进程调用 与 多线程编程

主要内容

- 进程调用
  - 调用 Activity
  - Content Provider　ContentResolver **访问其他应用程序中的数据**
  - Broadcast **广播**
  - Service 服务调用 AIDL
- 多线程编程
  - Handler 机制
  - Android多线程

主要内容

- 进程调用
  - 调用 Activity
    - **同一个进程访问，需要指定**Context**对象和**Activity**的**Class**对象**

```
Intent intent = new Intent(this, SecActivity.class);
startActivity(intent);
```

    - **跨进程访问，需要访问的**Activity**所对应的**Action,以及Uri。

```
Intent callIntent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:10086"));
startActivity(callIntent);
```

在AndroidManifest.xml**文件中使用标签在**android:name**属性中指定**Action

invoke 与 otherProcess 示例

- invoke 是调用者，可以调用另外APP中的activity
- otherProcess 是被调用者， 它有两个activity, 都可以被调用，需要设置intent-filter

示例工程

- invoke--other1.mp4
- invoke--other2.mp4

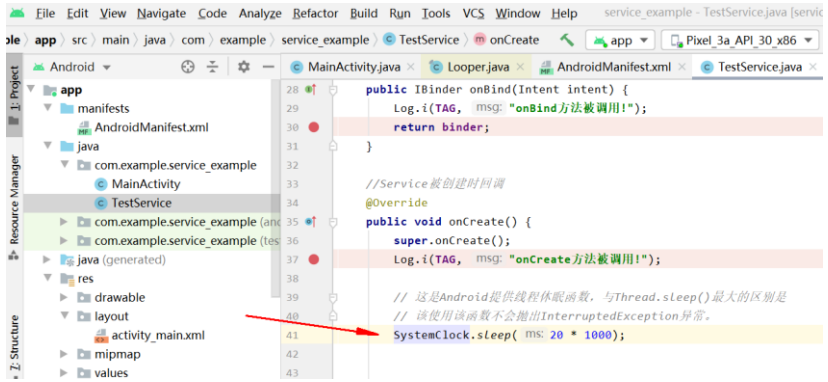
invoke 与 OtherProcess 示例

- invoke 是调用者，可以调用另外APP中的activity
- otherProcess 是被调用者， 它有两个activity, 都可以被调用， 需要设置intentfilter
- getIntent
- startActivityForResult

主要内容

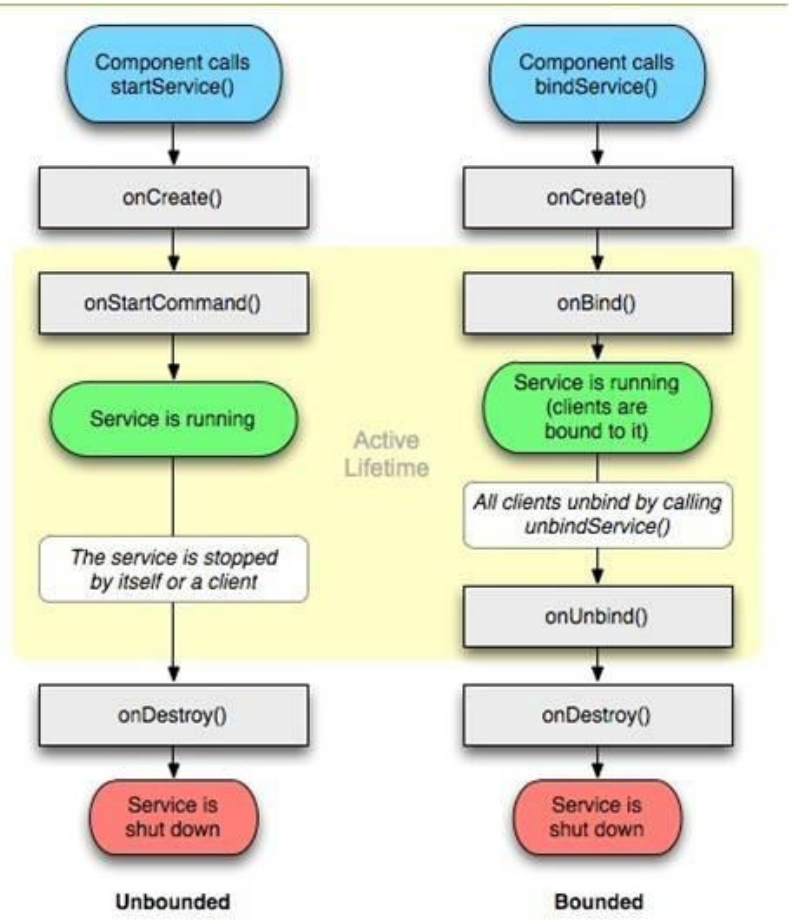
- 进程调用
  - 调用 Activity
  - Content Provider　ContentResolver **访问其他应用程序中的数据**
  - Broadcast **广播**
    - sendBroadcast receiveBroadcast
  - Service 服务调用 AIDL
    - Service与Activity在同一个进程
    - Service与Activity跨进程

Service与Activity



- 后台费时的操作。
- 同一线程，ANR现象。

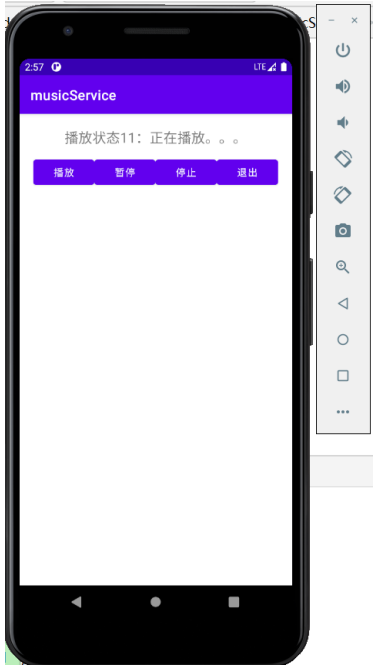
- Service通过startservice启动后，它就独立于调用者而运行。
- Bindservice调用后，可以通过 onBind(Intent) 这个方法返回了一个实现了 IBinder 接口的对象。



