该框架的核心是：

1、两个Transaction引擎

2、DataChannel兼有回调接口

3、NotifyTransaction直接有Transaction的句柄引用 直接可以调用mTran.onTransactionSuccess(mCode, mData);

4、TransactionMap管理本地列表

5、成功后更新界面的引擎就放在了onEgmTransactionSuccess的notifyMessage中

6、在这里

```
public void notifyMessage(int msgCode, Object arg3) {
          TransactionListener listener = getListener();

          if (listener != null) {
               listener.onTransactionMessage(msgCode, mType, mId, arg3);
          }
     }
```

其中每个Transaction自己保存自己的ID 事务类型等

7、如何保证的界面更新（mHandler是在主线程中创建的，这就保证了界面的更新）

```
     @Override
     synchronized public void onTransactionMessage(int code, int type, int tid, Object obj) {

          if(EgmServiceCode.TRANSACTION_SUCCESS != code)
               return;

          ResultInfo info = new ResultInfo(code,obj);
          Message message = mHandler.obtainMessage(TRANSACTION_SUCCESS, type, tid, info);
          message.sendToTarget();
     }
```

入手中线，发送一个请求，构造一个任务

开始契机

```
public int doGetRankListInfoInHome(String userId){
        GetRankInfoInHomeTransaction t = new GetRankInfoInHomeTransaction(userId);
        beginTransaction(t);
       return t.getId();
    }
```

EgmService（后台心脏）

```
 private EgmService() {

                         通信通道、通信的引擎
 super(new HttpDataChannel(new TransactionEngine(TransactionEngine.Priority,
CoreThreadCount), new HttpEngine(
                3, Thread.NORM_PRIORITY - 1)));
        mGroupListener = new GroupTransactionListener();
        mWattingLoginList = new LinkedList<Transaction>();
    }

@Override
    public int beginTransaction(Transaction trans) {
        trans.setListener(mGroupListener);
        return super.beginTransaction(trans);
    }
    public void addListener(EgmCallBack listener){
        mGroupListener.addListener(listener);
    }

    public void removeListener(EgmCallBack listener){
        mGroupListener.removeListener(listener);
    }
```

```
class GroupTransactionListener implements TransactionListener {
     WeakHashMap<EgmCallBack, Void> mCallbacks =
            new WeakHashMap<EgmCallBack, Void>();
     Handler mHandler = new InternalHandler();
```

TransactionEngine（Transaction的多线程永动机）内含了HttpEngine，网络通信的永动机

```
     public TransactionEngine(int priority, final int coreThreadCount) {
          mTransactionQueue = new LinkedBlockingQueue<Runnable>();
          mTransactionMap = new ConcurrentHashMap<Integer, Transaction>();

          mThreadPool = new ThreadPoolExecutor(coreThreadCount, 30, 2000,
                    TimeUnit.MICROSECONDS, mTransactionQueue);
          mThreadPool.setThreadFactory(new ThreadFactory() {

               @Override
               public Thread newThread(Runnable r) {
                    Thread thread = new Thread(r);
                    thread.setPriority(coreThreadCount);
                    return thread;
               }
          });
     }

     public void beginTransaction(Transaction tx) {
          if (tx != null) {
               tx.setTransactionEngine(this);
               Integer key = Integer.valueOf(tx.getId());

               mTransactionMap.put(key, tx);

               try {
                    mThreadPool.execute(tx);
               } catch (RejectedExecutionException e) {
                    e.printStackTrace();
                    if (tx instanceof AsyncTransaction) {
                         ((AsyncTransaction) tx).onTransactionError(0, null);
                    }
                    else {
                         tx.notifyError(0, null);
                    }
               }
          }
     }
```

