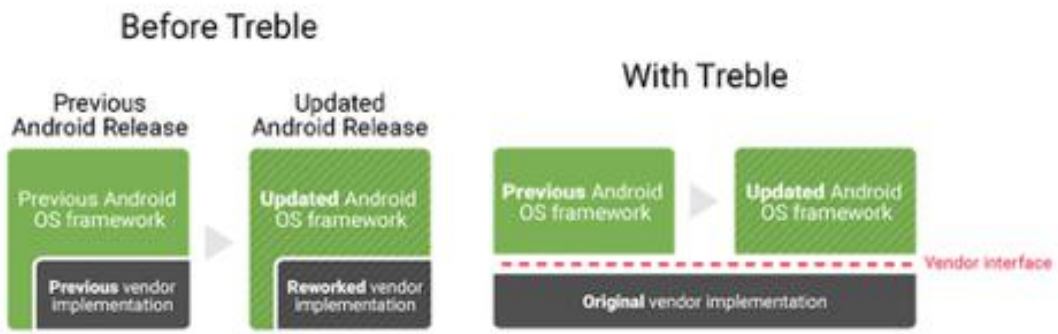
1、Android 8.0中提出了Project Treble，试分析：

(1) 提出Project Treble的目的

(2) 新架构的主要变化

答：（1）提出目的：Project Treble是谷歌在Android 8.0(Android O)开始定义的一个技术框架，服务于Android版本的升级。在原来的Android系统中，驱动、重要的运行库等都必须和系统版本严格对应。对于同样的硬件，适配于一个版本系统的驱动通常无法在另一个版本的系统中运行。同时Android设备的部分模块的驱动往往并不会面向公众提供，而是由芯片厂商直接以代码的形式交付给手机厂商，然后再由手机厂商直接把驱动整合到做好的系统更新里去。由于上述原因，谷歌发布新的Android版本的流程变得十分繁琐、缓慢，同时还进一步加大了Android系统的碎片化。Project Treble提出的目的是解除驱动和系统版本的紧密耦合，使厂商可以推出长期兼容未来新版本的驱动，并且保证它能够在以后的新版Android中无需修改也能正常使用。

（2）. 主要内容(答出2点及以上即可)：



(i) 新的HAL类型:Project Treble提出了绑定式 HAL、直通 HAL、与Same-Process (SP) HAL。

(ii) HIDL的提出：包含关于HAL接口定义语言（简称 HIDL）的一般信息，HIDL是用于指定HAL和其用户之间接口的接口描述语言 (IDL)。此外，Project Treble还包含了关于为 HIDL接口创建C++实现的详情与关于HIDL接口的Java前端的详情。

(iii) ConfigStore HAL：Project Treble 提供了关于ConfigStore HAL的说明，该HAL提供了一组API，可供访问用于配置Android框架的只读配置项。

(iv) 设备树叠加层：Project Treble提供了关于在Android中使用设备树叠加层(DTO)的详情。

(v) 供应商原生开发套件(VNDK)：Project Treble提供了关于VNDK（专门用来让供应商实现其HAL的一组库）的说明。

(vi) 供应商接口对象(VINTF)：Project Treble定义了VINTF对象，这些VINTF对象整合了关于设备的相关信息，并让这类信息可通过可查询API提供。

(vii)SELinux for Android 8.0：提供了关于 SELinux 变更和自定义的详情。

2、（1）Android 为什么设计 Binder 通信机制（结合 Linux 提供的通信机制来分析）

（2）Binder 只需要一次数据拷贝的秘密

答：（1）设计 Binder 主要有两方面原因，一是传输性能，二是安全性能

Linux 本身提供的通信机制有很多，比如管道、信号、消息队列、共享内存和插口（socket）等。但是这些通信机制都存在着一些问题。

从传输性能看，插口（socket）传输效率低，开销大，主要用在跨网络的进程间通信和本机上进程间的低速通信。消息队列和管道采用存储-转发方式，即数据先从发送方缓存区拷贝到内核开辟的缓存区中，然后再从内核缓存区拷贝到接收方缓存区，至少有两次拷贝过程。共享内存虽然只有一次拷贝，但是一般要结合其他进程间通信机制来同步信息。

