云数据库 HBase

HBase Solr 全文引擎

HBase Solr 全文引擎

快速入门

服务介绍

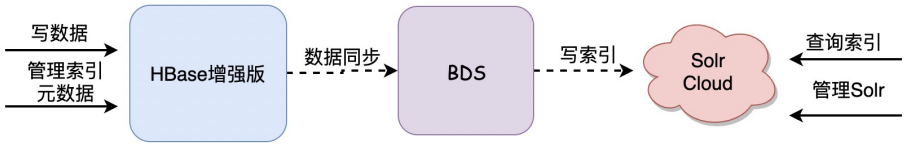
简介

Solr是分布式全文检索的最佳实践之一。Solr支持各种复杂的条件查询和全文索引。通过结合HBase、Solr，可以最大限度发挥HBase和Solr各自的优点，从而使得我们可以构建复杂的大数据存储和检索服务。
HBase+Solr适用于：**需要保存大数据量数据，而查询条件的字段数据仅占原数据的一小部分，并且需要各种条件组合查询的业务。**例如：

- 常见物流业务场景，需要存储大量轨迹物流信息，并需根据多个字段任意组合查询条件
- 交通监控业务场景，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
- 各种网站会员、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。

云HBase增强版提供的全文索引服务深度融合了HBase和Solr，**能够自动将用户写入HBase的数据实时数据写入到Solr中，用户无需双写HBase和Solr。**

在使用过程中，用户全程只需要和HBase和Solr交互即可，BDS做为HBase到Solr的同步通道，完全对用户透明，用户只需要通过HBase增强版对外提供的建立、修改外部索引的接口操作，即可完成索引元数据的管理。



在此方案中，HBase，数据同步通道BDS和Solr都是以独立集群的方式存在，因此用户可以对其分开管理。如果Solr能力不足，只需要库容Solr集群。如果BDS的同步能力不足，可以独立扩容BDS。HBase/BDS/Solr可以针对使用的场景选择不同的机型。并且三者之间相互独立，不会因为Solr占用过多CPU资源而影响HBase，或者反之。独立的部署形态大幅降低了使用成本，提升了使用灵活性和稳定性。

购买指南

使用HBase+Solr的方案需要先购买HBase增强版和BDS集群。并且在HBase控制台页面点击开通全文索引功能，完成Solr的购买和关联。具体参见购买指南

使用指南

参见快速入门和索引管理

Solr最佳实践

参见Solr最佳实践

开通指南

开通前准备

开通Solr全文索引服务必须先购买一个HBase增强版集群和一个在同一个VPC内的BDS集群。如果需要在已经购买的HBase和BDS上开通全文索引服务，则可以跳过此步。购买HBase增强版和BDS可以在HBase产品首页点击购买按钮选择相应的集群和配置，或者在HBase控制台点击创建HBase集群和BDS集群

开通全文索引服务

1. 在购买的HBase增强版集群的控制台页面左边的Tab中点击”全文索引”字样开通服务

2. 在弹出的开通页面中，选择所使用的BDS。**注意：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Solr的机器规格搜索引擎Solr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Solr的机器选型有疑问，可以在钉钉咨询HBase答疑或者提工单咨询。

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-t4n33q8he022k7g4v。BDS实例bds-t4n579vl2f74wqa1（可以在服务 / 版本 / 主实例一栏中看到关联了ld-t4n33q8he022k7g4v），和Solr实例ld-t4n33q8he022k7g4v-m1-se（可以在服务 / 版本 / 主实例一栏中看到关联了ld-t4n33q8he022k7g4v）。当所有实例都从创建中变成运行中，即可以开始使用

快速开始

下载HBase Shell

从HBase增强版Shell访问页面下载最新版本的HBase Shell。并根据Shell使用指导的指导完成Shell访问HBase的相关配置。在启动Shell连接之前，请确认已经配置好白名单

在HBase中建表

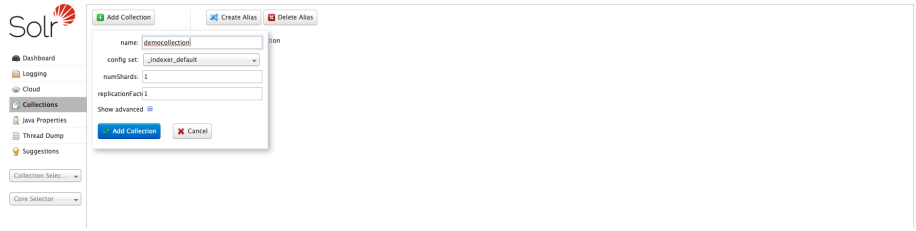
```
hbase(main):002:0> create 'testTable', {NAME => 'f'}
```

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

在Solr中创建Collection

在Solr集群的控制台中，点击Web页面进入Solr的WebAdmin。注意使用前需要设置好访问白名单和访问密码

如果没有对Solr有特殊的配置，请选择_indexer_default 配置目录 如需要修改Solr的配置，请参考Solr schema管理， 如下图：



建立映射关系

假设我们需要在testTable这个表中f:name这一列（family为f，qualifier为name ）映射到Solr中 ’ democollection’ 这个collection中的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"
}
]
}
```

具体的Schema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。

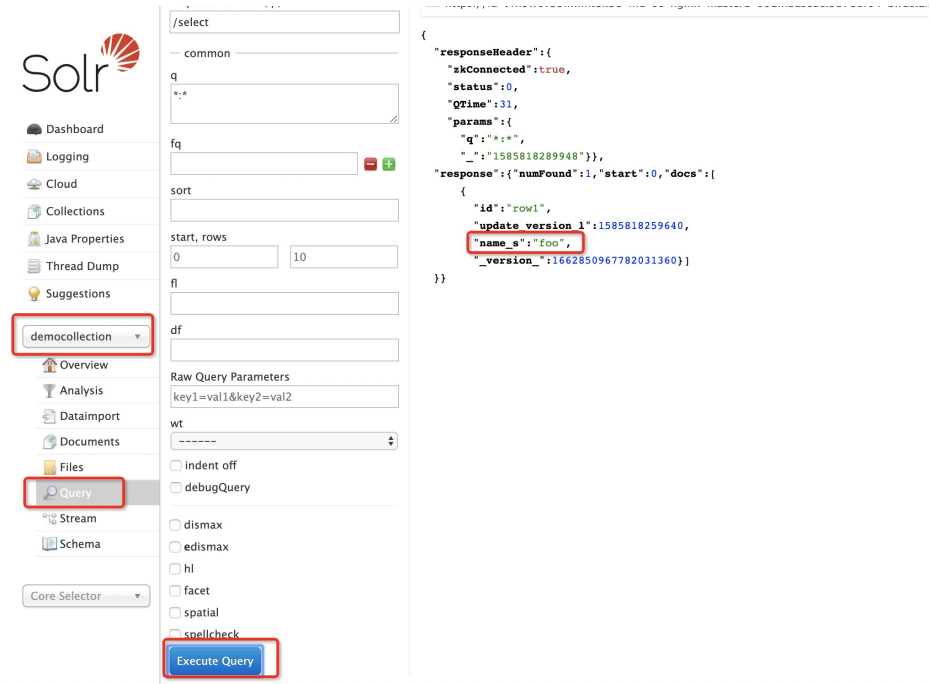
在HBase中写入数据

写入数据参见JavaAPI章节，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
```

在Solr中查询数据

在SolrWebUI中，选中刚才建立的collection，点击Query查询，就可以查到刚才写入的一行数据。



回查HBase

在通常的HBase+Solr使用场景中，会将HBase的一行中部分列同步到Solr中做全文索引，利用Solr强大的搜索能力提升多维查询能力。在Solr返回的结果中，会有每个document的idfield。这个field即为这一行在HBase中的rowkey。拿到这个rowkey后，回查HBase，就可以获得完整数据。

注意事项

1. HBase自定义时间戳的支持

当用户在给表中的列加上Solr索引之后，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），当用户写入的数据带时间戳时，会抛出User defined timestamp is not allowed when external index is enabled...的Exception。由于Solr不支持列时间戳，不带时间戳可以简化HBase同步Solr的逻辑。如果：

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。

用户可以通过修改表属性的方式，打开对自定义时间戳的支持。用户可以通过Shelll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. HBase中删除的支持

HBase中可以删除整行，删除某个family下的所有列，或者删除某个固定的列,HBase还支持删除指定某个版本的列。由于Solr并不支持多版本，如果在HBase删除指定的某个版本，会导致HBase和Solr中的数据不一致。因此当用户给HBase中的列加上Solr索引之后，删除这个列的指定版本的行为将会被禁止。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famliy里所有的列 --支持
delete.addFamily(Bytes.toBytes("f"));
//删除f这个famliy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，Solr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Solr，不会命中这种只有id的行。但如果查询到只有id这个field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Solr的TTL功能在一定时间后自动删除这些行。

