网页查看器

功能：通过输入一个地址，获得这个网页编写的源代码。然后将获取的源代码显示在屏幕上

思路：

1、获取地址

```
String  path  =  etxt.getText().toString();//将文本框中的地址获取出来
```

2、通过拿到的地址，去连接访问网络

```
URL  u  =  new  URL(path);
HttpURLConnection  connection  =  (HttpURLConnection)  u.openConnection()
```

3、设置访问网络的方式、网络响应等待时间及风络响应的状态

```
connection.setRequestMethod("GET");
connection.setReadTimeout(10000);
int  code  =  connection.getResponseCode();//获取访问地址的响应状态
```

4、做出判断是否连接成功

```
if  (code  ==  200)  {
```

5、如果连接成功，就获取网页的源代码,获取源代码的操作可以封装成一个类调用。

```
InputStream  input  =  connection.getInputStream();
ByteArrayOutputStream  array  =  new  ByteArrayOutputStream();
int  len;
byte[]  arr  =  new  byte[1024];
while  ((len  =  in.read(arr))  !=  -1)  {
    array.write(arr,  0,  len);
String   string =  new  String(array.toByteArray());
```

```
public final class StrFac {
    public static String getString(InputStream in){
        ByteArrayOutputStream array = new ByteArrayOutputStream();
        int len;
        byte[] arr = new byte[1024];
        try {
            while ((len = in.read(arr)) != -1) {
                array.write(arr, 0, len);
            }
            String  str = new String(array.toByteArray());
            return str;
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
```

6、然后对读取的内容再做出判断，如果读取的内容为空，就返回一个读取为空的消息

```
if  (("".equals(string)){    这里判断获取的字符是否为空也可以用
if(TextUtils.isEmpty(string)){
```

```
mag.obj = "读取为空";
mag.what = S;
mag.obj = string;
mag.what = C;
input.close();
} catch (Exception e) {
mag.obj = "网络异常";
mag.what = W;
```

出现异常及处理方法：

1、应用需要连接网络必须添加网络连接权限

```
<uses-permission android:name="android.permission.INTERNET"/>
```

2、ANR异常：有耗时操作在主线程时，会出现应用无响应异常。如果在主线程运行耗时操作会导致整个应用处于卡住的情况，对于用户的体验及其不好。解决办法，开启子线程将耗时操作在子线程中处理。

开启一个子线程在子线程中重写run()方法，将要操作的联网或其它耗时操作写入run()方法中

3、UI组件不可操作异常：由于规定在子线程中不可以对UI组件操作。解决办法，使用Handler类的消息传递机制，重写HandlerMessage（ ）方法。这个Handler类是一个信息处理类，依赖于Message，只要Message发现子线程中有数据，Handler就拿到这个数据（消息）马上处理，在这里是将这个数据给到UI显示出来。

