**package** com.zimi.bookstore.common.base  
  
**import** android.annotation.SuppressLint  
**import** android.os.Bundle  
**import** android.support.v7.app.AppCompatActivity  
**import** com.jaeger.library.StatusBarUtil  
  
*/\*\*  
 \** ***@author*** *崔兴旺  
 \** ***@description*** *\** ***@date*** *2019/7/19 23:54  
 \*/***open class** BaseActivity : AppCompatActivity() {  
  
 */\*\*  
 \* 在执行super.onCreate之前执行的回调函数  
 \*/* **open fun** onPreCreate() {  
  
 }  
  
 **override fun** onCreate(savedInstanceState: Bundle?) {  
 onPreCreate()  
 **super**.onCreate(savedInstanceState)  
  
 **if**(isNeedInitStatusNavigationBar()){  
 initStatusNavigationBar()  
 }  
 }  
  
 @SuppressLint(**"MissingSuperCall"**)  
 **override fun** onSaveInstanceState(outState: Bundle) {  
  
 }  
  
 **open fun** isNeedInitStatusNavigationBar():Boolean{  
 **return true** }  
  
 **private fun** initStatusNavigationBar(){  
 StatusBarUtil.setTransparentForImageView(**this**, **null**)  
 }  
  
  
  
}

**package** com.zimi.bookstore.common.base  
  
**import** android.databinding.DataBindingUtil  
**import** android.databinding.ViewDataBinding  
**import** android.graphics.drawable.ColorDrawable  
**import** android.os.Bundle  
**import** android.support.v4.content.res.ResourcesCompat  
**import** com.alick.holderview.HolderView  
**import** com.alick.holderview.api.IFailView  
**import** com.alick.holderview.api.OnReloadListener  
**import** com.zimi.bookstore.common.R  
  
  
*/\*\*  
 \** ***@author*** *崔兴旺  
 \** ***@title****:  
 \** ***@description*** *\** ***@date*** *2019/4/9 12:00  
 \*/***abstract class** BaseBindingActivity<Binding : ViewDataBinding> : BaseActivity(), IViewHelper, OnReloadListener, IFailView {  
  
 **lateinit var** mBinding: Binding  
 **var** mHolderView: HolderView? = **null** */\*\*  
 \* 获取布局ID  
 \*  
 \** ***@return*** *\*/* **protected abstract val** layoutId: Int  
  
 **override fun** onCreate(savedInstanceState: Bundle?) {  
 **super**.onCreate(savedInstanceState)  
 mBinding = DataBindingUtil.setContentView(**this**, layoutId)  
 window.setBackgroundDrawable(ColorDrawable(ResourcesCompat.getColor(resources,R.color.activity\_bg,theme)))  
 mHolderView = findViewById(R.id.id\_holder\_view)  
 mHolderView?.setOnReloadListener(**this**)  
 init()  
 initData()  
 initListener()  
 initViews()  
 }  
  
 **protected open fun** init() {  
  
 }  
  
 */\*\*  
 \* 设置错误信息  
 \*  
 \** ***@param errorMsg*** *\*/* **override fun** setErrorMsg(errorMsg: String) {  
 **if** (mHolderView != **null**) {  
 mHolderView!!.setErrorMsg(errorMsg)  
 }  
 }  
  
 */\*\*  
 \* 设置错误图片  
 \*  
 \** ***@param errorImgResId*** *\*/* **override fun** setErrorImg(errorImgResId: Int) {  
 **if** (mHolderView != **null**) {  
 mHolderView!!.setErrorImg(errorImgResId)  
 }  
 }  
  
  
}

**package** com.zimi.bookstore.common.base  
  
**import** android.databinding.DataBindingUtil  
**import** android.databinding.ViewDataBinding  
**import** android.os.Bundle  
**import** android.view.LayoutInflater  
**import** android.view.View  
**import** android.view.ViewGroup  
  
**import** com.alick.holderview.HolderView  
**import** com.alick.holderview.api.IFailView  
**import** com.alick.holderview.api.OnReloadListener  
**import** com.zimi.bookstore.common.R  
  
*/\*\*  
 \** ***@author*** *崔兴旺  
 \** ***@title****:  
 \** ***@description*** *\** ***@date*** *2019/4/16 11:33  
 \*/***abstract class** BaseBindingFragment<Binding : ViewDataBinding> : BaseFragment(), IViewHelper, OnReloadListener, IFailView {  
 **lateinit var** mBinding: Binding  
 **var** mHolderView: HolderView? = **null** */\*\*  
 \* 获取布局ID  
 \*  
 \** ***@return*** *\*/* **protected abstract val** layoutId: Int  
  
 **override fun** onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {  
 mBinding = DataBindingUtil.inflate(inflater, layoutId, container, **false**)  
 **return** mBinding.root  
 }  
  
 **override fun** onActivityCreated(savedInstanceState: Bundle?) {  
 **super**.onActivityCreated(savedInstanceState)  
  
 **val** root = mBinding.root  
 **if** (root **is** HolderView) {  
 mHolderView = root  
 } **else** {  
 mHolderView = mBinding.root.findViewById(R.id.id\_holder\_view)  
 }  
  
 mHolderView?.setOnReloadListener(**this**)  
  
 init()  
 initData()  
 initListener()  
 initViews()  
 }  
  
 **protected open fun** init() {  
  
 }  
  
 */\*\*  
 \* 设置错误信息  
 \*  
 \** ***@param errorMsg*** *\*/* **override fun** setErrorMsg(errorMsg: String) {  
 mHolderView?.setErrorMsg(errorMsg)  
 }  
  
 */\*\*  
 \* 设置错误图片  
 \*  
 \** ***@param errorImgResId*** *\*/* **override fun** setErrorImg(errorImgResId: Int) {  
 mHolderView?.setErrorImg(errorImgResId)  
 }  
}

**package** com.zimi.bookstore.common.base;  
  
**import** com.alick.refreshlayoutlibrary.ListHolderView;  
**import** com.alick.refreshlayoutlibrary.WySmartRefreshLayout;  
**import** com.zimi.bookstore.common.base.adapter.BaseAdapter;  
  
**import** java.util.List;  
  
*/\*\*  
 \** ***@author*** *崔兴旺  
 \** ***@createTime*** *2019/6/24 14:27  
 \** ***@description*** *\*/***public interface** IBindingListApi<Adapter **extends** BaseAdapter> **extends** WySmartRefreshLayout.OnWyRefreshListener, WySmartRefreshLayout.OnWyLoadMoreListener {  
 **void** initHolderView();  
  
 */\*\*  
 \* 刷新数据  
 \*  
 \** ***@param newData*** *\** ***@param isSuccess*** *\*/* **void** updateData(List newData, **boolean** isSuccess);  
  
 */\*\*  
 \* 是否没有更多数据了  
 \*  
 \** ***@param pageSize*** *\** ***@return*** *\*/* **boolean** isNoMoreData(**int** pageSize, List<?> newData);  
  
 */\*\*  
 \* 是否禁用下拉刷新  
 \*  
 \** ***@return*** *\*/* **boolean** isDisableRefresh();  
  
 */\*\*  
 \* 是否禁用加载更多  
 \*  
 \** ***@return*** *\*/* **boolean** isDisableLoadMore();  
  
 */\*\*  
 \* 获取自定义的每页请求数据量  
 \** ***@return*** *\*/* **int** getCustomPageSize();  
  
 */\*\*  
 \* 重置页码  
 \*/* **void** resetPageNum();  
  
 */\*\*  
 \* 获取适配器  
 \*  
 \** ***@return*** *\*/* Adapter getAdapter();  
  
 */\*\*  
 \* 获取ListHolderView  
 \*  
 \** ***@return*** *\*/* ListHolderView getListHolderView();  
}

**package** com.zimi.bookstore.common.base  
  
*/\*\*  
 \** ***@author*** *崔兴旺  
 \** ***@package*** *com.alick.mvvmlearn.base  
 \** ***@title****:  
 \** ***@description*** *\** ***@date*** *2019/4/9 13:22  
 \*/***interface** IViewHelper {  
 */\*\*  
 \* 初始化数据  
 \*/* **fun** initData()  
  
 */\*\*  
 \* 初始化监听  
 \*/* **fun** initListener()  
  
 */\*\*  
 \* 初始化视图  
 \*/* **fun** initViews()  
  
  
}

*/\*\*  
 \** ***@author*** *崔兴旺  
 \** ***@description*** *\** ***@date*** *2019/6/16 1:36  
 \*/***class** BaseFailResponse<Data>(msg: String?) : BaseResponse<Data>(**null**, msg, -1) {  
  
}

**package** com.zimi.bookstore.common.bean  
  
*/\*\*  
 \** ***@author*** *崔兴旺  
 \** ***@title****:  
 \** ***@description*** *\** ***@date*** *2019/7/21 15:31  
 \*/***open class** BaseResponse<Data>(**var data**: Data?, **var msg**: String?, **var code**: Int)

**package** com.zimi.bookstore.common.bean  
  
*/\*\*  
 \** ***@author*** *崔兴旺  
 \** ***@description*** *\** ***@date*** *2019/6/16 1:33  
 \*/***class** BaseSuccessResponse<Data>(data: Data) : BaseResponse<Data>(data, **""**,0)

**package** com.zimi.bookstore.common.utils;  
  
**import** android.text.TextUtils;  
  
**import** com.alick.utilslibrary.BLog;  
  
**import** java.io.IOException;  
**import** java.nio.charset.Charset;  
**import** java.nio.charset.UnsupportedCharsetException;  
  
