云数据库 HBase

HBase 标准版

HBase 标准版

产品简介

HBase标准版产品简介

云HBase标准版在开源HBase的基础上，做了大量的改进、稳定性优化和运维能力增强，吸收了众多阿里内部成功经验，比社区HBase版本具有更好的稳定性和可运维能力，同时100%兼容开源社区HBase。与开源产品的优势对比，可以参照这篇文章。目前云HBase标准版提供1.1版本和2.0版本供用户选择。

HBase1.1版本

1.1版本基于HBase社区1.1.2版本开发。

HBase2.0版本

2.0版本是基于社区2018年发布的HBase2.0.0版本开发的全新版本。同样，在此基础上，做了大量的改进和优化，吸收了众多阿里内部成功经验，比社区HBase版本具有更好的稳定性和性能。

访问准备

访问准备

第一步 准备ECS

目前HBase提供了内网访问及公网访问的模式。内网访问比较稳定，一般在生产使用；公网访问延迟相对较高

，比较适合开发测试使用，主要是满足开发测试的需求。 公网访问参考文档:公网访问方案。

经典网络

在经典网络环境中，您可以在HBase所在Region的任意Zone购买一台ECS，然后设置HBase的白名单即可访问。

例如：现在HBase创建在华东1-可用区b，我们可以在华东1-可用区b，可用区c，可用区d， 可用区e中任何一个可用区上创建ECS，然后连接HBase。

专有网络（VPC）

VPC场景下，用户可以创建任何可用区位置的ECS，只要ECS和HBase在同一个VPC下，即可正常通信。通过创建不同可用区的vswitch将ECS和HBase加入到同一个VPC后连接起来。

首先需要在这个Region中创建一个VPC，VPC控制台。

比如，我在杭州，那么在杭州创建一个VPC，叫myvpc。

在这个VPC下，对应HBase所在的可用区创建一个vswitch（虚拟交换机）。

比如，我们的HBase是在杭州的可用区e，那么我们在VPC中创建一个在可用区e的vswitch，叫switch-zone-e，对应的网段可以根据需要自行选择。

在HBase的购买页面，购买VPC设置为 myvpc和switch-zone-e。

创建ECS使用的vswitch。

a. 如果ECS可以购买和HBase相同的可用区，比如可用区e，那么可以直接到第五步购买。

b. 若不能购买HBase所在可用区的ECS，请查看下能够购买的ECS的可用区，比如是可用区b，然后在myvpc下创建对应的vswitch switch-zone-b。

创建ECS。

a. 若能够选择和HBase相同的可用区，如可用区e，那么直接购买，并在选择VPC的时候选择和HBase相同的VPC和vswitch。

b. 若不在同一个可用区，则按照上一步的选择的可用区，比如可用区b创建ECS，并在选择VPC的时候，选择之前创建的VPCmyvpc和switch-zone-b。

第二步 开启白名单

创建好ECS后，我们还需要在HBase上设置访问的白名单，将这台ECS的内网IP添加到访问白名单中，详细步骤

可以参考这里。

连通性检测

请参考 连通性检测

HBase客户端下载

Java SDK 安装请参考JAVA SDK安装

Java API的使用和连接参数请见下一篇文档HBase Java API 访问

使用Java Client 访问

使用JavaClient访问 HBase

访问准备

需要在相同的Region内准备一台ECS。如果已经有ECS了那么请继续下一步。如果还没有，您可以在ECS的购买页面上购买一台按量的ECS进行测试。设置请参考这里

也可以采取公网访问的方式：公网访问方案

使用 Client 读写 HBase

云HBase企业标准版可以直接使用社区开源版客户端，也可以使用阿里定制的HBase客户端访问（**注：如果是公网访问必须使用阿里提供的客户端**）。下载和依赖客户端的方式请见 客户端下载这篇文档

如果您的运行环境上并没有HBase的client运行需要的jar包，请在maven中添加如下的shade配置，把所有hbase-client的依赖都打到最终的jar包中。

```
  <build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-shade-plugin</artifactId>
<version>3.0.0</version>
```

```
<executions>
<execution>
<phase>package</phase>
<goals>
<goal>shade</goal>
</goals>
<configuration>
<artifactSet>
</artifactSet>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

获取集群的 ZK 连接地址

进入 HBase 控制台集群详情页面，在**网络信息**部分查看**ZK连接地址**，可以看到类似如下的ZooKeeper的连接地址。

```
 hb-bp1f5xxxx48a0r17i-001.hbase.rds.aliyuncs.com:2181
hb-bp1f5xxxx48a0r17i-002.hbase.rds.aliyuncs.com:2181
hb-bp1f5xxxx48a0r17i-003.hbase.rds.aliyuncs.com:2181
```

注意：如果您使用的是HBase增强版集群，请参照增强版的帮助文档HBase Java API 访问

配置 ZK 地址，连接集群

将集群的zk地址替换代码中的zk地址，就可以使用如下的示例代码来进行hbase集群的访问了。 例子中展示了创建表、写入数据、读取数据 三种场景。

```
 private static final String TABLE_NAME = "mytable";
private static final String CF_DEFAULT = "cf";
public static final byte[] QUALIFIER = "col1".getBytes();
private static final byte[] ROWKEY = "rowkey1".getBytes();

public static void main(String[] args) {
Configuration config = HBaseConfiguration.create();
String zkAddress = "hb-bp1f5xxxx48a0r17i-001.hbase.rds.aliyuncs.com:2181,hb-bp1f5xxxx48a0r17i-
002.hbase.rds.aliyuncs.com:2181,hb-bp1f5xxxx48a0r17i-003.hbase.rds.aliyuncs.com:2181";
config.set(HConstants.ZOOKEEPER_QUORUM, zkAddress);
Connection connection = null;

try {
connection = ConnectionFactory.createConnection(config);

HTableDescriptor tableDescriptor = new HTableDescriptor(TableName.valueOf(TABLE_NAME));
tableDescriptor.addFamily(new HColumnDescriptor(CF_DEFAULT));
System.out.print("Creating table. ");
Admin admin = connection.getAdmin();
```

```
admin.createTable(tableDescriptor);
System.out.println(" Done.");
Table table = connection.getTable(TableName.valueOf(TABLE_NAME));
try {
Put put = new Put(ROWKEY);
put.addColumn(CF_DEFAULT.getBytes(), QUALIFIER, "this is value".getBytes());
table.put(put);
Get get = new Get(ROWKEY);
Result r = table.get(get);
byte[] b = r.getValue(CF_DEFAULT.getBytes(), QUALIFIER); // returns current version of value
System.out.println(new String(b));
} finally {
if (table != null) table.close();
}
} catch (Exception e) {
e.printStackTrace();
} finally {
if (connection != null) {
try {
connection.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
}
```

代码样例

您也可以在这里下载我们提供的Java代码工程，**替换其中的ZooKeeper变量部分后使用。**

HBase访问Demo下载

公网访问方案

HBase提供混合访问的方案

公网访问

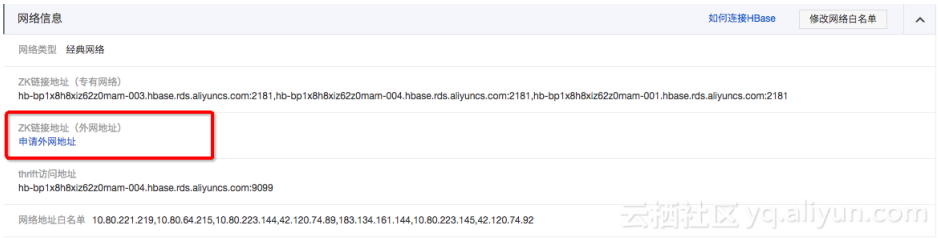
- 提供开发测试的便利，一般测试开发环境在线下，比如个人的电脑
- 客户hbase上云，从线下把数据同步上云

当前公网访问流量免费，内网访问流量免费

特别说明：公网访问平台不保障QPS及延迟

如果在想在公网访问阿里云的HBase需要有以下步骤：

1、申请公网访问域名



zk：域名用包含 proxy-pub的域名

2、添加公网访问白名单

经典访问VPC环境

- 主要是为了迁移的需求

需要提工单找产品同学打通，目前界面没有直接提供

以上功能需要使用阿里云定制的客户端：

相关hbase的maven及tar包：阿里云开源HBase客户端

VPN方案

在VPC环境下，可以与公司环境拉一根专线，打通阿里云跟内部环境

使用 Shell 访问

使用 HBase Shell 访问

准备访问用ECS

因为云HBase只提供了内网的访问，所以如果要访问HBase，需要在相同的Region内准备一台ECS。如果已经有ECS了那么请继续下一步。如果还没有，您可以在ECS的购买页面上购买一台按量的ECS进行测试。

设置请参考这里

下载HBase Shell

下载地址：云HBase控制台 -> 数据库连接 -> 连接信息 -> hbase shell下载。

下载完成以后，解压缩

```
tar -zxf xxx.tar.gz
```

注意：如果您使用的是HBase增强版集群，请参照增强版的Shell帮助文档

配置 ZK 地址 解压后，修改 conf/hbase-site.xml 文件，添加集群的 ZK 地址，如下所示：

```
<configuration>
<property>
<name>hbase.zookeeper.quorum</name>
<value>$ZK_IP1,$ZK_IP2,$ZK_IP3</value>
</property>
</configuration>
```

其中的$ZK_IP1,$ZK_IP2,$ZK_IP3请参考上一节中的**获取集群的 ZK 连接地址**

访问集群

通过如下命令就可以访问集群了。

