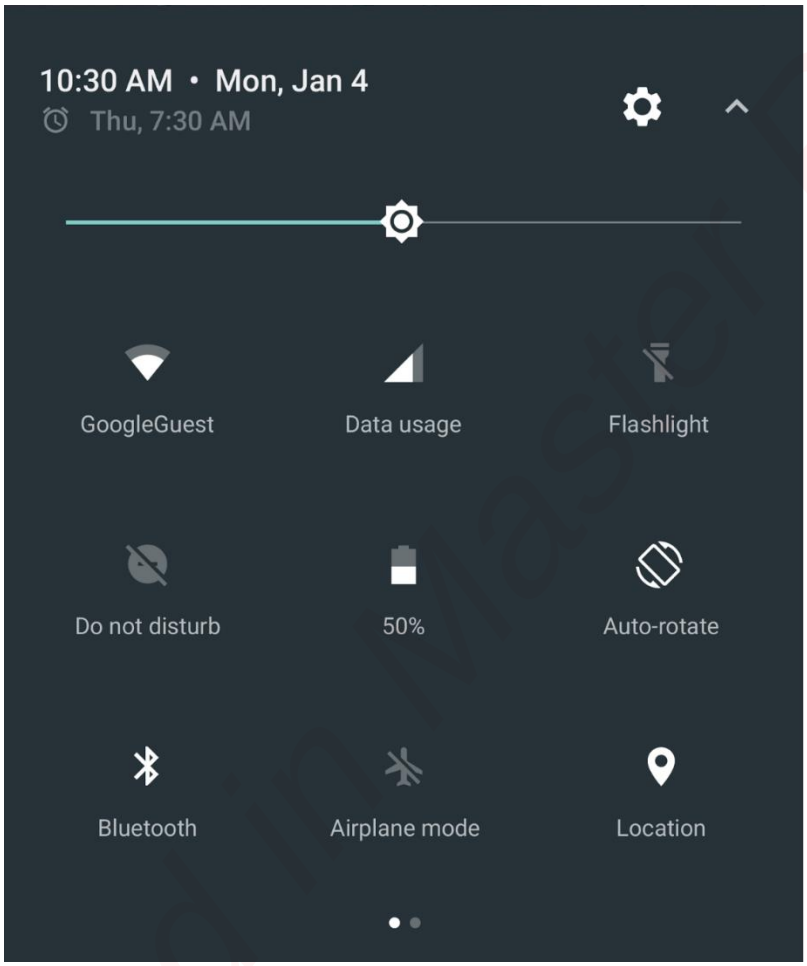
掌握 Android 7.0 新增特性 Quick Settings

本文中，我们详细了解一下 Android N（7.0）上的新增特性：Quick Settings。

Quick Settings 功能如下图所示：



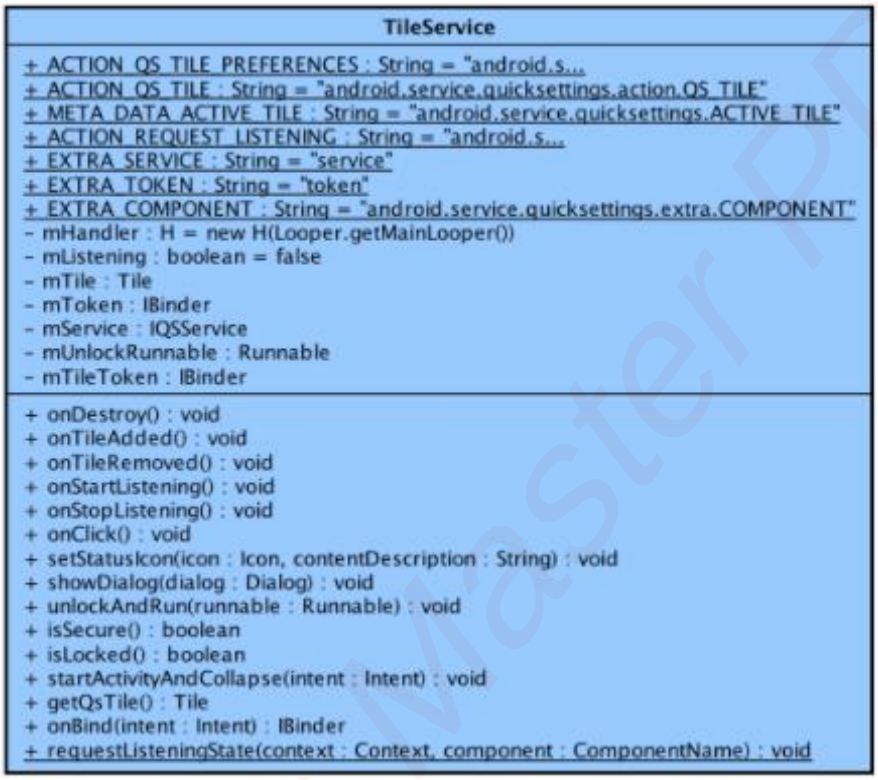
该功能位于下拉的通知面板中，在用户单手指下拉通知面板的时候，Quick Settings 区域显示成一个长条，用户可以点击右上角的尖号展开这个区域。