**import** okhttp3.Headers;  
**import** okhttp3.Interceptor;  
**import** okhttp3.MediaType;  
**import** okhttp3.Request;  
**import** okhttp3.RequestBody;  
**import** okhttp3.Response;  
**import** okhttp3.ResponseBody;  
**import** okio.Buffer;  
**import** okio.BufferedSource;  
  
*/\*\*  
 \* 抽象的http拦截器  
 \* Created by cxw on 2016/9/30.  
 \*/***public abstract class** AbstractHttpInterceptor **implements** Interceptor {  
 **private static final** Charset UTF8 = Charset.forName(**"UTF-8"**);  
*// private boolean enableDebug;* **private** java.lang.String TAG = **"AbstractHttpInterceptor"**;  
  
 **public** AbstractHttpInterceptor() {  
*// this.enableDebug = enableDebug;* }  
  
  
 **public void** addHeader(Request.Builder builder) {  
 *//默认空实现* }  
  
 @Override  
 **public** Response intercept(Chain chain) **throws** IOException {  
 Request.Builder builder = chain.request().newBuilder();  
 addHeader(builder);  
 Request request = builder.build();  
 Response response = chain.proceed(request);  
  
  
 RequestBody requestBody = request.body();  
  
 **if** (BLog.debug) {  
 Headers headers = request.headers();  
 **int** headerSize = headers.size();  
 StringBuffer sb = **new** StringBuffer();  
 sb.append(**" \n++++++++++++++++++++++++++++++++++++ OKHttp request log ++++++++++++++++++++++++++++++++++++\n"**);  
 sb.append(**"----------请求头----------\n"**);  
 **for** (**int** i = 0; i < headerSize; i++) {  
 sb.append(headers.name(i)).append(**":"**).append(headers.value(i)).append(**"\n"**);  
 }  
  
  
 sb.append(**"url:"**).append(request.url()).append(**"\n"**);  
 Charset charset = UTF8;  
 **if** (requestBody != **null**) {  
 sb.append(**"Content-Type:"**).append(requestBody.contentType()).append(**"\n"**);  
  
 Buffer buffer = **new** Buffer();  
 requestBody.writeTo(buffer);  
  
 MediaType contentType = requestBody.contentType();  
 **if** (contentType != **null**) {  
 charset = contentType.charset(UTF8);  
 }  
 String requestBodyLog = buffer.readString(charset);  
 sb.append(**"----------请求体----------\n"**).append((!TextUtils.isEmpty(requestBodyLog) ? requestBodyLog : **"无"**)).append(**"\n"**);  
 }  
 sb.append(**"++++++++++++++++++++++++++++++++++++ OKHttp response log ++++++++++++++++++++++++++++++++++++\n"**);  
 sb.append(**"----------响应体----------\n"**);  
  
*// 响应体部分============================================* ResponseBody responseBody = response.body();  
 **long** contentLength = responseBody.contentLength();  
 sb.append(**"responseCode:"**).append(response.code()).append(**" "**).append(response.message()).append(**"\n"**);  
  
 BufferedSource source = responseBody.source();  
 source.request(Long.MAX\_VALUE); *// Buffer the entire body.* Buffer responseBuffer = source.buffer();  
  
 MediaType responseContentType = responseBody.contentType();  
 **if** (responseContentType != **null**) {  
 **try** {  
 charset = responseContentType.charset(UTF8);  
 } **catch** (UnsupportedCharsetException e) {  
 BLog.e(TAG, **"Couldn't decode the response body; charset is likely malformed."**);  
 BLog.e(TAG, **"<-- END HTTP"**);  
 **return** response;  
 }  
 }  
  
 **if** (contentLength != 0) {  
 String jsonResult = sb.append(responseBuffer.clone().readString(charset)).toString();  
 BLog.i(StringUtils.decodeUnicode(jsonResult));  
 } **else** {  
 BLog.i(sb.toString());  
 }  
 }  
  
  
 **return** response;  
 }  
  
 */\*========================set/get方法-begin========================\*/  
  
 /\*========================set/get方法-end========================\*/*}

**package** com.zimi.bookstore.common.utils;  
  
**import** android.app.ActivityManager;  
**import** android.content.Context;  
**import** android.content.Intent;  
**import** android.net.Uri;  
**import** android.os.Build;  
  
**import** com.alick.utilslibrary.BLog;  
  
**import** java.io.File;  
**import** java.util.List;  
  
*//跟App相关的辅助类***public class** AppUtils {  
 **private static final** String ***TAG*** = **"AppUtils"**;  
  
 */\*\*  
 \* 安装apk(已兼容Android7.0)  
 \*  
 \** ***@param context*** *上下文  
 \** ***@param filePath*** *文件路径  
 \*/* **public static void** installApkFile(Context context, String filePath) {  
 *installApkFile*(context,**new** File(filePath));  
 }  
  
 */\*\*  
 \* 安装apk(已兼容Android7.0)  
 \*  
 \** ***@param context*** *上下文  
 \** ***@param file*** *文件对象  
 \*/* **public static void** installApkFile(Context context, File file) {  
 context.startActivity(*buildInstallApkIntent*(context,file));  
 }  
  
 */\*\*  
 \* 安装apk(已兼容Android7.0)  
 \** ***@param context*** *上下文  
 \** ***@param uri*** *uri对象  
 \*/* **public static void** installApkFile(Context context,Uri uri) {  
 context.startActivity(*buildInstallApkIntent*(uri));  
 }  
  
 */\*\*  
 \* 构建一个用于安装apk的intent对象  
 \** ***@param context*** *上下文  
 \** ***@param filePath*** *文件路径  
 \** ***@return*** *\*/* **public static** Intent buildInstallApkIntent(Context context, String filePath){  
 **return** *buildInstallApkIntent*(context,**new** File(filePath));  
 }  
  
 */\*\*  
 \* 构建一个用于安装apk的intent对象  
 \** ***@param uri*** *文件对象  
 \** ***@return*** *\*/* **public static** Intent buildInstallApkIntent(Uri uri){  
 Intent intent = **new** Intent(Intent.***ACTION\_VIEW***);  
 intent.setFlags(Intent.***FLAG\_ACTIVITY\_NEW\_TASK***);  
 **if** (Build.VERSION.***SDK\_INT*** >= Build.VERSION\_CODES.***N***) {  
 *//添加这一句表示对目标应用临时授权该Uri所代表的文件* intent.addFlags(Intent.***FLAG\_GRANT\_READ\_URI\_PERMISSION***);  
 }  
 intent.setDataAndType(uri, **"application/vnd.android.package-archive"**);  
 **return** intent;  
 }  
  
  
 */\*\*  
 \* 构建一个用于安装apk的intent对象  
 \** ***@param context*** *上下文  
 \** ***@param file*** *文件对象  
 \** ***@return*** *\*/* **public static** Intent buildInstallApkIntent(Context context, File file){  
 **return** *buildInstallApkIntent*(UriUtils.*getUriCompatibleN*(context, file));  
 }  
  
  
 **public static** <Activity **extends** android.app.Activity> **boolean** isTopActivity(Context context, Class<Activity> clazz) {  
 **return** *isTopActivity*(context, clazz.getName());  
 }  
  
 **public static boolean** isTopActivity(Context context, String activitySimpleName) {  
 **if** (context == **null** || (activitySimpleName == **null** || activitySimpleName.trim().length() == 0)) {  
 **return false**;  
 }  
  
 ActivityManager activityManager = (ActivityManager) context.getSystemService(Context.***ACTIVITY\_SERVICE***);  
 List<ActivityManager.RunningTaskInfo> tasksInfo = activityManager.getRunningTasks(1);  
 **if** (tasksInfo.isEmpty()) {  
 **return false**;  
 }  
 **try** {  
 String shortClassName = tasksInfo.get(0).**topActivity**.getClassName();  
 BLog.*i*(***TAG***,**"栈顶Activity名称:"** + shortClassName);  
 **return** activitySimpleName.contains(shortClassName);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
  
  
  
  
  
  
}

**package** com.zimi.bookstore.common.utils;  
  
**import** android.content.Context;  
**import** android.content.pm.PackageInfo;  
**import** android.content.pm.PackageManager;  
**import** android.text.TextUtils;  
**import** android.widget.EditText;  
  
**import** java.lang.reflect.Field;  
**import** java.lang.reflect.InvocationTargetException;  
**import** java.lang.reflect.Method;  
**import** java.math.RoundingMode;  
**import** java.text.DecimalFormat;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Random;  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
  
*/\*\*  
 \* 公共的工具类  
 \*/***public class** CommonUtils {  
  
 **private** CommonUtils() {  
 }  
  
 */\*\*  
 \* 检查电话号码的格式  
 \*  
 \** ***@param phoneNum*** *\** ***@return*** *\*/* **public static boolean** isPhoneNum(String phoneNum) {  
 Pattern p = Pattern  
 .compile(**"^((1[0-9])|(1[0-9])|(1[0-9])|(1[0-9]))\\d{9}$"**);  
 Matcher m = p.matcher(phoneNum);  
 **return** m.matches();  
 }  
  
 */\*\*  
 \* 检查邮箱格式  
 \*  
 \** ***@param*** *email  
 \** ***@return*** *\*/* **public static boolean** isEmail(String email) {  
 String str = **"^([a-zA-Z0-9\_\\-\\.]+)@((\\[[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.)|(([a-zA-Z0-9\\-]+\\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\\]?)$"**;  
 Pattern p = Pattern.compile(str);  
 Matcher m = p.matcher(email);  
  
 **return** m.matches();  
 }  
  
 */\*\*  
 \* 检查邮政编码格式  
 \*  
 \** ***@param*** *postcode  
 \** ***@return*** *\*/* **public static boolean** isPostcode(String postcode) {  
 String format = **"\\p{Digit}{6}"**;  
 **return** postcode.matches(format);  
 }  
  
