Android工作经验之开发技术分享

---

1. 全部**Activity**可继承自**BaseActivity**，便于统一风格与处理公共事件，构建对话框统一构建器的建立，万一需要整体变动，一处修改到处有效。

2. 数据库表段字段常量和**SQL**逻辑分离，更清晰，建议使用**Lite**系列框架**LiteOrm**库，超级清晰且重心可以放在业务上不用关心数据库细节。

3. 全局变量放全局类中，模块私有放自己的管理类中，让常量清晰且集中.

4. 不要相信庞大的管理类的东西会带来什么好处，可能是一场灾难，而要时刻注意单一职责原则，一个类专心做好一件事情更为清晰。

5. 如果数据没有必要加载，数据请务必延迟初始化，谨记为用户节省内存，总不会有坏处。

6. 异常抛出，在合适的位置处理或者集中处理，不要搞的到处是**catch**，混乱且性能低，尽量不要在循环体中捕获异常，以提升性能。

7. 地址引用链长时（**3**个以上指向）小心内存泄漏，和警惕堆栈地址指向，典型的易发事件是：数据更新了，**ListView**视图却没有刷新，这时**Adapter**很可能指向并的并不是你更新的数据容器地址（一般为**List**）。

8.信息同步：不管是数据库还是网网络操作，新插入的数据注意返回**ID**（如果没有赋予唯一**ID**），否则相当于没有同步。

9.多线程操作数据库时，**db**关闭了会报错，也很可能出现互锁的问题，推荐使用事务，推荐使用自动化的**LiteOrm**库操作。

10.做之前先考虑那些可以公用，资源，**layout**，类，做一个结构、架构分析以加快开发，提升代码可复用度。

11. 有序队列操作**add**、**delete**操作时注意保持排序，否则你会比较难堪喔。

12. 数据库删除数据时，要注意级联操作避免出现永远删不掉的脏数据喔。

13. 关于形参实参：调用函数时参数为基本类型传的是值，即传值；参数为对象传递的是引用，即传址。

14. listview在数据未满一屏时，**setSelection**函数不起作用；**ListView**批量操作时各子项和视图正确对应，可见即所选。

15 控制**Activity**的代码量，保持主要逻辑清晰。其他类遵守**SRP**（单一职能），**ISP**（接口隔离）原则。

16. arraylist执行**remove**时注意移除**int**和**Integer**的区别。你懂得。

17. Log请打上**Tag**，调试打印一定要做标记，能定位打印位置，否则尴尬是：不知道是哪里在打印。

18. 码块**/**常量**/**资源可以集中公用的一定共用，即使共用逻辑稍复杂一点也会值得，修改起来很轻松，修改一种，到处有效。

19. setSelection不起作用，尝试**smoothScrollToPosition**。**ListView**的**LastVisiblePosition**（最后一个可见子项）会随着**getView**方法执行位置不同变动而变。

20.与**Activity**通讯使用**Handler**更方便； 如果你的框架回调链变长，考虑监听者模式简化回调。

21.监听者模式不方便使用时，推荐**EventBus**框架库，使用时间总线，没接触过的同学可以自行脑补一下哦。

22. Handler在子线程线程使用Looper.prepare，或者new的时候给构造函数传入MainLooper来确保在主线程run。

23. timepicker 点击确定后需要clearFocus才能获取手动输入的时间。

24. 构造函数里面极度不推荐启动异步线程，会埋下隐患。比如：异步线程调用了本例的示例，就会悲剧等着崩溃吧。

25. 千万不要理所当然的以为一个对象不会为空，充分的做好容错处理；另外注意null也可以插入ArrayList等容器中。

26. ExpandableListView的子列表不能点击（禁用）要把Adapter的isChildSelectable方法返回true。

27. UI显示注意内容过长的情形要提前使用ScrollView否则在小手机上尴尬你懂得。

28. 注意按钮的感应范围不小于9mm否则不易点击；输入框注意光标的位置更易用户输入。

29. 服务器和客户端尽量统一唯一标识（有可能是ID），否则多少会有歧义和问题。

30. 注释，尽量去写足够的注释，去描述一下思路，达到看了可以明白某一块代码的效果。

31. 完整型数据一定要用Sqlite的Transaction，大数据一定要用。粗略测试插入100个数据有20倍的提速，插入1000个数据就有100多倍的提速。

32. 避免String="null"的情况出现String = null,=""都可以。避免出现title="无主题"这样的数据提交到数据库浪费空间。

33. 存在多个不同的dbhelper实例情况下，sqlitedatabase对象必然存在不同的实例，多线程同时写入数据，轮流写入数据时会不定时的报db is locked，引起崩溃，不管

是操作同张表还是异表。读和写可以同时并发，轮流无规律的交替执行。同时写入数据时解决方案是用并发的每个线程都用事务，db则不会lock，按次整体写入。

34. 建议整个应用维护一个dbhelper实例，只要db没有关闭，全局就只有一个db实例，多线程并发写入db不会lock，严格交替进行写入：123123123。。。（123代表不同线程，轮流插入一个记录），读和写均不会锁住db，读写交替并没有规律，执行次数和程度看cpu分配给哪个线程的时间片长。

35. 一个任务使用事务嵌套N个事务，N个事务中有一个失败，这个任务整体失败，全部成功后，数据才写入，具有安全性，整体性。并且事务写入大批量数据的效率经实际测试成百上千倍的高于一般的单个写入。数据库大量数据、多线程操作建议使用LiteOrm数据库框架，更稳定简单。