Quick Settings 提供给用户非常便捷的按钮，用户甚至无需解锁就可以操作这个区域，通过点击 Quick Settings 中的 Tile 来切换某个功能的状态，

例如打开/关闭手电筒，蓝牙，Wifi 等功能。这对于用户来说是非常便捷的。

开发者 API

使用 Quick Settings 功能非常的简单，只需要与 Tile 和 TileService 两个类打交道即可。它们的类图如下图所示：



TileService 是 android.app.Service 的子类，开发者通过继承 TileService 并覆写其对应的方法来完成功能的实现。TileService 中提供的状态回调方法如下：

方法名	说明
onClick()	当前 Tile 被点击了
onDestroy()	当前 Tile 将要被销毁
onStartListening()	当前 Tile 将要进入监听状态

方法名	说明
onStopListening()	当前 Tile 将要退出监听状态
onTileAdded()	当前 Tile 被添加到 Quick Settings 中
onTileRemoved()	当前 Tile 被从 Quick Settings 中删除

在这些状态变更的时候，开发者可以根据状态的不同来调整 Tile 的状态。调整的方法就是：先通过 TileService.getQsTile()获取到当前 Tile，然后通过 Tile 的 setXXX 方法来修改。最后调用 Tile.updateTile()来使刚刚的设置生效。

下面是一段代码示例。这段代码的功能是根据用户点击来将 Tile 在 Active 和非 Active 状态之间进行切换。

```
private static final String SERVICE_STATUS_FLAG = "serviceStatus";
private static final String PREFERENCES_KEY =
    "com.google.android_quick_settings";

@Override
public void onClick() { ①
    Log.d("QS", "Tile tapped");
    updateTile();
}

// Changes the appearance of the tile.
private void updateTile() {

    Tile tile = this.getQsTile(); ②
    boolean isActive = getServiceStatus();

    Icon newIcon;
```

```
    String newLabel;
    int newState;

    // Change the tile to match the service status.
    if (isActive) {

        newLabel = String.format(Locale.US,
                        "%s %s",
                        getString(R.string.tile_label),
                        getString(R.string.service_active));

        newIcon = Icon.createWithResource(getApplicationContext(),
                      R.drawable.ic_android_black_24dp);

        newState = Tile.STATE_ACTIVE;

    } else {
        newLabel = String.format(Locale.US,
                "%s %s",
                getString(R.string.tile_label),
                getString(R.string.service_inactive));

        newIcon =
                Icon.createWithResource(getApplicationContext(),
                        android.R.drawable.ic_dialog_alert);

        newState = Tile.STATE_INACTIVE;
    }

    // Change the UI of the tile.
    tile.setLabel(newLabel); ③
    tile.setIcon(newIcon);
    tile.setState(newState);
```

```
    // Need to call updateTile for the tile to pick up changes.
    tile.updateTile(); ④
}
```

这段代码说明如下：

1. 处理用户的点击事件
2. 获取自身的 Tile 对象
3. 设置 Tile 的状态，包括：Label，Icon，State
4. 设置完成之后真正让状态生效