具体代曲体现：

```
//使用常量描述消息状态                                    这是好习惯
private static final int CONTENT_EMTPEY = 1;
private static final int RESPONSE_ERROR = 2;
private static final int NET_ERROR = 3;
private static final int SUCCESS = 4;

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    url_edit = (EditText)findViewById(R.id.url_edit);
    code_txt = (TextView)findViewById(R.id.code_txt);
    handler = new Handler(){          消息处理器 ，使用它的原因，是在子线程无法操作UI。
        @Override
        public void handleMessage(Message msg) {  在主线程中处理的方法
            switch (msg.what) {
            case CONTENT_EMTPEY:
            case RESPONSE_ERROR:
            case NET_ERROR:
                Toast.makeText(getApplicationContext(), msg.obj.toString(), 0).show();
                break;
            case SUCCESS:
                String code = msg.obj.toString();
                code_txt.setText(code);//更新UI
                break;
            }
        }
    };
```

```
public void readurl(View v){
    new Thread(){                                          对应耗时操作，我们启动子线程
        Message msg = new Message();
        @Override
        public void run() {
            String path = url_edit.getText().toString();
            try {
                //一大步：获取源代码//需要注意的是：耗时操作（包括网络读取）都应该放到子线程
                URL url = new URL(path);                   获取网络的内容靠这两个对象
                HttpURLConnection connection = (HttpURLConnection)url.openConnection();
                //1.1对连接进行设置
                connection.setRequestMethod("GET");
                connection.setReadTimeout(5000);
                //1.2获取状态码
                int state = connection.getResponseCode();
                if(state == 200){//响应成功
                    //1.3 获取源代码                                    获取服务器发过来的流
