**电影信息爬虫系统说明书**

**（V1.0）**

|  |  |
| --- | --- |
| 编写人： | 张晓瑞 |
| 版本： | V1.0 |
| 编写日期： | 2015年3月10日 |

目 录

[1 文档介绍](#_Toc390334731)

[1.1 文档目的 3](#_Toc390334732)

[1.2 文档范围 3](#_Toc390334733)

[1.3 读者对象 3](#_Toc390334734)

[1.4 名词和缩写语 3](#_Toc390334735)

[2 系统总体设计](#_Toc390334736)

[2.1 系统架构设计 4](#_Toc390334737)

[2.2 系统模块划分 5](#_Toc390334738)

[3 系统详细说明](#_Toc390334739)

[3.1 系统流程图 7](#_Toc390334740)

[3.2 爬取目标的确定 10](#_Toc390334741)

[3.3 爬行策略 12](#_Toc390334742)

[3.4 网页解析 16](#_Toc390334743)

[3.5 URL处理 18](#_Toc390334744)

[3.6 增量爬行 21](#_Toc390334745)

[3.7 线程检测重启 21](#_Toc390334746)

[4 系统运行](#_Toc390334747)

[4.1 系统运行环境 23](#_Toc390334748)

[4.2 系统配置 23](#_Toc390334749)

[4.3 系统启动与结束 25](#_Toc390334750)

[4.4 爬取商品展示 28](#_Toc390334751)

[4.5 系统实现文件说明 29](#_Toc390334752)

# 1 文档介绍

## 1.1 文档目的

此文档是为了让读者了解电影信息爬虫的设计原理、所使用的技术以及各组成部分。通过本文档，可以快速理解、掌握电影信息爬虫的实现细节，从而为其添加新功能或构建其它用途爬虫打下基础。

## 1.2 文档范围

本文档描述整个爬虫系统所用到的技术以及组成部分，细化到影片类型确定、爬行策略、网页解析等具体实现。

## 1.3 读者对象

电影信息爬虫系统的使用者、开发者。

## 1.4 名词和缩写语

| **序号** | **名词及缩略语** | **定义** |
| --- | --- | --- |
| 1 | 系统 | 电影信息爬虫系统 |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |

# 2 系统总体设计

本章重点阐述电影信息爬虫系统的总体设计和各模块的划分，为下一章详细叙述系统的实现打下理论基础。

## 2.1 系统架构设计

要想更好更快地实现电影信息爬虫系统，需要对系统进行总体设计，然后再逐步细分功能。系统的总体设计主要包括系统架构设计和系统各功能模块的划分两个部分。



图2.1分布式架构图

考虑到集中式与分布式架构的优缺点，电影信息爬虫系统的架构图如图2.2所示，该架构是一种集中式与分布式的混合架构。该系统架构与图2.1中的分布式架构图最显著的区别是多一个中心结点。该中心节点负责爬虫的加入和退出、各爬行结点信息的维护，爬虫运行状态的监控，并进行适当的任务调度以达到负载均衡。其它任务均在爬行结点上处理，特别是与爬行密切相关的操作，如发送URL请求、URL去重，链接分发等。该架构吸取了集中式架构和分布式架构的优点，但本质上还是分布式架构，区别在中心节点（也可以称为管理节点）的个数。图2.1这种分布式架构准确来说应该称作完全分布式架构，即每个爬行节点也是中心节点，不仅负责爬取页面、提取链接，还负责管理自己、与外界通信；而图2.2中的分布式则只有一个中心节点，只负责简单的任务调度，爬行监控，从而不会成为系统的瓶颈，同时又保证了系统的扩展性。这样做的最大好处是容易实现，也便于管理维护。这在谷歌公司众多分布式系统上得到体现。



图2.2 带中心节点的分布式系统架构图

## 2.2 系统模块划分

针对电影信息爬虫的特点，在传统的网络爬虫基础上，系统各模块设计如图2.3所示。各模块主要围绕着对URL的处理运转，模块之间的交互通过箭头表示，箭头指向的方向表示被调用模块。同时，图中还给出了各模块包含的子功能。

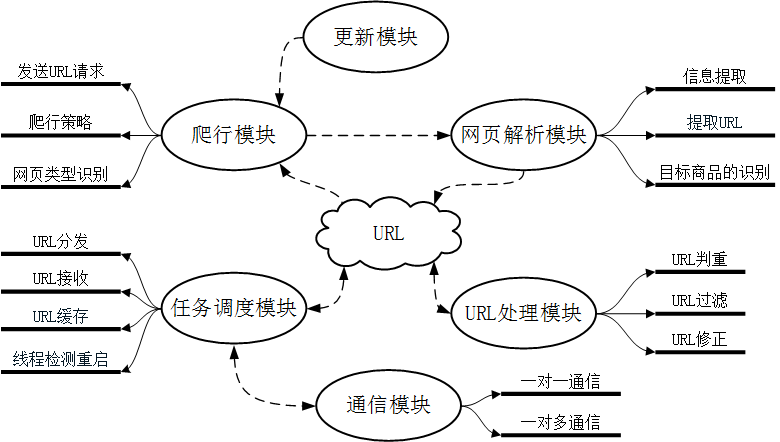


图2.3 系统各模块关系图

# 3 系统详细说明

本章重点阐述电影信息爬虫系统实现中使用的技术。

## 3.1 系统流程图

### 3.1.1 系统运行流程图

图3.1为系统运行流程图。流程图介绍系统执行过程中的主要部分。爬行系统一般都是不停地循环执行，除非遇到特殊情况，一般不存在结束的情况。图中画出的结束需要人为触发。图3.2为系统结束流程图，主要包括结束各线程，并保存相关信息。