TTL的支持

HBase和Solr都支持数据TTL过期，但是HBase的TTL机制与Solr的TTL机制不一样，HBase是单个KV过期，而Solr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，同时Solr中数据过期后，如果删除事件还没有完成，过期数据仍然可能读到，而HBase不会有这样的问题。因此在开启TTL后，会导致Solr和HBase中数据短暂的不一致。比如在Solr中查到对应的rowkey再反查HBase时，由于数据在HBase中已经过期，数据可能查不到，在开启TTL后，用户需要处理此类case。

HBase索引管理

管理HBase全文索引

准备工作

学习快速入门部分，并在Shell页面下载好最新版本的Shell，并配置好连接和在HBase的控制台中开启白名单。

映射Schema详解

HBase索引映射schema即管理HBase表的列映射到Solr的Collection Field和具体的类型。HBase增强版采用Json方式管理映射Schema。一个典型的映射Schema如下所示：

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

这个schema代表将namespace名为default，表名为testTable的HBase表的数据同步到Solr的democollection这个Collection中。其中f:name这一列（Family名和列名用冒号隔开）映射到Solr中的name_s这一列，f:age这一列映射到Solr中的age_l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	映射的Solr的collection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
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这个参数是HBase表rowkey映射到Solr Document中id这个unique field(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Solr Document中的id field。用户在solr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Solr Document中的id field。用户在solr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。

注意：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Solr 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	Solr中对应的列名，Solr中每一列都必须预定义，或者使用动态列，如name_s，Solr的schema配置参见Solr Schema配置
type	HBase中存入的这一列的type，**注意是HBase中source这一列的类型，非Solr中targetField的类型**。可以配置的值为INT/LONG/STRING/BOOLEAN/FLOAT/DOUBLE/SHORT/BIGDECIMAL

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Solr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solr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Solr中映射的列。Solr是一个强Schema的系统，每一列都必须在Solr的managed_schema中预先定义（Solr的schema配置参见Solr Schema配置 ）。但是这里我们推荐使用Solr的dynamicField功能，通过后缀自动识别这一列的类型，这样不用修改Solr的Schema就可以往Solr中写入新的列。比如name_s代表这一列在Solr中的类型为String。

HBase中source的类型type并不需要和Solr field的数据类型一一对应。比如用户可以定义Source列f:age的type为String，而solr的targetField为age_l(代表solr中这一列的类型为long)，在写入时，Solr会自动把String转化成long。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Solr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', TARGETFIELD => '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'
```

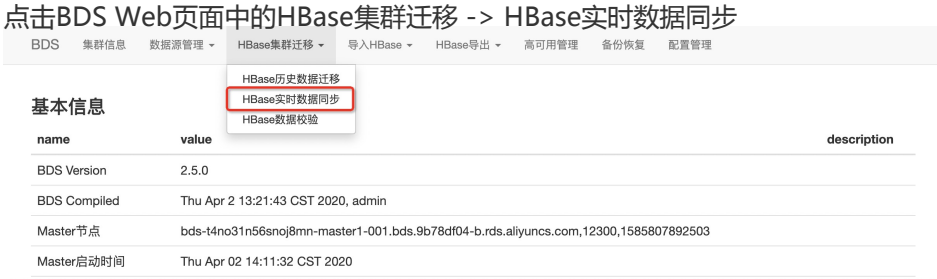
查看索引同步状态

查看状态

HBase和Solr的数据同步均由关联的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. 点击BDS Web页面中的HBase集群迁移 -> HBase实时数据同步



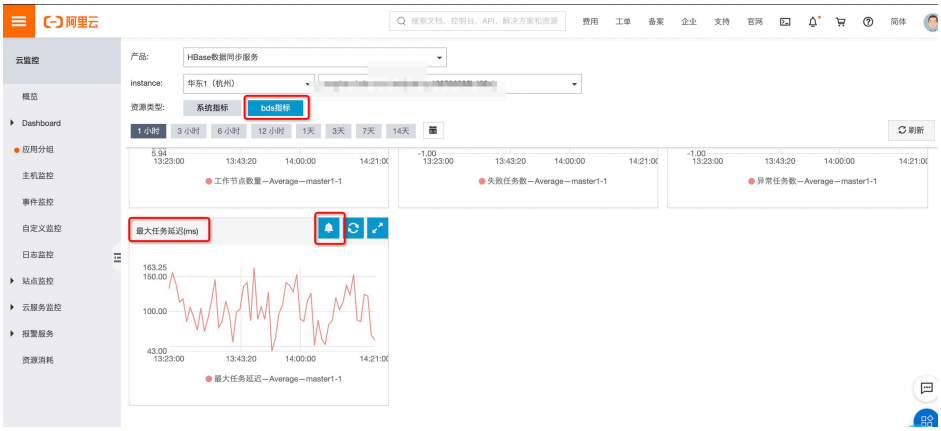
3. 选中Solr的通道名，点击进入后就可以看到同步详情。通常情况下，用户只需要关注HBase到Solr的同步状态和延迟这两个状态

监控和报警

HBase和Solr的数据同步均由关联的BDS完成，因此数据同步时的延迟指标在对应的BDS实例中，可以通过查看BDS的监控获取同步延迟信息，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，点击铃铛按钮配置相关报警



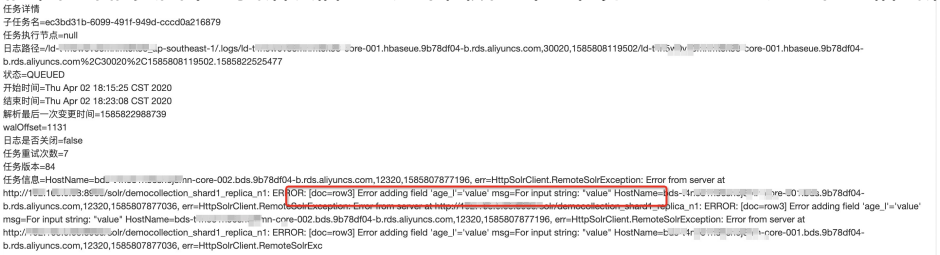
常见问题处理

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

1. 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写入Solr失败。由于HBase同步Solr采用的是同步HBase WAL的方式进行，一旦WAL中有错误数据，同步就会卡住。

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。如下图所示中的任务显示：row3这一行的格式错误导致无法写入solr。



为了解决这个问题，用户可以选择在HBase中删除row3这一行，或者修改row3这一行对应的列的数据，将其修改正确，同步就可以继续进行下去。如果遇到无法解决的问题，可以在钉钉上联系hbase答疑或者提工单咨询。

2. 同步速度慢

如果观察到同步点位仍然在推进，但是界面中的写入详情和读详情中读取Log和写入Solr的速度都很慢，可能是Solr集群的写入能力达到了瓶颈，需要扩容或者升配，具体可以参考Solr集群的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hbase答疑或者提工单咨询。

3. 通道队列中的Log积压严重

如果观察到界面中的写入详情和读详情中Log和写入Solr的速度都不慢，但是仍然在通道队列中显示在队列中的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。也有可能是达到了Solr的写入瓶颈，需要扩容Solr集群，具体可以参考BDS和Solr集群的监控，如果有相关的问题，可以在钉钉上联系hbase答疑或者提工单咨询。

全量数据索引构建

HBase到Solr的实时同步只会对当前写入的数据在Solr中构建索引。之前写入的历史数据，必须通过全量数据索引构建功能完成构建。

全量数据构建

在HBase Shell中调用build_external_index命令为HBase中的历史数据建立索引。该命令会为该表中的所有数据，按照solr索引映射关系（之前使用alter_external_index或者add_external_index_field命令加入的映射关系）在solr中全量建立索引。**注意：在全量build索引过程中，对表的schema修改操作会卡住，直到构建完成才能继续执行**

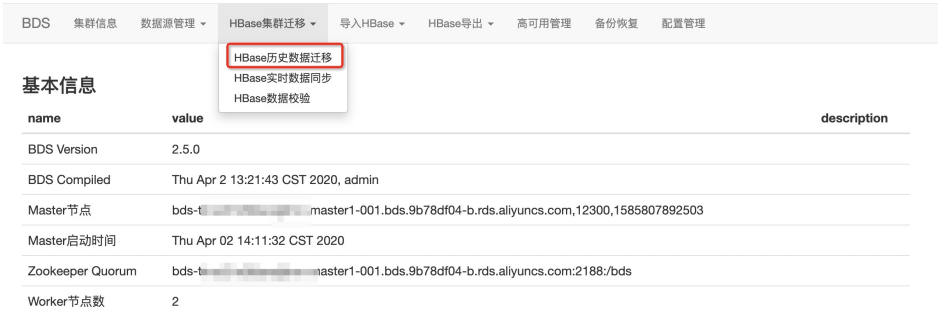
```
hbase shell> build_external_index 'testTable'
```

查看全量构建进度

HBase中历史数据的全量构建由关联的BDS完成，因此在关联的BDSWeb页面可以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. 点击BDS Web页面中的HBase集群迁移 -> HBase历史数据迁移



3. 点击相应的任务名，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。用户可以查看每个子任务的状态，如果有失败，可以点击详情查看失败原因。如果有相关的问题，可以在钉钉上联系hbase答疑或者提工单咨询。

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'
```

