**一、启动流程**

DropBoxManagerService(简称DBMS) 记录着系统关键log信息，主要功能用于Debug调试。 Android系统启动过程SystemServer进程时，在startOtherServices()过程会启动DBMS服务，如下：

**1.1 启动DBMS**

[-> SystemServer.java]

private void startOtherServices() {

//初始化DBMS，并登记该服务【见小节1.2】

ServiceManager.addService(Context.DROPBOX\_SERVICE,

new DropBoxManagerService(context, new File("/data/system/dropbox")));

...

}

其中DROPBOX\_SERVICE = “dropbox”, DBMS工作目录位于”/data/system/dropbox”，这个过程向ServiceManager 登记名为“dropbox”的服务。那么可通过dumpsys dropbox来查看该dropbox服务信息。

**1.2 初始化DBMS**

[-> DropBoxManagerService.java]

public final class DropBoxManagerService extends IDropBoxManagerService.Stub {

public DropBoxManagerService(final Context context, File path) {

mDropBoxDir = path; // 目录/data/system/dropbox

mContext = context;

mContentResolver = context.getContentResolver();

IntentFilter filter = new IntentFilter();

// 监听存储设备可用空间低的广播

filter.addAction(Intent.ACTION\_DEVICE\_STORAGE\_LOW);

// 监听开机完毕的广播

filter.addAction(Intent.ACTION\_BOOT\_COMPLETED);

context.registerReceiver(mReceiver, filter);

// Settings数据库变化时则回调广播接收者的onReceive方法,此处CONTENT\_URI=content://settings/global"

mContentResolver.registerContentObserver(

Settings.Global.CONTENT\_URI, true,

new ContentObserver(new Handler()) {

public void onChange(boolean selfChange) {

mReceiver.onReceive(context, (Intent) null);

}

});

mHandler = new Handler() {

public void handleMessage(Message msg) {

// 发送广播

if (msg.what == MSG\_SEND\_BROADCAST) {

mContext.sendBroadcastAsUser((Intent)msg.obj, UserHandle.OWNER,

android.Manifest.permission.READ\_LOGS);

}

}

};

}

}

该方法主要功能是给dropbox目录所对应的存储空间进行瘦身:

* 存储设备可用空间低；
* 开机完毕；
* Settings数据库变化；

当发生任一以上情况都会触发触发执行mReceiver的onReceive方法,接下来看看该onReceive()过程.

**1.3 mReceiver.onReceive**

[-> DropBoxManagerService.java]

private final BroadcastReceiver mReceiver = new BroadcastReceiver() {

public void onReceive(Context context, Intent intent) {

if (intent != null && Intent.ACTION\_BOOT\_COMPLETED.equals(intent.getAction())) {

mBooted = true;

return;

}

//收到ACTION\_DEVICE\_STORAGE\_LOW，则强制重新check存储空间

mCachedQuotaUptimeMillis = 0;

//创建工作线程来执行init和trim操作

new Thread() {

public void run() {

try {

init(); //【见小节1.3.1】

trimToFit(); //【见小节1.3.2】

} catch (IOException e) {

...

}

}

}.start();

}

};

**1.3.1 init**

private synchronized void init() throws IOException {

if (mStatFs == null) {

if (!mDropBoxDir.isDirectory() && !mDropBoxDir.mkdirs()) {

...

}

mStatFs = new StatFs(mDropBoxDir.getPath());

mBlockSize = mStatFs.getBlockSize(); //mBlockSize=4096

}

if (mAllFiles == null) {

File[] files = mDropBoxDir.listFiles();

// 列举所有的dropbox文件

mAllFiles = new FileList();

mFilesByTag = new HashMap<String, FileList>();

for (File file : files) {

if (file.getName().endsWith(".tmp")) {

file.delete(); //删除后缀为.tmp文件

continue;

}

// 创建dropbox的实体文件对象, 根据文件名来获取相应的时间戳

EntryFile entry = new EntryFile(file, mBlockSize);

if (entry.tag == null) {

continue; //忽略tag为空的文件

} else if (entry.timestampMillis == 0) {

file.delete(); //删除时间戳为0的文件

continue;

}

//将entry加入到mAllFiles对象

enrollEntry(entry);

}

}

}

该方法主要功能：

* 创建目录/data/system/dropbox;
* 列举该目录下所有文件,并对其进行:
  + 将每一个dropbox文件都对应于一个EntryFile对象,根据文件名来获取相应的时间戳
  + 删除后缀为.tmp的文件;
  + 删除时间戳为0的文件.