```
bin/hbase shell
```

视频资料

以上操作如果还有疑惑，可以点击这里查看视频说明。

相关资源

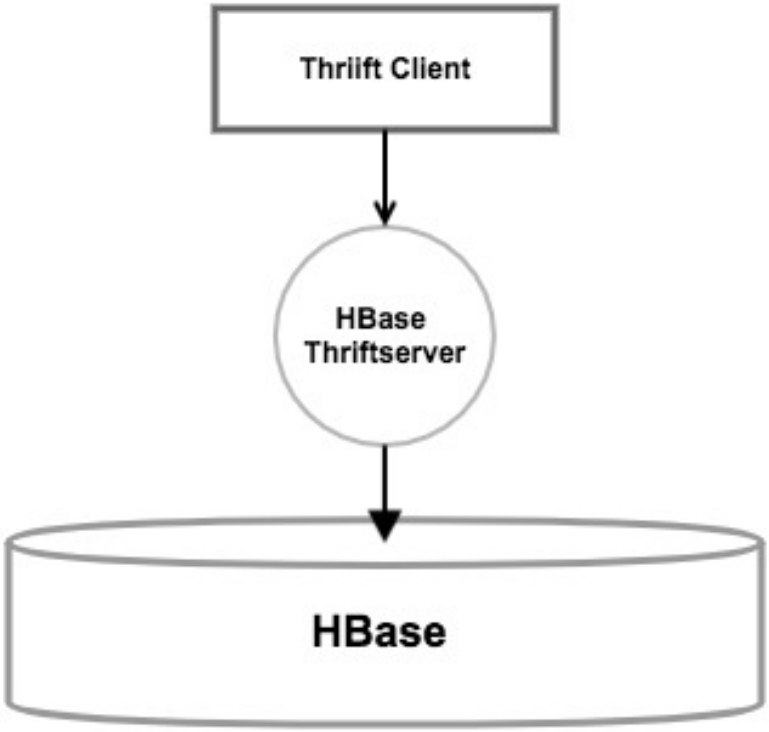
- HBase shell 命令列表
- HBase shell 命令练习

更多开发介绍，请参考 Apache HBase 官网。

多语言支持(thrift)

1、 产品支持

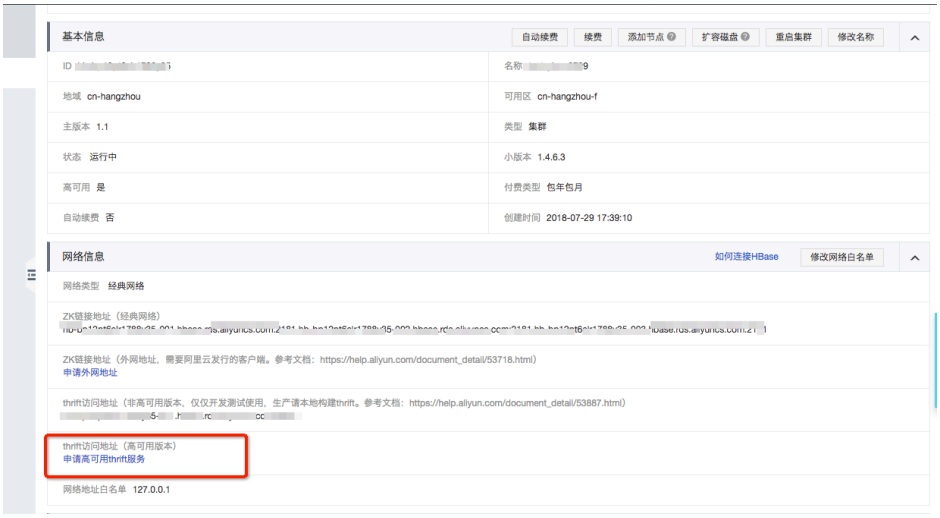
Thrift 提供多语言访问HBase的能力，支持的语言包从Thrift官网看括: C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml , Delphi 以及别的语言。主要流程是用户thrift Client 通过Thrift协议访问HBase的thriftserver,thriftserver请求转发给HBase的存储服务.大概架构图如下：



注意：如果您使用的是HBase增强版集群，请参照增强版的多语言访问帮助文档

开通HBase thriftserver服务：

管控页面点击 开通thriftserver服务化（高可用版本thriftserver），会得到一个 host:port的访问入口;



2、使用说明

2.1 用户Thrift client访问：

一般客户常见的访问方式是python及php ，这里给出php的访问方式；

2.1.1 以php走thrift访问HBase：

我们云HBase的thrift环境是0.9.0，所以建议客户客户端的版本也为 0.9.0，可以从这里下载thrift的0.9.0 版本，下载的源码包我们后面会用到，这里需要先安装thrift编译环境，对于源码安装可以参考thrift官网；

通过如下命令可以看出安装thrift的版本信息；

```
thrift --version
```

2.1.2. 生成thrift访问client的访问文件；

我们从这里下载云HBase的Hbase.thrift文件，云HBase使用的是thrift1协议

编译命令如下：

```
thrift --gen <language> Hbase.thrift
```

上述 language 是语言的缩写，常见的有如下：

```
thrift --gen php Hbase.thrift
thrift --gen cpp Hbase.thrift
thrift --gen py Hbase.thrift
```

执行thrift —gen php Hbase.thrift ，在目录下得到gen-php ，这个是需要的函数包文件；

```
thrift git:(last_dev)  ll
total 56
-rw-r--r-- 1 xuanling.gc staff 24K 3 5 15:06 Hbase.thrift
drwxr-xr-x 3 xuanling.gc staff 96B 8 1 16:03 gen-php
```

将下载到的Thrift源码文件夹下的/lib/php/lib下面的Thrift文件夹以及gen-php一起放在我们的业务逻辑代码一个src目录下面，加上我们自己的client.php的代码，目录结果如下所示：

```
[root@xxxxxxxxxxx thrift_client]# ll
total 12
-rw-r--r-- 1 zookeeper games 2743 Aug 2 11:16 client.php
drwxr-xr-x 3 zookeeper games 4096 Aug 2 01:22 gen-php
drwxr-xr-x 12 zookeeper games 4096 Aug 2 01:22 Thrift
```

php访问代码编写；

上述的Thrift文件夹以及gen-php文件夹，可以随自己项目以及个人风格命名，这里方便大家搞清目录结构，就保留原来风格；下面贴出php的代码，下面的程序是在HBase 建了一张表”new”：

```
<?php
ini_set('display_errors', E_ALL);
$GLOBALS['THRIFT_ROOT'] = "/root/thrift_client";
/* Dependencies. In the proper order. */
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/Transport/TTransport.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/Transport/TSocket.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/Protocol/TProtocol.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/Protocol/TBinaryProtocol.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/Protocol/TBinaryProtocolAccelerated.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/Transport/TBufferedTransport.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/Type/TMessageType.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/Factory/TStringFuncFactory.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/StringFunc/TStringFunc.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/StringFunc/Core.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/Type/TType.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/Exception/TException.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/Exception/TTransportException.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/Thrift/Exception/TProtocolException.php';

require_once $GLOBALS['THRIFT_ROOT'] . '/gen-php/Hbase/Types.php';
require_once $GLOBALS['THRIFT_ROOT'] . '/gen-php/Hbase/Hbase.php';

use Thrift\Protocol\TBinaryProtocol;
use Thrift\Transport\TBufferedTransport;
use Thrift\Transport\TSocket;
use Hbase\HbaseClient;
use Hbase\ColumnDescriptor;
use Hbase\Mutation;

$host='hb-bp12pt6alr1788y35-001.hbase.rds.aliyuncs.com';
$port=9099;

$socket = new TSocket($host, $port);
```

```
$socket->setSendTimeout(10000); // 发送超时，单位毫秒
$socket->setRecvTimeout(20000); // 接收超时，单位毫秒
$transport = new TBufferedTransport($socket);
$protocol = new TBinaryProtocol($transport);
$client = new HbaseClient($protocol);

$transport->open();

####列出表####
echo "----list tables----\n";
$tables = $client->getTableNames();
foreach ($tables as $name) {
var_dump($tables);
}

$tablename='new';
####写数据####
echo "----write data----\n";
$row = 'key';
$value = 'value';
$atrribute = array();
$mutations = array(
new Mutation(array(
'column' => 'info:cn1',
'value' => $value
)),
);

try {
$client->mutateRow($tablename, $row, $mutations, $atrribute);
} catch (Exception $e) {
var_dump($e);//这里自己打log
}

###读数据####
echo "---read data---\n";
$result = $client->getRow($tablename, $row, $atrribute);
var_dump($result);

###删数据####
echo "---delete data---\n";
$client->deleteAllRow($tablename, $row, $atrribute);
echo "---get data---\n";
$result = $client->getRow($tablename, $row, $atrribute);
var_dump($result);

###扫描数据###
$row = 'ID1';
$value = 'v1';
$mutations = array(
new Mutation(array(
'column' => 'info:c1',
'value' => $value
)),
);
```

```
try {
$client->mutateRow($tablename, $row, $mutations, $atrribute);
} catch (Exception $e) {
var_dump($e);
}

$row = 'ID2';
$value = 'v2';
$mutations = array(
new Mutation(array(
'column' => 'info:c1',
'value' => $value
)),
);
try {
$client->mutateRow($tablename, $row, $mutations, $atrribute);
} catch (Exception $e) {
var_dump($e);
}

$row = 'ID3';
$value = 'v3';
$mutations = array(
new Mutation(array(
'column' => 'info:c1',
'value' => $value
)),
);
try {
$client->mutateRow($tablename, $row, $mutations, $atrribute);
} catch (Exception $e) {
var_dump($e);
}

echo 'prefix scan';
$scan = $client->scannerOpenWithPrefix($tablename, 'ID', null, null);

$nbRows = 100;
$arr = $client->scannerGetList($scan, $nbRows);
echo 'count of result :'.count($arr)."\n";
var_dump($arr);
foreach ($arr as $k => $TRowResult) {
echo "\trow:$TRowResult->row\tcolumns(array):";
foreach ($TRowResult->columns as $key => $value) {
echo "key:$key\tvalue:$value->value\ttimestamp:$value->timestamp\n";
}
}

echo 'range scan';
$scan = $client->scannerOpenWithStop($tablename, 'ID0', 'ID2', null, null);

$nbRows = 100;
$arr = $client->scannerGetList($scan, $nbRows);
echo 'count of result :'.count($arr)."\n";
var_dump($arr);
foreach ($arr as $k => $TRowResult) {
```

```
echo "\trow:$TRowResult->row\tcolumns(array):";
foreach ($TRowResult->columns as $key => $value) {
echo "key:$key\tvalue:$value->value\ttimestamp:$value->timestamp\n";
}
}

###Increment 操作###
echo "do increment on a new row";
$row = 'ID4';
try {
$newCount = $client->atomicIncrement($tablename, $row, 'info:c1', 1234);
} catch (Exception $e) {
var_dump($e);
}

echo "new count $newCount\n";

###读写 long型数据到hbase###

$row = 'ID5';
$value = pack("J", 4567);

$mutations = array(
new Mutation(array(
'column' => 'info:c1',
'value' => $value
)),
);
try {
$client->mutateRow($tablename, $row, $mutations, null);
} catch (Exception $e) {
var_dump($e);
}

echo "---read data and print it as long ---\n";
$result = $client->getRow($tablename, $row, null);

foreach ($result[0]->columns as $key => $value) {
$count = unpack("J*mycount", $value->value);
var_dump($count);
}

?>
```

代码执行结果如下：

```
[root@xxxxxxxxxxx thrift_client]# php client.php
----list tables----
array(1) {
[0]=>
string(3) "new"
}
----write data----
---read data---
array(1) {
```

