# Phoenix定义

Phoenix最早是saleforce的一个开源项目，后来成为Apache基金的顶级项目。

Phoenix是构建在HBase上的一个SQL层，能让我们用标准的JDBC APIs而不是HBase客户端APIs来创建表，插入数据和对HBase数据进行查询。

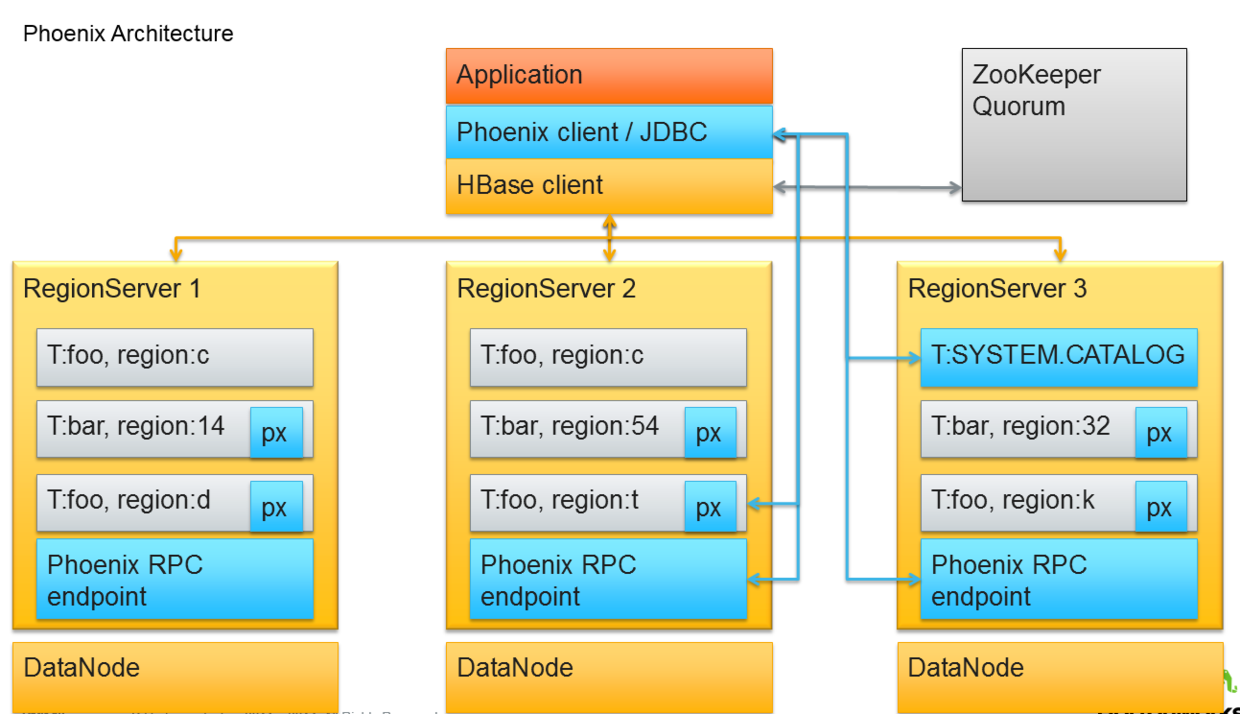
Phoenix完全使用Java编写，作为HBase内嵌的JDBC驱动。Phoenix查询引擎会将SQL查询转换为一个或多个HBase扫描，并编排执行以生成标准的JDBC结果集。直接使用HBase API、协同处理器与自定义过滤器，对于简单查询来说，其性能量级是毫秒，对于百万级别的行数来说，其性能量级是秒。

