7. 掌握SESSION入库的实现(原理、数据库设计、应用场景)

随着 session 的增加，管理已经不方便。这时，考虑使用 mysql 数据库管理。建立一个表管理 session 。 更改 session 的存储机制，让 session 不再存在文件中，而是入库。更该存储机制，只需要在文件中增加函数 session\_set\_save\_handler() 便可。代码如下：

// 重写 session 的存储机制

function sess\_open(){

  echo \_\_FUNCTION\_\_;

}

function sess\_close(){

  echo \_\_FUNCTION\_\_;

}

function sess\_read($sess\_id){

  echo \_\_FUNCTION\_\_;

}

function sess\_write($sess\_id, $sess\_data){

  echo \_\_FUNCTION\_\_;

}

function sess\_destroy(){

  echo \_\_FUNCTION\_\_;

}

function sess\_gc(){

  echo \_\_FUNCTION\_\_;

}

session\_set\_save\_handler(

  'sess\_open',

  'sess\_close',

  'sess\_read',

  'sess\_write',

  'sess\_destroy',

  'sess\_gc'

);

session\_start();

这要写了 session\_set\_save\_handler() ，php 就知道要更改 session 的存储机制了。如果没写，那还是存到文件中。这函数的参数，分别对应了 session 运行机制的各个方面。参数名字，也就是函数名，可以改变，不一定这样写，这样写是为了便于阅读理解。 执行顺序也不是从上到下，而是：open -> reader -> write -> close .

总结 session 运行机制：

1. 打开 session 时，语法上执行函数 session\_start() ，php 的 session 机制读取浏览器端的 cookie，语法上表示为 $\_cookie['PHPSESSID']。

2. 根据 cookie 找到存储在服务器端的 session 数据。

3. 把 session 数据反序列化，赋值给变量 $\_SESSION 。

4. 之后对变量 $\_SESSION 的操作都是对变量的操作，不会更新 session 文件。

5. 是否执行了 session\_destroy() 函数，如果执行了，那么删除服务器端的 session 文件。

6. 脚本结束时，判断是否有 sessin 文件，或者说是否执行过 session\_destroy() 方法。如果没有执行过，则把 $\_SESSION 变量中的数据写入到 session 文件中。如果执行过，那么什么也不做。

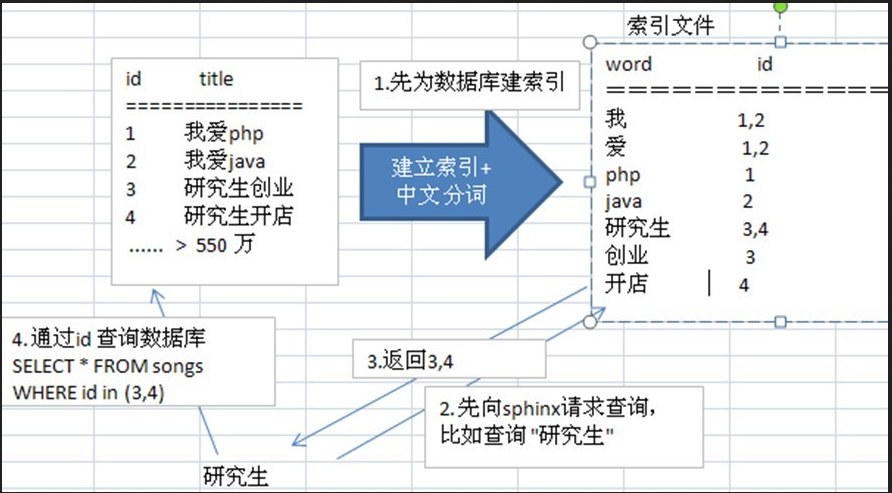
8. 掌握sphinx服务器的搭建、检索实现原理

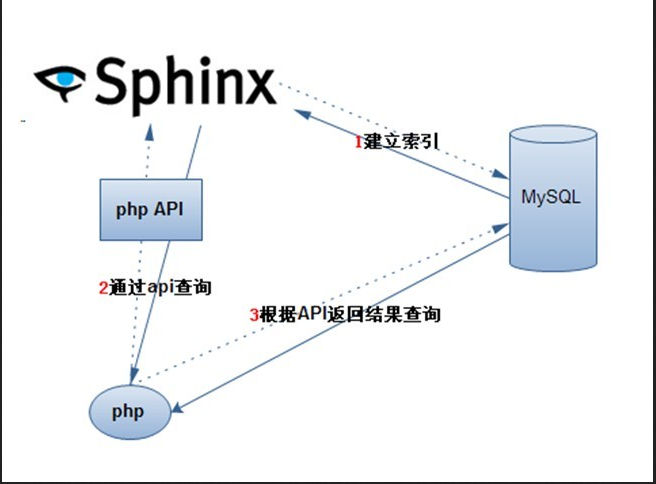
在 使用mysql数据库过程中，如果想实现全文检索的优化，可以使用mysql自带全文索引，但是不支持中文。。关于sphinx的安装网上很多教程写的都 不错比如：http://www.coreseek.cn/products-install/。这里就不再说明安装方法了。有兴趣的可以自己参考。

     MySQL在高并发连接、数据库记录数较多的情况下，SELECT ... WHERE ... LIKE '%...%'的全文搜索方式不仅效率差，而且以通配符%和\_开头作查询时，使用不到索引，需要全表扫描，对数据库的压力也很大。MySQL针对这一问题提供了一种全文索引解决方案，这不仅仅提高了性能和效率（因为MySQL对这些字段做了索引来优化搜索），而且实现了更高质量的搜索。但是，至今为止，MySQL对中文全文索引无法正确支持。

可 以使用Sphinx（一种全文检索引擎）技术，Sphinx默认不支持中文索引及检索。以前用Coreseek的补丁来解决，目前Coreseek 不单独提供补丁文件，而基于sphinx开发了Coreseek 全文检索服务器，Coreseek应该是现在用的最多的sphinx中文全文检索，它提供了为Sphinx设计的中文分词包LibMMSeg包含 mmseg中文分词。

**Sphinx的基本原理与检索流程**





这种检索流程使用sphinx官方为我们提供的API文件（php使用sphinxapi.php），首先php通过这个api连接sphinx服务，获取查询结果的id信息，然后再通过这些id从mysql数据库中 取得相关的数据。

**sphinx分布式索引原理：**

当searchd收到一个对分布式索引的查询时，它做如下操作：

1. 连接到远程代理

2. 执行查询

3. （在远程代理执行搜索的同时）对本地索引进行查询

4. 接收来自远程代理的搜索结果

5. 将所有结果合并，删除重复项

6. 将合并后的结果返回给客户端

在应用程序看来，普通索引和分布式索引完全没有区别。

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。

更好的HA支持已在计划之中，到时将允许指定哪些代理之间互相备份、有效性检查、跟踪

运行中的代理、对检索请求进行负载均衡，等等。