ALIBABA CLOUD

阿里云

云数据库HBase版
HBase Search全文索引

文档版本：20220705

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.使用前必读

> **注意** HBase增强版已全新升级为云原生多模数据库Lindorm，Lindorm全文索引功能也大幅增强，新场景请参见Lindorm SearchIndex。

2.快速入门

2.1. 服务介绍

Search服务用来解决复杂的多维查询和全文检索。

Solr是构建在Apache Lucene上的企业级搜索平台，是分布式全文检索的最佳实践之一，支持各种复杂的条件查询和全文检索，具有广泛的用户基础。通过深度融合HBase与Solr，我们推出了既能满足大数据海量存储，又可以支持复杂多维查询和全文检索的Search服务。

Search服务适用于：**需要保存海量数据，并且需要各种条件组合查询的业务。**例如：

- 物流场景，需要存储大量轨迹物流信息，并需根据任意多个字段组合查询。
- 交通监控场景，保存大量过车记录，同时会根据车辆信息任意条件组合检索出感兴趣的记录。
- 网站会员、商品信息检索场景，一般保存大量的商品/会员信息，并需要根据少量条件进行复杂且任意的查询，以满足网站用户任意搜索需求等。

Search服务的整体数据流如上图，数据写入HBase后，BDS负责将数据实时同步到Search索引中。在此架构下，HBase服务、数据同步通道BDS和Search索引都是以独立集群的方式存在，您可以分别对各个集群进行管理：如果Search索引处理能力不足，只需要扩容Search集群；如果BDS同步能力不足，可以单独扩容BDS。HBase/BDS/Search可以针对不同的使用场景选择不同的机型，独立的部署形态大幅提升了系统的稳定性。

与二级索引的区别

HBase增强版提供二级索引，可以低成本的解决非主键查询问题，适用于查询列比较固定的场景。如果业务场景需要复杂的多维组合查询，需要考虑使用Search服务。

与开源Solr的区别

Search服务深度融合HBase和Solr，用户无需关注各个服务的运行，只需要通过简单的API/Shell操作就可以将HBase与Solr建立关联。

Search服务基于开源Solr深度定制，完全兼容开源Solr API，在系统稳定性、读写性能、监控告警上做了大量工作，提供更加可靠、高性能的企业级搜索平台。

服务开通

开通Search服务需要三步：

1. 创建HBase集群，服务类型选择 `增强版` ；
2. 创建BDS集群；
3. HBase集群创建成功后，在HBase控制台页面点击 `全文索引` ，完成Search实例的购买和关联。

具体参见开通指南

使用指南

参见快速开始和索引管理

Search最佳实践

参见Search最佳实践

2.2. 开通指南

开通Search服务前需要先购买HBase增强版集群和同一个VPC内的BDS集群，如果需要在已经购买的HBase增强版和BDS上开通Search服务，则可以跳过此步。

购买HBase增强版和BDS可以在HBase产品首页单击购买按钮选择相应的集群和配置，或者在HBase控制台单击**创建HBase集群**和**创建BDS集群**。

开通全文索引（Search）服务

1. 在购买的HBase增强版集群的控制台页面左边单击**全文索引**开通服务。
2. 在弹出的开通页面中，关联一个空闲的BDS集群。

   > **说明** 只有在同一个VPC下，且没有被其他HBase集群关联到全文索引链路的BDS，才能被选择到。

3. 选择机器规格Search服务对内存和磁盘要求比较高，尽量选择内存大的机型，以及SSD云盘和本地SSD盘。如果对机器选型有疑问，可以在钉钉咨询 `云HBase答疑` 或者提工单咨询。

> **说明** 在开通过程中，HBase集群和BDS集群都会自动升级到最新版，升级过程中会有轻微抖动，请在业务低峰期开通。所选择的BDS集群不能再关联其他集群和做为其他用途使用，请谨慎选择。

查看开通进度

购买完成后，在控制台可以看到3个实例。如在下图中，分别是HBase增强版实例ld-t4n33q6tj8hexxxx，BDS实例bds-t4n579vl2f74xxxx（可以在 `服务 / 版本 / 主实例` 一栏中看到关联了ld-t4n33q8he022kxxxx），和Search实例ld-t4n33q8he022kxxxx-m1-se（可以在 `服务 / 版本 / 主实例` 一栏中看到关联了ld-t4n33q8he022kxxxx）。当所有实例都从创建中变成运行中，即可以开始使用。

2.3. 快速开始

从HBase增强版Shell访问页面下载最新版本的HBase Shell，并根据Shell使用指导完成Shell访问HBase的相关配置，同时需要确认配置好白名单。

HBase实例中创建表

```
hbase(main):002:0> create 'testTable', {NAME => 'f'}
```

上面的命令在HBase中创建了一张名为 `testTable` ，列族ColumnFamily为 `f` 的表。

Search实例中创建索引

在Search实例的控制台，单击 `数据库连接` 可以看到 `WebUI访问` 。注意：使用前需要设置好访问白名单和访问密码。

`config set` 可以选择 `_indexer_default` 作为配置集， `numShards` 设置为节点个数，其它采用默认值即可。

建立映射关系

假设我们需要将 `testTable` 表中 `f:name` 这一列（列族为f，列名为name ）映射到索引表 `democollection` 的 `name_s` 这一列。于是我们将下述JSON写入到一个文件中，如 `schema.json`

```
{
  "sourceNamespace": "default",
  "sourceTable": "testTable",
  "targetIndexName": "democollection",
  "indexType": "SOLR",
  "rowkeyFormatterType": "STRING",
  "fields": [
    {
      "source": "f:name",
      "targetField": "name_s",
      "type": "STRING"
    }
  ]
}
```

备注：JSON文件中各个参数的含义参见HBase索引管理。

在HBase shell中执行：

```
hbase(main):006:0> alter_external_index 'testTable', 'schema.json'
```