                    InputStream is = connection.getInputStream();
                    String code = StringFactory.getString(is);
                    is.close();
                    if(TextUtils.isEmpty(code)){                       传递的内容，给obj
                        msg.obj = "内容为空";//将消息内容封装到msg对象中.
                        msg.what = CONTENT_EMTPEY; //what在次可理解为：这是什么消息
                    }
                    //二大步：将源代码显示到UI:要注意，使用子线程解决掉ANR异常后，会引发子线程不可操作UI的异常
                    //此时可使用Handler进行消息传到
                    msg.obj = code;//将源代码封装到msg对象中
                    msg.what = SUCCESS;
                }else{
                    msg.obj = "响应异常";
                    msg.what = RESPONSE_ERROR;
                }
            } catch (Exception e) {
                msg.obj = "网络异常";
                msg.what = NET_ERROR;
            }
            handler.sendMessage(msg);//发送消息              利用Message对象，传递消息
        }
    }.start();//记得启动线程
```

总结:

Android下的网络请求与响应

请求以及响应 通过：HttpURLConnection 处理

请求的时候：需要设置提交方式（GET，POST）、设置超时，

获取响应状态码（200代表成功）。

响应的时候：通过 输入流 读取服务器对请求做出的响应。

由于请求网络是个耗时操作，为了用户体验，我们将所有耗时操作放在子线程。

网络操作需要请求权限。

子线程操作UI会报异常。所以我们通过Handler对象在子线程发送数据，到UI（主）线程更新UI。

Handler需要Message对象的支持。Message对象的obj可用来带数据，what用来带状态。

对于字符串处理**Android**为我们提供了一个简单实用的**TextUtils**类
主要的功能如下**:**

是否为空字符 **static boolean isEmpty(CharSequence str)**

拆分字符串 **public static String[] split (String text, String expression)**，

拆分字符串使用正则 **public static String[] split (String text, Pattern pattern)**

确定大小写是否有效在当前位置的文本**TextUtils.getCapsMode(CharSequence cs, int off, int reqModes)**

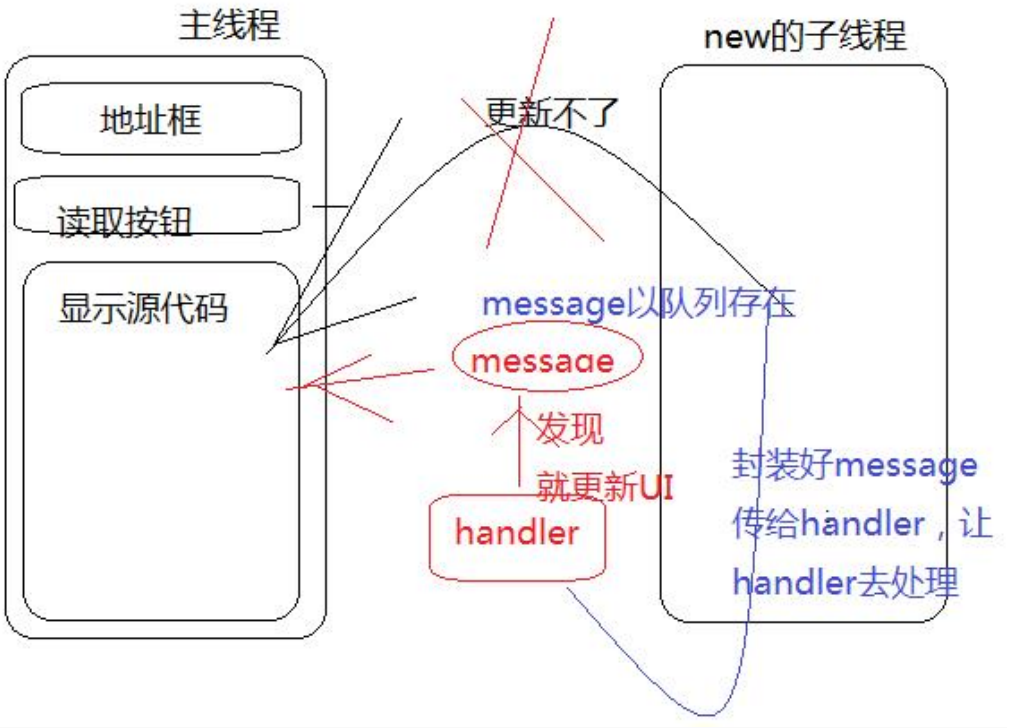
使用**HTML**编码这个字符串 **static String TextUtils.htmlEncode(String s)**