HBase的查询工具有很多，如：Hive、Tez、Impala、Spark SQL、Phoenix等。

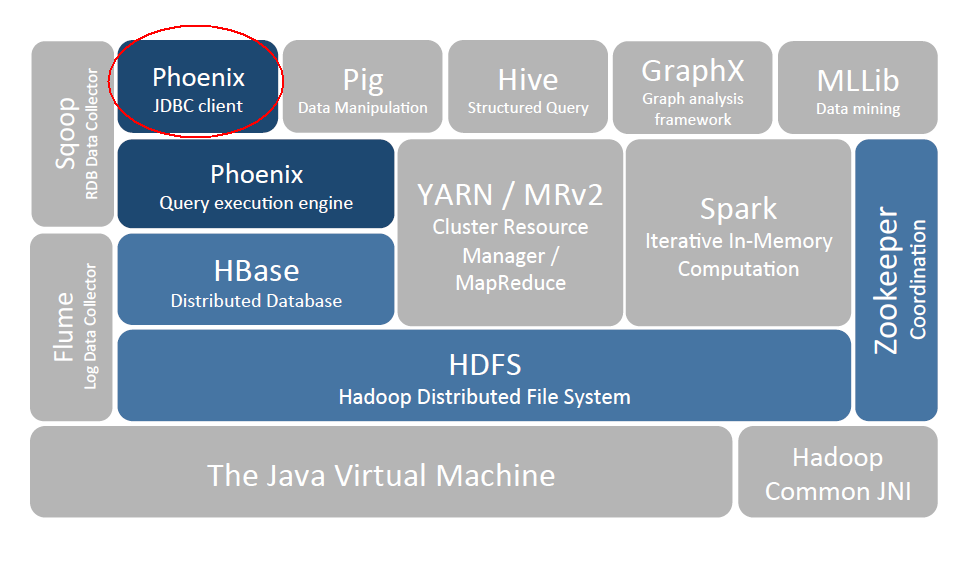
Phoenix通过以下方式使我们可以少写代码，并且性能比我们自己写代码更好：

* 将SQL编译成原生的HBase scans。
* 确定scan关键字的最佳开始和结束
* 让scan并行执行

## Phoenix架构和组成



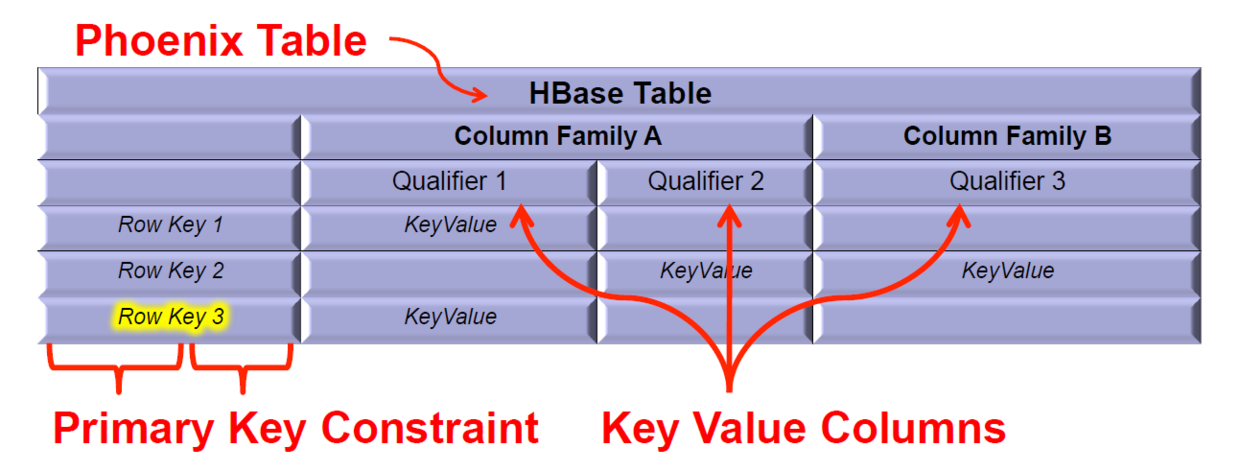
**Phoenix架构图**



**Phoenix在Hadoop生态系统中的位置**

## 数据存储

Phoenix将HBase的数据模型映射到关系型世界



## 特性

### [Transactions (beta)](http://phoenix.apache.org/transactions.html) 事务

该特性还处于beta版，并非正式版。通过集成[Tephra](http://tephra.incubator.apache.org/),Phoenix可以支持ACID特性。Tephra也是Apache的一个项目,是事务管理器，它在像HBase这样的分布式数据存储上提供全局一致事务。HBase本身在行层次和区层次上支持强一致性，Tephra额外提供交叉区、交叉表的一致性来支持可扩展性。