等待命令结束后，索引映射关系就已经创建完毕。

HBase中写入数据

写入数据参见使用Java API访问增强版集群，这里使用Shell写入一条示例数据

```
hbase(main):008:0> put 'testTable', 'row1', 'f:name', 'foo'
Took 0.1697 seconds
```

Search实例中查询数据

在Search的WebUI中，选中刚才创建的索引 `democollection` ，单击 `Query` 查询，就可以查到刚才写入的一行数据。

查询HBase

在Search的使用场景中，一般是将HBase表一行数据的部分列同步到Search实例，即只需要将那些需要多维查询的列配置到映射JSON文件中。通过Search查询的结果中，每行数据都会有一个列 `id` ，这个 `id` 即为这一行数据在HBase表中的rowkey。拿到这个rowkey后，回查HBase，就可以获得完整数据。

2.4. 注意事项

当HBase表配置Search索引后，带时间戳的写入将会被禁止掉（非索引列不会禁止带时间戳写入），会抛出 `User defined timestamp is not allowed when external index is enabled...` 的Exception。

- 默认不支持自定义时间戳写入。
- 不支持多版本。
- 有限支持表TTL。

1. 自定义时间戳

1. 使用的场景一定要带入时间戳；
2. 写入HBase的数据是通过BDS同步过来（如主备同步，RDS的增量导入等），BDS有可能会加上写入时间戳。

可以通过使用HBaseue Shell访问增强版集群修改表的Mutability属性为MUTABLE_ALL来打开对自定义时间戳的支持。打开时间戳支持会有一些性能损耗，但通常不会非常明显。

```
# 打开时间戳支持
hbase(main):002:0> alter 'testTable', MUTABILITY=> 'MUTABLE_ALL'
# 关闭时间戳支持
hbase(main):002:0> alter 'testTable', MUTABILITY=> 'MUTABLE_LATEST'
```

2. 多版本

当HBase表配置Search索引后，如果数据表有多版本，会导致HBase与Search实例的数据不一致。需要通过使用HBaseue Shell访问增强版集群显示的将HBase表版本修改为1。另外，也不再支持删除指定的某个版本，这种删除行为将会被禁止。举例来说

```
    //如果构造的Delete对象不加任何Family和Qualifier，则代表删除整行 --支持
    Delete delete = new Delete(Bytes.toBytes("row"));
    //删除f:q1这一列 --支持
    delete.addColumns(Bytes.toBytes("f"), Bytes.toBytes("q1"));
    //删除时间戳在ts1和ts1之前的所有数据  -- 开自定义时间戳（MUTABLE_ALL）后支持
    delete.addColumns(Bytes.toBytes("f"), Bytes.toBytes("q1"),  ts1);
    //删除f这个family里所有的列 --支持
    delete.addFamily(Bytes.toBytes("f"));
    //删除f这个family里时间戳在ts1和ts1之前的所有列数据 -- 开自定义时间戳（MUTABLE_ALL）后支持
    delete.addFamily(Bytes.toBytes("f"),  ts1);
    //删除f:q1这一列的最新版本  --不支持
    delete.addColumn(Bytes.toBytes("f"), Bytes.toBytes("q1"));
    //删除f:q1这一列中时间戳（版本）为ts1的数据  --不支持
    delete.addColumn(Bytes.toBytes("f"), Bytes.toBytes("q1"), ts1);
```

注意： 为了防止数据不一致，在HBase中执行删除某一行后，Search中对应的Document不会删除，而是会删除这个Document中除了 `id` 这个field（Rowkey映射）以外的其他所有field。一般情况下，用户都是带一定条件去查询Search，不会命中这种只有 `id` 的行。但如果查询到只有 `id` 的行，代表此行已经删除，用户需要自行过滤。我们后续将考虑使用Search的TTL功能在一定时间后自动删除这些行。

3. TTL

HBase和Search都支持数据TTL过期，但是HBase的TTL机制与Search的TTL机制不一样，HBase是单个KV过期，而Search中只能按照Document（对应HBase的一行）过期，而且过期的时间不会完全一致。如果业务中需要支持TTL，可以单独联系 `云HBase答疑` 或提工单。

3.索引管理

3.1. 管理HBase全文索引

准备工作

学习快速开始部分，在使用HBaseue Shell访问增强版集群下载好并配置好最新版本的Shell。

HBase表与Search索引的映射

表和索引的映射采用JSON方式实现，一个典型的映射配置如下：

```
{
  "sourceNamespace": "default",
  "sourceTable": "testTable",
  "targetIndexName": "democollection",
  "indexType": "SOLR",
  "rowkeyFormatterType": "STRING",
  "fields": [
    {
      "source": "f:name",
      "targetField": "name_s",
      "type": "STRING"
    },
     {
      "source": "f:age",
      "targetField": "age_i",
      "type": "INT"
    }
  ]
}
```

上述示例的含义：将HBase表 `testTable` 的数据同步到Search索引 `democollection` 中。其中 `f:name` 这一列（列族名和列名用冒号隔开）映射到索引中的 `name_s` 这一列， `f:age` 这一列映射到索引中的 `age_i` 这一列。下面将解释每个配置项的具体含义和可以配置的参数值。

参数名	含义
sourceNamespace	HBase表的namespace名，如果表没有namespace，这个参数可以不配，或者配置为‘default’
sourceTable	HBase表的表名，不含namespace的部分
targetIndexName	Search索引名
indexType	此项固定为 `SOLR`
rowkeyFormatterType	hbase中rowkey的格式，此处可以填 `STRING` 或者 `HEX` ，具体含义见下方解释

