目 录

CONTENTS

Doze与Standby模式

Doze低电耗模式

StandBy待机模式

白名单

监控电池电量和充电状态

耗电检测

Battery Historian

Energy Profiler

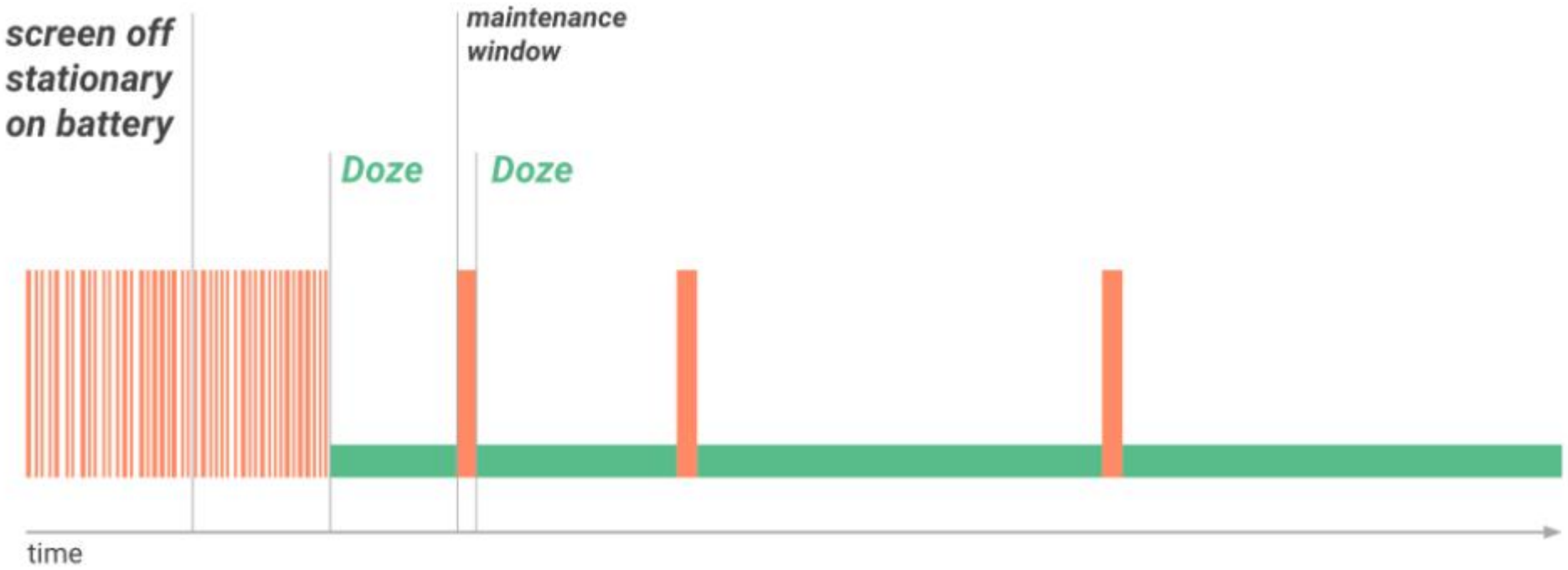
网络优化

protobuf使用

Doze低电耗模式

未充电、屏幕熄灭、让设备在一段时间内保持不活动状态。

延迟应用的后台 CPU 和网络活动，从而降低耗电量



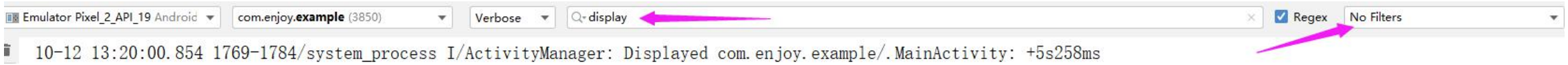
Doze测试

```
#启用Doze
adb shell dumpsys deviceidle enable
#强制进入doze模式 (同时还需要关闭屏幕)
adb shell dumpsys deviceidle force-idle

#退出doze模式
adb shell dumpsys deviceidle unforce
#关闭doze
adb shell dumpsys deviceidle disable
#重置设备
adb shell dumpsys battery reset


#查看doze白名单
adb shell dumpsys deviceidle whitelist
```

