**建造者模式 ：**

我们采用了建造者模式。建造者模式（Builder Pattern）使用多个简单的对象一步一步构建成一个复杂的对象。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。这里以announce为例，这里有Announces、AnnounceDao、AnnounceService和AnnounceAction，四个类依次递进，Announces包含一个公告最基本的属性，如

**private** Integer id;

**private** String announcement;

**private** String title;

**private** Date thetime;

AnnounceDao类中包含对声明的基本操作，

**public** List<Announces> getIndexAnno(**int** index);

**public** List<Announces> getAll();

**public** Announces find(**int** id);

**public** **void** update(Announces announce);

**public** **void** add(Announces announce);

AnnounceService类中是对声明的管理操作，

**public** List<Announces> getIndexAnno(**int** index);

**public** List<Announces> getAll();

**public** Pages ManageAllForPages(**int** pageSize, **int** nowPage);

**public** Announces find(**int** id);

**public** **void** ManageUpdate(Announces announce);

**public** **void** ManageAdd(Announces announce);

AnnounceAction类中是对这些操作的进一步封装。

这样做的好处有：

1.客户端不必知道产品内部组成的细节，将产品本身与产品的创建过程解耦，使得相同的创建过程可以创建不同的产品对象。

2.每一个具体建造者都独立，因此可以方便地替换具体建造者或增加新的具体建造者， 用户使用不同的具体建造者即可得到不同的产品对象 。

3.可以更加精细地控制产品的创建过程 。将复杂产品的创建步骤分解在不同的方法中，使得创建过程更加清晰，也更方便使用程序来控制创建过程。

4.增加新的具体建造者无须修改原有类库的代码，指挥者类针对抽象建造者类编程，系统扩展方便，符合“开闭”。

**中介者模式：**

中介者模式（Mediator Pattern）是用来降低多个对象和类之间的通信复杂性。这种模式提供了一个中介类，该类通常处理不同类之间的通信，并支持松耦合，使代码易于维护。中介者模式属于行为型模式。

我们项目还采用中介者模式，我们这里以Topic类为例，这里的TopicDao就充当中介者，Topic类中包含一些基本信息：

**private** **int** id;

**private** String title; // 标题

**private** String content; // 内容

**private** **int** countComment = 0; // 评论数

**private** Date topicTime; // 发布时间

**private** **int** niceTopic = 0; // 精品帖：0不是，1是

**private** **int** status = 0; // 帖子状态：0未结帖，1已结贴

**private** **int** integral = 10; // 默认帖子发布所耗积分

// 多对一

**private** Users topicsUser;

**private** Types topicsType;

// 集合，一对多

TopicDao对外提供了对Topic操作的接口

**public** **boolean** add(Topics topic);

**public** Topics find(**int** id);

**public** **boolean** update(Topics topic);

**public** List<Topics> getIndexFresh(**int** index);

**public** List<Topics> getAll();

**public** List<Topics> getIndexHot(**int** index);

**public** List<Topics> getByType(List<Types> listType);

**public** **void** delete(Topics topic);

**public** List<Topics> getIndexNice(**int** index);

TopicAction和TopicService就可以之间使用TopicDao提供的接口而非之间对Topic操作，把同事类原来一对多的依赖变成一对一的依赖，降低同事类的耦合度，同时也符合了迪米特原则。