- 学习超时自动拨号的demo
- 学习播放音乐的demo

两个Demo运行示例

service-超时后自动拨打电话.mp4



service--音乐播放器.mp4

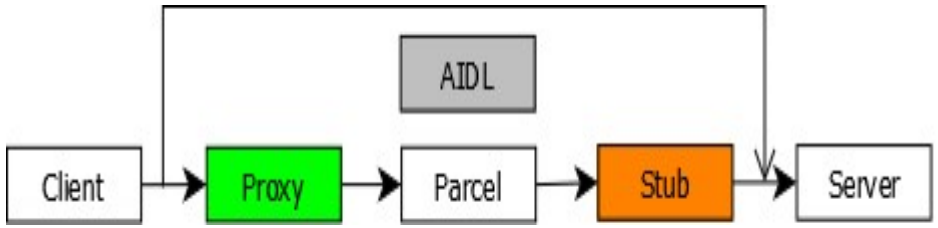
startActivity 与 Binder 进程通信在安全性的核心差异

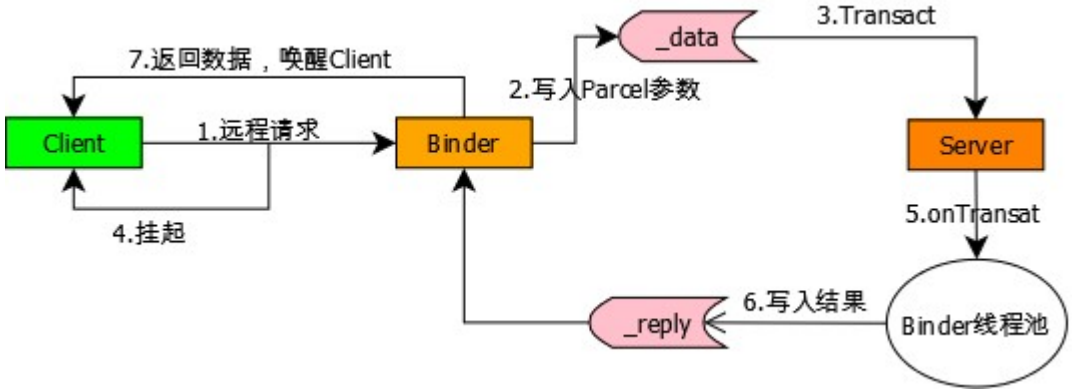
- 1. 身份验证与权限控制机制
  - startActivity：
    - 依赖 Intent 显式/隐式调用规则 与 组件声明权限（如 android:permission 属性）。若目标 Activity 未限制 exported 属性或未声明权限，可能被恶意应用劫持隐式 Intent，导致界面跳转或数据泄露。
    - 示例风险：未受保护的隐式 Intent 可能被其他应用注册相同 Action 的 Activity 截获，窃取敏感数据。
  - Binder：
    - 系统级 UID/PID 验证：Binder 驱动在内核层校验通信双方进程的 UID 和 PID，确保仅授权进程可访问服务。
    - 接口级权限声明：服务端需通过 <permission> 标签或 checkCallingPermission() 显式声明访问权限，客户端需在 AndroidManifest.xml 中声明对应权限才可调用。

- 2. 数据传输安全性
  - startActivity：
    - 通过 Intent 传递的 Bundle 数据可能被恶意应用通过 组件劫持 或 日志截获（如未加密的 Intent 内容）窃取。
    - 局限性：无法直接验证数据接收方的合法性，需依赖开发者主动加密敏感数据（如 Parcelable 加密实现）。
  - Binder：
    - 内核驱动管控：数据通过 Binder 驱动在内核共享内存中传输，用户态进程无法直接读写，避免中间人攻击。
    - 序列化安全：Parcel 支持加密传输（如实现 Parcelable 接口时封装加密逻辑），且通信通道由内核隔离，降低数据泄露风险。

Service与Activity跨进程通信

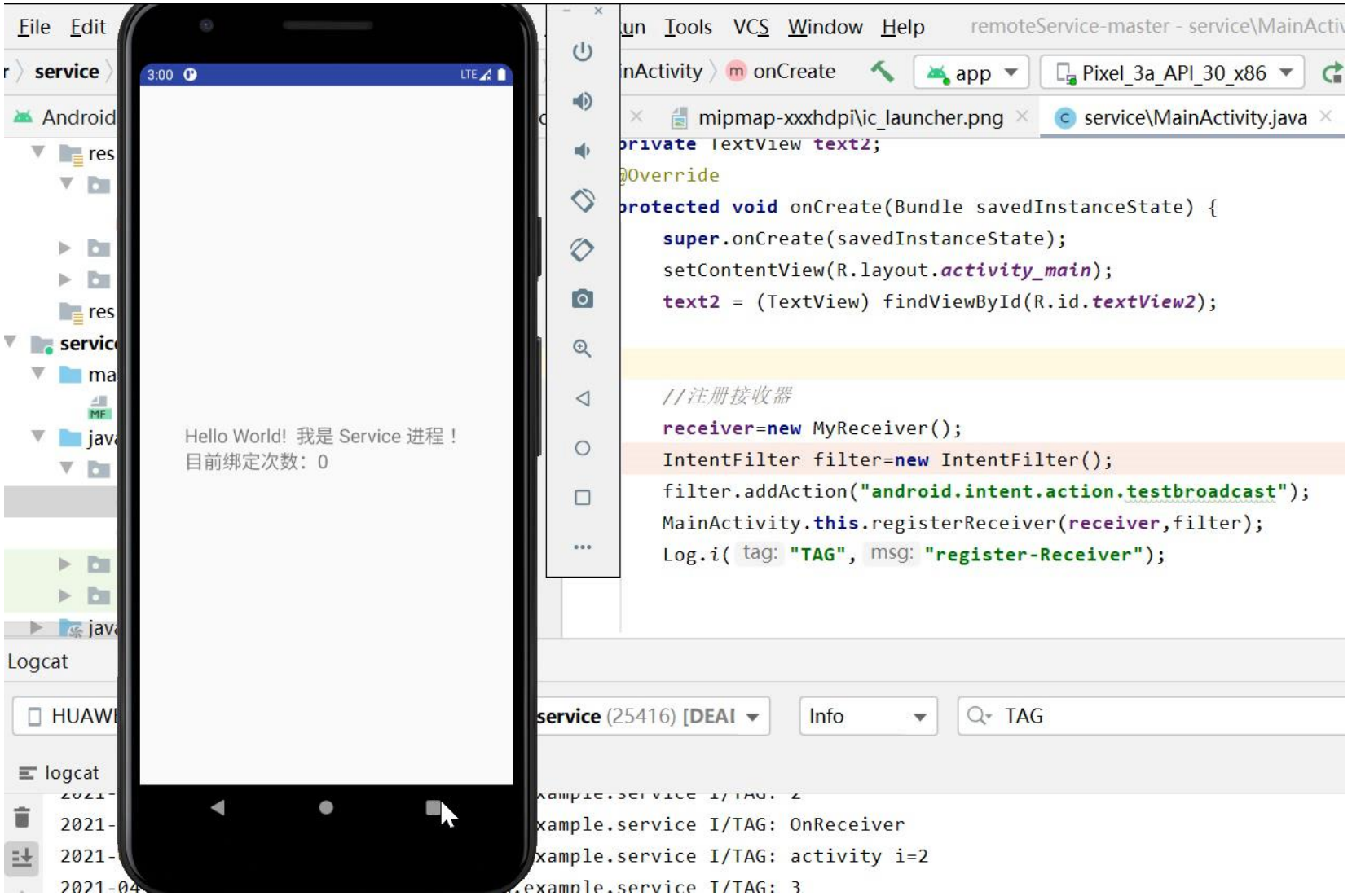
- 有什么方法？
- **AIDL: Android Interface Definition Language,即Android接口定义语言。**





