云数据库 HBase 版

HBase SQL服务(Phoenix)

HBase SQL服务(Phoenix)

HBase SQL(Phoenix)与Spark的选择

Phoenix 与 Spark的区别

ApsaraDB Phoenix是ApsaraDB HBase提供的SQL层，主要为了解决高并发、低延迟、简单查询场景，当然也可以解决一定的分析需求。 必须命中索引 且 命中后 返回的数据较少，如果是join，则join任意一则返回的数据量在10w以下，且另一侧必须命中索引。 为了保障集群稳定性，一些复杂的sql及耗时的sql会被平台拒绝运行。

ApsaraDB Spark是ApsaraDB HBase提供的分析引擎，满足 低并发，高延迟，复杂计算 场景。 不管怎么复杂的SQL，都可以完成。 另外 Spark可以支持sql、scala、java、python语言，支持流、OLAP、离线分析、数据清洗、支持多源(HBase、MongoDB、Redis、OSS等)。 (Spark Streaming支持准实时的在线流，不在此讨论访问内)

对比项目	Phoenix	Spark
SQL复杂度	简单查询， 必须命中索引 且 命中后 返回的数据较少，如果是join，则join任意一则返回的数据量在10w以下，且另一侧必须命中索引。 为了保障集群稳定性，一些复杂的sql及耗时的sql会被平台拒绝运行。	全部支持执行完成，支持Spark映射到Phoenix，做到Spark在简单SQL查询能到Phoenix同样的性能，不过Spark定位为 分析的场景，与Phoenix 纯TP有本质的区别
集群	HBase共享一个集群，本质是HBase提供的SQL	Spark需要单独购买集群
并发	单机 1w-5w左右	Spark最高不超过100
延迟	延迟在ms级别，一些命中较多的数据的sql会到 秒	一般延迟在300ms以上，大部分sql需要秒，分钟，甚至小时
更新	Phoenix支持	Spark不支持
支持业务	在线业务	离线业务 或者 准在线业务

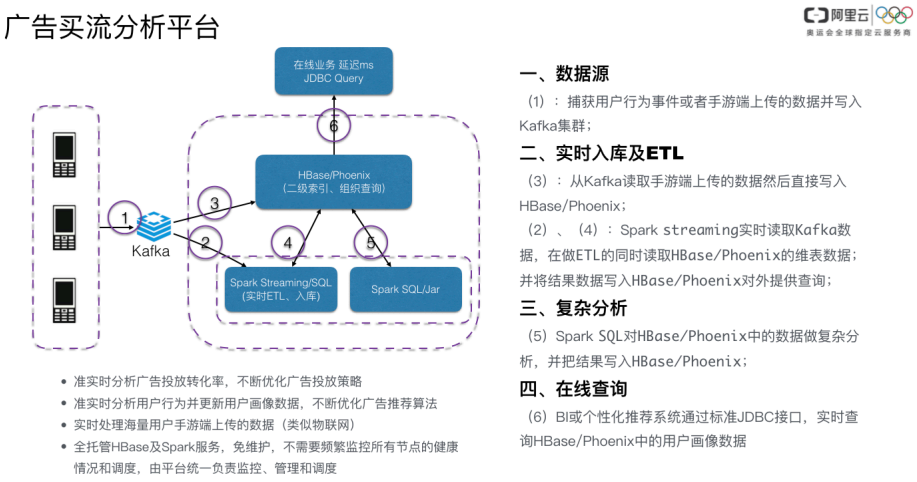
所以：

- 简单查询、 高并发、低延迟、 、在线业务 选择 Phoenix
- 复杂计算、低并发、高延迟、离线业务、准在线业务 选择 Spark

案例

可以看出：spark主要做流ETL及数据的二次加工，在线的查询还是通过Phoenix完成的

广告买流分析平台



HBase SQL(Phoenix) 4.x 使用说明

Phoenix是什么

Phoenix查询引擎支持使用SQL进行HBase数据的查询，会将SQL查询转换为一个或多个HBase API，协同处理器与自定义过滤器的实现，并编排执行。使用Phoenix进行简单查询，其性能量级是毫秒。

更多的信息可以参考官网：http://phoenix.apache.org/

Ali-Phoenix 说明

1. 兼容开源客户端(开源4.12)
2. 支持公网访问
3. 修复多个开源BUG
4. 新增功能和性能优化

参考：版本说明

Ali-Phoenix 客户端下载地址

Phoenix-4.12.0-AliHBase-1.1-0.9 下载 (JDK7编译)

使用说明

准备工作

准备一个内网的ECS，需要和HBase处在同一个网络内。

例如HBase是在经典网络的，那么就准备一个经典网络的ECS，如果HBase是在VPC的，那么就在需要在同一个VPC内的ECS

按照下载地址下载 Phoenix 客户端

在这台ECS上下载HBase对应版本的Phoenix客户端，这里以4.12.0-AliHBase-1.1-0.9版本举例

```
wget https://hbase-opt.oss-cn-hangzhou.aliyuncs.com/ali-phoenix-4.12.0-AliHBase-1.1-0.9.tar.gz
```

3. 解压缩压缩包

```
tar zxvf ali-phoenix-4.12.0-AliHBase-1.1-0.9.tar.gz
```

在HBase的网络白名单中开启访问节点的IP白名单

查看这台ECS的内网IP

```
hostname -i
```

然后把他加到HBase的网络白名单中。加入的方法请参考HBase白名单控制。

启动sqlline

在HBase产品的集群详情页面查看ZooKeeper的连接地址，然后使用如下的方式启动。启动命令sqlline.py在bin目录下

```
./sqlline.py hb-bp19142ir9xxxxxx-001.hbase.rds.aliyuncs.com,hb-bp19142ir9ruxxxxx-002.hbase.rds.aliyuncs.com,hb-bp19142ir9ruxxxxx-004.hbase.rds.aliyuncs.com
```

界面如果显示出类似jdbc:xxxx>这样的一个提示的时候，就表示启动成功了。

验证

我们在这个命令行中输入

```
!tables
```

如果看到一个表的列表，那么就说明我们配置成功了。可以开始更加深入的使用了。

退出

使用quit命令退出Phoenix

```
!quit
```

更多资料

Phoenix的入门教学Phoenix的深入使用

HBase SQL(Phoenix) 5.x 使用说明

Phoenix 5.x 说明

HBase SQL服务基于Phoenix 5.x为HBase 2.x提供SQL能力，通过轻客户端即可快速连接访问。SQL服务挂载了SLB负载均衡，通过round robin模式将请求均匀分发在每个query server节点。此外轻客户端还支持Python、Go等多语言访问。