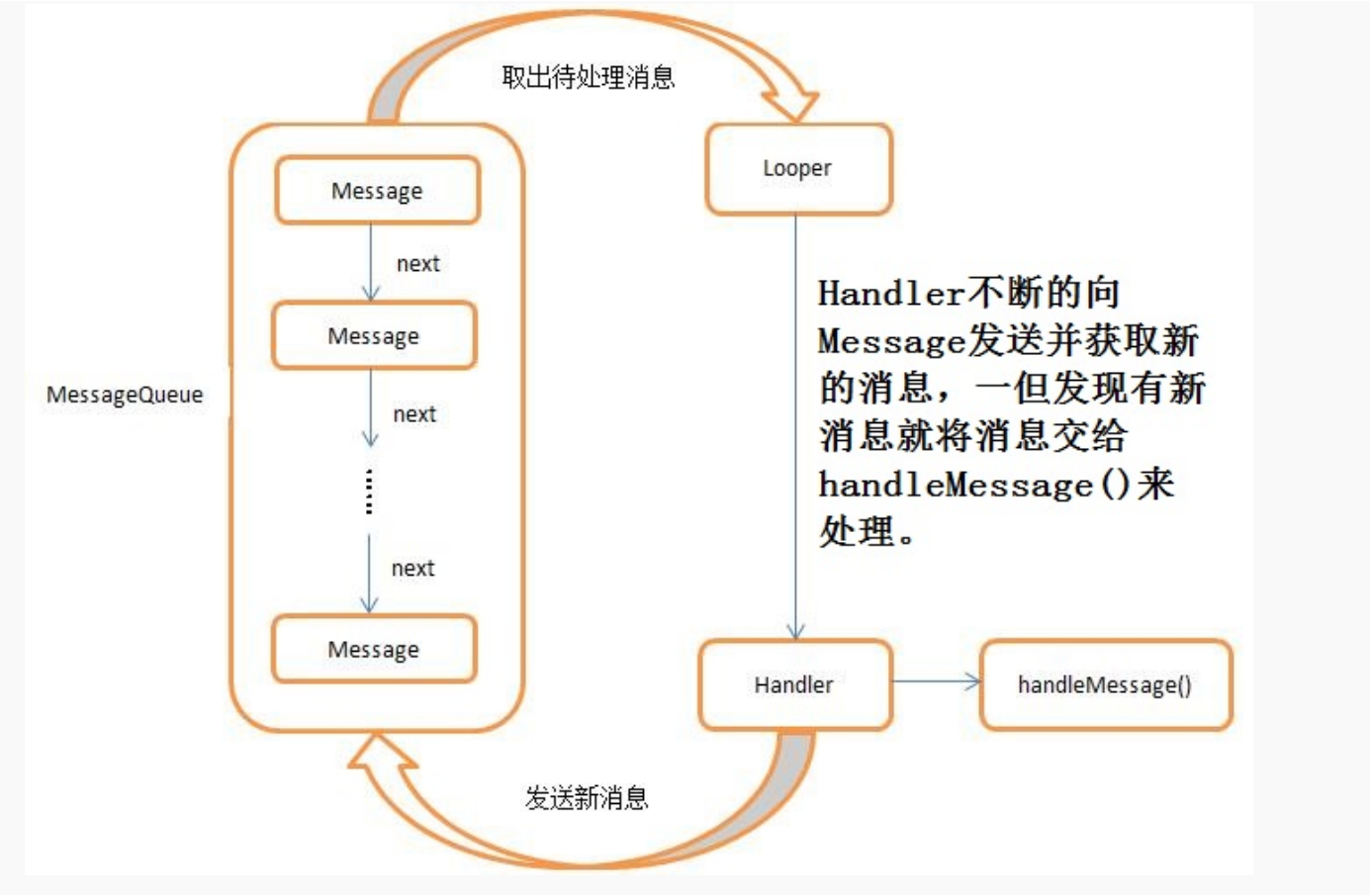
Handler消息机制的原理

出于性能优化的考虑,Android UI操作并不是线程安全,如果有多个线程并发操作UI组件,可能导致线程安全问题。如果在一个Activity中有多个线程去更新UI,并且都没有加锁机制,可能会导致界面混乱,如果加锁的话可以避免该问题但又会导致性能下降。因此,Android规定只允许UI线程修改Activity的UI组件。当程序第一次启动时,Android会同时启动一条主线程(Main Thread),主线程主要负责处理与UI相关的事件,比如用户按钮事件,并把相关的事件分发到对应的组件进行处理,因此主线程又称为UI线程。那么在新启动的线程中更新UI组件,这就需要借助handler的消息传递机制来实现了。

Handler类包含如下方法用于发送、处理消息:

- ♦ void handlerMessage(Message msg):处理消息的方法,该方法通常用于被重写。
- ♦ final boolean hasMessage(int what):检查消息队列中是否包含what属性为指定值的消息。
- ♦ sendEmptyMessage(int what):发送空消息
- ♦ final boolean sendMessage(Message msg):立即发送消息,注意这块返回值,如果message成功的被放到message queue里面则返回true,反之,返回false;





ScrollView控件只支持垂直滚动。

HorizontalScrollView除只支持水平滚动外，其他都和ScrollView一样。

如果要屏幕支持垂直滚动和水平滚动，那么就要让HorizontalScrollView作为ScrollView的直接子view，或者让ScrollView作为HorizontalScrollView的直接子view。

网络图片浏览器

图片的存储方式：图片是用点的方式存储，通过每个点上的纵横位置和颜色，再进行计算得到一个数用来保存这个点的图像，最终整张图片会生成一长乘宽的数组来存储。

与获取网页源代码基本相同，不同点是在在访问网页连接成功后获取到流，然后通过图片工厂类BitmapFactory，将流转为图片产生一个Bitmap对象。　再使用Message()对象将图片做为消息传回给ImageView这个UI显示出来。

代码体现;

```
handler = new Handler(){
    @Override
    public void handleMessage(Message msg) {
        if(msg.what==1){
            Bitmap bmp = (Bitmap)msg.obj;
            //将图片显示到界面
            img.setImageBitmap(bmp);
        }
    }
};
```

