

# 大规模分布式搜索引擎Search

庞磊 | 2020年11月

# 目录 > CONTENTS

- 1 Search简介
- 2 Search原理
- 3 Search安装与配置
- 4 Search使用

The background features a complex network diagram with numerous nodes (dots) and connecting lines, forming a spherical shape on the left and right sides of the slide. The nodes are of varying sizes and are connected by thin, light gray lines, creating a web-like structure.

# 1 chapter

## Search简介

- ✓ 什么是Search
- ✓ 适用场景

### ➤ 概念

- 基于ElasticSearch的大规模分布式搜索引擎
  - 通过Esdrive实现了SQL的全方位支持
  - 对ElasticSearch进行了全面升级和深度优化

### ➤ 特点

- 分布式实时存储
  - 独创的分层存储结构，支持“内存/SSD/SATA分层存储”
  - 堆外内存管理，单节点存储能力从6~8TB提升至30TB
  - 支持结构化、半结构化和非结构化数据
  - PB级数据规模
- 分布式实时搜索
  - 冷热数据分级存储，毫秒级实时关键字检索
  - 计算紧贴数据和索引，亚秒级在线分析



### ➤ 特点

- SQL引擎与搜索引擎相融合
  - 通过Inceptor和Esdrive，实现以SQL方式创建索引和全文搜索
- 高扩展
  - 支持上百个节点的横向线性扩展

### ➤ 在TDH平台中，Search扮演两种角色

- 作为Hyperbase全文检索的底层实现
- 作为独立的服务，既是一个分布式文件存储系统，又是一个强大的搜索引擎

### ➤ 文档数据库

- 存储半结构化、非结构化数据
- 功能和性能均优于Mongodb

### ➤ 日志分析与监控

- 统计和日志类时间序列数据的存储和分析


### ➤ 舆情分析

- 高聚合率的统计分析，如：热词跟踪

### ➤ 搜索引擎

- 多条件模糊查询
- 语义查询
- 不指定列的全文搜索





# 2 chapter

## Search原理

- ✓ 数据模型
- ✓ 分词与索引
- ✓ 系统架构
- ✓ 访问机制



### ➤ Index (索引)

- Search以Index为单位组织数据（Document），一个Index中的数据通常具有相似的特征
  - 例如：为员工信息创建一个Index，或为商品信息创建一个Index
- 与关系数据库的索引不是一个概念，这里是指Search的数据对象

### ➤ Type (分类)

- Type是Index的逻辑分类，如何分类由用户决定，一个Index可定义一个或多个Type
  - 例如：员工信息Index可按部门分类，包括财务部Type、销售部Type、研发部Type等

### ➤ Document (文档)

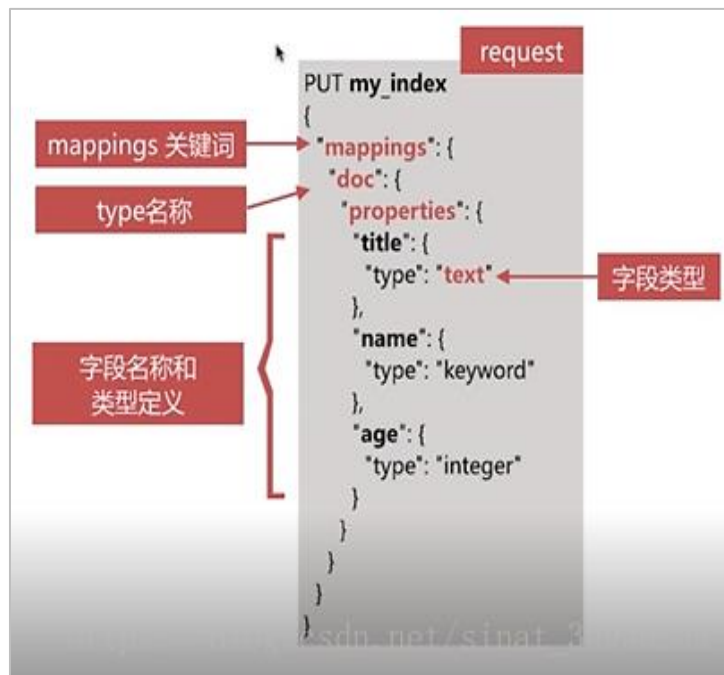
- Search的最基础数据单元，以JSON格式存储
  - 例如：员工的基本信息{"name": "zhangsan", "age": 30, "on\_board\_data": "2016-10-01", "address": {"province": "jiangsu", "city": "nanjing", ...} ...}可作为一个Document，保存到员工信息Index中

- Field（字段）
  - Document中的数据存储在Field中
- Search数据对象与关系数据库的映射关系
  - Type是Index的逻辑分类，不映射为关系数据库中的数据对象