HttpDataChannel（血管、数据通道）

Baseservice

```
public BaseService(DataChannel dataChanel) {
          mDataChanel = dataChanel;
          mTransactionEngine = dataChanel.getTransactionEngine();
     }

public int beginTransaction(Transaction trans) {
          int ret = -1;

          if (trans != null) {
               ret = trans.getId();
               if (trans instanceof AsyncTransaction) {
                    ((AsyncTransaction) trans).setDataChannel(mDataChanel);
               }
               if (mTransactionEngine != null) {
                    mTransactionEngine.beginTransaction(trans);
               }
          }return ret;
     }
```

```
public abstract class Transaction implements Runnable, Comparable<Transaction> {

     public void run() {
          try {
               if ( !isCancel()) {
                    onTransact();
               }
          } catch (Exception e) {
               e.printStackTrace();

               doCancel();
               onTransactException(0, e);
          }

          mTransMgr.endTransaction(this);
     }
     /**
      * Transaction 任务执行入口
      */
     public abstract void onTransact();
     /**
```

```
class GetRankInfoInHomeTransaction extends EgmBaseTransaction

   public void onTransact() {
  THttpRequest request =EgmProtocol.getInstance().createGetRankListInHomeRequest(mUserId);
      sendRequest(request);
   }

   protected void onEgmTransactionSuccess(int code, Object obj) {
      RankListInfoInHome mRankListInfo = null;
      if (obj != null && obj instanceof JsonElement) {
         Gson gson = new Gson();
         mRankListInfo = gson.fromJson((JsonElement)obj, RankListInfoInHome.class);
      }
      if (mRankListInfo.rankInfoList != null) {
         notifyMessage(EgmServiceCode.TRANSACTION_SUCCESS, mRankListInfo);
      } else {
         notifyDataParseError();
      }
   }
```

```
public class EgmHttpRequest extends THttpRequest {
    public static String APP_OS_NAME  = "androidPhone";
    public static final String USERAGENT =  "date/"
            + EgmUtil.getNumberVersion(BaseApplication.getAppInstance()) + " ("
            + APP_OS_NAME + "; " + android.os.Build.VERSION.RELEASE + "; " + android.os.Build.MODEL+ "; "
            + PlatformUtil.getResolution(BaseApplication.getAppInstance())+ "; "
            + EgmUtil.getAppChannelID(BaseApplication.getAppInstance()) + ") ";

    public EgmHttpRequest(String url) {
        super(url);
        init();

    }

    public EgmHttpRequest(String url, THttpMethod type) {
        super(url, type);
        init();
    }
    public void init() {
        addHeader("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
        addHeader("Accept-Encoding", "gzip, deflate");
        addHeader("PLATFORM", String.valueOf(EgmProtocolConstants.PLATFORM_ANROID));
        addHeader("APPVERSION",EgmUtil.getNumberVersion(BaseApplication.getAppInstance()));
        addHeader("TOKEN",EgmProtocol.getInstance().getUrsToken());
        addHeader("DEVICEID",EgmProtocol.getInstance().getUrsId());
        addHeader("user-agent",USERAGENT);
        addHeader("CHANNEL",EgmUtil.getAppChannelID(BaseApplication.getAppInstance()));
        addHeader("PHONETYPE",android.os.Build.MODEL);
        addHeader("MEID",PlatformUtil.getDeviceID(BaseApplication.getAppInstance()));
    }
```

```
    /** 获取主页面排行榜信息 */
    public THttpRequest createGetRankListInHomeRequest(String userId){
        EgmHttpRequest request = new
EgmHttpRequest(getRequestUrl(URL_GET_RANK_LIST_IN_HOME), THttpMethod.GET);
        request.setUrlLocalParam(userId);
        request.setCacheDatabase();
        request.setCacheFile();

        return request;
    }
```

```
public abstract class AsyncTransaction extends Transaction {
protected void sendRequest(Object obj) {
          if (mDataChannel != null) {
               mDataChannel.sendRequest(obj, this);
          }
     public void setDataChannel(DataChannel dataChannel) {
          mDataChannel = dataChannel;
     }
     protected void sendRequest(Object obj, AsyncTransaction trans) {
          if (mDataChannel != null) {
               mDataChannel.sendRequest(obj, trans);
          }
     }
```

```
     public HttpEngine(int httpThread, int threadPriority) {
          mRunningRequests = new HashMap<String, List<THttpRequest>>();
          mRequestQueue = new PriorityBlockingQueue<THttpRequest>();
          mHttpThreads = new LinkedList<HttpThread>();

          mBytePool = new ByteArrayPool(8 * 1024);

          for (int i = 0; i < httpThread; i++) {
               HttpThread thread = new HttpThread();
               thread.setPriority(threadPriority);
               thread.start();

               mHttpThreads.add(thread);
          }
     }
```

```
     private class HttpThread extends Thread {

          @Override
          public void run() {
               while (! mIsClosed) {
                    try {
                         THttpRequest request = mRequestQueue.take();
                         execute(request);
                    } catch (InterruptedException e) {
                         e.printStackTrace();
                    }
               }
          }
```