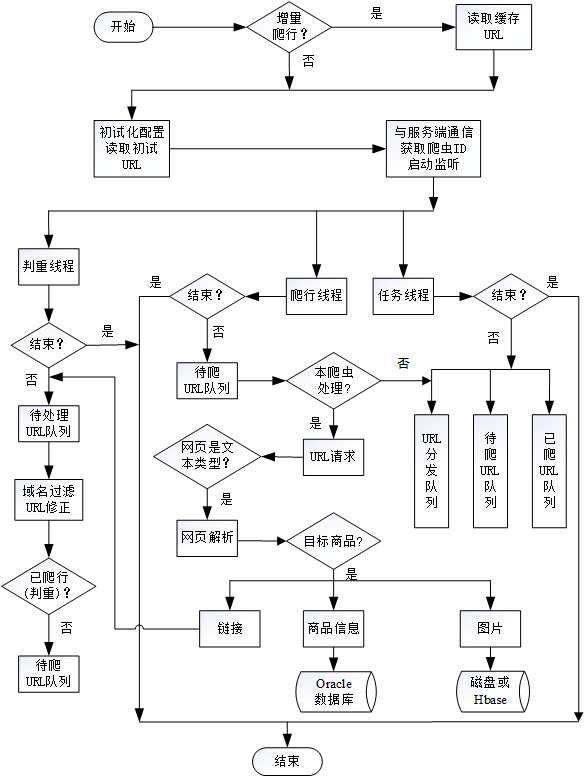


图 3.1 系统运行流程图

### 3.1.2 系统结束流程图

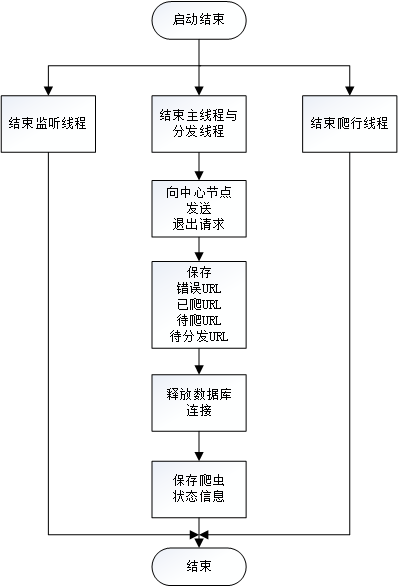


图 3.2 系统结束流程图

### 3.1.3 URL处理图

爬虫主要是围绕着处理URL运行。URL处理图正是以URL的视角介绍系统各个线程对URL的处理。如图3.3所示。

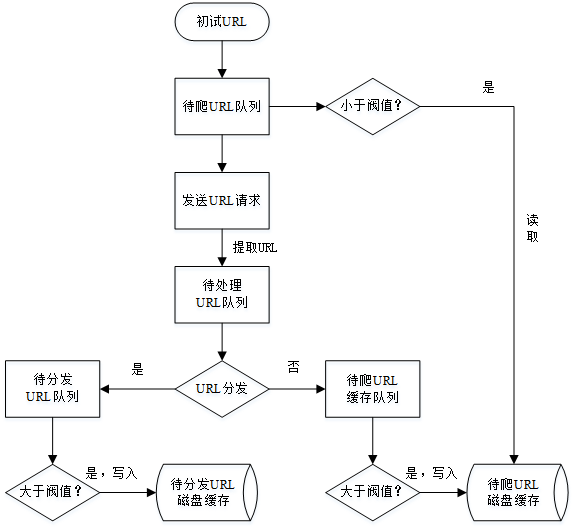


图 3.3 URL处理图

## 3.2 爬取目标的确定

电影信息爬行系统属于聚焦网络爬虫。而聚焦网络爬虫在对网页做信息提取前，必须确定网页是否为爬取目标。在本系统中，具体表现为对影视类别的确定。系统主要采用基于目标网页特征来确定影视类型。

由于主要针对m1905电影网和豆瓣电影网进行爬行，所以我们只用对这两个网站中的网页源代码进行分析即可。通过分析，我们发现name等于keywords或description的<meta>标签中都含有大量包的关键字，如图3.4所示。该图为豆瓣电影网上一部电影的网页源代码（http://movie.douban.com/subject/11026735/?from=showing）。然后检查<title>或<meta>标签中是否含有该集合中的关键词，如电影简介，以此来确定网页是否为电影类页面。