**1.3.2 trimToFit**

private synchronized long trimToFit() {

int ageSeconds = Settings.Global.getInt(mContentResolver,

Settings.Global.DROPBOX\_AGE\_SECONDS, DEFAULT\_AGE\_SECONDS);

int maxFiles = Settings.Global.getInt(mContentResolver,

Settings.Global.DROPBOX\_MAX\_FILES, DEFAULT\_MAX\_FILES);

long cutoffMillis = System.currentTimeMillis() - ageSeconds \* 1000;

while (!mAllFiles.contents.isEmpty()) {

EntryFile entry = mAllFiles.contents.first();

//当最老的文件时间戳在3天之内，且文件个数低于1000，则跳出循环

if (entry.timestampMillis > cutoffMillis

&& mAllFiles.contents.size() < maxFiles) break;

FileList tag = mFilesByTag.get(entry.tag);

if (tag != null && tag.contents.remove(entry)) tag.blocks -= entry.blocks;

if (mAllFiles.contents.remove(entry)) mAllFiles.blocks -= entry.blocks;

if (entry.file != null) entry.file.delete(); //删除文件

}

long uptimeMillis = SystemClock.uptimeMillis();

//除非接收设备存储低的广播，否则间隔5s才能再次执行restat

if (uptimeMillis > mCachedQuotaUptimeMillis + QUOTA\_RESCAN\_MILLIS) {

int quotaPercent = Settings.Global.getInt(mContentResolver,

Settings.Global.DROPBOX\_QUOTA\_PERCENT, DEFAULT\_QUOTA\_PERCENT);

int reservePercent = Settings.Global.getInt(mContentResolver,

Settings.Global.DROPBOX\_RESERVE\_PERCENT, DEFAULT\_RESERVE\_PERCENT);

int quotaKb = Settings.Global.getInt(mContentResolver,

Settings.Global.DROPBOX\_QUOTA\_KB, DEFAULT\_QUOTA\_KB);

//重新统计文件

mStatFs.restat(mDropBoxDir.getPath());

int available = mStatFs.getAvailableBlocks();

int nonreserved = available - mStatFs.getBlockCount() \* reservePercent / 100;

int maximum = quotaKb \* 1024 / mBlockSize;

//可用的块数量

mCachedQuotaBlocks = Math.min(maximum, Math.max(0, nonreserved \* quotaPercent / 100));

mCachedQuotaUptimeMillis = uptimeMillis;

}

if (mAllFiles.blocks > mCachedQuotaBlocks) {

//公平地限制所有tag的空间

int unsqueezed = mAllFiles.blocks, squeezed = 0;

TreeSet<FileList> tags = new TreeSet<FileList>(mFilesByTag.values());

for (FileList tag : tags) {

if (squeezed > 0 && tag.blocks <= (mCachedQuotaBlocks - unsqueezed) / squeezed) {

break;

}

unsqueezed -= tag.blocks;

squeezed++;

}

int tagQuota = (mCachedQuotaBlocks - unsqueezed) / squeezed;

//移除每个tags中的旧items

for (FileList tag : tags) {

if (mAllFiles.blocks < mCachedQuotaBlocks) break;

while (tag.blocks > tagQuota && !tag.contents.isEmpty()) {

EntryFile entry = tag.contents.first();

if (tag.contents.remove(entry)) tag.blocks -= entry.blocks;

if (mAllFiles.contents.remove(entry)) mAllFiles.blocks -= entry.blocks;

try {

if (entry.file != null) entry.file.delete();

enrollEntry(new EntryFile(mDropBoxDir, entry.tag, entry.timestampMillis));

} catch (IOException e) {

Slog.e(TAG, "Can't write tombstone file", e);

}

}

}

}

return mCachedQuotaBlocks \* mBlockSize;