- 学习跨进程Demo
  - app与service通信
  - 另： Demo中service这一侧MyService与MainActivity利用Broadcast通信

跨进程通信运行截屏


File Edit
service
Android
res
res
java
java
3:00
Hello World! 我是 Service 进程！
目前绑定次数：0
un Tools VCS Window Help
remoteService-master - service\MainActi
nActivity
onCreate
app
Pixel_3a_API_30_x86
mipmap-xxxhdpi\ic_launcher.png
service\MainActivity.java
rivate TextView text2;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
text2 = (TextView) findViewById(R.id.textView2);
//注册接收器
receiver=new MyReceiver();
IntentFilter filter=new IntentFilter();
filter.addAction("android.intent.action.testbroadcast");
MainActivity.this.registerReceiver(receiver,filter);
Log.i(tag: "TAG", msg: "register-Receiver");
Logcat
HUAWE
service (25416) [DEAI
Info
TAG
logcat
2021-
xample.service I/TAG: OnReceiver
2021-
xample.service I/TAG: activity i=2
2021-04
example.service I/TAG: 3

跨进程通信运行代码

- service--跨进程调用以及broadcast.mp4

跨进程通信为何推荐使用 AIDL?

- 1. AIDL 的核心作用
  - AIDL 的核心价值在于自动化生成 IPC 通信代码，通过定义接口描述语言，自动生成 Stub（服务端）和 Proxy（客户端）类，开发者无需手动处理以下复杂逻辑：
  - 数据序列化与反序列化：跨进程数据需转为字节流，AIDL 自动实现 Parcelable 或 Serializable 的封装与解析。
  - Binder 事务管理：包括线程调度、事务队列、异常处理等底层细节。
  - 跨进程对象传递：支持接口回调（如 oneway 异步调用）和复杂对象传递（需显式声明 in、out、inout 参数方向）。
- 2. AIDL 的不可替代性场景
  - 复杂对象传输：传递自定义对象（如 User 类）需显式声明字段序列化规则。
  - 高性能需求：AIDL 的 Binder 机制比 Messenger 或 ContentProvider 更高效，适合高频调用（如实时数据传输）。
  - 跨应用服务共享：第三方应用调用服务时，需通过 AIDL 接口明确权限和数据类型约束。
- 3. 替代方案的适用边界
  - 单向数据传输：简单数据传递（如通知状态）可用 Intent、Broadcast 或 Messenger。
  - 同一进程内通信：组件间直接通过 Binder 引用调用方法，无需序列化开销。
  - 极简接口：仅需单个方法调用时，手动实现 Binder 可能更轻量（但需权衡维护成本）。

- AIDL 并非强制要求，但它是 Android 官方为降低跨进程开发复杂度设计的核心工具，尤其在需要高效、稳定、可扩展的 IPC 场景中不可或缺。

作业

- 编程实现一个音乐播放器app，除了播放、暂停等功能，还可以显示播放进度。

思考探索

- 站在一个系统设计者的角度， 你如何提供进程通信的机制。

主要内容

- 进程调用
- 多线程编程
  - Handler 机制
  - Android多线程

Handler 机制 （Android的消息队列机制）

Handler的使用场景

先看这样两个例子：

1.启动今日头条app的时候，展示了一个开屏广告，默认播放x秒；在x秒后，需跳转到主界面。

2.用户在抖音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:

1. 调度（Schedule）Android系统在某个时间点执行特定的任务
    a. Message(android.os.Message)
    b. Runnable(java.lang.Runnable)
2. 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即可
- 取消调度
  - ✓ 通过Handler取消已经发送过的Message/Runnable

Handler的使用举例

启动今日头条app的时候，展示了一个开屏广告，默认播放3秒；在3秒后，需跳转到主界面。

```
Handler handler = new Handler();
Runnable runnable = new Runnable() {
    @Override
    public void run() {
        // 跳转首页
        jumpToMainActivity();
    }
};
handler.postDelayed(runnable, delayMillis: 3000);
```