 **public static boolean** isEmptyEditText(EditText editText){  
 **return** TextUtils.isEmpty(editText.getText().toString().trim());  
 }  
  
 **public static boolean** isEmpty(String text){  
 **return** TextUtils.isEmpty(text) || TextUtils.isEmpty(text.trim());  
 }  
  
   
 */\*\*  
 \* 隐藏手机尾号  
 \** ***@param*** *phoneNum  
 \** ***@return*** *\** ***@since*** *2015-8-9上午9:45:54  
 \** ***@author*** *cuixingwang  
 \*/* **public static** String hiddenMobileTail(String phoneNum){  
 **if**(phoneNum!=**null** && phoneNum.length()>4){  
 **return** phoneNum.subSequence(0, phoneNum.length()-4)+**"\*\*\*\*"**;  
 }**else**{  
 **return** phoneNum;  
 }  
 }  
   
 */\*\*  
 \* 将以逗号分割的String转为List  
 \** ***@param*** *string  
 \** ***@return*** *\** ***@since*** *2015-8-10下午5:23:10  
 \** ***@author*** *cuixingwang  
 \*/* **public static** List<String> parseString2List(String string){  
 List<String> list=**new** ArrayList<>();  
 **if**(TextUtils.isEmpty(string)){  
 **return** list;  
 }**else**{  
 String[] array=string.split(**","**);  
 **for** (**int** i = 0; i < array.length; i++) {  
 list.add(array[i]);  
 }  
 **return** list;  
 }  
 }  
   
 */\*\*  
 \* 将list转换为以逗号分割的字符串  
 \** ***@param*** *list  
 \** ***@return*** *形如:a,b,c,d  
 \** ***@since*** *2015-8-10下午5:24:58  
 \** ***@author*** *cuixingwang  
 \*/* **public static** <W> String parseList2String(List<W> list){  
 **return** parseList2String(**""**,list);  
 }  
  
 */\*\*  
 \* 将list转换为以逗号分割的字符串  
 \** ***@param*** *wrapStr 妹子字符串外面包裹的字符串  
 \** ***@param*** *list  
 \** ***@param*** <*W*>  
 *\** ***@return*** *形如:'a','b','c','d'  
 \*/* **public static** <W> String parseList2String(String wrapStr,List<W> list){  
 **if**(list!=**null** && list.size()>0){  
 StringBuilder sb=**new** StringBuilder();  
 **int** count=list.size();  
 **for** (**int** i = 0; i < count; i++) {  
 sb.append(wrapStr+list.get(i)+wrapStr).append(**","**);  
 }  
 **return** sb.toString().substring(0, sb.toString().length()-1);  
 }**else**{  
 **return ""**;  
 }  
 }  
  
  
 */\*\*  
 \* 将list转换为以separator的字符串  
 \** ***@param*** *list  
 \** ***@param*** *separator  
 \** ***@return*** *\*/* **public static** <W> String parseList2String(List<W> list,String separator){  
 **if**(list!=**null** && list.size()>0){  
 StringBuilder sb=**new** StringBuilder();  
 **int** count=list.size();  
 **for** (**int** i = 0; i < count; i++) {  
 W str = list.get(i);  
 **if**(str!=**null** && !**""**.equals(str)){  
 sb.append(str).append(separator);  
 }  
 }  
 String string = sb.toString();  
 **if**(string.length()==0){  
 **return ""**;  
 }  
 **return** string.substring(0, string.length()-1);  
 }**else**{  
 **return ""**;  
 }  
 }  
  
 **public static** String[] parseList2Array(List<String> list){  
 **if**(list==**null** || list.isEmpty()){  
 **return new** String[]{};  
 }  
 **int** count=list.size();  
 String[] strings=**new** String[count];  
 **for** (**int** i = 0; i < count; i++) {  
 strings[i]=list.get(i);  
 }  
 **return** strings;  
 }  
  
 */\*\*  
 \* 将list中model的某个属性取出来,用逗号分割,拼接成字符串  
 \** ***@param*** *list  
 \** ***@param*** *fieldName  
 \** ***@param*** <*W*>  
 *\** ***@return*** *\*/* **public static** <W> String buildStringsFromList(List<W> list,String fieldName){  
 **int** count = list.size();  
 StringBuilder sb=**new** StringBuilder();  
 String methodName=**"get"**+fieldName.substring(0,1).toUpperCase()+fieldName.substring(1);  
 **for** (**int** i = 0; i < count; i++) {  
 W w = list.get(i);  
 **try** {  
 String value = (String) w.getClass().getMethod(methodName).invoke(w);  
 **if**(TextUtils.isEmpty(value)){  
 **continue**;  
 }  
 sb.append(value).append(**","**);  
 } **catch** (IllegalAccessException e) {  
 e.printStackTrace();  
 } **catch** (InvocationTargetException e) {  
 e.printStackTrace();  
 } **catch** (NoSuchMethodException e) {  
 e.printStackTrace();  
 }  
 }  
 **if**(TextUtils.isEmpty(sb)){  
 **return ""**;  
 }  
 **return** sb.toString().substring(0, sb.toString().length()-1);  
 }  
  
 */\*\*  
 \* 将model集合转换成其中的某个属性集合  
 \** ***@param*** *list  
 \** ***@param*** *fieldName  
 \** ***@param*** <*W*>  
 *\** ***@param*** <*Q*>  
 *\** ***@return*** *\*/* **public static** <W, Q> List<Q> parseModels2Fields(List<W> list,String fieldName) {  
 **int** count = list.size();  
 List<Q> qlist = **new** ArrayList<>();  
 String methodName = **"get"** + fieldName.substring(0, 1).toUpperCase()+fieldName.substring(1);  
 **for** (**int** i = 0; i < count; i++) {  
 W w = list.get(i);  
 **try** {  
 Method method = w.getClass().getMethod(methodName);  
 Q qValue = (Q) method.invoke(w);  
 qlist.add(qValue);  
 } **catch** (NoSuchMethodException e) {  
 e.printStackTrace();  
 } **catch** (InvocationTargetException e) {  
 e.printStackTrace();  
 } **catch** (IllegalAccessException e) {  
 e.printStackTrace();  
 } **catch** (Exception e){  
 e.printStackTrace();  
 }  
 }  
 **return** qlist;  
 }  
  
  
  
 */\*\*  
 \* 手机号加"\*"\*/* **public static** String changePhone(String str){  
 String phone=**""**;  
 **if** (str.length()==11) {  
 phone = str.substring(0,3)+**"\*\*\*\*\*\*"**+str.substring(9,11);  
 }  
 **return** phone;  
 }  
  
 **public static boolean** isNumber(String str){  
 **return** isInteger(str) || isDecimal(str);  
 }  
  
 */\*\*  
 \* 判断该字符串是否为整数  
 \** ***@param*** *str  
 \** ***@return*** *\** ***@since*** *2015-8-29上午1:44:56  
 \** ***@author*** *cuixingwang  
 \*/* **public static boolean** isInteger(String str) {  
 **if**(TextUtils.isEmpty(str)){  
 **return false**;  
 }  
 Pattern pattern = Pattern.compile(**"[0-9]\*"**);  
 **return** pattern.matcher(str).matches();  
 }  
  
 **public static** String round(**int** number, **int** precision){  
 **return** round(String.valueOf(number),precision);  
 }  
  
 **public static** String round(**double** number, **int** precision){  
 **return** round(String.valueOf(number),precision);  
 }  
  
 **public static** String round(**float** number, **int** precision){  
 **return** round(String.valueOf(number),precision);  
 }  
  
 */\*\*  
 \* 将字符串保留N位小数  
 \** ***@param*** *str  
 \** ***@param*** *precision  
 \** ***@return*** *\*/* **public static** String round(String str, **int** precision){  
 String reslut = **null**;  
 **boolean** isMinus;  
 **try** {  
 **if**(str==**null** || **""**.equals(str)){  
 **return ""**;  
 }  
 StringBuilder sb=**new** StringBuilder();  
 sb.append(**"#"**);  
 **if**(precision>0){  
 sb.append(**"."**);  
 }  
 **for** (**int** i = 0; i < precision; i++) {  
 sb.append(**"0"**);  
 }  
  
 String format=sb.toString();  
 DecimalFormat df = **new** DecimalFormat(format);  
  
 **double** number = Double.parseDouble(str);  
  
 isMinus=number<0;  
  
 number=Math.abs(number);  
  
 reslut = df.format(number);  
  
 **int** indexOf = reslut.indexOf(**"."**);  
  
 **if**(indexOf!=-1){  
 String prefix=reslut.substring(0,reslut.indexOf(**"."**));  
  
 **if**(!isNumber(prefix)){  
 String str1 = reslut.substring(0, indexOf);  
 String str2 = reslut.substring(indexOf, reslut.length());  
 reslut= str1 +**"0"**+ str2;  
 }  
 }  
 } **catch** (NumberFormatException e) {  
 e.printStackTrace();  
 **return ""**;*//-77.68* }  
 **if**(isMinus){  
 **return "-"**+reslut;  
 }  
  
 **return** reslut;  
 }  
  
 */\*\*  
 \* 是否为小数  
 \** ***@param*** *orginal  
 \** ***@return*** *\*/* **public static boolean** isDecimal(String orginal) {  
 **return** isMatch(**"[-+]{0,1}\\d+\\.\\d\*|[-+]{0,1}\\d\*\\.\\d+"**, orginal);  
 }  
  
 **private static boolean** isMatch(String regex, String orginal) {  
 **if** (orginal == **null** || orginal.trim().equals(**""**)) {  
 **return false**;  
 }  
  
