Sphinx 使用手册

在上一篇中，我们完成了 Sphinx 的安装，在这篇中我们使用 php 程序操作 Sphinx，做个小的站内搜素引擎。

Sphinx 集成到程序中，有两种方式：

Sphinxapi 类　　、　　SphinxSE 存储引擎。

我们要使用 Sphinx 需要做以下几件事 ：

1、首先得有数据　　（安装篇我们已经导入数据了）

2、建立 Sphinx 配置文件　　（在安装篇我们已经配置完成了）

3、生成索引　　（在安装篇我们也做了）

4、启动 Sphinx　　（从这里开始吧）

5、使用之　　（调用 api 进行查询 或 使用 sphinxSE）

一、启用 sphinx 服务。

想要在程序中使用 Sphinx 必须开启 Sphinx 服务。

启动进程命令：searchd

-c　　指定配置文件

--stop　　是停止服务

--pidfile　　用来显式指定一个 PID 文件。最好指定，要不然，合并索引时，会报错

-p　　指定端口

```
/usr/local/coreseek/bin/searchd -c /usr/local/coreseek/etc/sphinx.conf
```

注意：这里启动的服务是 searchd ，不是 search。

Sphinx 默认的端口是 9312 端口。

如果出现这个问题：

```
using config file '/usr/local/coreseek/etc/sphi
listening on all interfaces, port=9312
bind() failed on 0.0.0.0, retrying...
bind() failed on 0.0.0.0, retrying...
bind() failed on 0.0.0.0, retrying...
bind() failed on 0.0.0.0, retrying...
```

说明端口已经被占用了，可以用 netstat -tnl 查看下，9312 已经运行。

解决的办法是：netstat -apn | grep 9312　找出进程 ID

kill -9　进程 id　再开启就可以了。

二、 使用 php 程序使用 sphinx　（sphinxapi 类）

(1)、在 php 手册中有相应的函数。

需要到 coreseek 解压包中找到 sphinxapi.php 文件，放到程序目录下。

```
cp /lamp/coreseek-3.2.14/csft-3.2.14/api/sphinxapi.php /usr/local/apache2/htdocs/
include 'sphinxapi.php';                    // 加载 Sphinx API
$sphinx = new SphinxClient();               //创建 sphinx 对象
$sphinx->SetServer("localhost", 9312); //建立连接，第一个参数 sphinx 服务器地址，第二个 sphinx 监听端口
$result = $sphinx->query($keyword,"*"); // 执行查询，第一个参数查询的关键字，第二个查询的索引名称，多个索引名称用（逗号）分开,也可以用*表示全部索引。
print_r($result);//打印结果。
```

找到下面这一段，是我匹配的数据。(根据个人的数据不同，仅供参考)

```
[matches] => Array (  //匹配的结果
            [6] => Array
                    [weight] => 4
                    [attrs] => Array
                            [group_id] => 1
                            [date_added] => 1319127367
```

//一个多维的数组，下标[6] 是你匹配包含关键字的文档 id，id 对应的数组，[weight]是权

重，[attrs]是属性，我们在配置文件中指定的。

这段也是我们需要的数据。

[total] => 2　　此查询在服务器检索所得的匹配文档总数

[total_found] => 2　　索引中匹配文档的总数

[time] => 0.009　　这个是我们这次查询所用的时间

[words] => Array　　(分词的结果,将我输入的关键字进行分词,

“兄弟连”Sphinx 给分成了,[兄弟][连] 就是说我输入[兄弟]或[连] 都可以找到这个文档。

[兄弟] => Array

[docs] => 2　在文档中出现多少次　（content 字段中）

[hits] => 6　一共出现多少次

[连] => Array

[docs] => 2

[hits] => 6

(2)、取得数据 摘要 并 高亮显示。

Matches 中就是我们匹配的结果，但是仿佛不是我们想要的数据，比如 titile，content 字段的内容就没有匹配出来，根据官方的说明是 Sphinx 并没有连接到 MySQL 去取数据，只是根据它自己的索引内容进行计算，因此如果想用 Sphinx 提供的 API 去取得我们想要的数据，还必须以查询的结果为依据，再次查询 MySQL 从而得到我们想要的数据。