Handler的使用举例

启动今日头条app的时候，展示了一个开屏广告，默认播放3秒；在3秒后，需跳转到主界面。**如果用户点击了跳过，则应该直接进入主界面。**

```
final Handler handler = new Handler();
final Runnable runnable = new Runnable() {
    @Override
    public void run() {
        // 跳转首页
        jumpToMainActivity();
    }
};
handler.postDelayed(runnable, delayMillis: 3000);
mSkipButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        handler.removeCallbacks(runnable);
        jumpToMainActivity();
    }
});
```

Handler的使用举例

用户在抖音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。

补充知识点:
Android中，UI控件并非是线程安全的，只能在主线程内调用，所以所有对于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的使用举例

用户在抖音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。

```
    View mDownLoadButton = findViewById(R.id.mSkipButton);
    mDownLoadButton.setOnClickListener((v) -> {
            new DownLoadThread(mVideoId).start();
    });
}
private class DownLoadThread extends Thread {
    String mVideoId;
    public DownLoadThread(String videoId) { this.mVideoId = videoId; }
    @Override
    public void run() {
        mHandler.sendMessage(Message.obtain(mHandler, MSG_START_DOWNLOAD));
        try {
            String videoPath = downloadVideo(mVideoId);
            mHandler.sendMessage(Message.obtain(mHandler, MSG_DOWNLOAD_SUCCESS, videoPath));
        } catch (Exception e) {
            mHandler.sendMessage(Message.obtain(mHandler, MSG_DOWNLOAD_FAIL));
        }
    }
    private String downloadVideo(String videoId) {}
}
```