 Pattern pattern = Pattern.compile(regex);  
 Matcher isNum = pattern.matcher(orginal);  
 **return** isNum.matches();  
 }  
  
  
 */\*\*  
 \* 检测Sdcard是否存在  
 \*   
 \** ***@return*** *\*/* **public static boolean** isExitsSdcard() {  
 **return** android.os.Environment.getExternalStorageState().equals(  
 android.os.Environment.MEDIA\_MOUNTED);  
 }  
  
 */\*\*  
 \* JsonStr字符串BOM头处理  
 \*   
 \** ***@param*** *data  
 \** ***@return*** *\*/* **public static** String getNoBOMStr(String data) {  
 **if** (data != **null** && data.startsWith(**"\ufeff"**)) {  
 **return** data.substring(1);  
 }  
 **return** data;  
 }  
  
 **public static** String removeBOM(String data) {  
  
 **if** (TextUtils.isEmpty(data)) {  
 **return** data;  
 }  
  
 **if** (data.startsWith(**"\ufeff"**)) {  
 *// Log.e(TAG, "Json字符串BOM头处理");* **return** data.substring(1);  
 } **else** {  
 **return** data;  
 }  
 }  
  
 **public static boolean** isAvilible(Context context, String packageName) {  
 **final** PackageManager packageManager = context.getPackageManager();*// 获取packagemanager* List<PackageInfo> pinfo = packageManager.getInstalledPackages(0);*// 获取所有已安装程序的包信息* List<String> pName = **new** ArrayList<String>();*// 用于存储所有已安装程序的包名  
 // 从pinfo中将包名字逐一取出，压入pName list中* **if** (pinfo != **null**) {  
 **for** (**int** i = 0; i < pinfo.size(); i++) {  
 String pn = pinfo.get(i).packageName;  
 pName.add(pn);  
 }  
 }  
 **return** pName.contains(packageName);*// 判断pName中是否有目标程序的包名，有TRUE，没有FALSE* }  
  
 **public static** String processNullStr(Object originalStr){  
 **return** processNullStr(originalStr, **"暂无"**);  
 }  
  
 */\*\*  
 \* 处理空字符串  
 \** ***@param*** *originalStr  
 \** ***@param*** *defaultStr  
 \** ***@return*** *\*/* **public static** String processNullStr(Object originalStr,String defaultStr){  
 **if**(originalStr==**null** || **""**.equals(originalStr.toString()) || **"null"**.equalsIgnoreCase(originalStr.toString())){  
 **return** defaultStr;  
 }  
 **try** {  
 **if**(Double.parseDouble(originalStr.toString())!=0){  
 **return** originalStr+**""**;  
 }  
 } **catch** (NumberFormatException e) {  
 **return** originalStr.toString();  
 }  
 **return** defaultStr;  
  
 }  
  
 */\*\*  
 \* 拷贝  
 \** ***@param*** *fromObj  
 \** ***@param*** *toObj  
 \** ***@return*** *\*/* **public static boolean** copy(Object fromObj, Object toObj) {  
 Field[] fromObjFields = fromObj.getClass().getDeclaredFields();  
 Field[] toObjFields = toObj.getClass().getDeclaredFields();  
  
 **int** fromObjMethodsCount = fromObjFields.length;  
 **int** toObjMethodsCount = toObjFields.length;  
  
 **if** (fromObjMethodsCount != toObjMethodsCount) {  
 **return false**;  
 }  
 **for** (**int** i = 0; i < fromObjMethodsCount; i++) {  
 Field field = fromObjFields[i];  
 Class<?> type = field.getType();  
 *//属性名* String fieldName=field.getName();  
 String upperFieldName=fieldName.substring(0,1).toUpperCase()+fieldName.substring(1);  
 *//根据属性名得到对应的方法名* String methodNameGet=**"get"**+upperFieldName;  
 **if**(type==**boolean**.**class**){  
 methodNameGet=**"is"**+upperFieldName;  
 }  
 String methodNameSet=**"set"**+upperFieldName;  
 *//调用方法* Method methodGet = **null**;  
 **try** {  
 methodGet = fromObj.getClass().getMethod(methodNameGet);  
 Object valueGet=methodGet.invoke(fromObj);  
 Method methodSet = toObj.getClass().getMethod(methodNameSet, type);  
 methodSet.invoke(toObj,valueGet);  
 } **catch** (NoSuchMethodException | SecurityException e) {  
 e.printStackTrace();  
 } **catch** (IllegalAccessException e) {  
 e.printStackTrace();  
 } **catch** (IllegalArgumentException e) {  
 e.printStackTrace();  
 } **catch** (InvocationTargetException e) {  
 e.printStackTrace();  
 } **catch** (Exception e){  
 e.printStackTrace();  
 }  
 }  
 **return false**;  
 }  
  
 */\*\*  
 \* 获得百分比  
 \** ***@param*** *aDouble  
 \** ***@return*** *\*/* **public static** String getPercent(Double aDouble){  
 DecimalFormat decimalFormat=**new** DecimalFormat(**"0.00%"**);  
 decimalFormat.setRoundingMode(RoundingMode.HALF\_UP);  
 **return** decimalFormat.format(aDouble);  
 }  
  
 */\*\*  
 \* 获得粗略的double  
 \** ***@param*** *aDouble  
 \** ***@return*** *\*/* **public static double** getRoughlyDouble(Double aDouble){  
 DecimalFormat df=**new** DecimalFormat(**"0.00000"**);  
 **return** Double.parseDouble(df.format(aDouble));  
 }  
  
  
  
  
 */\*\*  
 \* 将秒数转换成分秒  
 \** ***@param*** *second  
 \** ***@return*** *\*/* **public static** String parseDuration(**int** second){  
 **if**(second<=60){  
 **return** second+**"\""**;  
 }**else**{  
 **int** modulo=second%60;  
 **if**(modulo==0){  
 **return** second/60+**"'"**;  
 }**else**{  
 **return** second/60+**"'"**+modulo+**"\""**;  
 }  
 }  
 }  
  
  
 */\*\*  
 \* String数组转为String字符串  
 \** ***@param*** *strings  
 \** ***@return*** *\*/* **public static** String array2String(String[] strings){  
 StringBuilder sb=**new** StringBuilder();  
 **for** (String str:strings) {  
 sb.append(str);  
 }  
 **return** sb.toString();  
 }  
  
 */\*\*  
 \* 获得该字符串中小数点后面有几位  
 \** ***@param*** *str  
 \** ***@return*** *\*/* **public static int** getPrecision(String str){  
 **int** index = str.indexOf(**"."**);  
 **if**(index==-1){  
 **return** 0;  
 }**else**{  
 **return** str.length()-index-1;  
 }  
 }  
  
 */\*\*  
 \* 将字符串转换为全角  
 \** ***@param*** *input  
 \** ***@return*** *\*/* **public static** String toSBC(String input) {  
 **char** c[] = input.toCharArray();  
 **for** (**int** i = 0; i < c.length; i++) {  
 **if** (c[i] == **' '**) {  
 c[i] = **'\u3000'**;  
 } **else if** (c[i] < **'\177'**) {  
 c[i] = (**char**) (c[i] + 65248);  
 }  
 }  
 String s = **new** String(c);  
 **return** s;  
 }  
  
 */\*\*  
 \* 取随机数  
 \** ***@param*** *min  
 \** ***@param*** *max  
 \** ***@return*** *\*/* **public static int** getRandom(**int** min, **int** max){  
 Random random = **new** Random();  
 **int** value = random.nextInt(max-min+1) + min;  
 **return** value;  
 }  
  
 **public static** String getFileSize4MB(**long** size) {  
 *//如果字节数少于1024，则直接以B为单位，否则先除于1024，后3位因太少无意义* **if** (size < 1024) {  
 **return** String.valueOf(round((**float**)size/1024/1024,2)) + **"MB"**;  
 } **else** {  
 size = size / 1024;  
 }  
 *//如果原字节数除于1024之后，少于1024，则可以直接以KB作为单位  
 //因为还没有到达要使用另一个单位的时候  
 //接下去以此类推* **if** (size < 1024) {  
 **return** String.valueOf(round((**float**)size/1024,2)) + **"MB"**;  
 } **else** {  
 size = size / 1024;  
 }  
 **if** (size < 1024) {  
 *//因为如果以MB为单位的话，要保留最后1位小数，  
 //因此，把此数乘以100之后再取余* size = size \* 100;  
 **return** String.valueOf((size / 100)) + **"."** + String.valueOf((size % 100)) + **"MB"**;  
 } **else** {  
 *//否则如果要以GB为单位的，先除于1024再作同样的处理* size = size \* 100 / 1024;  
 **return** String.valueOf((size / 100)) + **"."** + String.valueOf((size % 100)) + **"GB"**;  
 }  
  
 }  
  
 **public static** String getFileSize(**long** size) {  
 *//如果字节数少于1024，则直接以B为单位，否则先除于1024，后3位因太少无意义* **if** (size < 1024) {  
 **return** String.valueOf(size) + **"B"**;  
 } **else** {  
 size = size / 1024;  
 }  
 *//如果原字节数除于1024之后，少于1024，则可以直接以KB作为单位  
 //因为还没有到达要使用另一个单位的时候  
 //接下去以此类推* **if** (size < 1024) {  
 **return** String.valueOf(size) + **"KB"**;  
 } **else** {  
 size = size / 1024;  
 }  
 **if** (size < 1024) {  
 *//因为如果以MB为单位的话，要保留最后1位小数，  
 //因此，把此数乘以100之后再取余* size = size \* 100;  
 **return** String.valueOf((size / 100)) + **"."** + String.valueOf((size % 100)) + **"MB"**;  
 } **else** {  
 *//否则如果要以GB为单位的，先除于1024再作同样的处理* size = size \* 100 / 1024;  
 **return** String.valueOf((size / 100)) + **"."** + String.valueOf((size % 100)) + **"GB"**;  
 }  
 }  
  
