说明：本次说的系统服务，是指捆绑在system\_process进程中的服务。

  仔细阅读源码可以发现，在frameworks/base/services/java/com/android/server/SystemServer.java中添加了很多服务，什么熵信息、电量、wife等等服务，这些服务启动都在launcher之前，一般要继承Binder或者继承一个AIDL。下面试着来添加一个简单系统服务

一。在frameworks/base/core/java/android/os/ 增加一个aidl文件，最后用aidl工具生产一个Java文件，这样可以方便后面：

**[html]** [view plaincopy](http://blog.csdn.net/leolaurel/article/details/8136804)

1. package android.os;

**[html]** [view plaincopy](http://blog.csdn.net/leolaurel/article/details/8136804)

1. /\*\*
2. \* Interface that allows controlling and querying YtTts connectivity.
3. \*
4. \* {@hide}
5. \*/
6. interface IYtTtsService{
7. /\*\*
8. \* 播放Text文字
9. \*/
10. int speak(String text);
11. }

然后在修改/frameworks/base/Android.mk，在LOCAL\_SRC\_FILES里添加一行：core/java/android/os/IYtTtsService.aidl \

不确定补充：把AIDL文件写在这个地方似乎可以避免加@hide

二 。在frameworks/base/services/java/com/android/server/ 新建要增加的service，继承上面aidl生产的java：

**[html]** [view plaincopy](http://blog.csdn.net/leolaurel/article/details/8136804)

1. package com.android.server;
2. import android.content.Context;
3. import android.os.IYtTtsService;
4. import android.os.RemoteException;
5. import android.util.Log;

**[html]** [view plaincopy](http://blog.csdn.net/leolaurel/article/details/8136804)

1. public class YtTtsService  extends IYtTtsService.Stub {
2. private WorkerThread mWorker;
3. public YtTtsService(Context context) {
4. super();
5. mWorker = new WorkerThread("YtTtsServiceWorker");
6. mWorker.start();
7. Log.i("[YT]", "In YtTtsService--**>**"+context);
8. }
9. @Override
10. public int speak(String text) throws RemoteException {
11. Log.i("[YT]", "In YtTtsService--**>**"+text);
12. return 0;
13. }
14. private class WorkerThread extends Thread{
15. int i = 0;
16. public WorkerThread(String name) {
17. super(name);
18. }
19. public void run() {
20. while(true){
21. i++;
22. Log.i("[YT]", "In YtTtsService--**>**WorkerThread running "+i);
23. try {
24. Thread.sleep(3000);
25. } catch (InterruptedException e) {
26. e.printStackTrace();
27. }
28. }
29. }
30. }
31. }

三。在frameworks/base/services/java/com/android/server/SystemServer.java, ServerThread中run()方法中增加代码：

**[html]** [view plaincopy](http://blog.csdn.net/leolaurel/article/details/8136804)

1. try {
2. Log.i("[YT]", "In SystemServer, YtTts add..");
3. ServiceManager.addService("yttts", new YtTtsService(context));
4. } catch (Throwable e) {
5. Log.i("[YT]", "In SystemServer, YtTts add err..");
6. }

5.在/frameworks/base/core/java/cn/youngtone/tts建一个YtTtsManager类,用于管理YtTtsService：

**[html]** [view plaincopy](http://blog.csdn.net/leolaurel/article/details/8136804)

1. package cn.youngtone.tts;
3. import android.content.Context;
4. import android.os.IYtTtsService;
5. import android.os.RemoteException;
6. import android.util.Log;
8. public class YtTtsManager {
9. IYtTtsService service;
11. public YtTtsManager(Context context ,IYtTtsService service) {
12. this.service = service;
13. }
14. public void speak(String text) {
15. try {
16. Log.e("[YT]" , "In YtTtsManage--**>**speak():"+text);
17. service.speak(text);
18. } catch (RemoteException e) {
19. Log.e("[YT]" , "In YtTtsManage--**>**E:RemoteException");
20. e.printStackTrace();
21. }
22. }
23. }

四。 在ContextImpl getSystemService中添加代码：

**[java]** [view plaincopy](http://blog.csdn.net/leolaurel/article/details/8136804)

1. **if** ("yttts".equals(name)) {

**[java]** [view plaincopy](http://blog.csdn.net/leolaurel/article/details/8136804)

1. **if** (mYtTtsManager == **null**) {
2. IBinder b = ServiceManager.getService("yttts");
3. IYtTtsService service = IYtTtsService.Stub.asInterface(b);
4. mYtTtsManager = **new** YtTtsManager(**this**, service);
5. }
6. **return** mYtTtsManager;
7. }