参数名	含义
fields	具体映射的列以及类型，fields配置项是一个json array，多个列的配置用逗号隔开，具体参见示例。具体配置详见下方解释。

rowkeyFormatterType

rowkeyFormatterType代表HBase表rowkey映射到索引Document中 `id` （数据类型为string）的方式。目前支持两种方式：

- `STRING` ：如果用户HBase表的rowkey是String，如 `row1` `order0001` ，`12345` （注意12345为字符串，非数字）可以使用此配置。该方式使用Bytes.toString(byte[])，函数将rowkey转成索引Document中的 `id` 。用户在Search索引中查出对应Document后，可以使用Bytes.toBytes(String)，函数将 `id` 转成byte[]做为rowkey反查HBase。
- `HEX` ：如果用户HBase表的rowkey不是String，则使用此方式，比如用户的rowkey是数字12345，或者具有多个含义的字段（有些字段是非String）拼接而成。该方法使用org.apache.commons.codec.binary.Hex包中的encodeAsString(byte[])）函数将rowkey转成索引Document中的 `id` 。用户在Search索引中查出对应Document后，可以使用Hex.decodeHex（String.toCharArray()）函数将idString转成byte[]做为rowkey反查HBase。

> **说明** 如果HBase表的Rowkey并非由String组成（即不是用 `Bytes.toBytes(String)` 方法当做rowkey存入HBase），请使用HEX方式，否则在将索引Document中的ID反转成bytes后有可能会和原rowkey不一样从而反查失败。

fields

每一个field映射都由以下三个参数组成：

参数名	含义
source	HBase表中需要映射的列名，其中family和qualifier的名字用冒号隔开，如 `f:name`
targetField	索引表中的列名，上述示例中给出的列名都是 **动态列** ，如 `name_s` 、 `age_i` ，这样的用法不需要事先定义，直接使用即可，Search服务会自动识别。更多动态列用法参见schema配置
type	HBase中的列在写入时的数据类型，**是HBase中source这一列的类型**。可以配置的值为INT/LONG/STRING/BOOLEAN/FLOAT/DOUBLE/SHORT/BIGDECIMAL，大小写敏感。

理解数据类型type

在HBase中，是**没有数据类型**这个概念的，所有类型的数据，包括中文字符，都是用户自己调用 `Bytes.toBytes(String/long/int/...)` 方法，把对应的String/long/int等类型转化成bytes存储在HBase的列中。配置type字段，就是要告诉系统，你存入到HBase的这一列，是使用哪种方法存入HBase的该列中的。比如

```
int age = 25;
byte[] ageValue = Bytes.toBytes(age);
put.addColumn(Bytes.toBytes("f"), Bytes.toBytes("age"), ageValue);
String name = "25";
byte[] nameValue = Bytes.toBytes(name);
put.addColumn(Bytes.toBytes("f"), Bytes.toBytes("name"), nameValue);
```

上述代码中 `f:age` 这一列的type为 `INT` 的值,而“f:name”这一列的type为 `STRING` 的列,**而非是一个INT**。填对type类型对正确地将数据同步到Search索引至关重要。因为系统会根据用户填入的type类型来从bytes数组中反解出原始数值来同步到Search索引里。在上述的列子中，如果用户错误地把 `f:name` 这一列的type填写为'INT'，系统会调用Bytes.toInt()方法反解原始值，很显然反解出来的值一定是错误的。

理解targetField

targetField表示HBase中source这一列将会在Search索引中映射的列。Search服务是一个强Schema的系统，每一列都必须在配置集的managed_schema中预先定义（schema配置参见schema配置 ）。但是这里我们推荐使用Search的动态列（dynamicField）功能，通过后缀自动识别这一列的类型。比如 `name_s` 代表这一列在Search索引中的类型为String。

HBase中source的类型type并不需要和索引中的列数据类型一一对应。比如用户可以定义Source列 `f:age` 的type为STRING，而索引中的targetField为 `age_i` （代表索引中这一列的类型为INT），在写入索引时，Search服务会自动将STRING转化成INT。但是如果用户往 `f:age` 列中写入了一个无法转换成数字的STRING，那么写入索引时，一定会报错。

管理Schema

修改映射Schema

用户可以将上一节介绍的JSON格式的schema存储在一个文件中，如schema.json，然后在HBase Shell中直接调用 `alter_external_index` 命令完成对HBase映射Schema的修改。 `schema.json` 文件需要放在启动HBase Shell的目录，或者使用绝对或者相对路径指向该文件。

```
hbase(main):006:0> alter_external_index 'HBase表名', 'schema.json'
```

使用JSON管理可以快速地添加、删除、修改多列。同时，也可以将fields中的映射列全部删掉，从而达到删除HBase表所有映射的目的。比如：

```
{
  "sourceNamespace": "default",
  "sourceTable": "testTable",
  "targetIndexName": "democollection",
  "indexType": "SOLR",
  "rowkeyFormatterType": "STRING",
  "fields": []
}
```

如果用户只想在原有的映射Schema上添加一列或者几列，可以采用 `add_external_index_field` 命令去添加一列或者多列。

```
 hbase shell> add_external_index_field 'testTable', {FAMILY => 'f', QUALIFIER => 'money', T
ARGETFIELD => 'money_f', TYPE => 'FLOAT' }
```

注意：只有使用了alter_external_index添加过映射schema的表，才能使用add_external_index_field的方式单独添加列。每次修改映射Schema，HBase的表都需要经历一次完整的alter Table流程，如果需要修改的列比较多，推荐采用alter_external_index的方式一次完成

如果用户只想在原有的映射schema上删除一列或者几列，可以采用 `remove_external_index` 完成。

```
hbase shell> remove_external_index 'testTable', 'f:name', f:age'
```

注意：每次修改映射Schema，HBase的表都需要经历一次完整的alter Table流程，如果需要修改的列比较多，推荐采用alter_external_index的方式一次完成。

查看目前的映射schema

在HBase Shell中使用 `describe_external_index` 命令，就可以得到当前表的映射Schema的完整JSON描述。

```
hbase(main):005:0> describe_external_index 'testTable'
```