或者用户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。

Solr最佳实践

访问Solr WebUI

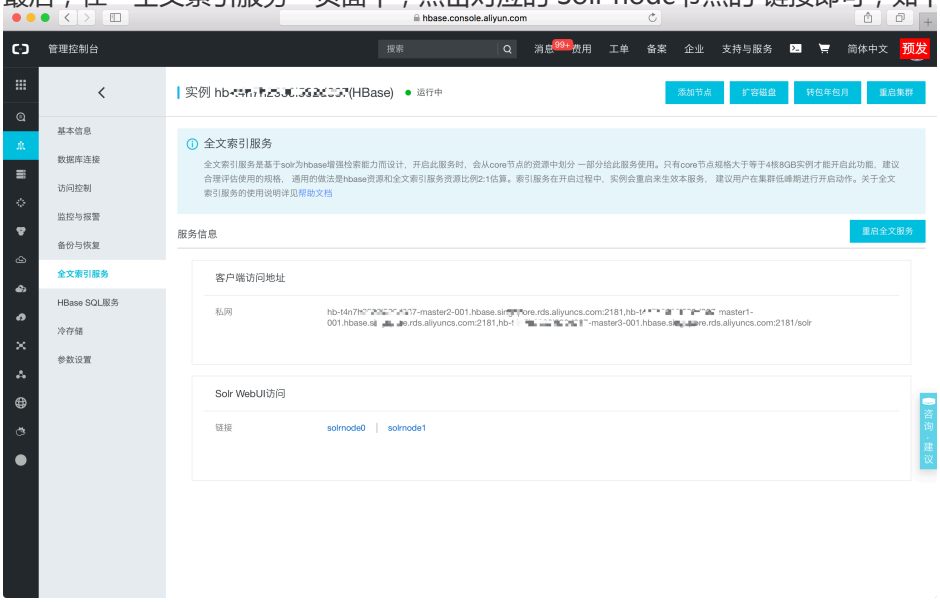
Solr WebUI访问

本文简述Solr WebUI访问方法，其大体与HBase WebUI访问流程一样。下面再进行简单的描述。

访问Solr WebUI前提

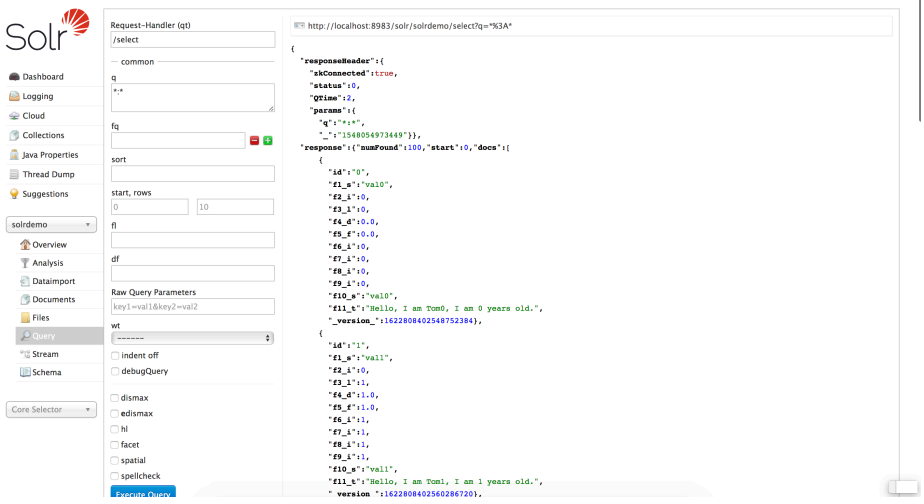
- Solr WebUI访问的用户，和HBase WebUI访问一样，需要设置用户本机的ip到HBase实例白名单中，详见参考
- 第一次访问时，需要设置好WebUI访问账户密码，详见参考

最后，在“全文索引服务”页面中，点击对应的 Solr node节点的 链接即可，如下图：



Solr WebUI界面功能简单介绍

SolrCloud是由多个Solr Server通过 Zookeeper协作起来，每个Solr Server都是一个单独的个体，每个Solr实例的UI界面如下：



从左边栏可以看出，主要关注的基本功能如下：

- Dashboard 本Solr node节点概况
- Logging 日志查看及日志级别设置

- Cloud 查看各个 collection的shard/replica 分布与状态概况
- Collections 查看各个collection基本shards属性
- Thread Dump 显示本次访问时，jvm的thread dump快照
- Collection下拉框，支持对collection进行简单的查询和插入操作，以及schema查看
- replica下拉框，查看具体的replica的简单统计情况

更多关于Solr界面的介绍与基础使用，请参考链接 Overview-Solr-Admin-UI

如何配置Schema

Solr 客户端工具包准备

首先需要下载客户端并解压，下载地址

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

修改 alisolr-7.3.8-bin/bin/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"
```

此地址见solr开通页面的客户端访问地址，如下：

注：上述配置流程中的 ZK 地址是内网地址，如果是想通过公网访问，请填写 ${公网 ZK 地址}/solr 即可。公网 ZK 地址获取，请参考公网访问。

下载默认的配置集模板

进入命令行，下载提供的默认配置集_indexer_default，在此基础上进行编辑，添加业务自定义的配置。

```
cd alisolr-7.3.8-bin/bin
./solr zk ls /configs // 查询当前的配置集列表
./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，将会在当前目录下看到一个conf的目录。在其中有两个重要的文件：managed-schema和solrconfig.xml，索引列相关的配置都在managed-schema中，详细的配置说明可参考社区文档。

创建新的配置集

下面给出一个简单的示例：

- 打开managed-schema文件
- 增加两个新的索引列定义

```
<field name="name" type="string" indexed="true" stored="true" required="false" multiValued="false"
/>
<field name="age" type="pint" indexed="true" stored="true" docValues="true" multiValued="false" />
```

name是string类型，age是基本int类型(pint代表的就是int，plong代表的就是long)，两个列都需要建立索引，并且需要存储。

上面每个列都需要自己定义，当列非常多时，定义起来会比较复杂，此时，可以使用Solr提供的动态列能力，可以参考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。

```
<!--
  动态定义字段，方便业务使用，不需要按照上面的格式一个一个的定义字段。
  例如，
  当您写入myfield_i时，会自动识别为*_i
  当您写入myfield_l时，会自动识别为*_l
  当您写入myfield_s时，会自动识别为*_s
-->
<dynamicField name="*_i"  type="pint"    indexed="true"  stored="true"/>
<dynamicField name="*_is" type="pints"    indexed="true"  stored="true"/>
<dynamicField name="*_s"  type="string"  indexed="true"  stored="true" />
<dynamicField name="*_ss" type="strings"  indexed="true"  stored="true"/>
<dynamicField name="*_l"  type="plong"   indexed="true"  stored="true"/>
<dynamicField name="*_ls" type="plongs"    indexed="true"  stored="true"/>
<dynamicField name="*_t" type="text_general" indexed="true" stored="true" multiValued="false"/>
<dynamicField name="*_txt" type="text_general" indexed="true" stored="true"/>
<dynamicField name="*_b"  type="boolean" indexed="true" stored="true"/>
<dynamicField name="*_bs" type="booleans" indexed="true" stored="true"/>
<dynamicField name="*_f"  type="pfloat"  indexed="true"  stored="true"/>
<dynamicField name="*_fs" type="pfloats"  indexed="true"  stored="true"/>
<dynamicField name="*_d"  type="pdouble" indexed="true"  stored="true"/>
<dynamicField name="*_ds" type="pdoubles" indexed="true"  stored="true"/>
<dynamicField name="*_str" type="strings" stored="false" docValues="true" indexed="false" />
<dynamicField name="*_dt"  type="pdate"    indexed="true"  stored="true"/>
<dynamicField name="*_dts" type="pdate"    indexed="true"  stored="true" multiValued="true"/>
<dynamicField name="attr_*" type="text_general" indexed="true" stored="true" multiValued="true"/>
```

