1. 设计模式

- 创建型设计模式
    1. 抽象工厂(Abstract Factory)：优势在于创建一组相关联的对象，易于拓展
    2. 构建器(Builder)：优势在于创建复杂对象时，独立于创建组成部分的方式和组装的方式
    3. 工厂方法(Factory Method)：优势在于创建一个对象时，不同的子类有不同的创建方法，同时又可以共用父类的模板方法
    4. 原型(Prototype)：优势在于克隆效率高
    5. 单例(Singleton)：优势在于确保重量级的对象只有一个实例
- 结构型设计模式
    1. 适配器(Adapter)：用于转换接口
    2. 桥接(Bridge)：抽象类间持有引用，运行时动态调用
    3. 组合(Compoment)：用树状结构表达"部分与整体"的关系
    4. 装饰器(Decorator)：动态添加额外的职责
    5. 外观(Facade)：给复杂系统提供更简单的接口
    6. 享元(Flyweight)：共享同状态的对象，支持大量细粒度对象
    7. 代理(Proxy)：屏蔽被代理对象，用于控制访问
- 行为型设计模式
    1. 职责链(Chain of Responsibility)：不指明接收者，多个对象都有机会处理请求
    2. 命令(Command)：将调用者和执行者解耦，将请求单独封装
    3. 解释器(Interpreter)：设计简单文法并提供解释器
    4. 迭代器(Iterator)：提供统一遍历访问接口，隐藏内部实现细节
    5. 中介器(Mediator)：大量复杂通信中，统一都与中介进行交互
    6. 备忘录(Memento)：无需破坏对象封装的情况下，实现保存和恢复对象的snapshot
    7. 观察者(Observer)：被观察者抽象地通知观察者们，观察者实际地更新状态
    8. 状态(State)：仅通过切换子类实例，而不改变抽象接口，实现状态切换
    9. 策略(Strategy)：将不同算法单独封装，方便增减和切换
    10. 模板方法(Template Method)：定义好顶层设计模板，而只留下部分抽象/具体/钩子方法由继承类实现
    11. 访问者(Visitor)：两个类层次实现不同的人访问不同的内容有不同的方式

创建型设计模式	实例
抽象工厂(Abstract Factory)	支持多种观感地用户界面工具箱，游戏中的多风格系列场景
构建器(Builder)	游戏中地图和人物建造其各个组分
工厂方法(Factory Method)	
原型(Prototype)	复制粘贴
单例(Singleton)	数据库连接池，Hibernate中的sessionFactory
结构型设计模式	**实例**
适配器(Adapter)	外部接口修改
桥接(Bridge)	蜡笔和毛笔的故事

创建型设计模式	实例
组合(Composite)	文件系统
装饰器(Decorator)	定制新增的各种变化
外观(Facade)	一次性接受参数以提供简单接口
享元(Flyweight)	编辑器系统对字母的处理，Token对象
代理(Proxy)	文档编辑器对图像的显示，网络代理
行为型设计模式	**实例**
职责链(Chain of Responsibility)	拦截器
命令(Command)	菜单命令
解释器(Interpreter)	自定义文法
迭代器(Iterator)	各种语言中封装好的迭代器
中介器(Mediator)	对话框组件中的复杂依赖，WTO
备忘录(Memento)	事务
观察者(Observer)	时钟，监听
状态(State)	TCP连接，消息队列
策略(Strategy)	排序算法簇
模板方法(Template Method)	平台，框架，架构，领域中间件
访问者(Visitor)	编译器的语法检查