使用步骤

开通HBase SQL服务

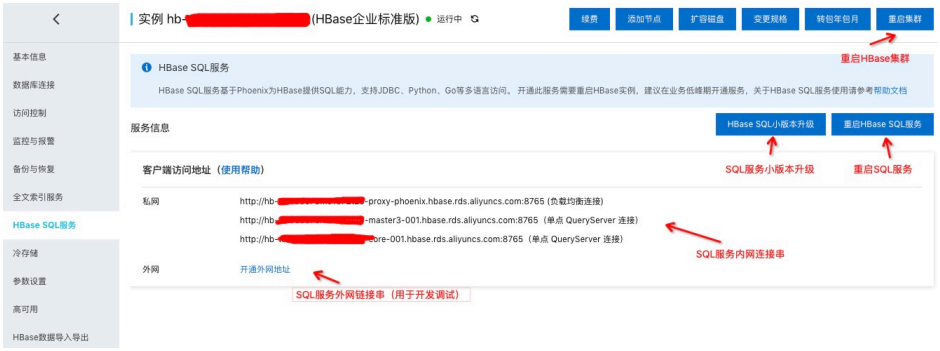
HBase实例开通后，进入控制台管理界面，在左侧菜单栏可以点击HBase SQL服务项->立即开通，即可开通HBase SQL服务。

开通HBase SQL服务时需要重启HBase实例，并占用部分内存资源，请在业务低峰期进行开通，开通过程需要十分钟左右，节点数越多时间会相对越长。

开通之后，后续对SQL服务重启和小版本升级也在该界面，点击相应按钮即可。

SQL服务管理

服务开通后，在HBase SQL服务栏即可看到如下界面：



用户可以在上图所示控制台中对SQL服务进行日常管理，包括SQL服务小版本升级，SQL服务重启，开通公网访问。

需要说明的是，SQL服务连接地址分为负载均衡连接和单点连接，分别在不同的场景中使用：

- 负载均衡连接：使用SLB进行负载均衡，对于高并发读写请求能够均发至后端的多个QueryServer中处理，提升集群的整体吞吐能力。
- 单点连接：使用大表创建索引或使用UPSERT...SELECT导数据以及复杂查询等单点请求场景，由于请求时间过长使用负载均衡连接会出现连接超时断掉情况，需使用单点QueryServer连接。

客户端准备

准备一台ECS，需要和HBase处在同一个网络内，建议是同一个vpc内部的ECS实例，否则需要打通网络，或者使用外网地址连接。

查看这台ECS的内网IP

```
hostname -i
```

然后把他加到HBase的网络白名单中。加入的方法请参考HBase白名单控制。

客户端访问HBase SQL服务

- 在准备好的ECS客户机上下载最新版本的HBase SQL客户端：

```
wget http://hbase-opt.oss-cn-hangzhou.aliyuncs.com/ali-phoenix-5.2.0-HBase-2.x.tar.gz
```

- 解压客户端压缩包

```
tar zxvf ali-phoenix-5.2.0-HBase-2.x.tar.gz
```

- 启动轻客户端工具

```
bin/sqlline-thin.py http://xxx-proxy-phoenix.hbase.rds.aliyuncs.com:8765
```

- 验证

我们在这个命令行中输入:

```
!tables
```

如果看到一个表的列表，那么就说明我们配置成功了。可以开始更加深入的使用了。

- 退出

使用quit命令退出Phoenix

```
!quit
```

注意：首次连接会创建meta表，需等待一段时间。

Phoenix 5.x vs Phoenix-4.x

1. Phoenix-5.x对于时区的处理逻辑做了统一优化，请参考云栖文章：Phoenix关于时区的处理方式说明。
2. Phoenix-5.x使用轻客户端模式，而Phonix-4.x默认提供重客户端模式。轻客户端和重客户端模式请参考云栖文章:Phoenix客户端优化之由重到轻。

HBase SQL(Phoenix) 入门

Phoenix 入门

本篇介绍简单的Phoenix使用方法，进行初步的数据查询。

在开始之前，请先确认您已经准备好了Phoenix的运行环境，若还没有，请参考这里

快速开始

1. 创建一个us_population表

```
CREATE TABLE IF NOT EXISTS us_population (
state CHAR(2) NOT NULL,
city VARCHAR NOT NULL,
population BIGINT
CONSTRAINT my_pk PRIMARY KEY (state, city));
```

写入数据

```
UPSERT INTO us_population VALUES('NY','New York',8143197);
UPSERT INTO us_population VALUES('CA','Los Angeles',3844829);
UPSERT INTO us_population VALUES('IL','Chicago',2842518);
UPSERT INTO us_population VALUES('TX','Houston',2016582);
UPSERT INTO us_population VALUES('PA','Philadelphia',1463281);
UPSERT INTO us_population VALUES('AZ','Phoenix',1461575);
UPSERT INTO us_population VALUES('TX','San Antonio',1256509);
UPSERT INTO us_population VALUES('CA','San Diego',1255540);
UPSERT INTO us_population VALUES('TX','Dallas',1213825);
UPSERT INTO us_population VALUES('CA','San Jose',912332);
```

查询SQL

```
SELECT state as "State",count(city) as "City Count",sum(population) as "Population Sum"
FROM us_population
GROUP BY state
ORDER BY sum(population) DESC;
```

结果验证

```
csv columns from database.
CSV Upsert complete. 10 rows upserted
Time: 0.159 sec(s)

St                              City Count                          Population Sum
-- ------------------------------------ ------------------------------------
NY                                     1                                8143197
CA                                     3                                6012701
TX                                     3                                4486916
IL                                     1                                2842518
PA                                     1                                1463281
AZ                                     1                                1461575
Time: 0.16 sec(s)
```

API访问Phoenix JDBC

Phoenix 5.x SDK maven依赖

```
<dependency>
<groupId>com.aliyun.phoenix</groupId>
<artifactId>ali-phoenix-queryserver-client</artifactId>
<version>5.2.1-HBase-2.x</version>
</dependency>
```

Phoenix 4.x SDK maven依赖

```
<dependency>
<groupId>com.aliyun.phoenix</groupId>
<artifactId>ali-phoenix-core</artifactId>
<version>${参考FAQ中最新版本说明}</version>
</dependency>
```

代码示例请参考：云HBase Demo

使用前必读

Phoenix定位为OLTP和操作型分析（operational analytics），大多用于在线业务，稳定性要求第一位。Phoenix的功能很强大，也很灵活，驾驭好对用户来说还是有一定难度，本文结合大部分用户的经验，给出一些使用建议，希望能帮助更多用户少走弯路。

一. 用户手册

Phoenix SQL基于SQL-92标准，但是还是有很多方言，软件架构也跟传统单机数据库有较大区别，用户在使用前务必阅读相关文档资料。以下罗列系列资料供用户参考：

1、Phoenix社区官方文档

http://phoenix.apache.org/，其中经常会翻到的链接如下：

Phoenix SQL语法

Phoenix数据类型

Phoenix内置函数

SQLLine 手册

2、阿里云HBase团队云栖中文系列文章和文档

Phoenix用户手册

HBase进化之从NoSQL到NewSQL，凤凰涅槃成就Phoenix

阿里云HBase SQL（Phoenix）服务深度解读

二. 使用建议

1、 二级索引使用指南

二级索引是Phoenix中非常重要的功能，用户在使用前需要深入了解，可以参考资料：二级索引社区文档，二级索引中文文档，全局索引设计实践。

下面介绍一些在使用二级索引过程中的注意事项：

1）是否需要使用覆盖索引？

覆盖索引需要将查询返回字段加入到索引表中，这样在命中索引时，只需要查询一次索引表即可，非覆盖索引，要想拿到完整结果则需要回查主表。不难理解，覆盖索引查询性能更好，但是会浪费一定存储空间，影响一定写性能。非覆盖索引使用时，有时执行计划并不能默认命中索引，此时，用户需要加索引Hint。

