5.外观模式

影院管理项目

组建一个家庭影院：

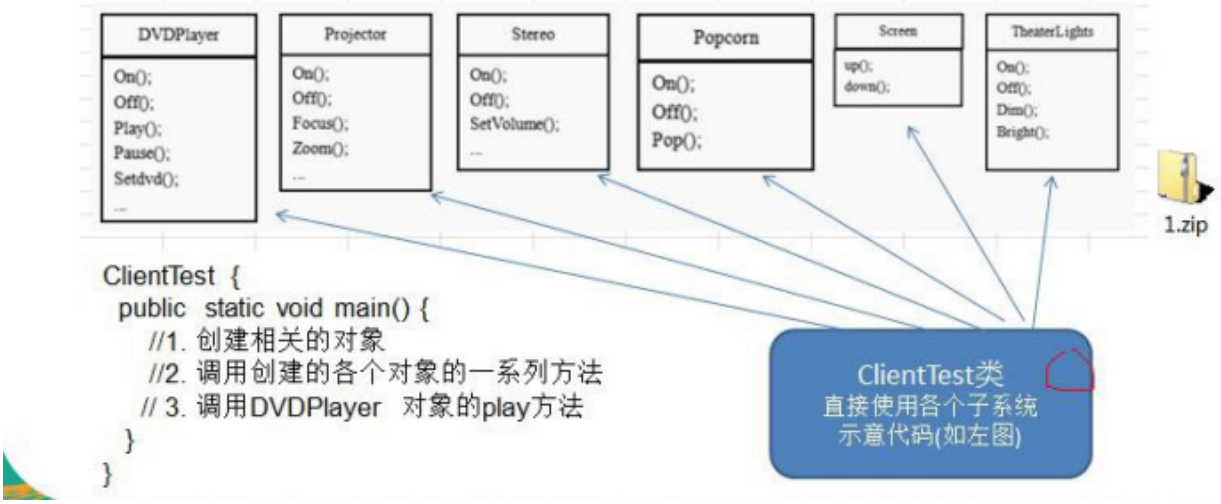
DVD 播放器、投影仪、自动屏幕、环绕立体声、爆米花机,要求完成使用家庭影院的功能，其过程为： 直接用遥控器：统筹各设备开关

开爆米花机放 下 屏 幕 开 投 影 仪 开音响开 DVD，选 dvd

去拿爆米花调 暗 灯 光 播放

观影结束后，关闭各种设备

传统方式解决影院管理



传统方式解决影院管理问题分析

1) 在 ClientTest 的 main 方法中，创建各个子系统的对象，并直接去调用子系统(对象)相关方法，会造成调用过程混乱，没有清晰的过程

2) 不利于在 ClientTest 中，去维护对子系统的操作

3) 解决思路：定义一个高层接口，给子系统中的一组接口提供一个一致的界面(比如在高层接口提供四个方法ready, play, pause, end)，用来访问子系统中的一群接口

4) 也就是说 就是通过定义一个一致的接口(界面类)，用以屏蔽内部子系统的细节，使得调用端只需跟这个接口发生调用，而无需关心这个子系统的内部

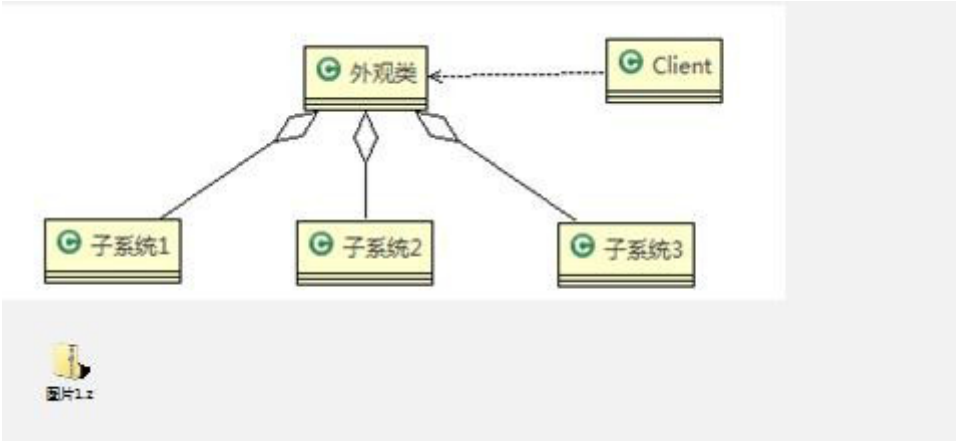
细节 => 外观模式

外观模式基本介绍

1） 外观模式（Facade），也叫“过程模式：外观模式为子系统中的一组接口提供一个一致的界面，此模式定义了一个高层接口，这个接口使得这一子系统更加容易使用

2） 外观模式通过定义一个一致的接口，用以屏蔽内部子系统的细节，使得调用端只需跟这个接口发生调用，而无需关心这个子系统的内部细节

外观模式原理类图



对类图说明(分类外观模式的角色)

