HBase RowKey与索引设计

设计技巧/原则/案例分享

http://www.nosqlnotes.com

NoSQL漫谈

整体思路

“无聊的话题，但多数HBase开发者却被时常困扰...

第一部分： 15分钟HBase基础知识

HBase基础；RowKey与Region；数据分片方式

第二部分： 合理的需求调研

设计目标；负载特点；查询场景；数据特点

第三部分： RowKey与索引设计

技巧与原则；二级索引；组合索引设计原则

第四部分： RowKey设计案例分享

OpenTSDB/JanusGraph/GeoMesa

目录

提纲：15分钟HBase基础

- 基本概念与数据模型

  KeyValue ; Region; 进程角色; 灵活的列设计

- 快速浏览读写流程

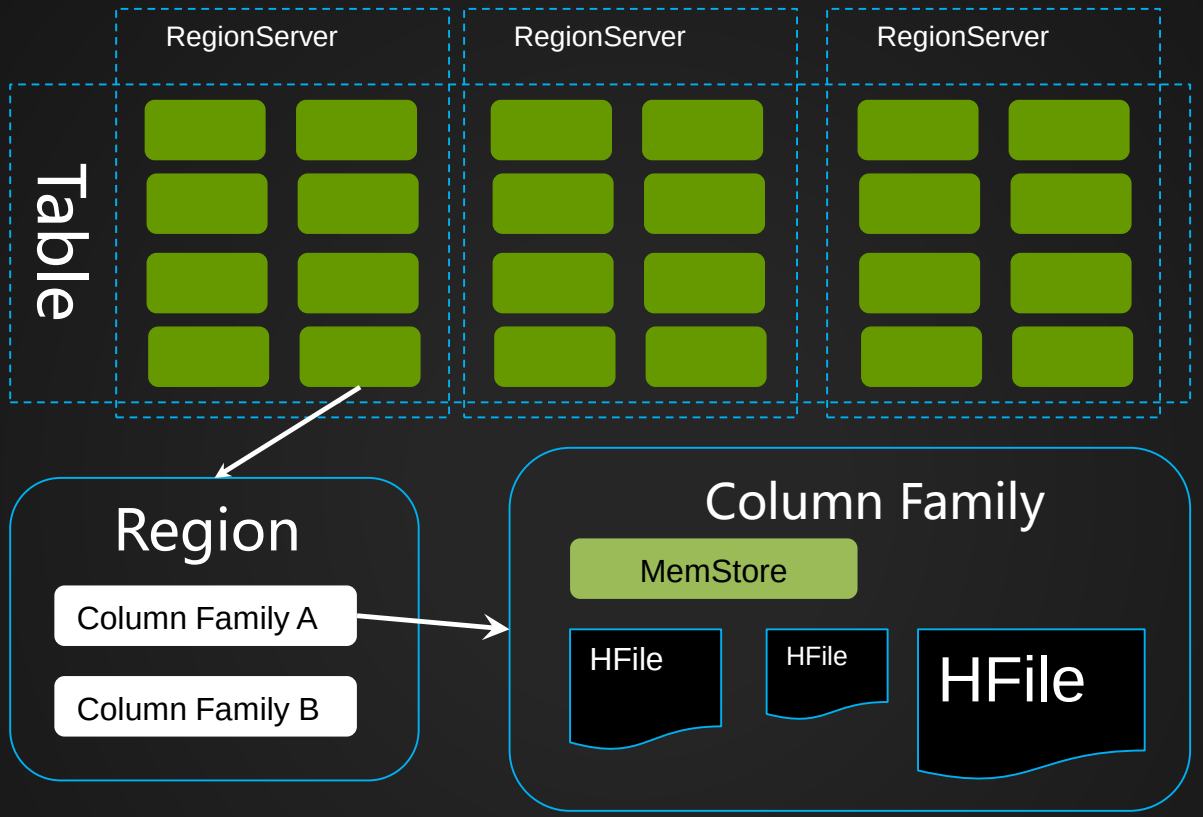
  Write; Flush; Compaction; Read; HFile

- RowKey与索引简介

  RowKey在读写流程中发挥的作用; 索引与RowKey

基本概念

Table: 可理解成传统数据库中的一个表，但因为SchemaLess的设计，它较之传统数据库的表而言，在设计上可以更加灵活



RegionServer: 数据服务进程。Region必须被部署在某一个RegionServer上才可以提供读写服务

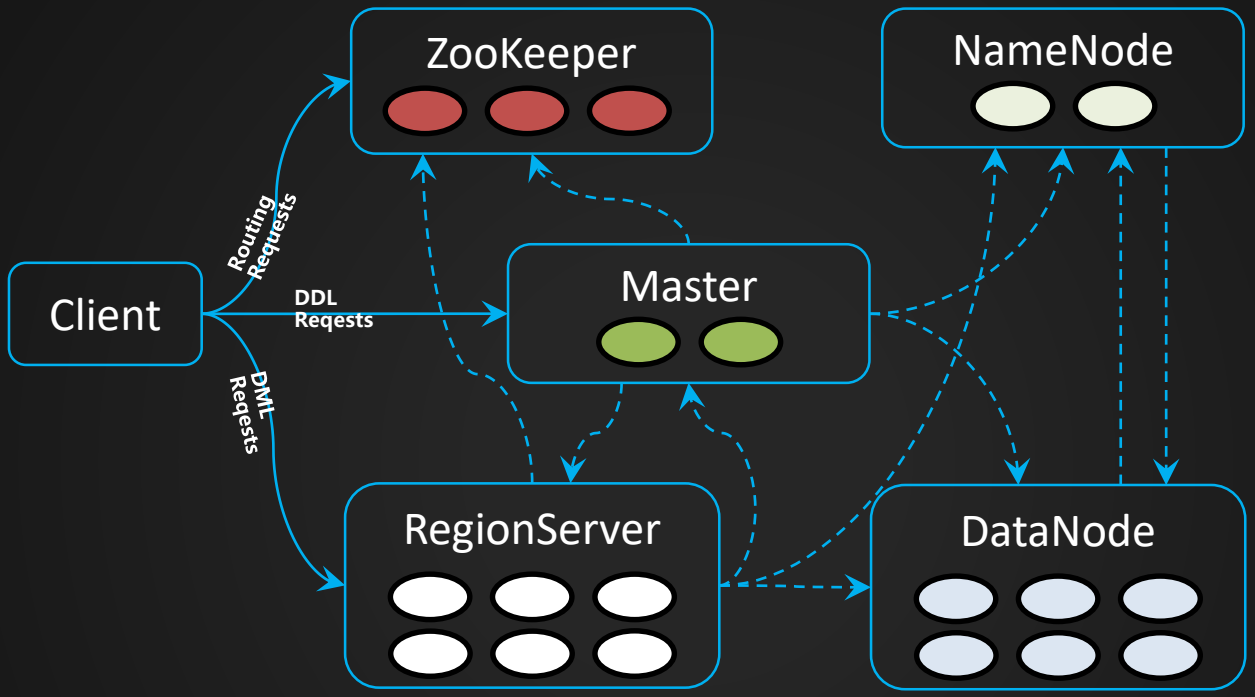
MemStore: 用来在内存中缓存一定大小的数据，达到一定大小后批量写入到底层文件系统中。数据是有序的。

Region：将Table横向切割成一个个子表，子表在HBase中被称之为Region

Column Family: 一些列的集合。不同的Column Family数据文件被存储在不同的路径中

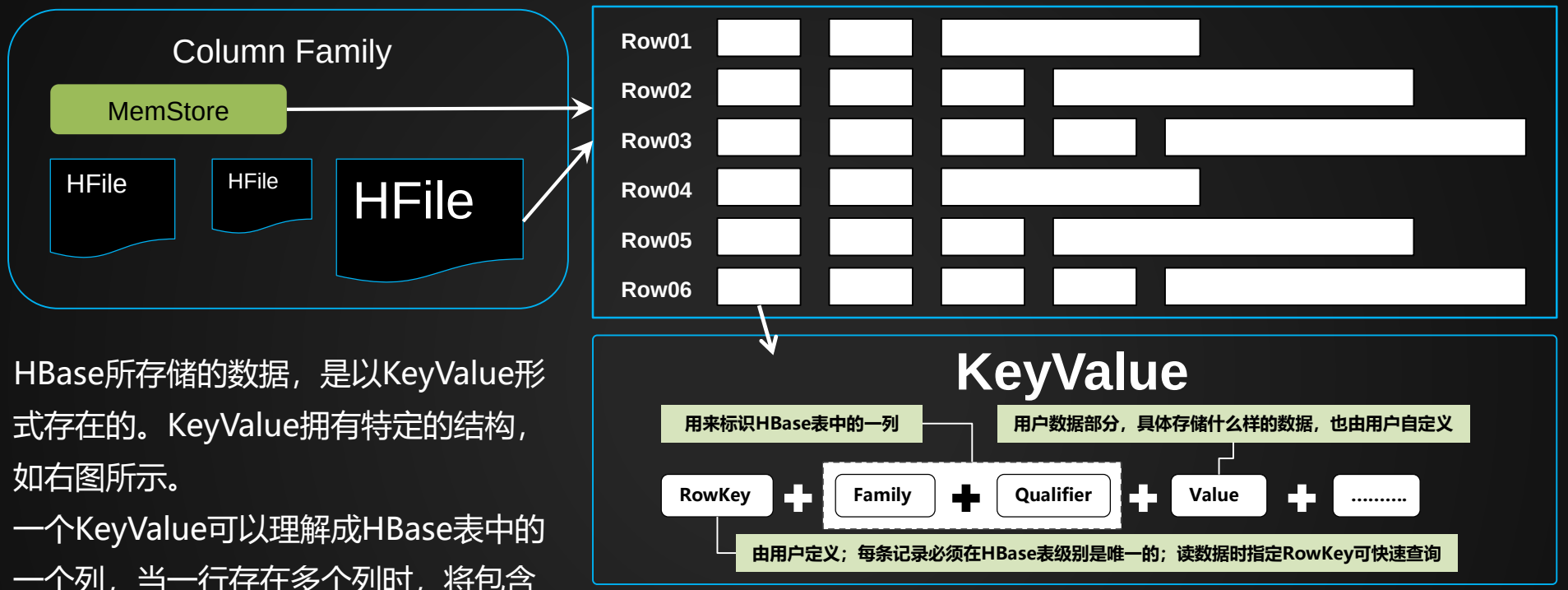
HFile: HBase数据在底层分布式文件系统中的文件组织格式

关键进程角色



Meta表的路由信息在ZooKeeper中；Master负责表管理操作，Region到各个RegionServer的分配以及RegionServer Failover的处理等；RegionServer提供数据读写服务。HBase的所有数据文件都存放在HDFS中。

