dragondog的专栏

个人资料

bluedog

访问： 93634次
积分： 1068
等级： BLOG 4
排名： 千里之外

原创： 12篇 转载： 0篇
译文： 0篇 评论： 16条

Activity的启动流程

标签： token null thread application string object

2011-03-09 18:56 19373人阅读 评论(7) 收藏 举报

分类： Android基础框架（3）

以前看了很多，时间长了都忘了，所以还是勤快点，把看到的都记下来，算是给自己点积累。

Activity启动分为很多种情况，这里说的是打开新的应用程序第一个Activity的流程。

1. AcitivityManager产生新进程，新进程从**Android**.app.ActivityThread.main开始运行。这里就是一般意义上的程序入口点，类似于C的main函数。

ActivityManagerService.**Java**

```
 private final void startProcessLocked(ProcessRecord app, String hostingType, String hostingNameStr) {

    // Process里面通知Zygote服务，Zygote真正产生新的进程（准确说是复制一个）

    int pid = Process.start("android.app.ActivityThread",
                mSimpleProcessManagement ? app.processName : null, uid, uid,

 }
```

2.ActivityThread的main函数中将Looper准备起来，prepareMainLooper标志着这是应用程序主线程的Looper对象。

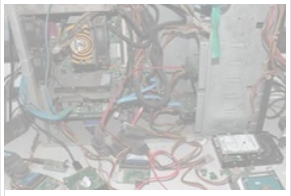
ActivityThread.java

```
 public static final void main(String[] args) {

   Looper.prepareMainLooper();


   ActivityThread thread = new ActivityThread();
   thread.attach(false);
   // 这里闭合消息循环
   Looper.loop();
}
```

3. 接下来调用attach，参数为false，表明这不是系统进程，是给普通应用

ActivityThread.java

Android ListView快速查找 (2)
Android性能优化总结 (2)
Activity的启动流程（二） (1)
Android的CursorAdapter (0)
Android数据库接口性能分 (0)
HandlerThread (0)
Android数据库随笔 (0)



```
private final void attach(boolean system) {

        ViewRoot.addFirstDrawHandler(new Runnable() {
            public void run() {
                ensureJitEnabled();
            }
        });
        RuntimeInit.setApplicationObject(mAppThread.asBinder());
        IActivityManager mgr = ActivityManagerNative.getDefault();
        mgr.attachApplication(mAppThread);

}
```

mAppThread是ApplicationThread对象，是提供给ActivityManagerService控制ActivityThread的Bi

```
    private final class ApplicationThread extends ApplicationThreadNative {

      public final void schedulePauseActivity(IBinder token, boolean finished,boolean userLeaving, int
    configChanges){

            queueOrSendMessage(finished ? H.PAUSE_ACTIVITY_FINISHING : H.PAUSE_ACTIVITY,
                token, (userLeaving ? 1 : 0),  configChanges);
        }

      public final void scheduleStopActivity(IBinder token, boolean showWindow, int configChanges) {
            queueOrSendMessage(showWindow ? H.STOP_ACTIVITY_SHOW : H.STOP_ACTIVITY_HIDE,
                token, 0, configChanges);
        }

      .........

    }
```

ActivityManagerService要Pause当前Activity，就会调用schedulePauseActivity想本地的消息循环中加入一个H.PAUSE_ACTIVITY的消息，然后立即返回以避免ActivityManagerService的阻塞。

4.现在又回到了ActivityManagerService中

ActivityManagerService.java

```
    // ActivityManagerService中XXXLocked函数才是真正干活的地方，XXX只是个套

    public final void attachApplication(IApplicationThread thread) {
        int callingPid = Binder.getCallingPid();
        attachApplicationLocked(thread, callingPid);
    }


    private final boolean attachApplicationLocked(IApplicationThread thr

        // 这里取出对应该Pid的ProcessRecord对象，如果取不出来，你就

        app = mPidsSelfLocked.get(pid)

        // 为ProcessRecord对象补充信息

        app.thread = thread;
        app.curAdj = app.setAdj = -100;
```


```
        app.curSchedGroup = Process.THREAD_GROUP_DEFAULT;
        app.setSchedGroup = Process.THREAD_GROUP_BG_NONINTERACTIVE;
        app.forcingToForeground = null;
        app.foregroundServices = false;
        app.debugging = false;
        // 清除timeout监测
        mHandler.removeMessages(PROC_START_TIMEOUT_MSG, app);


        // 回调之前的ActivityThread，让它记住自己在ActivityManagerService中的相关信息，传这么大坨东西，真给力

        thread.bindApplication(processName, app.instrumentationInfo != null
                ? app.instrumentationInfo : app.info, providers,
                app.instrumentationClass, app.instrumentationProfileFile,
                app.instrumentationArguments, app.instrumentationWatcher, testMode,
                isRestrictedBackupMode || !normalMode,
                mConfiguration, getCommonServicesLocked());

        // topRunningActivityLocked的意思没看太明白

        HistoryRecord hr = topRunningActivityLocked(null);

        // 启动Activity

        realStartActivityLocked(hr, app, true, true);

    }


    private final boolean realStartActivityLocked(HistoryRecord r,
            ProcessRecord app, boolean andResume, boolean checkConfig){

        // 这里又跑到ActivityThread中去了

        app.thread.scheduleLaunchActivity(new Intent(r.intent), r, System.identityHashCode(r),
                r.info, r.icicle, results, newIntents, !andResume, isNextTransitionForward());

    }
```