}

**package** com.zimi.bookstore.common.utils;  
  
**import** android.content.Context;  
**import** android.os.Environment;  
  
**import** java.io.File;  
**import** java.math.BigDecimal;  
  
**import** io.reactivex.ObservableEmitter;  
  
**public class** DataCleanManager {  
  
 **public interface** OnGetCacheSizeListener{  
 **void** onGetCacheSize(String cacheSize);  
 }  
  
 */\*\*  
 \* 获取缓存大小  
 \** ***@param context*** *\** ***@return*** *\** ***@throws*** *Exception  
 \*/* **public static** String getTotalCacheSize(Context context) {  
 **if**(context==**null**){  
 **return "0M"**;  
 }  
 **long** cacheSize = *getFolderSize*(context.getCacheDir());  
 **if** (Environment.*getExternalStorageState*().equals(Environment.***MEDIA\_MOUNTED***)) {  
 cacheSize += *getFolderSize*(context.getExternalCacheDir());  
 }  
 **return** *getFormatSize*(cacheSize);  
 }  
  
 **public static void** getAsyncTotalCacheSize(**final** Context context, **final** OnGetCacheSizeListener onGetCacheSizeListener) {  
 **if**(context==**null**){  
 onGetCacheSizeListener.onGetCacheSize(**"0M"**);  
 **return**;  
 }  
  
 RxJavaUtils.*createObservable*(**new** RxJavaUtils.RxJavaCallback<String>() {  
 @Override  
 **public void** subscribe(ObservableEmitter<String> emitter) **throws** Exception {  
 **long** cacheSize = *getFolderSize*(context.getCacheDir());  
 **if** (Environment.*getExternalStorageState*().equals(Environment.***MEDIA\_MOUNTED***)) {  
 cacheSize += *getFolderSize*(context.getExternalCacheDir());  
 }  
 emitter.onNext(getFormatSize(cacheSize));  
 emitter.onComplete();  
 }  
  
 @Override  
 **public void** onNext(String s) {  
 onGetCacheSizeListener.onGetCacheSize(s);  
 }  
  
 @Override  
 **public void** onError(Throwable e) {  
 onGetCacheSizeListener.onGetCacheSize(**"error"**);  
 }  
 });  
  
  
 }  
  
  
  
 */\*\*  
 \* 清除缓存  
 \** ***@param*** *context  
 \*/* **public static void** clearAllCache(Context context) {  
 **if**(context==**null**){  
 **return**;  
 }  
 deleteDir(context.getCacheDir());  
 **if** (Environment.getExternalStorageState().equals(Environment.MEDIA\_MOUNTED)) {  
 deleteDir(context.getExternalCacheDir());  
 }  
 }  
  
 **private static boolean** deleteDir(File dir) {  
 **if** (dir != **null** && dir.isDirectory()) {  
 String[] children = dir.list();  
 **for** (String aChildren : children) {  
 **boolean** success = deleteDir(**new** File(dir, aChildren));  
 **if** (!success) {  
 **return false**;  
 }  
 }  
 }  
 **return** dir != **null** && dir.delete();  
 }  
  
 *// 获取文件大小  
 //Context.getExternalFilesDir() --> SDCard/Android/data/你的应用的包名/files/ 目录，一般放一些长时间保存的数据  
 //Context.getExternalCacheDir() --> SDCard/Android/data/你的应用包名/cache/目录，一般存放临时缓存数据* **public static long** getFolderSize(File file) {  
 **long** size = 0;  
 **try** {  
 File[] fileList = file.listFiles();  
 **for** (**int** i = 0; i < fileList.length; i++) {  
 *// 如果下面还有文件* **if** (fileList[i].isDirectory()) {  
 size = size + getFolderSize(fileList[i]);  
 } **else** {  
 size = size + fileList[i].length();  
 }  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return** 0;  
 }  
 **return** size;  
 }  
  
 */\*\*  
 \* 格式化单位  
 \** ***@param*** *size  
 \** ***@return*** *\*/* **public static** String getFormatSize(**double** size) {  
 **double** kiloByte = size / 1024;  
 **if** (kiloByte < 1) {  
*// return size + "Byte";* **return "0M"**;  
 }  
  
 **double** megaByte = kiloByte / 1024;  
 **if** (megaByte < 1) {  
 BigDecimal result1 = **new** BigDecimal(Double.toString(kiloByte));  
 **return** result1.setScale(2, BigDecimal.ROUND\_HALF\_UP)  
 .toPlainString() + **"K"**;  
 }  
  
 **double** gigaByte = megaByte / 1024;  
 **if** (gigaByte < 1) {  
 BigDecimal result2 = **new** BigDecimal(Double.toString(megaByte));  
 **return** result2.setScale(2, BigDecimal.ROUND\_HALF\_UP)  
 .toPlainString() + **"M"**;  
 }  
  
 **double** teraBytes = gigaByte / 1024;  
 **if** (teraBytes < 1) {  
 BigDecimal result3 = **new** BigDecimal(Double.toString(gigaByte));  
 **return** result3.setScale(2, BigDecimal.ROUND\_HALF\_UP)  
 .toPlainString() + **"GB"**;  
 }  
 BigDecimal result4 = **new** BigDecimal(teraBytes);  
 **return** result4.setScale(2, BigDecimal.ROUND\_HALF\_UP).toPlainString()  
 + **"TB"**;  
 }  
}

**package** com.zimi.bookstore.common.utils;  
  
**import** java.util.Collection;  
**import** java.util.List;  
  
*/\*\*  
 \* 数据工具类  
 \* Created by cxw on 2017/3/1.  
 \*/***public class** DataUtils {  
 **public interface** FilterCallback<E>{  
 */\*\*  
 \* 是否需要移除  
 \** ***@param model*** *\** ***@return*** *\*/* **boolean** isNeedRemove(E model);  
 }  
  
 */\*\*  
 \* 过滤集合中的数据  
 \** ***@param list*** *待处理的数据集合  
 \** ***@param filterCallback*** *\** ***@param <E>*** *\*/* **public static** <E> **void** filterList(List<E> list,FilterCallback<E> filterCallback){  
 **if**(list==**null**){  
 **return**;  
 }  
 **int** size = list.size();  
 **for** (**int** i = 0; i < size; i++) {  
 **if**(filterCallback.isNeedRemove(list.get(i))){  
 list.remove(i);  
 i--;  
 size--;  
 }  
 }  
 }  
  
 */\*\*  
 \* 判断集合数据是否为空  
 \** ***@param collection*** *\** ***@return*** *\*/* **public static boolean** isEmpty(Collection collection){  
 **return** collection==**null** || collection.isEmpty();  
 }  
  
  
}

**package** com.zimi.bookstore.common.utils;  
  
**import** android.app.Activity;  
**import** android.content.DialogInterface;  
**import** android.graphics.Color;  
**import** android.support.annotation.NonNull;  
**import** android.text.InputType;  
**import** android.view.View;  
  
**import** com.afollestad.materialdialogs.DialogAction;  
**import** com.afollestad.materialdialogs.MaterialDialog;  
**import** com.zimi.bookstore.common.R;  
  
**import** java.util.Collection;  
  
  
*/\*\*  
 \* Created by cxw on 2016/4/22.  
 \*/***public class** DialogUtils {  
  
  
  
 **private static** MaterialDialog *materialDialog*;  
  
 **public static abstract class** OnButtonClickListener {  
 **public abstract void** onConfirmButtonClick();  
  
 **public void** onCancelButtonClick() {  
  
 }  
  
 **public void** onDismiss(**boolean** cancelable){  
  
 }  
 }  
  
 **public static abstract class** OnItemClickListener {  
 **public abstract void** onSelection(**int** which, CharSequence text);  
 }  
  
 **public static** MaterialDialog showSingleButtonDialog(Activity activity, String message, String positiveText) {  
 **return** showCustomMessageDialog(activity, message, **null**, positiveText, **null**);  
 }  
  
 **public static** MaterialDialog showSingleButtonDialog(Activity activity, String message, String positiveText, OnButtonClickListener onButtonClickListener) {  
 **return** showSingleButtonDialog(activity, message, positiveText, **true**, onButtonClickListener);  
 }  
  
 **public static** MaterialDialog showSingleButtonDialog(Activity activity, String message, String positiveText, **boolean** cancelabl, OnButtonClickListener onButtonClickListener) {  
 **return** showSingleButtonDialog(activity, **null**, message, positiveText, cancelabl, onButtonClickListener);  
 }  
  
 **public static** MaterialDialog showSingleButtonDialog(Activity activity, String title, String message, String positiveText, **boolean** cancelabl, OnButtonClickListener onButtonClickListener) {  
 **return** showCustomMessageDialog(activity, title, message, **null**, positiveText, cancelabl, onButtonClickListener);  
 }  
  
 **public static** MaterialDialog showNormalDialog(Activity activity, String message, **final** OnButtonClickListener onButtonClickListener) {  
 **return** showCustomMessageDialog(activity, message, **"取消"**, **"确定"**, onButtonClickListener);  
 }  
  
 **public static** MaterialDialog showCustomMessageDialog(Activity activity, String message, String negativeText, String positiveText, **final** OnButtonClickListener onButtonClickListener) {  
 **return** showCustomMessageDialog(activity, **"提示"**, message, negativeText, positiveText, **true**, onButtonClickListener);  
  