2）应该使用local Index还是global Index？实现上,一个global index表对应着一个hbase 表，local index是在主表上新增一列存储索引数据。

适用场景上，global index 适用于多读的场景，但存在同步索引时带来网络开销较大的问题。而local由于和原数据存储在一张表中同步索引数据会相对快一点。虽然local index也有一定适用场景，但仍然推荐使用global index, 其原因有以下几点：

1. 当前版本的phoneix的local index的实现相对global index不太完善，问题较多，使用存在一定的风险。
2. local index不太完善，大的改动后，可能会存在不兼容，升级流程比较复杂。
3. 在大数据量下，原始数据和索引数据放在一起会加剧region分裂，且分裂后索引数据的本地性也会丧失。

因此，在阿里云HBase SQL服务中**LOCAL INDEX功能已经被禁止！**

3）索引表最多可以创建多少个？

索引会保证实时同步，也会引来写放大问题，一般建议不超过10个，如果超过建议使用SearchIndex（全文索引）。

4) 构建索引需要注意哪些事项？

使用创建索引语句（CREATE INDEX）时，如果指定async参数，则为异步构建，语句完成时，会在SYSTEM.CATALOG表中简历索引表的元信息，并建立跟主表的关系，但是状态是building，索引表中没有数据，也不可查，需要后续用REBUILD语句来构建，或者借助MR工具来构建索引，并将状态更新为active。如果不指定，则为同步构建，一般建议数据量小于1亿行都使用同步构建即可，比较简单。

2、 加盐注意事项

注意：本特性有副作用，不建议使用。使用前请确保您已经充分了解其原理与适用场景。如果您无法确定您的场景是否适合，请钉钉与我们联系，我们会帮您进行评估。

加盐通常用来解决数据热点和范围查询同时存在的场景。原理介绍可以参考：Phoenix社区文档和中文文档。

加盐有较强的适用场景要求，场景不合适将适得其反：

- 写热点或写不均衡：比如以时间作为第一列主键，永远写表头或者表尾
- 需要范围查询：要按第一列主键进行范围查询，故不能使用hash打散

有热点就要打散，但打散就难以做范围查询。因此，要同时满足这对**相互矛盾的需求**，必须有一种**折中**的方案：既能在一定程度上打散数据，又能保证有序。这个解决方案就是加盐，亦称分桶(salt buckets)。**数据在桶内保序，桶之间随机**。写入时按桶个数取模，数据随机落在某个桶里，保证写请求在桶之间是均衡的。查询时读取所有的桶来保证结果集的有序和完备。

一般来说，严格满足上述条件的业务场景并不常见。大多数场景都可以找到其他的业务字段来协助散列。考虑到其严重的副作用，我们不建议使用这个特性。

副作用：

- **写瓶颈**：一般全表只有buckets个region用于承担写。当业务体量不断增长时，因为无法调整bucket数量，不能有更多的region帮助分担写，会导致写入吞吐无法随集群扩容而线性增加。导致写瓶颈，从而限制业务发展。
- **读扩散**：select会按buckets数量进行拆分和并发，每个并发都会在执行时占用一个线程。select本身一旦并发过多会导致线程池迅速耗尽或导致QueryServer因过高的并发而FGC。同时，本应一个RPC完成的简单查询，现在也会拆分成多个，使得查询RT大大增加。

这两个副作用会制约业务的发展，尤其对于大体量的、发展快速的业务。因为桶个数不能修改，写瓶颈会影响业务的扩张。读扩散带来的RT增加也大大降低了资源使用效率。

常见的使用**误区**：

- **预分区**：用分桶来实现建表的预分区最常见的误用。这是因为Phoenix提供的split on预分区语法很难使用。目前可用hbase shell的建表，指定预分区，之后关联为Phoenix表。在海量数据场景，合理的预分区是一个很有挑战的事情，我们后续会对此进行专题讨论，敬请期待。
- **伪热点**：写入热点或不均衡大多数情况都是假象，通常还有其他字段可用于打散数据。比如监控数据场景，以metric名字的hash值做首列主键可有效解决写入均衡的问题。

一定不要为了预分区而使用加盐特性，要结合业务的读写模式来进行表设计。如果您无法做出准确的判断，可以钉钉联系云HBase值班咨询。

Buckets个数跟机型配置和数据量有关系，可以参考下列方式计算，其中 N 为 Core/RS 节点数量：

- 单节点内存 8G: 2*N
- 单节点内存 16G: 3*N
- 单节点内存 32G: 4*N
- 单节点内存 64G: 5*N
- 单节点内存 128G: 6*N

注意：索引表默认会继承主表的盐值；bucket的数目不能超过256；一个空的Region在内存中的数据结构大概2MB，用户可以评估下单个RegionServer承载的总Region数目，有用户发生过在低配置节点上，建大量加盐表直接把集群内存耗光的问题。

3、 慎用扫全表、OR、Join和子查询

虽然Phoenix支持各种Join操作，但是Phoenix主要还是定位为在线数据库，复杂Join，比如子查询返回数据量特别大或者大表Join大表，在实际计算过程中十分消耗系统资源，会严重影响在线业务，甚至导致OutOfMemory异常。对在线稳定性和实时性要求高的用户，建议只使用Phoenix的简单查询，且查询都命中主表或者索引表的主键。另外，建议用户在运行SQL前都执行下explain，确认是否命中索引，或者主键，参考explain社区文档和explain中文文档。

4、 Phoenix不支持复杂查询

Phoenix的二级索引本质还是前缀匹配，用户可以建多个二级索引来增加对数据的查询模式，二级索引的一致性是通过协处理器实现的，索引数据可以实时可见，但也会影响写性能，特别是建多个索引的情况下。对于复杂查询，比如任意条件的and/or组合，模糊查找，分词检索等Phoenix不支持，建议使用SearchIndex（全文索引）。需要说明的是，相比于二级索引，SearchIndex采用后台异步索引，对在线业务影响较小，但会引入一定延迟，默认10s。

5、 Phoenix不支持复杂分析

Phoenix定位为操作型分析（operational analytics），对于复杂分析，比如前面提到的复杂join则不适合，这种建议用Spark这种专门的大数据计算引擎来实现，参考X-Pack Spark分析服务， HBase SQL(Phoenix)与

Spark的选择。

6、 Phoenix是否支持映射已经存在的HBase表？

支持，参考社区相关文档。用户可以通过Phoenix创建视图或者表映射已经存在的HBase表，注意如果使用表的方式映射HBase表，在Phoenix中执行DROP TABLE语句同样也会删除HBase表。另外，由于column family和列名是大小写敏感的，必须一一对应才能映射成功。另外，Phoenix的字段编码方式大部分跟HBase的Bytes工具类不一样，一般建议如果只有varchar类型，才进行映射，包含其他类型字段时不要使用映射。

