跨进程

ContentProvider了解多少?

- 技术点：ContentProvider
- 思路：ContentProvider功能
- 参考回答：
  作为四大组件之一，ContentProvider主要负责存储和共享数据。与文件存储、SharedPreferences存储、SQLite数据库存储这几种数据存储方法不同的是，后者保存下的数据只能被该应用程序使用，而前者可以让不同应用程序之间进行数据共享，它还可以选择只对哪一部分数据进行共享，从而保证程序中的隐私数据不会有泄漏风险。 引申：谈谈ContentProvider底层使用Binder机制原理

Android中提供哪些数据持久存储的方法?

- 技术点：数据持久化
- 思路：分条解释每种数据持久存储的特点
- 参考回答：Android平台实现数据存储的常见几种方式：
  - File 文件存储：写入和读取文件的方法和 Java中实现I/O的程序一样。
  - SharedPreferences存储：一种轻型的数据存储方式，常用来存储一些简单的配置信息，本质是基于XML文件存储key-value键值对数据。
  - SQLite数据库存储：一款轻量级的关系型数据库，它的运算速度非常快，占用资源很少，在存储大量复杂的关系型数据的时可以使用。
  - ContentProvider：四大组件之一，用于数据的存储和共享，不仅可以让不同应用程序之间进行数据共享，还可以选择只对哪一部分数据进行共享，可保证程序中的隐私数据不会有泄漏风险。

Java中的I/O流读写怎么做?

- 技术点：数据持久化（文件存储）
- 思路：大致介绍核心类和核心方法
- 参考回答：
  和 Java中实现I/O的程序是一样的，Context类中提供了openFileInput()和openFileOutput()方法来打开数据文件里的文件IO流

SharePreferences适用情形？使用中需要注意什么?

- 技术点：数据持久化（SharePreferences存储）
- 参考回答：
  SharePreferences是一种轻型的数据存储方式，适用于存储一些简单的配置信息，如int、string、boolean、float和long。由于系统对SharedPreferences的读/写有一定的缓存策略，即在内存中有一份该文件的缓存，因此在多进程模式下，其读/写会变得不可靠，甚至丢失数据。

Android中进程和线程的关系?

- 技术点：进程、线程
- 关系：一个APP一般对应一个进程和有限个线程
  - 一般对应一个进程，当然，可以在AndroidMenifest中给四大组件指定属性android:process开启多进程模式
  - 有限个线程：线程是一种受限的系统资源，不可无限制的产生且线程的创建和销毁都有一定的开销。

为何需要进行IPC？多进程通信可能会出现什么问题?

- 技术点：多进程通信
- 思路：讨论多进程通信会出现的问题得出IPC的必要性
- 参考回答：
  - （1）多进程造成的影响可总结为以下四方面：
    - 静态变量和单例模式失效：由独立的虚拟机造成
    - 线程同步机制失效：由独立的虚拟机造成
    - SharedPreference的不可靠下降：不支持两个进程同时进行读写操作，即不支持并发读写，有一定几率导致数据丢失
    - Application多次创建： Android系统会为新的进程分配独立虚拟机，相当于系统又把这个应用重新启动了一次。
  - （2）需要进程间通信的必要性：所有运行在不同进程的四大组件，只要它们之间需要通过内存在共享数据，都会共享失败。这是由于Android为每个应用分配了独立的虚拟机，不同的虚拟机在内存分配上有不同的地址空间，这会导致在不同的虚拟机中访问同一个类的对象会产生多份副本。

什么是序列化？Serializable接口和Parcelable接口的区别？为何推荐使用后者?

- 技术点：序列化
- 参考回答：
  序列化表示将一个对象转换成可存储或可传输的状态。序列化后的对象可以在网络上进行传输，也可以存储到本地。

- 应用场景：需要通过Intent和Binder等传输类对象就必须完成对象的序列化过程。

- 两种方式：实现Serializable/Parcelable接口。
  不同点：

name	Serializable接口	Parcelable接口
平台	Java的序列化接口	Android序列化接口
序列化原理	将对象转换成可存储或可传输的状态	将一个对象进行分解，且分解的每一部分都是传递可支持的数据类型
优缺点	简单、效率低、开销大。 由于序列化和反序列化都需要进行大量的IO操作	高效，但是使用麻烦
使用场景	适合将对象序列化到存储设备，将对象序列化后通过网络设备传输	主要用在内存的序列化

Android中为何新增Binder来作为主要的IPC方式?