从安全性考虑，传统 IPC 的接收方无法获得对方进程的可靠 UID/PID（用户/进程 ID），无法鉴别对方身份，只能由用户在数据包里填入 UID/PID，不可靠。其次传统 IPC 访问接入点是开放的，无法建立私有通道。因此安卓需要建立一套新的 IPC 机制来满足系统对传输性能和安全性的要求

（2）驱动为接收方分担了最为繁琐的任务：分配/释放大小不等，难以预测的有效负荷缓存区，而接收方只需要提供缓存来存放大小固 定，可以预测的消息头即可。在效率上，由于 mmap()分配的内存是映射在接收方用户空间里的，所有总体效果就相当于对有效负荷数据做了一次从发送方用户 空间到接收方用户空间的直接数据拷贝，省去了内核中暂存这个步骤，提升了一倍的性能。mmap()分配的内存除了映射进了接收方进程里，还映射进了内核空间。所以调用 copy_from_user()将数据拷贝进内核 空间也相当 于拷贝进了接收方的用户空间，这就是 Binder 只需一次拷贝的‘秘密’。

3、请简述 Zygote 启动 system_server 的过程

答：(1)ZygoteInit fork 出一个新的进程，这个进程就是 SystemServer 进程。

(2)fork 出来的子进程在 handleSystemServerProcess 里开始初始化工作，初始化分两步，一部在 native 完成，另外一部分（大部分）在 Java 端完成。 native 端的工作在 AppRuntime(AndroidRuntime 的子类)::onZygoteInit()完成，做的一件事情就是启动了一个 Thread，这个 Thread 是 SystemServer 的主线程，负责接收来至其他进程的 Binder 调用请求。

(3)nativeZygoteInit() 完成后，接下来开始 Java 层的初始化，这个流程比较长，也比较复杂，我们分成很多步进行讲解。初始化的入口是 SystemServer 的 main()函数，这里又调用了 Native 的 Init1(). Init1 实现在 com_android_server_SystemServer.cpp， 最终调用到的函数是 system_init()。

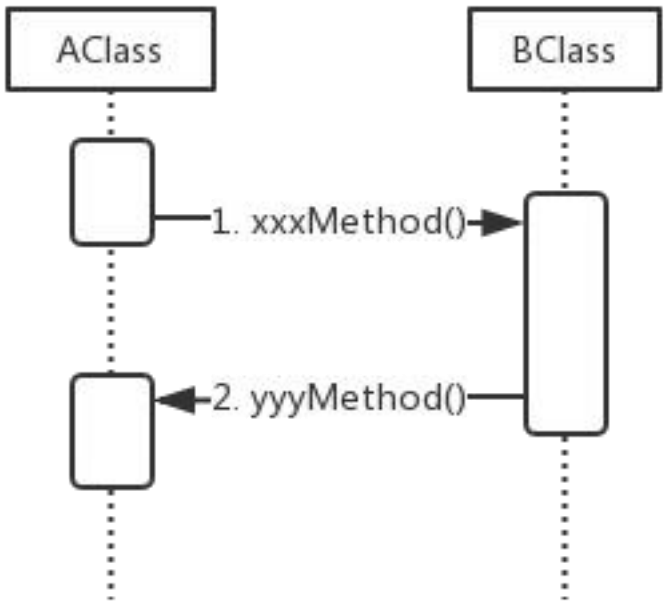
(4)system_init() 最后，join_threadpool() 将当前线程挂起，等待 binder 的请求。

(5)init2: 至此，system server 的 native 初始化工作完成，又重新回到了 Java 端，在这里，很多非常重要的系统服务将被启动。 这些工作将在一个新的线程内开始，线程名"android.server.ServerThread"。在 ServerThread 里，SystemServer 首