3.2. 查看索引同步状态

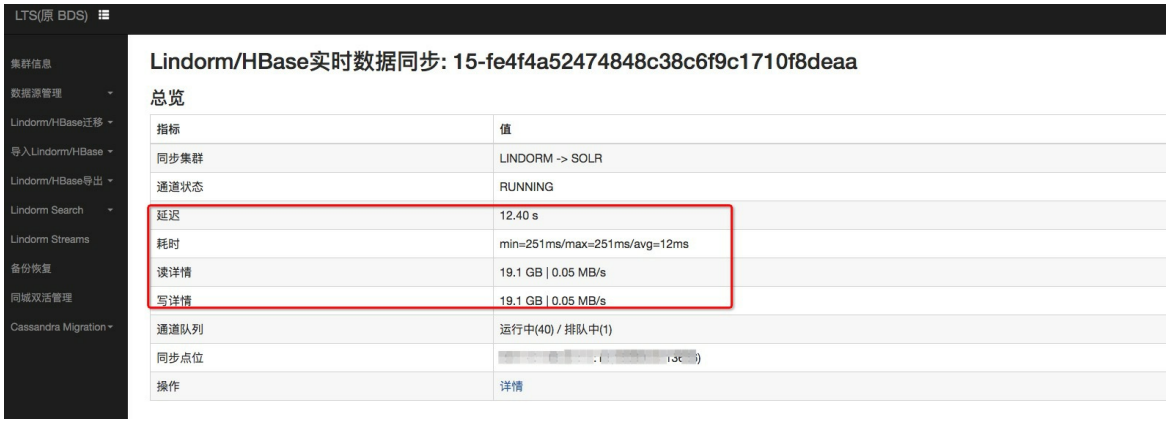
查看状态

HBase和Search索引的数据同步由关联的BDS完成，在BDS中可以看到索引同步状态。

1. 在与HBase关联的BDS中，找到WebUI入口。

2. 在BDS Web页面中单击 `Lindorm Search` 选择 **实时同步**

3. 选中通道名，点击进入后就可以看到同步详情。

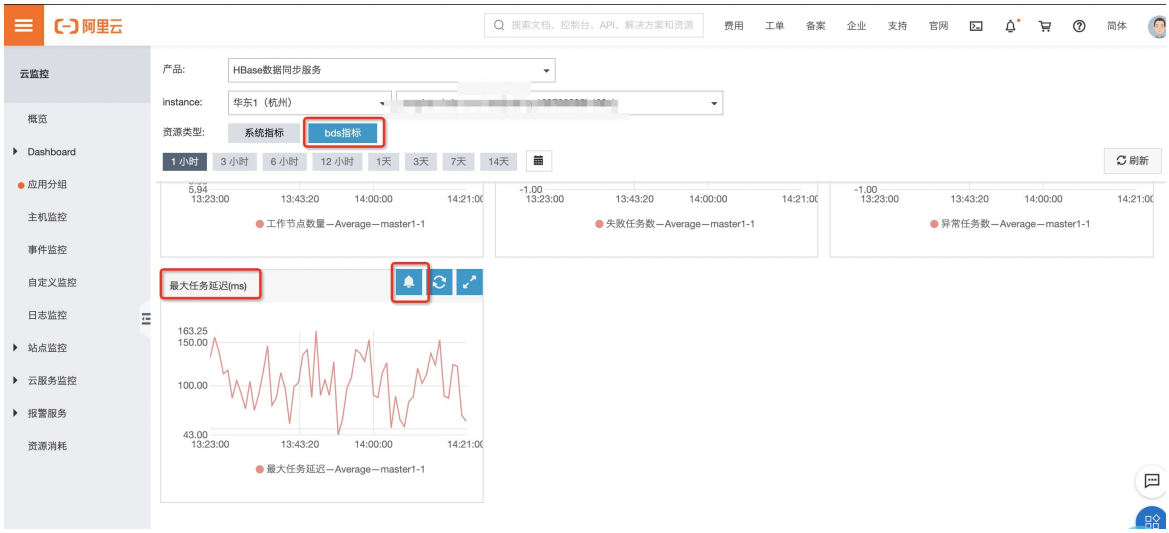


监控和报警

数据同步的延迟指标可以通过查看BDS的监控获取，并可以在云监控上订阅相关的报警。

1. 从BDS的控制台页面点击云监控进入云监控。

2. 选择BDS指标，查看最大任务延迟（ms）即可获得当前的延迟状态，单击铃铛按钮配置相关报警。

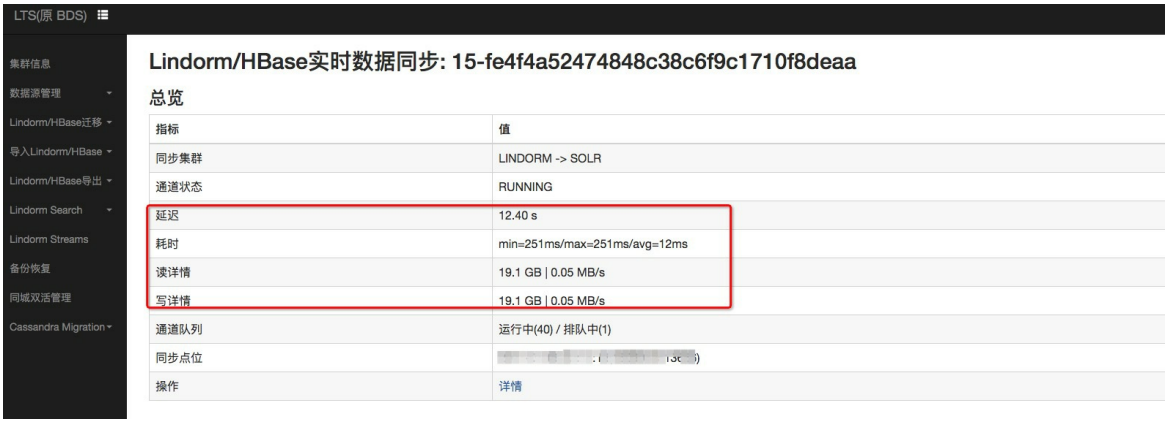


常见问题处理

如果BDS界面中显示同步的延迟过大，超出业务承受的范围，用户可以通过自查解决问题。

同步点位不再推进

如果发现BDS界面中的同步点位不再推进，通常是在HBase中写入了不合法的数据导致同步失败。



此时用户可以点击右侧的详情，在详情页面中查找错误信息。如果遇到无法解决的问题，可以在钉钉上联系 `云HBase答疑` 或者提工单咨询。

同步速度慢

如果观察到同步点位仍然在推进，但是界面中的 `写详情` 和 `读详情` 速度都很慢，可能是Search服务的集群写入能力达到瓶颈，需要扩容或者升配，具体可以参考Search实例云监控中显示的服务器负载情况。如果有相关的问题，可以在钉钉上联系 `云HBase答疑` 或者提工单咨询。

