Android移动开发

Android Mobile Application Development

第10讲 多线程

吴以凡
计算机学院 一教505
yfwu@hdu.edu.cn

Handler机制

Handler的使用场景

两个例子：

- 启动App时，展示了一个开屏广告，默认播放n秒；在n秒后，需跳转到主界面
- 用户在App中，点击下载视频，下载过程中需要弹出Loading弹窗，下载结束后提示用户下载成功/失败

Handler概念

Handler机制为Android系统解决了以下两个问题:

- 调度（Schedule）Android系统在某个时间点执行特定的任务
  - Message (android.os.Message)
  - Runnable (java.lang.Runnable)
- 将需要执行的任务加入到用户创建的线程的任务队列中

From Android Developer Website: There are two main uses for a Handler:
(1) to schedule messages and runnables to be executed at some point in the future;
(2) to enqueue an action to be performed on a different thread than your own.

Handler常用方法

- 立即发送消息

```
public final boolean sendMessage(Message msg)
public final boolean post(Runnable r);
```

- 延时发送消息

```
public final boolean sendMessageDelayed(Message msg, long delayMillis)
public final boolean postDelayed(Runnable r, long delayMillis);
```

- 定时发送消息

```
public boolean sendMessageAtTime(Message msg, long uptimeMillis);
public final boolean postAtTime(Runnable r, long uptimeMillis);
public final boolean postAtTime(Runnable r, Object token, long uptimeMillis);
```

- 取消消息

```
public final void removeCallbacks(Runnable r);
public final void removeMessages(int what);
public final void removeCallbacksAndMessages(Object token);
```

Handler的使用

- 调度Message

  新建一个Handler，实现handleMessage()方法

  在适当的时候给上面的Handler发送消息

- 调度Runnable

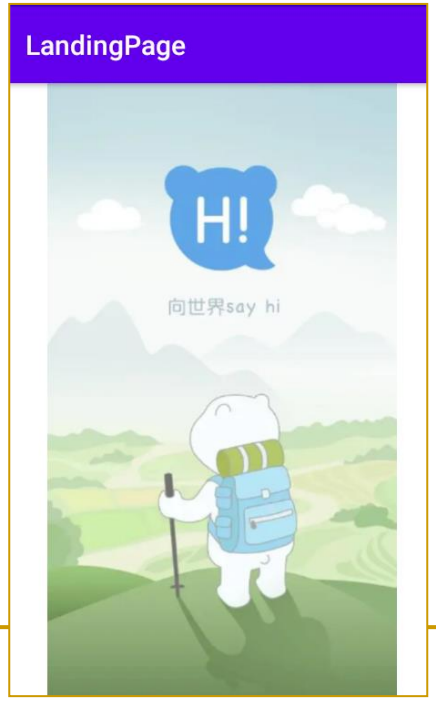
  新建一个Handler，然后直接调度Runnable

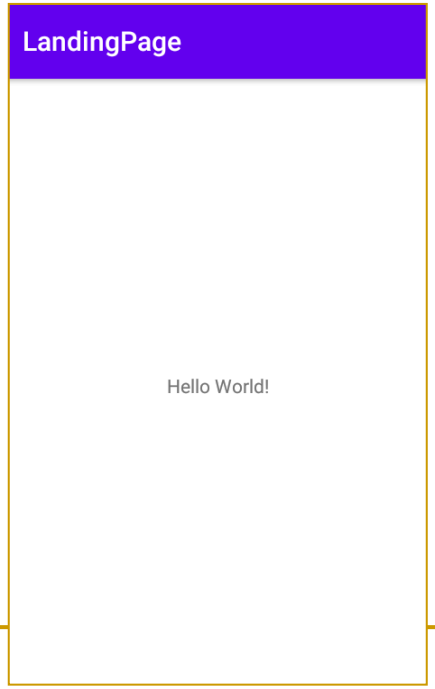
- 取消调度

  通过Handler取消已经发送过的Message/Runnable

Handler实例

- 启动app，展示一个开屏广告（着陆页LandingPage、SplashScreen）
  - 5秒后，自动跳转到主界面
  - 如果用户点击了屏幕或“跳过”按钮，直接进入主界面





