# 1、上次课程回顾

上次课程视频：链接:http://pan.baidu.com/s/1eSxnRGI 密码:9hfe

单线程爬虫实现步骤：

确定种子 --> 提取数据 --> 分析链接地址，放到一个队列 --> 对集合中的所有连接也进行相同的操作，直到数据提取完成。

注意：

地址不能重复（Set集合）

广度优先遍历

优化操作：

深度判断

HTMLParser工具简化代码

# 2、本次课程内容

## 2.1、多线程技术

我们编写的程序，默认会开启一个主线程。

通过编写线程的操作，可以开启多个子线程的操作。

通过多线程的形式，来提升程序的性能。

但是使用多线程会造成数据的安全隐患，这点在编写时需要注意。

因此，在编写多线程的程序时，需要注意使用同步的关键字，来避免这种安全问题。

Java实现多线程的主要方法：

1. 继承Thread类
2. 实现Runnable接口

必须覆写run()方法，通过start()来启动线程序。

|  |
| --- |
| **public** **class** ThreadDemo {  **public** **static** **void** main(String[] args) {  MyThread t1 = **new** MyThread();  MyThread t2 = **new** MyThread();  MyThread t3 = **new** MyThread();  t1.start();  t2.start();  t3.start();  }    }  **class** MyThread **extends** Thread {    @Override  **public** **void** run() {  // 子线的处理操作  System.***out***.println("当前子线程为：" + Thread.*currentThread*().getName());  }    } |
| **public** **class** ThreadDemo {  **public** **static** **void** main(String[] args) {  // MyThread t1 = new MyThread();  // MyThread t2 = new MyThread();  // MyThread t3 = new MyThread();  // t1.start();  // t2.start();  // t3.start();  MyRunnable r = **new** MyRunnable();  Thread t1 = **new** Thread(r);  Thread t2 = **new** Thread(r);  Thread t3 = **new** Thread(r);  t1.start();  t2.start();  t3.start();  }  }  **class** MyRunnable **implements** Runnable {  @Override  **public** **void** run() {  System.***out***.println("当前子线程为：" + Thread.*currentThread*().getName());  }  } |

两种方式的区别：

1. Thread方式更简单，但是不支持数据的通信。
2. Runnable方式代码稍微多一点，但是可以支持数据的通信。

Java中除了以上两种以外，还提供了Timer，TimerTask来支持多线程实现定时任务的操作。

还有在Android移动开发中，提供了AsyncTask的多线程支持，完成多线程任务操作。

## 2.2、网络爬虫中的多线程应用

分析之前的网络爬虫的操作过程，需要找到哪些操作可以多线程并行操作，哪些操作必须按顺序执行，不能并行操作，否则会有安全隐患。

提取数据的操作，可以支持并行完成，但是分配连接的操作，一定要按顺序完成（同步完成）。

放入连接的操作一般也建议单线程完成。

如果想编写多线程的爬虫操作，可以先准备好一些线程对象，并让这些对象等待有任务时，再进行操作。

这里就涉及一个怎样标记线程当前是空闲的问题，一般我们建议使用Object类的wait()和notify()

|  |
| --- |
| **public** **class** DangdangThreadCrawler {  // 所有等待爬取数据的集合  **private** **static** List<String> *allWaitUrls* = **new** ArrayList<String>();  // 准备一个Object类型的对象，相当于一个闹钟，通过这个对象来实现睡眠和唤醒线程。  **private** **static** Object *obj* = **new** Object();  **public** **static** **void** main(String[] args) {  **for** (**int** i = 0;i < 10;i++) {  MyThread t = **new** DangdangThreadCrawler().**new** MyThread();  t.start();  }  }  **class** MyThread **extends** Thread {  @Override  **public** **void** run() {  System.***out***.println("子线程初始化完成，当前线程为：" + Thread.*currentThread*().getName());  // 不能让线程自己结束，因此必须编写一个死循环。  **while** (**true**) {  // 判断当前是否有活要干，如果有，则进行处理，如果没有，则休息。  // 编写一个同步操作，同时只能有一个线程来判断allWaitUrls中的连接数，以及提取链接  String readyUrl = **null**;  // 通过一个同步块来实现同步的操作  **synchronized** (*allWaitUrls*) {  **if** (*allWaitUrls*.size() > 0) {  readyUrl = *allWaitUrls*.get(0);  *allWaitUrls*.remove(0);  }  }  **if** (readyUrl != **null**) {  // 有任务    } **else** {  // 没有任务，需要等待  **try** {  **synchronized** (*obj*) {  *obj*.wait();  }  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  }  }  } |

