Android (/tags/#Android) PowerManager (/tags/#PowerManager) Doze (/tags/#Doze)

Android电源管理之Doze模式专题系列（十一）

省电策略之网络控制

Posted by Cheson on May 12, 2017

今天是汶川地震9周年纪念日，早上看手机铺天盖地的新闻报道祭奠的盛况，犹如将一场悲剧的纪念日，不见其悲凉之景，或许只有当年亲历之人能懂其切骨之痛，虽时隔九年，犹能在亲人碑前潸然泪下。然细思其中滋味，痛着尤痛，何必揭开其伤疤，使其黯然。看到一同事朋友圈回忆其在成都的经历，虽无惊心动魄，但也是切实的震区感受。想当年我还在西安上学，也经历过类似的露宿操场的几天，对于未感受过大自然神力的我们来说，又怎么体会到那一份真正的恐惧，更多的怕是集体露宿的新鲜吧。写了一串不着边际的话，说是对社会的思考也好，亦或是对自己的思考也罢，每个人总会有自己不愿意去面对的痛。

借机煽情结束，来看这篇的主要内容吧。这是Doze模式中功耗优化的策略的第二篇，上一篇介绍了Idle状态下如何屏蔽电源锁的申请，这篇介绍Idle状态下如何控制网络连接来降低功耗。

在收到MSG_REPORT_IDLE_ON的消息进入到idle状态时，第二个省电策略的动作就是mNetworkPolicyManager.setDeviceIdleMode(true)

```
case MSG_REPORT_IDLE_ON: {
    EventLogTags.writeDeviceIdleOnStart();
    mLocalPowerManager.setDeviceIdleMode(true);
    try {
        /// M: integrate Doze and App Standby @{
        if(null != getDataShapingService()) {
                mDataShapingManager.setDeviceIdleMode(true);
        }
        /// integrate Doze and App Standby @}
        mNetworkPolicyManager.setDeviceIdleMode(true);
        mBatteryStats.noteDeviceIdleMode(true, null, Process.myUid());
    } catch (RemoteException e) {
    }
    getContext().sendBroadcastAsUser(mIdleIntent, UserHandle.ALL);
       EventLogTags.writeDeviceIdleOnComplete();
} break;
```

通过代码分析下网络部分是如何实现省电策略的，第一步当然是找到这个方法的具体定义的的地方了，可以看到mNetworkPolicyManager是INetworkPolicyManager接口类型的 `private INetworkPolicyManager mNetworkPolicyManager;` 从以往代码的经验来看可以推测实现应该是在一个叫做NetworkPolicyManagerService中，我们在framework中搜索下，果然存在这个服务 `./services/core/java/com/android/server/net/NetworkPolicyManagerService.java` ，而且确实实现了INetworkPolicyManager接口 `public class NetworkPolicyManagerService extends INetworkPolicyManager.Stub` 。对于这种方式，我们也是比较熟悉了，通过binder实现跨进程调用的一种方法。找到了代码位置，那么直接来看NetworkPolicyManagerService中setDeviceIdleMode方法实现。

```
@Override
public void setDeviceIdleMode(boolean enabled) {
    mContext.enforceCallingOrSelfPermission(MANAGE_NETWORK_POLICY, TAG);

    synchronized (mRulesLock) {
        if (mDeviceIdleMode != enabled) {
            mDeviceIdleMode = enabled;
            if (mSystemReady) {
                updateRulesForDeviceIdleLocked();
            }
            if (enabled) {
                EventLogTags.writeDeviceIdleOnPhase("net");
            } else {
                EventLogTags.writeDeviceIdleOffPhase("net");
            }
        }
    }
}
```