- 上传配置集

修改完后，可以创建一个属于自己的配置集（建议每个collection对应一个配置集），命令如下：

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

- 在Solr WEB界面查看配置是否上传成功

建议

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Collection有自己的配置集，不建议多个Collection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修改，_indexer_default 配置集中有HBase同步Solr的几个关键配置，如solrconfig.xml中的DocBasedVersionConstraintsProcessorFactory UpdateProcessor配置和managed_schema中的uniquekey配置。

Solr Java API访问

Solr支持多语言的访问，本篇主要介绍如何使用Java访问Solr服务。

1、获取集群连接地址

在全文索引开通页面，点击进入到“Solr实例”后，查看“数据库连接” ，参见下图

如果需要公网访问地址，可以点击“开通外网地址”即可，然后替换到应用中，如果需要停止公网访问，请点击“释放外网地址”。

2、配置客户端SDK依赖全文服务兼容开源社区协议，可直接依赖开源客户端版本。

```
<dependency>
<groupId>org.apache.solr</groupId>
<artifactId>solr-solrj</artifactId>
<version>7.3.1</version>
</dependency>
```

3、代码示例样例代码：基于查询条件获取id，这里的id值是和HBase rowkey对应的。

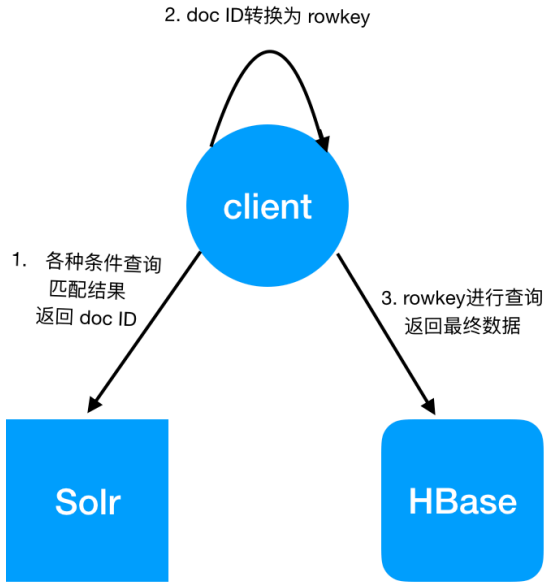
```
String zkHost = "zk1:2181,zk2:2181,zk3:2181/solr"
CloudSolrClient cloudSolrClient = new CloudSolrClient.Builder(Collections.singletonList(zkHost),
Optional.empty()).build(); //CloudSolrClient是线程安全的，应用多线程可以共享一个对象
SolrQuery solrQuery = new SolrQuery("f1_s:val99");
QueryResponse response = client.query(collection, solrQuery);
SolrDocumentList documentList = response.getResults();
for(SolrDocument doc : documentList){
String id = (String)doc.getFieldValue("id");
//do something
}
cloudSolrClient.close();
```

4、更多样例代码，参考。

索引查询示例

索引查询示例

本文介绍如何进行各种条件查询索引，返回匹配结果id后，转换为rowkey查询HBase取出最终完整原数据。流



程如下：

客户端准备

本示例展示使用Java客户端SolrJ来操作solr，并使用Java的HBase API访问HBase。

Java项目工程添加依赖如下：

```
<dependency>
<groupId>org.apache.solr</groupId>
<artifactId>solr-solrj</artifactId>
<version>7.3.1</version>
</dependency>
```

各种条件查询，获取doc ID

```
String zkHost = "zk1:2181,zk2:2181,zk3:2181/solr"
CloudSolrClient cloudSolrClient = new CloudSolrClient.Builder(Collections.singletonList(zkHost),
Optional.empty()).build();
SolrQuery solrQuery = new SolrQuery("f1_s:val99");
QueryResponse response = client.query(collection, solrQuery);
SolrDocumentList documentList = response.getResults();
for(SolrDocument doc : documentList){
String id = (String)doc.getFieldValue("id");
//do something
}
```

更多精确、模糊、范围、facet、stats、and/or组合等查询，见 github demo地址

doc ID转换成rowkey

1. string类型doc id如果默认 index_conf.xml配置中，不指定unique-key-formatter，或者显式指定使用string，如：

   ```
   <indexer table="solrdemo" unique-key-formatter="string">
   ...
   </indexer>
   ```

   那么，doc id转成rowkey过程如下：

   ```
   // String id = "xxxx";
   org.apache.hadoop.hbase.util.Bytes.toBytes(docId)
   ```

2. hex类型doc id如果默认 index_conf.xml配置中，unique-key-formatter指定使用hex，如：

   ```
   <indexer table="solrdemo" unique-key-formatter="hex">
   ...
   </indexer>
   ```

   那么，doc id转成rowkey过程如下：

```
// String id = "xxxx";
org.apache.commons.codec.binary.Hex.decodeHex(id.toCharArray());
```

此过程借助 commons-codec-1.9.jar的方法转换。此jar包依赖如下：

```
<dependency>
<groupId>commons-codec</groupId>
<artifactId>commons-codec</artifactId>
<version>1.9</version>
</dependency>
```

获取最终数据

最终我们拿根据各种条件查询匹配到的rowkey，如需获取这个rowkey的完整数据, 只要进行HBase的 get操作即可。HBase的查询访问支持Java api原生方式 ，详见参考; 也可以通过thrift支持c#、python、go等多语言, 详见参考

Solr分库分表(Alias功能)

前言

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。

注意：下面文档中关于curl命令如何访问，具体可参考。

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&collections=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。

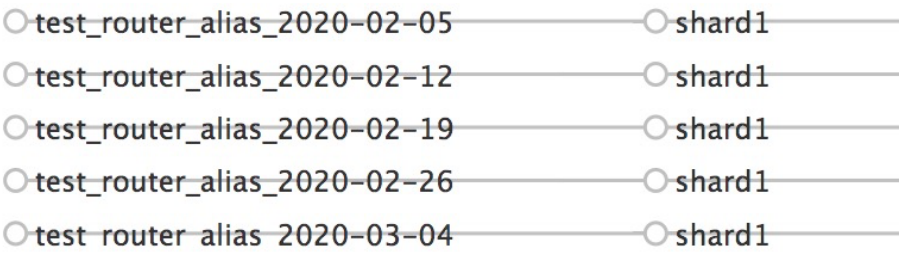
```
curl
"http://solrhost:8983/solr/admin/collections?action=CREATEALIAS&name=test_router_alias&router.start=NOW-30DAYS/DAY&router.autoDeleteAge=/DAY-90DAYS&router.field=your_timestamp_l&router.name=time&router.interval=%2B7DAY&router.maxFutureMs=8640000000&create-collection.collection.configName=_indexer_default&create-collection.numShards=1"
```

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
create-collection.numShards	1	创建的索引表shard个数，默认为1

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。

效果如下(如何访问Solr Web)



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#createalias
https://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<>(aliasedCollections.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, collectionName));
}

// 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();
}
}
```

分词使用说明

分词使用说明

全文索引服务涵盖丰富的查询功能，facet、排序/分页、任意复杂条件组合查询、function查询、stats统计等，其中还有一项重要的功能，就是分词。如我们常见的视频标题关键字搜索、商品标题关键字搜索等，都是利用了全文引擎的分词功能。
本文重点介绍两个类型分词器，分别是默认的英文分词、ik中文分词。

英文分词器

默认schema中定义了 text_general字段类型，此字段类型的分词器配置如下：

```
<!-- A general text field that has reasonable, generic
cross-language defaults: it tokenizes with StandardTokenizer,
removes stop words from case-insensitive "stopwords.txt"
(empty by default), and down cases. At query time only, it
also applies synonyms.
-->
<fieldType name="text_general" class="solr.TextField" positionIncrementGap="100" multiValued="true">
<analyzer type="index">
```

```
<tokenizer class="solr.StandardTokenizerFactory"/>
<filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
<!-- in this example, we will only use synonyms at query time
<filter class="solr.SynonymGraphFilterFactory" synonyms="index_synonyms.txt" ignoreCase="true"
expand="false"/>
<filter class="solr.FlattenGraphFilterFactory"/>
-->
<filter class="solr.LowerCaseFilterFactory"/>
</analyzer>
<analyzer type="query">
<tokenizer class="solr.StandardTokenizerFactory"/>
<filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
<filter class="solr.SynonymGraphFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true"/>
<filter class="solr.LowerCaseFilterFactory"/>
</analyzer>
</fieldType>
```

配置的分词组件功能如：

- StandardTokenizerFactory 标准切词器，负责切分分词汇
- StopFilterFactory 停用词过滤器，过滤类似 “the”、 “are”这种停用词
- SynonymGraphFilterFactory 近义词过滤器
- FlattenGraphFilterFactory 配合上述近义词过滤器完成近义词的过滤替换处理
- LowerCaseFilterFactory 小写过滤器

例如有句子description为： “A contented mind is the greatest blessing a man can enjoy in this world”
按照分词的短语查询如下：