五。测试代码

**[java]** [view plaincopy](http://blog.csdn.net/leolaurel/article/details/8136804)

1. YtTtsManager mManager = (YtTtsManager)getSystemService("yttts");
2. mManager.speak("这是一个简单的系统服务!");

“存储空间不足”data过小的通知分析DeviceStorageMonitorService类

位置

DeviceStorageMonitorService.java (frameworks\base\services\java\com\android\server\)

一. 三个关键变量:

private static final int DEFAULT\_THRESHOLD\_PERCENTAGE = 10;

private static final int DEFAULT\_FULL\_THRESHOLD\_BYTES = 1\*1024\*1024; // 1MB

private static final long DEFAULT\_DISK\_FREE\_CHANGE\_REPORTING\_THRESHOLD = 2 \* 1024 \* 1024; // 2MB

二. 参数调用的地方  
在整个android目录中, 三个常量被调用的地方, 都只是在DeviceStorageMonitorService.java中

DEFAULT\_THRESHOLD\_PERCENTAGE(10) : private long getMemThreshold();

return 1300234 \* 10 = 13,002,340

DEFAULT\_FULL\_THRESHOLD\_BYTES(10M) : private int getMemFullThreshold()

return 10,485,760

DEFAULT\_DISK\_FREE\_CHANGE\_REPORTING\_THRESHOLD(5M=5,242,880) : private final void restatDataDir();

threshold = 5,242,880 delta = mFreeMem = 101,482,696

三. 函数被调用的地方  
DeviceStorageMonitorService构造中

private long mMemLowThreshold = getMemThreshold();

四. DeviceStorageMonitorService的调用追踪

private final void checkMemory(boolean checkCache) -> 判断() ->

private final void sendNotification()

在sendNotification()中将剩余内存的值存入intent中, 然后广播出去, 广播类型为ACTION\_MANAGE\_PACKAGE\_STORAGE  
此通知的标题为 “存储空间不足(Low on space)”  
通知的内容为 “手机内存空间所剩不多了(Phone storage space is getting low)”

五. DEFAULT\_FULL\_THRESHOLD\_BYTES的调用追踪

private final void checkMemory(boolean checkCache) -> 判断() ->

private final void sendFullNotification() ->

sendStickyBroadcast(intent.ACTION\_DEVICE\_STORAGE\_FULL)

若data目录小于DEFAULT\_FULL\_THRESHOLD\_BYTES, 则不在接收短信.

六. DEFAULT\_DISK\_FREE\_CHANGE\_REPORTING\_THRESHOLD 调用追踪

private final void checkMemory(boolean checkCache) ->

EventLog.writeEvent(EventLogTags.FREE\_STORAGE\_CHANGED, mFreeMem)

经过jwisp烧录到真机上发现, 真正有可能有用的是百分比报警, 那只需要设置下面这个参数即可:  
DEFAULT\_THRESHOLD\_PERCENTAGE

在负责文件系统模块的过程中，经常会碰到由于系统空间被消耗完而导致的问题，因此要确保为系统功能（如数据库同步）保留一定的空间。在功能机中一般是由文件系统模块预留，那么在Android系统是怎样对设备存储空间进行管理和监控的呢？

如果你在使用Android手机时有过把memory填满或者即将填满的经历，也许你会注意到在这种情况下手机的Notifications栏会有“**Storage****space****running****out**”的通知。当点开该通知你会发现Setting–>**Storage** settings –>Device memory 下会有如下提示：Not enough**storage space**.

这个服务的实现是在android/framework/base/services/java/com/android/server/DeviceStorageMonitorService.java。DeviceStorageMonitorService类实现了一个监控设备上存储空间的服务。如果设备的剩余存储空间小于某一个阀值（默认是存储空间的10%，这个值可以设置）时将会向用户发送剩余空间不足的警告，让用户释放一些空间。

下面就分析一下这个类。首先看一下该服务是如何被添加进来的。在android/frameworks/base/services/java/com/android/server/SystemServer.java中使用ServiceManager.addService（）来添加系统服务:

在SystemServer中添加DSMS服务：

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 |  | try { Slog.i(TAG, “Device **Storage** Monitor”); ServiceManager.addService(DeviceStorageMonitorService.SERVICE, new DeviceStorageMonitorService(context)); } catch (Throwable e) { reportWtf(“starting DeviceStorageMonitor service”, e); } |

