目录

范例清单

第 1 章 简介

目录

1.1. 什么是 CoreSeek/Sphinx

Sphinx 是一个在 GPLv2 下分发的全文检索引擎；Coreseek 是一个可供企业使用的、基于 Sphinx（可独立于 Sphinx 原始版本运行）的中文全文检索引擎，按照 GPLv2 协议发行。商业使用（例如, 嵌入到其他程序中）需要联系我们以获得商业授权。

一般而言，Sphinx 是一个独立的全文搜索引擎；而 Coreseek 是一个支持中文的全文搜索引擎，意图为其他应用提供高速、低空间占用、高相关度结果的中文全文搜索能力。CoreSeek/Sphinx 可以非常容易的与 SQL 数据库和脚本语言集成。

应用程序可以通过三种不同的接口方式来与 Sphinx 搜索服务(searchd)通信： a) 通过原生的搜索 API (SphinxAPI), b) 通过 Sphinx 自身支持的 MySQL 网络协议 (使用命名为 SphinxQL 的 SQL 精简子集), 或者 c) 通过 MySQL 服务端的存储插件引擎 (SphinxSE)。当然， 还可以通过可以使用 a)、b)、c) 的应用程序来构建 webservice 来为其他应用程序提供通信

在 Sphinx 发行版本中提供的原生搜索 API 支持 PHP、Python、Perl、Rudy 和 Java。搜索 API 非常轻量化，可 以在几个小时之内移植到新的语言上。第三方 API 接口和插件提供了对 Perl、C#、Haskell、Ruby-on-Rails 支持，以及对其他可能的语言或者框架的 支持。

从版本 1.10-beta 开始，Sphinx 支持两种不同的索引后端：“磁盘（disk）”索引后端和“实时索引 （realtime）”（RT）索 引后端。磁盘索引支持在线全文索引重建，但是仅支持非文本（属性）数据的在线更新。RT 实时索引在此基础上，又增加了在线的全文索引更新。在此之前的版本 仅支持磁盘索引。

使用命名为数据源的接口，数据可以被加载到磁盘索引。当前系统内置 MySQL 和 PostgreSQL 以及 ODBC 兼容（MS SQL、Oracle 等） 数据库数据源的支持，也支持从管道标准输入读取特定格式的 XML 数据。通过适当修改源代码，用户可以自行增加新的数据源驱动（例如：对其他类型的 DBMS 的原生支持）。在 Coreseek 发行的版本中，用户还可以使用 Python 脚本作为数据源来获取任何已知世界和未知世界的数据，这极大的扩展了数据源的 来源。从 1.10-beta 版本开始的 RT 实时索引，只能使用 MySQL 接口通过 SphinxQL 来操作。

Sphinx 是 SQL Phrase Index 的缩写，但不幸的和 CMU 的 Sphinx 项目重名。

Coreseek (http://www.coreseek.cn/) 为 Sphinx 在中国地区的用户提供支持服务，如果您不希望纠缠与琐碎的技术细节，请直接联系我们。

本参考手册基于 Sphinx 2.0.1-beta 最新文档，可能存在潜在的翻译错误，如果您发现本文的翻译错误，请联系我们。

我们的联系方式： coreseek@gmail.com 李沫南(nzinfo) honestqiao@gmail.com 乔楚(HonestQiao 13581882013)

1.2. CoreSeek/Sphinx 的特性

Sphinx 的主要特性：

- 索引和搜索性能优异；
- 先进的索引和查询工具 (灵活且功能丰富的文本分析器，查询语言，以及多种不同的排序方式等等);
- 先进的结果集分析处理 (SELECT 可以使用表达式, WHERE, ORDER BY, GROUP BY 等对全文搜索结果集进行过滤);
- 实践证实可扩展性支持数十亿文档记录，TB 级别的数据，以及每秒数千次查询;
- 易于集成 SQL 和 XML 数据源，并可使用 SphinxAPI、SphinxQL 或者 SphinxSE 搜索接口
- 易于通过分布式搜索进行扩展

Sphinx 的详细特性：

- 高速的索引建立(在当代 CPU 上，峰值性能可达到 10 ~ 15MB/秒);
- 高性能的搜索 (在 1.2G 文本，100 万条文档上进行搜索，支持高达每秒 150~250 次查询);
- 高扩展性 (最大的索引集群超过 30 亿条文档，最繁忙时刻的查询峰值达到每天 5 千万次);
- 提供了优秀的相关度算法，基于短语相似度和统计（BM25）的复合 Ranking 方法;
- 支持分布式搜索功能;
- 提供文档片段（摘要以及高亮）生成功能;
- 内建支持 SphinxAPI 和 SphinxQL 搜索箭口，也可作为 MySQL 的存储引擎提供搜索服务;
- 支持布尔、短语、词语相似度等多种检索模式;
- 文档支持多个全文检索字段(缺省配置下，最大不超过 32 个);
- 文档支持多个额外的属性信息(例如：分组信息，时间戳等);
- 支持查询停止词;
- 支持词形学处理;
- 支持特殊词汇处理;
- 支持单一字节编码和 UTF-8 编码;
- 内建支持英语、俄语、捷克语词干化处理; 对 法语，西班牙语，葡萄牙语，意大利语，罗马尼亚语，德国，荷兰，瑞典，挪威，丹麦，芬兰，匈牙利等语言 的支持可通过第三方的 libstemmer 库建立);
- 原生的 MySQL 支持(同时支持 MyISAM 、InnoDB、NDB、Archive 等所有类型的数据表);
- 原生的 PostgreSQL 支持;
- 原生的 ODBC 兼容数据库支持 (MS SQL, Oracle, 等) ;
- …多达 50 多项其他功能没有在此一一列出，请参阅 API 和配置手册！

1.3. 如何得到 CoreSeek/Sphinx

Sphinx 原始版本可以从 Sphinx 官方网站 sphinxsearch.com/，Coreseek 可以从 Coreseek 官方网站 www.coreseek.cn/下载。

目前，CoreSeek/Sphinx 的发布包包括如下软件：

- `indexer`: 用于创建全文索引;
- `search`: 一个简单的命令行(CLI) 的测试程序，用于测试全文索引;
- `searchd`: 一个守护进程，其他软件（例如 WEB 程序）可以通过这个守护进程进行全文检索;
- `sphinxapi`: 一系列 searchd 的客户端 API 库，用于流行的 Web 脚本开发语言(PHP, Python, Perl, Ruby, Java).
- `spelldump`: 一个简单的命令行工具，用于从 `ispell` 或者 `MySpell` (OpenOffice 内置绑定) 格式的字典中提取词条。当使用 wordforms 时可用这些词条对索引进行定制.
- `indextool`: 工具程序，用来转储关于索引的多项调试信息。此工具是从版本 Coreseek 3.1(Sphinx 0.9.9-rc2)开始加入的。
- `mmseg`: 工具程序和库，Coreseek 用于提供中文分词和词典处理。

1.4. 许可协议

1.5. 作者和贡献者

作者

Sphinx 的最初作者 (诞生以来的主要开发者):

- Andrew Aksyonoff, http://shodan.ru

Sphinx 的团队

Sphinx 技术公司参与过或正在参与 Sphinx 开发的工作人员(按照字母排序):

- Alexander Klimenko
- Alexey Dvoichenkov
- Alexey Vinogradov
- Ilya Kuznetsov
- Stanislav Klinov

Coreseek 的团队

Coreseek 参与 CoreSeek/Sphinx 开发的工作人员:

- nzinfo, <li.monan(at)gmail.com>
- HonestQiao, <honestqiao(at)gmail.com>

贡献者

为 Sphinx 的开发出过力的人员和他们的贡献如下（以下排名不分先后）:

- Robert “coredev” Bengtsson (Sweden), 贡献了 PostgreSQL 数据源的初始版本
- Len Kranendonk, Perl API
- Dmytro Shteflyuk, Ruby API
- blueflycn, sphinx-for-chinese，另一个支持 MMSEG 中文分词的个人发布版本

此外，还有许多人提出了宝贵的想法、错误报告以及修正。在此一并致谢！对 CoreSeek/Sphinx 所采用依赖和工具软件的作者，在此也表示由衷的感谢与敬意。

1.6. 历史

Coreseek 的开发工作类似 Sphinx（起始于 2001 年），可以上溯到 2006 年（Sphinx 于该年开始对外提供），那时我们试图为一个数据库驱动的网站寻找一 个可接受的中文搜索的解决方案，但是当时没有任何方案能够完全而又直接的满足要求。事实上，主要是如下问题：

- 搜索质量(例如：类似 Google 的有效的相关度算法)
    - 单纯的统计学方法的效果非常糟糕，特别是在大量的短篇文档的集合上，例如：论坛、博客等等
- 搜索速度
    - 特别是当搜索的短语包括“停止词”时，表现的尤其明显，例如：英文（”to be or not to be”）、中文（“我的你的他的”）
- 建立索引时，磁盘 IO 和 CPU 消耗需要可控
    - 在共享的主机环境中，这一点的重要性更甚于索引速度
- 中文搜索的准确性和效率
    - 因为众所周知的原因，只有准确的中文分词才能提高中文搜索的准确性，并大大降低计算量.

通过网络，我们还了解到有无数的人存在类似的需求，但是没有一款真真合适的工具来解决这个问题。尔后我们进行了不同途径的探索，尝试 了不同的方法， 经过了反复的实践，幸运的是我们找到了 Sphinx，最终选择基于 Sphinx、结合 MMSeg，开发出 Coreseek 中文全文检索引擎，并按照 GPLv2 协议发行，以供企业和个人解决中文搜索问题。

随着时间的流逝，其他的解决方案有了很多改进，新的方案也不断涌现，但是，我们一致认为仍然没有一种解决方案是足够的好，以至于能让我们抛弃 Sphinx 将搜索平台迁移过去。

近年来，CoreSeek/Sphinx 的用户给了我们很多正面的反馈和建议，我们也不断改进和提高，提供了新的实时索引，并增加了 Python 数 据源，将 CoreSeek/Sphinx 的应用范围从已知世界扩展到未知世界，其应用场景也达到无限种可能；因此，显而易见的，CoreSeek /Sphinx 的开发过程将会继续（也许将持续到世界末日）。

第 2 章 安装

目录

2.1. 支持的操作系统

在绝大多数现代的 Unix 类操作系统（例如 Linux、BSD 等）上，只需要一个 C++编译器和基本的系统工具就可以编译并运行 CoreSeek/Sphinx，而不需要对源码进行任何改动。

目前，CoreSeek/Sphinx 可以在以下系统上运行：

- Linux 2.4.x, 2.6.x (包括各种发行版，如 Redhat、Centos、Fedora、Gentoo、Debian、Ubuntu、Slackware、OpenSuse、ArchLinux 等)
- Windows 2000, 2003, XP, Vista, Windows7, Windows2008
- FreeBSD 4.x, 5.x, 6.x, 7.x,8.x
- NetBSD 1.6, 3.0,4.x,5.x
- Solaris 9, 11
- Mac OS X

支持的 CPU 种类包括 X86, X86-64, AMD64, SPARC64。目前经过实际测试可以在主流 BSD 平台、Linux 平台和 Windows 平台运行，详情可以查看 Coreseek 测试运行环境列表。

我们希望 CoreSeek/Sphinx 也能够在其他的类 Unix 操作系统平台上工作，为所有需要解决中文搜索问题的用户服务，如果你运行 CoreSeek/Sphinx 索使用的操作系统不在上面的名单中，请告诉我们（HonestQiao, <honestqiao(at)gmail.com>）。

目前的阶段，CoreSeek/Sphinx 的 Windows 版可用于测试、调试和普通生产环境，但不建议用于负载量较大的生产系 统。限于 Windows 操作系统自身的限制，最突出的两个问题是：1）并发查询的支持不好；2）缺少索引数据热切换的支持。虽然目前已经有用户成功的在生产环境克 服了这两个问题，但是我们仍然不推荐在 Windows 下运行 CoreSeek/Sphinx 提供高强度的搜索服务。我们推荐使用 Linux 或者 BSD 作为 运行的操作系统平台，并可提供 Linux、BSD 系统下针对性的系统架构和性能优化支持服务。

2.2. 需要的工具

在类 UNIX 操作系统平台上，你需要以下的工具用来编译和安装 CoreSeek/Sphinx:

- C++编译器。GNU gcc 就能够干这个活.
- make 程序。GNU make 就能够干这个活.
- iconv 库。GNU libiconv 就能够提供支持.
- Python2.6（如果启用 Python 数据源支持）

在 Windows 平台上，你需要 Microsoft Visual C/C++ Studio .NET 2003 或者 2005 或者 2008。如果你还需要使用 Python 数据源，推荐安装 ActiveState Python2.6。其他的编译器/开发环境也许也能搞定这件事，但你可能需要自己手工制作他们所需的 Makefile 或者工程文件。

2.3. 在 Linux、BSD 上安装 CoreSeek/Sphinx

将你下载的 tar 包解压，并进入 `coreseek` 子目录： (我们在例子中使用的 2.0.1-beta 版本；你要根据你所使用的实际版本对指令和路径进行对应的修改)

```
$ tar xzvf sphinx-2.0.1-beta.tar.gz
$ cd sphinx
```

首先安装 MMSeg:

```
$ cd mmseg
```

$ ./configure –prefix=/usr/local/mmseg

$ make

$ make install

$ cd ..

运行配置 程序:

```
$ ./configure
```

configure 程序有很多运行选项。完整的列表可以通过使用 `--help` 开关得到。最重要的如下：

1.

- `--prefix`, 定义将 Coreseek 安装到何处;比如`--prefix=/usr/local/sphinx`(以下全部示例都假定 Coreseek 安装在这个位置)
- `--with-mysql`，当自动检测失败时，需要指出在那里能找到 MySQL 头文件和库文件;
- `--with-pgsql` 指出在那里能找到 PostgreSQL 头文件和库文件.
- `--with-mmseg`，启用基于 MMSeg 的中文分词法，并指出在那里能找到 MMSeg 头文件和库文件.
- `--with-python`，启用 Python 数据源支持. 需要预先安装 Python2.6.

编译源代码生成二进制程序:

```
$ make
```

安装二进制程序到你设定的目录下: (类 Unix 操作系统下默认为`/usr/local/bin/`，但是可以被 `./configure --prefix` 修改安装目录）

```
$ make install
```

2.4. 在 Windows 上安装 CoreSeek/Sphinx

在 Windows 上安装通常比在 Linux 环境下容易一些。要不是为了给代码制作 patch，一般安装预先编译好的二进制文件即可，它们可以在网站的下载区找到.

1. 将你下载的.zip 文件解压，比如 `sphinx-2.0.1-beta-win32.zip`, 或者 `sphinx-2.0.1-beta-win32-pgsql.zip` 如 果你需要 PostgresSQL 支持。 (我们在例子中使用的 2.0.1-beta 版本；你要根据你所使用的实际版本对指令和路径进行对应的修改) Windows XP 及其后续版本都可以直接解压.zip 压缩包，用免费的解压缩程序也可，比如 7Zip.在本教程的其余部分，我们假定上述压缩包被解压到 `C:\Sphinx`, 这样 `searchd.exe` 对应的路径就是 `C:\Sphinx\bin\searchd.exe`. 如果你打算给安装目录或者配置文件用个不同的路径，请在相应的地方自行修改路径。
2. 编辑 sphinx.conf.in – 需要修改@CONFDIR@相关项目 – 针对你的系统的实际情况进行修改.
3. 把 `searchd` 服务安装成一个 Windows 服务 :`C:\Sphinx\bin> C:\Sphinx\bin\searchd --install --config C:\Sphinx\sphinx.conf.in --servicename SphinxSearch`
4. 这样 `searchd` 服务应该出现在“控制面板->系统管理->服务”的列表中了，但还没有被启动，因为在启动它之前，我们还需要做些配置并用 `indexer` 建立索引 . 这些可以参考 快速入门教程. 在接下来的安装步骤(其中最多涉及到的是类似在 Linix 环境下运行 indexer 命令 ）中，你可能会得到一个有关 libmysql.dll 没有找到的错误提示。 如果安装过 MySQL，你应该找一个 Windows 系统目录中， 或者有时在 Windows \ System32 目录中，或者在 MySQL 的安装目录中，找到 libmysql.dll 文件。如果你 确实收到一个这样的错误，请将找到的 libmysql.dll 拷贝到 bin 目录中。

2.5. 已知的安装问题和解决办法

如果 `configure` 程序没有找到 MySQL 的头文件和库文件,, 请试试检查是否安装了 `mysql-devel` 或者 `mysql-client` 依赖包. 在有些系统上，默认安装包括这个包. 类似如此，libiconv 等也可能会有类似的提示。

如果 `make` 程序给出如下错误提示

```
/bin/sh: g++: command not found
make[1]: *** [libsphinx_a-sphinx.o] Error 127
```

请检查是否安装了 `gcc-c++` 包.

如果你在编译时得到如下错误

```
sphinx.cpp:67: error: invalid application of `sizeof' to
incomplete type `Private::SizeError<false>'
```

这意味着某些编译时的类型检查失败了，一个最有可能的原因是在你的系统上类型 off_t 的长度小于 64bit。一个快速的修复手段 是，你可以修改 src/sphinx.h ，将在定义类型 SphOffset_t 处，将 off_t 替换成 DWORD，需要注意，这种改动将使你的全文索引文件不能超过 2GB。即便这种修改有用，也请汇报这一问题，在汇报中请包括具体的错误信息以及操作 系统编译器的配置情况。这样，我们可能能够在下一个版本中解决这一问题。

如何你遇到了其他的任何问题，或者前面的建议对你没有帮助，别犹豫，请立刻联系我们.

2.6. CoreSeek/Sphinx 快速入门教程

以下所有的例子都假设你将 CoreSeek/Sphinx 安装在目录 `/usr/local/sphinx`，并且 `searchd` 对应的路径为 `/usr/local/sphinx/bin/searchd`.

为了使用 CoreSeek/Sphinx，你需要:

创建配置文件.缺省的配置文件名为 `sphinx.conf`. 全部的 CoreSeek/Sphinx 提供的程序默认都在当前工作的目录下寻找该文件.由 `configure` 程序生成的示例配置文件 `sphinx.conf.dist` 中包括全部选项的注释，复制并编辑这个文件使之适用于你的具体情况： (请确认 CoreSeek/Sphinx 安装在 `/usr/local/coreseek/`)

```
$ cd /usr/local/sphinx/etc
$ cp sphinx.conf.dist sphinx.conf
$ vi sphinx.conf
```

在示例配置文件中，将试图对 MySQL 数据库 `test` 中的 `documents` 表建立索引；因此在这里还提供了 `example.sql` 用于给测试表增加少量数据用于测试:

```
$ mysql -u test < /usr/local/sphinx/etc/example.sql
```

运行 `indexer` 为你的数据创建全文索引:

```
$ cd /usr/local/sphinx/etc
$ /usr/local/sphinx/bin/indexer --all
```

检索你新创建的索引!

你可以使用 `search`（注意，是 `search` 而不是 `searchd`）实用程序从命令行对索引进行检索:

```
$ cd /usr/local/sphinx/etc
$ /usr/local/sphinx/bin/search test
```

如果要从 PHP 脚本检索索引，你需要:

运行守护进程 searchd，PHP 脚本需要连接到 searchd 上进行检索:

```
$ cd /usr/local/sphinx/etc
$ /usr/local/sphinx/bin/searchd
```

运行 PHP API 附带的 test 脚本（运行之前请确认 searchd 守护进程已启动）:

```
$ cd sphinx/api
$ php test.php test
```

将 API 文件(位于 `api/sphinxapi.php`) 包含进你自己的脚本，开始编程.

祝你搜索愉快!

第 3 章 建立索引

目录

3.1. 数据源

索引的数据可以来自各种各样不同的来源：SQL 数据库、纯文本、HTML 文件、邮件等等。从 CoreSeek/Sphinx 的视角看，索引数据是一个结构化的*文档*的集合，其中每个*文档*是字段的集合，这种结构更利于 SQL 数据获取，其中每一行代表一个文档，每一列代表一个字段。

由于数据来源的不同，需要不同的代码来获取数据、处理数据以供 CoreSeek/Sphinx 进行索引的建立。这种代码被称之为*数据源驱动程序*（简称：*驱动*或*数据源*）。

在本文撰写时，CoreSeek/Sphinx 中包括 MySQL 和 PostgreSQL 数据源的驱动程序，这些驱动使用数据库系统提 供的 C/C++ 原生接口连接到数据库服务器并获取数据。此外，CoreSeek/Sphinx 还提供了额外的被称为 xmlpipe 的数据源驱动，该驱动运行某个具体的命 令，并从该命令的 `stdout` 中读入数据。数据的格式在 第 3.8 节 “xmlpipe 数据源” 中有介绍。经过长足的发展，Coreseek 还提供了具有特色的 Python 数据源驱动，可以使用 Python 编写数据获取脚本自定义数据源，从而得以获取任何已知世界和未知世界的数据。

如果确有必要，一个索引的数据可以来自多个数据源。这些数据将严格按照配置文件中定义的顺序进行处理。所有从这些数据源获取到的文档将被合并，共同产生一个索引，如同他们来源于同一个数据源一样。

3.2. 属性

属性是附加在每个文档上的额外的信息（值），可以在搜索的时候用于过滤和排序。

搜索结果通常不仅仅是进行文档的匹配和相关度的排序，经常还需要根据其他与文档相关联的值，对结果进行额外的处理。例如，用户可能需 要对新闻检索结 果依次按日期和相关度排序，检索特定价格范围内的产品，检索某些特定用户的 blog 日志，或者将检索结果按月分组。为了高效地完成上述工作，Sphinx 允许给文档附加一些额外的*属性*，并把这些值存储在全文索引中，以便在对全文匹配结果进行过滤、排序或分组时使用。

属性与字段不同，不会被全文索引。他们仅仅是被存储在索引中，属性进行全文检索式不可能的。如果要对属性进行全文检索，系统将会返回一个错误。

例如，如果 column 被设置为属性，就不能使用扩展表达式@column `1` 去匹配 column 为 1 的文档；如果数字字段按照普通的方式被索引，那么就可以这样来匹配。

属性可用于过滤，或者限制返回的数据，以及排序或者 结果分组; 也有可能是完全基于属性排序的结果, 而没有任何搜索相关功能的参与. 此外, 属性直接从搜索服务程序返回信息, 而被索引的文本内容则没有返回.

论坛帖子表是一个很好的例子。假设只有帖子的标题和内容这两个字段需要全文检索，但是有时检索结果需要被限制在某个特定的作者的帖子 或者属于某个子 论坛的帖子中（也就是说，只检索在 SQL 表的 author_id 和 forum_id 这两个列上有特定值的那些行），或者需要按 post_date 列对匹配 的结果排序，或者根据 post_date 列对帖子按月份分组，并对每组中的帖子计数。

为实现这些功能，可以将上述各列（除了标题和内容列）作为属性做索引，之后即可使用 API 调用来设置过滤、排序和分组。以下是一个例子：

示例： sphinx.conf 片段:

```
...
sql_query = SELECT id, title, content, \
author_id, forum_id, post_date FROM my_forum_posts
sql_attr_uint = author_id      #用于过滤的属性，类型为无符号整形
sql_attr_uint = forum_id       #同上
```

```
sql_attr_timestamp = post_date #设置用于过滤的属性，为时间戳类型
...
```

示例： 应用程序代码（使用 PHP）:

```
// 仅搜索 ID 为 123 的作者发布的内容
$cl->SetFilter ( "author_id", array ( 123 ) );

// 仅在 id 为 1，3，7 的子论坛中搜索
$cl->SetFilter ( "forum_id", array ( 1,3,7 ) );

// 按照发布时间倒序排列获取的结果
$cl->SetSortMode ( SPH_SORT_ATTR_DESC, "post_date" );
```

可以通过名字来指示特定的属性，并且这个名字是大小写无关的（注意：直到目前为止，Sphinx 还不支持中文作为属性的名称）。属性并*不会*被全文索引，他们只是按原封不动的存储在索引文件中。目前支持的属性类型如下：

- 无符号整数（1-32 位宽）；
- UNIX 时间戳（timestamps）；
- 浮点值（32 位，IEEE 754 单精度）；
- 字符串序列 (专指尤其计算出来的字符串序列整数值);
- 字符串 (版本 1.10-beta 开始支持);
- 多值属性 MVA(32 位无符号整型值的变长序列).

由各个文档的全部的属性信息构成了一个集合，它也被称为文档信息 *docinfo*. 文档信息可以按如下两种方式之一存储：

- 与全文索引数据分开存储（“外部存储 extern”，在 `.spa` 文件中存储），或者
- 在全文索引数据中，每出现一次文档 ID 就出现相应的文档信息（“内联存储 inline”，在 `.spd` 文件中存储）、

当采用外部存储方式时，`searchd` 总是在 RAM 中保持一份 `.spa` 文件的拷贝（该文件包含所有文档的所有文档信息）。这是主要是为了提高性能，因为磁盘的随机访问太慢了。相反，内联存储并不需要任何额外的 RAM，但代价是索引文件的体积大大地增加了；请注意，*全部*属性值在文档 ID 出现的*每一处*都被复制了一份，而文档 ID 出现的次数恰是文档中不同关键字的数目。仅当有一个很小的属性集、庞大的文本数据集和受限的 RAM 时，内联存储才是一个可考虑的选择。在大多数情况下，外部存储可令建立索引和检索的效率都*大幅提高*。

检索时，采用外部存储方式产生的的内存需求为 (1+属性总数)*文档总数*4 字节，也就是说，带有两个属性和一个时间戳的 1 千万篇文档会消耗(1+2+1)*10M*4 = 160 MB 的 RAM。这是每个检索的守护进程（*PER DAEMON*）消耗的量，而不是每次查询，`searchd` 仅在启动时分配 160MB 的内存，读入数据并在不同的查询之间保持这些数据。子进程并*不会*对这些数据做额外的拷贝。

3.3. MVA（多值属性）

多值属性 MVA (multi-valued attributes)是文档属性的一种重要的特例，MVA 使得向文档附加一系列的值作为属性的想法成为可能。这对文章的 tags，产品类别等等非常有用。MVA 属性支持过滤和分组（但不支持分组排序）。

目前 MVA 列表项的值被限制为 32 位无符号整数。列表的长度不受限制，只要有足够的 RAM，任意个数的值都可以被附加到文档上（包含 MVA 值的 `.spm` 文件会被 `searchd` 预缓冲到 RAM 中）。MVA 的源数据来源既可以是一个单独的查询，也可以是文档属性，参考 sql_attr_multi 中的来源类型。在第一种情况中，该查询须返回文档 ID 和 MVA 值的序对；而在第二种情况中，该字段被分析为整型值。对于多值属性的输入数据的顺序没有任何限制，在索引过程中这些值会自动按文档 ID 分组（而相同文档 ID 下的数据也会排序）。

在过滤过程中，MVA 属性中的*任何*一个值满足过滤条件，则文档与过滤条件匹配（因此通过排他性过滤的文档不会包含 任何被禁止的 值）。按 MVA 属性分组时，一篇文档会被分到与多个不同 MVA 值对应的多个组。例如，如果文档集只包含一篇文档，它有一个叫做 tag 的 MVA 属性，该属性 的值是 5、7 和 11，那么按 tag 的分组操作会产生三个组，它们的@count 都是 1，@groupby 键值分别是 5、7 和 11。还要注意，按 MVA 分组 可能会导致结果集中有重复的文档：因为每篇文文档可能属于不同的组，而且它可能在多个组中被选为最佳结果，这会导致重复的 ID。由于历史原因，PHP API 对结果集的行进行按文档 ID 的有序 hash，因此用 PHP API 进行对 MVA 属性的分组操作时你还需要使用 SetArrayResult().

3.4. 索引

为了快速地相应响应查询，Sphinx 需要从文本数据中建立一种为查询做优化的特殊的数据结构。这种数据结构被称为索引（*index*）；而建立索引的过程也叫做索引或建立索引（*indexing*）。

不同的索引类型是为不同的任务设计的。比如，基于磁盘的 B-Tree 存储结构的索引可以更新起来比较简单（容易向已有的索引插入新的文档），但是搜起来就相当慢。因此 Sphinx 的程序架构允许轻松实现多种不同的*索引类型*。

目前在 Sphinx 中实现的唯一一种索引类型是为最优化建立索引和检索的速度而设计的。随之而来的代价是更新索引相当的很慢。理论上讲，更新这种索引甚至可能比从头重建索引还要慢。不过大多数情况下这可以靠建立多个索引来解决索引更新慢的问题，细节请参考 第 3.11 节 “实时索引更新”.

实现更多的索引类型支持，已列入计划；目前已经包括一种可以实时更新的类型。

每个配置文件都可以按需配置足够多的索引。`indexer` 工具可以将它们同时重新索引（如果使用了`--all`选项）或者仅更新明确指出的一个。 `searchd` 工具会为所有被指明的索引提供检索服务，而客户端可以在运行时指定使用那些索引进行检索。

3.5. 源数据的限制

CoreSeek/Sphinx 索引的源数据有一些限制，其中最重要的一条是:

所有文档的 ID 必须是唯一的无符号非零整数（根据 Sphinx 构造时的选项，可能是 32 位或 64 位）

如果不满足这个要求，各种糟糕的情况都可能发生。例如，CoreSeek/Sphinx 建立索引时可能在突然崩溃，或者由于冲突的文档 ID 而在索引结果中产生奇怪的结果。也可能，一只重达一吨的恐龙最后跳出你的电脑，向你扔臭蛋。不要说我没有告诉过你咯！

3.6. 字符集、大小写转换和转换表

当建立索引时，Sphinx 从指定的数据源获得文本文档，将文本分成词的集合，再对每个词做大小写转换，于是“Abc”，“ABC”和“abc”都被当作同一个词（word，或者更学究一点，词项 *term*）

为了正确完成上述工作，Sphinx 需要知道：

- 源文本是什么编码的;
- 那些字符是字母，哪些不是;
- 哪些字符需要被转换，以及被转换成什么.

这些都可以用 charset_type 和 charset_table 选项为每个索引单独配置. charset_type 指 定文档的编码是单字节的（SBCS）还是 UTF-8 的。在 Coreseek 中，如果通过 charset_dictpath 设置中文词典启动了中文分词模式 后，不仅可以使用 UTF-8 编码的，还可以使用 GBK 及 BIG5 的编码（需要编译时提供 iconv 支持）；但是在内部实现中，仍然是预先转换成 UTF-8 编码在进行处理的. charset_table 则指定了字母类字符到它们的大小写转换版本的对应表，没有在这张表中出现的字符被认为是非字母类字符，并且在建立索引和检索时被当作词的分割符来看待。

注意，尽管默认的转换表并不包含空格符 (ASCII code 0×20, Unicode U+0020)，但是这么做是 *完全合法*的。 这在某些情况下可能有用，比如在对 tag 云构造索引的时候，这样一个用空格分开的词集就可以被当作一个*单独的*查询项了.

默认转换表目前包括英文和俄文字符。欢迎提交您为其他语言撰写的转换表！

在 Coreseek 中，启用中文分词后，系统会使用 MMSeg 内置的码表（被硬编码在 MMSeg 的程序中），因此，charset_table 在启用分词后将失效。

3.7. SQL 数据源 (MySQL, PostgreSQL)

对于所有的基于 SQL 驱动，建立索引的过程如下：

- 连接到数据库;
- 执行预查询 (参见 第 11.1.11 节 “sql_query_pre：待索引数据获取前查询”)，以便完成所有必须的初始设置，比如为 MySQL 连接设置编码;
- 执行主查询 (参见 第 11.1.12 节 “sql_query：获取待索引数据查询”)， 其返回的的数据将被索引;
- 执行后查询 (参见 第 11.1.30 节 “sql_query_post：数据获取后查询”)， 以便完成所有必须的清理工作;
- 关闭到数据库的连接;
- 对短语进行排序 (或者学究一点, 索引类型相关的后处理);
- 再次建立到数据库的连接;
- 执行索引后查询 (参见 第 11.1.31 节 “sql_query_post_index：数据索引后查询”)， 以便完成所有最终的清理善后工作;
- 再次关闭到数据库的连接.

大多数参数是很直观的，例如数据库的用户名、主机、密码。不过，还有一些细节上的问题需要讨论。

区段查询

索引系统需要通过主查询来获取全部的文档信息，一种简单的实现是将整个表的数据读入内存，但是这可能导致整个表被锁定并使得其他操作 被阻止（例如： 在 MyISAM 格式上的 INSERT 操作），同时，将浪费大量内存用于存储查询结果，诸如此类的问题吧。 为了避免出现这种情况，CoreSeek/Sphinx 支持一种被称为 *区段查询*的技术. 首先，CoreSeek/Sphinx 从数据库中取出文档 ID 的最小值和最大值，将由最大值和最小值定义自然数区间分成若干份，一次获取数据，建立索引。现举例如下：

例 3.1. 范围查询用法举例

```
# in sphinx.conf

sql_query_range= SELECT MIN(id),MAX(id) FROM documents
sql_range_step = 1000
sql_query = SELECT * FROM documents WHERE id>=$start AND id<=$end
```

如果这个表（documents）中，字段 ID 的最小值和最大值分别是 1 和 2345，则 sql_query 将执行 3 次：

1. 将 `$start` 替换为 1，并且将 `$end` 替换为 1000;
2. 将 `$start` 替换为 1001，并且将 `$end` 替换为 2000;
3. 将 `$start` 替换为 2001，并且将 `$end` 替换为 2345.

显然，这对于只有 2000 行的表，分区查询与整个读入没有太大区别，但是当表的规模扩大到千万级（特别是对于 MyISAM 格式的表），分区区段查询将提供一些帮助。

后查询（sql_post） 与 索引后查询（sql_post_index）的差异

后查询和索引后查询的区别在于，当 Sphinx 获取到全部文档数据后，立即执行后查询，但是构建索引的过程仍然 **may** 因为某种原因失败。在另一方面，当索引后查询被执行时，可以**理所当然的认为**索引已经成功构造完了。因为构造索引可能是个漫长的过程，因此对与数据库的连接在执行后索引操作后被关闭，在执行索引后操作前被再次打开。

3.8. xmlpipe 数据源

xmlpipe 数据源是处于让用户能够将现有数据嵌入 Sphinx 而无需开发新的数据源驱动的目的被设计和提供的。它将每篇文档限制为只能包括两个可全文索引的字段，以及只能包括两个属性。xmlpipe 数据源已经被废弃，在第 3.9 节 “xmlpipe2 数据源”中描述了 xmlpipe 的替代品 xmlpipe2 数据源。对于新的数据，建议采用 xmlpipe2。

为了使用 xmlpipe，需要将配置文件改为类似如下的样子：

```
source example_xmlpipe_source
{
type = xmlpipe
xmlpipe_command = perl /www/mysite.com/bin/sphinxpipe.pl
}
```

`indexer` 实用程序将要运行 xmlpipe_command, 所指定的命令，而后读取其向标准输出 `stdout` 上输出的数据，并对之进行解析并建立索引。 严格的说，是索引系统打开了一个与指定命令相连的管道，并从这个管道读取数据。

indexer 实用程序假定在从标准输入读入的 XML 格式的数据中中存在一个或更多的文档。下面是一个包括两个文档的文档数据流的例子：

例 3.2. XMLpipe 文档数据流

```
<document>
<id>123</id>
<group>45</group>
<timestamp>1132223498</timestamp>
<title>test title</title>
<body>
this is my document body
</body>
</document>

<document>
```

```
<id>124</id>
<group>46</group>
<timestamp>1132223498</timestamp>
<title>another test</title>
<body>
this is another document
</body>
</document>
```

遗留的 xmlpipe 数据驱动使用内置的解析器来解析 xml 文档，这个解析器的速度非常快，但是并没有提供对 XML 格式完整支持。这个解析器需要文档中*必须*包括全部的字段，并且*严格*按照例子中给出的格式给出，而且字段的出现顺序需要*严格*按照例子中给出的顺序。仅有一个字段 `timestamp` 是可选的，它的缺省值为 1。

3.9. xmlpipe2 数据源

xmlpipe2 使你可以用另一种自定义的 XML 格式向 Sphinx 传输任意文本数据和属性数据。数据模式（即数据字段的集合或者属性集）可以由 XML 流本身指定，也可以在配置文件中数据源的配置部分中指定。

在对 xmlpipe2 数据源做索引时，索引器运行指定的命令，打开一个连接到前述命令标准输出的管道，并等待接受具有正确格式的 XML 数据流。以下是一个数据流的样本：

例 3.3. xmlpipe2 文档数据流

```
<?xml version="1.0" encoding="utf-8"?>
<sphinx:docset>

<sphinx:schema>
<sphinx:field name="subject"/>
<sphinx:field name="content"/>
<sphinx:attr name="published" type="timestamp"/>
<sphinx:attr name="author_id" type="int" bits="16" default="1"/>
</sphinx:schema>

<sphinx:document id="1234">
<content>this is the main content <![CDATA[[and this <cdata> entry
must be handled properly by xml parser lib]]></content>
<published>1012325463</published>
<subject>note how field/attr tags can be
in <b>randomized</b> order</subject>
<misc>some undeclared element</misc>
</sphinx:document>

<sphinx:document id="1235">
<subject>another subject</subject>
<content>here comes another document, and i am given to understand,
that in-document field order must not matter, sir</content>
<published>1012325467</published>
</sphinx:document>

<!-- ... even more sphinx:document entries here ... -->

<sphinx:killlist>
<id>1234</id>
<id>4567</id>
</sphinx:killlist>

</sphinx:docset>
```

任意多的数据字段和属性都是允许的。数据字段和属性在同一文档元素中出现的先后顺序没有特别要求。。单一字段数据的最大长度有限制，超过 2MB 的数据会被截短到 2MB（但这个限制可以在配置文件中数据源部分中修改）。

XML 数据模式（Schema），即数据字段和属性的完整列表，必须在任何文档被分析之前就确定。这既可以在配置文件中用 xmlpipe_field 和 xmlpipe_attr_XXX 选 项指定，也可以就在数据流中用<sphinx:schema>元素指定。 <sphinx:schema>元素是可选的，但如果出现，就必须是<sphinx:docset>元素的第一个子元素。如果没 有在数据流中内嵌的数据模式定义，配置文件中的相关设置就会生效，否则数据流内嵌的设置被优先采用。

未知类型的标签（既不是数据字段，也不是属性的标签）会被忽略，但会给出警告。在上面的例子中，<misc>标签会被忽略。所有嵌入在其他标签中的标签及其属性都会被无视（例如上述例子中嵌入在<subject>标签中的<b>标签）

支持输入数据流的何种字符编码取决于系统中是否安装了 iconv. xmlpipe2 是用 libexpat 解析器解析的，该解析器内置对 US-ASCII, ISO-8859-1, UTF-8 和一些 UTF-16 变体的支持. CoreSeek/Sphinx 的 configure 脚本也会检查 libiconv 是否存在并使用它来处理其他的字符编码。 libexpat 也隐含的要求在 CoreSeek/Sphinx 端使用 UTF-8，因为它返回的分析过的数据总是 UTF-8 的。

xmlpipe2 可以识别的 XML 元素（标签）（以及前述元素可用的属性）如下：

sphinx:docset
　　顶级元素，用于标明并包括 xmlpipe2 文档.
sphinx:schema
　　可选元素，它要么是 sphinx:docset 的第一个子元素，要么干脆不出现。声明文档的模式。包括数据字段和属性的声明。
　　若此元素出现，则它会覆盖配置文件中对数据源的设定.
sphinx:field
　　可选元素，sphinx:schema 的子元素。声明一个全文数据字段. 可用的属性包括:

- “name”， 它指定了字段的名称，后续数据文档中具有此名称的元素的数据都被当作待检索的全文数据对待.
- “attr”， 用于特别指出将该字段作为字符串或者词汇数统计属性. 可以设置的值为 “string” 或 “wordcount”. 版本 1.10-beta 开始引入.

sphinx:attr
　　可选元素，sphinx:schema 的子元素。用于声明具体属性。可用的属性包括:

- “name”， 设定该属性名称，后续文档中具有该名称的元素应被当作一个属性对待.
- “type”， 设定该属性的类型。可能的类型包括 “int”， “timestamp”， “str2ordinal”， “bool”， “float” 和 “multi”.
- “bits”， 设定 “int” 型属性的宽度，有效值为 1 到 32.
- “default”， 设定该属性的默认值，若后续文档中没有指定这个属性，则使用此默认值.

sphinx:document
　　必须出现的元素，必须是 sphinx:docset 的子元素。包含任意多的其他元素，这些子元素带有待索引的数据字段和属性值，而这些数据字段或 属性值既可以是用 sphinx:field 和 sphinx:attr 元素声明的，也可以在配置文件中声明。唯一的已知属性是 “id”， 它必须包含一个唯一 的整型的文档 ID.
sphinx:killlist
　　可选元素，必须是 sphinx:docset 的子元素。 它包含一批“id“元素，其内容是文档编号 ID，将被作为当前索引的 失效名单列表 。

3.10. Python 数据源

Coreseek 支持使用 Python 编写数据源脚本，从而可以很方便的扩展 CoreSeek/Sphinx 的功能，来轻易的从任何 Python 可以操作的地方获取需要进行检索的数据。当前，Python 几乎支持所有的 SQL 数据库以及 NoSql 存储系统，可以查看 Python DatabaseInterfaces 获得详细列表。

```
python #用于配置 Python 数据源程序的 PYTHONPATH
{
path = /usr/local/coreseek/etc/pysource
path = /usr/local/coreseek/etc/pysource/csft_demo
}

source sourcename
{
type = python              #数据类型
name = csft_demo.MainSource    #调用的 python 的类名称
}
```

在以上配置中，对应的 Python 数据源脚本，为/usr/local/coreseek/etc/pysource/csft_demo/__init__.py，执行索引操作时，将从该脚本获取数据，请查看第 12.3 节 “Python 数据源程序接口”了解细节。

3.11. 实时索引更新

当前主要有两种方法来维护全文索引的内容是最新的。请注意，但是，这两种处理方法 的任务是应对*全文数据更新*，而不是 属性更新。从版本 0.9.8 开始支持即时的属性更新。参看 UpdateAttributes()API 调用了解详情。

方法一，使用基于磁盘的索引，手动分区，然后定期重建较小的分区（被称为“增量”）。通过尽可能的减小重建部分的大小，可以将平均索引滞后时间降低到 30~60 秒。在 0.9.x 版本中，这是唯一可用的方法。在一个巨大的文档集上，这可能是最有效的一种方法。参看第 3.12 节 “增量索引更新”了解详情。

方法二，版本 1.x（从版本 1.10-beta 开始）增加了实时索引（简写为 Rt 索引）的支持，用于及时更新全文数据。在 RT 索引上的更新，可以在 1~2 毫秒（0.001-0.002 秒）内出现在搜索结果中。然而，RT 实时索引在处理较大数据量的批量索引上效率并不高。参看 第 4 章 RT 实时索引 了解详情。

3.12. 增量索引更新

有这么一种常见的情况：整个数据集非常大，以至于难于经常性的重建索引，但是每次新增的记录却相当地少。一个典型的例子是：一个论坛有 1000000 个已经归档的帖子，但每天只有 1000 个新帖子。

在这种情况下可以用所谓的“主索引＋增量索引”（main+delta）模式来实现“近实时”的索引更新。

这种方法的基本思路是设置两个数据源和两个索引，对很少更新或根本不更新的数据建立主索引，而对新增文档建立增量索引。在上述例子 中，那 1000000 个已经归档的帖子放在主索引中，而每天新增的 1000 个帖子则放在增量索引中。增量索引更新的频率可以非常快，而文档可以在出现几分种内就 可以被检索到。

确定具体某一文档的分属那个索引的分类工作可以自动完成。一个可选的方案是，建立一个计数表，记录将文档集分成两部分的那个文档 ID，而每次重新构建主索引时，这个表都会被更新。

例 3.4. 完全自动化的实时索引

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

# in sphinx.conf
source main
{
# ...
sql_query_pre = SET NAMES utf8
sql_query_pre = REPLACE INTO sph_counter SELECT 1, MAX(id) FROM documents
sql_query = SELECT id, title, body FROM documents \
WHERE id<=( SELECT max_doc_id FROM sph_counter WHERE counter_id=1 )
}

source delta : main
{
sql_query_pre = SET NAMES utf8
sql_query = SELECT id, title, body FROM documents \
WHERE id>( SELECT max_doc_id FROM sph_counter WHERE counter_id=1 )
}

index main
{
source = main
path = /path/to/main
# ... all the other settings
}

# note how all other settings are copied from main,
# but source and path are overridden (they MUST be)
index delta : main
{
source = delta
```

```
path = /path/to/delta
}
```

请注意，上例中我们显示设置了数据源 delta 的 `sql_query_pre` 选项，覆盖了全局设置。必须显示地覆盖这个选项，否则对 delta 做索引的时候也会运行那条 `REPLACE` 查询，那样会导致 delta 源中选出的数据为空。可是简单地将 delta 的 `sql_query_pre` 设置成空也不行，因为在继承来的数据源上第一次运行这个指令的时候，继承来的*所有*值都会被清空，这样编码设置的部分也会丢失。因此需要再次显式调用编码设置查询。

3.13. 索引合并

合并两个已有的索引比重新对所有数据做索引更有效率，而且有时候必须这样做（例如在“主索引＋增量索引”分区模式中应合并主索引和增量索引，而不是简单地重新索引“主索引对应的数据）。因此 `indexer` 有这个选项。合并索引一般比重新索引快，但在大型索引上仍然*不是*一蹴而就。基本上，待合并的两个索引都会被读入内存一次，而合并后的内容需要写入磁盘一次。例如，合并 100GB 和 1GB 的两个索引将导致 202GB 的 IO 操作（但很可能还是比重新索引少）

基本的命令语法如下：

```
indexer --merge DSTINDEX SRCINDEX [--rotate]
```

SRCINDEX 的内容被合并到 DSTINDEX 中，因此只有 DSTINDEX 索引会被改变。若 DSTINDEX 已经被 `searchd` 于提供服务，则`--rotate` 参数是必须的。 最初设计的使用模式是，将小量的更新从 SRCINDEX 合并到 DSTINDEX 中。 因此，当属性被合并时，一旦出现了重复的文档 ID，SRCINDEX 中的属性值更优先（会覆盖 DSTINDEX 中的值）。 不过要注意，“旧的”关键字在这个过程中并*不*会被自动删除。 例如，在 DSTINDEX 中有一个叫做“old”的关键字与文档 123 相关联，而在 SRCINDEX 中则有关键字“new”与同一个文档相关，那么在合并后用这两个关键字*都*能找到文档 123。 您可以给出一个显式条件来将文档从 DSTINDEX 中移除，以便应对这种情况，相关的开关是`--merge-dst-range`:

