组件通信：
消息机制和广播

杨刚
中国人民大学

本章学习目标

- 掌握广播机制的原理和使用方法
- 掌握消息机制的原理和使用方法，包括Message、Handler、Looper类的使用
- 掌握通过消息机制动态获取信息；
- 掌握异步处理工具类：AsyncTask的使用

广播机制

- 利用Intent发送广播消息
  - Broadcast是一种广泛运用在应用程序之间异步传输信息的机制。Android系统通过发出广播消息，来通知各应用组件一些系统事件，
  - 应用程序和Android系统都可以使用Intent发送广播消息
  - 广播消息的内容可以与应用程序密切相关的自定义数据信息，也可以Android的系统信息
    - 网络连接变化
    - 电池电量变化
    - 接收到短信
    - 系统设置变化
  - BroadcastReceiver用于接收并处理广播消息。不管是系统消息还是自定义消息，都可以通过BroadcastReceiver来进行处理。形象的比喻，Intent是一种一对一的通信，广播消息是一种一对多的通信。

广播消息

- 使用Intent发送广播消息非常简单
  - 只需创建一个Intent
  - 并调用sendBroadcast()函数把Intent携带的信息广播出去

```
1. String UNIQUE_STRING =
   “com.example.BroadcastReceiverDemo";
2. Intent intent = new Intent(UNIQUE_STRING);
3. intent.putExtra("key1", "value1");
4. intent.putExtra("key2", "value2");
5. sendBroadcast(intent);
```

  - 在构造Intent时必须定义一个**全局唯一的字符串**，用来标识其要执行的动作，通常使用应用程序包的名称
  - 要在Intent传递额外数据，可以用Intent的putExtra()方法

广播消息

- 接收消息

（1）在AndroidManifest.xml文件中注册BroadcastReceiver

```
1.  <receiver android:name=".MyBroadcastReceiver">
2.        <intent-filter>
3.        <action
    android:name="com.example.BroadcastReceiverDemo" />
4.        </intent-filter>
5.        </receiver>
```

（2）创建BroadcastReceiver需继承BroadcastReceiver类，并重载onReceive()方法。示例代码如下：

```
1.  public class MyBroadcastReceiver extends
    BroadcastReceiver {
2.        @Override
3.        public void onReceive(Context context, Intent intent)
    {   //TODO: React to the Intent received.
4.        } }
```

广播机制

- 广播消息的过程
  - 消息发送
    - 无序广播sendBroadcast
    - 有序广播sendOrderBroadcast
    - 持续广播sendStickyBroadcast
      - 广播消息的实质就是一个Intent对象。使用sendBroadcast ()或sendStickyBroadcast ()方法发出去的Intent，所有满足条件的BroadcastReceiver都会随机地执行其onReceive()方法；而sendOrderBroadcast ()发出去的Intent，会根据BroadcastReceiver注册时Intent Filter 设置的优先级的顺序来执行，相同优先级的BroadcastReceiver则是随机执行。sendStickyBroadcast()方法主要的不同的是，Intent在发送后一直存在，并且在以后调用registerReceiver注册相匹配的BroadcastReceiver时会把这个Intent直接返回。

广播机制

- BroadcastReceiver的生命周期
  - BroadcastReceiver的onReceive()方法执行完成后，BroadcastReceiver的实例就会被销毁。如果onReceive()方法在10s内没有执行完毕，Android会认为改程序无响应。所以在BroadcastReceiver里不能做一些比较耗时的操作，否则会弹出“Application NoResponse”对话框。特别说明的是，这里不能使用子线程来解决，因为BroadcastReceiver的生命周期很短，子线程可能还没有结束BroadcastReceiver就先结束了。BroadcastReceiver一旦结束，此时它所在的进程很容易在系统需要内存时被优先杀死，因为它属于空进程。