比如：

```
$mysqli = new mysqli("localhost","root","password","test");
$ids= join(',',array_keys($result['matches']));  要把需要的 id 取出来。
$sql="select title,content,date_added from documents where id in({$ids})" ;
$result=$mysqli->query($sql);
while($row=$result->fetch_row()){//循环体开始
```

解析看下结果。

下面我们在输出结果的时候需要生成摘要，高亮（就是和百度一样，关键字飘红）

我们需要用到 buildExcerpts 这个函数，（php 手册中）

语法格式：

```
public array SphinxClient::buildExcerpts ( array $docs , string $index , string $words [, array $opts ]
//返回的是一个数组，一共有四个参数
//第一个参数是从数据库中查询的结果集
//第二个参数是索引的名字。
//第三个参数是要高亮显示的关键字。
//第四个参数是显示的字 格式化。
$opts = array( // 格式化摘要，高亮字体设置。
//在匹配关键字之前插入的字符串，默认是<b>
    "before_match"  => "<span style='font-weight:bold;color:red'>",
//在匹配关键字之后插入的字符串，默认是</b>
    "after_match"  => "</span>",
//在摘要段落之前插入的字符串默认 …
    "chunk_separator" => " ... ",
    );
    $res=$sphinx->buildExcerpts($row,"index",$keyword,$opts);
    echo "<font size=4>".$res[0]."</font></a></br>"; 标题
    echo "<font size=2>".$res[1]."</font></br>";  摘要
    echo $res[2]."</p>";  添加时间
    }//循环体结束。
```

到这里呢，我们用 php 程序调用 sphinxapi 实现了高亮摘要功能。

三、SphinxSE 的使用

SphinxSE 是一个可以编译进 MySQL 5.x 版本的 MySQL 存储引擎，尽管被称作“存储引擎”，SphinxSE 自身其实并不存储任何数据。它其实是一个允许 MySQL 服务器与 searchd 交互并获取搜索结果的嵌入式客户端,所有的索引和搜索都发生在 MySQL 之外。

它有一个很大的特点呢，就是如果不支持 Sphinxapi 的语言，也可以使用 Sphinx，理论上说，Sphinapi 能做的，SphinxSE 都能做。

第一步、 安装SphinxSE

SphinxSE的插件，在Sphinx(Coreseek)解压文件中/sphinx/mysqlse

(1)、删除mysql

因为安装sphinxSE是嵌入到MySQL中，所以我们要重新编译安装一次MySQL。

(2)、复制sphinx中的mysqlse

把sphinx源码文件夹/lamp/sphinx-0.9.9/mysqlse下的所有文件复制到mysql源码文件中/lamp/mysql-5.1.59/storage/sphinx下，编译myslq的时候把mysqlse一起编译。

创建sphinx文件夹：

```
mkdir/lamp/mysql-5.1.59/storage/sphinx
```

复制mysqlse文件夹到mysql的制定目录下

```
cp/lamp/sphinx-0.9.9/mysqlse/*/lamp/mysql-5.1.59/storage/sphinx
```

(3)、编译安装

复制完后进入到mysql源码文件进行编译安装，如下：

```
cd/lamp/mysql-5.1.59/
```

环境检测：

```
./configure --prefix=/usr/local/mysql --with-charset=utf8
--enable-thread-safe-client --enable-assembler --with-readline --with-big-tables
--with-named-curses-libs=/usr/lib/libncursesw.so.5 --with-plugins=sphinx
```

环境检测的时候如果出现这个configure: error: unknown plugin: sphinx错误

没执行sh BUILD/autorun.sh的原因。

```
sh BUILD/autorun.sh
```

执行后还会报个错误：

```
BUILD/autorun.sh: line 41: aclocal: command not foundCan't execute aclocal
```

是因为aclocal的问题，需要安装3个依赖包在我们的镜像中都有，直接yum安装就可以。

imakeautomakelibtool三个包。

如：`yum install imake automake libtool`

接下来再编译安装

```
make && make install
```

(4)、安装完成后，配置mysql

*添加用户组mysql ，将mysql用户默认组设置为mysql用户组

```
groupadd mysql
useradd -g mysql mysql
```

