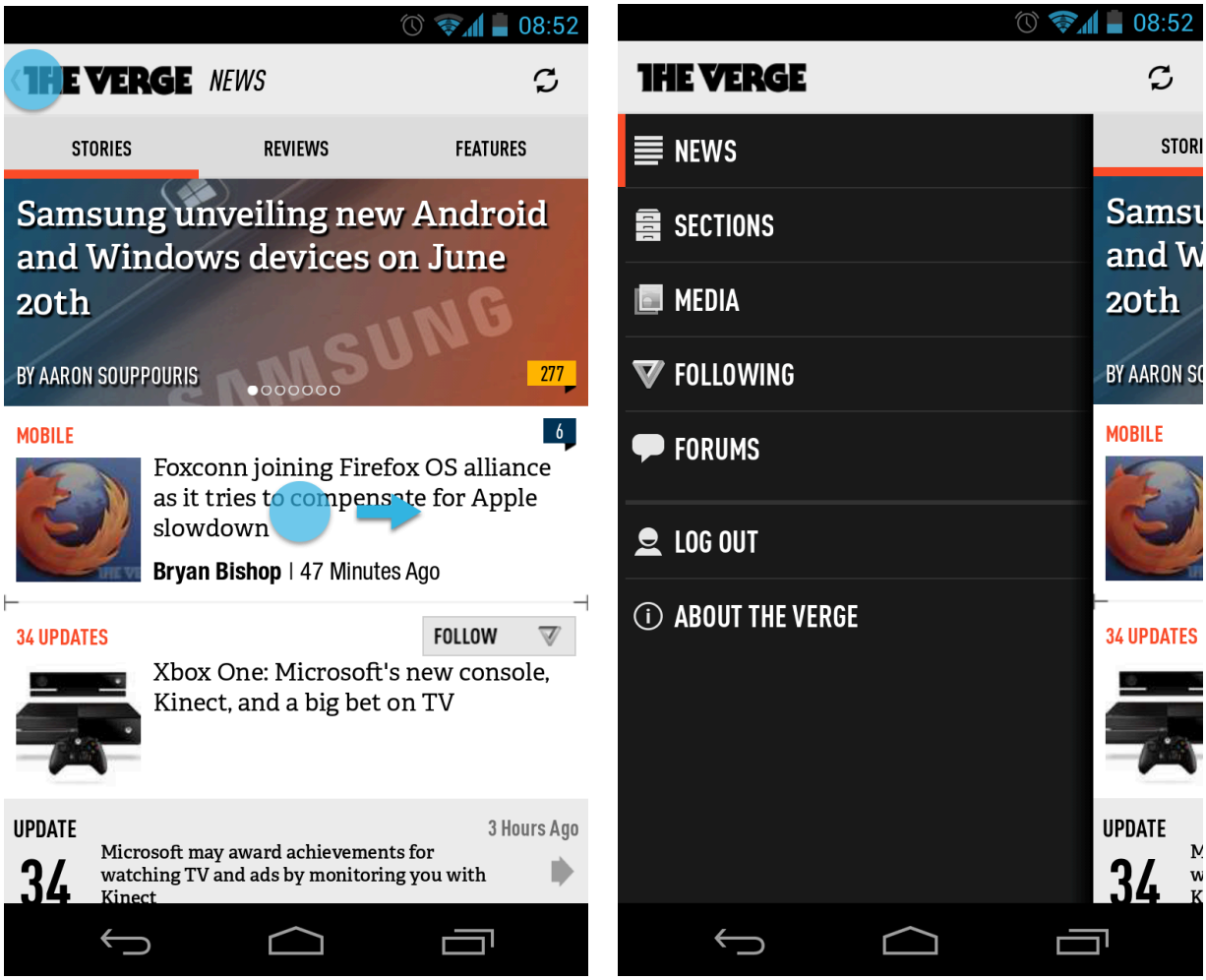
译者按

在Google推出Navigation Drawer之前，android生态系统中并不缺乏使用Drawer的应用，但是这些应用看起来使用了同样的交互，却又各有不同，我暂且从三个地方来描述这些应用之间的区别：