36. 经常需要用ListView或者其它显示大量Items的控件实时跟踪或者查看信息，并且希望最新的条目可以自动滚动到可视范围内。通过设置的控件transcriptMode属性可以将Android平台的控件（支持ScrollBar）自动滑动到最底部。

37. Long a; 判断a有没有赋值，if(a == 0)在a没有赋值情况下会报错。应该if(a == null)，Integer、Floag等也一样，原因你懂，只是提醒你要小心喔。

38. 编码遇到读写、出入等逻辑要双向考虑，文件导入导出，字符字节相互转换都要两边转码。

39. 一个 int 值与一个 Integer 对象（能包含 int 值的最小对象）的大小比率约为 1:4（32位和64位机器有不同）。额外的开销源于 JVM 用于描述 Java 对象的元数据也就是 Integer，（Long、Double等也是）。

40. 对象由元数据和数据组成。元数据包括类（指向类的指针，描述了类的类型），标记（描述了对象状态，如散

列码、形状等），锁（对象同步信息）。数组对象还包括大小的元数据。

41. 一个在 **32** 位 **Java** 运行时中使用 **1GB Java** 堆的 **Java** 应用程序在迁移到 **64** 位 **Java** 运行时之后，通常需要使用 **1.7GB** 的 **Java** 堆。

42. Hash 集合的访问性能比任何 **List** 的性能都要高，但每条目的成本也要更高。由于访问性能方面的原因，如果您正在创建大集合（例如，用于实现缓存），那么最好使用基于 **Hash** 的集合，而不必考虑额外的开销。

43. 对于并不那么注重访问性能的较小集合而言，**List** 则是合理的选择。**ArrayList** 和 **LinkedList** 集合的性能大体相同，但其内存占用完全不同：**ArrayList** 的每条目大小要比 **LinkedList** 小得多，但它不是准确设置大小的。**List** 要使用的正确实现是 **ArrayList** 还是 **LinkedList** 取决于 **List** 长度的可预测性。如果长度未知，那么正确的选择可能是 **LinkedList**，因为集合包含的空白空间更少。如果大小已知或可预知或比较小，那么 **ArrayList** 的内存开销会更低一些。

43. 选择正确的集合类型使你能够在集合性能与内存占用之间达到合理的平衡。除此之外，你可以通过正确调整集合大小来最大化填充率、最小化未得到利用的空间，从而最大限度地减少内存占用。

44. 充分利用封装（提供接口类来控制访问数据）和委托（**helper**对象来实施任务）两种理念。

45. 延迟分配 **Hashtable**：如果 **Hashtable** 为空是经常发生的普遍现象，那么仅在存在需要存储的数据时分配 **Hashtable** 应该是一种合理的做法。将 **Hashtable** 分配为准确的大小：虽然会有默认大小，但建议使用更为准确的初始大小。

46. EditText在**setText**时不要忘记是否需要**setSelection**。在大多数情况下是需要设置的。

47. XML两种情况要注意：**1** 属性名字时候有重复；**2** 注意文本是否包含非法字符，注意使用**CDATA**包裹。

48. 当逻辑没有明显问题时考虑对象属性、函数参数、网络传输参数是否全部了解，是否设置正确。

49. 当出现编译或者运行时错误，别人那没问题时，考虑你的编译环境和环境版本是否有问题。

50. 由于**String**类的**immutable**性质，当**String**变量需要经常变换其值时，应该考虑使用**StringBuilder**提升性能，多线程使用**StringBuffer**操作**string**提高程序效率。

51. java 栈的优势是比堆速度快，可共享，主要存放临时变量、参数等，堆的优势是可动态分配内存大小。

52. 只要是用**new()**来新建对象的，都会在堆中创建，而且其数据是单独存值的，即使与栈中的数据（值）相同，也不会与栈中的数据共享。

53. 基本数据类型定义的变量称自动变量，存的是‘字面值’，存在于栈中，可共享（存在即不新建）。

54. 多个**RandomAccessFile**对象指向同一个文件，可使用多个线程一起写入无需再自己加锁，经试验结论：三个线程分别写入**100**万次数据，使用锁约**12**秒，不使用约**8.5**秒。**100**个线程分别写入**1**万次数据使用锁耗时约**4.2**秒，不使用锁耗时约**3**秒。

55. XmlPullParser解析慎用**nextText()**方法，**xml**比较复杂，含有空标签、重复名字标签时容易出现异常问题；**TEXT**中使用**getText()**方法代替**START_TAG**中使用**nextText()**方法；**START_TAG**，**TEXT**，**END_TAG**三个事件配合使用。注意每个**xml**节点之间（不管是开始节点还是结束节点）都会出现**TEXT**事件。

56. 改变逻辑的时候考虑全部用到这项功能的地方，分散的地方多了，容易大意。

57. 当系统原生组件出现问题时，查看错误栈信息，自己写一个该组件的子类，并在合适的地方将出错方法复写一下，加上**try catch**保证不崩溃掉。不要扰乱了该系统控件的正常逻辑。

58. 输入控件注意对空格、换行等符号的控制；输入框里内容注意和左右控件的空间，防止误点击。

59. 注意函数参数里的**++**或者**–**操作。是**++c** 还是 **c++**，区别很大。

60. 各种地方、永远的不要小看**null**指针问题，甚至有些场合宁可错杀（**try catch**），不可放过。

个人资料

code_li