*生成MySQL配置文件
```
cp support-files/my-medium.cnf /etc/my.cnf
```
*创建数据库授权表
```
/usr/local/mysql/bin/mysql_install_db --user=mysql
```
* 更改安装目录和数据目录的所有者、所属组
```
chown -R root /usr/local/mysql
chown -R mysql /usr/local/mysql/var
chgrp -R mysql /usr/local/mysql
```
* 启动MySQL服务
```
/usr/local/mysql/bin/mysqld_safe --user=mysql &
```
* 登录MySQL客户端控制台设置指定root密码
```
/usr/local/mysql/bin/mysql -u root
SET PASSWORD FOR 'root'@'localhost'=PASSWORD('Am@ri31n');
```
* 添加MySQL启动脚本，设置为只有运行级别3自启动
```
cp /usr/local/mysql/share/mysql/mysql.server /etc/rc.d/init.d/mysqld
chown root.root /etc/rc.d/init.d/mysqld
chmod 755 /etc/rc.d/init.d/mysqld
chkconfig --add mysqld
chkconfig --list mysqld
chkconfig --levels 245 mysqld off
```
* 进入mysql
```
/usr/local/mysql/bin/mysql-uroot-p密码;(如进入失败,需要重新启动mysql)
```

(5)、检测sphinxse

输入这个命令 `show engines;`

```
+------------+---------+----------------------------------------------------------------+--------------+------+----
| Engine     | Support | Comment                                                        | Transactions | XA   | Sav
+------------+---------+----------------------------------------------------------------+--------------+------+----
| MRG_MYISAM | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO
| CSV        | YES     | CSV storage engine                                             | NO           | NO   | NO
| MyISAM     | DEFAULT | Default engine as of MySQL 3.23 with great performance         | NO           | NO   | NO
| InnoDB     | YES     | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES
| SPHINX     | YES     | Sphinx storage engine 0.9.9                                    | NO           | NO   | NO
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO
+------------+---------+----------------------------------------------------------------+--------------+------+----
6 rows in set (0.00 sec)
```

显示红格出现的内容就表示 sphinxSE 安装成功!

第二步：创建 SphinxSE 引擎表。

```
CREATE TABLE t1(
    id          INTEGER UNSIGNED NOT NULL,
    weight      INTEGER NOT NULL,
    query       VARCHAR(3072) NOT NULL,
```

```
    INDEX(query)
) ENGINE=SPHINX CONNECTION="sphinx://localhost:9312/*";
```

也可以用“*”表示所有索引。

搜索表前三列（字段）的类型必须是 整型(前两个) 和 字符串，这三列分别对应文档 ID，匹配权值和搜索查询。这前三个列的映射关系是固定的，你不能忽略这三列中的任何一个，或者移动其位置，或者改变其类型。

第三步：创建 sql 语句进行连表查询。

```
$sql="select d.id,d.title,d.content,d.date_added from t1 join documents as d on t1.id = d.id and t1.query='{$keyword};index=test1'";
```
多个索引用逗号分隔。

这个操作主要是通过 SphinxSE 引擎进行索引的检索，返回的结果交给 mysql 进行处理。

第四步：处理结果集。

解析出来的结果是一个关联数组，包含字段的值。
如果想获得其他以外的结果，如查询文档数，所用时间等。
用 `show engine sphinx status;`
注意：这句话要在在主查询语句后，立即执行。

第五步：生成摘要，高亮。

从版本 0.9.9 版本开始，SphinxSE 提供了一个 UDF 函数，允许用户通过 MySQL 创建摘要。这个功能的作用与 API 调用 BuildExcerprts 的功能相似，但可以通过 MySQL+SphinxSE 来访问。

SphinxSE 的摘要，高亮。

在 MySQL 中创建一个函数：

```
CREATE FUNCTION sphinx_snippets RETURNS STRING SONAME 'sphinx.so';
```