1. action bar是否固定
2. 触发区域
3. 是否能所有界面都可呼出

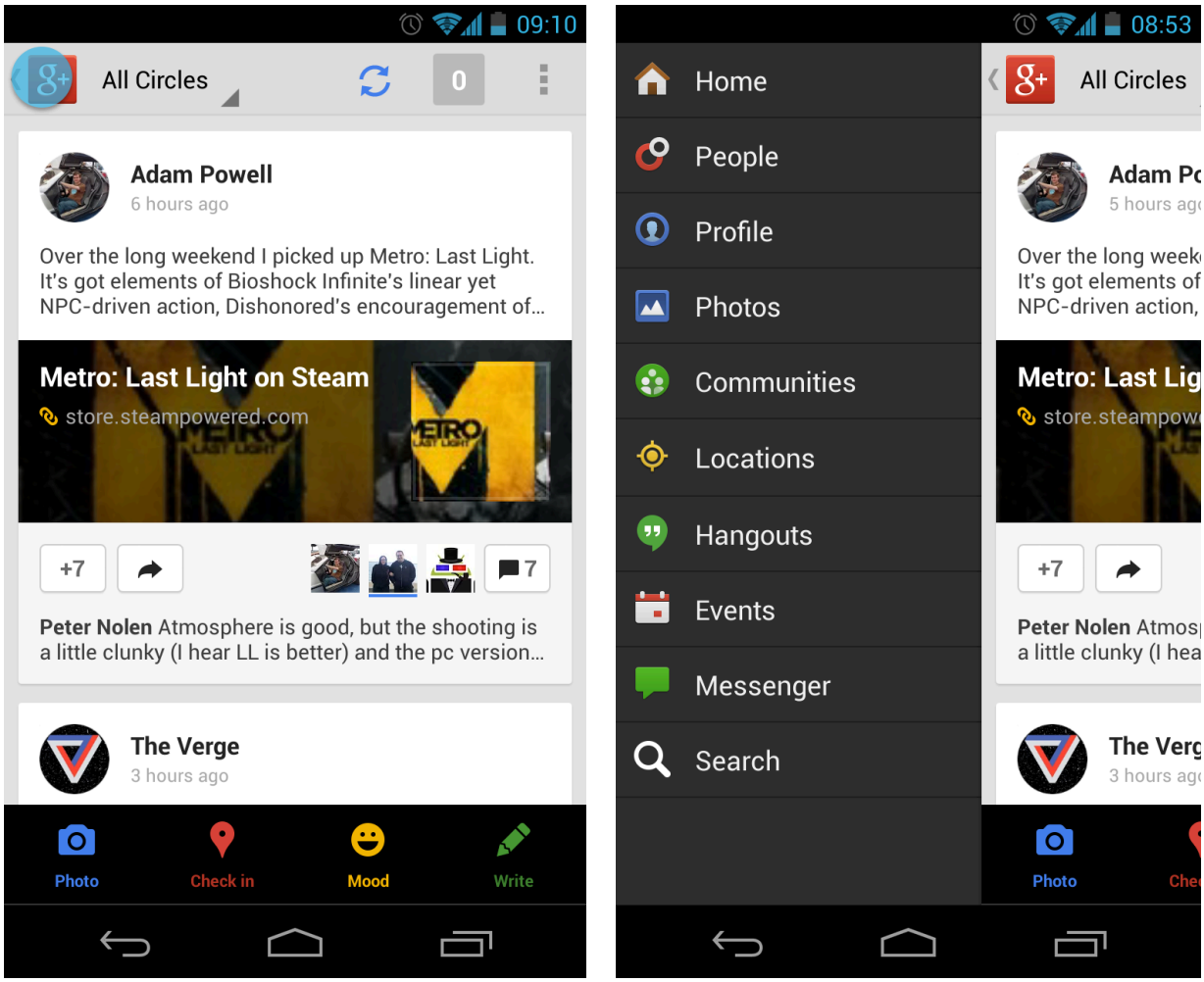
现状：

the verge



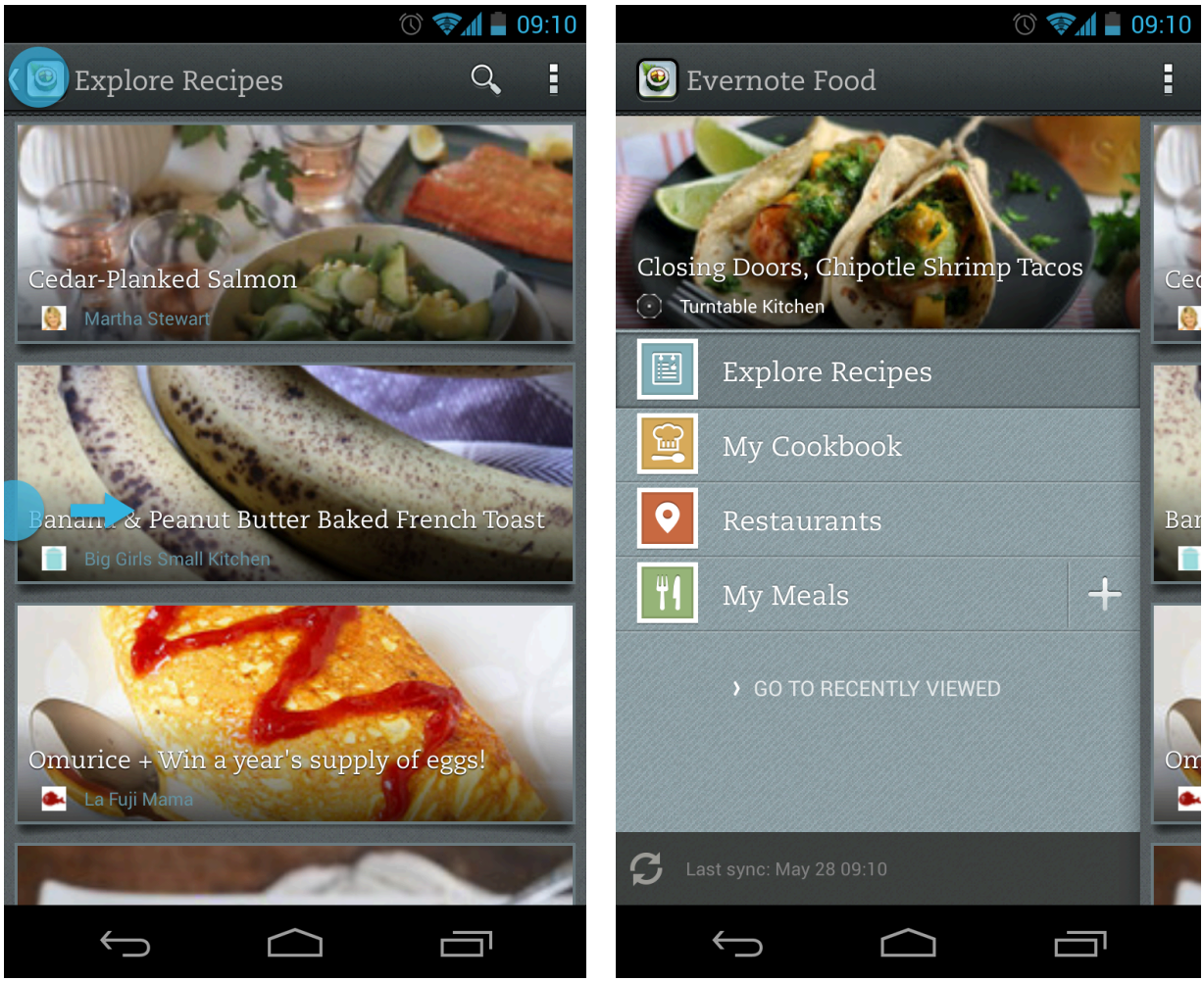
1. action bar固定
2. 整屏触发
3. 几乎所有界面都可呼出

Google+



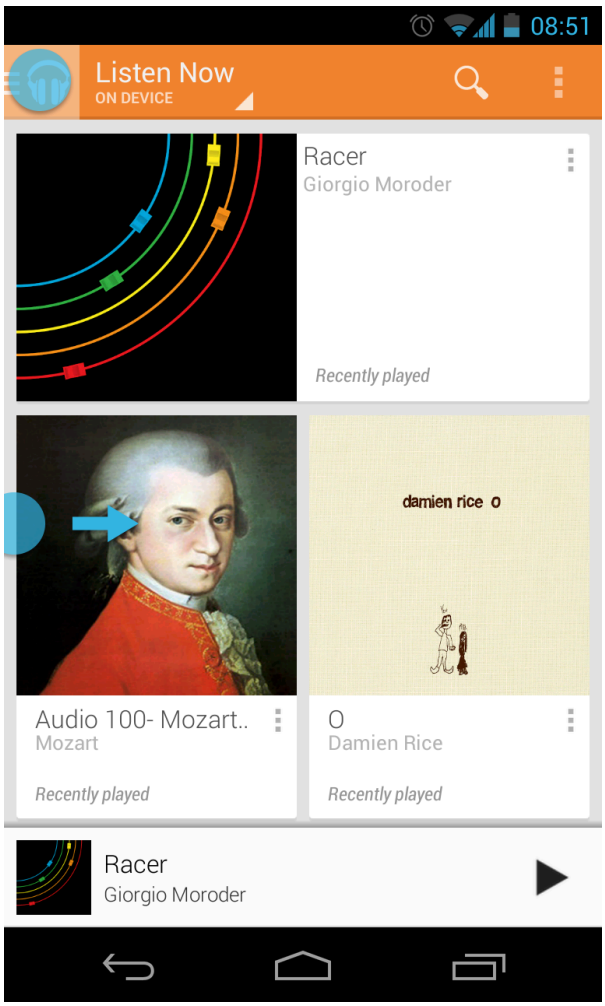
1. action bar不固定
2. 滑动不可触发抽屉（我认为这是Android平台亟待更新的一个应用）
3. 并非所有界面都能呼出导航抽屉

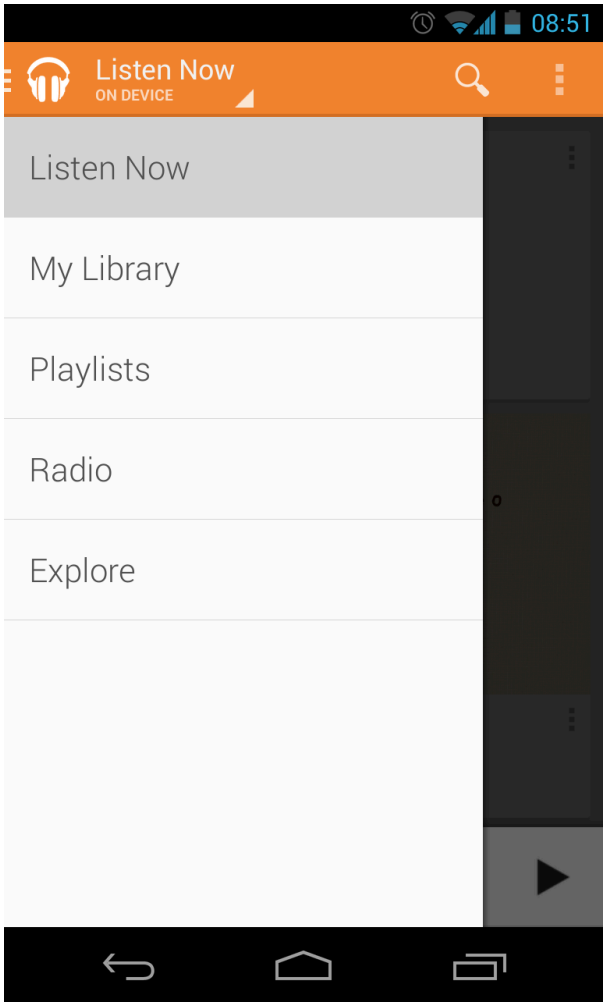
Evernote Food



1. action bar固定
2. 滑动边缘可触发抽屉
3. 所有界面都能呼出导航抽屉

规范的Navigation Drawer的做法





1. action bar固定 - 常用操作根据所处界面变化
2. 边缘滑入呼出抽屉导航
3. 几乎所有界面均可呼出

正文

DrawerLayout and the associated Android Design guidelines made their debut in the Google Shopper and Google Earth apps before Google I/O. At the conference+**Roman Nurik** and +**Nick Butcher** gave a great talk introducing them to engineers while explaining other UI patterns (https://developers.google.com/events/io/sessions/326204977) and +**Richard Fulcher** and +**Jens Nagel** gave another on related structural patterns as well. (https://developers.google.com/events/io/sessions/326301704) As an adaptation of a long-standing pattern from the developer community, we had a big task ahead of us in standardizing a pattern for the sliding drawers that users have come to know and love while still meeting some very general requirements from the rest of the app ecosystem.

