并行预处理说明

一、 功能概述

该预处理程序主要解决了一下功能

1.从源文件中提取用户基本信息并将日期和时间进行分解（生成数据库插入指令和 IP 解析指令并将它们加入队列）

2.简化 app_id 生成对应的映射文件

3.调用 API 将文件中的 IP 地址转化为上网地区和上网类型并进行分解（生成数据库插入指令并将它们加入队列）

4.各进程从共享队列中取出数据库插入指令并执行

5.对数据库中的省份信息、品牌信息和版本信息进行更新

二、 类概述

AppInfoAnalyse:解析 Ip 信息，生成映射文件

DataBaseConnection:建立与数据库的连接

CommonRequest:向数据库中插入或更新词条的指令

CommonRequestQueue:存放 CommonRequest 元素的队列

IpRequest:解析 Ip 地址的指令

IpRequestQueue:存储 IpRequest 元素的队列

UpdateAppVersion:更新应用版本

UpdateBrandInfo:更新品牌信息

UpdateProvinceInfo:更新省份信息

Pre_Thread:预处理线程，从源文件中读取信息

Ip_Thread:Ip 处理线程，解析 Ip 信息

Common_Thread:插入线程，处理向数据库中插入词条

Main_Thread:主线程

三、 具体类的功能

1).AppInfoAnalyse

概述：此类用于解析 app_id 的信息，生成对应的映射文件

方法：

1.AppInfoAnalyse()

构造函数用于把应用信息存入 vector

2.String getInfo(String id)

参数：信号

返回值：app 应用信息

从向量中取出第 id 个值

3.String getMark(String id)

参数：原 app_id

返回值：简略版的 app_id

4.void initial()

用于初始化生成映射文件

2)DataBaseConnection

此类主要用于数据库的连接与关闭

3)CommonRequest,IpRequest

这两个类类似，用于存放指令，其中 CommonRequest 用于存放向数据库中插入或更新信息的指令，三个参数 type,row,item 分别表示插入类型（一般信息或 Ip 信息或结束信号）、用户 id（也就是插入的行数）和插入内容（用字符串数组表示）；IpRequest 用于存放解析 Ip 地址信息的指令，两个参数 row,item 分别表示用户 id 和待解析的 Ip 地址。

4)CommonRequestQueue,IpRequestQueue

这两个类类似，分别是用于存放 CommonRequest 和 IpRequest 的队列，定义了三个接口。

1.int getsize()

返回当前队列中的元素的个数

2.synchronized CommonRequest/IpRequest getRequest()

从当前队列中取出一个元素，当队列中没有元素时，让该线程等待。为了保证共享队列的安全性，同一时间只允许一个线程调用此方法。

3.synchronized void putRequest(CommonRequest/IpRequest)

向当前队列中加入一个元素，为了保证共享队列的安全性，同一时间只允许一个线程调用此方法。

5)UpdateAppVersionInfo,UpdateBrandInfo,UpdateProvinceInfo

这三个类实现的功能类似，主要用于在数据库基本建立完毕之后更新数据库中的信息，使相关的信息格式一致，以便后续处理。其中 UpdateAppVersionInfo 用于将形如 1.0、2.0 的版本标识改为 1、2；UpdateBrandInfo 用于统一品牌的格式（统一用小写字母表示，形如 moto 之类的简称改为全称）；UpdateProvinceInfo 用于将形如“北京市”、“上海市”等直辖市信息写入省份字段。

6)Pre_Thread

该类从 Thread 派生而来。

成员：static int count:静态变量，用于记录 id

int begin:开始读取的行数

int end:结束的行数（如果不存在该行，则一直读取到最后停止）

IpRequestQueue ip:存放 IpRequest 指令的目标队列

CommonRequestQueue common:存放 CommonRequest 指令的目标队列

方法：

void run():用于从源文件中读取信息，按#号进行分割并进行简单处理之后（日期时间分割，品牌标签小写字母化）之后，生成插入指令和解析 IP 信息的指令，并把它们加入到对应的共享队列中。

针对源文件中部分异常数据（多一个”#”号）作了分类处理。

7)Ip_Thread

成员：IpRequestQueue ip:读取 IpRequest 指令的队列

CommonRequestQueue common:存放 CommonRequest 指令的目标队列

Pattern patternLocation:对对 URL 上内容进行匹配的正则表达式

方法：

1.String getUrlContent(String strUrl):从网络上打开 URL，抓取上面的内容，将其转化为字符串返回

2.String[] splitIp(String temp):将从网页上提取的 IP 所在地信息按照”省”、”市”等关键字

进行分解，得到三个字符串，分别表示 IP 所在省份、城市和 IP 的类型

3.void run():从共享的 IpRequestQueue 队列中取出指令，访问网上的 Ip 归属地查询系统，调用 getUrlContent 方法抓取相关网页进行关键码匹配，再调用 splitIp 方法将 ip 地址进行分解，生成请求数据库插入 IP 信息的 CommonRequest 对象，加入共享的 CommonRequestQueue 队列中

8)Common_Thread

常规的向数据库中插入或更新数据的线程

成员:CommonRequestQueue common:读取 CommonRequest 指令的队列

方法:

void run():连接相应的数据库，从共享的 CommonRequestQueue 中取出元素，根据数据库中是否已经存在该元素判定执行插入还是更新操作。

9)Main_Thread

主线程，用于建立共享队列和启动各子线程

四、 执行顺序

首先执行 Main_Thread 主线程建立数据库，当所有元素都插入或更新完成后，调用 UpdateAppVersionInfo、UpdateBrandInfo、UpdateProvinceInfo 中的相关方法对省份信息、品牌信息和版本信息的格式进行标准化