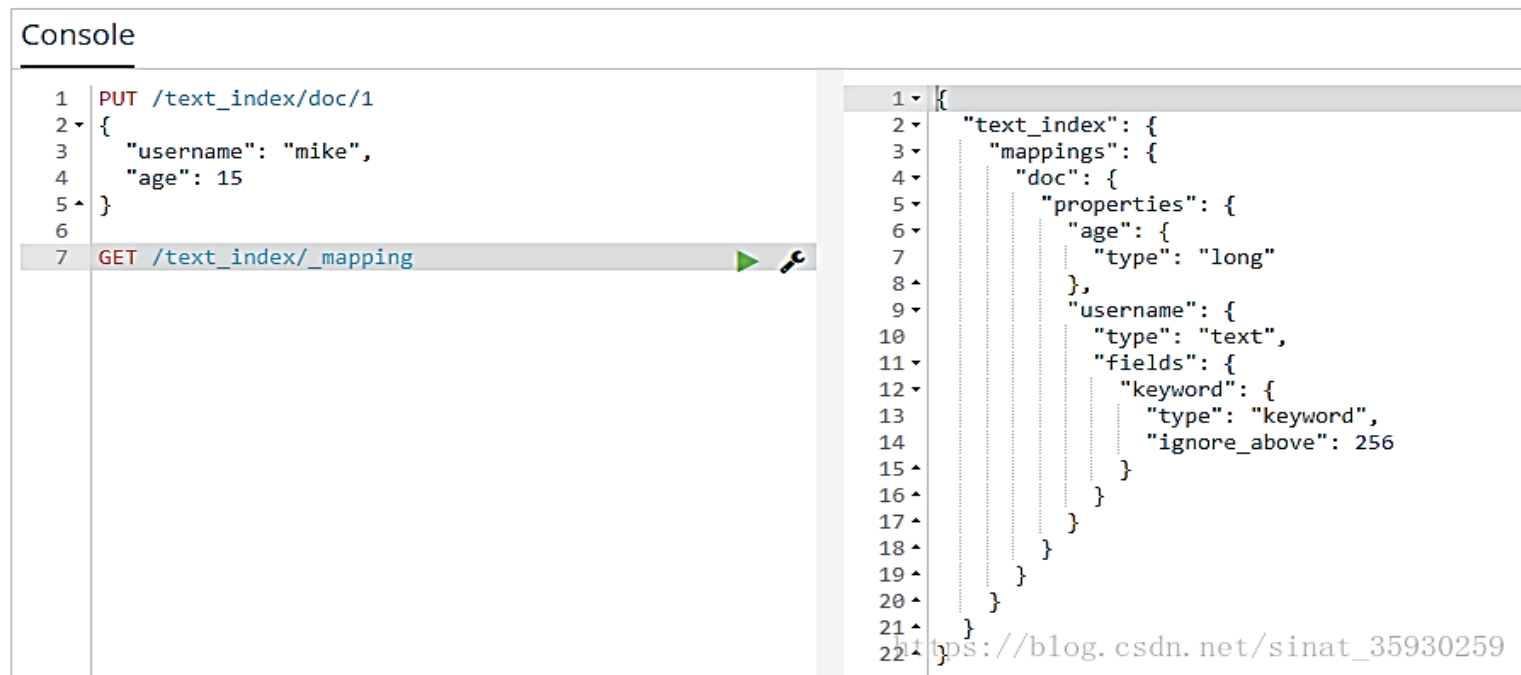
Search	关系数据库
Index（索引）	Table（表）
Document（文档）	Row（行）
Field（字段）	Column（列）

## ➤ Mapping (映射)

- 相当于关系数据库中的表结构定义 (Schema)
  - 定义Index的字段名和字段类型, 比如: 数值型、浮点型、布尔型等
  - 定义字段的索引设置, 比如: 是否索引、分词器、记录position等
  - 可以自定义创建, 也可以根据数据自动创建



自定义创建



根据数据自动创建

### ➤ 分词

- 基本过程

- 将文档拆分成一组单独的词（term）
- 将词转换为标准形式，以提高查全率，如：电脑 → 计算机

- 分词器

- 英文分词器：standard、English
- 中文分词器：ik、mmseg
- 不同的分词器会产生不同的分词结果，产生不同的索引，所以相同的查询条件会产生不同的结果

### ➤ 倒排索引

- 假设有两篇文档

- 文档1的内容：Tom lives in Shanghai, I live in Shanghai too.
- 文档2的内容：He once lived in Beijing.

### ➤ 倒排索引

- 对文档进行分词
  - 文档1的分词结果: [Tom] [live] [Shanghai] [i] [live] [Shanghai]
  - 文档2的分词结果: [he] [live] [Beijing]
- 根据分词结果, 构建倒排索引

词	文档号[出现频率]	出现位置
shanghai	1[2]	3, 6
he	2[1]	1
i	1[1]	4
live	1[2], 2[1]	2, 5, 2
beijing	2[1]	3
tom	1[1]	1



### ➤ 节点与集群

- 节点：一个运行中的ElasticSearch实例
- 集群：由一个或多个拥有相同cluster.name配置的节点组成，它们共同承担数据和负载压力；  
当节点加入集群或从集群中移除时，会进行数据重分布

### ➤ 主节点 (MasterNode)

- 负责管理集群内的所有变更，如增删节点、增删索引、分配分片等，不负责文档更新和搜索
- 每个集群只有一个主节点，默认情况下任何节点都可能被选为主节点
- 硬件配置：普通服务器（CPU、内存消耗一般）

### ➤ 数据节点 (DataNode)

- 负责存储数据，即文档的增删改查
- 分离主节点和数据节点是一个比较好的选择，因为索引和搜索操作会消耗大量资源
- 硬件配置：较高配置服务器（主要消耗磁盘和内存）

### ➤ 客户端节点 (ClientNode / 路由节点)

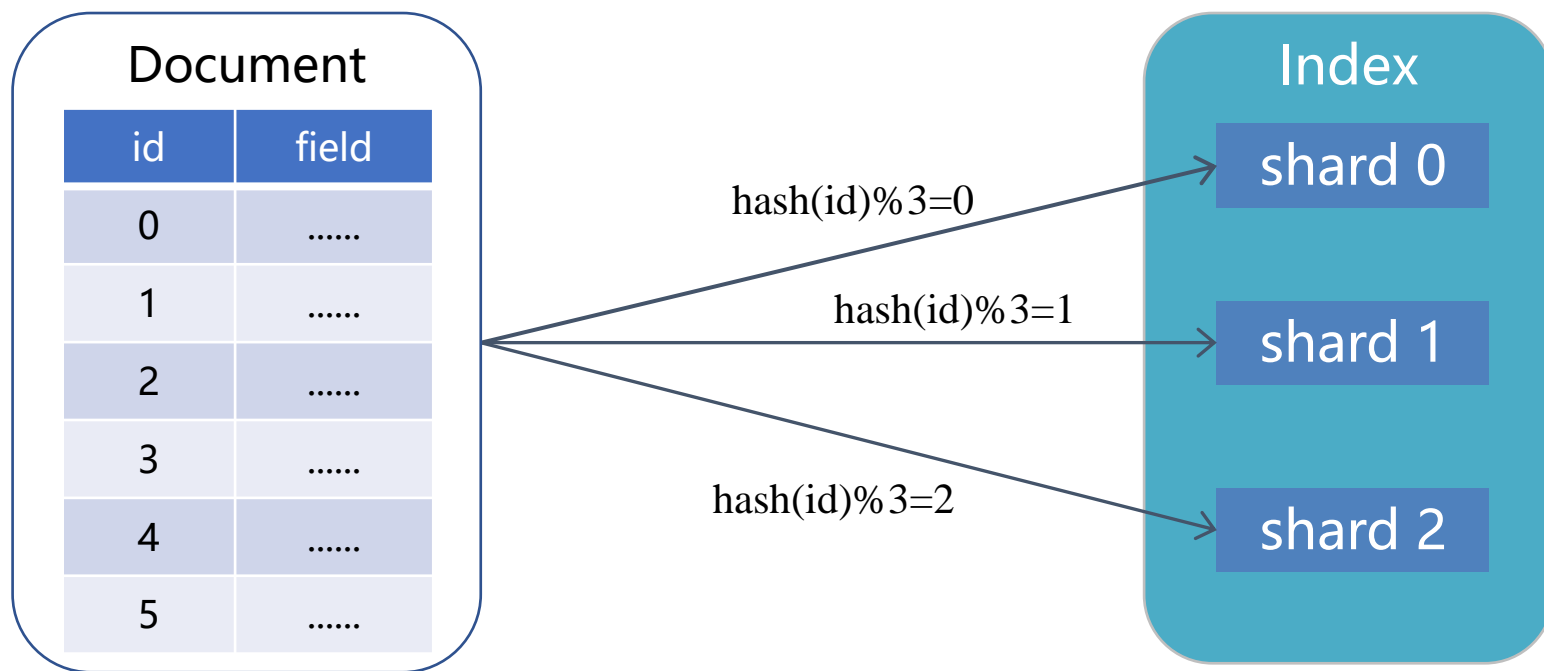
- 负责路由请求，实现集群访问的负载均衡
- 集群规模较大时非常有用，协调主节点和数据节点，根据集群状态直接路由请求