理解KeyValue

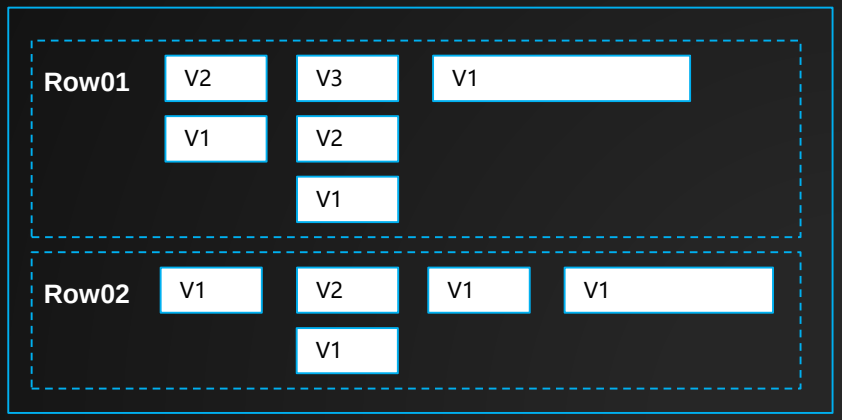


HBase所存储的数据，是以KeyValue形式存在的。KeyValue拥有特定的结构，如右图所示。

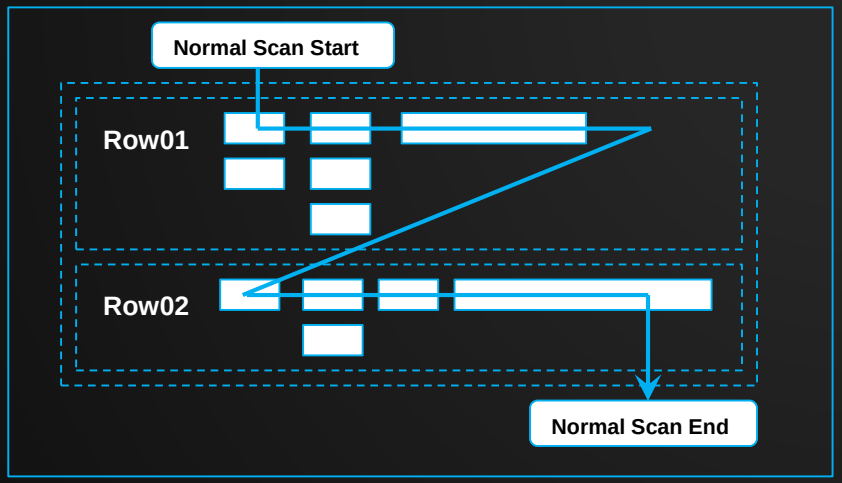
一个KeyValue可以理解成HBase表中的一个列，当一行存在多个列时，将包含多个KeyValue。同一行的KeyValue有可能存在于不同的文件中，但在读取的时候，将会按需合并在一起返回给客户端。

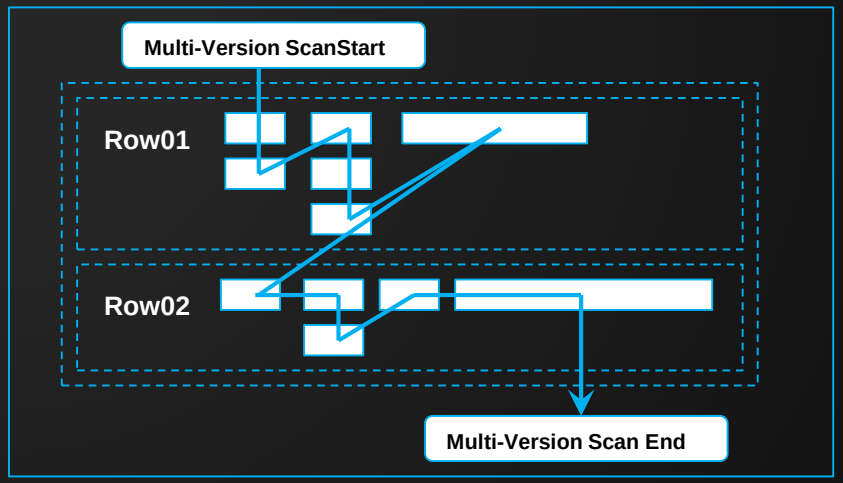
用户写数据时，需要定义用户数据的RowKey，指定每一列所存放的Column Family，并且为其定义相应的Qualifier(列名)，Value部分存放用户数据数据。HBase中每一行可拥有不同的KeyValues，这就是HBase Schema-less的特点。

KeyValue多版本



HBase中支持数据的多版本，通过带有不同时间戳的多个KeyValue版本来实现的。如左图所示。HBase所保存的版本数目是可配置的，默认存放3个版本。在普通的读取流程中，旧版本的数据是不可见的，但通过指定版本数或者版本号的读取，可以获取旧版本数据。下图是普通读取流程与多版本读取流程的对比。





灵活的列定义(1)

00001	{Name->Lily}	{City->GuangZhou}	{Phone->13333333333}	{Gender->Male}
00002	{Name->Siky}	{City->ShenZhen}	{Phone->13423333222}	{Gender->Female}
00003	{Name->Lina}	{City->Shanghai}	{Phone->13322222222}	{Gender->Male}
00004	{Name->Susan}	{City->JiNan}	{Phone->14233884444}	{Gender->Male}
……..	{Name->……..}	{City->……..}	{Phone-> ……..}	{Gender->……}
99999	{Name->Jane}	{City->XiaMen}	{Phone->18666666666}	{Gender->Male}

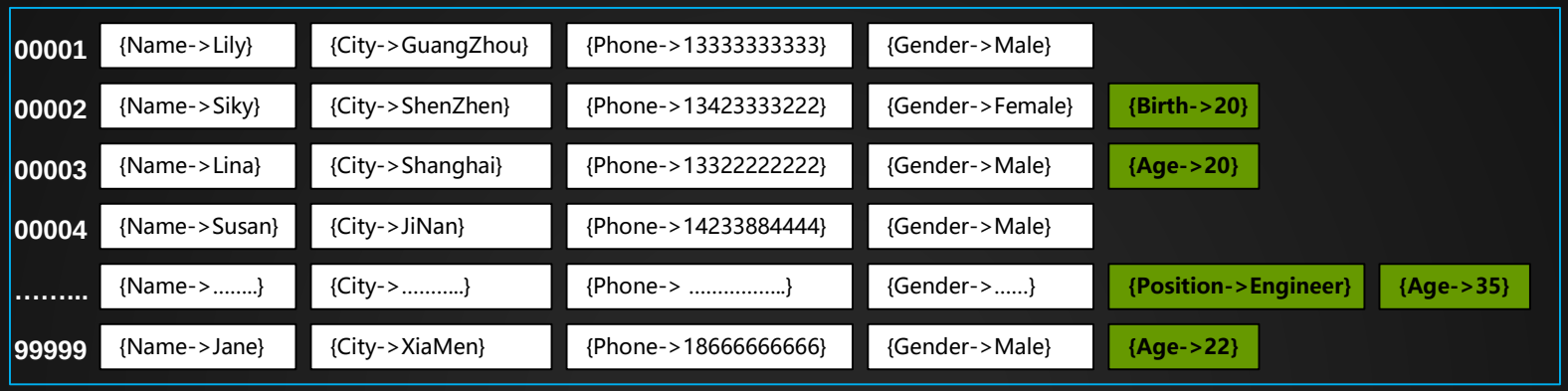
图1 最简单的HBase列设计

00001	{Name->Lily}	{Val->GuangZhou,13333333333,Male}
00002	{Name->Siky}	{Val->ShenZhen,134233333222,Female}
00003	{Name->Lina}	{Val->Shanghai,13322222222,Male}
00004	{Name->Susan}	{Val->JiNan,14233884444,Male}
……..	{Name->……..}	{Val->……..}
99999	{Name->Jane}	{Val->XiaMen,18666666666,Male}

图2 基于组合列存储的设计

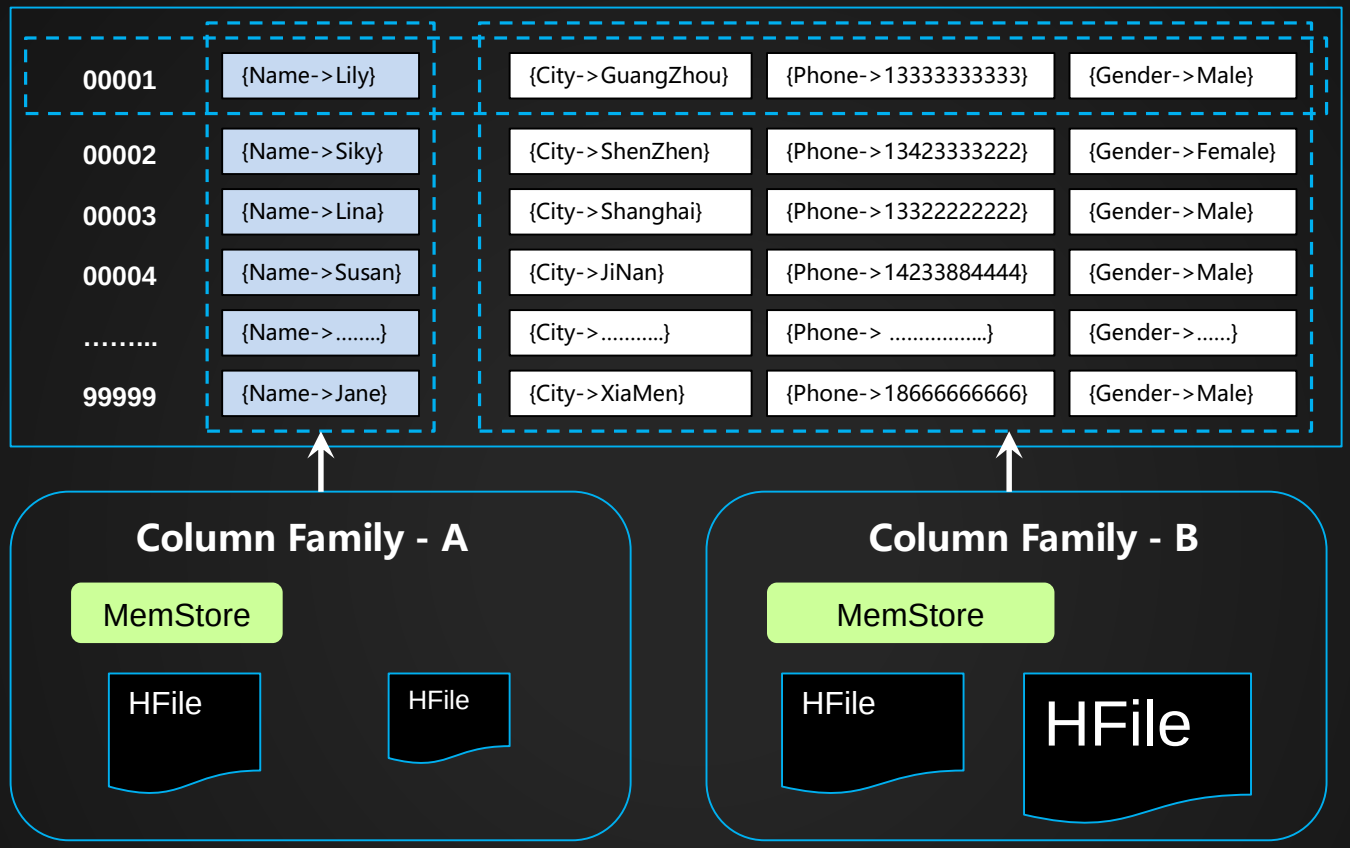
用户数据存入到HBase表中时，需要进行Qualifier(KeyValue/列)设计。一个最简单的设计是保持HBase的列与用户数据的列一致，如上图1的设计。这种设计，基本上与关系型数据库的设计是一致的，但这种设计会带来较大的数据冗余(KeyValue结构化开销)。但HBase基于KeyValue的接口，决定了这种设计可以是非常灵活的，例如，我们也可以考虑为HBase的每一行只设置两个列，其中，Name为一个列，其它内容合并到一个列中，如上图2所示。

