云数据库 HBase

HBase 增强版(Lindorm)

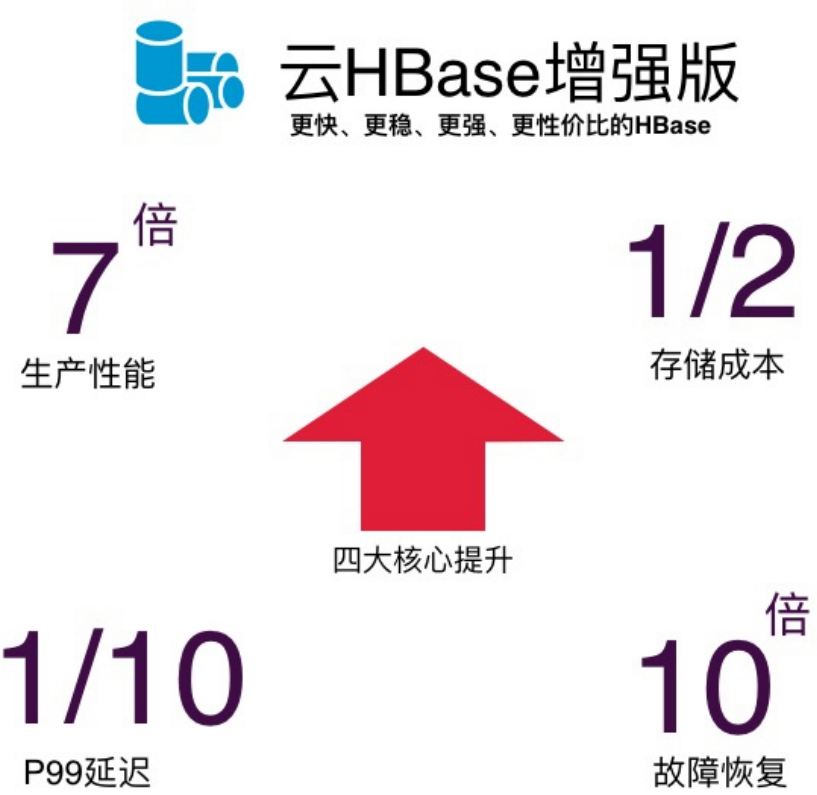
HBase 增强版(Lindorm)

产品简介

基本介绍

简介

阿里云数据库HBase增强版，是基于阿里集团内部使用的Lindorm产品研发的、完全兼容HBase的云上托管数据库，从2011年开始正式承载阿里内部业务的海量数据实时存储需求，支撑服务了淘宝、支付宝、菜鸟、优酷、高德等业务中的大量核心应用，历经双十一、春晚、十一出行节等场景的大规模考验，在成本、性能、稳定性、功能、安全、易用性等方面相比社区版拥有诸多优势和企业级能力



Lindorm是阿里HBase的内部分支，非常适合于对规模、吞吐、性能、可用性等有更高要求的企业级场景，面对大数据(无限扩展、高吞吐)、在线服务(低延时、高可用)、多功能查询的诉求，其可为用户提供无缝扩展、高吞吐、持续可用、毫秒级稳定响应、强弱一致可调、低存储成本、丰富索引的数据实时混合存取能力。

Lindorm的发展

时间	事件
2011.3	基于hbase研发的Lindorm-1.0在淘宝上线，服务于数据魔方、日志队列等场景
2012.3	数据规模达到100TB，秒请求10万
2013.5	大集群功能上线，开启多租户服务
2014.1	SQL功能上线，兼容Phoenix
2015.11	高可用架构落地，峰值秒请求超5000万
2016.11	内部机器规模达到1万台，作为中台服务高德、优酷、文娱等全经济体
2017.8	HBase社区版在阿里云商业化，正式服务云上企业级客户
2018.7	Lindorm-2.0发布，内部引擎大幅升级优化，实现性能7倍于开源HBase

2019.8	增强版(Lindorm)商业化，在存储、索引、容灾等方面发布多项独家功能
2019.12	基于Lindorm研发的Serverless版，最低月价格小于10元，满足小规模生产需求

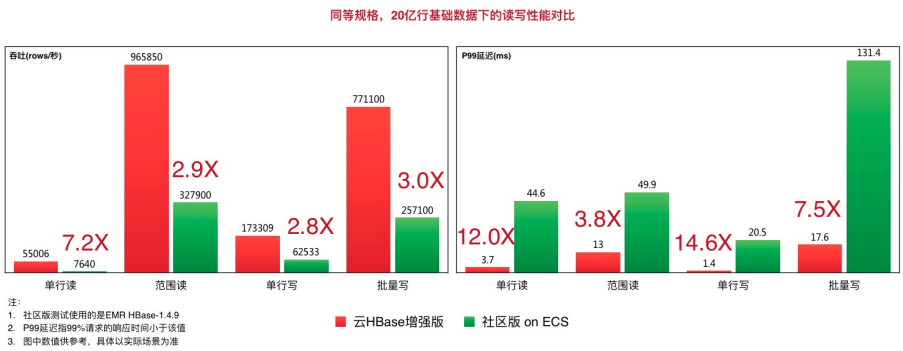
核心特性

Lindorm作为面向大数据领域的NoSQL服务，专注于低成本、高扩展、智能化的半结构化和非结构化存储场景，完全兼容HBase协议。目前，lindorm是阿里集团最基础存储设施之一，在多年的阿里巴巴双十一全球狂欢节上提供领先的大数据在线存储能力，支撑了数百PB规模的存储和其每秒数亿次的峰值访问、每日数十万亿次的海量吞吐。

相比HBase开源版，其拥有诸多强悍的企业级特性，您可以在**产品优势**中查看这些特性的简介，或者单击其中的链接获取更多信息。

核心优势

相比于标准版，云HBase增强版在如下几个方面拥有十分明显的优势，完整能力列表请参考**产品优势**中的与开源HBase对比

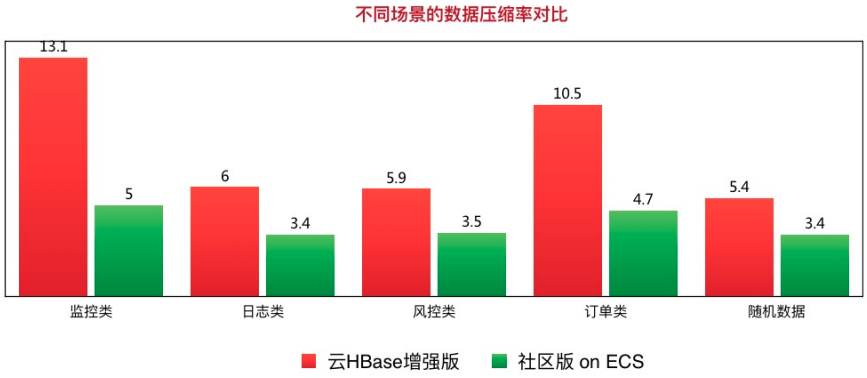


高性能

HBase增强版在RPC、内存管理、缓存、日志写入等方面深度优化，通过高性能数据结构、协程、合并提交、可检索编码等核心技术大幅提升读写的性能，在相同资源规格下，吞吐可达**HBase社区版的7倍**以上，毛刺可低至**HBase社区版的1/10**，更多数据参见**性能白皮书**

低成本

- 高压缩：HBase增强版独家内置深度优化的Zstandard算法，最高可达**13倍数据压缩比**，比常用SNAPPY**提升50%+**



- 冷热分离：面对大多场景中近热远冷的数据特点，增强版内置全自动的分层存储能力，应用无需任何改造，系统支持自动冷热分离，冷数据使用高压缩和廉价存储介质，减少70%成本，热数据可提升访问性能15%，详情参考**链接**
- 丰富的存储介质：支持高效云盘、SSD云盘、本地HDD、本地SSD。独家支持冷存储(OSS)、容量型云盘(超性价比云盘，即将发布)

高可用

HBase增强版对MTTR（平均故障恢复时间）做了深度的优化和改进，故障恢复速度可以达到**HBase的10倍**以上。并且其基于日志即存储思想和PACELC理论构建的分区多副本复制架构，可以提供多种数据一致性等级，方便应用在一致性、可用性、延迟和可编程性之间的灵活选择，充分满足应用的同城双活、异地多活、三机房强一致容灾等高可用需求(目前仅提供单可用区部署，更多高级形态将在未来逐渐开放)。

丰富检索

HBase增强版内置高性能的**全局二级索引**，满足数据非主键查询的需求，业务使用透明，查询自优化支持，可按需冗余非索引列；同时，增强版也提供**全文检索服务**，智能集成搜索引擎Solr，提供面向海量数据的存储、多维查询、全文索引等统一访问的混合型能力。

多租户

内置面向多租户的数据安全和资源隔离能力，提供标准的用户名密码认证、ACL、Quota、Resource Group等特性，满足大客户构建一站式企业HBase平台、提升开发效率、优化资源利用率的需求。更多内容请参考**集群管理**章节

快速入门

JAVA SDK安装

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

连接集群

HBase增强版支持通过Java/C++/Python/Go等API进行访问，访问集群需要两个信息：连接地址和用户名密码，您可以通过控制台获得。

连接地址

HBase增强版支持通过内网(VPC)和公网进行访问，两者使用不同的地址域名，并且不同方式(Java API、C++/Python/Go等API), 需要使用不同的访问端口，用户可以在集群的控制台中的“数据库连接”页面中看到连接地址和对应的端口。

HBase增强版集群一共提供了两类访问地址，两者者的地址都相同，只是端口不一样。

Java API的连接地址

与HBase标准版提供Zookeeper集群地址做为访问地址不同的是，HBase增强版使用一个地址域名作为访问的统一入口，该地址域名背后是一组高可用的服务器，能够帮助HBase客户端与后端的RegionServer建立直连，以进行高性能地访问。Java API访问指南请参考Java SDK安装和Java API访问

C++/Python/Go等API的访问地址

HBase增强版通过Thrift的方式支持C++/Python/Go等非Java访问HBase，如果您有非Java语言的应用需要访

问HBase，可以参考多语言访问指南。

公网访问

以上两种访问方式均支持从公网直接访问，只需要点击上述图片中的“申请外网地址”连接，在上述地址栏中，就会出现公网访问地址。用户只需要在应用中讲地址替换成公网地址即可。如果需要停止公网访问，请点击“释放外网地址”

注1：公网访问只用作开发调试或者测试，不可用于生产，公网访问下不保障SLA协议 。公网带宽及网络延迟都有限，性能会有很大衰退。 注2：切勿在VPC网络下使用公网域名，否则所有流量都会走公网，严重影响访问性能

白名单

为了连接安全，无论是公网访问，还是内网访问，均需要**将访问数据库的地址加入到白名单中。**加入白名单请参考设置白名单

用户名密码

HBase增强版内置用户认证和ACL的安全体系，使用可参考用户和ACL管理。HBase增强版在实例创建后会默认新增用户名为root，密码为root的超级账号，拥有集群的所有权限，您可以使用该root账号访问这个集群。也可以在用户和ACL管理页面中新增账号、删除或者修改root账号的密码。如果不需要访问时提供用户名密码，可以在用户和ACL管理页面中将此项功能关闭。

HBase Java API 访问

准备

1. 完成Java SDK安装（必看！开源客户端无法访问增强版）
2. 获取集群的连接地址

配置客户端参数

您可以通过以下两种方式，来配置访问HBase增强版的客户端参数.

方式一：配置文件

hbase-site.xml 中增加下列配置项：

```
<configuration>
<!--
集群的连接地址，在控制台页面的数据库连接界面获得(注意公网地址和VPC内网地址)
-->
<property>
<name>hbase.zookeeper.quorum</name>
<value>ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020</value>
</property>
<!--
设置用户名密码，默认root:root，可根据实际情况调整
-->
<property>
<name>hbase.client.username</name>
<value>root</value>
</property>
<property>
<name>hbase.client.password</name>
<value>root</value>
</property>
<!--
如果您直接依赖了阿里云hbase客户端，则无需配置connection.impl参数，如果您依赖了alihbase-connector，则需要配置此参数
-->
<!--property>
<name>hbase.client.connection.impl</name>
<value>org.apache.hadoop.hbase.client.AliHBaseUEClusterConnection</value>
</property-->
</configuration>
```

方式二：代码

通过代码Create Configuration，然后增加相关配置

```
// 新建一个Configuration
Configuration conf = HBaseConfiguration.create();
// 集群的连接地址，在控制台页面的数据库连接界面获得(注意公网地址和VPC内网地址)
conf.set("hbase.zookeeper.quorum", "ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020");
// 设置用户名密码，默认root:root，可根据实际情况调整
conf.set("hbase.client.username", "root")
conf.set("hbase.client.password", "root")

// 如果您直接依赖了阿里云hbase客户端，则无需配置connection.impl参数，如果您依赖了alihbase-connector，则需要配置此参数
//conf.set("hbase.client.connection.impl", AliHBaseUEClusterConnection.class.getName());
```

注：如果您依赖了老版本的alihbase-connector（版本号小于1.0.9/2.0.9,请参考老版本的参数配置文档）

创建连接

通过配置conf创建Connection，Conf的生成详见上一节中的代码访问方式。

```
// 创建 HBase连接，在程序生命周期内只需创建一次，该连接线程安全，可以共享给所有线程使用。
// 在程序结束后，需要将Connection对象关闭，否则会造成连接泄露。
// 也可以采用try finally方式防止泄露
Connection connection = ConnectionFactory.createConnection(conf);
```

使用API

建立完连接后，即可使用Java API访问HBase增强版集群。下面提供一些简单的Java 示例。

DDL操作

```
try (Admin admin = connection.getAdmin()){
// 建表
HTableDescriptor htd = new HTableDescriptor(TableName.valueOf("tablename"));
htd.addFamily(new HColumnDescriptor(Bytes.toBytes("family")));
// 创建一个只有一个分区的表
// 在生产上建表时建议根据数据特点预先分区
admin.createTable(htd);

// disable 表
admin.disableTable(TableName.valueOf("tablename"));

// truncate 表
admin.truncateTable(TableName.valueOf("tablename"), true);

// 删除表
admin.deleteTable(TableName.valueOf("tablename"));
}
```

DML操作

```
//Table 为非线程安全对象，每个线程在对Table操作时，都必须从Connection中获取相应的Table对象
try (Table table = connection.getTable(TableName.valueOf("tablename"))) {
// 插入数据
Put put = new Put(Bytes.toBytes("row"));
put.addColumn(Bytes.toBytes("family"), Bytes.toBytes("qualifier"), Bytes.toBytes("value"));
table.put(put);

// 单行读取
Get get = new Get(Bytes.toBytes("row"));
```

```
Result res = table.get(get);

// 删除一行数据
Delete delete = new Delete(Bytes.toBytes("row"));
table.delete(delete);

// scan 范围数据
Scan scan = new Scan(Bytes.toBytes("startRow"), Bytes.toBytes("endRow"));
ResultScanner scanner = table.getScanner(scan);
for (Result result : scanner) {
// 处理查询结果result
// ...
}
scanner.close();
}
```

HBase Shell访问

准备

1.获取HBase tar包。有两种方式可以获得访问HBase增强版的tar包

- 方式一：直接下载完整压缩包，里面已经拥有了访问HBase增强版所需的全部依赖，并做了功能增强。下载地址：云HBase控制台 -> 数据库连接 -> 连接信息 -> hbaseue shell下载。
- 方式二：直接从HBase官方网站下载你所需版本的HBase的tar包。并按照Java SDK 安装的指导，将相应版本的alihbase-connector jar包拷贝至tar包解压后的lib目录中

2.获取集群的连接地址， HBase Shell使用的连接地址为连接信息中的Java API访问地址部分（注意专有网络地址和外网地址的区别）。

配置

在解压后的tar包中的conf/目录下的hbase-site.xml文件中，加入如下配置：

```
<configuration>
<!--
集群的连接地址，在控制台页面的数据库连接界面获得(注意公网地址和VPC内网地址)
-->
<property>
<name>hbase.zookeeper.quorum</name>
<value>ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020</value>
</property>
```

```
<!--
设置用户名密码，默认root:root，可根据实际情况调整
-->
<property>
<name>hbase.client.username</name>
<value>root</value>
</property>
<property>
<name>hbase.client.password</name>
<value>root</value>
</property>
</configuration>
```

使用shell访问

进入tar包解压路径里的bin/目录，输入

```
./hbase shell
```

就可以开始使用原生的HBase Shell访问HBase增强版（默认log文件在hbase tar包解压后的logs目录下）。Shell的详细使用方法可以参看Shell入门.使用Shell连接增强版只能做简单的DDL操作和数据读写操作,管理接口如balance,move等集群管理命令都已经被禁用,详细请参考使用限制部分。

多语言(C++/Python/Go等)API访问

简介

HBase增强版通过Thrift支持多语言访问，只要是Thrift支持的语言，都可以访问HBase增强版。HBase增强版服务端的Thrift版本是0.12.0，虽然说thrift支持向后兼容，但如果有条件的用户，最好还是下载0.12.0的thrift，点击这里下载。一些语言提供了管理依赖的方法，可以遵循这些语言的使用习惯来安装thrift，如Python语言可以通过pip install thrift来安装，Go语言可以直接在代码里import {"github.com/apache/thrift/lib/go/thrift"}即可。

HBase增强版使用的Thrift接口定义是HBase的thrift2。因此需要下载thrift2的定义文件生成相应语言的接口。相比thrift1定义，HBase中的thrift2接口定义更加清晰，用户可以获得和Java语言类似的API调用体验。之前的thrift2之所以使用不广泛是因为thrift2的接口不全，不支持建表删表等DDL操作。但是目前thrift2相关的API定义已经由阿里云的开发人员补全并回馈给了社区（HBASE-21649）。目前thrift2比thrift1的功能更全而且更容易使用。

准备

1. 下载Thrift安装包，点击这里下载。
2. 下载**HBase Thrift2定义文件**
3. 获取**集群的连接地址**,多语言访问使用的连接为连接信息中的“非 JAVA 语言 Thrift2 访问地址”访问地址部分（注意专有网络地址和外网地址的区别）。

访问

Thrift的使用帮助可以参考Apache Thrift的**官方文档**,下面给出简单的使用方法。

1.生成对应语言的接口定义文件

从上述地址下载接口定义文件后，按照如下语法生成接口定义文件

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

例如：

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

2. 构造客户端访问HBase增强版

HBase增强版Thrift服务器端的**transport层使用的是HTTP**，因此在构造客户端时，需要thrift中的ThttpClient（各个语言都有相应实现）。并且在ACL开启的情况下，需要在ThttpClient上加上两个header来向服务器传输用户名和密码进行认证（如果关闭ACL则不需要）。Thrift在每个语言实现的ThttpClient都有加定制header的函数。如以Python语言的话，用以下方法构造客户端和设置连接串/用户名密码。更多语言示例可以参见demo。

```
# -*- coding: utf-8  -*-
# 以下两个模块可以通过 pip install thrift 安装获得
from thrift.protocol import TBinaryProtocol
from thrift.transport import THttpClient

# 下面的模块通过 thrift --gen py hbase.thrift 来生成
from hbase import THBaseService
from hbase.ttypes import TColumnValue, TColumn, TTableName, TTableDescriptor, TColumnFamilyDescriptor,
TNamespaceDescriptor, TGet, TPut, TScan

# 连接地址
```

```
url = "http://host:9190"
transport = THttpClient.THttpClient(url)
headers = {}
# 用户名
headers["ACCESSKEYID"]="root";
# 密码
headers["ACCESSSIGNATURE"]="root"
transport.setCustomHeaders(headers)
protocol = TBinaryProtocol.TBinaryProtocolAccelerated(transport)
client = THBaseService.Client(protocol)
transport.open()
# 具体操作，最后close连接
transport.close()
```