Handler实例

```
handler = new Handler();

runnable = new Runnable() {
   public void run() {
      jumpToMainActivity();
   }
};

handler.postDelayed(runnable, 5000);
```

```
public void onClick(View v) {
   handler.removeCallbacks(runnable);

   jumpToMainActivity();
}
```

问题：为什么按返回按钮回到SplashActivity后，自动跳转失效？

Handler实例

- 用户在App中，点击下载视频，下载过程中需要弹出Loading窗，下载结束后提示用户下载成功/失败。

直接在主线程执行下载视频操作？

* 补充知识点: Android中，UI控件并非是线程安全的，Android系统要求对于所有UI控件的调用，必须在主线程。因此，通常我们也把主线程也叫做**UI线程**。

Handler实例

- 用户在App中，点击下载视频，下载过程中需要弹出Loading窗，下载结束后提示用户下载成功/失败。

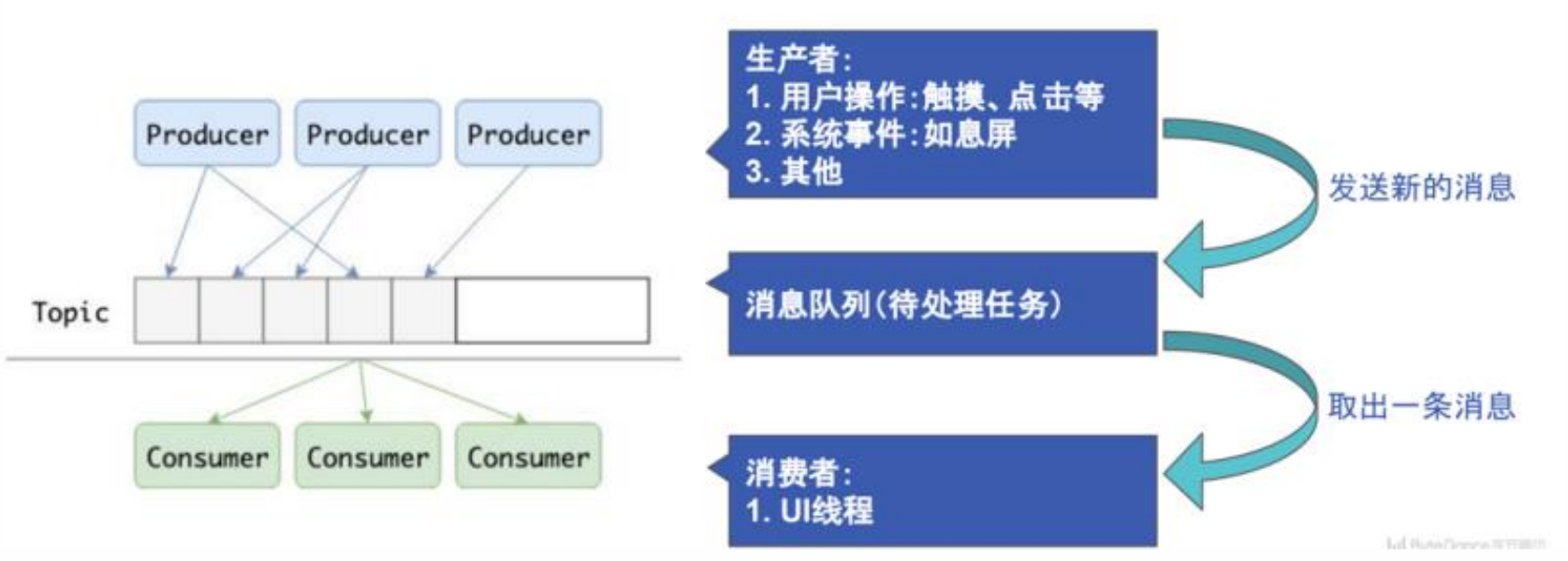
主线程与子线程的通信

```
DownloadThread().start() //开始下载

class DownloadThread : Thread() {
    override fun run() {
        handler.sendMessage(Message.obtain(handler, MSG_START_DOWNLOAD))
        try {
            downloadVideo()
            handler.sendMessage(Message.obtain(handler, MSG_SUCESS_DOWNLOAD))
        }catch (e: Exception){
            e.printStackTrace()
            handler.sendMessage(Message.obtain(handler, MSG_FAIL_DOWNLOAD))
        }

    }
}
```

