Android线程模型与service简介

徐涛

大纲

- 1. 线程介绍
- 2. 用java创建一个线程
- 3. android的线程模型
- 4.主线程与其它线程通信

线程简介

- 一般来说,我们把正在计算机中执行的程序叫做"进程"(Process) ,而不将其称为程序(Program)。所谓"线程"(Thread),是"进程"中某个单一顺序的控制流。
- 从os的角度看，进程是资源分配的单位，线程是调度的基本单位
- 常见的操作系统如linux，windows等都支持多线程。并且一般来说，一个进程内都包含多个执行的线程

线程简介

为什么需要多线程？

- 设想这样的场景：当你通过socket从一个网络读取数据时，有时候会读不到数据，因为数据可能在网络传输过程中被延迟了。如果程序阻塞在read上，那么直到数据到达前它就什么都不能做。假如用户界面上有一个按钮，并且用户在程序阻塞时点击这个按钮，那么因为此时程序不能处理鼠标事件,也不能执行与按钮事件相关连的处理方法，所以什么都不会发生。这种使用户觉得程序被挂起的情况会让用户感到沮丧。
- 我们需要额外的线程处理一些事物，而不是把什么都交给主线程

线程简介
为什么要用多线程

- 单线程模型会在没有考虑到它的影响的情况下引起Android应用程序性能低下，因为所有的任务都在同一个线程中执行，如果执行一些耗时的操作，如访问网络或查询数据库，会阻塞整个用户界面。当在执行一些耗时的操作的时候，不能及时地分发事件，包括用户界面重绘事件。从用户的角度来看，应用程序看上去像挂掉了。更糟糕的是，如果阻塞应用程序的时间过长（现在大概是5秒钟）Android会向用户提示一些信息，即打开一个“应用程序没有相应（application not responding）”的对话框。

线程简介
多线程的好处

- 许多情况下，在一个程序中使用多线程是有益处的：
- **1.与用户的更好交互（Better Interaction with the User）**
- **2.模拟同时进行的活动（Simulation of Simultaneous Activities）**
- **3.开发利用多处理器（Exploitation of Multiple Processors）**
- **4.当等待慢的I/O操作时，可以做其他事情（Do Other Things While Waiting for Slow IO Operations）**

创建一个线程

```
class WorkThread extends Thread {
    @Override
    public void run() {
        while (true) {
            //TODO: add implements code here
        }
    }

}
WorkThread work = new WorkThread();
work.start();
```

android线程简介

- 在Android 里，在默认情况下，一个程序用到的各种“组件”(如Activity，BroadcastReceiver 或Service等)都会在同一个进程(Process)里执行，而且由该进程的主线程负责执行之。