```
[0]=>
object(Hbase\TRowResult)#8 (3) {
["row"]=>
string(3) "key"
["columns"]=>
array(1) {
["info:cn1"]=>
object(Hbase\TCell)#10 (2) {
["value"]=>
string(5) "value"
["timestamp"]=>
int(1533179795969)
}
}
["sortedColumns"]=>
NULL
}
}
---delete data---
---get data---
array(0) {
}
```

2.2.python访问流程；

此外还有常见的python的客户，对于python的话，有happybase这种python的第三方包含thrift的库，我们见过一些客户使用Happybase进行访问HBase thrift，参见文章；此外，python 有丰富的库，我们通过pip可以安装thrift，以及访问HBase的thrift库；执行流程如下,假设用户已经安装python以及pip：

```
pip install thrift //安装thrift默认最新版本
pip install hbase-thrift //安装hbase thrift接口库
```

上面2步执行完成以后，既可以编写访问HBase的代码：

```
import sys
import time
import os

from thrift import Thrift
from thrift.transport import TSocket, TTransport
from thrift.protocol import TBinaryProtocol
from thrift.protocol.TBinaryProtocol import TBinaryProtocolAccelerated
from hbase import ttypes
from hbase.Hbase import Client, ColumnDescriptor, Mutation

def printRow(entry):
print "row: " + entry.row + ", cols:",
for k in sorted(entry.columns):
print k + " => " + entry.columns[k].value,
print


transport = TSocket.TSocket('hb-bp12pt6alr1788y35-001.hbase.rds.aliyuncs.com', 9099)
```

```
transport = TTransport.TBufferedTransport(transport)
protocol = TBinaryProtocol.TBinaryProtocolAccelerated(transport)
client = Client(protocol)
transport.open()

print "---list table--"
print client.getTableNames()

table="new"
row="key"

print "---write data---"
mutations = [Mutation(column="info:cn1", value="value")]
client.mutateRow(table, row, mutations)

print "---get data----"
printRow(client.getRow(table, row)[0])

print "---delete data---"
client.deleteAllRow(table, row)
print "---end----"

transport.close()
```

对应上述的程序执行的结果如下：

```
[root@Test ~]# python Hbase_client.py
---list table--
['new']
---write data---
---get data----
row: key, cols: info:cn1 => value
---delete data---
---end----
```

Go语言访问HBase

下载优化后的thrift访问压缩包（也可以从https://github.com/sdming/goh 下载原始版本），解压后放到$GOPATH/src下

```
wget http://public-hbase.oss-cn-hangzhou.aliyuncs.com/thrift/goh.tar.gz
tar -xvzf goh.tar.gz
mv github.com $GOPATH/src
```

示例代码请参考 $GOPATH/src/github.com/sdming/goh/demo/client.go 包含了DDL以及数据读写的代码示例

访问HBase HDFS

在一些场景下，比如需要bulkload导入数据，需要打开HBase集群的HDFS端口。

- 注意：hdfs端口打开后，因误操作hdfs导致的数据丢失等问题客户自身承担，客户需要对hdfs的操作比较了解

开通HDFS端口

- 首先联系云HBase答疑(钉钉号)，开通HDFS（ 由于hdfs的开放可能造成用户的恶意攻击，引起集群不稳定甚至造成破坏。因此此功能暂时不直接开放给用户，当用户特别需要的情况下，我们通过云HBase答疑后台开通，随后客户使用完成，再关闭）

验证

- 检查端口是否可以正常使用通过一个hdfs client访问云hbase上的hdfs（目标集群）
- 创建一个hadoop客户端配置目录conf(如果使用客户端已存在这个目录则不需要另行创建)

添加以下两个hdfs配置到hadoop客户端conf目录({hbase-header-1-host}和{hbase-header-1-host}可以咨询 云HBase答疑

- core-site.xml

```
<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://hbase-cluster</value>
</property>
</configuration>
```

- hdfs-site.xml

```
<configuration>
<property>
<name>dfs.nameservices</name>
<value>hbase-cluster</value>
</property>
<property>
<name>dfs.client.failover.proxy.provider.hbase-cluster</name>
<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>
</property>
<property>
<name>dfs.ha.automatic-failover.enabled.hbase-cluster</name>
<value>true</value>
</property>
<property>
```

```
<name>dfs.namenode.http-address.hbase-cluster.nn1</name>
<value>{hbase-header-1-host}:50070</value>
</property>
<property>
<name>dfs.namenode.http-address.hbase-cluster.nn2</name>
<value>{hbase-header-2-host}:50070</value>
</property>
<property>
<name>dfs.ha.namenodes.hbase-cluster</name>
<value>nn1,nn2</value>
</property>
<property>
<name>dfs.namenode.rpc-address.hbase-cluster.nn1</name>
<value>{hbase-header-1-host}:8020</value>
</property>
<property>
<name>dfs.namenode.rpc-address.hbase-cluster.nn2</name>
<value>{hbase-header-2-host}:8020</value>
</property>
</configuration>
```

添加conf到hadoop 客户端classpath中

- 读写验证hdfs端口能否正常访问

```
echo "hdfs port test"  >/tmp/test
hadoop dfs -put /tmp/test /
hadoop dfs -cat /test
```

Hive 读写 HBase 指南

云 HBase 支持使用 Hive 读写里面的数据，配置起来也很简单。

环境准备

- 将 Hive 所在的 Hadoop 集群所有的节点的IP加入到云HBase白名单；
- 获取云HBase的zookeeper访问地址，可在云HBase控制台查看。

修改配置

- 进入hive配置目录 /etc/ecm/hive-conf/
- 修改 hbase-site.xml，将 hbase.zookeeper.quorum 修改为云HBase的zookeeper访问连接，如下

：

```
<property>
<name>hbase.zookeeper.quorum</name>
<value>hb-xxx-001.hbase.rds.aliyuncs.com,hb-xxx-002.hbase.rds.aliyuncs.com,hb-xxx-
003.hbase.rds.aliyuncs.com</value>
</property>
```

在Hive中读写HBase表

如果HBase表不存在，可在Hive中直接创建云HBase关联表

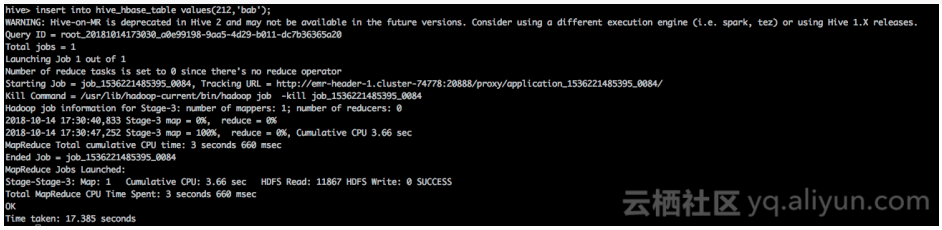
- 进入hive cli命令行

- 创建HBase表

```
CREATE TABLE hive_hbase_table(key int, value string)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val")
TBLPROPERTIES ("hbase.table.name" = "hive_hbase_table", "hbase.mapred.output.outputtable" =
"hive_hbase_table");
```

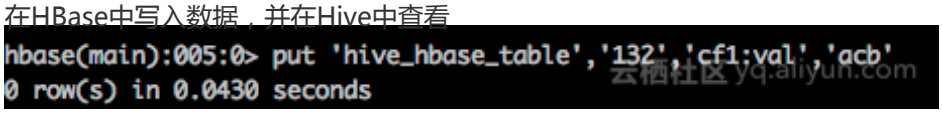
- Hive中向hbase插入数据

```
insert into hive_hbase_table values(212,'bab');
```

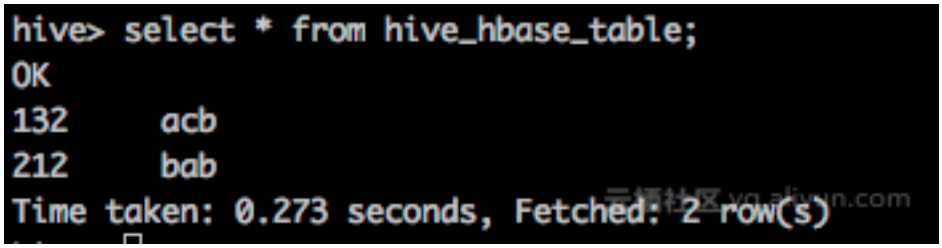


- 查看云HBase表,hbase表已创建，数据也已写入

在HBase中写入数据，并在Hive中查看

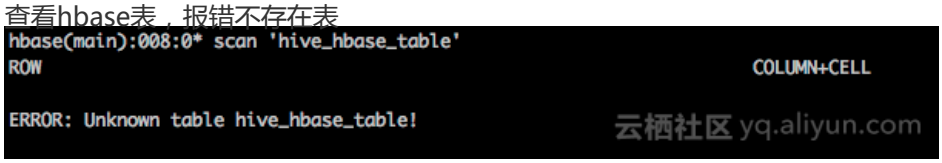


在Hive中查看：



Hive删除表，HBase表也删除

查看hbase表，报错不存在表



如果HBase表已存在，可在Hive中HBase外表进行关联，外部表在删除时不影响HBase已创建表

- 云hbase中创建hbase表，并put测试数据

```
hbase(main):020:0* create 'hbase_table','f'
0 row(s) in 1.3010 seconds

=> Hbase::Table - hbase_table
hbase(main):021:0> put 'hbase_table','1122','f:col1','hello'
0 row(s) in 0.0190 seconds

hbase(main):022:0> put 'hbase_table','1122','f:col2','hbase'
0 row(s) in 0.0110 seconds
```


- Hive中创建HBase外部关联表，并查看数据

```
hive> create external table hbase_table(key int, col1 string, col2 string)
    > STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
    > WITH SERDEPROPERTIES ("hbase.columns.mapping" = "f:col1,f:col2")
    > TBLPROPERTIES("hbase.table.name" = "hbase_table", "hbase.mapred.output.outputtable" = "hbase_table");
OK
Time taken: 0.129 seconds
hive> select * from hbase_table;
OK
1122    hello   hbase
Time taken: 0.181 seconds, Fetched: 1 row(s)
hive>
```


删除Hive表不影响HBase已存在表