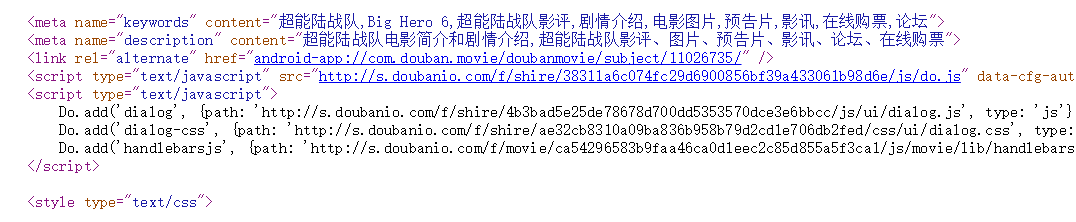


图3.4 网页源代码

测试发现，这种方法简单可行，正确率非常高。不过，还是出现了下载到非电影类图片。例如，一些电视剧的图片，不过总得来说，这种情况相对较少见，目前对m1905电影网和豆瓣电影网的图片爬取正确率均在99%以上。

## 3.3 爬行策略

互联网是网状结构，由众多链接相互连接。电影网站的规模虽然无法与整个互联网相比，但由于影片种类繁多、数量巨大，网站内部链接也是交错指向，同样可看做网状结构。因此，要想尽可能全地爬取电影信息，必须制定可行的爬行策略。通用型网络爬虫的爬取策略一般可以分为深度优先和广度优先两种。深度优先的原则是在一条路上尽可能深入爬行，这给深度优先带来陷入（trapped）问题。而网站通常会将重要的网页放在比较靠前的地方（链接层级少），不重要的链接层级往往较多，所以使用深度优先往往导致重要链接较慢甚至无法爬取。为了解决陷入问题，通常是通过设置爬行的最大深度来避免。而实际情况是，大多数爬虫使用的是广度优先。

### 3.3.1 广度优先

广度优先搜索（BFS: Breadth Fist Search），也叫宽度优先搜索，同样也是图的一种遍历算法，是指在爬取过程中，在完成当前层次的搜索后，才进入下一层次的搜索。以图3.5为例，图中七个点表示七个URL，之间的连接关系表示URL之间有链接，虚线代表层级关系，从A点开始，广度优先爬取顺序依次为A🡪B🡪C🡪D🡪E🡪F🡪G🡪H，而其深度优先的顺序为A🡪B🡪D🡪H🡪E🡪C🡪F🡪G，两种爬行策略有很明显的区别。

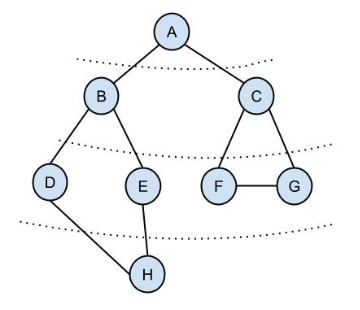


图3.5 广度优先示意图

网页设计一个重要原则是越重要的网页其层级应该越少。这样做不仅符合人的行为习惯，还可以降低用户获取的成本，便于搜索引擎收录。广度优先正好与这原则相符。另外，与深度优先处理一个链接完才转入下一个链接相比，广度优先可以并行处理，更适用于分布式系统。而广度优先的实现更是简单，只需一个队列。因而在爬虫中被广泛使用。

## 3.4 网页解析

电影分类信息网上，往往会在电影页面上放置了大量介绍、宣传电影的信息，特别是图片。要想从中获取到电影的简介、评分、详情等有用信息，需要对网页进行解析。互联网上的网页多为半结构化的文本，有着大量的标签和嵌套的结构。当我们自己开发一些处理网页的应用程序时，会想到要开发一个单独的网页解析器，这一部分的工作必定需要付出相当的精力和时间。为此，系统首先使用了开源解析器HtmlParser来完成网页解析部分。

### 3.4.1 解析器

HtmlParser 是SourceForge上的一个开源项目，它提供了线性和嵌套两种方式来解析网页，主要用于HTML 网页的转换（Transformation） 以及网页内容的抽取 （Extraction）。HtmlParser是一个设计简洁、运行速度快并经过严格测试的组件。它提供了强大而灵活易用的开源类库，支持过滤器（Filters）和访问者模式（Visitors）遍历HTML标签，并可以处理自定义标签。

## 3.5 URL处理

URL处理主要包括URL判重、URL过滤以及URL修正三个方面。

### 3.5.1 URL判重

各网页中的URL相互指向，在提取出来的链接中，经常会出现一些爬取过的。为了避免重复爬行，需要在爬取URL之前对其做是否已爬行过的判断，即URL判重。这问题听起来很简单，但由于数据量巨大，使得这一问题解决起来并不简单，同时，如果去重算法设计的不够高效，将会严重影响爬行效率。目前，URL判重研究主要围绕以尽可能小的内存消耗获取尽可能高效判重。