DSMS的构造函数的代码如下：

DSMS服务的构造函数：

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 |  | /\*\* \* Constructor to run service. initializes the disk **space** threshold value \* and posts an empty message to kickstart the process. \*/ public DeviceStorageMonitorService(Context context) { mLastReportedFreeMemTime = 0; mContext = context; mContentResolver = mContext.getContentResolver(); //create StatFs object mDataFileStats = new StatFs(DATA\_PATH); //获取Data分区信息； mSystemFileStats = new StatFs(SYSTEM\_PATH); //获取System分区信息； mCacheFileStats = new StatFs(CACHE\_PATH); //获取Cache分区信息； //initialize total **storage** on device // 初始化设备总空间信息 // mTotalMemory 用于保存Data分区总空间； mTotalMemory = ((long)mDataFileStats.getBlockCount() \* mDataFileStats.getBlockSize())/100L; /\* 创建3个Intent，分别用于通知存储空间不足（ACTION\_DEVICE\_**STORAGE**\_LOW）、 存储空间回复正常（ACTION\_DEVICE\_**STORAGE**\_OK）和存储空间满（ACTION\_DEVICE\_**STORAGE**\_FULL）。 由于每个Intent都设置了FLAG\_RECEIVER\_REGISTERED\_ONLY\_BEFORE\_BOOT标志，因此这三个Intent只 能由注册了的BroadcastReceiver接收。 \*/ mStorageLowIntent = new Intent(Intent.ACTION\_DEVICE\_**STORAGE**\_LOW); mStorageLowIntent.addFlags(Intent.FLAG\_RECEIVER\_REGISTERED\_ONLY\_BEFORE\_BOOT); mStorageOkIntent = new Intent(Intent.ACTION\_DEVICE\_**STORAGE**\_OK); mStorageOkIntent.addFlags(Intent.FLAG\_RECEIVER\_REGISTERED\_ONLY\_BEFORE\_BOOT); mStorageFullIntent = new Intent(Intent.ACTION\_DEVICE\_**STORAGE**\_FULL); mStorageFullIntent.addFlags(Intent.FLAG\_RECEIVER\_REGISTERED\_ONLY\_BEFORE\_BOOT); mStorageNotFullIntent = new Intent(Intent.ACTION\_DEVICE\_**STORAGE**\_NOT\_FULL); mStorageNotFullIntent.addFlags(Intent.FLAG\_RECEIVER\_REGISTERED\_ONLY\_BEFORE\_BOOT); // cache **storage** thresholds /\* 查询Seetings数据库中sys\_**storage**\_threshod\_percentage的值，默认是10，即当DATA\_PATH 目录下剩余空间少于其总空间的10%时，认为空间不足（ACTION\_DEVICE\_**STORAGE**\_LOW）。 \*/ mMemLowThreshold = getMemThreshold(); /\* 查询Settings数据库中的sys\_**storage**\_full\_threshold\_bytes的值，默认是1MB，即当DATA\_PATH 目录下剩余空间小于等于1M时，任务空间已满，剩余的部分是保留给系统使用的。 \*/ mMemFullThreshold = getMemFullThreshold(); /\* 开始检查，存储空间； \*/ checkMemory(true); } |