在Google I/O大会之前，DrawLayout和其相关的Android设计指南就首次出现在Google Shopper和Google Earth中。**Roman Nurik**和**Nick Butcher**在大会上很好地将其介绍给工程师们，同时还解释了其他UI模式（https://developers.google.com/events/io/sessions/326204977），并且**Richard Fulcher**和**Jens Nagel**也解释了相关的结构模式（https://developers.google.com/events/io/sessions/326301704）。作为一个在开发者社群中长期存在的交互模式，在标准化一个用户开始熟知并且喜爱的滑动式抽屉模式之前，我们有一个艰巨的任务，那就是我们还需要满足应用生态系统中其他的一些非常常规的要求。

We started from both low-level details and high-level conceptual structure at once. Everyone uses nav drawers just a little bit differently and those differences have implications for the rest of the app's navigation hierarchy. After a while we distilled down to a set of goals that we wanted to achieve:

我们同时从底层细节和高层概念结构着手。每个人使用导航抽屉都有一些小小的不同，而这些不同之处对应用其他的导航层级产生了一些影响。然后，我们提炼出一组我们需要实现的目标：

1) Consistency across apps. This was the whole reason we started on the project; we wanted to set forth some common patterns that developers could agree on so that users would find these constructs instantly recognizable from app to app. It also couldn't deviate too radically from existing navigation designs. Apps using a navigation drawer should still conform to general navigation principles like Up navigation, placement of universal items such as Settings and Help and so forth.

1）跨应用的一致性。这是我们启动整个项目的首要原因；我们希望提出一些被开发者认可的普适模式，这样用户立刻在不同的应用中辨认出这些结构。它也不能过于偏离现有的导航设计。使用导航抽屉的应用必须依旧符合一般导航原则，比如像“向上”（Up navigation）按钮，和“设置”，“帮助”的摆放位置。

2) Suitability for many different apps. The most common version of navigation drawers in the wild bulldozes the main content of a window to one side to reveal the drawer. This had some issues, even

though in some apps it can look flashy and feel satisfying.

2）不同应用之间的适应性。最普遍的导航抽屉版本就是将主体内容从一边滑到另一边来露出抽屉。这有一些问题，即使在一些应用里它看起来很华丽，感觉也很舒适。

3) Discoverability. A major navigation pattern that users will need to rely on needs permanent on-screen cues and affordances so that users don't miss large parts of an app's functionality.

3）可发现性。一个将被用户依赖的主要导航模式，在屏幕上需要永久显示的提示和隐喻，这样用户才不会错过一个应用的大部分功能。

4) Deep drawers. The more we kept playing with the idea of navigation drawers, the more we liked the idea of having the option of the drawer at lower levels of the app's hierarchy as a shortcut to get around. Leading into Android 4.x we did a lot of work to clarify the navigation hierarchy of apps and move away from the "big ball of activities" unstructured flows of 2.x and earlier, but the consequence of this was that getting from one leaf node of an app's hierarchy to another now meant more steps along the way.

4）深层导航。我们越深入思考导航抽屉的想法，我们越希望将导航放进低层级界面作为快捷入口。刚进入Android4.x时代时，我们在澄清应用的导航层级上花了不少功夫，移除了Android2.x或更早版本中的非结构化流，但是这样做的后果是从应用某个层次的节点到另一个节点现在需要花费更多的步骤。

5) Ease of development. The best pattern can be completely defeated if it's too complicated for developers to implement. Whatever we did had to be dead simple to add to apps, ideally with a pre-baked implementation in the Android support library.

5）易于开发。最好的交互模式也会因为开发过于复杂而被轻易击败。我们所做的一切都必须足够简单地被添加到应用中去，最好是基于Android支持库上实现。

6) Multiple screen size support. Developing for multiple screen sizes and great tablet support is something we'd like to continue improving. As we explored different drawer options, we couldn't help but notice that drawers held a lot in common with the left pane of multi-pane tablet layouts. Maybe there was something we could use there.

6）支持多种尺寸屏幕。为多种屏幕尺寸开发和支持大的平板是我们希望持续改进的。当我们探索不同的抽屉选项时，我们注意到抽屉和multi-pane平板布局中的左侧pane有很多共同点。也须有一些可以借鉴的地方。

7) It had to feel awesome. The best UI is one you find yourself idly fidgeting with.

7）必须看起来很棒。你无聊时把玩的视觉效果便是最好的。

One of the first decisions we made around navigation drawers was to leave the action bar fixed and we published this stipulation in the Design Guide very early. This was for a couple of reasons.

对于导航抽屉，我们做出的第一个决定就是将action bar固定，并且我们将这个规定很早就发布到设计指南中了。这其中有几个原因。

First, the action bar is part of what we call the window decor. A window's decor is extra decoration around the window's primary content that is owned and controlled by the framework. Nothing actually guarantees than an app can walk up the view hierarchy and see something consistent from version to version or that inserting a new, arbitrary parent view above the action bar in order to move it and the rest of the content around is going to actually work without crashing on later platform versions.

首先，action bar是window decor的一部分。decor 是窗口主要内容周围的部分，它是framework的一部分，并由framework控制。实际上，没有一个应用可以做到跨越视图层级，为actionbar添加一个新的父view来实现在移动actionbar的时候主要区域可以正常工作并保证在新版本系统中不会崩溃。

Second, the action bar forms an important structural anchor for the activity. It's always contextually relevant and provides a well-known place for navigation and actions. Making it inaccessible when a navigation drawer is open would be counterproductive in many situations and make the drawer feel more modal than it needs to. Apps that do this often end up creating secondary bars for actions within their drawers to compensate.

第二，action bar为activity构成了一个重要的结构锚点。它总是与内容相关的，并且为导航和操作提供了一个被人熟知的位置。当navigation drawer被打开而导致无法进入action bar时，将会在很多情况下出现问题，而且使drawer感觉更加形式化。这样做的应用一般都会在它们的抽屉里创建一个用于操作的二级菜单作为补偿。

We always prefer direct manipulation when it comes to touch. If the main window content moves to reveal a drawer the user should always be able to grab it and drag it to one side to make the drawer visible. In the general case this has issues with the window's gesture space.

当一个操作可以被点击时，我们通常选择直接操作。如果主窗口内容移动来露出一个抽屉，用户必须每次都要滑动来使抽屉显示出来。在通常的案例中，这样做还会与窗口手势操作区域有冲突。

A navigation drawer that is revealed by swiping the main activity content to one side means the horizontal swipe gesture space is now occupied. In effect, a navigation drawer in this style would mean you couldn't have ViewPagers, swipe to dismiss, MapViews, WebViews, etc. in the main content without a conflict. That's awfully limiting for a major UI pattern.

