[Gfw设置 3](#_Toc506481296)

[导入上拉加载更多、下拉刷新控件（以WebView为例）： 5](#_Toc506481297)

[1.在Module:app的build.gradle文件中的dependencies加入compile 'com.github.Aspsine:SwipeToLoadLayout:1.0.3' 5](#_Toc506481298)

[2.在Project:(项目名)的build.gradle文件中的repositories加入maven { url "https://jitpack.io" } 5](#_Toc506481299)

[3.在布局文件中引用刷新布局、头部和底部： 6](#_Toc506481300)

[4.创建头文件和底部文件： 7](#_Toc506481301)

[5.在Activity中引用: 13](#_Toc506481302)

[导在线阅读PDF控件（方法一）： 18](#_Toc506481303)

[1.在settings.gradle中导入':android-pdfview' 18](#_Toc506481304)

[2.往APP目录下的libs目录导入xUtils-2.6.14jar包 18](#_Toc506481305)

[3.在build.gradle文件的dependencies中导入compilefiles('libs/xUtils-2.6.14.jar') compileproject(':android-pdfview') 18](#_Toc506481306)

[4.在需要的XML中引用： 18](#_Toc506481307)

[5在需要的Activity中引用： 19](#_Toc506481308)

[方法二： 23](#_Toc506481309)

[1.在build.gradle的dependencie中引用 compile 'com.lidong.pdf:android\_pdf:1.0.1' 23](#_Toc506481310)

[2.在需要引用的XML中引用： 23](#_Toc506481311)

[3.在Activity中的使用： 23](#_Toc506481312)

[Android Stuido 修改包名 29](#_Toc506481313)

[AS导入CrosswalkWebViewModule 32](#_Toc506481314)

[AS接入融云即时通讯sdk 36](#_Toc506481315)

[Android 7.0 权限变更 37](#_Toc506481316)

[FileProvider 38](#_Toc506481317)

[常见使用场景 41](#_Toc506481318)

[历史版本问题 44](#_Toc506481319)

# Gfw设置

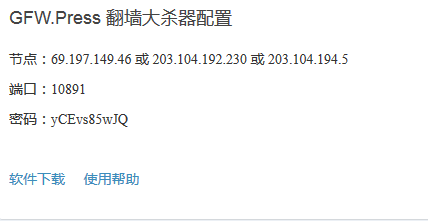
[](https://gfw.press/blog/?attachment_id=84)双击下载的安装包，按提示进行安装

安装过程通常只需点击几次 Next 按钮；

建议不要修改安装路径，如果修改，路径不要包含空格；

如果是 Windows 10 系统，请使用兼容模式进行安装；

如果安装失败，一般是安装包下载出错，请重新下载；

第一次运行，或者还没有进行过配置，会自动弹出配置窗口：

节点地址：填右图的节点中其中的一个；（节点地址可能会变，如果翻不了墙，去官网看一下节点地址是否改变）

节点端口：填右图的端口；

连接密码：填右图的密码；

本地端口就是代理服务器端口，默认是3128，通常不需要修改；

填写配置信息完成后，点击“确定”按钮，大杀器保存配置，开始运行，窗口缩小为右下角的一个绿色方块图标；如需打开配置窗口，双击图标即可；

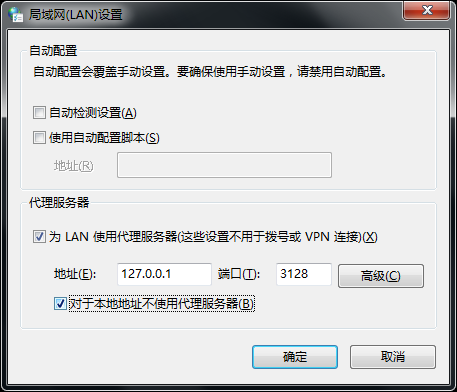
设置代理服务器

设置了代理服务器，浏览器就可以通过大杀器翻墙上网了；

点击Windows的“开始菜单》控制面板》Internet选项》连接》局域网设置”后，将出现配置窗口，按下图设置即可，端口对应就是大杀器配置的“本地端口”

地址： 127.0.0.1

端口：3128

[](https://gfw.press/blog/?attachment_id=85)

配置完成，点击“确定”按钮，然后打开IE、Chrome等浏览器，就可以翻墙访问谷歌之类被屏蔽的网站了。

注意：需要先翻墙

官网地址：https://gfw.press//

账号：[576804093@163.com](mailto:576804093@163.com)

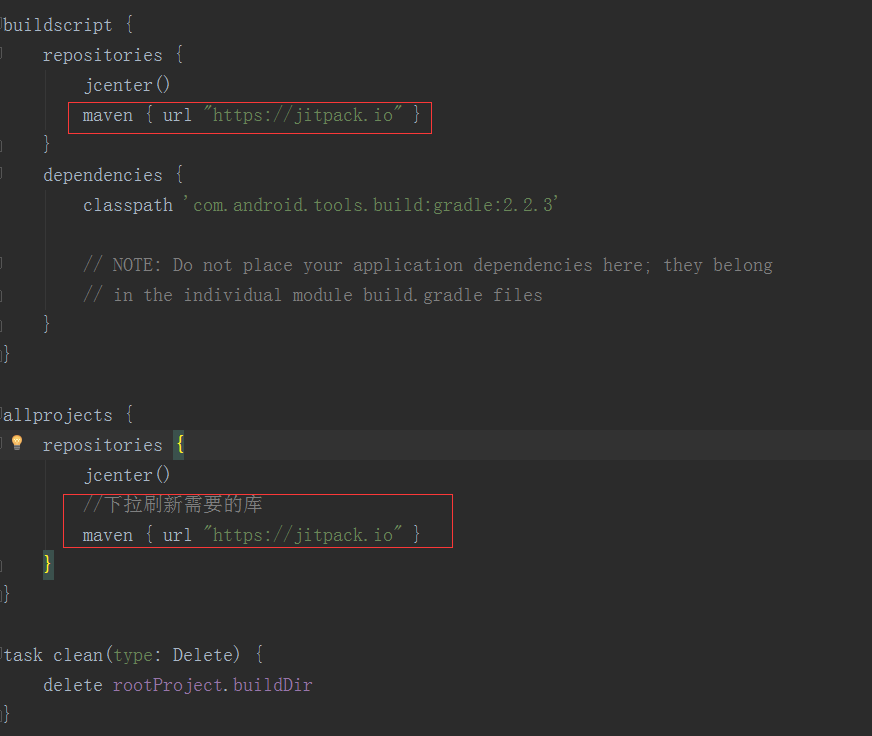
密码：3060371LOVE

# 导入上拉加载更多、下拉刷新控件（以WebView为例）：

1.在Module:app的build.gradle文件中的dependencies加入compile 'com.github.Aspsine:SwipeToLoadLayout:1.0.3' **如**



2.在Project:(项目名)的build.gradle文件中的repositories加入maven { url "https://jitpack.io" } **如**



### 3.在布局文件中引用刷新布局、头部和底部：

<com.aspsine.swipetoloadlayout.SwipeToLoadLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/swipeToLoadLayout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:layout\_weight="1">

<com.example.administrator.signaturetest.PullToRefreshAndLoad.RefreshHeaderView

android:id="@id/swipe\_refresh\_header"

android:layout\_width="match\_parent"

android:gravity="center"

android:layout\_height="50dp" />

<WebView

android:id="@id/swipe\_target"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:layout\_weight="1"/>

<com.example.administrator.signaturetest.PullToRefreshAndLoad.LoadMoreFooterView

android:id="@id/swipe\_load\_more\_footer"

