数据解析工具设计文档

学习总结　多线程　数据解析

一、设计目标

- 设计一个相对通用的数据解析工具，高效解析通过抓包工具抓取的网络数据内容
- 只需要添加少量代码，实现不同数据格式的解析，并支持不同文件格式保存

二、实施步骤

总的来说，就是使用MonkeyRunner自动测试脚本对手机进行某一些具体的操作时，通过Fiddler工具截取手机App的Http通讯数据，通过对这些数据进行解析从而达到获取应用数据的过程。

2.1、编写MonkeyRunner自动脚本

- 环境和工具：Python，Android SDK开发环境， Android手机一台（debug选项连接PC）
- 脚本目标：循环对手机屏幕进行上划操作
- 代码：

```
#coding=utf-8
from com.android.monkeyrunner import
MonkeyRunner,MonkeyDevice,MonkeyImage
device = MonkeyRunner.waitForConnection(5,'351BBJHUHDPU')
sleepTime = 0.2
for x in xrange(1,10000):
    MonkeyRunner.sleep(sleepTime)
    device.drag((436,1234),(378,346),1,10)
```

- 注：MonkeyRunner.waitForConnection方法参数为adb devices 下的手机设备名

2.2、抓取应用流量信息

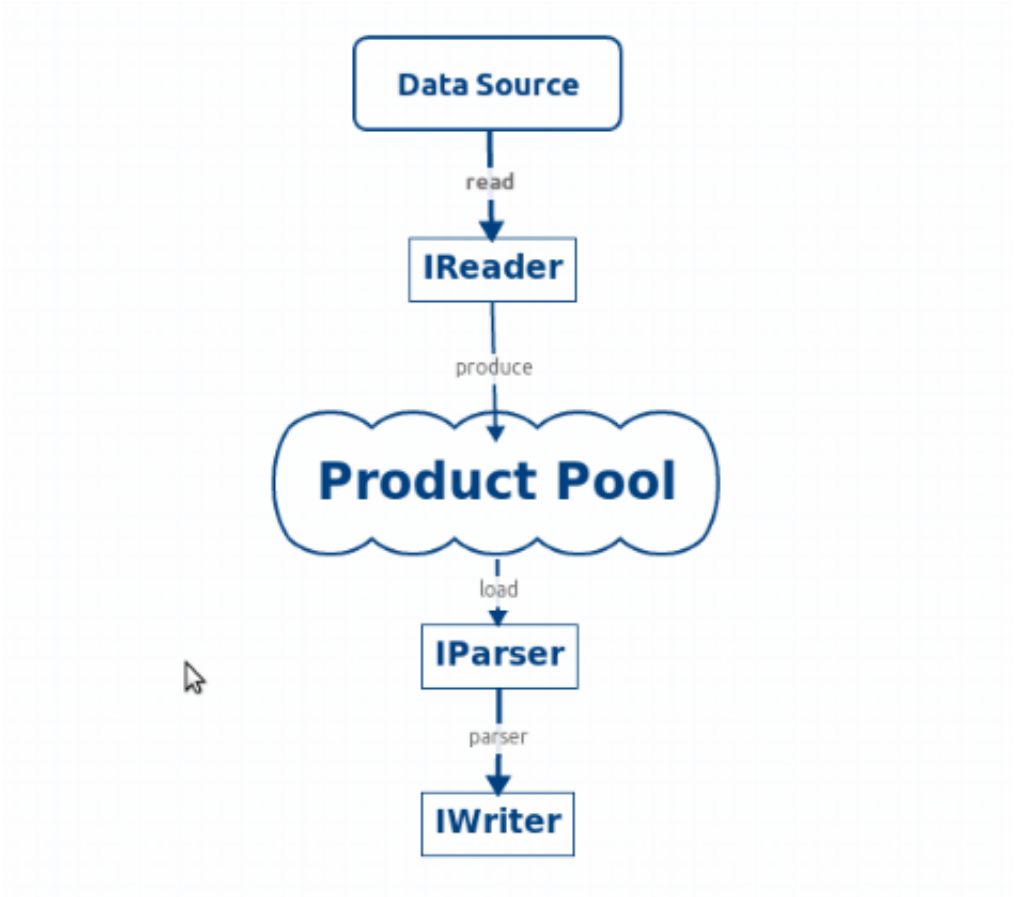
- 下载并安装需要抓取数据的Android应用
- Google搜索获得Android手机设置代理上网方法，并成功设置手机通过PC代理上网功