```
SolrQuery solrQuery = new SolrQuery("description:\"greatest blessing\"");
QueryResponse response = client.query(collection, solrQuery);
```

就可以匹配到这个句子。另外属于分词的其他查询方式如：按某词汇term查询、短语查询、近似查询，详见 github demo例子
要使用这个英文的分词，只要在schema中设置字段类型为“text_general”即可。如需定制分词，可以进一步了解 Solr的analyzers章节

中文分词器

中文分词器有很多，这里介绍一款开源的ik分词器，官方介绍地址
默认配置集_indexer_default中定义了text_ik的字段类型，如下：

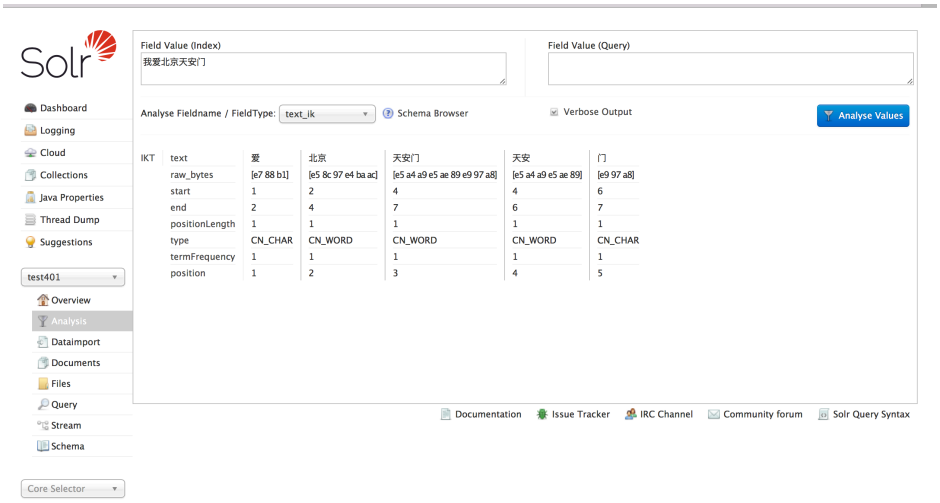
```
<fieldType name="text_ik" class="solr.TextField">
<analyzer type="index">
<tokenizer class="org.wltea.analyzer.lucene.IKTokenizerFactory" useSmart="false" />
</analyzer>
<analyzer type="query">
<tokenizer class="org.wltea.analyzer.lucene.IKTokenizerFactory" useSmart="true" />
</analyzer>
</fieldType>
```

配置这个类型后，只要定义字段的类型为 “text_ik” ，那么它就可以默认按照中文分词了。

```
<field name="desc_ik" type="text_ik" indexed="true" stored="true" docValues="false" />
```

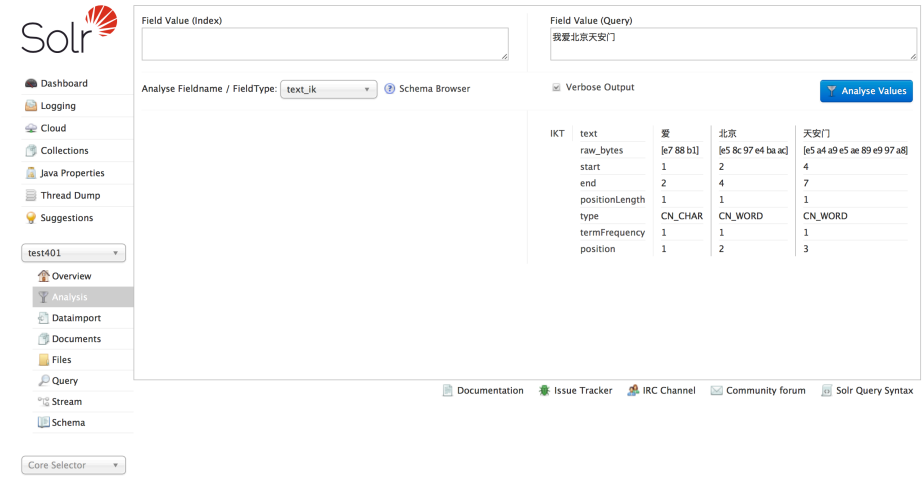
useSmart表示是否开启智能分词，智能分词会根据分词语法分词后，根据一些规则进一步挑选更合理的词汇，例如最长的完整词汇等规则。我们可以在Solr WebUI里尝试分词效果，如下：

index阶段 useSmart=false，效果如下：



IKT	text	爱	北京	天安门	天安	门
	raw_bytes	[e7 88 b1]	[e5 8c 97 e4 ba ac]	[e5 a4 a9 e5 ae 89 e9 97 a8]	[e5 a4 a9 e5 ae 89]	[e9 97 a8]
	start	1	2	4	4	6
	end	2	4	7	6	7
	positionLength	1	1	1	1	1
	type	CN_CHAR	CN_WORD	CN_WORD	CN_WORD	CN_CHAR
	termFrequency	1	1	1	1	1
	position	1	2	3	4	5

query阶段 useSmart=true，效果如下：



IKT	text	爱	北京	天安门
	raw_bytes	[e7 88 b1]	[e5 8c 97 e4 ba ac]	[e5 a4 a9 e5 ae 89 e9 97 a8]
	start	1	2	4
	end	2	4	7
	positionLength	1	1	1
	type	CN_CHAR	CN_WORD	CN_WORD
	termFrequency	1	1	1
	position	1	2	3

中文分词的词库扩展

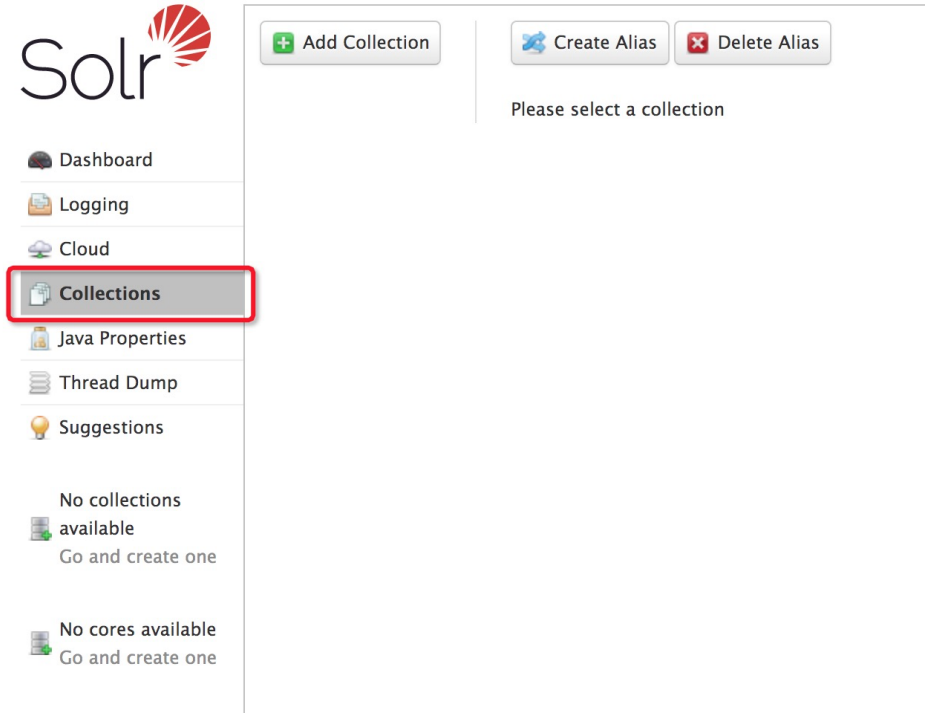
如需扩展ik的中午字典库，请联系“云HBase答疑”客服。
如果还有自定义的分词器，也可以联系“云HBase答疑”客服 咨询如何集成。

如何创建Collection

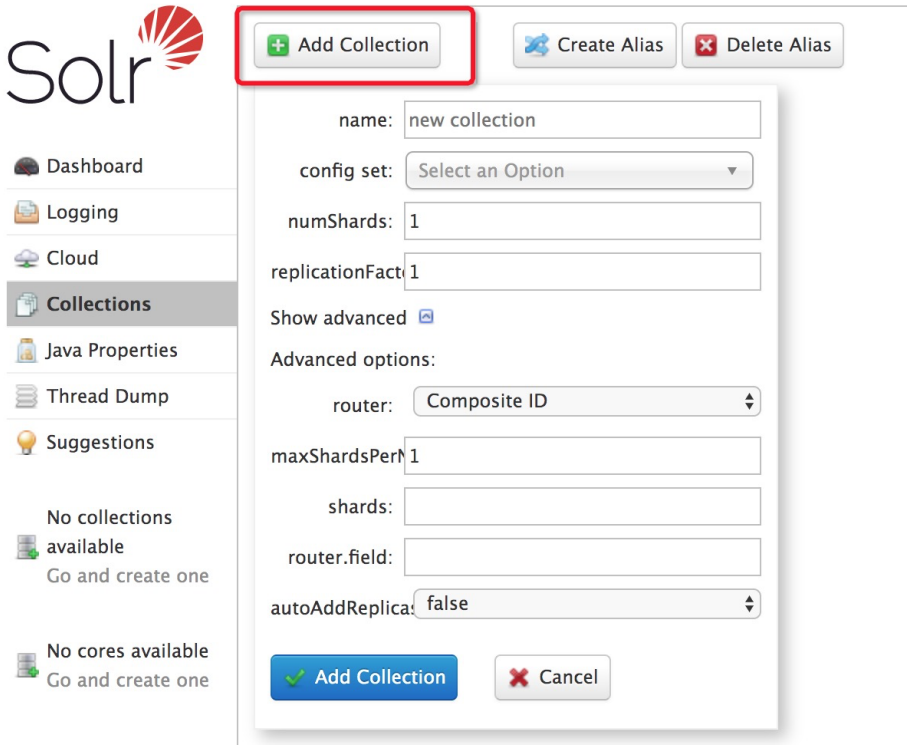
可以通过Java API或者Solr WEB来操作Collection，下面主要介绍如何在Solr WEB上创建和删除Collection。

创建

首先要登录到Solr的WEB界面，可参考：**如何登录Solr WEB**。可以看到有如下的界面：



可以点击“Add Collection”来创建新的Collection：



图中各个参数的含义

name：Collection的名称，需要自己定义，遵循字母、数字、下划线的命名规则。必填。

config set：选择一个配置集，配置集将会与这个Collection产生关联，建议每个Collection对应一个属于自己的config set，如果没有特殊配置，可以直接选择_indexer_default这个config set。如何创建新的配置集，参考。必填。

numShards：分片个数，一般与节点数保持一致，这样可以确保数据随机分配到不同的节点上。如果分片数大于节点个数，需要配合下面的maxShardsPerNode参数来设置，默认是1。

replicationFactor：每个分片的副本数，默认是1。在全文索引服务中，建议采用默认值1。

router：数据路由规则，默认是采用hash方式来将数据分配到不同的分片上，建议采用默认值Composite ID。

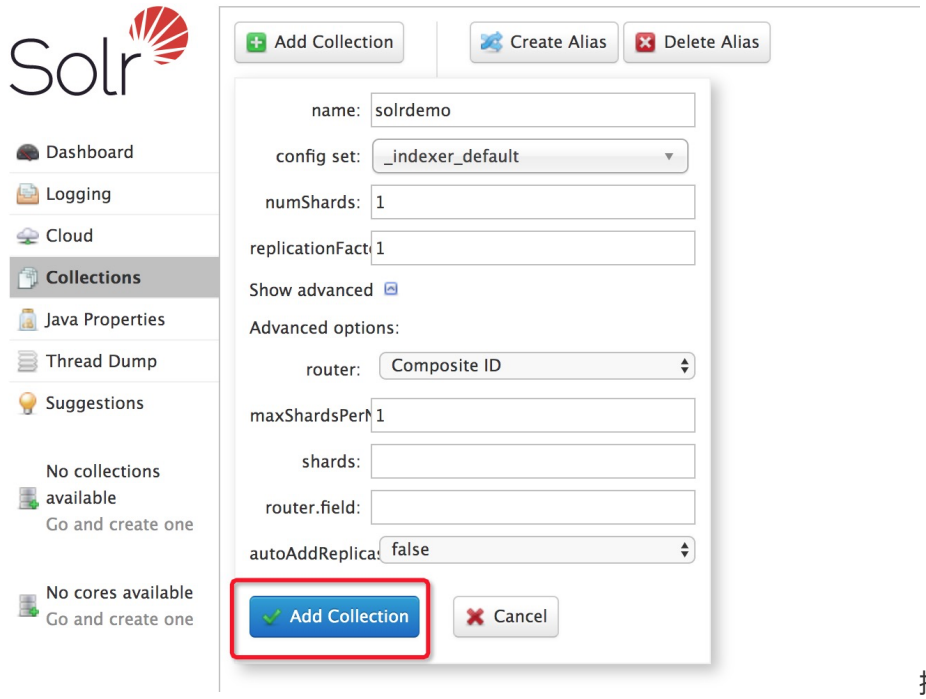
maxShardsPerNode：每个节点上最多有几个当前Collection的分片，在replicationFactor=1的前提下，需要确保下面的公式成立