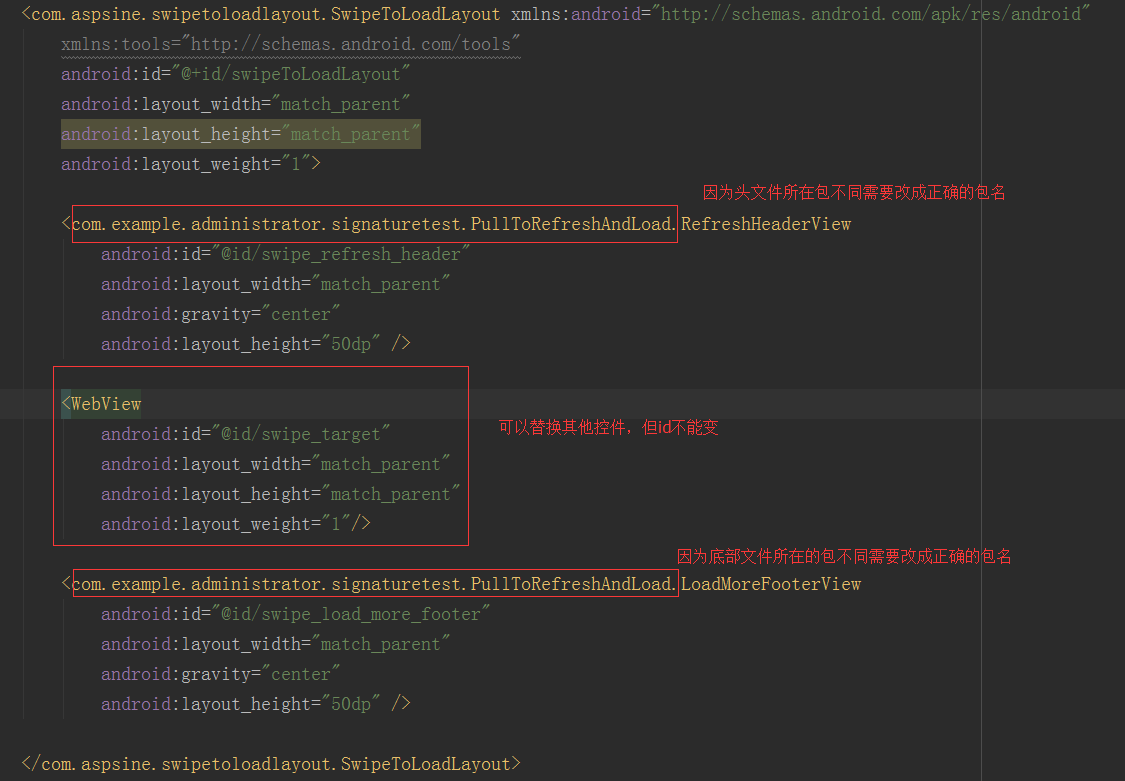
android:layout\_width="match\_parent"

android:gravity="center"

android:layout\_height="50dp" />

</com.aspsine.swipetoloadlayout.SwipeToLoadLayout>

其中四个控件ID不能改为其他



### 4.创建头文件和底部文件：

/\*\*

\* Created by Administrator on 2017/7/9.

\* 刷新头部

\*/

public class RefreshHeaderView extends TextView implements SwipeRefreshTrigger, SwipeTrigger {

public RefreshHeaderView(Context context) {

super(context);

}

public RefreshHeaderView(Context context, AttributeSet attrs) {

super(context, attrs);

}

@Override

public void onRefresh() {

// setText("REFRESHING");

setText("正在刷新中……");

}

@Override

public void onPrepare() {

setText("");

}

@Override

public void onMove(int yScrolled, boolean isComplete, boolean automatic) {

if (!isComplete) {

if (yScrolled >= getHeight()) {

// setText("RELEASE TO REFRESH");

setText("松开刷新");

} else {

// setText("SWIPE TO REFRESH");

setText("最后刷新时间: " + getLastUpdateTime());

}

} else {

// setText("REFRESH RETURNING");

setText("刷新成功");

}

}

@Override

public void onRelease() {

}

@Override

public void onComplete() {

setText("刷新成功");

}

@Override

public void onReset() {

setText("");

}

/\*\*

\* 获得系统的最新时间

\*

\* @return

\*/

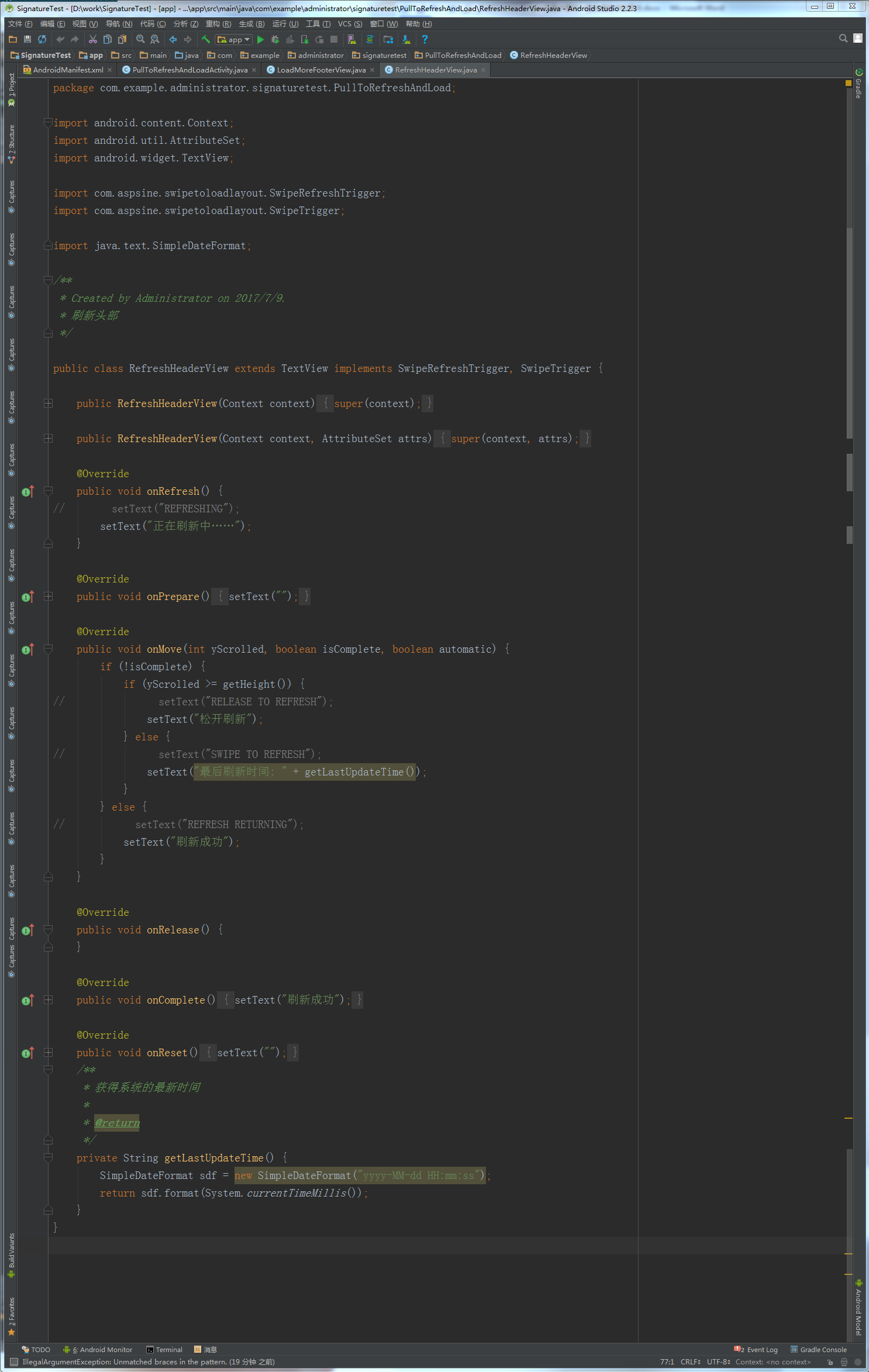
private String getLastUpdateTime() {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

return sdf.format(System.currentTimeMillis());

}

}



/\*\*

\* Created by Administrator on 2017/7/9.

\* 加载更多尾部

\*/

public class LoadMoreFooterView extends TextView implements SwipeTrigger, SwipeLoadMoreTrigger {

public LoadMoreFooterView(Context context) {

super(context);

}

public LoadMoreFooterView(Context context, AttributeSet attrs) {

super(context, attrs);

}

@Override

public void onLoadMore() {

// setText("LOADING MORE");

setText("加载更多……");

}

@Override

public void onPrepare() {

setText("");

}

@Override

public void onMove(int yScrolled, boolean isComplete, boolean automatic) {

if (!isComplete) {

if (yScrolled <= -getHeight()) {

// setText("RELEASE TO LOAD MORE");

setText("松开加载更多");

} else {

// setText("SWIPE TO LOAD MORE");

}

} else {

// setText("LOAD MORE RETURNING");

}

}

@Override

public void onRelease() {

// setText("LOADING MORE");

setText("加载更多……");

}

@Override

public void onComplete() {

// setText("COMPLETE");