```
hive> drop table hbase_table;
OK
Time taken: 0.102 seconds
hive>
```



```
hbase(main):023:0> scan 'hbase_table'
ROW                                             COLUMN+CELL
 1122                                           column=f:col1, timestamp=1539510170256, value=hello
 1122                                           column=f:col2, timestamp=1539510181752, value=hbase
1 row(s) in 0.0160 seconds
```


Hive更多操作HBase步骤，可参考https://cwiki.apache.org/confluence/display/Hive/HBaseIntegration

如果使用ECS自建mr集群的 Hive时，操作步骤跟EMR操作类似，需要注意的是自建Hive的hbase-site.xml部分配置项可能与云HBase不一致，简单来说网络和端口开放后，只保留hbase.zookeeper.quorum即可与云Hbase进行关联。

修改配置

修改完成后，需要**重启集群** ，采取滚动重启的方式，对集群影响较小，且规模越大，影响较小

修改完后，有历史记录

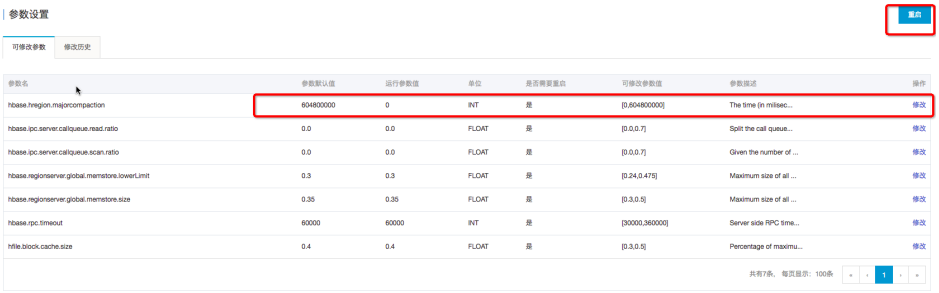
后续优化：后续有一些参数可以采取直接应用的方式，无需重启集群

目前开放的一些配置（仅仅列举一些，具体查看管控页面）

参数名称	参数默认值	描述
hbase.hregion.majorcompaction	604800000	默认为一周，可以改为0，不触发，后续产品支持周期性触发

		，或者时间段内触发major
hbase.ipc.server.callqueue.read.ratio	0	可以调整read与scan的队列大小，控制读写分离
hbase.ipc.server.callqueue.scan.ratio	0	读写分离控制
hbase.regionserver.global.memstore.lowerLimit	0.3	memstore的占比
hbase.regionserver.global.memstore.size	0.35	memstore的占比
hbase.rpc.timeout	60000	rpc超时时间，可以延长
hfile.block.cache.size	0.4	如果读比较多，可以调整威0.5

见：

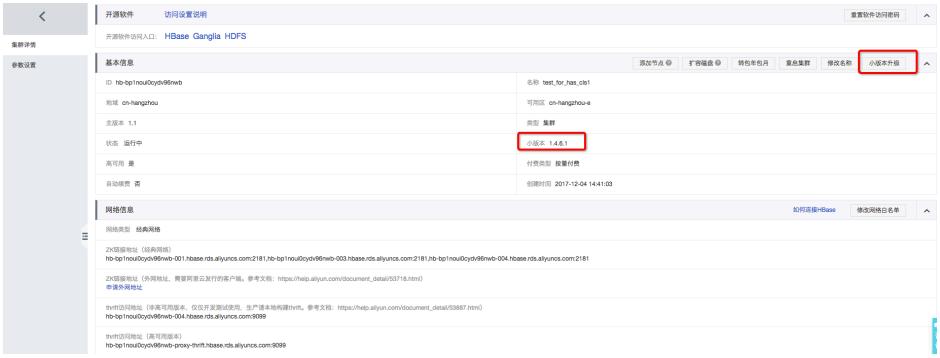


小版本升级

关于升级

阿里云云HBase组，在不断fix线上bug及改进性能，小版本升级是保证完全兼容的。一些严重的bug，我们会邮件通知，请关注我们的邮件。为了不影响客户的业务，我们不会主动升级客户的集群，我们建议客户在业务低峰期自主升级小版本。

我们当前不支持从1.x系列自动升级到2.x系列。 我们也会一直维护1.x及2.x的稳定性，在1.x上也会打上一些性能优化的patch。2.x主要有一些新的功能，如果客户需要，建议购买2.x集群



升级流程比较简单

- 选择在业务低峰期，点击升级按钮 小版本升级
- 升级时间跟随着集群的大小，region的个数多少相关，越大越多时间越长，一般4台的集群，100个region会在10分钟以内
- 后台会自动升级集群到最新对应的小版本，升级过程会滚动重启集群，基本对业务无感知
- 升级完成后，关注是否对业务有影响，如果有影响，迅速 通过工单 或者 云HBase答疑 钉钉号，跟我们联系

版本更新说明

1.x系列

版本1.4.6.2 是社区版本1.1 发展过来，做了大量的性能、稳定性、功能优化

1.4.6.2

- 解决WAL可能由于HDFS BUG导致损坏的问题
- 解决写入HDFS瞬间异常的情况下，可能导致整个RegionServer卡死，无法恢复。
- 解决Region分裂时候，客户端无法正常收到NotServingRegionException异常，会看到 “IllegalStateException: No result & no exception”
- 解决读写分离配置可能会导致RegionServer无法启动
- 解决社区Phoenix可能会把Zookeeper连接打满的BUG
- 优化配置，减少RegionServer被写出OOM的情况出现，减少大范围scan对服务端的内存消耗。
- 内核一些优化

1.4.6.3

PHOENIX

- 修复IndexTool的数据表为小写表名字时找不到索引表
- 修复客户端配置时区失效问题（当前默认时区GTM）
- 优化默认配置参数，更容易适配云上的小规格实例。
- 调整默认最大mutation size和bytes， 防止因为cell较大时导致创建索引失败。
- 通过raw scan替换major compaction索引表被disable方案，减小索引被dsiable风险。
- 写索引失败时，通过无限重试的强同步方案，规避索引表DISABLE问题，防止查询退化为扫全表问题

。
- upsert-select和create index时disable scan bolck cache, 减小不必要的内存使用和GC次数。

HBASE

- 删除列族后replication移除stale KV报NPE
- MiniHBaseCluster 支持可配置端口

1.4.9.1

- 同一集群支持冷存储 和 热存储
- 支持ZSTD，LZO压缩格式
- 修复MVCC卡死的BUG

1.5.0

- 支持Replication，支持主备集群同步
- 修复了一个内存泄露问题，已经回馈给社区HBASE-21228
- 修复了Reader线程在遇到OOM后退出而不abort RS的问题，已经回馈给社区HBASE-21357

1.5.1

- 修复ZSTD压缩算法引起的不兼容问题
- 修复冷存储的一些性能问题
- 修复Phoenix二级索引的一些问题
- 修复一个内存泄露问题，已经回馈给社区：HBASE-22169
- 其他一些bugfix

1.5.2

- 新增支持磁盘剩余空间过少自动锁定。
- 建表时不在支持PREFIXTREE Encoding。
- Backport社区Bug fix

1.5.3

- 新增日志订阅动态开关。
- 默认禁止hbck。
- 添加Thrift Server活跃链接数监控。
- 其他一些bugfix

2.x系列

2.0.2

- 发布商业化版本
- 阿里1.x版本优化同步到2.x版本，
- 修复大量的bug，已经贡献给社区，参考：

- HBASE-21237,Use CompatRemoteProcedureResolver to dispatch open/close region requests to RS
- HBASE-21228,Memory leak since AbstractFSWAL caches Thread object and never clean later
- HBASE-21212,Wrong flush time when update flush metric
- HBASE-21113,” Apply the branch-2 version of HBASE-21095, The timeout retry logic for several procedures are broken after master restarts”
- HBASE-21085,Adding getter methods to some private fields in ProcedureV2 module
- HBASE-21083,Introduce a mechanism to bypass the execution of a stuck procedure
- HBASE-21051,Possible NPE if ModifyTable and region split happen at the same time
- HBASE-21050,Exclusive lock may be held by a SUCCESS state procedure forever
- HBASE-21041,Memstore’ s heap size will be decreased to minus zero after flush
- HBASE-21039,Procedure worker should not quit when getting unexpected error
- HBASE-21035,Meta Table should be able to online even if all procedures are lost
- HBASE-21031,Memory leak if replay edits failed during region opening
- HBASE-21029,Miscount of memstore’ s heap/offheap size if same cell was put
- HBASE-21003,Fix the flaky TestSplitOrMergeStatus
- HBASE-20990,One operation in procedure batch throws an exception will cause all RegionTransitionProcedures receive the same exception
- HBASE-20978,[amv2] Worker terminating UNNATURALLY during MoveRegionProcedure
- HBASE-20976,SCP can be scheduled multiple times for the same RS
- HBASE-20975,Lock may not be taken or released while rolling back procedure
- HBASE-20973,ArrayIndexOutOfBoundsException when rolling back procedure
- HBASE-20921,Possible NPE in ReopenTableRegionsProcedure
- HBASE-20903,” backport HBASE-20792 “” info:servername and info:sn inconsistent for OPEN region” ” to branch-2.0”
- HBASE-20893,Data loss if splitting region while ServerCrashProcedure executing
- HBASE-20878,Data loss if merging regions while ServerCrashProcedure executing
- HBASE-20870,Wrong HBase root dir in ITBLL’ s Search Tool
- HBASE-20867,RS may get killed while master restarts
- HBASE-20864,RS was killed due to master thought the region should be on a already dead server
- HBASE-20860,Merged region’ s RIT state may not be cleaned after master restart
- HBASE-20854,Wrong retries number in RpcRetryingCaller’ s log message
- HBASE-20846,Restore procedure locks when master restarts
- HBASE-20727,Persist FlushedSequenceId to speed up WAL split after cluster restart
- HBASE-20679,Add the ability to compile JSP dynamically in Jetty
- HBASE-20611,UnsupportedOperationException may thrown when calling getCallQueueInfo()
- HBASE-20601,Add multiPut support and other miscellaneous to PE

2.0.3

- 修复若干AssignmentManger v2的稳定性问题

2.0.4

- 同一集群支持冷存储 和 热存储
- 修复若干稳定性问题

2.0.5

- 修复与Phoenix5.x的兼容性问题
- 修复冷存储的一些性能问题

2.0.6

- 新增支持ZSTD压缩算法
- 修复若干稳定性问题，均已回馈给社区，如：HBASE-22128 HBASE-21751 HBASE-22169
- 其他一些bugfix

2.0.7

- 新增支持磁盘剩余空间过少自动锁定。
- 解决开启phoenix wal无法清理的bug。
- 支持RS日志订阅消费

2.0.8

- 新增日志订阅动态开关。
- 默认禁止hbck。
- 其他一些bugfix

2.0.9

- 修复DFSClient内存泄漏问题。
- 修复SplitTable阻塞ModifyTable问题。
- 修复开启ASYNC_WAL造成WAL损坏问题.

HBase冷存储(即将下线)

冷存储功能目前已经全面升级，使用老架构的标准版冷存储功能即将下线，新用户请移步增强版的冷存储功能.

HBase增强版内置一体化冷热分离功能，在一张表内全透明实现数据的冷热分离，无需区分“热表”，“冷表”，业务0改造便可获得极致成本，如果有冷热分离需求，请移步HBase增强版的帮助文档冷存储和冷热分离章节

介绍

冷数据是大数据存储当中常见的场景。阿里云HBase针对冷数据存储的场景，提供一种新的冷存储介质，其存储成本仅为高效云盘的1/3，写入性能与云盘相当，并能保证数据随时可读。冷存储适用于数据归档、访问频率较低的历史数据等各种冷数据场景。冷存储的使用非常简单，用户可以在购买云HBase实例时选择冷存储作为一个附加的存储空间，并通过建表语句指定将冷数据存放在冷存储介质上面，从而降低存储成本。

开通冷存储

冷存储可以独立购买，作为一个附加存储空间使用。购买冷存储介质后，可以在建表时候中指定把表创建在冷存储上（即冷表），默认是创建在云盘介质上（即热表）。

创建新的HBase实例时，可在购买页面选择是否选购冷存储和冷存储的容量。

也可以对一个已有集群增加冷存储介质。方法是在实例的控制台点击“冷存储设置”-“立即开通”，选择冷存储容量后，即可开通冷存储。

使用冷存储

开通冷存储之后，可以在创建表时通过HFILE_STORAGE_POLICY属性指定要把是表创建在普通存储介质上（热表）还是创建在冷存储上（冷表），不指定默认为热表。如下语句表明test被创建为一个冷表，之后test表的数据将被存储在冷介质上：