启动耗时统计

- 系统日志统计

在 Android 4.4（API 级别 19）及更高版本中，logcat 包含一个输出行，其中包含名为 Displayed 的值。此值代表从启动进程到在屏幕上完成对应 Activity 的绘制所用的时间。



- adb命令统计

adb shell am start -S -W [packageName]/[activityName]

```
C:\Users\Administrator>adb shell am start -S -W com.enjoy.example/.MainActivity
WARNING: linker: libdvm.so has text relocations. This is wasting memory and is a security risk. Please fix.
Stopping: com.enjoy.example
Starting: Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.enjoy.example/.MainActi
vity }
Status: ok
Activity: com.enjoy.example/.MainActivity
ThisTime: 5024
TotalTime: 5024
Complete
```

Standby待机模式

应用待机模式会延迟用户近期未与之交互的应用的后台网络活动。

当用户有一段时间未触摸应用并且应用没有以下表现，则Android系统就会使应用进入空闲状态

- 用户明确启动应用
- 应用当前有一个进程在前台运行（作为活动或前台服务，或者正在由其他活动或前台服务使用）。
- 应用生成用户可在锁定屏幕或通知栏中看到的通知。

当用户将设备插入电源时，系统会从待机状态释放应用，允许它们自由访问网络并执行任何待处理的作业和同步。如果设备长时间处于闲置状态，系统将允许闲置应用访问网络，频率大约每天一次。

白名单

系统提供了一个可配置的白名单，将部分免除低电耗模式和应用待机模式优化的应用列入其中。 在低电耗模式和应用待机模式期间，列入白名单的应用可以使用网络并保留部分唤醒锁定。

```
public static void addWhileList(Context context) {
    PowerManager pm = (PowerManager) context.getSystemService(Context.POWER_SERVICE);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (!pm.isIgnoringBatteryOptimizations(context.getPackageName())) {
            Intent intent = new Intent(Settings.ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS);
            intent.setData(Uri.parse("package:" + context.getPackageName()));
            context.startActivity(intent);
            context.startActivity(new Intent(Settings.ACTION_IGNORE_BATTERY_OPTIMIZATION_SETTINGS));
        }
    }
}
```