灵活的列定义(2)



尽管我们在使用HBase表存放数据的时候，需要预先做好列的设计。但这个设计仅仅由应用层感知，HBase并没有存放任何的Schema信息来描述这个设计。也就是说，应用层需要知道为每一个表/每一行设计了什么样的列（KeyValue），然后在读取的时候做相应的解析。既然HBase中并没有Schema信息，那么，每一行中的列，也可以是任意添加的。如上图所示，绿色背景的KeyValue为后续增加的。

Column Family

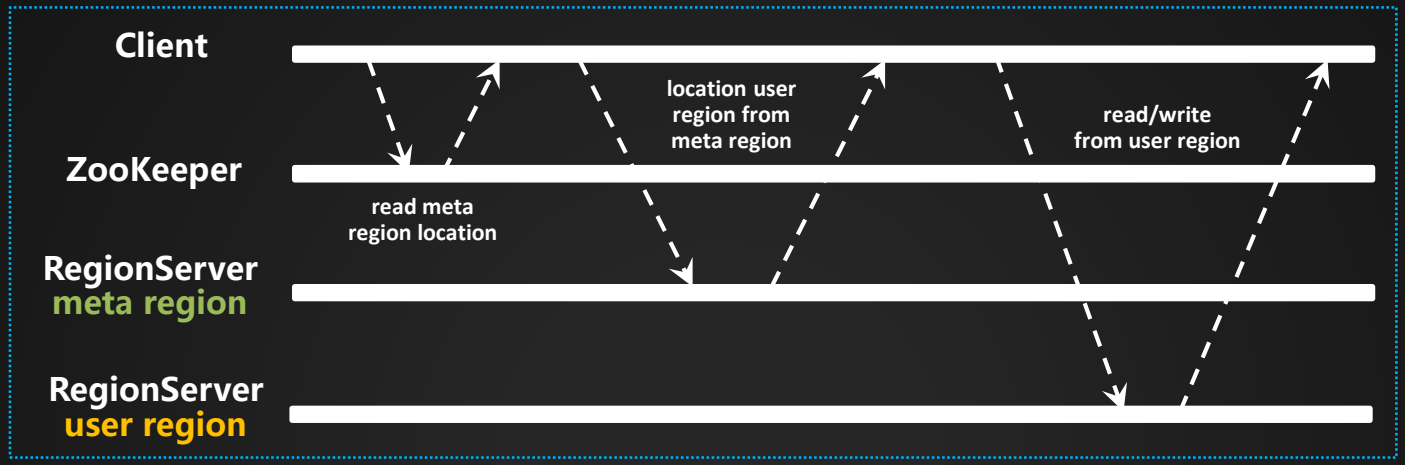


假设为表设置了两个列族，而且，定义了每一个列族中要存放的列，如左图所示：

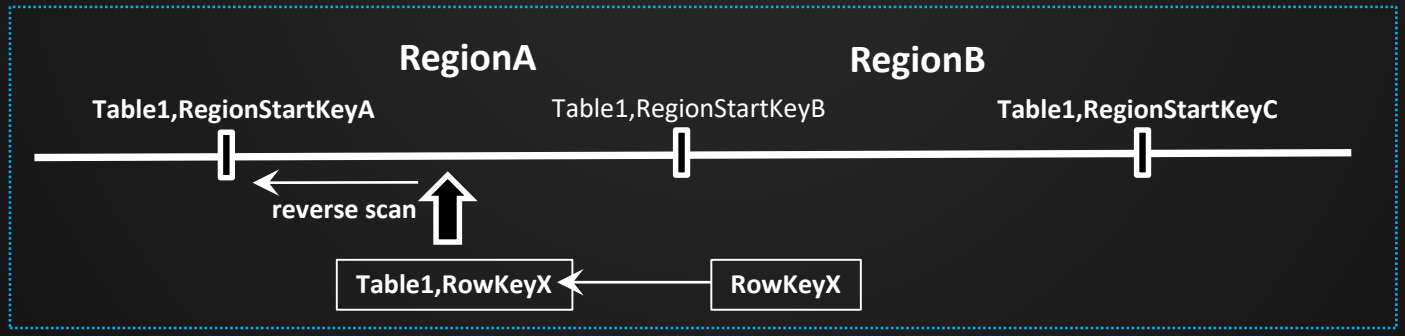
{Name} -> Column Family-A,　{City, Phone, Gender} -> Column Family-B

不同列族的数据会被存储在不同的路径中。即，设置多个列族时一行数据可能存在于两个路径中。整行读取的时候，需要将两个路径中的数据合并在一起才可以获取到完整的一行记录。但如果仅仅读取Name一列的话，只需要读取Column Family-A即可。