### ➤ 分片 (Shard)

- Shard是Search的数据存储单元，是数据的容器，Document保存在Shard中
- 每个Shard都是一个Lucene实例，是一个完整的单机搜索引擎，提供对外读写能力
- Index是逻辑概念，Shard是物理概念，创建Index时会指定划分为一个或多个Shard，然后分布到集群的各节点中，Document通过“哈希取模”（ $\text{Shard编号} = \text{hash}(\text{Document id}) \% \text{Shard数量}$ ）的方式分配到不同的Shard中
- Shard不可分割，但可以收缩（Shrink），可将分片数进行收缩成它的因数（如15个Shard可收缩成5/3/1个），这样在文档批量写入时，通过设置足够多的Shard，提升并行写入效率，当写操作完成后，收缩为少量的Shard，提高查询性能

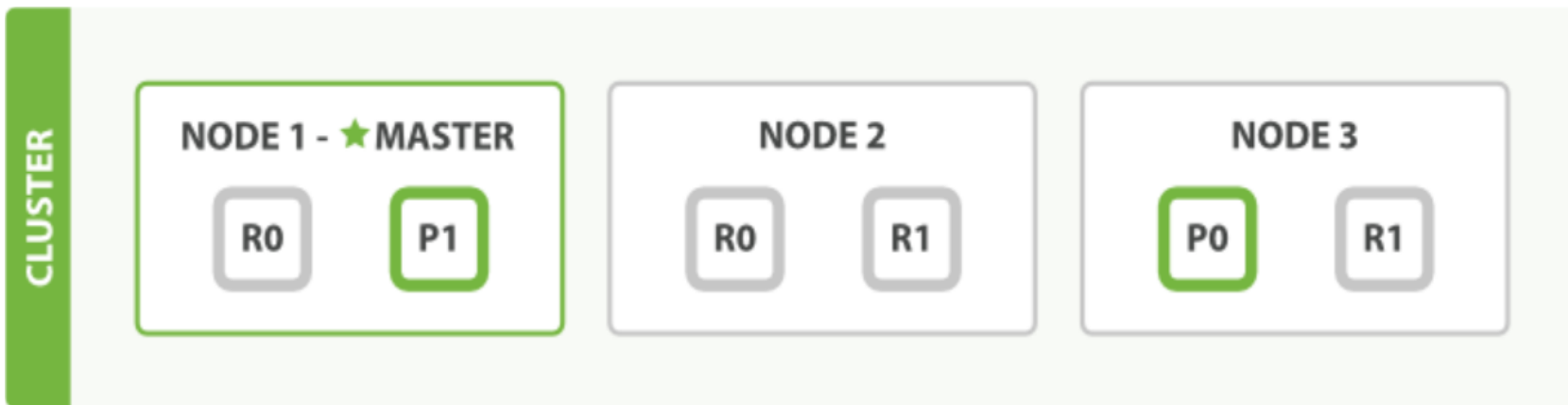
### ➤ 分片 (Shard)

- 当集群规模扩大或缩小时，系统会自动迁移分片，使数据均匀分布
- 虽然Document被存储或索引到Shard中，但应用程序直接与Index而不是Shard进行交互



