2019年3月30日 星期六

设计考试

主题

- 1. 工厂方法
- 2. 抽象工厂
- 3. 单例
- 4. 适配器
- 5. 装饰模式（重要）
- 6. 建造者模式：掌握，
  - 动机：
    - 复杂对象有成员属性，客户不希望了解建造过程且各部分有依赖关系
  - 模式结构：掌握指挥者作用：1. 希望分离建造与使用 2. 创建过程抽象，指定组件顺序，可以复用
- 7. 原型模式
  - 定义
  - 结构
  - 在java中的实现
  - 深克隆和浅克隆
- 8. 单例是重点
- **9. 桥接模式（必考）**
  - **给情景使用桥接模式**
  - 动机：产生不同行为的对象，
  - 与装饰者区别：通过原有的复杂的行为，拆分为可以独立变化的部分，从每部分中选取一个行为进行组合
  - 变化维度：

- 需要识别出抽象部分和实现部分 概念
  - 抽象化：抽象化出来的是实体本身，抽象掉的是实现部分
  - 脱耦：将抽象部分和实现部分
  - 跨平台视频播放器
- 优点
  - 桥接模式比多继承能更好遵循单一职责（策略、状态）
- 缺点
- 在什么情况下使用桥接模式
  - 每一个维度的变化在三个以上
- 适配器模式与桥接模式联用

- **10. 组合模式（重点，考察模式联用）**
  - **容器，只要是能用容器的模式都可以与组合模式联用**
  - **命令、迭代、观察，只要有数组**
- 11. 外观模式
  - 概念，与其他与接口打交道的模式的区别
  - 好处：“单一职责”，“迪米特法则”
  - 缺点：注意哪里不支持“开闭模式”
- 12. 享元模式（Flyweight）
  - 可以共享的相同内容：**内部状态，不能共享需要外部环境来设置的：外部状态（区别出）**
  - 外部状态与内部状态不一定要用组合才能构成一个对象，可以用各种方法，如构造注入，参数注入
  - 通常会出现工厂模式，享元工厂
  - 设计为较小的对象，内部状态较少，细粒度对象（fine-grained）
  - 实例二：共享网络设备

    - 外部状态通过参数来设置
  - 缺点：

- **13. 代理模式（可考的点很多）**
  - 动机：新的对象，实现对真实对象的操作，作为替身
  - Surrogate别名
  - 类图简单，主要是灵活运用
    - 注意Proxy中不是只能实现request方法，也可以再包括其他方法
  - 三种
    - **远程代理：数学运算代理**
      - **rmi流程**
        - **谁是代理**
        - **共同的接口从哪里来**
        - **如何注册目录服务**
        - **如何获取到本地？还是不需要管？**
    - 保护代理：判断权限 权限需要控制的
    - 虚拟代理
      - 需要创建一个非常大的对象时，用小对象代替大对象
      - 图片代理就是虚拟代理一种
      - 节省内存也可能会用

- 14. 命令模式
    - 分清命令模式中个角色职责
      - 发送者：Client 吃饭者
      - 接收者：Receiver 厨师
      - 调用者：Invoker：决定何时执行 服务员

    - Command：菜单
  - 与其他模式联用
  - 动机
    - 发送者与接收者接耦
    - 命令很多种，参数化
    - 队列相关
  - 很多任务，子任务很多，如何使用命令模式和其他模式（组合模式）
  - Undo和Redo怎么做
  - 宏命令，组合命令：数组实现（组合模式来替代数组）
  - 场景 uml类图
  - 鸭子
  - 联用的时候注意识别抽象的部分，很多时候是共享抽象部分
- 15. 观察者模式
  - 一对多的依赖关系好处
  - Java
    - Observable类，Observer是接口
    - 怎么notify
    - 最终使用这种策略的原因：注意有结合到Java本身的特点
  - 推和拉
  - 内部观察者和外部观察者（pattern of patterns）
  - 观察者支持**广播通信，通知哪些对象不一定总是同一组，可以有多个观察队列**
  - **通知部分的时候，可以用组合和迭代器**
  - **观察者为啥要保持对主题的引用**
    - **加加减减**

        - **拉数据**

- 16. 状态模式
    - 根据状态图给一个灵活的实现
    - 与策略差异
        - intent
        - 透明性：状态要对客户透明，策略不能对客户透明
    - 专门用一个抽象类来表示状态：单一职责原则（每种状态一个类）
    - 如果是通过判断状态属性数据来判断状态转变的，那么就去看**数据在哪边，可以通过调用持有分数数据的地方的方法来进行转换**

- **17.** 策略模式体现了什么设计原则

- 18. 模板方法
    - 其中的模板方法设为final，防止被改