广播机制

- **sendBroadcast和sendStickyBroadcast的区别**
  - sendBroadcast中发出的intent在ReceiverActivity不处于onResume状态是无法接受到的，即使后面再次使其处于该状态也无法接受到。
    而sendStickyBroadcast发出的Intent当ReceverActivity重新处于onResume状态之后就能重新接受到其Intent.这就是the Intent will be held to be re-broadcast to future receivers这句话的表现。就是说sendStickyBroadcast发出的最后一个Intent会被保留，下次当Recevier处于活跃的时候，又会接受到它。

消息机制

- 基本介绍
  - 采用消息机制的目的是完成主线程与子线程之间的消息传递
  - 当一个程序第一次启动的时候，Android会启动一个LINUX进程和一个主线程（Main Thread）。默认的情况下，所有该程序的组件都将在该进程和线程中运行。

消息机制

- 基本介绍
  - 主线程主要负责处理与UI相关的事件，如：用户的按键事件，用户接触屏幕的事件以及屏幕绘图事件，并把相关的事件分发到对应的组件进行处理。所以主线程通常又被叫做**UI线程**。
  - 在开发Android应用时必须遵守**单线程模型**的原则：Android UI操作并不是线程安全的，并且这些操作必须在UI线程中执行。
    - 1. 不要阻塞UI线程
    - 2. 确保只在UI线程中访问Android UI toolkit，禁止子线程更新主线程的UI组件

消息机制

- 基本概念
  - 1. Message
    - 线程间通讯的数据单元。例如后台线程在处理数据完毕后需要更新UI，则可发送一条包含更新信息的Message给UI线程。
  - 2. Message Queue
    - 消息队列，用来存放通过Handler发布的消息，按照先进先出执行。消息队列通常附属于某一个创建它的线程。
  - 3. Handler
    - 是Message的主要处理者，负责将Message添加到消息队列以及对消息队列中的Message进行处理。

消息机制

- 4. Looper
  - Message Queue和Handler之间的桥梁，循环取出Message Queue里面的Message，并交付给相应的Handler进行处理。

- 5. 线程
  - UI thread 通常就是主线程，而Android启动程序时会替它建立一个Message Queue。
  - 每一个线程里可含有一个Looper对象以及一个MessageQueue数据结构。在应用程序里，可以定义Handler的子类别来接收Looper所送出的消息。