android线程简介

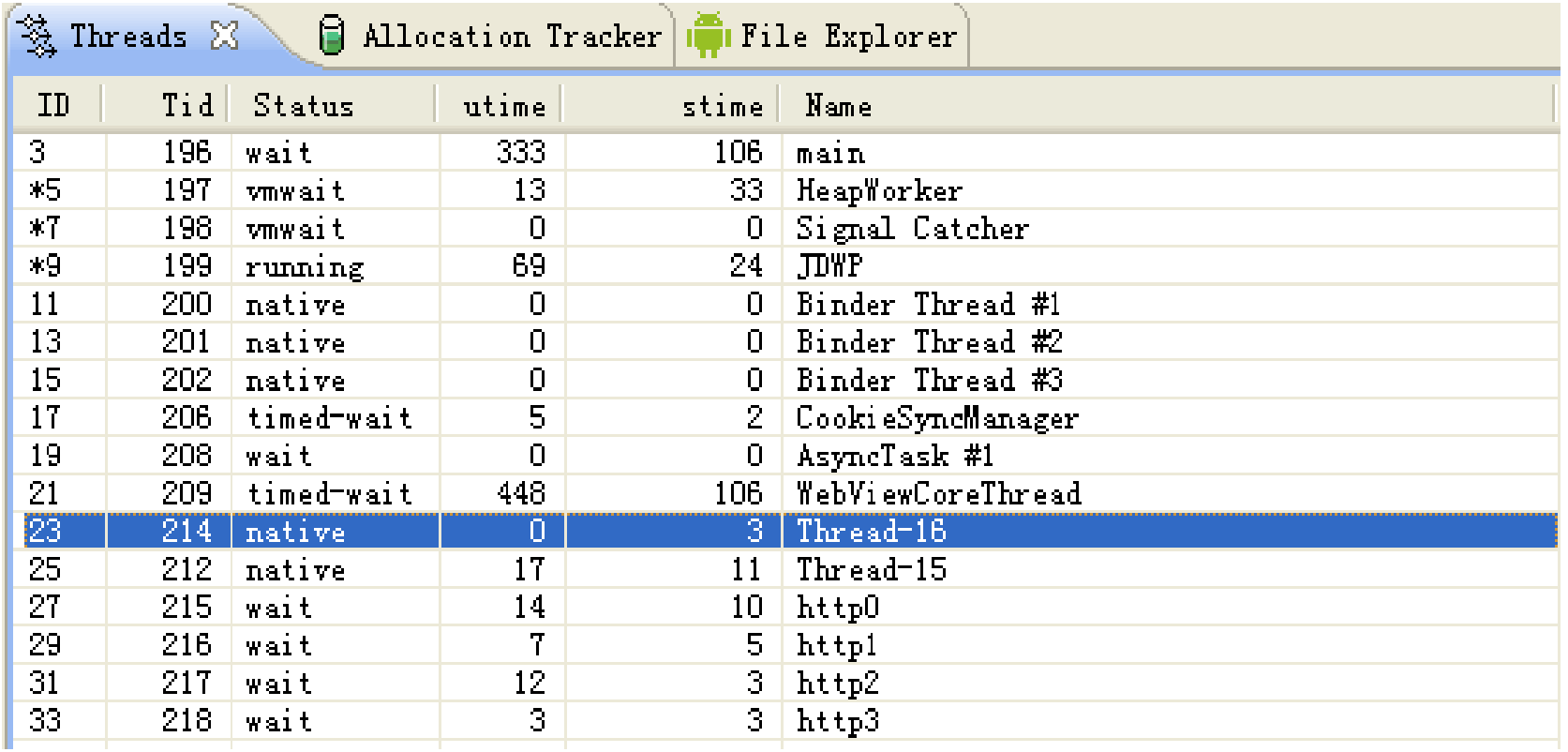
- 在Android 里，如果有特别指定，也可以让特定“组件”在不同的进程里执行。无论这些组件在哪一个进程里执行，默认情况下，他们都是由该进程里的主线程来负责执行之。

android线程简介

- 主线程除了要处理Activity 的UI 事件，又要处理Service 后台服务工作， 通常会忙不过来。该如何化解这种困境呢? 此时， 多线程 (Multi-thread)的并行(Concurrent)派上用场了，其可以化解主线程太过于忙碌的情 形。也就是说，主线程可以诞生多个子线程来分担其工作，尤其是比较冗长费时的后台服务工作，例如播放动画的背景音乐、或从网络上下载电影等。于是，主线程就能专心于处理UI画面的事件了。