```
create 'test',"info",CONFIGURATION => {'HFILE_STORAGE_POLICY'=>'COLD'}
```

冷表创建出来之后，其使用和数据写入与普通的表是一样的，通过HBase API操作即可。例如，我们可以通过HBase Shell写入和查询数据：

```
hbase(main):017:0> put 'test','row1','info:a','a'
Took 0.0154 seconds
hbase(main):018:0> put 'test','row2','info:a','b'
Took 0.0048 seconds
hbase(main):019:0> flush 'test'
Took 2.2975 seconds
hbase(main):020:0> scan 'test'
ROW COLUMN+CELL
row1 column=info:a, timestamp=1539573371860, value=a
row2 column=info:a, timestamp=1539573377499, value=b
2 row(s)
Took 0.1031 seconds
```

冷热分离场景下的自动同步

自动同步功能适用于如下场景：数据刚写进去时是热数据（需要比较好的查询性能），随着时间推移逐渐变成冷数据（几乎不查）。例如对于一个订单系统，新创建的订单通常是热数据，但是半年前的订单就是冷数据了。针对这样的场景，自动同步功能通过配置表属性将热表和冷表关联起来，并指定数据在热表中的存活时间TTL，将热表中超过TTL的数据自动移动到冷表中。

前提

- 自动同步功能从HBase2.0.4和1.5.1版本开始支持，如果您的实例低于此版本，请先进行升级。
- 使用自动同步功能，要求热表和冷表具有一致的结构（列族相同）。
- 自动同步功能并不能保证超过TTL的数据立刻被移动到冷表中，而是有一个过程。因此如果使用开源客户端，需要注意有一部分冷数据仍然存在于热表中。也可以使用阿里客户端，我们对这种情况做了处理，使得在应用看来冷热数据是严格按照TTL切分的。

使用开源客户端

举例：我们已经建立好了目标表cold，使用如下语句在新建hot表时指定将hot表中超过1天的数据同步到

cold表中：

```
create 'hot','info',CONFIGURATION => {'hbase.hstore.engine.class' =>
'org.apache.hadoop.hbase.regionserver.TransferStoreEngine','hbase.transfercompactor.sink.table'=> 'cold',
'hbase.transfercompactor.source.ttl' =>86400}
```

如果hot表是一张已经存在的表，可以使用如下语句修改表的属性：

```
alter'hot',CONFIGURATION => {'hbase.hstore.engine.class' =>
'org.apache.hadoop.hbase.regionserver.TransferStoreEngine','hbase.transfercompactor.sink.table'=> 'cold',
'hbase.transfercompactor.source.ttl' => 86400}
```

字段说明：

- hbase.hstore.engine.class指定为 org.apache.hadoop.hbase.regionserver.TransferStoreEngine表示需要在compact的时候进行数据迁移。
- hbase.transfercompactor.sink.table 要同步到的目标表名称，用户需要确保目标表已经建好且结构和源表一致。
- hbase.transfercompactor.source.ttl 表示要将多久之前的数据移动到目标表。单位秒。

使用阿里客户端

使用阿里客户端2.0.2版本（参见阿里云开源HBase客户端）。在HBase Shell中使用如下语句建表：

```
create_layered 'hot','cold',86400, 'info',...（其他参数同create）
```

这个语句会同时创建hot和cold，将cold设置为冷表，并配置hot中的数据在1天后被移动到cold中。使用如下代码创建connection：

```
    config.set(ClusterConnection.HBASE_CLIENT_CONNECTION_IMPL,
LayeredConnectionImplementation.class.getName());
connection = ConnectionFactory.createConnection(config);
```

使用针对冷热分离场景做了特殊处理的LayeredConnectionImplementation类，这样就可以做到应用看来冷热数据是严格按照TTL切分的。

注意事项

1.冷存储的读IOPS能力很低（每个节点上限为25），所以冷表只适合低频查询场景。

2.写入吞吐上，冷表和基于高效云盘的热表相当，可以放心写入数据。

3.冷表不适合并发大量读请求，如果有这种行为可能会导致请求异常。

4.购买冷存储空间特别大的客户可以酌情调整 读IOPS 能力，详情工单。

5.建议平均每个core节点管理冷数据不要超过30T。如果是同时有冷热表的集群，需要看region数量来衡量。如果需要单个core节点管理更大数据量的冷数据，可以工单咨询优化建议。

6.冷表无法转换成热表

7.如果未购买冷存储，但是建表时候指定了冷存储，会导致实际开通冷存储后表里数据迁移到冷存储。这个过程会影响表的访问，非常不建议这么操作。

性能数据

测试场景1：随机写

环境

Core配置：

- ecs.sn2ne.2xlarge（8核32G ）
- 300G高效云盘 * 4
- 6台

Master配置：

- ecs.sn1ne.2xlarge（8核16G）
- 2台

HBase配置：

- 表 BLOCKCACHE => ‘false’
- short-circuit 关闭
- 1台Core启动RS

HDFS配置：

- 6台Core都启动DataNode

测试机器：

- ecs.sn1ne.2xlarge（8核16G）

测试

```
hbase pe --nomapred --valueSize=100 --rows=1000000 --table=test  --presplit=64 randomWrite 120
```

结果

存储类型	TPS(120线程/无batch/value100B)	TPS(120线程/无batch/value100B/SKIP_WAL)
冷表	193116.68	211393.4
热表	192819.72	219569.1

测试场景2：Get延迟对比

环境

Core配置：

- ecs.sn2ne.2xlarge（8核32G ）
- 300G高效云盘 * 4
- 6台

Master配置：

- ecs.sn1ne.2xlarge（8核16G）
- 2台

HBase配置：

- 表 BLOCKCACHE => ‘false’
- short-circuit 关闭
- 只有1台Core启动RS

HDFS配合：

- 6台均Core启动DataNode

测试机器：

- ecs.sn1ne.2xlarge（8核16G）

数据准备

```
#冷表
hbase pe --nomapred --oneCon=true --valueSize=1024 --presplit=32 --rows=2000000 --table=$TABLE --
autoFlush=true --storagePolicyHFile=COLD sequentialWrite 15
#热表
hbase pe --nomapred --oneCon=true --valueSize=1024 --presplit=32 --rows=2000000 --table=$TABLE --
autoFlush=true sequentialWrite 15
#大约30G数据, 生成完毕后flush一次表
```

测试

```
hbase pe --nomapred --oneCon=true --rows=2000 --table=$TABLE randomRead 15
```

结果

存储类型	平均延迟us	p999延迟us
冷表	13414	320414
热表	3862	265905

测试场景3：Scan顺序读耗时对比

环境

Core配置：

- ecs.sn2ne.2xlarge （8核32G ）
- 300G高效云盘 * 4
- 6台

Master配置：

- ecs.sn1ne.2xlarge（8核16G）
- 2台

HBase配置：

- 表 BLOCKCACHE => ‘false’
- short-circuit 关闭
- hbase.storescanner.use.pread=false
- 只有1台Core启动RS

HDFS配合：

- 6台均Core启动DataNode

测试机器：

- ecs.sn1ne.2xlarge（8核16G）

数据准备

```
#冷表
```

```
hbase pe --nomapred --oneCon=true --valueSize=1024 --rows=2000000 --table=$TABLE --autoFlush=true --
storagePolicyHFile=COLD sequentialWrite 1
#热表
hbase pe --nomapred --oneCon=true --valueSize=1024 --rows=2000000 --table=$TABLE --autoFlush=true
sequentialWrite 1
#大约2G数据, 2个region，生成完毕后flush一次表，并做一个major，HDFS数据全在本地
```

测试