读写数据路由机制



Client获取Region路由信息的流程

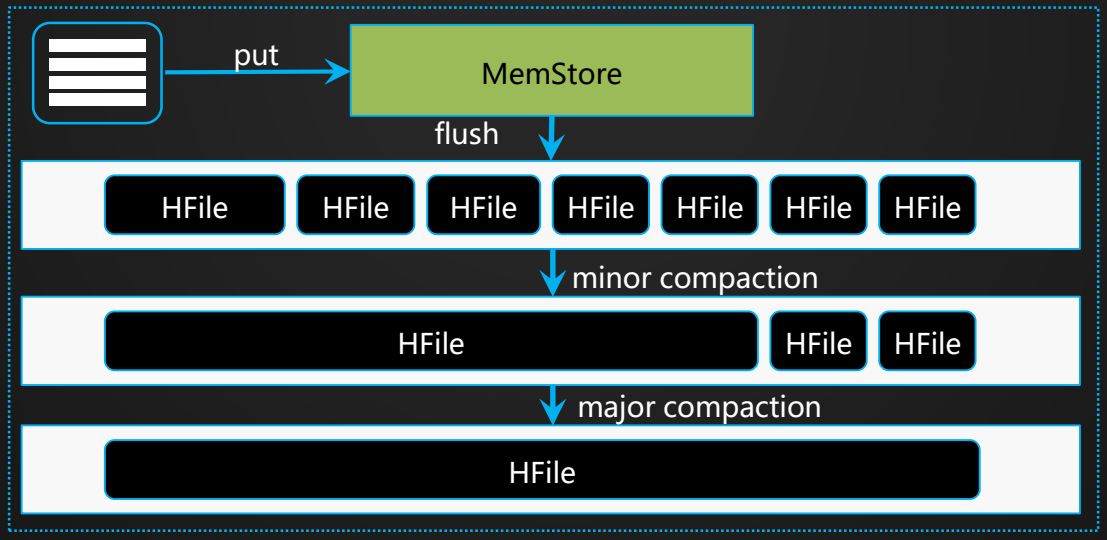


基于RowKey从Meta表定位关联Region的方法

写入流程

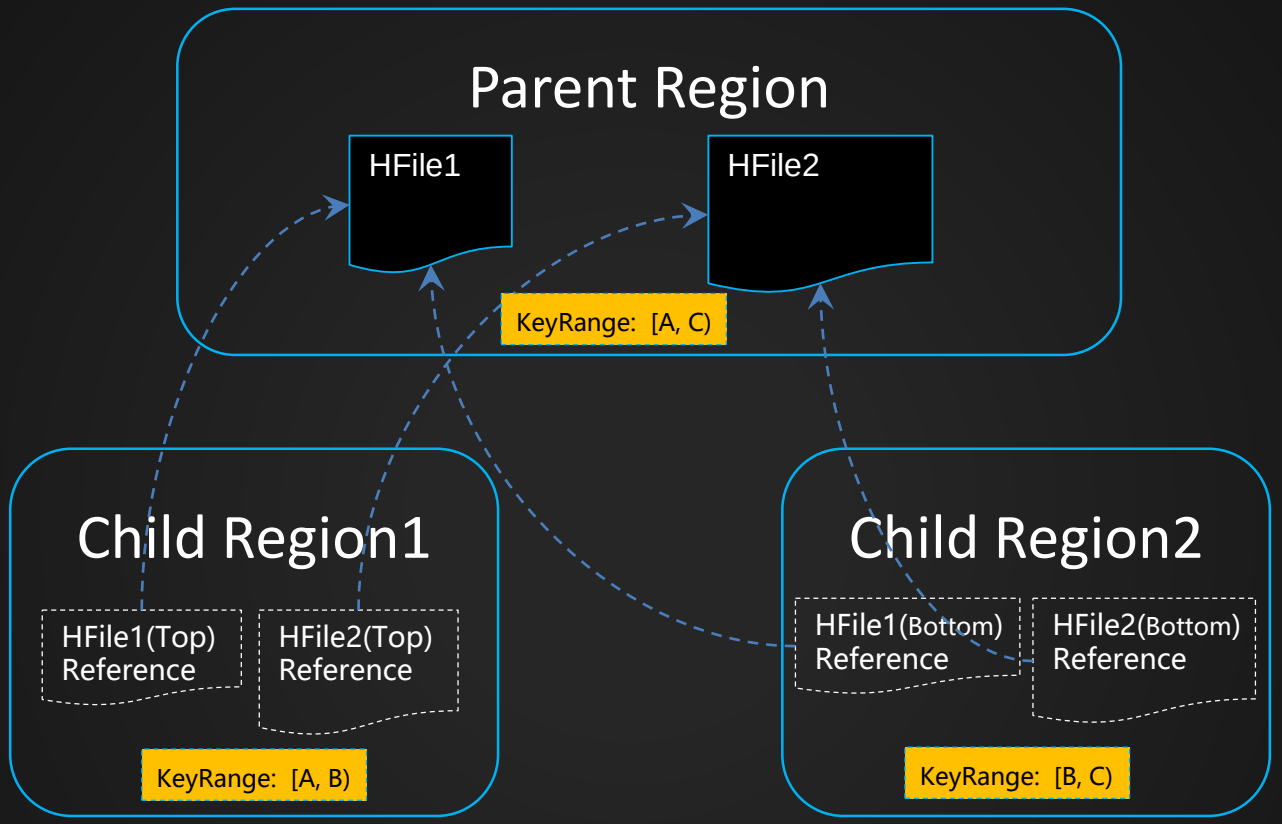
假设In-memory Flush&Compaction未开启

RegionServer侧写入



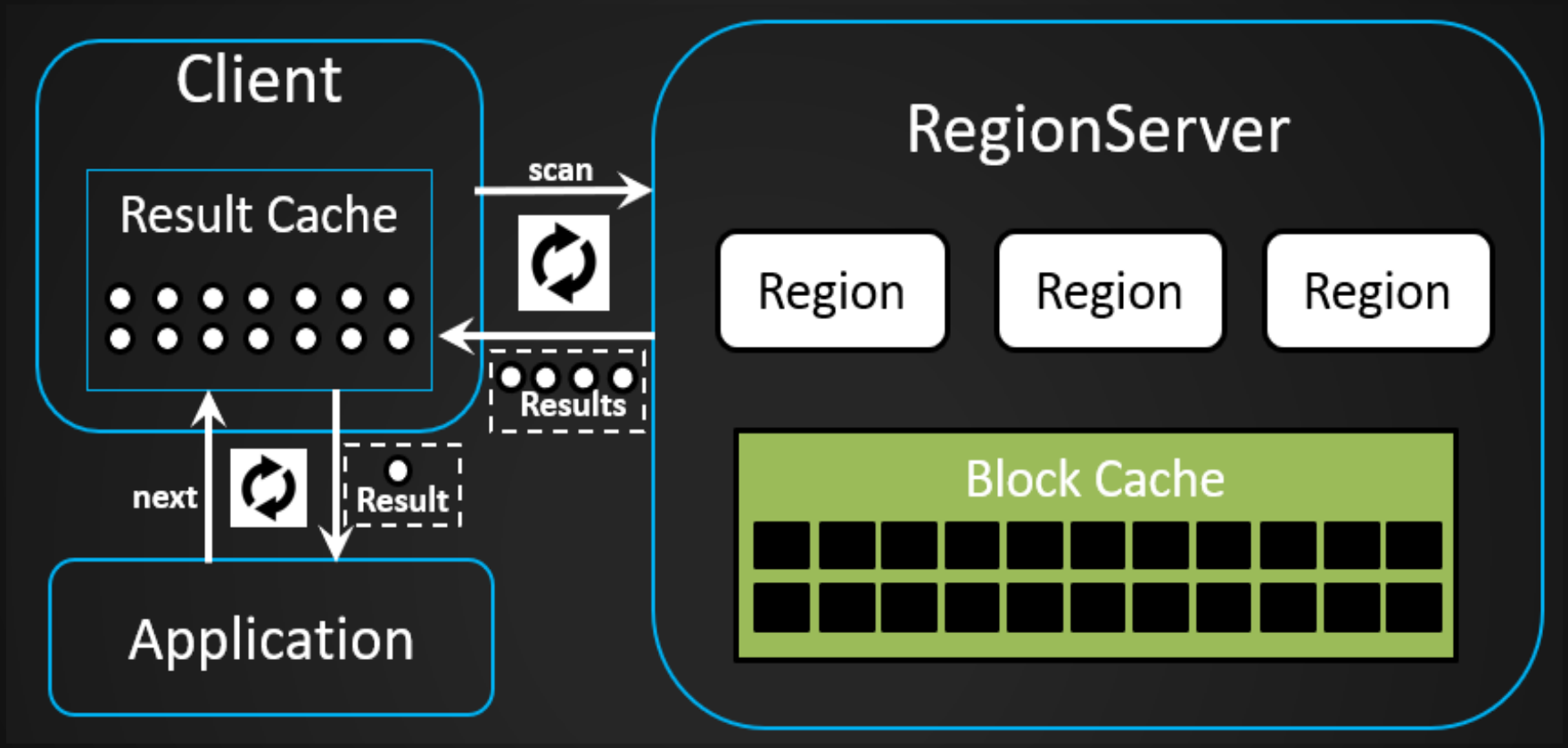
Flush & Compaction

Region Split



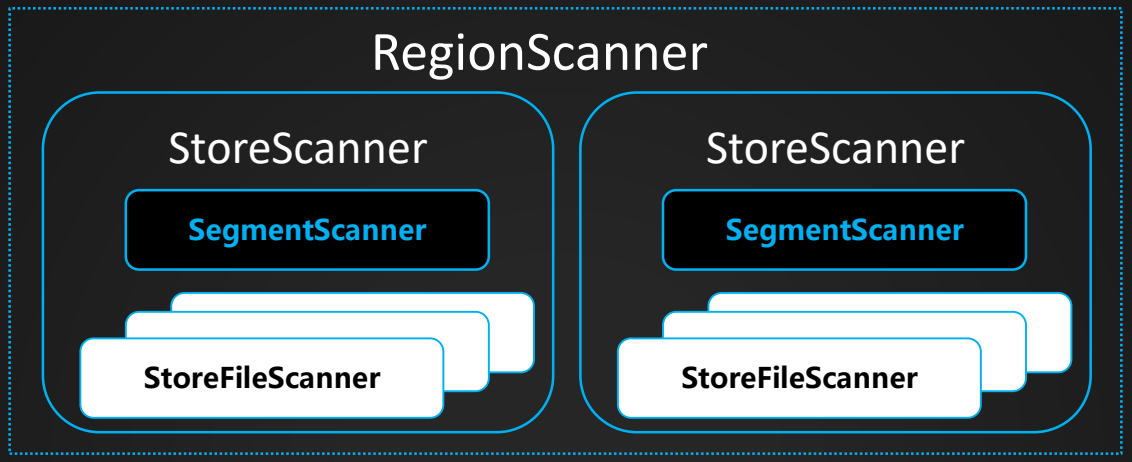
当一个Region变的过大后，会触发Split操作，将一个Region分裂成两个子Region。Region Split过程并不会真正将父Region中的HFile数据搬到子Region目录中。Split过程仅仅是在子Region中创建了到父Region的HFile的引用文件，子Region1中的引用文件指向原HFile的上部，而子Region2的引用文件指向原HFile2的下部。数据的真正搬迁工作是在Compaction过程完成的。

读取流程

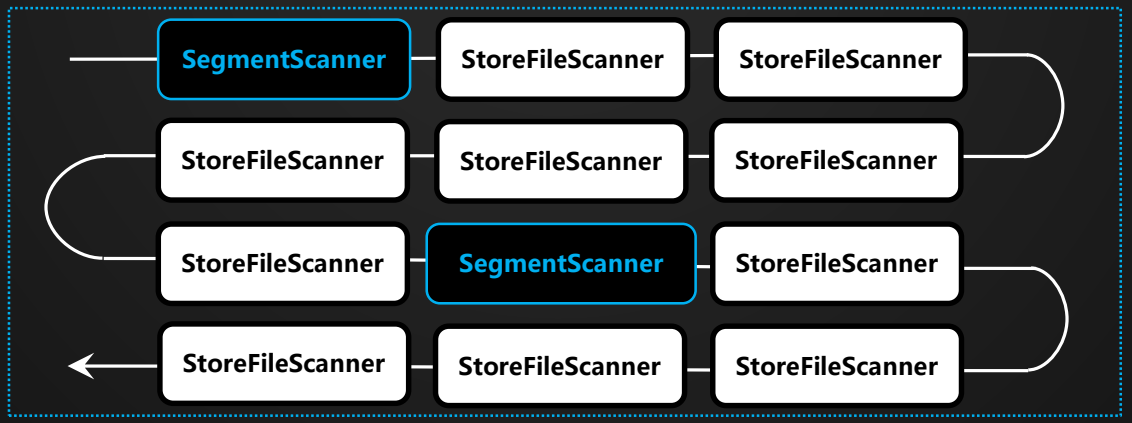

Client
Result Cache
scan
Results
next
Result
Application
RegionServer
Region
Region
Region
Block Cache

