StringObjectRequest

1、相对简单，不直接支持多界面的刷新，也就是一个任务只可以刷新一个界面。

2、没有采用线程池，直接采用队列RequestQueue、并实现了线程数组

3、相对简单

RequestQueue使用流程

RequestQueue是多任务的核心

1、第一步

mVolleyQueue = Volley.newRequestQueue(this,new SslHttpStack(true));
其中直接创建了NetworkDispatcher[]作为线程池

2、第二步：创建请求，其中内涵回调函数，由于是String类的请求，所以返回中的result时String类型

StringRequest stringRequest = new StringRequest(Request.Method.GET, url, new Response.Listener<String>() {}
stringRequest.setShouldCache(true);
stringRequest.setTag(TAG_REQUEST);
mVolleyQueue.add(stringRequest);

3、类型的区分恐怕是依靠StringRequest、JsonRequest等来区分的

```
public RequestQueue(Cache cache, Network network, int threadPoolSize, ResponseDelivery delivery)
{
        this.mSequenceGenerator = new AtomicInteger();
        this.mWaitingRequests = new HashMap();
        this.mCurrentRequests = new HashSet();
        this.mCacheQueue = new PriorityBlockingQueue();
        this.mNetworkQueue = new PriorityBlockingQueue();
        this.mCache = cache;
        this.mNetwork = network;
        this.mDispatchers = new NetworkDispatcher[threadPoolSize];
        this.mDelivery = delivery;}
public RequestQueue(Cache cache, Network network, int threadPoolSize) {
    this(cache, network, threadPoolSize, new ExecutorDelivery(new
Handler(Looper.getMainLooper())));}

public class ExecutorDelivery implements ResponseDelivery { 作用是用来派发结果
```

请求的发送

NetWorkDispatcher【多线程逻辑】

请求派发，实际是一个线程
线程内部请求并获取结果，利用泛型发送派发结果

```
NetworkResponse e = this.mNetwork.performRequest(request);

不同的Requet利用自己的逻辑解析自己的返回数据

Response response = request.parseNetworkResponse(e);

private byte[] entityToBytes(HttpEntity entity) throws
IOException, ServerError {
          ; ; ; ; ;
           int count;
 //由于存在压缩，所以不要根据长度，而要根据-1来判断，

           while((count = in.read(buffer)) != -1) {
               bytes.write(buffer, 0, count);
           }

       bytes.close();

数据的读取是必须的，无论是文件下载还是String JsonString获取，都是必须的
```

BasicNetwork network1 = new BasicNetwork((HttpStack)stack);
RequestQueue queue1 = new RequestQueue(new DiskBasedCache(cacheDir), network1);

缓存机制：直接采用了Map，而没用数据库之类的永久缓存机制，所以说，App重启之后，就重新清空缓存。

也是采用了Key与File结合的方式来实现

结果的回调

ExecutorDelivery【多线程】

```
public ExecutorDelivery(final Handler handler) {
   this.mResponsePoster = new Executor() {
       public void execute(Runnable command) {
           handler.post(command);
       }
   };
}

this.mResponsePoster.execute(new ExecutorDelivery.ResponseDeliveryRunnable(request, response,
(Runnable)null));
private class ResponseDeliveryRunnable implements Runnable {

其中的Handler时关键，是刷新UI的核心，保证异步与界面刷新的关系
```