函数的名字必须是 `sphinx_snippets`，而不能随便取名。

函数的参数表必须如下：

原型： `function sphinx_snippets ( document, index, words, [options] );`

Documents 参数：索引的字段名。

Index 参数：索引的名称。

Words 参数：高亮的关键字。

Options 参数（额外选项）必须这样指定：‘值’ AS 选项名。关于支持的所有选项，可以参见 API 调用 BuildExcerprts()。

实例：
```
SELECT sphinx_snippets('hello world doc', 'main', 'world',
'sphinx://192.168.1.1/' AS sphinx, true AS exact_phrase,
'[b]' AS before_match, '[/b]' AS after_match) FROM documents;

SELECT title, sphinx_snippets(text, 'index', 'mysql php') AS text
FROM sphinx, documents
WHERE query='mysql php' AND sphinx.id=documents.id;
```

四、排序，匹配，权重

匹配模式：SetMatchMode （设置匹配模式）

原型：function SetMatchMode ($mode)

SPH_MATCH_ALL 匹配所有查询词（默认模式）.

SPH_MATCH_ANY 匹配查询词中的任意一个.

SPH_MATCH_PHRASE 将整个查询看作一个词组，要求按顺序完整匹配.

SPH_MATCH_BOOLEAN 将查询看作一个布尔表达式.

SPH_MATCH_EXTENDED 将查询看作一个 Sphinx 内部查询语言的表达式.

SPH_MATCH_FULLSCAN 使用完全扫描，忽略查询词汇.

SPH_MATCH_EXTENDED2 类似 SPH_MATCH_EXTENDED ，并支持评分和权重

当如下条件满足时，SPH_MATCH_FULLSCAN 模式自动代替其他指定的模式被激活:

在完整扫描模式中，全部已索引的文档都被看作是匹配的。比如在论坛搜索用户帖子，用 SetFilter()指定用户 ID，就会返回，所有属于这个用户的帖子。

1.查询串是空的（即长度字符串为零）

2.docinfo 存储方式为 extern

注意，在完整扫描模式中，文档必须有至少一个属性。否则，即便设置 docinfo 的存储方式为 extern，也无法启用完整扫描模式。

权值计算：采用何种权值计算函数取决于查询的模式。

权值计算函数进行如下两部分主要部分：

1. 词组评分：

   词组评分根据文档和查询的最长公共子串（LCS，longest common subsequence）的长度进行。因此如果文档对查询词组有一个精确匹配（即文档直接包含该词组），那么它的词组评分就取得了可能的最大值，也就是查询中词的个数。

2. 统计学评分：

   统计学评分基于经典的 BM25 函数，该函数仅考虑词频。如果某词在整个数据库中很少见（即文档集上的低频词）或者在某个特定文档中被经常提及（即特定文档上的高频词），那么它就得到一个较高的权重。最终的 BM25 权值是一个 0 到 1 之间的浮点数。

通过这两个评分：最终算出权重值。

注意：根据不同的匹配模式，权值计算的方式也不一样，

如：在 SPH_MATCH_BOOLEAN 模式中，不做任何权重估计，每一个匹配项的权重都是 1。

在 SPH_MATCH_ALL 和 SPH_MATCH_PHRASE 模式中，最终的权值是词组评分的加权和。

在 SPH_MATCH_EXTENDED 模式中，最终的权值是带权的词组评分和 BM25 权重的和，再乘以 1000 并四舍五入到整数。

对于除布尔模式以外的全部模式中，都是子词组的匹配越好则评分越高，精确匹配（匹配整个词组）评分最高。这种方式有着更高的搜索质量。

权重设置：SetFieldWeights （设置字段权重）

原型： function SetFieldWeights ($weights)

按字段名称设置字段的权值。参数必须是一个 hash（关联数组），该 hash 将代表字段名字的字符串映射到一个整型的权值上。

例子：$sphinx->SetFieldWeights(array("title" => 10, "content" => 5));

给属性字段附个初始值，参与计算，权重默认为 1。

设置权重索引：SetIndexWeights

原型：function SetIndexWeights ($weights)

设置索引的权重，并启用不同索引中匹配结果权重的加权和。参数必须为在代表索引名的字符串与整型权值之间建立映射关系的 hash（关联数组）。默认值是空数组，意思是关闭带权加和。.