基于内存的Hash表的方法存在一个本质的问题，就是它消耗的内存是随着URL的增加而不断增长的。在要找一个类似于Hash表的但所消耗的内存相对固定而不会不断增长的方案的思路下，Burton Howard Bloom于1970年提出布隆过滤器 (Bloom Filter)，它是一种space efficient的概率型数据结构，用于判断一个元素是否在集合中。由于内存消耗小，判断速度快，在垃圾邮件过滤的黑白名单方法以及网络爬虫的URL判重模块中经常被用到。与哈希表判断元素是否在集合中相比，Bloom Filter只需要哈希表的1/8或1/4的空间复杂度就能完成同样的问题。

理论上，Bloom Filter存在误报（False Positive），但不存在漏报（False Negative）。误报是指将一些正常的样本（爬虫中就是没有爬取过的URL）过滤掉，而且随着元素的增加，误报率也随之增加。误报率取决于Bloom Filter算法中m，k，n这个三个参数的设置。其中，n为待判断元素个数，m为二进制位（bit）个数，k为哈希函数个数。相关文献研究表明，当时，可使误报率最低。



图3. 6判重流程图

专利针对分布式系统，在Bloom Filter基础上提出了一种新的去重方法。首先通过一致性哈希算法（Consistent Hashing）解决分布式系统的容错性、提高了扩展性以及增加了负载均衡，再将一致性哈希中用到的虚拟节点与Bloom Filter个数对应起来，使得每个Bloom Filter分担一部分URL，这样整个去重算法可以应对更大规模的数据。本文正是使用这种判重方法，图3.6构图，图中T为虚拟节点个数，K为哈希函数个数。

### 3.5.2 URL过滤

电影信息爬虫系统只爬行已实现解析的电影网站，对于其它网站，一律过滤掉。具体实现是将已实现解析的网站域名作为白名单设置在配置文件中。除了这种白名单机制的顶级域名过滤外，针对爬行的电影网站，还实现了黑名单机制的二级域名过滤。

电影网站上不仅有影片分类和电影页面，还有很多与影片无关的页面。特别是每个网页上的一些固定链接，如登录、注册、帮助、论坛等。对于这些链接，可以直接通过二级域名来过滤而不必等到URL判重，从而及早的淘汰非目标影片链接。此外，由于系统实现上只处理电影，对于其他影片的域名，也可以直接过滤掉，如豆瓣电视剧的地址为http://movie.douban.com/tv，可直接通过tv这个域名来过滤掉。

### 3.5.3 URL修正

网购链接地址经常会带有参数，造成同一部电影页面可能有不同的链接。所以，在发送URL请求前，需要对URL进行修正。

1. 统一电影页面链接形式

电影网站上，经常遇到同一页面不同的链接形式。例如豆瓣电影网上http://movie.douban.com/subject/3993588/?tag=%E7%83%AD%E9%97%A8&from=gaia、http://movie.douban.com/subject/3993588/?from=playing\_poster和http://movie.douban.com/subject/3993588/以及m1905电影网上的http://www.1905.com/mdb/film/2219617/和http://www.1905.com/mdb/film/2219617/?fr=wwwmdb\_data\_basicinfo\_7\_20140918#comment均指向的是同一电影页面。其中有些看起来很复杂的链接是因为该链接是通过站内搜索关键字得到的，从而链接中带有大量的参数。上述这些情况如果不处理，虽然链接不一样，但下载的电影海报是一样的，会引起重复下载。同时，这是无法通过URL去重算法解决的。对于各电影评分网站的这种情况，系统会在URL预处理阶段，统一修正为各网站链接的最简形式。如豆瓣电影网会修正为类似于http://movie.douban.com/subject/3993588/，而m1905电影网的最简形式为http://www.1905.com/mdb/film/2219617。

1. 加强去重

为了提高用户体验，电影评分网站往往大量使用Ajax这种异步传输数据技术，当用户查看电影评价等本页面未显示的信息时，这时只会更新页面部分内容，而不会加载整个页面，但页面网址有时会发生微小的改变。如m1905电影网的http://www.1905.com/mdb/film/2219617/?fr=wwwmdb\_data\_basicinfo\_7\_20140918#comment，就是http://www.1905.com/mdb/film/2219617/点击页面上的“发表影评”出现的。

系统解决这种问题的方法也很简单，直接对链接进行修正，截掉链接尾部多余的部分，从而使这些链接地址在URL判重算法中被过滤掉，再次达到避免重复下载的目的。

## 3.6 增量爬行

虽然系统通过线程检测重启和中心节点备份，极大地提高系统的稳定性，但系统还是面临一些不可抗拒的灾害，例如停电，导致系统在运行过程中突然挂掉。为了使系统再次启动时，还能继续之前的爬行，系统实现了增量爬行。

增量爬行的实现主要是在系统启动时通过读取之前爬行时缓存在本地磁盘上的链接以及相应的状态数据，使得爬虫开始爬行的状态近似故障前的。由于系统出故障时，内存里的数据来不及写到磁盘上，造成这部分数据丢失，故增量爬行并不能完全恢复到出故障时的状态，只是近似之前的状态。增量爬行的启用只需设置配置文件里的对应选项即可。