 }  
  
  
 **public static** MaterialDialog showCustomMessageDialog(Activity activity, String title, String message, String negativeText, String positiveText, **final boolean** cancelable, **final** OnButtonClickListener onButtonClickListener) {  
 materialDialog = **new** MaterialDialog.Builder(activity)  
 .title(title)  
 .content(message)  
 .cancelable(cancelable)  
 .negativeText(negativeText)  
 .positiveText(positiveText)  
 .negativeColor(activity.getResources().getColor(R.color.gray\_7f))  
 .positiveColor(activity.getResources().getColor(R.color.app\_color))  
 .onNegative(**new** MaterialDialog.SingleButtonCallback() {  
 @Override  
 **public void** onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {  
 **if** (onButtonClickListener != **null**) {  
 onButtonClickListener.onCancelButtonClick();  
 }  
 }  
 })  
 .onPositive(**new** MaterialDialog.SingleButtonCallback() {  
 @Override  
 **public void** onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {  
 **if** (onButtonClickListener != **null**) {  
 onButtonClickListener.onConfirmButtonClick();  
 }  
 }  
 })  
 .dismissListener(**new** DialogInterface.OnDismissListener() {  
 @Override  
 **public void** onDismiss(DialogInterface dialogInterface) {  
 onButtonClickListener.onDismiss(cancelable);  
 }  
 }).build();  
 **try** {  
 materialDialog.show();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** materialDialog;  
 }  
  
  
 **public static** MaterialDialog showCustomMessageDialog(Activity activity, String title, String message, String negativeText,  
 String positiveText, **final** OnButtonClickListener onButtonClickListener) {  
 materialDialog = **new** MaterialDialog.Builder(activity)  
 .title(title)  
 .content(message)  
 .negativeText(negativeText).negativeColor(activity.getResources().getColor(R.color.gray\_7f))  
 .positiveText(positiveText).positiveColor(activity.getResources().getColor(R.color.app\_color))  
 .onNegative(**new** MaterialDialog.SingleButtonCallback() {  
 @Override  
 **public void** onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {  
 **if** (onButtonClickListener != **null**) {  
 onButtonClickListener.onCancelButtonClick();  
 }  
 }  
 })  
 .onPositive(**new** MaterialDialog.SingleButtonCallback() {  
 @Override  
 **public void** onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {  
 **if** (onButtonClickListener != **null**) {  
 onButtonClickListener.onConfirmButtonClick();  
 }  
 }  
 }).build();  
 **try** {  
 materialDialog.show();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** materialDialog;  
 }  
  
 **public static** MaterialDialog showSingleDialog(Activity activity, String title, Collection<String> sList, **final** OnItemClickListener onItemClickListener) {  
 materialDialog = **new** MaterialDialog.Builder(activity)  
 .title(title)  
 .items(sList)  
 .itemsCallback(**new** MaterialDialog.ListCallback() {  
 @Override  
 **public void** onSelection(MaterialDialog dialog, View itemView, **int** which, CharSequence text) {  
 onItemClickListener.onSelection(which, text);  
 }  
 }).build();  
 **try** {  
 materialDialog.show();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** materialDialog;  
 }  
  
  
 **public static boolean** isShowing() {  
 **return** materialDialog != **null** && materialDialog.isShowing();  
 }  
  
 **public static void** closeDialog() {  
 **if** (materialDialog != **null** && materialDialog.isShowing()) {  
 materialDialog.dismiss();  
 }  
 }  
  
 **public static void** showEditTextDialog(Activity activity,String title,String content,String inputHint,MaterialDialog.InputCallback inputCallback) {  
 materialDialog=**new** MaterialDialog.Builder(activity)  
 .backgroundColor(Color.WHITE)  
 .content(content)  
 .inputType(InputType.TYPE\_CLASS\_TEXT)  
 .input(inputHint,**""**,inputCallback)  
 .negativeText(**"取消"**)  
 .positiveText(**"确定"**)  
 .negativeColor(activity.getResources().getColor(R.color.black\_66))  
 .positiveColor(activity.getResources().getColor(R.color.app\_color))  
 .onNegative(**new** MaterialDialog.SingleButtonCallback() {  
 @Override  
 **public void** onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {  
 dialog.dismiss();  
 }  
 })  
 .widgetColor(Color.parseColor(**"#d5d5d5"**))  
 .autoDismiss(**false**)  
 .show();  
 }  
  
}

**package** com.zimi.bookstore.common.utils;  
  
**import** android.app.Activity;  
**import** android.content.ActivityNotFoundException;  
**import** android.content.Context;  
**import** android.content.Intent;  
**import** android.content.pm.PackageManager;  
**import** android.net.Uri;  
**import** android.os.Build;  
  
*/\*\*  
 \** ***@author*** *崔兴旺  
 \** ***@description*** *\** ***@date*** *2019/8/23 2:15  
 \*/***public class** DownloadManagerUtils {  
 **public static boolean** checkDownloadManagerEnable(Context context) {  
 **int** state = context.getApplicationContext().getPackageManager()  
 .getApplicationEnabledSetting(**"com.android.providers.downloads"**);  
  
 **if** (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.JELLY\_BEAN\_MR2) {  
 **return** !(state == PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED ||  
 state == PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED\_USER  
 || state == PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED\_UNTIL\_USED);  
 } **else** {  
 **return** !(state == PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED ||  
 state == PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED\_USER);  
 }  
 }  
  
 **public static void** gotoDownloadManageActivity(Activity activity) {  
 **try** {  
 *//Open the specific App Info page:* Intent intent = **new** Intent(android.provider.Settings.ACTION\_APPLICATION\_DETAILS\_SETTINGS);  
 intent.setData(Uri.parse(**"package:"** + **"com.android.providers.downloads"**));  
 activity.startActivity(intent);  
  
 } **catch** (ActivityNotFoundException e) {  
 e.printStackTrace();  
  
 *//Open the generic Apps page:* Intent intent = **new** Intent(android.provider.Settings.ACTION\_MANAGE\_APPLICATIONS\_SETTINGS);  
 activity.startActivity(intent);  
 }  
 }  
}

**package** com.zimi.bookstore.common.utils;  
  
**import** android.text.Editable;  
**import** android.text.TextUtils;  
**import** android.widget.EditText;  
**import** android.widget.TextView;  
  
**import** com.zimi.bookstore.common.base.adapter.SimpleTextWatcherAdapter;  
  
  
*/\*\*  
 \* Created by cxw on 2016/4/15.  
 \*/***public class** EditTextUtils {  
 */\*\*  
 \* 将光标移动到末尾  
 \*/* **public static void** moveCursorToLast(TextView textView){  
 **if**(textView **instanceof** EditText){  
 ((EditText)textView).setSelection(textView.getText().length());  
 }  
 }  
  
 */\*\*  
 \* 限制文本内容长度,超出时,光标移动到末尾  
 \** ***@param textView*** *\*/* **public static void** limitCount(**final** TextView textView, **final int** limit){  
 textView.addTextChangedListener(**new** SimpleTextWatcherAdapter() {  
 @Override  
 **public void** afterTextChanged(Editable s) {  
 String str = textView.getText().toString();  
 **if**(str.length()>limit){  
 str=str.substring(0,limit);  
  
 *// 如果内容是表情,还需要用SmileUtils类转换一下* textView.setText(str);  
  
 *//如果是输入框,则将光标移动到末尾  
 moveCursorToLast*(textView);  
 }  
 }  
 });  
 }  
  
  
 **public static void** setTextComplatedListener(**final** IOnTextComplatedListener iOnTextComplatedListener, **final** TextView...textViews){  
 **for** (TextView textView:textViews) {  
 textView.addTextChangedListener(**new** SimpleTextWatcherAdapter() {  
 @Override  
 **public void** afterTextChanged(Editable s) {  
 **for** (TextView textView:textViews) {  
 **if**(TextUtils.*isEmpty*(textView.getText().toString().trim())){  
 iOnTextComplatedListener.onTextChanged(**false**);  
 **return**;  
 }  
 }  
 iOnTextComplatedListener.onTextChanged(**true**);  
 }  
 });  
 }  
 }  
  
 **public interface** IOnTextComplatedListener{  
 **void** onTextChanged(**boolean** isCompated);  
 }  
}

**package** com.zimi.bookstore.common.utils;  
  
**import** android.content.Context;  
**import** android.graphics.Bitmap;  
**import** android.graphics.Bitmap.CompressFormat;  
**import** android.graphics.BitmapFactory;  
**import** android.os.Environment;  
**import** android.text.TextUtils;  
**import** android.util.Log;  
  
**import** com.alick.utilslibrary.BLog;  
  
**import** java.io.ByteArrayOutputStream;  
**import** java.io.File;  
**import** java.io.FileInputStream;  
**import** java.io.FileNotFoundException;  
**import** java.io.FileOutputStream;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.io.UnsupportedEncodingException;  
  
*/\*\*  
 \** ***@author*** *gyw  
 \** ***@version*** *1.0  
 \** ***@time:*** *2015-6-11 上午11:19:02  
 \** ***@fun:*** *文件工具类  
 \*/***public class** FileUtils {  
  
 **public static final** String ***ROOT\_DIR*** = **"qingyin"**;  
 **public static final** String ***DOWNLOAD\_DIR*** = **"download"**;  
 **public static final** String ***CACHE\_DIR*** = **"cache"**;  
 **private static final** String ***TAG*** = **"FileUtil"**;  
  
 */\*\* 判断SD卡是否挂载 \*/* **public static boolean** isSDCardAvailable() {  
 **return** Environment.***MEDIA\_MOUNTED***.equals(Environment  
 .*getExternalStorageState*());  
 }  
  