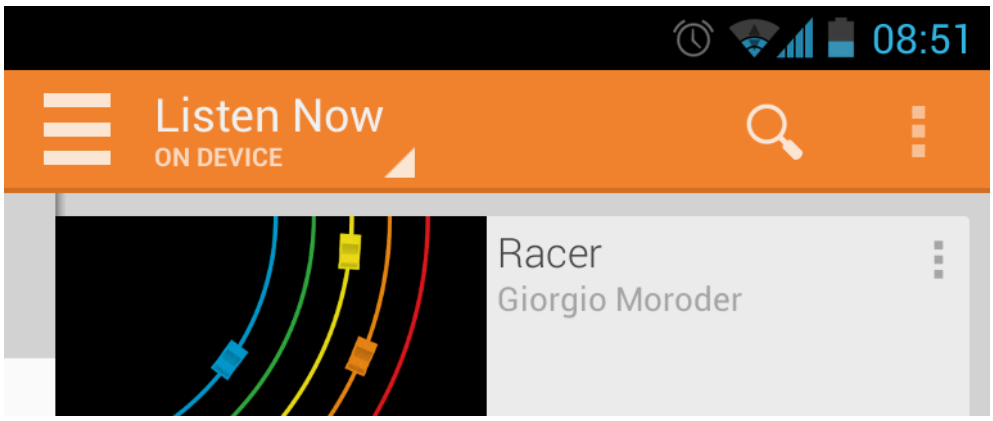
通过将main activity内容向一边滑动来露出导航抽屉，表明水平滑动手势区域被占用了。简单的说就是，如果导航抽屉是这样的呼出形式，意味着你将不可能在主内容中同时使用ViewPagers，滑动以隐藏，地图视图等等，而不引发冲突。这对于大多数交互模式都是一个要命的限制。

An overlay drawer with an edge swipe solved many of our requirements all at once. It retains direct manipulation and feels particularly nice on Nexus 4's curved glass edges. It keeps the gesture space open over the main window content. It can be implemented as a simple ViewGroup that you can drop in at the top level of your layout with one view becoming the drawer itself. android.support.v4.widget.DrawerLayout was born and met our goals of consistency, suitability across diverse apps and ease of development.

一个通过边缘呼出的覆盖式抽屉一下子就解决了大部分的需求。它保留了日常操作，并且在Nexus 4的弯曲玻璃边缘上表现的尤为出色。它为主窗口内容保留了手势区域。你可以在你布局的顶层放置一个的ViewGroup，然后将其中的一个view变为抽屉，实施起来很简单。android.support.v4.widget.DrawerLayou诞生了，并且达到了多种程序的一致性，适应性和易于开发的目标。

The next issue to tackle was discoverability. Thanks to work already done by the developer community users were already accustomed to reading a three horizontal line icon at the left side of the action bar as a menu/drawer button. (During user testing one user referred to it as "the hamburger" and the name stuck.) We knew this approach was a sure thing fori discoverability but it didn't quite sit right. Replacing the app icon meant potentially losing a big part of the app's identity.

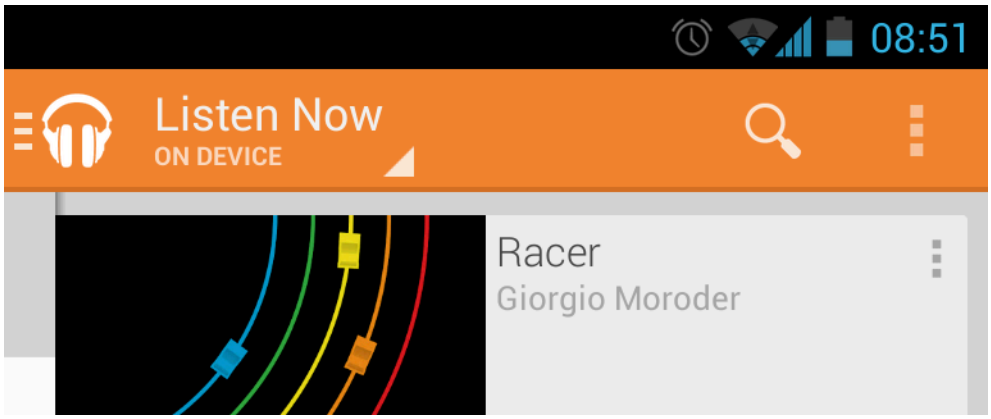
下一个需要解决的问题就是可发现性。多谢那些开发者社群中的用户所做的工作，已经习惯将三条水平线的图标放在action bar左边作为一个菜单/抽屉按钮。（在用户测试过程中，一位用户将其认为是一个"汉堡包"。）我们知道这种途径无疑增强了可发现性，但有一点不对劲。替换应用图标意味着丢失大量应用特性。



The apps that made the full-size three-line button icon in the upper left popular are themselves popular apps with strong branding throughout their UI. For other apps, using that full-size hamburger icon in place of the app's own icon on the action bar meant losing a glanceable identifying characteristic. It ran the risk