要想让Phoenix支持事务特性，需要以下步骤：

* **配置客户端hbase-site.xml**

<property>

<name>phoenix.transactions.enabled</name>

<value>true</value>

</property>

* **配置服务端hbase-site.xml**

<property>

<name>data.tx.snapshot.dir</name>

<value>/tmp/tephra/snapshots</value>

</property>

<property>

<name>data.tx.timeout</name>

<value>60</value>

<description> set the transaction timeout (time after which open transactions become invalid) to a reasonable value.</description>

</property>

* 配置$HBASE\_HOME并启动Tephra

./bin/tephra

通过以上配置，Phoenix已经支持了事务特性，但创建表的时候默认还是不支持的。如果想创建一个表支持事务特性，需要显示声明，如下：

CREATE TABLE my\_table (k BIGINT PRIMARY KEY, v VARCHAR) TRANSACTIONAL=true;

就是在建表语句末尾增加 TRANSACTIONAL=true。

原本存在的表也可以更改成支持事务的，需要注意的是，事务表无法改回非事务的，因此更改的时候要小心。一旦改成事务的，就改不回去了。

ALTER TABLE my\_other\_table SET TRANSACTIONAL=true;

### [User-defined functions(UDFs)](http://phoenix.apache.org/udf.html) 用户定义函数

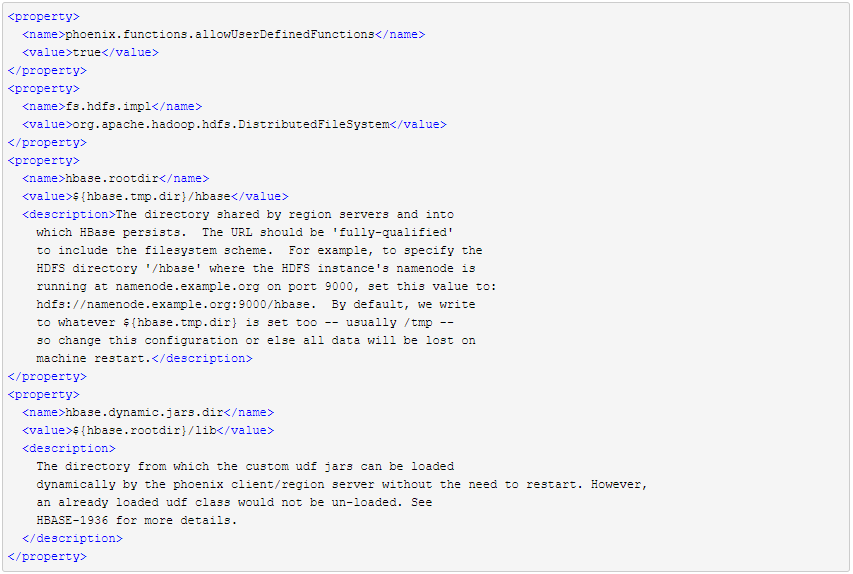
#### 概述

Phoenix从4.4.0版本开始支持用户自定义函数。

用户可以创建临时或永久的用户自定义函数。这些用户自定义函数可以像内置的create、upsert、delete一样被调用。临时函数是针对特定的会话或连接，对其他会话或连接不可见。永久函数的元信息会被存储在一张叫做SYSTEM.FUNCTION的系统表中，对任何会话或连接均可见。

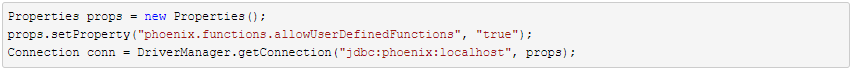
#### 配置

* Hive-site.xml

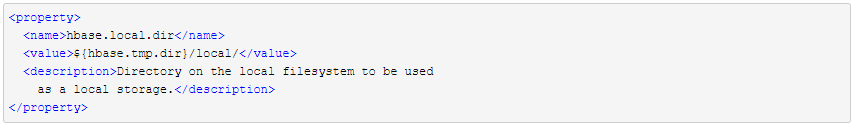


后两个配置需要跟hbse服务端的配置一致。

以上配置完后，在JDBC连接时还需要执行以下语句：



以下是可选的配置，用于动态类加载的时候把jar包从hdfs拷贝到本地文件系统.