获取充电状态

为了减少电池续航被我们软件的影响，我们可以通过检查电池状态以及电量来判断是否进行某些操作。比如我们可以在充电时才进行一些数据上报之类的操作。

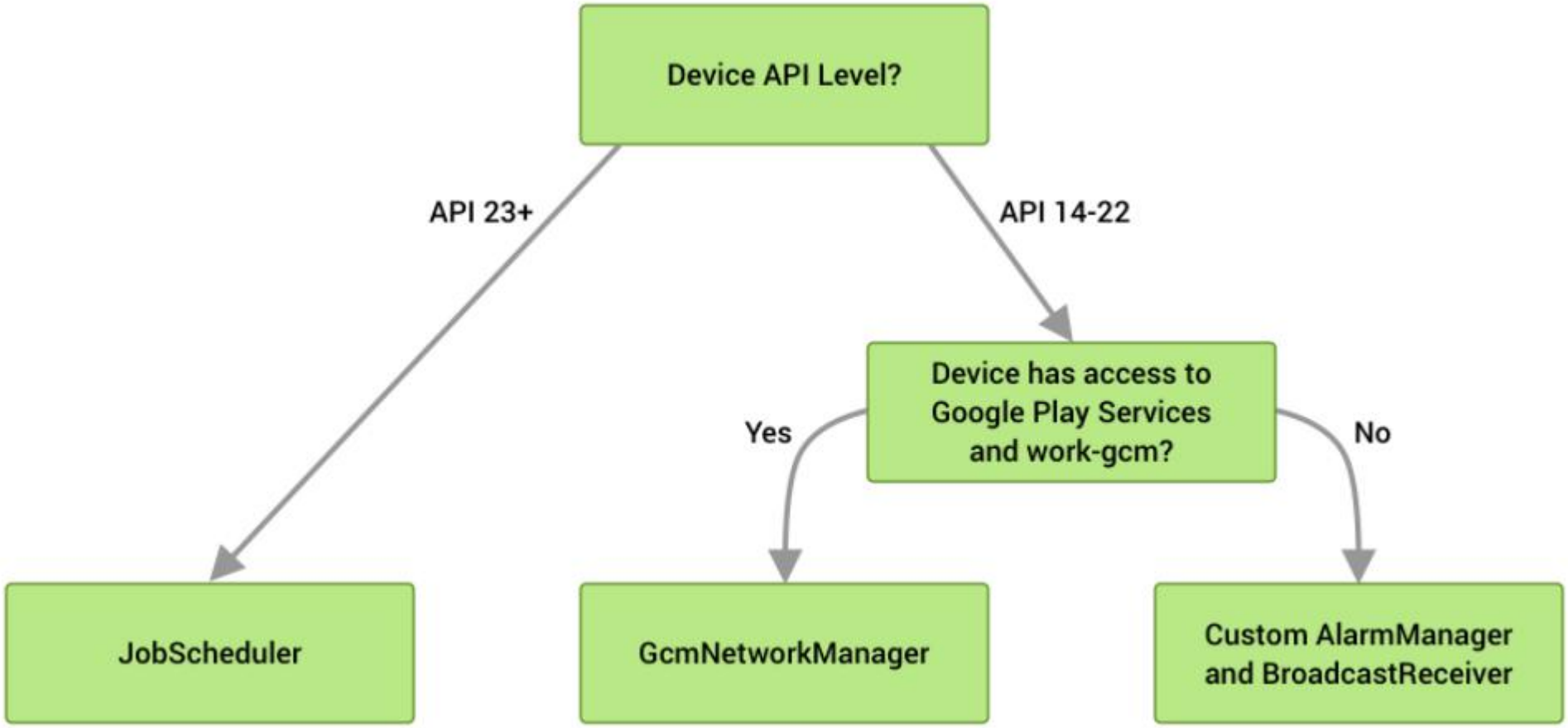
```
IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent batteryStatus = registerReceiver(null, ifilter);

// 是否正在充电
int status = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
boolean isCharging = status == BatteryManager.BATTERY_STATUS_CHARGING ||
                status == BatteryManager.BATTERY_STATUS_FULL;

// 什么方式充电?
int chargePlug = batteryStatus.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
//usb
boolean usbCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_USB;
//充电器
boolean acCharge = chargePlug == BatteryManager.BATTERY_PLUGGED_AC;

Log.e(TAG, "isCharging: " + isCharging + " usbCharge: " + usbCharge + " acCharge:" + acCharge);
```

监控充电状态变化

```
//注册广播
IntentFilter ifilter = new IntentFilter();
//充电状态
ifilter.addAction(Intent.ACTION_POWER_CONNECTED);
ifilter.addAction(Intent.ACTION_POWER_DISCONNECTED);
//电量显著变化
ifilter.addAction(Intent.ACTION_BATTERY_LOW); //电量不足
ifilter.addAction(Intent.ACTION_BATTERY_OKAY); //电量从低变回高
powerConnectionReceiver = new PowerConnectionReceiver();
registerReceiver(powerConnectionReceiver, ifilter);

public class PowerConnectionReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(Intent.ACTION_POWER_CONNECTED)) {
            Toast.makeText(context, "充电状态: CONNECTED", Toast.LENGTH_SHORT).show();
        } else if (intent.getAction().equals(Intent.ACTION_POWER_DISCONNECTED)) {
            Toast.makeText(context, "充电状态: DISCONNECTED", Toast.LENGTH_SHORT).show();
        } else if (intent.getAction().equals(Intent.ACTION_BATTERY_LOW)) {
            Toast.makeText(context, "电量过低", Toast.LENGTH_SHORT).show();
        } else if (intent.getAction().equals(Intent.ACTION_BATTERY_OKAY)) {
            Toast.makeText(context, "电量从低变回高", Toast.LENGTH_SHORT).show();
        }
    }
}
```

WorkManager

WorkManager API 是一个针对原有的 Android 后台调度 API 整合的建议替换组件。 如果设备在 API 级别 23 或更高级别上运行，系统会使用 JobScheduler。在 API 级别 14-22 上，系统会使用 GcmNetworkManager（如果可用），否则会使用自定义 AlarmManager 和 BroadcastReciever 实现作为备用。