1） 外观类(Facade)：为调用端提供统一的调用接口，外观类知道哪些子系统负责处理请求,从而将调用端的请求代理给适当子系统对象

2） 调用者(Client)：外观接口的调用者

3） 子系统的集合：指模块或者子系统，处理 Facade 对象指派的任务，他是功能的实际提供者

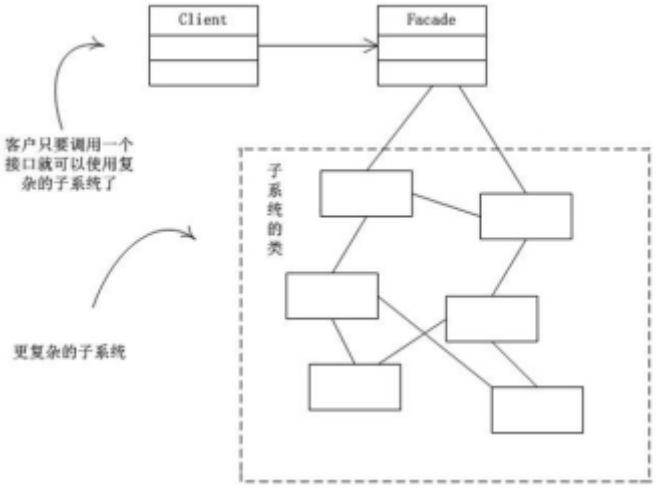
外观模式解决影院管理

传统方式解决影院管理说明

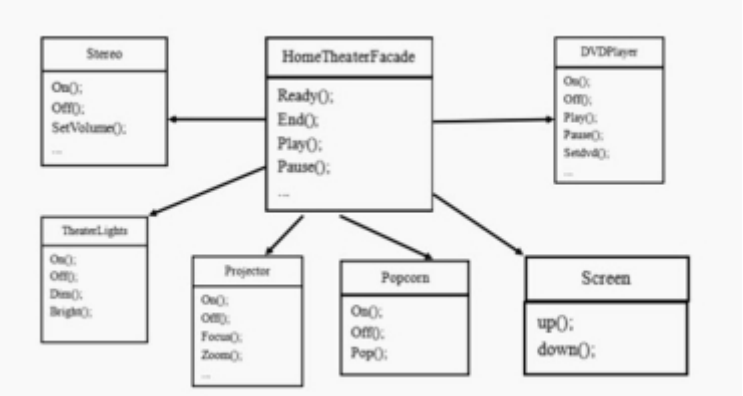
1） 外观模式可以理解为转换一群接口，客户只要调用一个接口，而不用调用多个接口才能达到目的。比如：在 pc 上安装软件的时候经常有一键安装

选项（省去选择安装目录、安装的组件等等），还有就是手机的重启功能（把关机和启动合为一个操作）。

2) 外观模式就是解决多个复杂接口带来的使用困难，起到简化用户操作的作用



外观模式应用实例




```
public class Screen {
    //使用饿汉式
   private static Screen instance = new Screen();
   public static Screen getInstance() {
      return instance;
   }
   private Screen(){}
   public void up() {
      System.out.println(" Screen up ");
   }
   public void down() {
      System.out.println(" Screen down ");
   }
}
```