## 3.7 线程检测重启

为了应对爬行过程可能出现的各种状况，程序里包含大量的异常处理代码，但在运行过程中还是难免遇到未知的问题，例如一些异常来自于Socket通信过程中字节流错误，另一些异常甚至来源于使用的开源包。问题都出现在爬行过程中，在分布式多线程环境下，出现的问题很难重现并锁定，更没办法直接调试解决。因此，为了增强系统的健壮性，使系统在某些线程出现问题的情况下能自我修复，系统在任务线程中增加了对判重线程和爬行线程的状态检测功能。该任务线程属于定时任务，会定期检查线程状态，若检测某个线程的状态异常，就会立即重启该线程，从而保证系统能够持久运行。

# 4 系统运行

## 4.1 系统运行环境

系统开发、运行环境如表4.1所示。

表4.1 系统开发运行环境

|  |  |  |
| --- | --- | --- |
| **名 称** | **内 容** | **备 注** |
| 操作系统 | 推荐Windows 7 旗舰版 64位 | Java跨平台，其它系统也可以。但Hbase存图片只适用Linux环境。 |
| 开发工具 | Eclipse IDE for Java Developers Version：Kepler Service Release 1 | 尽量用最新的版本 |
| 开发语言(JDK版本) | Java (1.8.0) | 尽量用最新的JDK |
| 数据库 | Oracle 11g R2 for Windows | 存储电影信息。如价格 |
| 版本控制 | Apache Subversion (SVN) | 其它工具也行 |
| 网络带宽 | 带宽尽可能大 | 单机单线程下需要  下行带宽 > 200KB/s |

## 4.2 系统配置

### 4.2.1 Eclipse配置

由于爬虫中大量注释、输出信息使用中文，为了使系统在Linux和Windows下运行过程中避免出现中文乱码。系统统一使用“UTF-8”编码。

在Windows下，Eclipse默认编码为GBK，需要修改为“UTF-8”。具体操作为：Window—>Preferences—General>—>Workspace中的Text file encoding设置为“Other”中的“UTF-8”。如图4.1所示。

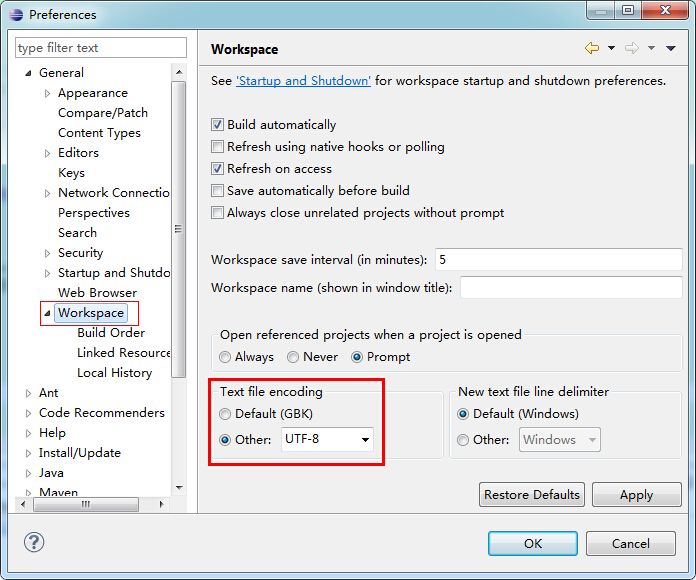


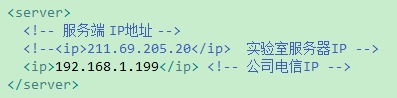
图4. 1 Eclipse编码设置

### 4.2.2 爬虫配置

与爬虫相关的设置主要在配置文件property.xml中。该配置文件中设置选项均有对应的注释，这里就不一一介绍，只重点介绍可能需要改动的设置。

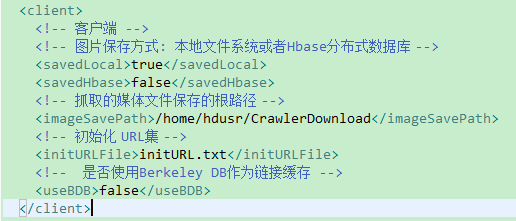
(1) 服务端IP地址

若在一台电脑上同时运行服务端和客户端，就直接设置本机的IP地址。

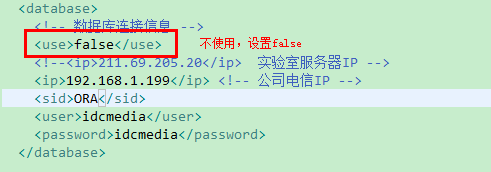


(2) 客户端设置

其中，<savedHbase>为Hbase保存图片，只适用于Linux系统。<savedHbase>设为true后，<imageSavePath>无效。



(3) 数据库



### 4.2.3 数据库安装配置

本项目有两种数据库可用，分别是MySql数据库和Oracle数据库，至于使用哪一种数据库，则视情况而定。其中，Oracle数据库的安装与配置，请参考Oracle数据库安装与部署（V1.0）.doc。数据库创建用户参考create\_user.txt。表字段含义说明参考Oracle电影信息爬虫系统数据库表（V1.0）.doc，建表语句参考create\_tables.txt；MySql数据库的安装与配置，请参考MySql数据库安装与部署（V1.0）.doc。数据库表字段含义说明参考MySql电影信息爬虫系统数据库表（V1.0）.doc。