线程简介

android浏览器里的多线程

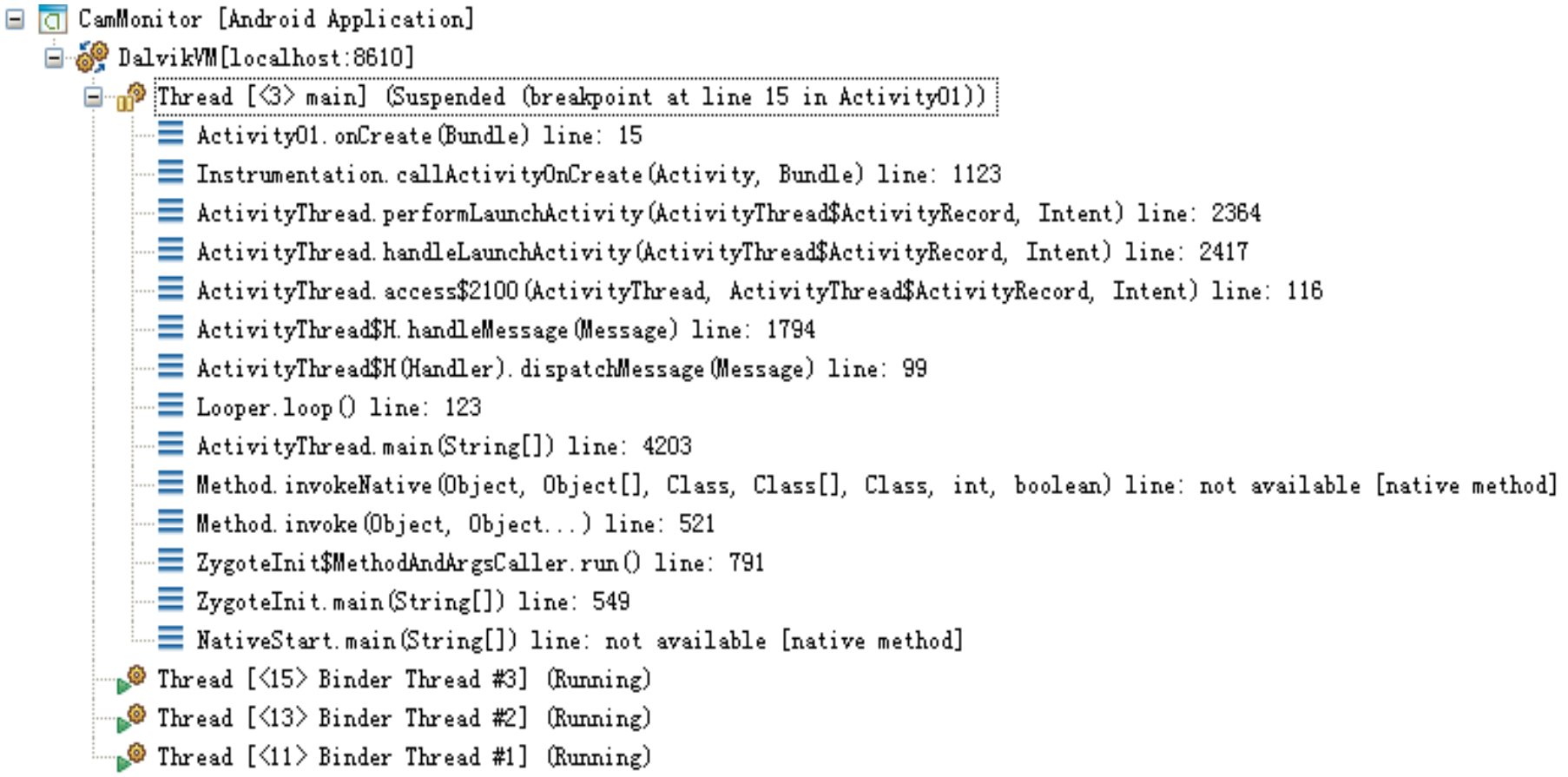


ID	Tid	Status	utime	stime	Name
3	196	wait	333	106	main
*5	197	vmwait	13	33	HeapWorker
*7	198	vmwait	0	0	Signal Catcher
*9	199	running	69	24	JDWP
11	200	native	0	0	Binder Thread #1
13	201	native	0	0	Binder Thread #2
15	202	native	0	0	Binder Thread #3
17	206	timed-wait	5	2	CookieSyncManager
19	208	wait	0	0	AsyncTask #1
21	209	timed-wait	448	106	WebViewCoreThread
23	214	native	0	3	Thread-16
25	212	native	17	11	Thread-15
27	215	wait	14	10	http0
29	216	wait	7	5	http1
31	217	wait	12	3	http2
33	218	wait	3	3	http3