通道队列中的Log积压严重

如果观察到界面中的 `写详情` 和 `读详情` 速度都不慢，但是在 `通道队列` 中显示队列中的Log积压越来越多，则有可能是HBase写入过快，BDS同步能力已经跟不上，需要扩容BDS集群。也可能是Search服务的集群写入能力达到瓶颈，需要扩容或者升配，具体可以查看BDS和Search实例的云监控，如果有相关的问题，可以在钉钉上联系 `云HBase答疑` 或者提工单咨询。

3.3. 全量数据索引构建

当配置好HBase表与Search索引的映射后，实时写入HBase中的数据将自动同步到Search索引中。对于HBase表中的存量历史数据，需要手工执行一次全量构建才可以完成数据同步。

全量数据构建

在HBase Shell中执行 `build_external_index` 为HBase表中的历史数据构建索引，该命令是异步执行的。

> **说明** 全量构建索引过程中，会阻塞HBase表的DDL操作，直到构建完成才能继续执行，但不会影响表的读写。

```
hbase shell> build_external_index 'testTable'
```

查看全量构建进度

历史数据的全量构建由关联的BDS完成，可以在关联的BDS Web页面查看到全量构建的进度。

1. 在与HBase关联的BDS中，找到WebUI入口。

2. 在BDS Web页面中单击 `Lindorm Search` 选择 **全量同步**

3. 单击相应的任务名，就可以查看当前全量任务的状态，如果状态显示为SUCCESS，则说明构建完成。

取消构建任务

如果想停止执行build 索引任务，则可以通过下述方式停止：

```
hbase> cancel_build_external_index 'testTable'
```

或者可以在查看全量构建进度的BDS Web页面中直接点击停止后删除任务。

4.最佳实践

4.1. Java API访问

云数据库HBase增强版全文索引服务支持多语言访问，并且完全兼容开源Apache Solr API，本文介绍如何使用Solr Java API访问云数据库HBase增强版全文索引服务。

使用Solr Java API访问云数据库HBase增强版全文索引服务的操作和使用Solr Java API访问云原生多模数据库Lindorm搜索引擎的操作相同，具体请参见使用示例。

4.2. Shell访问

下载Shell压缩包

下载并解压，下载地址

```
tar -zxvf alisolr-7.3.8-bin.tar.gz
```

配置

修改 alisolr-7.3.8-bin/conf/solr.in.sh 文件，去掉 SOLR_ZK_HOST 前面的注释 #，并修改如下：

```
SOLR_ZK_HOST="ld-xxxx-proxy-zk.hbaseue.9b78df04-b.rds.aliyuncs.com:2181/solr"
```

上面配置的地址可以在Search实例控制台查看，单击 **数据库连接** ，查看 **客户端访问地址** ，如下：

注：上述配置示例中的 `SOLR_ZK_HOST` 是私网地址，如果是想通过外网访问，请先 **开通外网地址** ，配置成外网地址即可。

使用Shell访问

进入到 alisolr-7.3.8-bin/bin目录。

```
./solr
```

- 创建索引表

  ```
  ./solr create_collection -c testIndex -n _indexer_default -shards 2
  ```

  索引名为 `testIndex` ，使用默认配置集 `_indexer_default` ，分片数设置为 `2` 。
- 查看索引表

  ```
  ./solr list_collections
  ```
- 下载配置集

  ```
  ./solr zk downconfig -d . -n _indexer_default
  ```

  其中 `_indexer_default` 是Search服务提供的默认配置集，执行上述命令后，当前目录会自动创建一个名为 `conf` 的子文件夹，里面存储的就是 `_indexer_default` 的配置集合。
- 上传配置集

  ```
  ./solr zk upconfig -d conf -n myConf
  ```

  基于默认的配置集 `_indexer_default` ，我们可以修改后上传为自定义的配置集名 `myConf` 。
- 查看配置集

  ```
  ./solr zk ls /configs
  ```
- 创建基于自定义配置集的索引表

  ```
  ./solr create_collection -c myIndex -n myConf -shards 2
  ```

4.3. 更新配置集

本文介绍HBase Search全文索引更新配置集的方法。

前提条件

下载并安装Shell，具体请参见Shell访问指导。

注意事项

- 建议使用dynamicField功能，不单独定义每个索引列，避免频繁修改 `managed-schema` 文件。
- 每个索引需要有自己的配置集，不建议多个索引共享配置集。
- 如果需要自己定义配置，请下载 `_indexer_default` 配置集后，在此基础上进行修改。

下载默认的配置集模板

执行以下命令下载默认的配置集模板 `_indexer_default` ，在其基础上进行编辑，添加业务自定义的配置。

```
cd alisolr-7.3.8-bin/bin
./solr zk ls /configs          // 查询已有的配置集列表
./solr zk downconfig -d . -n _indexer_default      // 下载配置集_indexer_default到当前目录
```

执行上述命令成功后，在当前目录下会生成一个*conf*的目录，其中有两个重要的文件：`managed-schema` 和 `solrconfig.xml` 。

创建新的配置集

下面给出一个简单的示例：

1. 打开 `managed-schema` 文件。
2. 增加两个新的索引列定义，增加内容如下：

   ```
   <field name="name" type="string" indexed="true" stored="true" required="false" multiVal
   ued="false" />
   <field name="age" type="pint" indexed="true" stored="true" docValues="true"  multiValue
   d="false" />
   ```