## 4.3 系统启动与结束

爬虫系统分为服务端和客户端。服务端相当于中心节点，客户端相当于爬行节点。

### 4.3.1 服务端

服务端主要是CrawlerServer项目。在第一次运行时，需要进行运行配置。具体操作为：Run—>Run Configurations…，然后选中“Server”，在“Arguments”栏中的VM arguments添加“-Djava.security.policy=Rmi.policy”。如图4.2所示。

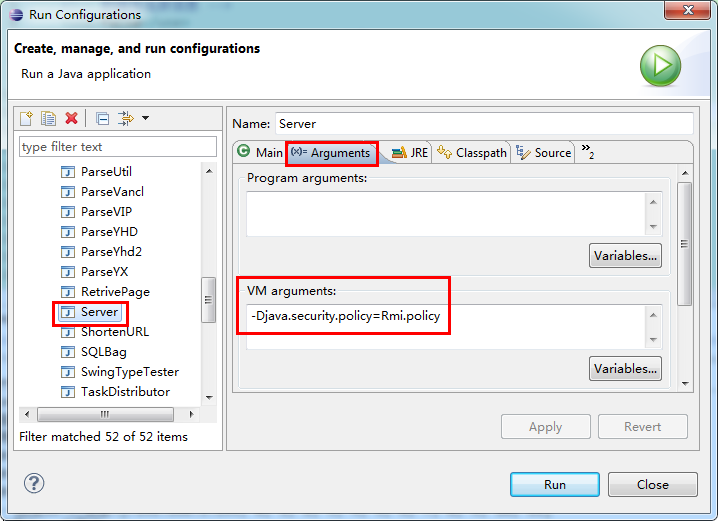


图 4.2运行配置

完成运行配置后。直接运行crawler.server包中的Server.java文件，即可启动服务端。服务端无界面，只有简单的输出，如图4.3所示。

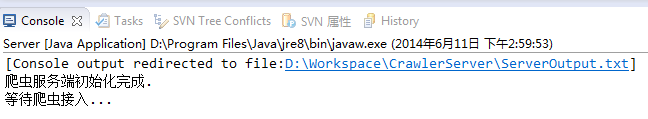


图 4.3服务端运行

### 4.3.2 客户端

客户端主要是MovieCrawler项目。和服务端一样，在第一次运行时，需要进行运行配置。具体操作和服务端一样，只是将选中的程序由“Server”替换为“Client”。接着，运行crawler.client包中的Client.java，出现下图4.4界面。初次运行时，建议测试下服务器的连接。若使用了数据库，也测试下数据库连接。单击“开始”按钮后，客户端开始运行。客户端界面只包含一些简单的设置选项，更多设置选项在配置文件中。客户端除了用于启动、结束爬虫外，一个重要的作用就是实时显示爬虫的运行情况，图中显示的信息为已爬取URL个数，下载的商品图片个数以及待爬URL个数。该时间间隔为1分钟，也可以另外设置。

说明：初始化URL来自服务端和客户端的initURL.txt文件。



图 4.4 爬虫客户端

### 4.3.2 定时爬行任务客户端

定时爬行任务客户端主要是MovieCrawler项目。运行crawler.timer包中的CrawlerTimer.java，出现下图4.4界面。初次运行时，建议测试下服务器的连接。若使用了数据库，也测试下数据库连接。单击“开始”按钮后，定时爬行任务客户端开始运行。客户端界面只包含一些简单的设置选项，更多设置选项在配置文件中。

说明：初始化URL来自服务端和客户端的initURL.txt文件，执行此定时任务爬行客户端时，需在配置文件中设置定时爬行任务可用，否则，爬虫定时任务界面的开始按钮不可用。