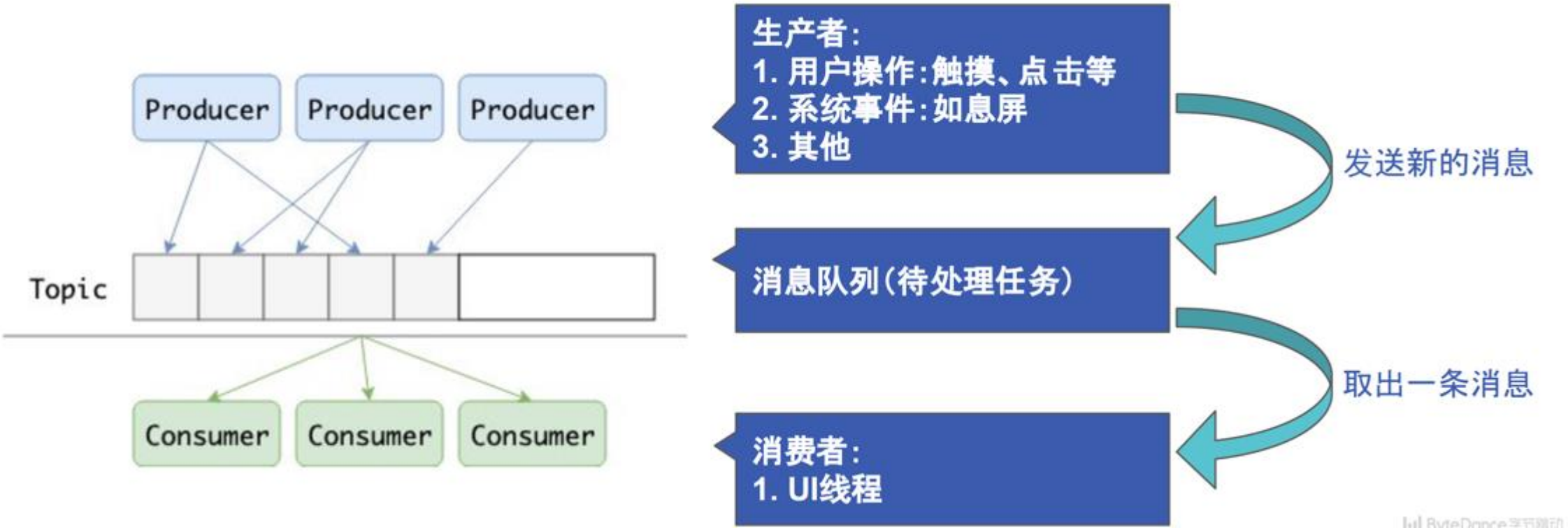
Handler的使用举例

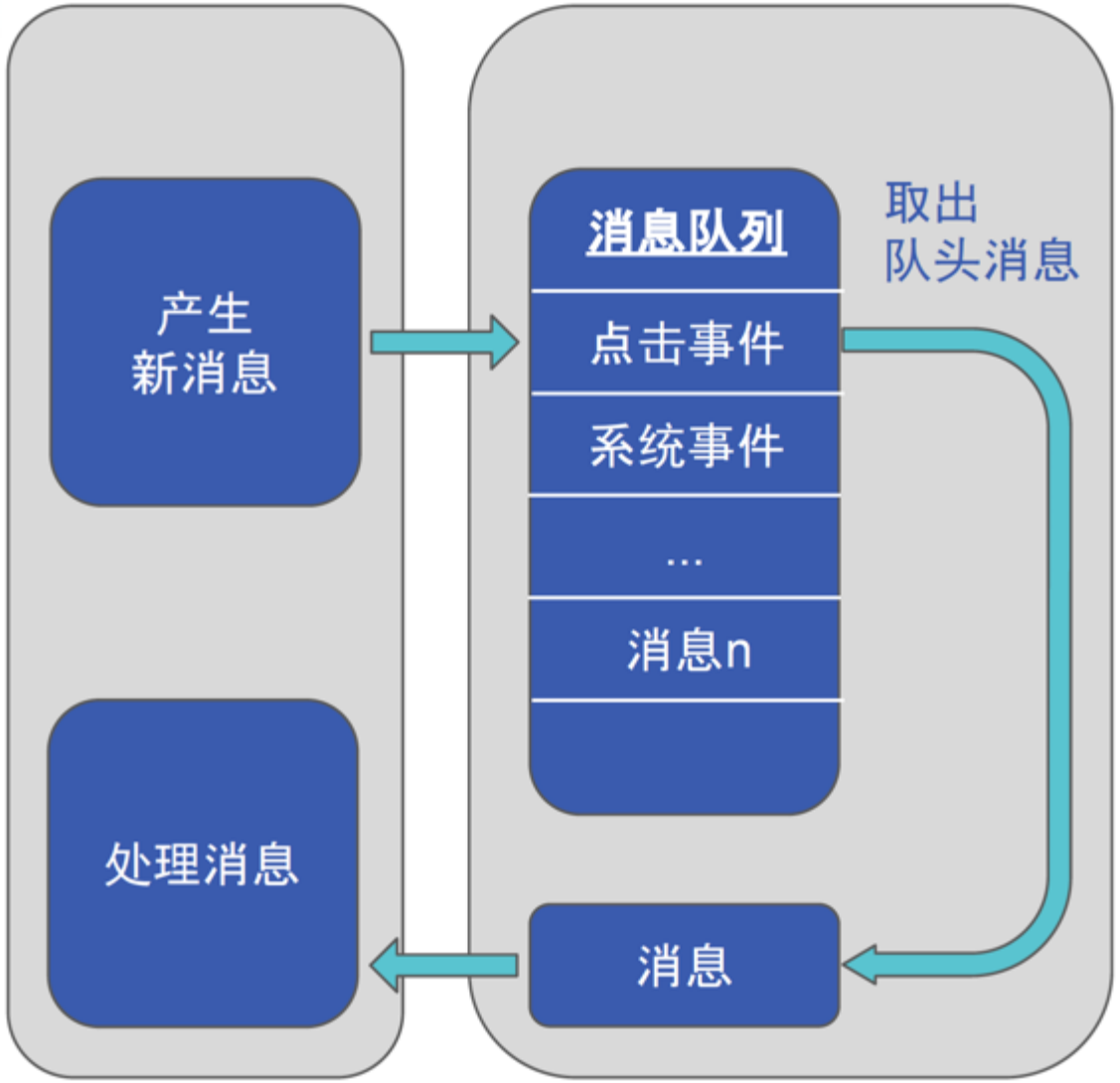
用户在抖音App中，点击下载视频，下载过程中需要弹出Loading窗，下载结束后提示用户下载成功/失败。

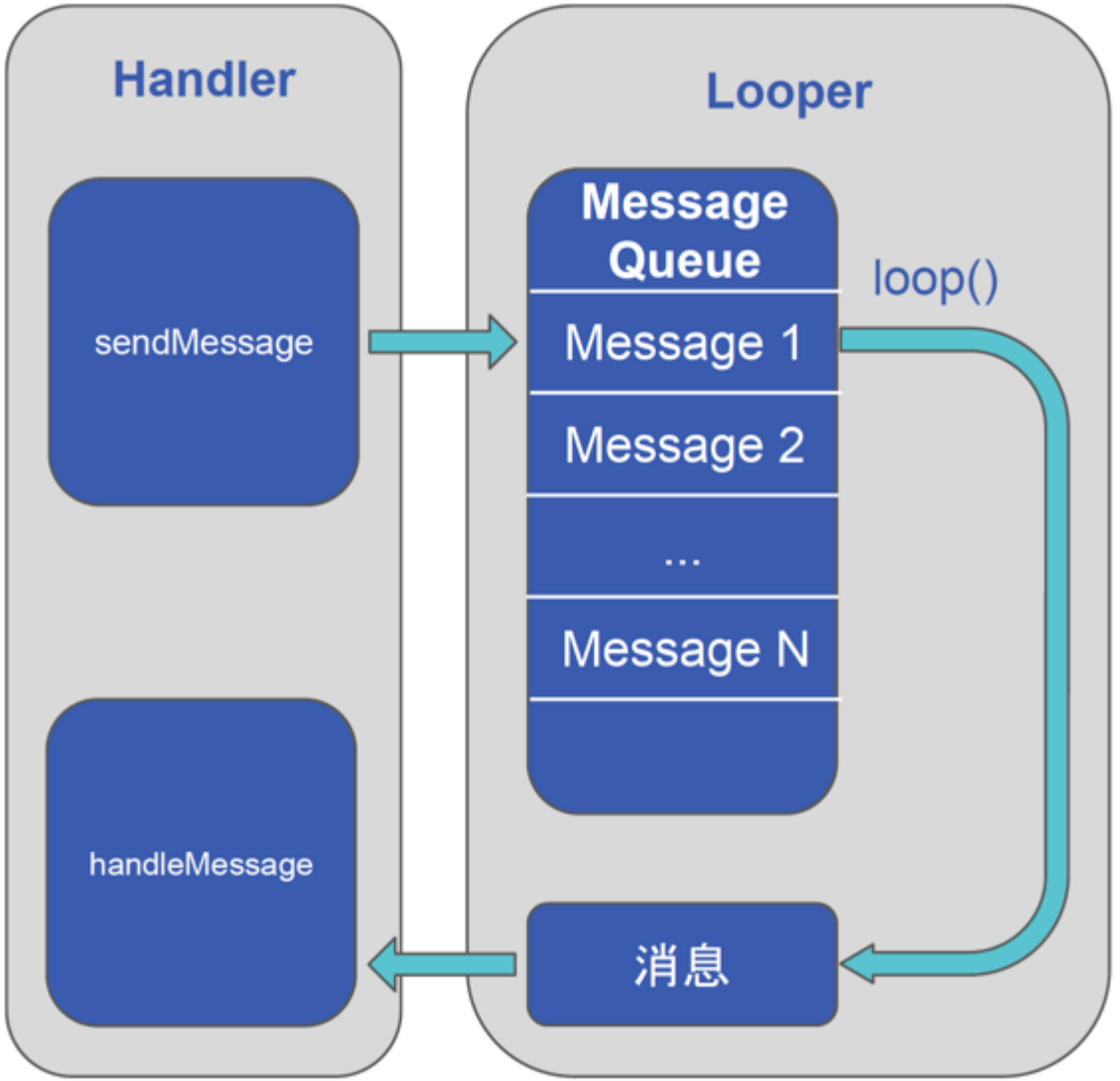
```
private static final int MSG_START_DOWNLOAD = 1;
private static final int MSG_DOWNLOAD_SUCCESS = 2;
private static final int MSG_DOWNLOAD_FAIL = 3;
private Handler mHandler = new Handler() {
    @Override
    public void handleMessage(@NonNull Message msg) {
        super.handleMessage(msg);
        switch (msg.what) {
            case MSG_START_DOWNLOAD:
                toast( msg: "开始下载");
                showLoading();
                break;
            case MSG_DOWNLOAD_SUCCESS:
                toast( msg: "下载成功");
                hideLoading();
                break;
            case MSG_DOWNLOAD_FAIL:
                toast( msg: "下载失败");
                hideLoading();
                break;
            default:
                break;
        }
    }
};
```