之后就需要开始准备数据爬取的操作了。

这里使用一个更容易的网页数据爬取方式：JSoup支持库。

|  |
| --- |
| **public** **class** DangdangThreadCrawler {  // 所有等待爬取数据的集合  **private** **static** List<String> *allWaitUrls* = **new** ArrayList<String>();  // 准备一个Object类型的对象，相当于一个闹钟，通过这个对象来实现睡眠和唤醒线程。  **private** **static** Object *obj* = **new** Object();  // url不重复的Set集合  **private** **static** Set<String> *allOverUrls* = **new** HashSet<String>();  // 记录深度  **private** **static** Map<String, Integer> *allUrlDepth* = **new** HashMap<String, Integer>();  // 允许的最大深度  **private** **static** **int** *MAX\_DEPTH* = 3;  **private** **static** String *SAVE\_PATH* = "/Users/kkb/Documents/dangdang\_data/";  **public** **static** **void** main(String[] args) {  *allUrlDepth*.put("http://category.dangdang.com/cp01.05.16.00.00.00.html", 1);  *allWaitUrls*.add("http://category.dangdang.com/cp01.05.16.00.00.00.html");  **for** (**int** i = 0; i < 10; i++) {  MyThread t = **new** DangdangThreadCrawler().**new** MyThread();  t.start();  }  // getUrlData("http://product.dangdang.com/23407125.html");  }  // 提取数据的操作方法  **public** **static** **void** getUrlData(String urlStr) {  **try** {  // 判断重复问题  **if** (!*allOverUrls*.contains(urlStr)) {  // 判断深度是否满足要求  **int** depth = *allUrlDepth*.get(urlStr);  **if** (depth <= *MAX\_DEPTH*) {  // JSoup是通过标准的DOM解析来提取数据的  // 注意，导包时，需要导入带 jsoup 包的支持类  Document doc = Jsoup.*connect*(urlStr).get();  // System.out.println(doc.html());  // 提取有效的数据  // 判断url是否是需要提取数据的url  **if** (urlStr.matches(".\*product\\.dangdang\\.com.\*\\d+\\.html")) {  // 满足要求，再进行数据的提取  // 根据要求提取标题和价格数据  // 标题要找的是h1  Element titleEl = doc.getElementsByTag("article").get(0);  // 提取里面的文字内容  String title = titleEl.text().trim();  // 提取价格  Element priceEl = doc.getElementById("main\_price");  String price = priceEl.text().trim();  // 保存文件  String fileName = System.*currentTimeMillis*() + ".txt";  PrintWriter writer = **new** PrintWriter(**new** File(*SAVE\_PATH* + fileName));  writer.println(title);  writer.println(price);  writer.print(urlStr);  writer.close();  }  // 提取所有的页面链接，加入到集合中  Elements allHref = doc.getElementsByTag("a");  **for** (Element aEl : allHref) {  // 取得里面的href属性  String href = aEl.attr("href");  **if** (href.startsWith("http:")) {  // 有用的链接就需要放入到集合中  **synchronized** (*allWaitUrls*) {  *allWaitUrls*.add(href);  **if** (!*allUrlDepth*.containsKey(href)) {  *allUrlDepth*.put(href, depth + 1);  }  }  // 需要唤醒之前睡眠的线程，继续干活  **synchronized** (*obj*) {  *obj*.notify();  }  }  }  System.***out***.println("当前线程：" + Thread.*currentThread*().getName() + " , 已经处理完成链接：" + urlStr);  // 记录处理好的链接  *allOverUrls*.add(urlStr);  // 判断是否有必要结束爬取操作  **if** (*allWaitUrls*.size() == 0) {  System.*exit*(0);  }  }  }  } **catch** (Exception e) {  e.printStackTrace();  }  }  **class** MyThread **extends** Thread {  @Override  **public** **void** run() {  System.***out***.println("子线程初始化完成，当前线程为：" + Thread.*currentThread*().getName());  // 不能让线程自己结束，因此必须编写一个死循环。  **while** (**true**) {  // 判断当前是否有活要干，如果有，则进行处理，如果没有，则休息。  // 编写一个同步操作，同时只能有一个线程来判断allWaitUrls中的连接数，以及提取链接  String readyUrl = **null**;  // 通过一个同步块来实现同步的操作  **synchronized** (*allWaitUrls*) {  **if** (*allWaitUrls*.size() > 0) {  readyUrl = *allWaitUrls*.get(0);  *allWaitUrls*.remove(0);  }  }  **if** (readyUrl != **null**) {  // 有任务  // 调用读取数据的方法  *getUrlData*(readyUrl);  } **else** {  // 没有任务，需要等待  **try** {  **synchronized** (*obj*) {  *obj*.wait();  }  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  }  }  } |