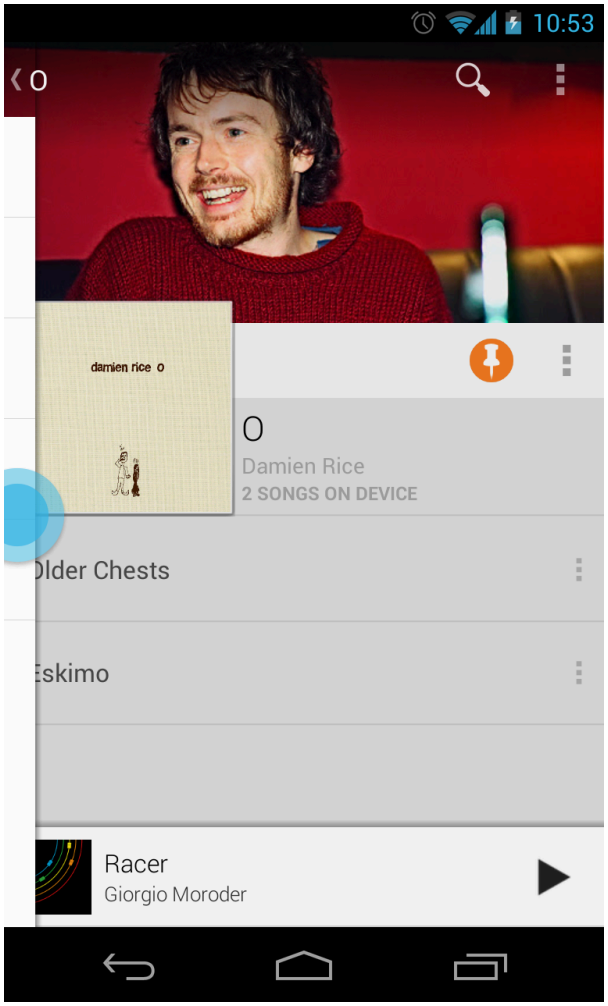
of apps looking too “samey.” We decided to play around with changing the Up chevron to the left of the icon as an alternative to a full hamburger.

那些流行将整个三条线的按钮放在左上角的应用,他们可以通过自己的UI来表达强烈的品牌感。对于其他的应用，使用全尺寸汉堡包图标替换应用action bar上自己的图标意味着失去自己的识别特征。它会使众多应用看起来过于相似。我们决定将向上箭头替换为左移后的汉堡包图标，来代替整个图标。



A first draft of the small hamburger (affectionately called “the slider”) didn’t test well. Making it the same width as the Up chevron made it too easy for users to miss in a quick visual scan. Some of our visual design team got to work creating a number of alternatives for later review. In the meantime we added the long-edge touch “peek” to DrawerLayout’s behavior as a hint to a drawer’s presence, but we knew this wouldn’t be sufficient for discoverability on its own.

第一版小汉堡(亲切地称为“滑块”）的草图感觉不太好。如果将其宽度设置得跟向上图标一样，用户在快速扫视的过程中很容易忽视掉。我们视觉设计团队的成员又做了一些候选的版本。与此同时，我们将“触摸整条屏幕边缘即可窥视抽屉”加入了抽屉式布局的交互行为当中，它可以作为一个抽屉存在的提示，但是我们知道这是不够的。



Meanwhile we were still looking into whether we could address adapting to multiple screen sizes with the same DrawerLayout pattern and we were running into a wall. Many apps like Gmail were already using fragments to great effect on larger screens by simply placing item list and item detail views side by side where those same views would represent deeper navigational hierarchy on handsets.

同时我们依旧在看我们能否使抽屉式布局模式适应于各种屏幕尺寸，但是我们遇到了困难。许多像Gmail的应用，他们在大尺寸屏幕上通过fragments将项目列表和项目详情view并排放在一起，但是相同的views在手持设备上却表示更深的导航层次。

YouTube was already using a content-sliding drawer to handle this case and it worked quite well. The new Hangouts app wanted to do something similar, especially to handle the case of Nexus 7-sized tablets in portrait mode. They wanted to keep the conversation list partially visible as context even when there wasn't space to show it fully, but to show the full list and current chatlog on large enough screens.

Youtbe已经使用了内容滑动抽屉来处理这种情况，并且表现得非常好。新的Hangouts想做一些类似的事情，特别是处理Nexus7在竖屏模式下的情况。他们希望即使没有足够的地方显示完整的会话列表，也希望将其保持部分可见，而在足够大的屏幕上将列表和会话详情全部显示出来。

The further we looked at these use cases the more clear it became that DrawerLayout and what Hangouts wanted were distinct. The key difference was that our navigation drawer pattern was independent of screen size; in cases where we wanted a nav drawer we wanted it tucked away until summoned. A larger screen didn't mean that there was some responsive design inflection point where the drawer became permanently visible.

我们越进一步地看这些用例，越清楚地认识到抽屉式布局和Hangouts想要的布局是不一样的。关键的不同点在于是我们的导航抽屉模式是独立于屏幕尺寸的；在我们希望包含导航抽屉的案例里，我们希望抽屉隐藏起来，在被召唤的时候才显示出来。大的屏幕并不意味着有一些响应式的设计拐点，比如抽屉永久显示。

android.support.v4.widget.SlidingPaneLayout now formed the basis for the new Hangouts app's UI. We kept the bulldozing behavior here even though we rejected it for DrawerLayout. Its use case is more narrow than nav drawers and it wasn't designed to be as strong of a prescription for this specific use case, merely another tool in the box. We were willing to concede the potential for gesture space conflict and leave that to apps choosing to use it.

android.support.v4.widget.SlidingPaneLayout现在组成了Hangouts的UI基础。虽然我们拒绝在这里使用抽屉式布局，但我们会持续攻破难点。它的用例比导航抽屉的用例更狭隘，因为它是为这一个特殊用例设计的，所以它并不是能成为一个强有力的处方，而仅仅是盒子中的另一个工具。我们承认这里存在着潜在的手势操作区域冲突，并且将这个问题留给那些选择这样做的应用。

This had some other implications for Up navigation. While we were settling on using the upper left corner as a toggle with a different glyph for navigation drawers using DrawerLayout, SlidingPaneLayout was becoming a different story. SPL's primary purpose was fitting a large-screen layout onto a smaller screen. You could build something equivalent using fragments and list/detail activities, SlidingPaneLayout was just meant to give that flow a more engaging interaction.