   > **说明**
   > - `name` 是string类型， `age` 是基本int类型（pint代表int，plong代表long），两个列都需要建立索引 `indexed=true` ，并且都需要存储原始数据 `stored=true` 。
   > - 每增加一个新列，都需要在文件中定义好，当需要增加非常多的列时，定义起来会比较复杂。此时，可以使用Search服务提供的动态列能力，参考 `managed-schema` 中的 `dynamicField` 定义，有了它之后不需要额外定义每个列，只需要在写入数据时指定的列名称后缀与定义保持一致即可。
   > - 例如： `name_s` 可以自动匹配 `*_s` 。 `age_i` 可以自动匹配 `*_i` 。

3. 上传配置集。

   修改完后，可以上传自定义的配置集（建议每个collection对应一个配置集），执行如下命令：

   ```
   ./search-cli zk upconfig -d conf/ -n myconf
   ```

4. 在集群管理系统查看配置集是否上传成功，登录集群管理系统请参见登录集群管理系统。

   选择**Cloud > Tree > /configs**，查看配置集 `myconf` 是否上传成功。

4.4. 索引数据可见性

写入Search服务中的数据，只有等到 `commit` 后才是可见的，何时执行commit是可以配置的，当前有两种commit方式：soft commit、hard commit。

配置solrconfig.xml

创建索引时需要指定配置集 `config set` ，配置集中的 `solrconfig.xml` 文件可以用来控制索引数据的可见性。

- soft commit

```
<autoSoftCommit>
    <maxTime>${solr.autoSoftCommit.maxTime:15000}</maxTime>
</autoSoftCommit>
```

数据写入默认15秒后可以查询。

- hard commit

```
<autoCommit>
  <maxTime>${solr.autoCommit.maxTime:30000}</maxTime>
  <openSearcher>false</openSearcher>
</autoCommit>
```

数据写入默认30秒后刷新到磁盘中。

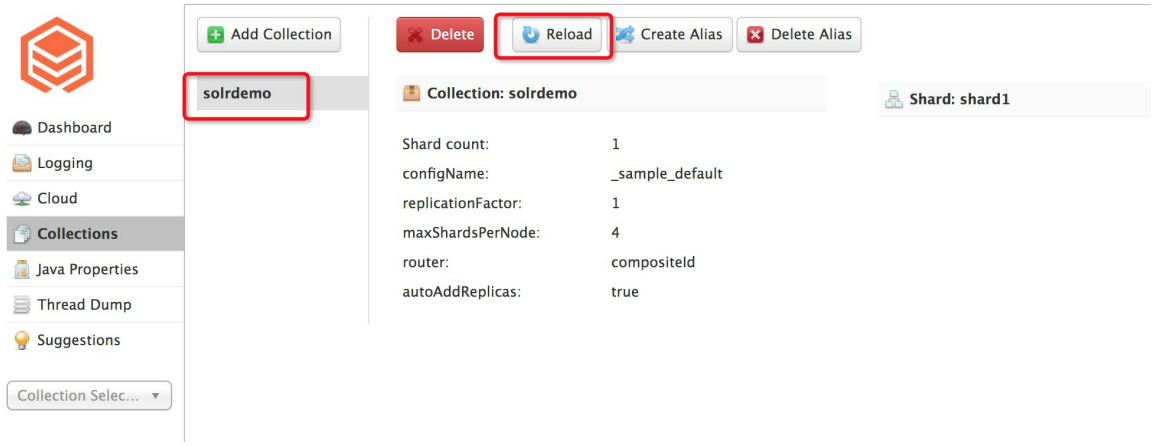
> 说明
>
> 1. soft commit的时间要小于hard commit时间。
> 2. 一般不建议配置较小的时间，这会导致服务端频繁刷新和open searcher，影响写入性能。采用默认值即可。

如何生效

1. 下载需要修改的配置集。
2. 修改 `soft commit` 和 `hard commit` 的时间。
3. 更新配置集。

```
./solr zk upconfig -d conf/ -n myconf
```

4. Reload Collection



客户端参数

如果不想修改服务端的solrconfig.xml文件，在单独写数据到Search时可以显示的调用commit来达到数据可见的目的。

- commitWithinMs

```
// 第二个参数commitWithinMs可以控制当前写入的数据多久后可以查询
public UpdateResponse add(SolrInputDocument doc, int commitWithinMs) throws SolrServerExc
eption, IOException;
```

例如：代表10秒后数据可查询。

```
cloudSolrClient.add(doc, 1000);
```

- commit API

```
// 写完数据后，显示调用commit接口，让服务端在多久后执行commit，保证数据可查询
public UpdateResponse commit(boolean waitFlush, boolean waitSearcher, boolean softCommit)
throws SolrServerException, IOException;
```

例如：服务端立即执行soft commit。

```
cloudSolrClient.commit(false, false, true);
```

参考文档

UpdateHandlers in SolrConfig

4.5. 分库分表（Alias功能）

背景信息

设想您有没有遇到过这样的问题：

1. 表变更业务逻辑中设置了访问某个表A，突然有一天需要修改为表B，此时只能修改配置进行线上变更。
2. 分库分表

   业务大部分场景只访问最近一周的数据，可以每隔一周新建一张表来存储，这样可以确保高效的查询热数据。在这个场景中需要自己来维护表的创建和删除，带来一定的业务复杂性。

   本文介绍的Alias（别名）将会完美的解决上面两个问题。

适用场景

时间序列场景

业务数据具有明显的时间特性，可以基于时间来创建不同的索引，这样既能降低单个索引的大小，又能提升查询性能。整个过程中业务不需要自己维护索引创建和删除。

重建索引场景

在不影响已有索引查询下，重建新的索引，待索引建完后，指向新的索引访问。整个过程中业务不需要代码变更。

如何使用Alias

基本功能：创建Alias指向已有的索引表

```
curl "http://solrhost:8983/solr/admin/collections?action=CREATEALIAS&name=your_alias_name&c
ollections=your_collection_name_A"
```