- 技术点：Binder机制
- 思路：回答Binder优点
- 参考回答：Binder机制有什么几条优点：
    - 传输效率高、可操作性强：传输效率主要影响因素是内存拷贝的次数，拷贝次数越少，传输速率越高。从Android进程架构角度分析：对于消息队列、Socket和管道来说，数据先从发送方的缓存区拷贝到内核开辟的缓存区中，再从内核缓存区拷贝到接收方的缓存区，一共两次拷贝。而对于Binder来说，数据从发送方的缓存区拷贝到内核的缓存区，而接收方的缓存区与内核的缓存区是映射到同一块物理地址的，节省了一次数据拷贝的过程。由于共享内存操作复杂，综合来看，Binder的传输效率是最好的。
    - 实现C/S架构方便：Linux的众IPC方式除了Socket以外都不是基于C/S架构，而Socket主要用于网络间的通信且传输效率较低。Binder基于C/S架构 ，Server端与Client端相对独立，稳定性较好。
    - 安全性高：传统Linux IPC的接收方无法获得对方进程可靠的UID/PID，从而无法鉴别对方身份；而Binder机制为每个进程分配了UID/PID且在Binder通信时会根据UID/PID进行有效性检测。

使用Binder进行数据传输的具体过程?

- 技术点：Binder机制
- 思路：通过AIDL实现方式解释Binder数据传输的具体过程
- 参考回答：
  服务端中的Service给与其绑定的客户端提供Binder对象，客户端通过AIDL接口中的asInterface()将这个Binder对象转换为代理Proxy，并通过它发起RPC请求。客户端发起请求时会挂起当前线程，并将参数写入data然后调用transact()，RPC请求会通过系统底层封装后由服务端的onTransact()处理，并将结果写入reply，最后返回调用结果并唤醒客户端线程。

是否了解AIDL？原理是什么？如何优化多模块都使用AIDL的情况?

- 技术点：AIDL

- 思路：
- 参考回答：
  - AIDL(Android Interface Definition Language，Android接口定义语言)：如果在一个进程中要调用另一个进程中对象的方法，可使用AIDL生成可序列化的参数，AIDL会生成一个服务端对象的代理类，通过它客户端实现间接调用服务端对象的方法。
  - AIDL的本质是系统提供了一套可快速实现Binder的工具。关键类和方法：
    - AIDL接口：继承IInterface。
    - Stub类：Binder的实现类，服务端通过这个类来提供服务。
    - Proxy类：服务器的本地代理，客户端通过这个类调用服务器的方法。
    - asInterface()：客户端调用，将服务端的返回的Binder对象，转换成客户端所需要的AIDL接口类型对象。返回对象：
      - 若客户端和服务端位于同一进程，则直接返回Stub对象本身；
      - 否则，返回的是系统封装后的Stub.proxy对象。

    - asBinder()：根据当前调用情况返回代理Proxy的Binder对象。
    - onTransact()：运行服务端的Binder线程池中，当客户端发起跨进程请求时，远程请求会通过系统底层封装后交由此方法来处理。
    - transact()：运行在客户端，当客户端发起远程请求的同时将当前线程挂起。之后调用服务端的onTransact()直到远程请求返回，当前线程才继续执行。

  - 当有多个业务模块都需要AIDL来进行IPC，此时需要为每个模块创建特定的aidl文件，那么相应的Service就会很多。必然会出现系统资源耗费严重、应用过度重量级的问题。解决办法是建立Binder连接池，即将每个业务模块的Binder请求统一转发到一个远程Service中去执行，从而避免重复创建Service。
    - 工作原理：每个业务模块创建自己的AIDL接口并实现此接口，然后向服务端提供自己的唯一标识和其对应的Binder对象。服务端只需要一个Service，服务器提供一个queryBinder接口，它会根据业务模块的特征来返回相应的Binder对像，不同的业务模块拿到所需的Binder对象后就可进行远程方法的调用了

Android中有哪些基于Binder的IPC方式？简单对比下？

name	优点	缺点	试用场景
Bundle	简单易用	只能传输Bundle支持的类型	四大组件间的进程间的通讯
文件共享	简单易用	不适合高并发场景，并且无法做到进程间的即时通讯	无并发访问场景，交换简单的数据实时性不高的场景
AIDL	功能强大，支持一对多并发通讯，支持实时通讯	使用稍复杂，需要处理好线程同步	一对多通讯，且有RPC需求
Messenger	功能一般，支持一对多串行通讯，支持实时通讯	不能很好的处理高并发情形，不支持RPC，数据通过Messenge进行传输，因此只能支持Bundle支持的类型	低并发一对多即时通讯，无RPC需求，或者无需返回结果的RPC需求
ContentProvider	在数据源访问方面功能强大，支持一对多并发数据共享，可通过Call方法扩展其他操作	可以理解为受约束的AIDL，主要提供数据源的CRUD操作	一对多进程间的数据共享