Loaders 使用

一、 实验目的

1. 了解和掌握 loader 的原理，机制。
2. 学习和掌握 loader 的使用。

二、 实验条件

PC 机

JDK（建议安装 JDK8 及其以上版本）、Android Studio

三、 实验原理

1、Loader

Loader 是一种异步处理数据的类。当 Loaders 时 active 时，它们会管理数据源、当内容发生改变会发送新的结果。新的结果会发送到注册的监听器上（通常是 LoaderManager）。

2、LoaderManager

LoaderManager 用来负责管理与 Activity 或者 Fragment 联系起来的一个或多个 Loaders 对象。每个 Activity 或者 Fragment 都有唯一的一个 LoaderManager 实例，用来启动、停止、保持、重启、关闭它的 Loaders。这些事件有时直接在客户端通过调用 initLoader()/restartLoader()/destroyLoader()函数来实现。通常这些事件通过主要的 Activity/Fragment 声明周期事件来触发，而不是手动（当然也可以手动调用）。比如，当一个 Activity 关闭时（destroyed），改活动将指示它的 LoaderManager 来销毁并且关闭它的 Loaders（当然也会销毁并关闭与这些 Loaders 关联的资源，比如 Cursor）。

3、实现 LoaderManager.LoaderCallbacks<D>接口

LoaderManager.LoaderCallbacks<D>接口 LoaderManager 用来向客户返回数据的方式。每个 Loader 都有自己的回调对象供与 LoaderManager 进行交互。

```
public class SampleActivity extends Activity implements LoaderManager.LoaderCallbacks<D> {

  public Loader<D> onCreateLoader(int id, Bundle args) { ... }

  public void onLoadFinished(Loader<D> loader, D data) { ... }

  public void onLoaderReset(Loader<D> loader) { ... }

  /* ... */
}
```

(1)onCreateLoader 是用来返回一个新的 Loader。LoaderManager 将会在它第一次创建 Loader 的时候调用该方法，需要在这边创建一个新的 Loader。一般是创建 CursorLoader，也可以创建自己定义的 Loader,基础 AysnctaskLoader。

(2)onLoadFinished 方法将在 Loader 创建完毕的时候自动调用。典型用法是，当载入数据完毕，客户端（译者注：调用它的 Activity 之类的）需要更新应用 UI。客户端假设每次有新数据的时候，新数据都会返回到这个方法中。记住，检测数据源是 Loader 的工作，Loader 也会执行实际的同步载入操作。一旦 Loader 载入数据完成，LoaderManager 将会接受到这些载入数据，并且将将结果传给回调对象的 onLoadFinished 方法，这样客户端（比

如 Activity 或者 Fragment）就能使用该数据了。

(3)当 Loader 们的数据被重置的时候将会调用 onLoadReset。该方法让你可以从就的数据中移除不再有用的数据。

4、AsynctaskLoader<D>

自定义实现 Loader。

必须有公共构造函数 public mLoader(Context context){super(context)} 和 loadInBackground()。

D 是数据类，一般用于定义你要返回的数据类型。注意：loadInBackgrond()的类型和 D 的类型必须一致！

四、 实验步骤

在课程讨论的 APP 程序中,修改 MainActivity.java 中:

① 初始定义 private Context mContext =null;

② 在适当位置（e.g.末尾）插入 LoadManager.callback 方法:

```
    private class mManagerLoader implements LoaderManager.LoaderCallbacks<String>{
        @Override
        public Loader onCreateLoader(int i, Bundle bundle) {
            return new mLoader1(mContext);
        }
        @Override
        public void onLoadFinished(Loader<String> stringLoader, String s) {

/*
这里要把 hander 的内容插入到这边，意思是当你创建的 AsynctaskLoader 的任务执行完成之后，就会跳到这边执行。注意这边的 String s 就是你 AsynctaskLoader 中 loadInBackground 的返回值。
把 msg.what 的判断改为对 s 的判断。
*/

        }
        @Override
        public void onLoaderReset(Loader loader) {

        }
    }
```