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：




产生
新消息
处理消息
消息队列
点击事件
系统事件
...
消息n
消息
取出
队头消息

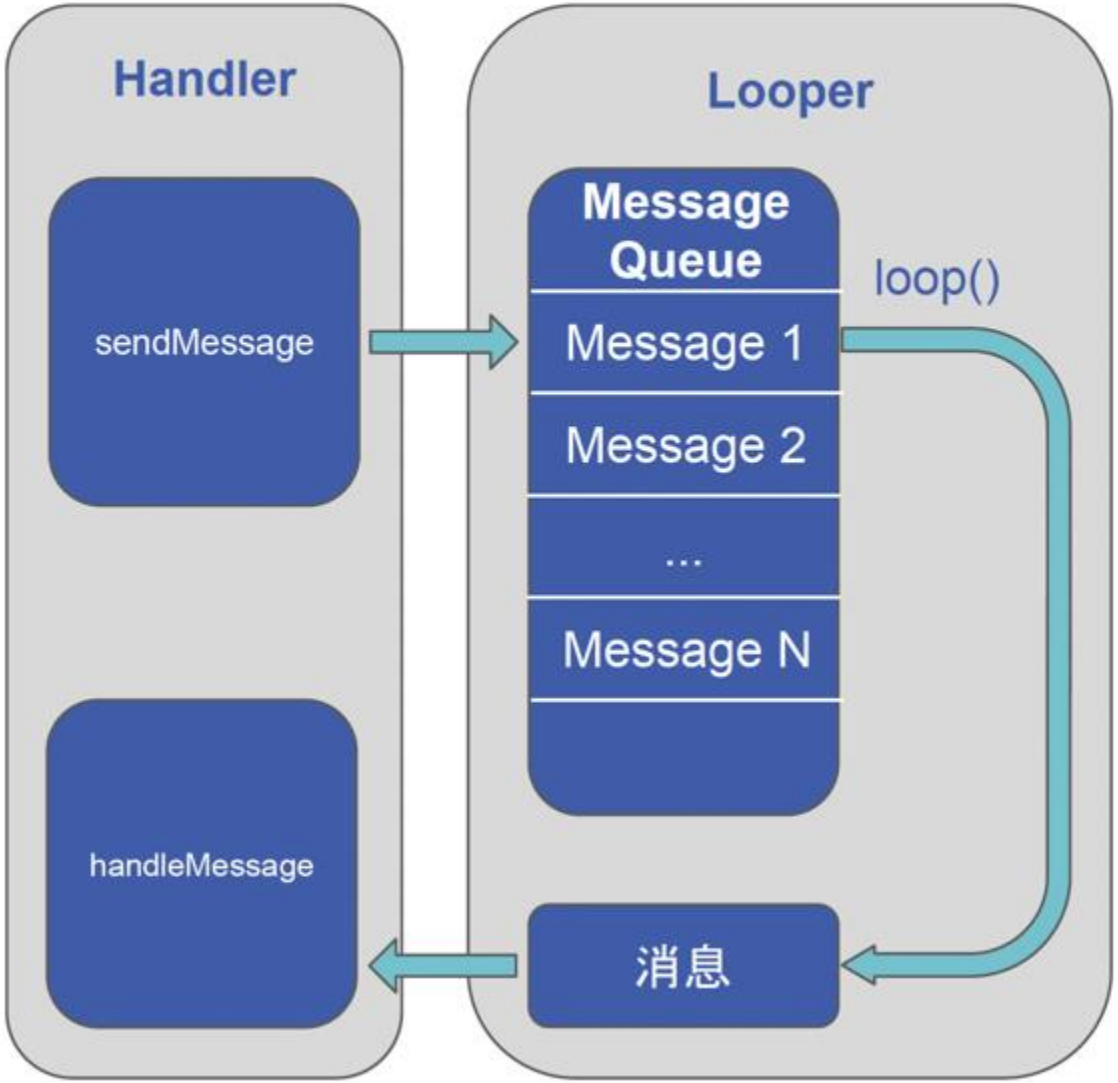

Handler
sendMessage
handleMessage
Looper
Message
Queue
Message 1
Message 2
...
Message N
消息
loop()

- 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

- Looper：
  - ➢ 消息轮询器，不断的从MessageQueue中抽取Message并执行。



辨析Runnable/Message

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

```
Handler handler = new Handler();
Runnable runnable = new Runnable() {
    @Override
    public void run() {
        // 跳转首页
        jumpToMainActivity();
    }
};
handler.postDelayed(runnable, delayMillis: 3000);
```