5.ActivityThread开始调度用户的Activity启动了

ActivityThread.java

```
private final void handleLaunchActivity(ActivityRecord r, Intent customIntent) {
        Activity a = performLaunchActivity(r, customIntent);

        .....

}
```



```
private final void performLaunchActivity(ActivityRecord r, Intent custom

        java.lang.ClassLoader cl = r.packageInfo.getClassLoader();
        // 产生Activity
        activity = mInstrumentation.newActivity(cl, component.getClassNa

        // 将新生的Activity与当前应用关联
```

```
        activity.attach(appContext, this, getInstrumentation(), r.token,
                        r.ident, app, r.intent, r.activityInfo, title, r.parent,
                        r.embeddedID, r.lastNonConfigurationInstance,
                        r.lastNonConfigurationChildInstances, config);

        .....
}
```

6.Acivity与当前App关联，直到这里，平时应用所见的Activity才真正被构建

Actiivty.java

```
final void attach(Context context, ActivityThread aThread, Instrumentation instr, IBinder token, int
    Application application, Intent intent, ActivityInfo info,  CharSequence title, Activity parent, Strin
    Object lastNonConfigurationInstance, HashMap<String,Object> lastNonConfigurationChildInsta
            Configuration config) {
        // 记录传入ContexImpl实例，这就是平时所见的Activity Context
        attachBaseContext(context);
        // 创建与Activity关联的Window，可以简单理解为显示窗口
        mWindow = PolicyManager.makeNewWindow(this);

        mUiThread = Thread.currentThread();

        mMainThread = aThread;
        mInstrumentation = instr;
        mToken = token;
        mIdent = ident;
        mApplication = application;
        mIntent = intent;
        mComponent = intent.getComponent();
        mActivityInfo = info;
        mTitle = title;
        mParent = parent;
        mEmbeddedID = id;
       // 构建WindowManager的代理LocalWindowManager
        mWindow.setWindowManager(null, mToken, mComponent.flattenToString());
        if (mParent != null) {
            // 难道说整个Activity栈中的Activity都有同一个Container
            mWindow.setContainer(mParent.getWindow());
        }
        mWindowManager = mWindow.getWindowManager();

    }
```


7.ActivityThread

ActivityThread.java

```
private final void performLaunchActivity(ActivityRecord r, Intent custom

        .....

        // 回调Activity::onCreate
```

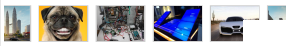
```
        mInstrumentation.callActivityOnCreate(activity, r.state);

        // 记录新产生的Activity
        mActivities.put(r.token, r);

    }


    private final void handleLaunchActivity(ActivityRecord r, Intent customIntent) {

        .....

        handleResumeActivity(r.token, false, r.isForward);

    }


    final void handleResumeActivity(IBinder token, boolean clearHide, boolean isForward) {
        ActivityRecord r = performResumeActivity(token, clearHide);



    }


    public final ActivityRecord performResumeActivity(IBinder token, boolean clearHide) {

        if (r.pendingIntents != null) {
            deliverNewIntents(r, r.pendingIntents);
            r.pendingIntents = null;
        }
        if (r.pendingResults != null) {
            deliverResults(r, r.pendingResults);
            r.pendingResults = null;
        }
        //回调Activity::onResume
        r.activity.performResume();

    }
```

到这里，Activity从产生到初始化的过程就全部结束了。之后就是等待用户界面的消息，根据消息进行相应的处理。整个过程中，在ActivityManagerServer、ActivityThread的互相协作下构建出Activity，并在相应的时机回调Activity的相应接口，完成Activity的初始化。

上一篇 Android的CursorAdapter与CursorFilter机制
下一篇 Android性能优化总结

相关文章推荐

- Activity启动流程
- Android A



查看评论

6楼 Boolean布尔 2016-07-28 09:11发表


博主，我要给你生孩子

5楼 qluojieq 2014-10-23 10:19发表


怎么这么厉害，你是怎么做到的，这个知识能告诉我是怎么样领悟到的呀，看源码吗！

4楼 放风筝的骚年 2012-05-09 14:58发表


很好 很感谢

3楼 virus026 2011-08-19 10:25发表


I love you man!

2楼 quechao1234 2011-08-10 10:36发表


您好，感谢楼主的分享
我有一个问题：Activity启动时是如何读取AndroidManifest.xml中的相关信息的？如screenOrientation属性，是在哪个函数中完成的.

Re: bluedog 2011-08-26 09:35发表


回复quechao1234：AndroidManifest.xml 在apk安装时，PackageMangerService解析出所有AndroidManifest中列出的Activity以及信息，其中就包括screenOrientation属性。
解析的过程在scanPackageLI()中，得到的Activity信息保存在ActivityInfo中。
Activity启动时，ActivityManagerService会从PackageMangerService读取该信息，

1楼 forever_crying 2011-07-03 02:13发表


很感谢能作者把这篇文章分享出来