```
hbase pe --nomapred --oneCon=true --rows=2000000 --caching=1000 --table=$TABLE scan 1
#一个线程顺序scan 2G数据
```

结果

存储类型	2G数据耗时(caching30)	2G数据耗时(caching1000)
冷表	464659ms, 4.23MB/s	109779ms, 17.91MB/s
热表	51457ms, 38.22MB/s	17764m, 110.7MB/s

标准版全文索引solr(即将下线)

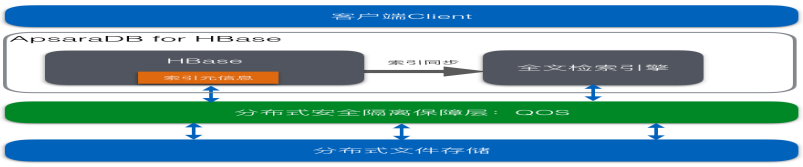
服务介绍

全文索引功能目前已经全面升级，使用老架构的标准版全文索引服务即将下线，新用户请移步增强版全文索引功能

服务介绍

概述

本文介绍“HBase全文索引服务”，及其常规使用的场景说明。“HBase全文索引服务”是基于稳定的阿里云HBase为底座，引进了使用广泛的Solr全文索引服务，进一步增强了HBase的检索能力，使得用户可以在充分发挥HBase KV能力的同时，也能利用全文检索构建复杂条件的查询业务。如图：



“HBase全文索引服务”属于ApsaraDB for HBase的增强服务，用户无需额外购买服务即可开启使用Solr全文索引功能，用户只需要合理规划集群规格即可便捷切换。

注意：目前只有2019年1月25日之后创建的新实例支持控制台有开启“全文索引服务”开关。如果2019年1月25之后创建的实例也没有对应Solr服务开启按钮，就是这个区开放时间是延后几天。另外所有2019年1月25日之前创建的旧实例目前不支持开启，如果需要，可以购买新实例，或者联系“云HBase答疑”客服申请旧实例迁移新版本实例。具体请与客服沟通。

使用场景

我们都知道HBase是大数据在线存储优秀选择，而Solr是分布式全文检索的最佳实践之一。HBase合适大数据存储，高并发高效KV查询，水平扩展性更强。Solr在分布式全文检索能力上功能完善，支持各种复杂的条件查询。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&检索服务。常见的使用场景可总结为：需要保存大数据量数据，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，还可能会使用高并发KV精确查询。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。

以上只是概述一些可能的场景，实际只要有以下几种类型的查询需求，都可以使用“HBase全文索引服务”来增强检索能力。几种查询类型如下：

- 任意个条件AND/OR组合查询
- facet按条件分类，统计匹配结果集中记录个数。常见的如网站的搜索结果侧边栏
- 复杂查询的多种排序，并分页
- 常见的条件筛选进行avg/min/max/sum等统计
- 分词，关键字查询。常见的商品标题、视频/新闻标题等关键字查询

以上细节详情见：Solr增强HBase检索能力PPT

快速入门

全文索引功能目前已经全面升级，使用老架构的标准版全文索引服务即将下线，新用户请移步增强版全文索引功能

本文从一个简单的 Demo 介绍开始，快速入门体验云 HBase “全文索引服务”的功能。整体流程是：开通全文服务 → 创建索引 → 写入HBase数据 → 查看 Solr 索引数据

1. 开通全文索引服务

在云 HBase 实例控制台左侧“全文索引服务”标签页，点击申请开通全文索引服务，等待片刻后服务即可使用。如图：

开通成功之后，如图：

2. 创建索引

创建索引核心分为3步：创建 HBase 表 solrdemo、创建 Solr 表 democollection、创建 solrdemo 与 democollection 的字段映射关系。下面介绍一个简单示例演示，快速体验服务使用流程。

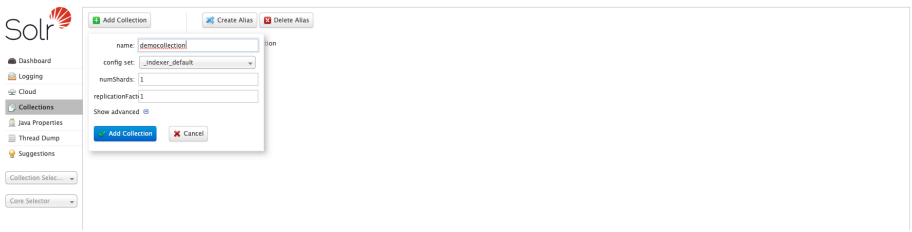
2.1 创建 HBase 表

使用 HBase Shell，创建 HBase 表如下：

```
hbase shell> create 'solrdemo', {NAME=>'info', REPLICATION_SCOPE=>'1'}
```

2.2 创建 Solr 表

进入 Solr WebUI 创建 democollection 表，选择 _indexer_default 配置目录 如下图：



如何访问 Solr WebUI 参考链接。

2.3 创建索引映射关系

然后我们创建 HBase 表 solrdemo 到 Solr 表 democollection 的索引字段映射关系配置文件，简单起见，我们假设solrdemo 的 info:name 映射到 democollection 的 name_s 字段，则编辑 demo.xml 文件内容如下：

```
<indexer table="solrdemo">
<field name="name_s" value="info:name" type="string" />
<param name="update_version_l" value="true"/>
</indexer>
```

然后下载 solr-7.3.1-ali-1.0.tar.gz 包并解压如下：

```
wget http://public-hbase.oss-cn-hangzhou.aliyuncs.com/installpackage/solr-7.3.1-ali-1.1.tgz
tar zxvf solr-7.3.1-ali-1.1.tgz
```

修改 solr-7.3.1-ali-1.1/bin/solr.in.sh 文件的 ZK_HOST 值，去掉注释，并设置

```
ZK_HOST="hb-xxxxxx-master2-001.hbase.rds.aliyuncs.com:2181,hb-xxxxxx-master3-
001.hbase.rds.aliyuncs.com:2181,hb-xxxxxx-master1-001.hbase.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 公网访问方案。

接下来运行以下命令创建 demoindex 索引，把 solrdemo 和 democollection 根据 demo.xml 关联起来，如下：

```
solr-7.3.1-ali-1.1/bin/solr-indexer add -n demoindex -f demo.xml -c democollection
```

接下来，我们运行 solr-indexer list命令查看刚刚添加的索引是否已经添加成功，如下：

```
solr-7.3.1-ali-1.1/bin/solr-indexer list
```

显示如下：

```
demoindex
```

```
.....
+ Connection params:
+ solr.zk = hb-xxxxxx-master2-001.hbase.rds.aliyuncs.com:2181,hb-xxxxxx-master3-
001.hbase.rds.aliyuncs.com:2181,hb-xxxxxx-master1-001.hbase.rds.aliyuncs.com:2181/solr
+ solr.collection = democollection
+ Indexer config:
142 bytes, use -dump to see content
+ Processes
+ 4 running processes //注：有多少个worker，就有多少个running，表示成功了
+ 0 failed processes
.....
```

有多少个 Worker，就有多少个 running process ，表示成功了。比如此实例规格是 4 个 Worker 的 HBase 实例，那这里就有 4 个 running process。如果没有4个，请检查上述步骤配置是否有错误，或联系“云HBase答疑”客服咨询 & 帮助。

注意：其他的 solr-indexer 命令请参阅命令 help，请仔细阅读 help，并确认命令效果后再执行。

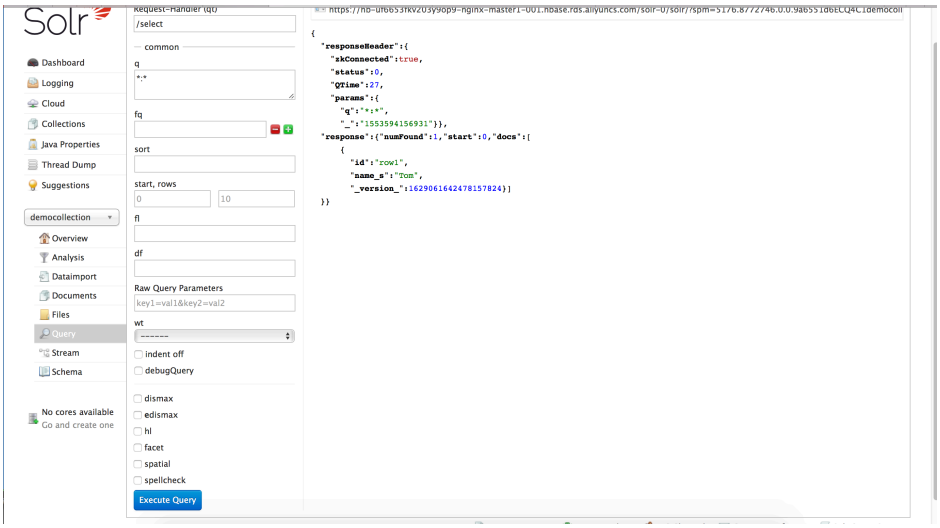
3. 写入HBase 数据

HBase Shell 写入简单一个数据如下：

```
hbase shell> put 'solrdemo','row1','info:name','Tom'
```

4. 查看 Solr 索引数据

进入 Solr WebUI 的选择对应 Collection，直接点击 query 按钮查看数据，如图：



可以看到，Solr 的 document 数据的 id 字段就是 HBase 的 RowKey。在实际进行条件查询的时候，我们可以使用 Solr API 进行查询，拿到 Solr 返回的 id 列表，逐个 id 当作 HBase 的 RowKey 调用 HBase Get 接口即可拿到符合各种复杂条件的 HBase数据了。

注：插入数据后，默认 Solr 延迟 15s 左右，即为 auto commit 时间间隔，所以可能需要等一下才能看到。当

然，也可以主动触发 commit 来立刻查看刚写入的数据。

索引重建

全文索引功能目前已经全面升级，使用老架构的标准版全文索引服务即将下线，新用户请移步增强版全文索引功能

HBase for Solr索引重建

EMR实例准备

EMR rebuild solr index需要访问HBase的hdfs/solr/hbase，则需要保证EMR实例和云HBase实例满足在同一个VPC下。注意，如果没有请事先创建vpc专有网，可能会在EMR实例购买安装是没有可选的VPC。

1. 购买安装EMR实例,与HBase同个可用区的实例，使用相同的 vpc id、vSwitch id。
2. 联系“云HBase答疑”，开通EMR访问云HBase实例hdfs/solr/hbase权限
3. 在HBase实例控制台中设置白名单，将EMR的ip列表设置到进去。
4. hbase-indexer-mr-2.0-job.jar到EMR机器上, 此包下载地址如下：
   使用的是HBase1.x版本实例时，使用如下：
   wget http://public-hbase.oss-cn-hangzhou.aliyuncs.com/installpackage/hbase-indexer-mr-2.0-job-for-hb1x.jar
   jar包md5值为：800162f9d7508ee6f87e238d78ddd7ba
   使用的是HBase2.x版本实例时，使用如下：
   wget http://public-hbase.oss-cn-hangzhou.aliyuncs.com/installpackage/hbase-indexer-mr-2.0-job-for-hb2x.jar
   jar包md5值为：c263a9b1825c6015a7a4848d4659fba7
   本jar包在EMR-3.14.0 版本测试兼容，其他版本环境若运行有问题，请联系“云HBase答疑”客服

作业配置

1. 在EMR机器上，拷贝 cp -r /etc/ecm/hadoop-conf-2.7.2-X.Y.Z ./custom_conf
2. 修改custom_conf/hdfs-site.xml，添加如下：

```
<property>
<name>dfs.nameservices</name>
<value>hbase-cluster</value>
</property>
<property>
<name>dfs.client.failover.proxy.provider.hbase-cluster</name>
<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>
</property>
```

```
<property>
<name>dfs.ha.automatic-failover.enabled.hbase-cluster</name>
<value>true</value>
</property>
<property>
<name>dfs.ha.namenodes.hbase-cluster</name>
<value>nn1,nn2</value>
</property>
<property>
<name>dfs.namenode.rpc-address.hbase-cluster.nn1</name>
<value>master1-1:8020</value>
</property>
<property>
<name>dfs.namenode.rpc-address.hbase-cluster.nn2</name>
<value>master2-1:8020</value>
</property>
```