通过setImageBitmap方法设置ImageView的显示内容。

```
String path = url_edit.getText().toString().trim();
try {
    URL url = new URL(path);//地址转为url链接
    //打开连接对象
    HttpURLConnection connection = (HttpURLConnection)url.openConnection();
    //设置连接对象
    connection.setRequestMethod("GET");
    connection.setReadTimeout(5000);
    //获取响应状态
    int state = connection.getResponseCode();
    if(state == 200){
        InputStream is = connection.getInputStream();//获取流
        Bitmap bmp = BitmapFactory.decodeStream(is);//通过图片工厂类，将流转为图
        片

        Message msg = new Message();
        msg.obj = bmp;
        msg.what = 1;//本案例其他状态不管。

        handler.sendMessage(msg);//发送
    }
} catch (Exception e) {
    Log.e("mylog", e.getMessage());
}
```

cache目录和files目录区别

cache目录:是应用用来存储一些运行缓存数据的文件夹，在清理手机垃圾时，这个文件夹会被清空。

files目录：是应用用来存储一些应用本身的数据文件，在清理手机垃圾时，不会被清空。

SmartImageView控件

第三方的UI控件　用来通过地址获取该地址的图片

通过WebImageCache对象，将你传入的图片地址的图片资源，缓存到应用的缓存目录之下。缓存成功以后，ImageView显示的实际是缓存目录中的本地文件。

第三方的工具包使用方法：

1、导入工具，注意在导入工具前在当前项目下一定要重新给建立一个包然后导入

2、使用该工具不需要Handler类的消息传递机制，Message类其内部已经给定义好方法直接使用

代码体现：

```
    handler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            if(msg.what==1){
                Bitmap bmp = (Bitmap)msg.obj;
                //将图片显示到界面
                img.setImageBitmap(bmp); 通过setImageBitmap方法
            }                            设置ImageView的显示内
        }                                容。
    };
```

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    etxt = (EditText) findViewById(R.id.etxt);
    img = (SmartImageView) findViewById(R.id.imgg);

}

public void huoqu(View v) {
String path = etxt.getText().toString();
img.setImageUrl(path);

}
```

要使用第三方的UI组件来显示图片
在布局中也是一样要定义第三方的UI组件

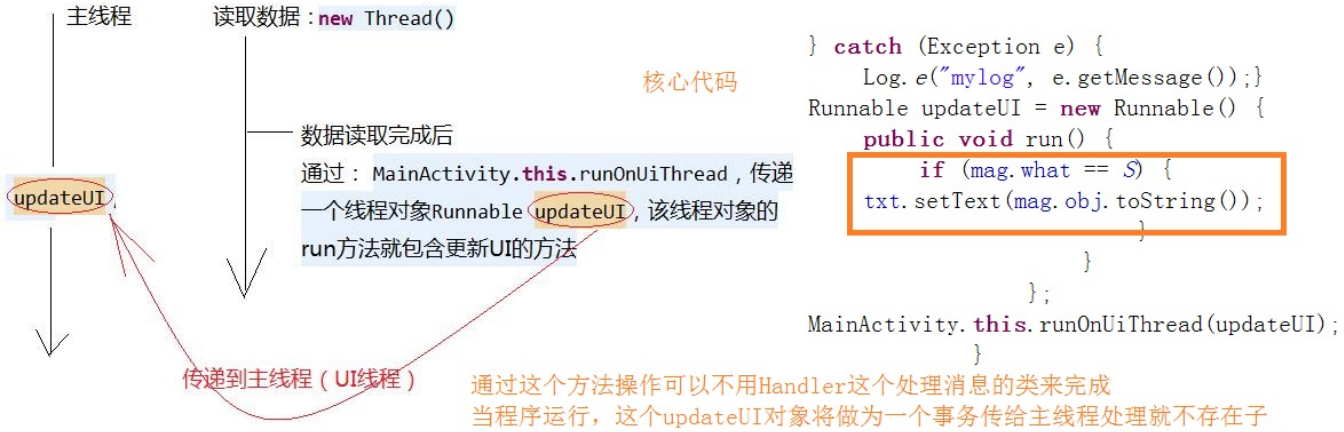
这个方法要注意是小写的Url，另外有一个URL的方法容易混淆

