J2EE设计模式

彭晨阳（板桥里人）

http://www.jdon.com

设计模式是系统架构之基础

- J2EE多层系统主要由架构设计、框架以及多个设计模式组成。
- 设计模式是一种实践的总结，是OOP最直接的表现。
- 掌握设计模式与否是衡量程序员设计水平高低的主要依据。

GoF设计模式

- GoF设计模式主要列举了常用的23种模式
- Java的GoF设计模式实现主要表现在面向接口编程。
- 工厂模式是最常用的一种设计模式

工厂模式简介

- 设定一个Class名称是AClass，在面向对象编程中，一般一个Class都会继承一个接口，设定AClass的接口为AInterface，那么生成AClass的对象方法如下：

```
AInterface  a = new AClass();
```

工厂模式简介（续）

改写成下列方式：

```
AInterface  a = Afactory.create();    //代码2
```

上面代码2这一行是使用Afactory的create方法来生成AInterface实例

工厂模式简介（续）

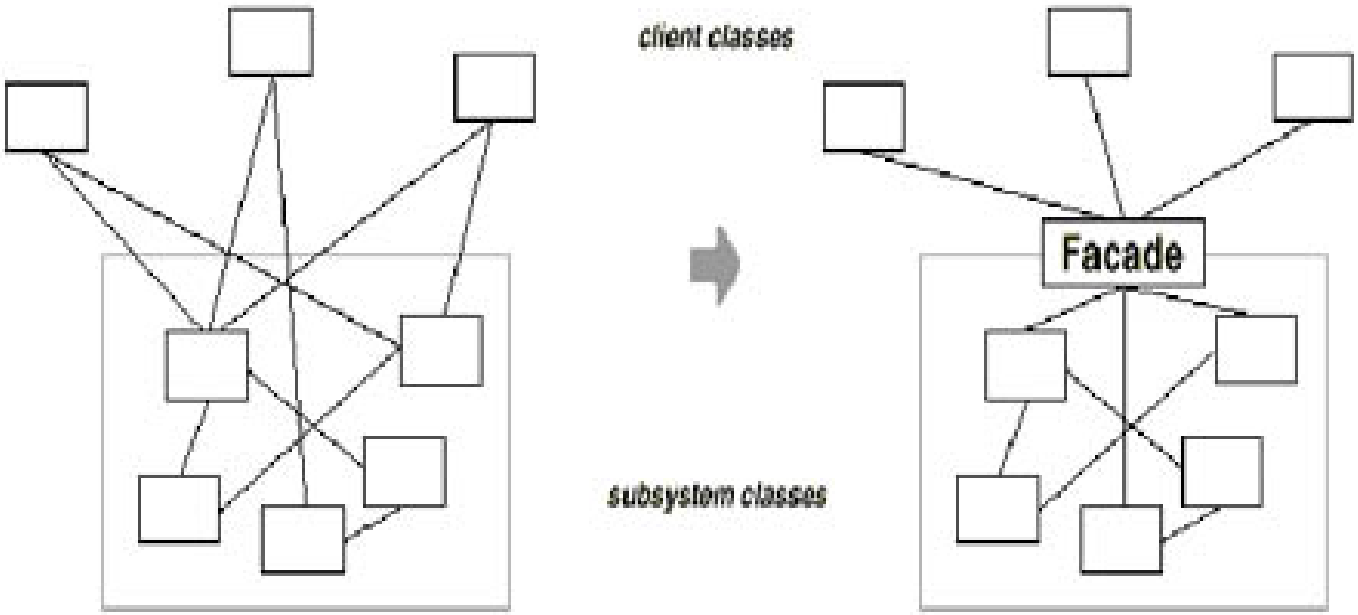
- Afactory的create方法封装了具体创建细节。
- 解耦了创建过程和使用过程，系统可扩展性增强，稳定性增强。
- Afactory的create方法代码：

```
public static AInterface create(){
        ……
      return new AClass();
}
```

EJB调用是工厂模式的实现

- 调用EJB 语法：
  EJBHome em = JNDIServer.getRemoteHome(EJB-JNDI-NAME);
  EJBObject myEJB = em.create();
- em.create()类似Afactory.create()；
- EJBObject 是接口