也就是说，如果文档 123 在索引 A 被找到，权值是 2，在 B 中也可找到，权值是 3，而且您调用了 SetIndexWeights （ array （ "A"=>100, "B"=>10 ）），那么文档 123 最终返回给客户端的权值为 2*100+3*10 = 230。

排序模式： SetSortMode （设置排序模式）

原型： function SetSortMode （ $mode, $sortby="" ）

设置排序模式，有两个参数，第一个参数是设置排序的模式，第二个是按属性排序。

SPH_SORT_RELEVANCE 模式, 按相关度降序排列（最好的匹配排在最前面）

SPH_SORT_ATTR_DESC 模式, 按属性降序排列 （属性值越大的越是排在前面）

SPH_SORT_ATTR_ASC 模式, 按属性升序排列（属性值越小的越是排在前面）

SPH_SORT_TIME_SEGMENTS 模式, 先按时间段（最近一小时/天/周/月）降序，再按相关度降序

SPH_SORT_EXTENDED 模式, 按一种类似 SQL 的方式将列组合起来，升序或降序排列。

SPH_SORT_EXPR 模式，按某个算术表达式排序。

属性说明：

SPH_SORT_RELEVANCE 忽略任何附加的参数，永远按相关度评分排序。

所有其余的模式都要求额外的排序子句，子句的语法跟具体的模式有关。

SPH_SORT_ATTR_ASC, SPH_SORT_ATTR_DESC 以及 SPH_SORT_TIME_SEGMENTS 这三个模式仅要求一个属性名。

SPH_SORT_RELEVANCE 模式等价于在扩展模式中按"@weight DESC, @id ASC"排序，

SPH_SORT_ATTR_ASC 模式等价于"attribute ASC, @weight DESC, @id ASC"，

而 SPH_SORT_ATTR_DESC 等价于"attribute DESC, @weight DESC, @id ASC"。

在 SPH_SORT_EXTENDED 模式中，您可以指定一个类似 SQL 的排序表达式，但涉及的属性（包括内部属性）不能超过 5 个，例如：

```
@weight DESC, price ASC, @id DESC
```

只要做了相关设置，不管是内部属性,还是用户定义的属性就都可以使用。内部属性的名字必须用特殊符号@开头，在上面的例子里，@weight 和@id 是内部属性，而 price 是用户定义属性。

五、实时更新索引 增量+合并

下面呢，我们讲增量索引还有合并索引。

数据库中的数据很大，然后我有些新的数据后来加入到数据库中，也希望能够检索到。全部重新建立索引很消耗资源，这样需要用到“主索引+增量索引”的思路来解决。

这个模式实现的基本原理是设置两个数据源和两个索引。

1、创建一个计数器。

一个简单的实现是，在数据库中增加一个计数表，记录将文档集分为两个部分的文档 ID,每次重新构建主索引是，更新这个表。

先在 mysql 中插入一个计数表。

```
CREATE TABLE sph_counter( counter_id INTEGER PRIMARY KEY NOT NULL,     max_doc_id INTEGER NOT NULL);
```

2、再次修改配置文件。

我们还需要修改一下配置文件文件。

主要是本着这个思路的，

主数据源，继承数据源，主索引，继承索引。（继承索引也就是增量索引）。

主数据源里面：我们需要把欲查询语句改成下面的语句： 13 行

```
Source main{
```

把 sql_query_pre 的改成下面的语句 79 行

```
sql_query_pre = REPLACE INTO sph_counter SELECT 1, MAX(id) FROM documents
    sql_query                = \
        SELECT id, group_id,UNIX_TIMESTAMP(date_added) AS date_added, title, content \
        FROM documents \
WHERE id<=( SELECT max_doc_id FROM sph_counter WHERE counter_id=1 )
}
```

继承数据源： 259 行

```
source delta : main
{
sql_query_pre = SET NAMES utf8
    sql_query                = \
        SELECT id, group_id,UNIX_TIMESTAMP(date_added) AS date_added, title, content
\
        FROM documents \
         WHERE id>( SELECT max_doc_id FROM sph_counter WHERE counter_id=1 )
}
```

主索引：　　281 行　把名字该成想对应的。

```
Index main {
source  = main                                  286 行
path = /usr/local/coreseek/var/data/main        290 行
}
```

继承索引（也是曾亮索引）　　494 行

```
index delta:main
{
    source          = delta
    path            = /usr/local/coreseek/var/data/delta
}
```