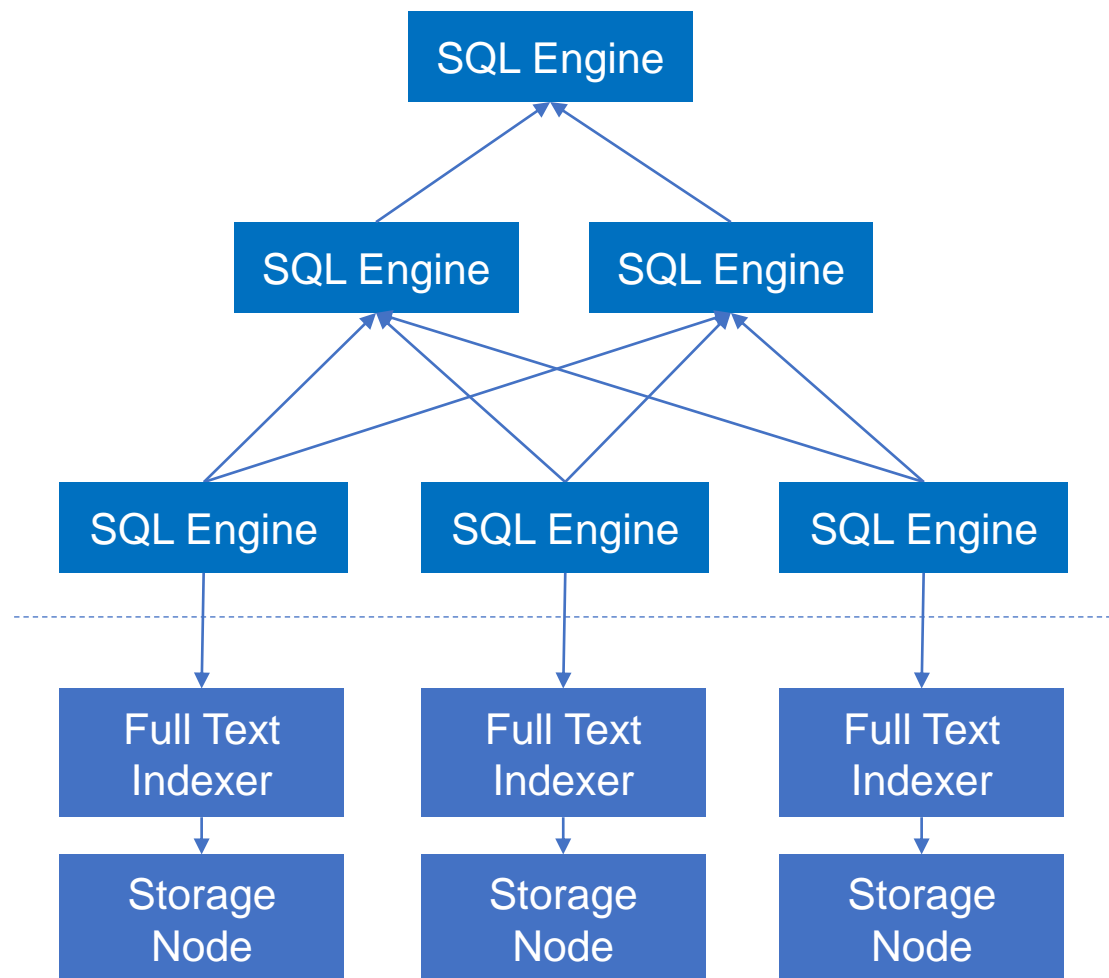
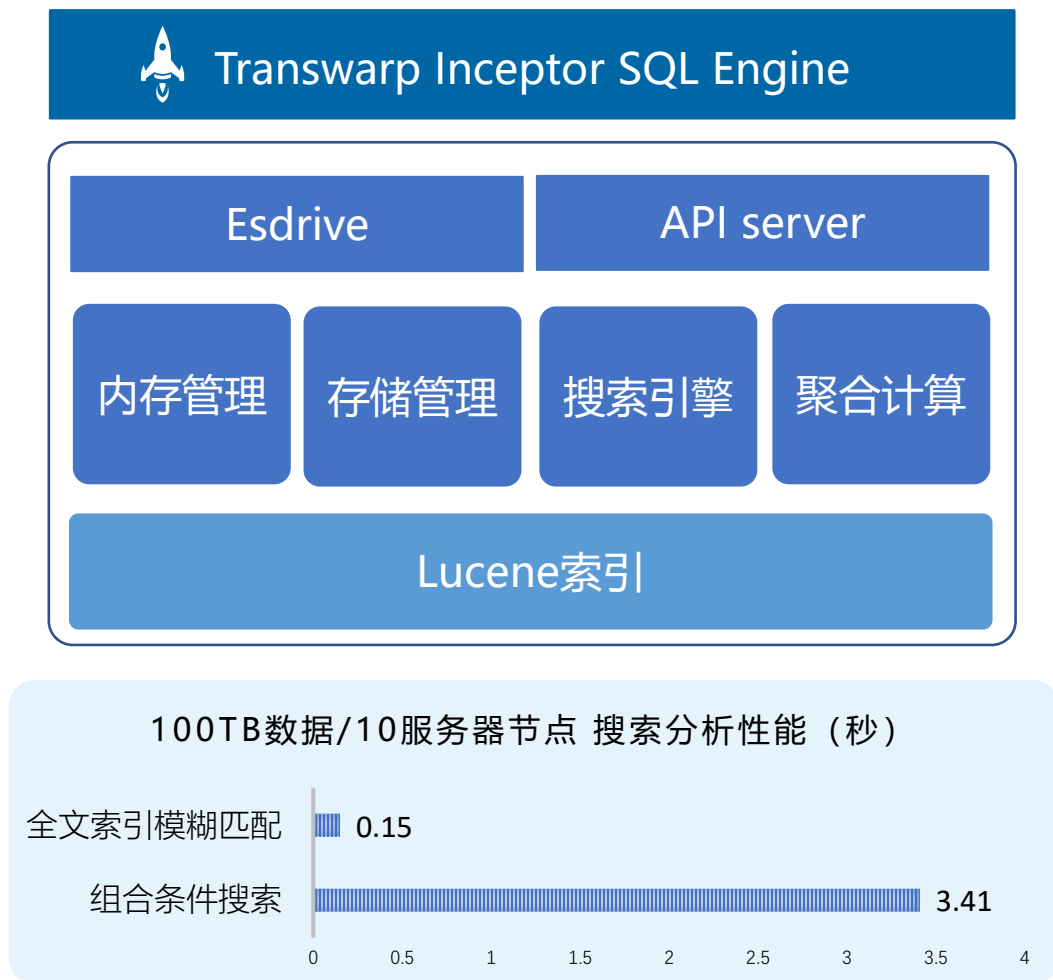
### ➤ 分片副本 (Shard Replica)

- Shard分为主Shard和副本Shard，后者是前者的精确复制，每个Shard可有零个或多个副本
- Index的任意一个Document都归属于一个主Shard，主Shard的数量决定了Index的最大数据量
- Index建立时必须明确主Shard数且不能修改，但副本Shard数可以随时修改
- 写操作只能被主Shard处理，读操作可同时被主Shard或副本Shard处理
  - 对于读操作，理论上拥有更多的副本，将拥有更高的吞吐量，但如果只在相同节点数目的集群上增加副本并不能提高性能，因为每个Shard获得的资源会变少，这时需要增加更多的硬件资源来提升吞吐量



