1. 代理设计模式概念

两个子类共同实现一个接口，其中一个子类负责真实业务实现，另外一个子类辅助真实业务的完成。就像战场上的狙击手都配有一名观察员，狙击手负责狙杀敌人，观察员负责辅助狙击手完成任务，两人都是为了狙杀敌人这个任务而生。

代理设计模式一般都与工厂设计模式相配合使用，对于工厂设计模式还有疑惑的可以参考：

> Java 反射3-简单工厂模式

```
// 业务接口
interface ISubject {
    public void eat();
}

// 真实业务类，吃饭
class RealSubject implements ISubject {
    @Override
    public void eat() {
        System.out.println("吃饭");
    }
}

// 代理类，我们将吃饭前的洗手，吃饭后的洗碗与吃饭这件事加在一起才能构成吃饭的完整过程
class ProxySubject implements ISubject {
    private ISubject iSubject = null;
    // 通过构造方法将真实业务传进来
    public ProxySubject(ISubject iSubject) {
        this.iSubject = iSubject;
    }
    // 辅助业务
    public void prepareEat() {
        System.out.println("洗手");
    }
    // 辅助业务
    public void afterEat() {
        System.out.println("洗碗");
    }

    // 真实业务和辅助业务结合在一起的产物，我们需要的就是它
    @Override
    public void eat() {
        this.prepareEat();
        this.iSubject.eat();
        this.afterEat();
    }
}

// 通过工厂我们要拿到代理类的对象
class Factory {
    private Factory() {
    }

    public static ISubject getInstance() {
        ISubject real = new RealSubject();
```

```
        ISubject proxy = new ProxySubject(real);
        return proxy;
    }
}

public class Test {
    public static void main(String[] args) throws Exception {
        ISubject iSubject = Factory.getInstance();
        iSubject.eat();
    }
}
```

上面的例子将代理设计模式大体框架展示了出来，但是，还有小瑕疵，如果按上面的工厂类设计模式，那么以后不同的业务类我们就得设计不同的工厂类，不是很麻烦吗。接下来，使用反射来解决这个问题。

2. 基础代理模式

使用反射将工厂类中的内容修改一下即可，修改如下：

```
class Factory {

    private Factory() {
    }

    // 输入真实业务类全名称，返回真实业务类
    public static <T> T getInstance(String className) throws Exception {
        return (T)Class.forName(className).newInstance();
    }

    // 输入代理业务类全名称，真实业务类，返回代理业务类
    public static <T> T getInstance(String className, Object object) throws Exception{
        Constructor<?> constructor =
Class.forName(className).getConstructor(object.getClass().getInterfaces()[0]);
        return (T)constructor.newInstance(object);
    }

}

public class Test {
    public static void main(String[] args) throws Exception {
        // version 1
        ISubject iSubject = Factory.getInstance("class13.ProxySubject",
Factory.getInstance("class13.RealSubject"));
        iSubject.eat();
    }
}
```

虽然使用了反射之后，使客户端的压力骤然减少，可是确让工厂类的代码变得一塌糊涂，我们进一步进行修改。

```
public static <T> T getInstance(String proxyName, String realName) throws Exception {
    T real = (T)Class.forName(realName).newInstance();
    Constructor<?> constructor =
Class.forName(proxyName).getConstructor(real.getClass().getInterfaces()[0]);
```

```
        return (T)constructor.newInstance(real);
    }
```

代理设计模式总体思路如图所示：