在实现完成这个 TileService 之后，我们还需要将其注册到 Manifest 中。TileService 需要设置一个特殊的权限和 Intent-Filter 的 Action，如下所示：

```
<service
    android:name=".QuickSettingsService"
    android:icon="@drawable/ic_android_black_dp"
    android:label="@string/tile_label"
    android:permission="android.permission.BIND_QUICK_SETTINGS_TILE">
    <intent-filter>
        <action android:name="android.service.quicksettings.action.QS_TILE" />
    </intent-filter>
</service>
```

当我们将包含这个 TileService 的应用安装到设备上之后，下划通知面板然后展开 Quick Settings 区域便可以看到我们开发的 Tile 了。

系统实现

我们可以通过前面提到的 Layout Inspector 工具来分析 Quick Settings 的结构。

Quick Settings 位于下拉的通知面板中。在布局上，这个部分通过 QSContainer 作为外部的容器，其中包含了一个 QSPanel。

QSPanel 中，包含了一个调节屏幕亮度的控件，这是通过一个 LinearLayout 来进行布局的，接下来就是 PagedTileLayout 中包含的多个 Tile 了，每个 Tile 用一个 QSTileView 来进行布局。PagedTileLayout 正如其名称所示，这是一个可以分页的 Layout。

QSContainer 中包含的元素如下图所示：

在 Android 系统中，包含两类 Tile：

- 一类是系统预置的
- 另一类的第三方应用中包含的

Quick Settings 功能实现主要位于这个目录中：
/frameworks/base/packages/SystemUI/src/com/android/systemui/qs。

系统预置 Tile

qs 目录下，包含了布局结构中用到的几个元素的实现类，包括：QSContainer，QSPanel，PagedTileLayout，QSTileView，QSIconView 等。

系统本身包含了一些预装的 Tile，例如：飞行模式的开关，位置信息的开关，热点功能的开关，手电筒功能开关等等。这些 Tile 的实现位于 qs/tiles 目录下，包含下面这些：

- AirplaneModeTile.java
- BatteryTile.java
- BluetoothTile.java
- CastTile.java
- CellularTile.java
- ColorInversionTile.java
- DataSaverTile.java
- DndTile.java
- FlashlightTile.java
- HotspotTile.java
- IntentTile.java
- LocationTile.java
- NightDisplayTile.java
- RotationLockTile.java
- UserTile.java
- WifiTile.java
- WorkModeTile.java

在 res 目录下，有一个名称为 `quick_settings_tiles_stock` 的字符串列出了所有系统内置的 Quick Setting 的名称，它们通过逗号进行分隔。

```
<string name="quick_settings_tiles_stock" translatable="false">
```

```
wifi,cell,battery,dnd,flashlight,rotation,bt,airplane,location,hotspot,inversion,saver,work,cast,night
</string>
```

QSTileHost 中为这里的名称和实现类做了映射：

```
// QSTileHost.java
public QSTile<?> createTile(String tileSpec) {
   if (tileSpec.equals("wifi")) return new WifiTile(this);
   else if (tileSpec.equals("bt")) return new BluetoothTile(this);
   else if (tileSpec.equals("cell")) return new CellularTile(this);
   else if (tileSpec.equals("dnd")) return new DndTile(this);
   else if (tileSpec.equals("inversion")) return new ColorInversionTile(this);
   else if (tileSpec.equals("airplane")) return new AirplaneModeTile(this);
   else if (tileSpec.equals("work")) return new WorkModeTile(this);
   else if (tileSpec.equals("rotation")) return new RotationLockTile(this);
   else if (tileSpec.equals("flashlight")) return new FlashlightTile(this);
   else if (tileSpec.equals("location")) return new LocationTile(this);
   else if (tileSpec.equals("cast")) return new CastTile(this);
   else if (tileSpec.equals("hotspot")) return new HotspotTile(this);
   else if (tileSpec.equals("user")) return new UserTile(this);
   else if (tileSpec.equals("battery")) return new BatteryTile(this);
   else if (tileSpec.equals("saver")) return new DataSaverTile(this);
   else if (tileSpec.equals("night")) return new NightDisplayTile(this);
   // Intent tiles.
   else if (tileSpec.startsWith(IntentTile.PREFIX)) return IntentTile.create(this,tileSpec);
   else if (tileSpec.startsWith(CustomTile.PREFIX)) return CustomTile.create(this,tileSpec);
   else {
       Log.w(TAG, "Bad tile spec: " + tileSpec);
       return null;
   }
}
```