```
indexer --merge main delta --merge-dst-range deleted 0 0
```

这个开关允许您在合并过程中对目标索引实施过滤。过滤器可以有多个，只有满足全部过滤条件的文档才会在最终合并后的索引中出现。在上述例子中，过滤器只允许“deleted”为 0 的那些条件通过，而去除所有标记为已删除（“deleted”）的记录（可以通过调用 UpdateAttributes()设置文档的属性）。

第 4 章 RT 实时索引

目录

实时索引（或者简称为 RT 索引）是一个新的索引后端，你可以实时的插入、更新或者删除文档（行）。RT 索引在版本 1.10-beta 中 加入。查询时 可以使用 SPhinxAPI、SphinxQL、SphinxSE，不过目前只能使用 SphinxQL 来更新。详细的 SphinxQL 参考请查看 第 7 章 SphinxQL 指南.

4.1. RT 实时索引概览

和其他类型的索引类似，RT 索引在 `sphinx.conf` 中定义。它与磁盘索引有着明显的差异，a) 数据源设置不需要而且会被忽略，b) 你应该明确的定义所有的文本字段，而不仅仅是属性。下面是例子

例 4.1. RT 实时索引定义

```
index rt
{
type = rt
path = /usr/local/sphinx/data/rt
rt_field = title
rt_field = content
rt_attr_uint = gid
}
```

RT 索引目前还在继续开发完善中（从版本 1.10-beta 开始）。 因此，他可能缺乏某些特定的功能：例如，前缀/中缀索引，MVA 属性等目前尚不支持。但是，所有常规索引功能和搜索功能都已经实现，并且经过了内部的测 试。我们还有一些实际运营生产环境（并非最不重要的部分）已经在使用 RT 索引，并且取得了较好的效果。

RT 索引可以使用 MySQL 协议访问，支持 INSERT, REPLACE, DELETE, 以及 SELECT 等 SQL 语句. 以下是一个在演示索引上的操作会话例子：

```
$ mysql -h 127.0.0.1 -P 9306
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 1.10-dev (r2153)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> INSERT INTO rt VALUES ( 1, 'first record', 'test one', 123 );
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO rt VALUES ( 2, 'second record', 'test two', 234 );
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM rt;
+------+--------+------+
| id   | weight | gid  |
+------+--------+------+
|    1 |      1 |  123 |
|    2 |      1 |  234 |
+------+--------+------+
2 rows in set (0.02 sec)

mysql> SELECT * FROM rt WHERE MATCH('test');
+------+--------+------+
| id   | weight | gid  |
+------+--------+------+
|    1 |   1643 |  123 |
|    2 |   1643 |  234 |
+------+--------+------+
2 rows in set (0.01 sec)

mysql> SELECT * FROM rt WHERE MATCH('@title test');
Empty set (0.00 sec)
```

并行和批量 INSERT 语法都支持，例如：可以先指定一个列的字段，然后同时插入多行。 删除可以使用 DELETE 语句，但是目前唯一支持的语法是 DELETE FROM <index> WHERE id=<id>。REPLACE 也支持，可以用于更新数据。

```
mysql> INSERT INTO rt ( id, title ) VALUES ( 3, 'third row' ), ( 4, 'fourth entry' );
Query OK, 2 rows affected (0.01 sec)

mysql> SELECT * FROM rt;
+------+--------+------+
| id   | weight | gid  |
+------+--------+------+
|    1 |      1 |  123 |
|    2 |      1 |  234 |
|    3 |      1 |    0 |
|    4 |      1 |    0 |
+------+--------+------+
4 rows in set (0.00 sec)

mysql> DELETE FROM rt WHERE id=2;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM rt WHERE MATCH('test');
+------+--------+------+
| id   | weight | gid  |
+------+--------+------+
|    1 |   1500 |  123 |
+------+--------+------+
```

```
1 row in set (0.00 sec)

mysql> INSERT INTO rt VALUES ( 1, 'first record on steroids', 'test one', 123 );
ERROR 1064 (42000): duplicate id '1'

mysql> REPLACE INTO rt VALUES ( 1, 'first record on steroids', 'test one', 123 );
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM rt WHERE MATCH('steroids');
+------+--------+------+
| id   | weight | gid  |
+------+--------+------+
|    1 |   1500 |  123 |
+------+--------+------+
1 row in set (0.01 sec)
```

RT 索引中存储的数据可以在正常关闭后继续存在。如果开启了二进制日志，则可以在崩溃或者非正常关闭后继续存在，并在随后的启动中恢复。

4.2. RT 实时索引使用注意事项

在版本 1.10-beta 时，RT 索引还处于特殊功能测试阶段：虽然没有大的问题，但是还有一些总所周知的以至于可以搅局的问题，以及一些特定的使用限制。这些使用限制在下面一一列出：

- 前缀和中缀索引尚不支持.
- MVA 属性尚不支持.
- 常规的磁盘块优化尚未实现
- 在索引初始化创建时，属性按照类别被重新排序，他们的顺序如下：uint, bigint, float, timestamp, string. 因此，在执行 INSERT 时，如果没有明确指定列名，就需要依次首先给出所有 uint 列的值，然后是 bigint 列的值，以此类推。
- 默认保留的内存块大小(`rt_mem_limit`)为 32M，对于大型索引而言可能性能不好，如果打算要索引 GB 级别的数据，你应该增大到 256~1024M。
- 较高比例的 DELETE/REPLACE 可导致大量的失效名单碎片和影响搜索性能。
- 目前的系统对事务处理没有做强制限制；太多并发的 INSERT/REPLACE 事务可能会耗费大量的内存。
- 在二进制日志损坏的情况下，恢复将在第一个损坏的事务处停止，即使在技术上可以继续搜寻随后完好的事务以及重新使用，也无济于事。当然，二进制日志文件中间损坏的情况（由于是片状磁盘/CDD/磁带？）是极为罕见的，甚至永远不会碰到。

4.3. RT 实时索引原理

RT 索引内部是区块化的。它保留了内存区块用于存储所有最新的变化。内存区块的内存使用由每个索引设置中的 rt_mem_limit 指令严格限制。一旦内存区块的增长超过这个限制，就会根据其内容创建一个新的磁盘区块，并将内存区块复位。因此，在 RT 索引中，虽然大多数的变化都会在内存中瞬间执行完成（毫秒级），这些内存区块溢满到磁盘区块建立的变化会有一定的延时（几秒钟）。

磁盘区块其实就是基于磁盘的索引。但是他们是 RT 索引的一部分并由其自动管理，所以无需手动配置，就算你想要管理目前也没有提供方式。因为只有在 RT 索引的内存区块达到了限制时才会创建一个新的磁盘区块，每个磁盘区块的大小基本上为 `rt_mem_limit` 个字节。

一般而言，最好设置一个较大限制值，使得刷新频率和索引碎片（磁盘区块）都尽可能最小化。例如，在一个专门的搜索服务器上要处理一个较大的 RT 索引，建议将 `rt_mem_limit` 设置为 1~2GB。全局的内存限制计划实现，但是在版本 1.10-beta 中尚未支持。

磁盘区块的全文索引数据并不会真的被修改，所以全文索引字段的变化（即删除行和更新）其实使用失效名单列表排除来自于磁盘区块的先前 版本的行，但是 实际上并不会从物理上清除数据。因此，在一个具有较高全文索引更新率的情况下，最终可能被先前版本的行污染，以及降低搜索性能。已经计划从索引中物理清除 以提高性能，但是在版本 1.10-beta 中尚未支持。

内存区块中的数据会在正常关闭时写入到磁盘，并在启动时再次载入。但是在守护进程或者服务器崩溃的情况下，内存区块中的更新数据可能会丢失。为了防止这种情况的发生，可以使用事务专用的二进制日志。查看 第 4.4 节 “二进制日志” 了解详情。

在 RT 索引中全文变化是基于事务的。他们会被存储到每个线程处理池中，并在 COMMIT 时立刻应用生效。在单次 COMMIT 中提交大量的批量更新有助于索引的快速生成。

4.4. 二进制日志

二进制日志本质上是一种故障恢复机制。当开启二进制日志时，`searchd` 将每条事务处理写入到二进制日志，并可以用于在非正常关闭后恢复数据。当正常关闭时，内存区块的数据会写入到磁盘，然后删除所有的二进制日志文件。

在正常的操作中，每当达到 binlog_max_log_size 的限制 (默认为 128M)时，就会打开一个新的二进制日志文件。已有的，已关闭的二进制日志文件会一直保存，直到它们存储的内容被刷新到磁盘区块中。如果设置限制为 0，则在 searchd 运行的过程中永远不会删除二进制日志文件；但是在正常关闭时，还是会删除掉。

目前有三种不同的二进制日志刷新策略，由 binlog_flush 选 项来控制，设置为 0 表示每秒将日志刷新一次到操作系统和同步到磁盘，设置为 1 表示表示每次事务处理都刷新和同步，设置为 2（默认模式）表示每次事务处理时 刷新单每秒同步一次。同步相对而言是比较慢的，因为他需要将数据物理写入到磁盘，所以模式 1 是最安全的模式（每次提交的事物处理，都确保是写入到磁盘 的）。刷新日志到操作系统是为了防止 searchd 崩溃时数据丢失，如果操作系统没有崩溃则数据可以安全写入到磁盘。模式 2 是默认的模式。

在非正常关闭后再次重启此进行数据恢复，二进制日志的内容会被重新执行，最后正常存储的所有事务处理都会被还原。事务处理都会被校验，以确保二进制日志文件中损坏的垃圾数据**不会**被 重新执行；因而一个破坏的事务处理将被检测到，在目前版本中它将不会被重新执行。事务处理都从一个特殊标志和时间戳开始，这样子为技术上处理跳过破坏的事 务处理并从下一条正确的开始继续执行、或者从某个指定时间戳开始重新执行事务提供了可能（时间点恢复），但是在版本 1.10-beta 时尚未实现该功能。

二进制日志存在一个副作用，那就是在一个小的 RT 索上更新时，除非正常关机，否则内存区块完全充足导致二进制日志文件不断增长而又永 远都不会被删除 掉。二进制日志基本上只追加相对于最后在磁盘上完好保存状态的增量，除非内存区块得到保存，否则它们不会被删除。一个不断增长的二进制日志对于磁盘使用和 崩溃恢复时间都不是好事。从版本 2.0.1-beta 开始，可以通过 rt_flush_period 指令配置 searchd 来周期性刷新内存区块到磁盘以解决这个问题。当周期性刷新启用时，searchd 会保留一个独立的线程用于检查 RT 索引的内存区块是否需要写回到磁盘。一旦写回到磁盘，各个二进制日志都可以（能够）安全的删除。

请注意 rt_flush_period 仅控制*检查*发生的频率。但是这并不*保证*特别是内存区块一定会得到保存。例如，定期重新保存一个巨大的内存区块但是仅更新了寥寥数行是没有任何意义的。searchd 守护进程将会通过某些启发式方法判断是否需要真的执行刷新。

第 5 章 搜索

目录

5.1. 匹配模式

有如下可选的匹配模式：

- 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 (参见 第 5.2 节 “布尔查询语法”);
- SPH_MATCH_EXTENDED, 将查询看作一个 CoreSeek/Sphinx 内部查询语言的表达式 (参见 第 5.3 节 “扩展查询语法”). 从版本 Coreseek 3/Sphinx 0.9.9 开始, 这个选项被选项 SPH_MATCH_EXTENDED2 代替，它提供了更多功能和更佳的性能。保留这个选项是为了与遗留的旧代码兼容——这样即使 Sphinx 及其组件包括 API 升级的时候，旧的应用程序代码还能够继续工作。
- SPH_MATCH_EXTENDED2, 使用第二版的“扩展匹配模式”对查询进行匹配.

- SPH_MATCH_FULLSCAN, 强制使用下文所述的“完整扫描”模式来对查询进行匹配。注意，在此模式下，所有的查询词都被忽略，尽管过滤器、过滤器范围以及分组仍然起作用，但任何文本匹配都不会发生.

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:

1. 查询串是空的（即长度字符串为零）
2. docinfo 存储方式为 `extern`.

在完整扫描模式中，全部已索引的文档都被看作是匹配的。这类匹配仍然会被过滤、排序或分组，但是并不会做任何真正的全文检索。这种模 式可以用来统一 全文检索和非全文检索的代码，或者减轻 SQL 服务器的负担（有些时候 Sphinx 扫描的速度要优于类似的 MySQL 查询）。 “在论坛中搜索帖子”这件事可用作完整搜索模式的例子：用 `SetFilter()`指定用户 ID 但不提供任何查询词，Sphinx 会匹配 `SetFilter()`所能匹配的全部文档，也就是这个用户 ID 对应的全部帖子。默认情况下，其结果的第一排序标准是相关度，其次是 Sphinx 文档 ID，正序（较老的文档在前）。

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。

5.2. 布尔查询语法

布尔查询 SPH_MATCH_BOOLEAN 允许使用下列特殊运算符:

- 显式的与（AND）运算符:

```
hello & world
```

-
- 或（OR）运算符:

```
hello | world
```

-
- 非（NOT）运算符:

```
hello -world
hello !world
```

-
- 分组（grouping）:

```
( hello world )
```

-

以下是一个使用了如上全部运算符的例子：

例 5.1. 布尔查询例子

```
( cat -dog ) | ( cat -mouse)
```

与(AND)运算符为默认操作，所以“hello world”其实就是“hello & world”

或(OR)运算符的优先级高于与运算符，因此“lookingfor cat | dog | mouse”意思是”looking for (cat | dog | mouse)” *而不是*”(looking for cat) | dog | mouse”

像“-dog”这种查询不能被执行，因为它差不多包括索引所有文档。这既有技术上的原因，也有性能上的原因。从技术上说，Sphinx 并不总是保持一个全部文档 ID 的列表。性能方面，当文档集非常大的时候（即 10-100M 个文档），对这种执行查询可能需要很长的时间。

5.3. 扩展查询语法

在扩展查询模式 SPH_MATCH_EXTENDED2 中可以使用如下特殊运算符：

- 或（OR）运算符:

```
hello | world
```

-
- 非（NOT）运算符:

```
hello -world
hello !world
```

-
- 字段（field）搜索符:

```
@title hello @body  world
```

-
- 字段限位修饰符（版本 Coreseek 3/Sphinx 0.9.9-rc1 中引入）:

```
@body[50] hello
```

-
- 多字段搜索符:

```
@(title,body) hello world
```

-
- 全字段搜索符:

```
@*  hello
```

-
- 词组搜索符:

```
"hello world"
```

-
- 近似距离搜索符:

```
"hello world"~10
```

-
- 阀值匹配符:

```
"the world is a wonderful place"/3
```

-
- 严格有序搜索符（即“在前”搜索符）:

```
aaa << bbb << ccc
```

-
- 严格形式修饰符（版本 Coreseek 3/Sphinx 0.9.9-rc1 中引入）:

```
raining =cats and =dogs
```

-
- 字段开始和字段结束修饰符 (版本 Coreseek 3.1/Sphinx 0.9.9-rc2 中引入):

```
^hello world$
```

-
- NEAR, 广义临近运算符 (版本 2.0.1-beta 中引入):

```
hello NEAR/3 world NEAR/4 "my test"
```

-
- SENTENCE 句子运算符 (版本 2.0.1-beta 中引入):

```
all SENTENCE words SENTENCE "in one sentence"
```

-

- PARAGRAPH 段落运算符 (版本 2.0.1-beta 中引入):

```
"Bill Gates" PARAGRAPH "Steve Jobs"
```

-
- ZONE 区域限制运算符 (版本 2.0.1-beta 中引入):

```
ZONE:(h3,h4) only in these titles
```

-

以下是上述某些运算符的示例：

例 5.2. 扩展匹配模式：查询例子

```
"hello world" @title "example program"~10 @body  python -(php|perl) @*  code
```

例子中查询的完整解释如下：

- 在文档的任意字段中找相邻的“hello”和“world”
- 不仅如此，符合上述条件的文档的 title 字段中还必须包含 “example”和“program”这两个词，并且他们之间至多有 10 个（不包括 10 个）其他的词（例如“example PHP program”可以匹配，但“example script to introduce outside data into the correct context for your program”就不行，因为中间有 10 个或以上的词。
- 同时，body 字段必须含有词“python”，但既没有“php”也没有“perl”
- 最后，任一字段中包含”code“.

与(AND)操作为默认操作，因此“hello world”意思是“hello”和“world”必须同时存在文档才能匹配。

或（OR）运算符的优先级要高于与运算符，因此”looking for cat | dog | mouse” 意思是”looking for (cat | dog | mouse)” 而*不是* “(looking for cat) | dog | mouse”；

字段限制(field limit)符(field limit)将其后指定的搜索限制在某个特定的字段中。通常，如果给出的字段名实际并不存在，你会得到一条错误信息。但可以通过在查询的最开始处加上@@relaxed 选项来放宽限制。

```
@@relaxed @nosuchfield my query
```

当搜索多个具有不同 schema 的索引时这可能有用。

版本 Coreseek 3/Sphinx 0.9.9-rc1 又引入了字段限位（field position limit）符。它把搜索限制在指定字段（一个或多个）的前 N 个位置。例如“@body[50] hello”**不会**匹配那些 body 字段包含“hello”，但它出现在第 51 个位置或者更靠后的文档。

近似距离以词为单位，随词数变化而变化，并应用于引号中的全部词。举个例子，”cat dog mouse”~5 这个查询的意思是必须有一个少于 8 个词的词串，它要包含全部的三个词，也就是说”CAT aaa bbb ccc DOG eee fff MOUSE” 这个文档*不会*匹配这个查询，因为这个词串正好是 8 个词。

阀值匹配符引入了一种模糊匹配。它允许至少含有某个阈值数量个匹配词的文档通过。上述例子（”the world is a wonderful place”/3）会匹配含有指定的六个词中的至少三个的那些文档。上面例子中的一个查询”the world is a wonderful place”/3 匹配的文档至少含有指定的 6 个词中的 3 个。

严格有序搜索符（即“在前”搜索符）是在版本 0.9.9-rc2 中引入的，它的几个参数在被匹配的文档中必须严格按查询中出现的顺序出现。例如，“black << cat”这个查询（不包括引号）可以匹配“black and white cat”，但*不能*匹配“the cat was black”。顺序运算符的优先级最低，它既可以应用在最简单的关键词上，也可以用在更复杂的表达式上，比如下面也是个正确的查询：

```
(bag of words) << "exact phrase" << red|green|blue
```

版本 0.9.9-rc1 引入了“严格形式”关键字修饰符，它保证关键词在匹配文档中严格以指定的形式出现，而默认行为只要求词根相同。例如，查询“runs”既可以匹配含有“runs”的文档，*也*可以匹配含有“running”的文档，因为这二者的词根都是“run”——而如果查询是“=runs”，那就只有前者能匹配。严格形式修饰符要求 index_exact_words 选项处于启用状态。这是个影响关键字的修饰符，可以与其他一些运算符混合使用，例如词组搜索符、近似搜索符和阈值搜索符等。

关键字修饰符“字段开始”和“字段结束”是在版本 Coreseek 3.1/Sphinx 0.9.9-rc2 中引入的，它们确保只在一个全文字段的最开始或最结束位置匹配关键字。例如，查询“^hello world$”(包括引号，也就是说这个查询是词组搜索符和字段起止修饰符的组合)匹配的文档必然包括某个严格只有“hello world”这个词组的字段。

自版本 Coreseek 3/Sphinx 0.9.9-rc1 始，可以嵌套任意层数的括号和“非”操作，但这类查询要想能够计算出结果，就必须保证不能隐含地涉及所有文档。

```
// 正确查询
aaa -(bbb -(ccc ddd))

// 不能处理的查询
-aaa
aaa | -bbb
```

NEAR 广义临近运算符, 版本 2.0.1-beta 引入, 是近似距离运算符的广义版本。其语法是 `NEAR/N`, 区分大小写, 并且不允许有空格在 NEAR 关键词与斜线以及距离值之间。

原始的近似距离运算符仅对给如的关键词集起作用。而 NEAR 则更加通用，可以接受任意的子表达式作为其参数，当两个表达式在 N 个词汇的距离中（不论先后顺序）发现对方时文档被选中。NEAR 是左结合的，与“在前”搜索符具有相同（最低）的优先级。

你需要明白为什么一个使用 NEAR 的查询`(one NEAR/7 two NEAR/7 three)` 并不完全等同于一个使用关键词近似距离的查询 `("one two three"~7)`。 这里的区别在于，近似距离运算符允许最多 6 个没有匹配的词汇出现在 3 个匹配到的词汇之间，但是 NEAR 的版本限制较少: 它允许最多 6 个词汇出现在 ‘one’ 与 ‘two’ 之间，然后允许匹配到的 two 关键词与’three’之间有最多 6 个词汇。

SENTENCE 句子 和 PARAGRAPH 段落运算符, 在版本 2.0.1-beta 中引入, 当它的参数都处于同一个句子或者段落中时文档被选中。参数可以是关键词或者短语，活着相同的运算符实例。下面是几个例子：

```
one SENTENCE two
one SENTENCE "two three"
one SENTENCE "two three" SENTENCE four
```

参数在句子或者段落中的顺序不产生影响。该功能仅在 index_sp (句子和段落索引功能) 启用时生效，否则作为单纯的 AND 操作对待。参考 `index_sp` 章节的文档了解系统是如何来判断一个句子或者段落的。

ZONE 区域限制运算符, 版本 2.0.1-beta 引入, 非常类似于字段限制运算符，但是限制在一个给定的字段内或者区域的列表。请注意，其随后的字表达式并*不*要求限定在一个给定的连续区域之内，可以跨越多个相同的区域进行匹配。例如, `(ZONE:th hello world)` 查询 *将* 匹配下面演示的文档:

```
<th>Table 1. Local awareness of Hello Kitty brand.</th>
.. some table data goes here ..
<th>Table 2. World-wide brand awareness.</th>
```

ZONE 运算符对查询的影响直到下一个查询字段或者区域限制符号，或者到右括号结束。它仅适用于启用了区域支持(参看 第 11.2.9 节 “index_zones：索引标签区域信息”) 的索引，否则将被忽略掉。

5.4. 权值计算

采用何种权值计算函数（目前）取决于查询的模式。

权值计算函数有如下两个主要部分：

1. phrase rank,
2. statistical rank.

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。

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。

在所有模式中，数据字段的词组评分是 LCS 乘以用户指定的数据字段权值。数据字段权值是整数，默认为 1，且字段的权值必须不小于 1。

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_ANY 模式中，于前面述两模式的基本思想类似，只是每个数据字段的权重都再加上一个匹配词数目。在那之前，带权的词组相关度被额外乘以一个足够大的数，以便确保**任何**一个有较大词组评分的数据字段都会使整个匹配的相关度较高，即使该数据字段的权重比较低。

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。

这个行为将来会被修改，以便使 MATCH_ALL 和 MATCH_ANY 这两个模式也能使用 BM25 算法。这将使词组评分相同的搜索结果片断得到改进，这在只有一个词的查询中尤其有用。

关键的思想（对于除布尔模式以外的全部模式中）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。作者的经验是，这种基于词组相似性的评分方法可以提供比任何单纯的统计模型（比如其他搜索引擎中广泛使用的 BM25）明显更高的搜索质量。

5.5. 表达式，函数，运算符

Sphinx 允许你在 SphinxQL 和 SphinxAPI 中使用任意的表达式，可以涉及属性的值，内部属性（文档编号和相关度），算术运算符，内置的函数和用户自定义的函数。本章节是支持的运算符和函数的说明文档。下面是用于快速查看的完整参考列表。

- 算术运算符: +, -, *, /, %, DIV, MOD
- 比较运算符: <, > <=, >=, =, <>
- 布尔运算符: AND, OR, NOT
- 位运算符: &, |
- ABS()
- BIGINT()
- CEIL()
- COS()
- CRC32()
- DAY()
- EXP()
- FLOOR()
- GEODIST()
- IDIV()
- IF()
- IN()
- INTERVAL()
- LN()
- LOG10()
- LOG2()
- MAX()
- MIN()
- MONTH()
- NOW()
- POW()
- SIN()
- SINT()
- SQRT()
- YEAR()
- YEARMONTH()
- YEARMONTHDAY()

5.5.1. 运算符

算术运算符: +, -, *, /, %, DIV, MOD

标准算术运算符。计算可以以三种不同的精度进行：(a) 单精度 32 位 IEEE754 浮点值（默认情况），(b) 32 位有符号整数，(c) 64 位有符号整数。如果没有任何返回浮点数的操作，表达式分析器会自动切换到整数模式，否则使用默认的浮点模式。比如，对于表达式“ a+b”，如果两个参数都是 32 位整数的，则它会被以 32 位整数模式计算，如果两个参数都是整数，而其中一个是 64 位的，则以 64 位整数模式计算，否则以浮点模式计算。然而表达式“ a/b”或者“ sqrt(a)”却总是使用浮点模式计算，因为这些操作返回非整数的结果，要让前者使用整数模式，可以使用 IDIV(a,b)或者 a DIV b。另外，如果两个参数都是 32 位的，表达式“ a*b”并不会自动提升到 64 位模式。要想强制 64 位模式，可以用 BIGINT()。（但要注意的是，如果表达式中同时有浮点数，那么 BIGINT()的命运就是简单地被忽略）

比较运算符: <, > <=, >=, =, <>

比较操作符（比如=和<=）在条件为真时返回 1.0，否则返回 0.0。例如 (a=b)+3 在属性“a”与属性“b”相等时返回 4，否则返回 3。与 MySQL 不同，相等性比较符（即=和<>）中引入了一个小的阈值（默认是 1e-6）。如果被比较的两个值的差异在阈值之内，则二者被认为相等。

布尔运算符: AND, OR, NOT

布尔操作符（AND,OR,NOT）是在版本 Coreseek 3.1/Sphinx 0.9.9-rc2 中引入的，其行为与一般的布尔操作没有两样。它们全部是左结合，而且比之其他操作符，它们有最低的优先级，其中 NOT 的优先级比 AND 和 OR 高，但仍旧低于所有其他操作符。AND 和 OR 有相同的优先级，因此建议使用括号来避免在复杂的表达式中出现混乱。

位运算符: &, |

这些运算符分别进行按位 AND 和 OR。操作数 必须是整数类型。版本 1.10-beta 引入。

5.5.2. 数值函数

ABS(x)

返回 x 的绝对值。

CEIL(x)

返回大于或等于该 x 的最小整数。

COS(x)

返回 x 的余弦。

EXP(x)

返回 e 的 X 乘方后的值（e=2.718…作为自然对数的底）。

FLOOR(x)

返回小于或等于该 x 的最大整数。

IDIV(x, y)

返回 x 整除 y 的结果。两个参数都必须是整数类型。

LN(x)

返回 x 的自然对数 (自然对数的底为 e=2.718…).

LOG10(x)

返回 x 的常用对数 (底为 10).

LOG2(x)

返回 x 的二进制对数 (底为 2).

MAX(x, y)

返回 x, y 中较大的参数.

MIN(x, y)

返回 x, y 中较小的参数.

POW(x, y)

返回 x 的 y 次方的冥。

SIN(x)

返回 x 的正弦.

SQRT(x)

返回 x 的平方根.

5.5.3. 日期和时间函数

DAY(t)

根据当前时区，返回时间戳 t 对应的所在月日期（范围在 1.. 31 ）。版本 2.0.1-beta 引入。

MONTH(t)

根据当前时区，返回时间戳 t 对应的月份（范围在 1.. 12 ）。版本 2.0.1-beta 引入。

NOW()

将当前时间戳作为整数返回。版本 0.9.9-rc1 引入。

YEAR(t)

根据当前时区，返回时间戳 t 对应的年份（范围在 1969.. 2038 ）。版本 2.0.1-beta 引入。

YEARMONTH(t)

根据当前时区，返回时间戳 t 对应的整数年份和月份（范围在 196912..203801 ）。版本 2.0.1-beta 引入。

YEARMONTHDAY()

根据当前时区，返回时间戳 t 对应的整数年份、月份和日期（范围在 19691231..20380119 ）。版本 2.0.1-beta 引入。

5.5.4. 类型转换函数

BIGINT()

它将它的整型参数强行提升到 64 位，而对浮点参数无效。引入它是为了可以强制某些表达式（如“ a*b”）用 64 位模式计算，即使所有的参数都是 32 位的。 版本 0.9.9-rc1 引入。

SINT()

强制将 32 位无符号整数重新诠释为有符号数，并提示道 64 位（因为 32 位类型是无符号的）。通过以下的例子很容易明白：通常 1-2 的计算结果（32 位无符 号整数）为 4294967295，但是 SINT(1-2)的计算结果为-1。 版本 2.0.1-beta 引入。

5.5.5. 比较函数

IF(expr, x, y)

IF() 行为与其对应的 MySQL 的有所不同。它接受 3 个参数，检查第一个参数 expr 是否为 0.0，若非零则返回第二个参数 x ，为零时则返回第三个参数 y 。注意，与比较操作符不同， IF()并 不使用阈值！因此在第一个参数中使用比较结果是安全的，但使用算术运算符则可能产生意料之外的结果。比如，下面两个调用会产生 *不同*的结果，虽然在逻辑上他们是等价的：

```
IF ( sqrt(3)*sqrt(3)-3<>0, a, b )
IF ( sqrt(3)*sqrt(3)-3, a, b )
```

在第一种情况下，由于有阈值，比较操作符<>返回 0.0（逻辑假），于是 IF()总是返回‘b’。在第二种情况下，IF()函数亲自擅自在*没有*阈值的情况下将同样的 sqrt(3)*sqrt(3)-3 与零值做比较。但由于浮点数运算的精度问题，该表达式的结果与 0 值会有微小的差异，因此该值与零值的相等比较不会通过，上述第二种情况中 IF()会返回‘a’做为结果。

IN(expr, val1, …)

版本 0.9.9-rc1 引入了函数 IN(expr, val1, val2, …)，它需要两个或更多参数，如果第一个参数与后续任何一个参数（val1 到 valN）相等则返回 1，否则返回 0。目前，所有被测试是否相等的参数（不包 括 expr 本身）必须是常量。（支持任意表达式在技术上是可以实现的，未来我们会这么做）。这些常量经过预先排序，测试相等时可以使用二元查找以提高效 率，因此即使参数列表很长 IN()也还可以提供较高的速度。自版本 0.9.9-rc2 始，第一个参数可以是一个 MVA 多值属性，这种情况下只要 MVA 中的 任何一个值与后面列表中的任何一个值相等 IN()就返回 1。 从版本 2.0.1-beta 开始，IN()也支持 IN(expr,@uservar)语法以检查其值是否在给定的全局用户变量列表中。

INTERVAL(expr, point1, …)

版本 0.9.9-rc1 引入了函数 INTERVAL(expr, point1, point2, point3, …)，它接受 2 个或更多参数，返回第一个小于第一个参数 expr 的参数的下标：如果 expr<point1，返回 0；如果 point1<=expr<point2，返回 1，一次类推。显然，必须有 point1<point2<…<pointN 才能保证这个函数正确工作。

5.5.6. 其它函数

CRC32(s)

返回字符串 s 的 CRC32 值. 版本 2.0.1-beta 引入。

GEODIST(lat1, long1, lat2, long2)

版本 0.9.9-rc2 引入函数 GEODIST(lat1, long1, lat2, long2)，它根据坐标计算两个指定点之间的地表距离。请注意经纬度都要以角度为单位，而结果是以米为单位的。四个参数都可以是任意表达式。当其中一对 参数引用的是文档属性对而另一对参数是常数，系统会自动选择一条优化的路径。

5.6. 排序模式

可使用如下模式对搜索结果排序：

- 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。

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”。

SPH_SORT_TIME_SEGMENTS 模式

在 SPH_SORT_TIME_SEGMENTS 模式中，属性值被分割成“时间段”，然后先按时间段排序，再按相关度排序。

时间段是根据搜索发生时的*当前时间戳*计算的，因此结果随时间而变化。所说的时间段有如下这些值：

- 最近一小时
- 最近一天
- 最近一周
- 最近一月
- 最近三月
- 其他.

时间段的分法固化在搜索程序中了，但如果需要，也可以比较容易地改变（需要修改源码）。

这种模式是为了方便对 Blog 日志和新闻提要等的搜索而增加的。使用这个模式时，处于更近时间段的记录会排在前面，但是在同一时间段中的记录又根据相关度排序－这不同于单纯按时间戳排序而不考虑相关度。

SPH_SORT_EXTENDED mode

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：

```
@relevance DESC, price ASC, @id DESC
```

只要做了相关设置，不管是内部属性（引擎动态计算出来的那些属性）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，用户属性按原样使用就行了。在上面的例子里，`@relevance` 和`@id` 是内部属性，而 `price` 是用户定义属性。

可用的内置属性：

- @id (匹配文档的 ID)
- @weight (匹配权值)
- @rank (等同 weight)
- @relevance (等同 weight)
- @random (随机顺序返回结果)

`@rank` 和 `@relevance` 只是 `@weight` 的别名.

SPH_SORT_EXPR 模式

表达式排序模式使您可以对匹配项按任何算术表达式排序，表达式中的项可以是属性值，内部属性（@id 和@weight），算术运算符和一些内建的函数。例如：

```
$cl->SetSortMode ( SPH_SORT_EXPR,
"@weight + ( user_karma + ln(pageviews) )*0.1" );
```

在表达式中支持的运算符和函数请参看专门的章节第 5.5 节 “表达式，函数，运算符”.

5.7. 结果分组（聚类）

有时将搜索结果分组（或者说“聚类”）并对每组中的结果计数是很有用的－例如画个漂亮的图来展示每个月有多少的 blog 日志，或者把 Web 搜索结果按站点分组，或者把找到的论坛帖子按其作者分组等等。

理论上，这可以分两步实现：首先在 Sphinx 中做全文检索，再在 SQL 服务器端对得到的 ID 分组。但是现实中在大结果集（10K 到 10M 个匹配）上这样做通常会严重影响性能。

为避免上述问题，Sphinx 提供了一种“分组模式”，可以用 API 调用 SetGroupBy()来开启。在分组时，根据 group-by 值给匹配项赋以一个分组。这个值用下列内建函数之一根据特定的属性值计算：

- SPH_GROUPBY_DAY, 从时间戳中按 YYYYMMDD 格式抽取年、月、日;
- SPH_GROUPBY_WEEK, 从时间戳中按 YYYYNNN 格式抽取年份和指定周数（自年初计起）的第一天;
- SPH_GROUPBY_MONTH, 从时间戳中按 YYYYMM 格式抽取月份;
- SPH_GROUPBY_YEAR, 从时间戳中按 YYYY 格式抽取年份;
- SPH_GROUPBY_ATTR, 使用属性值自身进行分组.

最终的搜索结果中每组包含一个最佳匹配。分组函数值和每组的匹配数目分别以“虚拟”属性 @group 和 @count 的形式返回.

结果集按 group-by 排序子句排序，语法与 SPH_SORT_EXTENDED 排序子句的语法相似。除了`@id` 和`@weight`，分组排序子句还包括：

- @group (groupby 函数值),
- @count (组中的匹配数目).

默认模式是根据 groupby 函数值降序排列，即按照 "@group `desc`".

排序完成时，结果参数 `total_found` 会包含在整个索引上匹配的组的总数目。

注意: 分组操作在固定的内存中执行，因此它给出的是近似结果；所以 `total_found` 报告的数目可能比实际给出的个分组数目的和多。@count 也可能被低估。要降低不准确性，应提高 `max_matches`。如果 `max_matches` 允许存储找到的全部分组，那结果就是百分之百准确的。

例如，如果按相关度排序，同时用 `SPH_GROUPBY_DAY` 函数按属性"`published`"分组，那么：

- 结果中包含每天的匹配结果中最相关的那一个，如果那天有记录匹配的话,
- 结果中还附加给出天的编号和每天的匹配数目,
- 结果以天的编号降序排列（即最近的日子在前面）.

从版本 0.9.9-rc2 开始，当使用 GROUP BY 时，可以通过 SetSelect() API 调用聚合函数 (AVG(), MIN(), MAX(), SUM())

5.8. 分布式搜索

为提高可伸缩性，Sphnix 提供了分布式检索能力。分布式检索可以改善查询延迟问题（即缩短查询时间）和提高多服务器、多 CPU 或多核环境下的吞吐率（即每秒可以完成的查询数）。这对于大量数据（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。

其关键思想是对数据进行水平分区（HP，Horizontally partition），然后并行处理。

分区不能自动完成，您需要

- 在不同服务器上设置 Sphinx 程序集（`indexer` 和 `searchd`）的多个实例;
- 让这些实例对数据的不同部分做索引（并检索）;
- 在 `searchd` 的一些实例上配置一个特殊的分布式索引;
- 然后对这个索引进行查询.

这个特殊索引只包括对其他本地或远程索引的引用，因此不能对它执行重新建立索引的操作，相反，如果要对这个特殊索引进行重建，要重建的是那些被这个索引被引用到的索引。

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.

在应用程序看来，普通索引和分布式索引完全没有区别。 也就是说，分布式索引对应用程序而言是完全透明的，实际上也无需知道查询使用的索引是分布式的还是本地的。 (就算是在 0.9.9 之中， Sphinx 也不支持以其他的方式来结合分布式索引进行搜索, 也许在将来会去掉该限制.)

任一个 `searchd` 实例可以同时做为主控端（master，对搜索结果做聚合）和从属端（只做本地搜索）。这有如下几点好处：

1. 集群中的每台机器都可以做为主控端来搜索整个集群，搜索请求可以在主控端之间获得负载平衡，相当于实现了一种 HA（high availability，高可用性），可以应对某个节点失效的情况。
2. 如果在单台多 CPU 或多核机器上使用，一个做为代理对本机进行搜索的 searchd 实例就可以利用到全部的 CPU 或者核。

如果在单台多 CPU 或多核机器上使用，一个做为代理对本机进行搜索的 searchd 实例就可以利用到全部的 CPU 或者核。

5.9. 搜索服务(searchd) 查询日志格式

在版本 2.0.1-beta 中支持下述的两种查询日志格式。以前的版本只支持自定义的纯文本格式，目前也仍然是默认的日志格式。 然而，虽然它可能会方便人工监测和审查，但难以回放以便做性能测试，它只能 记录*搜索*查询而不能记录搜索请求的其他信息，也不总是包含完整的搜索查询数据等。 默认纯文本格式也很难（有时不可能）为性能测试的目的而回放。新的 `sphinxql` 格式则缓解了这个问题，其目的是完整和自动化的记录信息，即使降低了简洁和可读性也无所谓。

5.9.1. 纯文本日志格式

默认情况下，`searchd` 将全部成功执行的搜索查询都记录在查询日志文件中。以下是一个类似记录文件的例子：

```
[Fri Jun 29 21:17:58 2007] 0.004 sec [all/0/rel 35254 (0,20)] [lj] test
[Fri Jun 29 21:20:34 2007] 0.024 sec [all/0/rel 19886 (0,20) @channel_id] [lj] test
```

日志格式如下

```
[query-date] query-time [match-mode/filters-count/sort-mode
total-matches (offset,limit) @groupby-attr] [index-name] query
```

匹配模式（match-mode）可以是如下值之一

- “all” 代表 SPH_MATCH_ALL 模式;
- “any” 代表 SPH_MATCH_ANY 模式;
- “phr” 代表 SPH_MATCH_PHRASE 模式;
- “bool” 代表 SPH_MATCH_BOOLEAN 模式;
- “ext” 代表 SPH_MATCH_EXTENDED 模式;
- “ext2″ 代表 SPH_MATCH_EXTENDED2 模式;
- “scan” 代表使用了完整扫描模式，这可能是由于设置了 SPH_MATCH_FULLSCAN 模式导致的，也可能是因为查询是空的。 (参见文档 匹配模式)

排序模式（sort-mode）可以取如下值之一:

- “rel” 代表 SPH_SORT_RELEVANCE 模式;
- “attr-” 代表 SPH_SORT_ATTR_DESC 模式;
- “attr+” 代表 SPH_SORT_ATTR_ASC 模式;
- “tsegs” 代表 SPH_SORT_TIME_SEGMENTS 模式;
- “ext” 代表 SPH_SORT_EXTENDED 模式.

此外，如果 `searchd` 启动的时候带参数 `--iostats`，那么在列出被搜索的全部索引后还会给出一块数据。

一个查询日志项看起来就像这样:

```
[Fri Jun 29 21:17:58 2007] 0.004 sec [all/0/rel 35254 (0,20)] [lj]
[ios=6 kb=111.1 ms=0.5] test
```

多出来的这块数据是关于搜索中执行的 I/O 操作的信息，包括执行的 I/O 操作次数、从索引文件中读取数据的 kb 数和 I/O 操作占用的时间（尽管这个时间还包括一个后台处理组件所占用的，但主要是 I/O 时间）

5.9.2. SphinxQL 日志格式

这是版本 2.0.1-beta 引入的新的日志格式，其目标是开始以一种简单的格式去自动记录所有的一切（例如为了自动回放）。 新的格式可以通过配置文件中的 query_log_format 指令来启用， 或者在访问过程中通过 SphinxQL 语句 SET GLOBAL query_log_format=... 进行来回切换。在新的格式中，前面的例子将如下所示：（为了方便阅读进行了换行，但是在实际的日志中每行一个查询）

```
/* Fri Jun 29 21:17:58.609 2007 2011 conn 2 wall 0.004 found 35254 */
SELECT * FROM lj WHERE MATCH('test') OPTION ranker=proximity;

/* Fri Jun 29 21:20:34 2007.555 conn 3 wall 0.024 found 19886 */
SELECT * FROM lj WHERE MATCH('test') GROUP BY channel_id
OPTION ranker=proximity;
```

请注意， **所有**的请求都将以此格式记录， 包括那些通过 SphinxAPI 和 SphinxSE 发送的请求，而不只是那些通过 SphinxQL 发送的请求。 还需要注意的是，这种日志记录方式仅适合纯文本日志格式，如果使用'syslog'记录则不会工作。

SphinxQL 日志格式相对默认的纯文本日志格式的特点如下：

- 所有类型的请求都会被记录。 (该工作还在继续和完善.)
- 全部报表数据将被尽可能的记录。
- 错误和提醒也会被记录。
- 该日志可以通过 SphinxQL 自动回放。
- 额外的性能计数器（目前，记录了每个 Agent 的分布式查询时间）被记录。

每个请求（包括 SphinxAPI 和 SphinxQL ）都正好对应一个日志记录行。 所有的请求类型，包括 INSERT, CALL SNIPPETS 等，都会被记录（但 编写本手册时，这方面的工作还在继续和完善）。 每个记录行都是有效的 SphinxQL 语句，可以用于完整的重构请求，但是请求记录如果太大则需要考虑性能原因而缩短。 其他信息，例如性能计数器等会被作为注释记录在请求后。

5.10. MySQL 协议支持与 SphinxQL

Sphinx 的 searchd 守护程序从 版本 0.9.9-rc2 开始支持 MySQL 二进制网络协议，并且能够通过标准的 MySQL API 访问。例如，"mysql"命令行程序可以很好地工作。以下是用 MySQL 客户端对 Sphinx 进行查询的例子：

```
$ mysql -P 9306
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 0.9.9-dev (r1734)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT * FROM test1 WHERE MATCH('test')
-> ORDER BY group_id ASC OPTION ranker=bm25;
+------+--------+----------+------------+
| id   | weight | group_id | date_added |
+------+--------+----------+------------+
|    4 |   1442 |        2 | 1231721236 |
|    2 |   2421 |      123 | 1231721236 |
|    1 |   2421 |      456 | 1231721236 |
+------+--------+----------+------------+
3 rows in set (0.00 sec)
```

请注意 mysqld 甚至根本没有在测试机上运行。所有事情都是 searchd 自己搞定的。

新的访问方法是对原生 API 的一种 *补充*，原生 API 仍然完美可用。事实上，两种访问方法可以同时使用。另外，原生 API 仍旧是默认的访问方法。MySQL 协议支持需要经过额外的配置才能启用。当然这只需要更动一行配置文件，加入一个协议为 mysql41 的监听器（listener）就可以了：

```
listen = localhost:9306:mysql41
```

如果仅仅支持这个协议但不支持 SQL 语法，那没什么实际意义。因此 Sphinx 现在还支持 SQL 的一个很小的子集，我们给这个子集起 个绰号，叫 SphinxQL。它支持在所有类型的索引上进行标准 SELECT 查询，支持在 RT 索引上使用 INSERT、REPLACE、DELETE 来更改信息，以 及更多的处理。 详细的 SphinQL 指南请查看 第 7 章 SphinxQL 指南.

5.11. 批量查询

多查询，或者批量查询，用于一次发送多条查询到 Sphinx（更专业的说法是再一次网络请求中）。

目前有两个方法来实现批量查询，他们是 AddQuery() 和 RunQueries()。 你也可以通过 SphinxQL 来执行批量查询，参见 第 7.18 节 "多结果集查询（批量查询）". (事实上，常规的 Query() 调 用在内部实现上是一次单独的 AddQuery()调用紧跟 RunQueries()调用。) AddQuery()会保持前面的 API 调用对查询设置的当前状态，并且记住这些查询。RunQueries()实际上发送所有记住的查询，并依次返回多 个结果集。 对于查询本身没有任何限制，但在一次批量查询中会进行查询数目的完成检查 (参见 第 11.4.25 节 "max_batch_queries：最大批量查询").

为什么要使用批量查询呢？ 通常是是为了提升性能。 首先，在一次批量查询中将多个请求一次发送到 searchd 而不是由一个接一个的发送，或多或少会减少网络通信。 其次，更重要的是，在批量查询中发送多条查询时，searchd 会执行某些内部优化。随着时间的推移逐渐增加了新的优化措施，在可能的情况下将所有的查询打包为批量查询是有意义的，因此简单地升级 Sphinx 到一个新的版本将自动启用新的优化措施。 在处理时如果没有任何可能的批量查询优化可应用，查询在内部将被一个接一个的处理。

为什么（或者说什么时候）不使用批量查询呢？ 批量查询要求一次处理中的查询都是相互独立的，但是有时候并非如此。 也就是说，有时候查询 B 是基于查询 A 的结果的，只能在查询 A 运行完成以后才能生成新的查询。例如，你可能需要在主索引查不到任何结果的时候，再从第二个索 引查询显示结果。或者，有时候需要根据第一次查询的结果数去指定第二次查询的结果偏移量。诸如此类情况下，你不得不使用多次分开的查询（或者多次分开的批 量查询）。

截止版本 0.9.10，有两个主要的优化措施需要注意： 公共查询优化（从版本 0.9.8 开始提供）；公共子树优化（从版本 0.9.10 开始提供）。

公共查询优化 是指 searchd 发现批量查询中的查询仅排序和分组设置不同时，将*只执行一次搜索*。 例如，如果一次批量查询包含三个查询，查询关键字全部都是“ipod nano”，但是第一个查询取按照价格排序前十的结果，第二个查询按照供应商分组并取评分最高的前五个供货商，而第三个查询取最高价格的，则针 对”ipod nano”的全文查询仅执行一次，但其结果会重复使用来建立三个不同的结果集。

从这个优化措施得到好处的一个典型的案例就是所谓的 **faceted 分面搜索**。事实上，分面搜索可以通过执行一定数量的查询，一来获取搜索结果本身，同时进行相同全文关键字的一些其他查询，如根据不同的分组设置所有必要的分组结果（前三作者，前五供货商等等）。而这些只需要保持全文检索和过滤设置相同，公共查询优化将会触发并大幅提高查询性能。

公共子树优化 更加有趣。 它能让 searchd 分析利用批量全文查询中的相似之处。它可以识别所有查询中的公共全文查询部分（子树），并且在查询中缓存其结果。例如，我们可以看下面的批量查询：

