链滴

Java Design Pattern

作者：CM

原文链接：https://ld246.com/article/1486373415047

来源网站：链滴

Factory Design Pattern

工厂模式，简单来描述就是，一个类拥有类似工厂的生产模式，根据用户的要求，返回指定的类型。
们用一个Dog的示例来描述

Interface Dog

```
public interface Dog{
    void speak();
}
```

Poodle Dog

```
public class  Poodle implements Dog{

    @overview
    public void speak(){
        sout("poodle say:wrf!");
    }
}
```

Rottweiler Dog

```
public class Rottweiler implements Dog{
    @overview
    public void speak(){
        sout("rottweiler say loudly:woof!");
    }
}
```

Factory Class

```
//factory to make dog
public class DogFactory{
    //get dog by type ,param [type] dog type
    public static Dog getDog(DogTypeEnum type)throws Exception{
        Dog dog = null;
        switch(type){
            case POODLE:
                dog=new Poodler();
            break;
            case ROTTWEILER:
                dog = new Rottweiler();
            break;
            default:
            throw new RuntimeException("unkonwn dog type!");
        }
        return dog;
    }
}
```

如此，通过工厂类，构建Dog对象是，根据不同的DogTypeEnum值，可以得到的对象实力不同，即单的工厂设计模式。

顺便补充一下，工厂模式常用的场景例如WEB开发中需要的SmsSender 、MailSender的编写，需要多种短信运营商或者邮箱类型提供不同的Sender对象时可用。具体参照各自的项目。

—

Abstract Factory Mode

抽象工厂模式，抽象工厂可以理解为工厂的工厂。抽象工厂类生产的对象即为工厂。可以将之理解为Factory Producer.
延续上一章节的demo，我们假设（现实中不一定存在这样的情况），每一种dog都有自己的毛发颜，因此，为了这个颜色，我们构建一个工厂类ColorFactory，用于生产颜色。

```
//颜色接口
public interface Color{
    void getColor(){};
}

public class Blue implements Color{

    @overview
    public void getColor(){
        sout("color filled:Blue()");
    }
}

public class Red implements Color{

    @overview
    public void getColor(){
        sout("color filled:Red()");
    }
}

public class Green implements Color{

    @overview
    public void getColor(){
        sout("color filled:Green()");
    }
}

//颜色工厂类
public class ColorFactory{
    //获取颜色，简单描述为直接构建实例并调用getColor()方法
    public static Color getColor(ColorTypeEnum type){
        switch(type){
        case blue:
            return new Blue().getColor();
            break;
        case red:
            return new Red().getColor();
            break;
```

```
        case green:
            return new Green().getColor();
            break;
        default:
            return null;
        }
    }
}
```

那么，现在我们具有了两个工厂，一个是DogFactory ，一个是ColorFactory.那么我们如何组建一个象工厂来实现，每只狗有自己对应的肤色呢？那就让我们来构建一个抽象工厂类。

```
//抽象工厂类，用于提供指定的对象接口
public abstract AbstractFactory{
    abstract Dog getDog(DogTypeEnum type);
    abstract Color getColor(ColorTypeEnum type);
}

//分别让两个工厂 继承这个抽象工厂类

public class DogFactory extends AbstractFactory {
    @overview
    public Dog getDog(DogTypeEnum type)throws Exception{
        Dog dog = null;
        switch(type){
            case POODLE:
                dog=new Poodler();
            break;
            case ROTTWEILER:
                dog = new Rottweiler();
            break;
            default:
            throw new RuntimeException("unkonwn dog type!");
        }
        return dog;
    }
    @overview
    public Color getColor(ColorTypeEnum type){
        return null;
    }
}

public class ColorFactory{
    //获取颜色，简单描述为直接构建实例并调用getColor()方法
    @overview
    public Color getColor(ColorTypeEnum type){
        switch(type){
        case blue:
            return new Blue().getColor();
            break;
        case red:
            return new Red().getColor();
            break;
        case green:
```

```
            return new Green().getColor();
            break;
        default:
            return null;
        }
    }

    public Dog getDog(DogTypeEnum type){
        return null;
    }
}
```

此时我们将两个工厂类相对的统一了。那么我们这时可以提供一个统一的工厂生产类，FactoryProduer

```
public class FactoryProducer{
    public AbstractFactory getFactory(FactoryTypeEnum type){
        AbstractFactory factory = null;
        switch(type){
        case dog:
            factory  = new DogFactory();
            break;
        case color:
            factory = new ColorFactory();
            break;
        default:
            //null
        }
        return factory;
    }
}
```

由此，统一了工厂的生产类。此时你可以通过getFactory();并调用对应的getDog()或者getColor()方获取对象实例，不同的狗狗拥有不同的肤色。就好像QQ的皮肤一样，一换可以换一整套