多语言Demo

所有的Demo的完整的代码都上传到了GitHub上，包括该语言thrift定义文件，以及所依赖的libray（某些语言适用）。用户可以直接下载Github上相应语言的代码适用

Python

https://github.com/aliyun/aliyun-apsaradb-hbase-demo/tree/master/hbase/thrift2/python

Go

https://github.com/aliyun/aliyun-apsaradb-hbase-demo/tree/master/hbase/thrift2/go

C++

https://github.com/aliyun/aliyun-apsaradb-hbase-demo/tree/master/hbase/thrift2/cpp

node.js

https://github.com/aliyun/aliyun-apsaradb-hbase-demo/tree/master/hbase/thrift2/nodejs

PHP

https://github.com/aliyun/aliyun-apsaradb-hbase-demo/tree/master/hbase/thrift2/php

更多语言

更多语言的使用请参见Thrfit官方文档

监控与报警

HBase增强版使用**云监控**（开箱即用的企业级开放型一站式监控解决方案，完整使用请参考其**相关介绍**）实现系统的监控与报警。本文介绍如何配置监控和配置报警。

监控

HBase增强版使用新版云监控，以**实例**为粒度组织和展示监控数据。目前有两种方式来查看某个HBase增强版实例的监控数据。

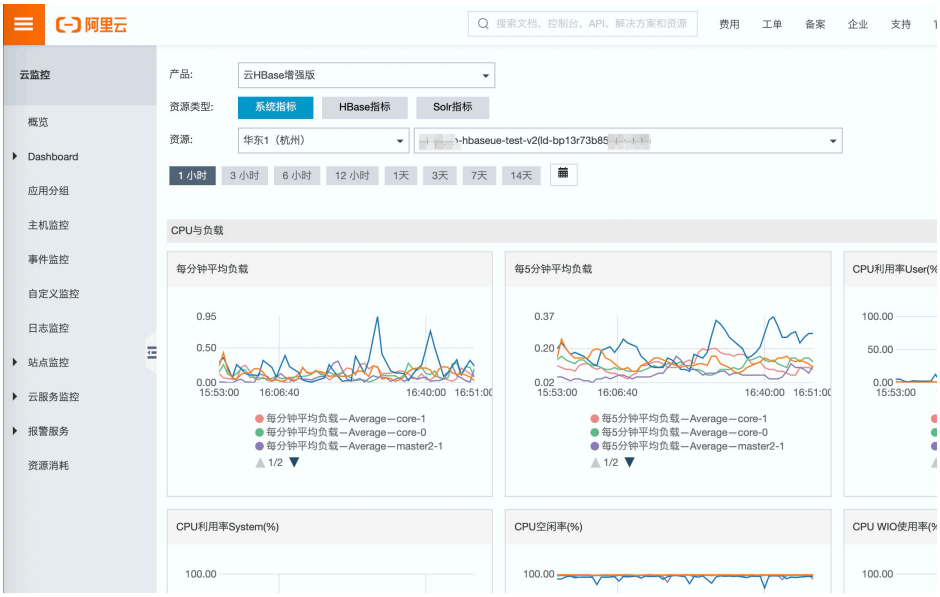
- (**推荐**)通过HBase控制台跳转到该实例的云监控页面
- 通过云监控的Dashboard按region和实例id搜索

注意：

通过HBase控制台查看监控(推荐)

（1）进入**HBase控制台**，选择您要查看监控数据的HBase增强版实例，点击实例名，进入实例的基本信息页面；点击左侧的监控与报警按钮，进入监控跳转页面，如下图所示：

（2）点击跳转至云监控，即可进入该实例的云监控页面。如下图所示：



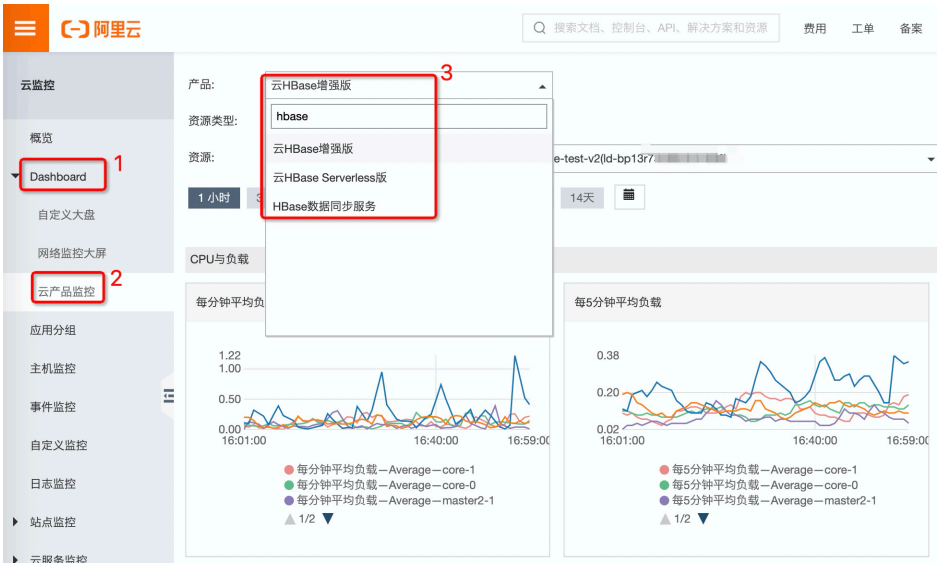
特别说明：

- HBase增强版的监控数据组织**不再**依赖云监控的应用分组特性，HBase控制台**不会**再为每个实例自动创建应用分组，也**不再**需要用户的RAM授权。如果您只看监控，不配报警，是不需要操作云监控的应用分组的
- 对于独立部署的Solr实例，可以从Solr控制台的监控报警页面跳转，也可以从其关联的HBase增强版实例进行跳转，二者均会跳转到同一个监控页面。此时，虽然Solr是独立部署的，但该实例**归属于**某个HBase增强版实例，所有的监控数据都以HBase增强版实例进行组织的
- 本方法适用于云HBase的**所有**产品形态，包括HBase标准版/HBase增强版/BDS/Solr等

通过云监控Dashboard按Region和实例Id查看监控

通过云监控的Dashboard功能，可以搜索并查看你的账号下的所有云HBase实例的监控。步骤如下：

（1）进入**云监控控制台**，点击左侧的Dashboard -> 云产品监控，并在右侧的产品列表中，输入hbase。云监控会自动列出云HBase的具体部署形态，选择HBase增强版来查看增强版的监控，如下图所示：



（2）在资源列表中，选择您的实例所在的Region和实例Id，查看该实例的监控：

通过选择region和实例id，可以查看当前账号下购买的所有HBase实例的监控。

特别说明

- 本方法不适用于HBase标准版，适用于其他所有产品形态
- 对于独立部署的Solr实例，若要使用本方法查看监控，请在实例列表中选择与Solr实例关联的HBase增强版实例的ID，而不是Solr实例的ID。

报警

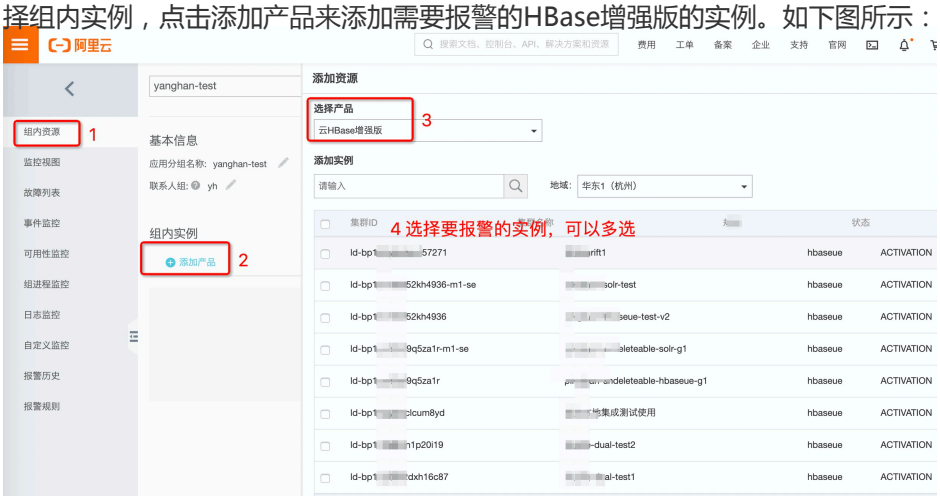
云监控提供了多种报警配置方式，这里，我们推荐使用应用分组来为HBase进行报警配置。云监控引入应用分组的概念来组织被监控的实例，并以应用分组为粒度进行报警配置。主要的操作有：创建应用分组(或使用已经存在的分组)，向分组中添加要报警的HBase实例，配置报警规则。下面我们详细介绍每一步操作：

（1）创建(或复用)应用分组

进入云监控的控制台，点击左侧应用分组，选择您要配置报警的分组。或者创建新的分组。注意，创建新分组时，不勾选初始化安装监控插件，不需要动态添加实例。

（2）向应用分组中添加需要报警的实例

准备好应用分组后，我们可以向其中添加需要报警的HBase实例。点击您的应用分组，进入分组的详情页。选择组内实例，点击添加产品来添加需要报警的HBase增强版的实例。如下图所示：



（3）为应用分组添加报警规则

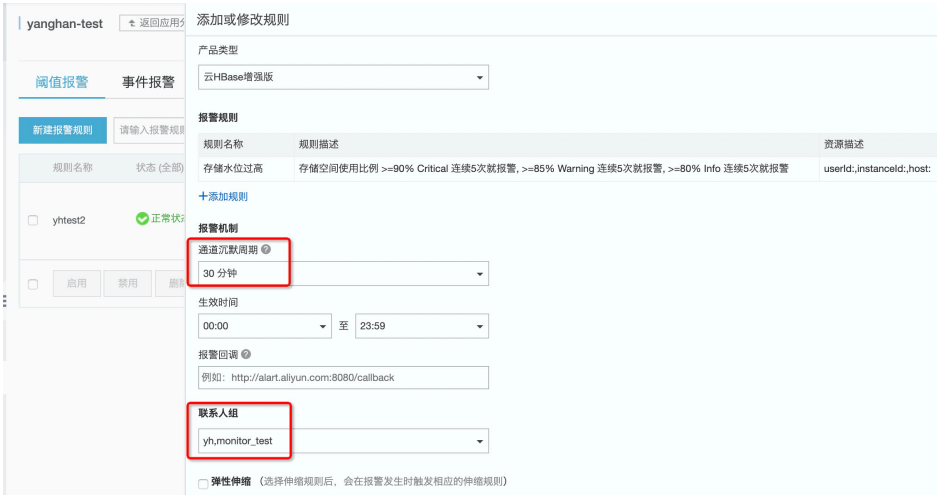
在准备好分组，以及分组中的实例后，我们可以为这个分组添加一系列的报警规则，这些规则会对分组中的所有实例生效。点击左侧的报警规则，点击新建报警规则，创建报警规则，如下图所示：

注意：报警指标的单位是预置的，不需要用户输入。比如，存储空间使用百分比，单位是%。如您想配置存储水位超过80%报警，则应在报警条件中输入80，而不是0.8。如下图所示：

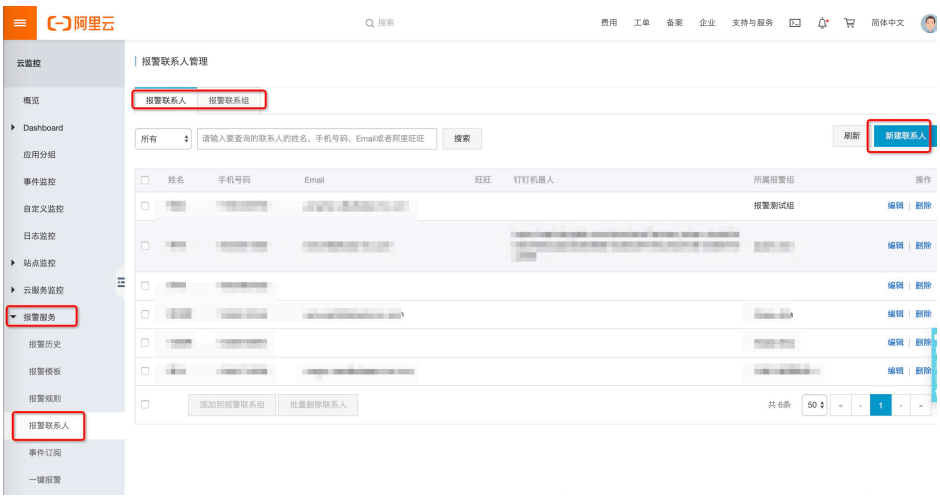
（4）配置报警的其他信息(如联系人)

配置好报警规则后，需要关注报警的另外两个信息：

- 通道沉默周期：表示连续触发报警的最小周期。如下图所示，设置为30分钟，则表示30分钟内该指标在这个应用分组内最多触发一次
- 联系人组：表示这个报警在触发后，通知给谁



如果您还未配置联系人分组，请在云监控的控制台中配置，如下图所示：



其他特别说明(重要)

关于实例存储水位报警的特别说明

HBase实例在存储使用比例达到95%时会自动锁定集群，禁止写入。如果发生禁写，会影响业务。因此，所有的HBase实例都会默认配置一个存储水位报警规则，在水位达到**80%**时报警。这个规则是对用户透明的，如果您想修改这个规则，或者禁用此规则，请钉钉联系云HBase答疑来修改。

另外，如果您通过云监控为存储使用比例显式配置了一个报警规则，系统默认的报警规则**依然有效**(**不会**被您配置的规则覆盖)。此时，如果您希望移除系统默认规则，请钉钉联系云HBase答疑进行修改。

使用限制

HBase增强版支持通过HBase原生API进行访问，但目前在以下几方面的使用存在限制：

不支持HBase的系统管理API ：HBase增强版不支持集群管理相关操作的接口，如assign region、stopRegionServer等等，用户对集群的日常管理，可以通过控制台和我们提供的集群管理系统完成。**但日常可能会用到的flush、compaction支持通过Shell或者Java API调用。**

暂不支持Coprocessor：计划后续支持，如果有该需求的客户可以钉钉询问云HBase答疑或者提工单咨询。

有限开放底层的HDFS：如果有数据导入导出需求，可以参考**导入导出指南**。如果需要使用Spark分析，可以采用spark-connector的方式访问或者直接在Spark中调用HBase API或者使用MR的TableInputFormat。**但增强版支持开放HDFS来进行bulkload操作**

不支持用户自定义的Filter：自定义Filter指的是用户将自身实现的Filter类代码封装为jar，上传到HBase的类路径下或者HDFS中，并重启HBase或者动态加载，使之生效。HBase增强版不支持此使用方式。

暂不支持Phoenix SQL: 目前HBase增强版暂不支持Phoenix

如对以上的使用限制存在疑问或需求，欢迎提工单，或者在钉钉上找“云HBase答疑”钉钉号咨询解决。

小版本升级

关于版本升级

阿里云HBase团队，会不断fix线上bug及改进性能，同时保证小版本升级是保证完全兼容的。如果版本有一些严重的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。

升级流程

用户在**业务低峰期**，在集群控制台点击升级按钮 小版本升级即可。

升级过程中HBase服务不会停止，只会逐台滚动重启升级，RegionServer重启前我们会把所有的Region全部移到其他服务器上，尽可能少地减少对业务影响。但尽管如此，业务在升级过程中可能遇到一些抖动，但整个HBase集群都会保持可用状态，用户可以放心升级。

升级时间跟随着集群的大小，region的个数多少相关，集群越大越多时间越长。一般2个Core节点，共有100个region的集群会在10分钟以内升级完成。如果集群较大，Region数目上千，升级时间超过一个小时也是正常现象。

如果在升级过程中遇到什么问题，请通过工 或者云HBase答疑 钉钉号，跟我们联系

版本更新说明

2.1.8

支持冷存储和冷热分离功能

2.1.9

支持全文索引Solr，其他一些bugfix

2.1.10

支持高性能原生二级索引,优化冷热分离功能功能

2.1.12

优化高性能原生二级索引功能，修复bulkload过程中可能遇到的错误。

2.1.15

优化全文索引Solr，以及其他一些bugfix。

2.1.17

优化全文索引Solr的使用体验。

数据通道

数据导入与迁移

HBase增强版支持从阿里云RDS，MySQL，不同HBase版本（含阿里云HBase和自建HBase集群）导入数据，并进行实时的增量同步，也支持将数据迁移到不同版本的HBase中。数据的导入和迁移，都是基于BDS服务实现。想了解BDS服务的详情，请参考使用说明。如果您有数据想从其他数据源，如自建HBase集群，云HBase集群或者RDS，MySQL等需要导入到HBase增强版，可以在钉钉上联系“云HBase答疑”寻求帮助。

RDS/MySQL历史数据导入

将RDS/MySQL的历史数据导入到HBase增强版，可以使用BDS，请参考使用BDS将RDS全量数据导入到HBase

RDS增量数据同步

通过BDS数据同步服务可以将RDS增量数据实时同步到HBase增强版中，使用说明：

1. 开通BDS服务
2. 在BDS中添加HBase增强版集群关联，具体操作方法参见添加HBase增强版数据源
3. 参考RDS实时数据同步HBase完成配置

HBase->HBase增强版 数据导入和增量同步

使用阿里云BDS数据同步服务可以将阿里云HBase标准版或开源HBase的历史和增量数据迁移到HBase增强版，详见BDS使用文档

数据导入（历史数据迁移）

1. 开通BDS服务
2. 在BDS中添加HBase集群和HBase增强版集群，具体操作方法参见添加HBase增强版数据源
3. 添加历史数据迁移任务

增量数据同步

1. 开通BDS服务
2. 在BDS中添加HBase集群和HBase增强版集群，具体操作方法参见添加HBase增强版数据源
3. 添加实时数据同步任务

HBase增强版 ->HBase 数据导入和增量同步

使用阿里云BDS数据同步服务可以将HBase增强版历史和增量数据迁移到阿里云HBase标准版或开源HBase，详见BDS使用文档

数据导入（历史数据迁移）

1. 开通BDS服务
2. 在BDS中添加HBase集群和HBase增强版集群，具体操作方法参见添加HBase增强版数据源
3. 添加历史数据迁移任务

增量数据同步

1. 开通BDS服务
2. 在BDS中添加HBase集群和HBase增强版集群，具体操作方法参见添加HBase增强版数据源

3. 添加实时数据同步任务

使用DataWorks/DataX导入数据

适用场景

云HBase提供BDS服务能够支持各种HBase版本之间相互迁移和实时同步，并且支持同步RDS、Loghub的实时数据到HBase。如果您有这方面的需求，请移步**BDS介绍**。如果您需要从BDS暂不支持的异构数据源导入数据，如从**MaxCompute（原ODPS）**导入数据到云HBase，则需要使用到DataX这个产品。DataX 是阿里巴巴集团内被广泛使用的离线数据同步工具/平台，实现包括 MySQL、Oracle、SqlServer、Postgre、HDFS、Hive、ADS、HBase、TableStore(OTS)、MaxCompute(ODPS)、DRDS 等各种异构数据源之间高效的数据同步功能。