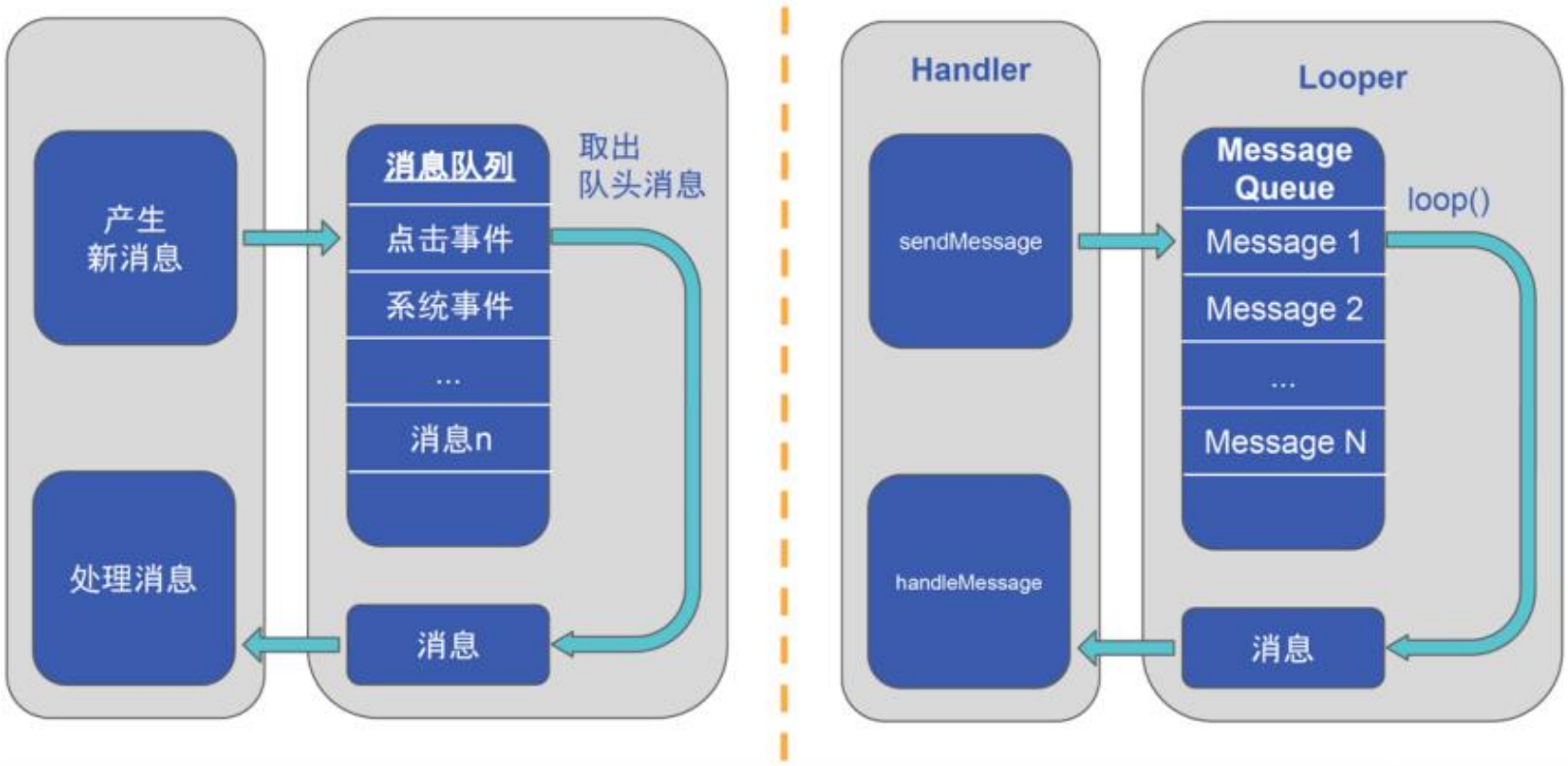
```
companion object{
    const val MSG_START_DOWNLOAD = 0
    const val MSG_SUCCESS_DOWNLOAD = 1
    const val MSG_FAIL_DOWNLOAD = 2
}
val handler = object : Handler() {
    override fun handleMessage(msg: Message) {
        super.handleMessage(msg)
        when(msg.what){
            MSG_START_DOWNLOAD -> {
                toast("开始下载")
                showLoading()
            }
            MSG_SUCCESS_DOWNLOAD -> {
                toast("下载成功")
                hideLoading()
            }
            MSG_FAIL_DOWNLOAD -> {
                toast("下载失败")
                hideLoading()
            }
        }
    }
}
```