 */\*\* 获取缓存目录 \*/* @Deprecated  
 **public static** String getCacheDir(Context context) {  
 **return** *getDir*(context, ***CACHE\_DIR***);  
 }  
  
 */\*\* 获取下载目录 \*/* **public static** String getDownloadDir(Context context) {  
 **return** *getDir*(context, DOWNLOAD\_DIR);  
 }  
  
 */\*\* 获取应用目录，当SD卡存在时，获取SD卡上的目录，当SD卡不存在时，获取应用的cache目录 \*/* **public static** String getDir(Context context, String name) {  
 StringBuilder sb = **new** StringBuilder();  
 *// boolean isFirstInstallSd = SharedPreUtil.getBooleanSharedPre(context,  
 // "first\_install\_sd", true);* **if** (isSDCardAvailable()) {  
 sb.append(getExternalStoragePath());  
 } **else** {  
 sb.append(getCachePath(context));  
 }  
 sb.append(name);  
 sb.append(File.separator);  
 String path = sb.toString();  
 **if** (createDirs(path)) {  
 **return** path;  
 } **else** {  
 **return null**;  
 }  
 }  
  
 */\*\* 获取SD下的应用目录  
 \*  
 \*/* **public static** String getExternalStoragePath() {  
 StringBuilder sb = **new** StringBuilder();  
 sb.append(Environment.getExternalStorageDirectory().getAbsolutePath());  
 sb.append(File.separator);  
 sb.append(ROOT\_DIR);  
 sb.append(File.separator);  
 **return** sb.toString();  
 }  
  
 */\*\* 获取应用的cache目录 \*/* **public static** String getCachePath(Context context) {  
 File f = context.getCacheDir();  
 **if** (**null** == f) {  
 **return null**;  
 } **else** {  
 **return** f.getAbsolutePath() + **"/"**;  
 }  
 }  
  
 */\*\* 创建文件夹 \*/* **public static boolean** createDirs(String dirPath) {  
 File file = **new** File(dirPath);  
 **if** (!file.exists() || !file.isDirectory()) {  
 **return** file.mkdirs();  
 }  
 **return true**;  
 }  
  
 **public static boolean** createFile(File file) {  
 **if** (!file.exists()) {*// 如果文件不存在，或者是文件夹* String parent = file.getParent();  
 File parentFile = **new** File(parent);*// 根据父路径创建文件对象* **if** (!parentFile.exists() || !parentFile.isDirectory()) {  
 parentFile.mkdirs();  
 }  
 **try** {  
 file.createNewFile();*// 创建文件* } **catch** (Exception e) {  
 BLog.e(**"创建文件失败:"**+e.getMessage());  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
 **return true**;  
 }  
  
 */\*\* 创建文件 \*/* **public static boolean** createFile(String filePath) {  
 File file = **new** File(filePath);  
 **return** createFile(file);  
 }  
  
 */\*\* 判断文件是否存在 \*/* **public static boolean** isExistFile(String filePath) {  
 **if**(TextUtils.isEmpty(filePath)){  
 **return false**;  
 }  
 File file = **new** File(filePath);  
 **return** file.exists();  
 }  
  
 */\*\* 删除文件 \*/* **public static void** deleteFile(Context context, String filePath) {  
 File file = **new** File(filePath);  
 **if** (file.exists()) { *// 判断文件是否存在* file.delete(); *// 删除文件* } **else** {  
 *// ToastUtil.showShortToast(context, "文件不存在或已删除");* }  
 }  
  
 */\*\* 删除文件 \*/* **public static boolean** deleteFile(String filePath) {  
 File file = **new** File(filePath);  
 **if** (file.exists()) { *// 判断文件是否存在* **return** file.delete(); *// 删除文件* }  
 **return false**;  
 }  
  
 */\*\*  
 \*   
 \** ***@param*** *file  
 \*/* **public static void** deleteFileFolder(File file) {  
 **if** (file.exists()) {  
 **if** (file.isFile()) {  
 file.delete();  
 **return**;  
 }  
 **if** (file.isDirectory()) {  
 File[] childFile = file.listFiles();  
 **if** (childFile == **null** || childFile.length == 0) {  
 file.delete();  
 **return**;  
 }  
 **for** (File f : childFile) {  
 deleteFileFolder(f);  
 }  
 file.delete();  
 }  
 }  
 }  
  
 **public static boolean** renameFile(File srcfile,String newFileName){  
 **if**(srcfile==**null** || !srcfile.exists()){  
 **return false**;  
 }  
 **return** srcfile.renameTo(**new** File(srcfile.getParentFile(),newFileName));  
 }  
  
 **private static** String IMAGE\_PATH = **""**;  
 **private static** String FILE\_NAME= **"/icon\_app.png"**;  
 **private static final int** sharePictureResId = 0;  
 */\*\*  
 \* 创建本地分享图片  
 \** ***@return*** *\*   
 \*/* **public static** String createLocalPic(Context context){  
 **if** (Environment.getExternalStorageState().equals(  
 Environment.MEDIA\_MOUNTED)) {  
 IMAGE\_PATH = Environment  
 .getExternalStorageDirectory()  
 .getAbsolutePath()  
 + FILE\_NAME;  
 } **else** {  
 IMAGE\_PATH = context.getFilesDir()  
 .getAbsolutePath() + FILE\_NAME;  
 }  
 File shareAppFile = **new** File(IMAGE\_PATH);  
 **if** (!shareAppFile.exists()) {  
 **try** {  
 shareAppFile.createNewFile();  
 Bitmap bt = BitmapFactory.decodeResource(  
 context.getResources(), sharePictureResId);  
 FileOutputStream fos = **new** FileOutputStream(  
 shareAppFile);  
 bt.compress(CompressFormat.JPEG, 100, fos);  
 fos.flush();  
 fos.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 **return** IMAGE\_PATH;  
 }  
  
 **public static void** bytes2File(**byte**[] bytes,String filePath){  
 bytes2File(bytes,filePath,**false**);  
 }  
 **public static void** bytes2File(**byte**[] bytes,String filePath,**boolean** isAppend){  
 FileOutputStream out=**null**;  
 **try** {  
 out=**new** FileOutputStream(filePath,isAppend);  
 out.write(bytes,0,bytes.length);  
 out.flush();  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **catch** (Exception e){  
 e.printStackTrace();  
 }**finally** {  
 **if**(out!=**null**){  
 **try** {  
 out.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
  
 */\*\*  
 \* 将文件转换成byte数组  
 \** ***@param*** *file  
 \** ***@return*** *\** ***@since*** *2015-8-24下午9:53:49  
 \** ***@author*** *cuixingwang  
 \*/* **public static byte**[] file2Bytes(File file){  
 **if**(file==**null** || !file.exists()){  
 **return null**;  
 }  
 **byte**[] buf=**new byte**[1024];  
 ByteArrayOutputStream out=**new** ByteArrayOutputStream();  
 InputStream in=**null**;  
 **try** {  
 in=**new** FileInputStream(file);  
 **int** len=0;  
 **while**((len=in.read(buf))!=-1){  
 out.write(buf,0,len);  
 }  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 } **catch** (UnsupportedEncodingException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **finally**{  
 **if**(out!=**null**){  
 **try** {  
 out.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 **if**(in!=**null**){  
 **try** {  
 in.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 **return** out.toByteArray();  
 }  
  
 */\*\*  
 \* 复制文件  
 \** ***@param*** *fromFile  
 \** ***@param*** *toFile  
 \*/* **public static void** copyfile(String fromFile, String toFile){  
 copyfile(**new** File(fromFile),**new** File(toFile));  
 }  
  
 **public static void** cutFile(String fromFile, String toFile){  
 cutFile(**new** File(fromFile),**new** File(toFile));  
 }  
  
 */\*\*  
 \* 剪切文件  
 \** ***@param*** *fromFile  
 \** ***@param*** *toFile  
 \*/* **public static void** cutFile(File fromFile, File toFile){  
 copyfile(fromFile,toFile);  
 fromFile.delete();  
 }  
  
  
 */\*\*  
 \* 复制文件  
 \*  
 \** ***@param*** *inputStream  
 \** ***@param*** *toFile  
 \** ***@since*** *2015-9-22下午3:00:15  
 \** ***@author*** *zhanghebin  
 \*/* **public static boolean** copyfile(InputStream inputStream, File toFile) {  
 FileOutputStream fosto = **null**;  
 **try** {  
 **boolean** newFile = createFile(toFile);  
 BLog.i(**"新建文件是否成功:"**+newFile);  
 **if**(!newFile){  
 **return false**;  
 }  
 fosto = **new** FileOutputStream(toFile);  
 **byte** bt[] = **new byte**[1024\*3];  
 **int** c;  
 **while** ((c = inputStream.read(bt)) > 0) {  
 fosto.write(bt, 0, c); *// 将内容写到新文件当中* }  
 **return true**;  
 } **catch** (Exception ex) {  
 Log.e(**"readfile"**, ex.getMessage());  
 **return false**;  
 } **finally** {  
 BLog.i(**"cxw"**,**"关闭流"**);  
 **try** {  
 **if** (inputStream != **null**) {  
 inputStream.close();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **try** {  
 **if** (fosto != **null**) {  
 fosto.close();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 }  
  