```
barack obama president
barack obama john mccain
barack obama speech
```

他们有一个共同的部分是两个单词(“barack obama”)，这将被计算一次，然后缓存并在查询之间共享。这个工作就是公共子树优化来做的。每个查询的缓存大小由 subtree_docs_cache 和 subtree_hits_cache 指令严格控制（使得即使缓存匹配了“i am”的超多文档也不会耗尽内存和导致服务服务器挂掉）

下面演示的代码（使用 PHP ）展示了在同一查询中使用 3 种不同 排序方式：

```
require ( "sphinxapi.php" );
$cl = new SphinxClient ();
$cl->SetMatchMode ( SPH_MATCH_EXTENDED2 );

$cl->SetSortMode ( SPH_SORT_RELEVANCE );
$cl->AddQuery ( "the", "lj" );
$cl->SetSortMode ( SPH_SORT_EXTENDED, "published desc" );
$cl->AddQuery ( "the", "lj" );
$cl->SetSortMode ( SPH_SORT_EXTENDED, "published asc" );
$cl->AddQuery ( "the", "lj" );
$res = $cl->RunQueries();
```

如何确认批量查询的请求真的被优化了呢？ 如果优化过，在他们各自的查询日志将有一个“倍数“区段 特别之处有多少个查询是一起处理的：

```
[Sun Jul 12 15:18:17.000 2009] 0.040 sec x3 [ext2/0/rel 747541 (0,20)] [lj] the
[Sun Jul 12 15:18:17.000 2009] 0.040 sec x3 [ext2/0/ext 747541 (0,20)] [lj] the
[Sun Jul 12 15:18:17.000 2009] 0.040 sec x3 [ext2/0/ext 747541 (0,20)] [lj] the
```

请注意 “x3″ 区段. 这意味着该查询是优化的并且在同一批次中处理了 3 个查询。作为参考，没有进行批量查询处理的多次查询的通常日志会是下面这样：

```
[Sun Jul 12 15:18:17.062 2009] 0.059 sec [ext2/0/rel 747541 (0,20)] [lj] the
[Sun Jul 12 15:18:17.156 2009] 0.091 sec [ext2/0/ext 747541 (0,20)] [lj] the
[Sun Jul 12 15:18:17.250 2009] 0.092 sec [ext2/0/ext 747541 (0,20)] [lj] the
```

请注意每个查询多时间查询案件是由一个因素改善 的 1.5 倍至 2.3 倍，这取决于在特定的排序模式。 其实在这两方面， 常见的查询优化和共同子树，还有的报告和 3 倍 甚至更多的改进，而且从生产情况的，而不仅仅是 综合测试。 值得提醒的是批量查询方式中每个查询的时间会改善 1.5 倍到 2.3 倍，具体如何取决于特定的排序模式。其实，不仅在综合测试中，公共查询优化和公共子树优 化都会提升 3 倍或者更多的改善，在实际生产运营环境中也是如此。

5.12. 字符串排序规则

Sphinx 从版本 2.0.1-beta 开始支持字符串排序，而排序规则势必会影响字符串属性的比较。Sphinx 在进行字符串属性的 ORDER BY 或者 GROUP BY 操作时所进行的字符串比较，与字符集的编码设定和策略有关。

在索引时，字符串属性仅仅是存储起来，没有字符集 或语言信息附加给他们。 这没关系，Sphinx 只需要存储并逐字返回字符串给调用的应用程序。 但是如果你让 Sphinx 进行字符串值排序，这个请求就会产生歧义。

首先，单字节（ASCII 码或 ISO – 8859 – 1 或 Windows – 1251 ）字符串 处理不同于 UTF – 8 编码的字符串（可能 每一个字符包含可变数目的字节）。 因此，我们需要知道字符集的类型，以便能够正确地解析每个原始字节为有意义的字符。

第二，我们还需要知道特定语言的 字符串排序规则。 例如，当排序根据美国规则并 在 en_US 区域时，重音字符’ ï’（带分音符的小写字母 i ） 在一些地方应该放在’Z ‘的寿面。 然而，根据法国规则排序 并在 fr_FR 区域的语言环境中时，它应该放在’i'和’J'之间。 而一些 其他语言规则可以选择忽略所有的口音，使’ ï’ 和’i'可以任意混合。

第三位，但并非最不重要的，我们也许需要区分在某些场景下大小写相关的排序，而在其他一些情况下大小写无关的排序。

排序规则结合以上所述：字符集， 语言规则， ，大小写区分。Sphinx 目前提供以下四种 排序规则。

1. `libc_ci`
2. `libc_cs`
3. `utf8_general_ci`
4. `binary`

前两个排序规则依赖于几个标准 C 库（libc）的调用， 因此，可以支持任何在您的系统中安装的语言环境。他们分别提供对大小写无关和大小写敏感的排序规则支持。 默认情况下，会使用 C 语言环境，按位进行逐字节对比。你可以通过使用 collation_libc_locale 指令来制定一个在当前系统中可用的其他语言环境设置来改变它。还可以通过 `locale` 命令来获得当前系统上可用的语言环境列表：

```
$ locale -a
C
en_AG
en_AU.utf8
en_BW.utf8
en_CA.utf8
en_DK.utf8
en_GB.utf8
en_HK.utf8
en_IE.utf8
en_IN
en_NG
en_NZ.utf8
en_PH.utf8
en_SG.utf8
en_US.utf8
en_ZA.utf8
en_ZW.utf8
es_ES
fr_FR
POSIX
ru_RU.utf8
ru_UA.utf8
```

不同系统上的语言环境列表可能有所不同。请参考操作系统的文档，以安装更多所需的语言环境。

`utf8_general_ci` 与 `binary` 语言环境是 Sphinx 内建支持的。第一个是 UTF-8 数据的通用字符串校对类型（没有针对任何语言进行定制）。它与 MySQL 的 `utf8_general_ci` 排序规则类似。第二个是简单的字节对比。

字符集校对可以在每次会话过程中通过 SphinxQL 的 `SET collation_connection` 语句来覆盖全局设置。所有后续的 SphinxQL 查询都将使用该字符集校对。SphinxAPI 和 SphinxSE 查询都将使用由 collation_server 配置指令专门指定的服务器默认字符集校对。当前，Sphinx 的默认字符集校对为 `libc_ci`。

排序规则会影响所有的字符串属性比较操作，包括在 ORDER BY 和 GROUp By 中的使用，根据所选择的排序规则将返回不同的排序或者分组结果。

5.13. 用户自定义函数（UDF）

从版本 2.0.1-beta 开始，Sphinx 支持用户自定义函数，或者简称为 UDF。 他们可以被 `searchd` 动态的加载或者卸载，而不用重启驻守进程， 并可以在搜索时用在表达式中。UDF 的功能如下所述：

- 函数可以接受整数 (32-bit 与 64-bit), 浮点数, 字符串, 或者 MVA 参数.
- 函数可以返回整数或者浮点数值.
- 函数可以检查参数数量、类型和名称，以及引发错误.
- 目前仅支持简单的函数（即非聚合的）.

用户自定义函数需要你的系统支持动态库调用（也就是共享对象）。大部分现代操作系统都支持，包括 Linux、Windows、MacOS、Solaris、BSD 以及其他的。 （内部测试是在 Linux 和 Windows 上进行的。）UDF 库需要放置在有 plugin_dir 指令专门指定的目录中，并且服务需要配置为 `workers = threads` 模式。库文件不能使用相对路径。 一旦库文件成功编译构建并拷贝到正确的位置，就可以使用 CREATE FUNCTION and DROP FUNCTION 语句来分别动态的安装或者反安装这些函数。 单个库可以包含多个函数。每个库会在第一次安装其中的某个函数时加载，并在反安装库中的所有函数后被卸载。

库函数将实现可被 SQL 语句调用的 UDF，他们需要遵循 C 调用规则，以及简单的命名约定。 Sphinx 源代码发行包中包含了一个简单的示例文件， src/udfexample.c, 其中定义了一些简单的函数来演示如何使用证书、字符串以及 MVA 参数；你可以使用其中的某一个为样板来创建你的新函数。 它包含了 UDF 接口的头文件 src/sphinxudf.h, 其中定义了所需的类型和结构。 `sphinxudf.h` 头是独立的，也就是说不需要 Sphinx 源代码的任何其他部分就可以编译。

你想要用在 SELECT 语句中的任何一个函数，都需要至少两个相应的 C/C++函数：初始化调用和函数本身的调用。 你也可以根据需要定义反初始化调用以便函数需要在查询后进行清理。 SQL 中的函数名是大小写无辜啊的，但是 C 函数不是，他们需要全部为小写。 函数名称错误将会阻止 UDF 调入。 你还需要特别注意编译时的调用约定，包括参数顺序和类型，以及主调用函数的返回类型。 任何疏忽都可能导致服务崩溃，最好的情况也许是返回了不可预期的结果。 最后但并非最不重要的，所有的函数都必须是线程安全的。

假设因为因为测试的原因，你在 SphinxQL 中的 UDF 起名为 `MYFUNC`。初始化、主调用与卸载函数需要按照以下方式命名并使用对应的参数：

```
/// 初始化
/// 查询初始化时调用 called once during query initialization
/// returns 0 on success
/// returns non-zero and fills error_message buffer on failure
int myfunc_init ( SPH_UDF_INIT * init, SPH_UDF_ARGS * args,
char * error_message );

/// 主调用
/// 返回计算的结果
/// 给 error_flag 写入非零值表示出现错误
RETURN_TYPE myfunc ( SPH_UDF_INIT * init, SPH_UDF_ARGS * args,
char * error_flag );

/// 可选的卸载函数
/// 调用一次清理一次以完成查询处理
void myfunc_deinit ( SPH_UDF_INIT * init );
```

上面提到的两个结构体， `SPH_UDF_INIT` 和 `SPH_UDF_ARGS` 在 `src/sphinxudf.h` 街头头文件中定义并说明。 主调用的 `RETURN_TYPE` 可以是以下之一：

- `int` 表示函数返回值为 INT.
- `sphinx_int64_t` 表示函数返回值为 BIGINT.
- `float` 表示函数返回值为 FLOAT.

他们的调用顺序如下. `myfunc_init()` 在查询初始化时调用一次。它可以返回非零值表示失败；在此情况下查询将不会被执行，并可以通过 `error_message` 缓冲区来返回错误信息。然后，`myfunc()` 会在每行处理中调用，`myfunc_deinit()` 则在查询结束时调用。`myfunc()` 可以通过写入非零值到 `error_flag` 来表示错误，此时它将不再被序列中的行调用，而是用默认值 0 来取代。Sphinx 可能会也可能不会选择提前终止此类查询，目前不保证一定会如何选择。

第 6 章 命令行工具参考

目录

就像其他地方已经提到的，Sphinx 不是个名叫“sphinx”的单独可执行程序，而是由四个独立的程序共同组成的。本节介绍这些工具和他们的用法。

6.1. indexer：索引命令参考

indexer 是 Sphinx 的两个关键工具之一。不管是从命令行直接调用，还是作为一个较大的脚本的一部分使用，indexer 都只负责一件事情——收集要被检索的数据。.

indexer 的调用语法基本上是这样：

```
indexer [OPTIONS] [indexname1 [indexname2 [...]]]
```

用户可以在 sphinx.conf 中设置好可能有哪些索引（index）（这些索引可以在晚些时候别搜索），因此在调用 indexer 的时候，最简单的情况下，只需要告诉它你要建立哪个（或者哪些）索引就行了。

假设 sphinx.conf 包含了两个索引的具体设置， mybigindex 和 mysmallindex，，你可以这么调用：

```
$ indexer mybigindex
$ indexer mysmallindex mybigindex
```

在配置文件 sphinx.conf 里面，用户可以为他们的数据指定一个或多个索引。然后调用 indexer 来对其中一个特定的索引进行重新编制索引操作，或者是重新编制所有索引——不限于某一个或同时全部，用户总是可以指定现有索引的一个组合。

indexer 的大部分选项都可以在配置文件中给出，然而有一部分选项还需要在命令行上指定，这些选项影响编制索引这一操作是如何进行的。这些选项列举如下：

- --config <file> (简写为-c <file>) 使 indexer 将指定的文件 file 作为配置文件。 通常，indexer 是会在安装目录（例如 e.g. /usr/local/sphinx/etc/sphinx.conf ， 如果 sphinx 被安装在 /usr/local/sphinx）中寻找 sphinx.conf，若找不到，则继续在用户在 shell 中调用 indexer 时所在的目录中寻找。 这个选项一般在共享 sphinx 安装的情况下使用，比如二进制文件安装在/usr/local/sphinx，而不同用户都有权定制自己的 sphinx 设置。或者在同一个服务器上运行多个实例的情况下使用。在上述两中情况中，用户可以创建自己的 sphinx.conf 文件，然后把它做为参数传给 indexer。例如：

```
$ indexer --config /home/myuser/sphinx.conf myindex
```

-
- --all 使 indexer 对 sphinx.conf 文件中列出的所有索引进行重新编制索引，这样就不比一次列出每个索引的名字了。这个选项在配置文件较小的情况下，或者在类似基于 cron 的维护工作中很有用。在上述情况中，整个索引集每天或每周或别的什么合适的时间间隔中就重新建立一次。用法示例：

```
$ indexer --config /home/myuser/sphinx.conf --all
```

-
- --rotate 用于轮换索引。对新的文档建立索引时几乎肯定都确保搜索服务仍然可用，除非你有信心在搜索服务停止同时不给你的用户带来困扰。--rotate 建立一个额外的索引，并列于原有索引（与原有索引在相同目录，简单地在原有索引文件名基础上加一个.new 后缀）。一旦这个额外的索引建立完成，indexer 给 searchd 发一个 SIGHUP 信号做为通知。searchd 会尝试将索引重新命名（给原有索引加上.old 后缀，而把带有.new 后缀的新索引改为原名，以达替换之目的），继而用新的文件重启服务。依 seamless_rotate 选项设定之不同，在新索引可用之前可能有一点小的延迟。用法示例：

```
$ indexer --rotate --all
```

-
- --quiet 使 indexer 不输出除错误（error）外的任何东西。这个选项通常用在 cron 定时任务的情境下或者脚本中，这些情况下大部分输出是无关紧要或完全没用的，除非是发生了某些种类的错误。用法示例：

```
$ indexer --rotate --all --quiet
```

-

- --noprogress 不随时显示进度信息，而是仅在索引结束时报告最终的状态细节（例如为哪些文档建立了索引，建立索引的速度等）。当脚本没有运行在一个控制台（console，或“tty”）时，这个选项是默认的。用法示例：

```
$ indexer --rotate --all --noprogress
```

-
  - --buildstops <outputfile.text> <N> 像建立索引一样扫描索引对应的数据源，产生一个最终会被加入索引的词项的列表。换种说法，产生一个用这个索引可以检索的词项的列表。注意，这个选项使 indexer 并不真正更新指定的索引，而只是“假装”建在立索引似地处理一遍数据，包括运行 sql_query_pre 或者 sql_query_post 选项指定的查询。outputfile.txt 文件最终会包含一个词表，每行一个词，按词频排序，高频在前。参数 N 指定了列表中最多可出现的词项数目，如果 N 比索引中全部词项的数目还大，则返回的词项数就是全部词项数。客户端应用程序利用这种字典式的词表来提供“您是要搜索。。。吗？(Did you mean…)”的功能，通常这个选项与下面要讲的--buildfreqs 选项一同使用。示例：

```
$ indexer myindex --buildstops word_freq.txt 1000
```

-

这条命令在当前目录产生一个 word_freq.txt 文件，内含 myindex 这个索引中最常用的 1000 个词，且最常用的排在最前面。注意，当指定了多个索引名或使用了--all 选项（相当于列出配置文件中的所有索引名）时，这个选项对其中的最后一个索引起作用。

-
- --buildfreqs 与 --buildstops 一同使用 (如果没有指定 --buildstops 则--buildfreqs 也被忽略). 它给 --buildstops 产 生的词表的每项增加一个计数信息，即该词在索引中共出现了多少次，这在建立停用词（stop words，出现特别普遍的词）表时可能有用。在开发“您是要搜索。。。吗？(Did you mean…)”的功能时这个选项也能帮上忙，因为有了它你就能知道一个词比另一个相近的词出现得更频繁的程度。示例：

```
$ indexer myindex --buildstops word_freq.txt 1000 --buildfreqs
```

-

这个命令将产生一个类似于上一条命令的 word_freq.txt ，但不同在于，每个词的后面都会附加一个数字，指明在指定的索引中这个词出现了多少次。

-
- --merge <dst-index> <src-index> 用于在物理上将多个索引合并，比方说你在使用“主索引＋增量索引”模式，主索引很少改变，但增量索引很频繁地重建，而--merge 选项允许将这两个索引合而为一。操作是从右向左进行的，即先考察 src-index 的内容，然后在物理上将之与 dst-index 合并，最后结果留在 dst-index 里。用伪代码说就是 dst-index += src-index。示例：

```
$ indexer --merge main delta --rotate
```

-

上例中 main 是主索引，很少更动，delta 是增量索引，频繁更新。上述命令调用 indexer 将 delta 的内容合并到 main 里面并且对索引进行轮换。

-
- --merge-dst-range <attr> <min> <max> 在合并索引的时候运行范围过滤。具体地说，向目标索引 (是 --merge 的一个参数，如果没有指定 --merge， 则--merge-dst-range 也被忽略)合并时，indexer 会对将要合并进去的文档做一次过滤，只有通过过滤才能最终出现在目标索引中。举一个实用的例子，假设某个索引有一个“已删除（deleted）”属性，0 代表“尚未删除”。这样一个索引可以用如下命令进行合并：

```
$ indexer --merge main delta --merge-dst-range deleted 0 0
```

-

这样标记为已删除的文档（值为 1）就不会出现在新生成的目标索引中了。这个选项可以在命令行上指定多次，以便指定多个相继的过滤，这样一个文档要想合并到最终的目标索引中去，就必须依次通过全部这些过滤。

-

- --dump-rows <FILE> 用于从 SQL 数据源导出获取的数据到指定的文件，每行数据使用 MySQL 兼容的格式。导出的数据用于完整的再现 indexer 收到的数据并帮助重现索引时的问题。
- --verbose 确保每条导致索引出现问题的行都会报告出来 (重复, 为零, 或者没有文档编号 ID; 或者外部文件字段的 Io 问题; 等等) . 默认情况下，该选项是关闭的，作为替代的是报告问题摘要。
- --sighup-each 有助于重建多个较大的索引并在每个索引轮换完成后马上生效可被 searchd 使用. 使用 --sighup-each, indexer 会在每个索引成功完成各项工作后给 searchd 发送 SIGHUP 信号。 (默认做法是在全部索引建立后发送 SIGHUP 信号)

- `--print-queries` prints out SQL queries that `indexer` sends to the database, along with SQL connection and disconnection events. That is useful to diagnose and fix problems with SQL sources.

6.2. `searchd`：搜索服务端命令参考

`searchd` 也是 sphinx 的两个关键工具之一。 `searchd` 是系统实际上处理搜索的组件，运行时它表现得就像一种服务，他与客户端应用程序调用的五花八门的 API 通讯，负责接受查询、处理查询和返回数据集。

不同于 `indexer`, `searchd` 并不是设计用来在命令行或者一般的脚本中调用的， 相反，它或者做为一个守护程序（daemon）被 init.d 调用（在 Unix/Linux 类系统上），或者做为一种服务（在 Windows 类系统上），因此并不是所有的命令行选项都总是有效，这与构建时的选项有关。

使用 `searchd` 就像这么简单：

```
$ searchd [OPTIONS]
```

不管 `searchd` 是如何构建的，下列选项总是可用：

- `--help` (可以简写为`-h`) 列出可以在你当前的 `searchd` 构建上调用的参数。.
- `--config <file>` (可简写为`-c <file>`) 使 `searchd` 使用指定的配置文件，与上述 `indexer` 的`--config` 开关相同。
  - `--stop` 用于异步停掉 `searchd`，使用 `sphinx.conf` 中所指定的 PID 文件，因此您可能还需要用`--config` 选项来确认 `searchd` 使用哪个配置文件。值得注意的是，调用 `--stop` 会确保用 UpdateAttributes() 对索引进行的更动会反应到实际的索引文件中去。示例：

```
$ searchd --config /home/myuser/sphinx.conf --stop
```

  -
    - `--stopwait` 用于同步停止 `searchd`.`--stop` 本质上是通知运行实例退出 (通过发送 SIGTERM 信号) 然后立刻返回.`--stopwait` 将会一直等待直到 `searchd` 实例确实的完成了关闭(例如，保存所有在内存中的属性改变) 并退出. 示例：

```
$ searchd --config /home/myuser/sphinx.conf --stopwait
```

    -

可能的退出代码如下：

    -

      - 0 表示成功;
      - 1 表示连接到运行的 searchd 守护进程失败。;
      - 2 表示守护进程在关闭时报告了错误;
      - 3 表示守护进程关闭时崩溃.

  - `--status` 用来查询运行中的 `searchd` 实例的状态，，使用指定的（也可以不指定，使用默认）配置文件中描述的连接参数。它通过配置好的第一个 UNIX 套接字或 TCP 端口与运行中的实例连接。一旦连接成功，它就查询一系列状态和性能计数器的值并把这些数据打印出来。在应用程序中，可以用 Status() API 调用来访问相同的这些计数器。示例：

```
$ searchd --status
$ searchd --config /home/myuser/sphinx.conf --status
```

  -
  - `--pidfile` 用来显式指定一个 PID 文件。PID 文件存储着关于 `searchd` 的进程信息，这些信息用于进程间通讯（例如 `indexer` 需要知道这个 PID 以便在轮换索引的时候与 `searchd` 进行通讯）`searchd` 在正常模式运行时会使用一个 PID（即不是使用–console 选项启动的），但有可能存在 `searchd` 在控制台（`--console`）模式运行，而同时正在索引正在进行更新和轮换操作的情况，此时就需要一个 PID 文件。

```
$ searchd --config /home/myuser/sphinx.conf --pidfile /home/myuser/sphinx.pid
```

  -
  - `--console` is used to force `searchd` 用来强制 `searchd` 以控制台模式启动；典型情况下 `searchd` 像一个传统的服务器应用程序那样运行，它把信息输出到（`sphinx.conf` 配 置文件中指定的）日志文件中。但有些时候需要调试配置文件或者守护程序本身的问题，或者诊断一些很难跟踪的问题，这时强制它把信息直接输出到调用他的控制 台或者命令行上会使调试工作容易些。同时，以控制台模式运行还意味着进程不会 fork（因此搜索操作都是串行执行的），也不会写日志文件。（要特别注意，`searchd` 并不是被主要设计用来在控制台模式运行的）。可以这样调用 `searchd`：

```
$ searchd --config /home/myuser/sphinx.conf --console
```

- --logdebug 让额外的调试信息输出到守护进程日志中。使用的较少，用于帮助调试难以重现的请求。
- --iostats 当使用日志时（必须在 sphinx.conf 中启用 query_log 选项）启用--iostats 会对每条查询输出关于查询过程中发生的输入输出操作的详细信息，会带来轻微的性能代价，并且显然会导致更大的日志文件。更多细节请参考 query log format 一节。可以这样启动 searchd：

```
$ searchd --config /home/myuser/sphinx.conf --iostats
```

- --cpustats 使实际 CPU 时间报告（不光是实际度量时间（wall time））出现在查询日志文件（每条查询输出一次）和状态报告（累加之后）中。这个选项依赖 clock_gettime()系统调用，因此可能在某些系统上不可用。可以这样启动 searchd：

```
$ searchd --config /home/myuser/sphinx.conf --cpustats
```

- --port portnumber (可简写为-p) 指定 searchd 监听的端口，通常用于调试。这个选项的默认值是 9312，但有时用户需要它运行在其他端口上。在这个命令行选项中指定端口比配置文件中做的任何设置优先级都高。有效的端口范围是 0 到 65535，但要使用低于 1024 的端口号可能需要权限较高的账户。使用示例：

```
$ searchd --port 9313
```

- --listen (address ":" port | port | path) [":" protocol] (or -l for short) Works as --port, but allow you to specify not only the port, but full path, as IP address and port, or Unix-domain socket path, that searchd will listen on. Otherwords, you can specify either an IP address (or hostname) and port number, or just a port number, or Unix socket path. If you specify port number but not the address, searchd will listen on all network interfaces. Unix path is identified by a leading slash. As the last param you can also specify a protocol handler (listener) to be used for connections on this socket. Supported protocol values are ‘sphinx’ (Sphinx 0.9.x API protocol) and ‘mysql41′ (MySQL protocol used since 4.1 upto at least 5.1).
- --index <index> (或者缩写为 -i <index>) 强制 searchd 只提供针对指定索引的搜索服务。跟上面的--port 相同，这主要是用于调试，如果是长期使用，则应该写在配置文件中。使用示例：

```
$ searchd --index myindex
```

- --strip-path 用于去除索引中所有引用到的文件名中的路径名 (stopwords, wordforms, exceptions, 等等). 这有助于继续使用在其他可能路径布局不同的机器上建立的索引。

searchd 在 Windows 平台上有一些特有的选项，与它做为 windows 服务所产生的额外处理有关，这些选项只存在于 Windows 二进制版本。

注意，在 Windows 上 searchd 默认以--console 模式运行，除非用户将它安装成一个服务。

- --install installs searchd 将 searchd 安装成一个微软管理控制台（Microsoft Management Console， 控制面板 / 管理工具 / 服务）中的服务。如果一条命令指定了--install，那么同时使用的其他所有选项，都会被保存下来，服务安装好后，每次启动都会调用这些命令。例如，调用 searchd 时，我们很可能希望用--config 指定要使用的配置文件，那么在使用--install 的同时也要加入这个选项。一旦调用了这个选项，用户就可以在控制面板中的管理控制台中对 searchd 进行启动、停止等操作，因此一切可以开始、停止和重启服务的方法对 searchd 也都有效。示例：

```
C:\WINDOWS\system32> C:\Sphinx\bin\searchd.exe --install
--config C:\Sphinx\sphinx.conf
```

如果每次启动 searchd 你都希望得到 I/O stat 信息，那就应该把这个选项也用在调用--install 的命令行里：

```
C:\WINDOWS\system32> C:\Sphinx\bin\searchd.exe --install
--config C:\Sphinx\sphinx.conf --iostats
```

- --delete 在微软管理控制台（Microsoft Management Console）和其他服务注册的地方删除 searchd，当然之前要已经通过--install 安装过 searchd 服务。注意，这个选项既不删除软件本身，也不删除任何索引文件。调用这个选项之后只是使软件提供的服务不能从 windows 的服务系统中调用，也不能在机器重启后自动启动了。如果调用时 searchd 正在做为服务运行中，那么现有的

示例并不会被结束（一直会运行到机器重启或调用--stop）。如果服务安装时（用--servicename）指定了自定义的名字，那在调用此选项卸载服务时里也需要用--servicename 指定相同的名字。示例：

```
C:\WINDOWS\system32> C:\Sphinx\bin\searchd.exe --delete
```

- --servicename <name> 在安装或卸载服务时指定服务的名字，这个名字会出现在管理控制台中。有一个默认的名字 searchd，但若安装服务的系统可能有多个管理员登录，或同时运行多个 searchd 实例，那么起一个描述性强的名字将是个好好主意。注意，只有在与--install 或者--delete 同时使用的时候--servicename 才有效，否则这个选项什么都不做。示例：

```
C:\WINDOWS\system32> C:\Sphinx\bin\searchd.exe --install
--config C:\Sphinx\sphinx.conf --servicename SphinxSearch
```

- --ntservice 在 Windows 平台，管理控制台将 searchd 做为服务调用时将这个选项传递给它。通常没有必要直接调用这个开关，它是为 Windows 系统准备的，当服务启动时，系统把这个参数传递给 searchd。然而理论上，你也可以用这个开关从命令行将 searchd 启动成普通服务模式（与--console 代表的控制台模式相对）

最后但并非最不重要的，类似其他的守护进程（daemon），searchd 支持多种信号。

SIGTERM
进行一次平滑的重启。新的请求不会被接受；但是已经开始的请求不会被强行中断。
SIGHUP
启动索引轮换。取决于 seamless_rotate 的设置，新的请求可能会在短期内陷入停顿；客户端将接收到临时错误。
SIGUSR1
强制重新打开 searchd 日志和查询日志，使得日志轮换可以进行。

6.3. search：命令行搜索命令参考

search 是 Sphinx 中的一个辅助工具。searchd 负责服务器类环境中的搜索，而 search 专注于在命令行上对索引进行快速测试，而不需要构建一个复杂的架构来处理到服务器端的连接和处理服务器返回的响应。

注意：search 并不是设计用来做为客户端应用程序的一部分。我们强烈建议用户不要针对 search 编写接口，相反，应该针对 searchd。Sphinx 提供的任何客户端 API 也都不支持这种用法。（任何时候 search 总是每次都重新调入索引，而 searchd 会把索引缓冲在内存中以利性能）。

澄清了这些我们就可以继续了。很多通过 API 构造的查询也可以用 search 来做到，然而对于非常复杂的查询，可能还是用个小脚本和对应的 API 调用来实现比较简单。除此之外，可能有些新的特性先在 searchd 系统中实现了而尚未引入到 search 中。

search 的调用语法如下：

```
search [OPTIONS] word1 [word2 [word3 [...]]]
```

调用 search 并不要求 searchd 正在运行，只需运行 search 的账户对配置文件和索引文件及其所在路径有读权限即可。

默认行为是对在配置文件中设置的全部索引的全部字段搜索 word1（AND word2 AND word3….）。如果用 API 调用来构建这个搜索，那相当于向 SetMatchMode 传递参数 SPH_MATCH_ALL，然后在调用 Query 的时候指定要查询的索引是*。

search 有很多选项。首先是通用的选项：

- --config <file> (可简写为-c <file>) 使 search 使用指定的配置文件，这与上述 indexer 的对应选项相同。
- --index <index> (可简写为-i <index>) 使 search 仅搜索指定的索引。通常它会尝试搜索 sphinx.conf 中列出的全部物理索引，不包括分布式索引。
- --stdin 使 search 接受标准输入（STDIN）上传入的查询，而不是命令行上给出的查询。有时你要用脚本通过管道给 search 传入查询，这正是这个选项的用武之地。

设置匹配方式的选项：

- --any (可简写为-a) 更改匹配模式，匹配指定的任意一个词（word1 OR word2 OR word3），这对应 API 调用中向 SetMatchMode 传递参数 SPH_MATCH_ANY。
- --phrase (可简写为-p) 更改匹配模式，将指定的全部词做为一个词组（不包括标点符号）构成查询，这对应 API 调用中向 SetMatchMode 传递参数 SPH_MATCH_PHRASE。
- --boolean (可简写为-b) 将匹配模式设为 Boolean matching。注意如果在命令行上使用布尔语法，可能需要对某些符号（用反斜线“\”）加以转义，以避免外壳程序（shell）或命令行处理器对这些符号做特殊理解，例如，在 Unix/Linux 系统上必须转义“&”以防止 search 被

fork 成一个后台进程，尽管这个问题也可以像下文一样通过使用--stdin 选项来解决。这个选项对应 API 调用中向 SetMatchMode 传递参数 SPH_MATCH_BOOLEAN。

- --ext (可简写为-e) 将匹配模式设为 Extended matching。这对应与 API 调用中向 SetMatchMode 传递参数 SPH_MATCH_EXTENDED。要注意的是因为已经有了更好的扩展匹配模式版本 2，所以并不鼓励使用这个选项，见下一条说明。
- --ext2 (可简写为-e2) 将匹配模式设为 扩展比配，版本 2。这个选项对应在 API 调用中向 SetMatchMode 传递参数 SPH_MATCH_EXTENDED2。要注意这个选项相比老的扩展匹配模式更有效也提供更多的特性，因此推荐使用这个新版的选项。
- --filter <attr> <v> (可简写为-f <attr> <v>) 对结果进行过滤，只有指定的属性 attr 匹配指定的值 v 时才能通过过滤。例如--filter deleted 0 只匹配那些有 deleted 属性，并且其值是 0 的文档。也可以在命令行上多次给出--filter 以便指定多重过滤，但是如果重复定义针对同一个属性的过滤器，那么第二次指定的过滤条件会覆盖第一次的。

用于处理搜索结果的选项：

- --limit <count> (可简写为-l count) 限制返回的最多匹配结果数。如果指定了分组（group）选项，则表示的是返回的最多匹配组数。默认值是 20 个结果（与 API 相同）
- --offset <count> (可简写为-o <count>) 从第 count 个结果开始返回，用于给搜索结果分页。如果想要每页 20 个结果，那么第二页就从偏移量 20 开始，第三页从偏移量 40 开始，以此类推。
- --group <attr> (可简写为-g <attr>) 搜索结果按照指定的属性 attr 进行分组。类似 SQL 中的 GROUP BY 子句，这会将 attr 属性值一致的结果结合在一起，返回的结果集中的每条都是一组中最好的那条结果。如果没有特别指定，那“最好”指的是相关度最大的。
- --groupsort <expr> (可简写为-gs <expr>) 使搜索结果根据-group 分组后，再用表达式<expr>的值决定分组的顺序。注意，这个选项指定的不是各组内部哪条结果是最好的，而是分组本身返回的顺序。
- --sortby <clause> (可简写为-s <clause>) 指定结果按照<clause>中指定的顺序排序。这使用户可以控制搜索结果展现时的顺序，即根据不同的列排序。例如，--sortby "@weight DESC entrytime DESC"的意思是将结果首先按权值（相关度）排序，如果有两条或以上结果的相关度相同，则他们的顺序由时间值 entrytime 决定，时间最近（值最大）的排在前面。通常需要将这些项目放在引号里(--sortby "@weight DESC")或者用逗号隔开(--sortby @weight,DESC)，以避免它们被分开处理。另外，与通常的排序模式相同，如果指定了--group(分组)，这个选项就影响分组内部的结果如何排序。
- --sortexpr expr (可简写为-S expr) 搜索结果展现的顺序由指定的算术表达式 expr 决定。例如： --sortexpr "@weight + (user_karma + ln(pageviews))*0.1"（再次提示，要用引号来避免 shell 对星号*做特殊处理）。扩展排序模式在排序模式 一章下的 SPH_SORT_EXTENDED 条目下具体讨论。
- --sort=date 搜索结果按日期降序（日期最新的在前）排列。要求索引中有一个属性被指定为时间戳。
- --rsort=date 特别指定结果升序排列（日期中较老的在前）。要求索引中有一个属性被指定为时间戳。
- --sort=ts 搜索结果按时间戳分成组。先返回时间戳在最近一小时内的这组结果，在组内部按相关度排序。其后返回时间戳为最近一天之内的结果，也按相关度排序。再之后是最近一周的，最后是最近一个月的。在排序模式 一章的 SPH_SORT_TIME_SEGMENTS 条目下对此有更详细的讨论。

其他选项：

- --noinfo (可简写为-q) 使 search 不在 SQL 数据库中查询文档信息（Document Info）。具体地说，为了调试 search 和 MySQL 共同使用时出现的问题，你可以在使用这个选项的同时提供一个根据文档 ID 搜索整个文章全文的查询。细节可参考 sql_query_info 指令。

6.4. spelldump：拼写信息导出命令参考

spelldump 是 Sphinx 的一个辅助程序。

用于从 ispell 或者 MySpell 格式的字典文件中可用来辅助建立词形列表（*wordforms*）的内容——词的全部可能变化都预先构造好。

一般用法如下：

```
spelldump [options] <dictionary> <affix> [result] [locale-name]
```

两个主要参数是词典的主文件（[language-prefix].dict）和词缀文件（[language-prefix].aff）；通常这两种文件被命名为[语言简写].dict 和[语言简写].aff，大多数常见的 Linux 发行版中都有这些文件，网上也到处找得到。

[result] 指定的是字典数据的输出位置，而[locale-name]指定了具体使用的区域设置(locale)。

还有一个-c [file]选项，用来指定一个包含大小写转换方面细节的文件。

用法示例：

```
spelldump en.dict en.aff
spelldump ru.dict ru.aff ru.txt ru_RU.CP1251
spelldump ru.dict ru.aff ru.txt .1251
```

结果文件会包含字典中包含的全部词，字典序排列，wordforms 文件格式。可以根据具体的使用环境定制这些文件。结果文件的一个例子：

```
zone > zone
zoned > zoned
zoning > zoning
```

6.5. indextool：索引信息导出命令参考

`indextool` 是版本 0.9.9-rc2 中引入的辅助工具。用于输出关于物理索引的多种调试信息。（未来还计划加入索引验证等功能，因此起名较 indextool 而不是 indexdump）。 基本用法如下：

```
indextool <command> [options]
```

唯一一个所有命令都有的选项是`--config`，用于指定配置文件：

- `--config <file>` (可简写为`-c <file>`) 覆盖默认的配置文件名。

其他可用的命令如下：

- `--dumpheader FILENAME.sph` 在不涉及任何其他索引文件甚至配置文件的前提下，快速输出索引头文件的内容，包括索引的全部设置，尤其是完整的属性列表、字段列表。在版本 0.9.9-rc2 之前，这个命令是由 search 工具提供的。 CLI search utility.
- `--dumpconfig FILENAME.sph` 从给定的索引头文件导出 `sphinx.conf` 文件格式兼容（基本上）的索引配置信息。版本 2.0.1-beta 新增。
- `--dumpheader INDEXNAME` 输出给定索引名的索引头内容，索引头文件的路径是在配置文件中查得的。
- `--dumpdocids INDEXNAME` 输出给定索引名涉及的文档 ID。数据是从属性文件(.spa)中抽取的，因此要求 doc_info=extern 正常工作。
- `--dumphitlist INDEXNAME KEYWORD` 输出指定关键字 KEYWORD 在执行索引中的的全部出现。
- `--dumphitlist INDEXNAME --wordid ID` 输出指定关键字 KEYWORD 在执行索引中的的全部出现，但是关键字使用特定的内部数字编号。
- `--htmlstrip INDEXNAME` 使用指定的索引中的 HTML 剥离器设置来过滤标准输入并打印过滤结果到标准输出。需要注意的是配置信息从 sphinx.conf 中获得，而不是从索引头文件中获得。
- `--check INDEXNAME` 检查可能由于 `indexer` 的 bug 或者硬件问题导致的索引数据一致性错误。
- `--strip-path` 用于去除索引中所有引用到的文件名中的路径名 (stopwords, wordforms, exceptions, 等等). 这有助于继续使用在其他可能路径布局不同的机器上建立的索引。

第 7 章 SphinxQL 指南

目录

SphinxQL 是有我们自己特色的 SQL 子集，通过使用标准的 SQL 语句并加上 Sphinx 特定的扩展，将搜索驻守进程的功能对外提 供。所有通过 SphinxAPI 提供的功能都可以通过 SphinxQL 访问，不过反过来则不行，例如对 RT 索引的写操作只能通过 SPhinxQL。本章的文档列出了所 支持的 SphinxQL 语句的语法。

7.1. SELECT（搜索查询）语法

```
SELECT
select_expr [, select_expr ...]
FROM index [, index2 ...]
[WHERE where_condition]
[GROUP BY {col_name | expr_alias}]
[ORDER BY {col_name | expr_alias} {ASC | DESC} [, ...]]
[WITHIN GROUP ORDER BY {col_name | expr_alias} {ASC | DESC}]
[LIMIT offset, row_count]
[OPTION opt_name = opt_value [, ...]]
```

SELECT 语句在版本 0.9.9-rc2 引入。 其语法基于通常的 SQL，但是加入了一些 Sphinx 特定的扩展，并省略了一些功能（例如（目前）不支持多表联合(JOIN)）。特别的，

- Column list clause. Column names, arbitrary expressions, and star ('*') are all allowed (ie. `SELECT @id, group_id*123+456 AS expr1 FROM test1` will work). Unlike in regular SQL, all computed expressions must be aliased with a valid identifier. Starting with version 2.0.1-beta, `AS` is optional. Special names such as @id and @weight should currently be used with leading at-sign. This at-sign requirement will be lifted in the future.
- FROM clause. FROM clause should contain the list of indexes to search through. Unlike in regular SQL, comma means enumeration of full-text indexes as in Query() API call rather than JOIN.
- WHERE clause. This clause will map both to fulltext query and filters. Comparison operators (=, !=, <, >, <=, >=), IN, AND, NOT, and BETWEEN are all supported and map directly to filters. OR is not supported yet but will be in the future. MATCH('query') is supported and maps to fulltext query. Query will be interpreted according to full-text query language rules. There must be at most one MATCH() in the clause. Starting with version 2.0.1-beta, `{col_name | expr_alias} [NOT] IN @uservar` condition syntax is supported. (Refer to 第 7.7 节 "SET（设置服务端变量）语法" for a discussion of global user variables.)
    - GROUP BY clause. Currently only supports grouping by a single column. The column however can be a computed expression:

```
SELECT *, group_id*1000+article_type AS gkey FROM example GROUP BY gkey
```

Aggregate functions (AVG(), MIN(), MAX(), SUM()) in column list clause are supported. Arguments to aggregate functions can be either plain attributes or arbitrary expressions. COUNT(*) is implicitly supported as using GROUP BY will add @count column to result set. Explicit support might be added in the future. COUNT(DISTINCT attr) is supported. Currently there can be at most one COUNT(DISTINCT) per query and an argument needs to be an attribute. Both current restrictions on COUNT(DISTINCT) might be lifted in the future.

```
SELECT *, AVG(price) AS avgprice, COUNT(DISTINCT storeid)
FROM products
WHERE MATCH('ipod')
GROUP BY vendorid
```

Starting with 2.0.1-beta, GROUP BY on a string attribute is supported, with respect for current collation (参见 第 5.12 节 "字符串排序规则").

- WITHIN GROUP ORDER BY clause. This is a Sphinx specific extension that lets you control how the best row within a group will to be selected. The syntax matches that of regular ORDER BY clause:

```
SELECT *, INTERVAL(posted,NOW()-7*86400,NOW()-86400) AS timeseg
FROM example WHERE MATCH('my search query')
GROUP BY siteid
WITHIN GROUP ORDER BY @weight DESC
ORDER BY timeseg DESC, @weight DESC
```

Starting with 2.0.1-beta, WITHIN GROUP ORDER BY on a string attribute is supported, with respect for current collation (参见 第 5.12 节 “字符串排序规则”).

- ORDER BY clause. Unlike in regular SQL, only column names (not expressions) are allowed and explicit ASC and DESC are required. The columns however can be computed expressions:

```
SELECT *, @weight*10+docboost AS skey FROM example ORDER BY skey
```

Starting with 2.0.1-beta, ORDER BY on a string attribute is supported, with respect for current collation (参见 第 5.12 节 “字符串排序规则”).

- LIMIT clause. Both LIMIT N and LIMIT M,N forms are supported. Unlike in regular SQL (but like in Sphinx API), an implicit LIMIT 0,20 is present by default.
- OPTION clause. This is a Sphinx specific extension that lets you control a number of per-query options. The syntax is:

```
OPTION <optionname>=<value> [ , ... ]
```

Supported options and respectively allowed values are:

- ‘ranker’ – any of ‘proximity_bm25′, ‘bm25′, ‘none’, ‘wordcount’, ‘proximity’, ‘matchany’, or ‘fieldmask’
- ‘max_matches’ – integer (per-query max matches value)
- ‘cutoff’ – integer (max found matches threshold)
- ‘max_query_time’ – integer (max search time threshold, msec)
- ‘retry_count’ – integer (distributed retries count)
- ‘retry_delay’ – integer (distributed retry delay, msec)
- ‘field_weights’ – a named integer list (per-field user weights for ranking)
- ‘index_weights’ – a named integer list (per-index user weights for ranking)
- ‘reverse_scan’ – 0 or 1, lets you control the order in which full-scan query processes the rows

Example:

```
SELECT * FROM test WHERE MATCH('@title hello @body  world')
OPTION ranker=bm25, max_matches=3000,
field_weights=(title=10, body=3)
```

7.2. SHOW META（显示查询状态信息）语法

```
SHOW META
```

SHOW META shows additional meta-information about the latest query such as query time and keyword statistics:

```
mysql> SELECT * FROM test1 WHERE MATCH('test|one|two');
+------+--------+----------+------------+
| id   | weight | group_id | date_added |
+------+--------+----------+------------+
|    1 |   3563 |      456 | 1231721236 |
|    2 |   2563 |      123 | 1231721236 |
|    4 |   1480 |        2 | 1231721236 |
+------+--------+----------+------------+
3 rows in set (0.01 sec)

mysql> SHOW META;
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| total         | 3     |
| total_found   | 3     |
```

```
| time          | 0.005 |
| keyword[0]    | test  |
| docs[0]       | 3     |
| hits[0]       | 5     |
| keyword[1]    | one   |
| docs[1]       | 1     |
| hits[1]       | 2     |
| keyword[2]    | two   |
| docs[2]       | 1     |
| hits[2]       | 2     |
+---------------+-------+
12 rows in set (0.00 sec)
```

7.3. SHOW WARNINGS（显示查询警告信息）语法

```
SHOW WARNINGS
```

SHOW WARNINGS statement, introduced in version 0.9.9-rc2, can be used to retrieve the warning produced by the latest query. The error message will be returned along with the query itself:

```
mysql> SELECT * FROM test1 WHERE MATCH('@@title hello') \G
ERROR 1064 (42000): index test1: syntax error, unexpected TOK_FIELDLIMIT
near '@title hello'

mysql> SELECT * FROM test1 WHERE MATCH('@title -hello') \G
ERROR 1064 (42000): index test1: query is non-computable (single NOT operator)

mysql> SELECT * FROM test1 WHERE MATCH('"test doc"/3') \G
*************************** 1. row ***************************
id: 4
weight: 2500
group_id: 2
date_added: 1231721236
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS \G
*************************** 1. row ***************************
Level: warning
Code: 1000
Message: quorum threshold too high (words=2, thresh=3); replacing quorum operator
with AND operator
1 row in set (0.00 sec)
```

7.4. SHOW STATUS（显示服务端状态信息）语法

SHOW STATUS, introduced in version 0.9.9-rc2, displays a number of useful performance counters. IO and CPU counters will only be available if searchd was started with –iostats and –cpustats switches respectively.