下面再来看一下checkMemory()方法的实现。

checkMemory()方法：

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 |  | private final void checkMemory(boolean checkCache) { //if the thread that was started to clear cache is still **running** do nothing till its //finished clearing cache. Ideally this flag could be modified by clearCache // and should be accessed via a lock but even if it does this test will fail now and //hopefully the next time this flag will be set to the correct value. //如果线程正在清除缓存CACHE\_PATH ，那么不进行空间检查。 if(mClearingCache) { if(localLOGV) Slog.i(TAG, “Thread already **running** just skip”); //make sure the thread is not hung for too long long diffTime = System.currentTimeMillis() – mThreadStartTime; if(diffTime > (10\*60\*1000)) { Slog.w(TAG, “Thread that clears cache file seems to run for ever”); } } else { //重新计算3个分区的剩余空间大小； restatDataDir(); if (localLOGV)  Slog.v(TAG, “freeMemory=”+mFreeMem);  //post intent to NotificationManager to display icon if necessary ////如果剩余空间低于mMemLowThreshold，先做一次缓存清理； if (mFreeMem < mMemLowThreshold) { if (!mLowMemFlag) { if (checkCache) { // See if clearing cache helps // Note that clearing cache is asynchronous and so we do a // memory check again once the cache has been cleared. //首先清除缓存 mThreadStartTime = System.currentTimeMillis(); mClearSucceeded = false; clearCache(); } else { //如果空间仍然低于mMemLowThreshold,发送广播并在状态来设置一个 //警告通知； Slog.i(TAG, “**Running** low on memory. Sending notification”); sendNotification(); mLowMemFlag = true; } } else { if (localLOGV) Slog.v(TAG, “**Running** low on memory ” + “notification already sent. do nothing”); } } else { if (mLowMemFlag) { //剩余空间不小于mMemLowThreshold，且已经设置了mLowMemFlag，则 //取消空间不足广播。 Slog.i(TAG, “Memory available. Cancelling notification”); cancelNotification(); mLowMemFlag = false; } } if (mFreeMem < mMemFullThreshold) { //如果空间已满，则发送空间已满的广播； if (!mMemFullFlag) { sendFullNotification(); mMemFullFlag = true; } } else { if (mMemFullFlag) { //空间不满且已经发送了空间已满的广播，则在此取消。 cancelFullNotification(); mMemFullFlag = false; } } } if(localLOGV) Slog.i(TAG, “Posting Message again”); //keep posting messages to itself periodically //DEFAULT\_CHECK\_INTERVAL为1分钟，即每1分钟会触发一次检查 postCheckMemoryMsg(true, DEFAULT\_CHECK\_INTERVAL); } //mLowMemFlag和mMemFullFlag为是否发送了广播的标识。  当空间不足时，DSMS会先尝试clearCache函数，该函数内部会与PackageManager-Service（以下简称PKMS）交互，其代码如下：  [-->DeviceStorageManagerService.java::clearCache]  private final void clearCache() {  if (mClearCacheObserver == null) {     //创建一个CachePackageDataObserver对象,当PKMS清理完空间时会回调该对象的     //onRemoveCompleted函数      mClearCacheObserver = new CachePackageDataObserver();  }  mClearingCache = true;//设置mClearingCache的值为true，表示我们正在清理空间  try {    //调用PKMS的freeStorageAndNotify函数以清理空间，这个函数在分析PKMS时再介绍   IPackageManager.Stub.asInterface(           ServiceManager.getService("package")).           freeStorageAndNotify(mMemLowThreshold, mClearCacheObserver); //该函数在PackageManagerservice.java } ......  }  CachePackageDataObserver是DSMS定义的内部类，其onRemoveCompleted函数很简单，就是重新发送消息DEVICE\_MEMORY\_WHAT  (postCheckMemoryMsg(false, 0);)，让DSMS再检测一次存储空间。如果剩余空间小于10%，则发送sendNotification()提示 storage space running out。 DeviceStorageManagerService的功能单一，没有重载dump函数。而DiskStats-Service唯一有用的就是dump功能了。不知Google的工程师为什么没有把DeviceStorage-ManagerService和DiskStatsService的功能整合到一起。  ps: 你饿了,想吃饭,就一会去问你妈一声"开饭没有啊?"这就正常函数调用.  但是今天你妈包饺子,花的时间比较长,你跑啊跑啊,就烦了.于是你给你妈说,我先出去玩会,开饭的时候打我手机.等过了一阵,你妈给你打电话说"开饭啦,快回来吃饭吧!"  其中,你告诉你妈打手机找你,就是个你把回调函数句柄保存到你妈的动作.你妈打电话叫你,就是个回调过程. 总结： （1）首先在构造函数中，获取data，system，cache分区信息，然后注册四个intent,分别为低内存，内存ok，内存满，内存没有满四种情况。然后获取settings数据库里的data目录下剩余空间少于其总空间的百分比值，获取数据库中data目录下剩余空间的大小临界值（用于提示用户空间已满）。然后开始检查，存储空间；checkMemory(true); （2） 在检查存储空间时，首先判断如果线程正在清除缓存CACHE\_PATH ，那么不进行空间检查。否则重新计算3个分区的剩余空间大小。如果剩余空间低于百分比10%，如果需要做缓存清理，先做一次缓存清理；清理完毕后会再次进行新一轮的checkMemory，如果剩余空间低于百分比10%并不用做缓存清理并且没有发通知，则发送通知告诉用户内部空间超出最低值10%，如果此时空间百分比正常，但已发送通知，则将通知取消。同样的，如果空间已满，大于full的临界值，则发送空间已满的广播；空间不满且已经发送了空间已满的广播，则取消。最后会每1分钟会触发一次检查空间checkMemory。 如下：清理完毕后会再次进行新一轮的checkMemory: 做一下onRemoveCompleted动作，该动作发送检查空间的消息【postCheckMemoryMsg(false, 0);这里的false就是不再执行clearCache】，然后handle处理该消息DEVICE\_MEMORY\_WHAT，再次进入checkMemory（false），发送通知告诉用户storage space running out，空間剩10%時會出現该提示。 |