读取流程 – Scanner抽象

假设In-memory Flush&Compaction未开启

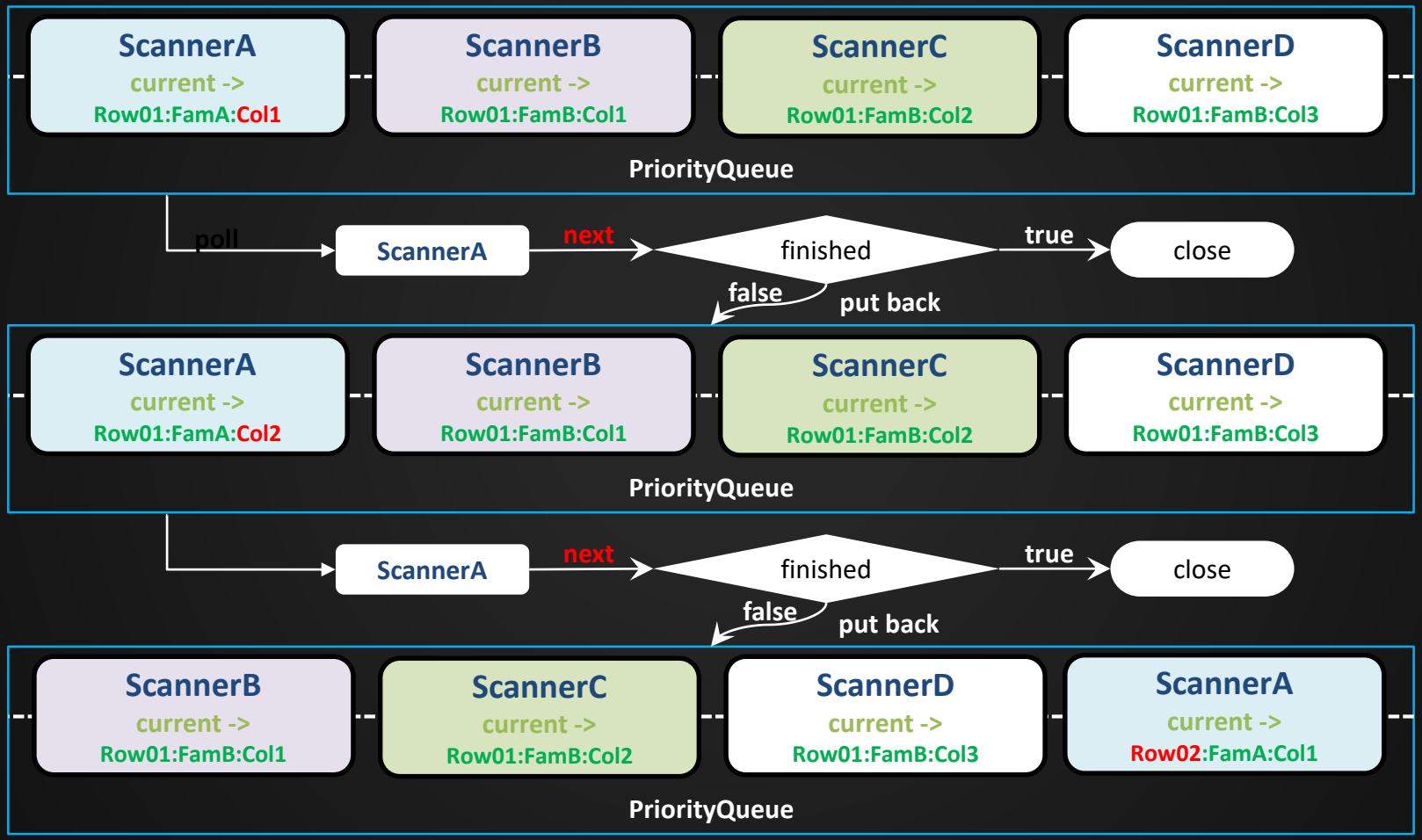


RegionScanner的抽象

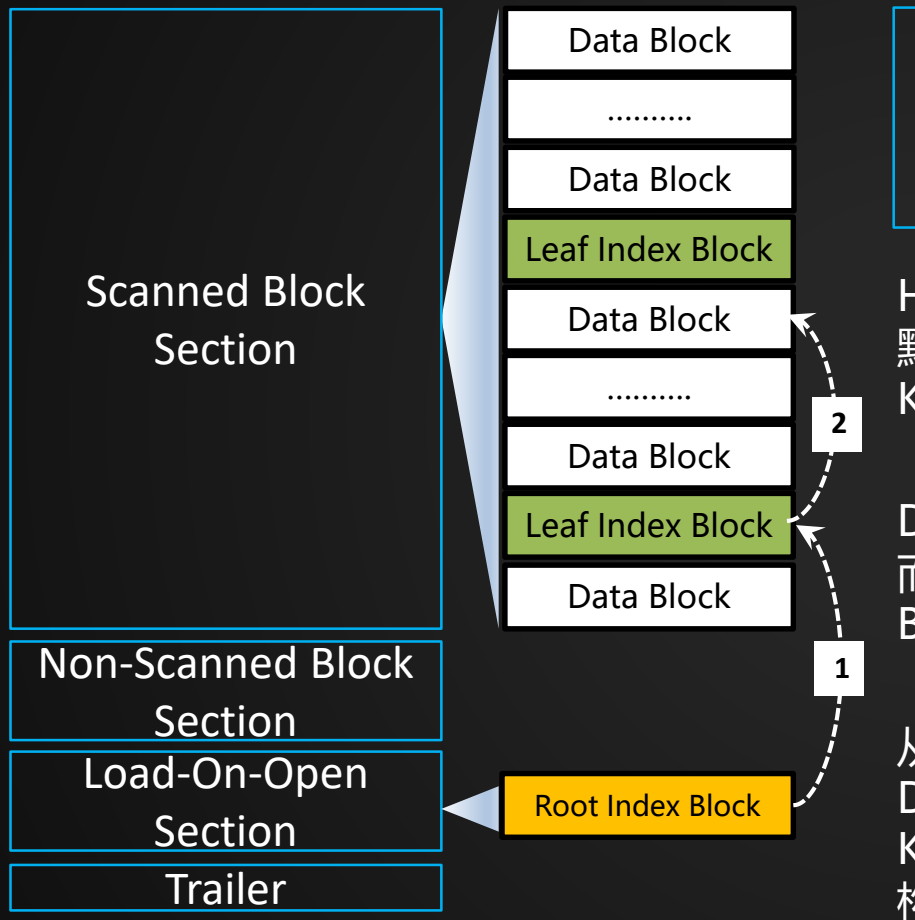


多个Scanner被组织在一个优先级队列中

读取流程 – Scaner Next请求调用



读取流程 – Scaner Next请求调用



HFile中的数据按Block组织，一个Data Block的默认大小为64KB，Data Block中直接存储了KeyValue信息。

Data Block的索引信息存储在Leaf Index Block中。而Leaf Index Block的信息存储在Root Index Block中(不考虑Intermediate Index Block情形)。

从Root Index Block到Leaf Index Block再到Data Block，以及从Data Block到用户数据KeyValue的数据组织，正是一种典型的B+ Tree结构。

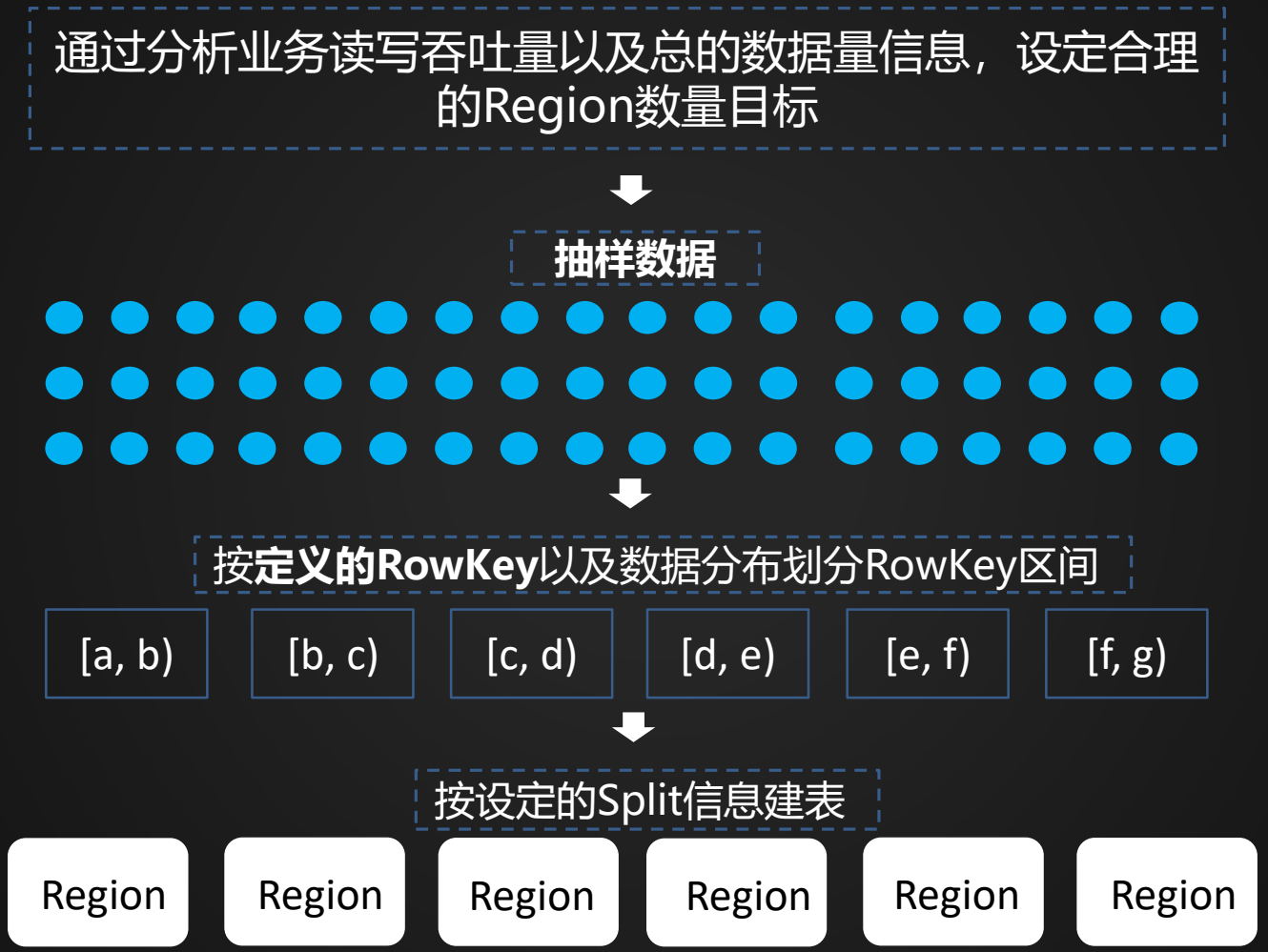
RowKey在读写流程中发挥的作用

回顾整个读写流程：

- 读写数据时通过RowKey路由到对应的Region
- MemStore中的数据按RowKey排序
- HFile中的数据按RowKey排序

> **HBase的数据排序方式**
>
> HBase中的数据按RowKey的字典数据存放，下面的例子将帮助你理解字典排序的原理：
>
> *RowKey列表：*
> *{“abc”，“a”，“bdf”，“cdf”，“defg”}*
> *按字典排序后的结果为：*
> *{“a”，“abc”，“bdf”，“cdf”，“defg”}*
>
> 也就是说，当两个RowKey进行排序时，先对比两个RowKey的第一个字节，如果相同，则对比第二个字节，依此类推...如果在对比到第M个字节时，已经超出了其中一个RowKey的字节长度，那么，短的RowKey要被排在另外一个RowKey的前面