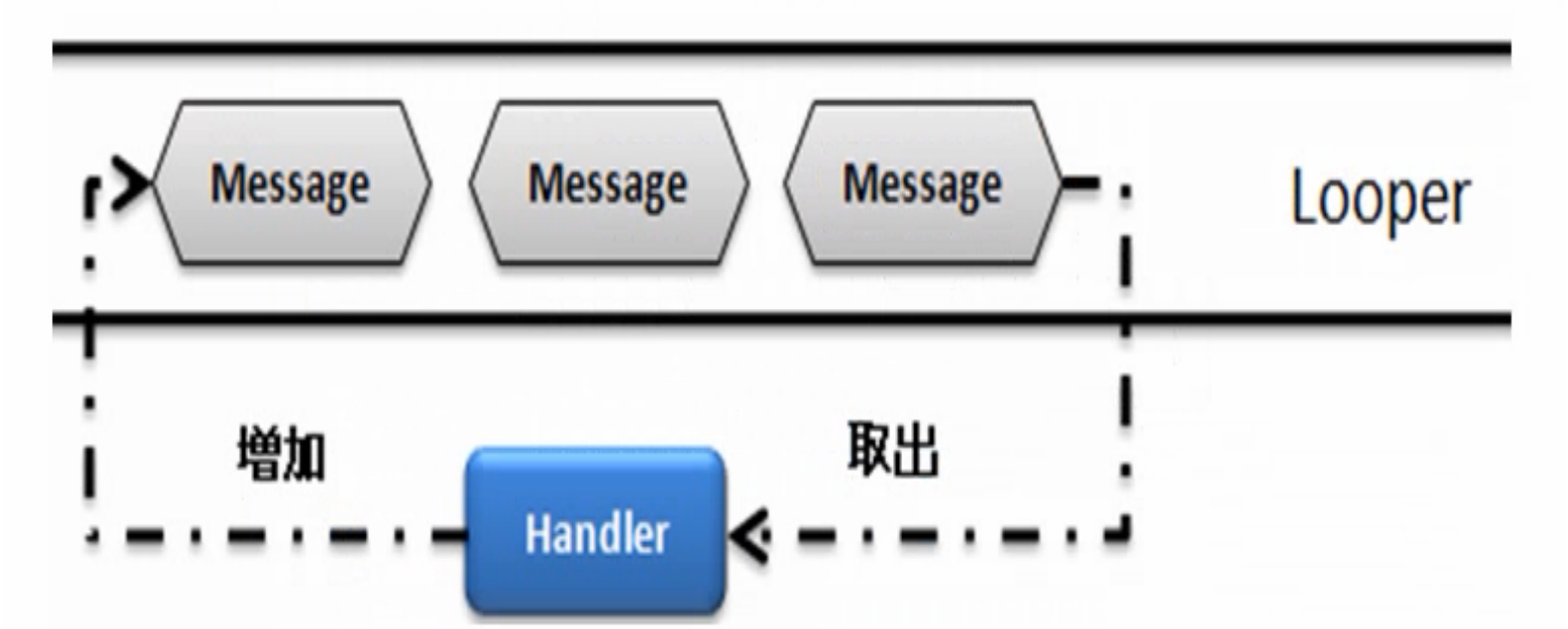
消息机制

- 运行机制：
  - 每个线程都**可以并仅可以**拥有一个Looper实例，消息队列MessageQueue在Looper的构造函数中被创建并且作为成员变量被保存，也就是说MessageQueue相对于**线程也是唯一**的。
  - Android应用在启动的时候会默认为主线程创建一个Looper实例，并借助相关的Handler和Looper里面的MessageQueue完成对Activities、Services、Broadcase Receivers等的管理。