上面的URL功能：创建一个Alias名为your_allias_name，其指向一个索引表your_collection_name_A。这样业务逻辑中可以只设置访问your_alias_name，内核会自动转发请求到真实的索引表上。假设某一天需要变更索引表名为your_collection_name_B，执行一次更改Alias命令。

修改Alias

```
curl "http://solrhost:8983/solr/admin/collections?
action=ALIASPROP&name=your_alias_name&collections=your_collection_name_B"
```

这样，业务代码上不需要任何变更即可访问新的索引表。

高级功能：自动分表

搜索服务支持按照时间字段自动分表，大大简化业务逻辑。下面以具体的示例来介绍：业务要求以周为单位创建索引表，并且能够自动删除旧的索引表。

```
curl "http://solrhost:8983/solr/admin/collections?action=CREATEALIAS&name=test_router_alias
&router.start=NOW-30DAYS/DAY&router.autoDeleteAge=/DAY-90DAYS&router.field=your_timestamp_l
&router.name=time&router.interval=%2B7DAY&router.maxFutureMs=8640000000&create-collection.c
ollection.configName=_indexer_default&create-collection.numShards=2"
```

参数	值	说明
router.start	NOW-30DAYS/DAY	第一个collection创建的时间点，样例中给出的NOW-30DAYS/DAY代表以30天前开始新建索引
router.interval	+7DAY	间隔多久创建新的索引表，样例中给出的是每隔7天新建一个索引表
router.autoDeleteAge	/DAY-90DAYS	自动淘汰多久前的索引表，样例中给出的是淘汰90天前的索引表，其值必须大于router.start
router.field	your_timestamp_l	分表的时间字段，默认业务中需要携带这个字段，并指定时间值，例如当前的系统时间戳System.currentTimeMillis()
router.maxFutureMs	8640000000	代表最大容忍写入的时间字段your_date_dt与当前时间的差值，防止写入过大的时间字段或者过小的时间值，样例中给出的是100天，即不能写入一条数据的时间比当前时间大100天或小100天
collection.collection.configName	_indexer_default	代表创建的索引表依赖的配置集，可以设置为自己的配置集名称，参考配置集更新
create-collection.numShards	2	创建的索引表shard个数，默认为2

上面的业务含义，以30天前（假设今天是3月7日）开始创建索引，每隔7天新建一个索引，your_timestamp_l，并且它的值与当前时间在100天以内，周期性的删除90天过期的索引。

注意事项
1.业务必须带有时间字段，可以为Date类型，也可以为Long类型。
2.查询时，默认是查询全部索引表。此时，可以单独指定查询某个索引表。需要通过API或者URL获取到所有collection列表，然后提取其中的时间字段来判断真实访问的collection，可参见下面的代码样例。

删除Alias

普通的Alias可以直接通过下面的命令删除。

```
curl "http://solrhost:8983/solr/admin/collections?action=DELETEALIAS&name=your_alias_name"
```

具有自动分表的Alias删除，还需要主动删除关联的collection。

1. 取得关联Alias的collection列表

   ```
   curl "http://solrhost:8983/solr/admin/collections?action=LIST"
   ```

   其中collection名称以 `test_router_alias` 开头的都是关联该Alias的collection。

2. 删除Alias

   ```
   curl "http://solrhost:8983/solr/admin/collections?action=DELETEALIAS&name=test_router_alias"
   ```

3. 删除所有collection

   ```
   curl "http://solrhost:8983/solr/admin/collections?action=DELETE&name=collection_name"
   ```

参考文档

https://lucene.apache.org/solr/guide/7_3/collections-api.html#createaliashttps://lucene.apache.org/solr/guide/7_3/collections-api.html#list

如何指定long型时间进行查询

```
import org.apache.solr.client.solrj.SolrQuery;
import org.apache.solr.client.solrj.impl.CloudSolrClient;
import org.apache.solr.client.solrj.impl.ClusterStateProvider;
import org.apache.solr.client.solrj.response.QueryResponse;
import org.apache.solr.common.SolrDocument;
import org.apache.solr.common.util.StrUtils;
import java.time.Instant;
import java.time.ZoneOffset;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeFormatterBuilder;
import java.time.temporal.ChronoField;
import java.util.AbstractMap;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Optional;
public class SolrDemo {
  private static final DateTimeFormatter DATE_TIME_FORMATTER = new DateTimeFormatterBuilder
```

```
()
      .append(DateTimeFormatter.ISO_LOCAL_DATE).appendPattern("[_HH[_mm[_ss]]]")
      .parseDefaulting(ChronoField.HOUR_OF_DAY, 0)
      .parseDefaulting(ChronoField.MINUTE_OF_HOUR, 0)
      .parseDefaulting(ChronoField.SECOND_OF_MINUTE, 0)
      .toFormatter(Locale.ROOT).withZone(ZoneOffset.UTC);
  private static final String zkHost = "localhost:2181/solr";
  private CloudSolrClient cloudSolrClient;
  private ClusterStateProvider clusterStateProvider;
  public SolrDemo() {
    cloudSolrClient = new CloudSolrClient.Builder(
        Collections.singletonList(zkHost), Optional.empty()).build();
    cloudSolrClient.connect();
    clusterStateProvider = cloudSolrClient.getClusterStateProvider();
  }
  public void close() throws Exception {
    if (null != cloudSolrClient) {
      cloudSolrClient.close();
    }
  }
  private List<String> findCollection(String aliasName, long start, long end) {
    List<String> collections = new ArrayList<>();
    if (start > end) {
      return collections;
    }
    //基于[start, end]寻找合适的collection
    if (clusterStateProvider.getState(aliasName) == null) {
      // 拿到当前aliasName对应的所有collection列表
      // test_router_alias_2020-03-04, test_router_alias_2020-02-26, test_router_alias_2020
-02-19, test_router_alias_2020-02-12, test_router_alias_2020-02-05
      List<String> aliasedCollections = clusterStateProvider.resolveAlias(aliasName);
      // 从collection名称中提取出时间日期
      // 2020-03-04T00:00:00Z=test_router_alias_2020-03-04,
      // 2020-02-26T00:00:00Z=test_router_alias_2020-02-26,
      // 2020-02-19T00:00:00Z=test_router_alias_2020-02-19,
      // 2020-02-12T00:00:00Z=test_router_alias_2020-02-12,
      // 2020-02-05T00:00:00Z=test_router_alias_2020-02-05
      List<Map.Entry<Instant, String>> collectionsInstant = new ArrayList<>(aliasedCollecti
ons.size());
      for (String collectionName : aliasedCollections) {
        String dateTimePart = collectionName.substring(aliasName.length() + 1);
        Instant instant = DATE_TIME_FORMATTER.parse(dateTimePart, Instant::from);
        collectionsInstant.add(new AbstractMap.SimpleImmutableEntry<>(instant, collectionNa
me));
      }
      // 根据查询时间找出合理的collection
      Instant startI = Instant.ofEpochMilli(start);
      Instant endI = Instant.ofEpochMilli(end);
      for (Map.Entry<Instant, String> entry : collectionsInstant) {
        Instant colStartTime = entry.getKey();
        if (!endI.isBefore(colStartTime)) {
          collections.add(entry.getValue());
          System.out.println("find collection: " + entry.getValue());
          if (!startI.isBefore(colStartTime)) {
```

```
            break;
          }
        }
      }
    } else {
      collections.add(aliasName);
    }
    System.out.println("query " + collections);
    return collections;
  }
  public void run() throws Exception {
    try {
      // [2020-03-07 2020-03-10]
      long start = 1583538686312L;
      long end = 1583797886000L;
      String aliasName = "test_router_alias";
      String collections = StrUtils.join(findCollection(aliasName, start, end), ',');
      QueryResponse res = cloudSolrClient.query(collections, new SolrQuery("*:*"));
      for (SolrDocument sd : res.getResults()) {
        System.out.println(sd.get("id") + " " + sd.get("gmtCreate_l"));
      }
    } finally {
      cloudSolrClient.close();
    }
  }
  public static void main(String[] args) throws Exception {
    SolrDemo solrDemo = new SolrDemo();
    solrDemo.run();
    solrDemo.close();
  }
}
```