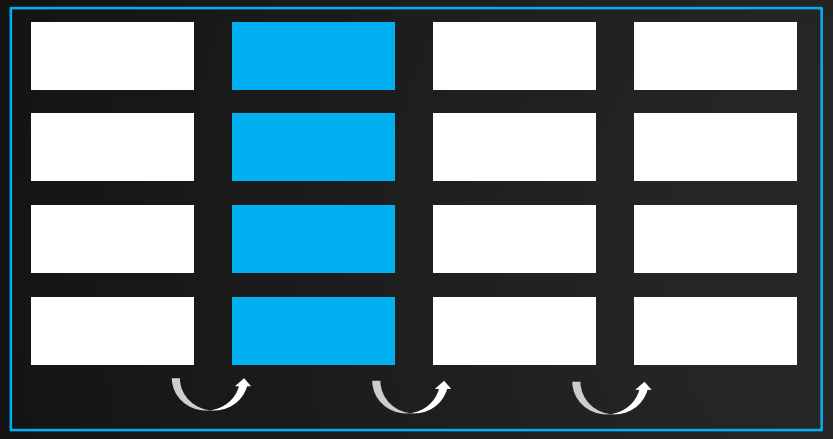
RowKey设计直接关乎Region的划分



RowKey对Compaction的影响

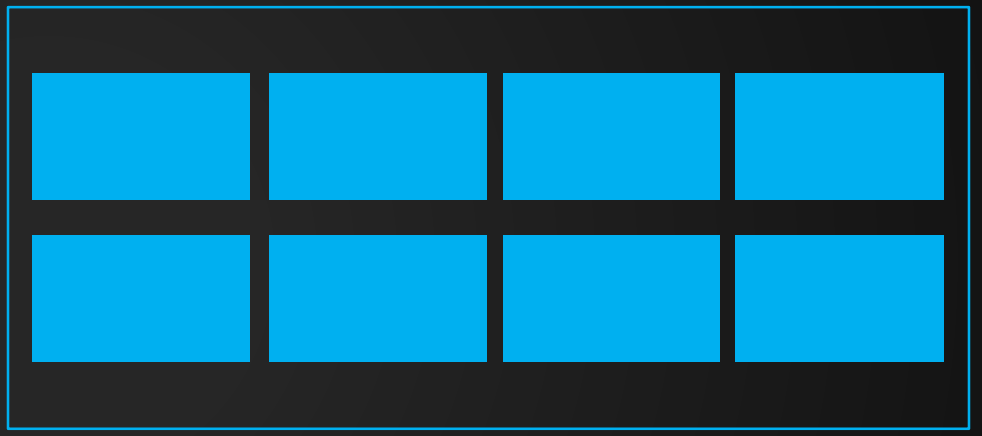
处于写活跃状态的Region

RowKey: DayInfo + ID



随着时间的推移，写活跃的Region会出现变化。Major Compaction的数据总量随着时间的推移呈现周期性波动

RowKey: ID



任意时刻，所有Region都是写活跃的。Major Compaction的数据总量随着时间的推移不断增大。

RowKey对查询的影响

Name	Phone	ID	.....
Ariya	1332222	I0000005	....
...	...	...	....
Li	1384444	I9999999	....
Li	1385555	I0000008	....
Wang	1332323	I0000007	....
Wang	1333232	I0000002	....
...	...	...	....
Wang	1334988	I0012322	....
Wang	1335465	I0040007	....
Wang	1366134	I0070007	....
Wang	1397323	I0052322	....
Wang	1388866	I0112322	....
Wang	1388888	I0900007	....
...	...	...	....
Xiao	1332222	I0000009	....

基于Name+Phone+ID构建RowKey

可很好/较好支持的查询场景：

查询场景1：Name:Li AND Phone:1384444 AND ID:I9999999
查询场景2：Name:Wang AND Phone:1384444
查询场景3：Name:Wang
查询场景4：Name:Wang AND Phone: 133%
查询场景2：Name:Wang AND Phone: 133?323

难以支持的查询场景：

查询场景5： Phone: 1384444
查询场景6： ID: I9999999

RowKey即索引...

数据库查询可简单分解为两个步骤：

1）**键的查找**　2）**数据的查找**

因这两种数据组织方式的不同，在RDBMS领域有两种常见的数据组织表结构：

索引组织表：键与数据存放在一起，查找到键所在的位置则意味着查找到数据本身。

堆表：键的存储与数据的存储是分离的。查找到键的位置，只能获取到数据的物理地址，还需要基于该地址去获取数据。

HBase数据表其实是一种**索引组织表结构**：查找到**RowKey**所在的位置则意味着找到数据本身。因此，**RowKey本身就是一种索引**。

RowKey查询的局限性/二级索引需求背景

Name	Phone	ID	.....
Ariya	1332222	I0000005	....
...	...	...	....
Li	1384444	I9999999	....
Li	1385555	I0000008	....
Wang	1332323	I0000007	....
Wang	1333232	I0000002	....
...	...	...	....
Wang	1334988	I0012322	....
Wang	1335465	I0040007	....
Wang	1366134	I0070007	....
Wang	1397323	I0052322	....
Wang	1388866	I0112322	....
Wang	1388888	I0900007	....
...	...	...	....
Xiao	1332222	I0000009	....

基于Name+Phone+ID构建RowKey

如果提供的查询条件能够**尽可能丰富**的描述RowKey的**前缀信息**，则**查询时延**越能得到保障。如下面几种组合条件场景：

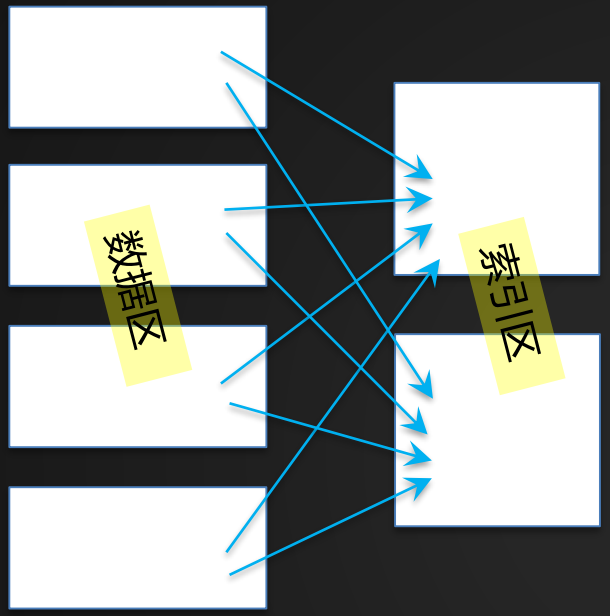
- *Name + Phone + ID*
- *Name + Phone*
- *Name*

如果查询条件不能提供Name信息，则RowKey的前缀条件是无法确定的，此时只能通过**全表扫描**的方式来查找结果。

一种业务模型的用户数据RowKey，只能采用单一结构设计。但事实上，查询场景可能是多维度的。例如，在上面的场景基础上，还需要单独基于Phone列进行查询。这是HBase二级索引出现的背景。即，二级索引是为了让HBase能够提供更多维度的查询能力。

注：HBase原生并不支持二级索引方案，但基于HBase的KeyValue数据模型与API，可以轻易的构建出二级索引数据。Phoenix提供了两种索引方案，而一些大厂家也都提供了自己的二级索引实现。

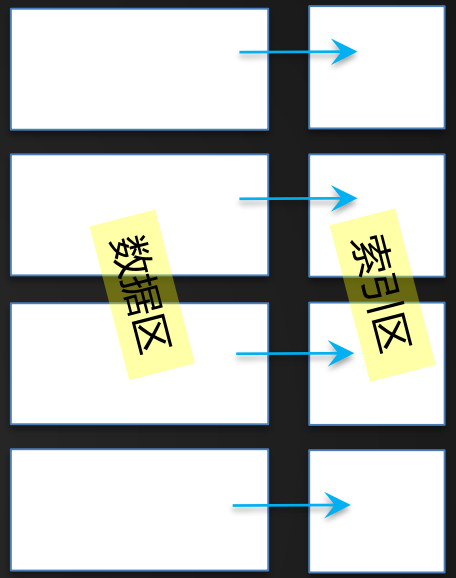
常见的两种二级索引



方案优点：数据按索引字段全部排序，基于索引字段的小批次查询性能高。能够支持更大的查询并发数。方法的复杂度较低。

方案缺点：生成索引数据对实时写入的影响较大。

不同用户Region的索引数据混杂在一起



方案优点：每一个用户Region都拥有独立的索引数据，目前最佳的实践是将这部分索引数据存放于一个**独立的列族**中。

方案缺点：按索引字段进行查询时，需要访问所有的索引Region，随着数据量和Region数目的不断增多，查询时延无保障，查询所支持的并发数也会降低。

每个用户Region索引数据独立存在

索引即RowKey...

举例说明如何基于HBase原生API构建二级索引的方法：

假设原始数据的RowKey与列定义如下：

Put: *RowKey -> ID ， 列 -> {Name, Phone, Address, ...}*

如果我们希望基于Phone列构建二级索引，我们可以基于原数据，构建一条新的记录，这个记录的RowKey为：

Put: *RowKey -> Phone + ID, 列 -> {...}*

在二级索引的RowKey中，包含了数据的RowKey信息。这样，既能有效的从Phone索引到ID信息，又能保证索引RowKey的唯一性。

构建二级索引的核心在于**如何设计一个合理的索引RowKey**。

更多HBase基础内容...