备注：其他使用中常见问题请多翻看FAQ文档

全文索引（Search Index）

简介

一. SearchIndex解决哪些用户问题？

1. HBase不支持复杂查询

HBase作为海量在线存储引擎，被广泛应用于推荐、风控、物联网、画像、表单等大数据场景。Phoenix作为HBase的SQL层，极大降低了用户使用门槛，并且实现了二级索引、加盐表、动态列等大量实用功能。
HBase底层存储基于LSM，LSM能将业务的随机写转为顺序写，能有效提升写吞吐，但是其查询只适合于Rowkey的前缀匹配，查询模式单一；Phoenix二级索引，底层是跟原表关联的索引表，同样也是前缀匹配，一个表可以有多个索引，这样可以增加查询模式，但是索引数目不能太多，否则写放大的问题会比较严重。

对于更加复杂的查询场景，比如表单、日志查询里面的模糊查找，用户画像里面的随机条件组合等等，HBase + Phoenix的组合就不能支持。该问题是基于LSM的NoSQL数据库的通用问题，除了HBase，Cassandra、LevelDB、RocksDB、MongoDB引擎等都有相同的问题。

如果用户选择强行在OLTP数据库上做复杂查询，会导致在线库写性能严重下降，影响在线业务，甚至引发故障，做过在线业务的用户都深有感受。当然用户也可以选择在备库上做复杂查询，不过前面提到在线库本身的查询能力往往有限，满足不了在线复杂查询的相应要求。

2. 双写遇到的问题

为了解决问题1，用户自然会想到借助检索引擎，比如ES、Solr、Lucene等来解决该问题。不少用户选择的是双写的方式，也就是每一条记录同时写在线库和检索引擎，该方式看起来简单，但实际使用过程中问题很多。

我们了解到的case，把这套方案解决较好的客户往往都是要投入月级别的时间和大量人力。下面以双写HBase和Solr为例，举几个用户遇到比较多的问题。

1. 一致性难以保证双写很难保证在线库跟检索引擎的一致性。比如，两个链接并发双写，并且有修改的操作，那么很难保证HBase中同一字段的写入顺序跟Solr中同一个doc的修改顺序一致，那HBase和Solr中数据就出现了不一致，而且出现问题很难排查；另外，在线库往往只需要保存最近一段时间的数据，超过TTL的数据会被自动清理掉，而Solr中同样会有这个需求。但是HBase是按照KV做TTL的，Solr是按照doc，那两者在做数据清理的时候同样会出现不一致。不一致的场景有很多，这里就不一一介绍了。
2. 写入性能下降相同配置下，HBase的吞吐要比Solr高很多，这源于软件设计的出发点不同，优化的方向不同等诸多因素。如果双写，那势必会导致Solr的写吞吐限制了HBase的写吞吐。
3. 历史数据的同步双写只是解决了新数据的问题，对于历史数据则不适用，用户需要自己解决历史数据批量同步问题。特别是，对于不能停机的场景，在历史数据rebuild过程中，如何解决跟新数据跟历史数据相互覆盖的问题，也是十分棘手的问题。
4. 冗余存储空间检索引擎专门解决索引问题，其数据存储格式要比在线库要更复杂，一份在线库的数据在检索引擎中可能需要存储多份，比如原始数据存储，倒排索引存储，为提升聚合和排序的列存DocValue的存储。那么，势必有存储冗余的问题，如何降成本也是一大挑战。
5. 稳定性双写要求HBase和Solr同时保证稳定性，如果Solr出现故障，写流程会被block住，对在线业务造成影响。

3. HBase + Solr易用性不足

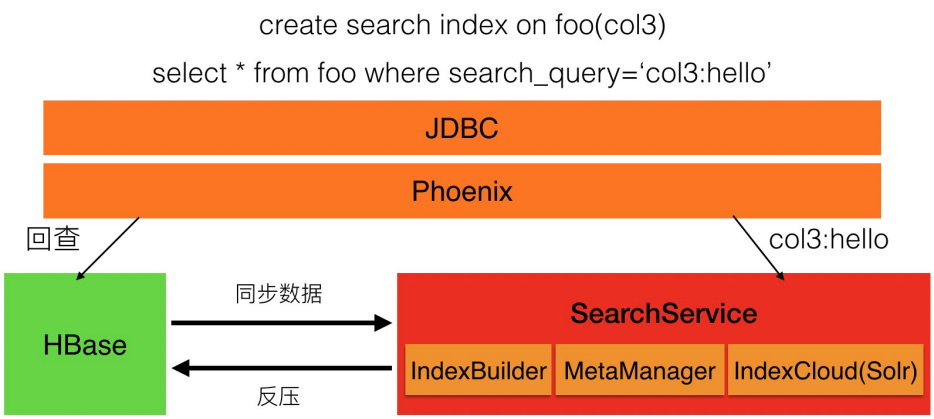
阿里云HBase Solr全文检索引擎，采用在系统层做数据转换和同步的方式一站式解决了用户使用双引擎遇到的大部分问题。但是，试用过的用户会有一个体会，就是使用太灵活了，步骤也比较繁琐，容易出问题，如果不是资深玩家难以驾驭。下面举几个用户痛点：

使用门槛高用户需要同时理解HBase、Solr、Indexer（数据同步服务），同时操作HBase Shell，Indexer命令行，Solr界面三个途径才能把流程走通。

Schemaless的HBase跟强Schema的Solr数据类型难以保证对齐首先，用户要自己定义从HBase column到Solr field的映射；其次，用户要自己保证实际写入到HBase中的类型正确。比如HBase中一个列对应Solr中一个long类型，因为HBase API并不检查用户实际写入的数值是否合法，导致写入HBase成功，但是同步到Solr是通不过的。这就要求用户要自己基于HBase API写一套类型检查系统，费时费力。

HBase + Solr对于数据冗余存储的问题解决不友好用户需要自己决定Solr中是否开启stored，docValued选项，对于只开启indexed选项的Field，用户可以通过回读HBase的方式来拿到最终结果数据，而对于开启了stored或者docValued的Field，直接从Solr中返回结果性能会更好。这套优化的逻辑需要用户自己管理和实现。

二. SearchIndex架构



SearchIndex是阿里云HBase SQL（Phoenix）基于HBase + Solr双引擎的新的索引实现，其架构如上图所示。Phoenix层将SQL（DDL、DML）语句转化为对HBase和Solr的具体操作，SearchService负责索引同步，一致性，元数据管理等。

SearchIndex解决了前面提到的所有问题，用户只需要几分钟，几条SQL语句就可以跑通整个流程；Phoenix强类型直接映射Solr类型，并支持分词、Array等复杂类型；自适应回查的优化策略更好解决了数据冗余存储问题。相比于HBase Solr全文检索引擎，大大提高了易用性，并且覆盖绝大部分的场景和需求。但目前SearchIndex还不能完全取代HBase + Solr，对于资深玩家，比较喜欢直接写HBase API和Solr API带来的灵活性，仍然可以选择使用HBase Solr全文检索引擎的方式。