使用Datax进行数据同步

使用Datax有两种方案：

1. 使用阿里云Dataworks的数据集成服务配置datax任务
2. 使用开源datax配置同步任务

方案一：使用Dataworks运行datax配置步骤

创建工作空间

创建工作空间的方法详见DataWorks文档

创建资源组

资源组类型	配置文档	特点	注意事项
独享资源组	配置文档	独享资源组的机器由DataWorks自动买出，运维完全托管于系统，您无需担心机器服务和可用性等问题	独享资源不支持跨地域使用。例如，华东2（上海）地域的独享资源，只能给华东2（上海）地域的工作空间使用（无法绑定其他区域的VPC），并且独享资源组不能夸Vswtich访问HBase集

			群
自定义资源组	配置文档	**仅DataWorks企业版及以上版本支持自定义资源组**。自定义资源组的ECS机器由用户自己买出，用户可以将ECS买在HBase的VPC内，从而用内网访问HBase，否则只能用外网访问	自定义资源组的机器完全可控、可登录访问，但是需要自行安装/运维/升级DataX版本(配置文档中有安装方法)
默认资源组	无	默认资源组机器无法从内网访问HBase所在VPC，只能通过**公网访问**HBase	公网访问HBase会在DataWorks产生额外费用，详见文档

我们推荐使用独享资源组和自定义资源组的方式访问HBase。两者的详细差别见文档

配置网络

独享资源组网络配置

1. 绑定专有网络(VPC)，将独享资源组与HBase所在VPC绑定
2. 在VPC控制台中找到独立资源组所绑定的VPC和Vswtich所在的网段。如下图中的192.168.0/24。由于无法知道独立资源组内机器的具体IP，所以必须将整个网段加入HBase的白名单，才可以正常访问HBase。 
3. 添加步骤2中获得的IP段到HBase访问白名单中，添加方法见文档

自定义资源组网络配置

自定义资源组的机器都是用户自己购买，因此能看到每台ECS的具体IP，将这些IP全部配置到HBase访问白名单中即可，添加方法见文档

默认资源组网络配置

默认资源组机器的IP段详见文档,将区域对应的IP添加到HBase访问白名单中即可，添加方法见文档。

创建同步任务并绑定资源组

1. 创建同步任务，具体方法参见文档
2. 修改插件配置，读写HBase分别使用hbase11xwritter，hbase11xreader插件。

相关的配置可以参考具体插件的帮助。但是hbase增强版”hbaseconfig”部分不再使用Zookeeper.quorum这个参数连接，而是使用增强版特有的endpoint形式，配置示例如下:

```
"hbaseConfig": {
"hbase.client.connection.impl" : "com.alibaba.hbase.client.AliHBaseUEConnection",
"hbase.client.endpoint" : "host:30020",
"hbase.client.username" : "root",
"hbase.client.password" : "root"
}
```

说明：

- hbase.client.connection.impl - 固定配置不需要修改
- hbase.client.endpoint - 用户控制台上提供的Java API访问地址，用户可以参考连接集群获得。
- hbase.client.username和password - HBase增强版中用户自己创建的用户名和密码（默认均为root），用户必须保证提供 的用户有读写HBase增强版中表的权限（默认提供的root用户已经具有读写所有表的权限）。关于用户和ACL，用户可以参考连接集群章节.

3.配置任务资源组为上一步创建的资源组

方案二 使用开源datax配置步骤

下载DataX安装包

点击此处直接下载集成了访问HBase增强版所需jar包的DataX安装包。下载完成后解压DataX的tar包。

如果是已有的DataX版本，或者从GitHub地址下载最新版本的安装包，则需要加入所需的jar包，加入方法如下：

从JAVA SDK安装中的下载压缩包章节下载1.x版本的alihbase-connector-1.x jar文件，注意只需要alihbase-connector-1.x jar这一个jar文件即可（1.x代表版本号，具体数字为最新的版本号），无需整个压缩包。把下载好的jar包放入datax/plugin/writer/hbase11xwriter/libs目录中。如果需要用DataX读取HBase增强版的数据，则需将此jar包也放入datax/plugin/reader/hbase11xreader/libs目录中。

编辑配置文件

在DataX中，读取HBase增强版的插件为hbase11xreader，此插件的具体配置可参见hbase11xreader的帮助文档。写入HBase增强版的插件为hbase11xwriter,此插件的具体配置可参见hbase11xwriter的帮助文档。读写HBase增强版的配置与官方的配置除了hbaseconfig部分，其他部分完全一致。hbaseconfig部分不再使用Zookeeper.quorum这个参数连接，而是使用增强版特有的endpoint形式，配置示例如下：

```
...
"hbaseConfig": {
```

```
"hbase.client.connection.impl" : "com.alibaba.hbase.client.AliHBaseUEConnection",
"hbase.client.endpoint" : "host:30020",
"hbase.client.username" : "root",
"hbase.client.password" : "root"
}
...
```

其中hbase.client.connection.impl为固定配置，用户将其设置为“com.alibaba.hbase.client.AliHBaseUEConnection”即可启用增强版的Connection。
hbase.client.endpoint即用户控制台上提供的Java API访问地址，用户可以参考连接集群获得。
hbase.client.username和password为HBase增强版中用户自己创建的用户名和密码（默认均为root），用户必须保证提供 的用户有读写HBase增强版中表的权限（默认提供的root用户已经具有读写所有表的权限）。关于用户和ACL，用户可以参考连接集群章节。

启动DataX开始迁移数据

DataX的具体使用方式大家可以参考官方文档

注意事项：

迁移开始前，请仔细阅读连接集群章节，根据DataX所部署的ECS需要添加白名单，才能正确访问HBase增强版。同时，如果ECS与HBase增强版不在一个VPC内，则需要使用公网地址访问。

通过Spark访问HBase增强版

访问准备

HBase增强版支持从Spark访问，用户只需要加入阿里云发布的HBase客户端，或者alihbase-connector的依赖即可，最新版本详见JAVA SDK安装

获取访问地址

参见连接集群，使用地址中Java API访问地址，默认端口为30020，如果是公网访问，请使用公网域名

获取用户名和密码

参见连接集群，默认的用户名为root，密码为root。或者在集群管理页面中关闭ACL功能后，无需再提供用户

名密码

添加HBase增强版访问配置

方式一：配置文件

hbase-site.xml 中增加下列配置项：

```
<configuration>
<!--
集群的连接地址，在控制台页面的数据库连接界面获得(注意公网地址和VPC内网地址)
-->
<property>
<name>hbase.zookeeper.quorum</name>
<value>ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020</value>
</property>
<!--
设置用户名密码，默认root:root，可根据实际情况调整
-->
<property>
<name>hbase.client.username</name>
<value>root</value>
</property>
<property>
<name>hbase.client.password</name>
<value>root</value>
</property>
<!--
如果您直接依赖了阿里云hbase客户端，则无需配置connection.impl参数，如果您依赖了alihbase-connector，则需要配置此参数
-->
<!--property>
<name>hbase.client.connection.impl</name>
<value>org.apache.hadoop.hbase.client.AliHBaseUEClusterConnection</value>
</property-->
</configuration>
```

方式二：代码

通过代码在Configuration中添加参数：

```
// 新建一个Configuration
Configuration conf = HBaseConfiguration.create();
// 集群的连接地址，在控制台页面的数据库连接界面获得(注意公网地址和VPC内网地址)
conf.set("hbase.zookeeper.quorum", "ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020");
// 设置用户名密码，默认root:root，可根据实际情况调整
conf.set("hbase.client.username", "root")
conf.set("hbase.client.password", "root")
```

```
// 如果您直接依赖了阿里云hbase客户端，则无需配置connection.impl参数，如果您依赖了alihbase-connector，则需要配置此参数
//conf.set("hbase.client.connection.impl", AliHBaseUEClusterConnection.class.getName());
```

Spark访问示例

```
test(" test the spark sql count result") {
//1. 添加hbase ue访问配置
var conf = HBaseConfiguration.create
conf.set("hbase.zookeeper.quorum", "ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020")
conf.set("hbase.client.username", "test_user")
conf.set("hbase.client.password", "password")

//2. 创建表
val hbaseTableName = "testTable"
val cf = "f"
val column1 = cf + ":a"
val column2 = cf + ":b"
var rowsCount: Int = -1
var namespace = "spark_test"
val admin = ConnectionFactory.createConnection(conf).getAdmin()
val tableName = TableName.valueOf(namespace, hbaseTableName)
val htd = new HTableDescriptor(tableName)
htd.addFamily(new HColumnDescriptor(cf))
admin.createTable(htd)

//3. 插入测试数据
val rng = new Random()
val k: Array[Byte] = new Array[Byte](3)
val famAndQf = KeyValue.parseColumn(Bytes.toBytes(column))
val puts = new util.ArrayList[Put]()
var i = 0
for (b1 <- ('a' to 'z')) {
for (b2 <- ('a' to 'z')) {
for (b3 <- ('a' to 'z')) {
if(i < 10) {
k(0) = b1.toByte
k(1) = b2.toByte
k(2) = b3.toByte
val put = new Put(k)
put.addColumn(famAndQf(0), famAndQf(1), ("value_" + b1 + b2 + b3).getBytes())
puts.add(put)
i = i + 1
}
}
}
}
val conn = ConnectionFactory.createConnection(conf)
val table = conn.getTable(tableName)
table.put(puts)
```

```
//4. 创建spark表
val sparkTableName = "spark_hbase"
val createCmd = s"""CREATE TABLE ${sparkTableName} USING org.apache.hadoop.hbase.spark
| OPTIONS ('catalog'=
| '{"table":{"namespace":"$${hbaseTableName}", "name":"${hbaseTableName}"},"rowkey":"rowkey",
| "columns":{
| "col0":{"cf":"rowkey", "col":"rowkey", "type":"string"},
| "col1":{"cf":"cf1", "col":"a", "type":"string"},
| "col2":{"cf":"cf1", "col":"b", "type":"String"}}}'
| )""".stripMargin

println(" createCmd: \n" + createCmd + " rows : " + rowsCount)
sparkSession.sql(createCmd)

//5. 执行count sql
val result = sparkSession.sql("select count(*) from " + sparkTableName)
val sparkCounts = result.collect().apply(0).getLong(0)
println(" sparkCounts : " + sparkCounts)
```

通过Hive访问HBase增强版

替换Hive中的hbase相关的jar包

HBase增强版支持Hive访问。但是Hive调用HBase的方式并非标准用法，而是直接调用了HBase内部类。因此无法采用直接加入alihbase-connector的jar的方式做兼容，需要替换hive/lib下已有的hbase jar包。

```
[hadoop@hadoop16 lib]$ ll |grep hbase
-rw-rw-r--  1 hadoop hadoop 26687536 Oct 19 15:4
-rw-rw-r--  1 hadoop hadoop   546195 Oct 19 15:4
-rw-r--r--  1 hadoop hadoop    20781 May 23  2017 hbase-annotations-1.0.2.jar
-rw-r--r--  1 hadoop hadoop     8870 May 23  2017 hbase-annotations-1.0.2-tests.jar
-rw-r--r--  1 hadoop hadoop     9362 May 23  2017 hbase-checkstyle-1.0.2.jar
-rw-r--r--  1 hadoop hadoop  1118767 May 23  2017 hbase-client-1.0.2.jar
-rw-r--r--  1 hadoop hadoop   512484 May 23  2017 hbase-common-1.0.2.jar
-rw-r--r--  1 hadoop hadoop   191960 May 23  2017 hbase-common-1.0.2-tests.jar
-rw-r--r--  1 hadoop hadoop   122950 May 23  2017 hbase-examples-1.0.2.jar
-rw-r--r--  1 hadoop hadoop    86957 May 23  2017 hbase-hadoop2-compat-1.0.2.jar
-rw-r--r--  1 hadoop hadoop    35461 May 23  2017 hbase-hadoop-compat-1.0.2.jar
-rw-r--r--  1 hadoop hadoop    12585 May 23  2017 hbase-it-1.0.2.jar
-rw-r--r--  1 hadoop hadoop   422209 May 23  2017 hbase-it-1.0.2-tests.jar
-rw-r--r--  1 hadoop hadoop   102077 May 23  2017 hbase-prefix-tree-1.0.2.jar
-rw-r--r--  1 hadoop hadoop  3690744 May 23  2017 hbase-protocol-1.0.2.jar
-rw-r--r--  1 hadoop hadoop    56001 May 23  2017 hbase-resource-bundle-1.0.2.jar
-rw-r--r--  1 hadoop hadoop   384728 May 23  2017 hbase-rest-1.0.2.jar
-rw-r--r--  1 hadoop hadoop  3631080 May 23  2017 hbase-server-1.0.2.jar
-rw-r--r--  1 hadoop hadoop  4712641 May 23  2017 hbase-server-1.0.2-tests.jar
-rw-r--r--  1 hadoop hadoop    12547 May 23  2017 hbase-shell-1.0.2.jar
-rw-r--r--  1 hadoop hadoop    10973 May 23  2017 hbase-testing-util-1.0.2.jar
-rw-r--r--  1 hadoop hadoop  2477029 May 23  2017 hbase-thrift-1.0.2.jar
-rw-rw-r--  1 hadoop hadoop   115935 Jun 19  2015 hive-hbase-handler-1.2.1.jar
```

1. 删除hive/lib中hbase 开头的所有的jar。如图中红框所示的所有jar文件。注意不要删除hive-hbase-handler-{version}.jar，这是Hive访问HBase的逻辑代码jar包。
2. 点此下载alihbase兼容客户端jar包文件（所有以alihbase-开头的jar包，其他的依赖视情况而定，如果原来就有。可以不放）,全部放入hive/lib目录中
3. 如果之前在hive/.hiverc中或者在启动Hive时使用--auxpath参数指定了hbase依赖的，需要把加载

的jar包换成新的alihbase开头的jar

连接前的准备

获取连接地址

参见连接集群，使用地址中Java API访问地址，如果是公网访问，请使用公网域名

获取用户名密码

参见连接集群，默认的用户名为root，密码为root。或者在集群管理页面中关闭ACL功能后，无需再提供用户名密码

将访问HBase的Hive机器IP加入HBase白名单

所有访问HBase的Hive机器的IP，必须加入HBase集群的白名单中，否则无法访问，添加白名单请参考访问白名单

在Hive中配置连接参数

在Hive中配置连接HBase的参数有两种参数，一种是直接配置在hive-site.xml文件中。在这个文件中加入如下配置项：

```
    <configuration>
<!--
集群的连接地址，在控制台页面的数据库连接界面获得(注意公网地址和VPC内网地址)
-->
<property>
<name>hbase.zookeeper.quorum</name>
<value>ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020</value>
</property>
<!--
设置用户名密码，默认root:root，可根据实际情况调整
-->
<property>
<name>hbase.client.username</name>
<value>root</value>
</property>
<property>
<name>hbase.client.password</name>
<value>root</value>
</property>
</configuration>
```

另一种方式是在Hive的Client中直接用命令的方式配置：

```
set hbase.zookeeper.quorum=ld-xxxx-proxy-hbaseue.hbaseue.xxx.rds.aliyuncs.com:30020
set hbase.client.username=root
```

```
set hbase.client.password=root
```

当完成这些配置后，就可以在Hive中自由地使用HBase外表。

Hive简单使用指南

如果HBase表不存在，可在Hive中直接创建云HBase关联表

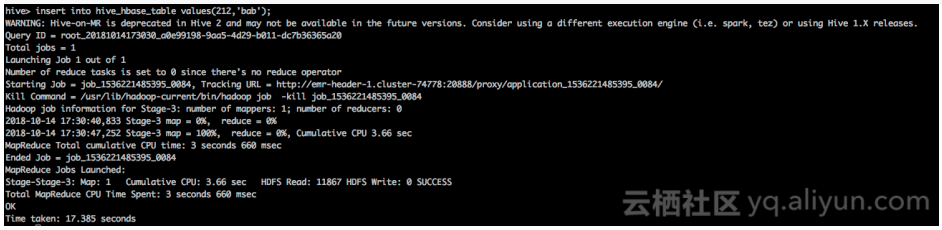
- 进入hive cli命令行

- 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

```
hbase(main):005:0> put 'hive_hbase_table','132','cf1:val','acb'
0 row(s) in 0.0430 seconds
```

在Hive中查看：

```
hive> select * from hive_hbase_table;
OK
132     acb
212     bab
Time taken: 0.273 seconds, Fetched: 2 row(s)
```

Hive删除表，HBase表也删除

```
hive>
    >
    >
    > drop table hive_hbase_table;
OK
Time taken: 6.307 seconds
```

查看hbase表，报错不存在表

```
hbase(main):008:0* scan 'hive_hbase_table'
ROW                                                    COLUMN+CELL

ERROR: Unknown table hive_hbase_table!
```

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>
```


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

注意：目前暂不支持Hive关联HBase增强版的Snapshot来直接读取HFile文件。如果是采用关联/创建外表的方式读写是完全兼容的。

通过实时计算(Flink)访问HBase增强版

云HBase增强版支持阿里云实时计算服务（Apache Flink）访问。用户可以把HBase中的表当做Flink 中的维表或者结果表。具体的使用方法参见阿里云实时计算服务帮助中的创建云数据库 HBase版结果表和创建云数据库 HBase版维表。

在使用HBase增强版的表作为Flink的维表或者结果表时，创建表的DDL语句时，需要使用增强版的连接地址。增强版的访问地址请参见连接集群章节。Flink访问HBase增强版使用的是Java API访问地址。用户名密码默认为root，root。如果使用新创建的用户，请确保该用户拥有访问与Flink关联的表的读写权限，详见用户和ACL管理。示例的DDL语句如下：

创建维表

```
CREATE TABLE hbase (
`key` varchar,
`name` varchar,
PRIMARY KEY (`key`), -- HBase中的rowkey字段。
PERIOD FOR SYSTEM_TIME --维表标识。
) with (
TYPE = 'cloudhbase',
endpoint = 'host:port',-- HBase增强版的Java API访问地址
columnFamily = 'xxxxxx',
userName = 'root', -- 用户名
password = 'root', -- 密码
```

