- 设计模式
  - 设计原则
    - 0.单一职责原则 一个类承担一种职责
    - 1.开闭原则 对扩展开放, 对修改关闭
    - 2.里氏替换原则 LSP继承复用(子对父继承与新增)
    - 3.依赖倒转原则 针对接口编程,依赖抽象而不依赖具体: 把父类替换成子类,程序行为没发生变化
    - 4.接口隔离原则 使用多接口,降低依赖,降低耦合: 一个接口担当一种角色
    - 5.合成/聚合复用原则 尽量使用合成/聚合,而不是继承: 新对象包含已有对象, 避免修改父类,影响全部子类
    - 6.最小知识原则(迪米特) 模块相互独立,尽量少与其他模块相互作用
  - 创建型模式
    - 工厂模式 在创建对象时不会对客户端暴露创建逻辑，并且是通过使用一个共同的接口来指向新创建的对象。
    - 抽象工厂模式 工厂模式的超级组合
    - 单例模式 保证一个类仅有一个实例，并提供一个访问它的全局访问点
    - 建造者模式 将一个复杂的构建与其表示相分离，使得同样的构建过程可以创建不同的表示
    - 原型模式 从一个对象再创建另外一个可定制的对象,而且不知道任何创建的细节(深拷贝)
  - 结构型模式
    - 适配器模式 作为两个不兼容的接口之间的桥梁
    - 桥接模式 将抽象部分与实现部分分离，使它们都可以独立的变化
    - 过滤器模式 允许开发人员使用不同的标准来过滤一组对象，通过逻辑运算以解耦的方式把它们连接起来
    - 组合模式 将对象组合成树形结构以表示"部分-整体"的层次结构
    - 装饰器模式 为已有功能动态增加工作功能,把类的核心职责和装饰功能拆分开
    - 外观模式 为子系统中的一组接口提供一个一致的界面，用外观模式定义一个高层接口来便捷访问子系统
    - 享元模式 运用共享技术有效地支持大量细粒度的对象
    - 代理模式 为其他对象提供一种代理以控制对这个对象的访问
  - 行为模式
    - 责任链模式 避免请求发送者与接收者耦合在一起，让多个连接成一条链的对象都有可能接收请求.
    - 命令模式 将一个请求封装成一个对象，从而使您可以用不同的请求对客户进行参数化
    - 解释器模式 给定一个语言，定义它的文法表示，并定义一个解释器，这个解释器使用该标识来解释语言中的句子
    - 迭代器模式 提供一种方法顺序访问一个聚合对象中各个元素, 而又无须暴露该对象的内部表示
    - 中介者模式 提供了一个中介类, 用来降低多个对象和类之间的通信复杂性。
    - 备忘录模式 在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态
    - 观察者模式 对象间的一种一对多关系，当一个对象的状态发生改变时，所有依赖对象都得到通知并被自动更新
    - 状态模式 对象的行为依赖于它的状态（属性），并且可以根据它的状态改变而改变它的相关行为
    - 策略模式 一个类的行为或其算法可以在运行时更改.
    - 模板模式 一个抽象类公开定义了执行它的方法的方式/模板。它的子类可以按需要重写方法实现.
    - 访问者模式 主要将稳定的数据结构与易变数据操作分离
  - 表示模式
    - MVC模式 典型的MVC三层结构
    - 业务代表模式 用于对表示层和业务层解耦
    - 组合实体模式 当更新一个组合实体时，内部依赖对象会自动更新
    - 数据访问对象模式 把低级的数据访问 API 或操作从高级的业务服务中分离出来
    - 传输对象模式 从客户端向服务器一次性传递带有多个属性的数据
  - UML类图建模
    - 图例三层
      - 1. 类的名称: 抽象类用斜体
      - 2. 类的特性: 字段和属性
      - 3. 类的操作: 方法和行为
    - 符号意义
      - 1. + 表示public
      - 2. - 表示private
      - 3. # 表示pretected
      - 4. <<interface>> 接口
      - 5. ↑ 表示继承
    - 关系
      - 1. 继承关系
      - 2. 实现接口
      - 3. 依赖关系
      - 4. 聚合关系
      - 5. 关联关系