快速开始

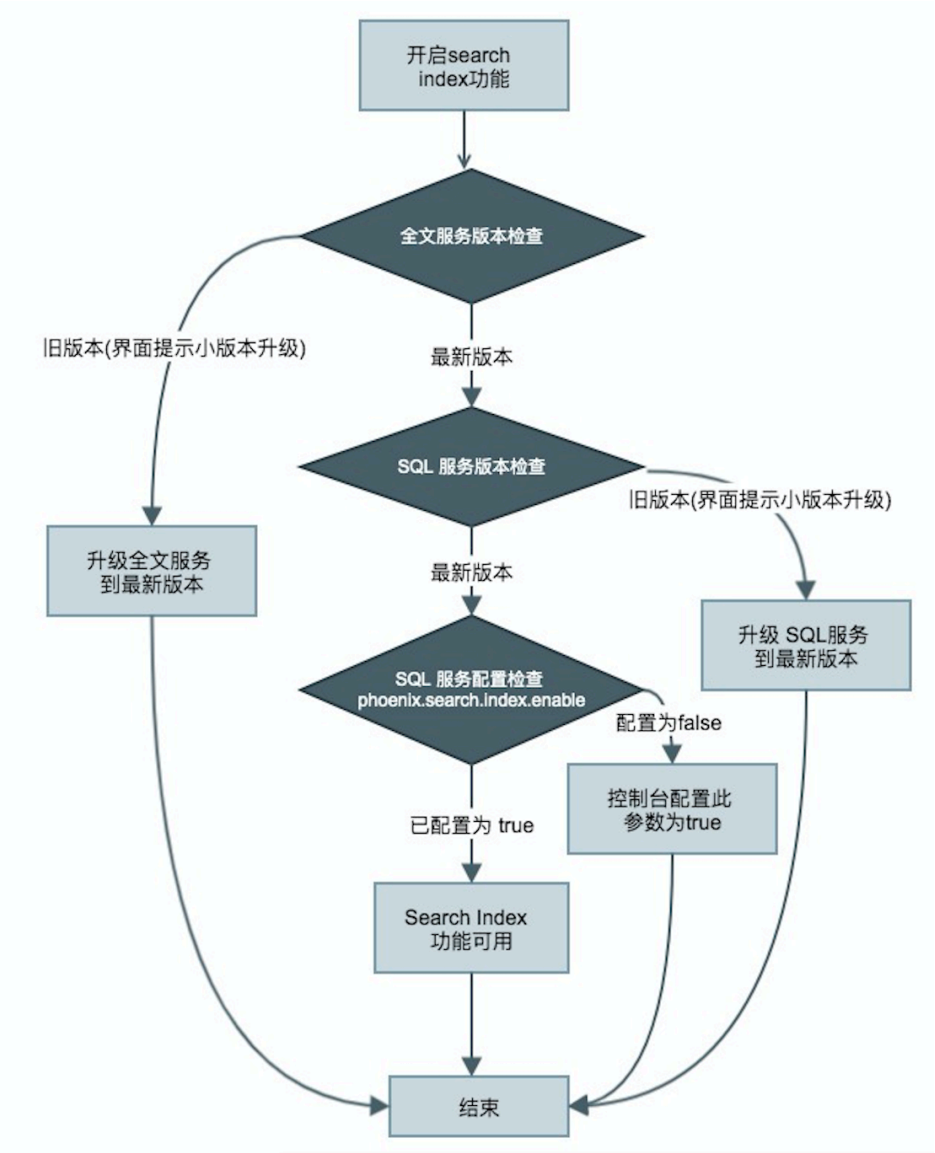
1. 开通 Search Index

Search index 功能只被支持在云 HBase 2.x, 依赖全文检索服务和 SQL 服务。开通此功能，主要包括以下三大步骤：

1. 开通全文检索服务，如已开通请升级至最新版本（界面没有提示小版本升级，当前版本为最新版本）
2. 开通SQL服务，如已开通请升级至最新版本（界面没有提示小版本升级，当前版本为最新版本）
3. 在控制台参数配置中，配置参数phoenix.search.index.enable为 true, 并重启 SQL 服务。

> 以上升级皆为 server 端操作，客户端不需要升级。

可以按照以下流程图进行检查：



后续版本升级只需单独升级SQL服务和全文检索服务即可，但需要**特别注意：在升级SQL服务前要先检查全文检索服务是否是最新版本，也就是要先升级全文检索服务，否则可能会带来兼容性问题。**

2. sqlline

开启交互式命令行，bin/sqlline-thin.py

```
-- 创建数据表
create table test (id varchar not null primary key,C1 varchar,C2 varchar, C3 integer)SALT_BUCKETS=2;

-- 写入数据
upsert into test values('a','word','Learn the definition of a phrase as a grammatical unit', 38000);
upsert into test values('b', 'say hello', 'I confused what truth is', 3000);

-- 创建同步索引
create search index on test (C1, C2 {ANALYZER=standard});
```

```
-- 索引查询 & 数据表条件过滤
select id from test where search_query='C2:truth OR C2:phrase' and C3 = 3000;
```

3. JDBC API DEMO

```
 Connection conn = DriverManager.getConnection(url);
String tableName = "test";
Statement stmt = conn.createStatement();
stmt.execute("create table " + tableName
+ "(id varchar not null primary key,"
+ "C1 varchar,"
+ "C2 varchar, "
+ "C3 integer )");

for (int i = 0; i < 15; i++) {
stmt.executeUpdate("upsert into " + tableName
+ String.format(" values('id_%d', 'c1_%d', 'c2_%d', %d)", i, i % 3, i % 3, i % 2));
}
conn.commit();

//create search index async
stmt.execute("CREATE SEARCH INDEX ON " + tableName + " (C1, C2) ASYNC
SHARD_NUM=1,REPLICATION_NUM=1");

stmt.execute("ALTER SEARCH INDEX ON " + tableName + " REBUILD");
stmt.execute("ALTER SEARCH INDEX ON " + tableName + " COMMIT");

ResultSet rs = stmt.executeQuery(
"select * from " + tableName + " where SEARCH_QUERY='C2:c2_2' and c3 = 1");
while (rs.next()) {
// fetch data
}
rs.close();

stmt.execute("DROP SEARCH INDEX ON " + tableName);
stmt.execute("DROP TABLE " + tableName);

stmt.close();
conn.close();
```

DDL语法

1. 创建全文索引

```
create search index [if not exists] on [schema.]table_name
```

```
(column_def[,...])
[ASYNC]
[index_options]

column_def:
column_name [{column_option,..}[,...]]

column_option:
docvalues=true | false // defalut value: false
| analyer=standard | ik // defalut value: none
| stored=true | false // defalut value: false
| indexed=true | false // defalut value: true

index_options:
SHARD_NUM = [integer number] // defalut value: rs size * 2
| REPLICATION_NUM = [integer number] // default value: 1
| COMMIT_INTERVAL_TIME = [integer number] // default value: 15000 (单位ms)
```

说明

其中 column_option 和 index_options 可省略，省略后使用默认的值。

- SHARD_NUM 表示 solr collection 的 shard 数量。
- REPLICATION_NUM 表示 solr collection 的 replication 数量。
- COMMIT_INTERVAL_TIME 表示写collection时每次自动commit的间隔时间。值越小索引可见间隔越短，但写入吞吐会下降。

分词器：分词器：standard是solr自带的默认分词器，能够支持对英文文本进行分词，但是如果是对中文文本进行分词则无法很好的支持。我们引入了ik分词器，需要对中文文本进行分词处理的请选择这个分词器。

索引数据同步：

1. 创建同步索引时，会自动发起索引 rebuild 操作，至到索引数据全部生成后，建表操作完成。
2. 创建异步索引时（使用ASYNC 关键字），不会自动发起索引的 rebuild，需要再次执行 alter rebuild 完成索引同步。

