[Xutils3网络框架的二次封装](http://blog.csdn.net/u010456903/article/details/52995484)

**[Android](http://lib.csdn.net/base/android" \o "Android知识库" \t "http://blog.csdn.net/u010456903/article/details/_blank)**的网络请求框架真是太多了，不管使用哪个，我感觉做个二次封装比较好，不管需求用哪个改起来方便。

这是Xutils3的封装，下次出个OkHttp3的，道理都是一样的。代码亲测了异步get获取数据是OK的，文件上传没有亲测，如有问题咱们一起沟通交流，谢谢。

Xutils3的配置

1、在gradle中添加

compile 'org.xutils:xutils:3.3.22'

2、在Application的onCreate中初始化

public class MyApplication extends Application {

@Override

public void onCreate() {

super.onCreate();

x.Ext.*init*(this);

}

}

在配置清单文件中引用

<application android:name=".MyApplication"

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

然后就是重点 Xutils3的封装类

package org.xutils.sample.http;  
  
import android.os.Handler;  
import android.os.Looper;  
import android.widget.ImageView;  
  
import org.xutils.common.Callback;  
import org.xutils.http.RequestParams;  
import org.xutils.image.ImageOptions;  
import org.xutils.sample.R;  
import org.xutils.x;  
  
import java.io.File;  
import java.util.Map;  
  
  
public class Xutils {  
  
 private volatile static Xutils *instance*;  
 private Handler handler;  
 private ImageOptions options;  
  
 private Xutils() {  
 handler = new Handler(Looper.*getMainLooper*());  
 }  
  
 */\*\*  
 \* 单例模式  
 \*/* public static Xutils getInstance() {  
 if (*instance* == null) {  
 synchronized (Xutils.class) {  
 if (*instance* == null) {  
 *instance* = new Xutils();  
 }  
 }  
 }  
 return *instance*;  
 }  
  
 */\*\*  
 \* 异步get请求  
 \*/* public void get(String url, Map<String, String> maps, final XCallBack callBack) {  
 RequestParams params = new RequestParams(url);  
 if (maps != null && !maps.isEmpty()) {  
 for (Map.Entry<String, String> entry : maps.entrySet()) {  
 params.addQueryStringParameter(entry.getKey(), entry.getValue());  
 }  
 }  
 x.*http*().get(params, new Callback.CommonCallback<String>() {  
  
 @Override  
 public void onSuccess(String result) {  
  
 onSuccessResponse(result, callBack);  
 }  
  
 @Override  
 public void onError(Throwable ex, boolean isOnCallback) {  
 ex.printStackTrace();  
 }  
  
 @Override  
 public void onCancelled(CancelledException cex) {  
  
 }  
  
 @Override  
 public void onFinished() {  
  
 }  
 });  
 }  
  
 */\*\*  
 \* 异步post请求  
 \*/* public void post(String url, Map<String, String> maps, final XCallBack callback) {  
 RequestParams params = new RequestParams(url);  
 if (maps != null && !maps.isEmpty()) {  
 for (Map.Entry<String, String> entry : maps.entrySet()) {  
 params.addBodyParameter(entry.getKey(), entry.getValue());  
 }  
 }  
  
 x.*http*().post(params, new Callback.CommonCallback<String>() {  
  
 @Override  
 public void onSuccess(String result) {  
 onSuccessResponse(result, callback);  
 }  
  
 @Override  
 public void onError(Throwable ex, boolean isOnCallback) {  
  
 }  
  
 @Override  
 public void onCancelled(CancelledException cex) {  
  
 }  
  
 @Override  
 public void onFinished() {  
  
 }  
 });  
 }  
  
 */\*\*  
 \* 带缓存数据的异步get请求  
 \*/* public void getCache(String url, Map<String, String> maps, final boolean cache,  
 final XCallBack callback) {  
  
 RequestParams params = new RequestParams(url);  
 if (maps != null && !maps.isEmpty()) {  
 for (Map.Entry<String, String> entry : maps.entrySet()) {  
 params.addQueryStringParameter(entry.getKey(), entry.getValue());  
 }  
 }  
 x.*http*().get(params, new Callback.CacheCallback<String>() {  
 @Override  
 public void onSuccess(String result) {  
 onSuccessResponse(result, callback);  
 }  
  
 @Override  
 public void onError(Throwable ex, boolean isOnCallback) {  
  
 }  
  
 @Override  
 public void onCancelled(CancelledException cex) {  
  
 }  
  
 @Override  
 public void onFinished() {  
  
 }  
  
 @Override  
 public boolean onCache(String result) {  
 if (cache) {  
 onSuccessResponse(result, callback);  
 }  
 return cache;  
 }  
 });  
 }  
  
 */\*\*  
 \* 带缓存数据的异步 post请求  
 \*/* public void postCache(String url, Map<String, String> maps, final boolean cache,  
 final XCallBack callback) {  
 RequestParams params = new RequestParams(url);  
 if (maps != null && !maps.isEmpty()) {  
 for (Map.Entry<String, String> entry : maps.entrySet()) {  
 params.addBodyParameter(entry.getKey(), entry.getValue());  
 }  
 }  
  
 x.*http*().post(params, new Callback.CacheCallback<String>() {  
 @Override  
 public void onSuccess(String result) {  
 onSuccessResponse(result, callback);  
 }  
  