主要就是检查权限，判断条件然后核心部分就是调用updateRulesForDeviceIdleLocked来进入下一步操作。这边的逻辑是：1）将白名单进程和前台进程加入到防火墙的例外列表中（uidRules）；2）设置防火墙规则

```
void updateRulesForDeviceIdleLocked() {
    if (mDeviceIdleMode) {
        // sync the whitelists before enable dozable chain.  We don't care about the rules if
        // we are disabling the chain.
        SparseIntArray uidRules = new SparseIntArray();
        final List<UserInfo> users = mUserManager.getUsers();
        for (int ui = users.size() - 1; ui >= 0; ui--) {
            UserInfo user = users.get(ui);
            for (int i = mPowerSaveTempWhitelistAppIds.size() - 1; i >= 0; i--) {
                if (mPowerSaveTempWhitelistAppIds.valueAt(i)) {
                // idle状态下允许联网的临时白名单（来自用户定义）
                    int appId = mPowerSaveTempWhitelistAppIds.keyAt(i);
                    int uid = UserHandle.getUid(user.id, appId);
                    uidRules.put(uid, FIREWALL_RULE_ALLOW);
                }
            }
            for (int i = mPowerSaveWhitelistAppIds.size() - 1; i >= 0; i--) {
            // idle状态下允许联网的白名单（来自系统）
                int appId = mPowerSaveWhitelistAppIds.keyAt(i);
                int uid = UserHandle.getUid(user.id, appId);
                uidRules.put(uid, FIREWALL_RULE_ALLOW);
            }
        }
        for (int i = mUidState.size() - 1; i >= 0; i--) {
            if (isProcStateAllowedWhileIdle(mUidState.valueAt(i))) {
            // foreground进程
                uidRules.put(mUidState.keyAt(i), FIREWALL_RULE_ALLOW);
            }
        }
        setUidFirewallRules(FIREWALL_CHAIN_DOZABLE, uidRules);
    }
    enableFirewallChainLocked(FIREWALL_CHAIN_DOZABLE, mDeviceIdleMode);
}
```

这里需要说明的是mPowerSaveTempWhitelistAppIds这个白名单来自于用户自定义，具体设置在settings->battery->Battery optimization中设置不需要优化的app。mPowerSaveWhitelistAppIds白名单列表来自于系统配置，后续白名单篇中会专门讲解。mUidState存储的是forgound进程，这里判断是否加入到防火墙例外名单中的规则为，其优先级需要大于PROCESS_STATE_FOREGROUND_SERVICE

```
static boolean isProcStateAllowedWhileIdle(int procState) {
    return procState <= ActivityManager.PROCESS_STATE_FOREGROUND_SERVICE;
}
```

然后来看最重要的一步，设置防火墙规则

```
/**
 * Set uid rules on a particular firewall chain. This is going to synchronize the rules given
 * here to netd.  It will clean up dead rules and make sure the target chain only contains rul
 * specified here.
 */
private void setUidFirewallRules(int chain, SparseIntArray uidRules) {
    try {
        int size = uidRules.size();
        int[] uids = new int[size];
        int[] rules = new int[size];
        for(int index = size - 1; index >= 0; --index) {
            uids[index] = uidRules.keyAt(index);
            rules[index] = uidRules.valueAt(index);
        }
        mNetworkManager.setFirewallUidRules(chain, uids, rules);
    } catch (IllegalStateException e) {
        Log.wtf(TAG, "problem setting firewall uid rules", e);
    } catch (RemoteException e) {
        // ignored; service lives in system_server
    }
}
```

这里将uidRules中的key和values读出来分别存放到uids和rules中，然后通过NetManager来设置防火墙规则。这里会走到NetworkManagementService中的setFirewallUidRules方法。

```
@Override
public void setFirewallUidRules(int chain, int[] uids, int[] rules) {
    enforceSystemUid();
    synchronized (mQuotaLock) {
        SparseIntArray uidFirewallRules = getUidFirewallRules(chain);
        SparseIntArray newRules = new SparseIntArray();
        // apply new set of rules
        for (int index = uids.length - 1; index >= 0; --index) {
            int uid = uids[index];
            int rule = rules[index];
            setFirewallUidRule(chain, uid, rule);
            newRules.put(uid, rule);
        }
        // collect the rules to remove.
        SparseIntArray rulesToRemove = new SparseIntArray();
        for (int index = uidFirewallRules.size() - 1; index >= 0; --index) {
            int uid = uidFirewallRules.keyAt(index);
            if (newRules.indexOfKey(uid) < 0) {
                rulesToRemove.put(uid, FIREWALL_RULE_DEFAULT);
            }
        }
        // remove dead rules
        for (int index = rulesToRemove.size() - 1; index >= 0; --index) {
            int uid = rulesToRemove.keyAt(index);
            setFirewallUidRuleInternal(chain, uid, FIREWALL_RULE_DEFAULT);
        }
    }
}
```

然后调用了setFirewallUidRule方法对每个uid和rule进行设置

```
@Override
public void setFirewallUidRule(int chain, int uid, int rule) {
    enforceSystemUid();
    setFirewallUidRuleInternal(chain, uid, rule);
}
```