③ 再插入 AsynctaskLoader 继承类

```
    private static class mLoader1 extends AsyncTaskLoader<String>{
        public mLoader2(Context context){super(context);}

        @Override
        protected void onStartLoading() {
            super.onStartLoading();
```

```
            forceLoad();
        }

 /*
下面函数蓝色部分就是把之前 onOptionItemSelected()中的 case R.id.HttpGet 下的 new Thread()(……)里面的内容搬过来。
不过要记得把相对应的变量变成 static 类型；把 mHandle…替换成 load_flags 的操作；把 MainActicity.this 替换成 getContext();
 */
        @Override
        public String loadInBackground() {
         //这边要定义一个 String 的标志变量，用来 onLoadFinished 的操作。
            String load_flags = null;
            try {
//                                                  String respond = new WebAccessTools(MainActivity.this).doHttpsGet(URL + "?name=query");
                    String respond = new WebAccessTools(getContext()).javaHttpsGet(URL + "?name=query");
                    Log.v("response",respond);
                    if(respond != "nothing"){

                        JSONArray json=new JSONArray(respond);  // 如果服务器连接失败，这句将抛出异常，而不是 doHttpsGet,respond == [ 04-10 11:19:41.635 3212: 3410 W/System.err ]

                        datas.clear();
                        datas_string.clear();
                        for(int i =0;i<json.length();i++){
                            JSONObject obj = json.getJSONObject(i);
                            Log.i("name",obj.getString("name"));
                            Log.i("content",obj.getString("content"));
                            Log.i("start",obj.getString("start"));
                            Log.i("stop",obj.getString("stop"));

                            timeset_1 = Long.parseLong(obj.getString("start"));
                            boolean added = false;

                            if (datas.size() == 0) {
                                new_added = 0;
                                added = true;
                                Log.i(TAG, "datas.size() == 0");
                            } else {
                                for (int j = 0; j < datas.size(); j++) {
                                    if (timeset_1 <= datas.get(j)[0]) {
```

```
                                    new_added = j;
                                    added = true;
                                    Log.i(TAG, "i == " + j + "时 break");
                                    break;
                                }
                            }
                            if (!added) {
                                new_added = datas.size();
                            }
                        }

                        timeset_2 = Long.parseLong(obj.getString("stop"));
                        Long[] long_tmp = {timeset_1, timeset_2};
                        String[]    string_tmp    =    {obj.getString("name"),
obj.getString("content")};
                        datas.add(new_added, long_tmp);
                        datas_string.add(new_added, string_tmp);
                    }
                    load_flags = "2";
                    //mHander.obtainMessage(2).sendToTarget();
                    saveArray(datas, datas_string);
                    Log.i(TAG, "datas.add");
                }else{
                    datas.clear();
                    datas_string.clear();
                    load_flags = "2";
                    //mHander.obtainMessage(2).sendToTarget();
                    saveArray(datas, datas_string);
                }
            } catch (Exception e) {
                load_flags = "3";
                //mHander.obtainMessage(3).sendToTarget();
                e.printStackTrace();
            }
              return load_flags;
        }
    }
```

④ 增加初始定义 private mManagerLoader mLoader = new mManagerLoader();

⑤ 在 onCreate()里面初始化 loader:

增加 getLoaderManager().initLoader(0,null,mLoader);

//如果你要用到多个 loader，这边就要初始化多个

//参数意义：

第一个代表你定义 loader 的序列号，一般为 0；

第二个一般为 null；

第三个就是你要用到的 AsynctaskLoader(或者 CursorLoader)

⑥ 把 onOptionItemSelected()中的 case R.id.HttpGet 下的 new Thread()(……)(整个一大串函数）替换成 getLoaderManager().restartLoader(0,null,mLoader)；
//参数意义同上。

本实验的效果可以通过 Log 输出来判断是否使用成功，也可以通过 App 的 UI 操作进行判别。

五、 实验报告内容与要求

实验报告中要包含以下几个部分：

1、实验目的
2、实验条件
3、实验原理
4、实验步骤分析
5、实验结果与总结
6、实验思考题

实验步骤要详细，关键步骤要有截图，运行结果也要有截图。

六、 思考题

1、 把 APP 中加号操作（创建新的讨论组）也用 LoaderManger 处理。

参考资料

http://blog.csdn.net/murphykwu/article/details/35287883
http://developer.android.com/reference/android/app/LoaderManager.html
http://www.cnblogs.com/wchhuangya/p/3277148.html