图 4.5 定时任务爬行客户端

## 4.4 爬取电影海报展示

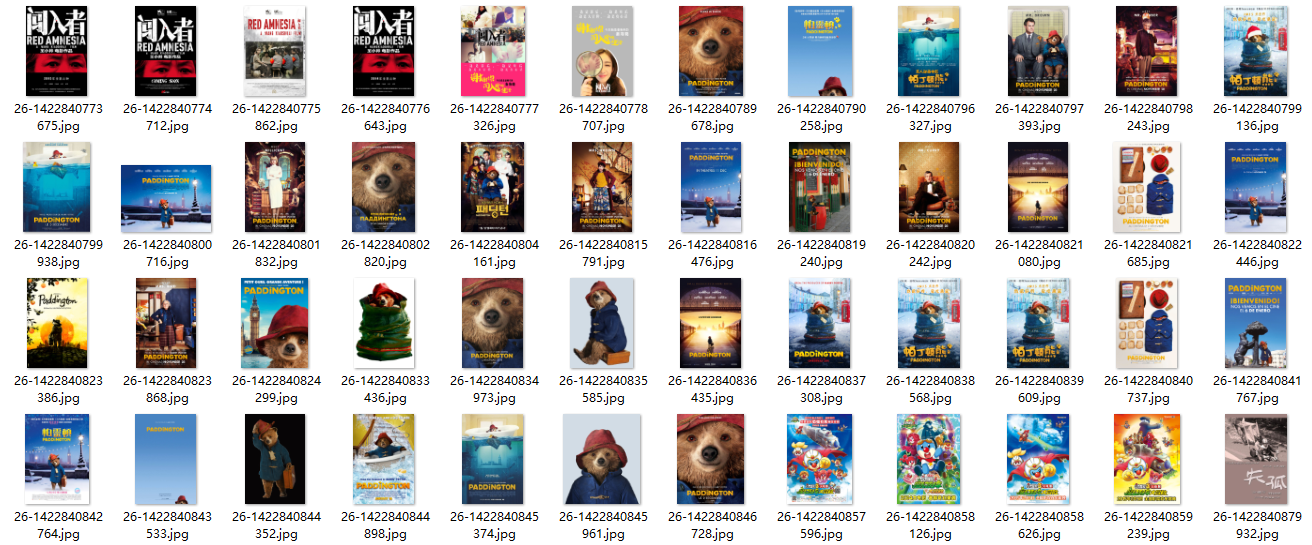


图 4.6 爬取的电影海报图片

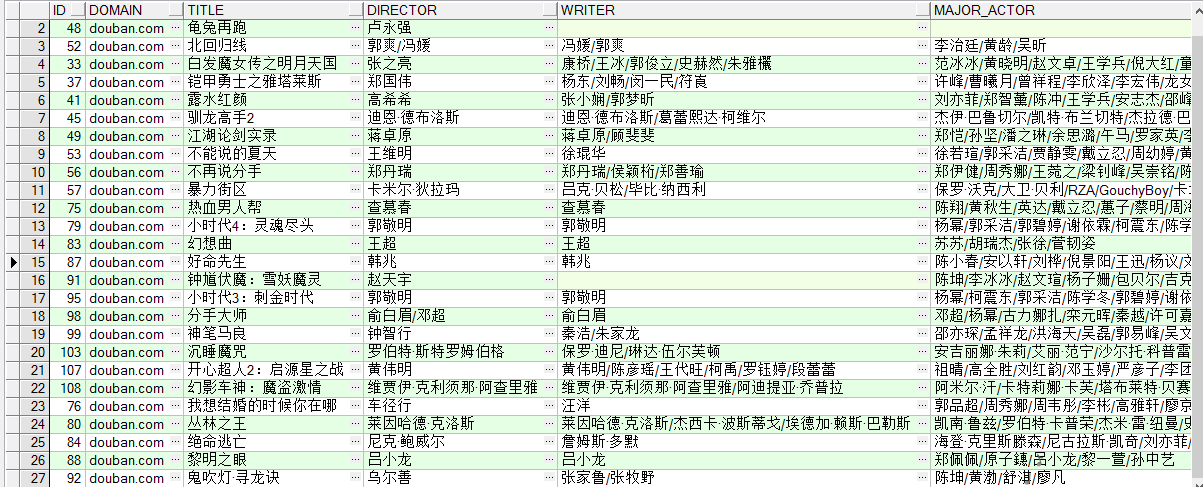


图 4.7 Oracle数据库中的电影信息

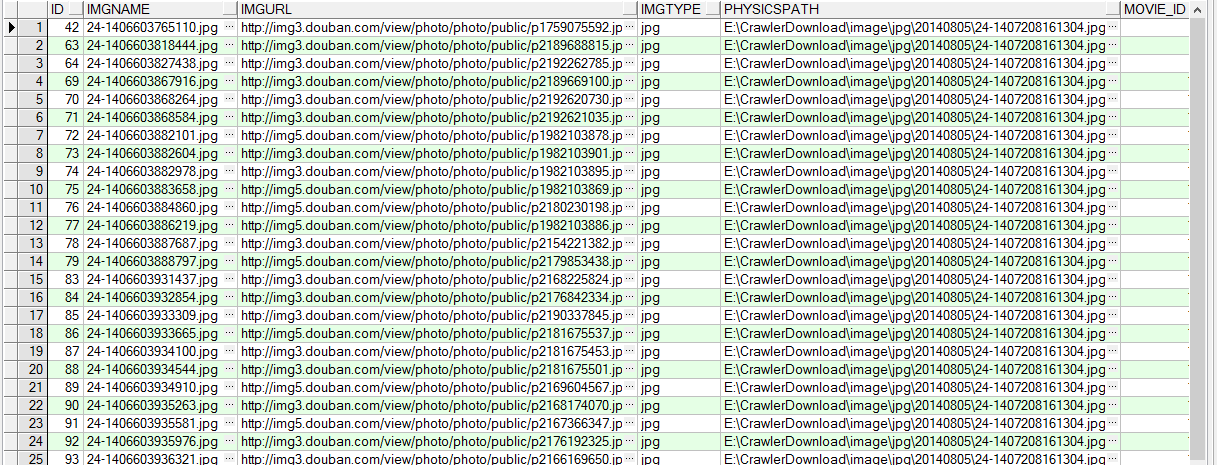


图 4.8 Oracle数据库中的电影海报信息

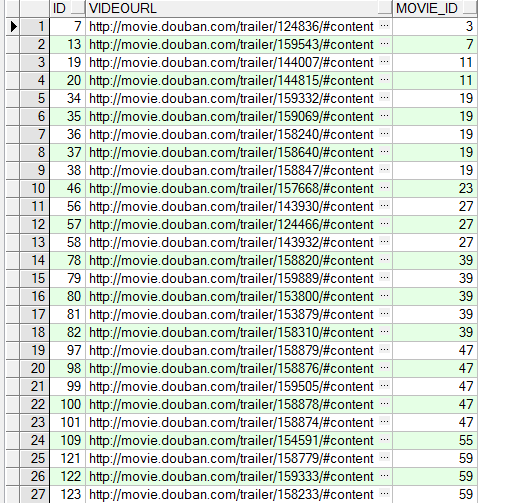


图 4.9 Oracle数据库中的电影预告片信息

图4.6为部分爬取的电影海报图片。该图片存储在爬虫节点的磁盘上。为了与特征提取、建索引等系统对接，系统实现了对爬取的图片直接存到Hbase数据库上。可以爬虫启动前在爬虫客户端上设置（图4.4）。另外，为了将电影海报图片与数据库中电影信息对应起来，图片均使用了唯一命名。即通过电影海报图片的名称，就可以在数据库中找到该电影的完整信息。