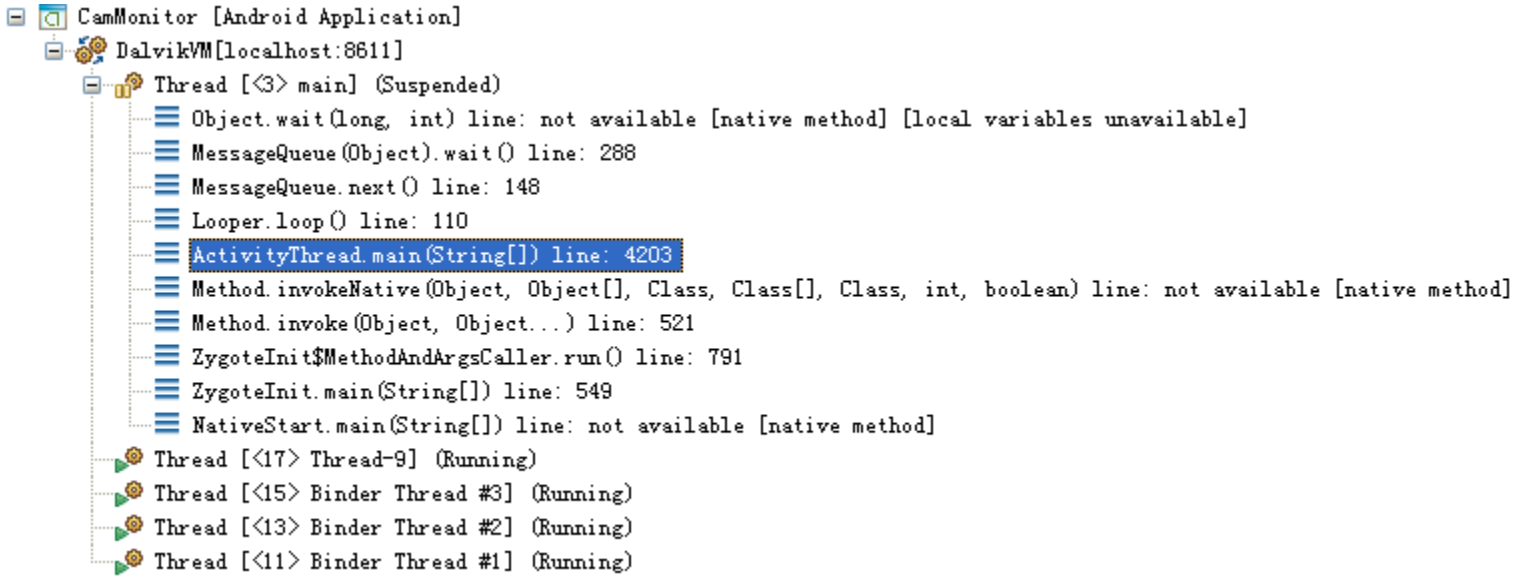
Android线程模型

- Android的主线程里有一个消息队列，线程在一个循环中检测是否有新的消息到来，如果没有消息在队列中，则主线程挂起等待（如下图）。

Android线程模型



Android线程模型
(主线程等待示意图)



Android线程模型

----指导思想

- 把费时的工作交给子线程
- 子线程把结果反馈给主线程
- android线程之间传递数据的桥梁：**android.os.Handler**

Android线程模型

handler示例：创建handler


```
static final int MSG_GET_NBA_SCHEDULE = 0x100;
MyH mHandler = new MyH();
class MyH extends Handler{

    @Override
    public void handleMessage(Message msg) {
        switch(msg.what){
        case MSG_GET_NBA_SCHEDULE:
            String schedule = (String) msg.obj;
            TextView tv = (TextView)findViewById(R.id.schedule);
            tv.setText(schedule);
        }
    }

}
```

Android线程模型

handler示例：使用handler投递消息到主线程的消息队列

```
class Worker extends Thread{

    @Override
    public void run() {

        //从网络获取赛程信息
         try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //......略
        //将数据送给UI线程显示

        Message msg= Message.obtain();
        msg.obj = "Rocket vs sun , 2010.10.31  9:30  ";
        msg.what = MSG_GET_NBA_SCHEDULE;
        mHandler.sendMessage(msg);
    }

}
```

Android线程模型

handler示例：在主线程中使用启动子线程

```
public class TestHandler extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);


        Worker w = new Worker();
        w.start();
    }
```

Android线程模型

UI元素只能由主线程修改

- Andriod提供了几种在其他线程中访问UI线程的方法：
- 1. Activity.runOnUiThread(Runnable)
- 2. View.post(Runnable)
- 3. View.postDelayed(Runnable, long)
- 4. Hanlder

Android线程模型

AsyncTask类

- 使用handler的方法会使你的代码很较难理解，当你需要实现一些很复杂的操作并需要频繁地更新UI时这会变得更糟糕。为了解决这个问题，Android 1.5提供了一个工具类：AsyncTask，它使创建需要与用户界面交互的长时间运行的任务变得更简单。

Android线程模型
AsyncTask类


```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

Android线程模型

- UI线程向子线程发送消息（注意不是子线程向ui线程发送消息）

Android线程模型
子线程的handler定义

```
class Worker extends Thread{
    private WorkerHandler mWorkerHandler = new WorkerHandler();
    class WorkerHandler extends Handler{
        @Override
        public void handleMessage(Message msg) {
            Toast.makeText(TestHandler.this,(CharSequence) msg.obj,Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void run() {

           Message msg = Message.obtain();
            msg.obj="hello,i'm worker thread";

            mHandler.sendMessage(msg );