```
mysql> SHOW STATUS;
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| uptime             | 216   |
| connections        | 3     |
| maxed_out          | 0     |
| command_search     | 0     |
| command_excerpt    | 0     |
| command_update     | 0     |
| command_keywords   | 0     |
| command_persist    | 0     |
| command_status     | 0     |
```

```
| agent_connect      | 0     |
| agent_retry        | 0     |
| queries            | 10    |
| dist_queries       | 0     |
| query_wall         | 0.075 |
| query_cpu          | OFF   |
| dist_wall          | 0.000 |
| dist_local         | 0.000 |
| dist_wait          | 0.000 |
| query_reads        | OFF   |
| query_readkb       | OFF   |
| query_readtime     | OFF   |
| avg_query_wall     | 0.007 |
| avg_query_cpu      | OFF   |
| avg_dist_wall      | 0.000 |
| avg_dist_local     | 0.000 |
| avg_dist_wait      | 0.000 |
| avg_query_reads    | OFF   |
| avg_query_readkb   | OFF   |
| avg_query_readtime | OFF   |
+--------------------+-------+
29 rows in set (0.00 sec)
```

7.5. INSERT 和 REPLACE（数据插入和替换）语法

```
{INSERT | REPLACE} INTO index [(column, ...)]
VALUES (value, ...)
[, (...)]
```

INSERT statement, introduced in version 1.10-beta, is only supported for RT indexes. It inserts new rows (documents) into an existing index, with the provided column values.

ID column must be present in all cases. Rows with duplicate IDs will **not** be overwritten by INSERT; use REPLACE to do that.

`index` is the name of RT index into which the new row(s) should be inserted. The optional column names list lets you only explicitly specify values for some of the columns present in the index. All the other columns will be filled with their default values (0 for scalar types, empty string for text types).

Expressions are not currently supported in INSERT and values should be explicitly specified.

Multiple rows can be inserted using a single INSERT statement by providing several comma-separated, parens-enclosed lists of rows values.

7.6. DELETE（数据删除）语法

```
DELETE FROM index WHERE {id = value | id IN (val1 [, val2 [, ...]])}
```

DELETE statement, introduced in version 1.10-beta, is only supported for RT indexes. It deletes existing rows (documents) from an existing index based on ID.

`index` is the name of RT index from which the row should be deleted. `value` is the row ID to be deleted. Support for batch `id IN (2,3,5)` syntax was added in version 2.0.1-beta.

Additional types of WHERE conditions (such as conditions on attributes, etc) are planned, but not supported yet as of 1.10-beta.

7.7. SET（设置服务端变量）语法

```
SET [GLOBAL] server_variable_name = value
SET GLOBAL @user_variable_name = (int_val1 [, int_val2, ...])
```

SET statement, introduced in version 1.10-beta, modifies a server variable value. The variable names are case-insensitive. No variable value changes survive server restart. There are the following classes of the variables:

1. per-session server variable (1.10-beta and above)
2. global server variable (2.0.1-beta and above)
3. global user variable (2.0.1-beta and above)

Global user variables are shared between concurrent sessions. Currently, the only supported value type is the list of BIGINTs, and these variables can only be used along with IN() for filtering purpose. The intended usage scenario is uploading huge lists of values to `searchd` (once) and reusing them (many times) later, saving on network overheads. Example:

```
// in session 1
mysql> SET GLOBAL @myfilter=(2,3,5,7,11,13);
Query OK, 0 rows affected (0.00 sec)

// later in session 2
mysql> SELECT * FROM test1 WHERE group_id IN @myfilter;
+------+--------+----------+------------+-----------------+------+
| id   | weight | group_id | date_added | title           | tag  |
+------+--------+----------+------------+-----------------+------+
|    3 |      1 |        2 | 1299338153 | another doc     | 15   |
|    4 |      1 |        2 | 1299338153 | doc number four | 7,40 |
+------+--------+----------+------------+-----------------+------+
2 rows in set (0.02 sec)
```

Per-session and global server variables affect certain server settings in the respective scope. Known per-session server variables are:

AUTOCOMMIT = {0 | 1}
: Whether any data modification statement should be implicitly wrapped by BEGIN and COMMIT. 版本 2.0.1-beta 引入。

COLLATION_CONNECTION = collation_name
: Selects the collation to be used for ORDER BY or GROUP BY on string values in the subsequent queries. Refer to 第 5.12 节 “字符串排序规则” for a list of known collation names. 版本 2.0.1-beta 引入。

CHARACTER_SET_RESULTS = charset_name
: Does nothing; a placeholder to support frameworks, clients, and connectors that attempt to automatically enforce a charset when connecting to a Sphinx server. 版本 2.0.1-beta 引入。

Known global server variables are:

QUERY_LOG_FORMAT = {plain | sphinxql}
: Changes the current log format. 版本 2.0.1-beta 引入。

LOG_LEVEL = {info | debug | debugv | debugvv}
: Changes the current log verboseness level. 版本 2.0.1-beta 引入。

Examples:

```
mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> SET GLOBAL query_log_format=sphinxql;
Query OK, 0 rows affected (0.00 sec)
```

7.8. BEGIN, COMMIT, 以及 ROLLBACK（事务处理）语法

```
START TRANSACTION | BEGIN
COMMIT
ROLLBACK
SET AUTOCOMMIT = {0 | 1}
```

BEGIN, COMMIT, and ROLLBACK statements were 版本 2.0.1-beta 引入。 BEGIN statement (or its START TRANSACTION alias) forcibly commits pending transaction, if any, and begins a new one. COMMIT statement commits the current transaction, making all its changes permanent. ROLLBACK statement rolls back the current transaction, canceling all its changes. SET AUTOCOMMIT controls the autocommit mode in the active session.

AUTOCOMMIT is set to 1 by default, meaning that every statement that perfoms any changes on any index is implicitly wrapped in BEGIN and COMMIT.

Transactions are limited to a single RT index, and also limited in size. They are atomic, consistent, overly isolated, and durable. Overly isolated means that the changes are not only invisible to the concurrent transactions but even to the current session itself.

7.9. CALL SNIPPETS（摘要生成）语法

```
CALL SNIPPETS(data, index, query[, opt_value AS opt_name[, ...]])
```

CALL SNIPPETS statement, introduced in version 1.10-beta, builds a snippet from provided data and query, using specified index settings.

`data` is the source data string to extract a snippet from. `index` is the name of the index from which to take the text processing settings. `query` is the full-text query to build snippets for. Additional options are documented in 第 8.7.1 节 “BuildExcerpts （产生文本摘要和高亮）”. Usage example:

```
CALL SNIPPETS('this is my document text', 'test1', 'hello world',
5 AS around, 200 AS limit)
```

7.10. CALL KEYWORDS（关键词生成）语法

```
CALL KEYWORDS(text, index, [hits])
```

CALL KEYWORDS statement, introduced in version 1.10-beta, splits text into particular keywords. It returns tokenized and normalized forms of the keywords, and, optionally, keyword statistics.

`text` is the text to break down to keywords. `index` is the name of the index from which to take the text processing settings. `hits` is an optional boolean parameter that specifies whether to return document and hit occurrence statistics.

7.11. SHOW TABLES（显示当前提供搜索服务的索引列表）语法

```
SHOW TABLES
```

SHOW TABLES statement, introduced in version 2.0.1-beta, enumerates all currently active indexes along with their types. As of 2.0.1-beta, existing index types are `local`, `distributed`, and `rt`respectively. Example:

```
mysql> SHOW TABLES;
+-------+-------------+
| Index | Type        |
+-------+-------------+
| dist1 | distributed |
| rt    | rt          |
| test1 | local       |
| test2 | local       |
+-------+-------------+
4 rows in set (0.00 sec)
```

7.12. DESCRIBE（显示指定搜索服务索引的字段信息）语法

```
{DESC | DESCRIBE} index
```

DESCRIBE statement, introduced in version 2.0.1-beta, lists index columns and their associated types. Columns are document ID, full-text fields, and attributes. The order matches that in which fields and attributes are expected by INSERT and REPLACE statements. As of 2.0.1-beta, column types are `field`, `integer`, `timestamp`, `ordinal`, `bool`, `float`, `bigint`, `string`, and `mva`. ID column will be typed either `integer` or `bigint` based on whether the binaries were built with 32-bit or 64-bit document ID support. Example:

```
mysql> DESC rt;
+---------+---------+
| Field   | Type    |
+---------+---------+
| id      | integer |
| title   | field   |
| content | field   |
| gid     | integer |
+---------+---------+
4 rows in set (0.00 sec)
```

7.13. CREATE FUNCTION（添加自定义函数）语法

```
CREATE FUNCTION udf_name
RETURNS {INT | BIGINT | FLOAT}
SONAME 'udf_lib_file'
```

CREATE FUNCTION statement, introduced in version 2.0.1-beta, installs a user-defined function (UDF) with the given name and type from the given library file. The library file must reside in a trusted plugin_dir directory. On success, the function is available for use in all subsequent queries that the server receives. Example:

```
mysql> CREATE FUNCTION avgmva RETURNS INT SONAME 'udfexample.dll';
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT *, AVGMVA(tag) AS q from test1;
+------+--------+---------+-----------+
| id   | weight | tag     | q         |
+------+--------+---------+-----------+
|    1 |      1 | 1,3,5,7 | 4.000000  |
|    2 |      1 | 2,4,6   | 4.000000  |
|    3 |      1 | 15      | 15.000000 |
|    4 |      1 | 7,40    | 23.500000 |
+------+--------+---------+-----------+
```

7.14. DROP FUNCTION（删除自定义函数）语法

```
DROP FUNCTION udf_name
```

DROP FUNCTION statement, introduced in version 2.0.1-beta, deinstalls a user-defined function (UDF) with the given name. On success, the function is no longer available for use in subsequent queries. Pending concurrent queries will not be affected and the library unload, if necessary, will be postponed until those queries complete. Example:

```
mysql> DROP FUNCTION avgmva;
Query OK, 0 rows affected (0.00 sec)
```

7.15. SHOW VARIABLES（显示服务器端变量）语法

```
SHOW VARIABLES
```

Added in version 2.0.1-beta, this is currently a placeholder query that does nothing and reports success. That is in order to keep compatibility with frameworks and connectors that automatically execute this statement.

```
mysql> SHOW VARIABLES;
Query OK, 0 rows affected (0.00 sec)
```

7.16. SHOW COLLATION（显示字符集校对）语法

```
SHOW COLLATION
```

Added in version 2.0.1-beta, this is currently a placeholder query that does nothing and reports success. That is in order to keep compatibility with frameworks and connectors that automatically execute this statement.

```
mysql> SHOW COLLATION;
Query OK, 0 rows affected (0.00 sec)
```

7.17. UPDATE（数据更新）语法

```
UPDATE index SET col1 = newval1 [, ...] WHERE ID = docid
```

UPDATE statement was added in version 2.0.1-beta. It can currently update 32-bit integer attributes only. Multiple attributes and values can be specified. Both RT and disk indexes are supported. Updates on other attribute types are also planned.

```
mysql> UPDATE myindex SET enabled=0 WHERE id=123;
Query OK, 1 rows affected (0.00 sec)
```

7.18. 多结果集查询（批量查询）

Starting version 2.0.1-beta, SphinxQL supports multi-statement queries, or batches. Possible inter-statement optimizations described in 第 5.11 节 “批量查询” do apply to SphinxQL just as well. The batched queries should be separated by a semicolon. Your MySQL client library needs to support MySQL multi-query mechanism and multiple result set. For instance, mysqli interface in PHP and DBI/DBD libraries in Perl are known to work.

Here’s a PHP sample showing how to utilize mysqli interface with Sphinx.

```
<?php

$link = mysqli_connect ( "127.0.0.1", "root", "", "", 9306 );
if ( mysqli_connect_errno() )
die ( "connect failed: " . mysqli_connect_error() );

$batch = "SELECT * FROM test1 ORDER BY group_id ASC;";
$batch .= "SELECT * FROM test1 ORDER BY group_id DESC";

if ( !mysqli_multi_query ( $link, $batch ) )
die ( "query failed" );

do
{
// fetch and print result set
if ( $result = mysqli_store_result($link) )
{
while ( $row = mysqli_fetch_row($result) )
printf ( "id=%s\n", $row[0] );
mysqli_free_result($result);
}

// print divider
if ( mysqli_more_results($link) )
printf ( "------\n" );

} while ( mysqli_next_result($link) );
```

Its output with the sample `test1` index included with Sphinx is as follows.

```
$ php test_multi.php
id=1
id=2
id=3
id=4
------
id=3
id=4
id=1
id=2
```

The following statements can currently be used in a batch: SELECT, SHOW WARNINGS, SHOW STATUS, and SHOW META. Arbitrary sequence of these statements are allowed. The results sets returned should match those that would be returned if the batched queries were sent one by one.

7.19. COMMIT（注释）语法

Since version 2.0.1-beta, SphinxQL supports C-style comment syntax. Everything from an opening `/*` sequence to a closing `*/` sequence is ignored. Comments can span multiple lines, can not nest, and should not get logged. MySQL specific `/*! ... */` comments are also currently ignored. (As the comments support was rather added for better compatibility with`mysqldump` produced dumps, rather than improving generaly query interoperability between Sphinx and MySQL.)

```
SELECT /*! SQL_CALC_FOUND_ROWS */ col1 FROM table1 WHERE ...
```

7.20. SphinxQL 保留关键字列表

A complete alphabetical list of keywords that are currently reserved in SphinxQL syntax (and therefore can not be used as identifiers).

```
AND
AS
ASC
AVG
BEGIN
BETWEEN
BY
CALL
COLLATION
COMMIT
COUNT
DELETE
DESC
DESCRIBE
DISTINCT
FALSE
FROM
GLOBAL
GROUP
ID
IN
INSERT
INTO
LIMIT
MATCH
MAX
META
MIN
NOT
NULL
OPTION
OR
ORDER
REPLACE
ROLLBACK
SELECT
SET
SHOW
START
STATUS
SUM
TABLES
TRANSACTION
TRUE
UPDATE
VALUES
VARIABLES
WARNINGS
WEIGHT
```

WHERE
WITHIN

7.21. SphinxQL 升级备注, version 2.0.1-beta

This section only applies to existing applications that use SphinxQL versions prior to 2.0.1-beta.

In previous versions, SphinxQL just wrapped around SphinxAPI and inherited its magic columns and column set quirks. Essentially, SphinxQL queries could return (slightly) different columns and in a (slightly) different order than it was explicitly requested in the query. Namely, `weight` magic column (which is not a real column in any index) was added at all times, and GROUP BY related @count , @group , and `@distinct` magic columns were conditionally added when grouping. Also, the order of columns (attributes) in the result set was actually taken from the index rather than the query. (So if you asked for columns C, B, A in your query but they were in the A, B, C order in the index, they would have been returned in the A, B, C order.)

In version 2.0.1-beta, we fixed that. SphinxQL is now more SQL compliant (and will be further brought in as much compliance with standard SQL syntax as possible). That is not yet a breaking change, because `searchd` now supports compat_sphinxql_magics directive that flips between the old "compatibility" mode and the new "compliance" mode. However, the compatibility mode support is going to be removed in future, so it's strongly advised to update SphinxQL applications and switch to the compliance mode.

The important changes are as follows:

- `@ID` **magic name is deprecated in favor of** `ID`**.** Document ID is considered an attribute.
    - `WEIGHT` **is no longer implicitly returned**, because it is not actually a column (an index attribute), but rather an internal function computed per each row (a match). You have to explicitly ask for it, using the `WEIGHT()` function. (The requirement to alias the result will be lifted in the next release.)

```
SELECT id, WEIGHT() w FROM myindex WHERE MATCH('test')
```

    - **You can now use quoted reserved keywords as aliases.** The quote character is backtick ("`", ASCII code 96 decimal, 60 hex). One particularly useful example would be returning`weight` column like the old mode:

```
SELECT id, WEIGHT() `weight` FROM myindex WHERE MATCH('test')
```

- The column order is now different and should now match the one expliclitly defined in the query. So if you are accessing columns based on their position in the result set rather than the name (for instance, by using `mysql_fetch_row()` rather than `mysql_fetch_assoc()` in PHP), **check and fix the order of columns in your queries.**
    - `SELECT *` return the columns in index order, as it used to, including the ID column. However, `SELECT *` **does not automatically return WEIGHT().** To update such queries in case you access columns by names, simply add it to the query:

```
SELECT *, WEIGHT() `weight` FROM myindex WHERE MATCH('test')
```

Otherwise, i.e., in case you rely on column order, select ID, weight, and then other columns:

```
SELECT id, *, WEIGHT() `weight` FROM myindex WHERE MATCH('test')
```

    - **Magic** @count **and** `@distinct` **attributes are no longer implicitly returned**. You now have to explicitly ask for them when using GROUP BY. (Also note that you currently have to alias them; that requirement will be lifted in the future.)

```
SELECT gid, COUNT(*) q FROM myindex WHERE MATCH('test')
GROUP BY gid ORDER BY q DESC
```

-
-

第 8 章 API 参考

目录

Sphnix 有几种不同编程语言的原生 searchd 客户端 API 的实现。在本文完成之时，我们对我们自己开发的 PHP，Python 和 java 客户端实现提供官方支持。此外，也有一些针对 Perl，Ruby 和 C++的第三方免费、开源 API 实现。

API 的参考实现是用 PHP 写成的，因为（我们相信）较之其他语言，Sphinx 在 PHP 中应用最广泛。因此这份参考文档基于 PHP API 的参考，而且这节中的所有的代码样例都用 PHP 给出。

当然，其他所有 API 都提供相同的方法，也使用完全相同的网络协议。因此这份文档对他们同样适用。在方法命名习惯方面或者具体数据结构的使用上可能会有小的差别。但不同语言的 API 提供的功能上绝不会有差异。

8.1. 通用 API 方法

8.1.1. GetLastError （错误信息）

原型： function GetLastError()

以可读形式返回最近的错误描述信息。如果前一次 API 调用没有错误，返回空字符串。

任何其他函数（如 Query()）失败后（函数失败一般返回 false），都应该调用这个函数，它将返回错误的描述。

此函数本身并*不*重置对错误描述，因此如有必要，可以多次调用。

8.1.2. GetLastWarning （告警信息）

原型： function GetLastWarning ()

以可读格式返回最近的警告描述信息。如果前一次 API 调用没有警告，返回空字符串。

您应该调用这个函数来确认您的请求（如 Query()）是否虽然完成了但产生了警告。例如，即使几个远程代理超时了，对分布式索引的搜索查询也可能成功完成。这时会产生一个警告信息。

此函数本身*不*会重置警告信息，因此如有必要，可以多次调用。

8.1.3. SetServer （设置搜索服务）

原型： function SetServer ($host, $port)

设置 `searchd` 的主机名和 TCP 端口。此后的所有请求都使用新的主机和端口设置。默认的主机和端口分别是“localhost”和 9312。

8.1.4. SetRetries （设置失败重试）

原型： function SetRetries ($count, $delay=0)

设置分布式搜索重试的次数和延迟时间。

对于暂时的失败，`searchd` 对每个代理重试至多`$count` 次。`$delay` 是两次重试之间延迟的时间，以毫秒为单位。默认情况下，重试是禁止的。注意，这个调用**不会**使 API 本身对暂时失败进行重试，它只是让 `searchd` 这样做。目前暂时失败包括 connect() 调用的各种失败和远程代理超过最大连接数（过于繁忙）的情况。

8.1.5. SetConnectTimeout （设置超时时间）

原型： function SetConnectTimeout ($timeout)

设置连接超时时间，在与服务器连接时，如果超过这个时间没有连上就放弃。

有时候服务器在响应上会有所延迟，这有可能由于网络的延时，也有可能是因为服务器未处理完的查询太多，堆积所致。不管是什么情况，有了这个选项，就给客户端应用程序提供了一定的控制权，让它可以决定当 `searchd` 不可用的时候如何处理，而且可以避免脚本由于超过运行限制而运行失败（尤其是在 PHP 里）

当连接失败的而时候，会将合适的错误码返回给应用程序，以便在应用程序级别进行错误处理和通知用户。

8.1.6. SetArrayResult （设置结果返回格式）

原型： function SetArrayResult ($arrayresult)

PHP 专用。控制搜索结果集的返回格式（匹配项按数组返回还是按 hash 返回）

`$arrayresult` 参数应为布尔型。如果`$arrayresult` 为 `false`（默认），匹配项以 PHP hash 格式返回，文档 ID 为键，其他信息（权重、属性）为值。如果`$arrayresult` 为 `true`，匹配项以普通数组返回，包括匹配项的全部信息（包含文档 ID）。

这个调用是对 MVA 属性引入分组支持时同时引入的。对 MVA 分组的结果可能包含重复的文档 ID。因此需要将他们按普通数组返回，因为 hash 对每个文档 ID 仅能保存一个记录。

8.1.7. IsConnectError （检查链接错误）

原型： function IsConnectError ()

检查上一个错误是 API 层面的网络错误还是 searchd 返回的远程错误。如果是上一次连接 searchd 的尝试在 API 层面失败了，返回真，否则返回假（错误发生在远程，或者根本没有尝试连接）。 这是在版本 0.9.9-rc1 引入的。

8.2. 通用搜索设置

8.2.1. SetLimits （设置结果集偏移量）

原型： function SetLimits ($offset, $limit, $max_matches=0, $cutoff=0)

给服务器端结果集设置一个偏移量（`$offset`）和从那个偏移量起向客户端返回的匹配项数目限制（`$limit`）。并且可以在服务器端设定当前查询的结果集大小（`$max_matches`），另有一个阈值（`$cutoff`），当找到的匹配项达到这个阀值时就停止搜索。全部这些参数都必须是非负整数。

前两个参数的行为与 MySQL LIMIT 子句中参数的行为相同。他们令 `searchd` 从编号为`$offset` 的匹配项开始返回最多`$limit` 个匹配项。偏移量(`$offset`)和结果数限制(`$limit`)的默认值分别是 0 和 20，即返回前 20 个匹配项。

`max_matches` 这个设置控制搜索过程中 `searchd` 在内存中所保持的匹配项数目。一般来说，即使设置了 `max_matches` 为 1，**全部**的匹配文档也都会被处理、评分、过滤和排序。但是任一时刻只有最优的 N 个文档会被存储在内存中，这是为了性能和内存使用方面的原因，这个设置正是控制这个 N 的大小。注意，`max_matches` 在**两个**地方设置。针对单个查询的限制由这个 API 调用指定。但还有一个针对整个服务器的限制，那是由配置文件中的 `max_matches` 设置控制的。为防止滥用内存，服务器不允许单个查询的限制高于服务器的限制。

在客户端不可能收到超过 `max_matches` 个匹配项。默认的限制是 1000，您应该不会遇到需要设置得更高的情况。1000 个记录足够向最终用户展示了。如果您是想将结果传输给应用程序以便做进一步排序或过滤，那么请注意，在 Sphinx 端完成效率要**高**得多。

`$cutoff` 设置是为高级性能优化而提供的。它告诉 `searchd` 在找到并处理`$cutoff` 个匹配后就强制停止。

8.2.2. SetMaxQueryTime （设置最大搜索时间）

原型： function SetMaxQueryTime ($max_query_time)

设置最大搜索时间，以毫秒为单位。参数必须是非负整数。默认值为 0，意思是不做限制。

这个设置与 SetLimits()中的`$cutoff` 相似，不过这个设置限制的是查询时间，而不是处理的匹配数目。一旦处理时间已经太久，本地搜索查询会被停止。注意，如果一个搜索查询了多个本地索引，那这个限制独立地作用于这几个索引。

8.2.3. SetOverride （设置临时属性值覆盖）

原型： function SetOverride ($attrname, $attrtype, $values)

设置一个临时的（只对单个查询有效）针对不同文档的属性值覆盖。只支持标量属性。$value 是一个哈希表，他的键是要覆盖属性的文档 ID，之是对应该文档 ID 的要覆盖的值。 于版本 0.9.9-rc1 引入。

属性覆盖特性使用户可以针对一次查询“临时性地”修改一些文档的值，不影响其他查询。这个函数可以用来进行数据个性化。例如，假设 正在实现一个个性 化搜索函数，用来将朋友推荐的帖子排在前面，这类数据不仅是动态的，而且是个性化的，因此不能简单地把这种数据放入索引，因为不能影响其他用户的搜索。而 覆盖机制是针对单个查询的，不会影响其他人。因此可以，比如说，给每个文档设置一个“friends_weight”属性，默认值是 0，然后临时将文档 123，456，789（当前用户的朋友推荐的）的这个属性设置为 1，最后用这个值进行相关度计算。

8.2.4. SetSelect（设置返回信息的内容）

原型： function SetSelect ($clause)

设置 select 子句，列出具体要取出的属性以及要计算并取出的 expressions。子句的语法模仿了 SQL。于版本 0.9.9-rc1 引入。

SetSelect()于标准 SQL 查询中 SELECT 和 FROM 之间的部分非常相近。它允许你指定要取出哪些属性（列），以及在 这些列上要计算和 取出哪些表达式。与 SQL 语言的区别是，表达式必须用关键字 AS 给每个表达式取一个别名，别名必须是有效的标识符（由字母和数字组成）。在 SQL 里面可以 这样做，但是不是强制的。Sphinx **强制**必须有别名，以便计算结果总是可以以一个“正常”的名字在结果集中返回，或者在其他子句中引用，等等。

其他方面基本上等同于 SQL。支持星号（“*”），支持函数，支持任意数目的表达式。计算出的表达式可以用于排序、过滤和分组，这与其他常规属性相同。

从版本 0.9.9-rc2 开始，允许使用 GROUP BY 的时候使用聚集函数（AVG(), MIN(), MAX(), SUM()）。

表达式排序（第 5.6 节 “SPH_SORT_EXPR 模式”）和地理距离计算函数（第 8.4.5 节 “SetGeoAnchor （设置地表距离锚点）”）现在的内部实现就是这种表达式计算机制，分别使用“魔法名字”“@expr”和“@geodist”。

示例：

```
$cl->SetSelect ( "*, @weight+(user_karma+ln(pageviews))*0.1 AS myweight" );
$cl->SetSelect ( "exp_years, salary_gbp*{$gbp_usd_rate} AS salary_usd,
IF(age>40,1,0) AS over40" );
$cl->SetSelect ( "*, AVG(price) AS avgprice" );
```

8.3. 全文搜索设置

8.3.1. SetMatchMode （设置匹配模式）

原型： function SetMatchMode ($mode)

设置全文查询的匹配模式，见第 5.1 节 “匹配模式”中的描述。参数必须是一个与某个已知模式对应的常数。

警告： （仅 PHP）查询模式常量**不能**包含在引号中，那给出的是一个字符串而不是一个常量：

```
$cl->SetMatchMode ( "SPH_MATCH_ANY" ); // INCORRECT! will not work as expected
$cl->SetMatchMode ( SPH_MATCH_ANY ); // correct, works OK
```

8.3.2. SetRankingMode （设置评分模式）

原型： function SetRankingMode ($ranker)

设置评分模式。目前只在 SPH_MATCH_EXTENDED2 这个匹配模式中提供。参数必须是与某个已知模式对应的常数。

Sphinx 默认计算两个对最终匹配权重有用的因子。主要是查询词组与文档文本的相似度。其次是称之为 BM25 的统计函数，该函数值根据关键字文档中的频率（高频导致高权重）和在整个索引中的频率（低频导致高权重）在 0 和 1 之间取值。

然而，有时可能需要换一种计算权重的方法——或者可能为了提高性能而根本不计算权值，结果集用其他办法排序。这个目的可以通过设置合适的相关度计算模式来达到。

已经实现的模式包括：

- SPH_RANK_PROXIMITY_BM25, 默认模式，同时使用词组评分和 BM25 评分，并且将二者结合。
- SPH_RANK_BM25, 统计相关度计算模式，仅使用 BM25 评分计算（与大多数全文检索引擎相同）。这个模式比较快，但是可能使包含多个词的查询的结果质量下降。
- SPH_RANK_NONE, 禁用评分的模式，这是最快的模式。实际上这种模式与布尔搜索相同。所有的匹配项都被赋予权重 1。
- SPH_RANK_WORDCOUNT, 根据关键词出现次数排序。这个排序器计算每个字段中关键字的出现次数，然后把计数与字段的权重相乘，最后将积求和，作为最终结果。
- SPH_RANK_PROXIMITY, 版本 0.9.9-rc1 新增，将原始的词组相似度作为结果返回。在内部，这个模式被用来模拟 SPH_MATCH_ALL 的查询。
- SPH_RANK_MATCHANY, 版本 0.9.9-rc1 新增，返回之前在 SPH_MATCH_ANY 中计算的位次，在内部这个模式用于模拟 SPH_MATCH_ANY 的查询。
- SPH_RANK_FIELDMASK, 版本 0.9.9-rc2 新增，返回一个 32 位掩码，其中第 N 位对应第 N 个全文字段，从 0 开始计数，如果某个字段中出现了满足查询的关键词，则对应的标志位被置 1。
- SPH_RANK_SPH04, 版本 1.10-beta 新增, 基本功能给予默认的 SPH_RANK_PROXIMITY_BM25 评分, 但是当他们在一段文本的最开始或者最后匹配时作了进一步处理。也就是说，当一个字段严格匹配查询时，相对于一个包含查询而非完全相同的字段，SPH04 会 给予更高的评分。（例如，当查询为”Hyde Park”时，一篇名称为”Hyde Park”的文档取得的评分将高于一篇名称为 “Hyde Park, London” 或者 “The Hyde Park Cafe”的文档。）

8.3.3. SetSortMode （设置排序模式）

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

设置匹配项的排序模式，见第 5.6 节 “排序模式”中的描述。参数必须为与某个已知模式对应的常数。

警告： （仅 PHP）查询模式常量**不能**包含在引号中，那给出的是一个字符串而不是一个常量：

```
$cl->SetSortMode ( "SPH_SORT_ATTR_DESC" ); // INCORRECT! will not work as expected
$cl->SetSortMode ( SPH_SORT_ATTR_ASC ); // correct, works OK
```

8.3.4. SetWeights （设置权重）

原型： function SetWeights ($weights)

按在索引中出现的先后顺序给字段设置权重。 **不推荐**使用，建议使用 SetFieldWeights()。

8.3.5. 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。

字段权重影响匹配项的评级。第 5.4 节 “权值计算” 解释了词组相似度如何影响评级。这个调用用于给不同的全文数据字段指定不同于默认值的权值。

给定的权重必须是正的 32 位整数。最终的权重也是个 32 位的整数。默认权重为 1。未知的属性名会被忽略。

目前对权重没有强制的最大限制。但您要清楚，设定过高的权值可能会导致出现 32 位整数的溢出问题。例如，如果设定权值为 10000000 并在扩展模式中进行搜索，那么最大可能的权值为 10M（您设的值）乘以 1000（BM25 的内部比例系数，参见第 5.4 节 “权值计算”, “权值计算”）再乘以 1 或更多（词组相似度评级）。上述结果最少是 100 亿，这在 32 位整数里面没法存储，将导致意想不到的结果。

8.3.6. SetIndexWeights （设置索引权重）

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。

当在不同的本地索引中都匹配到相同的文档 ID 时，Sphinx 默认选择查询中指定的最后一个索引。这是为了支持部分重叠的分区索引。

然而在某些情况下索引并不仅仅是被分区了，您可能想将不同索引中的权值加在一起，而不是简单地选择其中的一个。`SetIndexWeights()`允许您这么做。当开启了加和功能后，最后的匹配权值是各个索引中的权值的加权合，各索引的权由本调用指定。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。

8.4. 结果集过滤设置

8.4.1. SetIDRange （设置查询 ID 范围）

原型： function SetIDRange ($min, $max)

设置接受的文档 ID 范围。参数必须是整数。默认是 0 和 0，意思是不限制范围。

此调用执行后，只有 ID 在`$min` 和`$max`（包括`$min` 和`$max`）之间的文档会被匹配。

8.4.2. SetFilter （设置属性过滤）

原型： function SetFilter ($attribute, $values, $exclude=false)

增加整数值过滤器。

此调用在已有的过滤器列表中添加新的过滤器。`$attribute` 是属性名。`$values` 是整数数组。`$exclude` 是布尔值，它控制是接受匹配的文档（默认模式，即`$exclude` 为 false 时）还是拒绝它们。

只有当索引中`$attribute` 列的值与`$values` 中的任一值匹配时文档才会被匹配（或者拒绝，如果`$exclude` 值为 true）

8.4.3. SetFilterRange （设置属性范围）

原型： function SetFilterRange ($attribute, $min, $max, $exclude=false)

添加新的整数范围过滤器。

此调用在已有的过滤器列表中添加新的过滤器。`$attribute` 是属性名, `$min` 、`$max` 定义了一个整数闭区间，`$exclude` 布尔值，它控制是接受匹配的文档（默认模式，即`$exclude` 为 false 时）还是拒绝它们。

只有当索引中`$attribute` 列的值落在`$min` 和 `$max` 之间（包括`$min` 和 `$max`），文档才会被匹配（或者拒绝，如果`$exclude` 值为 true）。

8.4.4. SetFilterFloatRange （设置浮点数范围）

原型： function SetFilterFloatRange ($attribute, $min, $max, $exclude=false)

增加新的浮点数范围过滤器。

此调用在已有的过滤器列表中添加新的过滤器。`$attribute` 是属性名, `$min` 、`$max` 定义了一个浮点数闭区间，`$exclude` 必须是布尔值，它控制是接受匹配的文档（默认模式，即`$exclude` 为 false 时）还是拒绝它们。

只有当索引中`$attribute` 列的值落在`$min` 和 `$max` 之间（包括`$min` 和 `$max`），文档才会被匹配（或者拒绝，如果`$exclude` 值为 true）。

8.4.5. SetGeoAnchor （设置地表距离锚点）

原型： function SetGeoAnchor ($attrlat, $attrlong, $lat, $long)

为地理距离计算设置锚点，并且允许使用它们。

`$attrlat` 和 `$attrlong` 是字符串，分别指定了对应经度和纬度的属性名称。`$lat` 和 `$long` 是浮点值，指定了锚点的经度和纬度值，以角度为单位。

一旦设置了锚点，您就可以在您的过滤器和/或排序表达式中使用`"@geodist"`特殊属性。Sphinx 将在每一次全文检索中计算给定经纬度与锚点之前的地表距离，并把此距离附加到匹配结果上去。`SetGeoAnchor` 和索引属性数据中的经纬度值都是角度。而结果会以米为单位返回，因此地表距离 1000.0 代表 1 千米。一英里大约是 1609.344 米。

8.5. 分组设置

8.5.1. SetGroupBy （设置分组的属性）

原型： function SetGroupBy ($attribute, $func, $groupsort="@group desc")

设置进行分组的属性、函数和组间排序模式，并启用分组（参考第 5.7 节 “结果分组（聚类）”中的描述）。

`$attribute` 是字符串，为进行分组的属性名。`$func` 为常数，它 指定内建函数，该 函数以前面所述的分组属性的值为输入，目前的可选的值为： SPH_GROUPBY_DAY、SPH_GROUPBY_WEEK、 SPH_GROUPBY_MONTH、 SPH_GROUPBY_YEAR、SPH_GROUPBY_ATTR 。 `$groupsort` 是控制分组如何排序的子句。其语法与第 5.6 节 “SPH_SORT_EXTENDED mode”中描述的相似。

分组与 SQL 中的 GROUP BY 子句本质上相同。此函数调用产生的结果与下面伪代码产生的结果相同。

```
SELECT ... GROUP BY $func($attribute) ORDER BY $groupsort
```

注意，影响最终结果集中匹配项顺序的是$groupsort。排序模式（见第 8.3.3 节 “SetSortMode （设置排序模式）”）影响每个分组*内*的顺序，即每组内哪些匹配项被视为最佳匹配。比如，组之间可以根据每组中的匹配项数量排序的同时每组组内又根据相关度排序。

从版本 0.9.9-rc2 开始，聚合函数 (AVG(), MIN(), MAX(), SUM()) 可以在 GROUP BY 时被 SetSelect() API 调用。

从版本 2.0.1-beta 开始，支持按照字符串属性分组（根据其当前字符集校对）。

8.5.2. SetGroupDistinct （设置分组计算不同值的属性）

原型： function SetGroupDistinct ($attribute)

设置分组中需要计算不同取值数目的属性名。只在分组查询中有效。

$attribute 是包含属性名的字符串。每个组的这个属性的取值都会被储存起来（只要内存允许），其后此属性在此组中不同值的总数会被计算出来并返回给客户端。这个特性与标准 SQL 中的 COUNT(DISTINCT)子句类似。因此如下 Sphinx 调用

```
$cl->SetGroupBy ( "category", SPH_GROUPBY_ATTR, "@count  desc" );
$cl->SetGroupDistinct ( "vendor" );
```

等价于如下的 SQL 语句：

```
SELECT id, weight, all-attributes,
COUNT(DISTINCT vendor) AS @distinct,
COUNT(*) AS @count
FROM products
GROUP BY category
ORDER BY @count  DESC
```

在上述示例伪代码中，SetGroupDistinct()调用只与 COUNT(DISINCT vendor)对应。GROUP BY，ORDER By 和 COUNT(*)子句则与 SetGroupBY()调用等价。两个查询都会在每类中返回一个匹配的行。除了索引中的属性，匹配项还可以包含每类的匹配项计数和每类中不同来源 ID 的计数。

8.6. 搜索数据

8.6.1. Query （查询）

原型： function Query ($query, $index=”*”, $comment=”")

连接到 searchd 服务器，根据服务器的当前设置执行给定的查询，取得并返回结果集。

$query 是查询字串，$index 是包含一个或多个索引名的字符串。一旦发生一般错误，则返回假并设置 GetLastError()信息。若成功则返回搜索的结果集。 此外, $comment 将被发送到查询日志中搜索部分的前面，这对于调试是非常有用的。目前，注释的长度限制为 128 个字符以内。

$index 的默认值是"*"，意思是对全部本地索引做查询。索引名中允许的字符包括拉丁字母（a-z），数字（0-9），减号（-）和下划线（_），其他字符均视为分隔符。因此，下面的示例调用都是有效的，而且会搜索相同的两个索引：

```
$cl->Query ( "test query", "main delta" );
$cl->Query ( "test query", "main;delta" );
$cl->Query ( "test query", "main, delta" );
```

给出多个索引时索引的顺序是有意义的。如果同一个文档 ID 的文档在多个索引中找到，那么权值和属性值会取最后一个索引中所存储的作为该文档 ID 的权值和属性值，用于排序、过滤，并返回给客户端（除非用 SetIndexWeights()显式改变默认行为）。因此在上述示例中，索引“delta”中的匹配项总是比索引“main”中的更优先。

如果搜索成功，Query()返回的结果集包含找到的全部匹配项中的一部分（根据 SetLimits()之设定）和与查询相关的统计数据。结果集是 hash（仅 PHP，其他语言的 API 可能使用其他数据结构） ，包含如下键和值：

“matches”:
: 是一个 hash 表，存储文档 ID 以及其对应的另一个包含文档权重和属性值的 hash 表（或者，如果启用了 SetArrayResult() 则返回数组）。

“total”:
: 本查询在 *搜索服务器*检索所得的匹配文档总数（即服务器端结果集的大小）。这是在当前设置下，用当前查询可以从服务器端获得的搜索结果（匹配文档）数目的上限。其最大限制值由 SetLimits()设定。

“total_found”:
（搜索服务器上找到和处理了的）索引中匹配文档的总数。其与 total 的差别在于：total 表示客户端实际可以读取的结果数目，而 total_found 仅表示服务器端存在多少匹配的文档（永远不会比 total 小）。
“words”:
hash 表，它将查询关键字（关键字已经过大小写转换，取词干和其他处理）映射到一个包含关于关键字的统计数据（“docs”——在多少文档中出现，“hits”——共出现了多少次）的小 hash 表上。
“error”:
searchd 报告的错误信息（人性化可读的字符串）。若无错误则为空字符串。
“warning”:
searchd 报告的警告信息（人类可读字符串）。若无警告则为空串。

需要指出的是 Query() 索执行的操作，与没有中间步骤的 AddQuery()和 RunQueries() 相同 ； 它类似一次单独的 AddQuery() 调用，紧跟一次相应的 RunQueries()调用，然后返回匹配的第一个数组元素 （从第一次，也是仅有的一次查询返回）。

8.6.2. AddQuery （增加批量查询）

原型： function AddQuery ($query, $index=”*”, $comment=”")

向批量查询增加一个查询。$query 为查询串。$index 为包含一个或多个索引名的字符串。 此外，如果提供了$comment，它 将被发送到查询日志中搜索部分的前面，这对于调试是非常有用的。目前，注释的长度限制为 128 个字符以内。返回 RunQueries() 返回的数组中的一个下标。

批量查询（或多查询）使 searchd 能够进行可能的内部优化，并且无论在任何情况下都会减少网络连接和进程创建方面的开销。相对于单独的查询，批量查询不会引入任何额外的开销。因此当您的 Web 页运行几个不同的查询时，一定要考虑使用批量查询。

例如，多次运行同一个全文查询，但使用不同的排序或分组设置，这会使 searchd 仅运行一次开销昂贵的全文检索和相关度计算，然后在此基础上产生多个分组结果。

有时您不仅需要简单地显示搜索结果，而且要显示一些与类别相关的计数信息，例如按制造商分组后的产品数目，此时批量查询会节约大量 的开销。若无批量 查询，您会必须将这些本质上几乎相同的查询运行多次并取回相同的匹配项，最后产生不同的结果集。若使用批量查询，您只须将这些查询简单地组成一个批量查 询，Sphinx 会在内部优化掉这些冗余的全文搜索。

AddQuery()在内部存储全部当前设置状态以及查询，您也可在后续的 AddQuery()调用中改变设置。早先加入的查询不会被影响，实际上没有任何办法可以改变它们。下面是一个示例：

```
$cl->SetSortMode ( SPH_SORT_RELEVANCE );
$cl->AddQuery ( "hello world", "documents" );

$cl->SetSortMode ( SPH_SORT_ATTR_DESC, "price" );
$cl->AddQuery ( "ipod", "products" );

$cl->AddQuery ( "harry potter", "books" );

$results = $cl->RunQueries ();
```

用上述代码，第一个查询会在“documents”索引上查询“hello world”并将结果按相关度排序，第二个查询会在“products”索引上查询“ipod”并将结果按价格排序，第三个查询在“books”索引上搜 索“harry potter”，结果仍按价格排序。注意，第二个 SetSortMode() 调用并不会影响第一个查询（因为它已经被添加了），但后面的两个查询都会受影响。

此外，在 AddQuery()之前设置的任何过滤，都会被后续查询继续使用。因此，如果在第一个查询前使用 SetFilter()，则通过 AddQuery()执行的第二个查询（以及随后的批量查询）都会应用同样的过滤，除非你先调用 ResetFilters()来清除过滤规则。同时，你还可以随时加入新的过滤规则

AddQuery()并不修改当前状态。也就是说，已有的全部排序、过滤和分组设置都不会因这个调用而发生改变，因此后续的查询很容易地复用现有设置。

AddQuery()返回 RunQueries()结果返回的数组中的一个下标。它是一个从 0 开始的递增整数，即，第一次调用返回 0，第二次返回 1，以此类推。这个方便的特性使你在需要这些下标的时候不用手工记录它们。

8.6.3. RunQueries （执行批量查询）

原型： function RunQueries ()

连接到 searchd，运行由 `AddQuery()` 添加的全部查询，获取并返回它们的结果集。若发生一般错误（例如网络 I/O 失败）则返回假并设置 `GetLastError()` 信息。若成功则返回结果集的简单数组。

该数组中的每一个结果集都跟 Query()返回的结果集完全相同。

注意，批量查询请求自身几乎总是成功——除非有网络错误、正在进行索引轮换，或者其他导致整个查询无法被处理的因素。

然而其中的单个的查询很可能失败。此时与之对应的结果集只包含一个非空的`"error"`信息，而没有关于匹配或查询的统计信息。在极端情况下，批量查询中的所有单个查询可能都失败。但这仍然不会导致报告一般错误，因为 API 已经成功地连接到 `searchd`，提交了批量查询并得到返回结果——但每个结果集都只包含特定的错误信息。

8.6.4. ResetFilters （清除当前设置的过滤器）

原型： function ResetFilters ()

清除当前设置的过滤器。

通常此调用在使用批量查询的时候会用到。您可能需要为批量查询中的不同查询提供不同的过滤器，为达到这个目的，您需要调用 `ResetFilters()` 然后用其他调用增加新的过滤器。

8.6.5. ResetGroupBy （清除现有的分组设置）

原型： function ResetGroupBy ()

清除现有的全部分组设置，并取消分组设定。

通常此调用在使用批量查询的时候会用到。单独的分组设置可以用 `SetGroupBy()` 和 `SetGroupDistinct()` 来改变，但它们不能关闭分组。`ResetGroupBy()` 将之前的分组设置彻底重置并在当前状态下关闭分组模式，因此后续的 `AddQuery()` 可以进行无分组的搜索。

8.7. 附加方法

8.7.1. BuildExcerpts （产生文本摘要和高亮）

原型： function BuildExcerpts ($docs, $index, $words, $opts=array())

该函数用来产生文档片段（摘要）。连接到 `searchd`，要求它从指定文档中产生片段（摘要），并返回结果。

`$docs` 为包含各文档内容的数组。`$index` 为包含索引名字的字符串。给定索引的不同设置（例如字符集、形态学、词形等方面的设置）会被使用。`$words` 为包含需要高亮的关键字的字符串。它们会按索引的设置被处理。例如，如果英语取词干（stemming）在索引中被设置为允许，那么即使关键词是“shoe”，“shoes”这个词也会被高亮。从版本 0.9.9-rc1 开始，关键字可以包含通配符，与查询支持的 star-syntax 类似。`$opts` 为包含其他可选的高亮参数的 hash 表：

“before_match”：
在匹配的关键字前面插入的字符串。从版本 1.10-beta 开始，可以在该字符串中使用%PASSAGE_ID%宏。该宏会被替换为当前片段的递增值。递增值的起始值默认为 1，但是可以通 过”start_passage_id”设置覆盖。在多文档中调用时，%PASSAGE_ID%会在每篇文档中重新开始。默认为””。
“after_match”：
在匹配的关键字后面插入的字符串。从版本 1.10-beta 开始，可以在该字符串中使用%PASSAGE_ID%宏。默认为 “”。
“chunk_separator”：
在摘要块（段落）之间插入的字符串。默认为” … ”.
“limit”：
摘要最多包含的符号（码点）数。整数，默认为 256。
“around”：
每个关键词块左右选取的词的数目。整数，默认为 5.
“exact_phrase”：
是否仅高亮精确匹配的整个查询词组，而不是单独的关键词。布尔值，默认为 false.
“single_passage”：
是否仅抽取最佳的一个区块。布尔值，默认为 false.
“use_boundaries”：
是否跨越由 phrase_boundary 选项设置的词组边界符。布尔型，默认为 false.
“weight_order”：
对于抽取出的段落，要么根据相关度排序（权重下降），要么根据出现在文档中的顺序（位置递增）。布尔型，默认是 false.

“query_mode”：
版本 1.10-beta 新增。设置将$words 当作 扩展查询语法的 查询处理，还是当做普通的文本字符串处理（默认行为）。例如，在查询模式时，（“one two” | “three four”）仅高亮和包含每个片段中出现”one two” 或 “three four”的地方及相邻的部分。而在默认模式时， 每个单独出现”one”， “two”， “three”， 或 “four”的地方都会高亮。布尔型，默认是 false。
“force_all_words”：
版本 1.10-beta 新增. 忽略摘要的长度限制直至包含所有的词汇。布尔型，默认为 false.
“limit_passages”：
版本 1.10-beta 新增. 限制摘要中可以包含的最大区块数。整数值，默认为 0 (不限制).
“limit_words”：
版本 1.10-beta 新增. 限制摘要中可以包含的最大词汇数。整数值，默认为 0 (不限制).
“start_passage_id”：
版本 1.10-beta 新增. 设置 %PASSAGE_ID% 宏的起始值 (在 before_match, after_match 字符串中检查和应用). 整数值，默认为 1.
“load_files”：
版本 1.10-beta 新增. 设置是否将$docs 作为摘要抽取来源的数据(默认行为),或者将其当做文件名。从版本 2.0.1-beta 开始，如果该标志启用，每个请求将创建最多 dist_threads 个工作线程进行并发处理。布尔型，默认为 false.
“html_strip_mode”：
版本 1.10-beta 新增. HTML 标签剥离模式设置。默认为”index”，表示使用 index 的设置。可以使用的其他值为”none”和”strip”，用于强制跳过或者应用剥 离，而不管索引如何设置的。还可以使用”retain”，表示保留 HTMK 标签并防止高亮时打断标签。”retain”模式仅用于需要高亮整篇文档，并且 不能设置限制片段的大小。字符型，可用值为”none”，”strip”，”index”或者”retain”。
“allow_empty”：
版本 1.10-beta 新增. 允许无法产生摘要时将空字符串作为高亮结果返回 (没有关键字匹配或者不符合片段限制。). 默认情况下，原始文本的开头会被返回而不是空字符串。布尔型，默认为 false.
“passage_boundary”：
版本 2.0.1-beta 新增. 确保区块不跨越句子，段落或者 zone 区域（仅当每个索引的设置启用时有效）。字符型，可用值为 “sentence”， “paragraph”， 或者 “zone”.
“emit_zones”：
版本 2.0.1-beta 新增. 在每个区块前使用区域对应的 HTML 标签来封闭区域。布尔型，魔默认为 false。

摘要提取算法倾向于提取更好的片段（与关键词完全匹配），然后是不在摘要中但是包含了关键词的片段。 通常情况下，它会尝试高亮查询的最佳匹配，并且在限制条件下尽可能的高亮查询中的所有关键词。 如果文档没有匹配查询，默认情况下将根据限制条件返回文档的头部。从版本 1.10-beta 开始，可以通过设置 allow_empty 属性位 true 以返 回空的片段来替代默认方式。

失败时返回 false。成功时返回包含有片段（摘要）字符串的数组。

8.7.2. UpdateAttributes （更新属性）

原型： function UpdateAttributes ($index, $attrs, $values)

立即更新指定文档的指定属性值。成功则返回实际被更新的文档数目（0 或更多），失败则返回-1。

`$index` 为待更新的（一个或多个）索引名。$attrs 为属性名字符串的数组，其所列的属性会被更新。`$attrs` 为 hash 表，`$values` 表的键为文档 ID，`$values` 表的值为新的属性值的简单数组。

`$index` 既可以是一个单独的索引名，也可以是一个索引名的列表，就像 `Query()` 的参数。与 `Query()` 不同的是不允许通配符，全部待更新的索引必须明确指出。索引名列表可以包含分布式索引。对分布式索引，更新会同步到全部代理上。

只有在 `docinfo=extern` 这个存储策略下才可以运行更新。更新非常快，因为操作完全在内存中进行，但它们也可以变成持久的，更新会在 `searchd` 干净关闭时（收到 SIGTERM 信号时）被写入磁盘。在额外限制条件下，MVA 属性也可以被更新，参见 mva_updates_pool 详细了解。

使用示例：