### [Secondary Indexing](http://phoenix.apache.org/secondary_indexing.html) 二级索引

在HBase中，只有一个单一的按照字典序排序的rowKey索引，当使用rowKey来进行数据查询的时候速度较快，但是如果不使用rowKey来查询的话就会使用filter来对全表进行扫描，很大程度上降低了检索性能。而Phoenix提供了二级索引技术来应对这种使用rowKey之外的条件进行检索的场景。

* Covered Indexes

只需要通过索引就能返回所要查询的数据，所以索引的列必须包含所需查询的列(SELECT的列和WHRER的列)

* Functional Indexes

从Phoeinx4.3以上就支持函数索引，其索引不局限于列，可以合适任意的表达式来创建索引，当在查询时用到了这些表达式时就直接返回表达式结果

* Global Indexes

Global indexing适用于多读少写的业务场景。  
使用Global indexing的话在写数据的时候会消耗大量开销，因为所有对数据表的更新操作（DELETE, UPSERT VALUES and UPSERT SELECT）,会引起索引表的更新，而索引表是分布在不同的数据节点上的，跨节点的数据传输带来了较大的性能消耗。在读数据的时候Phoenix会选择索引表来降低查询消耗的时间。在默认情况下如果想查询的字段不是索引字段的话索引表不会被使用，也就是说不会带来查询速度的提升。

* Local Indexes

Local indexing适用于写操作频繁的场景。  
与Global indexing一样，Phoenix会自动判定在进行查询的时候是否使用索引。使用Local indexing时，索引数据和数据表的数据是存放在相同的服务器中的避免了在写操作的时候往不同服务器的索引表中写索引带来的额外开销。使用Local indexing的时候即使查询的字段不是索引字段索引表也会被使用，这会带来查询速度的提升，这点跟Global indexing不同。一个数据表的所有索引数据都存储在一个单一的独立的可共享的表中。

### [Statistics Collection](http://phoenix.apache.org/update_statistics.html) 统计信息收集

UPDATE STATISTICS可以更新某张表的统计信息，以提高查询性能

### [Row timestamp](http://phoenix.apache.org/rowtimestamp.html) 时间戳

从4.6版本开始，Phoenix提供了一种将HBase原生的row timestamp映射到Phoenix列的方法。这样有利于充分利用HBase提供的针对存储文件的时间范围的各种优化，以及Phoenix内置的各种查询优化。

### [Paged Queries](http://phoenix.apache.org/paged.html) 分页查询

Phoenix支持分页查询：

Row Value Constructors (RVC)

OFFSET with limit

### [Salted Tables](http://phoenix.apache.org/salted.html) 散步表

如果row key是自动增长的，那么HBase的顺序写会导致region server产生数据热点的问题，Phoenix的Salted Tables技术可以解决region server的热点问题

### [Skip Scan](http://phoenix.apache.org/skip_scan.html) 跳跃扫描

可以在范围扫描的时候提高性能

### [Views](http://phoenix.apache.org/views.html) 视图

标准的SQL视图语法现在在Phoenix上也支持了。这使得能在同一张底层HBase物理表上创建多个虚拟表。

### [Multi tenancy](http://phoenix.apache.org/multi-tenancy.html) 多租户

通过指定不同的租户连接实现数据访问的隔离

### [Dynamic Columns](http://phoenix.apache.org/dynamic_columns.html) 动态列

Phoenix 1.2, specifying columns dynamically is now supported by allowing column definitions to included in parenthesis after the table in the FROM clause on a SELECT statement. Although this is not standard SQL, it is useful to surface this type of functionality to leverage the late binding ability of HBase.