```
节点个数*maxShardsPerNode >= numShards
```

shards：默认不需要填。

router.field：默认不需要填。

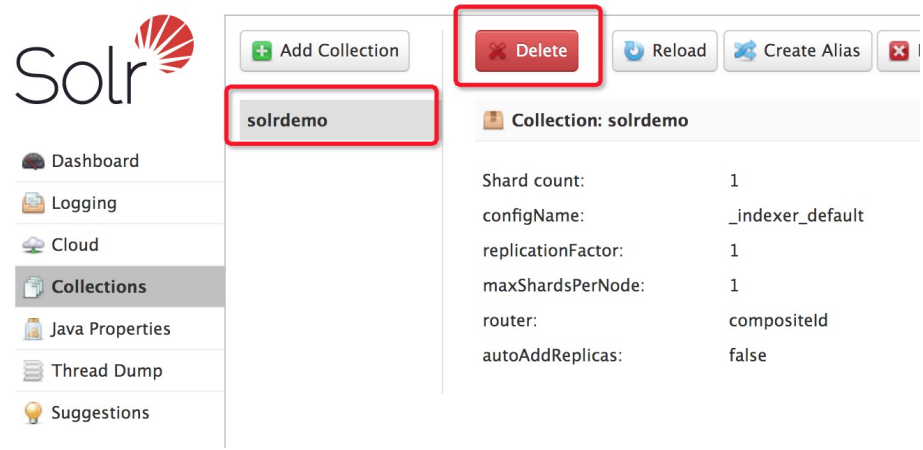
autoAddReplicas：当一个节点宕掉时，当前Collection的异常分片是否需要自动移动到其它节点上提供服务。默认值为false。



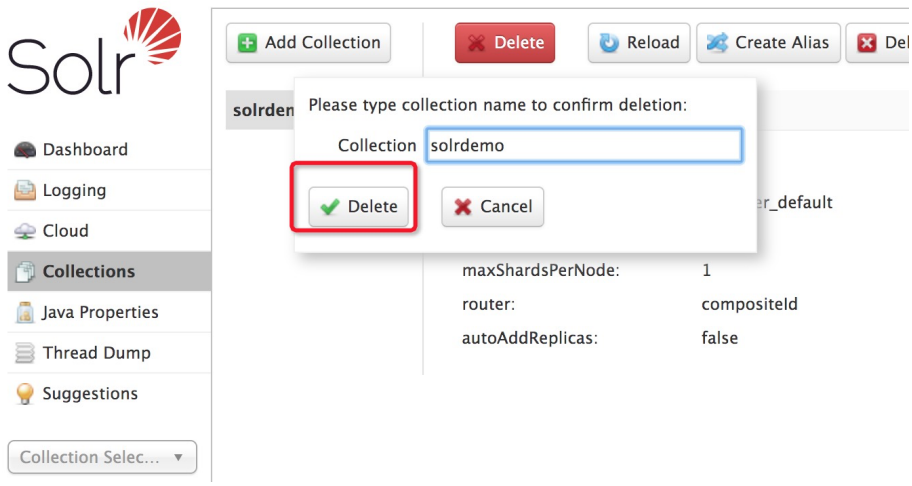
按照上面的配置，点击“Add Collection”，即可成功创建一个新的Collection：solrdemo。

删除

删除Collection，操作如下，选中需要删除Collection名称，点击“Delete”



在弹出的对话框中输入删除的Collection名称即可。



建议

replicationFactor保持默认值1，当需要提高查询并发时，再考虑动态添加副本
删除操作需要慎重，索引数据将会一并删除。

索引数据可见性

写入Solr中的数据，只有等到commit才是可见的，何时执行commit是需要配置的，当前有两种commit方式：soft commit、hard commit。

solrconfig.xml中的配置

Collection在创建时需要指定配置集，其中的solrconfig.xml文件可以用来控制索引数据的可见性。如何获取solrconfig.xml，可参考。

soft commit

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

写入的数据多久后可以查询，默认是15秒。

hard commit