```
$cl->UpdateAttributes ( "test1", array("group_id"), array(1=>array(456)) );
$cl->UpdateAttributes ( "products", array ( "price", "amount_in_stock" ),
array ( 1001=>array(123,5), 1002=>array(37,11), 1003=>(25,129) ) );
```

第一条示例语句会更新索引“test1”中的文档 1，设置“group_id”为 456.第二条示例语句则更新索引 “products”中的文档 1001，1002 和 1003。文档 1001 的“price”会被更新为 123，“amount_in_stock”会被更新为 5；文档 1002，“price”变为 37 而“amount_in_storage”变为 11，等等。

8.7.3. BuildKeywords （获取分词结果）

原型： function BuildKeywords ($query, $index, $hits)

根据指定索引的符号化（tokenizer）方式的设置，从查询中抽取关键词，也可以同时返回每个关键词出现次数的统计信息。返回一个数组，其元素是一些字典，每个字典包含一个关键字的信息。

`$query` 是抽取关键字的目标。`$index` 是某个索引的名字，系统会使用这个索引的符号化（tokenizer）设置，关键词出现次数的统计信息也从这个索引中得出。`$hits` 是一个布尔值，它指定了是否需要返回关键词出现此处的信息。

使用示例：

```
$keywords = $cl->BuildKeywords ( "this.is.my query", "test1", false );
```

8.7.4. EscapeString （转义特殊字符）

原型： function EscapeString ($string)

查询语言分析器将某些字符理解成特殊操作符，这个函数对字符串中的那些有特殊意义的字符进行转义。返回转义后的字符串。

`$string` 是待转义的字符串。

表面上看这个函数是多余的，因为可以很容易地在可能调用这个函数的程序里实现这个转义功能。然而这些特殊字符的集合可能随着时间而改变，因此理应提供一个 API 调用来完成这个功能，并保证任何时候都可以正确地转义全部特殊字符。

使用示例：

```
$escaped = $cl->EscapeString ( "escaping-sample@query/string" );
```

8.7.5. Status （查询服务状态）

原型： function Status ()

查询 searchd 的状态，返回一个数组，数组元素是由状态变量名和值的键值对构成。

使用示例：

```
$status = $cl->Status ();
foreach ( $status as $row )
print join ( ": ", $row ) . "\n";
```

8.7.6. FlushAttributes （强制更新属性到磁盘）

原型： function FlushAttributes ()

强制 `searchd` 刷新等待更新的属性到磁盘，并阻塞访问直到完成。 成功时返回一个非负的内部刷新标记值。错误时返回-1，并设置错误信息。 版本 2.0.1-beta 引入。

使用 UpdateAttributes()API 调用更新的属性值将一直保存在内存中，直到一次这样的刷新（将当前所有可能需要更新的属性值写回到磁盘）。FlushAttributes()调用将强制执行一次更新。 调用后将会阻塞访问直到 `searchd` 将数据全部写入到磁盘，这个过程可能会需要数秒或者数分钟，这取决于所有待更新数据的大小（.spa 文件的大小）。所有当前被更新的索引都会被刷新。

刷新标记仅仅是一个不断增长的不定数值，并没有实际意义。它一定是一个非负值，并且随着时间而增长，但是并不一定是连续增长；例 如，两个调用分别返 回 10 和 1000，他们都是有效的返回值。如果两个 FlushAttributes()调用返回相同的标记值，就意味着没有在他们两者之间任何实际的属性 更新，英雌当前的刷新状态将保持不变（所有索引）。

使用示例：

```
$status = $cl->FlushAttributes ();
if ( $status<0 )
print "ERROR: " . $cl->GetLastError();
```

8.8. 持久连接

“持久连接”特性允许利用一个单独的网络连接来运行本来需要多个连接的多个命令。

8.8.1. Open（打开连接）

原型： function Open ()

打开到服务器的持久连接。

8.8.2. Close （关闭连接）

原型： function Close ()

关闭先前打开的持久连接。

第 9 章 MySQL 存储引擎 (SphinxSE)

目录

9.1. SphinxSE 概览

SphinxSE 是一个可以编译进 MySQL 5.x 版本的 MySQL 存储引擎，它利用了该版本 MySQL 的插件式体系结构。SphinxSE 不能用于 MySQL 4.x 系列，它需要 MySQL 5.0.22 或更高版本；或 MySQL 5.1.12 或更高版本；或 MySQL 5.5.8 或更高版本

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。

显然，SphinxSE 的适用于：

- 使将 MySQL FTS 应用程序移植到 Sphinx；
- 使没有 Sphinx API 的那些语言也可以使用 Sphinx；
- 当需要在 MySQL 端对 Sphinx 结果集做额外处理（例如对原始文档表做 JOIN，MySQL 端的额外过滤等等）时提供优化。

9.2. 安装 SphinxSE

你需要获得 MySQL 的源代码，然后重新编译 MySQL。MySQL 的源代码 (mysql-5.x.yy.tar.gz) 可在 dev.mysql.com 网站获得.

针对某些版本的 MySQL，Sphinx 网站提供了包含支持 SphinxSE 的打过补丁 tarball 压缩包。将这些文件解压出来替换原始文件，就可以配置(configure)、构建(build)以生成带有内建 Shpinx 支持的 MySQL 了。

如果网站上没有对应版本的 tarball，或者由于某种原因无法工作，那您可能需要手工准备这些文件。您需要一份安装好的 GUN Autotools 框架（autoconf，automake 和 libtool）来完成这项任务。

9.2.1. 编译支持 SphinxSE 的 MySQL 5.0.x

如果使用我们事先做好的打过补丁的 tarball，那请跳过步骤 1-3。

1. 将 `sphinx.5.0.yy.diff` 补丁文件复制到 MySQL 源码目录并运行

```
patch -p1 < sphinx.5.0.yy.diff
```

2.

如果没有与您的 MySQL 版本完全匹配的.diff 文件，请尝试一个最接近版本的.diff 文件。确保补丁顺利应用，没有 rejects。

3.

4. 在 MySQL 源码目录中运行

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

5.

6. 在 MySQL 源码目录中建立 `sql/sphinx` 目录，并把 Sphinx 源码目录中 `mysqlse` 目录下的全部文件拷贝到这个目录。示例：

```
cp -R /root/builds/sphinx-0.9.7/mysqlse/* /root/builds/mysql-5.0.24/sql/sphinx
```

7.

8. 配置（configure）MySQL，启用 Sphinx 引擎：

```
./configure --with-sphinx-storage-engine
```

9.

10. 构建（build）并安装 MySQL：

```
make
make install
```

11.

9.2.2. 编译支持 SphinxSE 的 MySQL 5.1.x

如果使用我们事先做好的打过补丁的 tarball，那请跳过步骤 1-2。

1. 在 MySQL 源码目录中建立 `storage/sphinx` 目录，并将 Sphinx 源码目录中的 `mysqlse` 目录下的全部文件拷贝到这个目录。示例：

```
cp -R /root/builds/sphinx-0.9.7/mysqlse/* /root/builds/mysql-5.1.14/storage/sphinx
```

2.

3. 在 MySQL 源码目录运行

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

4.

5. 配置（configure）MySQL，启用 Sphinx 引擎

```
./configure --with-plugins=sphinx
```

6.

7. 构建（build）并安装 MySQL

```
make
make install
```

8.

9.2.3. 检查 SphinxSE 安装与否

为了检查 SphinxSE 是否成功地编入了 MySQL，启动新编译出的 MySQL 服务器，运行 mysql 客户端，执行 `SHOW ENGINES` 查询，这会显示一个全部可用引擎的列表。Sphinx 应该出现在这个列表中，而且在“Support”列上显示“YES”：

```
mysql> show engines;
+------------+----------+----------------------------------------------------------------+
| Engine     | Support  | Comment                                                        |
+------------+----------+----------------------------------------------------------------+
| MyISAM     | DEFAULT  | Default engine as of MySQL 3.23 with great performance         |
...
| SPHINX     | YES      | Sphinx storage engine                                          |
...
+------------+----------+----------------------------------------------------------------+
13 rows in set (0.00 sec)
```

9.3. 使用 SphinxSE

要通过 SphinxSE 搜索，您需要建立特殊的 ENGINE=SPHINX 的“搜索表”（中间表，没有实际数据，查询时从 searchd 获取数据返回给 MySQL），然后使用 SELECT 语句从中检索，把全文查询放在 WHERE 子句中。

让我们从一个 create 语句和搜索查询的例子开始：

```
CREATE TABLE t1
(
id          INTEGER UNSIGNED NOT NULL,
weight      INTEGER NOT NULL,
query       VARCHAR(3072) NOT NULL,
group_id    INTEGER,
INDEX(query)
) ENGINE=SPHINX CONNECTION="sphinx://localhost:9312/test";

SELECT * FROM t1 WHERE query='test it;mode=any';
```

搜索表前三列（字段）的类型*必须*是 INTEGER UNSINGED（或者 BIGINT），INTEGER（或者 BIGINT）和 VARCHAR（或者 TEXT），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。搜索查询列必须被索引，其他列必须无索引。列的名字会被忽略，所以可以任意命名。

除此之外，其他列（字段）的类型必须是 INTEGER、TIMESTAMP、BIGINT、VARCHAR 或者 FLOAT 之一。它们必须与 Sphinx 结果集中提供的属性按名称绑定，即它们的名字必须与 sphinx.conf 中指定的属性名一一对应。如果 Sphinx 搜索结果中没有某个属性名，该列的值就为 NULL.

特殊的内部“虚拟”属性名也可以与 SphinxSE 列绑定。但特殊符号@用_sph_代替。例如，要取得@group 和@count 或者@distinct 内部属性，列名应分别使用_sph_group、_sph_count 或者_sph_distinct。

可以使用字符串参数 CONNECTION 来指定用这个表搜索时的默认搜索主机、端口号和索引。如果 CREATE TABLE 中没有使用连接（connection）串，那么默认使用索引名“*”（搜索所有索引）和 localhost:9312。连接串的语法如下：

```
CONNECTION="sphinx://HOST:PORT/INDEXNAME"
```

默认的连接串也可以需要时改变：

```
ALTER TABLE t1 CONNECTION="sphinx://NEWHOST:NEWPORT/NEWINDEXNAME";
```

也可以在查询中覆盖全部这些选项。

如例子所示，查询文本和搜索选项都应放在 WHERE 子句中对 query 列的限制中（即第三列），选项之间用分号分隔，选项名与选项值用等号隔开。可以指定任意数目的选项。可用的选项如下：

- query – 查询文本；
- mode – 匹配模式.必须是 “all”, “any”, “phrase”, “boolean”, 或者 “extended”, 或者 “extended2″ 之一。默认为 “all”；
    - sort – 匹配项排序模式 必须是 “relevance”, “attr_desc”, “attr_asc”, “time_segments”, 或者 “extended” 之一。除了“relevance”模式，其他模式中还必须在一个冒号后附上属性名（或“extended”模式中的排序子句）：

```
... WHERE query='test;sort=attr_asc:group_id';
... WHERE query='test;sort=extended:@weight desc, group_id asc';
```

    -
- offset – 结果集中的偏移量，默认是 0;
- limit – 从结果集中获取的匹配项数目，默认为 20;
    - index – 待搜索的索引：

```
... WHERE query='test;index=test1;';
... WHERE query='test;index=test1,test2,test3;';
```

    -
- minid, maxid – 匹配文档 ID 的最小值和最大值；
    - weights – 逗号分隔的列表，指定 Sphinx 全文数据字段的权值：

```
... WHERE query='test;weights=1,2,3;';
```

    -

- filter, !filter – 逗号分隔的列表，指定一个属性名和一系列可匹配的属性值：

```
# only include groups 1, 5 and 19
... WHERE query='test;filter=group_id,1,5,19;';

# exclude groups 3 and 11
... WHERE query='test;!filter=group_id,3,11;';
```

-
- range, !range – 逗号分隔的列表，指定一个属性名和该属性可匹配的最小值和最大值：

```
# include groups from 3 to 7, inclusive
... WHERE query='test;range=group_id,3,7;';

# exclude groups from 5 to 25
... WHERE query='test;!range=group_id,5,25;';
```

-
- maxmatches – 此查询最大匹配的数量，与 SetLimits() API 调用的 max_matches 参数类似：

```
... WHERE query='test;maxmatches=2000;';
```

-
- cutoff – 此查询最大允许的匹配数，与 SetLimits() API 调用的参数 cutoff 类似：

```
... WHERE query='test;cutoff=10000;';
```

-
- maxquerytme – 此查询最大允许的查询时间 (单位为毫秒)，如同 SetMaxQueryTime() API 调用：

```
... WHERE query='test;maxquerytime=1000;';
```

-
- groupby – 分组（group-by）函数和属性，对应 SetGroupBy() API 调用:

```
... WHERE query='test;groupby=day:published_ts;';
... WHERE query='test;groupby=attr:group_id;';
```

-
- groupsort – 分组（group-by）排序子句:

```
... WHERE query='test;groupsort=@count  desc;';
```

-
- distinct – 分组（group-by）时用于计算 COUNT(DISTINCT) 的属性名称，如同 SetGroupDistinct() API 调用：

```
... WHERE query='test;groupby=attr:country_id;distinct=site_id';
```

-
- indexweights – 逗号分隔的列表，指定一系列索引名和搜索时这些索引对应的权值：

```
... WHERE query='test;indexweights=idx_exact,2,idx_stemmed,1;';
```

-
- comment – 用于查询日志标记本次查询的字符串 (对应 Query() API 调用的$comment 参数):

```
... WHERE query='test;comment=marker001;';
```

-
- select – 用于计算的表达式 (对应 SetSelect() API 调用):

```
... WHERE query='test;select=2*a+3*b as myexpr;';
```

-
- host, port – 分别表示本地或者远端 `searchd` 主机名称和 TCP 端口:

```
... WHERE query='test;host=sphinx-test.loc;port=7312;';
```

-

- ranker – 当匹配模式（查询语法）为 extended 时用于评分模式，对应 SetRankingMode() API 调用. 可用值包括 “proximity_bm25″, “bm25″, “none”, “wordcount”, “proximity”, “matchany”, 和 “fieldmask”.

```
... WHERE query='testrmode=extended2;anker=bm25;';
```

-
- geoanchor – 地理坐标锚点，对应 SetGeoAnchor() API 调用. 参数有四个分别是维度与经度属性的名称以及定位点的坐标：

```
... WHERE query='test;geoanchor=latattr,lonattr,0.123,0.456';
```

-

一个**非常重要**的注意事项：让 Sphinx 来对结果集执行排序、过滤和切片（slice）要比提高最大匹配项数量然后在 MySQL 端用 WHERE、ORDER BY 和 LIMIT 子句完成对应的功能来得高效得**多**。这有两方面的原因。首先，Sphinx 对这些操作做了一些优化，比 MySQL 效率更高一些。其次，searchd 可以打包更少的数据，SphinxSE 也可以传输和解包更少的数据。

从版本 0.9.9-rc1 开始，除了结果集，额外的查询信息可以用 `SHOW ENGINE SPHINX STATUS` 语句获得：

```
mysql> SHOW ENGINE SPHINX STATUS;
+--------+-------+-----------------------------------------------+
| Type   | Name  | Status                                        |
+--------+-------+-----------------------------------------------+
| SPHINX | stats | total: 25, total found: 25, time: 126, words: 2 |
| SPHINX | words | sphinx:591:1256 soft:11076:15945              |
+--------+-------+-----------------------------------------------+
2 rows in set (0.00 sec)
```

查询状态信息可以通过状态变量名来访问。值得提醒的是，访问这些信息不需要超级用户权限。

```
mysql> SHOW STATUS LIKE 'sphinx_%';
+------------------+--------------------------------+
| Variable_name    | Value                          |
+------------------+--------------------------------+
| sphinx_total       | 25                           |
| sphinx_total_found | 25                           |
| sphinx_time        | 126                          |
| sphinx_word_count  | 2                            |
| sphinx_words       | sphinx:591:1256 soft:11076:15945 |
+------------------+--------------------------------+
5 rows in set (0.00 sec)
```

可以对 SphinxSE 搜索表和其他引擎的表之间使用 JOIN，以下是一个例子，例中“documents”来自 `example.sql`：

```
mysql> SELECT content, date_added FROM test.documents docs
-> JOIN t1 ON (docs.id=t1.id)
-> WHERE query="one document;mode=any";
+----------------------------------+---------------------+
| content                          | docdate             |
+----------------------------------+---------------------+
| this is my test document number two | 2006-06-17 14:04:28 |
| this is my test document number one | 2006-06-17 14:04:28 |
+----------------------------------+---------------------+
2 rows in set (0.00 sec)

mysql> SHOW ENGINE SPHINX STATUS;
+--------+-------+-----------------------------------------+
| Type   | Name  | Status                                  |
+--------+-------+-----------------------------------------+
| SPHINX | stats | total: 2, total found: 2, time: 0, words: 2 |
| SPHINX | words | one:1:2 document:2:2                    |
+--------+-------+-----------------------------------------+
2 rows in set (0.00 sec)
```

9.4. 通过 MySQL 生成片段（摘要）

从版本 0.9.9-rc2 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 的功能非常相似，但可以通过 MySQL+SphinxSE 来访问。

提供这个 UDF 的二进制文件叫做 `sphinx.so`，当安装 SphinxSE 本身的时候，这个文件会自动地被创建，并且安装到合适的位置。 但如果由于某种原因它没能自动安装，那就请在创建 SphinxSE 的目录中寻找 `sphinx.so` 文件，并把它拷贝到你的 MySQL 实例的 plugins 目录下。然后用下面语句来注册这个 UDF：

```
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 和 words 这两个参数可以是字符串或者数据库表的列。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()。只有一个选项不被 API 支持而只能用于 UDF，这个选项叫做`'sphinx'`，用于指定 searchd 的位置（服务器和端口）。

使用示例：

```
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;
```

第 10 章 报告 BUG

很不幸，Sphinx 还没有达到 100%无 bug（尽管我们正向这个目标努力），因此您可能偶尔遇到些问题。

对于每个问题的报告越详细越好，这很重要——因为要想修复 bug，我们必须重现 bug 并调试它，或者根据您提供的信息来推断出产生 bug 的原因。因此在此提供一些如何报告 bug 的指导。

编译时的问题

如果 Sphinx 构建失败，请您按照以下步骤进行：

1. 确认您的 DBMS（数据库）的头文件和库文件都正确安装了（例如，检查 `mysql-devel` 包已经安装）
2. 报告 Sphinx 的版本和配置文件（别忘了删除数据库连接密码），MySQL（或 PostgreSQL）配置文件信息，gcc 版本，操作系统版本和 CPU 类型（例如 x86、x86-64、PowerPC 等）：

```
mysql_config
gcc --version
uname -a
```

3.
4. 报告 `configure` 脚本或者 `gcc` 给出的错误信息（只需错误信息本身，不必附上整个构建日志）

运行时的问题

如果 Sphinx 已经成功构建并能运行，但运行过程中出现了问题，请您按照以下步骤进行：

1. 描述您遇到的 bug（即，您预期的行为和实际发生的行为），以及重现您遇到的问题需要的步骤；
2. 附带 Sphinx 的版本和配置文件（别忘了删除密码），MySQL（或 PostgreSQL）配置文件信息，gcc 版本，操作系统版本和 CPU 类型（例如 x86、x86-64、PowerPC 等）：

```
mysql --version
gcc --version
uname -a
```

3.
4. 构建、安装和运行调试版本的全部 Sphinx 程序（这会启用很多内部校验，或称断言(assertions)）：

```
make distclean
./configure --with-debug
make install
killall -TERM searchd
```

5.
6. 重新索引，检查是否有断言(assertions)被触发（如果是，那很可能是索引损坏了并造成了问题）；
7. 如果 bug 在调试版本中没有重现，请回到非调试版本并在报告中说明这个情况；
8. 如果 bug 可以在您的数据库的很小的子集（1-100 条记录）上重现，请提供一个该子集的 gzip 压缩包；
9. 如果问题与 `searchd` 有关，请在 bug 报告中提供 `searchd.log` 和 `query.log` 中的相关条目；
10. 如果问题与 `searchd` 有关，请尝试在 console 模式下运行它并检查它是否因断言失败而退出。

```
./searchd --console
```

11.
12. 如果任何一个程序因断言失败而退出，请提供断言(assertions)信息。

调试断言，崩溃和挂断

如果任何一个程序因断言(assertions)失败而退出，崩溃或停止响应，您可以额外生成一个内存转储文件并检查它。

1. 启用内存转储。在大多数 Linux 系统上，可以用 `ulimit` 命令启用它：

```
ulimit -c 32768
```

2.
3. 运行程序，尝试重现 bug；
4. 如果程序崩溃（可能有断言失败的情况也可能没有），在当前目录下找到内存转储文件（一般会打印“Segmentation fault (core dumped)”消息）；
5. 如果程序停止响应，在另一个控制台上用 `kill -SEGV` 强制退出并获得内存转储：

```
kill -SEGV HANGED-PROCESS-ID
```

6.
7. 使用 `gdb` 检查转储文件，查看 backtrace：

```
gdb ./CRASHED-PROGRAM-FILE-NAME CORE-DUMP-FILE-NAME
(gdb) bt
(gdb) quit
```

8.

提示： HANGED-PROCESS-ID（停止响应的进程 ID）, CRASHED-PROGRAM-FILE-NAME（崩溃程序的文件名） and CORE-DUMP-FILE-NAME（核心转储文件的文件名）应该被换成具体的数字和文件名。例如，一次对停止响应的 searchd 的调试会话看起来 应该像下面这样：

```
# kill -SEGV 12345
# ls *core*
core.12345
# gdb ./searchd core.12345
(gdb) bt
...
(gdb) quit
```

注意 `ulimit` 并不是整个服务器范围的，而是仅影响当前的 shell 会话。因此您不必还原任何服务器范围的限制——但是一旦重新登陆，您就需要再次设置 `ulimit`。

核心内存转储文件会存放在当前工作目录下（Sphinx 的各个程序不会改变工作目录），因此它们就在那。

不要立刻删除转储文件，从它里面可能获得更多有用的信息。您不需要把这个文件发送给我们（因为调试信息与您的系统本身紧密相关），但我们可能会向您询问一些与之相关的问题。

第 11 章 sphinx.conf/csft.conf 配置选项参考

目录

11.1. 数据源配置选项

11.1.1. type：数据源类型

数据源类型。必须选项，无默认值。 可用的类型包括 `mysql`, `pgsql`, `mssql`, `xmlpipe` and `xmlpipe2`, `odbc`， 以及 `python`.

在 CoreSeek 的分发版本中，新增了 `python` 数据源类型，从而得以启用 Python 数据源支持。

所有其他与数据源相关的选项都依赖于这个选项指定的源类型。与 SQL 数据源（即 MSSQL、MySQL 和 PostgreSQL）相关的选项以“sql_”开头，而与 xmlpipe 和 xmlpipe2 数据源相关的选项则以“xmlpipe_”开头。 除了 `xmlpipe` 是默认支持外，其他数据源类型的支持是有前提条件的；依赖与您的设置和已安装的数据库客户端库文件，它们可能被支持或者不被支持。例如，`mssql` 仅在 Windows 系统提供支持。`odbc` 在 Windows 系统上是原生支持，而在 Linux 系统上通过 UnixODBC library 支持。

在 CoreSeek 的分发版本中，`python` 通过 Python 提供支持，在安装了 Pythin 的 Windows 系统和 Linux 系统上都可以支持。

示例：

```
type = mysql
```

11.1.2. sql_host：数据库服务器

要连接的 SQL 服务器主机地址。必须选项，无默认值。仅对 SQL 数据源（`mysql`, `pgsql`, `mssql`）有效。

最简单的情形下，Sphinx 与 MySQL 或 PostgreSQL 服务器安装在同一台主机上，此时您只须设置为 localhost 即可。注 意，MySQL 客户端库根据主机名决定是通过 TCP/IP 还是 UNIX socket 连接到服务器。一般来说，“localhost”使之强制使用 UNIX socket 连接（这是默认的也是推荐的模式），而“127.0.01”会强制使用 TCP/IP。细节请参考 MySQL manual

示例：

```
sql_host = localhost
```

11.1.3. sql_port：数据库端口

要连接的 SQL 服务器的 IP 端口。可选选项，默认值为 `mysql` 端口 3306，`pgsql` 端口 5432。仅适用于 SQL 数据源（`mysql`, `pgsql`, `mssql`）。注意，此选项是否实际被使用依赖于 sql_host 选项。

示例：

```
sql_port = 3306
```

11.1.4. sql_user：数据库用户名

连接到 sql_host 时使用的 SQL 用户名。必须选项，无默认值。仅适用于 SQL 数据源（`mysql`, `pgsql`, `mssql`）。

示例：

```
sql_user = test
```

11.1.5. sql_pass：数据库密码

连接到 sql_host 时使用的 SQL 用户密码。必须选项，无默认值。 仅适用于 SQL 数据源（`mysql`, `pgsql`, `mssql`）。

示例：

```
sql_pass = mysecretpassword
```

11.1.6. sql_db：数据库名称

连接到 SQL 数据源之后使用的 SQL 数据库，此后的查询均在此数据库上进行。必须选项，无默认值。 仅适用于 SQL 数据源（`mysql`, `pgsql`, `mssql`）。

示例：

```
sql_db = test
```

11.1.7. sql_sock：数据库 Socket 文件

连接到本地 SQL 服务器时使用的 UNIX socket 名称。可选选项，默认值为空（使用客户端库的默认设置）。 仅适用于 SQL 数据源（`mysql`, `pgsql`, `mssql`）。

在 Linux 上，通常是`/var/lib/mysql/mysql.sock`。 而在 FreeBSD 上通常是`/tmp/mysql.sock`。注意此选项是否实际被使用依赖与 sql_host 的设置。

示例：

```
sql_sock = /tmp/mysql.sock
```

11.1.8. mysql_connect_flags：MySQL 连接参数

MySQL 客户端的连接标志（connection flags）。可选选项，默认值为 0（不设置任何标志）。仅适用于 `mysql` 数据源。

此选项必须包含各标志相加所得的整型值。此整数将被原样传递给 mysql_real_connect() 。 可用的标志在 mysql_com.h 中列举。下面列举的是几个与索引相关的标志和它们的值：

- CLIENT_COMPRESS = 32; 允许使用压缩协议 protocol
- CLIENT_SSL = 2048; 握手后切换到 SSL
- CLIENT_SECURE_CONNECTION = 32768; 新的 MySQL 4.1 版本身份认证

例如，标志 2080（2048+32）代表同时使用压缩和 SSL，32768 代表仅使用新的身份验证。起初这个选项是为了在 `indexer` 和 `mysql` 位 于不同主机的情况下使用压缩协议而引入的。尽管降低了网络带宽消耗，但不管在理论上还是在现实中，在 1Gbps 的链路上启用压缩很可能恶化索引时间。然而 在 100Mbps 的连输上启用压缩可能会明显地改善索引时间（有报告说总的索引时间降低了 20-30%）。根据您网络的连接情况，您获得的改善程度可能会 有所不同。

示例：

```
mysql_connect_flags = 32 # 启用压缩
```

11.1.9. mysql_ssl_cert, mysql_ssl_key, mysql_ssl_ca：MySQL 的 SSL 连接

连接 MySQL 服务器时使用的 SSL 认证选项。可选参数，默认值是空串（即不使用 SSL 认证）。 仅适用于 `mysql` 数据源。

这些指令用来在 `indexer` 和 MySQL 之间建立安全的 SSL 连接。关于怎样建立认证机制和设置 MySQL 服务器的信息可以参考 MySQL 文档。

示例：

```
mysql_ssl_cert = /etc/ssl/client-cert.pem
mysql_ssl_key = /etc/ssl/client-key.pem
mysql_ssl_ca = /etc/ssl/cacert.pem
```

11.1.10. odbc_dsn：ODBC 连接字符串(DSN)

要连接的 ODBC DSN。必须选项，没有默认值。 仅适用于 `odbc` 数据源。

DBC DSN（数据源名字，Data Source Name）指定了连接 ODBC 数据源时使用的认证选项（主机地址，用户名，密码等）。具体的格式与 ODBC 的具体驱动有关。

示例：

```
odbc_dsn = Driver={Oracle ODBC Driver};Dbq=myDBName;Uid=myUsername;Pwd=myPassword
```

11.1.11. sql_query_pre：待索引数据获取前查询

索引数据获取前执行的查询（pre-fetch query），或预查询（pre-query）。多值选项，可选选项，默认为一个空的查询列表。 仅适用于 SQL 数据源（`mysql`,`pgsql`,`mssql`）。

多值意思是您可以多次设置该指令，从而指定多个预查询。它们在索引数据获取查询 sql_query 之前执行，而且会严格按照在配置文件中出现的顺序执行。预查询的结果会被忽略。

预查询在很多时候有用。它们被用来设置字符集编码，标记待索引的记录，更新内部计数器，设置 SQL 服务器连接选项和变量等等。

也许预查询最常用的一个应用就是用来指定服务器返回行时使用的字符编码。这**必须**与 Sphinx 期待的编码相同（在 charset_type 和 charset_table 选项中设置）。以下是两个与 MySQL 有关的设置示例：

```
sql_query_pre = SET CHARACTER_SET_RESULTS=cp1251
sql_query_pre = SET NAMES utf8
```

对于 MySQL 数据源，在预查询中禁用查询缓冲（query cache）（仅对 indexer 连接）是有用的，因为索引查询一般并会频繁地重新运行，缓冲它们的结果是没有意义的。这可以按如下方法实现：

```
sql_query_pre = SET SESSION query_cache_type=OFF
```

示例：

```
sql_query_pre = SET NAMES utf8
sql_query_pre = SET SESSION query_cache_type=OFF
```

11.1.12. sql_query：获取待索引数据查询

获取即将索引的文档（数据）的主查询。必须的选项，无默认选项。 仅适用于 SQL 数据源（`mysql`,`pgsql`,`mssql`）。

只能有一个主查询。它被用来从 SQL 服务器获取文档（文档列表）。可以指定多达 32 个全文数据字段（严格来说是在 sphinx.h 中定义的 SPH_MAX_FIELDS 个）和任意多个属性。所有既不是文档 ID（第一列）也不是属性的列的数据会被用于建立全文索引。

文档 ID **必须**是第一列，而且**必须是唯一的正整数值（不能是 0 也不能是负数）**，既可以是 32 位的也可以是 64 位的，这要根据 Sphinx 是如何被构建的，默认情况下文档 ID 是 32 位的，但在运行 `configure` 脚本时指定`--enable-id64` 选项会打开 64 位文档 ID 和词 ID 的支持。

示例：

```
sql_query = \
SELECT id, group_id, UNIX_TIMESTAMP(date_added) AS date_added, \
title, content \
FROM documents
```

11.1.13. sql_joined_field：SQL 连接字段设置

连接/有效载荷字段获取查询。 多值选项，可选，默认值为空。 仅对 SQL 数据源有效 (`mysql`, `pgsql`, `mssql`).

`sql_joined_field` 提供两种不同的方式：连接字段，或者有效载荷（有效载荷字段）。其语法格式如下：

```
sql_joined_field = FIELD-NAME 'from' ( 'query' | 'payload-query' ); \
QUERY [ ; RANGE-QUERY ]
```

where

- FIELD-NAME 是 连接/有效载荷 字段名称;
- QUERY 是一个用于获取数据到索引的 SQL 查询.
- RANGE-QUERY 是一个可选的用于获取范围知道索引的 SQL 查询. (版本 2.0.1-beta 增加.)

Joined fields let you avoid JOIN and/or GROUP_CONCAT statements in the main document fetch query (sql_query). This can be useful when SQL-side JOIN is slow, or needs to be offloaded on Sphinx side, or simply to emulate MySQL-specific GROUP_CONCAT funcionality in case your database server does not support it.

The query must return exactly 2 columns: document ID, and text to append to a joined field. Document IDs can be duplicate, but they **must** be in ascending order. All the text rows fetched for a given ID will be concatented together, and the concatenation result will be indexed as the entire contents of a joined field. Rows will be concatenated in the order returned from the query, and separating whitespace will be inserted between them. For instance, if joined field query returns the following rows:

```
( 1, 'red' )
( 1, 'right' )
( 1, 'hand' )
( 2, 'mysql' )
( 2, 'sphinx' )
```

then the indexing results would be equivalent to that of adding a new text field with a value of ‘red right hand’ to document 1 and ‘mysql sphinx’ to document 2.

Joined fields are only indexed differently. There are no other differences between joined fields and regular text fields.

Starting with 2.0.1-beta, **ranged queries** can be used when a single query is not efficient enough or does not work because of the database driver limitations. It works similar to the ranged queries in the main indexing loop, 参见 第 3.7 节 “区段查询”. The range will be queried for and fetched upfront once, then multiple queries with different `$start` and `$end`substitutions will be run to fetch the actual data.

Payloads let you create a special field in which, instead of keyword positions, so-called user payloads are stored. Payloads are custom integer values attached to every keyword. They can then be used in search time to affect the ranking.

The payload query must return exactly 3 columns: document ID; keyword; and integer payload value. Document IDs can be duplicate, but they **must** be in ascending order. Payloads must be unsigned integers within 24-bit range, ie. from 0 to 16777215. For reference, payloads are currently internally stored as in-field keyword positions, but that is not guaranteed and might change in the future.

Currently, the only method to account for payloads is to use SPH_RANK_PROXIMITY_BM25 ranker. On indexes with payload fields, it will automatically switch to a variant that matches keywords in those fields, computes a sum of matched payloads multiplied by field wieghts, and adds that sum to the final rank.

示例：

```
sql_joined_field = \
tagstext from query; \
SELECT docid, CONCAT('tag',tagid) FROM tags ORDER BY docid ASC
```

11.1.14. sql_query_range：分区查询范围

分区查询设置。可选选项，默认为空。 仅适用于 SQL 数据源（`mysql`,`pgsql`,`mssql`）。

设置这个选项会启用文档的区段查询（参看第 3.7 节 “区段查询”）。分区段查询有助于避免在索引大量数据时发生 MyISAM 表臭名昭著的死锁问题。（同样有助于解决其他不那么声名狼藉的问题，比如大数据集上的性能下降问题，或者 InnoDB 对多个大型读事务（read transactions）进行序列化时消耗额外资源的问题。）

此选项指定的查询语句必须获取用于分区的最小和最大文档 ID。它必须返回正好两个整数字段，先是最小 ID 然后是最大 ID，字段的名字会被忽略。

当启用了分区段查询时，sql_query 要求包括`$start` 和 `$end` 宏（因为重复多次索引整个表显然是个错误）。注意，`$start` .. `$end` 所指定的区间不会重叠，因此**不会**在查询中删除 ID 正好等于`$start` 或 `$end` 的文档。第 3.7 节 “区段查询”中的例子解释了这个问题，注意大于等于或小于等于比较操作是如何被使用的。

示例：

```
sql_query_range = SELECT MIN(id),MAX(id) FROM documents
```

11.1.15. sql_range_step：分区查询步进值

区段查询的步进值。可选选项，默认为 1024。 仅适用于 SQL 数据源（`mysql`,`pgsql`,`mssql`）。

仅当启用 ranged queries 时有效。用 sql_query_range 取得的文档 ID 区间会被以这个不小的间隔步数跳跃遍历。例如，如果取得的最小和最大 ID 分别是 12 和 3456，而间隔步数是 1000，那么 indexer 会用下面这些值重复调用几次 sql_query：

- $start=12, $end=1011
- $start=1012, $end=2011
- $start=2012, $end=3011
- $start=3012, $end=3456

示例：

```
sql_range_step = 1000
```

11.1.16. sql_query_killlist：Kill-list 查询

用于得到 Kill-list 的查询。可选选项，默认为空（不设定查询）。 仅适用于 SQL 数据源（`mysql`,`pgsql`,`mssql`）。 版本 0.9.9-rc1 引入.

这个查询返回的结果集应该只有一列，每行是一个文档 ID。返回的这些文档 ID 将被存储在一个索引里。根据查询中提到的索引的顺序，一个索引的 kill-list 会抑制来自*其他*（顺序在其前面）索引的结果。这个设计的目的是要帮助用户实现在现有索引上的删除或者更新，而不用重建索引（甚至根本不用访问这个索引），尤其是为了结果解决“幽灵结果”问题。

让我们来分析一个实际的例子。假设我们有两个索引，‘main’和‘delta’。假设文档 2、3 和 5 在上一次重建索引 ‘main’的时候就被删除 了，而文档 7 和文档 11 则被更新了（即他们的文字内容发生了变化）。假设在建立索引‘main’的时候，关键字‘test’在所有这些提到的文档中都出现 了。而当我们建立索引‘delta’的时候文档 7 中也出现了关键字‘test’，但是文档 11 中不再有关键字‘test’了。现在我们重新建立索引 ‘delta’，然后以合适的顺序（较旧的排在较新的之前）对这两个索引进行检索：

```
$res = $cl->Query ( "test", "main delta" );
```

首先，我们要正确地处理删除的情况。结果集合不应该包含文档 2、3 或 5。其次，我们也要避免出现幽灵结果。如果我们不做点什么，文档 11 *就会*出现在搜索结果中。因为它会被在‘main’中查到（但在‘delta’中查不到它），并出现在最终的结果集合中，除非我们做点什么防止这种情况的发生。

Kill-list，或者缩写成 K-list 就是我们要做的。附加在‘delta’索引上的 Kill-list 会屏蔽掉前面**所有**各索引中检索到的特定行，在这个例子中，也就是‘main’中的行。因此，想要得到预期的结果，我们应该将更新了的*和*删除了的文档 ID 都放进 Kill-list。

示例：

```
sql_query_killlist = \
SELECT id FROM documents WHERE updated_ts>=@last_reindex UNION \
SELECT id FROM documents_deleted WHERE deleted_ts>=@last_reindex
```

11.1.17. sql_attr_uint：整数属性

声明无符号整数属性(attribute)。可声明同一类型的多个不同名称的属性，可选项。 仅适用于 SQL 数据源（mysql，pgsql，mssql）。

被声明的列的值必须在 32 位无符号整型可表示的范围内。超出此范围的值也会被接受，但会溢出。例如-1 会变成 2^32-1 或者说 4,294,967,295。

您可以在属性名后面附加":BITCOUNT"（见下面的示例）以便指定整型属性的位数。属性小于默认 32 位（此时称为位域）会有损性能。但它们在外部存储（extern storage）模式下可以节约内存：这些位域被组合成 32 位的块存储在.spa 属性数据文件中。如果使用内联存储（inline storage），则位宽度的设置会被忽略。

示例：

```
sql_attr_uint = group_id
sql_attr_uint = forum_id:9 # 9 bits for forum_id
```

11.1.18. sql_attr_bool：布尔属性

声明布尔属性（attribute）。可声明同一类型的多个不同名称的属性，可选项。仅适用于 SQL 数据源（mysql 和 pgsql）。 仅适用于 SQL 数据源（mysql，pgsql，mssql）。 等价于用 sql_attr_uint 声明为 1 位。

示例：

```
sql_attr_bool = is_deleted # 将被编码为 1 个 bit
```

11.1.19. sql_attr_bigint：长整型属性

64 位整数属性（attribute）声明。多个值（可以同时声明多个属性），可选选项。 仅适用于 SQL 数据源（mysql，pgsql，mssql）。注意，与 sql_attr_uint 不同，这些值是**有符号**的。于版本 0.9.9-rc1 引入。

示例：

```
sql_attr_bigint = my_bigint_id
```

11.1.20. sql_attr_timestamp：UNIX 时间戳属性

声明 UNIX 时间戳属性（attribute）。可声明同一类型的多个不同名称的属性，可选项。 仅适用于 SQL 数据源（mysql，pgsql，mssql）。

这个列的值必须是 UNIX 格式的时间戳，即 32 位无符号整数表示的自格林尼治平时 1970 年 1 月 1 日午夜起过去的秒数。时间戳在内 部是按整数值存储 和处理的。但除了将时间戳按整数使用，还可以对它们使用多种与日期相关的函数——比如时间段排序模式，或为分组（GROUP BY）抽取天/星期/月/年。注意 MySQL 中的 DATE 和 DATETIME 列类型**不能**直接作为时间戳使用，必须使用 UNIX_TIMESTAMP 函数将这些列做显式转换。

请注意 MySQL 的 DATE 或者 DATETIME 字段类型**不能**直接在 Sphinx 之中作为 timestamp 属性使用；如果这样的字段需要在 Sphinx 中进行范围过滤，在 sql_query 中你需要明确使用 MySQL 的 UNIX_TIMESTAMP 函数来转换这样的字段。

还必须注意的是 UNIX 时间戳不支持 1970 年 1 月 1 日之前的日期，在 MySQL 中使用 UNIX_TIMESTAMP()时将不会 返回预期的结果。 如果你只需要处理日期而不是时间（小时或者分钟或者秒），可以考虑使用 MySQL 的 TO_DAYS()函数（从年份 0 开始的天数）。

示例：

```
sql_attr_timestamp = UNIX_TIMESTAMP(added_datetime) AS added_ts
```

11.1.21. sql_attr_str2ordinal：字符串序数排序属性

声明字符串序数属性（属性 ）。可声明同一类型的多个不同名称的属性，可选项。 仅适用于 SQL 数据源（mysql，pgsql，mssql）。

这个属性类型（简称为字串序数）的设计是为了允许按字符串值排序，但不存储字符串本身。对字串序数做索引时，字符串值从数据库中取出、暂存、排序然后用它们在该有序数组中的序数代替它们自身，因此字串序数是个整型，对它们的大小比较与在原字串上做字典序比较结果相同。

早期版本上，对字串序数做索引可能消耗大量的 RAM。自 svn r1112 起，字串序数的积累和排序也可在固定大小的内存中解决了（代价是额外的临时磁盘空间），并受 mem_limit 设置限制。

理想中字符串可以根据字符编码和本地字符集（locale）排序。例如，如果已知字符串为 KOI8R 编码下的俄语字串，那么对字节 0xE0,0xE1 和 0xE2 排序结果应为 0xE1,0xE2 和 0xE0，因为 0xE0 在 KOI8R 中代表的字符明显应在 0xE1 和 0xE2 之后。但很不 幸，Sphinx 目前不支持这个功能，而是简单地按字节值大小排序。

请注意，这里的序号是每个索引根据自身数据计算的，因此在同时读取多个索引事实无法同时保留正确的顺序进行合并的。处理后的字符串 被替换为处理时其 在索引中的序列号，但是不同的索引具有不同的字符串集。例如，如果'main'索引包含字符串"aaa", "bbb", "ccc", 直到 "zzz"，它们将会被分别分配数值为 1,2,3,直到 26。但是'delta'如果仅包含"zzz"，则会被分配数值 1。那么在合并后，该顺序将被打 乱。不幸的是，在不存储原始字符串的情况下，这个问题无法解决（一旦存储原始字符串，序号将没有任何用处了）。

示例：

```
sql_attr_str2ordinal = author_name
```

11.1.22. sql_attr_float：浮点数属性

声明浮点型属性 attribute 。 可声明同一类型的多个不同名称的属性，可选项。 仅适用于 SQL 数据源（mysql, pgsql, mssql）。

属性值按单精度 32 位 IEEE754 格式存储。可表示的范围大约是 1e-38 到 1e+38。可精确表示的小数大约是 7 位。浮点属性的一个重要应用是存储经度和纬度值（以角度为单位），经纬度值在查询时的地理位置距离计算中有用。

示例：

```
sql_attr_float = lat_radians
sql_attr_float = long_radians
```

11.1.23. sql_attr_multi：多值属性(MVA)属性

声明多值属性（多值属性，MVA)）. 可声明同一类型的多个不同名称的属性，可选项。 仅适用于 SQL 数据源（mysql, pgsql, mssql）。

简单属性每篇文档只允许一个值。然而有很多情况（比如 tags 或者类别）需要将多个值附加给同一个属性，而且要对这个属性值列表做过滤或者分组。

声明格式如下（用反斜线只是为了清晰，您仍可以在一行之内完成声明）：

```
sql_attr_multi = ATTR-TYPE ATTR-NAME 'from' SOURCE-TYPE \
[;QUERY] \
[;RANGE-QUERY]
```

其中

- ATTR-TYPE 是 'uint' 或者 'timestamp' 之一
- SOURCE-TYPE 是 'field', 'query', 或者 'ranged-query' 之一
- QUERY 是用来取得全部（文档 ID，属性值）序对的 SQL 查询
- RANGE-QUERY 是用来取得文档 ID 的最小值与最大值的 SQL 查询，与'sql_query_range'类似

示例：

```
sql_attr_multi = uint tag from query; SELECT id, tag FROM tags
sql_attr_multi = uint tag from ranged-query; \
SELECT id, tag FROM tags WHERE id>=$start AND id<=$end; \
SELECT MIN(id), MAX(id) FROM tags
```

11.1.24. sql_attr_string：字符串属性(可返回原始文本信息)

字符串属性定义。 多值选项（即可以有一个或多个这样子的定义），可选项。仅对 SQL 数据源类型有效 (`mysql`, `pgsql`, `mssql`) 。 版本 2.0.1-beta 引入。

String attributes can store arbitrary strings attached to every document. There's a fixed size limit of 4 MB per value. Also, `searchd` will currently cache all the values in RAM, which is an additional implicit limit.