```
tableName = 'xxxxxx'
);
```

创建结果表

```
create table liuxd_user_behavior_test_front (
row_key varchar,
from_topic varchar,
origin_data varchar,
record_create_time varchar,
primary key (row_key)
) with (
type = 'cloudhbase',
endpoint = 'host:port', -- HBase增强版的Java API访问地址
userName = 'root', -- 用户名
password = 'root', -- 密码
columnFamily = '<yourColumnFamily>',
tableName = '<yourTableName>',
batchSize = '500'
)
```

网络打通

如果使用共享版的Flink集群，请参照阿里云实时计算的帮助文档VPC访问授权进行VPC的网络打通。如果使用独立的Flink集群，请确保Flink集群和HBase集群在同一个VPC下。否则只能用公网访问地址链接HBase（参见连接集群)。无论是哪种链接方式，都需要确保Flink的机器的IP加到HBase访问白名单中

企业特性

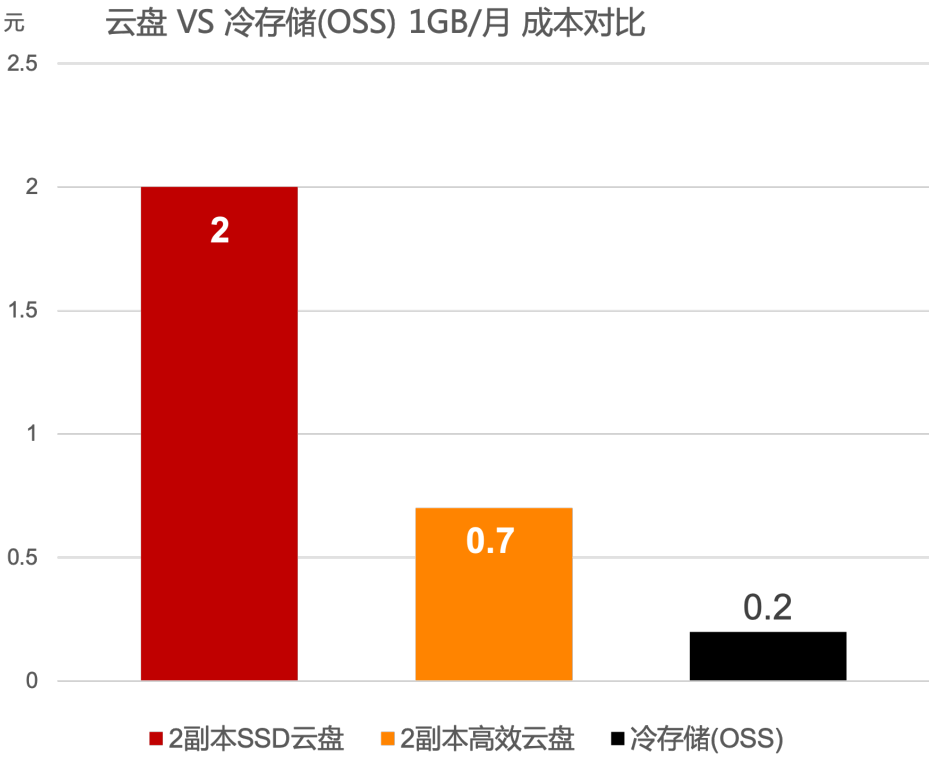
使用冷存储

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。

同时HBase增强版还基于冷存储实现了在同一张表内的冷热分离功能，能够自动将表中将热数据放到读写速度快的热存储中，而把不常访问的数据放到冷存储中降低成本，如果用户有冷热分离的需求，可以移步冷热分离



介绍章节

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。

创建新的HBase增强版实例时，可在购买页面选择是否选购冷存储和冷存储的容量。

如果您在创建实例时没有选择开通冷存储，也没有关系。在集群控制台上，找到冷存Tab，点击开通。

注意：只有HBase增强版

2.1.8版本以上才支持冷存储，如果低于此版本在开通过程中会自动升级到最新版本

使用冷存储

注意：冷存储功能需要HBase增强版服务端升级到2.1.8版本以上，客户端依赖要求AliHBase-Connector 1.0.7/2.0.7以上，Shell要求alihbase-2.0.7-bin.tar.gz以上

HBase增强版支持在ColumnFamily（列簇）级别设置存储属性。可以将表的某个列簇（或者所有列簇）的Storage设为冷存储。一旦设置为冷存储后，那么这个表中该列簇（或者所有列簇）的数据，都会存储在冷存储中，并不会占用该集群的HDFS空间。设置的方法可以在建表时指定，也可以在建好表后，对列簇的属性进行修改。

建表和修改表属性均可以使用Java API和HBase shell完成，在使用Java API前请 按照HBase Java API 访问文档中的步骤完成Java SDK安装和参数配置。在使用HBase shell前，请按照HBase Shell访问文档中的步骤完成Shell的下载和配置。

建表时指定冷存储

HBase Shell

```
hbase(main):001:0> create 'coldTable', {NAME => 'f', STORAGE_POLICY => 'COLD'}
```

Java API

```
 Admin admin = connection.getAdmin();
HTableDescriptor descriptor = new HTableDescriptor(TableName.valueOf("coldTable"));
HColumnDescriptor cf = new HColumnDescriptor("f");
cf.setValue("STORAGE_POLICY", AliHBaseConstants.STORAGETYPE_COLD);
descriptor.addFamily(cf);
admin.createTable(descriptor);
```

修改表属性指定冷存储

如果表已经建立后，可以通过修改表中列簇的属性来设置冷存储的列簇。**如果这个列簇中已经有数据，那么只有在major compaction之后，数据才会进入到冷存储**

HBase Shell

```
hbase(main):011:0> alter 'coldTable', {NAME=>'f', STORAGE_POLICY => 'COLD'}
```

Java API

```
Admin admin = connection.getAdmin();
TableName tableName = TableName.valueOf("coldTable");
```

```
HTableDescriptor descriptor = admin.getTableDescriptor(tableName);
HColumnDescriptor cf = descriptor.getFamily("f".getBytes());
// 设置表的存储类型为冷存储
cf.setValue("STORAGE_POLICY", AliHBaseConstants.STORAGETYPE_COLD);
admin.modifyTable(tableName, descriptor);
```

将表属性改回热存储

如果表的列存存储类型为冷存储，想改回到热存储，可以通过修改表属性的方式实现。**如果这个列簇中已经有数据，那么只有在major compaction之后，数据才会回到热存储中**

HBase Shell

```
hbase(main):014:0> alter 'coldTable', {NAME=>'f', STORAGE_POLICY => 'DEFAULT'}
```

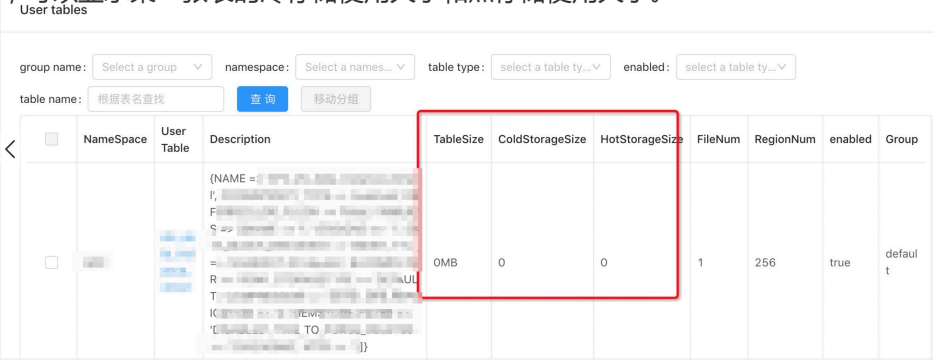
Java API

```
// 参见创建连接: https://help.aliyun.com/document_detail/119570.html
Admin admin = connection.getAdmin();
TableName tableName = TableName.valueOf("coldTable");
HTableDescriptor descriptor = admin.getTableDescriptor(tableName);
HColumnDescriptor cf = descriptor.getFamily("f".getBytes());
// 设置表的存储类型为默认存储，默认存储为热存储
cf.setValue("STORAGE_POLICY", AliHBaseConstants.STORAGETYPE_DEFAULT);
admin.modifyTable(tableName, descriptor);
```

查看冷存储使用情况

在控制台的冷存储界面，可以查看整体的冷存储使用状况，并可以扩容冷存储。

在集群管理系统的表Tab中，可以显示某一张表的冷存储使用大小和热存储使用大小。



性能测试

环境说明

Master: ecs.c5.xlarge, 4core 8G, 20G高效云盘*4RegionServer: ecs.c5.xlarge, 4core 8G, 20G高效云盘*4测试机器：ecs.c5.xlarge, 4core 8G

写性能

表类型	avg rt	p99 rt
热表	1736 us	4811 us
冷表	1748 us	5243 us

说明: 每条记录10列，每列100B，也就是单行1k， 16线程写

随机Get性能

表类型	avg rt	p99 rt
热表	1704 us	5923 us
冷表	14738 us	31519 us

说明: 关闭表的BlockCache，完全读盘。每条记录10列，每列100B，也就是单行1k。8线程读，每次读出1k。

范围Scan性能

表类型	avg rt	p99 rt
热表	6222 us	20975 us
冷表	51134 us	115967 us

说明: 关闭表的BlockCache，每条记录10列，每列100B，也就是单行1k。8线程读，每次读出1k。Scan的Caching设为30

注意事项

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

2.写入吞吐上，冷存储和基于高效云盘的热存储相当，可以放心写入数据。

3.冷存储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。

冷热分离

背景

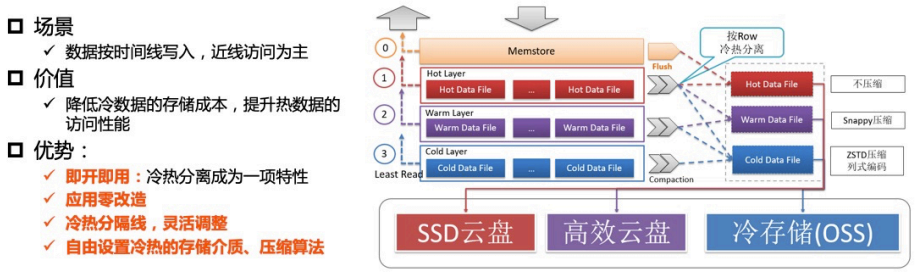
在海量大数据场景下，一张表中的部分业务数据随着时间的推移仅作为归档数据或者访问频率很低，同时这部分历史数据体量非常大，比如订单数据或者监控数据，降低这部分数据的存储成本将会极大的节省企业的成本。如何以极简的运维配置成本就能为企业极大降低存储成本，阿里云HBase增强版冷热分离功能应运而生。阿里云HBase增强版为冷数据提供新的存储介质，新的存储介质存储成本仅为高效云盘的1/3。

HBase增强版在同一张表里实现了数据的冷热分离，系统会自动根据用户设置的冷热分界线自动将表中的冷数据归档到冷存储中。在用户的访问方式上和普通表几乎没有任何差异，在查询的过程中，用户只需配置查询Hint或者TimeRange，系统根据条件自动地判断查询应该落在热数据区还是冷数据区。对用户而言始终是一张表，对用户几乎做到完全的透明。

详细介绍请参考云栖社区文章——《面向海量数据的极致成本优化-云HBase的一体化冷热分离》

原理介绍

用户在表上配置数据冷热时间分界点，HBase增强版依赖用户写入数据的时间戳(毫秒)和时间分界点来判断数据的冷热。数据最初始在热存储上，随意时间的推移慢慢往冷数据迁移。同时用户可以**任意变更数据的冷热分界点**，数据可以从热到冷，也可以从冷到热。



使用方法

注意：冷存储功能需要HBase增强版服务端升级到2.1.8版本以上，客户端依赖要求AliHBase-Connector 1.0.7/2.0.7以上，Shell要求alihbase-2.0.7-bin.tar.gz以上

在使用Java API前请 按照HBase Java API 访问文档中的步骤完成Java SDK安装和参数配置。

在使用HBase shell前，请按照HBase Shell访问文档中的步骤完成Shell的下载和配置。

开通冷存储功能

请参照冷存储中的指导开通集群的冷存储功能

为表设置冷热分界线

用户在使用过程中可以随时调整COLD_BOUNDARY来划分冷热的边界。COLD_BOUNDARY的单位为秒，如COLD_BOUNDARY => 86400 代表 86400秒（一天）前写入的数据会被自动归档到冷存储介质上。

在冷热分离使用过程中，无需把列簇的属性设置为COLD，如果已经把列簇的属性设置为了COLD，请依照冷存储文档中的指导将冷存储的属性去除

Shell

```
// 创建冷热分离表
hbase(main):002:0> create 'chsTable', {NAME=>'f', COLD_BOUNDARY=>'86400'}
// 取消冷热分离
hbase(main):004:0> alter 'chsTable', {NAME=>'f', COLD_BOUNDARY=>""}
// 为已经存在的表设置冷热分离,或者修改冷热分离分界线，单位为秒
hbase(main):005:0> alter 'chsTable', {NAME=>'f', COLD_BOUNDARY=>'86400'}
```

Java API方式

```
// 新建冷热分离表
Admin admin = connection.getAdmin();
TableName tableName = TableName.valueOf("chsTable");
HTableDescriptor descriptor = new HTableDescriptor(tableName);
HColumnDescriptor cf = new HColumnDescriptor("f");
// COLD_BOUNDARY 设置冷热分离时间分界点，单位为秒, 示例表示1天之前的数据归档为冷数据
cf.setValue(AliHBaseConstants.COLD_BOUNDARY, "86400");
descriptor.addFamily(cf);
admin.createTable(descriptor);

// 取消冷热分离
// 注意：需要做major compaction，数据才能从冷存储上回到热存储上
HTableDescriptor descriptor = admin
.getTableDescriptor(tableName);
HColumnDescriptor cf = descriptor.getFamily("f".getBytes());
// 取消冷热分离
cf.setValue(AliHBaseConstants.COLD_BOUNDARY, null);
admin.modifyTable(tableName, descriptor);

// 为已经存在的表设置冷热分离功能，或者修改冷热分离分界线
HTableDescriptor descriptor = admin
.getTableDescriptor(tableName);
HColumnDescriptor cf = descriptor.getFamily("f".getBytes());
```

```
// COLD_BOUNDARY 设置冷热分离时间分界点，单位为秒, 示例表示1天之前的数据归档为冷数据
cf.setValue(AliHBaseConstants.COLD_BOUNDARY, "86400");
admin.modifyTable(tableName, descriptor);
```

数据写入

冷热分离的表与普通表的数据写入方式完全一致，用户可以参照HBase Java API 访问文档中的方式或者使用非Java语言对表进行数据写入。数据的写入的时间戳使用的是当前时间。数据先会存储在热存储（云盘）中。随着时间的推移，如果这行数据的写入时间超过COLD_BOUNDARY设置的值，就会在major_compact时归档到冷数据，此过程完全对用户透明。

数据查询

由于冷热数据都在同一张表中，用户全程只需要和一张表交互。在查询过程中，如果用户明确知道需要查询的数据在热数据里（写入时间少于COLD_BOUNDARY设置的值），可以在Get或者Scan上设置HOT_ONLY的Hint来告诉服务器只查询热区数据。或者在Get/Scan上设置TimeRange来限定查询数据的时间，系统会根据设置TimeRange决定是查询热区，冷区还是冷热都查。查询冷区数据延迟要比热区数据延迟高的多，并且**查询吞吐受到冷存储限制**（详见冷存储文档）。

查询示例

Get

Shell

```
// 不带HotOnly Hint的查询，可能会查询到冷数据
hbase(main):013:0> get 'chsTable', 'row1'
// 带HotOnly Hint的查询，只会查热数据部分，如row1是在冷存储中，该查询会没有结果
hbase(main):015:0> get 'chsTable', 'row1', {HOT_ONLY=>true}
// 带TimeRange的查询，系统会根据设置的TimeRange与COLD_BOUNDARY冷热分界线进行比较来决定查询哪个区域的数据（注意TimeRange的单位为毫秒时间戳）
hbase(main):016:0> get 'chsTable', 'row1', {TIMERANGE => [0, 1568203111265]}
```

Java

```
Table table = connection.getTable("chsTable");
// 不带HotOnly Hint的查询，可能会查询到冷数据
Get get = new Get("row1".getBytes());
System.out.println("result: " + table.get(get));
// 带HotOnly Hint的查询，只会查热数据部分，如row1是在冷存储中，该查询会没有结果
get = new Get("row1".getBytes());
get.setAttribute(AliHBaseConstants.HOT_ONLY, Bytes.toBytes(true));
// 带TimeRange的查询，系统会根据设置的TimeRange与COLD_BOUNDARY冷热分界线进行比较来决定查询哪个区域的数据（注意TimeRange的单位为毫秒时间戳）
get = new Get("row1".getBytes());
get.setTimeRange(0, 1568203111265)
```

Scan

注意：如果scan不设置Hot Only，或者TimeRange包含冷区时间，则会并行访问冷数据和热数据来合并结果，这是由于HBase的Scan原理决定的

Shell

```
// 不带HotOnly Hint的查询，一定会查询到冷数据
hbase(main):017:0> scan 'chsTable'， {STARTROW =>'row1', STOPROW=>'row9'}
// 带HotOnly Hint的查询，只会查询热数据部分
hbase(main):018:0> scan 'chsTable'， {STARTROW =>'row1', STOPROW=>'row9'， HOT_ONLY=>true}
// 带TimeRange的查询，系统会根据设置的TimeRange与COLD_BOUNDARY冷热分界线进行比较来决定查询哪个区域的数据（注意TimeRange的单位为毫秒时间戳）
hbase(main):019:0> scan 'chsTable'， {STARTROW =>'row1', STOPROW=>'row9'， TIMERANGE => [0, 1568203111265]}
```

Java