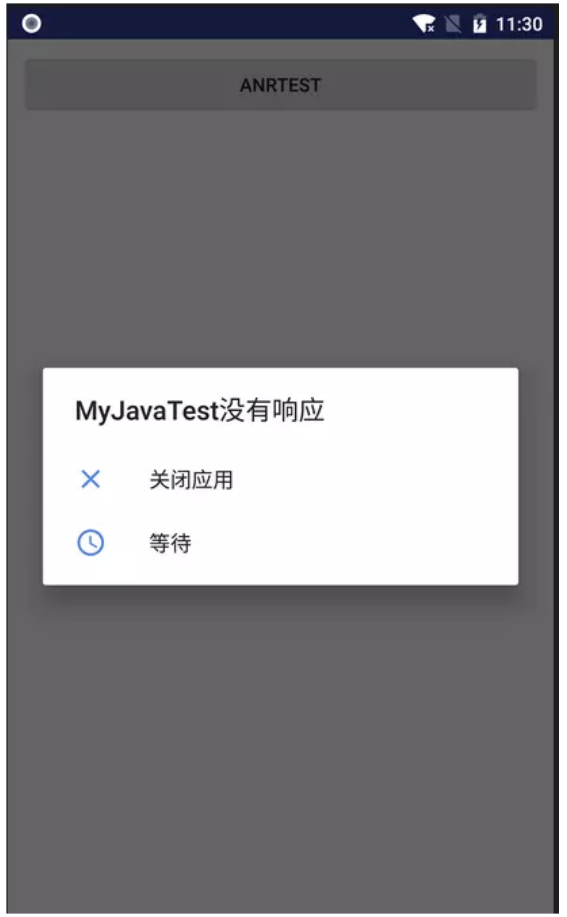
=

```
Handler handler = new Handler() {
    @Override
    public void handleMessage(@NonNull Message msg) {
        super.handleMessage(msg);
        if (msg.what == MSG_GO_MAIN_ACTIVITY) {
            // 跳转首页
            jumpToMainActivity();
        }
    }
};
handler.sendMessageDelayed(
        Message.obtain(handler, MSG_GO_MAIN_ACTIVITY),
        delayMillis: 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)

- ANR : Application Not Responding，程序无响应。
- 界面卡顿
- 原因：UI线程执行单个任务时间过长。
- UI线程中不要执行耗时操作。



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.SendMessage将待处理的Message加到队列里面，Looper负责队列管理，HandleMessage负责处理消息。
- ✓掌握Handler的基本用法：立即/延时/定时发送消息、取消消息；

主要内容

- 进程调用
- 多线程编程
  - Handler 机制
  - Android多线程

进程与线程

进程（Process）是一个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，需要设置AndroidManifest

```
<activity
  android:name=".SecondActivity"
  android:process=":remote" />
```

```
C:\Users\Administrator\AppData\Local\Android\Sdk\platform-tools>adb    -s  00008af5af904ee0  shell
PD2046:/ $
PD2046:/ $
PD2046:/ $  getprop | grep heap
[dalvik.vm.heapgrowthlimit]: [256m]
[dalvik.vm.heapmaxfree]: [8m]
[dalvik.vm.heapminfree]: [2m]
[dalvik.vm.heapsize]: [512m]
[dalvik.vm.heapstartsize]: [16m]
[dalvik.vm.heaptargetutilization]: [0.75]
[init.svc.heapprofd]: [stopped]
PD2046:/ $
PD2046:/ $
PD2046:/ $
```

进程与线程

使用多进程的好处、缺点。

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中的常用线程

- ✓Thread
- ✓ThreadPool
- ✓AsyncTask *
- ✓HandlerThread *
- ✓IntentService *
- ✓**RunOnUIthread**

Thread

```
class MyThread extends Thread{
    @Override
    public void run() {
        super.run();
        // do something
    }
}
```

一个简单的Thread的例子

```
Thread thread = new Thread() {
    @Override
    public void run() {
        super.run();
        while(!isInterrupted()){
            // do something
        }
    }
};
thread.start();
thread.interrupt();
```

怎样优雅的启动和停止一个Thread

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(Runnbale): 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?

1. 新建和销毁线程，如此一来会大大降低系统的效率
2. 而线程是可以重用的

如何建立线程池？？

ThreadPool

```
poolarray[count];
init_threadpool(poolfun,count);
{
    for(i :1 to count)
    {
        poolarray[i].thread = createThread(poolfun);
        poolarray[i].idle = 0;
        lock_signal(poolarray[i].signal);
    }
}
poolfun()
{
    while(1)
    {
        wait_signal(Actionfun);
        Actionfun(param);
        lock_signal();
    }
}
```

```
thread get_freeThread(Actionfun,param)
{
    for(i: 1 to count)
    {
        if poolarray[i].idle ==0
        {
        poolarray[i].idle=1
        poolarray[i].param = param;
        unlock_signal(poolarray[i].signal);
        }
    }
}
```

ThreadPool

介绍几种常用的线程池：

➢单个任务处理时间比较短且任务数量很大（多个线程的线程池）：

- 网络库：FixedThreadPool 定长线程池
- DB操作：CachedThreadPool 可缓存线程池

➢执行定时任务（定时线程池）：

- 定时上报性能日志数据： ScheduledThreadPool 定时任务线程池

➢特定单项任务（单线程线程池）：

- 日志写入：SingleThreadPool 只有一个线程的线程池

AsyncTask

回到之前的例子：
用户在抖音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。

Handler模式来实现的异步操作，代码相对臃肿，在多个任务同时执行时，不易对线程进行精确的控制。

Android提供了工具类AsyncTask，它使创建异步任务变得更加简单，不再需要编写任务线程和Handler实例即可完成相同的任务

AsyncTask

AsyncTask的定义及重要函数：

1. AsyncTask<Params, Progress, Result>： UI线程

2. onPreExecute： UI线程

3. doInBackground： 非UI线程

4. publishProgress： 非UI线程

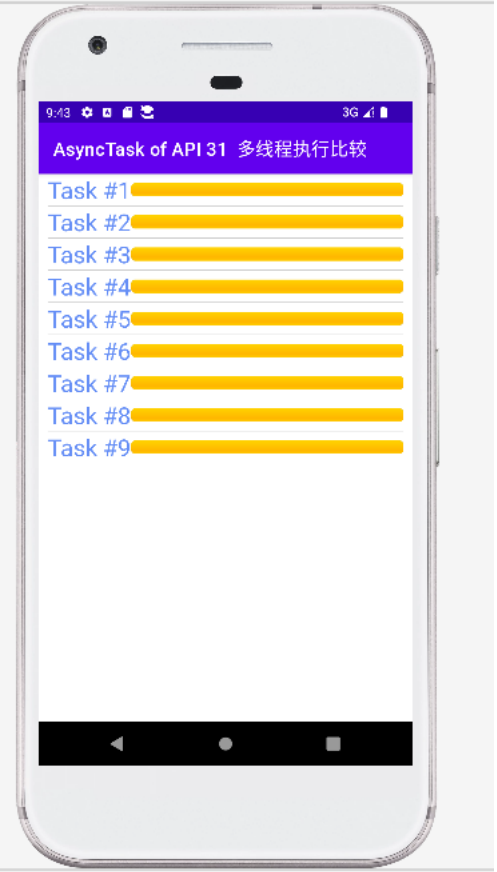
5. onProgressUpdate： UI线程

6. onPostExecute： UI线程