As of 1.10-beta, strings can only be used for storage and retrieval. They can not participate in expressions, be used for filtering, sorting, or grouping (ie. in WHERE, ORDER or GROUP clauses). Note that attributes declared using `sql_attr_string` will **not** be full-text indexed; you can use sql_field_string directive for that.

示例：

```
sql_attr_string = title # will be stored but will not be indexed
```

11.1.25. sql_attr_str2wordcount：文档词汇数记录属性

词汇数属性定义。多值选项（即可以有一个或多个这样子的定义），可选项。仅对 SQL 数据源类型有效 (`mysql`, `pgsql`, `mssql`) 。 版本 2.0.1-beta 引入。

Word-count attribute takes a string column, tokenizes it according to index settings, and stores the resulting number of tokens in an attribute. This number of tokens ("word count") is a normal integer that can be later used, for instance, in custom ranking expressions (boost shorter titles, help identify exact field matches, etc).

示例：

```
sql_attr_str2wordcount = title_wc
```

11.1.26. sql_column_buffers：结果行缓冲大小

每行缓冲大小。 可选项，默认值为空（自动计算大小）。 仅对 SQL 数据源类型有效 (`mysql`, `pgsql`, `mssql`) 。 版本 2.0.1-beta 引入。

ODBC and MS SQL drivers sometimes can not return the maximum actual column size to be expected. For instance, NVARCHAR(MAX) columns always report their length as 2147483647 bytes to `indexer` even though the actually used length is likely considerably less. However, the receiving buffers still need to be allocated upfront, and their sizes have to be determined. When the driver does not report the column length at all, Sphinx allocates default 1 KB buffers for each non-char column, and 1 MB buffers for each char column. Driver-reported column length also gets clamped by an upper limie of 8 MB, so in case the driver reports (almost) a 2 GB column length, it will be clamped and a 8 MB buffer will be allocated instead for that column. These hard-coded limits can be overridden using the `sql_column_buffers` directive, either in order to save memory on actually shorter columns, or overcome the 8 MB limit on actually longer columns. The directive values must be a comma-separated lists of selected column names and sizes:

```
sql_column_buffers = <colname>=<size>[K|M] [, ...]
```

示例：

```
sql_query = SELECT id, mytitle, mycontent FROM documents
sql_column_buffers = mytitle=64K, mycontent=10M
```

11.1.27. sql_field_string：字符串字段(可全文搜索，可返回原始文本信息)

组合字符串属性和全文字段定义。 Combined string attribute and full-text field declaration. 多值选项（即可以有一个或多个这样子的定义），可选项。 仅对 SQL 数据源类型有效 (`mysql`, `pgsql`,`mssql`) 。 版本 2.0.1-beta 引入。

sql_attr_string only stores the column value but does not full-text index it. In some cases it might be desired to both full-text index the column and store it as attribute. `sql_field_string` lets you do exactly that. Both the field and the attribute will be named the same.

示例：

```
sql_field_string = title # will be both indexed and stored
```

11.1.28. sql_field_str2wordcount：文档词汇数记录字段(可全文搜索，可返回原始信息)

组合词汇数属性与全文字段定义。 多值选项（即可以有一个或多个这样子的定义），可选项。 仅对 SQL 数据源类型有效 (`mysql`, `pgsql`, `mssql`) 。 版本 2.0.1-beta 引入。

sql_attr_str2wordcount only stores the column word count but does not full-text index it. In some cases it might be desired to both full-text index the column and also have the count.`sql_field_str2wordcount` lets you do exactly that. Both the field and the attribute will be named the same.

示例：

```
sql_field_str2wordcount = title # will be indexed, and counted/stored
```

11.1.29. sql_file_field：外部文件字段

实际数据存储于文件的字段定义。 仅对 SQL 数据源类型有效 (`mysql`, `pgsql`, `mssql`) 。 版本 2.0.1-beta 引入。

This directive makes `indexer` interpret field contents as a file name, and load and index the referred file. Files larger than max_file_field_buffer in size are skipped. Any errors during the file loading (IO errors, missed limits, etc) will be reported as indexing warnings and will **not** early terminate the indexing. No content will be indexed for such files.

示例：

```
sql_file_field = my_file_path # load and index files referred to by my_file_path
```

11.1.30. sql_query_post：数据获取后查询

取后查询。可选项，默认值为空。 仅适用于 SQL 数据源（`mysql`,`pgsql`,`mssql`）。

此查询在 sql_query 成功执行后立即执行。如果取后取查询产生了错误，该错误被当作警告被报告，但索引**不会**因此终止。取后查询的结果会被忽略。注意当取后查询执行时索引还**尚未**完成，而后面的索引仍然可能失败。因此在这个查询中不应进行任何永久性的更新。例如，不应在此查询中更新辅助表中存储的最近成功索引的文档 ID 值，请在后索引查询（索引后查询 sql_query_post_index ）中操作。

示例：

```
sql_query_post = DROP TABLE my_tmp_table
```

11.1.31. sql_query_post_index：数据索引后查询

索引后查询。可选项，默认值为空。 仅适用于 SQL 数据源（`mysql`,`pgsql`,`mssql`）。

此查询在索引完全成功结束后执行。如果此查询产生错误，该错误会被当作警告报告，但索引**不会**因此而终止。该查询的结果集被忽略。此查询中可以使用宏`$maxid`，它会被扩展为索引过程中实际得到的最大的文档 ID。如果没有文档被索引，则$maxid 会设置为 0。

示例：

```
sql_query_post_index = REPLACE INTO counters ( id, val ) \
VALUES ( 'max_indexed_id', $maxid )
```

11.1.32. sql_ranged_throttle：分区查询间隔时间

分区查询的间隔时间（throttling），单位是毫秒。可选选项，默认值为 0（无间隔时间）。 仅适用于 SQL 数据源（`mysql`,`pgsql`,`mssql`）。

此选项旨在避免 indexer 对数据库服务器构成了太大的负担。它会使 indexer 在每个分区查询的步之后休眠若干毫秒。休眠无条件执行，并在取结果的查询之前执行。

示例：

```
sql_ranged_throttle = 1000 # sleep for 1 sec before each query step
```

11.1.33. sql_query_info_pre：命令行信息获取前查询

命令行查询前查询。 可选选项，默认为空。 仅对 `mysql` 数据源有效。

仅被命令行搜索所用，用来在命令行查询之前执行查询，一般用于设置查询的字符集编码

示例：

```
sql_query_info_pre = SET NAMES utf8
```

11.1.34. sql_query_info：命令行信息获取查询

文档信息查询。 可选选项，默认为空。 仅对 `mysql` 数据源有效。

仅被命令行搜索所用，用来获取和显示文档信息，目前仅对 MySQL 有效，且仅用于调试目的。此查询为每个文档 ID 获取 CLI 搜索工具要显示的文档信息。 它需要包含`$id` 宏，以此来对应到查询的文档的 ID。

示例：

```
sql_query_info = SELECT * FROM documents WHERE id=$id
```

11.1.35. xmlpipe_command：数据获取命令

调用 xmlpipe 流提供者的 Shell 命令。必须选项。仅对 `xmlpipe` 和 `xmlpipe2` 数据源有效。

指定的命令会被运行，其输出被当作 XML 文档解析。具体格式描述请参考 第 3.8 节 “xmlpipe 数据源” 和 第 3.9 节 “xmlpipe2 数据源” 。

示例：

```
xmlpipe_command = cat /home/sphinx/test.xml
```

11.1.36. xmlpipe_field：字段设置

声明 xmlpipe 数据字段。可声明同一类型的多个不同名称的属性，可选项。仅对 `xmlpipe2` 数据源有效。参考 第 3.9 节 “xmlpipe2 数据源”.

示例：

```
xmlpipe_field = subject
xmlpipe_field = content
```

11.1.37. xmlpipe_field_string：字符串字段

xmlpipe 字段和字符串属性定义。 多值，可选项。 仅对 `xmlpipe2` 数据源来性有效。参见 第 3.9 节 “xmlpipe2 数据源”. 版本 2.0.1-beta 引入。

Makes the specified XML element indexed as both a full-text field and a string attribute. Equivalent to <sphinx:field name=”field” attr=”string”/> declaration within the XML file.

示例：

```
xmlpipe_field_string = subject
```

11.1.38. xmlpipe_field_wordcount：词汇数存储字段

xmlpipe 字段和词汇数属性定义。 多值，可选项。 仅对 `xmlpipe2` 数据源来性有效。参见 第 3.9 节 “xmlpipe2 数据源”. 版本 2.0.1-beta 引入。

Makes the specified XML element indexed as both a full-text field and a word count attribute. Equivalent to <sphinx:field name=”field” attr=”wordcount”/> declaration within the XML file.

示例：

```
xmlpipe_field_wordcount = subject
```

11.1.39. xmlpipe_attr_uint：整数属性

声明 xmlpipe 整型属性。可声明同一类型的多个不同名称的属性，可选项。仅对 `xmlpipe2` 数据源有效。语法与 sql_attr_uint 相同。

示例：

```
xmlpipe_attr_uint = author
```

11.1.40. xmlpipe_attr_bool：布尔属性

声明 xmlpipe 布尔型属性。可声明同一类型的多个不同名称的属性，可选项。 仅对 `xmlpipe2` 数据源有效。语法与 sql_attr_bool 相同。

示例：

```
xmlpipe_attr_bool = is_deleted # will be packed to 1 bit
```

11.1.41. xmlpipe_attr_timestamp：UNIX 时间戳属性

声明 xmlpipe UNIX 时间戳属性。可声明同一类型的多个不同名称的属性，可选项。 仅对 `xmlpipe2` 数据源有效。语法与 sql_attr_timestamp 相同。

示例：

```
xmlpipe_attr_timestamp = published
```

11.1.42. xmlpipe_attr_str2ordinal：字符串序列属性

声明 xmlpipe 字符序数属性。可声明同一类型的多个不同名称的属性，可选项。 仅对 `xmlpipe2` 数据源有效。语法与 sql_attr_str2ordinal 相同。

示例：

```
xmlpipe_attr_str2ordinal = author_sort
```

11.1.43. xmlpipe_attr_float：浮点数属性

声明 xmlpipe 浮点型属性。可声明同一类型的多个不同名称的属性，可选项。 仅对 `xmlpipe2` 数据源有效。语法与 sql_attr_float 相同。

示例：

```
xmlpipe_attr_float = lat_radians
xmlpipe_attr_float = long_radians
```

11.1.44. xmlpipe_attr_multi：多值属性

声明 xmlpipe MVA 属性。可声明同一类型的多个不同名称的属性，可选项。 仅对 `xmlpipe2` 数据源有效。

这个选项为 xmlpipe2 流声明一个 MVA 属性标签。该标签的内容会被试图解析成一个整型值的列表（数组），此列表构成一个 MVA 属性值，这与把 MVA 属性的数据源设置为“字段”时 sql_attr_multi 分析 SQL 列内容的方式类似。

示例：

```
xmlpipe_attr_multi = taglist
```

11.1.45. xmlpipe_attr_string：字符串属性

xmlpipe 字符串定义。 多值，可选项。 Applies to `xmlpipe2` source type only. 版本 2.0.1-beta 引入。

This setting declares a string attribute tag in xmlpipe2 stream. The contents of the specified tag will be parsed and stored as a string value.

示例：

```
xmlpipe_attr_string = subject
```

11.1.46. xmlpipe_fixup_utf8：UTF-8 修复设置

在 Sphinx 端进行 UTF-8 验证和过滤，防止 XML 分析器因为碰上非 UTF-8 文档而犯疯卡死。可选选项，默认值为 0。只适用于 xmlpipe2 数据源。 仅对 `xmlpipe2` 数据源有效。

在某些特定情况下很难甚至不可能保障输入的 xmlpipe2 文档体都是完美有效一致的 UTF-8 编码。例如，输入流中可能溜进一些 特定国家的单字节 编码文本。Libexpat 这个 XML 分析器非常脆弱，遇到这种情况就会停止工作。UTF-8 修复（UTF-8 fixup）功能能让这种情况得到避免。当启动了修复选项的时候，Sphinx 会对输入留进行预处理，其后再传给 XML 分析器，其间非法的 UTF-8 序列 全部被替换成空格。

示例：

```
xmlpipe_fixup_utf8 = 1
```

11.1.47. mssql_winauth：Windows 集成认证

标志位，代表是否使用 MS SQL 的 windows 身份认证。布尔型，可选选项，默认值是 0（假）。只适用于 `mssql` 数据源。于版本 0.9.9-rc1 引入。

这个选项指出在连接到 MS SQL Server 时时候使用现在正登录的 windows 账户的凭据作来进行身份验证。注意当 `searchd` 作为服务运行时，账户的用户名可能与安装这个服务的账户不同。

示例：

```
mssql_winauth = 1
```

11.1.48. mssql_unicode：Unicode 设置

MS SQL 编码类型标志位。布尔型，可选选项，默认值是 0（假）。只适用于 `mssql` 数据源。于版本 0.9.9-rc1 引入。

这个选项指出在对 MS SQL Server 进行查询时，是使用 Unicode 还是单字节数据。这个标志位**必须**与 charset_type 指令同步，也就是说，要索引 Unicode 数据，则既要设置 charset_type 选项（设为‘utf-8’），又要设置数据源的 `mssql_unicode` 选项（设为 1）。多说一句：MS SQL 实质上返回 UCS-2 编码的数据，而不是 UTF-8，但 Sphinx 可以自动处理这个情况。

示例：

```
mssql_unicode = 1
```

11.1.49. unpack_zlib：SQL 数据源解压字段设置

使用 zlib（即 gnuzip）来解压（unpack，deflate）的列。多个值，可选选项，默认值是列的空列表。仅适用于 SQL 数据源（`mysql`,`pgsql`,`mssql`）。于版本 0.9.9-rc1 引入。

`indexer` 会使用标准 zlib 算法（称作 deflate，`gunzip` 也实现了这个算法）对这个选项指定的那些列进行解压缩。当建立索引的动作发生在数据库所在机器以外的机器时，这个选项会降低数据库的负载，并节约网络带宽。要想使用这个特性，就必须保证在建立时 zlib 和 zlib-devel 都是可用的。

示例：

```
unpack_zlib = col1
unpack_zlib = col2
```

11.1.50. unpack_mysqlcompress：MySQL 数据源解压字段设置

使用 MySQL UNCOMPRESS()算法解压的列。多个值，可选选项，默认值是列的空列表。仅适用于 SQL 数据源（`mysql`,`pgsql`,`mssql`）。于版本 0.9.9-rc1 引入。

`indexer` 会使用 MySQL COMPRESS()和 UNCOMPRESS()使用的修改过的 zlib 算法对这个选项指定的那些列进行解压缩。 当建立索引的动作发生在数据库所在机器以外的机器时，这个选项会降低数据库的负载，并节约网络带宽。要想使用这个特性，就必须保证在建立时 zlib 和 zlib-devel 都是可用的。

示例：

```
unpack_mysqlcompress = body_compressed
unpack_mysqlcompress = description_compressed
```

11.1.51. unpack_mysqlcompress_maxsize：MySQL 数据源解压缓冲区设置

用于 UNCOMPRESS()解压后数据的缓冲区大小。可选选项，默认值是 16M。于版本 0.9.9-rc1 引入。

当使用 unpack_mysqlcompress 选项时，由于实现方法本质上的限制，不可能减小缓冲区大小方面的需求。因此缓冲区必须预先分配好，解压出的数据也不能超过缓冲区大小。这个选项控制这个缓冲区大小值，既可以用于限制 `indexer` 的内存使用，也可以用于在需要的时候使解压非常大的数据变为可能。

示例：

```
unpack_mysqlcompress_maxsize = 1M
```

11.2. 索引配置选项

11.2.1. type：索引类型设置

索引类型。可用的值包括 plain（普通本地索引）、distributed（分布式）或 rt（实时索引）。可选选项，默认值为 plain（索引为普通本地索引）。

Sphinx 支持几种不同的索引类型。 版本 0.9.x 支持两种索引类型：普通本地索引——在本机上存储和处理，和分布式索引——不仅涉及本地搜索，而且同时通过网络向远程 `searchd` 实例做查询（查看第 5.8 节 “分布式搜索”了解详情）。 版本 1.10-beta 也增加了被称为实时索引（简称为 RT 索引）的索引类型，它也支持在本机上存储和处理，除此之外还支持全文索引数据的即时更新（查看第 4 章 RT 实时索引了解详情）。 需要提醒的是 *attributes* 值在普通本地索引或者 RT 索引中都可以被即时更新。

你可以根据需要设置索引内行。索引默认是普通本地索引。

示例：

```
type = distributed
```

11.2.2. source：文档源

向本地索引增加文档源。可以出现多次，必须选项。

为当前索引指定一个从中可以获取文档的文档源。必须至少有一个文档源。可以有多个文档源，任何数据源类型都可接受：即您可以从 MySQL 服务器中获取一部分数据，从 PostgreSQL 中获取另一部分，再在文件系统上使用 xmlpipe2 获取一部分。

然而，对源数据却有一些限制。首先，文档 ID 必须在所有源的总体上是唯一的。如果这个条件不满足，那可能导致非预期的搜索结果。其次，源的模式必须相同，以便在同一个索引中存储。

数据来源的 ID 不会被自动存储。因此，为了获知匹配的文档是从哪个数据源中来的，需要手工存储一些额外的信息。通常有两种方法：

1. 修改文档 ID，将源 ID 编码进去：

```
source src1
{
sql_query = SELECT id*10+1, ... FROM table1
...
}

source src2
{
sql_query = SELECT id*10+2, ... FROM table2
...
}
```

2.
3. 将数据来源存储为一个属性：

```
source src1
{
```

```
sql_query = SELECT id, 1 AS source_id FROM table1
sql_attr_uint = source_id
...
}

source src2
{
sql_query = SELECT id, 2 AS source_id FROM table2
sql_attr_uint = source_id
...
}
```

4.

示例：

```
source = srcpart1
source = srcpart2
source = srcpart3
```

11.2.3. path：索引文件路径

索引文件的路径和文件名（不包括扩展名）。必须选项。

path 既包括文件夹也包括文件名，但不包括扩展名。`indexer` 在产生永久和临时索引文件的最终名字时会附加上不同的扩展名。永久数据文件有几个不同的扩展名，都以“.sp”开头，临时文件的扩展名以“.tmp”开头。如果 indexer 没有成功地自动删除`.tmp*`文件，手工删除是安全的。

以下是不同索引文件所存储的数据种类，供参考：

- `.spa` 存储文档属性（仅在 extern 文档信息存储模式中使用）；
- `.spd` 存储每个词 ID 可匹配的文档 ID 列表；
- `.sph` 存储索引头信息；
- `.spi` 存储词列表（词 ID 和指向`.spd` 文件的指针）；
- `.spk` 存储 kill-lists;
- `.spm` 存储 MVA 数据；
- `.spp` 存储每个词 ID 的命中（或者说记账，或者词的出现）列表；
- `.sps` 存储字符串属性数据.

示例：

```
path = /var/data/test1
```

11.2.4. docinfo：文档信息存储模式

文档信息(docinfo)的存储模式。可选选项，默认是“extern”，可用的值包括’none’, ‘extern’ 和 ‘inline’.。

此选项确切定义了文档信息在磁盘和 RAM 中的物理存储方式。“none”意思是根本不存储文档信息（没有任何属性）。通常并不需要 显式设置为 “none”因为当没有配置任何属性时 Sphinx 会自动选择“none”。“inline”代表文档信息与文档 ID 列表一同存储在在`.spd` 文件中。“extern”代表文档信息与文档 ID 分开（在外部）存储（在`.spa` 文件中）。

基本上，外部存储的文档信息在查询时必须保持在内存中。这是性能的原因。因此有时候“inline”是唯一的选择。然而这种情况并不多见，文档信息默认是“extern”存储的。深入的探讨和 RAM 使用的估计请参见第 3.2 节 “属性”。

示例：

```
docinfo = inline
```

11.2.5. mlock：缓冲数据内存锁定

已缓冲数据的内存锁定。可选选项，默认为 0（不调用 mlock()）

为提高性能，searchd 将.spa 和.spi 文件预取到内存中，并一直在内存中保存它们的拷贝。但如果一段时间内没有对该索引的搜索，则对这份缓冲的拷贝没有内存访问，而操作系统可能会决定将它交换到磁盘上去。对这些“冷却”了的索引的访问会导致其交换回内存并得到一个较大的延迟。

将 mlock 选项设置为 1 会使 Sphinx 使用 mlock(2)系统调用将存储上述缓冲了的数据的系统内存锁定，这将避免内存交换（详情参见 man 2 mlock）。mlock(2)是特权调用，因此可能需要 searchd 以 root 账户运行或通过其他办法赋予足够的权限。如果 mlock()失败会发出警告，但索引会继续进行。

示例：

```
mlock = 1
```

11.2.6. morphology：词形处理

词形处理器的列表。可选选项，默认为空（不使用任何词形处理器）。

词形处理器可以将待索引的词从各种形态变成基本的规则的形态。例如，英语词干提取器（English stemmer）可以将“dogs”和“dog”都变成“dog”，这使搜索这两个词的结果都相同。

内置的词形处理器包括英语词干提取器，俄语词干提取器（支持 UTF-8 和 Windows-1251 编码），Soundex 和 Metaphone。后面两个会将词替换成特殊的语音编码，这会使发音相近的词表示形式相同。Snowball 项目的 libstemmer 库提供的额外词干提取器可以通过在编译期对 configure 脚本使用--with-libstemmer 选 项来启用。内建的英语和俄语词干提取器要比它们在 libstemmer 中的对应物模块运行更快，但它们的结果可能略有不同，因为内建的版本基于较旧的版 本。Metaphone 基于 Double Metaphone 算法实现。Metaphone 是 基于 Double Metaphone 的算法和索引的主要代码实现的。

在 morphology 选项中可使用的内建值包括

- none – 不做任何词形处理，保持原样;
- stem_en – 应用实现的英语词干;
- stem_ru – 应用实现的俄语词干;
- stem_enru – 应用实现的英语和俄语词干;
- stem_cz – 应用实现的捷克语词干;
- soundex – 用关键字的 SOUNDEX 编码取代它;
- metaphone – 用关键字的 METAPHONE 编码取代它.

libstemmer 提供的额外值格式为“libstemmer_XXX”，XXX 指 libstemmer 算法的代号。（完整列表参见 libstemmer_c/libstemmer/modules.txt）

可以指定多个词干提取器（以逗号分隔）。这些提取器按列出的顺序应用于输入词串，整个处理会在第一个真正修改了原词的词干提取器之后停止。另外，当 wordforms 特性启用时，词会现在词形字典中查询，如果词典中有对应的条目，那么词干提取器干脆不会被使用。换个说法，wordforms 选项可以用来补充指定词干提取器的例外情况。

示例：

```
morphology = stem_en, libstemmer_sv
```

11.2.7. dict：关键字词典类型

关键字词典的类型。 可用值为 crc 或者 keywords。 可选项，默认为 crc。 版本 2.0.1-beta 引入。

The default dictionary type in Sphinx, and the only one available until version 2.0.1-beta, is a so-called CRC dictionary which never stores the original keyword text in the index. Instead, keywords are replaced with their control sum value (either CRC32 or FNV64, depending whether Sphinx was built with --enable-id64) both when searching and indexing, and that value is used internally in the index.

That approach has two drawbacks. First, in CRC32 case there is a chance of control sum collision between several pairs of different keywords, growing quadratically with the number of unique keywords in the index. (FNV64 case is unaffected in practice, as a chance of a single FNV64 collision in a dictionary of 1 billion entries is approximately 1:16, or 6.25 percent. And most dictionaries will be much more compact that a billion keywords, as a typical spoken human language has in the region of 1 to 10 million word forms.) Second, and more importantly, substring searches are not directly possible with control sums. Sphinx alleviated that by pre-indexing all the possible substrings as separate keywords (参见 第 11.2.18 节 “min_prefix_len：最小索引前缀长度”, 第 11.2.19 节 “min_infix_len：最小索引中缀长度” directives). That actually has an added benefit of matching substrings in the quickest way possible. But at the same time pre-indexing all substrings grows the index size a lot (factors of 3-10x and even more would not be unusual) and impacts the indexing time respectively, rendering substring searches on big indexes rather impractical.

Keywords dictionary, introduced in 2.0.1-beta, fixes both these drawbacks. It stores the keywords in the index and performs search-time wildcard expansion. For example, a search for a 'test*' prefix could internally expand to 'test|tests|testing' query based on the dictionary contents. That expansion is fully transparent to the application, except that the separate per-keyword statistics for all the actually matched keywords would now also be reported.

Indexing with keywords dictionary should be 1.1x to 1.3x slower compared to regular, non-substring indexing – but times faster compared to substring indexing (either prefix or infix). Index size should only be slightly bigger that than of the regular non-substring index, with a 1..10% percent total difference Regular keyword searching time must be very close or identical across all three discussed index kinds (CRC non-substring, CRC substring, keywords). Substring searching time can vary greatly depending on how many actual keywords match the given substring (in other words, into how many keywords does the search term expand). The maximum number of keywords matched is restricted by the expansion_limitdirective.

Essentially, keywords and CRC dictionaries represent the two different trade-off substring searching decisions. You can choose to either sacrifice indexing time and index size in favor of top-speed worst-case searches (CRC dictionary), or only slightly impact indexing time but sacrifice worst-case searching time when the prefix expands into very many keywords (keywords dictionary).

示例：

```
dict = keywords
```

11.2.8. index_sp：索引句子和段落信息

是否检测并索引句子和段落边界。 可选值，默认为 0（不检测和索引）。 版本 2.0.1-beta 引入。

This directive enables sentence and paragraph boundary indexing. It's required for the SENTENCE and PARAGRAPH operators to work. Sentence boundary detection is based on plain text analysis, so you only need to set `index_sp = 1` to enable it. Paragraph detection is however based on HTML markup, and happens in the HTML stripper. So to index paragraph locations you also need to enable the stripper by specifying `html_strip = 1`. Both types of boundaries are detected based on a few built-in rules enumerated just below.

Sentence boundary detection rules are as follows.

- Question and excalamation signs (? and !) are always a sentence boundary.
- Trailing dot (.) is a sentence boundary, except:
    - When followed by a letter. That's considered a part of an abbreviation (as in "S.T.A.L.K.E.R" or "Goldman Sachs S.p.A.").
    - When followed by a comma. That's considered an abbreviation followed by a comma (as in "Telecom Italia S.p.A., founded in 1994″).
    - When followed by a space and a small letter. That's considered an abbreviation within a sentence (as in "News Corp. announced in Februrary").
    - When preceded by a space and a capital letter, and followed by a space. That's considered a middle initial (as in "John D. Doe").

Paragraph boundaries are inserted at every block-level HTML tag. Namely, those are (as taken from HTML 4 standard) ADDRESS, BLOCKQUOTE, CAPTION, CENTER, DD, DIV, DL, DT, H1, H2, H3, H4, H5, LI, MENU, OL, P, PRE, TABLE, TBODY, TD, TFOOT, TH, THEAD, TR, and UL.

Both sentences and paragraphs increment the keyword position counter by 1.

示例：

```
index_sp = 1
```

11.2.9. index_zones：索引标签区域信息

字段内需要索引的 HTML/XML 区域的标签列表。 可选项，默认为空（不索引区域）。 版本 2.0.1-beta 引入。

Zones can be formally defined as follows. Everything between an opening and a matching closing tag is called a span, and the aggregate of all spans corresponding sharing the same tag name is called a zone. For instance, everything between the occurrences of <H1> and </H1> in the document field belongs to H1 zone.

Zone indexing, enabled by `index_zones` directive, is an optional extension of the HTML stripper. So it will also require that the stripper is enabled (with `html_strip = 1`). The value of the`index_zones` should be a comma-separated list of those tag names and wildcards (ending with a star) that should be indexed as zones.

Zones can nest and overlap arbitrarily. The only requirement is that every opening tag has a matching tag. You can also have an arbitrary number of both zones (as in unique zone names, such as H1) and spans (all the occurrences of those H1 tags) in a document. Once indexed, zones can then be used for matching with the ZONE operator, 参见 第 5.3 节 “扩展查询语法”.

示例：

```
index_zones = h*, th, title
```

11.2.10. min_stemming_len：词干化最小词长

启用词干化的最小词长。可选选项，默认为 1（对任何词都进行词干化）。于版本 0.9.9-rc1 引入。

词干化方法并不完美，有时会产生用户不想要的结果。例如，如果用英语的 Porter stemmer 词干化算法处理关键词“gps”，得到的结果是“gp”，显然这是不对的。`min_stemming_len` 这个特性允许根据词的长度来决定是否跳过词干化，即不对较短的词进行词干化。注意，词长等于这个选项设置的值的词**会**被词干化。因此要避免对 3 个字符长的关键词进行词干化，必须指定这个选项的值为 4。要活的更细粒度的控制，请参考 wordforms 这个特性。

示例：

```
min_stemming_len = 4
```

11.2.11. stopwords：停止词

停用词文件列表（空格分隔）。可选选项，默认为空。

停用词是不被索引的词。停用词表一般包括最常用的高频词，因为它们对搜索结果没有多大帮助却消耗很多处理资源。

可以指定多个文件名，用空格分隔。所有文件都会被载入。停用词文件的格式是简单的纯文本。其编码必须与 charset_type 选项所指定的索引编码相匹配。文件数据会根据 charset_type 选项的设置进行切分，因此您可以使用与待索引数据相同的分隔符。词干提取器（stemmers）也会在停用词文件的分析中使用。

尽管停用词不会被索引，它们却影响关键词的位置。例如，假设“the”是一个停用词，文档 1 包含一行“in office”，而文档 2 包含“in the office”。将“in office”作为确切词组搜索则只会得到文档 1，虽然文档 2 里的 the 是停用的。

示例：

```
stopwords = /usr/local/sphinx/data/stopwords.txt
stopwords = stopwords-ru.txt stopwords-en.txt
```

11.2.12. wordforms：词形字典

词形字典。 可选选项，默认为空。

词形字典在输入文档根据 charset_table 切 碎后使用。本质上，它使您可以将一个词替换成另一个。这通常被用来将不同的词形变成一个单一的标准形式（即将词的各种形态如 “walks”，“walked”，“walking”变为标准形式“walk”）。也可以用来实现取词根的例外情况，因为词形字典中可以找到的词不会经 过词干提取器的处理。

索引和搜索中的输入词都会利用词典做规则化。因此要使词形字典的更改起作用，需要重新索引并重启 `searchd`。

Sphnix 的词形支持被设计成可以很好地支持很大的字典。它们轻微地影响索引速度：例如，1M 个条目的字典会使索引速度下降 1.5 倍。搜索速度则 完全不受影响。额外的内存占用大体上等于字典文件的大小，而且字典是被多个索引共享的，即如果一个 50MB 的词形字典文件被 10 个不同的索引使用了，那么 额外的 `searchd` 内存占用就是大约 50MB。

字典文件的格式是简单的纯文本。每行包括一个源词形和一个目标词形，编码应与 charset_type 选项所指定的完全相同，二者用大于号分隔。文件载入时会经过 charset_table 选项指定的规则的处理。因此基本上在大小写是否敏感这个问题上它是与待索引的全文数据相同的，即通常是大小写无关的。一下是个文件内容的例子：

```
walks > walk
walked > walk
walking > walk
```

我们提供了一个 `spelldump` 工具，它可以帮您从 `ispell` 和 `MySpell`（OpenOffice 提供）格式的 `.dict` 和 `.aff` 字典文件生成 Sphinix 可接受的格式。

从版本 0.9.9-rc1 开始，可以将好几个源词对应到同一个目标词上。由于这个过程作用于符号化之后的结果而不是原始文本，空白字符和标记语言都被忽略。

```
core 2 duo > c2d
e6600 > c2d
core 2duo > c2d
```

Notice however that the *destination* wordforms are still always interpreted as a *single* keyword! Having a mapping like “St John > Saint John” will result in **not** matching “St John” when searching for “Saint” or “John”, because the destination keyword will be “Saint John” with a space character in it (and it’s barely possible to input a destination keyword with a space).

示例：

```
wordforms = /usr/local/sphinx/data/wordforms.txt
```

11.2.13. exceptions：词汇特例处理

Token 特例文件。 可选选项，默认为空。

对于使用 Coreseek 的中文用户，这一选项无效。Coreseek 为 Sphinx 贡献的中文分词法内置了 Token 特例化支持，具体参阅 Coreseek MMSeg 分词法的文档。不过，值得高兴的是，Token 特例化的文件格式两者是同样的。

此选项允许将一个或多个 Token（Token 中，可以包括在正常情况下会被过滤的字符）映射成一个单独的关键词。exceptions 选项与 wordforms 选项很类似，它们都代表某种映射，但有一些重要的不同点。

这些不同点简要总结如下：

- exceptions 大小写敏感, wordforms 大小写无关；
- exceptions 允许检测一串记号， wordforms 仅对单独的词有效；
- exceptions 可以使用 charset_table 中**没有**的特殊符号，wordforms 完全遵从 charset_table；
- exceptions 在大字典上性能会下降，wordforms 则对百万级的条目应对自如。

输入文件的格式仍然是纯文本，每行一个分词例外，而行的格式如下：

```
map-from-tokens => map-to-token
```

示例文件：

```
AT & T => AT&T
AT&T => AT&T
Standarten  Fuehrer => standartenfuhrer
Standarten Fuhrer => standartenfuhrer
MS Windows => ms windows
Microsoft Windows => ms windows
C++ => cplusplus
c++ => cplusplus
C plus plus => cplusplus
```

这里全部的符号都是大小写敏感的：它们**不会**按 charset_table 选项的规则处理。因此，在上述例外文件下，“At&t”会被分成两个关键字“at”和“t”，因为其中的小写字母。而“AT&T”却被精确匹配并产生一个单独的关键字“AT&T”。

需要注意的是，前述映射文件的目标关键词（右侧）a)总是被解释成一个*单独*的词，而且 b)不仅是大小写敏感的，而且是空白符号敏感的！在上述样例中，查询“ms windows”*不会*匹配包含“MS Windows”的文档。这个查询会被解释成两个词“ms”和“Windows”。而“MS Windows”映射到的是一个*单独*的 关键字“ms windows”，包括中间的空格。另一方面“standartenfuhrer”会取回带有“Standarten Fuhrer”或者“Standarten Fuehrer”内容的文档（大写字母必须与此处列出的完全相同），或者关键词本身大小写随意的任何版本，例如“staNdarTenfUhreR”。 （然而“standarten fuhrer”不会匹配。这段文本无法与列出的任何一个例外相匹配，因为大小写不同。因此被索引为两个分开的关键字）

映射源部分的空白符（white space）不会被忽略，但空白符的数量无所谓。任何数量的空白符都匹配已索引的文档或者查询中的任意数量的空白符。例如映射源部分（“=>”左 端）的“AT€&€T”可以匹配“AT€€&€T”，不管被映射部分或已索引全文数据中实际有几个空格。根据上述例子中的第一条，上述文 本会作为“AT&T”关键字被索引。 *对于使用 Coreseek 的中文用户，这个特性目前尚不被支持。Coreseek 将在后续版本支持这个特性。*

exceptions 选项也允许特殊字符（这是通用 charset_table 选项规则的例外（exception），此选项因而得名）。假设您一般不想把“+”当作有效的字符，但仍想搜索一些例外情况，比如“C++”。上述例子正好可以做到这点，完全不管哪些字符在表中，哪些字符不在。

Exceptions 选项被应用于原始输入文档和索引、搜索时的查询数据。因此要使文件的改动生效，需要重建索引并重启 searchd。

示例：

```
exceptions = /usr/local/sphinx/data/exceptions.txt
```

11.2.14. min_word_len：最小索引词汇长度

最小索引词长度。可选选项，默认为 1（索引任何词）

只有长度不小于这个最小索引词长度的词会被索引。例如，如果 min_word_len 为 4，那么“the”这个词不会被索引，但“they”会。

示例：

```
min_word_len = 4
```

11.2.15. charset_type：字符集编码

字符集编码类型。可选选项，默认为“sbcs”。可用的值包括“sbcs”和“utf-8”。 *对于使用 Coreseek 的中文用户，可选的值还可以有“zh_cn.utf-8 ”、“zh_cn.gbk”和“zh_cn.big5”（需要编译时提供 iconv 支持）。当设置 charset_type 值为上面的值时，系统默认您开启了中文分词特性。*

不同的编码将它们的内部字符代码映射到特殊字节序列的方法不同。目前两个最常见的方法是单字节编码和 UTF-8。它们对应的 charset_type 值分别是“sbcs”（代表 Single Byte Character Set 单字节字符集）和“utf-8”。选定的编码类型会在搜索被使用的任何情况下使用：索引数据时，对索引查询时，产生摘要时，等等。

注意，尽管“utf-8”暗示解码出来的值应按 unicode 码点数值对待，“sbcs”却对应一系列不同的编码，它们对不同字节值的处理不同，这要在 charset_table 设置中正确地反应出来。例如，同一个值 244（十六进制 0xE0）根据使用的是 koi-8r 还是 windows-1251 编码而映射到不同的俄语字符。

示例：

```
charset_type = utf-8
```

11.2.16. charset_table：字符表和大小写转换规则

接受的字符表和大小写转换规则。可选选项，默认值与 charset_type 选项的值有关。*对于使用 Coreseek 的中文用户，Coreseek 提供的 MMseg 分词法内置了可接受的字符表，并且用户不可修改。当启用分词功能时，自动开启。*

charset_table 频繁应用于 Sphinx 的分词过程，即从文档文本或查询文本中抽取关键字的过程。它控制哪些字符被当作有效字符接受，哪些相反，还有接受了的字符如何转换（例如大小写信息保留还是去除）。

可以把 charset_table 想成一个对超过 100K 个 Unicode 字符中每一个的映射关系的大表（或者一个 256 个字符的 小表，如果你使用 SBCS）。默认每个字符都对应 0，这表示它不在关键字中出现，应被视为分隔符。一旦在此表中被提及，字符就映射到另一个字符（通常是它自身或者自身的小 写版本），同时被当作一个可以出现在关键字中的有效字符。

值的格式是逗号分隔的映射列表。两种最简单的映射分别是声明一个字符为有效和将一个简单字符映射为另一个字符。但用这种格式指定整个表会导致其体积臃肿、无法管理。因此提供了一些语法上的快捷方式，用它们可以一次指定一定范围的字符。详细的列表如下：

A->a
: 单个字符映射，声明源字符“A”为允许出现在关键字中的字符，并将之映射到目的字符“a”（这并 *没有*声明“a”是允许的）。

A..Z->a..z
: 范围映射，声明源范围中的全部字符允许出现在关键字中，并将它们映射到目的范围。 *并不*声明目的范围是允许的。此外会检查 （长度必须相等）

a
: 单一字符映射，声明一个允许的字符，将它映射到它自身。相当于单个字符映射 a->a 。

a..z
: 杂散范围映射，声明范围中的全部字符为允许，将它们映射到自身。相当于范围映射 a..z->a..z 。

A..Z/2

棋盘范围映射。映射每相邻两个字符到其中的第二个。形式化地说，声明范围中的奇数字符为允许，将它们映射到偶数字符上。同时允许偶数字符并映射到 其自身。例如， A..Z/2 相当于 A->B, B->B, C->D, D->D, …, Y->Z, Z->Z。这个映射接将捷径方便声明大小写字符交替而非大小写字符分别连续的 Unicode 块。

编码为 0 到 31 之间的控制字符总是被视作分隔符。编码 32 到 127 的字符即 7 位 ASCII 字符可以原样使用在映射中。为避免配置文 件的编码问题，8 位 ASCII 字符和 Unicode 字符必须以 U+xxx 形式指定，“xxx”是码点对应的十六进制数。也可以用这个形式编码 7 位 ASCII 编码中的特殊字 符，例如用 U+20 来编码空格符，U+2E 来编码句点，U+2C 来编码逗号。

示例：

```
# 'sbcs' defaults for English and Russian
charset_table = 0..9, A..Z->a..z, _, a..z, \
U+A8->U+B8, U+B8, U+C0..U+DF->U+E0..U+FF, U+E0..U+FF

# 'utf-8' defaults for English and Russian
charset_table = 0..9, A..Z->a..z, _, a..z, \
U+410..U+42F->U+430..U+44F, U+430..U+44F
```

11.2.17. ignore_chars：忽略字符表

忽略字符表。 可选选项，默认为空。

有些字符，如软断字符(U+00AD)，不是仅仅要当作分隔符，而且应该被完全忽略。例如，如果“-”只是不在 charset_table 里，那么 “abc-def”会被当作两个关键字“abc”和“def”来索引。相反，如果将“-”加到 ignore_char 列表中，那么相同的文本会被当作一个 单独的关键字“abcdef”索引。

此选项的语法与 charset_table 相同，但只允许声明字符，不允许映射它们。另外，忽略的字符不能出现在 charset_table 里。

示例：

```
ignore_chars = U+AD
```

11.2.18. min_prefix_len：最小索引前缀长度

索引的最小前缀长度。可选选项，默认为 0（不索引前缀）。

前缀索引使实现“wordstart*”形式的通配符成为可能（通配符语法的细节请参考 enable_star 选项）。当最小前缀长度被设置为正值，indexer 除了关键字本身还会索引所有可能的前缀（即词的开头部分）。太短的前缀（小于允许的最小值）不会被索引。

例如，在 min_prefix_len=3 设置下索引关键字“example”会导致产生 5 个索引项“exa”, “exam”, “examp”, “exampl”和该词本身。对这个索引搜索“exam”会得到包含“example”的文档，即使该文档中没有“exam”自身。然而，前缀索引会使索 引体积急剧增大（因为待索引关键字增多了很多），而且索引和搜索的时间皆会恶化。

在前缀索引中没有自动的办法可以提高精确匹配（整个词完全匹配）的评分，但有一些技巧可以实现这个功能。首先，可以建立两个索引，一个带有前缀索引，另一个没有，同时在这两个索引中搜索，然后用 SetIndexWeights() 来设置二者的权重。其次，可以启用星号语法并重写扩展模式的查询：

```
# in sphinx.conf
enable_star = 1

// in query
$cl->Query ( "( keyword | keyword* ) other keywords" );
```

示例：

```
min_prefix_len = 3
```

11.2.19. min_infix_len：最小索引中缀长度

索引的最小中缀长度。可选选项，默认为 0（不索引中缀）。

中缀索引是实现“start*”, “*end”, and “*middle*”等形式的通配符成为可能（通配符语法的细节请参考 enable_star 选 项）。当最小中缀长度设置为正值，indexer 除了对关键字本身还会对所有可能的中缀（即子字符串）做索引。太短的中缀（短于允许的最小长度）不会被索 引。例如，在 min_infix_len=2 设置下索引关键字“test”会导致产生 6 个索引项 “te”, “es”, “st”, “tes”, “est”等中缀和词本身。对此索引搜索“es”会得到包含“test”的文档，即使它并不包含“es”本身。然而，中缀索引会使索引体积急剧增大（因为 待索引关键字增多了很多），而且索引和搜索的时间皆会恶化。

在中缀索引中没有自动的办法可以提高精确匹配（整个词完全匹配）的评分，但可以使用与 prefix indexes 选项中相同的技巧。

示例：

```
min_infix_len = 3
```

11.2.20. prefix_fields：前缀索引字段列表

做前缀索引的字段列表。可选选项，默认为空（所有字段均为前缀索引模式）。

因为前缀索引对索引和搜索性能均有影响，可能需要将它限制在某些特定的全文数据字段：例如，对 URL 提供前缀索引，但对页面内容不提供。prefix_fields 指定哪些字段要提供前缀索引，其他字段均会使用普通模式。值的格式是逗号分隔的字段名字列表。

示例：

```
prefix_fields = url, domain
```

11.2.21. infix_fields：中缀索引字段列表

做中缀索引的字段列表。可选选项，默认为空（所有字段均为中缀索引模式）。

与 prefix_fields 选项类似，但限制的是哪些字段做中缀索引。

示例：

```
infix_fields = url, domain
```

11.2.22. enable_star：星号语法

允许前缀/中缀索引上的星号语法（或称通配符）。可选选项，默认为 0（不使用通配符），这是为了与 0.9.7 版本的兼容性。可用的值为 0 和 1。

此特性启用搜索前缀或中缀索引时的“星号语法”，或者说通配符语法。仅影响搜索，因此要使改变生效只须重启 `searchd`，而不需要重新索引。

默认值为 0，意思是禁止星号语法，所有关键字都根据索引时的 min_prefix_len 和 min_infix_len settings 设置被视为前缀或者中缀。取值 1 的意思是星号（“*”）可以用在关键字的前面或后面。星号与零个或多个字符匹配。

例如，假设某索引启用了中缀索引，且 enable_star 值为 1。搜索过程按如下工作：

1. 查询 “abcdef” 仅匹配确切含有“abcdef”这个词的文档；
2. 查询 “abc*” 可匹配含有以“abc”开头的词的文档（包括精确匹配词“abc”的文档）；
3. 查询 “*cde*” 匹配在任何地方含有“cde”的词的文档（包括精确匹配词“cde”的文档）；
4. 查询 “*def” 匹配含有以“def”结束的词的文档（包括精确匹配词“def”的文档）

示例：

```
enable_star = 1
```

11.2.23. ngram_len：N-gram 长度

N-gram 索引的 N-gram 长度。可选选项，默认为 0（禁用 n-gram 索引）可用的值是 0 和 1（其他长度尚未实现）*对于使用 Coreseek 的中文用户，在启用了中文分词的情况下，本节内容可忽略。*

N-gram 提供对未分词 CJK（Chinese, Japanse, Koreasn 中日韩）文本的基本支持。CJK 搜索的问题在于词与词之前没有清晰的界限。理想中，文本可以通过一个称作分词程序（segmenter）的 特殊程序的过滤，之后分隔符即被加入到适当位置。然而分词过程缓慢而易错，因此通常会转而索引连续的一组 N 个字符，或称 N-gram。

启用此特性，CJK 字符流会被当作 n-gram 索引。例如，如果输入文本为“ABCDEF”（A 到 F 均代表 CJK 字符） ，而此选项设置的长度为 1，那它们会被当作“A B C D E F”而索引。（如果此选项设置的长度是 2，那会产生“AB BC CD DE EF”，但目前仅支持 1）。只有那些在 ngram_chars 选项表中列出的字符会这样分割，其他不受影响。

注意，如果搜索查询是已分词的，即单独的词之间有分隔符分隔，那么在扩展模式中将这些词放入引号中搜索会得到正确的匹配结果，即使文档**没有**分词。例如，假设原查询为“BC DEF”，在应用程序端用引号将索引包起来，看起来是“BC” “DEF”（*包括*引号），这个查询被传给 Sphinx 并在其内部分割成 1-gram，查询变成“B C” “D E F”，仍然包括作为词组查询操作符的引号。该查询会匹配正确的文本，即使文本中没有相应的分隔符。

即使搜索查询没有分词，Sphinx 也可以返回较好的结果，这要感谢基于词组的相关度计算：它会使相近的词组匹配（在 n-gram 中 CJK 词相当于多个字符的词匹配）排在前面。

示例：

```
ngram_len = 1
```

11.2.24. ngram_chars：N-gram 字符列表

N-gram 字符列表。 可选选项，默认为空。 *对于使用 Coreseek 的中文用户，在启用了中文分词的情况下，本节内容可忽略。*

与 ngram_len 选项联用，此列表定义了从中抽取 n-gram 的字符序列。其他字符组成的词不受 n-gram 索引特性的影响。值的格式与 charset_table 相同。

示例：

```
ngram_chars = U+3000..U+2FA1F
```

11.2.25. phrase_boundary：词组边界符列表

词组边界符列表。 可选选项，默认为空。

此列表控制哪些字符被视作分隔不同词组的边界，每到一个这样的边界，其后面的词的“位置”值都会被加入一个额外的增量，可以借此用近似搜索符来模拟词组搜索。语法与 charset_table 选项相似，但没有字符之间的映射关系，而且这些词组边界符不能重复出现在其他任何设置选项中。

自每个词组边界起，后面的词的“位置”都会被加入一个额外的增量（由 phrase_boundary_step 定 义）。这使通过近似搜索符实现词组搜索成为可能：不同词组中的词之间的距离肯定大于 phrase_boundary_step，因此相似距离小于 phrase_boundary_step 的近似搜索其实是在搜索在一个词组范围内出现了全部给定查询词的情况，相当于词组搜索。

只有词组边界符后面紧跟着一个分隔符时，词组边界才被激活，这是为了避免 S.T.A.L.K.E.R 或 URLs 等缩写被错当成若干个连续的词组（因为“.”属于词组边界符）。

示例：

```
phrase_boundary = ., ?, !, U+2026 # horizontal ellipsis
```

11.2.26. phrase_boundary_step：词组边界位置增量

词组边界上词位置的增量。可选选项，默认为 0。

在词组边界上，当前词位置会加上此选项设置的额外增量。详细请参考 phrase_boundary 选项。

示例：

```
phrase_boundary_step = 100
```

11.2.27. html_strip：HTMl 标记清理

是否从输入全文数据中去除 HTML 标记。可选标记，默认为 0。已知值包括 0（禁用）和 1（启用）。

HTML 标签和实体都可以被标记并且得到处理。

HTML 标签会被删除，但是其内容（即<P> 与 </P>之间的部分）默认情况下会保留。你可以选择标签中需要保留和索引的属性（例如 A 标签中的 HREF 属性，或者 IMG 标签中的 ALT 属 性）一些常见的内嵌标签会被完全清除，其他标签都被视为块级并被替换为空格。例如：'te<B>st</B>'文本将被单做一个 单一的关键字'test'被索引，但是，'te<P>st</P>'将被当做两个关键字'te'和'st'被索引。已知的内嵌标 签如下：A, B, I, S, U, BASEFONT, BIG, EM, FONT, IMG, LABEL, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TT.