}

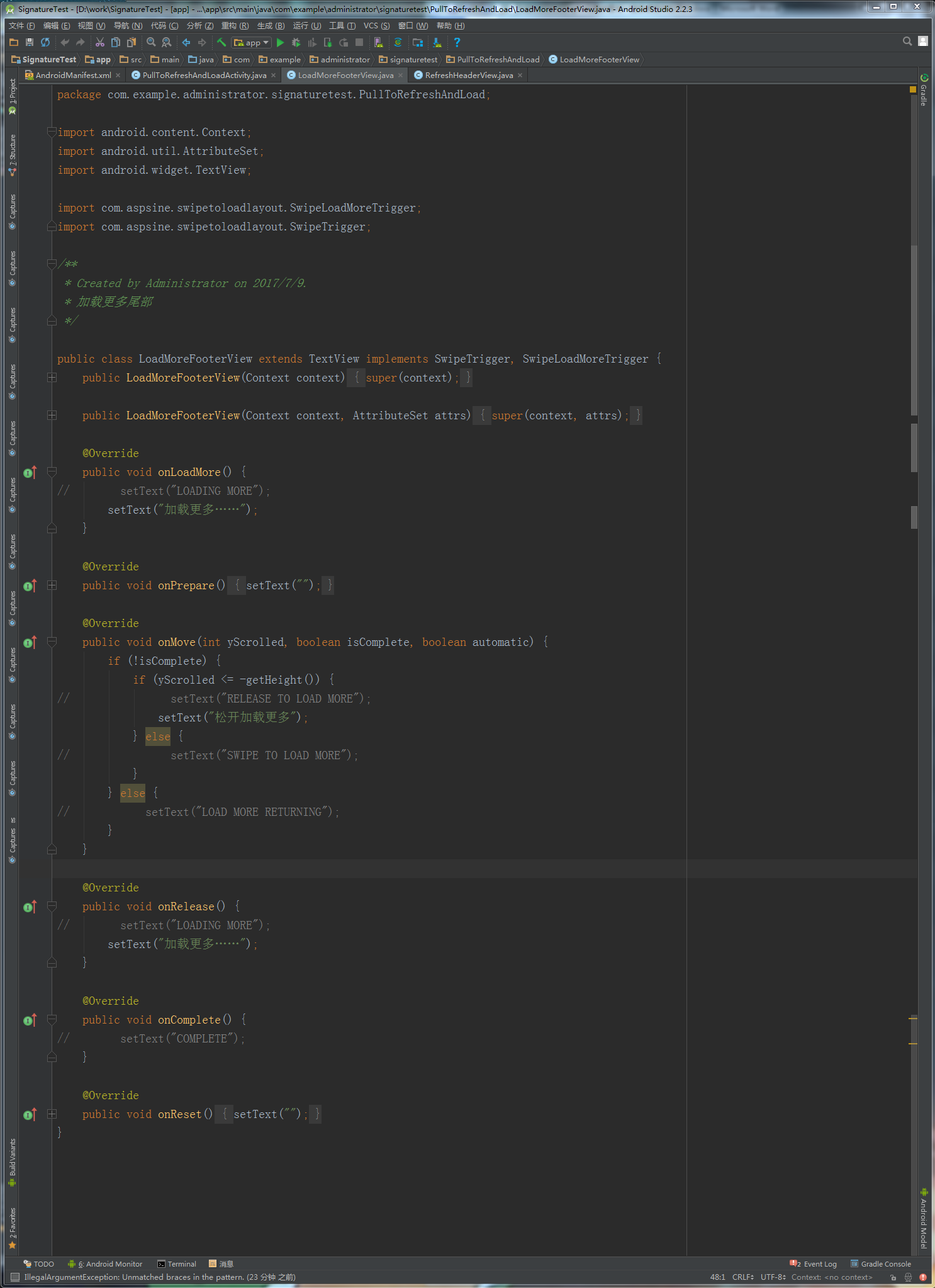
@Override

public void onReset() {

setText("");

}

}



### 5.在Activity中引用:

public class PullToRefreshAndLoadActivity extends AppCompatActivity implements OnRefreshListener, OnLoadMoreListener {

SwipeToLoadLayout swipeToLoadLayout;

WebView webView;

Handler handler;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

//取消标题栏

supportRequestWindowFeature(Window.FEATURE\_NO\_TITLE);

//设置布局

setContentView(R.layout.pull\_to\_refresh\_and\_load);

//获得刷新布局控件

swipeToLoadLayout = (SwipeToLoadLayout) findViewById(R.id.swipeToLoadLayout);

webView= (WebView) findViewById(R.id.swipe\_target);

//设置刷新监听

swipeToLoadLayout.setOnRefreshListener(this);

//设置加载更多监听

swipeToLoadLayout.setOnLoadMoreListener(this);

webView.setWebViewClient(new WebViewClient() {

@Override

public boolean shouldOverrideUrlLoading(WebView view, String url) {

view.loadUrl(url);

return true;

}

});

//WebChromeClient是辅助WebView处理Javascript的对话框，网站图标，网站title，加载进度等 :

webView.setWebChromeClient(new WebChromeClient() {

//处理alert弹出框

@Override

public boolean onJsAlert(WebView view, String url, String message, JsResult result) {

Toast.makeText(getApplicationContext(), message, Toast.LENGTH\_SHORT).show();

result.cancel();

return true;

}

});

webView.getSettings().setJavaScriptEnabled(true); //设置WebView属性,运行执行js脚本

//设置自适应屏幕，两者合用

webView.getSettings().setUseWideViewPort(true);//将图片调整到适合webview的大小

webView.getSettings().setLoadWithOverviewMode(true);// 缩放至屏幕的大小

webView.loadUrl("http://www.baidu.com");

// autoRefresh();自动刷新方法

//给Handler赋值

handler=new Handler();

}

//下拉刷新

@Override

public void onRefresh() {

swipeToLoadLayout.postDelayed(new Runnable() {

@Override

public void run() {

swipeToLoadLayout.setRefreshing(false);

// mAdapter.add("REFRESH:\n" + new Date());

handler.post(new Runnable(){//用handler转交UI处理操作

@Override public void run(){

//在这写操作webview的代码

//请不要在这里放耗时代码，否则会卡住UI线程

webView.loadUrl("http://www.qq.com/");

}

});

}

}, 2000);

}

//上拉加载更多

@Override

public void onLoadMore() {

swipeToLoadLayout.postDelayed(new Runnable() {

@Override

public void run() {

swipeToLoadLayout.setLoadingMore(false);

Toast.makeText(getApplicationContext(),"aa",Toast.LENGTH\_SHORT).show();

}

}, 2000);

}

//自动刷新

private void autoRefresh() {

swipeToLoadLayout.post(new Runnable() {

@Override

public void run() {

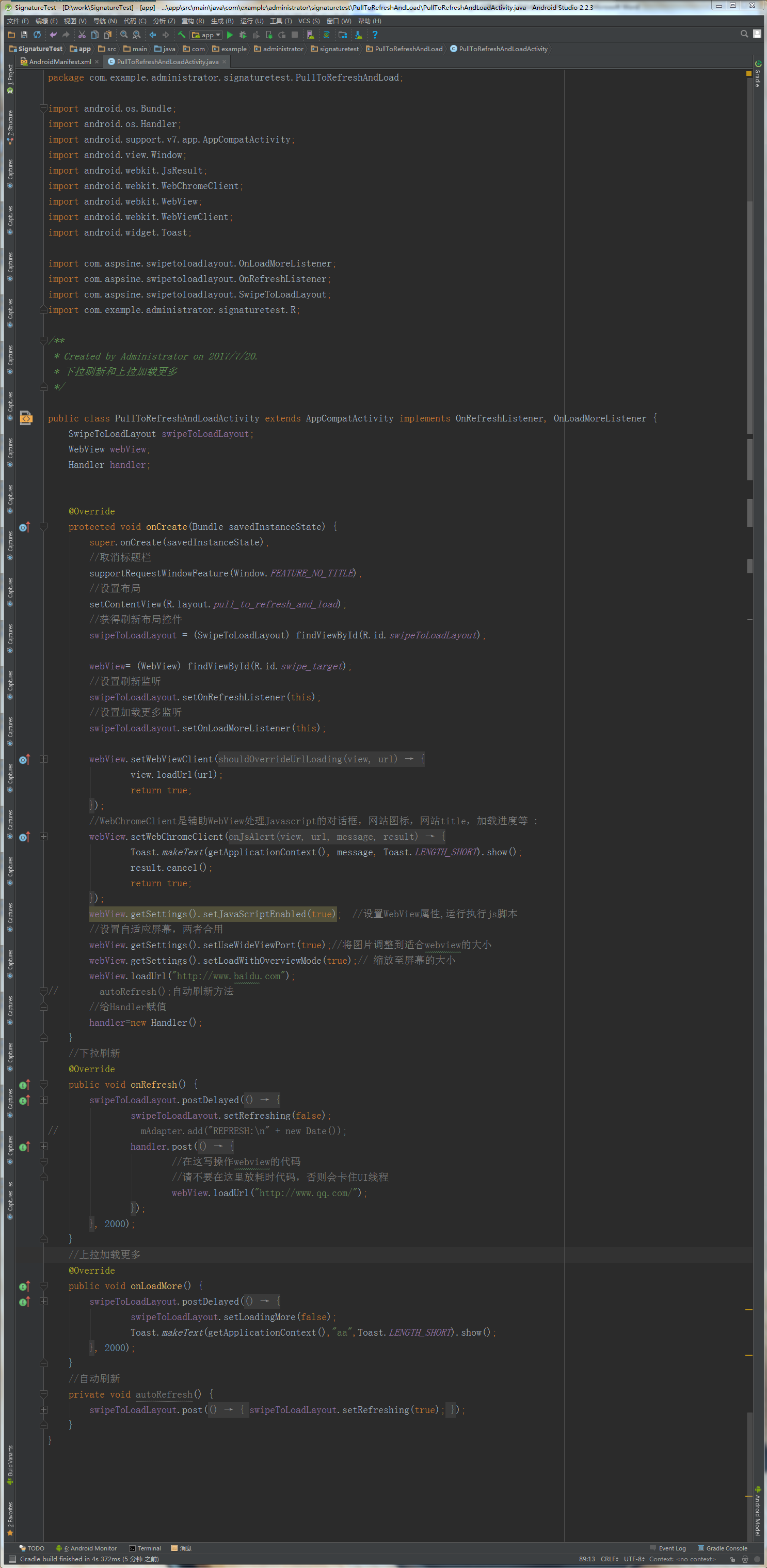
swipeToLoadLayout.setRefreshing(true);