```
TableName tableName = TableName.valueOf("chsTable");
Table table = connection.getTable(tableName);
// 不带HotOnly Hint的查询，一定会查询到冷数据
Scan scan = new Scan();
ResultScanner scanner = table.getScanner(scan);
for (Result result : scanner) {
System.out.println("scan result:" + result);
}
// 带HotOnly Hint的查询，只会查询热数据部分
scan = new Scan();
scan.setAttribute(AliHBaseConstants.HOT_ONLY, Bytes.toBytes(true));
// 带TimeRange的查询，系统会根据设置的TimeRange与COLD_BOUNDARY冷热分界线进行比较来决定查询哪个区域的数据（注意TimeRange的单位为毫秒时间戳）
scan = new Scan();
scan.setTimeRange(0, 1568203111265);
```

注意：

1. 冷热分离表中的冷区只是用来归档数据，查询请求应该非常的少，用户查询冷热分离表的绝大部分请求应该带上HOT_ONLY的标记（或者设置的TimeRange只在热区）。如果用户有大量请求需要去查冷区数据，则可能得考虑COLD_BOUNDARY冷热分界线的设置是否合理。

2. 如果一行数据已经在冷数据区域，但这一行后续有更新，更新的字段先会在热区，如果设置HOT_ONLY去查询这一行（或者设置的TimeRange只在热区），则只会返回这一行更新的字段（在热区）。只有在查询时去掉HOT_ONLY Hint，去掉TimeRange，或保证TimeRange覆盖了该行数据插入和更新时间，才能完整返回这一行。因此不建议对已经进入冷区的数据进行更新，如果有频繁更新冷数据的需求，则可能得考虑COLD_BOUNDARY冷热分界线的设置是否合理。

查看表中冷数据和热数据的大小

在集群管理系统的表Tab中，可以显示某一张表的冷存储使用大小和热存储使用大小。**如果数据还没有进入冷**

存储，有可能数据还在内存中，请执行flush，将数据刷写到盘上，再请执行major_compact完成后再查看

进阶功能

优先查询热数据

在范围查询（Scan）场景下，查询的数据可能横跨冷热区，比如查询一个用户的所有订单、聊天记录等。但查询的展示往往是从新到旧的分页展示，最先展示的往往是最近的热数据。在这个场景下，普通的Scan（不带Hot_Only）会并行地扫描冷热数据，导致请求性能下降。而在开启了优先查询热数据后，会优先只查热数据，只有热数据的条数不够显示（如用户点了下一页查看），才会去查询冷数据，减少冷存储的访问，提升请求响应。

开启热数据优先查询，只需在Scan上设置COLD_HOT_MERGE属性即可。该属性的含义是优先查询热存储中的数据, 若热存储中的数据查完了，用户仍然在调用next获取下一条数据，则会开始查询冷数据。

Shell

```
hbase(main):002:0> scan 'chsTable', {COLD_HOT_MERGE=>true}
```

Java

```
scan = new Scan();
scan.setAttribute(AliHBaseConstants.COLD_HOT_MERGE, Bytes.toBytes(true));
scanner = table.getScanner(scan);
```

注意

1. 若某一行同时包含热数据和冷数据(部分列属于热数据，部分列属于冷数据，比如部分列更新场景)，开启热数据优先功能，会使得该行的查询结果会分两次返回，即scanner返回的Result集合中，对于同一个Rowkey会有两个对应的Result。

2. 由于是先返回热数据，再返回冷数据，开启热数据优先功能后，无法保证后返回的冷数据结果的Rowkey一定大于先返回的热数据结果的Rowkey，即Scan得到的Result集不保序， 但热数据和冷数据的各自返回集仍保证按Rowkey有序(参见下面的demo)。在部分实际场景中， 用户可以通过Rowkey设计，保障scan的结果仍然保序，比如订单记录表，Rowkey=用户ID+订单创建时间，扫描某个用户的订单数据是有序的

```
//假设rowkey为"coldRow"的这一行是冷数据，rowkey为"hotRow"的这一行为热数据
//正常情况下，由于hbase的row是字典序排列，rowkey为"coldRow"的这一行会比"hotRow"这一行先返回。
hbase(main):001:0> scan 'chsTable'
ROW COLUMN+CELL
coldRow column=f:value, timestamp=1560578400000, value=cold_value
hotRow column=f:value, timestamp=1565848800000, value=hot_value
2 row(s)

// 设置COLD_HOT_MERGE时, scan的rowkey顺序被破坏，热数据比冷数据先返回，因此返回的结果中，"hot"排在了"cold"的前面
hbase(main):002:0> scan 'chsTable', {COLD_HOT_MERGE=>true}
ROW COLUMN+CELL
hotRow column=f:value, timestamp=1565848800000, value=hot_value
coldRow column=f:value, timestamp=1560578400000, value=cold_value
2 row(s)
```

根据Rowkey中的字段划分数据的冷热

除了通过写入KV的时间戳区分冷热，HBase增强版还支持通过Rowkey中的某些字段来区分数据的冷热，如用户的Rowkey中包含一个timestamp，增强版支持parse这个timestamp来划分这行数据的冷热，而不是根据KeyValue的写入时间戳。如果根据KV写入时间戳划分冷热无法满足用户需求，欢迎提工单或者钉钉上找云HBase答疑来询问使用方法。

注意事项

参见冷存储的使用注意事项。

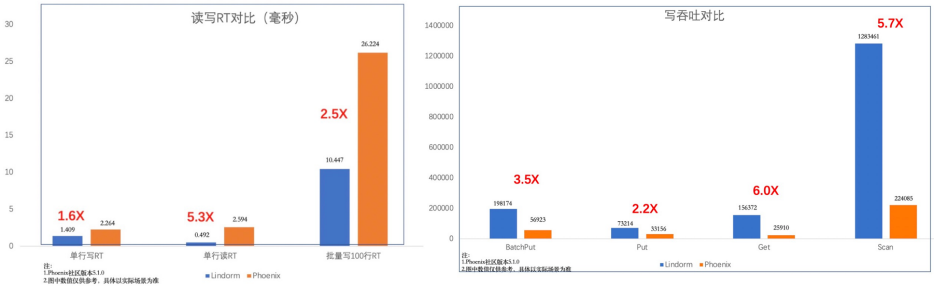
高性能原生二级索引

二级索引简介

HBase原生提供了主键索引，即按rowkey的二进制排序的索引。Scan可基于此rowkey索引高效的执行整行匹配、前缀匹配、范围查询等操作。但若需要使用rowkey之外的列进行查询，则只能使用filter在指定的rowkey范围内进行逐行过滤。若无法指定rowkey范围，则需进行全表扫描，不仅浪费大量资源，查询RT也无法保证。

有多种解决方案可解决HBase的多维查询的问题。比如以要查询的列再单独写一张表(用户自己维护二级索引)，或者将数据导出到Solr或者ES这样的外部系统进行索引。像Solr/ES这样的搜索引擎类产品，提供了强大的ad hoc查询能力，云HBase现已集成了全文索引服务。这样，可以节省掉这些外部系统的部署和运维成本。

Solr/ES固然强大，但对于大部分列较少且有固定查询模式的场景来说，有”杀鸡用牛刀”之感。为此，HBase增强版推出了原生的全局二级索引解决方案，以更低的成本解决此类问题。因内置于HBase，提供了强大的吞吐与性能。这个索引方案在阿里内部使用多年，经历了多次双11考验，尤其适合解决海量数据的全局索引场景。下图给出了HBase增强版与Phoenix在索引场景下的性能对比：



下面，我们先介绍索引的两个重要概念，然后介绍HBase增强版二级索引的DDL和DML操作，讨论一些高级主题，如rowkey的二进制排序问题以及查询优化的问题，最后，给出使用约束和FAQ。

基本概念

考虑如下主表和索引表：

```
create table 'dt' (rowkey varchar, c1 varchar, c2 varchar, c3 varchar, c4 varchar, c5 varchar constraint pk primary key(rowkey));
create index 'idx1' on 'dt' (c1);
create index 'idx2' on 'dt' (c2, c3, c4);
create index 'idx3' on 'dt' (c3) include (c1, c2, c4);
create index 'idx4' on 'dt' (c5) include (ALL);
```

索引列

索引表的主键列就是其索引列。比如idx1的c1，idx2的c2,c3,c4。索引列及其顺序决定了索引表能支持的查询场景。只有一个索引列的索引表称单列索引，有多个索引列的称组合索引。关于查询如何命中索引表，以及命中哪个索引表，可以参见文末的查询优化一节。

冗余列

如果查询中所需要的列在索引表里没有，则需要回查主表才能完成查询。在分布式场景下，回查主表可能带来额外的多次RPC，导致查询RT大幅度增加。因此，通过空间换时间的策略，将主表中的列冗余在索引表中，来避免命中索引的查询再回查主表。有冗余列的索引叫冗余索引(如idx3和idx4)，或覆盖索引(Covered Index)。

考虑到业务变化，可能会增加新列，因此，HBase增强版提供了’冗余所有列’的语义(即idx4)。索引表会自动冗余主表的所有列，从容应对业务变化。

使用前准备

- 服务器版本要求：2.1.10及以上，如果在此版本之下的集群，请在控制台上点击小版本升级
- 客户端版本要求： 需要alihbase-client 1.1.9/2.0.4以上，或者alihbase-connector1.0.9/2.09以上，详见Java SDK安装
- Shell版本要求： 需要 alihbase-2.0.9-bin.tar.gz以上版本，请访问HBase Shell下载最新版增强版Shell。

管理索引(DDL)

可通过HBase shell和Java API进行索引DDL操作。本节介绍如何使用HBase shell来进行索引DDL操作，关于Java API的使用，可以参见AliHBaseUEAdmin的相关接口注释。

下面的shell命令展示了几个常用的DDL操作：

```
# 创建索引
# 为主表dt创建索引idx1: create index idx1 on dt ('f1:c2', 'f1:c3');
# 冗余主表中的所有列，注意 COVERED_ALL_COLUMNS 是一个关键字，请不要在列名中使用它
hbase(main):002:0> create_index 'idx1', 'dt', {INDEXED_COLUMNS => ['f1:c2', 'f1:c3']}, {COVERED_COLUMNS =>
['COVERED_ALL_COLUMNS']}

# 查看索引schema
hbase(main):002:0>describe_index 'dt'

# 禁用dt的idx1索引，此时，更新dt不会再更新idx1
hbase(main):002:0>offline_index 'idx1', 'dt'

# 删除idx1这个表
hbase(main):002:0>remove_index 'idx1', 'dt'
```

下面进行详细介绍。

创建索引

```
hbase(main):002:0>create_index 'idx1', 'dt', {INDEXED_COLUMNS => ['f1:c2', 'f2:c3']}
```

为主表dt创建索引idx1，索引列有2个，f1列族下的c2列，f2列族下的c3列。没有冗余列。

```
hbase(main):002:0>create_index 'idx2', 'dt', {INDEXED_COLUMNS => ['f1:c1']}, {COVERED_COLUMNS => ['f2:c2']}
```

为主表dt创建索引idx1，索引列有2个，f1列族下的c1列，冗余f2列族下的c2列。

```
hbase(main):002:0>create_index 'idx3', 'dt', {INDEXED_COLUMNS => ['f1:c3']}, {COVERED_COLUMNS =>
['COVERED_ALL_COLUMNS']}
```

idx3会冗余dt的**所有列**。因此，命中idx3的查询一定不会回查主表。注意COVERED_ALL_COLUMNS是一个关键字，表示此索引表会冗余主表的所有列。因此，不要使用这个名字作为列名。

除设定索引表的schema(索引列/冗余列)之外，也支持设置索引表的存储特性，例如：

```
hbase(main):002:0>create_index 'idx1', 'dt', {INDEXED_COLUMNS => ['f1:c1', 'f2:c2']}, {DATA_BLOCK_ENCODING =>
'DIFF', BLOOMFILTER => 'ROW',COMPRESSION => 'LZO'}
```

查看索引schema

通过list命令可以查看主表及其所有的索引表，例如：

```
hbase(main):023:0> list
TABLE
dt
dt.idx1
dt.idx2
yh1
4 row(s)
Took 0.0129 seconds
=> ["dt", "dt.idx1", "dt.idx2", "yh1"]
```

通过describe_index命令可以查看指定主表的所有索引表的schema信息，例如：

```
hbase(main):024:0> describe_index 'dt'
Index [idx2] on [dt] (f1.c3 ASC,f2.c4 ASC)
Index [idx1] on [dt] (f1.c1 ASC,f2.c2 ASC) includes (@@ALL@@)
Total 2 indexes found for table dt
```

删除索引

与普通的HBase表一样，删除索引也需要先禁用(offline_index)，再删除(remove_index)，例如：

```
# 禁用dt的idx1索引，此时，更新dt不会再更新idx1
hbase(main):002:0>offline_index 'idx1', 'dt'
# 删除idx1这个表
hbase(main):002:0>remove_index 'idx1', 'dt'
```

注意，**不能使用disable命令来禁用索引。如果索引表offline，所有的查询都不会走这张索引表**

为有数据的表建索引时，历史数据同步问题

在为一张已经有数据的表添加新的索引时，create_index命令会**同时**将主表的历史数据同步到索引表中。因此

，当主表很大时，create_index会非常耗时。注意：create_index的数据同步任务是在服务端执行的，即使杀掉hbase shell进程，数据同步任务也会继续执行下去，直到任务完成。

未来我们会开放异步构建索引的能力，即create_index时不同步历史数据，而是用户显式执行一个命令来触发后台的数据同步流程，并通过检查索引状态是否为active来判断数据同步是否完成。

访问索引(DML)

本节介绍通过java API来访问索引(DML)。HBase Java API的基础使用，连接的创建请参见HBase Java API 访问文档。

数据写入

用户不需要主动向索引表写入数据。写主表时，HBase会自动将变更同步到其所有的索引表中。HBase增强版提供**同步更新**语义，即：写主表时会**同步**更新其**所有**索引表，待主表和索引表都写成功后，写操作才会返回到客户端。可以从以下两个角度来理解：

- 强一致：写主表成功后，本次更新可以立即被读到。
- 写进行中或超时：主表和索引表的一致性未决，但保证最终一致，即要么都更新，要么都不更新。

数据查询

与关系型数据库类似，用户不需要直接发起针对索引表的查询，只需按业务要求表达对主表的查询。HBase增强版会根据索引表的schema和查询模式自动选择最合适的索引表进行查询。通过Filter来描述基于非rowkey的查询条件，例如：

```
byte[] f = Bytes.toBytes("f");
byte[] c1 = Bytes.toBytes("c1");
byte[] value = Bytes.toBytes("yourExpectedValue");

// 等价于 select * from dt where f.c1 == value
Scan scan = new Scan();
SingleColumnValueFilter filter = new SingleColumnValueFilter(f, c1, EQUAL, value);
scan.setFilter(filter);
```

注意：

- **如果使用LESS，GREATER等条件，需要注意数据的排序问题，详情请见进阶功能中的有符号数的排序问题一节**
- 如果查询可以命中索引，系统会自动将setFilterIfMissing设置为true。如果查询无法命中索引，则会使用scan里配置的值

通过使用FilterList，可以实现and和or条件的组合与嵌套，以表达更复杂的查询条件，例如：

```
// 等价于 where f.c1 >= value1 and f.c1 < value2
```

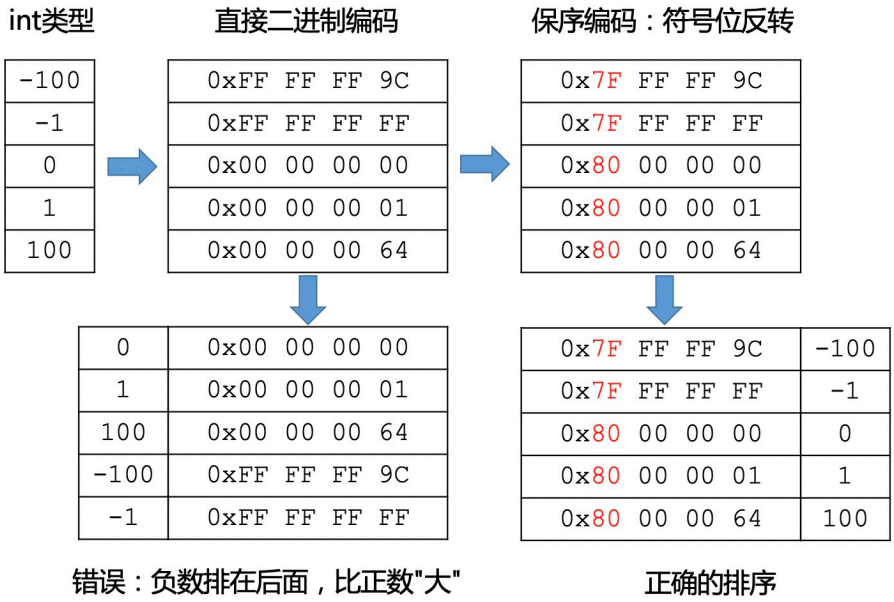
```
FilterList filters = new FilterList(FilterList.Operator.MUST_PASS_ALL);
filters.addFilter(new SingleColumnValueFilter(f, c1, GREATER_OR_EQUAL, value1));
filters.addFilter(new SingleColumnValueFilter(f, c1, LESS, value2));
```

HBase增强版会根据Filter以及索引的schema自动选择合适的索引表进行查询。具体请参见下文的查询优化一节。

进阶功能

有符号数的排序问题

HBase原生API只有一种数据类型byte[]，并以此进行二进制排序。因此，所有业务上使用的类型都需要转换为byte[]。这就涉及到数据的原始排序与转换成byte[]之后的排序问题。我们希望，在转换前后数据的排序保持不变。HBase client的Bytes类提供了各种类型与byte[]之间的相互转换，但这些接口仅适用于0和正数，而不适用于负数。下图以int类型为例描述了这个问题：



上图中，直接二进制编码就是Bytes.toBytes(int)。可见，在有负数参与的情况下，转换为byte[]之后的int不再保序。该问题可通过符号位反转来解决。对于byte，short，long，float等类型，都有此类问题。因此，HBase增强版提供了新的工具类org.apache.hadoop.hbase.util.OrderedBytes(依赖了alihbase-client或者alihbase-connector都可以找到该类)来解决这个问题。下面举例说明其用法：

```
// int转换为保序的byte[]
int x = 5;
byte[] bytes = OrderedBytes.toBytes(x);

// byte[]转换为int
int y = OrderedBytes.toInt(bytes);
```

更多用法可参见类注释。

查询优化

本节讨论HBase增强版如何根据查询进行索引选择。概括的说，就是前缀匹配。这是一种RBO(Rule Based Optimization)策略。根据查询条件中以and连接的等值比较的条件与匹配索引表的前缀，选择匹配程度最高的索引表作为本次查询使用的索引。下面我们通过一些示例来理解这一规则。

假设有如下主表和索引表：

```
create table 'dt' (rowkey varchar, c1 varchar, c2 varchar, c3 varchar, c4 varchar, c5 varchar constraint pk primary key(rowkey));
create index 'idx1' on 'dt' (c1);
create index 'idx2' on 'dt' (c2, c3, c4);
create index 'idx3' on 'dt' (c3) include (c1, c2, c4);
create index 'idx4' on 'dt' (c5) include (ALL);
```

考虑如下查询：