实际上就是检测了下调用的uid是否是SYSTEM_UID，然后通过setFirewallUidRuleInternal继续往下走

```
private void setFirewallUidRuleInternal(int chain, int uid, int rule) {
    synchronized (mQuotaLock) {
        SparseIntArray uidFirewallRules = getUidFirewallRules(chain);

        final int oldUidFirewallRule = uidFirewallRules.get(uid, FIREWALL_RULE_DEFAULT);
        if (DBG) {
            Slog.d(TAG, "oldRule = " + oldUidFirewallRule
                    + ", newRule=" + rule + " for uid=" + uid);
        }
        if (oldUidFirewallRule == rule) {
            if (DBG) Slog.d(TAG, "!!!!! Skipping change");
            // TODO: eventually consider throwing
            return;
        }

        try {
            String ruleName = getFirewallRuleName(chain, rule);
            String oldRuleName = getFirewallRuleName(chain, oldUidFirewallRule);

            if (rule == NetworkPolicyManager.FIREWALL_RULE_DEFAULT) {
                uidFirewallRules.delete(uid);
            } else {
                uidFirewallRules.put(uid, rule);
            }

            if (!ruleName.equals(oldRuleName)) {
                mConnector.execute("firewall", "set_uid_rule", getFirewallChainName(chain), ui
                        ruleName);
            }
        } catch (NativeDaemonConnectorException e) {
            throw e.rethrowAsParcelableException();
        }
    }
}
```

这里的核心内容在于通过getFirewallRuleName获取防火墙的规则是allow还是deny

```
private @NonNull String getFirewallRuleName(int chain, int rule) {
    String ruleName;
    if (getFirewallType(chain) == FIREWALL_TYPE_WHITELIST) {
        if (rule == NetworkPolicyManager.FIREWALL_RULE_ALLOW) {
            ruleName = "allow";
        } else {
            ruleName = "deny";
        }
    } else { // Blacklist mode
        if (rule == NetworkPolicyManager.FIREWALL_RULE_DENY) {
            ruleName = "deny";
        } else {
            ruleName = "allow";
        }
    }
    return ruleName;
}
```

如果规则有变化的话通过mConnector来进行新规则的执行，方法的定义在NativeDaemonConnector中，通过层层函数的调用最终走到的是

```
public NativeDaemonEvent[] executeForList(long timeoutMs, String cmd, Object... args)
        throws NativeDaemonConnectorException {
    final long startTime = SystemClock.elapsedRealtime();

    final ArrayList<NativeDaemonEvent> events = Lists.newArrayList();

    final StringBuilder rawBuilder = new StringBuilder();
    final StringBuilder logBuilder = new StringBuilder();
    final int sequenceNumber = mSequenceNumber.incrementAndGet();

    makeCommand(rawBuilder, logBuilder, sequenceNumber, cmd, args);
```

最后通过mOutputStream来执行这个command，实现了对网络防火墙规则的设定

```
synchronized (mDaemonLock) {
    if (mOutputStream == null) {
        throw new NativeDaemonConnectorException("missing output stream");
    } else {
        try {
            mOutputStream.write(rawCmd.getBytes(StandardCharsets.UTF_8));
        } catch (IOException e) {
            throw new NativeDaemonConnectorException("problem sending command", e);
        }
    }
}
```

PREVIOUS
ANDROID PERFORMANCE PATTERNS ——BATTERY PERFORMANCE **(/2017/04/19/ANDROID_PERF_PATTERNS_BATTERY/)**

NEXT
ANDROID电源管理之DOZE模式专题系列（十二） **(/2017/05/16/PM_DOZE_BATT_STAT/)**

FEATURED TAGS (/tags/)

前端 (/tags/#前端) Android (/tags/#Android) frameworks (/tags/#frameworks) AlarmManager (/tags/#AlarmManager) Performance (/tags/#Performance) systrace (/tags/#systrace) PowerManager (/tags/#PowerManager) Wakelock (/tags/#Wakelock) Guitar (/tags/#Guitar) 民谣 (/tags/#民谣) 赵雷 (/tags/#赵雷) Doze (/tags/#Doze) Android Performance Patterns (/tags/#Android Performance Patterns)

FRIENDS

待遇见志同道合的你 (https://github.com)　小明 (http://www.betterming.cn)

(https://twitter.com/chendongqi)

(https://www.zhihu.com/people/chendongqi)

(http://weibo.com/chendongqi) (https://www.facebook.com/chendongqi)

(https://github.com/chendongqi)

(https://www.linkedin.com/in/firstname-lastname-idxxxx)