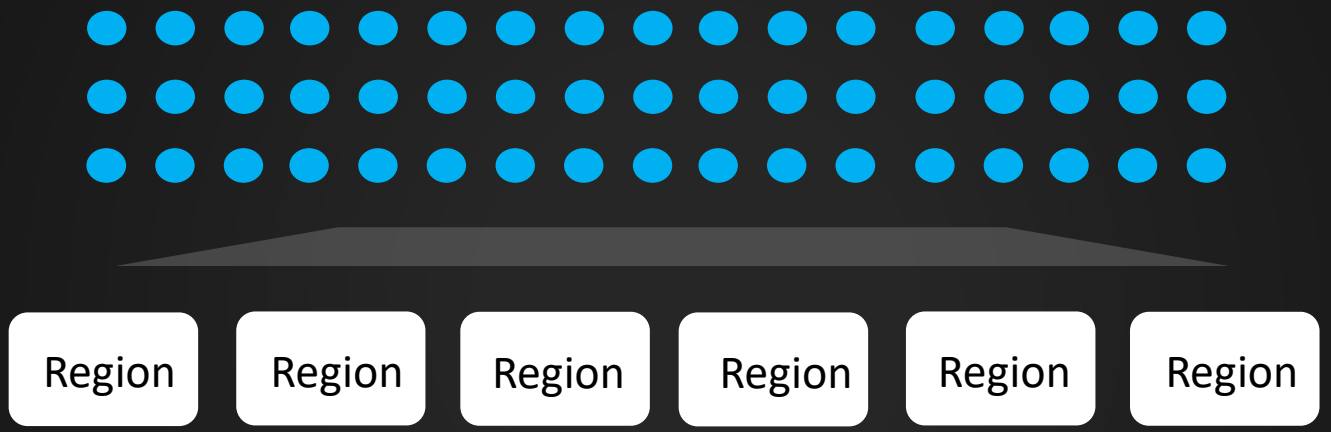
《一条数据的HBase之旅》系列文章：

1. 开篇基础内容
2. HBase写数据流程
3. Flush与Compaction
4. HBase读取流程
5. HFile原理剖析

...(更多分享内容敬请期待)...

目录

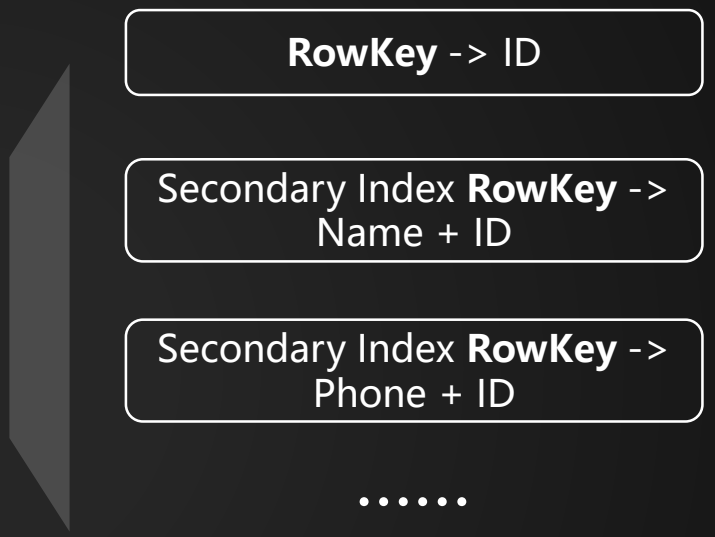
RowKey设计的目标



结合业务特点，将数据合理的分配到每一个Region中，从而很好的满足业务的读写需求

索引设计的目标

ID	Name	Phone	.....
I0000005	Ariya	1332222	....
...	...	...	....
I9999999	Lina	1384444	....
I0000003	Lina	1385555	....
I0000008	Lisa	1385555	....
I0000007	Wang	1332323	....
I0000002	Wang	1333232	....
...	...	...	....
I0000009	Xiao	1332222	....



为HBase提供更多维度的查询能力。在实际应用中应该通过构建尽量少的索引，来满足更多的查询场景

需求调研的关键维度

负载特点

查询场景

数据特点

负载特点

读写TPS；读写比重

重写轻读？重读轻写？读写相当？

数据负载均衡与高效读取时常是矛盾的。

在**重读轻写**的大数据场景中，RowKey设计应该更侧重于如何高效读取。

而在**重写轻读**的大数据场景中，在满足基本查询需求的前提下，应该更关注整体的吞吐量，这就对数据的负载均衡提出了很高的要求。

查询场景

需要支持哪些查询场景？时延要求？

最高频的查询场景是什么？

最有价值的数据排序场景是什么

是否有其它维度的价值查询场景？频度？

是否是组合字段场景？

各个字段的匹配类型？

Equal? Prefix Match? Wildcard? Text-Search?

数据特点

查询条件字段的离散度信息?

字段离散度的定义:

字段A的离散度 = (字段A的可能的枚举值数目)/数据总记录条数

查询条件字段的数据分布特点?

数据分布影响RowKey的设计，更进一步影响如何合理的划分Region信息

数据生命周期?

影响到一个表的一次Major Compaction发生时涉及到的最大数据量

目录

影响查询性能的关键因素

ID	NAME	PROVINCE	GENDER	PHONE	AGE
I0000001	Lily	Shandong	MALE	13322221111	20
I0000002	Wang	Guangdong	FEMAIL	13222221111	15
I0000003	Jaison	Shandong	FEMAIL	13522221111	13
I0000004	He	Henan	MALE	13333331111	18
I0000005	Ariya	Hebei	FEMAIL	13344441111	28
I0000006	Bai	Hunan	MALE	15822221111	30
I0000007	Wang	Hubei	FEMAIL	15922221111	35
I0000008	Jaison	Heilongjiang	MALE	15844448888	38
I0000009	Xiao	Jilin	MALE	13802514000	38
.....	.....	.....	.....	.....	....
I0099999	Jaison	LiaoNing	MALE	188776633444	28
.....	.....	.....	.....	.....	....
I9999999	Tan	Liaoning	MALE	13955225522	70

基于某一个索引/RowKey进行查询时，影响查询的最关键因素在于**能否将扫描的候选结果集限定在一个合理的范围内。**

知识点备注：查询驱动条件与查询过滤条件：直接影响数据扫描范围的查询条件，称之为**查询驱动条件**。而其它的能够起到过滤作用的查询条件，则称之为**查询过滤条件**。影响查询的关键因素在于如何合理的设置查询驱动条件。

RowKey字段的选取

遵循的**最基本原则**：

唯一性： RowKey必须能够唯一的识别一行数据

无论应用是什么样的负载特点，RowKey字段都应该参考**最高频的查询场景。**数据库通常都是以如何高效的读取和消费数据为目的，而不是数据存储本身。

而后，结合具体的负载特点，再对选取的RowKey字段值进行改造，组合字段场景下需要重点考虑**字段的顺序**。

避免数据热点的方法 - Reversing

如果经初步设计出的RowKey在数据分布上不均匀，但RowKey尾部的数据却呈现出了良好的随机性，此时，可以考虑将RowKey的信息翻转，或者直接将尾部的bytes提前到RowKey的前部。

后四位提前

完全翻转

缺点：更场景利于Get但不利于Scan，因为数据在原RowKey上的自然顺序已被打乱

避免数据热点的方法 - Salting

Salting的原理是在原RowKey的前面添加固定长度的随机bytes，随机bytes能保障数据在所有Regions间的负载均衡。

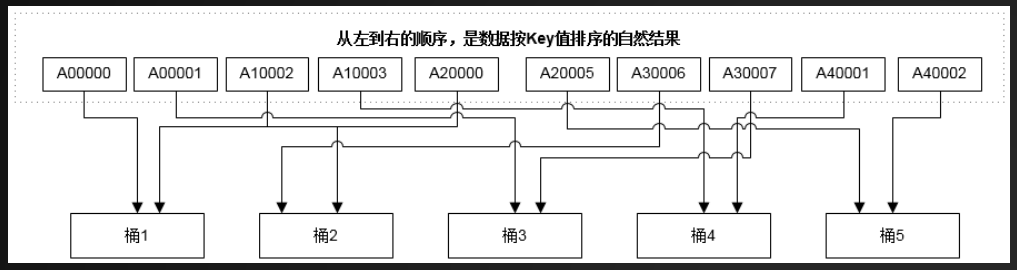
缺点：既然是随机bytes，基于原RowKey查询时无法获知随机bytes信息是什么，也就需要去各个可能的Regions中去查看。可见，Salting对于读取是利空的。

避免数据热点的方法 - Hashing

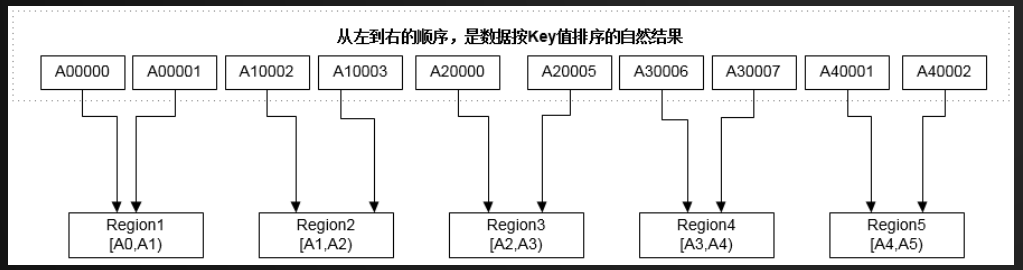
基于RowKey的完整或部分数据进行Hash，而后将Hashing后的值完整替换原RowKey或部分替换RowKey的前缀部分。