练习：

使用上面的代码模型，尝试爬取新浪新闻的数据。

|  |
| --- |
| **public** **class** SinaNewsThreadCrawler {  // 所有等待爬取数据的集合  **private** **static** List<String> *allWaitUrls* = **new** ArrayList<String>();  // 准备一个Object类型的对象，相当于一个闹钟，通过这个对象来实现睡眠和唤醒线程。  **private** **static** Object *obj* = **new** Object();  // url不重复的Set集合  **private** **static** Set<String> *allOverUrls* = **new** HashSet<String>();  // 记录深度  **private** **static** Map<String, Integer> *allUrlDepth* = **new** HashMap<String, Integer>();  // 允许的最大深度  **private** **static** **int** *MAX\_DEPTH* = 2;  **private** **static** String *SAVE\_PATH* = "/Users/kkb/Documents/sina\_data/";  **public** **static** **void** main(String[] args) {  *allUrlDepth*.put("http://news.sina.com.cn", 1);  *allWaitUrls*.add("http://news.sina.com.cn");  **for** (**int** i = 0; i < 10; i++) {  MyThread t = **new** SinaNewsThreadCrawler().**new** MyThread();  t.start();  }  // getUrlData("http://product.dangdang.com/23407125.html");  }  // 提取数据的操作方法  **public** **static** **void** getUrlData(String urlStr) {  **try** {  // 判断重复问题  **if** (!*allOverUrls*.contains(urlStr)) {  // 判断深度是否满足要求  **int** depth = *allUrlDepth*.get(urlStr);  **if** (depth <= *MAX\_DEPTH*) {  // JSoup是通过标准的DOM解析来提取数据的  // 注意，导包时，需要导入带 jsoup 包的支持类  Document doc = Jsoup.*connect*(urlStr).get();  // System.out.println(doc.html());  // 提取有效的数据  // 判断url是否是需要提取数据的url  **if** (urlStr.matches(".\*\\d{4}-\\d{2}-\\d{2}.\*\\d+\\.shtml")) {  // 满足要求，再进行数据的提取  // 根据要求提取标题和描述数据  // 标题  String title = doc.title();  // 描述  Element descEl = doc.getElementsByAttributeValue("name", "description").get(0);  String desc = descEl.attr("content");  // 保存文件  String fileName = System.*currentTimeMillis*() + ".txt";  PrintWriter writer = **new** PrintWriter(**new** File(*SAVE\_PATH* + fileName));  writer.println(title);  writer.println(desc);  writer.print(urlStr);  writer.close();  }  // 提取所有的页面链接，加入到集合中  Elements allHref = doc.getElementsByTag("a");  **for** (Element aEl : allHref) {  // 取得里面的href属性  String href = aEl.attr("href");  **if** (href.startsWith("http:")) {  // 有用的链接就需要放入到集合中  **synchronized** (*allWaitUrls*) {  *allWaitUrls*.add(href);  **if** (!*allUrlDepth*.containsKey(href)) {  *allUrlDepth*.put(href, depth + 1);  }  }  // 需要唤醒之前睡眠的线程，继续干活  **synchronized** (*obj*) {  *obj*.notify();  }  }  }  System.***out***.println("当前线程：" + Thread.*currentThread*().getName() + " , 已经处理完成链接：" + urlStr);  // 记录处理好的链接  *allOverUrls*.add(urlStr);  // 判断是否有必要结束爬取操作  **if** (*allWaitUrls*.size() == 0) {  System.*exit*(0);  }  }  }  } **catch** (Exception e) {  e.printStackTrace();  }  }  **class** MyThread **extends** Thread {  @Override  **public** **void** run() {  System.***out***.println("子线程初始化完成，当前线程为：" + Thread.*currentThread*().getName());  // 不能让线程自己结束，因此必须编写一个死循环。  **while** (**true**) {  // 判断当前是否有活要干，如果有，则进行处理，如果没有，则休息。  // 编写一个同步操作，同时只能有一个线程来判断allWaitUrls中的连接数，以及提取链接  String readyUrl = **null**;  // 通过一个同步块来实现同步的操作  **synchronized** (*allWaitUrls*) {  **if** (*allWaitUrls*.size() > 0) {  readyUrl = *allWaitUrls*.get(0);  *allWaitUrls*.remove(0);  }  }  **if** (readyUrl != **null**) {  // 有任务  // 调用读取数据的方法  *getUrlData*(readyUrl);  } **else** {  // 没有任务，需要等待  **try** {  **synchronized** (*obj*) {  *obj*.wait();  }  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  }  }  } |

