设计模式

1、你所知道的设计模式有哪些？

- 参考回答：
    - **创建型模式，共五种**：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。
    - **结构型模式，共七种**：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。
    - **行为型模式，共十一种**：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录　模式、状态模式、访问者模式、中介者模式、解释器模式。

2、谈谈 MVC、MVP 和 MVVM，好在哪里，不好在哪里　？

- 参考回答：
    - **MVC**:
        - 视图层(View)　　对应于 xml 布局文件和 java 代码动态 view 部分
        - 控制层(Controller)　　MVC 中 Android 的控制层是由 Activity 来承担的，Activity 本来主要是作为初始化页面，展示数据的操作，但是因为 XML 视图功能太弱，所

以 Activity 既要负责视图的显示又要加入控制逻辑，承担的功能过多。

- 模型层(Model)　针对业务模型，建立数据结构和相关的类，它主要负责网络请求，数据库处理，I/O 的操作。

- 总结

  - 具有一定的分层，model 彻底解耦，controller 和 view 并没有解耦层与层之间的交互尽量使用回调或者去使用消息机制去完成，尽量避免直接持有　　controller 和 view 在 android 中无法做到彻底分离，但在代码逻辑层面一定要分清业务逻辑被放置在 model 层，能够更好的复用和修改增加业务。

- **MVP**

  - 通过引入接口 BaseView，让相应的视图组件如 Activity，Fragment 去实现 BaseView，实现了视图层的独立，通过中间层 Preseter 实现了 Model 和 View 的完全解耦。MVP **彻底解决**了 MVC 中 View 和 Controller 傻傻分不清楚的问题，但是随着业务逻辑的增加，一个页面可能会非常复杂，UI 的改变是非常多，会有非常多的 case，这样就会造成 View 的接口会很庞大。

- **MVVM**
    - MVP 中我们说过随着业务逻辑的增加，UI 的改变多的情况下，会有非常多的跟 UI 相关的 case，这样就会造成 View 的接口会很庞大。而 MVVM 就解决了这个问题，通过双向绑定的机制，实现数据和 UI 内容，只要想改其中一方，另一方都能够及时更新的一种设计理念，这样就省去了很多在 View 层中写很多 case 的情况，只需要改变数据就行。
- 三者如何选择？
    - 如果项目简单，没什么复杂性，未来改动也不大的话，那就不要用设计模式或者架构方法，只需要将每个模块封装好，方便调用即可，不要为了使用设计模式或架构方法而使用。
    - 对于偏向展示型的 app，绝大多数业务逻辑都在后端，app 主要功能就是展示数据，交互等，建议使用 mvvm。
    - 对于工具类或者需要写很多业务逻辑 app，使用 mvp 或者 mvvm 都可。

3、封装 p 层之后.如果 p 层数据过大,如何解决?

- 参考回答：
    - 对于 MVP 模式来说，P 层如果数据逻辑过于臃肿，建议引入 **RxJava** 或则 **Dagger**，越是复杂的逻辑，越能体现 RxJava 的优越性

4、是否能从 Android 中举几个例子说说用到了什么设计模式 ？

- 参考回答：
    - **AlertDialog、Notification 源码中使用了 Builder（建造者）模式完成参数的初始化**
    - **Okhttp 内部使用了责任链模式来完成每个 Interceptor 拦截器的调用**
    - **RxJava 的观察者模式；单例模式；GridView 的适配器模式；Intent 的原型模式**
    - **日常开发的 BaseActivity 抽象工厂模式**

5、装饰模式和代理模式有哪些区别 ？

- 参考回答：
    - 装饰器模式与代理模式的区别就在于
        - 两者都是对类的方法进行**扩展**，但**装饰器模式**强调的是增强自身,在被装饰之后你能够在被增强的类上使用增强后的功能。

        - 而**代理模式**则强调要让别人帮你去做一些本身与你业务没有太多关系的职责（记录日志、设置缓存）代理模式是为了实现对象的控制，因为被代理的对象往往难以直接获得或者是其内部不想暴露出来。

6、实现单例模式有几种方法 ？懒汉式中双层锁的目的是什么 ？两次判空的目的又是什么 ？

- 参考回答：
    - 单例模式实现方法有多种：**饿汉，懒汉(线程安全，线程非安全)，双重检查(DCL),内部类，以及枚举**
    - 所谓**双层检验锁（在加锁前后对实例对象进行两次判空的检验）**：加锁是为了第一次对象实例化的线程同步，而锁内还要有第二层判空是因为可能会有多个线程进入第一层 if 判断内部，而在加锁代码块外排队等候，如果锁内不进行第二次检验，仍然会出现实例化多个对象的情况。

7、用到的一些开源框架，介绍一个看过源码的，内部实现过程。

- 参考回答：
    - 面试常客：Okhttp，Retrofit，Glide，RxJava，GreenDao，Dagger 等

8、Fragment 如果在 Adapter 中使用应该如何解耦？

- 参考回答：
    - 接口回调
    - 广播