目 录

CONTENTS

Doze与Standby模式

Doze低电耗模式

StandBy待机模式

白名单

监控电池电量和充电状态

耗电检测

Battery Historian

Energy Profiler

网络优化

protobuf使用

Battery Historian

Battery Historian是一个可以了解设备随时间的耗电情况的工具 。在系统级别，该工具以 HTML 的形式可视化来自系统日志的电源相关事件。在具体应用级别，该工具可提供各种数据，帮助您识别耗电的应用行为。

Battery Historian可以帮助我们查看应用是否具有以下耗电行为：

- 过于频繁地触发唤醒提醒（至少每 10 秒钟一次）。
- 持续保留 GPS 锁定。
- 至少每 30 秒调度一次作业。
- 至少每 30 秒调度一次同步。
- 使用手机无线装置的频率高于预期。

使用 Android 8.0 及以上版本的设备时，使用Energy Profiler 可以了解应用在哪里耗用了不必要的电量。

目 录

CONTENTS

Doze与Standby模式

Doze低电耗模式

StandBy待机模式

白名单

监控电池电量和充电状态

耗电检测

Battery Historian

Energy Profiler

网络优化

protobuf使用

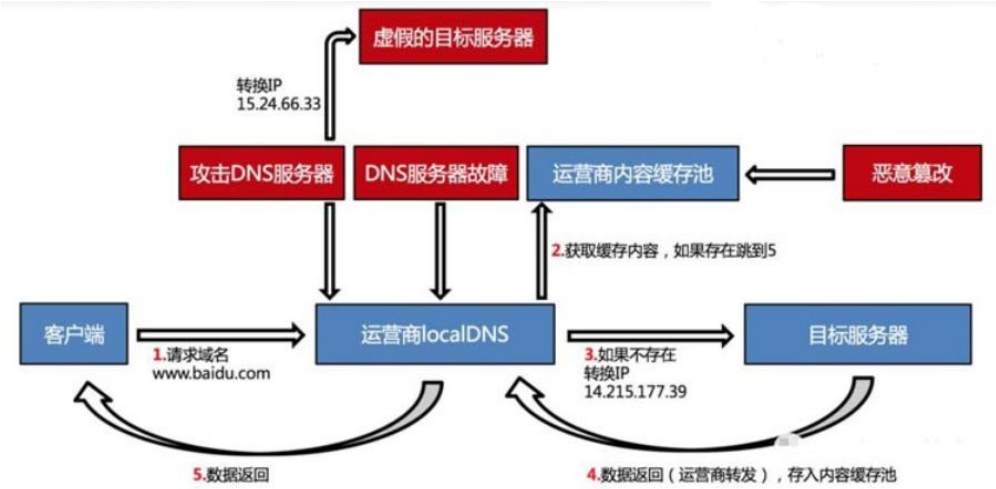
DNS优化

DNS（Domain Name System），它的作用是根据域名查出IP地址，它是HTTP协议的前提，只有将域名正确的解析成IP地址后，后面的HTTP流程才能进行。

DNS 完整的解析流程很长，会先从本地系统缓存取，若没有就到最近的 DNS 服务器取，若没有再到主域名服务器取，每一层都有缓存，但为了域名解析的实时性，每一层缓存都有过期时间。

传统的DNS解析机制有几个缺点：

- 缓存时间设置得长，域名更新不及时，设置得短，大量 DNS 解析请求影响请求速度；
- 域名劫持，容易被中间人攻击，或被运营商劫持，把域名解析到第三方 IP 地址，据统计劫持率会达到7%；
- DNS 解析过程不受控制，无法保证解析到最快的IP；
- 一次请求只能解析一个域名。



连接与数据优化

- 启用：‘keep-alive’（okhttp默认开启，但是仍需服务器支持）
- 使用http2
- 开启数据压缩(okhttp默认支持接收gzip压缩)
- 使用protobuf代替json、xml
- 使用webp代替png/jpg
- 不同网络的不同图片下发
- http开启缓存 / 关键数据加入缓存