## 2.3、文本预处理和数据存储

以新浪新闻数据爬取为例，如果提取下来数据想进行分析，必须先进行语义化的分词处理。

再将词的出现次数进行统计，并保存到数据库或类似的地方。

真正的保存方式，应该是将文本保存到HDFS中，再通过MR的方式来进行词频统计。

这里由于这些东西没学，所以我们简单的保存到MySQL数据库里即可。

这种文本语义分析的库网上有很多。

这个是跟着搜索引擎的发展出现的。

这里通过JE分词器，就可以完成新闻内容的分词操作，分词后，需要将词和出现次数结构化的保存到数据库中，方便我们进行统计操作。

这里我们就使用MySQL数据库来完成数据的保存。

安装配置好MySQL数据库以后（注意配置的端口号、编码、root用户的密码），可以通过

mysql -uroot -p密码 来登录数据库。

之后可以通过 show databases命令来显示所有的数据库。

我们这里通过create database 命令来建立一个新的数据库。

|  |
| --- |
| CREATE DATABASE sina\_news\_data CHARACTER SET 'utf8' ; |

切换到新创建好的数据库

|  |
| --- |
| USE sina\_news\_data; |

在这个库中建立新的表。

这张表中包含两个字段：

关键词keywords varchar(50)

出现次数counts int

|  |
| --- |
| CREATE TABLE sina\_news (  keywords varchar(50) primary key ,  counts int not null  ); |

之后就可以通过SQL语句，来完成数据的添加和查询等操作。

将之前的新浪新闻处理程序进行修改，读取到标题和新闻描述后，不再保存文本，而是直接拆分字符串。

统计每个关键词出现的次数。

这里为了避免线程安全问题，我们直接处理已经保存好的文本文件即可。

分词后，将出现次数统计好，然后保存到mysql数据库中。

|  |
| --- |
| **public** **class** SinaNewsDataSave {  **private** **static** Map<String, Integer> *keywordCounts* = **new** HashMap<String, Integer>();  **public** **static** **void** main(String[] args) **throws** Exception {  // 找到保存所有数据的文件夹，将这个文件夹下的所有文本文件读取出来，分别进行分词操作  File dir = **new** File("/Users/kkb/Documents/sina\_data/");  // 读取下面的所有文本文件  File[] allFiles = dir.listFiles();  MMAnalyzer mm = **new** MMAnalyzer();  **for** (File f : allFiles) {  BufferedReader reader = **new** BufferedReader(**new** FileReader(f));  // 只读取第一行和第二行的内容，进行内容的分析  String content = reader.readLine();  content += "|" + reader.readLine();  // 进行分词操作  String result = mm.segment(content, "|");  String[] allKeywords = result.split("\\|");  **for** (String keyword : allKeywords) {  // 先判断这个词在Map中是否已经保存过了  **if** (*keywordCounts*.containsKey(keyword)) {  // 如果之前保存过，在原有的出现次数基础上 ＋ 1  *keywordCounts*.put(keyword, *keywordCounts*.get(keyword) + 1);  } **else** {  // 这是第一次出现  *keywordCounts*.put(keyword, 1);  }  }  reader.close();  }    // 保存数据库  // 下面就需要掌握通过Java进行数据库操作的方法。  // 固定用到3个类，首先需要建立与数据库的链接  // 加载数据库驱动  Class.*forName*("org.gjt.mm.mysql.Driver");  // 根据数据库的位置和用户名密码，来链接数据库  // 注意 Connection 的包 使用 java.sql.包  Connection conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/sina\_news\_data?useUnicode=true&characterEncoding=UTF-8", "root", "root");    // 编写SQL语句，完成数据库的相关操作  String sql = "INSERT INTO sina\_news (keywords,counts) VALUES (?,?)" ;    // 循环所有的数据，准备进行添加  Set<String> allKeys = *keywordCounts*.keySet();  **for** (String key : allKeys) {  **int** value = *keywordCounts*.get(key);    // 建立数据库操作对象  PreparedStatement pst = conn.prepareStatement(sql);  // 传入?参数  pst.setString(1, key);  pst.setInt(2, value);  // 执行SQL命令，添加，修改和删除都使用 executeUpdate操作  pst.executeUpdate();  pst.close();  }    // 关闭链接等  conn.close();  }  } |

# 3、总结

1、多线程爬虫实现

2、数据的分词操作和保存处理

下次课程：

ECharts制作词云

Heritrix开源爬虫工具

JSON数据的解析处理（JD的商品评论数据）

大作业要求：

4人以下1组，要求自己定义爬虫的目的。

完成爬虫程序的设计和实现，通过任意一种可视化方式展现出结果即可。