集群示例

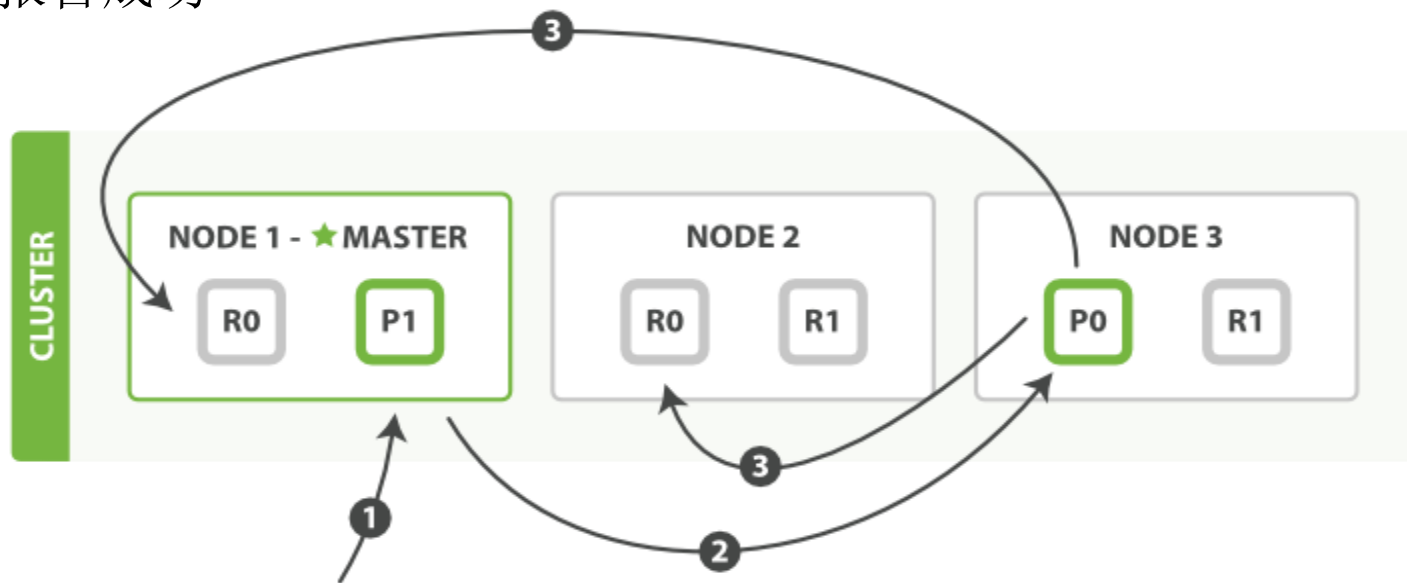
## ➤ Search系统架构





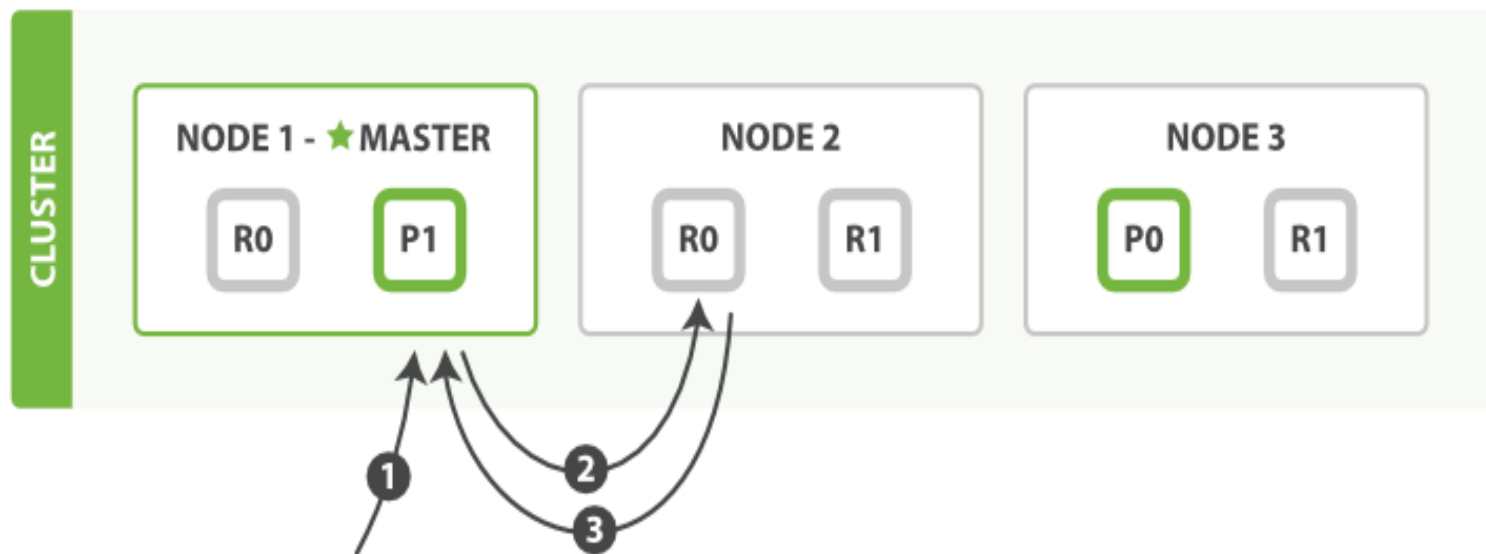
### ➤ 更新文档

- (1) 客户端向Node1（路由节点）发送新建、索引或删除文档请求
- (2) 通过文档id确定该文档属于分片0，请求被转发到Node3，因为分片0的主分片在Node3上
- (3) Node3在主分片上执行更新操作，如果成功了，Node3将请求并行转发到Node1和Node2的副本分片上，一旦所有副本分片都报告同步成功，Node3将向Node1报告更新成功，最后Node1向客户端报告成功




### ➤ 读取文档

- (1) 客户端向Node 1（路由节点）发送读取请求
- (2) Node 1通过文档id确定文档属于分片0，分片0的副本存储于所有的三个节点上，这种情况下它将请求转发到Node 2
- (3) Node 2将目标文档返回给Node 1，然后将文档返回给客户端



### ➤ 读取文档

- 路由节点在响应请求的时候，会选择不同的副本分片来达到负载均衡，即轮询所有的副本分片
- 文档被读取时，已经被索引的文档可能存在于主分片上，但还没复制到副本分片，这种情况下，副本分片可能会报文档不存在，但主分片能成功返回文档
- 一旦索引操作成功返回给用户，那么文档在主分片和副本分片都是可用的