示例

```
CREATE SEARCH INDEX ON TEST (NAME,ADDR);
CREATE SEARCH INDEX ON TEST (NAME,ADDR)SHARD_NUM=2, REPLICATION_NUM=3;
CREATE SEARCH INDEX ON TEST (NAME,ADDR{DOCVALUES=true}) COMMIT_INTERVAL_TIME=10000 ;
CREATE SEARCH INDEX ON TEST (NAME {DOCVALUES=true, ANALYZER= standard}, ADDR);
CREATE SEARCH INDEX ON TEST (NAME,ADDR)ASYNC SHARD_NUM = 1, REPLICATION_NUM = 1;
```

2. 删除全文索引

```
drop search index [if exists] on [schema.]table_name;
```

示例

```
drop search index on test;
drop search index on s.test;
drop search index if exists on test;
```

3. Alter 全文索引

```
alter search index [if exists] on [schema.]table_name actions;

actions:
commit // commit 索引数据，立即可见
| set option // 设置索引属性
| rebuild // rebuild search index
| reload //Solr配置生效
| drop column [if exists] column_name[,...]
| add [if not exists] column_def[,...]

option:
COMMIT_INTERVAL_TIME = [integer number] // default value: 15000

column_def:
column_name [{column_option,..}[,...]]

column_option:
docvalues=true | false // defalut value: false
| analyer=standard | ik // defalut value: none
| stored=true | false // defalut value: false
| indexed=true | false // defalut value: true
```

说明

- set option 当前只支持设置 COMMIT_INTERVAL_TIME。
- rebuld 表示根据索引信息将当前全量Phoenix 表数据同步到全文索引中，操作较重，建议在业务低峰期进行，在rebuild不需要停机，系统会自动解决历史数据和新增数据的一致性问题。
- add column执行成功后，只能看到该列成功时刻后的新增数据，对于该列的存量历史数据并不能立刻可见。如果要查询该列的存量历史数据，需要执行rebuild。
- drop column执行成功后，该列会立刻不可查，但是历史数据实际是存在的，会占用存储空间。如果需要完全同步，需要执行rebuild。

示例

```
alter search index on test commit;
alter search index on test set COMMIT_INTERVAL_TIME=10000;
alter search index if exists on test rebuild;
alter search index if exists on test drop if exists name, addr;
alter search index if exists on test add if not exists NAME,ADDR{DOCVALUES=true};
```

支持字段数据类型

支持在以下Phoenix类型字段(Phoenix支持字段类型点击 这里 以及他们的array类型上建立全文索引

类型	对应Array类型是否支持
Varchar	是
Integer	是
Float	是
Bigint	是
Double	是
Boolean	是
Timestamp	否

DML语法

数据写入与删除

通过 Phoenix 提供的 Upsert 语法写入，索引数据会异步同步到全文索引中。主要支持 UPSERT VALUES 和 UPSERT SELECT 两类语法，我们在此不做赘述，具体可参考 这里。

通过 Phoenix 提供的 Delete 语法删除数据，索引数据会异步同步到全文索引中，具体可参考 这里。

search index Select

全文索引查询语法：

```
SELECT projected_columns
FROM table_name
WHERE search_query = 'search_expression' [ (and common_column1_filter), (and common_column2_filter)...]
[ ORDER BY (column_name1, column_name2, column_nsearch_queryame3... ]
[ LIMIT n ];
```

search_expression:

- 基本表达式，对应 solr 中的 q 参数, 使用说明详见这里。
- Solr JSON query, 使用 JSON 表达式查询 Solr.

对于Phoenix中timestamp类型字段建立的索引，在solr中我们都以Long(unix毫秒时间戳)的方式去进行存储和处理，因此在进行查询的时候，要将查询的日期转换成unix毫秒时间戳来进行查询。例如我对A_TIMESTAMP这个字段建立了全文索引，然后想要查询 ‘2011-11-11 11:11:11’ 这个时间点的数据，那么我要把表达式写成search_query = ‘A_TIMESTAMP:1320981071000’ 。

Solr JSON Syntax

```
{
"q": query_expression (string),
"fq": filter_query_expression(s) (string_or_array_of_strings, ...),
"sort": sort_expression(String),
"start": start_index(number),
"rows": result_rows(number),
"cursor": cursor_mark_string
}
```

参数说明

- q: 检索查询
- fq: filter query, 查询结果不涉及打分，并且会查询结果会被cache到内存，建议对于会复用的查询用，可参考Solr fq文档。
- sort: 字段名字 + asc/desc
- start: 查询开始位置索引，默认从0开始
- rows: 每次查询返回结果集大小，默认10
- cursor: 使用游标做翻页查询，对于深翻查询建议使用，相对于start和rows性能更好，start和rows会重头查。可参考Solr分页文档 。

JSON 语法说明

语法说明示例表：

```
create table tableName(id integer primary key, name varchar, addr varchar);
upsert into tableName(id,name)values(10, 'cc');
upsert into tableName(id,name,addr)values(11, 'aa', 'ZheJiang');
upsert into tableName(id,name,addr)values(12, 'bb', 'BeiJin');
create search index on tableName(name);
```

1. q 和 fq 参数查询

查询结果只显示在 select 中指定的列，可以配合 limit 和 order by 子句使用, 也可以在 JSON 语法中设定分页排序相关的参数。

```
SELECT * FROM tableName WHERE search_query='{"q":"NAME:*"}';


+-----+-------+-----------+
| ID | NAME | ADDR |
+-----+-------+-----------+
| 10 | cc | |
| 11 | aa | ZheJiang |
| 12 | bb | BeiJin |
+-----+-------+-----------+


SELECT * FROM tableName WHERE search_query='{"q":"*:*","fq":"NAME:bb"}';

+-----+-------+---------+
| ID | NAME | ADDR |
+-----+-------+---------+
| 12 | bb | BeiJin |
+-----+-------+---------+
```

2. 排序

```
SELECT * FROM tableName WHERE search_query='{"q":"NAME:*","sort":"NAME desc"}';

+-----+-------+-----------+
| ID | NAME | ADDR |
+-----+-------+-----------+
| 10 | cc | |
| 12 | bb | BeiJin |
| 11 | aa | ZheJiang |
+-----+-------+-----------+
```

3. 分页查询

start + row 分页

此方法适用于小数据量分页查询，同时 start 参数值不宜太大。

```
SELECT * FROM tableName WHERE search_query='{"q":"NAME:*","start":"1","rows":"2"}';

+-----+-------+-----------+
| ID | NAME | ADDR |
+-----+-------+-----------+
| 11 | aa | ZheJiang |
| 12 | bb | BeiJin |
+-----+-------+-----------+


// 排序 + 分页
SELECT * FROM tableName WHERE search_query='{"q":"NAME:*","sort":"NAME desc","start":"1","rows":"2"}';
```

```
+-----+-------+-----------+
| ID | NAME | ADDR |
+-----+-------+-----------+
| 12 | bb | BeiJin |
| 11 | aa | ZheJiang |
+-----+-------+-----------+
```

cursor 分页

cursor 适用于深翻，其性能相对于前一种方式更加平稳。每次查询需要指定 cursor 参数，查询结果会返回下一次查询所需的 cursor 字符串（此字段为虚拟字段，字段名为 NEXT_CURSOR）。