  
 */\*\*  
 \* 复制文件  
 \*   
 \** ***@param*** *fromFile  
 \** ***@param*** *toFile  
 \** ***@since*** *2015-9-22下午3:00:15  
 \** ***@author*** *zhanghebin  
 \*/* **public static boolean** copyfile(File fromFile, File toFile) {  
 FileInputStream fosfrom = **null**;  
 FileOutputStream fosto = **null**;  
 **if** (!fromFile.exists()) {  
 **return false**;  
 }  
 **if** (!fromFile.isFile()) {  
 **return false**;  
 }  
 **if** (!fromFile.canRead()) {  
 **return false**;  
 }  
 **try** {  
 **boolean** newFile = createFile(toFile);  
 BLog.i(**"新建文件是否成功:"**+newFile);  
 **if**(!newFile){  
 **return false**;  
 }  
 fosfrom = **new** FileInputStream(fromFile);  
 fosto = **new** FileOutputStream(toFile);  
 **byte** bt[] = **new byte**[1024\*3];  
 **int** c;  
 **while** ((c = fosfrom.read(bt)) > 0) {  
 fosto.write(bt, 0, c); *// 将内容写到新文件当中* }  
 **return true**;  
 } **catch** (Exception ex) {  
 Log.e(**"readfile"**, ex.getMessage());  
 **return false**;  
 } **finally** {  
 BLog.i(**"cxw"**,**"关闭流"**);  
 **try** {  
 **if** (fosfrom != **null**) {  
 fosfrom.close();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **try** {  
 **if** (fosto != **null**) {  
 fosto.close();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 */\*\*  
 \* 获得文件扩展名(不包含.)  
 \** ***@param*** *filePathOrUrl  
 \** ***@return*** *\*/* **public static** String getExtName(String filePathOrUrl){  
 **if**(TextUtils.isEmpty(filePathOrUrl) || !filePathOrUrl.contains(**"."**)){  
 **return ""**;  
 }  
 **if**(filePathOrUrl.lastIndexOf(**"."**) < filePathOrUrl.lastIndexOf(**"/"**)){  
 **return ""**;  
 }  
  
 **try** {  
 **return** filePathOrUrl.substring(filePathOrUrl.lastIndexOf(**"."**)+1);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return ""**;  
 }  
 }  
  
 */\*\*  
 \* 获得文件扩展名  
 \** ***@param*** *file  
 \** ***@return*** *\*/* **public static** String getExtName(File file){  
 **return** getExtName(file.getAbsolutePath());  
 }  
  
 */\*\*  
 \* 获得文件名  
 \** ***@param*** *filePathOrUrl  
 \** ***@return*** *\*/* **public static** String getFileName(String filePathOrUrl){  
 **try** {  
 String fileName=filePathOrUrl.substring(filePathOrUrl.lastIndexOf(**"/"**)+1);  
 **return** fileName;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return ""**;  
 }  
  
 **public static void** writeFile(String filePath,String content){  
 **if**(TextUtils.isEmpty(filePath)){  
 **return**;  
 }  
 File file=**new** File(filePath);  
  
 **boolean** isExists = file.exists();  
 **if**(!isExists){  
 isExists=createFile(file);  
 }  
 **if**(isExists){  
 FileOutputStream out=**null**;  
 **try** {  
 out=**new** FileOutputStream(file,**true**);  
 **byte**[] bytes = content.getBytes();  
 out.write(bytes,0,bytes.length);  
 out.flush();  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **finally** {  
 **if**(out!=**null**){  
 **try** {  
 out.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 }  
  
}

**package** com.zimi.bookstore.common.utils;  
  
**import** android.annotation.SuppressLint;  
**import** android.content.ContentUris;  
**import** android.content.Context;  
**import** android.database.Cursor;  
**import** android.net.Uri;  
**import** android.os.Build;  
**import** android.os.Environment;  
**import** android.provider.DocumentsContract;  
**import** android.provider.MediaStore;  
  
**import** java.io.File;  
  
**import static** android.os.Build.VERSION\_CODES.ICE\_CREAM\_SANDWICH;  
  
**public class** ImageUtils {  
 @SuppressLint(**"NewApi"**)   
 **public static** String getPath(**final** Context context, **final** Uri uri) {   
 **final boolean** isKitKat = Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.KITKAT;   
 *// DocumentProvider* **if** (isKitKat && DocumentsContract.isDocumentUri(context, uri)) {   
 *// ExternalStorageProvider* **if** (isExternalStorageDocument(uri)) {   
 **final** String docId = DocumentsContract.getDocumentId(uri);   
 **final** String[] split = docId.split(**":"**);   
 **final** String type = split[0];   
   
 **if** (**"primary"**.equalsIgnoreCase(type)) {   
 **return** Environment.getExternalStorageDirectory() + **"/"** + split[1];   
 }   
   
 *// TODO handle non-primary volumes* }   
 *// DownloadsProvider* **else if** (isDownloadsDocument(uri)) {   
   
 **final** String id = DocumentsContract.getDocumentId(uri);   
 **final** Uri contentUri = ContentUris.withAppendedId(   
 Uri.parse(**"content://downloads/public\_downloads"**), Long.valueOf(id));   
   
 **return** getDataColumn(context, contentUri, **null**, **null**);   
 }   
 *// MediaProvider* **else if** (isMediaDocument(uri)) {   
 **final** String docId = DocumentsContract.getDocumentId(uri);   
 **final** String[] split = docId.split(**":"**);   
 **final** String type = split[0];   
   
 Uri contentUri = **null**;   
 **if** (**"image"**.equals(type)) {   
 contentUri = MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI;   
 } **else if** (**"video"**.equals(type)) {   
 contentUri = MediaStore.Video.Media.EXTERNAL\_CONTENT\_URI;   
 } **else if** (**"audio"**.equals(type)) {   
 contentUri = MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI;   
 }   
   
 **final** String selection = **"\_id=?"**;   
 **final** String[] selectionArgs = **new** String[] {   
 split[1]   
 };   
   
 **return** getDataColumn(context, contentUri, selection, selectionArgs);   
 }   
 }   
 *// MediaStore (and general)* **else if** (**"content"**.equalsIgnoreCase(uri.getScheme())) {   
   
 *// Return the remote address* **if** (isGooglePhotosUri(uri))   
 **return** uri.getLastPathSegment();   
   
 **return** getDataColumn(context, uri, **null**, **null**);   
 }   
 *// File* **else if** (**"file"**.equalsIgnoreCase(uri.getScheme())) {   
 **return** uri.getPath();   
 }   
   
 **return null**;   
 }   
   
 */\*\*   
 \* Get the value of the data column for this Uri. This is useful for   
 \* MediaStore Uris, and other file-based ContentProviders.   
 \*   
 \** ***@param*** *context The context.   
 \** ***@param*** *uri The Uri to query.   
 \** ***@param*** *selection (Optional) Filter used in the query.   
 \** ***@param*** *selectionArgs (Optional) Selection arguments used in the query.   
 \** ***@return*** *The value of the \_data column, which is typically a file path.   
 \*/* **public static** String getDataColumn(Context context, Uri uri, String selection,   
 String[] selectionArgs) {   
   
 Cursor cursor = **null**;   
 **final** String column = **"\_data"**;   
 **final** String[] projection = {   
 column   
 };   
   
 **try** {   
 cursor = context.getContentResolver().query(uri, projection, selection, selectionArgs,   
 **null**);   
 **if** (cursor != **null** && cursor.moveToFirst()) {   
 **final int** index = cursor.getColumnIndexOrThrow(column);   
 **return** cursor.getString(index);   
 }   
 } **finally** {   
 **if** (cursor != **null**)   
 cursor.close();   
 }   
 **return null**;   
 }   
   
   
 */\*\*   
 \** ***@param*** *uri The Uri to check.   
 \** ***@return*** *Whether the Uri authority is ExternalStorageProvider.   
 \*/* **public static boolean** isExternalStorageDocument(Uri uri) {   
 **return "com.android.externalstorage.documents"**.equals(uri.getAuthority());   
 }   
   
 */\*\*   
 \** ***@param*** *uri The Uri to check.   
 \** ***@return*** *Whether the Uri authority is DownloadsProvider.   
 \*/* **public static boolean** isDownloadsDocument(Uri uri) {   
 **return "com.android.providers.downloads.documents"**.equals(uri.getAuthority());   
 }   
   
 */\*\*   
 \** ***@param*** *uri The Uri to check.   
 \** ***@return*** *Whether the Uri authority is MediaProvider.   
 \*/* **public static boolean** isMediaDocument(Uri uri) {   
 **return "com.android.providers.media.documents"**.equals(uri.getAuthority());   
 }   
   
 */\*\*   
 \** ***@param*** *uri The Uri to check.   
 \** ***@return*** *Whether the Uri authority is Google Photos.   
 \*/* **public static boolean** isGooglePhotosUri(Uri uri) {   
 **return "com.google.android.apps.photos.content"**.equals(uri.getAuthority());   
 }  
  
 */\*\*  
 \* 将选择的图片Uri(形如:content://media/external/images/media/67800)  
 \* 转换成文件路径(形如:/sdcard/emulated/0/test.png)  
 \** ***@param*** *context  
 \** ***@param*** *uri  
 \** ***@return*** *\*/* **public static** String photoUri2FilePath(Context context, Uri uri){  
 String filePath;  
 String[] proj = {MediaStore.Images.Media.DATA};  
 Cursor cursor = context.getContentResolver().query(uri, proj, **null**, **null**, **null**);  
 **if** (cursor == **null**) {  
 filePath = uri.getPath();  
 } **else** {  
 **int** column\_index = cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA);  
 cursor.moveToFirst();  
 filePath = cursor.getString(column\_index);  
 }  
 **if** (Build.VERSION.SDK\_INT<ICE\_CREAM\_SANDWICH && cursor != **null** && !cursor.isClosed()) {  
 cursor.close();  
 }  
 **return** filePath;  
 }  
  
 **public static** File saveImage2sdcard(){  
  
  
 **return null**;  
 }  
}