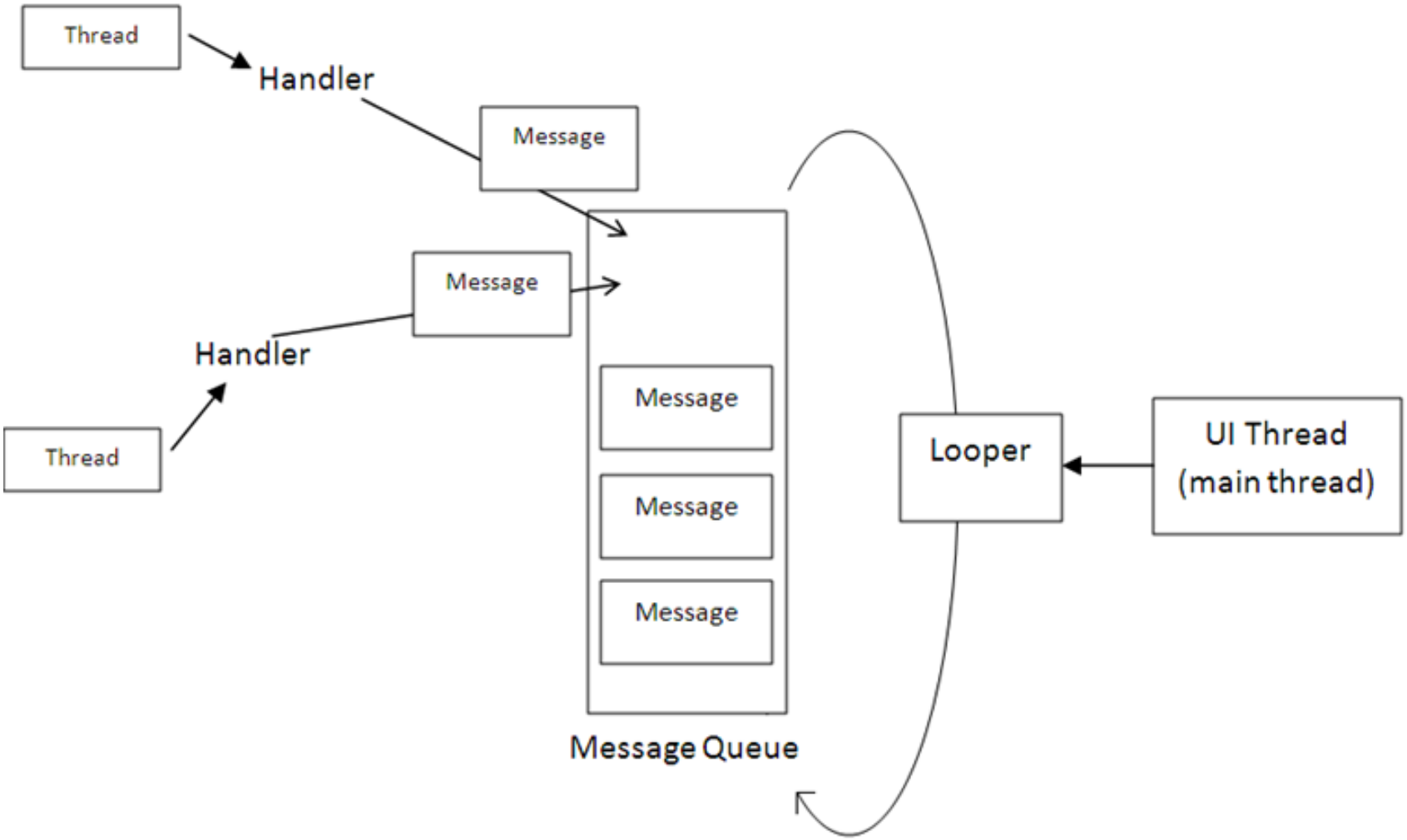
消息机制

- 运行机制：
  - 而在子线程中，Looper需要通过显式调用Looper.Prepare()方法进行创建。Prepare方法通过ThreadLocal来保证Looper在线程内的唯一性，如果Looper在线程内已经被创建并且尝试再度创建"Only one Looper may be created per thread"异常将被抛出。

消息机制

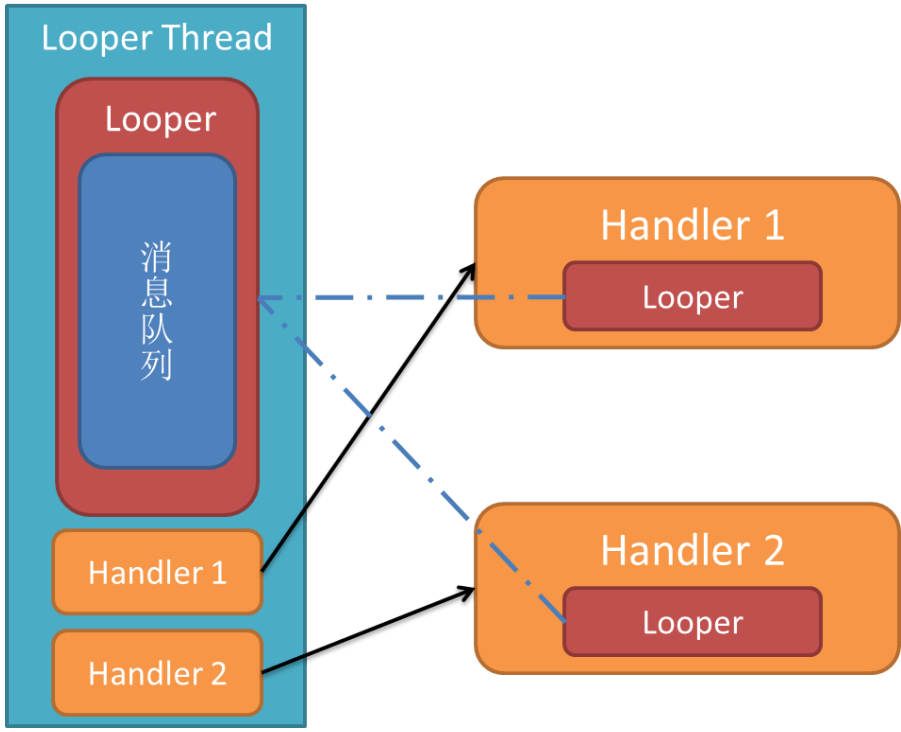


消息机制


Thread
Handler
Message
Message
Handler
Thread
Message
Message
Message
Message Queue
Looper
UI Thread
(main thread)