Popcorn,Projector,等类与Screen类似,只有一些方法不同

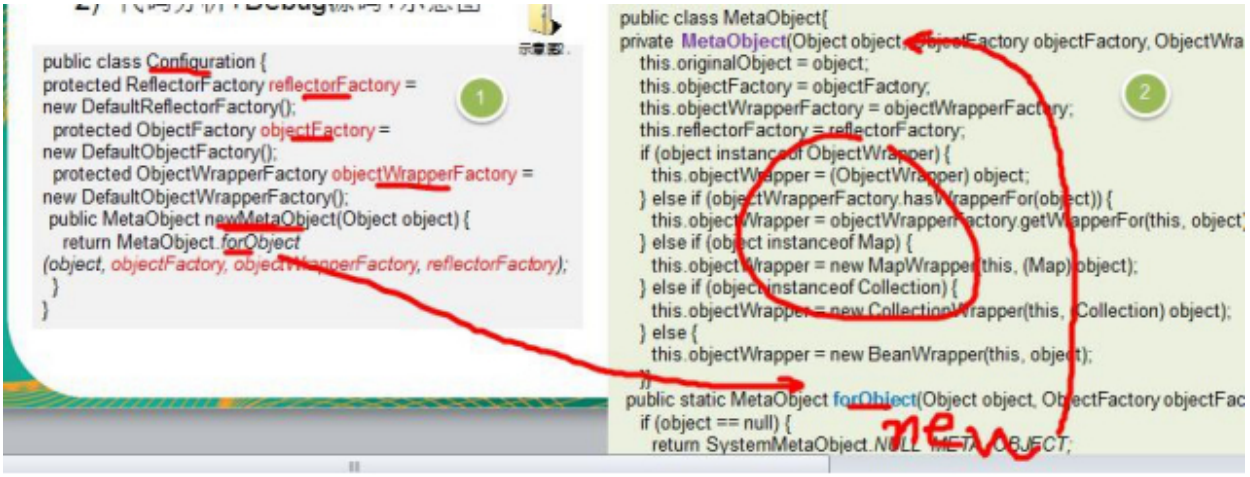
```
//外观类
public class HomeTheaterFacade {
   //定义各个子系统对象
```

```
  private TheaterLight theaterLight=null;
  private Screen screen=null;
   ……
  //构造器
  public HomeTheaterFacade() {
     super();
     this.theaterLight = TheaterLight.getInstance();
     this.screen = Screen.getInstance();
   …….
  }

  //操作分成 4 步
  public void ready() {
     popcorn.on();
     screen.down();
    ……
  }

  public void play() {
     dVDPlayer.play();
  }

  public void pause() {
     dVDPlayer.pause();
  }

  public void end() {
     theaterLight.bright();
     screen.up();
   ……
  }




}
```

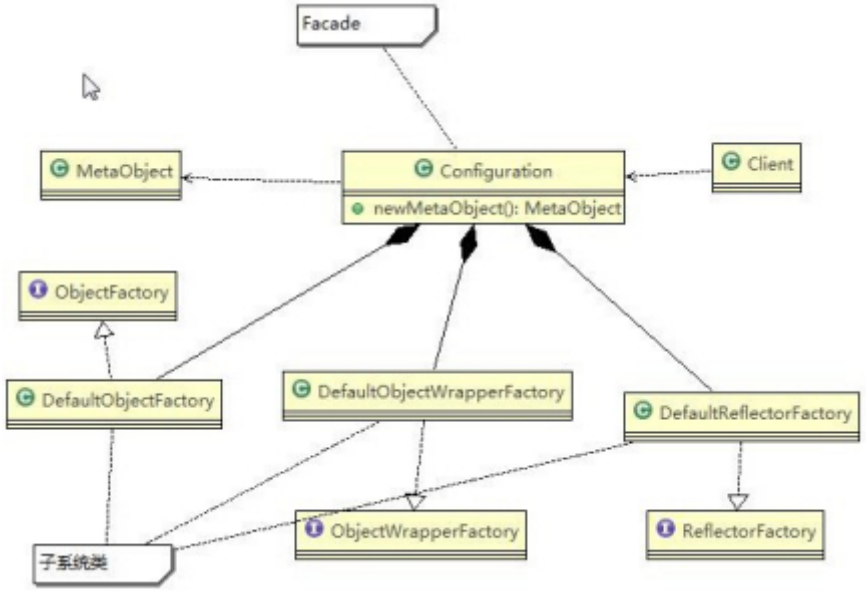
外观模式在 MyBatis 框架应用的源码分析

1） MyBatis 中的 Configuration 去创建 MetaObject 对象使用到外观模式

2） 代码分析+Debug 源码+示意图



1） 对源码中使用到的外观模式的角色类图



外观模式的注意事项和细节

1） 外观模式对外屏蔽了子系统的细节，因此外观模式降低了客户端对子系统使用的复杂性

2） 外观模式对客户端与子系统的耦合关系 - 解耦，让子系统内部的模块更易维护和扩展

3） 通过合理的使用外观模式，可以帮我们更好的划分访问的层次

4） 当系统需要进行分层设计时，可以考虑使用 Facade 模式

5） 在维护一个遗留的大型系统时，可能这个系统已经变得非常难以维护和扩展，此时可以考虑为新系统开发一个Facade 类，来提供遗留系统的比较清晰简单的接口，让新系统与 Facade 类交互，提高复用性

6） 不能过多的或者不合理的使用外观模式，使用外观模式好，还是直接调用模块好。要以让系统有层次，利于维护为目的。