Android

Android设计哲学

即使平台之间有很大的不同，但是如何利用API创建应用程序的学习过程是大同小异的。一般来说，有两个步骤：首先，应该知道怎么用API实现你的功能。其次，要了解平台间的细微差别。换句话说，首先你应该学会如何创建应用程序（了解应用程序的基本结构等），然后就要学会根据具体情况实现这个应用程序。

相比而言，第二阶段（学习使用正确的方法来实现应用程序）通常需要很长一段时间，在这个过程中你会不断地写代码，犯错误，然后从错误中吸取教训。显然，这不是一个有效的学习方法，本小节和下面的一些连接针对这向你伸出援助之手，教你怎么学习创建你的应用程序。

在此之前，先讲一个要点：成功的应用程序往往提供一个突出的用户体验。当Android团队构建了一个有着健壮核心的系统时，大多数的用户体验将来源于用户和应用程序之间的的交互。因此，我们鼓励你们花时间去构建应用程序优秀的用户体验。

显著的用户体验体现在三个核心特征上：*1、快速；2、响应；3、无缝*。当然，自从计算机出现以后，每一个平台都曾经有过类似的三种性质。尽管如此，每个平台实现这些特性的方式也有所不同；下面将会简单的介绍在Android平台下面你的应用程序将如何达到这些要求。

快速（Fast）

Android程序执行应该是很快的。当然，准确来说它的程序应该执行的很有*效率*（有效率才会快）。在目前的计算机世界里哟这样一个倾向：假设摩尔定律能够最终解决我们所有的问题。当这种倾向遇到嵌入式应用程序的时候，摩尔定律就变得有些复杂了。

与桌面和服务应用程序不一样，摩尔定律在移动设备应用程序上面不是真正适用的。摩尔定律实际上是关于电子晶体管集成密度的，它原本的含义是：随着时间的流逝，在给定的电路尺寸上面可以集成更多的电路。对于桌面和服务应用陈旭来说，它的含义是：你可以打包更多的“速度”在一个大小差不多的芯片上面，速度的提高，其他相应的一些性能也会显著的提高。对以像手机这样的嵌入式应用程序来说，相反的，使用摩尔定律是为了使芯片变得更小。这样，随着芯片密度的提高，相同功能的芯片会变得越来越小，功耗会越来越低，从而使手机做的越来越小，电池的持续的时间越来越长。这样就导致手持嵌入式设备相对于桌面系统来说正在以一种比较慢二实际的速度在增涨。因而，对于嵌入式设备来说，摩尔定律就意味着更多的功能和更少的功耗，速度只是次要的。

这就是我们要写出高效代码的原因：你不能天真的认为电话在速度上面的增涨速度和桌面、服务应用程序是一样的。一般来说，高效的代码意味着最小的内存占用，意味着紧凑的风格，意味着避免了因为某种语言和编码习惯对性能的影响。我们用面向对象这个概念来理解，大部分这样的工作都是在*方法*层面上，与实际的代码，循环等等相类似。

在 “编写高效Android” 一文中，我们会对此做详细的介绍。

响应（Responsive）

我们有可能能够编写赢得世界上所有的性能测试的代码，但是用户使用起来会感到很恼火，这是因为应用程序没有足够的*响应性*——让人感觉反映迟钝，在关键的时刻失灵，或者处理输入太慢。在Android平台下，那些响应性不够的应用程序会经常弹出"Application Not Responding" (ANR)这样的致命消息。

通常，这会在应用程序不能响应用户的输入的情况下发生。例如，你的应用程序在一些I/O操作上（如网络接口调用）阻塞，这是主线程将不会处理用户的输入事件，一段时间之后应用系统就会认为你的程序挂起了，就会给一个选项给用户询问是否结束它。同样的，如果你的应用程序花费很多时间去构建内存中的一个结构、或者计算游戏的下一步，这时系统同样会认为程序已经挂起。当碰到上面情况的时候，要确保计算的高效性，但是即使是最高效的代码也需要花费时间。

在上面两个例子中，问题的解决方案是建立一个子线程来处理大部分的工作，这样就能保证你的主线程（响应用户界面事件）一直运行，这样防止系统认为你的程序已经僵化。因为这种线程的实现一般在*“类”（CLASS）*这个层次上，你可以把响应当成是类的问题来处理（这里，可以和*方法*层次描述的基本性能相比较）。

这里只是一个简单的介绍，在 “构建响应Android应用程序” 一文中对于应用程序的响应性有详细的介绍。

无缝性（Seamless）

即使是你的应用程序执行很快，并且具有很高的响应性，它仍然有可能让用户苦恼。一个常见的例子是后台进程(比如Android的 Service 和 BroadcastReceiver)对某些事件可能会突然弹出一个UI响应。这似乎是无关紧要的，并且一般开发者会认为这这是正常的，因为他们花费了戴亮时间去测试和使用自己的应用程序。可是，Android应用程序模型的构建是能够允许用户在不同的应用程序之间进行流畅的切换。这就意味着，当你的后台进程实际上弹出那个UI的时候，用户可能正在系统的其他部分中，做一些其他的事情，如在接电话。想像一下，如果SMS服务每次都会在文本消息传入时弹出一个对话框，这很快就会使用户崩溃。这就是为什么Android标准对于这些事件使用的是通知（Notifications）机制；这使用户能够自己控制。

这仅仅是一个例子，相似的例子数不胜数。比如，如果Activities没有正确的实现onPause()方法和其他生命周期方法，这将会导致数据丢失。或者如果你的应用程序有意的暴露数据给其他应用程序使用，你应该使用一个ContentProvider，而不是用一个路人皆可见的未加工过的文件或者数据库。

这些例子有一个共同的特点，他们都涉及到程序与程序或则程序与系统之间的交互。系统被设计为将多个应用程序视为一种松耦合组件的联合，而不是大块的代码黑盒。这就允许作为开发人员的你将整个系统看作是一个这些组件的大联合。这允许你干净地封装，无缝地和其他应用程序结合，因而你能设计自己喜欢的程序。

这使一种*组件*层次的概念（与性能和响应的*类*层次和*方法*层次相对应）。至于怎样编写无缝性能很高的代码，“与系统相结合”一文中将会对此做出介绍，提供代码提示和最佳实例。