消息机制

```
1.  public class LooperThread extends Thread {
2.      private Handler handler1;
3.
4.      private Handler handler2;
5.
6.      @Override
7.      public void run() {
8.          // 将当前线程初始化为Looper线程
9.          Looper.prepare();
10.         // 实例化两个handler
11.         handler1 = new Handler();
12.         handler2 = new Handler();
13.         // 开始循环处理消息队列
14.         Looper.loop();
15.     }
16. }
```



消息机制

- 说明
  - 在Android中，线程包括
    - a. 有消息循环的线程（该类型的线程一般都会有一个Looper）
    - b. 无消息循环的线程；
  - 线程分为主线程（UI线程）和子线程
    - 只要是关于UI相关的东西，就不能放在子线程中处理，因为子线程是不能操作UI的，只能进行数据、系统等其他非UI的操作
    - UI线程就是一个消息循环的线程
  - 每个线程都可以有自己的消息队列和消息循环，也可以没有。

消息机制

- 下面的程序对吗？
  - 创建一个新的线程来处理联网操作

```
1. public void onClick(View v){
2. new Thread(new Runnable(){
3. Bitma b = loadImageFromNetwork("http://example.com/image.png");
4. mImageView.setImageBitmap(b);
5. }).start();
6. }
```

  - 错在：在UI线程之外访问Android Ui tookit

消息机制

- 实现从其他线程中访问UI线程的方法：

1. Activity.runOnUiThread(Runnable)
2. View.post(Runnable)
3. View.postDelayed(Runnable, **long**)

- 正确的方法：

```
1.  public void onClick(View v){
2.  new Thread(new Runnable(){
3.  public void run(){
4.  final Bitmap bitmap = loadImageFromNetwork("http://example.com/image.png");
5.  mImageView.post(new Runnable(){
6.  public void run(){
7.  mImageView.setImageBitmap(bitmap);
8.  } }); } }).start();
9.  }
```

消息机制

- Message类
  - Message类定义的变量及常用方法

序号	变量或方法	类型	描述
1	public int arg1	变量	传递整型数据
2	public int arg2	变量	传递整型数据
3	public Object obj	变量	定义传递的信息数据
4	public int what	变量	定义此message属于何种操作
5	public Handler getTarget()	普通	取得操作此消息的Handler对象

消息机制

- Handler类
  - Message对象封装消息，而这些消息的操作需要android.os.Handler类完成
  - Handler类的常用操作方法

序号	方法
1	Handler()
2	Handler(Looper looper)
3	Message obtainMessage
4	HandleMessage
5	removeMessage
6	sendMessage

消息机制

- Handler的作用：
  - 1.是把消息加入特定的（Looper）消息队列中；
  - 2.分发消息；
  - 3.处理该消息队列中的消息；

- handler应该由处理消息的线程创建；也就是说，如果handler的函数handleMessage里处理的是UI的消息，也就是更新界面的事情，那么该handler需要在主线程中创建；

消息机制

- handler与创建它的线程之间的关系
  - handler与创建它的线程相关联，而且也只与创建它的线程相关联。
  - handler运行在创建它的线程中，所以，如果在handler中进行耗时的操作，会阻塞创建它的线程。

消息机制

- Looper类
  - 在使用Handler处理Message时，都需要依靠一个Looper通道完成，当用户取得一个Handler对象时，实际上都是通过Looper完成的。
  - 在一个Activity类中，会自动帮助用户启动Looper对象，而若是在一个用户自定义的类中，则需要用户手工调用Looper类中的若干方法，之后才可以正常启动Looper对象。

消息机制

- Looper类
  - 类的常用方法

序号	方法	说明
1	getMainLooper()	
2	myLooper()	返回当前线程的Looper
3	prepare()	初始化
4	prepareMainLooper()	初始化主线程Looper对象
5	quit	消息队列结束时调用
6	loop	启动消息队列