# 3 chapter

## Search安装与配置

- ✓ 系统安装
- ✓ 系统配置与管理

1.选择服务

2.分配角色

3.配置服务

4.配置安全

5.服务总览

6.安装

选择想要添加的服务，可能存在某些由于依赖服务未添加而无法添加的服务，请先添加其依赖的服务。

<input type="checkbox"/> <b>ZOOKEEPER</b> ZooKeeper用于协调同步其他服务	<input type="checkbox"/> <b>HDFS</b> HDFS是Hadoop应用的基本存储系统	<input type="checkbox"/> <b>YARN</b> YARN是资源管理框架
<input checked="" type="checkbox"/> <b>SEARCH</b> Search是一个分布式的搜索分析引擎	<input type="checkbox"/> <b>HYPERBASE</b> Hyperbase是实时在线事务处理引擎	<input type="checkbox"/> <b>TXSQL</b> TxSQL是一个分布式的关系型数据库
<input type="checkbox"/> <b>NOTIFICATION</b> 通知组件是一个提供相关组件中产生的消息的存储，查询，记录组件的工作状态的服务	<input type="checkbox"/> <b>INCEPTOR</b> Inceptor是基于内存的交互式SQL分析引擎	<input type="checkbox"/> <b>TRANSPORTER</b> TDT是数据整合工具
<input type="checkbox"/> <b>TRANSPEDIA</b> Transpedia是TDH产品的文档检索服务	<input type="checkbox"/> <b>OOZIE</b> Oozie是Hadoop的一个工作流调度系统	<input type="checkbox"/> <b>SQOOP</b> Sqoop 用于在Hadoop和其他结构化数据存储中传送数据
<input type="checkbox"/> <b>SOPHON</b> Sophon是一个深度学习交互式分析工具	<input type="checkbox"/> <b>SLIPSTREAM</b> Slipstream是基于Spark的实时流处理引擎	<input type="checkbox"/> <b>DISCOVER</b> Discover是基于内存的数据挖掘引擎
<input type="checkbox"/> <b>KAFKA</b>	<input type="checkbox"/> <b>WORKFLOW</b> Workflow是图形化的工作流设计、调试、调度和分析的	<input type="checkbox"/> <b>RUBIK</b>

取消

下一步



1.选择服务

2.分配角色

3.配置服务

4.配置安全

5.服务总览

6.安装

本页面为选中的服务分配角色，默认已按推荐方法分配，如果想要修改分配，先点选左侧菜单的一个角色，再在中间面板上选中/取消从而为该角色分配节点

▼ Search2

Search Server

搜索主机...

3 个节点在 /default-rack

☐

☒

☐

全选

全不选

全选

全不选

◀ 上一步

下一步 ▶



1.选择服务

2.分配角色

3.配置服务

4.配置安全

5.服务总览

6.安装

配置选中的服务

属性

基础参数

自定义参数

Search2

基础参数

值

path.data

<节点特定, 请点击以分别修改>

node.master

<节点特定, 请点击以分别修改>

node.data

<节点特定, 请点击以分别修改>

http.port

9200

transport.tcp.port

9300

network.bind\_host

<节点特定, 请点击以分别修改>

network.publish\_host

<节点特定, 请点击以分别修改>

es.heap.size

<节点特定, 请点击以分别修改>

search.container\_limits.memory

◀ 上一步

下一步 ▶



TRANSWARP  
DATA HUB

服务 管理 应用市场

Search1

HEALTHY

主 页 > Search1

可在本页编辑服务的配置。修改或者增加配置项后请点击“保存更改”按钮，更改的配置项才被保存。

搜索配置项...

全部配置

配置项	配置类型	配置文件	值	描述
cluster.name	预定义		cluster	恢复推荐值
cluster.routing.allocation.balance.index	预定义		0.999	恢复推荐值
cluster.routing.allocation.balance.shard	预定义		0.001	恢复推荐值
discovery.zen.fd.ping_interval	预定义		30s	恢复推荐值
discovery.zen.fd.ping_retries	预定义		6	恢复推荐值
discovery.zen.fd.ping_timeout	预定义		120s	恢复推荐值
discovery.zen.minimum_master_nodes	预定义		1	恢复推荐值
discovery.zen.ping_timeout	预定义		100s	恢复推荐值
es.heap.size	预定义		<节点特定，请点击以分别修改>	
gateway.recover_after_time	预定义		5m	恢复推荐值
head.port	预定义		9100	恢复推荐值
http.port	预定义		9200	恢复推荐值
index.shard.path.selector	预定义		tiered	恢复推荐值
index.store.type	预定义		niofs	恢复推荐值

Elasticsearch  连接 cluster 集群健康值: green (20 of 20) 信息

概览 索引 数据浏览 基本查询 [+] 复合查询 [+] 刷新

集群概览 集群排序 View Aliases Index Filter


**governor**  
size: 83.4ki (164ki)  
docs: 41 (82)  
信息 动作

● tdh-12	0	1	2	4	6	7	9
● tdh-13	1	2	3	5	6	8	
★ tdh-14	0	3	4	5	7	8	9

信息 动作

governor/oB8zMFYNT4mwcijpV9L8Xg [0]

```
{
  "state": "STARTED",
  "primary": false,
  "node": "oB8zMFYNT4mwcijpV9L8Xg",
  "relocating_node": null,
  "shard": 0,
  "index": "governor",
  "version": 10,
  "allocation_id": {
    "id": "6D2dNuTNSnuJS0OnwE_gzQ"
  }
}
```

The background of the slide features a complex network diagram. It consists of numerous small, dark grey circular nodes connected by thin, light grey lines. These connections form a dense web of triangles and other geometric shapes, creating a sense of interconnectedness. The overall color palette is light grey and white, with the dark grey nodes providing contrast.

# 4 chapter

## Search使用

- ✓ REST API
- ✓ Esdrive SQL



## ➤ 访问方法

- 在命令行中通过curl命令访问REST API
- 访问端口默认为9200

## ➤ 示例

- 任务：搭建一个员工信息Index（Index名为employee）
  - Index中的每个Document对应一名员工信息
  - Index的Type按部门分类：dev（研发部）、finance（财务部）和sales（销售部）
- 第1步：创建名为employee的Index

```
/* 创建Index employee， pretty表示以JSON格式返回结果 */  
curl -XPUT 'localhost:9200/employee/?pretty'  
/* 返回结果：true表示Index创建成功 */  
{  
  "acknowledged" : true  
}
```

## ➤ 示例

- 第2步：编入Document，即向Index中新增一条员工信息

```
/* 向Index employee的Type dev下编入Document */  
curl -XPUT 'localhost:9200/employee/dev/1?pretty' -d '{  
  "firstname": "San",  
  "lastname": "Zhang",  
  "age": 26,  
  "on_board_date": "2015-10-31",  
  "hometown": "Beijing",  
  "school": "Nanjing University",  
  "married": false,  
  "about": "I love Beijing Opera"  
}'
```

```
/* 返回结果：新增Document的元数据（JSON格式） */  
{  
  "_index": "employee", // index名  
  "_type": "dev",       // type名  
  "_id": "1",           // id  
  "_version": 1,        // 版本号  
  "_shards": {          // Shard信息  
    "total": 2,  
    "successful": 2,  
    "failed": 0  
  },  
  "created": true       // 创建成功  
}
```

## ➤ 示例

- 第3步：查看Document是否存在

```
/* 查看/employee/dev/1下是否存在Document, -i表示打印HTTP header */  
curl -i -XHEAD 'localhost:9200/employee/dev/1'
```