HTML 实体会被解码并替换为对应的 UTF-8 字符。剥离器同时支持数字形式（例如ï）和文本形式（例如& oacute;或& nbsp;）。所有由 HTML4 标准规定的实体都支持。

此特性对 `xmlpipe` 数据源无效（建议升级到 xmlpipe2 ）。这个去除 HTML 标记的模块应该很好地工作于正确格式化的 HTML 和 XHTML，但就像大多数浏览器一样，对于格式错误的文本（例如带有无法配对的<和>的 HTML）可能产生不希望的输出。

只有 HTML 标签和 HTML 注释会被删除。要同时删除标签的内容（例如要删除内嵌的脚本），请参考 html_remove_elements 选项。标签名没有限制，即任何看起来像有效的标签开头、结束或者注释的内容都会被删除。

示例：

```
html_strip = 1
```

11.2.28. html_index_attrs：HTML 标签属性索引设置

去除 HTML 标记但要索引的标记属性列表。可选选项，默认为空（不索引标记的属性）。

指定被保留并索引的 HTML 标记属性，而 HTML 标记本身会被被删除。格式是对每个标记列举可以索引的属性，请看下例：

示例：

```
html_index_attrs = img=alt,title; a=title;
```

11.2.29. html_remove_elements：HTML 元素清理

完全清空 HTML 标签列表，不仅这些标签本身会被删除，标签本身包含的内容也会被删除。可选选项，默认为空（不删除任何标签的内容）。

此特性允许删除标签包含的内容，即在开始标记和结束标记之间的所有东西。一般用于删除内嵌的脚本或 CSS 等。短的没有实际内容的标签（即
）也支持；类似**不会**这样的标签中的内容将被删除。

值为逗号分隔的标签名称列表。标签名大小写无关。

示例：

```
html_remove_elements = style, script
```

11.2.30. local：本地索引声明

分布式索引 distributed index 中的本地索引声明。可以出现多次， 可选选项，默认为空。

此设置用于声明分布式索引被搜索时要搜索的本地索引。全部本地索引会被**依次**搜索，仅使用 1 个 CPU 或核。要并行处理，可以配置 `searchd` 查询它自身（细节参考 第 11.2.31 节 "agent：远程索引声明" ）。可以为每个分布式索引声明多个本地索引。每个本地索引可以在其他分布式索引中多次引用。

示例：

```
local = chunk1
local = chunk2
```

11.2.31. agent：远程索引声明

分布式索引（distributed index）中的远程代理和索引声明。可以出现多次，可选选项，默认为空。

此设置用来声明搜索分布式索引时要搜索的远程代理。代理可以看作网络指针，它指定了主机、端口和索引名。在最基本的情况下，代理可以与远程物理主机对应。更严格的来说，这不一定总是正确：可以将多个代理指向同一台远程主机，甚至指向同一个 `searchd` 实例（以便利用多个 CPU 或核）。

值的格式如下：

```
agent = specification:remote-indexes-list
specification = hostname ″:″ port | path
```

“hostname”是远程主机名，“port”是远程 TCP 端口，”path”是 Unix 域套接字的路径， 而“remote-index-list”是一个逗号分隔的远程索引列表。

全部代理会被并行搜索。然而同一个代理的多个索引是依次搜索的。这使您可以根据硬件来优化配置。例如，如果两个远程索引存储在一个 相同的硬盘上，最 好是配置一个带有多个按顺序搜索的索引，避免频繁的磁头寻址。如果这些索引存储在不同的硬盘上，那配置两个代理会更有利，因为这可以使工作完全并行。对于 CPU 也是如此，虽然在两个进程间切换对性能的影响比较小而且常常被完全忽略。

在有多个 CPU 和硬盘的机器上，代理可以指向相同的机器以便并行地使用硬件，降低查询延迟。并不需要为此设置多个 searchd 实例，一个实例与自身通信是合法的。以下是一个示例设置，它是为一台有 4 个 CPU 的机器准备的，可以并行地使用 4 个 CPU，各处理一个查询：

```
index dist
{
type = distributed
local = chunk1
agent = localhost:9312:chunk2
agent = localhost:9312:chunk3
agent = localhost:9312:chunk4
}
```

注意其中一块是本地搜索的，而同一个 searchd 示例又向本身查询，以便并行地启动其他三个搜索。

示例：

```
agent = localhost:9312:chunk2 # contact itself
agent = /var/run/searchd.s:chunk2
agent = searchbox2:9312:chunk3,chunk4 # search remote indexes
```

11.2.32. agent_blackhole：远程黑洞代理

分布式索引（ distributed index）中声明远程黑洞代理。多个值，可选选项，默认是空。于版本 0.9.9-rc1 引入。

agent_blackhole 选项使用户可以向远程代理发送“即发即忘”的查询（fire-and-forget queries）。这在调试（或者仅仅是测试）即将投入生产的集群的时候很有用：可以设置一个单独的调试/测试 searchd 实例，然后从实际生产中使用的主服务器（master，或称聚集者 aggregator）实例向这个测试服务器转发查询请求，但不干扰生产系统的工作。主（master）searchd 会以正常的方式尝试连接和查询黑洞代理，但是它既不会等待也不会处理黑洞代理的反馈。同时，发生在黑洞代理上的全部网络错误也都被忽略。值的格式与普通的 agent 完全相同。

示例：

```
agent_blackhole = testbox:9312:testindex1,testindex2
```

11.2.33. agent_connect_timeout：远程查询时间

远程代理的连接超时时间，单位为毫秒。可选选项，默认为 1000（即 1 秒）。

连接到远程代理时，searchd 最多花这些时间等待 connet()调用成功完成。如果达到了超时时间 connect()仍没有完成，而 and retries 选项是启用的，那么将开始重试。

示例：

```
agent_connect_timeout = 300
```

11.2.34. agent_query_timeout：远程查询超时时间

远程代理查询超时时间，以毫秒为单位。可选选项，默认为 3000（即 3 秒）。

连接后，searchd 最多花这这些时间等到远程查询完成。这个超时时间与连接超时时间是完全独立的。因此一个远程代理最多能造成的延迟为 agent_connection_timeout 与 agent_query_timeout 之和。如果超时时间已到，查询**不会**再重试，同时产生警告。

示例：

```
agent_query_timeout = 10000 # our query can be long, allow up to 10 sec
```

11.2.35. preopen：索引文件预开启

预先打开全部索引文件还是每次查询时再打开索引。可选选项，默认为 0（不预先打开）。

此选项令 searchd 在启动时（或轮换索引时）预先开打全部索引文件并在运行过程中保持打开。目前，默认是不预先打开这些文件（此行为可能在未来改变）。预先打开的每个索引文件会占用若干（目前是两个）文件描述符。但每次查询可以节约两个 open() 调用而且不会受高负载情况下索引轮换过程中可能发生的微妙的竞争条件（race condition）的影响。另一方面，当提供很多索引服务（几百到几千）时，必须每次查询时打开索引文件以便节约文件描述符。

这个指令不影响 indexer 的任何行为，只对 searchd 有用。

示例：

```
preopen = 1
```

11.2.36. ondisk_dict：字典文件保持设置

指定是将字典文件(.spi)保持在磁盘上还是将它预先缓冲在内存中。可选选项，默认值是 0(预先缓冲在内存里)。于版本 0.9.9-rc1 引入。

字典文件（.spi）既可以驻留在内存中也可以保持在磁盘上。默认情况下是将整个字典缓冲在内存中。这样做提高性能，但可能带来过大的内存压力，尤其是使用了前缀或中缀的时候。启用 ondisk_dict 为每次查询的每个关键词带来一次磁盘 I/O 操作，但是会减少内存使用。

这个指令不影响 indexer 的任何行为，只对 searchd 有用。

示例：

```
ondisk_dict = 1
```

11.2.37. inplace_enable：原地索引倒转设置

是否启用原地索引倒转（in-place index inversion）。可选选项，默认值是 0（使用单独的临时文件）。于版本 0.9.9-rc1 引入。

inplace_enable 选项极大地减少了建立索引时的磁盘压力，代价是略慢的索引速度（少使用大约两倍的磁盘空间，速度方面能达到原有性能的 90-95%）

建立索引的过程有两个主要的阶段。第一个阶段主要是收集、处理文档以及对文档根据关键词进行部分的排序，中间结果被写入到临时文件 （.tmp*） 中。第二个阶段则对文档进行完全排序并创建总重的索引文件。因此，重建一个正在应用于生产的索引将导致一个三倍大的磁盘占用峰值：第一倍，中间结果临时文 件，第二，新建的副本，和第三，在一切发生时旧的索引仍然占用一份磁盘空间，以便继续服务。（中间结果的大小与最终索引的大小相当）。对于很大的数据集， 这将是一笔很大的磁盘开销，而 inplace_enable 选项用于减少这个开销。一旦启用这个选项，临时文件将被重复利用，最终的数据写回到临时文件中，最后改个名字就可以作为最终结果使用了。然而，这可能导致更多的临时数据块重新分配，因此性能会有一点损失。

这个指令不影响 searchd 的任何行为，只对 indexer 有用。

示例：

```
inplace_enable = 1
```

11.2.38. inplace_hit_gap：原地索引倒转匹配点空隙设置

微调原地倒转（In-place inversion）行为的选项。控制预先分配的匹配点列表（hitlist）的空隙的大小。可选选项，默认是 0。于版本 0.9.9-rc1 引入。

这个指令不影响 searchd 的任何行为，只对 indexer 有用。

示例：

```
inplace_hit_gap = 1M
```

11.2.39. inplace_docinfo_gap：原地索引倒转文档信息空隙设置

微调原地倒转（In-place inversion）行为的选项。控制预先分配的文档信息（docinfo）的空隙的大小。可选选项，默认是 0。于版本 0.9.9-rc1 引入。

这个指令不影响 searchd 的任何行为，只对 indexer 有用。

示例：

```
inplace_docinfo_gap = 1M
```

11.2.40. inplace_reloc_factor：原地索引倒转重定位内存设置

微调原地倒转（In-place inversion ）行为的选项。控制重定位缓冲区占用索引时的内存的比例。可选选项，默认是 0.1。于版本 0.9.9-rc1 引入。

这个指令不影响 searchd 的任何行为，只对 indexer 有用。

示例：

```
inplace_reloc_factor = 0.1
```

11.2.41. inplace_write_factor：原地索引倒转写缓冲内存设置

微调原地倒转(In-place inversion)行为的选项。控制原地写缓冲占用索引时的内存的比例。可选选项，默认是 0.1。于版本 0.9.9-rc1 引入。

这个指令不影响 searchd 的任何行为，只对 indexer 有用。

示例：

```
inplace_write_factor = 0.1
```

11.2.42. index_exact_words：词干化后原词索引

是否在词干化处理后还索引原词（是否在索引原关键词的词干化/重映射后的形式的同时也索引原词）。可选选项，默认值是 0（不索引额外的形式）。于版本 0.9.9-rc1 引入。

一旦启用，index_exact_words 强制 indexer 除了词干化的版本外，将原始的关键字也加入索引。这样做也使查询语言中的精确匹配搜索符可用。这个选项将对索引大小和索引时间带来延迟。然而搜索的性能不会被影响。

示例：

```
index_exact_words = 1
```

11.2.43. overshort_step：短词位置增量

在经过过短的词（比 min_word_len 短的词）处后增加位置值。 可选选项，允许的值是 0 或者 1，默认是 1。于版本 0.9.9-rc1 引入。

这个指令不影响 searchd 的任何行为，只对 indexer 有用。

示例：

```
overshort_step = 1
```

11.2.44. stopword_step：通用词位置增量

在经过 停用词 处后增加位置值可选选项，允许的值是 0 或者 1，默认是 1。于版本 0.9.9-rc1 引入。

这个指令不影响 searchd 的任何行为，只对 indexer 有用。

示例：

```
stopword_step = 1
```

11.2.45. hitless_words：位置忽略词汇列表

位置忽略词汇列表。 可选项，可用值为 all 或者列表所在的文件名。 版本 2.0.1-beta 引入。

By default, Sphinx full-text index stores not only a list of matching documents for every given keyword, but also a list of its in-document positions (aka hitlist). Hitlists enables phrase, proximity, strict order and other advanced types of searching, as well as phrase proximity ranking. However, hitlists for specific frequent keywords (that can not be stopped for some reason despite being frequent) can get huge and thus slow to process while querying. Also, in some cases we might only care about boolean keyword matching, and never need position-based searching operators (such as phrase matching) nor phrase ranking.

`hitless_words` lets you create indexes that either do not have positional information (hitlists) at all, or skip it for specific keywords.

Hitless index will generally use less space than the respective regular index (about 1.5x can be expected). Both indexing and searching should be faster, at a cost of missing positional query and ranking support. When searching, positional queries (eg. phrase queries) will be automatically converted to respective non-positional (document-level) or combined queries. For instance, if keywords “hello” and “world” are hitless, “hello world” phrase query will be converted to (hello & world) bag-of-words query, matching all documents that mention either of the keywords but not necessarily the exact phrase. And if, in addition, keywords “simon” and “says” are not hitless, “simon says hello world” will be converted to (“simon says” & hello & world) query, matching all documents that contain “hello” and “world” anywhere in the document, and also “simon says” as an exact phrase.

示例：

```
hitless_words = all
```

11.2.46. expand_keywords：词汇展开

尽可能展开关键字的精确格式或者型号形式。 可选项，默认为 0（不展开关键字）。 版本 2.0.1-beta 引入。

Queries against indexes with `expand_keywords` feature enabled are internally expanded as follows. If the index was built with prefix or infix indexing enabled, every keyword gets internally replaced with a disjunction of keyword itself and a respective prefix or infix (keyword with stars). If the index was built with both stemming and index_exact_words enabled, exact form is also added. Here’s an example that shows how internal expansion works when all of the above (infixes, stemming, and exact words) are combined:

```
running -> ( running | *running* | =running )
```

Expanded queries take naturally longer to complete, but can possibly improve the search quality, as the documents with exact form matches should be ranked generally higher than documents with stemmed or infix matches.

Note that the existing query syntax does not allowe to emulate this kind of expansion, because internal expansion works on keyword level and expands keywords within phrase or quorum operators too (which is not possible through the query syntax).

This directive does not affect `indexer` in any way, it only affects `searchd`.

示例：

```
expand_keywords = 1
```

11.2.47. blend_chars：混合字符列表

混合字符列表。 可选项，默认为空。 版本 2.0.1-beta 引入。

Blended characters are indexed both as separators and valid characters. For instance, assume that & is configured as blended and AT&T occurs in an indexed document. Three different keywords will get indexed, namely “at&t”, treating blended characters as valid, plus “at” and “t”, treating them as separators.

Positions for tokens obtained by replacing blended characters with whitespace are assigned as usual, so regular keywords will be indexed just as if there was no `blend_chars` specified at all. An additional token that mixes blended and non-blended characters will be put at the starting position. For instance, if the field contents are “AT&T company” occurs in the very beginning of the text field, “at” will be given position 1, “t” position 2, “company” positin 3, and “AT&T” will also be given position 1 (“blending” with the opening regular keyword). Thus, querying for either AT&T or just AT will match that document, and querying for “AT T” as a phrase also match it. Last but not least, phrase query for “AT&T company” will *also*match it, despite the position

Blended characters can overlap with special characters used in query syntax (think of T-Mobile or @twitter). Where possible, query parser will automatically handle blended character as blended. For instance, “hello @twitter” within quotes (a phrase operator) would

handle @-sign as blended, because @-syntax for field operator is not allowed within phrases. Otherwise, the character would be handled as an operator. So you might want to escape the keywords.

Starting with version 2.0.1-beta, blended characters can be remapped, so that multiple different blended characters could be normalized into just one base form. This is useful when indexing multiple alternative Unicode codepoints with equivalent glyphs.

示例：

```
blend_chars = +, &, U+23
blend_chars = +, &->+ # 2.0.1-beta and above
```

11.2.48. blend_mode：混合类型

混合类型列表。可选项，默认为 `trim_none`。 版本 2.0.1-beta 引入。

By default, tokens that mix blended and non-blended characters get indexed in there entirety. For instance, when both at-sign and an exclamation are in `blend_chars`, “@dude!” will get result in two tokens indexed: “@dude!” (with all the blended characters) and “dude” (without any). Therefore “@dude” query will *not* match it.

`blend_mode` directive adds flexibility to this indexing behavior. It takes a comma-separated list of options.

```
blend_mode = option [, option [, ...]]
option = trim_none | trim_head | trim_tail | trim_both | skip_pure
```

Options specify token indexing variants. If multiple options are specified, multiple variants of the same token will be indexed. Regular keywords (resulting from that token by replacing blended with whitespace) are always be indexed.

```
trim_none
      Index the entire token.
trim_head
      Trim heading blended characters, and index the resulting token.
trim_tail
      Trim trailing blended characters, and index the resulting token.
trim_both
      Trim both heading and trailing blended characters, and index the resulting token.
skip_pure
      Do not index the token if it’s purely blended, that is, consists of blended characters only.
```

Returning to the “@dude!” example above, setting `blend_mode = trim_head, trim_tail` will result in two tokens being indexed, “@dude” and “dude!”. In this particular example, `trim_both` would have no effect, because trimming both blended characters results in “dude” which is already indexed as a regular keyword. Indexing “@U.S.A.” with `trim_both` (and assuming that dot is blended two) would result in “U.S.A” being indexed. Last but not least, `skip_pure` enables you to fully ignore sequences of blended characters only. For example, “one @@@ two” would be indexed exactly as “one two”, and match that as a phrase. That is not the case by default because a fully blended token gets indexed and offsets the second keyword position.

Default behavior is to index the entire token, equivalent to `blend_mode = trim_none`.

示例：

```
blend_mode = trim_tail, skip_pure
```

11.2.49. rt_mem_limit：RT 索引内存限制

内存区块限制大小。 可选项，默认为空。 版本 2.0.1-beta 引入。

RT index keeps some data in memory (so-called RAM chunk) and also maintains a number of on-disk indexes (so-called disk chunks). This directive lets you control the RAM chunk size. Once there’s too much data to keep in RAM, RT index will flush it to disk, activate a newly created disk chunk, and reset the RAM chunk.

The limit is pretty strict; RT index should never allocate more memory than it’s limited to. The memory is not preallocated either, hence, specifying 512 MB limit and only inserting 3 MB of data should result in allocating 3 MB, not 512 MB.

示例：

```
rt_mem_limit = 512M
```

11.2.50. rt_field：字段设置

全文字段定义。 多值选项，必须设置。 版本 2.0.1-beta 引入。

Full-text fields to be indexed are declared using `rt_field` directive. The names must be unique. The order is preserved; and so field values in INSERT statements without an explicit list of inserted columns will have to be in the same order as configured.

示例：

```
rt_field = author
rt_field = title
rt_field = content
```

11.2.51. rt_attr_uint：整数属性

无符号整数属性定义。 多值选项（允许设置任意个此属性），可选项。 声明一个 32 位的无符号数属性。 版本 2.0.1-beta 引入。

示例：

```
rt_attr_uint = gid
```

11.2.52. rt_attr_bigint：长整数属性

BIGINT 属性定义 多值选项（允许设置任意个此属性），可选项。 声明一个 64 位的有符号数属性。 版本 2.0.1-beta 引入。

示例：

```
rt_attr_bigint = guid
```

11.2.53. rt_attr_float：浮点数属性

浮点数属性定义。 多值选项（允许设置任意个此属性），可选项。 声明一个单精度，32 为的 IEEE 754 格式的浮点数属性 版本 2.0.1-beta 引入。

示例：

```
rt_attr_float = gpa
```

11.2.54. rt_attr_timestamp：UNIX 时间戳属性

时间戳属性定义。 多值选项（允许设置任意个此属性），可选项。 版本 2.0.1-beta 引入。

示例：

```
rt_attr_timestamp = date_added
```

11.2.55. rt_attr_string：字符串属性

字符串属性定义。 多值选项（允许设置任意个此属性），可选项。 版本 2.0.1-beta 引入。

示例：

```
rt_attr_string = author
```

11.3. indexer 程序配置选项

11.3.1. mem_limit：索引内存限制

索引过程中内存使用限制。可选选项，默认 32M。

这是 indexer 不会超越的强制内存限制。可以以字节、千字节（以 K 为后缀）或兆字节（以 M 为后缀）为单位。参见示例。当过小的值导致 I/O 缓冲低于 8KB 时该限制会自动提高，此值的最低限度依赖于待索引数据的大小。如果缓冲低于 256KB，会产生警告。

最大可能的限制是 2047M。太低的值会影响索引速度，但 256M 到 1024M 对绝大多数数据集（如果不是全部）来说应该足够了。这个值设得太高可 能导致 SQL 服务器连接超时。在文档收集阶段，有时内存缓冲的一部分会被排序，而与数据库的通信会暂停，于是数据库服务器可能超时。这可以通过提高 SQL 服务器端的超时时间或降低 mem_limit 来解决。 *在 Coreseek 的分发版本中，如果使用 python 数据源，则在 Python 部分的处理不会受 mem_limit 的限制。*

示例：

```
mem_limit = 256M
# mem_limit = 262144K # same, but in KB
# mem_limit = 268435456 # same, but in bytes
```

11.3.2. max_iops：每秒 IO 操作限制

每秒最大 I/O 操作次数，用于限制 I/O 操作。可选选项，默认为 0（无限制）。

与 I/O 节流有关的选项。它限制了每秒钟最大的 I/O 操作（读或写）的次数。值 0 意思是不加限制。

indexer 在索引时可能导致突发的密集磁盘 I/O，因此需要限制它磁盘活动（给同一台机器上运行的其他程序留出一些资源，比如 searchd）。 I/O 节流就是用来实现上述功能的。它的工作原理是，在 indexer 的连续磁盘 I/O 操作之间强制增加一个保证的延迟。现代 SATA 硬盘每秒钟可以执行多达 70-100 以上次的 I/O 操作（主要受磁头寻道时间的限制）。将索引 I/O 限制为上述数值的几分之一可以减轻由索引带来的搜索性能下降。

示例：

```
max_iops = 40
```

11.3.3. max_iosize：最大 IO 操作限制

最大允许的 I/O 操作大小，以字节为单位，用于 I/O 节流。可选选项，默认为 0（不限制）。

与 I/O 节流有关的选项。它限制 indexer 文件 I/O 操作的单次最大大小。值 0 代表不加限制。超过限制的读写操作会被分成几个小的操作，并被 max_iops 计为多次。在本文写作时，全部 I/O 操作都被限制在 256KB 以下（默认的内部缓冲大小），因此大于 256KB 的 max_iosize 值没有任何作用。

示例：

```
max_iosize = 1048576
```

11.3.4. max_xmlpipe2_field：最大字段大小

对于 XMLLpipe2 数据源允许的最大的字段大小，以字节为单位。可选选项，默认值为 2MB。

示例：

```
max_xmlpipe2_field = 8M
```

11.3.5. write_buffer：写缓冲大小

写缓冲区的大小，单位是字节。可选选项，默认值是 1MB。

在建立索引过程中，写缓冲用于写入临时行而最终的索引文件。写缓冲区越大则所需的磁盘写入次数越少。缓冲区使用的内存不计入 mem_limit 选项的值。注意对于不同的文件，会分配多个缓冲区（目前最多 4 个），这会引起内存占用增加。

示例：

```
write_buffer = 4M
```

11.3.6. max_file_field_buffer：外部文件缓冲大小

文件字段可用的最大缓冲区大小，字节为单位。 可选项，默认为 8MB，最小值为 1MB.

File field buffer is used to load files referred to from sql_file_field columns. This buffer is adaptive, starting at 1 MB at first allocation, and growing in 2x steps until either file contents can be loaded, or maximum buffer size, specified by `max_file_field_buffer` directive, is reached.

Thus, if there are no file fields are specified, no buffer is allocated at all. If all files loaded during indexing are under (for example) 2 MB in size, but `max_file_field_buffer` value is 128 MB, peak buffer usage would still be only 2 MB. However, files over 128 MB would be entirely skipped.

示例：

```
max_file_field_buffer = 128M
```

11.4. `searchd` 程序配置选项

11.4.1. listen：监听设置

指定 `searchd` 监听的 IP 地址和端口，或 UNIX 域 socket 的路径。于版本 0.9.9-rc1 引入。

一个非正式的 `listen` 语法说明如下：

```
listen = ( address ":" port | port | path ) [ ":" protocol ]
```

即，用户可以指定 IP 地址（或主机名）和端口号，或者仅仅是端口号，或者 Unix socket 的路径。如果仅指定了端口号而没有地址，那么 `searchd` 会在所有的网络接口上监听。Unix 路径都带有一个前导的斜杠"/"。

从版本 0.9.9-rc2 开始，还可以指定一个协议处理器（protocl handler），或者叫监听器（listener），应用于这个 socket 上的连接。支持的协议值包括'sphinx'（Sphinx 0.9.x API 协议）和 mysql41（版本 4.1 到至少 5.1 的 MySQL 使用的协议）。关于 MySQL 协议的更多细节可以参考 第 5.10 节 "MySQL 协议支持与 SphinxQL" 。

示例：

```
listen = localhost
listen = localhost:5000
listen = 192.168.0.1:5000
listen = /var/run/sphinx.s
listen = 9312
listen = localhost:9306:mysql41
```

可以有多个 listen 指令，`searchd` 会在所有这些指令纸浆的端口和 socket 上监听，以备用户连接。如果没有找到 `listen` 指令，服务器会在所有的网络接口上用默认端口（9312）监听。 从版本 1.10-beta 开始，也会自动监听默认的 SphinxQL 端口 9306。这两个端口号是由 IANA（查看 http://www.iana.org/assignments/port-numbers 了解详情）所分配的，因此可以正常使用。

在 Windows 上不支持 Unix 域套接字。

11.4.2. address：监听地址

要绑定的接口 IP 地址。可选项，默认为 0.0.0.0（即在所有接口上监听）。 **不推荐**, 建议使用 listen 。

`address` 设置指定 `searchd` 在哪个接口上绑定、监听和接受输入的网络连接。默认值为 0.0.0.0，意思是在所有接口上监听。目前**不能**指定多个接口。

示例：

```
address = 192.168.0.1
```

11.4.3. port：监听端口

`searchd` 的 TCP 端口号。 **不推荐**, 建议使用 listen 。 必选项，默认为 9312。

示例：

```
port = 9312
```

11.4.4. log：搜索系统日志

日志文件名。可选项，默认为“searchd.log”。全部 `searchd` 运行时事件会被记录在这个日志文件中。

Also you can use the ‘syslog’ as the file name. In this case the events will be sent to syslog daemon. To use the syslog option the sphinx must be configured ‘–with-syslog’ on building.

示例：

```
log = /var/log/searchd.log
```

11.4.5. query_log：搜索查询日志

查询日志文件名。可选项，默认为空（不记录查询日志）。全部搜索查询会被记录在此文件中。其格式在第 5.9 节 “搜索服务(searchd) 查询日志格式”中描述。

In case of ‘plain’ format, you can use the ‘syslog’ as the path to the log file. In this case all search queries will be sent to syslog daemon with LOG_INFO priority, prefixed with ‘[query]‘ instead of timestamp. To use the syslog option the sphinx must be configured ‘–with-syslog’ on building.

示例：

```
query_log = /var/log/query.log
```

11.4.6. query_log_format：查询日志格式

查询日志格式。 可选项，可用值为 plain、sphinxql，默认为 plain。 版本 2.0.1-beta 引入。

Starting with version 2.0.1-beta, two different log formats are supported. The default one logs queries in a custom text format. The new one logs valid SphinxQL statements. This directive allows to switch between the two formats on search daemon startup. The log format can also be altered on the fly, using `SET GLOBAL query_log_format=sphinxql` syntax. Refer to 第 5.9 节 “搜索服务(searchd) 查询日志格式” for more discussion and format details.

示例：

```
query_log_format = sphinxql
```

11.4.7. read_timeout：远程读取超时时间

网络客户端请求的读超时时间，单位是秒。可选项，默认是 5 秒。 `searchd` 强制关闭在此时间内未能成功发出查询的客户端连接。

示例：

```
read_timeout = 1
```

11.4.8. client_timeout：客户端超时时间

在使用持久连接时，两次查询之间等待的最长时间（单位是秒）。可选选项，默认是 5 分钟。

示例：

```
client_timeout = 3600
```

11.4.9. max_children：子进程数目限制

子进程的最大数量（或者说，并行执行的搜索的数目）。可选项，默认为 0，不限制。

用来控制服务器负载。任何时候不可能有比此设置值更多的搜索同时运行。当达到限制时，新的输入客户端会被用临时失败（SEARCH_RETRY）状态码驳回，同时给出一个声明服务器已到最大连接限制的消息。

示例：

```
max_children = 10
```

11.4.10. pid_file：PID 文件

searchd PID 文件名。必选项。 Mandatory.

PID 文件会在启动时重建（并锁定）。主守护进程运行时它含有该进程的 ID，而当守护进程退出时该文件会被删除。这个选项是必须的，因为 Sphinx 在内部使用它做如下事：检查是否已有一个 searchd 示例；停止 searchd；通知 searchd 应该轮换索引了。也可以被各种不同的外部自动化脚本所利用。

示例：

```
pid_file = /var/run/searchd.pid
```

11.4.11. max_matches：最大返回匹配数

守护进程在内存中为每个索引所保持并返回给客户端的匹配数目的最大值。可选选项，默认值为 1000。

引入此选项是为了控制和限制内存使用，max_matches 设置定义了搜索每个索引时有多少匹配项会保存在内存中。每个找到的匹配项都会被*处理*，但只有它们中最佳的 N 个会在内存中保持并最终返回给客户端。假设索引中包括 2000000 个当前查询的匹配项，你几乎总是不需要它们中的*全部*。 通常您需要扫描它们并根据某种条件（即按相关度排序、或者价格、或者其他什么）选出最好的那些，比如 500 个，并以在页面上显示 20 到 100 项。只跟踪最 好的 500 个匹配要比保持全部的 2000000 个匹配项大大地节约内存和 CPU，之后可以对这些最佳匹配排序，然后丢弃除了要在页面上要显式的 20 项之外 的结果。max_matches 控制“最佳 N 个匹配”中的 N。

此参数明显影响每个查询消耗的内存和 CPU。1000 到 10000 的值通常就可以满足需求，但更高的值要小心使用。粗心地把 max_matches 增加到 1000000 意味着 searchd 被迫为*每一个*查询分配 1M 条匹配项的缓冲。这会明显增大查询的内存消耗，有时会明显影响性能。

特别注意！此限制还可在**另一个**地方指定。max_matches 可以通过对应的 API 调用实时降低，该调用的默认值**也**是 1000。因此要使应用程序获取超过 1000 个匹配结果，必须修改配置文件，重启 searchd，再用 SetLimits()调用设置合适的限制。还要注意，API 调用设置的限制不能大于.conf 文件中的设置，这是为了预防恶意的或错误的请求。

示例：

```
max_matches = 10000
```

11.4.12. seamless_rotate：无缝轮换

防止 searchd 轮换在需要预取大量数据的索引时停止响应。可选选项，默认为 1（启用无缝（seamless）轮换）。

索引可能包含某些需要预取到内存中的数据。目前.spa, .spi 和 .spm 文件会被完全预取到内存中（它们分别包含属性数据，MVA 数据和关键字索引）。若无无缝轮换，轮换索引时会尽量使用较小的内存，并如下工作：

1. 新的查询暂时被拒绝（用“retry”错误码）；
2. searchd 等待目前正在运行的查询结束；
3. 旧的索引被释放，文件被重命名；
4. 新的索引文件被重命名，分配所需的内存；
5. 新的索引属性和字典数据预调进内存；
6. searchd 恢复为新索引提供查询服务。

然而，如果有大量的属性或字典数据，那么预调数据的步骤可能消耗大量的时间——预调 1.5GB 的文件可能需要几分钟的时间。

当启用了无缝轮换，轮换按如下工作：

1. 为新索引分配内存；
2. 新索引的属性和字典数据异步地预调进内存；
3. 如果成功，旧的索引被释放，新旧索引文件被重命名；
4. 如果失败，释放新索引；
5. 在任意时刻，查询服务都正常运行——或者使用旧索引，或者使用新索引。

无缝轮换以轮换过程中更大的**峰值**内存消耗为代价（因为当预调新索引时.spa/.spi/.spm 数据的新旧拷贝需要同时保持在内存中）。平均内存耗用不变。

示例：

```
seamless_rotate = 1
```

11.4.13. preopen_indexes：索引预开启

是否在启动是强制重新打开所有索引文件。可选选项，默认为 1（全部打开）。

Starting with 2.0.1-beta, the default value for this option is now 1 (foribly preopen all indexes). In prior versions, it used to be 0 (use per-index settings).

设置为 1 时，该配置会覆盖并对所有提供服务的索引强制 打开 preopen 选项 They will be preopened, no matter what is the per-index preopen setting. When set to 0, per-index settings can take effect. (And they default to 0.)

Pre-opened indexes avoid races between search queries and rotations that can cause queries to fail occasionally. They also make searchd use more file handles. In most scenarios it's therefore preferred and recommended to preopen indexes.

示例：

```
preopen_indexes = 1
```

11.4.14. unlink_old：旧索引清理

索引轮换成功之后，是否删除以.old 为扩展名的索引拷贝。可选选项，默认为 1（删除这些索引拷贝）。

示例：

```
unlink_old = 0
```

11.4.15. attr_flush_period：属性刷新周期

用 UpdateAttributes() 实时更新文档属性时，所产生的变化首先写入到这些属性在内存中的一份拷贝中（必须将 docinfo 设置成 extern）。其后，一旦 searchd 正常关闭（通过发送 SIGTERM 信号），这些变化才写入磁盘。于版本 0.9.9-rc1 中引入。

从版本 0.9.9-rc1 开始，可以令 searchd 每隔一段时间就将变化写回磁盘，防止丢失这些变化。这个间隔时间通过 attr_flush_period 选项设置，单位是秒。

默认值是 0，即关闭隔一段时间就将变化写回磁盘的特性，但是正常关闭时的写回不被关闭。

示例：

```
attr_flush_period = 900 # persist updates to disk every 15 minutes
```

11.4.16. ondisk_dict_default：索引字典存储方式

对 ondisk_dict 指令的全局的默认值。 可选选项，默认值是 0（将字典预先缓冲到内存）。于版本 0.9.9-rc1 中引入。

这个选项用于为当前使用的这份 searchd 正在提供服务的所有索引指定 ondisk_dict 选项的默认值。如果某个索引的这个选项做了显式设定，那么这个设定覆盖上述实例级的默认设置，这种机制提供了细粒度的控制。

示例：

```
ondisk_dict_default = 1 # keep all dictionaries on disk
```

11.4.17. max_packet_size：最大包大小

网络通讯时允许的最大的包的大小。这个限制既对来自客户端的查询包有效，也对分布式环境下远程代理返回的响应包有效。只用于内部校验，不直接影响内存占用和性能。可选选项，默认值是 8M。于版本 0.9.9-rc1 引入。

示例：

```
max_packet_size = 32M
```

11.4.18. mva_updates_pool：MVA 更新共享内存

用于多值属性 MVA 更新的存储空间的内存共享池大小。可选选项，默认大小是 1M。于版本 0.9.9-rc1 引入。

这个设置控制用于存储多值属性 MVA 更新后的值共享存储池的大小。如果指定大小为 0 则意味着完全禁用多值属性 MVA 的更新。一旦达 到了这个内存池大 小的限制，尝试更新多值属性 MVA 将得到错误。但普通的（标量的）属性仍然可以更新。由于内部实现上的技术困难，一旦多值属性 MVA 有所更新，则索引上发 生的**任何**更新改变都**不能**在索引重建前被写入（store，flush）磁盘，尽管这可能在未来实现。同时，多值属性 MVA 是设计用来在索引重建前迅速反应数据库中的变化，而不是一种永久存储的机制。

示例：

```
mva_updates_pool = 16M
```

11.4.19. crash_log_path：崩溃日志

崩溃日志文件的路径（正式地说叫做前缀）。可选选项，默认值为空（不创建崩溃日志文件）。于版本 0.9.9-rc1 引入。 从版本 2.0.1-beta 开始不再推荐使用，崩溃调试信息将以文本格式会记录到 searchd.log 日志文件中，独立的二进制崩溃文件不再需要了。

11.4.20. max_filters：最大过滤器数目

每次查询允许设置的过滤器的最大个数。只用于内部检查，不直接影响内存使用或性能。可选选项，默认值是 256。于版本 0.9.9-rc1 引入。

示例：

```
max_filters = 1024
```

11.4.21. max_filter_values：单个过滤器最大过滤值数目

单个过滤器允许的值的最大个数。只用于内部检查，不直接影响内存使用或性能。可选选项，默认值是 4096。于版本 0.9.9-rc1 引入。

示例：

```
max_filter_values = 16384
```

11.4.22. listen_backlog：带处理监听队列

TCP 监听待处理队列长度。可选选项，默认值是 5。

在 Windows 系统上创建的 Sphinx 目前（版本 0.9.9）只能一个接一个地处理请求。同时发生的请求会被操作系统级别的 TCP 协议栈装入到 一个队列中，无法如对的请求立即失败并收到“连接被拒”错误信息。listen_backlog 选项控制这个连接队列的长度。非 Windows 平台上创建 的 Sphinx 使用默认值即可。

示例：

```
listen_backlog = 20
```

11.4.23. read_buffer：读缓冲区

每个关键字的读缓冲区的大小。可选选项，默认值是 256K。

对于每个搜索查询中的每个关键词，有两个相关的读缓冲区（一个针对文档列表，一个针对关键词出现位置列表）。本选项允许控制他们的大小，增加每次查询的内存占用，但可能会减少 IO 时间。

示例：

```
read_buffer = 1M
```

11.4.24. read_unhinted：无匹配时读取大小

无匹配时读操作的大小。可选选项，默认值是 32K。

当系统处理查询时，对于一些读取操作，系统预先就知道要读取的数据的确切长度，但是有一些却相反。其中对常见的是已匹配位置列表 （hitlist） 的长度目前是无法预先取得的。这个选项控制在这些情况下读取多少数据。它会影响 IO 时间，对于比本选项设置值大的列表，IO 时间减少，而对于那些较小的列 表则是 IO 时间增加。内存占用不受影响，因为读缓冲区已经是分配好了的。也因此这个选项的设置值不能超过选项 read_buffer 的设置值。

示例：

```
read_unhinted = 32K
```

11.4.25. max_batch_queries：最大批量查询

每次批量查询的查询数限制。 可选项，默认为 32 个。

Makes searchd perform a sanity check of the amount of the queries submitted in a single batch when using multi-queries. Set it to 0 to skip the check.

示例：

```
max_batch_queries = 256
```

11.4.26. subtree_docs_cache：子树优化文档缓存

每个查询的公共子树文档缓存大小。 可选值，默认为 0（禁止）。

Limits RAM usage of a common subtree optimizer (参见 第 5.11 节 “批量查询”). At most this much RAM will be spent to cache document entries per each query. Setting the limit to 0 disables the optimizer.

示例：

```
subtree_docs_cache = 8M
```

11.4.27. subtree_hits_cache：子树优化命中缓存

每个查询的公共子树命中缓存大小。 可选值，默认为 0（禁止）。

Limits RAM usage of a common subtree optimizer (参见 第 5.11 节 “批量查询”). At most this much RAM will be spent to cache keyword occurrences (hits) per each query. Setting the limit to 0 disables the optimizer.

示例：

```
subtree_hits_cache = 16M
```

11.4.28. workers：MPM 模式

多处理模式（MPM）。 可选项；可用值为 none、fork、prefork，以及 threads。 默认在 Unix 类系统为 form，Windows 系统为 threads。 版本 2.0.1-beta 引入。

Lets you choose how `searchd` processes multiple concurrent requests. The possible values are:

none
: All requests will be handled serially, one-by-one. Prior to 1.x, this was the only mode available on Windows.

fork
: A new child process will be forked to handle every incoming request. Historically, this is the default mode.

prefork
: On startup, searchd will pre-fork a number of worker processes, and pass the incoming requests to one of those children.

threads
: A new thread will be created to handle every incoming request. This is the only mode compatible with RT indexing backend.

Historically, `searchd` used fork-based model, which generally performs OK but spends a noticeable amount of CPU in fork() system call when there's a high amount of (tiny) requests per second. Prefork mode was implemented to alleviate that; with prefork, worker processes are basically only created on startup and re-created on index rotation, somewhat reducing fork() call pressure.

Threads mode was implemented along with RT backend and is required to use RT indexes. (Regular disk-based indexes work in all the available modes.)

示例：

```
workers = threads
```

11.4.29. dist_threads：并发查询线程数

Max local worker threads to use for parallelizable requests (searching a distributed index; building a batch of snippets). Optional, default is 0, which means to disable in-request parallelism. 版本 2.0.1-beta 引入。

Distributed index can include several local indexes. `dist_threads` lets you easily utilize multiple CPUs/cores for that (previously existing alternative was to specify the indexes as remote agents, pointing searchd to itself and paying some network overheads).

When set to a value N greater than 1, this directive will create up to N threads for every query, and schedule the specific searches within these threads. For example, if there are 7 local indexes to search and dist_threads is set to 2, then 2 parallel threads would be created: one that sequentially searches 4 indexes, and another one that searches the other 3 indexes.

In case of CPU bound workload, setting `dist_threads` to 1x the number of cores is advised (creating more threads than cores will not improve query time). In case of mixed CPU/disk bound workload it might sometimes make sense to use more (so that all cores could be utilizes even when there are threads that wait for I/O completion).

Note that `dist_threads` does **not** require threads MPM. You can perfectly use it with fork or prefork MPMs too.

Starting with version 2.0.1-beta, building a batch of snippets with `load_files` flag enabled can also be parallelized. Up to `dist_threads` threads are be created to process those files. That speeds up snippet extraction when the total amount of document data to process is significant (hundreds of megabytes).

示例：

```
index dist_test
{
type = distributed
local = chunk1
local = chunk2
local = chunk3
local = chunk4
}
# ...

dist_threads = 4
```

11.4.30. binlog_path：二进制日志路径

Binary log (aka transaction log) files path. Optional, default is build-time configured data directory. 版本 2.0.1-beta 引入。

Binary logs are used for crash recovery of RT index data that would otherwise only be stored in RAM. When logging is enabled, every transaction COMMIT-ted into RT index gets written into a log file. Logs are then automatically replayed on startup after an unclean shutdown, recovering the logged changes.

`binlog_path` directive specifies the binary log files location. It should contain just the path; `searchd` will create and unlink multiple binlog.* files in that path as necessary (binlog data, metadata, and lock files, etc).

Empty value disables binary logging. That improves performance, but puts RT index data at risk.

示例：

```
binlog_path = # disable logging
binlog_path = /var/data # /var/data/binlog.001 etc will be created
```

11.4.31. binlog_flush：二进制日志刷新

Binary log transaction flush/sync mode. Optional, default is 2 (flush every transaction, sync every second). 版本 2.0.1-beta 引入。

This directive controls how frequently will binary log be flushed to OS and synced to disk. Three modes are supported:

- 0, flush and sync every second. Best performance, but up to 1 second worth of committed transactions can be lost both on daemon crash, or OS/hardware crash.
- 1, flush and sync every transaction. Worst performance, but every committed transaction data is guaranteed to be saved.
- 2, flush every transaction, sync every second. Good performance, and every committed transaction is guaranteed to be saved in case of daemon crash. However, in case of OS/hardware crash up to 1 second worth of committed transactions can be lost.

For those familiar with MySQL and InnoDB, this directive is entirely similar to `innodb_flush_log_at_trx_commit`. In most cases, the default hybrid mode 2 provides a nice balance of speed and safety, with full RT index data protection against daemon crashes, and some protection against hardware ones.