Façade模式



会话 Bean和实体Bean

- 一个会话Bean中调用多个实体Bean
- 该会话Bean是一个Façade类/Manager类
- 使用Façade 会话Bean优点：
  1. 提供性能，节省客户端直接调用实体Bean的网络开销
  2. 解耦分层，利于扩展变化。

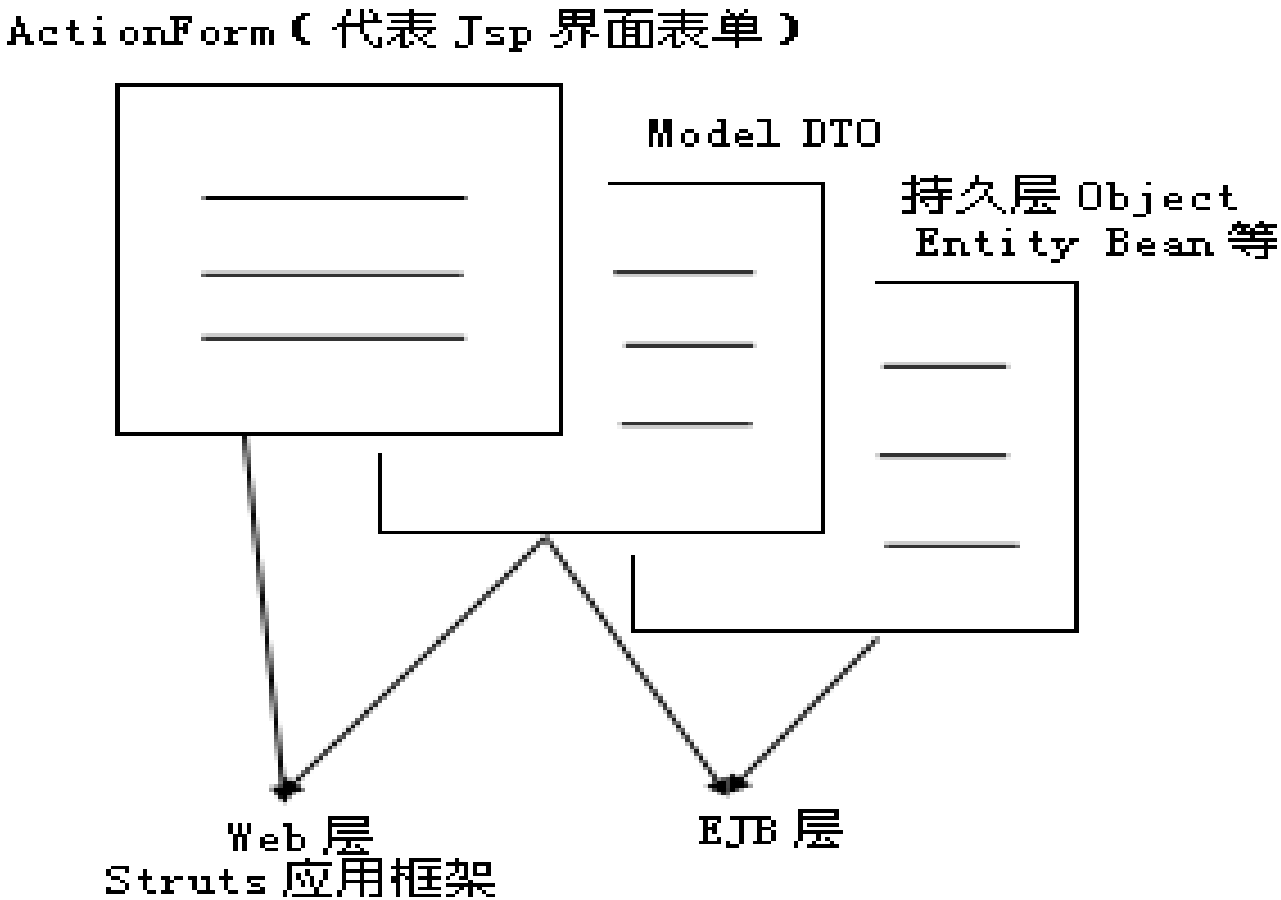
DTO模式

- DTO模式或称VO模式，是指将数据封装成普通的JavaBeans，在J2EE多个层次之间传输。
- DTO类似信使，是同步系统中的Message
- 该JavaBeans可以是一个数据模型Model