咨询“云HBase答疑” master1-1、master2-1的hostname是什么，替换上述那么master1-1、master2-1。

3. 添加custom_conf/hbase-site.xml文件，文件中添加属性如下：

```
<?xml version="1.0"?>
<configuration>
<property>
<name>hbase.zookeeper.quorum</name>
<value>zk1,zk2,zk3</value>
</property>
</configuration>
```

其中zk1,zk2,zk3即为云HBase的zookeeper链接地址

咨询“云HBase答疑”，提出开放 EMR访问 某个云HBase实例需求，请求打开EMR的mapreduce访问云HBase权限
通过telnet master1-1 8020初步验证是否已经打开。其中master1-1替换为具体长名字的地址，参考zk地址中的master1-001中缀的地址。

验证配置运行hadoop —config custom_conf fs -ls hdfs://hbase-cluster/查看是否可以访问阿里云HBase的 hdfs目录。

运行作业

最后，运行如下命令启动重建索引作业：

```
hadoop --config custom_conf jar hbase-indexer-mr-2.0-job.jar \
--zk-host zk1,zk2,zk3/solr \
--collection collection001 \
--reducers 0 \
--hbase-indexer-zk=zk1,zk2,zk3 \
--hbase-table-name testtable \
```

```
--hbase-indexer-name testtable_index01
```

如果需要传入更多HBase 过滤参数，完整如下：

```
hadoop  --config custom_conf  jar hbase-indexer-mr-2.0-job.jar \
--zk-host zk1,zk2,zk3/solr \
--collection collection001 \
--reducers 0 \
--hbase-indexer-zk=zk1,zk2,zk3 \
--hbase-table-name test:index001 \
--hbase-indexer-name myindex002 \
--hbase-start-row startRow \
--hbase-end-row endRow \
--hbase-timestamp-format yyyy-MM-dd'T'HH:mm:ss.SSSZ \
--hbase-start-time startTime \
--hbase-end-time endTime \
--verbose
```

注意：” hbase-start-row” 、” hbase-end-row” 使用hbase包中的Bytes.toStringBinary(byte[] startRow)和Bytes.toBytesBinary(String startRow) 进行 string <=> byte[]的转换，建议你写程序得到这个start/end row的string，然后填写到上面命令中。

“hbase-timestamp-format”即为SimpleDataFormat的format string参数，对应解析startTime/endTime的string得到时间。如果不提供”hbase-timestamp-format”，则程序直接Long.parse(startTime/endTime) 解析得到timestamp值。

案例如下：

```
hadoop  --config custom_conf   jar hbase-indexer-mr-2.0-job.jar \
--zk-host zk1,zk2,zk3/solr \
--collection collection001 \
--reducers 0 \
--hbase-indexer-zk=zk1,zk2,zk3 \
--hbase-table-name test:index001 \
--hbase-indexer-name myindex002 \
-D "mapred.child.java.opts=-Xmx2048m" \
--hbase-start-row 00000000000000000000000000 \
--hbase-end-row 00000000000000000000000390 \
--verbose

hadoop --config custom_conf jar hbase-indexer-mr-2.0-job.jar \
--zk-host zk1,zk2,zk3/solr \
--collection collection001 \
--reducers 0 \
--hbase-indexer-zk=zk1,zk2,zk3 \
--hbase-table-name test:index001 \
--hbase-indexer-name myindex002 \
--hbase-start-time 1530780396659 \
--hbase-end-time 1531119958636 \
--verbose

hadoop --config custom_conf jar hbase-indexer-mr-2.0-job.jar \
--zk-host zk1,zk2,zk3/solr \
```

```
--collection collection001 \
--reducers 0 \
--hbase-indexer-zk=zk1,zk2,zk3 \
--hbase-table-name test:index001 \
--hbase-indexer-name myindex002 \
--hbase-timestamp-format "yyyy-MM-dd'T'HH:mm:ss.SSSZ" \
--hbase-start-time 2018-07-05T16:46:36.659+0800 \
--hbase-end-time 2018-07-09T15:05:58.636+0800 \
--verbose

或者
--hbase-timestamp-format "yyyy-MM-dd'T'HH:mm:ss" \
--hbase-start-time 2018-07-05T16:46:36 \
--hbase-end-time 2018-07-09T15:05:58 \
```

注意，这里启动MR任务运行过程中，不要人为修改索引属性，否则过程执行可能会失败。

作业超时处理

由于重建索引会对集群资源负载变大，如果作业太大hbase scan timeout时，可以适当调整timeout参数，如在hbase-site.xml添加如下:
如异常：org.apache.hadoop.hbase.client.ScannerTimeoutException: 98566ms passed since the last invocation, timeout is currently set to 60000

```
  <property>
<name>hbase.client.scanner.timeout.period</name>
<value>12000</value>
</property>
```

如果碰到mapreduce task超时最终导致作业失败时，首先检查一下是否hbase原表有个别region太大，导致一个作业运行过长导致，可以手动split对应那个region。其次，可以mapred-site.xml添加如下配置适当增加超时参数。
如异常：AttemptID:attempt_1534767970337_0010_m_000032_0 Timed out after 1800 secs

```
 <property>
<name>mapreduce.task.timeout</name>
<value>1800000</value>
</property>
```

建立索引详细说明

全文索引功能目前已经全面升级，使用老架构的标准版全文索引服务即将下线，新用户请移步增强版全文索引功能

建立索引详细说明

在“快速入门”小节中，可以体验了创建索引、使用索引的大体流程。本文针对建立索引进行详细说明，描述索引的工作基本原理以及配置映射关系，帮助进一步熟悉全文索引服务的使用。

基本原理描述

如上图所示，demoindex 索引描述了HBase 的 solrdemo 表和 Solr 的 democollection 各个索引列字段映射关系。命令行中 -c democollection 指定了这个 demoindex 关联的 Solr Collection 为 democollection; -f demo.xml 描述了 demoindex 使用的配置文件是 demo.xml 文件；在 demo.xml 文件中，table="solrdemo" 表示此映射关联了 HBase 的 solrdemo 表；demo.xml 文件中，每一行 field 就描述了 HBase 的一个 column 映射到 Solr 的一个 field 字段。例如图中的 field标签，表示 HBase 中 info:name 这个列映射到 Solr 的 name_s，其中 type="string" 表示 HBase的这个 info:name 存储的 byte[] 是 Bytes.toBytes(String) 得到的数据。类似的，type 还有 int、long、double、float、short、boolean，分别对应 HBase 存储的数据是 Bytes.toBytes(int)、Bytes.toBytes(long)、Bytes.toBytes(double)、Bytes.toBytes(float)、Bytes.toBytes(short)、Bytes.toBytes(boolean) 得到的 byte[] 数据。

对于 Solr 来说，存储的类型就是 field 字段在 schema 中描述的类型，这两个类型要对应。比如这里 name_s 其实暗含着这个 name_s 在 Solr 里面是 string 类型。后缀方式暗示类型的预定义都已经在 schema 中默认加上了，如果你需要有不带后缀 _s、_i、_l、_b、_f等的字段名，那么你需要显式在 Solr 的 schema 中配置指定其对应类型。其他 solr 更多动态类型映射，请参考 默认schema中的动态类型。

客户端使用详情

由上述小节可以看到，整个链路里面涉及到：HBase Shell 命令建表、solr-indexer 命令管理索引、Solr 命令操作创建 collection，3 个部分的DDL操作。首先我们看对于 HBase Shell 的建表操作，我们需要做什么，如下：

HBase Shell 命令行建表

```
create 'solrdemo', {NAME=>'info', REPLICATION_SCOPE=> '1'}
```

唯一需要修改的地方，就是设置涉及到同步列所在的列簇 info 的 REPLICATION_SCOPE 为 1,这里不多赘述。

solr-indexer 命令管理索引

Solr 客户端工具包准备

首先需要下载解压客户端：

```
wget http://public-hbase.oss-cn-hangzhou.aliyuncs.com/installpackage/solr-7.3.1-ali-1.1.tgz
tar zxvf solr-7.3.1-ali-1.1.tgz
```

修改 solr-7.3.1-ali-1.1/bin/solr.in.sh 文件，去掉 ZK_HOST 前面注释 #，并修改如下：

```
ZK_HOST="hb-xxxxxx-master2-001.hbase.rds.aliyuncs.com:2181,hb-xxxxxx-master3-
001.hbase.rds.aliyuncs.com:2181,hb-xxxxxx-master1-001.hbase.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 公网访问方案。

Solr 命令上传自定义配置并创建 Collection

如“快速入门” 章节我们创建一个 Solr 的 Collection 时，其实使用的是 Solr WebUI 进行创建的，并且使用的 _indexer_default 的这个 Collection 配置。这个 _indexer_default 配置目录中，其核心包含了 managed-schema 和solrconfig.xml 核心的两个配置文件，分别描述了 Collection 的每个字段是什么类型，以及 Collection 的一些系统动作，如cache、autoCommit 等。

下面我们来简单认识下两个配置文件，并通过一个例子，来创建一个具有定制配置的 Collection。为了方便，我们已经把样例的配置放置在了 solr-7.3.1-ali-1.1/server/solr/configsets/_democonfig 目录下，如下：

```
solr-7.3.1-ali-1.1/server/solr/configsets/_democonfig/conf/managed-schema
solr-7.3.1-ali-1.1/server/solr/configsets/_democonfig/conf/solrconfig.xml
solr-7.3.1-ali-1.1/server/solr/configsets/_democonfig/conf/lang
solr-7.3.1-ali-1.1/server/solr/configsets/_democonfig/conf/stopwords.txt
solr-7.3.1-ali-1.1/server/solr/configsets/_democonfig/conf/... ...
```

这里我们只需关心 managed-schema 和 solrconfig.xml 两个配置即可。打开 managed-schema、solrconfig.xml 配置文件，上面有基本的配置项说明描述，详细参考 schema配置相关说明 以及 solrconfig.xml配置说明。

接下来我们创建一个 Collection 命名为 democollection2,并声明 name 字段为 string 类型、age为 int 类型，如下：

```
cp -r solr-7.3.1-ali-1.1/server/solr/configsets/_democonfig solr-7.3.1-ali-1.1/server/solr/configsets/_democollection2
vim solr-7.3.1-ali-1.1/server/solr/configsets/_democollection2/conf/managed-schema
```

在 managed-schema 中大概 127 行附近空白处，添加如下行：

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

注意，managed-schema 里面预定义的 int 类型是用 pint 代表，其他类型也是，可以参见配置文件描述。

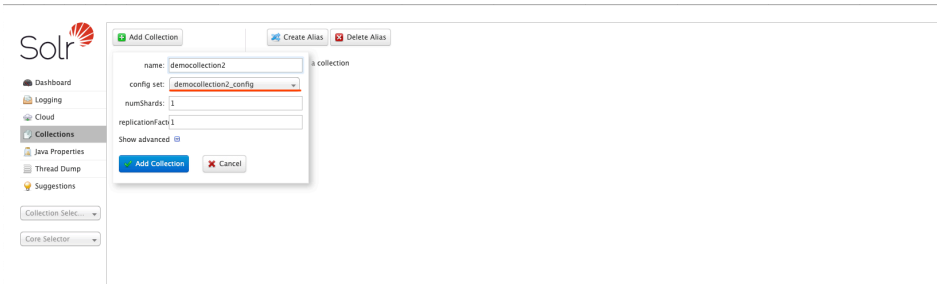
这里 name 字段是必需的，在使用此配置时，必需确保 HBase 的一行数据内必需包含 name 对应的列，否则会导致同步失败并阻塞。

接下来我们把刚刚修改的 _democollection 定制配置目录上传到 Solr 集群，执行命令如下：