能

- 安装Fiddler，通过设置Fiddler host过滤可以获取固定host的HTTP Request和Response
- 使用MonkeyRunner测试脚本，循环触发手机上划操作，自动浏览手机应用商店的应用分类界面，通过Fiddler抓取手机的HTTP请求和响应消息内容，并导出

2.3、数据解析实现

2.3.1、数据处理流程



2.3.2、使用生产者消费者模式处理数据

生产者线程读取源数据文件，生产出待解析的字符串数据放入数据池中。

```
/**
  * 生产者的生产容器
  */
private LinkedList<Object> mProductPool = new LinkedList<Object>
();

private void addProduct(List<Object> data) {
    while (!data.isEmpty()) {
        synchronized (mProductPool) {
             while (mProductPool.size() >= MAX_SIZE) {
                try {
                    mProductPool.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }


        getDataProducer().startProduce();

        for (int i = 0; i < Math.min(MAX_SIZE, data.size()) &&
                mProductPool.size() < MAX_SIZE; i++) {
                if (!data.isEmpty()) {
                    mProductPool.add(data.get(0));
                    data.remove(0);
                }
            }
            mProductPool.notifyAll();
        }
    }
}
```

多个消费者线程每次从数据池中获取一定数量的数据内容用来解析，并把解析结果放入最终的处理结果池中。这里并没有使用二级的消费者去做数据保存操作，因为可能需要把所有数据解析完成后做统一的分类、统计、筛选操作。

```
    /**
      * 消费者处理数据后的容器
      */
    private List<Unit> mConsumedPool = new ArrayList<Unit>();

    public class Consumer implements Runnable {

        @Override
        public void run() {
```

```
            while (true) {
                List<Object> queue = mEngine.getProductPool();
                if (queue == null) {
                    return;
                }

                List<Object> temp = new ArrayList<Object>();
                synchronized (queue) {
                    while (queue.size() == 0 &&
!mEngine.isProduceComplete()) {
                        try {
                            queue.wait();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                            queue.notify();
                        }
                    }

                    int size = queue.size();
                    int k = Math.max(0, size - NUM_PER_CONSUME);
                    for (int i = size - 1; i >= k; i--) {
                        temp.add(queue.get(i));
                        queue.remove(i);
                    }

                    queue.notifyAll();
                }

                for (Object str : temp) {
                    ArrayList<Unit> result = onParse((String)
str);
                    if (result != null) {
                        mEngine.addParsedUnit(result);
                    }
                }

                if (mEngine.isProduceComplete() &&
queue.isEmpty()) {
                    complete(this);
                    Thread t = Thread.currentThread();
                    t.interrupt();
                    return;
                }
            }
        }
    }
```

Engine.java作为整个解析过程的引擎，不同的数据需要不同的解析实现和保存格式实现,使用Reader读取数据的数据，将读取的数据填充到数据池中，当池满时，Reader线程阻塞当池空时，解析线程阻塞。在数据全部读取并解析完成之后，通过IWriter将结果保存到文件

```
public class Engine implements IParser.ParseListener {
    // 指定解析的数据路径
    public void setSourcePath(String path) {}
    // 设置Reader
    public void setReader(IReader reader) {}
    // 设置解析器
    public void setParser(IParser parser) {}
    // 设置Writer
    public void setWriter(IWriter writer) {}
    // 解析器解析完成的回调
    public void onParseComplete() {}
    // 启动，开始处理数据
    public void start()
}
```