数据建模

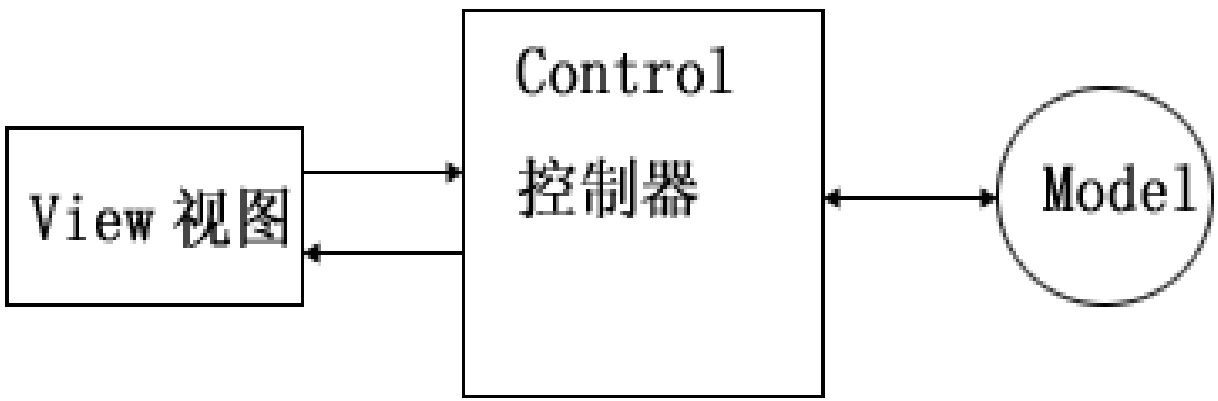
- Model 、 Domain Object以及DTO关系
  1. 分析提炼Model是系统设计之起端
  2. 系统设计之初，三者基本统一
  3. 系统复杂化后，DTO可能是多个Model组合实现；

Model与MDA


ActionForm（代表Jsp界面表单）
Model DTO
持久层Object
Entity Bean等
Web层
Struts应用框架
EJB层

MVC模式

- MVC模式是J2EE Web层的主要实现



Struts框架（Framework）

- Struts框架是MVC模式的实现（特例化）
- 框架（Framework）与模式（Pattern）的关系：
  1. 设计模式比框架更抽象；
  2. 设计模式是比框架更小的体系元素；
  3. 框架比设计模式更加特例化；

代理模式

- 代理模式是容器级别或框架级别的模式
- 代理模式可以强迫客户端对一个对象的方法调用间接通过代理类进行。
- 通常代理模式有以下几种：访问代理（Access Proxy）、虚拟代理和远程代理等。

代理模式

- 接口：
- public interface BaseIF{
- public Object myMethod();
- }
- 原始类：
- public class OriginClass implements BaseIF{
- public Object myMethod (){
- return " hello , It is me! ";
- }
- }

代理模式（续）

- 代理类：
- public class ProxyClass implements BaseIF{
- public Object myMethod (){
- //通过网络协议调用远程的OriginClass
- BaseIF instance = getRemoteOrigin ();
- Return instance.myMethod ();
- }
- …..
- }
- 客户端调用
- BaseIF instance = new Proxy Class();
- String = (String)instance.myMethod ();
- System.out.println(result); //将会输出 hello , It is me!

动态代理模式

- 动态代理利用Java的反射（Reflect）机制，可以在运行时刻将一个对象实例的方法调用分派到另外一个对象实例的调用。
- 动态代理模式可以在运行时刻创建继承某个接口的类型安全的代理对象，而无需在代码编译时编译这些代理类代码。是AOP的良好实现。
- 使用动态代理模式，基本可以实现完全解耦，例如JBoss 3.XX容器核心。

JdonSD框架

- JdonSD框架揉和了多种设计模式
- JdonSD框架融入了AOP、MDA最新设计思想
- 即将加入RBAC、工作流