剩下的基本不用改变。

注意：　如果

你增量索引的 source 配置中只有 id, title,content 三项，

而主索引的 source 配置中有 id,date_added,title,content 四项，合并的时候会报属性数量不匹配。　如：

```
delta : sql_query = SELECT id, title,content FROM documents
main :   sql_query = SELECT id,UNIX_TIMESTAMP(date_added) AS date_added, title,
content FROM documents
```

3、测试增量索引+合并索引。

如果想测试增量索引是否成功，往数据库表中插入数据，查找是否能够检索到，这个时候检索应该为空，然后，单独重建增量索引

```
/usr/local/coreseek/bin/indexer -c /usr/local/coreseek/etc/sphinx.conf delta
```

查看是否将新的记录进行了索引。如果成功，此时，再用 /usr/local/coreseek/bin/search 工具来检索，能够看到，在主索引中检索到的结果为 0，而在增量中检索到结果。当然，前提条件是，检索的词，只在后来插入的数据中存在。

接下来的问题是如何让增量索引与主索引合并
命令原型：

```
indexer --merge main delta
```

```
/usr/local/coreseek/bin/indexer -c /usr/local/coreseek/etc/sphinx.conf --merge main delta --rotate
```

--rotate 参数，是在不停用 sphinx 进程的基础上，直接创建索引的。

注意：可能你要反复测试，你需要把所有的索引全部删除，重新建立，

这时候你建立的索引可能是 main.new.sph 等的索引文件。

你在搜索的时候会搜索不到，为什么呢？

你可能用了—rotate 参数，试着把 searchd 服务停止掉，在创建索引就会好的。

4、自动实时更新索引。

我们需要建立两个脚本，还要用到计划任务。

建立一个主索引和增量索引的脚本

main.sh delta.sh （可以自己命名）

在增量索引中写下 delta.sh

```
/usr/local/coreseek/bin/indexer -c /usr/local/coreseek/etc/sphinx.conf delta --rotate >> /usr/local/coreseek/var/log/delta.log
```

主索引中写下：main.sh 意思就是合并索引

```
/usr/local/coreseek/bin/indexer -c /usr/local/coreseek/etc/sphinx.conf --merge main delta --rotate >> /usr/local/coreseek/var/log/main.log
```

最后，我们需要脚本能够自动运行，以实现，增量索引每 10 分钟重新建立，和主索引只在午夜 2：30 时重新建立。

脚本写好了，我们需要建立计划任务

crontab -e 来编辑 crontab 文件，如果之前没有使用，会是一个空的文件。写下下面两条语句

```
*/10 * * * *  /usr/local/coreseek/etc/delta.sh

30 2 * * *  /usr/local/coreseek/etc/main.sh
```

第一条是表示每 10 分钟运行。
第二条是表示 每天的 凌晨 2：30 分运行(查看系统时间)

如果提示权限问题，给脚本可执行的权限 **chmod 755**

保存好后：重新启动服务

```
service crond stop    service crond start
```

要验证的话 ，我们可以查看日志文件，或者加入数据搜索。

六、分布式

分布式是为了改善查询延迟问题和提高多服务器、多 CPU 或多核环境下的吞吐率。对于大量数据（即十亿级的记录数和 TB 级的文本量）上的搜索应用来说是很关键的。

分布式思想：对数据进行水平分区（HP，Horizontally partition），然后并行处理。

当 searchd 收到一个对分布式索引的查询时，它做如下操作

1. 连接到远程代理;
2. 执行查询;
3. （在远程代理执行搜索的同时）对本地索引进行查询;
4. 接收来自远程代理的搜索结果;
5. 将所有结果合并，删除重复项;
6. 将合并后的结果返回给客户端.

```
index dist  //配置文件 494 行
{
     type = distributed
     local = chunk1
     agent = localhost:9312:chunk2     本地
     agent = 192.168.2.22:9312:chunk3 远程
     agent = 192.168.2.23:9312:chunk4 远程

}
```

Sphinx 我们已经基本讲完。

包括 Sphinx，SphinxSE，中文分词，高亮，摘要，匹配，排序，权重，增量索引,合并索引，分布式。

你自己可以创建一个 1 千万的数据，做个测试吧。

LAMP 兄弟连 蔡老师
邮箱：caida@lampbrother.net