消息机制

- handler对象使用要点：
  - 1.handler对象在主线程中构造完成（并且启动工作线程之后不要再修改之，否则会出现数据不一致），然后在工作线程中可以放心的调用发送消息SendMessage等接口
  - 2. 除了所述的handler对象之外的任何主线程的成员变量如果在工作线程中调用，仔细考虑线程同步问题。
  - 3. 如果有必要需要加入同步对象保护该变量。

消息机制

- AsyncTask
  - 子线程无法直接对主线程组件进行更新，而且如果所有的开发都分别定义若干个子线程的操作对象，则这多个对象同时对主线程操作就会非常麻烦。
  - 为了解决该问题，提供了android.os.AsyncTask（异步任务）类，通过此类完成非阻塞的操作类。
  - 功能与handler类似，可以在后台进行操作之后更新主线程的UI，更简单。

消息机制

- AsyncTask
  - AsyncTask类的继承关系
    - java.lang.Object
      - ↳ android.os.AsyncTask<Params, Progress, Result>
    - Params：启动时需要的参数类型，比如HTTP请求的URL
    - Progress：后台执行任务的百分比，如进度条需要传递的是Integer
    - Result：后台执行完毕之后返回的信息，如完成数据信息显示传递的是String

消息机制

- AsyncTask
  - AsyncTask类的常用方法
    - onPreExecute(), 该方法将在执行实际的后台操作前被UI thread调用。可以在该方法中做一些准备工作，如在界面上显示一个进度条。
    - doInBackground(Params...), 将在onPreExecute 方法执行后马上执行，该方法运行在后台线程中。这里将主要负责执行那些很耗时的后台计算工作。可以调用 publishProgress方法来更新实时的任务进度。该方法是抽象方法，子类必须实现。
    - publishProgress，更新线程进度
    - onProgressUpdate(Progress...),在publishProgress方法被调用后，UI thread将调用这个方法从而在界面上展示任务的进展情况，例如通过一个进度条进行展示。
    - onPostExecute(Result), 在doInBackground 执行完成后，onPostExecute 方法将被UI thread调用，后台的计算结果将通过该方法传递到UI thread.

消息机制

- AsyncTask
  - 为了正确的使用AsyncTask类，以下是几条必须遵守的准则：
    - 1. Task的实例必须在UI thread中创建
    - 2. execute方法必须在UI thread中调用
    - 3. 不要手动的调用onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...)这几个方法
    - 4. 该task只能被执行一次，否则多次调用时将会出现异常
  - doInBackground方法和onPostExecute的参数必须对应，这两个参数在AsyncTask声明的泛型参数列表中指定，第一个为doInBackground接受的参数，第二个为显示进度的参数，第三个为doInBackground返回和onPostExecute传入的参数

消息机制

- 再进一步
  - 用AsyncTask类来完善

```
1.  public void onClick(View v){
2.  new DownloadImageTask().execute("http://example.com/image.png");
3.  }
4.  private class DownloadImageTask extends AsyncTask<String, Void, Bitmap>{
5.  /**The sytem calls this to perform work in worker thread and delivers it the para
    meters given to AsyncTask.execute()*/
6.  protected Bitmap doInBackground(String... urls){
7.  return loadImageFromNetwork(urls[0]);
8.  }
9.  /**系统调用该方法来在更新UI线程，并将doInbackground()的结果返回出来*/
10. protected void onPostExecute(Bitmap result){
11. mImageView.setImageBitmap(result);
12. } }
```

利用外部线程更改Ui主线程的方法

- 三种基础方法
  - **1.Thread+handler**
  - **2.TimerTask+handler**
  - **3.Runnable+Handler.postDelayed(runnable,time)**

在Handler 异步实现时,涉及到 Handler, Looper, Message,Thread四个对象，实现异步的流程是主线程启动Thread（子线程）→ thread(子线程)运行并生成Message→Looper获取Message并传递给Handler, → Handler逐个获取Looper中的Message，并进行UI变更。