```
select rowkey from dt where c1 = 'a';
select rowkey from dt where c2 = 'b' and c4 = 'd';
select * from dt where c2 = 'b' and c3 >= 'c' and c3 < 'f';
select * from dt where c5 = 'c';
```

下面逐个分析

（1）select rowkey from dt where c1 = 'a'

命中索引表idx1

（2）select rowkey from dt where c2 = 'b' and c4 = 'd'

命中索引表idx2，从中查找所有满足c2 = 'b'条件的行，然后逐行按c4 = 'd'进行过滤。虽然C4是索引列之一，但因where条件中缺少C3列，无法匹配上idx2的前缀。

（3）select * from dt where c2 = 'b' and c3 >= 'c' and c3 < 'f'

命中索引表idx2，完美匹配。但因为是select *，而索引表里并未包含主表的所有列，因此在查询索引之后，还要回查一次主表。回查主表时，回查的rowkey可能散布在主表的各个地方，因此，可能会消耗多次RPC。回查的数据量越大，RT越长。

（4）select * from dt where c5 = 'c'

命中索引表idx4，完美匹配。因为idx3是全冗余索引，所以，select *不需要回查主表。

因此，用户需要结合实际查询模式来进行索引表的设计，并考虑好未来一段时间中潜在的业务变化。限于篇幅，这里不对此进行详细展开讨论。有兴趣的读者可以阅读《数据库索引设计与优化》，以更进一步的了解如何用好索引。

约束与限制

- 不同主表可以有同名索引，如dt有索引idx1，foo也可以有索引idx1；但同一主表下不允许有同名索引
- 只能为version = 1的表建索引，不支持为多版本的表建索引
- 对有TTL的主表建索引，不能单独为索引表设置TTL：索引表会自动继承主表的TTL
- 索引列最多不超过3个
- 索引列 + 主表rowkey，总长度不能超过30KB；不建议使用大于100字节的列作为索引列
- 单个主表的索引表个数最多不超过5个
- 一次查询最多只能命中一个索引，不支持多索引联合查询(Index Merge Query)
- 创建索引时会将主表的数据同步到索引中，对大表建索引会导致create_index命令耗时过长。异步创建索引的功能会在未来开放。
- scan的filter中，仅支持BinaryComparator，若使用其他comparator，则查询会退化为主表的查询(不会抛错)

下列功能因为使用上有一些限制，所以，暂时未开放：

- 异步创建索引：只建索引，不同步历史数据，通过显式执行一个命令来为历史数据构建索引。
- 自定义时间戳写入数据

对于索引使用上的任何问题，欢迎钉钉联系云HBase答疑或者工单咨询。

全文索引服务

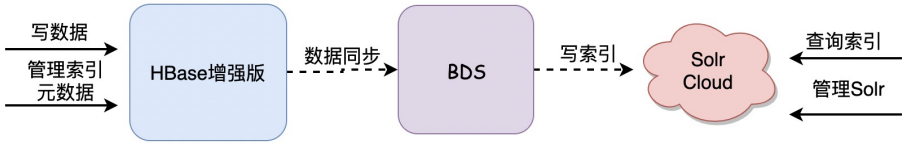
全文索引Search服务用来解决复杂的多维查询和全文检索。

简介

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

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

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



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

与二级索引的区别

HBase增强版提供高性能的原生二级索引，可以低成本的解决非主键查询问题，适用于查询列比较固定的场景。如果业务场景需要复杂的多维组合查询，需要考虑使用Search服务。

与开源Solr的区别

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

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

服务开通

开通Search服务需要三步：

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

具体参见开通指南

使用指南

参见快速入门和索引管理

最佳实践

参见最佳实践

性能白皮书

测试环境

- 本次测试对开源自建HBase与云HBase增强版进行了多个场景的性能对比
- 网络类型为VPC网络，并保证客户端与服务端处于同一服务可用区
- 所有测试均在华东2（上海）地域的可用区B完成
- 开源自建HBase使用社区1.4.9版本，为了对比方便，基于E-MapReduce提供的HBase服务替代(EMR版本: EMR-3.20.0)
- HBase增强版的版本为1.5.1.2

开源HBase集群所用配置:

名称	内容
Core节点配置	16C32G(ecs.c5.4xlarge)
Core节点数量	3
Core节点单节点磁盘大小	3.2TB
Core节点磁盘类型	高效云盘
Master节点配置	4c8g

HBase增强版集群所用配置：

名称	内容
Core节点配置	16C32G
Core节点数量	3
Core节点单节点磁盘大小	3.2TB
Core节点磁盘类型	高效云盘
Master节点配置	4c8g

压力端配置:

名称	内容
压力节点配置	16C32G(ecs.c5.4xlarge)
压力节点数量	1

测试工具

介绍

AHBench是由阿里云HBase团队研发的benchmark测试套件。

该测试套件集成了云HBase性能增强版性能白皮书所包含的YCSB(Yahoo! Cloud Serving Benchmark)测试集合、测试流程控制、结果汇聚等功能。借助该测试套件，您可以通过简单配置后，一键执行性能测试。

下载工具

从下载地址下载AHBench，上传到压测客户端并解压。

运行环境

请确保压测客户端运行环境满足：

- Linux系统
- JDK 1.8 +
- python 2.7
- 建议客户端CPU配置为独享16Core以上

配置

配置HBase集群地址(必须)

- 在AHBench/conf/hbase-site.xml中配置需要测试的HBase集群地址。
- 云HBase企业标准版可以参考 使用 HBase Shell 访问 中的配置zk地址一节。
- 云HBase性能增强版可以参考 HBase性能增强版-快速入门-HBase Shell访问中的配置一节。

配置运行时环境变量(必须)

打开 AHBench/conf/ahbench-env.properties ，配置工具运行的环境变量

```
vi AHBench/conf/ahbench-env.properties

# 配置JDK的安装地址。如果java已经在系统PATH中，可以跳过该配置
```

```
# JAVA_HOME=/usr/java/jdk1.8.0/

# 配置被测HBase集群的版本，如果为1.x版本则填1，如果为2.x版本则填2。
HBASE_VERSION=2
```

配置测试相关参数(可选)

打开AHBench/conf/ahbench-settings.properties，配置测试相关的参数，如压缩、编码、线程数、数据量、字段大小等，**默认不需要修改**，如有需求可根据场景进行定制修改。

说明：部分参数仅在特定HBase版本支持，如ZSTD压缩、INDEX编码仅在云HBase增强版支持，你可以通过配置ZSTD+INDEX获得更好性能

```
# 配置被测表的压缩算法，可选项有:
# NONE LZO ZSTD SNAPPY GZ LZ4 ZSTD 等
# 注意部分被测系统可能不支持指定的压缩算法
# 云HBase性能增强版推荐使用ZSTD
ahbench.table.compression=SNAPPY

# 配置被测表的编码算法，可选项有:
# NONE DIFF INDEX
# 注意部分被测系统可能不支持指定的编码算法
# 云HBase性能增强版推荐使用INDEX
ahbench.table.encoding=DIFF
```

启动测试

快速测试集

测试数据量1000万，整体运行时间大约40分钟(视被测HBase系统可能有变化)，至少需要总存储空间20GB

```
cd AHBench
./fast_test
```

完整测试集

测试数据量20亿，整体运行时间大约25小时(视被测HBase系统可能有变化)，至少需要总存储空间2TB

```
cd AHBench
./full_test
```

若要重复进行该测试，可以通过跳过数据导入阶段(上一次测试已成功运行)，减少运行时间。跳过导入阶段，整体测试运行时间大约3.5小时(视被测HBase系统可能有变化)

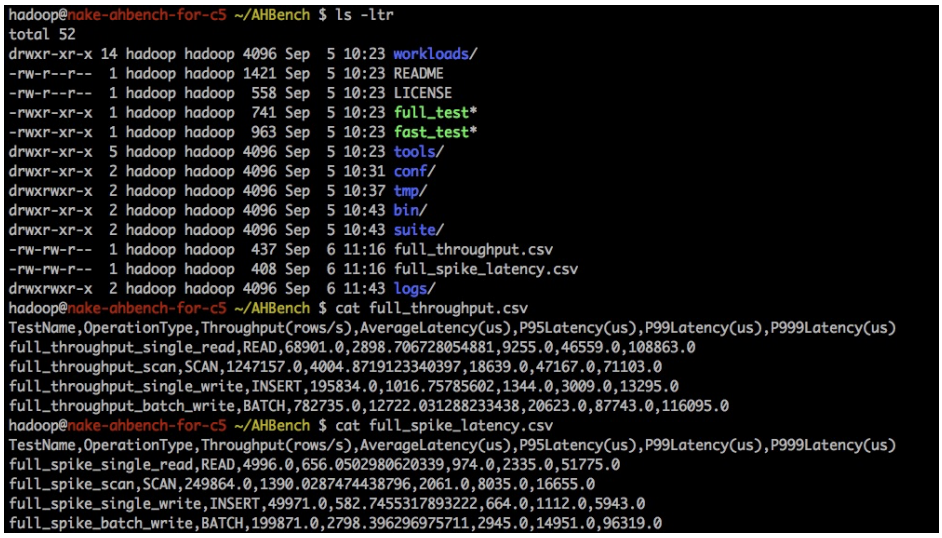
```
cd AHBench
```

```
./full_test --skipload
```

测试结果分析

测试case运行完毕后，会在当前目录生成csv文件。csv文件全称为逗号分隔值文件(Comma-Separated Values)。可以将测试结果复制到Excel/Numbers等数据分析软件中做进一步对比分析。

csv文件结果如图所示：



注意事项&常见问题

- 该测试套件使用”ahbenchtest-read”与”ahbenchtest-write” 两个表作为测试用表。测试过程中可能会删除再创建这两张表。请确保这两张表可以安全地被删除
- 测试压力可能将被测系统压垮，请勿在生产环境中运行该测试工具
- 如果测试中遇到错误退出，请检查如下事项:
    - JAVA_HOME是否正确设置，是否安装了python运行环境。
    - 被测集群地址是否填写正确
    - 被测集群HBase版本是否填写正确
    - 被测集群是否支持测试所指定的压缩算法
    - 被测集群状态是否正常服务
    - 被测集群存储空间是否足够
- ecs为虚拟运行环境，同一机型下的性能测试结果可能存在5～10%的波动，处于预期范围之内。

测试方法

本次测试分为吞吐量对比测试与毛刺率对比测试。吞吐量测试使用相同的线程数分别对社区版HBase和云HBase增强版进行对比测试，对比其吞吐量。毛刺率测试使用相同的吞吐压力分别对社区版HBase和云

HBase增强版进行对比测试，对比其毛刺率。压缩率测试使用相同的数据写入社区版HBase和云HBase增强版进行对比测试，对比其压缩比。

建表

在社区版HBase和云HBase增强版（Lindorm）集群中分别建表。所有case的表构造都相同，根据ycsb数据进行预分区，建表时预分200个分区。

云HBase增强版，使用独有的INDEX编码（当指定编码为DIFF时会自动升级为INDEX编码算法）和ZSTD压缩(参考其相关介绍)，建表语句为：

```
create 'test', {NAME => 'f', DATA_BLOCK_ENCODING => 'DIFF', COMPRESSION => 'ZSTD'}, {SPLITS =>
(1..199).map{|i| "user#{(i * ((2**63-1)/199)).to_s.rjust(19, "0")}"} }
```

社区版HBase，使用官方推荐的DIFF编码和SNAPPY压缩，建表语句为：

```
create 'test', {NAME => 'f', DATA_BLOCK_ENCODING => 'DIFF', COMPRESSION => 'SNAPPY'}, {SPLITS =>
(1..199).map{|i| "user#{(i * ((2**63-1)/199)).to_s.rjust(19, "0")}"} }
```

数据准备

对于单行读和范围读场景，需要对读取的数据进行数据准备。

单表准备20亿行基础数据，每行数据20列，value为20byte。

ycsb的配置文件为

```
recordcount=2000000000
operationcount=150000000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=false
fieldcount=20
fieldlength=20

readproportion=1.0
updateproportion=0.0
scanproportion=0
insertproportion=0

requestdistribution=uniform
```

启动命令为

```
bin/ycsb load hbase10 -P <workload> -p table=test -threads 200 -p columnfamily=f -s
```

测试场景

吞吐量对比测试

吞吐量测试为在固定线程数条件下，对比社区版HBase与云HBase增强版本的吞吐能力差异。一共有4个测试场景，每组测试场景均独立进行，测试场景之间没有前后依赖关系。

单行读场景

20亿行基础数据，每行数据20列，value为20byte，查询区间为1000万行。上述数据准备完成后，触发其major_compaction并等待完成。先进行预热测试20分钟，再进行20分钟的正式测试。

ycsb的workload配置如下：

```
recordcount=10000000
operationcount=2000000000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=false
fieldcount=1
fieldlength=20

readproportion=1.0
updateproportion=0.0
scanproportion=0
insertproportion=0

requestdistribution=uniform
```

启动命令为

```
bin/ycsb run hbase10 -P <workload> -p table=test -threads 200 -p columnfamily=f -p maxexecutiontime=1200
```

范围读场景

20亿行基础数据，每行数据20列，value为20byte。查询区间为1000万行，每次范围读取50行。上述数据准备完成后，触发其major_compaction并等待完成。先进行预热测试20分钟，再进行20分钟的正式测试。

ycsb的workload配置如下：

```
recordcount=10000000
operationcount=2000000000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=false
fieldcount=1
```

```
fieldlength=20

readproportion=0.0
updateproportion=0.0
scanproportion=1.0
insertproportion=0

requestdistribution=uniform
maxscanlength=50
hbase.usepagefilter=false
```

启动命令为

```
bin/ycsb run hbase10 -P <workload> -p table=test -threads 100 -p columnfamily=f -p maxexecutiontime=1200
```

单行写场景

每次插入1列，value为20byte。执行20分钟测试

ycsb的workload配置如下：

```
recordcount=2000000000
operationcount=100000000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=false
fieldcount=1
fieldlength=20

readproportion=0.0
updateproportion=0.0
scanproportion=0
insertproportion=1.0

requestdistribution=uniform
```

启动命令为

```
bin/ycsb run hbase10 -P <workload> -p table=test -threads 200 -p columnfamily=f -p maxexecutiontime=1200
```

批量写场景

每次插入1列，value为20byte，每次batch写入100行。执行20分钟测试：

```
recordcount=2000000000
operationcount=10000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
fieldcount=1
```

```
fieldlength=20
cyclickey=true

readallfields=false
readproportion=0
updateproportion=0
scanproportion=0
insertproportion=0.0
batchproportion=1.0
batchsize=100

requestdistribution=uniform
```

启动命令为

```
bin/ycsb run hbase10 -P <workload> -p table=test -threads 100  -p columnfamily=f -p maxexecutiontime=1200
```

毛刺率对比测试

毛刺率对比测试为在固定OPS（Operation per second）的条件下，对比社区版HBase与云HBase增强版本的毛刺差别。

单行读场景

20亿行基础数据，每行数据20列，value为20byte，查询区间为1000万行。OPS限制为5000上述数据准备完成后，触发其major_compaction并等待完成。先进行预热测试20分钟，再进行20分钟的正式测试。

ycsb的workload配置如下：

```
recordcount=10000000
operationcount=2000000000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=false
fieldcount=1
fieldlength=20

readproportion=1.0
updateproportion=0.0
scanproportion=0
insertproportion=0

requestdistribution=uniform
```

启动命令为

```
bin/ycsb run hbase10 -P <workload> -p table=test -threads 200 -p columnfamily=f -p maxexecutiontime=1200 -p
target=5000
```

范围读场景

20亿行基础数据，每行数据20列，value为20byte，查询区间为1000万行，每次范围读取50行。OPS限制为5000上述数据准备完成后，触发其major_compaction并等待完成。先进行预热测试20分钟，再进行20分钟的正式测试。

ycsb的workload配置如下：

```
recordcount=10000000
operationcount=2000000000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=false
fieldcount=1
fieldlength=20

readproportion=0.0
updateproportion=0.0
scanproportion=1.0
insertproportion=0

requestdistribution=uniform
maxscanlength=50
hbase.usepagefilter=false
```

启动命令为

```
bin/ycsb run hbase10 -P <workload> -p table=test -threads 100 -p columnfamily=f -p maxexecutiontime=1200 -p
target=5000
```

单行写场景

每次插入1列，value为20byte。执行20分钟测试OPS限制为50000

ycsb的workload配置如下：

```
recordcount=2000000000
operationcount=100000000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=false
fieldcount=1
fieldlength=20

readproportion=0.0
updateproportion=0.0
scanproportion=0
insertproportion=1.0

requestdistribution=uniform
```

启动命令为

```
bin/ycsb run hbase10 -P <workload> -p table=testwrite -threads 200 -p columnfamily=f -p
maxexecutiontime=1200 -p target=50000
```

批量写场景

每次插入1列，value为20byte，每次batch写入100行。执行20分钟测试：ops限制为2000

```
recordcount=2000000000
operationcount=10000000
workload=com.yahoo.ycsb.workloads.CoreWorkload
fieldcount=1
fieldlength=20
cyclickey=true

readallfields=false
readproportion=0
updateproportion=0
scanproportion=0
insertproportion=0.0
batchproportion=1.0
batchsize=100

requestdistribution=uniform
```

启动命令为

```
bin/ycsb run hbase10 -P <workload> -p table=testwrite -threads 100 -p columnfamily=f -p
maxexecutiontime=1200 -p target=2000
```

压缩率对比测试

以下所有压缩率对比测试均使用同一测试流程。通过ycsb写入500万行测试数据，手动触发flush与major_compaction并等待完成，统计其表大小。

每行列个数	每列value大小
1	10
1	100
20	10
20	20

ycsb的workload配置如下

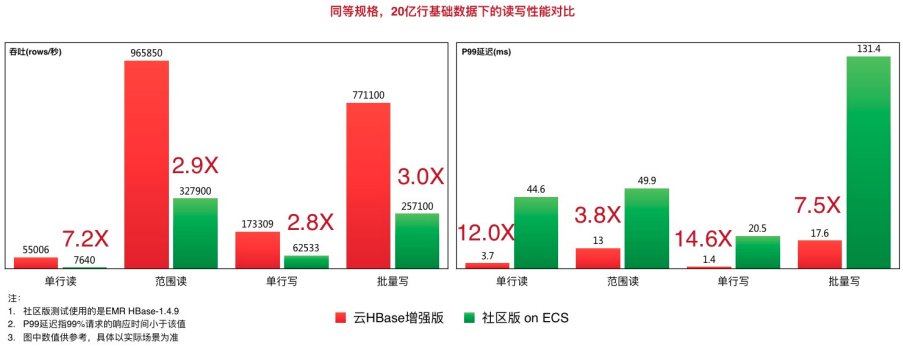
```
recordcount=5000000
operationcount=150000000
workload=com.yahoo.ycsb.workloads.CoreWorkload