Handler原理：UI线程与消息队列机制

- Android中，UI线程负责处理界面的展示，响应用户的操作



Handler原理：UI线程与消息队列机制



Handler原理：UI线程与消息队列机制

- Message 消息

  由MessageQueue统一队列，然后交由Handler处理

- MessageQueue 消息队列

  用来存放Handler发送过来Message，并且按照先入先出的规则执行

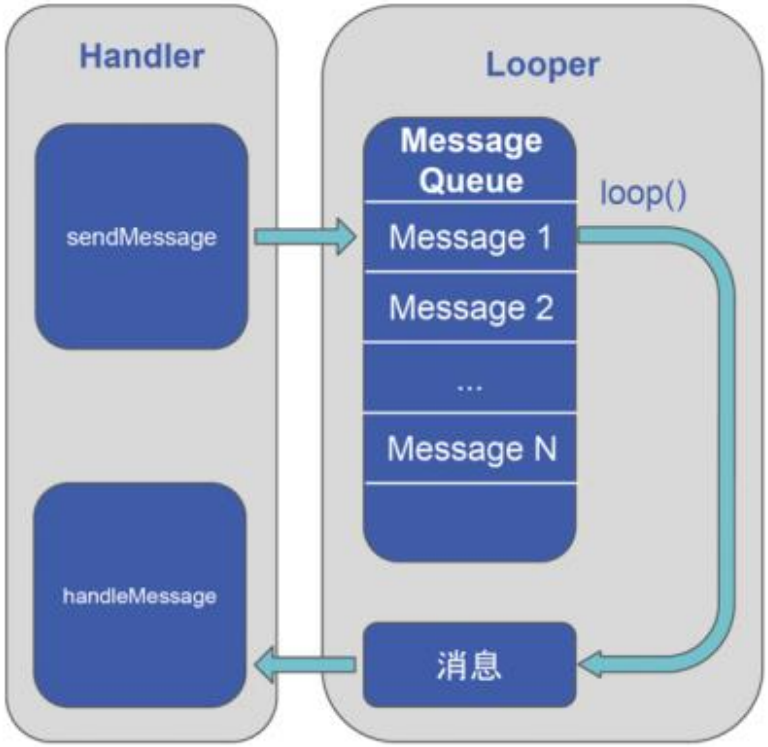
- Handler 处理者

  负责发送和处理Message，每个Message必须有一个对应的Handler

  - 发送：sendMessage()
  - 处理：handleMessage()