```
<autoCommit>
```

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

写入的数据多久后刷新到磁盘中，默认是30秒。

```
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

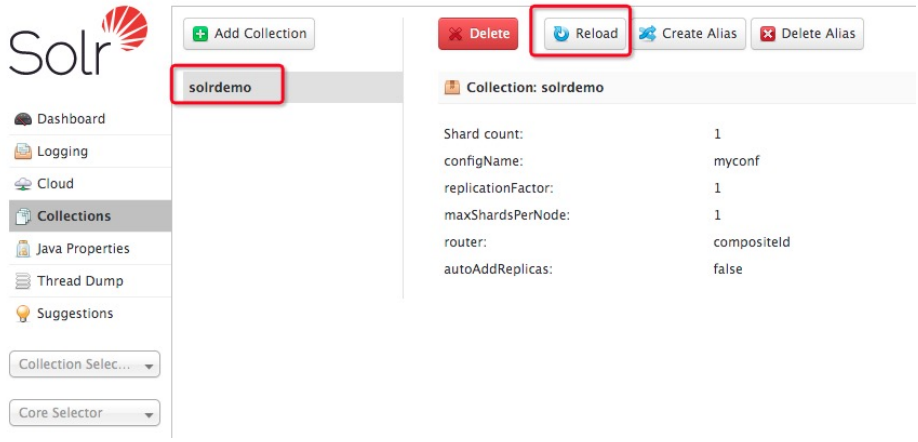
下载需要修改的配置集，参考

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，在单独写入Solr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 SolrServerException,
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

https://lucene.apache.org/solr/guide/7_3/updatehandlers-in-solrconfig.html#updatehandlers-in-solrconfig

数据预排序

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，如果索引数据在存储时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效果更明显。下面介绍如何在Solr中为每个Collection的数据开启预排序。

首先需要下载配置集，可参考。

设置预排序

1. 修改solrconfig.xml中的MergePolicy，详见链接。
2. 查询时，指定参数segmentTerminateEarly=true。

举个例子

1. 数据按照timestamp预排序（降序）

```
<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>
```

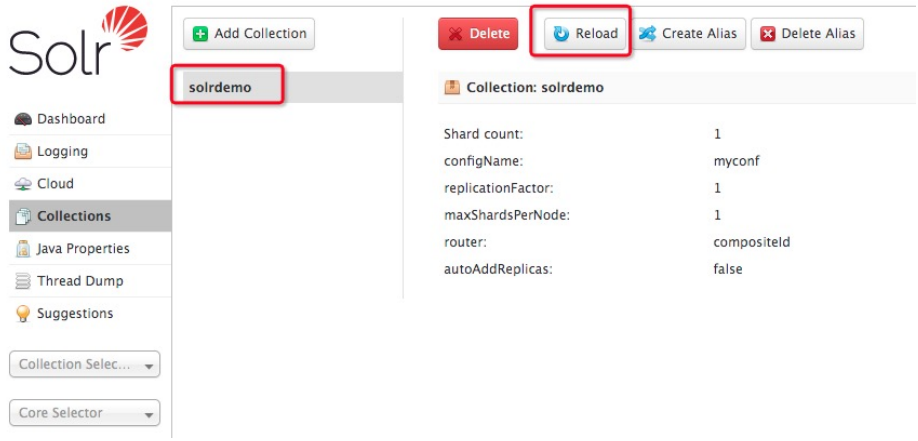
其中的< str name="sort">timestamp desc< /str>表示数据在写入时将会按照timestamp字段倒

序排序。

2. 更新配置集，参考。

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

3. Reload Collection



4. 执行如下查询：

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

注意

```
预排序返回的结果中，"numFound"数值是不准确的，但是整体的数据集是符合查询条件的。
多个排序字段，以逗号分隔：timestamp desc,age asc。
```

配置TTL

前言

对于大多数业务来说，数据可能只需要保留一段时间，过期的数据不再需要。从成本和性能考量上来说，能够清理掉旧数据是最好的，TTL可以完美解决这个场景，不需要业务逻辑主动delete数据，只需要设置合理的TTL即可，内核自动清理过期数据。下面介绍如何配置索引数据的TTL。

TTL原理介绍

写入的索引数据中必须包含有时间字段列，用来标识数据的起始时间，后台线程周期性的发起deleteByQuery:[* TO 当前时间-TTL值]。

```
<processor class="solr.processor.DocTTLUpdateProcessorFactory">
<int name="expirationFieldName">update_version_l</int>
<int name="ttlSeconds">2592000</int>
<int name="autoDeletePeriodSeconds">86400</int>
</processor>
```

expirationFieldName 写入的数据中必须携带这个时间字段，必须为long类型。示例中给出的是系统默认值，可以不需要更改，在从HBase同步到Solr时，我们自动会为该值设置系统时间。默认可不填。

ttlSeconds 代表索引数据的过期时间，单位为秒，默认为永久保存。示例中给出的是30天。

autoDeletePeriodSeconds 代表后台清理旧数据的周期，默认为-1。示例中给出的是1天执行一次，这个值不要配置的太短，建议为1天。

如何修改TTL

1. 下载配置集_indexer_default，参考。
2. 注释掉solrconfig.xml中default-chain的配置

```
<!-- 注释掉
<updateRequestProcessorChain name="default-chain" default="true">
<processor class="solr.LogUpdateProcessorFactory">
<int name="maxNumToLog">3</int>
<int name="slowUpdateThresholdMillis">5000</int>
</processor>
<processor class="solr.processor.DocBasedVersionConstraintsProcessorFactory">
<bool name="ignoreOldUpdates">true</bool>
<str name="versionField">update_version_l</str>
<str name="deleteVersionParam">del_version</str>
</processor>
<processor class="solr.DistributedUpdateProcessorFactory"/>
<processor class="solr.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>
-->
```

3. 打开ttl-chain的配置，根据业务需要修改TTL的配置值

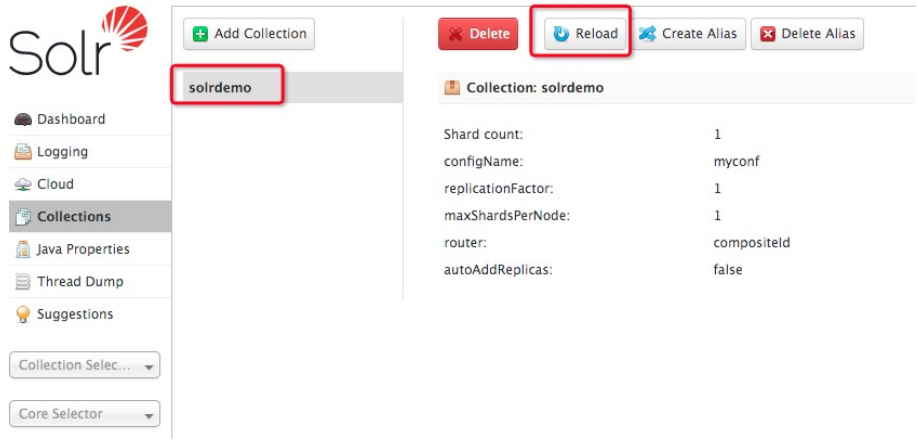
```
<updateRequestProcessorChain name="ttl-chain" default="true">
<processor class="solr.LogUpdateProcessorFactory">
<int name="maxNumToLog">3</int>
<int name="slowUpdateThresholdMillis">5000</int>
</processor>
<processor class="solr.processor.DocTTLUpdateProcessorFactory">
```

```
<int name="ttlSeconds">2147483647</int>
<int name="autoDeletePeriodSeconds">-1</int>
<bool name="autoSoftCommit">false</bool>
</processor>
<processor class="solr.processor.DocBasedVersionConstraintsProcessorFactory">
<bool name="ignoreOldUpdates">true</bool>
<str name="versionField">update_version_l</str>
<str name="deleteVersionParam">del_version</str>
</processor>
<processor class="solr.DistributedUpdateProcessorFactory"/>
<processor class="solr.RunUpdateProcessorFactory"/>
</updateRequestProcessorChain>
```

4. 更新配置集

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

5. Reload Collection



常见FAQ

常见FAQ

创建solr collection的shard个数、replica个数设置多少合适？

我们先列举一下几条规则，尽量满足如下规则即可：
1）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。
2）对于replicationFactor副本数，我们推荐默认设置为1，并且把autoAddReplicas属性设置为false。对于有特殊要求的，比如写入非常少，读非常多，且读可能会有大量热点，那可以考虑使用replicationFactor=2、