这里对于向上导航（Up navigation）有了一些新的含义。当我们使用DrawerLayout时，我们在左上角使用了一个不同的图标解决了“表示这是一个导航抽屉的开关”的问题，SlidingPaneLayout却有所不同。SlidingPaneLayout 的初衷是将一个大屏幕布局适应到小屏幕上去。你可以使用多个fragments和列表/详情activities来得到同样的效果，SlidingPaneLayout只是意味着给信息流一个更加迷人的交互。

With that in mind there was only one answer. SlidingPaneLayout would use normal Up navigation as if it were conceptually two different activities in the navigation hierarchy. The Up button should travel up from the detail pane to the item list, opening the pane, but never the reverse.

考虑到这一点只有一个答案。SlidingPaneLayout使用常规的Up导航，概念上就像在导航层面有两个不同的activity。Up按钮必须引导从详情pane到项目列表，打开pane，而不是相反。

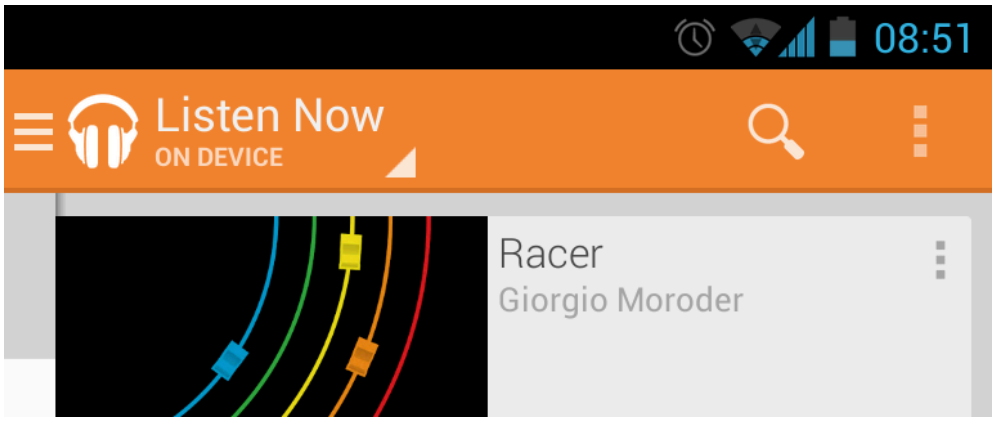
Making it easier and more satisfying for developers to handle multiple screen sizes: accomplished. The same code can be used on any device and SlidingPaneLayout will automatically lock open when enough horizontal space is available to fit both panes side by side.

让开发者更加轻松和满意的处理多种屏幕尺寸：完成。相同的代码可以在任何的设备上使用，并且在水平上有足够的空间容下两个并排的pane，SlidingPaneLayout 将会自动打开。

We came back to the discoverable affordance for nav drawers. In an effort to make it more visible the

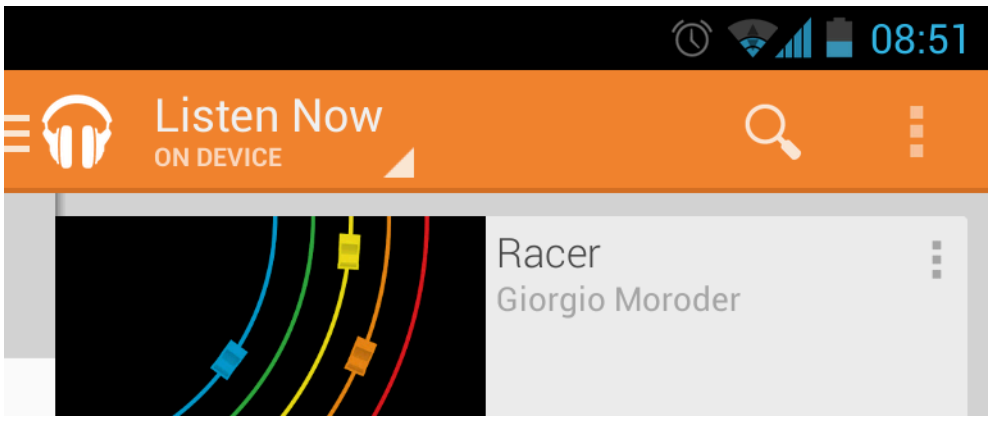
design team had experimented with slightly wider sliders that would shove the activity icon a bit to the right in the natural course of layout, but those solutions were rejected since it threw off the metrics and positioning of the icon. Someone decided to try eliminating the left side padding of the slider, allowing it to be wider and more visible as it touched the left edge of the screen.

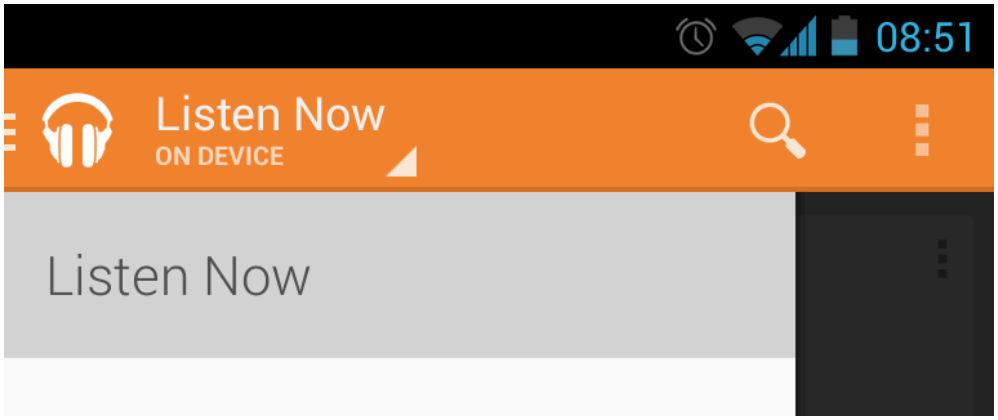
我们再来谈谈导航菜单的可发现性。为了让它更容易被发现，设计团队做了个实验，用稍宽的滑块将activity图标向右推动了一点，但是这些方案都被否决了，因为它搅乱了图标的度量和定位。某些人决定去掉滑块左边的内边距，使其挨着屏幕边缘，这样图标可以更宽一些，并且更可见。



This design came along with a small animation sample. Since the slider now touched the left edge, animating it off further to the left meant it became visually smaller. This subtle transition helped reinforce the button's purpose: touching the slider in the fully visible state meant the drawer would move into view from offscreen while the slider icon slid off in the opposite direction. This perfectly advertised the toggle open/closed behavior; touching the slider in its now mostly-offscreen state would make the drawer mimic the same motion, sliding back offscreen itself. android.support.v4.app.ActionBarDrawerToggle was written to provide the glue between a DrawerLayout and this behavior.