图4.7为数据库中电影表的部分记录，从中可以看到电影的名称、导演、编剧、主演以及评分等信息。图4.8为数据库中电影海报图片的部分记录，从中可以看到电影海报图片的链接地址以及其所对应的电影ID。图4.9为数据库中电影预告片的部分记录，从中可以看到电影预告片的链接地址以及其所对应的电影ID。

## 4.5 系统实现文件说明

### 4.5.1 服务端CrawlerServer项目

CrawlerServer项目主要包括crawler和crawler.server两个包。应用的包在lib文件夹中。项目具体组成如图4.7所示。

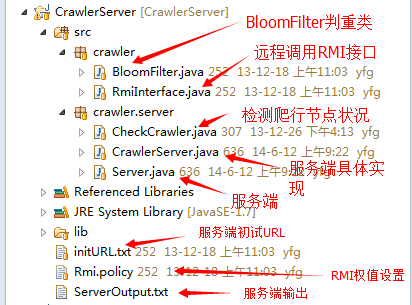


图 4.7 CrawlerServer实现文件说明

### 4.5.2 客户端MovieCrawler项目

MovieCrawler项目包含众多包和资源文件，具体组成如图4.8和图4.9所示。

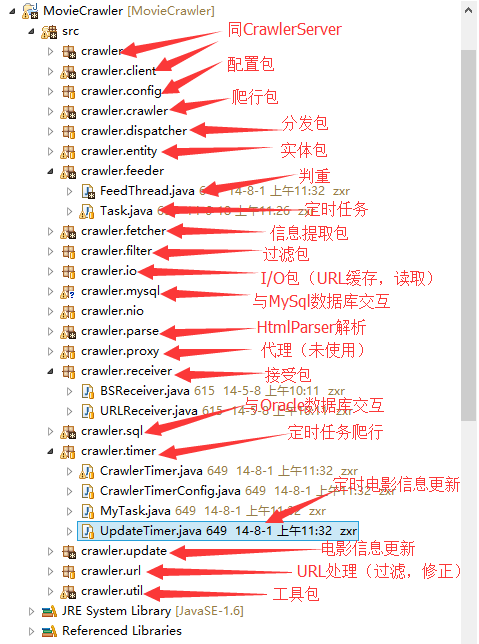


图 4.8 MovieCrawler实现文件说明

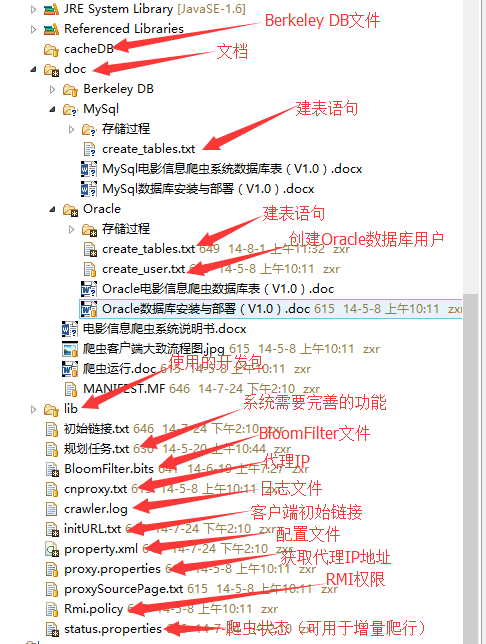


图 4.9 MovieCrawler资源文件说明