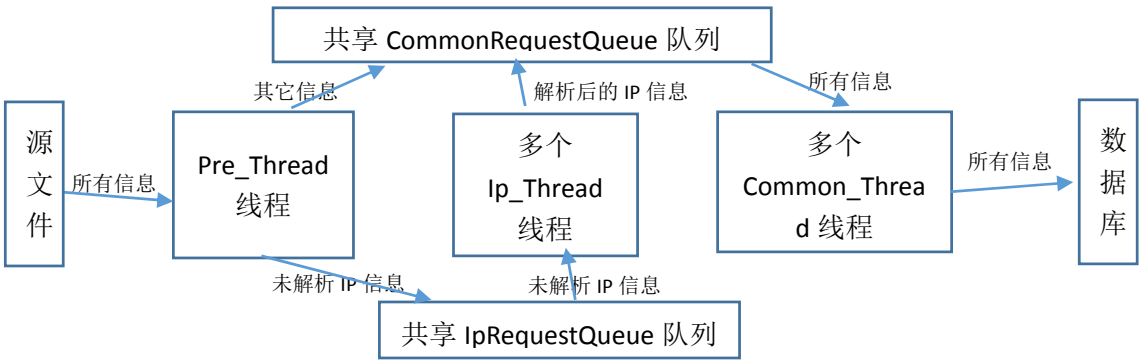
五、 各线程之间的配合

1.各线程之间的配合

在主线程中所有的 Common_Thread 线程、Ip_Thread 线程、Pre_Thread 线程都统一共享一个 CommonRequestQueue 和一个 IpRequestQueue，所有的 IP 信息都经过源文件-Pre_Thread –IpRequestQueue-Ip_Thread-CommonRequestQueue-Common_Thread-数据库的过程，其他信息都经过源文件-Pre_Thread-CommonRequestQueue-Common_Thread-数据库的过程。为了保证共享信息的安全性，上述两个队列在同一时间都只允许一个线程访问。

各线程的关系图如下

各线程截止信息传递图如下：



2.结束信号的传递

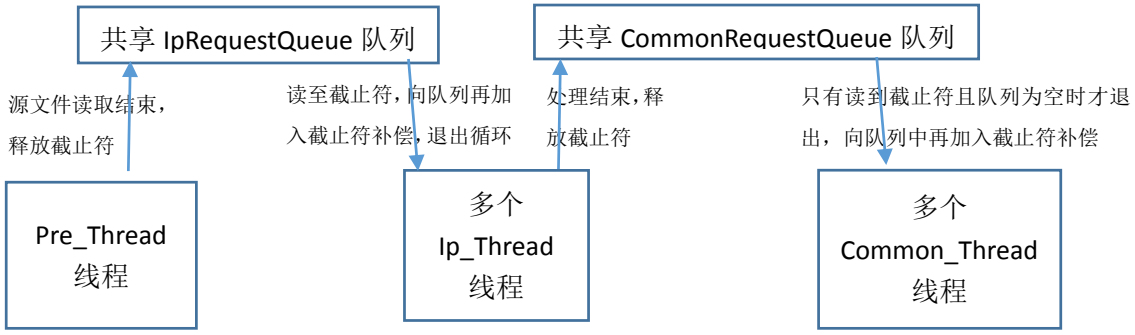
当 Pre_Thread 的文件读取工作结束后，线程会释放一个 row 值为-1 的 IpRequest 指令，提醒 Ip_Thread 线程读取工作已经结束，Ip_Thread 线程随后会释放一个 type 值为-1 的 CommonRequest 指令，提醒 Common_Thread 线程 Ip 解析工作已经结束。

由于可能存在多个 Ip_Thread 类型的进程，所以一旦 Ip_Thread 进程读到截止符时，它们再次向 IpRequestQueue 中加入一个相同的元素，以防止其它同类型的进程因读不到截止

符而进入死循环。

另一方面，由于 Common_Thread 读取的指令有来自 Pre_Thread 加入的，也有 Ip_Thread 加入的，所以在 CommonRequestQueue 队列中会有多个截止符。鉴于此，在 Common_Thread 类型的线程判定结束时，只认定在最后一个截止符（取出后 CommonRequestQueue 为空队列）被取出时退出线程，与 Ip_Thread 线程相同，此时我们也会向 CommonRequestQueue 中加入一个截止符以防止其它同类型线程进入死循环。

各线程截止信息传递图如下：



六、 性能测试

由于读取文件信息的速度>>IP 解析的速度>对数据库进行插入或更新的速度，并且 IP 信息的量远远小于其他信息的量，此外线程数过多会大大增加 CPU 的开销，所以在测试中，我们只开了 1 个 Pre_Thread 线程，开了 3 个 Ip_Thread 线程，而开了 15 个 Common_Thread 线程。

1.内存占用

第一组数据的量为 30000 行，程序运行稳定时内存的使用量约为 71.26M

第二组数据的量为 500000 行，程序运行稳定时内存的使用量约为 863M

假设内存的使用量与源文件中的行数成线性关系，由此估计测试 300 万行的数据，内存的使用量约为 5G

在服务器内存空间有限的情况下，我们可以采取将一个大文件拆成若干小文件的方式分步运行。

2.速度比较

序号	数据来源	数据量(行)	并行计算时间	速率(行/秒)	数据来源	数据量(行)	串行计算时间	速率(行/秒)	速率比
1	11 月前 30000	30000	3:20	150.00	10 月份	32929	20min	27.44	5.466
2	11 月前 500000	500000	49min	170.07					6.198

由此可见，使用并行程序之后，预处理的速度加快至原来的 5-6 倍。