测试索引

```
create search index on tableName(name{stored=true}, addr{stored=true});
```

Select 查询字段必须全部在索引表中，同时开了 store 或者 doc value 属性。

查询所有数据

```
 SELECT * FROM tableName WHERE search_query='{"q":"NAME:*", "cursor":"*"}';
+-----+-------+-----------+-------------------+
| ID | NAME | ADDR | NEXT_CURSOR |
+-----+-------+-----------+-------------------+
| 10 | cc | | AoEoODAwMDAwMGM= |
| 11 | aa | ZheJiang | AoEoODAwMDAwMGM= |
| 12 | bb | BeiJin | AoEoODAwMDAwMGM= |
+-----+-------+-----------+-------------------+
```

当 rows 不设置时，使用默认值

第一次查查询，利用 cursor 读取一行数据：

```
SELECT * FROM tableName WHERE search_query='{"q":"NAME:*", "cursor":"*","rows":"1"}';
+-----+-------+-------+-------------------+
| ID | NAME | ADDR | NEXT_CURSOR |
+-----+-------+-------+-------------------+
| 10 | cc | | AoEoODAwMDAwMGE= |
+-----+-------+-------+-------------------+
```

当 cursor 的值为 * 时，表示 start marker

第二次查询、读取剩余的两行数据：

```
SELECT * FROM tableName WHERE search_query='{"q":"NAME:*", "cursor":"AoEoODAwMDAwMGE=","rows":"2"}';
+-----+-------+-----------+-------------------+
```

```
| ID | NAME | ADDR | NEXT_CURSOR |
+-----+-------+-----------+------------------+
| 11 | aa | ZheJiang | AoEoODAwMDAwMGM= |
| 12 | bb | BeiJin | AoEoODAwMDAwMGM= |
+-----+-------+-----------+------------------+
```

第二次查询时 cursor 参数的值，为第一次查询 NEXT_CURSOR 字段的结果。除了第一次 cusor 的值是星号，之后都是上一次查询 NEXT_CURSOR 字段的值。

4. 作为查询条件的列的大小写

语法说明实例表:

```
create table tableName(id integer primary key, "name" varchar, addr varchar);
upsert into tableName(id,"name",addr)values(10, 'cc', 'ShangHai');
upsert into tableName(id,"name",addr)values(11, 'aa', 'ZheJiang');
upsert into tableName(id,"name",addr)values(12, 'bb', 'BeiJin');
create search index on tableName("name",addr);
```

进行查询:

```
select * from tableName where search_query = '{"q":"ADDR:ZheJiang"}';

+-----+-------+-----------+
| ID | name | ADDR |
+-----+-------+-----------+
| 11 | aa | ZheJiang |
+-----+-------+-----------+

select * from tableName where search_query = '{"q":"name:aa"}';

+-----+-------+-----------+
| ID | name | ADDR |
+-----+-------+-----------+
| 11 | aa | ZheJiang |
+-----+-------+-----------+
```

在Phoenix中，列的大小写默认情况下是非大小写敏感的，建表中如果没有把列用双引号包起来则列的大小写默认均为大写(详情请点击这里)。json查询条件中的大小写情况和源表是一致的，但与sql语句中不写双引号默认大写的情况不同，json查询条件中不会将列名自动转为大写，因此，在json查询条件中要注意列名的大小写情况，必须与源表的大小写情况一致，例如上面例子中的 ADDR和 name列。

5. 带自定义列簇的列的作为查询条件进行查询

语法说明示例表:

```
create table tableName(id integer primary key, name varchar, f.addr varchar);
```

```
upsert into tableName(id,name,f.addr)values(10, 'cc', 'ShangHai');
upsert into tableName(id,name,f.addr)values(11, 'aa', 'ZheJiang');
upsert into tableName(id,name,f.addr)values(12, 'bb', 'BeiJin');
create search index on tableName(f.addr);
```

进行查询:

```
select * from tableName where search_query = '{"q":"F_ADDR:ZheJiang"}';

+-----+-------+-----------+
| ID | NAME | ADDR |
+-----+-------+-----------+
| 11 | aa | ZheJiang |
+-----+-------+-----------+
```

在json语法中对带自定义列簇的列进行查询时，要注意做为查询条件的列名为 “列簇” +” _” +” 列名” ,即上面例子的F.ADDR,在JSON查询时为F_ADDR。

6. JSON查询语法限制

- 查询的 where 子句 只支持 search_query 条件查询
- 不支持 limit + offset、order by 子句
- 不支持复杂查询操作比如：Join、子查询、分组聚合。

在ARRAY类型字段上使用search index

语法说明示例表：

```
create table tableName(id integer primary key, name varchar, addr varchar array);
upsert into tableName(id,name,addr)values(11, 'aa', ARRAY['ZheJiang','Guangdong']);
upsert into tableName(id,name,addr)values(12, 'bb', ARRAY['BeiJin','ShangHai']);
create search index on tableName(addr);
```

进行查询:

```
SELECT * FROM tableName WHERE search_query = '{"q":"ADDR:ZheJiang"}';

+-----+-------+------------------------+
| ID | NAME | ADDR |
+-----+-------+------------------------+
| 11 | aa | [ZheJiang, Guangdong] |
+-----+-------+------------------------+
```

查询条件只要匹配到该行的ARRAY字段中任意一项，就代表查询命中。

查询计划

SearchIndex会根据元数据判断，Solr中数据是否可查（打开stored或者docValues），如果可查则直接从Solr中返回结果，如果不可查则会自动回查HBase拿到最终结果。使用 epxlain 可以查看查询计划。通过查询计划可以判断，是直接查询索引返回结果，还是查询完索引又回查了数据表。

当 select 的列可以在索引表中直接获取数据时，不需要回查数据表

```
create search index on test2(name{stored=true});

explain select name from test2 where search_query='NAME:*';

+-----------------------------------------------------+-----------------+----------------+
| PLAN | EST_BYTES_READ | EST_ROWS_READ |
+-----------------------------------------------------+-----------------+----------------+
| CLIENT EXECUTE QUERY: [ NAME:* ] ON SEARCH INDEX | null | null |
+-----------------------------------------------------+-----------------+----------------+
```

当 select 的列不可以在索引表中直接获取数据时，需要回查数据表

```
create search index on test2(name{stored=true});

explain select addr from test2 where search_query='NAME:*';
+-----------------------------------------------------+-----------------+----------------+
| PLAN | EST_BYTES_READ | EST_ROWS_READ |
+-----------------------------------------------------+-----------------+----------------+
| CLIENT EXECUTE QUERY: [ NAME:* ] ON SEARCH INDEX | null | null |
| CLIENT EXECUTE MULTI GET ON DATA TABLE: TEST2 | null | null |
+-----------------------------------------------------+-----------------+----------------+
```