```
solr-7.3.1-ali-1.1/bin/solr zk upconfig -d _democollection2 -n democollection2_config
```

-d _democollection2 指的是刚刚修改的目录名字，注意这个目录默认必须在 solr-7.3.1-ali-1.1/server/solr/configsets/ 下。-n democollection2_config 指的是我们把这个 -d 指定的目录上传到 Solr 上面时，重命名为 democollection2_config，当然你可以写不一样的名字，只要命名合法。

接下来，就是在 Solr WebUI 里面创建 Collection 时，选中对应的 democollection2_config 即可，这样就完成了一个定制 schema 配置的 Collection 创建。如图：



solr-indexer 创建索引

上述创建了HBase 表、自定义的 Collection 之后，接下来就是利用 solr-indexer 命令工具，创建索引把 solrdemo 和这个定制的 schema 配置的 democollection2 关联起来。

首先运行 solr-indexer 把“快速入门”中的 demoindex 删除掉，解除 solrdemo 表和 democollection 的映射关系，然后这里创建一个 demoindex2 索引，让 solrdemo 表和 democollection2 索引关联映射起来。如下：

```
solr-7.3.1-ali-1.1/bin/solr-indexer delete -n demoindex
```

此时“快速入门”中创建的 demoindex 已经被我们删除了。接下来就是创建demoindex2 了。

首先编辑准备 demoindexer2.xml 配置，描述 HBase 表 solrdemo 与 democollection2 的字段映射关系。demoindexer2.xml 配置如下：

```
<?xml version="1.0"?>
<indexer table="solrdemo">
<field name="name" value="info:name" type="string"/>
<field name="age" value="info:age" type="int"/>
<param name="update_version_l" value="true"/>
</indexer>
```

然后执行命令如下：

```
solr-7.3.1-ali-1.1/bin/solr-indexer add -n demoindex2 -f demoindexer2.xml -c democollection2
```

执行成功之后，可以运行 list 查看是否创建成功。

```
solr-7.3.1-ali-1.1/bin/solr-indexer list
```

到这里，我们就完成创建自定义 Collection 配置的索引创建了。

solr-indexer 更新索引

上述我们已经创建好了 Indexer，Indexer 已正常运行。此时若我们需要更新 Indexer，可以使用 solr-indexer 命令来完成。

若我们想增加一个索引列 info:desc，首先修改配置文件 demoindexer2.xml 如下：

```
<?xml version="1.0"?>
<indexer table="solrdemo">
<field name="name" value="info:name" type="string"/>
<field name="age" value="info:age" type="int"/>
<field name="desc_s" value="info:desc" type="string"/>
<param name="update_version_l" value="true"/>
</indexer>
```

运行更新命令如下：

```
solr-7.3.1-ali-1.1/bin/solr-indexer update -n demoindex2 -f demoindexer2.xml -c democollection2
```

之后我们查看一下更新是否生效：

```
solr-7.3.1-ali-1.1/bin/solr-indexer list -dump -t solrdemo
```

输出如下，可以看到 desc_s 已加入：

```
demoindex2
......
+ Connection params:
+ solr.zk = ......
+ solr.collection = democollection2
```

```
+ Indexer config:
<?xml version="1.0"?>
<indexer table="solrdemo">
<field name="name" value="info:name" type="string"/>
<field name="age" value="info:age" type="int"/>
<field name="desc_s" value="info:desc" type="string"/>
<param name="update_version_l" value="true"/>
</indexer>
+ Processes
......
```

Solr 从 HBase 同步数据有一定的提交延迟（默认15s左右），若需要修改此提交延迟，命令如下：

```
solr-7.3.1-ali-1.1/bin/solr-indexer update -n demoindex2 -f demoindexer2.xml -c democollection2 -w 3000
```

查看更新是否生效：

```
solr-7.3.1-ali-1.1/bin/solr-indexer list -dump -t solrdemo
```

输出如下，可以看到在 Connection params 中已加入了 solr.commitWithin：

```
demoindex2
......
+ Connection params:
+ solr.zk = ......
+ solr.collection = democollection2
+ solr.commitWithin = 3000
+ Indexer config:
<?xml version="1.0"?>
<indexer table="solrdemo">
<field name="name" value="info:name" type="string"/>
<field name="age" value="info:age" type="int"/>
<field name="desc_s" value="info:desc" type="string"/>
<param name="update_version_l" value="true"/>
</indexer>
+ Processes
......
```

注意：在更新 indexer 时，配置文件是必须的，若不想修改原来的 indexer 配置，则需要保持 -f 的配置文件内容与之前的配置一致。简而言之，更新 indexer 时，总是用 -f 指定的配置覆盖原有的配置。

solr-indexer 重建索引

由于 Indexer 只能够对启动后的增量数据进行索引，在实际的使用中，若想对存量数据进行索引，需要用到 solr-indexer 的索引重建功能。solr-indexer 关于索引重建的命令有 3 个：rebuild 启动索引重建任务，rebuild-status 查看索引重建任务状态、rebuild-cancel 取消索引重建任务。

在索引重建前，可以在之前的 HBase 表 solrdemo 中插入一些测试数据，并做预分区，以便更好的观察命令效果。

rebuild 命令 help 如下：

```
rebuild [-z zkConn] <-n indexName> <-t table> <-r maxRetries>
-z zkConn zookeeper connection string, such as local:2181/solr
if not specified, use ZK_HOST in solr.in.sh env instead
-n indexName the rebuild index name
-t tableName the rebuild table name
-r maxReties the max number of task retries
```

其中 -z、-n、-t 参数意义容易理解，注意 -t 指定的表名需要与 -n 指定的 Indexer 配置文件中的表一致。-r 指定了本次索引重建任务的失败尝试最大次数，一般情况下设置 10 即可。

执行 rebuild 命令，提交索引重建任务：

```
solr-7.3.1-ali-1.1/bin/solr-indexer rebuild -n demoindex2 -t solrdemo -r 10
```

我们还可以通过 rebuild-status 查看索引重建任务状态：

```
solr-7.3.1-ali-1.1/bin/solr-indexer rebuild-status -n demoindex2 -t solrdemo
```

输出信息类似如下：

```
Job Status information of job_solrdemo_demoindex2:

jobStatus=SUCCEEDED
jobStartTime=2019-07-xx xx:xx:xx xxx
jobEndTime=2019-07-xx xx:xx:xx xxx
completedRows=100000
tableName=solrdemo
indexName=demoindex2
totalTasks=3
completedTasks=3
runningTasks=0
pendingTasks=0
```

通过 rebuild-cancel 取消索引重建任务：

```
solr-7.3.1-ali-1.1/bin/solr-indexer rebuild-cancel -n demoindex2 -t solrdemo
```

再次通过 rebuild-status 查看索引重建任务状态，可以看到 jobStatus=CANCELLED：

```
Job Status information of job_solrdemo_demoindex2:

jobStatus=CANCELLED
jobStartTime=2019-07-xx xx:xx:xx xxx
jobEndTime=2019-07-xx xx:xx:xx xxx
completedRows=30000
tableName=solrdemo
```

```
indexName=demoindex2
totalTasks=3
completedTasks=3
runningTasks=0
pendingTasks=0
```

如果 rebuild-cancel 命令报错，说明本次索引重建任务已完成或已取消，可以重新执行 rebuild 命令，在任务执行过程中运行 rebuild-cancel。

注意：solr-indexer 命令的详细用法请参阅命令 help。请务必仔细阅读 help，并确认命令效果后再执行。

附一：索引映射文件 type 支持的类型

索引映射文件type支持的类型： int/short/long/string/boolean/float/double例如 new_index.xml：

```
<?xml version="1.0"?>
<indexer table="newtest">
<field name="name" value="info:name" type="string"/>
<field name="age" value="info:age" type="int"/>
<field name="man" value="info:man" type="boolean"/>
<field name="height" value="info:height" type="double"/>
... ...
<param name="update_version_l" value="true"/>
</indexer>
```

注意，这里描述的 type 指的是在 HBase 中，某个列 value 是以什么样的格式保存的 byte[] 值。比如 info:age 是type="int"，表示在 HBase 当中，info:age 这个列的值是通过 Bytes.toBytes(int) 得到的 4 个字节 byte[]。如果你写入的是其他类型的 byte[]，那么将会解析失败。这里建议用户写入 HBase 的时候，做好类型格式清洗，不要插入错误格式的数据，否则会导致同步失败。

附二：Collection 支持的字段类型

见 managed-schema 文件，分别支持 string/pint/plong/boolean/pdouble/pfloat 等，其他更多动态类型映射，请参考 默认schema中的动态类型。

附三：update_version_l 的作用

update_version_l 为固定写法，表示 Document 的最后更新时间，始终设置为 true。

附四：HBase 的 RowKey 映射与 Solr 中 Document 的 id 字段

默认情况下，HBase 的 RowKey 是通过 Bytes.toString(byte[]) 得到的 string 值作为 Solr Document 的 id 字段值进行保存。现支持两种格式映射关系，如下：

- string： 默认 HBase的 RowKey 以 Bytes.toString(byte[] rowkey) 转换为 Document id；
- hex： 提供 commons-codec 包的 Hex.encodeHexString(byte[] rowkey) 转换为 Hex 方式的 Document id。对应反解析即可。

指定方式如下：new_index.xml，指定 unique-key-formatter 为 hex 或者 string，默认是string。指定使用

unique-key-formatter=“hex” 如下：

```
<indexer table="solrdemo" unique-key-formatter="hex">
<field name="age" value="info:age" type="int"/>
... ...
</indexer>
```

指定使用 unique-key-formatter=“string” 如下：

```
<indexer table="solrdemo" unique-key-formatter="string">
<field name="age" value="info:age" type="int"/>
... ...
</indexer>
```

默认 unique-key-formatter 就是 string，如下：

```
<indexer table="solrdemo">
<field name="age" value="info:age" type="int"/>
... ...
</indexer>
```