        Looper.prepare();
        Looper.loop();
    }

    public Handler getWorkerHandler() {
        return mWorkerHandler;
    }
}
```

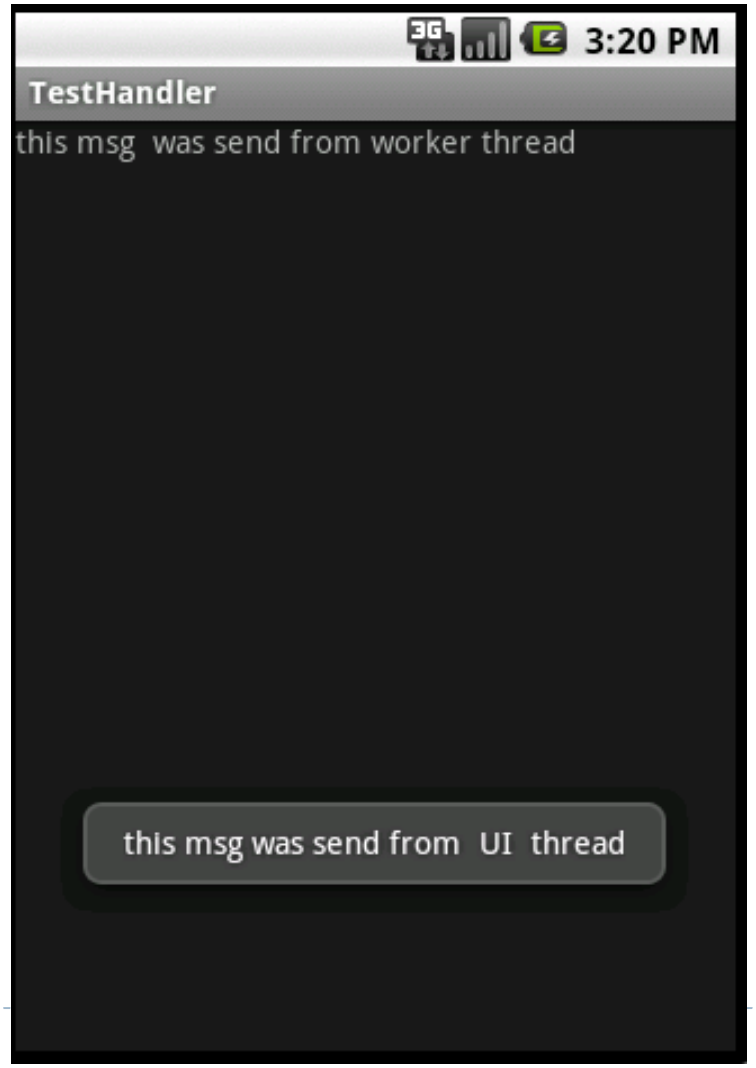
Android线程模型

主线程通过子线程的handler向其发送消息

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Worker w = new Worker();
    w.start();

    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }


    Message msg = Message.obtain();
    msg.obj="hello,i'm UI thread";

    w.getWorkerHandler().sendMessage(msg );

}
```

Android线程模型

运行结果



Services简介

徐涛

认识Service

- 1、后台运行，不可交互
- 2、需要通过某一个Activity或其他Context对象来启动：
  context.startService()或constext.bindService();
- 3、最好在Service中启动新线程来运行很耗时的任务

为什么不使用后台线程而使用Service

- 1、service可以放在独立的进程中，所以更安全
- 2、使用service可以依赖现有的binder机制，不需要在应用层面上处理线程同步的繁杂工作
- 3. 系统可以重新启动异常死去的service

Service生命周期

- 1、通过startService启动
  - （1）生成->开始（onCreate->onStart）过程
  - （2）Service停止的时候直接进入销毁（onDestroy）过程
  - （3）如果调用者直接退出而没有调用stopService，则会一直在后台运行
- 2、通过bindService
  - （1）只运行onCreate
  - （2）调用者退出，Service则调用onUnbind->onDestroyed停止

实现Service

- 实现onCreate()
- 实现onBind(Intent i)
- 实现onUnbind(Intent i)
- 实现onStart(Intent, int)
- 实现onDestroy()