HBase SQL(Phoenix) FAQ

1. ali-phoenix 最新版本在maven中央仓库发布了哪些jar包？

- ali-phoenix 4.x发布包如下：

GroupId	ArtifactId	Latest Version
com.aliyun.phoenix	ali-phoenix	4.12.0-AliHBase-1.1-0.9
com.aliyun.phoenix	ali-phoenix-hive	4.12.0-AliHBase-1.1-0.9
com.aliyun.phoenix	ali-phoenix-spark	4.12.0-AliHBase-1.1-0.9
com.aliyun.phoenix	ali-phoenix-pherf	4.12.0-AliHBase-1.1-0.9
com.aliyun.phoenix	ali-phoenix-queryserver	4.12.0-AliHBase-1.1-0.9

com.aliyun.phoenix	ali-phoenix-queryserver-client	4.12.0-AliHBase-1.1-0.9
com.aliyun.phoenix	ali-phoenix-pig	4.12.0-AliHBase-1.1-0.9
com.aliyun.phoenix	ali-phoenix-flume	4.12.0-AliHBase-1.1-0.9
com.aliyun.phoenix	ali-phoenix-core	4.12.0-AliHBase-1.1-0.9

- ali-phoenix 5.x发布包如下：

GroupId	ArtifactId	Latest Version
com.aliyun.phoenix	ali-phoenix-queryserver-client	5.1.0-HBase-2.0
com.aliyun.phoenix	ali-phoenix-shaded-thin-client	5.1.0-HBase-2.0

避免轻客户端中包与业务三方包冲突，推荐使用轻客户端shade包。

2. ali-phoenix JDBC URL格式是什么样的？

在启用query-server的时候需要使用轻客户端，否则使用重客户端。*ali-phoenix不支持以keytab的方式访问云HBASE，所以此处和apache phoenix的JDBC格式有所差异*

重客户端DRIVE

URL语法

```
jdbc:phoenix:[[ZK_HOST1:port, ZK_HOST2:port, ZK_HOST3:port | comma-separated ZooKeeper Quorum [:port] [:hbase root znode] ]
```

简单URL示例

```
jdbc:phoenix:localhost
jdbc:phoenix:localhost:123:/hbase
jdbc:phoenix:v1,v2,v3:123:/hbase
jdbc:phoenix:v1:2181,v2:2181,v3:2181:/hbase
jdbc:phoenix:v1:2181,v2:2181,v3:2181
```

轻客户端DRIVE

URL语法

```
jdbc:phoenix:thin:[key=value[;key=value...]]
```

简单URL示例

```
jdbc:phoenix:thin:url=http://localhost:8765;serialization=PROTOBUF
```

3. 是否支持QueryServer?

HBase1.x版本使用Phoenix4.x重客户端模式，需要用户自行搭建QueryServer服务。

HBase2.0版本增加HBase SQL服务，默认开启QueryServer。

4. 是否支持Tracing Web Application ?

当前云HBASE上的ali-phoenix此不支持，此功能正在开发中

5.构建同步的二级索引超时怎么办？

HBase1.0上Phoenix4.x版本，需要在客户加上如下配置，并重启客户端。

```
<property>
<name>hbase.rpc.timeout</name>
<value>60000000</value>
</property>
<property>
<name>hbase.client.scanner.timeout.period</name>
<value>60000000</value>
</property>
<property>
<name>phoenix.query.timeoutMs</name>
<value>60000000</value>
</property>
```

HBase2.0上Phoenix5.x版本，可在控制台参数管理中修改上述参数，并重启HBase SQL服务，注意不需要重启HBase，仅重启HBase SQL服务(Phoenix)即可。

6.开通Namespace Mapping

4.x开通步骤：

1). 在客户端增加以下配置， 2).找云 HBase 答疑开通 Server 端参数， 3). 重启 HBase 集群。

```
<configuration>
<property>
<name>phoenix.schema.isNamespaceMappingEnabled</name>
<value>true</value>
</property>
<property>
<name>phoenix.schema.mapSystemTablesToNamespace</name>
<value>true</value>
</property>
```

```
</configuration>
```

5.x开通步骤：

1). 控制台 -> 参数配置，配置phoenix.schema.isNamespaceMappingEnabled 和 phoenix.schema.mapSystemTablesToNamespace 为 true。 2). 重启 HBase 集群。 3). 重启 SQL 服务。

注意： 这里需要配置两组相同的参数，参数描述是不相同的，一组是 query server 的，一组是 HBase server 端的，都需要设置。

7.是否支持连接池

Phoenix4.x最新版本的4.12.0.X版本支持，具体参考PhoenixConnectionPool.java

Phoenix5.x版本基于轻客户端实现，PhoenixConnectionPool.java不能使用，建议使用社区第三方连接池，比如mybatis，可参考链接池demo

8.执行创建索引时间太长能否断开开户端链接？

不能断开客户端链接。执行create index主要有两个步骤，第一步在server端同步源表数据到索引表， 第二步在客户端发起请求修改索引表状态设置为active。其中第一步一般是是客户端发起请求在server端完成后。

9.创建同步索引表，同步索引数据的速度怎么样？

一般情况下1000W数据创建索引需要5-20min, 具体情况视集群配置和资源使用情况而定。

10.创建索引由于时间太长，客户端断开了链接怎么办？

一般情况下当前索引表的状态是building状态的（可以在sqlline中使用 !table命令查看），只有当索引表状态变为active才算真正完成了索引构建。此时有两种解决方法：第一、通过alter index命令rebuild索引。第二、删除building状态的索引表，配置更大的客户端超时时间，重新创建索引。

11.关系型数据库怎么导入云HBASE的phoenix表中？

通过datax(https://github.com/alibaba/DataX)或者CDP, Phoenix4.x可利用hbase11xsqlwriter插件写入到Phoenix表中，其中zk的zookeeper.znode.parent配置值为/hbase。Phoenix5.x可使用hbase20xsqlwriter插件写入到Phoenix表中.

12. 查询时发生遇到ERROR 599(42912): Default enable Force index, please set phoenix.force.index=false to disable it....， 应该怎么处理？

为了避免查询扫全表，会在SQL编译阶段,检查查询条件是否有主键或者索引列作为过滤条件，如果没有会产生此异常。如果查询确实需要非主键或非索引列作为过滤条件的列，phoenix4.x版本可以在客户端的hbase-site.xml文件中配置phoenix.force.index为false，重新打开客户端，即可生效。Phoenix5.x在控制台参数管理中修改phoenix.force.index为false，重启HBase SQL服务即可。

13. 通过springboot使用durid连接池报java.sql.SQLException:

java.lang.IllegalArgumentException: Connection is null or closed.

由于Phoenix内部会缓存链接，上层再使用一层连接池时，会出现部分链接被关闭的情况，所以目前不推荐使用此类方式。你可以尝试当前版本自带的连接池，参考问题10。

14. 使用python客户端连接QueryServer时，连接闲置一段时间后再进行读写时报错 phoenixdb.errors.InternalError: ('', None, None, None)

由于开源python客户端未实现连接空闲超时重建机制，通过SLB负载均衡连接超时后再次请求发送到其他QueryServer节点导致。

解决办法：

1. 下载阿里phoenix-python客户端：

   phoenix-python-client下载

2. 如果已安装phoenix-python驱动，需要进行删除

   ```
   rm -rf /usr/lib/python${version}/site-packages/phoenixdb*
   ```

3. 解压后在phoenixdb目录安装驱动

   ```
   python setup.py install
   ```

15、 Phoenix是否支持多租户？

暂时不支持，包括grant，revoke命令。