4.6. 常见问题

本文介绍您在使用全文索引过程中可能遇到的一些问题和解决方法。

创建索引时设置shard个数和replica个数

创建索引时设置shard个数和replica个数需要满足以下规则：

- 单个shard的最大document条数不能超过int的最大值，大概21亿。否则就会因lucene底层循环复用int值而导致覆盖。
- 对于replicationFactor副本数，推荐默认设置为1，并且把autoAddReplicas属性设置为false。对于有特殊要求的，例如写入非常少，读非常多，且读可能会有大量热点，那可以考虑使用replicationFactor=2或者3，可以缓解读负载均衡。
- 假设实例有n个服务节点，每个节点放m个shard，需要遵循如下公式：

  ```
  索引的总数据量 > n X m X 21亿
  ```

  如果一个服务节点的m个数太大了，例如超过节点CPU的个数，那就可能要考虑扩容集群或者升配。对于一个索引而言，每个服务节点放一个shard开始，即m=1，如果不满足再依次增加m的值，直到能满足上述公式为止即可。

- 对于单个shard在条数不能超过int的最大值，大概21亿的情况下，它的存储也尽量不能太大，例如一个shard保存了20亿，按照1KB一个doc，总数据量达到2TB左右，这对一个server来说可能会有点大了，对应如果大量扫描操作会出现异常，推荐扩容节点，分担大量存储扫描取数据的压力。控制在单个shard的总量数据也不要太大。当然这个要结合查询特点和数据特点，例如单个document就10KB和200字节碰到这种情况都是不一样的。

> **说明** shard和replica的个数后期可以调整，但建议在创建索引时提前设计好。

HBase同步数据到Search索引的延时是多少

```
索引同步的延时时间 = 数据同步延迟 + commit时间
```

没有堆积情况下，同步延时主要为框架开销，毫秒级别（如果有积压情况下，延时会变长，需要增加节点来增加同步能力）默认commit时间为15秒，对于写很少的客户可以设置为1秒、3秒、5秒，对于写入量大的用户，不推荐设置过小，不然小文件会比较多，服务会频繁的执行文件合并操作，影响整体的性能。

使用预排序功能

排序是非常消耗资源的，在数据量特别大的时候，不仅查的慢，还特别占用系统资源，如果本身存储的数据已经按照某个字段预先排好序，检索性能会有明显的提升，特别是在大数据量上对比的时候，此特点效果更明显。下面简单描述下如何配置预排序：

- 修改solrconfig.xml中的MergePolicy，具体操作请参见SolrConfig。
- 查询时指定参数 `segmentTerminateEarly=true` ，示例如下：

```
<mergePolicyFactory class="org.apache.solr.index.SortingMergePolicyFactory">
<str name="sort">timestamp desc</str>
<str name="wrapped.prefix">inner</str>
<str name="inner.class">org.apache.solr.index.TieredMergePolicyFactory</str>
<int name="inner.maxMergeAtOnce">10</int>
<int name="inner.segmentsPerTier">10</int>
</mergePolicyFactory>
```

示例中 `<str name="sort">timestamp desc</str>` 配置参数表示插入数据时会按照timestamp字段进行预先倒序排序，执行查询如下：

```
curl "http://localhost:8983/solr/testcollection/query?q=*:*&sort=timestamp+desc&rows=10&segmentTerminateEarly=true"
```

参数上加上 `segmentTerminateEarly=true` 后，显示效果会比没有设置预排序的快很多，尤其是排序数据量TB级别之后，效果更明显。

> **说明**
> - 查询时，指定的sort必须与配置的MergePolicy中指定的一致，否则不起效果。
> - 查询时需要指定segmentTerminateEarly参数，否则也会进行全排。
> - 使用了这个预排序返回的结果中，numFound是不准确的。