- Looper 消息轮询器

  不断的从MessageQueue中抽取Message并执行



辨析Runnable/Message

- Runnable会被打包成Message，所以实际上Runnable也是Message
- 没有明确的界限，取决于使用的方便程度

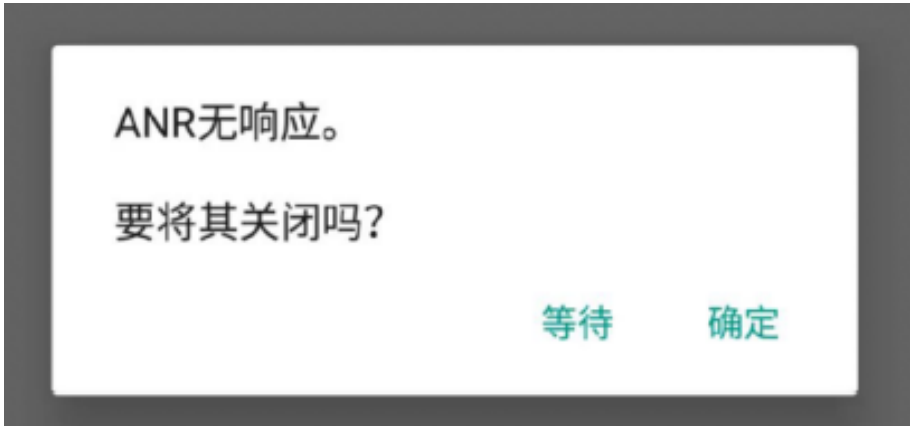
```
val handler = Handler()
val runnable = Runnable {
    //跳转首页
    jumpToMainActivity()
}
handler.postDelayed(runnable, 3000)
```

=

```
val handler = object : Handler() {
    override fun handleMessage(msg: Message) {
        super.handleMessage(msg)
        if (msg.what == MSG_GO_MAIN_ACTIVITY) {
            //跳转首页
            jumpToMainActivity()
        }
    }
}
handler.sendMessageDelayed(Message.obtain(handler, MSG_GO_MAIN_ACTIVITY), 3000)
```

扩展：ANR

- 主线程（UI线程）不能执行耗时操作，否则会出现 ANR (Application Not Responding)



Handler总结

- Handler就是Android中的消息队列机制的一个应用，可理解为是一种生产者消费者的模型，解决了Android中的线程内&线程间的任务调度问题
- Handler的本质就是一个死循环，待处理的Message加到队列里面，Looper负责轮询执行
- 掌握Handler的基本用法：立即/延时/定时发送消息、取消消息

Android多线程

进程与线程

- 进程（Process）是具有一定独立功能的程序关于某个数据集合的一次运行活动。它是操作系统动态执行的基本单元。在传统的操作系统中，进程既是基本的分配（资源）单元，也是基本的执行（调度）单元。
  - 一般情况下，android中的一个app是一个进程，如果需要使用多进程，需要手动开启
- 线程（Thread）是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。

Android多进程

- 一个应用可以创建多个进程 android:process

```
<activity
    android:name=".SecondActivity"
    android:configChanges="screenLayout"
    android:label="@string/app_name"
    android:process=":remote" />
```

- AndroidMantifest.xml中的activity、service、receiver和provider均支持android:process属性

Android多进程

优点

✓ 保护主进程减少崩溃概率

✓ 主进程退出子进程可以继续工作(push进程)

✓ 实现多模块功能与数据解耦

✓ 分散内存占用，降低被系统杀死的概率

缺点

✗ 多进程间通信（RPC ）开发比较困难

* 不要创建过多进程，通常保留最多两到三个进程

Android中的常用线程

- Thread
- ThreadPool
- HandlerThread
- IntentService
- AsyncTask

Thread

- 简单的创建线程的例子

```
Thread thread = new Thread() {
   public void run() {
      // do something
   }
};

thread.start();
```

启动线程

```
Thread thread = new Thread() {
   public void run() {
      while (!isInterrupted()) {
         // do something
      }
   }
};

thread.start();
thread.interrupt();
```

启动和停止线程

Thread

- 设置线程优先级

```
Thread thread = new Thread() {
   public void run() {
      Process.setThreadPriority(Process.THREAD_PRIORITY_DEFAULT);
   }
};

thread.start();
```

