**抓取系统**

# 1系统概况

evcrawler

消息队列文件

sender

Mysql链接库

Linkdbupdater

selector

Pagedbupdate

Mongo网页库

整个抓取系统是一个环的结构。

|  |  |
| --- | --- |
| 模块 | 简介 |
| Selector | 定期访问**evcrawler**，取得抓取的统计信息；根据统计信息，从**mysql链接库**中选取链接，发送到**evcrawler**，令其开始抓取 |
| Evcrawler | 基于libevent的抓取器，负责接受上游**selector**的url集合，抓取后把网页数据写入**消息队列文件**。 |
| 消息队列文件 | 从远程网站抓取下来的数据，存放于此，其结构是filelinkblock。 |
| Sender | 从**消息队列文件**中读取数据，发送给下游接受者，目前为**linkdbupdater**和**pagedbupdater**。 |
| Linkdbupdater | Linkdbupdater接受来自于sender的数据，更新此次抓取url的链接库信息以及从该页面中提取出新的url。 |
| Mysql链接库 | 存放url链接库 |
| Pagedbupdater | Pagedbupdater接受来自于sender的数据，把抓取数据存放于mongodb中。 |
| Mongo网页库 | 存放抓取的网页数据，以md5sum(url)为key。 |

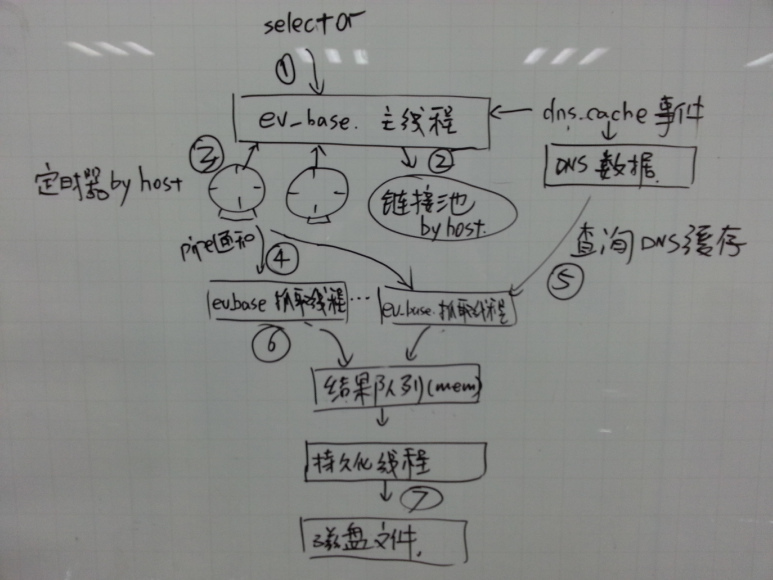
# 2模块详细介绍

## 2.1 Evcrawler

基于libevent实现，支持如下功能:

1. 使用libevent中的evhttp，实现了完全异步的抓取。
2. 从并发数和访问间隔两个角度实现抓取压力控制。
3. 支持优雅退出，退出之前会保存工作数据(待抓取的url链接)。
4. 支持dns cache机制。
5. 支持抓取数据实时的磁盘持久化。
6. 支持抓取状态统计。

模块内部流程如下。



如图，

1. 主线程中会启动一个event\_base，挂载在这个event\_base上的事件如下

|  |  |
| --- | --- |
| 事件名称 | 事件功能 |
| 监听socket | 该socket负责与上游的selector通信，注册好回调函数accept\_conn\_cb后，交给event\_base异步单线程处理。每次accept的新socket，也是注册好回调函数main\_process\_cb后，交给event\_base处理。 |
| 定时器 | 每个host都会创建一个自己的定时器事件，用来控制站点的访问间隔，当定时器事件被触发时，回调函数timer\_cb会通过pipe的方式，通知抓取线程开始抓取该站点的一个url。 |
| Dns异步更新 | Dns异步更新事件会负责定期更新dns cache，该事件会在timer\_cb中被调用。 |
| 信号事件 | 处理SIGTERM和SIGINT事件，会保存当前链接库中待抓取的链接于磁盘中，下次启动时，自动加载，马上开始抓取。 |