}

});

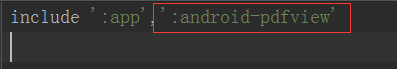
}

}



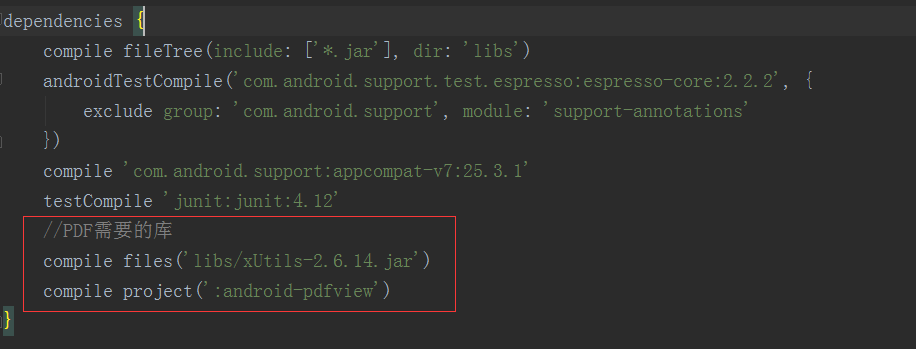
# 导在线阅读PDF控件（方法一）：

1.在settings.gradle中导入':android-pdfview'**，如：**



### 2.往APP目录下的libs目录导入xUtils-2.6.14jar包

### 3.在build.gradle文件的dependencies中导入compilefiles('libs/xUtils-2.6.14.jar') compileproject(':android-pdfview')



### 4.在需要的XML中引用：

**<com.joanzapata.pdfview.PDFView**

**android:id="@+id/pdfview"**

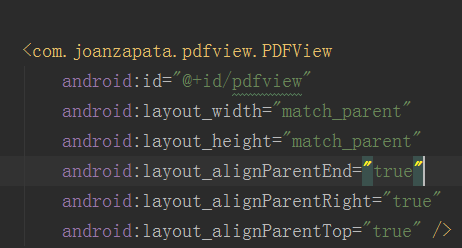
**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

**android:layout\_alignParentEnd="true"**

**android:layout\_alignParentRight="true"**

**android:layout\_alignParentTop="true" />**



### 5在需要的Activity中引用：

public class MainActivity extends AppCompatActivity implements OnPageChangeListener {  
 public static final String *Pdf\_Url*="http://123.56.187.157:8888/hlts/upload/1500882446308\_1.pdf";  
 private int pageNumber = 1;  
 private PDFView pdfview;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 getSupportActionBar().hide();//隐藏掉整个ActionBar，包括下面的Tabs  
 setContentView(R.layout.*activity\_pdfview*);  
 pdfview = (PDFView) findViewById(R.id.*pdfview*);  
 downloadPdf();  
 }  
 private void downloadPdf(){  
 try {  
 com.lidroid.xutils.HttpUtils http = new com.lidroid.xutils.HttpUtils();  
 http.download(*Pdf\_Url*, Environment.*getExternalStorageDirectory*()+"/provisional.pdf", true, false, new RequestCallBack<File>() {  
 @Override  
 public void onStart() {}  
 @Override  
 public void onLoading(long total, long current, boolean isUploading) {}  
 @Override  
 public void onFailure(HttpException error, String msg) {}  
 @Override  
 public void onSuccess(ResponseInfo<File> responseInfo) {  
 File file = new File(Environment.*getExternalStorageDirectory*()+"/provisional.pdf");  
 readPdf(file);  
 }  
 });  
 } catch (Exception e) {  
 Log.*d*("error", e.toString());  
 }  
 }  
 private void readPdf(File file) {  
 pdfview.fromFile(file)  
 .defaultPage(pageNumber)  
 .load();  
 }  
 //这个下载网络上的pdf到sd卡上的代码 然后进行本地读取 在失去焦点的时候删除下载的文件  
// 删除代码  
 @Override  
 protected void onPause() {  
 super.onPause();  
 File file = new File(Environment.*getExternalStorageDirectory*()+"/provisional.pdf");  
 if (file.exists()) {  
 file.delete();  
 }  
 }  
 //继承接口后可以继承的方法  
 @Override  
 public void onPageChanged(int page, int pageCount) {  
 pageNumber = page;  
 }  
 @Override  
 public void onBackPressed() {  
 super.onBackPressed();  
 }  
  
}

方法二：

1.在build.gradle的dependencie中引用 compile 'com.lidong.pdf:android\_pdf:1.0.1'

### 2.在需要引用的XML中引用：

<com.lidong.pdf.PDFView  
 android:id="@+id/pdfView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 />

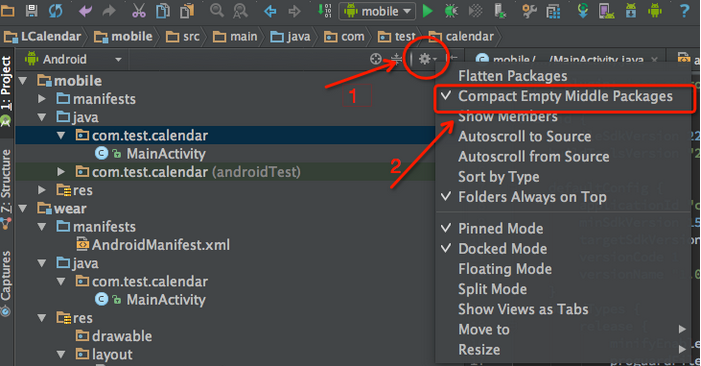
### 3.在Activity中的使用：

public class Download\_the\_PDF\_and\_display extends AppCompatActivity implements OnPageChangeListener  
 , OnLoadCompleteListener, OnDrawListener {  
 private PDFView pdfView;  
 private ProgressDialog pd;  
  
 @Override  
 protected void onCreate(@Nullable Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*pdf*);  
 pdfView = (PDFView) findViewById(R.id.*pdfView*);  
 displayFromFile1("http://123.56.187.157:8888/hlts/upload/1499049299321\_1.pdf", "1499049299321\_1.pdf");  
 //读取本地PDF  
// displayFromAssets("http://123.56.187.157:8888/hlts/upload/1499049299321\_1.pdf");  
 }  
 private void displayFromAssets(String assetFileName ) {  
 pdfView.fromAsset(assetFileName) //设置pdf文件地址  
 .defaultPage(1) //设置默认显示第1页  
 .onPageChange(this) //设置翻页监听  
 .onLoad(this) //设置加载监听  
 .onDraw(this) //绘图监听  
 .showMinimap(true) //pdf放大的时候，是否在屏幕的右上角生成小地图  
 .swipeVertical(true) //pdf文档翻页是否是垂直翻页，默认是左右滑动翻页  
 .enableSwipe(true) //是否允许翻页，默认是允许翻页  
 .load();  
 }  
 */\*\*  
 \* 获取打开网络的pdf文件  
 \*  
 \** ***@param*** *fileUrl  
 \** ***@param*** *fileName  
 \*/* private void displayFromFile1(String fileUrl, String fileName) {  
 showProgress();  
 pdfView.fileFromLocalStorage(this, this, this, fileUrl, fileName); //设置pdf文件地址  
  
 }  
  
 */\*\*  
 \* 翻页回调  
 \*  
 \** ***@param*** *page  
 \** ***@param*** *pageCount  
 \*/* @Override  
 public void onPageChanged(int page, int pageCount) {  
 /\*Toast.makeText(Download\_the\_PDF\_and\_display.this , "page= " + page +  
 " pageCount= " + pageCount , Toast.LENGTH\_SHORT).show();\*/  
 }  
  
 */\*\*  
 \* 加载完成回调  
 \*  
 \** ***@param*** *nbPages 总共的页数  
 \*/* @Override  
 public void loadComplete(int nbPages) {  
 Toast.*makeText*(Download\_the\_PDF\_and\_display.this, "加载完成" + nbPages, Toast.*LENGTH\_SHORT*).show();  
 hideProgress();  
 }  
  
 @Override  
 public void onLayerDrawn(Canvas canvas, float pageWidth, float pageHeight, int displayedPage) {  
 // Toast.makeText( MainActivity.this , "pageWidth= " + pageWidth + "  
 // pageHeight= " + pageHeight + " displayedPage=" + displayedPage , Toast.LENGTH\_SHORT).show();  
 }  
  