### [Bulk CSV Data Loading](http://phoenix.apache.org/bulk_dataload.html) 大量CSV数据加载

加载CSV数据到Phoenix表有两种方式：1. 通过psql命令以单线程的方式加载，数据量少的情况下适用。 2. 基于MapReduce的bulk load工具，适用于数据量大的情况

### [Query Server](http://phoenix.apache.org/server.html) 查询服务器

Phoenix4.4引入的一个单独的服务器来提供thin客户端的连接

### [Tracing](http://phoenix.apache.org/tracing.html) 追踪

从4.1版本开始Phoenix增加这个特性来追踪每条查询的踪迹，这使用户能够看到每一条查询或插入操作背后从客户端到HBase端执行的每一步。

### [Metrics](http://phoenix.apache.org/metrics.html) 指标

Phoenix提供各种各样的指标使我们能够知道Phoenix客户端在执行不同SQL语句的时候其内部发生了什么。这些指标在客户端JVM中通过两种方式来收集：

**Request level metrics** - collected at an individual SQL statement  
level

**Global metrics** - collected at the client JVM level

## SQL支持

### SELECT

SELECT \* FROM TEST LIMIT 1000;

SELECT \* FROM TEST LIMIT 1000 OFFSET 100;

SELECT full\_name FROM SALES\_PERSON WHERE ranking >= 5.0 UNION ALL SELECT reviewer\_name FROM CUSTOMER\_REVIEW WHERE score >= 8.0

### UPSERT VALUES

UPSERT INTO TEST VALUES('foo','bar',3);

UPSERT INTO TEST(NAME,ID) VALUES('foo',123);

### UPSERT SELECT

UPSERT INTO test.targetTable(col1, col2) SELECT col3, col4 FROM test.sourceTable WHERE col5 < 100

UPSERT INTO foo SELECT \* FROM bar;

### DELETE

DELETE FROM TEST;

DELETE FROM TEST WHERE ID=123;

DELETE FROM TEST WHERE NAME LIKE 'foo%';

### CREATE TABLE

CREATE TABLE my\_schema.my\_table ( id BIGINT not null primary key, date)

CREATE TABLE my\_table (

id INTEGER not null primary key desc,

date DATE not null,

m.db\_utilization DECIMAL,

i.db\_utilization) m.DATA\_BLOCK\_ENCODING='DIFF'

CREATE TABLE stats.prod\_metrics (

host char(50) not null,

created\_date date not null,

txn\_count bigint CONSTRAINT pk PRIMARY KEY (host, created\_date) )

CREATE TABLE IF NOT EXISTS "my\_case\_sensitive\_table" ( "id" char(10) not null primary key, "value" integer) DATA\_BLOCK\_ENCODING='NONE',VERSIONS=5,MAX\_FILESIZE=2000000 split on (?, ?, ?)

CREATE TABLE IF NOT EXISTS my\_schema.my\_table (org\_id CHAR(15), entity\_id CHAR(15), payload binary(1000),CONSTRAINT pk PRIMARY KEY (org\_id, entity\_id) )TTL=86400

### DROP TABLE

DROP TABLE my\_schema.my\_table;

DROP TABLE IF EXISTS my\_table; DROP TABLE my\_schema.my\_table CASCADE;

### CREATE FUNCTION

CREATE FUNCTION my\_reverse(varchar) returns varchar as 'com.mypackage.MyReverseFunction' using jar 'hdfs:/localhost:8080/hbase/lib/myjar.jar'

CREATE FUNCTION my\_reverse(varchar) returns varchar as 'com.mypackage.MyReverseFunction'

CREATE FUNCTION my\_increment(integer, integer constant defaultvalue='10') returns integer as 'com.mypackage.MyIncrementFunction' using jar '/hbase/lib/myincrement.jar'

CREATE TEMPORARY FUNCTION my\_reverse(varchar) returns varchar as 'com.mypackage.MyReverseFunction' using jar 'hdfs:/localhost:8080/hbase/lib/myjar.jar'

### CREATE VIEW

CREATE VIEW "my\_hbase\_table"( k VARCHAR primary key, "v" UNSIGNED\_LONG) default\_column\_family='a';