 @Override  
 public void onError(Throwable ex, boolean isOnCallback) {  
  
 }  
  
 @Override  
 public void onCancelled(CancelledException cex) {  
  
 }  
  
 @Override  
 public void onFinished() {  
  
 }  
  
 @Override  
 public boolean onCache(String result) {  
 if (cache) {  
 onSuccessResponse(result, callback);  
 }  
 return cache;  
 }  
 });  
 }  
  
  
 */\*\*  
 \* 正常图片显示  
 \*/* public void bindCommonImage(ImageView iv, String url, boolean option) {  
 if (option) {  
 options = new ImageOptions.Builder().setLoadingDrawableId(R.mipmap.icon\_stub)  
 .setFailureDrawableId(R.mipmap.icon\_error).build();  
 x.*image*().bind(iv, url, options);  
 } else {  
 x.*image*().bind(iv, url);  
 }  
 }  
  
 */\*\*  
 \* 圆形图片显示  
 \*/* public void bindCircularImage(ImageView iv, String url, boolean option) {  
 if (option) {  
 options = new ImageOptions.Builder().setLoadingDrawableId(R.mipmap.icon\_stub)  
 .setFailureDrawableId(R.mipmap.icon\_error).setCircular(true).build();  
 x.*image*().bind(iv, url, options);  
 } else {  
 x.*image*().bind(iv, url);  
 }  
 }  
  
  
 */\*\*  
 \* 文件上传  
 \*/* public void upLoadFile(String url, Map<String, String> maps, Map<String, File> file,  
 final XCallBack callback) {  
 RequestParams params = new RequestParams(url);  
 if (maps != null && !maps.isEmpty()) {  
 for (Map.Entry<String, String> entry : maps.entrySet()) {  
 params.addBodyParameter(entry.getKey(), entry.getValue());  
 }  
 }  
 if (file != null) {  
 for (Map.Entry<String, File> entry : file.entrySet()) {  
 params.addBodyParameter(entry.getKey(), entry.getValue().getAbsoluteFile());  
 }  
 }  
 // 有上传文件时使用multipart表单, 否则上传原始文件流.  
 params.setMultipart(true);  
 x.*http*().post(params, new Callback.CommonCallback<String>() {  
 @Override  
 public void onSuccess(String result) {  
 onSuccessResponse(result, callback);  
 }  
  
 @Override  
 public void onError(Throwable ex, boolean isOnCallback) {  
  
 }  
  
 @Override  
 public void onCancelled(CancelledException cex) {  
  
 }  
  
 @Override  
 public void onFinished() {  
  
 }  
 }  
  
 );  
 }  
  
  
 */\*\*  
 \* 文件下载  
 \*/* public void downLoadFile(String url, Map<String, String> maps, final XDownLoadCallBack callBack) {  
  
 RequestParams params = new RequestParams(url);  
 if (maps != null && !maps.isEmpty()) {  
 for (Map.Entry<String, String> entry : maps.entrySet()) {  
 params.addBodyParameter(entry.getKey(), entry.getValue());  
 }  
 }  
 params.setAutoRename(true);// 断点续传  
 params.setSaveFilePath(PublicParams.SAVE\_FILE\_PATH);  
 x.*http*().post(params, new Callback.ProgressCallback<File>() {  
 @Override  
 public void onSuccess(final File result) {  
 handler.post(new Runnable() {  
 @Override  
 public void run() {  
 if (callBack != null) {  
 callBack.onResponse(result);  
 }  
 }  
 });  
 }  
  
 @Override  
 public void onError(Throwable ex, boolean isOnCallback) {  
  
 }  
  
 @Override  
 public void onCancelled(CancelledException cex) {  
  
 }  
  
 @Override  
 public void onFinished() {  
 handler.post(new Runnable() {  
 @Override  
 public void run() {  
 if (callBack != null) {  
 callBack.onFinished();  
 }  
 }  
 });  
 }  
  
 @Override  
 public void onWaiting() {  
  
 }  
  
 @Override  
 public void onStarted() {  
  
 }  
  
 @Override  
 public void onLoading(final long total, final long current,  
 final boolean isDownloading) {  
 handler.post(new Runnable() {  
 @Override  
 public void run() {  
 if (callBack != null) {  
 callBack.onLoading(total, current, isDownloading);  
 }  
 }  
 });  
 }  
 });  
 }  
  
 */\*\*  
 \* 异步get请求返回结果,json字符串  
 \*/* private void onSuccessResponse(final String result, final XCallBack callBack) {  
 handler.post(new Runnable() {  
 @Override  
 public void run() {  
 if (callBack != null) {  
 callBack.onResponse(result);  
 }  
 }  
 });  
 }  
  
  
 public interface XCallBack {  
 void onResponse(String result);  
 }  
  
  
 public interface XDownLoadCallBack extends XCallBack {  
 void onResponse(File result);  
  
 void onLoading(long total, long current, boolean isDownloading);  
  
 void onFinished();  
 }  
  
}

差点忘了，还有一个需要配置的，那就是网络权限

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

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