}

trimToFit过程中触发条件是：当文件有效时长超过3天，或者最大文件数超过1000，再或者剩余可用存储设备过低；

DBMS有很多常量参数：

* DEFAULT\_AGE\_SECONDS = 3 \* 86400：文件最长可存活时长为3天
* DEFAULT\_MAX\_FILES = 1000：最大dropbox文件个数为1000
* DEFAULT\_QUOTA\_KB = 5 \* 1024：分配dropbox空间的最大值5M
* DEFAULT\_QUOTA\_PERCENT = 10：是指dropbox目录最多可占用空间比例10%
* DEFAULT\_RESERVE\_PERCENT = 10：是指dropbox不可使用的存储空间比例10%
* QUOTA\_RESCAN\_MILLIS = 5000：重新扫描retrim时长为5s

当然上面这些都是默认值，完全可以通过设置content://settings/global数据库中相应项来设定值。

**二、DropBox工作**

当发生以下任一场景，都会调用AMS.addErrorToDropBox()来触发DBMS工作。

* **crash:** 文章[理解Android Crash处理流程](http://gityuan.com/2016/06/24/app-crash/) [小节4]的AMS.handleApplicationCrashInner过程
* **anr:** 文章[android ANR原理分析](http://gityuan.com/2016/07/02/android-anr/)[小节3.1]的AMS.appNotResponding()过程；
* **watchdog:** 文章[WatchDog工作原理](http://gityuan.com/2016/06/21/watchdog/) [小节3.1]的Watchdog.run()过程;
* **native\_crash**: 当调用NativeCrashReporter.run()的过程;
* **wtf:** 当调用Log.wtf()或者Log.wtfQuiet()的过程；
* **lowmem:** 当内存较低时，触发AMS.reportMemUsage()过程；
* …

**2.1 AMS.addErrorToDropBox**

[–>ActivityManagerService.java]

public void addErrorToDropBox(String eventType, ProcessRecord process, String processName, ActivityRecord activity, ActivityRecord parent, String subject, final String report, final File logFile, final ApplicationErrorReport.CrashInfo crashInfo) {

//创建dropbox标签名【见小节2.1.1】

final String dropboxTag = processClass(process) + "\_" + eventType;

//获取dropbox服务的代理端

final DropBoxManager dbox = (DropBoxManager)

mContext.getSystemService(Context.DROPBOX\_SERVICE);

//当不需要输出dropbox报告则直接返回

if (dbox == null || !dbox.isTagEnabled(dropboxTag)) return;

final StringBuilder sb = new StringBuilder(1024);

//输出Process,flags,以及进程中所有package 【见小节2.1.2】

appendDropBoxProcessHeaders(process, processName, sb);

...

if (subject != null) {

sb.append("Subject: ").append(subject).append("\n");

}

sb.append("Build: ").append(Build.FINGERPRINT).append("\n");

sb.append("\n");

//创建新线程，避免将调用者阻塞在I/O

Thread worker = new Thread("Error dump: " + dropboxTag) {

@Override

public void run() {

if (report != null) {

//比如ANR时输出Cpuinfo，或者lowmem时输出的内存信息

sb.append(report);

}

if (logFile != null) {

//比如anr或者Watchdog时输出的traces文件(kill -3)，最大上限为256KB

sb.append(FileUtils.readTextFile(logFile, DROPBOX\_MAX\_SIZE,

"\n\n[[TRUNCATED]]"));

}

if (crashInfo != null && crashInfo.stackTrace != null) {

// 比如crash时输出的调用栈

sb.append(crashInfo.stackTrace);

}

String setting = Settings.Global.ERROR\_LOGCAT\_PREFIX + dropboxTag;

int lines = Settings.Global.getInt(mContext.getContentResolver(), setting, 0);

//当dropboxTag所对应的settings项不等于0，则输出logcat

if (lines > 0) {

//输出evets/system/main/crash这些log信息

java.lang.Process logcat = new ProcessBuilder("/system/bin/logcat",

"-v", "time", "-b", "events", "-b", "system", "-b", "main",

"-b", "crash",

"-t", String.valueOf(lines)).redirectErrorStream(true).start();

input = new InputStreamReader(logcat.getInputStream());

int num;

char[] buf = new char[8192];

//不断读取input中的log内容，并添加到sb

while ((num = input.read(buf)) > 0) sb.append(buf, 0, num);

...