在AndroidManifest.xml中申明Service

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="cn.coolworks.servicesdemo" android:versionCode="1"
  android:versionName="1.0.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".ServicesDemo" android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <service android:name=".Music">
      <intent-filter>
        <action android:name="com.liangshan.wuyong.START_AUDIO_SERVICE" />
        <category android:name="android.intent.category.default" />
      </intent-filter>
    </service>
  </application>
</manifest>
```

本地Service与远程service

- 本地的service在同一个进程内
- 远程的service在不同的进程内

- 与远程service的通信使用binder

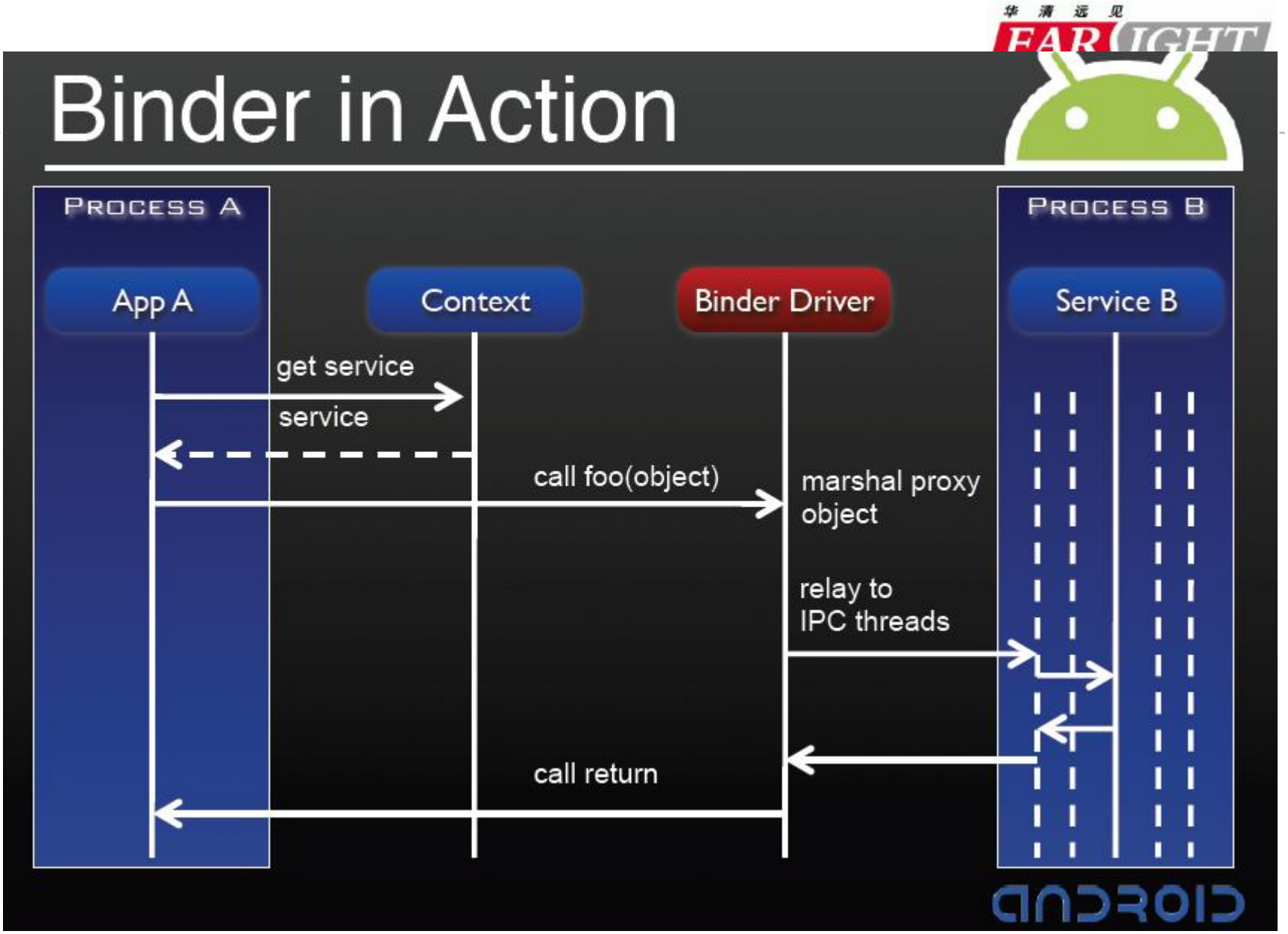
本地Service与远程service 的优劣势对比

- 本地的service在同一个进程内，可以直接函数调用，不用写IPC通讯
- 本地service容易调试
- 本地service 挂断会影响其它组件，如activity，导致整个应用被杀

- 与远程service的通信使用binder
- Service挂掉不会影响activity
- 写IPC机制较复杂

AIDL简介

- Android interface define language


华清远见
FARSIGHT
Binder in Action
PROCESS A
App A
Context
Binder Driver
PROCESS B
Service B
get service
service
call foo(object)
marshal proxy object
relay to IPC threads
call return
android

华清远见
FARSIGHT
嵌入式培训专家

Q&A
谢谢