```xml
<com.loopj.android.image.SmartImageView
    android:id="@+id/imgg"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="6" />
```

注意要使用的第三方UI组件

runOnUiThread()

通过输入一个地址，获得这个网页编写的源代码。然后将获取的源代码显示在屏幕上



核心代码

```java
} catch (Exception e) {
    Log.e("mylog", e.getMessage());}
Runnable updateUI = new Runnable() {
    public void run() {
        if (mag.what == 5) {
        txt.setText(mag.obj.toString());
                }
            }
        };
MainActivity.this.runOnUiThread(updateUI);
    }
```

通过这个方法操作可以不用Handler这个处理消息的类来完成当程序运行，这个updateUI对象将做为一个事务传给主线程处理就不存在子线程不可操作UI组件的问题，相当于主线程调用了一个子线程的事件来处理，而真正操作这个UI组件的还是子线程自己。

代码体现：

```java
public class MainActivity extends Activity {
    //使用常量描述消息状态
    private static final int CONTENT_EMTPEY = 1;
    private static final int RESPONSE_ERROR = 2;
    private static final int NET_ERROR = 3;
    private static final int SUCCESS = 4;
    private EditText url_edit;
    private TextView code_txt;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        url_edit = (EditText)findViewById(R.id.url_edit);
        code_txt = (TextView)findViewById(R.id.code_txt);

    }

    public void readurl(View v){
        new Thread(){
            String msg = "";
            int what = 0;
            @Override
            public void run() {
                String path = url_edit.getText().toString();
                try {
                    try {
                        //暂停updateUI线程的执行
                        Thread.sleep(15000);//该代码可以导致主线程被停止。
                    } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                    //一大步：获取源代码//需要注意的是：耗时操作（包括网络读取）都应该放到子线程
                    URL url = new URL(path);
                    HttpURLConnection connection = (HttpURLConnection)url.openConnection();
                    //1.1对连接进行设置
                    connection.setRequestMethod("GET");
                    connection.setReadTimeout(5000);
                    //1.2获取状态码
                    int state = connection.getResponseCode();
                    if(state == 200){//响应成功
                        //1.3 获取源代码
                        InputStream is = connection.getInputStream();
                        String code = StringFactory.getString(is);
                        is.close();
                        if(TextUtils.isEmpty(code)){
                            //? 之前我们使用handler对象进行数据的更新， 如果没有handler 数据如何更新到ui中？
                            msg = "内容为空";//将消息内容封装到msg对象中.
                            what = CONTENT_EMTPEY; //what在次可理解为：这是什么消息
                        }
                        //二大步：将源代码显示到UI:要注意，使用子线程解决掉ANR异常后，会引发子线程不可操作UI的异常
                        //此时可使用handler进行消息传到
                        msg = code;//将源代码封装到msg对象中
                        what = SUCCESS;

                    }else{
                        msg = "响应异常";
                        what = RESPONSE_ERROR;
                    }
                } catch (Exception e) {
                    msg = "网络异常";
                    what = NET_ERROR;
                }
                //创建更新UI的线程对象
                Runnable updateUI =  new Runnable() {
                    @Override
                    public void run() {
                        try {
                            //暂停updateUI线程的执行
                            Thread.sleep(4000);//该代码可以导致主线程被停止。
                        } catch (InterruptedException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }
                        switch (what) {
                        case CONTENT_EMTPEY:
                        case RESPONSE_ERROR:
                        case NET_ERROR:
                            Toast.makeText(getApplicationContext(), msg, 0).show();
                            break;
                        case SUCCESS:
                            code_txt.setText(msg);
                        }
                    }
                };
```

此处不再使用Handler消息处理的类，来处理Message收到的消息

获取到的数据只做封装不做处理，留给后面的线程来处理，给UI组件

在子线程中再创建一个线程来处理UI数据更新的问题

```
            //解决数据更新到UI的问题
            MainActivity.this.runOnUiThread(updateUI);

        }
    }.start();//记得启动线程
}
}
```

最后用主线程来调用处理UI更新的子线程