创建型模式

单例模式

- 定义
  - 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
- 关键点
  - 私有构造器
  - 线程安全
  - 延迟加载
  - 序列化与反序列化安全 — 定义 readResolve() 方法，返回单例对象
  - 反射安全 — 对于饿汉式或静态内部类方式，在构造函数中判断单例对象是否不为空，如果不为空就抛出异常
  - 唯一性前提 — 类被同一个类加载器加载
- 实现方式
  - 饿汉式 — 不支持延迟加载
  - 懒汉式 — 频繁调用（加锁、解锁）会产生性能瓶颈
  - 双重加锁 — 使用 volatile 关键字避免初始化重排序
  - 静态内部类 — 基于类的初始化锁
  - 枚举（推荐） — 可以避免使用序列化/反序列化或反射来创建多个实例
- 扩展
  - 线程唯一 — 使用 ThreadLocal
  - 多例模式：通过 Map 来存储对象类型和对象之间的对应关系，来控制对象的个数
- 适用场景
  - 生产唯一序列号
  - web 中的计数器
  - 配置信息类、连接池类
- 优缺点
  - 优点
    - 只有一个实例，减少了内存开销
    - 避免对资源的多重占用
    - 设置全局访问点，严格控制访问
  - 缺点 — 没有接口，扩展困难，与单一职责原则冲突
- 源码应用
  - JDK
    - Runtime
    - Unsafe
  - Spring — AbstractFactoryBean
  - Mybatis — ErrorContext
- 对比
  - 静态类 — 仅在类自给自足，且不依赖于复杂的初始化时，才可以考虑使用
  - 全局变量
    - 不能延迟初始化
    - 不能确保只有一个实例
    - 污染命名空间

工厂方法模式

- 定义 — 定义一个用于创建对象的接口，让子类决定将哪一个类实例化。工厂方法使一个类的实例化延迟到其子类
- 适用场景
  - 创建对象需要大量重复的代码
  - 客户端不依赖于产品类实例如何被创建
  - 一个类通过其子类来指定创建哪个对象
- 优缺点
  - 优点
    - 屏蔽产品具体实现，调用者只关心产品接口
    - 加入新产品只需新增工厂类，符合开闭原则
  - 缺点 — 类的个数容易过多，增加复杂度
- 源码应用
  - JDK — ThreadFactory
  - SLF4J — LoggerFactory

原型模式

- 定义 — 用原型实例指定创建对象的种类，并且通过拷贝这个原型来创建新的对象
- 要素
  - 实现 Cloneable 接口
  - 重写 Object的 clone 方法
- 适用场景
  - 类初始化消耗较多资源
  - new 对象过程繁琐（数据准备、访问权限等）
  - 构造函数比较复杂
  - 循环体中创建大量对象
- 优缺点
  - 优点
    - 性能比直接 new 对象性能高
    - 简化创建过程
  - 缺点
    - 必须配备克隆方法
    - 对克隆复杂对象或克隆出的对象进行复杂改造时，容易引入风险
    - 深拷贝、浅拷贝要运用得当
- 源码应用
  - JDK — ArrayList/HashMap

建造者模式

- 定义 — 将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示
- 适用场景
  - 需要生成的对象具有复杂的内部结构
  - 需要生成的对象内部属性本身相互依赖
- 优缺点
  - 优点
    - 建造者独立，易扩展
    - 便于控制细节风险
  - 缺点
    - 产品必须有共同点，范围有限制
    - 如内部变化复杂，会有很多的建造类
- 源码应用
  - JDK — StringBuilder/StringBuffer
  - Guava — ImmutableSet
  - Spring — BeanDefinitionBuilder
  - Elasticsearch — IndexQueryBuilder

抽象工厂模式

- 定义 — 提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类
- 适用场景
  - 客户端不依赖于产品类实例如何被创建
  - 强调一系列相关的产品对象一起使用创建对象需要大量重复的代码
  - 提供一个产品类的库，所有的产品以同样的接口出现，从而使客户端不依赖于具体实现
- 优缺点
  - 优点
    - 具体产品在应用层代码隔离，无需关心创建细节
    - 保证客户端始终只使用同一个产品族中的对象
  - 缺点 — 产品族中扩展新产品困难，需修改抽象工厂接口
- 源码应用
  - JDK — Connection
  - Mybatis — SqlSessionFactory