3这种，可以缓解读负载均衡。
3) 我们假设实例有n个solr server节点，每个节点放m个shard。我们得遵循如下公式：

```
这个collection未来的总数据量 > n X m X 21亿
```

对于一个collection而言，每个solr server节点放一个shard开始，即m=1，如果不满足，我们再m = 2、3...直到能满足“未来collection的总数据量 > n X m X 21亿” 这个公式为止即可。
如果你发现一个server要放的m的个数太大了，比如超过物理linux机器cpu的个数了，那就可能要考虑扩容节点了。推荐一个solr server节点尽量不要太多相同collection的shard
4) 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，如果比如一个shard保存了20亿，按照1k一个doc，总数据量达到2T左右, 这对一个server来说可能会有点大了，对应如果大量扫描估计扛不住，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，比如单个document就10k和200字节碰到这种情况都是不一样的。
注：对于solr的shard、replica分配，是可以后期再调整和split的，按照上述设置之后，后续有变化再调整也可以。另外，我们推荐用户后续如果需要调整shard、replica个数的，请联系“云hbase答疑”进行沟通，在solr在solr 7.3.1.4版本之前，请勿自行split shard。

hbase同步数据到solr索引的延时是多少？多少秒可见？

```
索引同步的延时时间 = 数据同步延迟 + solr commitWithin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)
对于solr 的commitWithin,默认是15000ms。再写入压力不大，没有积压的情况下，几乎主要决定索引的可见性即为 commitWithin时间，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，进而系统会频繁merge，也会侧面影响整体性能。

Solr如何使用预排序功能？

我们都知道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，那么solr的检索会有明显的提升，特别是在大数据量上对比的时候，此特点效果更明显。那么在solr层面是怎么支持的呢？下面我们简单描述下步骤：

- 修改solrconfig.xml中的MergePolicy, 详见链接
- 查询时，指定参数segmentTerminateEarly=true即可
  下面简单给个demo演示：

```
<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>
```

此时，我们主要关心“< str name=” sort” >timestamp desc< /str>”配置，此时插入数据将会按照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T级别之后，效果更明显。

需要注意的是:

- 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

关于solr的各种客户端连接使用说明

在云hbase 全文服务solr的使用中，我们一共提供了几个地址：zk内网地址、和zk公网地址、solr的webui公网地址、solr的CloudSolrClient连接地址。下面分别描述这几种地址的使用：

- solr CloudSolrClient内网链接地址

见上图，作为java api 中 CloudSolrClient的内网访问地址，此时CloudSolrClient的构造方法如下：

```
List<String> zkHostList = new ArrayList<>();
zkHostList.add("
hb-m5eXXXX-master1-001.hbase.rds.aliyuncs.com:2181");
zkHostList.add("
hb-m5eXXXX-master2-001.hbase.rds.aliyuncs.com:2181");
zkHostList.add("
hb-m5eXXXX-master3-001.hbase.rds.aliyuncs.com:2181");
CloudSolrClient cloudSolrClient = new Builder(zkHostList, Optional.of("/solr");
```

注意，替换“hb-m5eXXXX.....” 为你的对应zk 单个host地址。

- solr webui公网地址，见如下图

如上图，直接点击即可打开solr WebUI，其访问控制和hbase WebUI一样，详见文档链接
注意这个仅为公网浏览器打开的solr admin WebUI访问地址，不能用其浏览器显示的url进行solr数据访问。

- solr HttpSolrClient、curl 公网单点开发测试访问地址

如上图，当用户在“数据库链接”中，也打开 zk公网时，同样solr也具备了公网开发测试的单点访问功能，其开通的节点为 关键字带有“master3-1”的 host名字。如这里为的公网单点访问地址为：

```
http://hb-proxy-pub-m5eXXXX-master3-001.hbase.rds.aliyuncs.com:8983
```

此时，如果想公网访问solr进行开发测试，命令行运维是可以直接类似如下即可：

```
curl "http://hb-proxy-pub-m5eXXXX-master3-
001.hbase.rds.aliyuncs.com:8983/solr/admin/collections?action=list"
```

如果是想公网通过java api的HttpSolrClient单点公网开发测试访问solr，可以初始化实例如下：

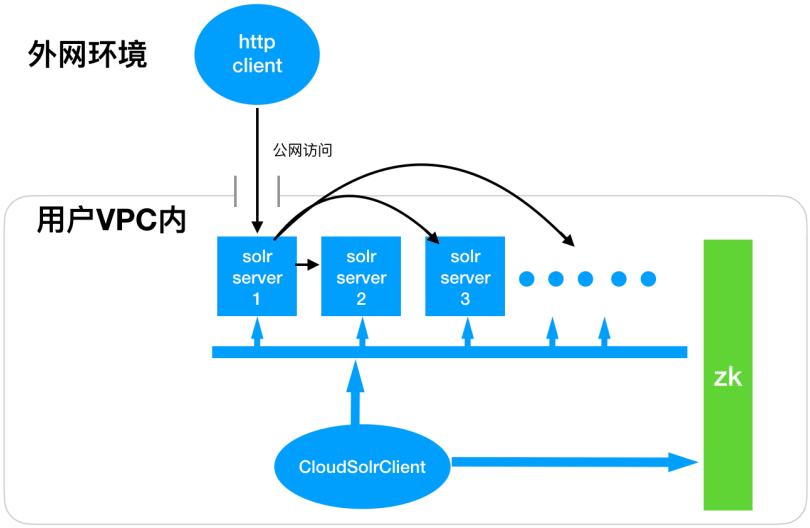
```
HttpSolrClient solrClient = new HttpSolrClient.Builder("http://hb-proxy-pub-m5eXXXX-master3-
001.hbase.rds.aliyuncs.com:8983/solr").build();
```

注意，solr的路由核心是在server端的，所以客户端访问任何一个节点都可以访问整个集群，当然如果您是内网访问solr集群，推荐使用CloudSolrClient api，它会有一些负责均衡相关的功能，这里的HttpSolrClient仅推荐用来做开发测试使用。

访问SolrServer

公网访问Solr

SolrCloud模式访问如同HBase一样，使用zookeeer进行进行获取可用的Solr Server进行访问。不同的是，Solr的访问不是在客户端路由，CloudSolrClient也只是封装load balance的http client循环访问不通的Solr Server进行负载均衡。真正发生路由是在一个Solr Server上，针对这点，就表明我们所有的访问都可以发送到一个Solr Server上。
另外，Solr使用http的方式访问，有大量的开源工具、代码库可以使用，当然也可以使用官网的各种语言的客户端。为了保持所有客户端都是和开源同步，为了公网访问Solr集群，我们只需要访问固定某个开好公网的Server进行访问即可，这样既可以满足开发测试阶段需求，也可以使用各种开源的http访问工具进行开发测试。



大致流程如下：

公网开放

Solr的公网开放和HBase的公网开放一样，当用户申请公网访问时，在“数据库链接”中显示如下：

申请完成后，可以看到，公网zk地址有3个，我们获取最后一个master3中缀的链接节点，即为我们可以进行公网访问的链接。如：

```
hb-proxy-pub-xxxxxxxxxxx-master3-001.hbase.singapore.rds.aliyuncs.com
```

进行公网访问

拿到了上面的master3的solr server公网地址后，可以使用这个链接进行访问solr。下面介绍两种访问方式时，设置的链接地址如何设置。

curl方式

node的hostname为上面拿到的公网master3地址，端口使用solr专用的8983

```
```

```
curl "http://hb-proxy-pub-xxxxxxxxxx-master3-
001.hbase.singapore.rds.aliyuncs.com:8983/solr/solrdemo/query?q=*:*"
```

注：如果上述curl无法访问，请确认是否白名单设置完成，详见hbase公网访问中的白名单设置
如果还不能访问，确认是否在开solr之前已经开过公网，如果是，可以尝试关闭公网再开启公网后验证一下。

SolrJ代码方式

代码构造SolrClient的时候，使用HttpSolrClient即可，如下：

```
HttpSolrClient solrClient = new HttpSolrClient.Builder("http://hb-proxy-pub-xxxxxxxxxx-master3-
001.hbase.singapore.rds.aliyuncs.com:8983/solr").build();
SolrQuery solrQuery = new SolrQuery("*:*");
QueryResponse response = solrClient.query("solrdemo", solrQuery);
//do something
```

注：开发测试使用如上方式进行公网访问，当上生产环境时，请使用CloudSolrClient 客户端进行构建应用，参考。