 */\*\*  
 \* 显示对话框  
 \*/* private void showProgress() {  
 pd = new ProgressDialog(Download\_the\_PDF\_and\_display.this);  
 pd.setProgressStyle(ProgressDialog.*STYLE\_SPINNER*);// 设置进度条风格，风格为圆形，旋转的  
 pd.setMessage("资源加载中,请稍候...");// 设置ProgressDialog提示信息  
 // 设置ProgressDialog 的进度条是否不明确 false 就是不设置为不明确  
 pd.setIndeterminate(false);  
 pd.setCancelable(false); // 设置ProgressDialog 是否可以按退回键取消  
 pd.show();  
 }  
  
 */\*\*  
 \* 关闭等待框  
 \*/* private void hideProgress() {  
 pd.dismiss();  
 }

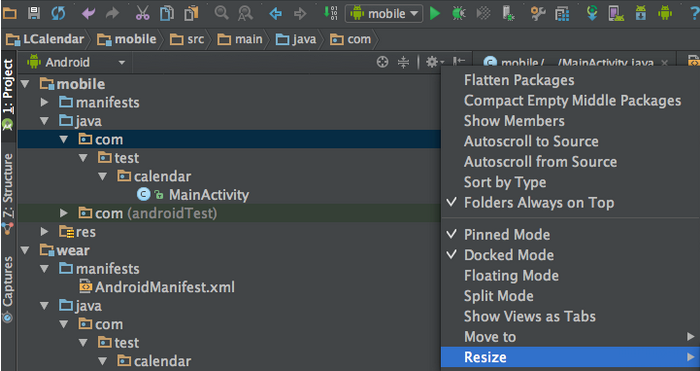
**注意：两种方法只能用一种，否则包会冲突（推荐使用第二种），两种方式都需要网络权限以及SD卡读写权限**

<!--网络权限-->  
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />  
<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

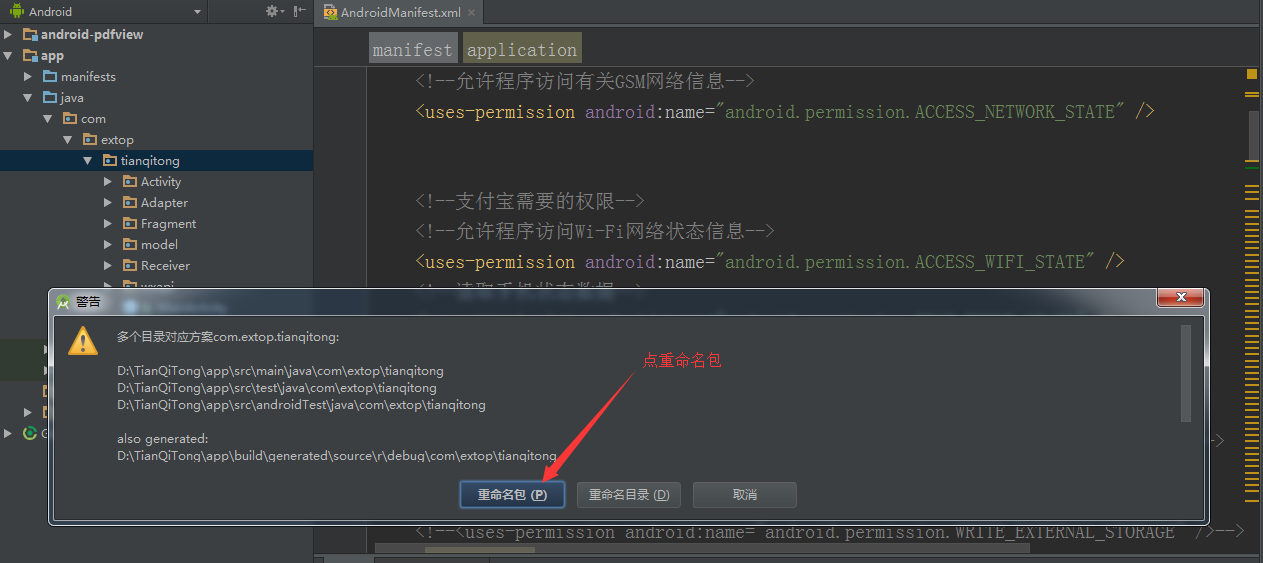
# Android Stuido 修改包名

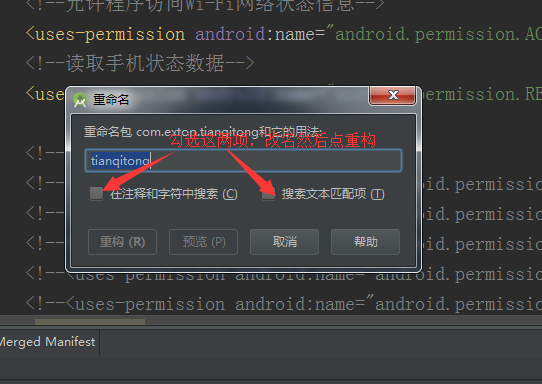


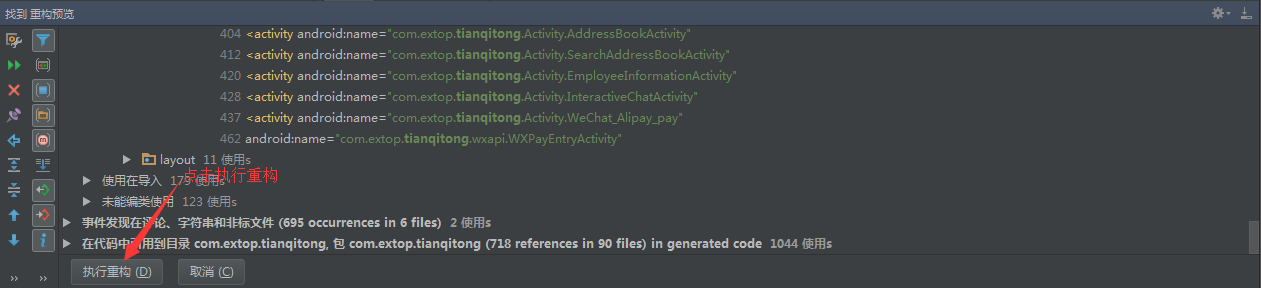
1.点击如一所示的图标，然后会出现2所示菜单，这个时候注意方框圈住的选项，默认是选中状态，我们把它取消，这时候可以看到工程的文件目录已经发生变化，这个时候变成如下图所示：

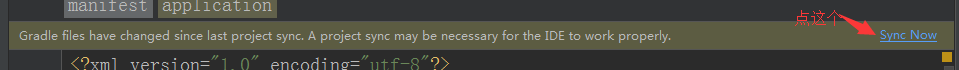


这个时候，我们就可以随意的修改包名了，选中想要修改的包名部分目录，Shift + F6 ,在弹出框选择Rename Package,稍等一会，可以看到包名已经发生了变化。

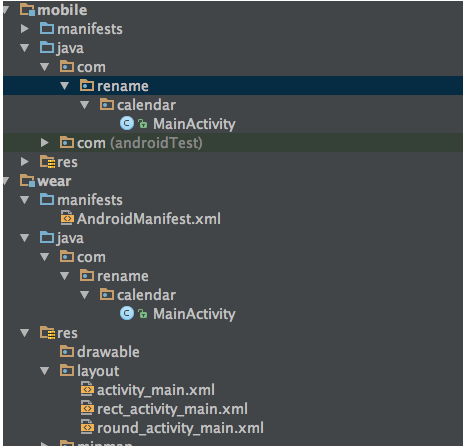








OK，下图为我把包名test部分改为rename.