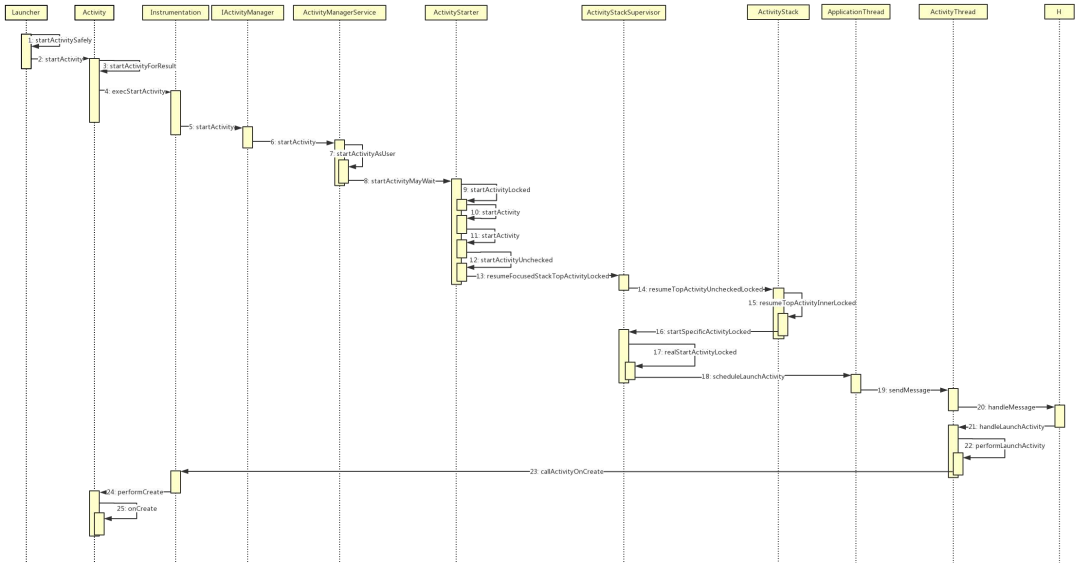
先创建了两个线程，UI thread 和 WindowManager thread，这两个 thread 的 handle 将被传给某些 Service 的构造函数，部分的启动工作会分发到这两个 Thread 内进行。
(6)接下来，System Server 会启动一系列的 Service，其中最重要的就是 Acitivity Manager 和 Window Manager。
(7)ActivityStack 是存在当前运行 Activity 的栈，resumeTopActivityLocked() 从其中找到要启动的那一个，如果该应用从来没有启动过，我们需要通过 AcitivyManagerService 为其创建一个进程。

4、 根据 ppt 上关于 activity 启动源码的粗略讲解，以及自己查阅 android8.0 中 activity 启动　相关源码，绘制该过程中（activity 启动）涉及到的类与主要方法的时序图。

附：时序图示例



答　　：



5、在介绍 4 大组件的时候，都有涉及到了一个组件 ActivityManagerService，该组件的主要功能有哪些？

答：(1)统一调度个应用的程序的 Activity。应用程序要运行 Activity，会首先报告 AMS，然后由 AMS 决定该 Activity 是否可以启动，如果可以，AMS 再通知应用程序指定的 Activity。换句话说说运行 Activity 时各应用程序的内政，AMS 并不干预，但是 AMS 必须知道各应用程序都运行了哪些 Activity。

(2)内存管理。Android 官方声称，Activity 退出后，其所在的进程并不会被立即杀死，从而在下次启动该 Activity 时能够提高启动速度，这些 Activity 只有当系统内存紧张时，才会被自动杀死，应用程序并不关心这些问题，这些都是在 AMS 中完成的。

(3)进程管理，AMS 向外提供了查询系统正在运行的进程信息的 API。

6、（1）简述 looper、handler、MessageQueue 以及 Message 的关系。
（2）异步线程通信机制的设计目的

答：（1）每一个 looper 中都包含着一个 MessageQueue 的引用，用来存放 Message 对象；handler 是用来向 MessageQueue 中插入消息的对象，handler 在哪个 looper 线程下创建，就持有 looper 引用，也就意味着持有 MessageQueue 的引用，那么 handler 就可以向 MessageQueue 中插入消息；最终处理消息的也是 handler，每一个 Message 对象中包含了发送它的 handler 对象的引用，最终处理消息时实际还是回调了 handler 自定义的方法。

（2）因为不能将一些耗时的操作放在主线程（UI 线程），需要交给其他线程异步执行，

最终通知 UI 线程结果更新对应的 UI。