TileQueryHelper 负责了 Tile 的初始化工作。在这个类中，会读取 R.string.quick_settings_tiles_stock 中的值，然后根据配置来初始化系统内置的 Quick Setting：

```
// TileQueryHelper.java
String possible = mContext.getString(R.string.quick_settings_tiles_stock);
String[] possibleTiles = possible.split(",");
final Handler qsHandler = new Handler(host.getLooper());
final Handler mainHandler = new Handler(Looper.getMainLooper());
for (int i = 0; i < possibleTiles.length; i++) {
  final String spec = possibleTiles[i];
  final QSTile<?> tile = host.createTile(spec);
  if (tile == null || !tile.isAvailable()) {
      continue;
  }
  tile.setListening(this, true);
  tile.clearState();
  tile.refreshState();
  tile.setListening(this, false);
  qsHandler.post(new Runnable() {
      @Override
      public void run() {
          final QSTile.State state = tile.newTileState();
          tile.getState().copyTo(state);
          // Ignore the current state and get the generic label instead.
          state.label = tile.getTileLabel();
          mainHandler.post(new Runnable() {
              @Override
              public void run() {
                  addTile(spec, null, state, true);
                  mListener.onTilesChanged(mTiles);
              }
          });
      }
```

```
  });
}
```

这段代码应该很简单，这里就不多做说明了。

第三方应用中包含的 Tile

对于 SystemUI 来说，除了要列出系统内置的 Quick Setting 之外，还有开发者开发的 Quick Setting 也需要读取。这部分逻辑通过 QueryTilesTask 以一个异步的 Task 来完成，这在这个异步任务中，会通过 PackageManager 查询所有开发者开发的 Quick Setting

```
// TileQueryHelper.java
private class QueryTilesTask extends
        AsyncTask<Collection<QSTile<?>>, Void, Collection<TileInfo>> {
    @Override
    protected Collection<TileInfo> doInBackground(Collection<QSTile<?>>... para
ms) {
        List<TileInfo> tiles = new ArrayList<>();
        PackageManager pm = mContext.getPackageManager();
        List<ResolveInfo> services = pm.queryIntentServicesAsUser(
                new Intent(TileService.ACTION_QS_TILE), 0, ActivityManager.getC
urrentUser()); ①
        String stockTiles = mContext.getString(R.string.quick_settings_tiles_st
ock);
        for (ResolveInfo info : services) { ②
            String packageName = info.serviceInfo.packageName;
            ComponentName componentName = new ComponentName(packageName, info.s
erviceInfo.name);

            // Don't include apps that are a part of the default tile set.
            if (stockTiles.contains(componentName.flattenToString())) { ③
                continue;
            }

            final CharSequence appLabel = info.serviceInfo.applicationInfo.load
Label(pm); ④
```

```
            String spec = CustomTile.toSpec(componentName);
            State state = getState(params[0], spec);
            if (state != null) {
                addTile(spec, appLabel, state, false);
                continue;
            }
            if (info.serviceInfo.icon == 0 && info.serviceInfo.applicationInfo.
icon == 0) {
                continue;
            }
            Drawable icon = info.serviceInfo.loadIcon(pm);
            if (!permission.BIND_QUICK_SETTINGS_TILE.equals(info.serviceInfo.pe
rmission)) {
                continue;
            }
            if (icon == null) {
                continue;
            }
            icon.mutate();
            icon.setTint(mContext.getColor(android.R.color.white));
            CharSequence label = info.serviceInfo.loadLabel(pm);
            addTile(spec, icon, label != null ? label.toString() : "null", appL
abel, mContext);
        }
        return tiles;
    }
```

这段代码说明如下：

1. 通过 PackageManager 查询所有设置了 TileService.ACTION_QS_TILE 的组件。PackageManager 负责了所有应用包信息的管理。
2. 遍历查询到的所有组件
3. 跳过系统预置的 Tile
4. 为每个 Tile 读取标签和图标

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通

12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计

3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计

7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究