readallfields=false
fieldcount=<每行列个数>
fieldlength=<每列value大小>

readproportion=1.0

requestdistribution=uniform
```

写入数据命令为

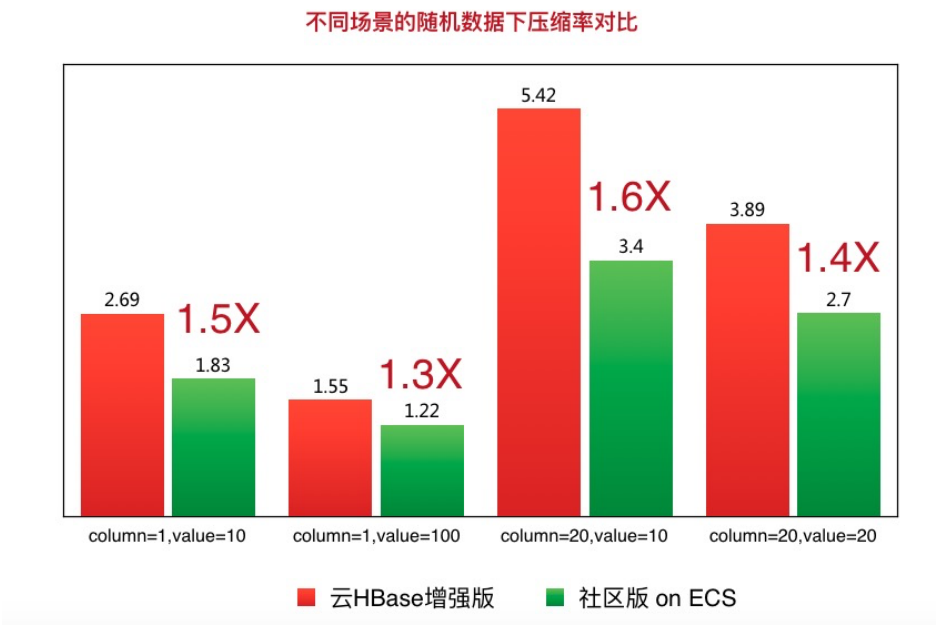
```
bin/ycsb load hbase10 -P <workload> -p table=test -threads 200 -p columnfamily=f -s
```

测试结果

吞吐量对比和毛刺率对比测试结果



压缩率对比测试结果



数据仅供参考，请以具体场景为准

集群管理

集群管理系统

使用前准备

添加白名单

将访问集群管理系统的电脑的IP加入到白名单中。关于白名单的设置，可以参照**IP白名单设置**。

设置访问密码

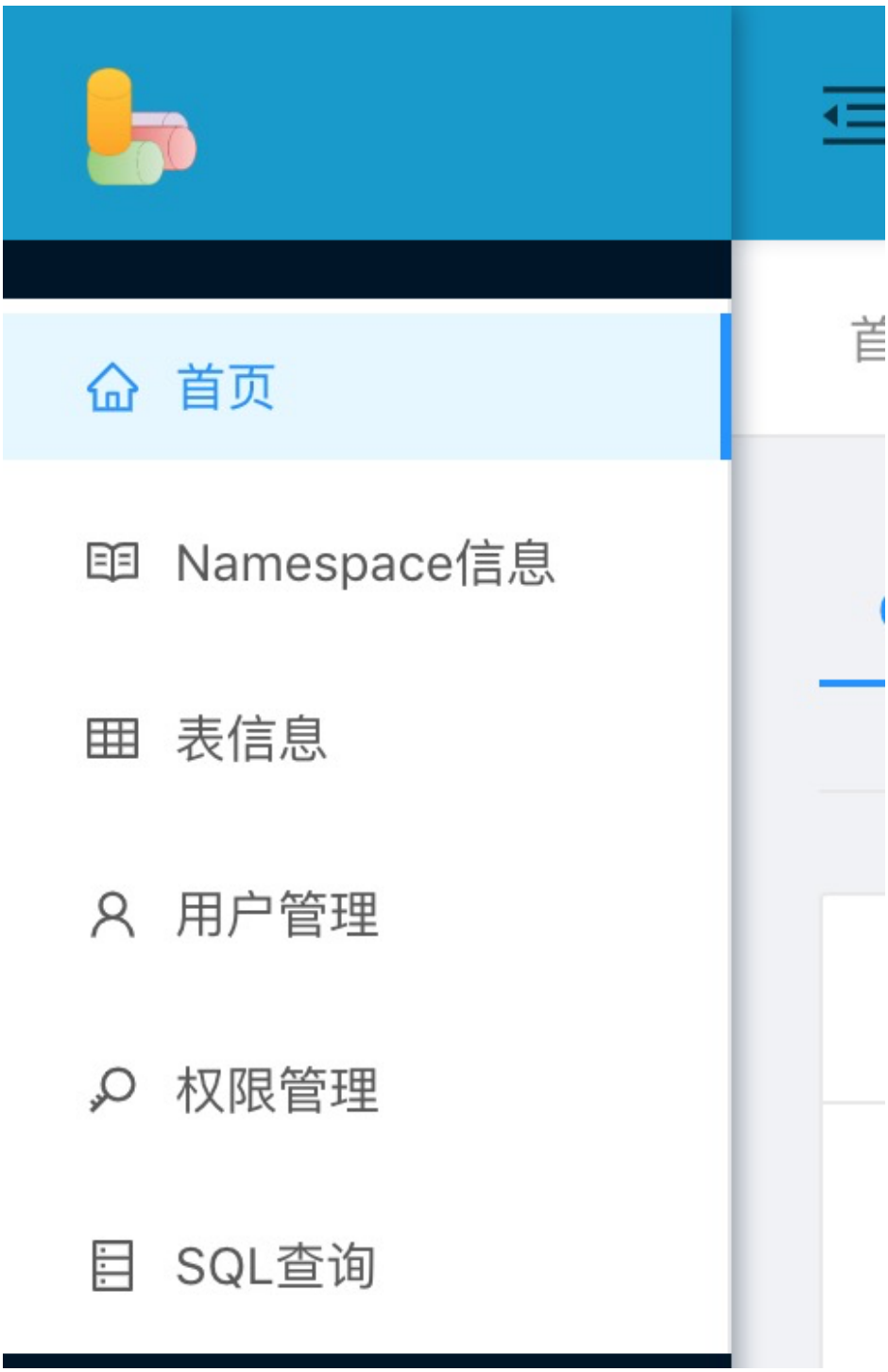
在访问之前，请确保已经正确设置了**访问密码**，点击下图中的“重置UI访问密码”可以重设密码。

基本介绍

HBase增强版为用户提供了一个全新的集群管理系统，通过这个系统可以看到集群的容量状况，Server列表，表属性，表大小等等信息。在集群管理系统中，还可以完成namespace管理、用户管理、ACL管理等功能。集群管理的入口在用户控制台中（参考上图），点击ClusterMananger链接即可进入。

集群管理系统分为以下几个页面：

主页



在主页中，可以看到集群以下信息：

集群空间使用情况

在此部分用户可以看到自己集群的空间的使用情况，是否需要扩容集群的磁盘空间

Group信息

在此部分用户可以看到集群中的所有group，并可以对group进行管理。关于Group的详细介绍，可以参考

Group管理章节

Server列表

在此部分用户可以看到所有在线的RegionServer，以及一些Server的状态，如果点击Server的名字，还可以查看Server的详情，包括此RegionServer上的Region信息等。用户可以把RegionServer分配到不同的Group里从而达到请求隔离的目的，关于Group的详细介绍，可以参考Group管理章节

集群健康信息

如果有宕机的RegionServer，或者没有上线的region，会在此部分展示。HBase在balance或者split region时，region会有短暂不在线，属于正常情况，如果存在长时间不在线的region或者RegionServer，请提工单或者在钉钉上联系“云HBase答疑”钉钉号。

Namespace信息

在Namespace信息页面中，用户可以看到集群中所有的namespace，并对namespace进行管理。详细的namespace管理请参照Group管理章节

表信息

在表信息页面中，用户可以看到集群中所有的表，以及属性，当点击表名后，可以看到表的详细信息，所有region信息等等。

用户管理

在用户管理页面中，用户可以看到当前集群中的用户，以及其所拥有的权限，用户可以进行创建用户，修改用户密码，删除用户等操作。详细的指南可以参见用户与ACL管理章节

权限管理

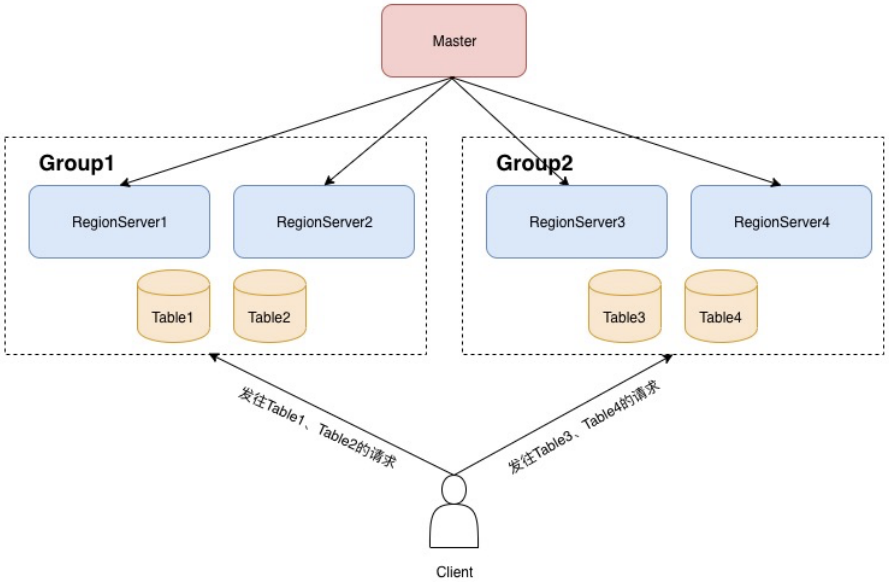
在权限管理页面中，用户可以管理相应用户所拥有的权限，可以赋予(grant)，回收（revoke）某个用户的一项或者多想权限。详细的指南可以参见用户与ACL管理章节

SQL查询

在SQL查询页面中，用户比较方便地使用select语句查询表内容，详细的介绍请参见SQL数据查询章节

Group管理

当多个用户或者业务在使用同一个HBase集群时，往往会存在资源争抢的问题。一些重要的在线业务的读写，可能会被离线业务批量读写所影响。而Group功能，则是HBase增强版提供的用来解决多租户隔离问题的功能。通过把RegionServer划分到不同的Group分组，每个分组上host不同的表，从而达到资源隔离的目的。



例如，在上图中，我们创建了一个Group1，把RegionServer1和RegionServer2划分到Group1中，创建了一个Group2，把RegionServer3和RegionServer4划分到Group2。同时，我们把Table1和Table2也移动到Group1分组。这样的话，Table1和Table2的所有region，都只会分配到Group1中的RegionServer1和RegionServer2这两台机器上。同样，属于Group2的Table3和Table4的Region在分配和balance过程中，也只会落在RegionServer3和RegionServer4上。因此，用户在请求这些表时，发往Table1、Table2的请求，只会由RegionServer1和RegionServer2服务，而发往Table3和Table4的请求，只会由RegionServer3和RegionServer4服务，从而达到资源隔离的目的。

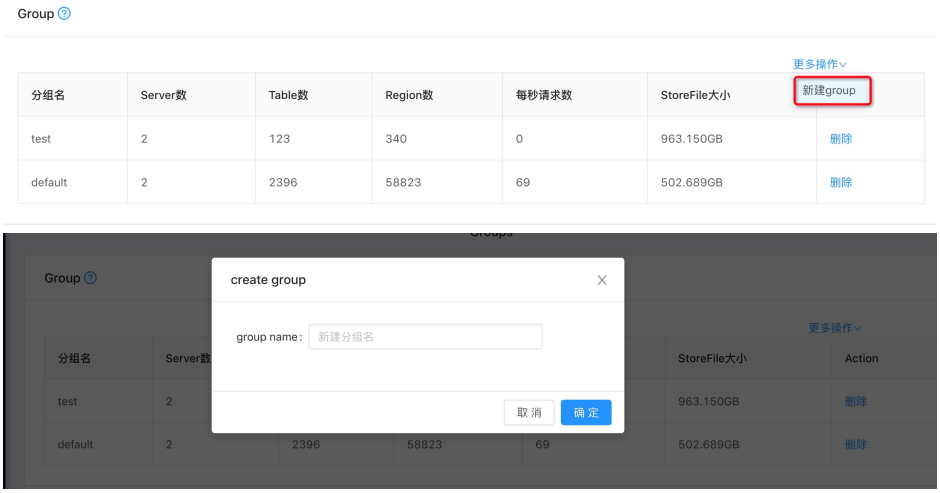
HBase增强版支持在集群管理系统管理Group。

查看Group信息

在集群管理界面的主页中，可以看到当前集群所有的Group信息。如果用户没有创建过任何group，系统会有一个默认的default group。所有的RegionServer和表都会隶属于这一个Group。

创建新Group

在集群管理页面中，点击group表的“更多操作—>新建group”，创建新的group。创建新的group后，这个group内的server数量和表数量都会为0，后续需要用户将Server和表移动至这一个group。

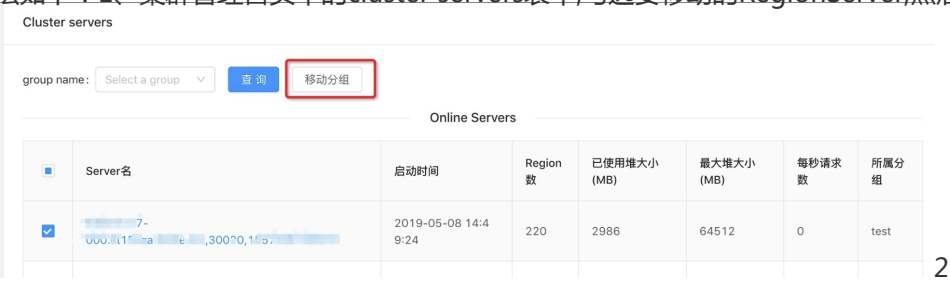


删除Group

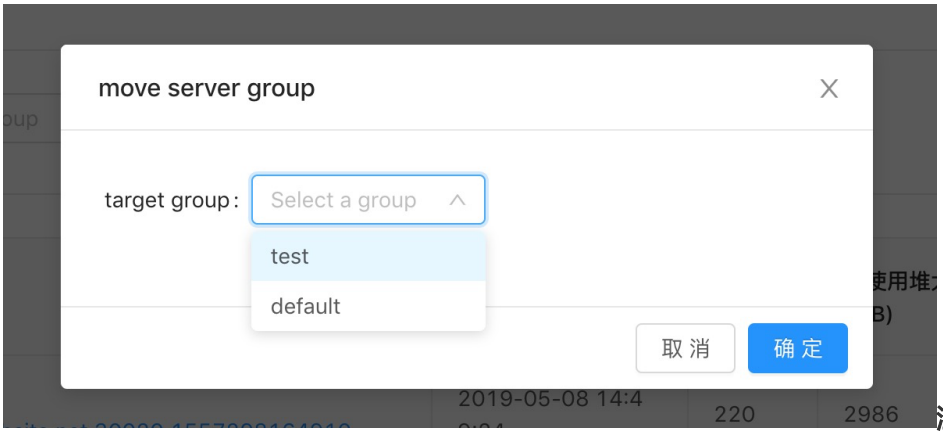
在集群管理首页中，点击group表对应行的“删除”，删除对应的group。**注意，只有当该group中的所有表和Server都被移出的时候，group才能被删除**

移动RegionServer的所属分组

默认状态下，所有的RegionServer都属于default分组，用户需要将Server移动到对应的Group中才能使用。移动分组的使用方法如下：1、集群管理首页中的cluster servers表中,勾选要移动的RegionServer,然后点击“移动分组”按钮.



2、在弹出框中选择目标group,然后点击“确定”即可.

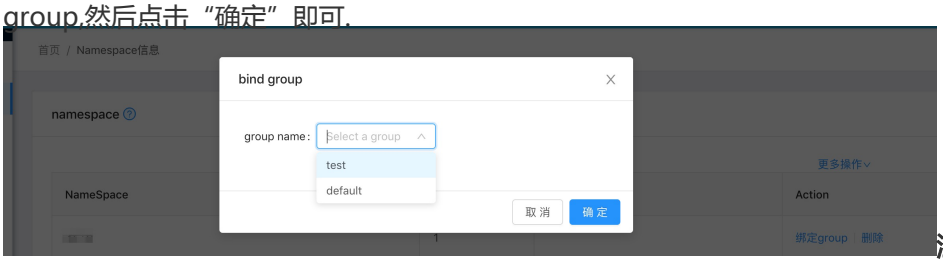


注意：1.如果用户将一张表移动到一个没有任何RegionServer的group，表的region会因为没有任何服务器可以上线从而无法访问。2.每一个分组最好至少拥有两台RegionServer，这样当一台RegionServer宕机后，表的region至少可以failover到同一分组的另外一台Server上。如果整个分组只有一台Server，当这台Server宕机后，这个分组所有的表将无法访问。3.在移动RegionServer的分组时，这个RegionServer上打开的region会被立刻重新balance到分组的其他机器上去

设置Namesapce所属分组

如果用户没有设置Namespace所属分组，用户新建的表默认都会属于default分组，其region会在default分组上线。如果用户不想每次在新建表后都手动将表移动到对应的分组，可以设置Namespace的所属分组。这样，在这个Namespace中新建的表，都会自动加入到对应的group分组。1、在集群管理的Namespace信息页面,点击对应namespace行的“绑定group”.

2、在弹出框中选择目标group,然后点击“确定”即可.

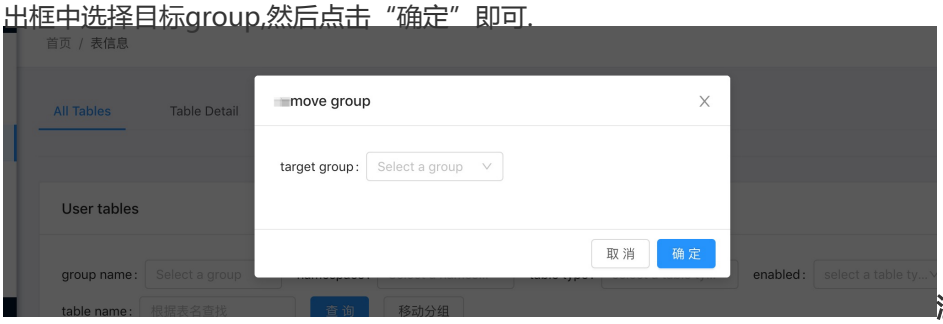


注意：1.修改Namespace分组的所属分组信息不会影响这个Namespace中已有的表，已有表的分组不会被改变。2.如果Namespace拥有所属分组，新建的表会自动属于这个group，但用户可以手工将这个表移动到另外的group

设置表的所属分组

用户可以手工将表移动到目标分组，操作方法如下：1、集群管理的表信息页中,勾选要移动分组的表,然后点击

“移动分组”按钮.2、在弹出框中选择目标group,然后点击“确定”即可.



注意：请勿将表移动到没有服务器的分组，这会造成表无法上线

Namespace管理

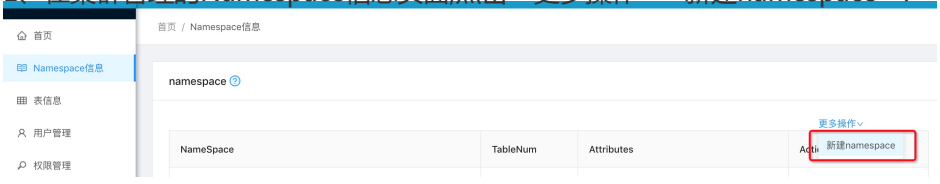
HBase中的Namesapce相当于表的一个逻辑分组。不同的Namespace下面的表允许重名。不同的租户拥有不同的namespace，彼此无法看到和影响。即不同用户的namespace将看到不同的表集合。HBase会默认提供一个叫做default的Namespace，如果用户没有指定使用Namespace，则会默认使用default Namespace。

创建删除Namespace

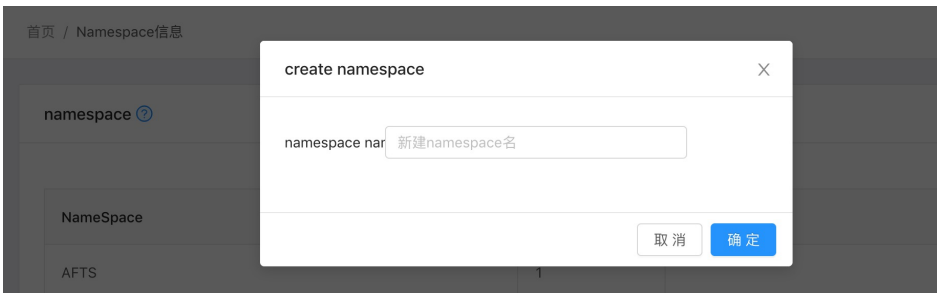
用户可以通过Java API和HBase的Shell来创建和删除Namespace。也可以使用集群管理系统来管理Namespace。

创建Namespace

1、在集群管理的Namespace信息页面,点击“更多操作—>新建namespace”.



2、在弹出的窗口填入namespace名称,然后点击“确定”即可.



删除Namespace