- 第4步：获取Document

```
/* 获取Index employee的Type dev下的Document */  
curl -XGET 'localhost:9200/employee/dev/1?pretty'  
/* 返回结果：Document对象（JSON格式） */  
{  
  "_index": "employee", "_type": "dev", "_id": "1", "_version": 1, "found": true,  
  "_source": {  
    "firstname": "San", "lastname": "Zhang", "age": 26, "on_board_date": "2015-10-31",  
    "hometown": "Beijing", "school": "Nanjing University", "married": false, "about": "I love Beijing Opera"  
  }  
}
```

## ➤ 示例

### • 第5步：更新Document

```
/* 将Document中的age改为30 */  
curl -XPUT 'localhost:9200/employee/dev/1?pretty' -d '{  
  "firstname": "San",  
  "lastname": "Zhang",  
  "age": 30,  
  "on_board_date": "2015-10-31",  
  "hometown": "Beijing",  
  "school": "Nanjing University",  
  "married": false,  
  "about": "I love Beijing Opera"  
}'
```

```
/* 返回结果：更新后的Document元数据，版本号加1 */  
{  
  "_index": "employee", // index名  
  "_type": "dev",       // type名  
  "_id": "1",           // id  
  "_version": 2,        // 版本号  
  "_shards": {          // Shard信息  
    "total": 2,  
    "successful": 2,  
    "failed": 0  
  },  
  "created": true        // 创建成功  
}
```

## ➤ 示例

### • 第6步：删除Document

```
/* 删除/employee/dev/1下的Document */  
curl -XDELETE 'localhost:9200/employee/dev/1?pretty'
```

### • 第7步：删除Index

```
/* 删除Index employee */  
curl -XDELETE 'localhost:9200/employee/?pretty'  
/* 返回结果：true表示Index删除成功 */  
{  
  "acknowledged": true  
}
```

➤ 数据类型

Esdrive SQL	ElasticSearch
String	String
Int	Integer
Boolean	Boolean
Tinyint	Byte
Smallint	Short
Bigint	Long
Float	Float
Double	Double
Date	Date
Timestamp	Long

## ➤ 创建Esdrive内表

- 在Search中创建Index，同时在Inceptor中创建对应的映射表（Esdrive内表）

```
CREATE TABLE <tableName> (<id> STRING, <col_name1> <data_type1>, <col_name2> <data_type2>, ...) ①  
  STORED AS ES  
  [WITH SHARD NUMBER <m>] ②  
  [REPLICATION <n>] ③  
  [TBLPROPERTIES('elasticsearch.tablename'='<dbName.tableName1>')]; ④
```

- ① 第一列<id>映射为Index的\_id，必须是String类型
- ② 可选项，指定Index的分片（Shard）数，默认值为10，建议每个Shard不超过25G，建表后不可改
- ③ 可选项，指定Shard的副本数，默认值为1，建表后可改
- ④ 可选项，<dbName.tableName1>指定新建表在Search中映射的Index名，Index必须是不存在的



### ➤ 创建Esdrive内表

- 示例：创建了一张名为esdrive\_inner\_table的Esdrive内表，字段包含所有的数据类型，Shard数为10，副本数为1，均采用默认值

```
create table esdrive_inner_table (key1 string, sv0 int, sv1 boolean, sv2 tinyint, sv3 smallint, sv4 bigint, sv5 float, sv6 double, sv7 string, sv8 date, sv9 timestamp) stored as ES;
```

The screenshot displays the Elasticsearch web interface. At the top, the URL is `http://shiva01:9200/` and the cluster health is `green (1799 of 1799)`. The main navigation bar includes tabs for '概览', '索引', '数据浏览', '基本查询 [+]', and '复合查询 [+]'. The '索引' tab is active, showing a list of indices. The index `default.esdrive_inner_table` is highlighted, with a red box around its name and a red arrow pointing to the text '==>Index名'. Below the index name, the size is `1.24ki (2.48ki)` and the number of documents is `docs: 0 (0)`. A red arrow points from the text '查看索引信息' to the '索引信息' button in the '动作' dropdown menu. The '动作' dropdown menu is open, showing options for '信息' and '动作'. The '信息' dropdown menu is also open, showing options for '索引状态' and '索引信息'. The '索引信息' option is selected, and the settings for the index `default.esdrive_inner_table` are displayed in a modal window. The settings include `"number_of_shards": "10"` (labeled '==>shard数') and `"number_of_replicas": "1"` (labeled '==>副本数'). The modal window also shows the index state as `"open"` and the creation date as `"1486969386884"`. The 'SHARP' logo is visible in the bottom right corner.

### ➤ 创建Esdrive外表

- 在Inceptor中创建映射表（Esdrive外表），与已存在的Index建立映射关系
- 创建Esdrive外表时，不可指定Shard和副本数

```
CREATE EXTERNAL TABLE <tableName> (  
  <id> STRING, <col_name1> <data_type1>, <col_name2> <data_type2>, ...  
)  
STORED BY 'io.transwarp.esdrive.ElasticSearchStorageHandler' ①  
[WITH SERDEPROPERTIES('elasticsearch.columns.mapping'='_id,<cl1>,<cl2>, ...')] ②  
TBLPROPERTIES('elasticsearch.tablename'='<dbName.tableName1>','elasticsearch.indextype'='default') ③  
);
```

① 指定Storage Handler，STORED AS ES的简写

② 可选项，指定外表的列和Index字段的映射关系

③ 必选项，建外表时必须指定Search中一个已存在的Index<dbName.tableName1>，支持指定Index Type

➤ 创建Esdrive外表

• 示例

```
CREATE EXTERNAL TABLE esdrive_external_table (  
  key1 string, ex0 int, ex1 bigint, ex2 double, ex3 string  
)  
STORED BY 'io.transwarp.esdrive.ElasticSearchStorageHandler'  
WITH SERDEPROPERTIES ('elasticsearch.columns.mapping'='_id,sv0,sv2,sv5,sv7')  
TBLPROPERTIES ('elasticsearch.tablename'='default.esdrive_inner_table');
```

Esdrive外表中的列	Key1(string)	ex0(int)	ex1(bigint)	ex2(double)	ex3(string)
ES索引的字段	_id(string)	sv0(integer)	sv2(byte)	sv5(float)	sv7(string)

### ➤ 分词器 (Analyzer)

- 创建Esdrive内表时，直接用SQL语句对列指定分词器
  - 只用于创建内表
  - 只用于内表中String类型的列，且不可以是内表的首列
- 不同语言类型，选择的分词器不同