最高优先级 -> 最低优先级：-20 -> +19

- THREAD_PRIORITY_DEFAULT: 0
- HREAD_PRIORITY_LOWEST: 19
- THREAD_PRIORITY_MORE_FAVORABLE: -1
- THREAD_PRIORITY_LESS_FAVORABLE: 1
- …

Thread

- Android中创建线程最终调用Linux中pthread库，通过系统调用最终请求内核创建线程
- 尽管我们创建线程数量没有限制，但线程数量过多会导致性能问题
  - CPU 实际上只能并行处理少量线程；一旦超限便会遇到优先级和调度问题
  - 每个线程至少需要占用64k 内存，线程数目过多容易引发OOM（java.lang.OutOfMemoryError）等问题
  - 每个线程占用一个fd，安卓中一旦fd超限制，也会引发OOM的问题

ThreadPool

- 接口Java.util.concurrent.ExecutorService 表述了异步执行的机制，并且可以让任务在一组线程内执行。
- 重要函数
  - execute(Runnable)
  - submit(Runnable): 有返回值（Future），可以cancel，更方便进行错误处理
  - shutdown()

ThreadPool

- 为什么要使用线程池？
  - 线程的创建和销毁的开销都比较大，降低资源消耗
  - 线程是可复用的，提高响应速度
  - 对多任务多线程进行管理，提高线程的可管理性

HandlerThread

- HandlerThread的本质：继承Thread类 & 封装Handler类

试想一款股票交易App：

- 由于因为股票的行情数据都是实时变化的
- app需要每隔一定时间向服务器请求行情数据

这个轮询的请求的调度应该放到非主线程，可由Handler + Looper去处理和调度

HandlerThread

- 在子线程中使用Handler：
  1. 先在thread的run方法里面调用 Looper.prepared 方式去初始化Looper
  2. 然后再创建一个Handler
  3. 主动调用 Looper.loop() 通知整个队列循环开始进行

```
Thread thread = new Thread() {
        public void run() {
            Looper.prepare();
            Handler handler = new Handler(); // 子线程中的handler
            Looper.loop();
        }
    };
```

- 主线程中默认可以直接使用Handler的原因：主线程ActivityThread被创建时就会初始化Looper

HandlerThread

- HandlerThread：带有Handler消息机制的线程
  - 通过延时消息完成轮询操作

```
class StockHandlerThread extends HandlerThread implements Handler.Callback {
    private Handler handler;
    public StockHandlerThread() {
        super("stock");
    }
    @Override
    protected void onLooperPrepared() {
        super.onLooperPrepared();
        handler = new Handler(getLooper(), this);
        handler.sendEmptyMessage(MSG_QUERY);
    }
    @Override
    public boolean handleMessage(@NonNull Message msg) {
        if (msg.what == MSG_QUERY) {
            handler.sendEmptyMessageDelayed(MSG_QUERY, 10 *1000);
        }
        return false;
    }
}
```