}

//将log信息输出到DropBox 【见小节2.2】

dbox.addText(dropboxTag, sb.toString());

}

};

if (process == null) {

//当进程为空，意味着system\_server进程崩溃，系统可能很快就要挂了,

//那么不再创建新线程，而是直接在system\_server进程中同步运行

worker.run();

} else {

//启动新线程

worker.start();

}

}

该方法主要功能是输出以下内容项：

1. Process,flags, package等头信息；
2. 当report不为空，则比如ANR时输出Cpuinfo，或者lowmem时输出的内存信息
3. 当logFile不为空，则比如anr或者Watchdog时输出的traces文件(kill -3)，最大上限为256KB；
4. 当stack不为空，则比如crash时输出的调用栈；
5. 输出logcat的events/system/main/crash信息。

**2.1.1 AMS.processClass**

private static String processClass(ProcessRecord process) {

//MY\_PID代表的是当前进程pid，正是system\_server进程

if (process == null || process.pid == MY\_PID) {

return "system\_server";

} else if ((process.info.flags & ApplicationInfo.FLAG\_SYSTEM) != 0) {

return "system\_app";

} else {

return "data\_app";

}

}

dropbox文件名格式为dropboxTag@xxx.txt xxx代表时间戳,例如system\_server\_crash@1465650845355.txt,则记录该文件时间戳为1465650845355. 文件后缀除了.txt，还有压缩格式.txt.gz. 对于dropboxTag是由processClass + eventType组合而成.

* processClass分为system\_server, system\_app, data\_app;
* eventType：分为crash,anr,wtf,native\_cras,lowmem, watchdog

列举部分常见tags以及含义:

| **dropboxTag** | **含义** |
| --- | --- |
| system\_server\_anr | system进程无响应 |
| system\_server\_watchdog | system进程发生watchdog |
| system\_server\_crash | system进程崩溃 |
| system\_server\_native\_crash | system进程native出现崩溃 |
| system\_server\_wtf | system进程发生严重错误 |
| system\_server\_lowmem | system进程内存不足 |

当然除了system\_server进程, 还有system\_app, data\_app类型的进程, 以上所有类型都适用,列举部分:

|  |  |
| --- | --- |
| system\_app\_crash | 系统app崩溃 |
| system\_app\_anr | 系统app无响应 |
| data\_app\_crash | 普通app崩溃 |
| data\_app\_anr | 普通app无响应 |

**2.1.2 AMS.appendDropBoxProcessHeaders**

private void appendDropBoxProcessHeaders(ProcessRecord process, String processName, StringBuilder sb) {

if (process == null) {

sb.append("Process: ").append(processName).append("\n");

return;

}

synchronized (this) {

sb.append("Process: ").append(processName).append("\n");

int flags = process.info.flags;

IPackageManager pm = AppGlobals.getPackageManager();

sb.append("Flags: 0x").append(Integer.toString(flags, 16)).append("\n");

for (int ip=0; ip<process.pkgList.size(); ip++) {

String pkg = process.pkgList.keyAt(ip);

sb.append("Package: ").append(pkg);

try {

PackageInfo pi = pm.getPackageInfo(pkg, 0, UserHandle.getCallingUserId());

if (pi != null) {

sb.append(" v").append(pi.versionCode);

if (pi.versionName != null) {

sb.append(" (").append(pi.versionName).append(")");

}

}

} catch (RemoteException e) {

...

}

sb.append("\n");

}

}

}

该方法输出的信息:

* 进程名;
* 进程的ApplicationInfo的flags信息;
* 进程中所有的包名以及版本信息;

这里列举头信息实例:

2016-11-11 22:22:22 system\_app\_anr (compressed text, 26165 bytes)

Process: com.android.systemui

Flags: 0x40d83e0d

Package: com.android.systemui v21 (5.0.2) Subject: Broadcast of Intent { act=android.intent.action.TIME\_TICK flg=0x50000014 (has extras) }

**2.2 DBM.addText**

[-> DropBoxManager.java]

public void addText(String tag, String data) {

try {

//data数据封装到Entry对象实例 【见小节2.3】

mService.add(new Entry(tag, 0, data));

} catch (RemoteException e) {

...