如果listen-socket收到请求，也就是步骤①，会把accept的新socket注册于event\_base中。当收到selector请求时，分为如下两种case处理。

|  |  |
| --- | --- |
| 请求名称 | 请求处理 |
| 统计请求 | Selector要求汇报各个站点的抓取统计情况，目前对**待抓取连接数**，**正在抓取连接数**，**该站点的最大并发抓取数**，**该站点的抓取间隔**进行统计汇报，host\_links对象负责更新这些数据。 |
| 添加链接 | Selector会把url链接批量发送过来，经过主线程处理后，插入到host\_links内存结构中，也就是步骤②。 |

1. 当某个host的定时器事件触发时(也就是图中很萌的闹钟图案，希望你能看明白)，也就是步骤③，该事件的回调函数timer\_cb会做如下事情:
   1. 如果dns\_cache的更新窗口打开，则发起该host的异步的dns查询。查询结果的更新工作交给回调函数dns\_query\_cb完成。
   2. 以roundrobin的轮询方式，通过主线程中建立的pipe管道，写入host数据，通知某个线程去抓取该host的某个url，这就是步骤④。
2. 抓取线程在创建之前，都各自绑定了一个pipe管道，抓取线程会把这个pipe的读事件注册到自己私有的event\_base上，因此timer\_cb可以通过pipe的方式触发抓取行为。Pipe的回调函数start\_crawling会从pipe中读到要抓取的host，然后抓取线程以这个host去host\_links中取得一个url链接，发起抓取行为。在抓取之前，抓取线程还要去dns\_cache中查询是否存在dns缓冲，这也就是步骤⑤。发起异步抓取后，当抓取完成时，其回调函数request\_cb会被调用，来处理抓取结果，并把抓取结果写入到内存结果队列中，并通过pthread的条件变量通知写入线程，这就是步骤⑥。
3. 写入线程收到通知后，会从内存结果队列中取得一个数据，如果未被压缩，则调用压缩函数对网页数据进行压缩处理，之后写入磁盘文件中。这就是步骤⑦。

## 2.2 消息队列文件

消息队列文件设计参见utils/filelinkblock，文件格式如下:

file\_link\_block\_head

Piece1

Piece2

消息队列文件就是上图所示的数据单元构成，每个文件大小不超过1G，evcrawler负责写入，而sender模块负责读取。

struct file\_link\_block\_head

{

uint32\_t magic\_num; //< 用于识别和校验head的开头

uint32\_t block\_id; ///< 本块中从1增的序号

uint32\_t timestamp; ///< 写入的时间点，可以用于回放时判断的依据

uint32\_t log\_id; ///< 用于追查和调试问题

uint32\_t check\_sum[2]; ///< 对变长数据部分(block\_buff)的校验值

uint8\_t queue\_type; ///< 队列ID

uint8\_t event\_type; ///< 事件ID

uint16\_t piece1\_len; ///< 数据第一部分的长度

uint32\_t reserved; ///< 保留位

uint32\_t block\_size; ///< 数据总长度

char block\_buff[0];

};

在这个抓取系统中，piece1部分存储json字符串，piece2部分存储压缩后的网页内容。

## 2.3 sender

Sender模块负责读取消息队列文件，逐条把消息发送给远程接受者。Sender的行为如下

1. sender启动后，会按照配置文件中每个channle的配置要求工作。
2. 对于每一个数据channel，sender会先读取offset下的进度文件，按照进度文件指示的位置开始发送数据。
3. Sender发送数据后，会等待一个正常的回应包，之后会记录进度于offset文件中，并开始发送新的消息。
4. 当遇到远程服务不能正常响应时，sender会休眠1s后，开始重新发送数据包，直到收到正常响应包，才会发送下一个数据包。
5. Sender可以支持数据rollback，不过这里的抓取系统没有用到。

## 2.4 linkdbupdater

Linkdbupdater从上游sender接受抓取的数据，主要做两个事情:

1. 更新链接库中此次抓取的url信息(抓取的时间和最后页面的返回状态码)。
2. 从此次抓取的url页面中，尝试提取新的url链接，插入到链接库中。如果是30X跳转系列，则会把Location当做是发现的扩散链接处理。链接提取仅限于host内部链接，采用正则表达式的方式提取链接，为了提高抓取效率，提取时会去掉url中?后面的参数。

## 2.5 Mysql链接库

链接库采用mysql实现，设计两个表，如下:

Link\_info表，采用innodb引擎。

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 说明 | 是否索引 |
| url\_no | int(11) | url的编号 | 自增主键 |
| url | varchar(255) | url |  |
| url\_type | tinyint(3) | url的类型(首页，索引页，详细页面) |  |
| host\_no | smallint(5) | url所在host的编号 | KEY |
| status\_code | smallint(5) | 最后一次的http-status-code |  |
| creat\_time | int(10) | url创建时间 |  |
| check\_time | int(10) | url最后一次访问时间 |  |
| refer\_sign | bigint(20) | url的refer的签名 |  |
| url\_sign | bigint(20) | url的签名，用于去重。 | UNIQUE KEY |

Host\_info表，innodb引擎

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 说明 | 是否索引 |
| host\_no | int(11) | host的编号 | 自增主键 |
| host | char(64) | host的字符串 |  |

## 2.6 selector

Selector负责定期请求evcrawler，获得每一个host的抓取的统计结果，并根据统计结果，如果某个host在evcrawler中的待抓取链接较少，则为每个host选取一定量的url，交给evcrawler供其抓取所用。

Selector目前的调度算法是只抓取新发现的url，selector会记录每个host当前选择到了最大的url\_no。

## 2.7 pagedbupdater

Pagedbupdater接受来自sender的请求。将http状态码为200的抓取结果，以md5sum(url)为key，插入到mongodb网页库中。

## 2.8 Mongo网页库

负责存储压缩后的网页。

# 3 部署及运维

## 3.1部署之前的环境准备工作。

软件安装(要求全部为64bit版本，注意数据存储的位置，不要默认存储，否则磁盘易满):

|  |  |
| --- | --- |
| 软件名称 | 简介 |
| mysql套件 | 包括mysql服务端，客户端等。修改mysql的配置文件my.cnf |
| mongodb | 从mongodb官网下载即可。 |
| Python2.7 | 官网下载后编译安装 |
| Python库 | 使用easy\_install安装pymongo/tornado/BeautifulSoup，MySQLdb采用源码build后安装。 |

## 3.2 部署爬虫系统

按照爬虫的全局示意图，做出如下部署。

1. Evcrawler/sender/page\_db\_updater/mongod部署于一台机器。这是考虑到mongod大量占用内存，而evcrawler/sender并不占用大内存的缘故。Page\_db\_updater可以就近更新mongod，减少网络开销。
2. Link\_db\_updater/selector/mysql部署于另一台机器，mysql对内存/CPU/磁盘有要求，link\_db\_updater和selector可以就近访问mysql。

## 3.3 爬虫系统各模块的配置及启动

|  |  |
| --- | --- |
| 模块 | 简介 |
| Selector | Python selector.py ./conf/selector.config.json ./data/host\_min\_urlno.json，目前和link\_db\_updater合用一个配置，但是evcrawler的host+port被写死在selector.py中，后续要分离出独立配置，把evcrawler的host和port写入selector的专属配置。 |
| Evcrawler | ./bin/evcrawler，默认配置为./conf/evcrawler.config.json |
| Sender | ./bin/senderer，默认配置为./conf/sender.config.json |
| Linkdbupdater | Python link\_db\_updater.py ./conf/link\_db\_updater.config.json |
| Pagedbupdater | Python page\_db\_updater.py ./conf/page\_db\_updater.config.json |

## 3.4 如何添加一个站点的抓取

(1) 在evcrawler的./conf/host\_visit\_interval.config.json中加入某个域名的访问压力配置信息。

(2) 使用python linkdbLogic.py <http://www.yoursite.com/> 在mysql链接库中加入种子连接。

(3) 重启selector。

# 4 后续的工作

Mysql链接库的性能问题，主要是两个索引，uniq key url\_sign用于url去重，和host\_no用于按照host进行url选取。

1. url\_sign可以挪出去，以前圈乐的disk\_dict模块可以做到210000/s，占用内存也较小，ttserver在1亿的数据量时，已经被证明随机插入速度不行。
2. 放弃host\_no的选取方式，改进为多个selector区域扫描的方式。比如每个selector负责1000万的区域，按照url的类型来控制url的访问间隔，比如首页的访问间隔为1小时，索引页访问间隔为1天，详细页面的访问间隔为3天。考虑到evcrawler中保存的待抓取链接数有限，每个selector还是要记住自己的扫描进度。