IntentService

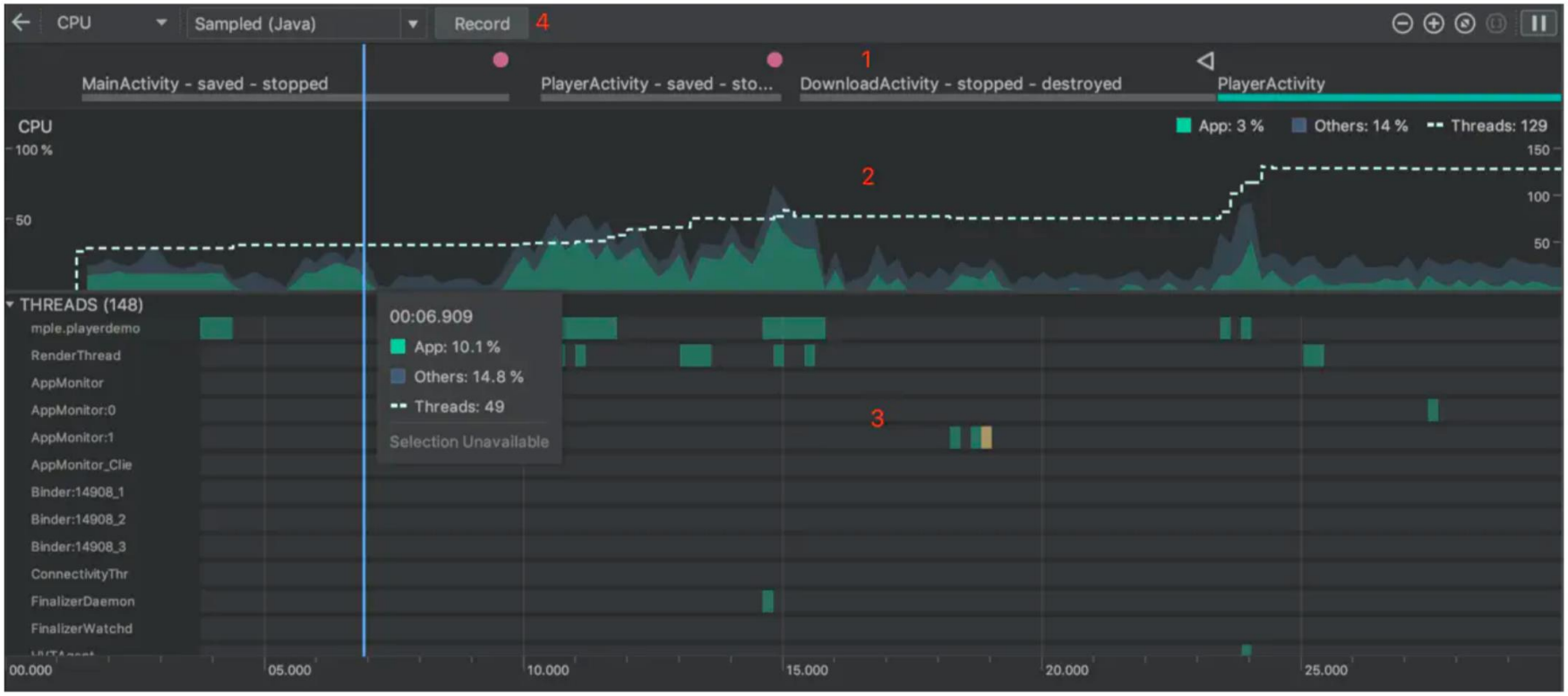
- Service
  - Service 是一个可以在后台执行长
  - 时间运行操作而不提供用户界面的应用组件
- 常见Service
  - 音乐播放
  - Push

IntentService

- Service是执行在主线程的。如果不希望在主线程执行某些操作任务，就应该使用IntentService
  - 比如：用Service下载文件
- IntentService 是 Service 的子类，它使用工作线程逐一处理所有启动请求。如果您不要求服务同时处理多个请求，这是最好的选择。

线程状态

- Profiler
  - View > Tool Windows > Profiler



AsyncTask异步类

- Android的UI线程主要负责处理用户的按键事件、触屏事件等，其他阻赛UI线程的操作不应该在主线程中操作。
- Android程序将耗时操作放在新线程中完成，但新线程可能需要动态更新UI组件，比如将从网络获取的图片用于UI。但新线程无法直接更新UI组件。
  - 解决方案：使用异步任务（AsyncTask）实现异步线程的操作