示例：

```
binlog_flush = 1 # ultimate safety, low speed
```

11.4.32. binlog_max_log_size：二进制日志大小限制

Maximum binary log file size. Optional, default is 0 (do not reopen binlog file based on size). 版本 2.0.1-beta 引入。

A new binlog file will be forcibly opened once the current binlog file reaches this limit. This achieves a finer granularity of logs and can yield more efficient binlog disk usage under certain borderline workloads.

示例：

```
binlog_max_log_size = 16M
```

11.4.33. collation_server：服务端默认字符集

Default server collation. Optional, default is libc_ci. 版本 2.0.1-beta 引入。

Specifies the default collation used for incoming requests. The collation can be overridden on a per-query basis. Refer to 第 5.12 节 “字符串排序规则” section for the list of available collations and other details.

示例：

```
collation_server = utf8_ci
```

11.4.34. collation_libc_locale：服务端 libc 字符集

Server libc locale. Optional, default is C. 版本 2.0.1-beta 引入。

Specifies the libc locale, affecting the libc-based collations. Refer to 第 5.12 节 “字符串排序规则” section for the details.

示例：

```
collation_libc_locale = fr_FR
```

11.4.35. plugin_dir：插件目录

Trusted location for the dynamic libraries (UDFs). Optional, default is empty (no location). 版本 2.0.1-beta 引入。

Specifies the trusted directory from which the UDF libraries can be loaded. Requires workers = thread to take effect.

示例：

```
workers = threads
plugin_dir = /usr/local/sphinx/lib
```

11.4.36. mysql_version_string：MySQL 版本设置

A server version string to return via MySQL protocol. Optional, default is empty (return Sphinx version). 版本 2.0.1-beta 引入。

Several picky MySQL client libraries depend on a particular version number format used by MySQL, and moreover, sometimes choose a different execution path based on the reported version number (rather than the indicated capabilities flags). For instance, Python MySQLdb 1.2.2 throws an exception when the version number is not in X.Y.ZZ format; MySQL .NET connector 6.3.x fails internally on version numbers 1.x along with a certain combination of flags, etc. To workaround that, you can use `mysql_version_string` directive and have `searchd` report a different version to clients connecting over MySQL protocol. (By default, it reports its own version.)

示例：

```
mysql_version_string = 5.0.37
```

11.4.37. rt_flush_period：RT 索引刷新周期

RT indexes RAM chunk flush check period, in seconds. Optional, default is 0 (do not flush). 版本 2.0.1-beta 引入。

Actively updated RT indexes that however fully fit in RAM chunks can result in ever-growing binlogs, impacting disk use and crash recovery time. With this directive the search daemon performs periodic flush checks, and eligible RAM chunks can get saved, enabling consequential binlog cleanup. 参见 第 4.4 节 “二进制日志” for more details.

示例：

```
rt_flush_period = 3600
```

11.4.38. thread_stack：线程堆栈

Per-thread stack size. Optional, default is 64K. 版本 2.0.1-beta 引入。

In the `workers = threads` mode, every request is processed with a separate thread that needs its own stack space. By default, 64K per thread are allocated for stack. However, extremely complex search requests might eventually exhaust the default stack and require more. For instance, a query that matches a few thousand keywords (either directly or through term expansion) can eventually run out of stack. Previously, that resulted in crashes. Starting with 2.0.1-beta, `searchd` attempts to estimate the expected stack use, and blocks the potentially dangerous queries. To process such queries, you can either the thread stack size by using the `thread_stack` directive (or switch to a different `workers` setting if that is possible).

A query with N levels of nesting is estimated to require approximately 30+0.12*N KB of stack, meaning that the default 64K is enough for queries with upto 300 levels, 150K for upto 1000 levels, etc. If the stack size limit is not met, `searchd` fails the query and reports the required stack size in the error message.

示例：

```
thread_stack = 256K
```

11.4.39. expansion_limit：关键字展开限制

The maximum number of expanded keywords for a single wildcard. Optional, default is 0 (no limit). 版本 2.0.1-beta 引入。

When doing substring searches against indexes built with `dict = keywords` enabled, a single wildcard may potentially result in thousands and even millions of matched keywords (think of matching ‘a*’ against the entire Oxford dictionary). This directive lets you limit the impact of such expansions. Setting `expansion_limit = N` restricts expansions to no more than N of the most frequent matching keywords (per each wildcard in the query).

示例：

```
expansion_limit = 16
```

11.4.40. compat_sphinxql_magics

Legacy SphinxQL quirks compatiblity mode. Optional, default is 1 (keep compatibility). Introduced in version 2.0.1-beta.

Starting with version 2.0.1-beta, we're bringing SphinxQL in closer compliance with standard SQL. However, existing applications must not get broken, and `compat_sphinxql_magics` lets you upgrade safely. It defauls to 1, which enables the compatibility mode. However, **SphinxQL compatibility mode is now deprecated and will be removed** once we complete bringing SphinxQL in line with standard SQL syntax. So it's advised to update the applications utilising SphinxQL and then switch the daemon to the new, more SQL compliant mode by setting`compat_sphinxql_magics = 0`. Please refer to 第 7.21 节 "SphinxQL 升级备注, version 2.0.1-beta" for the details and update instruction.

Example:

```
compat_sphinxql_magics = 0 # the future is now
```

11.4.41. watchdog

Threaded server watchdog. Optional, default is 1 (watchdog enabled). Introduced in version 2.0.1-beta.

A crashed query in `threads` multi-processing mode (workers `= threads`) can take down the entire server. With watchdog feature enabled, `searchd` additionally keeps a separate lightweight process that monitors the main server process, and automatically restarts the latter in case of abnormal termination. Watchdog is enabled by default.

Example:

```
watchdog = 0 # disable watchdog
```

第 12 章 Coreseek 配置选项参考

目录

12.1. 中文分词核心配置

关于中文分词的详细配置实例和分词词典的自定义设置，可以访问 Coreseek 网站中文分词核心配置查看。

核心配置：`charset_dictpath = /usr/local/mmseg3/etc/charset_type = zh_cn.utf-8 #charset_table = ..... #需将原有的该配置注释掉 ngram_len = 0`

12.1.1. charset_dictpath

设置中文分词词典所在的目录；

示例:

```
#Linux
charset_dictpath              = /usr/local/mmseg3/etc/
#Windows
charset_dictpath              = C:\usr\local\coreseek\etc
```

12.1.2. charset_type

设置文档的字符集，可选的值为“zh_cn.utf-8”、“zh_cn.gbk”和“zh_cn.big5”。“zh_cn.gbk”和“zh_cn.big5”需要 iconv 支持。

示例:

```
charset_type          =  zh_cn.utf-8
```

12.2. MMSEG 分词配置选项

mmseg 分词相关的配置选项，需要保存到文件 `mmseg.ini`，并将该配置文件放置到 `charset_dictpath` 所设置的目录中。

基本配置：

```
[mmseg]merge_number_and_ascii=0;       ;合并英文和数字 abc123/xnumber_and_ascii_joint=-;      ;定义可以连接英文
和数字的字符 compress_space=1;            ;暂不支持 seperate_number_ascii=0;      ;就是将字母和数字打散
其中，分号表示注释
```

12.2.1. merge_number_and_ascii

是否合并英文和数字，该选项设置是否将连接在一起的英文字母和数字作为一个整体看待

例如:

```
merge_number_and_ascii=0; ;abc123 将被切分为 abc、123
merge_number_and_ascii=1; ;abc123 将不被切分
```

12.2.2. number_and_ascii_joint

定义可以连接英文和数字的字符

例如:

```
number_and_ascii_joint=-_; ;abc_123、abc-123 都将作为整体
```

12.2.3. compress_space

预设配置，暂不支持

12.2.4. seperate_number_ascii

是否将字母和数字打散

例如:

```
seperate_number_ascii=0; ;abc 作为整体
seperate_number_ascii=1; ;abc 被切分为 a、b、c
```

12.3. Python 数据源程序接口

Python 数据源的各种实例程序，可以访问 Coreseek 网站 Python 数据源获取。该部分的相关文档，还在继续完善中。

```
#Python 数据源基本演示程序
#/usr/local/coreseek/etc/pysource/csft_demo/__init__.py
# -*- coding:utf-8 -*-
```

```
class MainSource(object):
def __init__(self, conf):
self.conf =  conf
self.data = [
{'id':1, 'subject':u"标题 1", 'date':1270131607},
{'id':2, 'subject':u'标题 2', 'date':1270135548},
]
self.idx = 0

def GetScheme(self):
return [
('id' , {'docid':True, } ),
('subject', { 'type':'text'} ),
('date', {'type':'integer'} ),
]

def GetKillList(self):
return [3,1]

def GetFieldOrder(self):
return ('subject')

def Connected(self):
pass

def OnBeforeIndex(self):
print 'before index called'
pass

def NextDocument(self, err):
if self.idx < len(self.data):
item = self.data[self.idx]
self.id = item['id']
self.subject = item['subject'].encode('utf-8')
self.date = item['date']
self.idx += 1
return True
else:
return False
pass

def OnAfterIndex(self):
print 'after index called'
pass

def OnIndexFinished(self):
print 'after index finished'
pass

if __name__ == "__main__":
source = MainSource()
source.Connected()

source.OnBeforeIndex()
while source.NextDocument():
print "id=%d, title=%s" % (source.docid, source.title)

source.OnAfterIndex()
source.OnIndexFinished()
pass
#eof
```

12.3.1. GetScheme() （设置检索字段的属性）

返回所有需要被索引的字段的属性

例如:

```
    def GetScheme(self):
return [
('id' , {'docid':True, } ),
('subject', { 'type':'text'} ),
('date', {'type':'integer'} ),
]
```

在检索字段设置中，必须且仅有一个字段需要设置为 docid，对应于 SQL 数据源中的文档编号。其他字段的属性，可以为 integer，对应于 `sql_attr_uint`，或者 text，对应于全文检索字段。目前支持的类型如下：

- docid：对应于 SQL 数据源中的文档编号 id
- integer：整数属性，参见 sql_attr_uint
- bool：布尔属性，参见 sql_attr_bool
- long：长整型属性，参见 sql_attr_bigint
- timestamp：UNIX 时间戳（整数）属性，参见 sql_attr_timestamp
- str2ord：字符串序数排序属性，参见 sql_attr_str2ordinal
- float：浮点数属性，参见 sql_attr_float
- list：多值属性(MVA)属性，参见 sql_attr_multi 的 field
- list-query：多值属性(MVA)属性，参见 sql_attr_multi 的 query
- string：字符串属性(可返回原始文本信息)，参见 sql_attr_string
- string_text：字符串字段(可全文搜索，可返回原始文本信息)，参见 sql_field_string
- text：全文检索字段，参见全文检索字段

12.3.2. GetKillList() （设置不参与检索的文档编号）

处于该列表之中的文档，将不被检索，对应于 SQL 数据源的 `sql_query_killlist`

例如:

```
    def GetKillList(self):
return [3,1]
```

12.3.3. GetFieldOrder() （设置字段的顺序）

全文字段被检索的顺序

例如:

```
    def GetFieldOrder(self):
return [('subject','content')]
```

12.3.4. Connected() （获取数据前的连接处理）

一般用于进行数据库的连接等预处理

例如:

```
    def Connected(self):
#在此进行数据库连接和处理
pass
```

12.3.5. OnBeforeIndex() （数据获取前处理）

类似 `sql_query_pre` 配置选项的作用

例如:

```
    def OnBeforeIndex(self):
print 'before index called'
pass
```

12.3.6. NextDocument() （文档获取处理）

获取实际的需要参与检索的数据，按条获取，需要获取的字段，作为 self 自身的属性给出，相当于 `sql_query` 的作用，每次读取一条数据

例如:

```
    def NextDocument(self, err):
if self.idx < len(self.data):
item = self.data[self.idx]
self.id = item['id']
self.subject = item['subject'].encode('utf-8')
self.date = item['date']
self.idx += 1
return True
else:
return False
pass
```

12.3.7. OnAfterIndex() （数据获取后处理）

类似 `sql_query_post` 配置选项的作用

例如:

```
    def OnAfterIndex(self):
print 'after index called'
pass
```

12.3.8. OnIndexFinished() （索引完成时处理）

类似 `sql_query_post_index` 配置选项的作用

例如:

```
    def OnIndexFinished(self):
print 'after index finished'
pass
```

附录 A. Sphinx revision history

目录

A.1. Version 2.0.1-beta, 22 apr 2011

New general features

- added remapping support to blend_chars directive
- added multi-threaded snippet batches support (requires a batch sent via API, dist_threads, and `load_files`)
- added collations (collation_server, collation_libc_locale directives)
- added support for sorting and grouping on string attributes (`ORDER BY`, `GROUP BY`, `WITHING GROUP ORDER BY`)
- added UDF support (plugin_dir directive; CREATE FUNCTION, DROP FUNCTION statements)
- added query_log_format directive, SET GLOBAL query_log_format | log_level = … statements; and connection id tracking
- added sql_column_buffers directive, fixed out-of-buffer column handling in ODBC/MS SQL sources
- added blend_mode directive that enables indexing multiple variants of a blended sequence
- added UNIX socket support to C, Ruby APIs
- added ranged query support to sql_joined_field
- added rt_flush_period directive
- added thread_stack directive
- added SENTENCE, PARAGRAPH, ZONE operators (and index_sp, index_zones directives)
- added keywords dictionary support (and dict, expansion_limit directives)
- added `passage_boundary`, `emit_zones` options to snippets
- added a watchdog process in threaded mode
- added persistent MVA updates
- added crash dumps to `searchd.log`, deprecated `crash_log_path` directive
- added id32 index support in id64 binaries (EXPERIMENTAL)
- added SphinxSE support for DELETE and REPLACE on SphinxQL tables

New SphinxQL features

- added new, more SQL compliant SphinxQL syntax; and a compat_sphinxql_magics directive
- added CRC32(), DAY(), MONTH(), YEAR(), YEARMONTH(), YEARMONTHDAY() functions
- added DIV, MOD, and % operators
- added reverse_scan=(0|1) option to SELECT
- added support for MySQL packets over 16M
- added dummy SHOW VARIABLES, SHOW COLLATION, and SET character_set_results support (to support handshake with certain client libraries and frameworks)
- added mysql_version_string directive (to workaround picky MySQL client libraries)
- added support for global filter variables, SET GLOBAL @uservar=(int_list)
- added DELETE … IN (id_list) syntax support
- added C-style comments syntax (for example, `SELECT /*!40000 some comment*/ id FROM test`)
- added UPDATE … WHERE id=X syntax support
- added SphinxQL multi-query support
- added DESCRIBE, SHOW TABLES statements

New command-line switches

- added `--print-queries` switch to `indexer` that dumps SQL queries it runs
- added `--sighup-each` switch to `indexer` that rotates indexes one by one
- added `--strip-path` switch to `searchd` that skips file paths embedded in the index(-es)
- added `--dumpconfig` switch to `indexer` that dumps an index header in `sphinx.conf` format

Major changes and optimizations

- changed default preopen_indexes value to 1
- optimized English stemmer (results in 1.3x faster snippets and indexing with morphology=stem_en)
- optimized snippets, 1.6x general speedup
- optimized const-list parsing in SphinxQL
- optimized full-document highlighting CPU/RAM use
- optimized binlog replay (improved performance on K-list update)

Bug fixes

- fixed #767, joined fields vs ODBC sources
- fixed #757, wordforms shared by indexes with different settings
- fixed #733, loading of indexes in formats prior to v.14
- fixed #763, occasional snippets failures
- fixed #648, occasionally missed rotations on multiple SIGHUPs
- fixed #750, an RT segment merge leading to false positives and/or crashes in some cases
- fixed #755, zones in snippets output

- fixed #754, stopwords counting at snippet passage generation
- fixed #723, fork/prefork index rotation in children processes
- fixed #696, freeze on zero threshold in quorum operator
- fixed #732, query escaping in SphinxSE
- fixed #739, occasional crashes in MT mode on result set send
- fixed #746, crash with a named list in SphinxQL option
- fixed #674, AVG vs group order
- fixed #734, occasional crashes attempting to report NULL errors
- fixed #829, tail hits within field position modifier
- fixed #712, missing query_mode, force_all_words snippet option defaults in Java API
- fixed #721, added dupe removal on RT batch INSERT/REPLACE
- fixed #720, potential extraneous highlighting after a blended keyword
- fixed #702, exceptions vs star search
- fixed #666, ext2 query grouping vs exceptions
- fixed #688, WITHIN GROUP ORDER BY related crash
- fixed #660, multi-queue batches vs dist_threads
- fixed #678, crash on dict=keywords vs xmlpipe vs min_prefix_len
- fixed #596, ECHILD vs scripted configs
- fixed #653, dependency in expression, sorting, grouping
- fixed #661, concurrent distributed searches vs workers=threads
- fixed #646, crash on status query via UNIX socket
- fixed #589, libexpat.dll missing from some Win32 build types
- fixed #574, quorum match order
- fixed multiple documentation issues (#372, #483, #495, #601, #623, #632, #654)
- fixed that ondisk_dict did not affect RT indexes
- fixed that string attributes check in indextool –check was erroneously sensitive to string data order
- fixed a rare crash when using BEFORE operator
- fixed an issue with multiforms vs BuildKeywords()
- fixed an edge case in OR operator (emitted wrong hits order sometimes)
- fixed aliasing in docinfo accessors that lead to very rare crashes and/or missing results
- fixed a syntax error on a short token at the end of a query
- fixed id64 filtering and performance degradation with range filters
- fixed missing rankers in libsphinxclient
- fixed missing SPH04 ranker in SphinxSE
- fixed column names in sql_attr_multi sample (works with example.sql now)
- fixed an issue with distributed local+remote setup vs aggregate functions
- fixed case sensitive columns names in RT indexes
- fixed a crash vs strings from multiple indexes in result set
- fixed blended keywords vs snippets
- fixed secure_connection vs MySQL protocol vs MySQL.NET connector
- fixed that Python API did not works with Python 2.3
- fixed overshort_step vs snippets
- fixed keyword staistics vs dist_threads searching
- fixed multiforms vs query parsing (vs quorum)
- fixed missed quorum words vs RT segments
- fixed blended keywords occasionally skipping extra character when querying (eg “abc[]“)
- fixed Python API to handle int32 values
- fixed prefix and infix indexing of joined fields
- fixed MVA ranged query
- fixed missing blended state reset on document boundary
- fixed a crash on missing index while replaying binlog
- fixed an error message on filter values overrun
- fixed passage duplication in snippets in weight_order mode
- fixed select clauses over 1K vs remote agents
- fixed overshort accounting vs soft-whitespace tokens
- fixed rotation vs workers=threads
- fixed schema issues vs distributed indexes
- fixed blended-escaped sequence parsing issue
- fixed MySQL IN clause (values order etc)
- fixed that post_index did not execute when 0 documents were succesfully indexed
- fixed field position limit vs many hits
- fixed that joined fields missed an end marker at field end
- fixed that xxx_step settings were missing from .sph index header
- fixed libsphinxclient missing request cleanup in sphinx_query() (eg after network errors)
- fixed that index_weights were ignored when grouping
- fixed multi wordforms vs blend_chars
- fixed broken MVA output in SphinxQL

- fixed a few RT leaks
- fixed an issue with RT string storage going missing
- fixed an issue with repeated queries vs dist_threads
- fixed an issue with string attributes vs buffer overrun in SphinxQL
- fixed unexpected character data warnings within ignored xmlpipe tags
- fixed a crash in snippets with NEAR syntax query
- fixed passage duplication in snippets
- fixed libsphinxclient SIGPIPE handling
- fixed libsphinxclient vs VS2003 compiler bug

A.2. Version 1.10-beta, 19 jul 2010

- added RT indexes support (第 4 章 RT 实时索引)
- added prefork and threads support (workers directives)
- added multi-threaded local searches in distributed indexes (dist_threads directive)
- added common subquery cache (subtree_docs_cache, subtree_hits_cache directives)
- added string attributes support (sql_attr_string, sql_field_string, xml_attr_string, xml_field_string directives)
- added indexing-time word counter (sql_attr_str2wordcount, sql_field_str2wordcount directives)
- added CALL SNIPPETS(), CALL KEYWORDS() SphinxQL statements
- added `field_weights`, `index_weights` options to SphinxQL SELECT statement
- added insert-only SphinxQL-talking tables to SphinxSE (connection='sphinxql://host[:port]/index')
- added `select` option to SphinxSE queries
- added backtrace on crash to `searchd`
- added SQL+FS indexing, aka loading files by names fetched from SQL (sql_file_field directive)
- added a watchdog in threads mode to `searchd`
- added automatic row phantoms elimination to index merge
- added hitless indexing support (hitless_words directive)
- added –check, –strip-path, –htmlstrip, –dumphitlist … –wordid switches to indextool
- added –stopwait, –logdebug switches to searchd
- added –dump-rows, –verbose switches to indexer
- added "blended" characters indexing support (blend_chars directive)
- added joined/payload field indexing (sql_joined_field directive)
- added FlushAttributes() API call
- added query_mode, force_all_words, limit_passages, limit_words, start_passage_id, load_files, html_strip_mode, allow_empty options, and %PASSAGE_ID% macro in before_match, after_match options to BuildExcerpts() API call
- added @groupby/@count/@distinct columns support to SELECT (but not to expressions)
- added query-time keyword expansion support (expand_keywords directive, SPH_RANK_SPH04 ranker)
- added query batch size limit option (max_batch_queries directive; was hardcoded)
- added SINT() function to expressions
- improved SphinxQL syntax error reporting
- improved expression optimizer (better constant handling)
- improved dash handling within keywords (no longer treated as an operator)
- improved snippets (better passage selection/trimming, around option now a hard limit)
- optimized index format that yields ~20-30% smaller indexes
- optimized sorting code (indexing time 1-5% faster on average; 100x faster in worst case)
- optimized searchd startup time (moved .spa preindexing to indexer), added a progress bar
- optimized queries against indexes with many attributes (eliminated redundant copying)
- optimized 1-keyword queries (performace regression introduced in 0.9.9)
- optimized SphinxQL protocol overheads, and performance on bigger result sets
- optimized unbuffered attributes writes on index merge
- changed attribute handling, duplicate names are strictly forbidden now
- fixed that SphinxQL sessions could stall shutdown
- fixed consts with leading minus in SphinxQL
- fixed AND/OR precedence in expressions
- fixed #334, AVG() on integers was not computed in floats
- fixed #371, attribute flush vs 2+ GB files
- fixed #373, segfault on distributed queries vs certain libc versions
- fixed #398, stopwords not stopped in prefix/infix indexes
- fixed #404, erroneous MVA failures in indextool –check
- fixed #408, segfault on certain query batches (regular scan, plus a scan with MVA groupby)
- fixed #431, occasional shutdown hangs in preforked workers
- fixed #436, trunk checkout builds vs Solaris sh
- fixed #440, escaping vs parentheses declared as valid in charset_table
- fixed #442, occasional non-aligned free in MVA indexing
- fixed #447, occasional crashes in MVA indexing
- fixed #449, pconn busyloop on aborted clients on certain arches

- fixed #465, build issue on Alpha
- fixed #468, build issue in libsphinxclient
- fixed #472, multiple stopword files failing to load
- fixed #489, buffer overflow in query logging
- fixed #493, Python API assertion after error returned from Query()
- fixed #500, malformed MySQL packet when sending MVAs
- fixed #504, SIGPIPE in libsphinxclient
- fixed #506, better MySQL protocol commands support in SphinxQL (PING etc)
- fixed #509, indexing ranged results from stored procedures

A.3. Version 0.9.9-release, 02 dec 2009

- added Open, Close, Status calls to libsphinxclient (C API)
- added automatic persistent connection reopening to PHP, Python APIs
- added 64-bit value/range filters, fullscan mode support to SphinxSE
- MAJOR CHANGE, our IANA assigned ports are 9312 and 9306 respectively (goodbye, trusty 3312)
- MAJOR CHANGE, erroneous filters now fail with an error (were silently ignored before)
- optimized unbuffered .spa writes on merge
- optimized 1-keyword queries ranking in extended2 mode
- fixed #441 (IO race in case of highly conccurent load on a preopened)
- fixed #434 (distrubuted indexes were not searchable via MySQL protocol)
- fixed #317 (indexer MVA progress counter)
- fixed #398 (stopwords not removed from search query)
- fixed #328 (broken cutoff)
- fixed #250 (now quoting paths w/spaces when installing Windows service)
- fixed #348 (K-list was not updated on merge)
- fixed #357 (destination index were not K-list-filtered on merge)
- fixed #369 (precaching .spi files over 2 GBs)
- fixed #438 (missing boundary proximity matches)
- fixed #371 (.spa flush in case of files over 2 GBs)
- fixed #373 (crashes on distributed queries via mysql proto)
- fixed critical bugs in hit merging code
- fixed #424 (ordinals could be misplaced during indexing in case of bitfields etc)
- fixed #426 (failing SE build on Solaris; thanks to Ben Beecher)
- fixed #423 (typo in SE caused crash on SHOW STATUS)
- fixed #363 (handling of read_timeout over 2147 seconds)
- fixed #376 (minor error message mismatch)
- fixed #413 (minus in SphinxQL)
- fixed #417 (floats w/o leading digit in SphinxQL)
- fixed #403 (typo in SetFieldWeights name in Java API)
- fixed index rotation vs persistent connections
- fixed backslash handling in SphinxQL parser
- fixed uint unpacking vs. PHP 5.2.9 (possibly other versions)
- fixed #325 (filter settings send from SphinxSE)
- fixed #352 (removed mysql wrapper around close() in SphinxSE)
- fixed #389 (display error messages through SphinxSE status variable)
- fixed linking with port-installed iconv on OS X
- fixed negative 64-bit unpacking in PHP API
- fixed #349 (escaping backslash in query emulation mode)
- fixed #320 (disabled multi-query route when select items differ)
- fixed #353 (better quorum counts check)
- fixed #341 (merging of trailing hits; maybe other ranking issues too)
- fixed #368 (partially; @field “” caused crashes; now resets field limit)
- fixed #365 (field mask was leaking on field-limited terms)
- fixed #339 (updated debug query dumper)
- fixed #361 (added SetConnectTimeout() to Java API)
- fixed #338 (added missing fullscan to mode check in Java API)
- fixed #323 (added floats support to SphinxQL)
- fixed #340 (support listen=port:proto syntax too)
- fixed #332 (\r is legal SphinxQL space now)
- fixed xmlpipe2 K-lists
- fixed #322 (safety gaps in mysql protocol row buffer)
- fixed #313 (return keyword stats for empty indexes too)
- fixed #344 (invalid checkpoints after merge)
- fixed #326 (missing CLOCK_xxx on FreeBSD)

A.4. Version 0.9.9-rc2, 08 apr 2009

- added IsConnectError(), Open(), Close() calls to Java API (bug #240)
- added read_buffer, read_unhinted directives
- added checks for build options returned by mysql_config (builds on Solaris now)
- added fixed-RAM index merge (bug #169)
- added logging chained queries count in case of (optimized) multi-queries
- added GEODIST() function
- added –status switch to searchd
- added MySpell (OpenOffice) affix file support (bug #281)
- added ODBC support (both Windows and UnixODBC)
- added support for @id in IN() (bug #292)
- added support for aggregate functions in GROUP BY (namely AVG, MAX, MIN, SUM)
- added MySQL UDF that builds snippets using searchd
- added write_buffer directive (defaults to 1M)
- added xmlpipe_fixup_utf8 directive
- added suggestions sample
- added microsecond precision int64 timer (bug #282)
- added listen_backlog directive
- added max_xmlpipe2_field directive
- added initial SphinxQL support to mysql41 handler, SELECT …/SHOW WARNINGS/STATUS/META are handled
- added support for different network protocols, and mysql41 protocol
- added fieldmask ranker, updated SphinxSE list of rankers
- added mysql_ssl_xxx directives
- added –cpustats (requires clock_gettime()) and –status switches to searchd
- added performance counters, Status() API call
- added overshort_step and stopword_step directives
- added strict order operator (aka operator before, eg. “one << two << three”)
- added indextool utility, moved –dumpheader there, added –debugdocids, –dumphitlist options
- added own RNG, reseeded on @random sort query (bug #183)
- added field-start and field-end modifiers support (syntax is “^hello world$”; field-end requires reindex)
- added MVA attribute support to IN() function
- added AND, OR, and NOT support to expressions
- improved logging of (optimized) multi-queries (now logging chained query count)
- improved handshake error handling, fixed protocol version byte order (omg)
- updated SphinxSE to protocol 1.22
- allowed phrase_boundary_step=-1 (trick to emulate keyword expansion)
- removed SPH_MAX_QUERY_WORDS limit
- fixed CLI search vs documents missing from DB (bug #257)
- fixed libsphinxclient results leak on subsequent sphinx_run_queries call (bug #256)
- fixed libsphinxclient handling of zero max_matches and cutoff (bug #208)
- fixed Java API over-64K string reads (eg. big snippets) in Java API (bug #181)
- fixed Java API 2nd Query() after network error in 1st Query() call (bug #308)
- fixed typo-class bugs in SetFilterFloatRange (bug #259), SetSortMode (bug #248)
- fixed missing @@relaxed support (bug #276), fixed missing error on @nosuchfield queries, documented @@relaxed
- fixed UNIX socket permissions to 0777 (bug #288)
- fixed xmlpipe2 crash on schemas with no fields, added better document structure checks
- fixed (and optimized) expr parser vs IN() with huge (10K+) args count
- fixed double EarlyCalc() in fullscan mode (minor performance impact)
- fixed phrase boundary handling in some cases (on buffer end, on trailing whitespace)
- fixes in snippets (aka excerpts) generation
- fixed inline attrs vs id64 index corruption
- fixed head searchd crash on config re-parse failure
- fixed handling of numeric keywords with leading zeroes such as “007″ (bug #251)
- fixed junk in SphinxSE status variables (bug #304)
- fixed wordlist checkpoints serialization (bug #236)
- fixed unaligned docinfo id access (bug #230)
- fixed GetRawBytes() vs oversized blocks (headers with over 32K charset_table should now work, bug #300)
- fixed buffer overflow caused by too long dest wordform, updated tests
- fixed IF() return type (was always int, is deduced now)
- fixed legacy queries vs. special chars vs. multiple indexes
- fixed write-write-read socket access pattern vs Nagle vs delays vs FreeBSD (oh wow)
- fixed exceptions vs query-parser issue
- fixed late calc vs @weight in expressions (bug #285)
- fixed early lookup/calc vs filters (bug #284)
- fixed emulated MATCH_ANY queries (empty proximity and phrase queries are allowed now)

- fixed MATCH_ANY ranker vs fields with no matches
- fixed index file size vs inplace_enable (bug #245)
- fixed that old logs were not closed on USR1 (bug #221)
- fixed handling of '!' alias to NOT operator (bug #237)
- fixed error handling vs query steps (step failure was not reported)
- fixed querying vs inline attributes
- fixed stupid bug in escaping code, fixed EscapeString() and made it static
- fixed parser vs @field -keyword, foo|@field bar, "" queries (bug #310)

A.5. Version 0.9.9-rc1, 17 nov 2008

- added min_stemming_len directive
- added IsConnectError() API call (helps distingusih API vs remote errors)
- added duplicate log messages filter to searchd
- added –nodetach debugging switch to searchd
- added blackhole agents support for debugging/testing (agent_blackhole directive)
- added max_filters, max_filter_values directives (were hardcoded before)
- added int64 expression evaluation path, automatic inference, and BIGINT() enforcer function
- added crash handler for debugging (crash_log_path directive)
- added MS SQL (aka SQL Server) source support (Windows only, mssql_winauth and mssql_unicode directives)
- added indexer-side column unpacking feature (unpack_zlib, unpack_mysqlcompress directives)
- added nested brackers and NOTs support to query language, rewritten query parser
- added persistent connections support (Open() and Close() API calls)
- added index_exact_words feature, and exact form operator to query language ("hello =world")
- added status variables support to SphinxSE (SHOW STATUS LIKE 'sphinx_%')
- added max_packet_size directive (was hardcoded at 8M before)
- added UNIX socket support, and multi-interface support (listen directive)
- added star-syntax support to BuildExcerpts() API call
- added inplace inversion of .spa and .spp (inplace_enable directive, 1.5-2x less disk space for indexing)
- added builtin Czech stemmer (morphology=stem_cz)
- added IDIV(), NOW(), INTERVAL(), IN() functions to expressions
- added index-level early-reject based on filters
- added MVA updates feature (mva_updates_pool directive)
- added select-list feature with computed expressions support (see SetSelect() API call, test.php –select switch), protocol 1.22
- added integer expressions support (2x faster than float)
- added multiforms support (multiple source words in wordforms file)
- added legacy rankers (MATCH_ALL/MATCH_ANY/etc), removed legacy matching code (everything runs on V2 engine now)
- added field position limit modifier to field operator (syntax: @title[50] hello world)
- added killlist support (sql_query_killlist directive, –merge-killlists switch)
- added on-disk SPI support (ondisk_dict directive)
- added indexer IO stats
- added periodic .spa flush (attr_flush_period directive)
- added config reload on SIGHUP
- added per-query attribute overrides feature (see SetOverride() API call); protocol 1.21
- added signed 64bit attrs support (sql_attr_bigint directive)
- improved HTML stripper to also skip PIs (<? … ?>, such as <?php … ?>)
- improved excerpts speed (upto 50x faster on big documents)
- fixed a short window of searchd inaccessibility on startup (started listen()ing too early before)
- fixed .spa loading on systems where read() is 2GB capped
- fixed infixes vs morphology issues
- fixed backslash escaping, added backslash to EscapeString()
- fixed handling of over-2GB dictionary files (.spi)

A.6. Version 0.9.8.1, 30 oct 2008

- added configure script to libsphinxclient
- changed proximity/quorum operator syntax to require whitespace after length
- fixed potential head process crash on SIGPIPE during "maxed out" message
- fixed handling of incomplete remote replies (caused over-degraded distributed results, in rare cases)
- fixed sending of big remote requests (caused distributed requests to fail, in rare cases)
- fixed FD_SET() overflow (caused searchd to crash on startup, in rare cases)
- fixed MVA vs distributed indexes (caused loss of 1st MVA value in result set)
- fixed tokenizing of exceptions terminated by specials (eg. "GPS AT&T" in extended mode)
- fixed buffer overrun in stemmer on overlong tokens occasionally emitted by proximity/quorum operator parser (caused crashes on certain proximity/quorum queries)

- fixed wordcount ranker (could be dropping hits)
- fixed –merge feature (numerous different fixes, caused broken indexes)
- fixed –merge-dst-range performance
- fixed prefix/infix generation for stopwords
- fixed ignore_chars vs specials
- fixed misplaced F_SETLKW check (caused certain build types, eg. RPM build on FC8, to fail)
- fixed dictionary-defined charsets support in spelldump, added \x-style wordchars support
- fixed Java API to properly send long strings (over 64K; eg. long document bodies for excerpts)
- fixed Python API to accept offset/limit of 'long' type
- fixed default ID range (that filtered out all 64-bit values) in Java and Python APIs

A.7. Version 0.9.8, 14 jul 2008

Indexing

- added support for 64-bit document and keyword IDs, –enable-id64 switch to configure
- added support for floating point attributes
- added support for bitfields in attributes, sql_attr_bool directive and bit-widths part in sql_attr_uint directive
- added support for multi-valued attributes (MVA)
- added metaphone preprocessor
- added libstemmer library support, provides stemmers for a number of additional languages
- added xmlpipe2 source type, that supports arbitrary fields and attributes
- added word form dictionaries, wordforms directive (and spelldump utility)
- added tokenizing exceptions, exceptions directive
- added an option to fully remove element contents to HTML stripper, html_remove_elements directive
- added HTML entities decoder (with full XHTML1 set support) to HTML stripper
- added per-index HTML stripping settings, html_strip, html_index_attrs, and html_remove_elements directives
- added IO load throttling, max_iops and max_iosize directives
- added SQL load throttling, sql_ranged_throttle directive
- added an option to index prefixes/infixes for given fields only, prefix_fields and infix_fields directives
- added an option to ignore certain characters (instead of just treating them as whitespace), ignore_chars directive
- added an option to increment word position on phrase boundary characters, phrase_boundary and phrase_boundary_step directives
- added –merge-dst-range switch (and filters) to index merging feature (–merge switch)
- added mysql_connect_flags directive (eg. to reduce indexing time MySQL network traffic and/or time)
- improved ordinals sorting; now runs in fixed RAM
- improved handling of documents with zero/NULL ids, now skipping them instead of aborting

Search daemon

- added an option to unlink old index on succesful rotation, unlink_old directive
- added an option to keep index files open at all times (fixes subtle races on rotation), preopen and preopen_indexes directives
- added an option to profile searchd disk I/O, –iostats command-line option
- added an option to rotate index seamlessly (fully avoids query stalls), seamless_rotate directive
- added HTML stripping support to excerpts (uses per-index settings)
- added 'exact_phrase', 'single_passage', 'use_boundaries', 'weight_order 'options to BuildExcerpts() API call
- added distributed attribute updates propagation
- added distributed retries on master node side
- added log reopen on SIGUSR1
- added –stop switch (sends SIGTERM to running instance)
- added Windows service mode, and –servicename switch
- added Windows –rotate support
- improved log timestamping, now with millisecond precision

Querying

- added extended engine V2 (faster, cleaner, better; SPH_MATCH_EXTENDED2 mode)
- added ranking modes support (V2 engine only; SetRankingMode() API call)
- added quorum searching support to query language (V2 engine only; example: "any three of all these words"/3)
- added query escaping support to query language, and EscapeString() API call
- added multi-field syntax support to query language (example: "@(field1,field2) something"), and @@relaxed field checks option
- added optional star-syntax ('word*') support in keywords, enable_star directive (for prefix/infix indexes only)
- added full-scan support (query must be fully empty; can perform block-reject optimization)
- added COUNT(DISTINCT(attr)) calculation support, SetGroupDistinct() API call

- added group-by on MVA support, SetArrayResult() PHP API call
- added per-index weights feature, SetIndexWeights() API call
- added geodistance support, SetGeoAnchor() API call
- added result set sorting by arbitrary expressions in run time (eg. “@weight+log(price)*2.5″), SPH_SORT_EXPR mode
- added result set sorting by @custom compile-time sorting function (see src/sphinxcustomsort.inl)
- added result set sorting by @random value
- added result set merging for indexes with different schemas
- added query comments support (3rd arg to Query()/AddQuery() API calls, copied verbatim to query log)
- added keyword extraction support, BuildKeywords() API call
- added binding field weights by name, SetFieldWeights() API call
- added optional limit on query time, SetMaxQueryTime() API call
- added optional limit on found matches count (4rd arg to SetLimits() API call, so-called ‘cutoff’)

APIs and SphinxSE

- added pure C API (libsphinxclient)
- added Ruby API (thanks to Dmytro Shteflyuk)
- added Java API
- added SphinxSE support for MVAs (use varchar), floats (use float), 64bit docids (use bigint)
- added SphinxSE options “floatrange”, “geoanchor”, “fieldweights”, “indexweights”, “maxquerytime”, “comment”, “host” and “port”; and support for “expr:CLAUSE”
- improved SphinxSE max query size (using MySQL condition pushdown), upto 256K now

General

- added scripting (shebang syntax) support to config files (example: #!/usr/bin/php in the first line)
- added unified config handling and validation to all programs
- added unified documentation
- added .spec file for RPM builds
- added automated testing suite
- improved index locking, now fcntl()-based instead of buggy file-existence-based
- fixed unaligned RAM accesses, now works on SPARC and ARM

Changes and fixes since 0.9.8-rc2

- added pure C API (libsphinxclient)
- added Ruby API
- added SetConnectTimeout() PHP API call
- added allowed type check to UpdateAttributes() handler (bug #174)
- added defensive MVA checks on index preload (protection against broken indexes, bug #168)
- added sphinx-min.conf sample file
- added –without-iconv switch to configure
- removed redundant -lz dependency in searchd
- removed erroneous “xmlpipe2 deprecated” warning
- fixed EINTR handling in piped read (bug #166)
- fixup query time before logging and sending to client (bug #153)
- fixed attribute updates vs full-scan early-reject index (bug #149)
- fixed gcc warnings (bug #160)
- fixed mysql connection attempt vs pgsql source type (bug #165)
- fixed 32-bit wraparound when preloading over 2 GB files
- fixed “out of memory” message vs over 2 GB allocs (bug #116)
- fixed unaligned RAM access detection on ARM (where unaligned reads do not crash but produce wrong results)
- fixed missing full scan results in some cases
- fixed several bugs in –merge, –merge-dst-range
- fixed @geodist vs MultiQuery and filters, @expr vs MultiQuery
- fixed GetTokenEnd() vs 1-grams (was causing crash in excerpts)
- fixed sql_query_range to handle empty strings in addition to NULL strings (Postgres specific)
- fixed morphology=none vs infixes
- fixed case sensitive attributes names in UpdateAttributes()
- fixed ext2 ranking vs. stopwords (now using atompos from query parser)
- fixed EscapeString() call
- fixed escaped specials (now handled as whitespace if not in charset)
- fixed schema minimizer (now handles type/size mismatches)
- fixed word stats in extended2; stemmed form is now returned

- fixed spelldump case folding vs dictionary-defined character sets
- fixed Postgres BOOLEAN handling
- fixed enforced “inline” docinfo on empty indexes (normally ok, but index merge was really confused)
- fixed rare count(distinct) out-of-bounds issue (it occasionaly caused too high @distinct values)
- fixed hangups on documents with id=DOCID_MAX in some cases
- fixed rare crash in tokenizer (prefixed synonym vs. input stream eof)
- fixed query parser vs “aaa (bbb ccc)|ddd” queries
- fixed BuildExcerpts() request in Java API
- fixed Postgres specific memory leak
- fixed handling of overshort keywords (less than min_word_len)
- fixed HTML stripper (now emits space after indexed attributes)
- fixed 32-field case in query parser
- fixed rare count(distinct) vs. querying multiple local indexes vs. reusable sorter issue
- fixed sorting of negative floats in SPH_SORT_EXTENDED mode

A.8. Version 0.9.7, 02 apr 2007

- added support for `sql_str2ordinal_column`
- added support for upto 5 sort-by attrs (in extended sorting mode)
- added support for separate groups sorting clause (in group-by mode)
- added support for on-the-fly attribute updates (PRE-ALPHA; will change heavily; use for preliminary testing ONLY)
- added support for zero/NULL attributes
- added support for 0.9.7 features to SphinxSE
- added support for n-grams (alpha, 1-grams only for now)
- added support for warnings reported to client
- added support for exclude-filters
- added support for prefix and infix indexing (see `max_prefix_len`, `max_infix_len`)
- added @* syntax to reset current field to query language
- added removal of duplicate entries in query index order
- added PHP API workarounds for PHP signed/unsigned braindamage
- added locks to avoid two concurrent indexers working on same index
- added check for existing attributes vs. `docinfo=none` case
- improved groupby code a lot (better precision, and upto 25x times faster in extreme cases)
- improved error handling and reporting
- improved handling of broken indexes (reports error instead of hanging/crashing)
- improved `mmap()` limits for attributes and wordlists (now able to map over 4 GB on x64 and over 2 GB on x32 where possible)
- improved `malloc()` pressure in head daemon (search time should not degrade with time any more)
- improved `test.php` command line options
- improved error reporting (distributed query, broken index etc issues now reported to client)
- changed default network packet size to be 8M, added extra checks
- fixed division by zero in BM25 on 1-document collections (in extended matching mode)
- fixed `.spl` files getting unlinked
- fixed crash in schema compatibility test
- fixed UTF-8 Russian stemmer
- fixed requested matches count when querying distributed agents
- fixed signed vs. unsigned issues everywhere (ranged queries, CLI search output, and obtaining docid)
- fixed potential crashes vs. negative query offsets
- fixed 0-match docs vs. extended mode vs. stats
- fixed group/timestamp filters being ignored if querying from older clients
- fixed docs to mention `pgsql` source type
- fixed issues with explicit ‘&’ in extended matching mode
- fixed wrong assertion in SBCS encoder
- fixed crashes with no-attribute indexes after rotate

A.9. Version 0.9.7-rc2, 15 dec 2006

- added support for extended matching mode (query language)
- added support for extended sorting mode (sorting clauses)
- added support for SBCS excerpts
- added `mmap()ing` for attributes and wordlist (improves search time, speeds up `fork()` greatly)
- fixed attribute name handling to be case insensitive
- fixed default compiler options to simplify post-mortem debugging (added `-g`, removed `-fomit-frame-pointer`)
- fixed rare memory leak
- fixed “hello hello” queries in “match phrase” mode
- fixed issue with excerpts, texts and overlong queries

- fixed logging multiple index name (no longer tokenized)
- fixed trailing stopword not flushed from tokenizer
- fixed boolean evaluation
- fixed pidfile being wrongly `unlink()ed` on `bind()` failure
- fixed `--with-mysql-includes/libs` (they conflicted with well-known paths)
- fixes for 64-bit platforms

A.10. Version 0.9.7-rc1, 26 oct 2006

- added alpha index merging code
- added an option to decrease `max_matches` per-query
- added an option to specify IP address for searchd to listen on
- added support for unlimited amount of configured sources and indexes
- added support for group-by queries
- added support for /2 range modifier in charset_table
- added support for arbitrary amount of document attributes
- added logging filter count and index name
- added `--with-debug` option to configure to compile in debug mode
- added `-DNDEBUG` when compiling in default mode
- improved search time (added doclist size hints, in-memory wordlist cache, and used VLB coding everywhere)
- improved (refactored) SQL driver code (adding new drivers should be very easy now)
- improved exceprts generation
- fixed issue with empty sources and ranged queries
- fixed querying purely remote distributed indexes
- fixed suffix length check in English stemmer in some cases
- fixed UTF-8 decoder for codes over U+20000 (for CJK)
- fixed UTF-8 encoder for 3-byte sequences (for CJK)
- fixed overshort (less than `min_word_len`) words prepended to next field
- fixed source connection order (indexer does not connect to all sources at once now)
- fixed line numbering in config parser
- fixed some issues with index rotation

A.11. Version 0.9.6, 24 jul 2006

- added support for empty indexes
- added support for multiple sql_query_pre/post/post_index
- fixed timestamp ranges filter in “match any” mode
- fixed configure issues with –without-mysql and –with-pgsql options
- fixed building on Solaris 9

A.12. Version 0.9.6-rc1, 26 jun 2006

- added boolean queries support (experimental, beta version)
- added simple file-based query cache (experimental, beta version)
- added storage engine for MySQL 5.0 and 5.1 (experimental, beta version)
- added GNU style `configure` script
- added new searchd protocol (all binary, and should be backwards compatible)
- added distributed searching support to searchd
- added PostgreSQL driver
- added excerpts generation
- added `min_word_len` option to index
- added `max_matches` option to searchd, removed hardcoded MAX_MATCHES limit
- added initial documentation, and a working `example.sql`
- added support for multiple sources per index
- added soundex support
- added group ID ranges support
- added `--stdin` command-line option to search utility
- added `--noprogress` option to indexer
- added `--index` option to search
- fixed UTF-8 decoder (3-byte codepoints did not work)
- fixed PHP API to handle big result sets faster
- fixed config parser to handle empty values properly
- fixed redundant `time(NULL)` calls in time-segments mode