}

}

在DropBoxManager中有addText, addData, addFile方法，三分归一统，对应于DBMS的add()方法。

**2.3 DBMS.add**

[ -> DropBoxManagerService.java]

public void add(DropBoxManager.Entry entry) {

File temp = null;

OutputStream output = null;

final String tag = entry.getTag();

try {

int flags = entry.getFlags();

...

init(); // 初始化【见小节1.3.1】

long max = trimToFit(); // 压缩空间【见小节1.3.2】

long lastTrim = System.currentTimeMillis();

byte[] buffer = new byte[mBlockSize];

InputStream input = entry.getInputStream();

int read = 0;

while (read < buffer.length) {

int n = input.read(buffer, read, buffer.length - read);

if (n <= 0) break;

read += n;

}

//创建临时文件，例如tid=1234的对应文件drop1234.tmp

temp = new File(mDropBoxDir, "drop" + Thread.currentThread().getId() + ".tmp");

int bufferSize = mBlockSize;

if (bufferSize > 4096) bufferSize = 4096;

if (bufferSize < 512) bufferSize = 512;

FileOutputStream foutput = new FileOutputStream(temp);

output = new BufferedOutputStream(foutput, bufferSize);

//创建gzip压缩文件

if (read == buffer.length && ((flags & DropBoxManager.IS\_GZIPPED) == 0)) {

output = new GZIPOutputStream(output);

flags = flags | DropBoxManager.IS\_GZIPPED;

}

//不断将temp文件数据写入buffer

do {

output.write(buffer, 0, read);

long now = System.currentTimeMillis();

if (now - lastTrim > 30 \* 1000) {

max = trimToFit(); //执行时间超过30s则执行trim

lastTrim = now;

}

read = input.read(buffer);

if (read <= 0) {

FileUtils.sync(foutput);

output.close();

output = null;

} else {

output.flush();

}

long len = temp.length();

if (len > max) {

temp.delete();

temp = null;

break;

}

} while (read > 0);

//[见小节2.3.1]

long time = createEntry(temp, tag, flags);

temp = null;

final Intent dropboxIntent = new Intent(DropBoxManager.ACTION\_DROPBOX\_ENTRY\_ADDED);

dropboxIntent.putExtra(DropBoxManager.EXTRA\_TAG, tag);

dropboxIntent.putExtra(DropBoxManager.EXTRA\_TIME, time);

if (!mBooted) {

dropboxIntent.addFlags(Intent.FLAG\_RECEIVER\_REGISTERED\_ONLY);

}

//发送广播MSG\_SEND\_BROADCAST

mHandler.sendMessage(mHandler.obtainMessage(MSG\_SEND\_BROADCAST, dropboxIntent));

} catch (IOException e) {

...

} finally {

if (output != null) output.close();

entry.close();

if (temp != null) temp.delete();

}

}

**2.3.1 DBMS.createEntry**

[ -> DropBoxManagerService.java]

private synchronized long createEntry(File temp, String tag, int flags) throws IOException {

long t = System.currentTimeMillis(); //当当前时间作为dropbox文件的时间戳

...

if (temp == null) {

enrollEntry(new EntryFile(mDropBoxDir, tag, t));

} else {

enrollEntry(new EntryFile(temp, mDropBoxDir, tag, t, flags, mBlockSize));

}

return t;

}

关于时间戳问题:

1. EntryFile(File file, int blockSize) : 从file文件获取时间戳,并保存到EntryFile.timestampMillis. init()过程使用.
2. 其他的构造方法则都会在创建时,将当前时间保存到EntryFile.timestampMillis.比如EntryFile(File dir, String tag, long timestampMillis)

**三. 总结**

DBMS服务的数据保存目录为/data/system/dropbox。

当出现crash, anr, wtf，lowmem，以及开机完成时都会通过DropBoxManager， 收集系统的重要信息： Process,flags, package等头信息和logcat信息。 另外就是根据不同场景输出相应的信息，例如：

1. CRASH：输出发生crash时的当前线程的调用栈信息；
2. ANR：输出Cpuinfo，以及重要进程的各个线程的traces文件(kill -3)；
3. Watchdog: 也输出重要进程的各个线程的traces文件(kill -3)

# 参考

http://gityuan.com/2016/06/12/DropBoxManagerService/