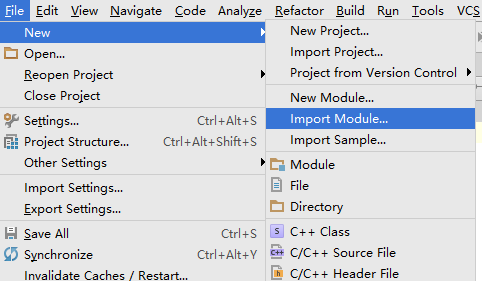
# AS导入CrosswalkWebViewModule

1.首先从官网（https://crosswalk-project.org/documentation/downloads.html）下载压缩包

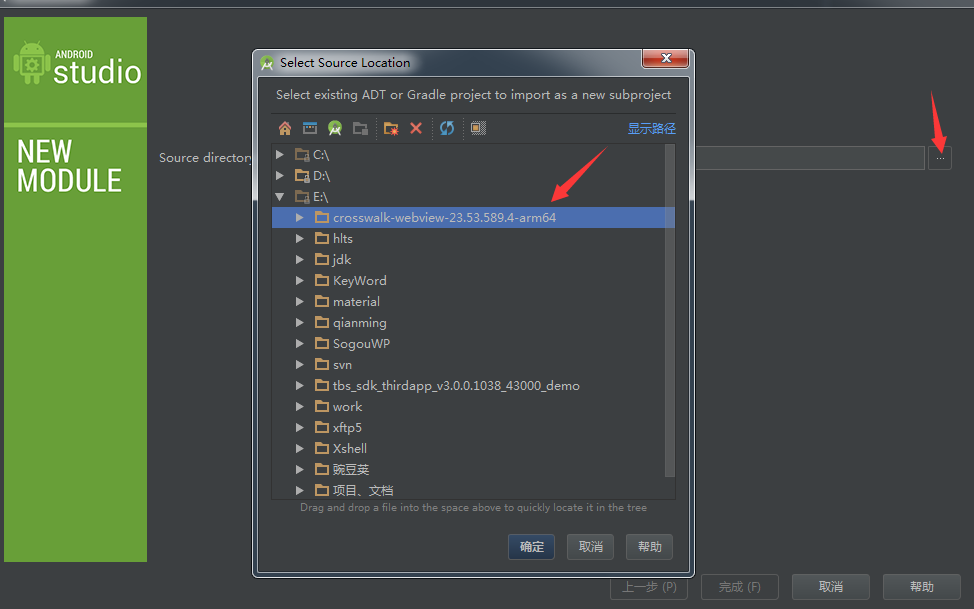
2.然后将下载好的压缩包解压

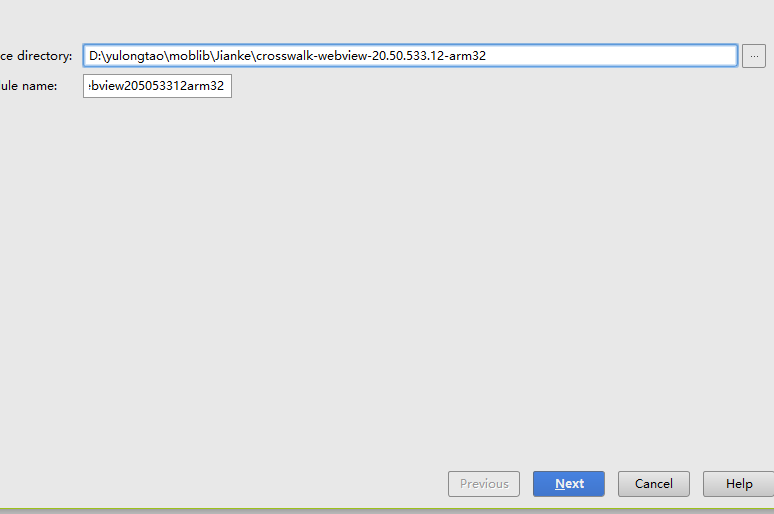
3.打开Android Studio 新建项目

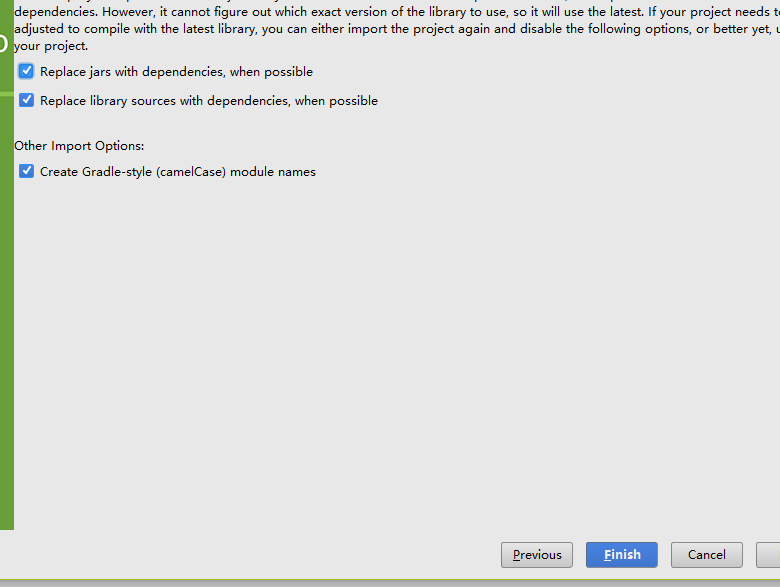
4.点击文件→new→Import Module



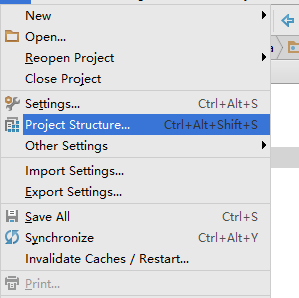
5. 选择解压的文件夹

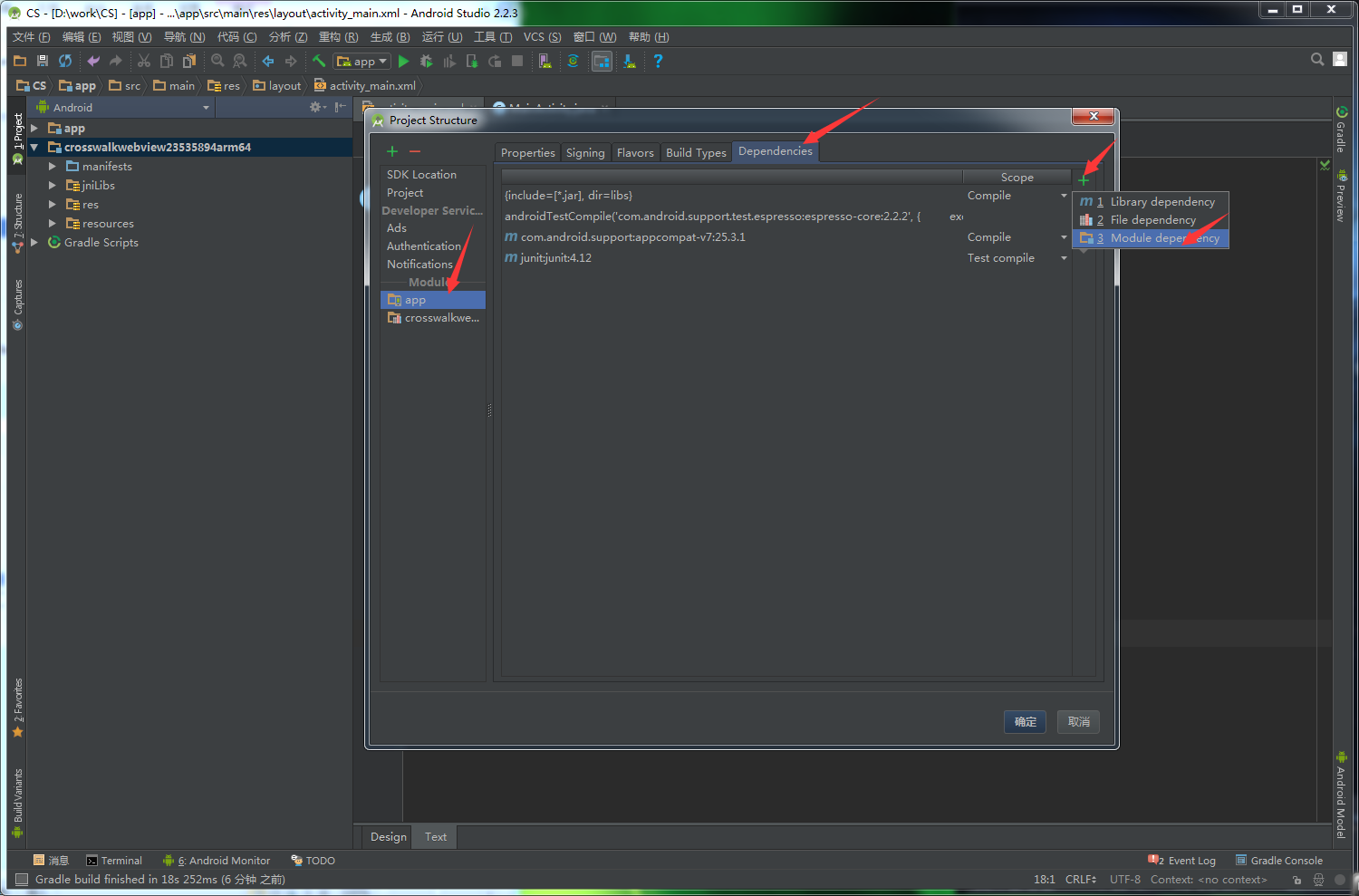


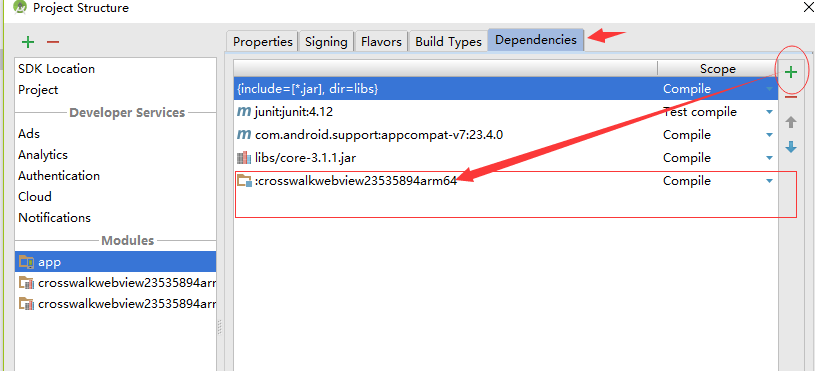


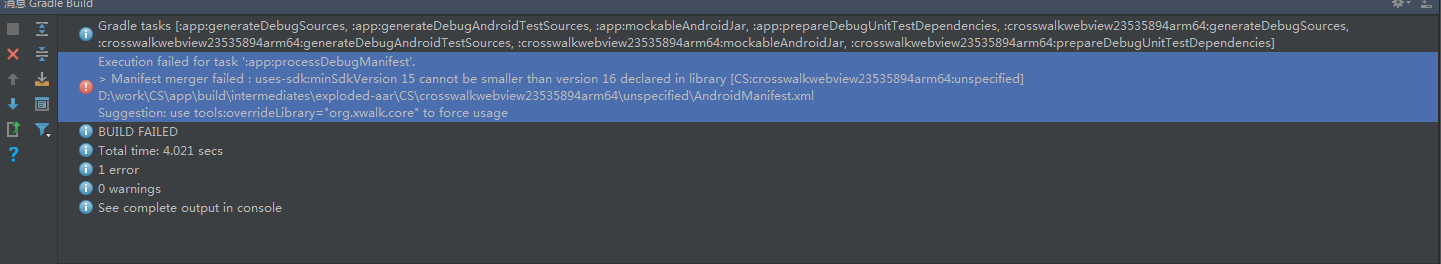


6.点击项目结构把新导入的Module与应用关联









点确定之后会报这个错，需要把项目build→minSdkVersion和crosswalkwebview的build→minSdkVersion改成一致的。

crosswalk-webviewx86（平板 笔记本）都用32位的兼容64

crosswalk-webview-arm64（手机）

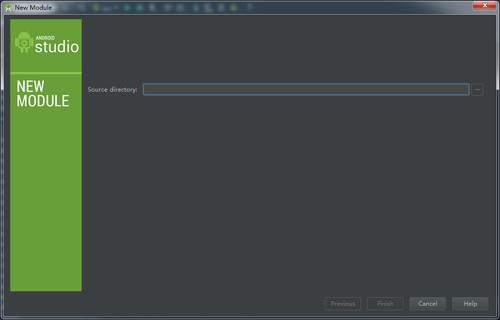
# AS接入融云即时通讯sdk

1.从官网(http://www.rongcloud.cn/downloads)下载sdk

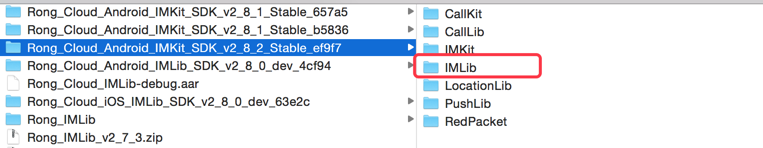
2.开发者在集成融云即时通讯、实时网络能力前，需前往融云官方网站注册创建融云开发者帐号。

3. 以 Module 形式导入前面下载的融云 SDK 里面的各个组件。

打开您的工程， File -> New -> Import Module

0

打开您从官网下载的融云 SDK，选择 IMLib。如图：



根据需要，以同样的步骤导入SDK里的其它组件： IMKit, CallKit, CallLib, RedPacket。

4.点击项目结构把新导入的Module与应用关联

# Android 7.0 权限变更

由于 Android 7.0 或更高版本的系统在国内手机市场上的占比不是很高，很多 Android 开发人员并没有做 7.0 适配工作，同时测试人员也容易忽视这方面的兼容问题。这导致 7.0 及以上版本的手机用户在使用到应用部分功能时可能出现 App 崩溃闪退。其中，大部分原因都是由项目中使用到 **file://** 类型的 URI 所引发的。本文我们便来一探究竟。

为了提高私有目录的安全性，防止应用信息的泄漏，从 Android 7.0 开始，应用私有目录的访问权限被做限制。具体表现为，开发人员不能够再简单地通过 **file://** URI 访问其他应用的私有目录文件或者让其他应用访问自己的私有目录文件。

同时，也是从 7.0 开始，Android SDK 中的 [StrictMode](https://developer.android.com/reference/android/os/StrictMode.html) 策略禁止开发人员在应用外部公开 **file://** URI。具体表现为，当我们在应用中使用包含 **file://** URI 的 Intent 离开自己的应用时，程序会发生故障。

开发中，如果我们在使用 **file://** URI 时忽视了这两条规定，将导致用户在 7.0 及更高版本系统的设备中使用到相关功能时，出现 **FileUriExposedException** 异常，导致应用出现崩溃闪退问题。而这两个过程的替代解决方案便是使用 FileProvider。

### FileProvider

作为四大组件之一的 ContentProvider，一直扮演着应用间共享资源的角色。这里我们要使用到的 FileProvider，就是 ContentProvider 的一个特殊子类，帮助我们将访问受限的 **file://** URI 转化为可以授权共享的 **content://** URI。

**第一步，注册一个 FileProvider**

作为系统四大组件之一的 ContentProvider，其子类FileProvider，也同样需要使用 元素在 Manifest 文件中添加注册信息，并按照要求设置相关属性值。

<application>

...

<provider