CREATE VIEW my\_view ( new\_col SMALLINT ) AS SELECT \* FROM my\_table WHERE k = 100; CREATE VIEW my\_view\_on\_view AS SELECT \* FROM my\_view WHERE new\_col > 70;

### DROP VIEW

DROP VIEW my\_view

DROP VIEW IF EXISTS my\_schema.my\_view

DROP VIEW IF EXISTS my\_schema.my\_view CASCADE

### CREATE SCHEMA

CREATE SCHEMA IF NOT EXISTS my\_schema

CREATE SCHEMA my\_schema

### USE

USE my\_schema

USE DEFAULT

### DROP SCHEMA

DROP SCHEMA IF EXISTS my\_schema

DROP SCHEMA my\_schema

# 安装部署

## 安装预编译的Phoenix

* 下载并解压最新版的phoenix-[version]-bin.tar包
* 将phoenix-[version]-server.jar放入服务端和master节点的HBase的lib目录下
* 重启HBase
* 将phoenix-[version]-client.jar添加到所有Phoenix客户端的classpath

## 使用Phoenix

### 命令行

若要在命令行执行交互式SQL语句：

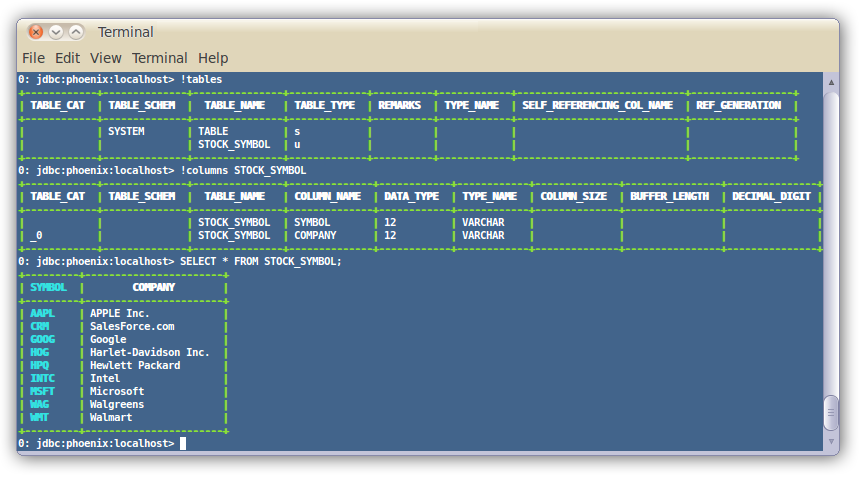
1.切换到bin目录

2.执行以下语句

$ sqlline.py localhost

若要在命令行执行SQL脚本

$ sqlline.py localhost ../examples/stock\_symbol.sql



### 客户端

[SQuirrel](http://squirrel-sql.sourceforge.net/)是用来连接Phoenix的客户端。

SQuirrel安装步骤如下：

1. Remove prior phoenix-[\*oldversion\*]-client.jar from the lib directory of SQuirrel, copy phoenix-[\*newversion\*]-client.jar to the lib directory (\*newversion\* should be compatible with the version of the phoenix server jar used with your HBase installation)

2. Start SQuirrel and add new driver to SQuirrel (Drivers -> New Driver)

3. In Add Driver dialog box, set Name to Phoenix, and set the Example URL to jdbc:phoenix:localhost.

4. Type “org.apache.phoenix.jdbc.PhoenixDriver” into the Class Name textbox and click OK to close this dialog.

5. Switch to Alias tab and create the new Alias (Aliases -> New Aliases)

6. In the dialog box, Name: \*any name\*, Driver: Phoenix, User Name: \*anything\*, Password: \*anything\*

7. Construct URL as follows: jdbc:phoenix: \*zookeeper quorum server\*. For example, to connect to a local HBase use: jdbc:phoenix:localhost

8. Press Test (which should succeed if everything is setup correctly) and press OK to close.

9. Now double click on your newly created Phoenix alias and click Connect. Now you are ready to run SQL queries against Phoenix.