这个设计带有一些列动画。当小滑块碰到左边缘，它向左移动有一个小动画，来表示此时菜单可见。这个微妙的过度加强了按钮的目的:轻触全部显示出来的小滑块表示抽屉会从屏幕外面(向右)移入视野，而此时小滑块向相反方向(左)滑动。这样完美的演绎了开关开启/关闭的行为;点击滑块(部分在屏幕外的状态）将会使抽屉做类似的运动，自动滑出屏幕以外。android.support.v4.app.ActionBarDrawerToggle就是用来关联DrawerLayout和这种交互的。





Our final nod to discoverability was to optionally open the navigation drawer on first launch of the app. While this guarantees that the user will know it's there, it doesn't mean that they will know how to summon it in the future. Apps that take this option should record the first time that the user successfully opens the drawer on their own and stop showing it open on startup.

我们关于可发现性的最后一点是在第一次启动应用时就打开抽屉。虽然这样可以保证用户知道抽屉在那里，但是并不意味着他知道以后如何呼出抽屉。当用户第一次自己成功打开抽屉，应用需要记录下来，然后在以后启动时不再显示抽屉。

At this point we were happy with the outcome. There were certainly compromises involved but we were confident that this pattern was universal enough to publish. Both DrawerLayout and SlidingPaneLayout felt good for their respective use cases. Only one issue remained, and it was one we had many discussions about through the entire process. That issue was the one of deep drawers - nav drawers at levels of an app's navigation hierarchy deeper than the root.

我非常开心现在终于有了成果。虽然有妥协，但是我们还是非常有信心发布这个足够通用的交互模式。DrawerLayout 和SlidingPaneLayout 对于他们的用例都足够友好。只有一个遗留问题，而这个问题是我们在整个过程中反复讨论的。那就是深层导航 - 比顶级导航要低的应用层及导航。

We were confident in overloading the left side action bar button for a drawer toggle at the root of an app's hierarchy because there was no potential for ambiguity. However at deeper levels of the navigation hierarchy this wasn't an option; that space was already reserved for Up. Removing Up from deeper activities with a drawer would mean invalidating a previous pattern and moving back toward the big ball of activities anti-pattern that we've worked hard to overcome for the last year or so. So we kept both.

我们非常自信地将应用顶级的action bar左边按钮作为抽屉的开关，因为这样做并没有潜在的歧义。但是在应用的低层级界面里，这不是问题；因为那时按钮已经被替换成向上按钮（Up）。如果将更深层级的activities 的向上按钮去掉而替换成抽屉将会意味着使我们努力工作一年而创造的交互模式作废。所以我们保留了两者。

Location：Communities > Android Design

Drawer entries are generally shortcuts to navigation hubs within an app. These hubs are often siblings at the root of your app's navigation hierarchy, but not always. As navigational flexibility is important from any of these hubs, the drawer should be prioritized using the ActionBarDrawerToggle-style affordance. From any other location within the app, the app should present Up navigation as usual.

抽屉通常是作为导航集合的快捷入口存在于应用中的。这些集合一般都是应用顶层的同级导航，但也不全是。因为对这些集合来说，导航灵活性是非常重要的，所以抽屉必须优先使用ActionBarDrawerToggle-style。在应用的任何其他位置，应用都必须像以往一样显示向上按钮（Up）。

Up may take priority over a drawer for the upper left affordance outside of navigation hubs, but there's another way to open nav drawers - the edge swipe. Absolute discoverability isn't critical at lower levels since a user always has the option of tapping Up repeatedly to return to the root, at which point the slider is available to open the drawer. Savvy users can simply swipe the drawer out from anywhere it's made available as a shortcut.

向上按钮的优先级可能高于抽屉，但是还有另一种方式可以打开导航抽屉 - 边缘滑入。完全的可见性在低层级

界面里并不至关重要，因为一个用户始终都可以通过不断点击向上按钮回到根目录，在那里用户可以点击小滑块来开启抽屉。请理解允许用户在任何界面通过边缘滑入来呼出抽屉只是一个快捷手段。

Drawer-based navigation is always absolute. It doesn't create navigation history per se, it instead behaves like navigation from a home screen widget or notification. That is, it involves replacing the task stack with TaskStackBuilder or similar. It represents a hard context switch to another part of the app. Of course if an app only uses it to switch top-level views and never deep links you can implement this with a simple FragmentTransaction.

抽屉导航始终是绝对的。对于每次的操作它并不创造历史记录，它的行为类似从一个桌面插件或者通知进入应用。它使用TaskStackBuilder之类的东西替换了任务堆栈。它是强行切换到应用其他位置的代表。当然如果一个应用只用它来在顶层视图之间切换且完全不加入深层链接，你用FragmentTransaction就可以很容易地做出来。

I hope you've enjoyed this short series. Creating patterns for the Android Design guide always carries more constraints than creating a UI for a single app since they have to meet more general use cases, but the reward is more consistent behavior across the ecosystem that users can immediately recognize and understand in your app from experience with others.

我希望你能享受这短暂的系列。为Android设计指南创造交互模式承载的制约因素总是多于单独为某个应用创作UI，因为需要面对更多更普遍的用例，但收获是，用户通过从其他应用中获得的经验可以快速识别并理解你的应用，这样的行为贯穿整个生态系统的。

原文：

均来自Adam Powell的Google+ posts

- Part 1
- Part 2
- Part 3
- Part 4