```
private class DownloadTask extends AsyncTask<String, Integer, String> {
    public static final String DOWNLOAD_FAIL = "download_fail";
    @Override protected void onPreExecute() {
        super.onPreExecute();
        toast( msg: "开始下载");
        showLoading();
    }
    @Override protected String doInBackground(String... strings) {
        String url = strings[0];
        try {
            return downloadVideo(url);
        } catch (Exception e) {
            return DOWNLOAD_FAIL;
        }
    }
    private String downloadVideo(String videoId) {
        int progress = 0;
        while (progress < 100) {
            publishProgress(progress);
            progress++;
        }
        return "local_url";
    }
    @Override protected void onProgressUpdate(Integer... values) {
        super.onProgressUpdate(values);
        Log.d( tag: "download",  msg: "下载进度: " + values[0]);
    }
    @Override protected void onPostExecute(String s) {
        super.onPostExecute(s);
        if (DOWNLOAD_FAIL.equals(s)) {
            hideLoading();
            toast( msg: "下载失败");
        } else {
            hideLoading();
            toast( msg: "下载成功: " + s);
```

AsyncTask 与 多线程

- asyncTask 与 线程池.mp4



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。
- 所以我们软件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是否可以放到非主线程，由Handler + Looper去处理和调度？

HandlerThread

```java
public class StockHandlerThread extends HandlerThread implements Handler.Callback {
    public static final int MSG_QUERY_STOCK = 100;
    // 与工作线程相关的Handler
    private Handler mHandler;
    public StockHandlerThread(String name) {
        super(name);
    }
    public StockHandlerThread(String name, int priority) {
        super(name);
    }
    @Override
    protected void onLooperPrepared() {
        mHandler = new Handler(getLooper(), callback: this);
        // 首次请求
        mHandler.sendEmptyMessage(MSG_QUERY_STOCK);
    }
    @Override
    public boolean handleMessage(@NonNull Message msg) {
        if (msg.what == MSG_QUERY_STOCK) {
            // 请求股票数据
            // ...
            // 回调主线程或者写入数据库
            // ...
            // 10s后再次请求
            mHandler.sendEmptyMessageDelayed(MSG_QUERY_STOCK, delayMillis: 10 * 1000);
        }
        return true;
    }
}
```

HandlerThread

源码：

```
public class HandlerThread extends Thread {
    int mPriority;
    int mTid = -1;
    Looper mLooper;
    private @Nullable Handler mHandler;

    public HandlerThread(String name) {...}

    /** Constructs a HandlerThread. ...*/
    public HandlerThread(String name, int priority) {...}

    /** Call back method that can be explicitly overridden if needed to execute some ...*/
    protected void onLooperPrepared() {}

    @Override
    public void run() {...}

    /** This method returns the Looper associated with this thread. If this thread not been started ...*/
    public Looper getLooper() {...}

    /** @return a shared {@link Handler} associated with this thread ...*/
    @NonNull
    public Handler getThreadHandler() {...}

    /** Quits the handler thread's looper. ...*/
    public boolean quit() {...}

    /** Quits the handler thread's looper safely. ...*/
    public boolean quitSafely() {...}

    /** Returns the identifier of this thread. See Process.myTid(). ...*/
    public int getThreadId() { return mTid; }
}
```

IntentService

回顾一下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

那什么是IntentService?

IntentService 是 Service 的子类，它使用工作线程逐一处理所有启动请求。如果不要求服务同时处理多个请求，这是最好的选择。

更通俗的讲：

Service是执行在主线程的。
而很多情况下，我们需要做的事情可能并不希望在主线程执行，那么就应该用IntentService。
比如：用Service下载文件

IntentService

```java
public class DownloadService extends IntentService {
    /**
     * Creates an IntentService.  Invoked by your subclass's constructor.
     *
     * @param name Used to name the worker thread, important only for debugging.
     */
    public DownloadService(String name) {
        super( name: "DownloadService");
    }
    @Override
    protected void onHandleIntent(@Nullable Intent intent) {
        if (intent != null) {
            try {
                String url = intent.getStringExtra( name: "url");
                // download file from url
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

Android中的常用线程

- ✓Thread
- ✓ThreadPool
- ✓AsyncTask *
- ✓HandlerThread *
- ✓IntentService *
- ✓**RunOnUIthread**

RunOnUIthread

```java
@Override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.activity_main);
    String title = "AsyncTask of API " + Build.VERSION.SDK_INT + "  多线程执行比较";
    setTitle(title);
    final ListView taskList = (ListView) findViewById(R.id.task_list);
    taskList.setAdapter(new AsyncTaskAdapter(getApplication(), TASK_COUNT));
    mProgressUI = findViewById(R.id.task_progress2);
    new Thread(new Runnable() {
        @Override
        public void run() {
            //do something takes long time in the work-thread
            while (prog < 101)
            {
                SystemClock.sleep( ms: 100);
                prog++;
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                            mProgressUI.setProgress(prog);
                    }
                });
                if (prog==101)
                    prog=1;
            }
        }
    }).start();
```

Android多线程与进程总结

Thread	多线程的基础
ThreadPool	对线程进行更好的管理
AsyncTask	Android中为了简化多线程的使用， 而设计的默认封装
HandlerThread	开启一个线程，就可以处理多个耗时任务
IntentService	Android中无界面异步操作的默认实现
runOnUiThread	子线程post消息到主线程