后到开发群告知管理员，管理员安排相关人员进行审核，审核通过之后即可。