在集群管理的Namespace信息页面,点击对应行的“删除”按钮,即可删除对应的namespace.

注意：删除Namespace前请先将该Namespace下所有的表都删除，否则会删除失败

绑定Namespace所属Group

绑定Namespace所属Group可以将该Namespace下所建立的新表自动关联到某一个group，详见Group管理中的设置Namesapce所属分组章节。

用户和ACL管理

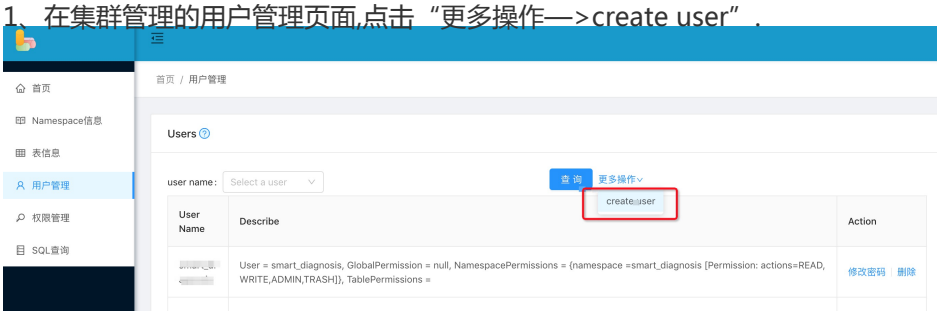
用户体系

HBase增强版提供一套简单易用的用户认证和ACL体系。用户的认证只需要在配置中简单的填写用户名密码即可。用户的密码在服务器端非明文存储，并且在认证过程中不会明文传输密码，即使验证过程的密文被拦截，用以认证的通信内容不可重复使用，无法被伪造。

您可以非常方便地通过**集群管理系统**来管理用户。在集群管理页面的用户管理页面中。会列出当前集群内的所有用户列表。在您购买集群后，系统会自动创建一个用户名为root，密码为root的用户，该用户拥有集群的所有权限，用户可以用这个用户访问集群，也可以通过管理系统修改这个用户的密码，或者删除这个用户。

创建新用户

1、在集群管理的用户管理页面,点击“更多操作—>create user”.

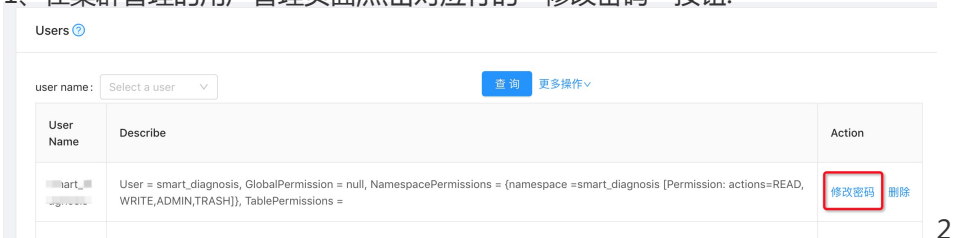


2、在弹出框中输入用户名密码,然后点击确定即可.

注意：1.HBase增强版在服务器端不会明文存储密码，因此创建用户后无法再看到该用户的密码，用户需要记住自己设置的密码，忘记密码后，只能通过修改用户密码的方式更改密码。2.新建的用户没有任何权限，需要在权限页面赋予相应的权限之后，才能够正常访问，详见ACL管理章节

修改用户密码

1、在集群管理的用户管理页面,点击对应行的“修改密码”按钮.

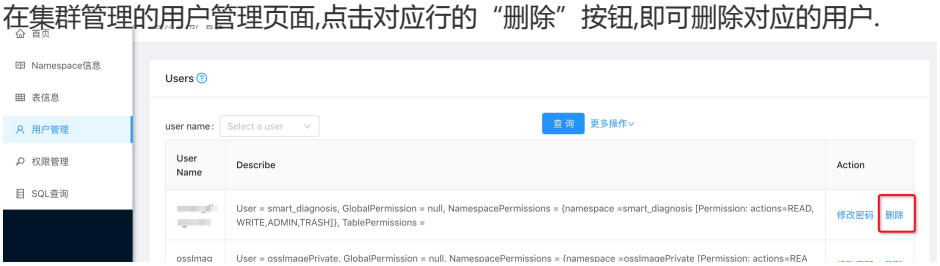


2、在弹出的对话框中填入新

密码,然后点击“确定”即可.

删除用户

在集群管理的用户管理页面,点击对应行的“删除”按钮,即可删除对应的用户.



ACL权限管理

权限种类

在HBase增强版中，服务器会根据每个用户拥有的权限去决定该用户是否能够做某项操作。如user1如果只有Table1的读权限，那么他在写Table1的时候就会报错，在读写Table2的时候都会被拒绝访问。目前，HBase增强版拥有的权限种类有以下几种：

WRITE 权限

与写HBase表相关的操作，如Put、Batch、Delete、Increment、Append、CheckAndMutate等

READ 权限

与读HBase表相关的操作，如Get、Scan、exist等；读取HBase表descriptor，namespace列表等相关操作，如getTableDescriptor、listTables、listNamespaceDescriptors等

ADMIN权限

不会涉及到表删除和表数据删除的DDL操作如createTable、enable/disableTable等，Namespace相关DDL操作，如createNamespace等。

TRASH权限

为了防止表误删和表数据被清空，只有授予了TRASH权限的用户，才能调用truncateTable和deleteTable这两个DDL操作

SYSTEM权限

只有赋予了SYSTEM权限的用户，才能执行Compact，flush等运维操作。另外，使用BDS迁移、同步HBase增强版时，使用的用户也必须有SYSTEM权限

权限分层

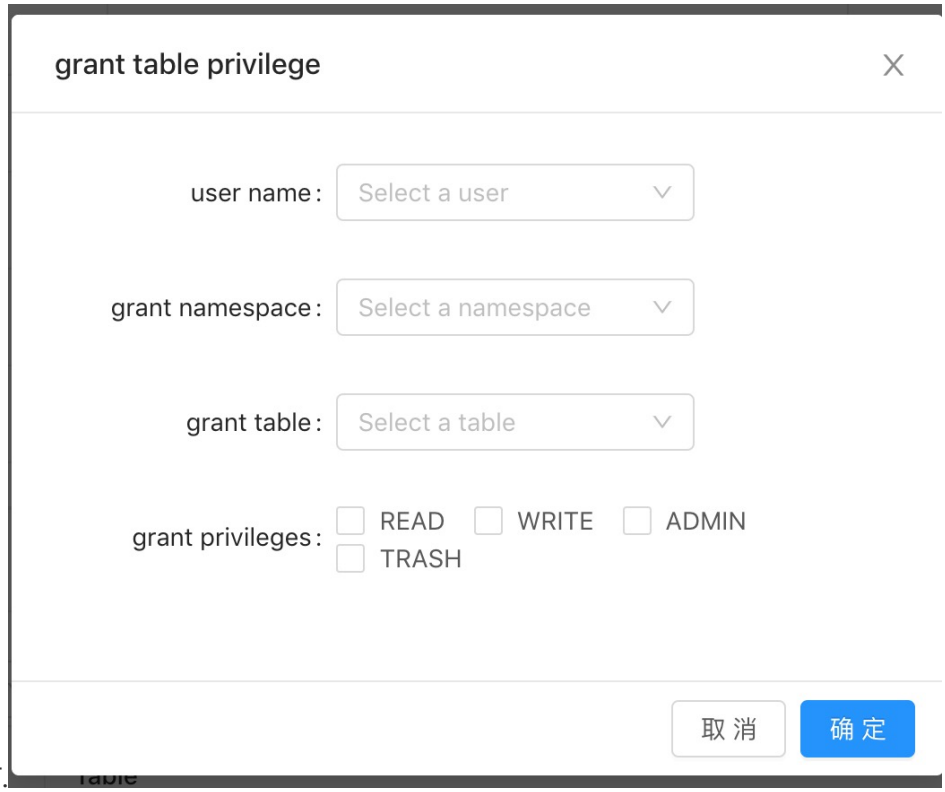
目前在HBase增强版中有三个权限层级。Global，Namespace和Table。这三者是相互覆盖的关系。比如给user1赋予了Global的读写权限，则他就拥有了所有namespace下所有Table的读写权限。如果给user2赋予了Namespace1的读写权限，那么他会自动拥有Namespace1中所有表的读写权限（包括在这个Namespace中新建的表）。**注意：只有拥有Global层级的ADMIN权限的用户，才能够Create和delete namespace**

权限管理

grant权限

在集群管理系统中的权限管理，可以给对应用户赋予相应权限。如果要给一个用户赋予某一个表的READ权限，可以这样操作：1、在集群管理的权限管理页面,点击Table权限表格的“更多操作—>grant privilege”按钮.

2、在弹出的grant table privilege对话框中,选择对应的用户、namespace和表,勾选READ权限,然后点击确定

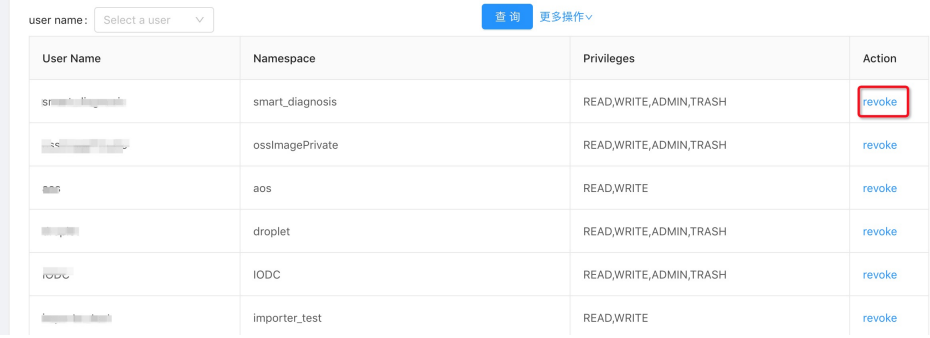


即可.

revoke权限

集群管理系统中的权限管理，可以收回对应用户的权限。每个用户可能有多个层级的权限。可以在对应层级的权限列表中找到对应用户，然后点击revoke按钮，然后选择需要revoke的权限,具体过程如下：

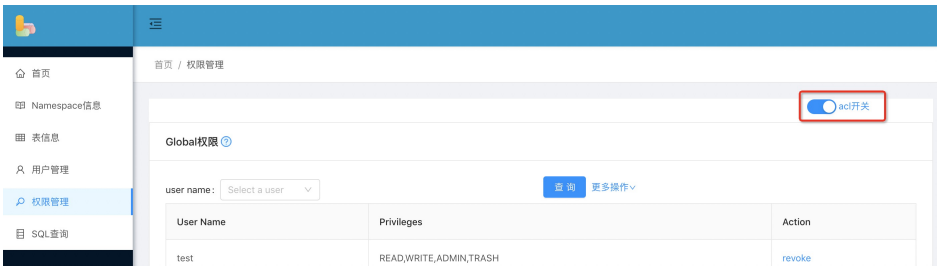
1、在集群管理的权限管理页面,找到对应的权限行,然后点击后面的revoke按钮.



2、在弹出的revoke权限对话框中,会列出当前用户对当前对象(global、表或者namespace)的所有权限,勾选需要revoke的权限,然后点击“确定”按钮即可.

打开关闭ACL功能

如果您不需要用户和ACL功能来进行访问控制，可以将ACL功能关闭。ACL功能关闭后，后续所有的访问（包括API访问，SQL访问和非JAVA访问）时都不需要提供用户名和密码，用户可以做任何操作而没有限制。ACL功能可以动态地打开和关闭，无需重启集群。但如果之前ACL是关闭状态，动态打开后，没有提供用户名和密码的客户端在重连时，会因为无法认证而报错。提供了用户名和密码的客户端在重连时会被正常认证，但在做越权的操作时，会被拒绝访问。



数据查询

在开发调试，或者在生产运维过程中，往往需要去HBase中查询某条数据。除了使用了HBase shell来写Get、Scan请求。HBase增强版还在**集群管理系统**中提供了一个简单的SQL查询入口。用户可以使用熟悉的SQL语法来查询HBase的表。

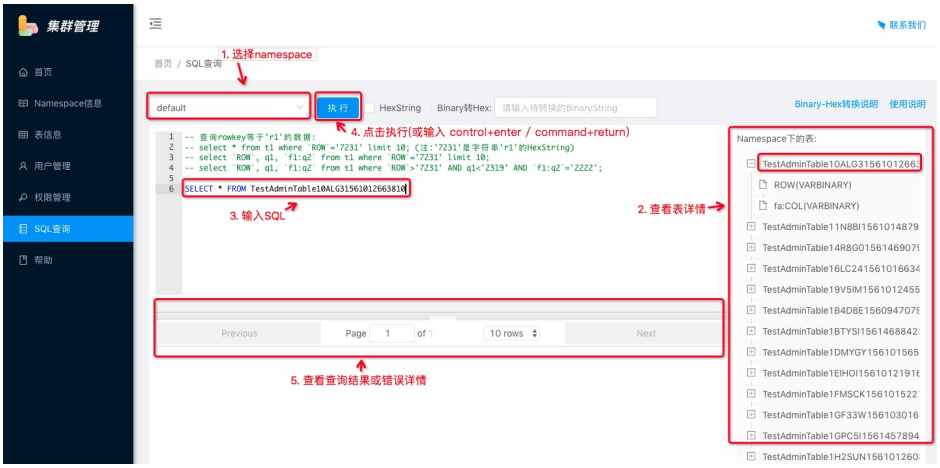
从集群管理系统中的SQL查询页面可以进入数据查询系统。在使用前，先要选择要查询的表所在namespace。选中了namespace后，会在页面右边的树形结构中列出这个namespace里所有的表。点击表名可以显示这个表的schema, ROW代表RowKey, COL是预置的列名。用户可以方便地根据这个表的schema来构造select请求。

一、简单入门

一个典型的SQL查询交互过程如下：

首先选择一个namespace，例如 'default'
在页面右侧中查看您需要查询的表结构
在编辑器中输入您的SQL
点击上面的 '执行' 按钮，或者通过键盘输入 'contrl+enter'(Windows)/'command+return' (MacOS) 来执行
查看编辑器下方的结果表格，如果执行出错，在此处也可看见错误详情

页面操作过程如下图所示：



注意：集群管理系统的SQL查询页面不支持use namespace语法,只能通过左上角的namespace列表选择/切换namespace。

二、使用说明

在您执行SQL前，请您务必先阅读下面的几点**重要说明**，能有效解决您使用时可能遇到的问题或疑惑：

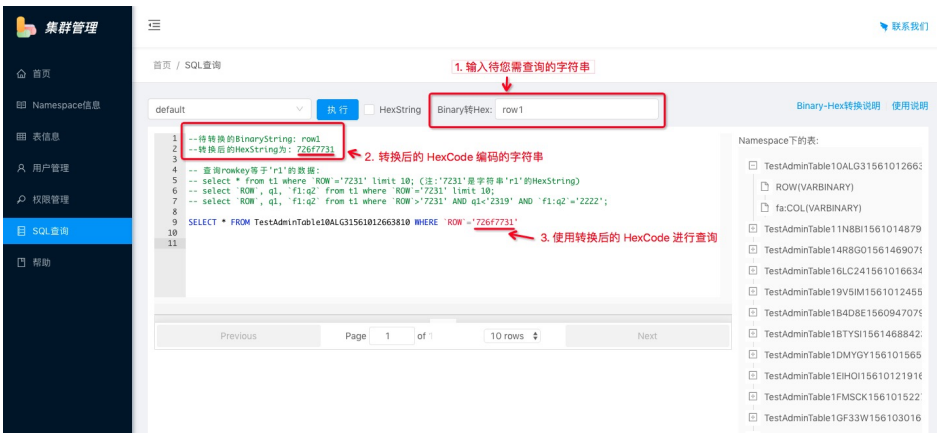
本系统只支持 'SELECT' 查询语句，如果需要变更数据操作，需要通过命令行或者使用产品提供的api自行开发应用。
为确保数据安全，本系统没次查询仅返回最多100条数据。
对所有varbinary类型的字段进行条件查询时，必须使用HexCode编码的字符串作为value 。
'ROW' 字段对应 HBase 的 rowkey，'ROW' 和 'qualifier' 都是 'varbinary' 类型，'qualifier' 如果不属于 family 'f'， 则需要指定 family，例如: 'select `ROW`, q1, `f1:q2` from ...'。
'ROW' 和 'COL' 是 SQL 保留字，查询时需要加反引号；'qualifier' 指定 family 时也需要加反引号。

三、Binary 编码字符与 HexCode 编码字符的转换

HBase内部使用 byte[] 的形式存储数据，在 SQL查询 页中返回的查询结果中，varbinary 类型的字段以 **HBase BinaryString** 的编码形式展示的。

在 varbinary 类型的字段进行条件查询时（即 where 子句中包含如 rowkey 等字段），必须使用 HexCode 编码的字符作为查询条件的 value。例如：查询 rowkey 为 r1 数据，则SQL查询中的 where 子句应该写成 where rowkey='7321'（注：字符串r1的HexCode编码字符为7321）。

为方便用户在查询时进行编码转换，系统在 SQL查询 页面中提供了一个简单的转换工具，只需将您的 Binary 编码字符串输入转换框中，即可在SQL编辑器内看到该字符串相应的 HexCode编码。详情如下图所示：



如果用户想查看 HexCode 编码的查询结果，可以勾选 执行 按钮后的 HexString 单选框，系统会将查询返回的结果中 varbinary 类型的字段都转为 HexCode 编码的字符串。详情如下图所示：