android:name="android.support.v4.content.FileProvider"

android:authorities="${applicationId}.yourname"

android:exported="false"

android:grantUriPermissions="true">

...

</provider>

...

</application>

其中，android:authorities 属性值是一个由 build.gradle 文件中的 **applicationId** 值和自定义的名称组成的 Uri 字符串（这样写是约定俗成的）。其他属性值使用如上固定值即可。

**第二步，添加共享目录**

在 res/xml 目录下新建一个 xml 文件，用于存放应用需要共享的目录文件。这个 xml 文件的内容类似这样：

<?xml version="1.0" encoding="utf-8"?>

<paths xmlns:android="http://schemas.android.com/apk/res/android">

<files-path name="my\_images" path="images/"/>

...

</paths>

元素必须包含一到多个子元素。这些子元素用于指定共享文件的目录路径，必须是这些元素之一：

* <files-path>：内部存储空间应用私有目录下的 files/ 目录，等同于 Context.getFilesDir() 所获取的目录路径；
* <cache-path>：内部存储空间应用私有目录下的 cache/ 目录，等同于 Context.getCacheDir() 所获取的目录路径；
* <external-path>：外部存储空间根目录，等同于 Environment.getExternalStorageDirectory() 所获取的目录路径；
* <external-files-path>：外部存储空间应用私有目录下的 files/ 目录，等同于 Context.getExternalFilesDir(null) 所获取的目录路径；
* <external-cache-path>：外部存储空间应用私有目录下的 cache/ 目录，等同于 Context.getExternalCacheDir()；

可以看出，这五种子元素基本涵盖内外存储空间所有目录路径，包含应用私有目录。同时，每个子元素都拥有 **name** 和 **path** 两个属性。

其中，path 属性用于指定当前子元素所代表目录下需要共享的子目录名称。注意：path 属性值不能使用具体的独立文件名，只能是目录名。

而 name 属性用于给 path 属性所指定的子目录名称取一个别名。后续生成 **content://** URI 时，会使用这个别名代替真实目录名。这样做的目的，很显然是为了提高安全性。

如果我们需要分享的文件位于同级别目录下不同的子目录中，就需要添加多个子元素逐一指定要分享的文件目录，或者共享他们通用的父目录也行。

添加完共享目录后，再在 <provider> 元素中使用 <meta-data> 元素将 res/xml 中的 path 文件与注册的 FileProvider 链接起来：

<provider

android:name="android.support.v4.content.FileProvider"

android:authorities="${applicationId}.yourname"

android:exported="false"

android:grantUriPermissions="true">

<meta-data

android:name="android.support.FILE\_PROVIDER\_PATHS"

android:resource="@xml/yourfilename" />

</provider>

**第三步，生成 Content URI**

在 Android 7.0 出现之前，我们通常使用 Uri.fromFile() 方法生成一个 File URI。这里，我们需要使用 FileProvider 类提供的公有静态方法 getUriForFile 生成 Content URI。比如：

Uri contentUri = FileProvider.getUriForFile(this,

BuildConfig.APPLICATION\_ID + ".myprovider", myFile);

需要传递三个参数。第二个参数便是 Manifest 文件中注册 FileProvider 时设置的 authorities 属性值，第三个参数为要共享的文件，并且这个文件一定位于第二步我们在 path 文件中添加的子目录里面。