AsyncTask异步类

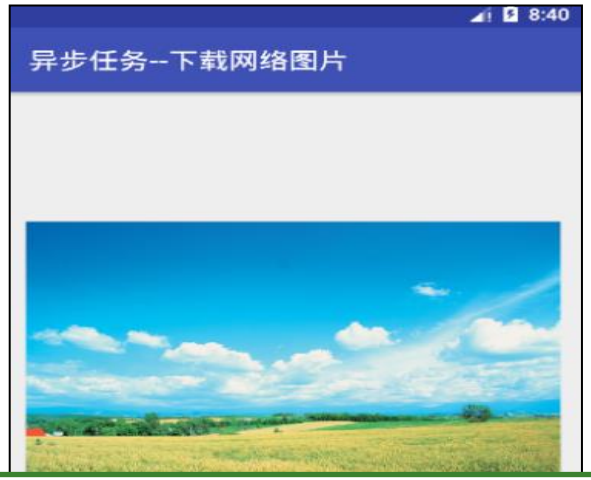
- AsyncTask<Params,Progress,Result>抽象类，在被继承时需要指定如下三个泛型参数：
  - Params：启动任务执行的输入参数的类型
  - Progress：后台任务完成的进度值的类型
  - Result：后台任务执行完成以后返回结果的类型

AsyncTask异步类

- 使用AsyncTask步骤：
  - 创建AsyncTask的子类，并指定参数类型。如果某个参数不需要，则指定为Void类型
  - 实现AsyncTask的方法
    - onPreExecute(): 启动异步任务时，先执行的方法
    - doInBackground(Params…): 启动子线程执行的方法，也即后台线程将要完成的功能，如：获取网络资源等耗时性的操作
    - onPostExecute(Result result): doInBackground()方法执行完以后，系统会自动调用onPostExecute()方法，并接受其返回值，一般负责更新UI线程等操作
  - 调用AsyncTask子类的实例的execute(Params… params)方法执行耗时操作

AsyncTask类实现网络下载图片

```
public class MainActivity extends AppCompatActivity {
   private ImageView mImageView = null;
   private ProgressBar mProgressBar = null;
   private String URLs="http://wallcoo.com/nature/iclickart_8_1024/
wallpapers/1280x1024/iclickart_nature_wallpaper_122414a.jpg";
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      //实例化控件
     this.mImageView = (ImageView)
                          findViewById(R.id.imageView1);
     this.mProgressBar = (ProgressBar)
                          findViewById(R.id.progressBar1);
      //实例化异步任务
      ImageDownloadTask task = new ImageDownloadTask();
      //执行异步任务
      task.execute(URLs);
   }
```




```
<ImageView
    android:id="@+id/imageView1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    />
  <ProgressBar
    android:id="@+id/progressBar1"
    android:visibility="gone"
    style="?android:attr/progressBarStyleLarge"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true" />
```

AsyncTask类实现网络下载图片

```
class ImageDownloadTask extends AsyncTask<String,Void, Bitmap> {
   @Override
   protected Bitmap doInBackground(String... params) {
      Bitmap bitmap = null;       // 待返回的结果
      String url = params[0];  // 获取URL
   URLConnection connection;    // 网络连接对象
      InputStream is;       // 数据输入流
      try {
         connection = new URL(url).openConnection();
          is = connection.getInputStream();   //获取输入流
          BufferedInputStream buf = new BufferedInputStream(is);
          // 解析输入流
          bitmap = BitmapFactory.decodeStream(buf);
          is.close();
          buf.close();
       }  catch (MalformedURLException e) {
            e.printStackTrace();
       }  catch (IOException e) {
            e.printStackTrace();
       }
       // 返回给后面调用的方法
       return bitmap;
   }
```

```
      @Override
      protected void onPreExecute() {
         //显示等待圆环
         mProgressBar.setVisibility(View.VISIBLE);
      }
      @Override
      protected void onPostExecute(Bitmap result) {
         //下载完毕，隐藏等待圆环
         mProgressBar.setVisibility(View.GONE);
         mImageView.setImageBitmap(result);
      }
}
```

AsyncTask异步类

- AsyncTask<Params,Progress,Result>抽象类，在被继承时需要指定如下三个泛型参数：

```
class ImageDownloadTask extends AsyncTask<String, Void, Bitmap>
```

  - Params：启动任务执行的输入参数的类型
    - String:

```
task.execute(URLs);
```

  - Progress：后台任务完成的进度值的类型
    - Void
  - Result：后台任务执行完成以后返回结果的类型
    - Bitmap:

```
@Override
protected Bitmap doInBackground(String... params)
```