三、针对第三方应用商店分类数据的解析

2.1、定义不同类型的解析器

在分析第三方商店数据的过程中发现，每个应用市场都会选择性的屏蔽竞品信息的行为，例如除非通过搜索功能，你很难在360应用商店看到直接百度的应用。淘宝应用商店的屏蔽行为相对较少，但是会刻意提高一些推广应用的下载排名。很难在应用商店中抓取完全的应用信息，可能需要解析一些自己手动收集的应用分类信息。

```
//做具体针对淘宝数据的JSON字符串进行数据解析
public class TBParser extends IParser {
    @Override
    public ArrayList<Unit> onParse(String source) {
    }
}
//做具体针对360的JSON字符串进行数据解析
public class QHParser extends IParser {
    @Override
    public ArrayList<Unit> onParse(String source) {
    }
}
// 针对自己定义的数据进行解析
public class CustomParser extends IParser {
    @Override
    public ArrayList<Unit> onParse(String source) {
    }
}
```

2.2、Zip压缩和Huffman编码数据压缩的比较

针对ZIP压缩，java提供了标准的接口
针对Huffman编码压缩文本的原理，可以参照这里,具体实现在工程代码在：
com.ivonhoe.parser.huffman.HuffmanCode.java的实现
比较发现针对字符的压缩，直接ZIP压缩比先Huffman压缩后ZIP压缩的压缩比率高。

2.3、保存文本格式

保存的数据文件包含两个部分,对应的文件：/data/app/category

- 第一部分表示各种分类数据的条目数量
  {健康运动:81}
  {通讯社交:326}
  {生活休闲:485}
  ……
- 第二部分表示所有的包名，按照文件开始部分的分类顺序和条目数依次保存
  com.hk515.patient
  com.guokr.zhixing
  com.yidian.health
  ……
- 如上所示，健康运动应用数量81个，分别是第二部分的1~81条目，通讯社交应用有326个，分别是第二部分的82~408条目，依次类推。

四、针对京东商品评论数据的解析

针对京东的评论数据信息，通过继承IParser的接口实现具体针对京东数据的解析方式。
针对解析数据的保存，通过继承IWriter的保存接口，实现针对Excel文件格式的保存

```
**
 * 针对JD的评论数据解析
 *
 * @author ivonhoe
 */
public class JDWebParser extends IParser {
    @Override
    public ArrayList<Unit> onParse(String source) {
    }
}

**
 * 通过接口保存到Excel表格中
 *
 * @author ivonhoe
 */
public class JDCommentWriter extends ExlWriter {
    @Override
    public void onWrite(List<Unit> list) {
        // TODO Auto-generated method stub
        exportToSheet("comments", TITLE_STRINGS, list);
    }
}
```

五、总结

- 在python部分，在实现上只需要通过简单的循环让手机屏幕不停的滑动就可以了。针对移动端数据传输的特点，绝大多数的客户端应用和服务器数据传输的格式都是通过JSON或者XML。你不再需要想办法在爬取网站页面的基础上进行数据收集，不再需要考虑如何从繁杂的网页上筛选你想要的正文。通过抓包分析的方式，这就像一个漏斗，你不需要再去筛选数据。你拿到的都是格式化的数据了。
- 如果能够集合一些更智能的自动化测试工具如百度的MTC测试工具，录制一些特殊的操作脚本，那应该可以做更多的事情。
- 如果针对复杂的业务需求，可以增加二级的消费者针对数据进行进一步的处理。
- 针对应用分类数据的筛选，因为不同数据源的信息（如下载量）也并不是完全可信的。还需要做进一步的筛选。目前的做法是根据不同数据源进行加权平均取排名的方式（测试效果并不好），或者根据数据源的并集和应用下载量的排名综合筛选，应该可以有更好的方式。