关于存储这块需要看看，数据已经到达了吗？还是数据未达到，仍然需要区网络读取呢？

```
                    try {
                         http = getHttp();
                         response = http.executeRequest(request);

                         Object ret = null;

                         if (response != null) {
                              int respCode = response.getResponseCode();
                              if (respCode == 304) {
                                   notifyReceived(request, respCode, httpCache);
                                   break;
                              }
```

成功之后

```
private boolean onRequestNotify(THttpRequest request, Object object,
                    int notifyType, int code) {

          AsyncTransaction tran = null;
          Integer key = Integer.valueOf(request.getRequestID());

          tran = mTransactionMap.remove(key);

          if (tran != null) {
               mRequestMap.remove(Integer.valueOf(tran.getId()));
          }

          boolean ret = false;
          if (tran != null) {
               ret = true;
               NotifyTransaction notify = tran.createNotifyTransaction(object,
                         notifyType, code);
               mTransactionEngine.beginTransaction(notify);
          }

          return ret;
     }
```

NotifyTransaction可以直接获取相目标Transaction应的引用，通知其内容

```
public final void onTransact() {
          doBeforeTransact();

          boolean doEnd = isNeedEnd();

          if (mTrans == null) {
               if (doEnd) {
                    mTran.doEnd();
               }
               if (!mTran.isCancel()) {
                    try {
                         if (mNotifyType == NOTIFY_TYPE_SUCCESS) {
                              mTran.onTransactionSuccess(mCode, mData);
                         } else {
                              mTran.onTransactionError(mCode, mData);
                         }
                    } catch (Exception e) {
                         e.printStackTrace();
                    }
               }
          }
```

NotifyTransaction一个Transanction一个通知，一一对应的关系，直接获取，设计模式（依赖倒置）

有关数据的存储

数据的存储：首先获取的返回可以看做是个接口，如果是图片还要通过读取数据流进行获取图片内容。此外，这还有缓存之类的问题。

首先读取一个头，然后读取其余的？？？

HTTP协议部分

```
 public void init() {
        addHeader("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
        addHeader("Accept-Encoding", "gzip, deflate");
        addHeader("PLATFORM", String.valueOf(EgmProtocolConstants.PLATFORM_ANROID));
        addHeader("APPVERSION",EgmUtil.getNumberVersion(BaseApplication.getAppInstance()));
        addHeader("TOKEN",EgmProtocol.getInstance().getUrsToken());
        addHeader("DEVICEID",EgmProtocol.getInstance().getUrsId());
        addHeader("user-agent",USERAGENT);
        addHeader("CHANNEL",EgmUtil.getAppChannelID(BaseApplication.getAppInstance()));
        addHeader("PHONETYPE",android.os.Build.MODEL);
        addHeader("MEID",PlatformUtil.getDeviceID(BaseApplication.getAppInstance()));
    }
```