举个例子：

String filePath = Environment.getExternalStorageDirectory() + "/images/"+System.currentTimeMillis()+".jpg";

File outputFile = new File(filePath);

if (!outputFile.getParentFile().exists()) {

outputFile.getParentFile().mkdir();

}

Uri contentUri = FileProvider.getUriForFile(this,

BuildConfig.APPLICATION\_ID + ".myprovider", outputFile);

生成的 Content URI 是这样的：

content://com.yifeng.samples.myprovider/my\_images/1493715330339.jpg

其中，构成 URI 的 host 部分为 <provider> 元素的 authorities 属性值（applicationId ＋ customname），path 片段 my\_images 为 res/xml 文件中指定的子目录别名（真实目录名为：images）。

**第四步，授予 Content URI 访问权限**

生成 Content URI 对象后，需要对其授权访问权限。授权方式有两种：

第一种方式，使用 Context 提供的 grantUriPermission(package, Uri, mode\_flags) 方法向其他应用授权访问 URI 对象。三个参数分别表示授权访问 URI 对象的其他应用包名，授权访问的 Uri 对象，和授权类型。其中，授权类型为 Intent 类提供的读写类型常量：

* FLAG\_GRANT\_READ\_URI\_PERMISSION
* FLAG\_GRANT\_WRITE\_URI\_PERMISSION

或者二者同时授权。这种形式的授权方式，权限有效期截止至发生设备重启或者手动调用 revokeUriPermission() 方法撤销授权时。

第二种方式，配合 Intent 使用。通过 setData() 方法向 intent 对象添加 Content URI。然后使用 setFlags() 或者 addFlags() 方法设置读写权限，可选常量值同上。这种形式的授权方式，权限有效期截止至其它应用所处的堆栈销毁，并且一旦授权给某一个组件后，该应用的其它组件拥有相同的访问权限。

**第五步，提供 Content URI 给其它应用**

拥有授予权限的 Content URI 后，便可以通过 startActivity() 或者 setResult() 方法启动其他应用并传递授权过的 Content URI 数据。当然，也有其他方式提供服务。

如果你需要一次性传递多个 URI 对象，可以使用 intent 对象提供的 setClipData() 方法，并且 setFlags() 方法设置的权限适用于所有 Content URIs。

### 常见使用场景

前面介绍的内容都是理论部分，在 [开发者官方 FileProvider 部分](https://developer.android.com/reference/android/support/v4/content/FileProvider.html) 都有所介绍。接下来我们看看，实际开发一款应用的过程中，会经常遇见哪些 FileProvider 的使用场景。

**自动安装文件**

版本更新完成时打开新版本 apk 文件实现自动安装的功能，应该是最常见的使用场景，也是每个应用必备功能之一。常见操作为，通知栏显示下载新版本完毕，用户点击或者监听下载过程自动打开新版本 apk 文件。适配 Android 7.0 版本之前，我们代码可能是这样：

File apkFile = new File(getExternalFilesDir(Environment.DIRECTORY\_DOWNLOADS), "app\_sample.apk");

Intent installIntent = new Intent(Intent.ACTION\_VIEW);

installIntent.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

installIntent.setDataAndType(Uri.fromFile(apkFile), "application/vnd.android.package-archive");

startActivity(installIntent);

现在为了适配 7.0 及以上版本的系统，必须使用 Content URI 代替 File URI。

在 res/xml 目录下新建一个 file\_provider\_paths.xml 文件（文件名自由定义），并添加子目录路径信息：

<?xml version="1.0" encoding="utf-8"?>

<paths xmlns:android="http://schemas.android.com/apk/res/android">

<external-files-path name="my\_download" path="Download"/>

</paths>

然后在 Manifest 文件中注册 FileProvider 对象，并链接上面的 path 路径文件：

<provider

android:name="android.support.v4.content.FileProvider"

android:authorities="com.yifeng.samples.myprovider"

android:exported="false"

android:grantUriPermissions="true">

<meta-data

android:name="android.support.FILE\_PROVIDER\_PATHS"

android:resource="@xml/file\_provider\_paths"/>

</provider>

修改 java 代码，根据 File 对象生成 Content URI 对象，并授权访问：

File apkFile = new File(getExternalFilesDir(Environment.DIRECTORY\_DOWNLOADS), "app\_sample.apk");

Uri apkUri = FileProvider.getUriForFile(this,

BuildConfig.APPLICATION\_ID+".myprovider", apkFile);

Intent installIntent = new Intent(Intent.ACTION\_VIEW);

installIntent.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

installIntent.addFlags(Intent.FLAG\_GRANT\_READ\_URI\_PERMISSION);

installIntent.setDataAndType(apkUri, "application/vnd.android.package-archive");

startActivity(installIntent);

如此这般，便完成了应用中调用系统功能打开 apk 文件的 7.0 适配工作。

**调用系统拍照**

调用系统拍照功能时也需要传递一个 Uri 对象，用于保存图片至指定目录，这里也需要适配 7.0 版本。其他步骤不再赘述，核心 java 代码如下（路径不同，注意添加 res/xml 中的 path 文件子目录）：

String filePath = Environment.getExternalStorageDirectory() + "/images/"+System.currentTimeMillis()+".jpg";

File outputFile = new File(filePath);

if (!outputFile.getParentFile().exists()) {

outputFile.getParentFile().mkdir();

}

Uri contentUri = FileProvider.getUriForFile(this,

BuildConfig.APPLICATION\_ID + ".myprovider", outputFile);

Intent intent = new Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);

intent.putExtra(MediaStore.EXTRA\_OUTPUT, contentUri);

startActivityForResult(intent, REQUEST\_TAKE\_PICTURE);

**调用系统裁剪**

调用系统裁剪的过程中涉及到两个 Uri 对象：inputUri 和 outputUri，较为复杂一些。通常，调用系统裁剪的来源为调用系统拍照或选择系统相册。前者返回的是一个 File URI 对象，后者返回的是一个 Content URI 对象。作为裁剪源，我们要做的就是对其做进一步处理。但是不能像上面那样使用 getUriForFile() 方法，这个并不难理解，因为如果是选择系统相册所得的图片，本身也不一定属于我们自己的应用。正确处理方式是这样：

private Uri getImageContentUri(String path){

Cursor cursor = getContentResolver().query(MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI,

new String[]{MediaStore.Images.Media.\_ID},

MediaStore.Images.Media.DATA + "=? ",

new String[]{path}, null);

if (cursor != null && cursor.moveToFirst()) {

int id = cursor.getInt(cursor.getColumnIndex(MediaStore.Images.Media.\_ID));

Uri baseUri = Uri.parse("content://media/external/images/media");

return Uri.withAppendedPath(baseUri, ""+id);

}else {

ContentValues contentValues = new ContentValues(1);

contentValues.put(MediaStore.Images.Media.DATA, path);

return getContentResolver().insert(MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI, contentValues);

}

}

拿到正确的 Content URI 后，作为 inputUri，传递给 Intent 对象：

Intent intent = new Intent("com.android.camera.action.CROP");

intent.addFlags(Intent.FLAG\_GRANT\_WRITE\_URI\_PERMISSION | Intent.FLAG\_GRANT\_READ\_URI\_PERMISSION);

intent.setDataAndType(inputUri, "image/\*");

intent.putExtra("crop", "true");

intent.putExtra("aspectX", 1);

intent.putExtra("aspectY", 1);

intent.putExtra(MediaStore.EXTRA\_OUTPUT, Uri.fromFile(outputFile));

startActivityForResult(intent, REQUEST\_PICK);

**注意：**这里的 outputUri 并没有改变，仍然使用的是 Uri.fromFile() 方法获取的 File URI 类型！这是很奇怪的一点，但是不得不这么做。事实上，使用这种方式调用系统裁剪功能本身就是有问题的！常见问题如：在部分机型上，调用系统裁剪并返回前一个页面时，在 onActivityResult() 方法中得到的 resultCode 值不等于 RESULT\_OK。Crop Intent 在官方文档中本来就无迹可寻，本身就是一种不推荐的用法！取而代之的是，我们可以使用 GitHub 上的一些开源库实现应用内的图片裁剪功能，比如 [uCrop](https://github.com/Yalantis/uCrop)、[cropper](https://github.com/edmodo/cropper" \t "_blank) 等。

### 历史版本问题

说了这么多，还有一个大家比较关心的问题就是：哪些已经上线的旧版本应用没有做 7.0 适配工作怎么办？关于这个问题，Google 已经提前帮我们想好解决方案啦。

还记得 6.0 运行时权限问题吗？如果你不想处理运行时权限事宜的话，只需要在 build.gradle 文件中将 targetSdkVersion 的值设为 23 以下即可。

同样的，只要 targetSdkVersion 值小于 24，File URI 的使用依旧可以出现在 7.0 及以上版本的设备中。不过需要注意的是，如前面所述，调用系统裁剪功能比较特殊，可能会出现一些问题。