/\* 对中文列指定分词器 \*/

<col\_name> STRING <WITH|APPEND> ANALYZER 'ZH' <ANALYZER\_NAME>

① 'ZH': 声明语言类型为中文，必须有单引号

② <WITH|APPEND>: 指定分词列的关键字。当该列只需要检索语义时，用WITH关键字；当该列在检索语义的基础上还需要精确查询时，用APPEND关键字

③ <ANALYZER\_NAME>: 可选的中文分词器有两个，ik和mmseg

/\* 对英文列指定分词器 \*/

<col\_name> STRING <WITH|APPEND> ANALYZER 'EN' <ANALYZER\_NAME>

① 'EN': 声明语言类型为英文，必须有单引号

② <ANALYZER\_NAME>: 可选的英文分词器有两个，standard和english

### ➤ 为Esdrive表增加列

- 只用于Esdrive内表，不能用于外表

```
ALTER TABLE <tableName> ADD COLUMNS (  
    <col_name1> <data_type1>, <col_name2> <data_type2>, ...  
);
```

### ➤ 清空Esdrive表

- 只能清空Esdrive内表，不能用于外表

```
TRUNCATE TABLE <tableName>;
```

### ➤ 删除Esdrive表

- 对于内表，删除ES中的Index（含数据），以及Inceptor中的Esdrive表（Metastore中的元数据）
- 对于外表，只删除Inceptor中的Esdrive表（Metastore中的元数据），不删除ES中的Index

```
DROP TABLE <tableName>;
```

## ➤ 插入数据

### • 单条插入

```
INSERT INTO esdrive_inner_table (key1,sv0,sv1,sv2,sv3,sv4,sv5,sv6,sv7,sv8,sv9)
VALUES ('k1',2,true,3,4,5,6.0,7.33,'s7ad','2010-06-30 08:54:42',1477881007946);
```

### • 批量插入

```
BATCHINSERT INTO esdrive_inner_table (key1,sv0,sv1,sv2,sv3,sv4,sv5,sv6,sv7,sv8,sv9)
BATCHVALUES ( VALUES ('k2',3,true,3,4,5,6.1,7.33,'dsfx','2010-06-30 10:07:48',1377832007946),
VALUES ('k3',4,true,3,4,5,6.2,7.22,'ngfh','2010-06-30 06:08:17',1277833007946),
VALUES ('k4',5,true,3,4,5,6.3,7.11,'ewwd','2010-06-30 18:52:37',1477844007946)
);
```

### • 插入时首列需唯一

- 首列key1映射为Index的\_id，首列相同的数据只会保留最后插入的数据

```
INSERT INTO TABLE esdrive_start(key1, content, tint, tbool) VALUES ('1', 'hello ES', 1 , true);
INSERT INTO TABLE esdrive_start(key1, content, tint, tbool) VALUES ('2', 'hello search', 2, false);
```

### ➤ 更新数据

```
UPDATE <tableName> SET <col_name> = <value> WHERE <filter_conditions>;
```

*eg:* update esdrive\_inner\_table set sv1=false where key1='k1';

### ➤ 删除数据

```
DELETE FROM <tableName> WHERE <filter_conditions>;
```

*eg:* DELETE FROM esdrive\_inner\_table WHERE key1='k1';



➤ 对于Select语句，Esdrive SQL与Inceptor SQL的用法完全相同

- 包括Where、Group By、Join、集合运算等

➤ Esdrive SQL检索语义的实现

- 第1步：对被查询文本进行分词，生成倒排索引
- 第2步：对查询条件进行分词，利用倒排索引实现全文检索

➤ Esdrive SQL检索语义的优势

- 语义更丰富
  - 标准SQL: like %word%模糊查询只有一种语义，无法描述复杂语义，如：查询指定间隔内的两个词
  - Esdrive SQL: 定义了多种模糊查询的语法，用于描述不同场景下的检索语义
- 查询性能更优
  - 标准SQL: like %word%模糊查询的算法复杂度为 $O(n)$
  - Esdrive SQL: 基于倒排索引的全文检索的算法复杂度为 $O(\log(n))$

## ➤ Contains函数

- 通过Contains函数对查询条件进行分词，实现分词检索

```
CONTAINS(  
  <col_name>, '<text_query>'  
)
```

- Near操作符

- 限制所查单词的间隔，提高查询结果的相关性，包含三个参数

- ① token1~n: 表示检索词（ $n \geq 2$ ），token应该是倒排索引中存在的单词，否则查询无意义
- ② slop: 表示token1与token2之间允许间隔的最大token数，该值是一个上限
- ③ in\_order: 可选项，表示是否按顺序依次匹配token，Boolean类型。默认值为false，不按顺序匹配，token1可以出现在token2之后；若为true，则按顺序匹配，token1必须出现在token2之前

```
CONTAINS(  
  <col_name>, 'NEAR((token1, token2[, token3, ...]), slop[, in_order])'  
);
```

## ➤ Contains函数

### • Near操作符

```
Select * from news_analyze_zh where contains(content, 'near((京东,阿里), 1, false)');
```

key1	content
7	阿里、京东、小米争相抢滩，“新零售”到底是什么

### • Fuzzy操作符

– 在Contains分词检索的基础上，查询语义相似的短语，包含2个参数

① **phrase**: 表示需要查询的短语。先经过分词得到多个token，查询结果必须包含分词后的所有token

② **fuzziness**: 表示最大编辑距离（Levenshtein距离），用于表达短语之间的语义相似度，中文较复杂

```
CONTAINS(  
  <col_name>, 'FUZZY(phrase, fuzziness)'  
);
```

➤ Contains函数

- Fuzzy操作符

```
Select * from news_analyze_zh where contains(content, 'fuzzy(阿里金融, 5)');
```

key1	content
2	重庆市携手阿里金融，共建新型智慧城市
4	马云剖析阿里未来三个业务：金融、平台和数据

- Contains函数语法举例

```
select dt from inceptor where contains(hphm, "term '鲁D528E8'" ) order by dt limit 100;  
select dt from inceptor where contains(hphm, "prefix '鲁D528'" ) order by dt limit 100;  
select dt from inceptor where contains(hphm, "wildcard '鲁D*28E8'" ) order by dt limit 100;  
select dt from inceptor where contains(dt, "term '9223370633649793807'" ) and  
contains(hphm, "term '鲁P56729'" ) order by dt limit 100;  
select dt from inceptor where contains(dt, "range'[9223370647735474807,9223370647735478807]'" );
```



# Q&A

**TRANSWARP**  
星环科技