缺点：与Reversing类似，Hashing也不利于Scan，因为打乱了原RowKey的自然顺序。

知识点：分布式数据库的常见数据分片方式



“Hash分片”方式：尽管可以保证所有数据到各个“桶”的分布均匀性。但打乱了数据原有的顺序，从而使得按顺序读取方式的性能偏低



“Range分片”方式： 每一个Region负责管辖一个Key值范围。这样的存储，保留了数据原有的顺序，从而按顺序读取方式的性能很高

有两种常见的基础数据分片方式：

- Hash分片
- Range分片

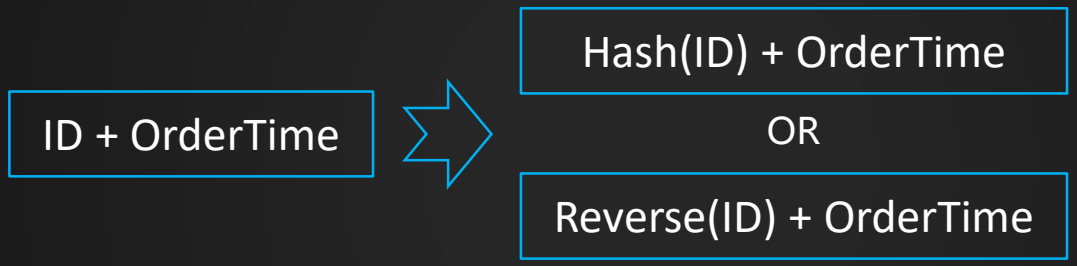
原理与区别可见左图。在实际应用中，两者还可以结合应用。

HBase采用了基于RowKey分区的方式。

实现Hash+Range分片的方法

RowKey构成：　ID + OrderTime

查询场景：　给定ID，查找该ID关联的所有订单



达成的效果：不同ID的数据可均匀打散到各个Region中，而同一个ID的数据则是相邻存放的。

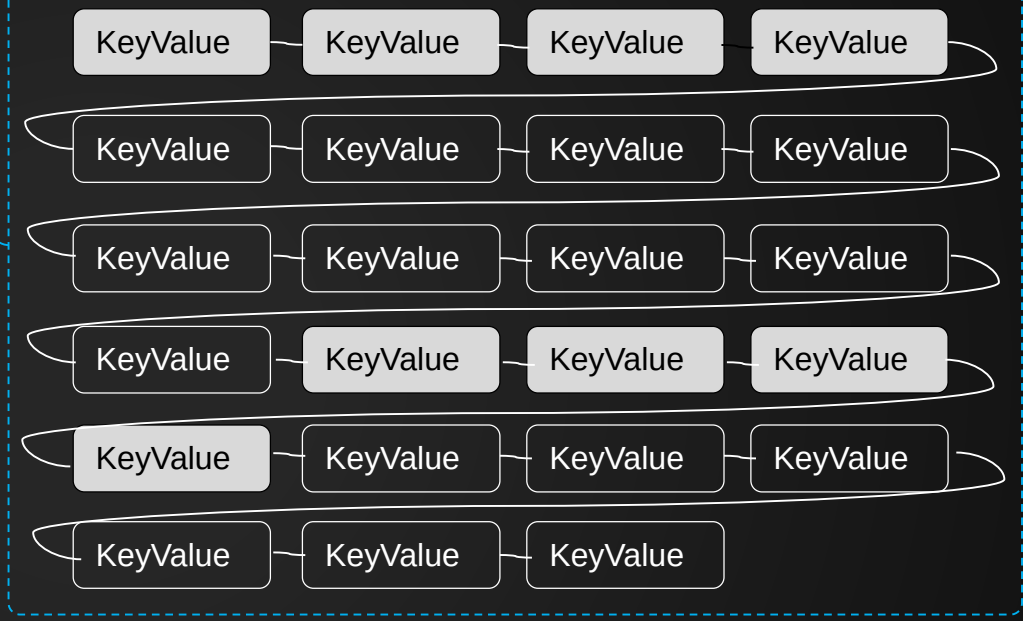
HBase基于Range的方式可以为使用者带来很大的灵活度，但却提升了使用门槛！

多Schema类型数据聚簇

举例：select a.account_id, a.amount, b.account_name, b.balance **from Transactions** a **left join AccountInfo** b **on** a.account_id = b.account_id **where** a.account_id = "xxxxxxx"

account_id	account_name	register_date	balance
A0001	Andy	12/12/2012	$90000
A0001	$100	09/12/2014 13:00:07	
A0001	$1020	10/12/2014 15:30:05	
A0001	$100	12/12/2014 18:00:02	
A0002	Lily	10/12/2010	$902323
A0002	$105	11/11/2014 18:15:00	
A0002	$129	11/12/2014 20:15:00	

account_id amount order_time

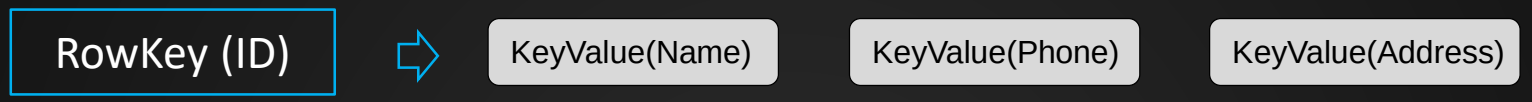


RowKey设计：
AccountInfo: Hash(account_id)
Transactions: Hash(account_id) + order_time

将AcountInfo与Transactions中的记录存放到一个HBase表中，通过RowKey上的一些特殊设计，使得：

- 通过RowKey可识别出一行数据属于哪一个表
- 两个表中的所有记录仍然以独立的行存在，属于同一个account_id的不同的行相邻存放

二级索引RowKey设计常见方法
无Schema模式



Global Index

Reverse(Phone) + **RowKey(ID)**

- 索引列Phone作为先导列，并可进行相关处理来避免热点
- 原数据RowKey放在末端，可用来查找原数据位置

Local Index

ShardKey + Reverse(Phone) + **RowKey(ID)**

- 确保每一个数据表Region所生成的索引数据在同一个索引Region中，通常选择使用数据表Region的StartKey作为这里的ShardKey

这是常见的设计思路，如果原数据RowKey中已经包含了索引列的信息，该设计容易导致数据冗余

二级索引RowKey设计常见方法
有Schema模式

原数据RowKey

假设数据表UserInfo包含下面5个columns：
ID, NAME, ADDRESS,
PHONE,DATE
原数据RowKey包含三个组成部分：
Section 1: ID
Section 2: NAME
Section 3: truncate(DATE, 8)
如下图所示：

二级索引RowKey

基于NAME列构建的索引结构示例如下：

基于PHONE列构建的索引结构示例如下：

注：用 标注的提示信息表明该列也在原数据RowKey中存在

当原数据RowKey中的列与索引列有重叠时，该设计能避免一个列在索引列中被重复存储。但该设计需要事先支持Schema，也就是需要事先定义原数据的RowKey结构以及索引的结构信息。

二级索引字段的选取

对所有的价值查询场景进行详细分析，基于确实能够缩小查询范围的一部分列来构建二级索引。即，我们应该基于离散度较好的一些列来构建索引。

列名称	中文描述	列值信息
ID	证件号码	Primary Key,全局唯一
NAME	姓名	可能会重复
PROVINCE	籍贯省份	共有34种枚举值
GENDER	性别	共有2种枚举值
PHONE	电话号码	每条记录均不重复
AGE	年龄	从1~100之间的值

如上图所示，字段ID, PHONE的离散度为1，基于这些字段构建索引是最佳的。而字段PROVINCE与GENDER,AGE的离散度较差，不适合用来构建二级索引。

组合索引适用场景/构建原则

NAME	ID	...	Phone
Ariya	I0000005	....	1332222
Bai	I0000006	....	1352222
He	I0000004	....	1362222
Lily	I0000001	....	1382222
Lina	I0000003	....	1385555
Lisa	I0000008	....	1385555
Wang	I0000002	....	1333232
Wang	I0000007	....	1332323
.....	.....	....	.....
Wang	I0012322	....	1334988
Wang	I0040007	....	1335465
Wang	I0052322	....	1397323
Wang	I0070007	....	1366134
Wang	I0112322	....	1388866
Wang	I0900007	....	1388888
.....	.....	....	....
Xiao	I0000009	....	1332222

图1 基于NAME + ID构建索引

NAME	Phone	ID	...
Ariya	1332222	I0000005	....
Bai	1352222	I0000006	....
He	1362222	I0000004	....
Lily	1382222	I0000001	....
Lina	1385555	I0000003	....
Lisa	1385555	I0000008	....
Wang	1332323	I0000007	....
Wang	1333232	I0000002	....
.....	.....	.....	....
Wang	1334988	I0012322	....
Wang	1335465	I0040007	....
Wang	1366134	I0070007	....
Wang	1397323	I0052322	....
Wang	1388866	I0112322	....
Wang	1388888	I0900007	....
.....	....	.....	....
Xiao	1332222	I0000009	....

图2 基于NAME+Phone+ID构建索引

举例：

如左图所示，假设查询条件为：

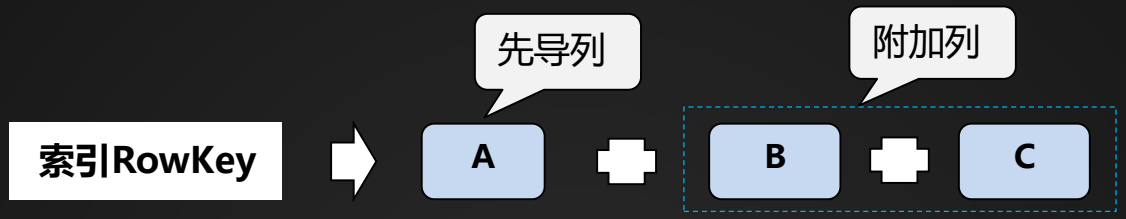
NAME = “Wang” && Phone = “1388888”

如果基于NAME构建索引，则依然可能需要扫描大量的数据，才可以找到一条目标记录。如左图1所示。

如果基于NAME + Phone构建索引，则可以更精确地命中目标记录。如左图2所示。查询性能可大幅度提升。

组合索引的创建，取决于对用户查询场景的详细分析。组合索引的确可极大的优化这些字段组合时的查询场景，但却会带来相对较大的数据膨胀。在不了解用户数据特点以及用户查询场景的情形下，盲目的构建组合索引，是要坚决避免的。

组合索引中字段组合的顺序



先导列的选取：

- 被选作先导列的列，一定是经常被用到的列
- 应选择设置了EQUALS查询条件的列作为先导列
- 先导列应该具备较好的离散度
- 尽量不要重复选择其它索引的先导列作为本索引的先导列

附加列的选取：

- 提供了EQUALS查询条件的列，应该放在前面部分
- 由于列的组合顺序将会影响到数据的排序，我们也应该考虑到业务场景关于排序的诉求
- 结合各个列的离散度进一步分析，将这些列组合之后，能否将数据限定在一个合理的范围之内？如果不能，需要结合查询场景设置更合理的组合列。

关于索引的其它建议

1. 严格控制二级索引的数量

每一个索引所关联的索引数据总条数，与用户数据的总条数是1:1的。因此，在为一个用户表定义二级索引时，应该要考虑多个索引所带来的存储空间膨胀以及性能下降问题。建议在充分分析了各种查询场景的情况下，通过构建尽量少的索引，来满足更多的查询场景

2. Global Index在高并发/小批次查询场景中更有利

查询一个Local Index时，需要去查看每一个Index Region才能获取到符合条件的完整结果集，这对高并发查询并不利。但Local Index对于并发分析场景却是有利的

3. 全文检索需求可考虑与Elasticsearch/Solr对接

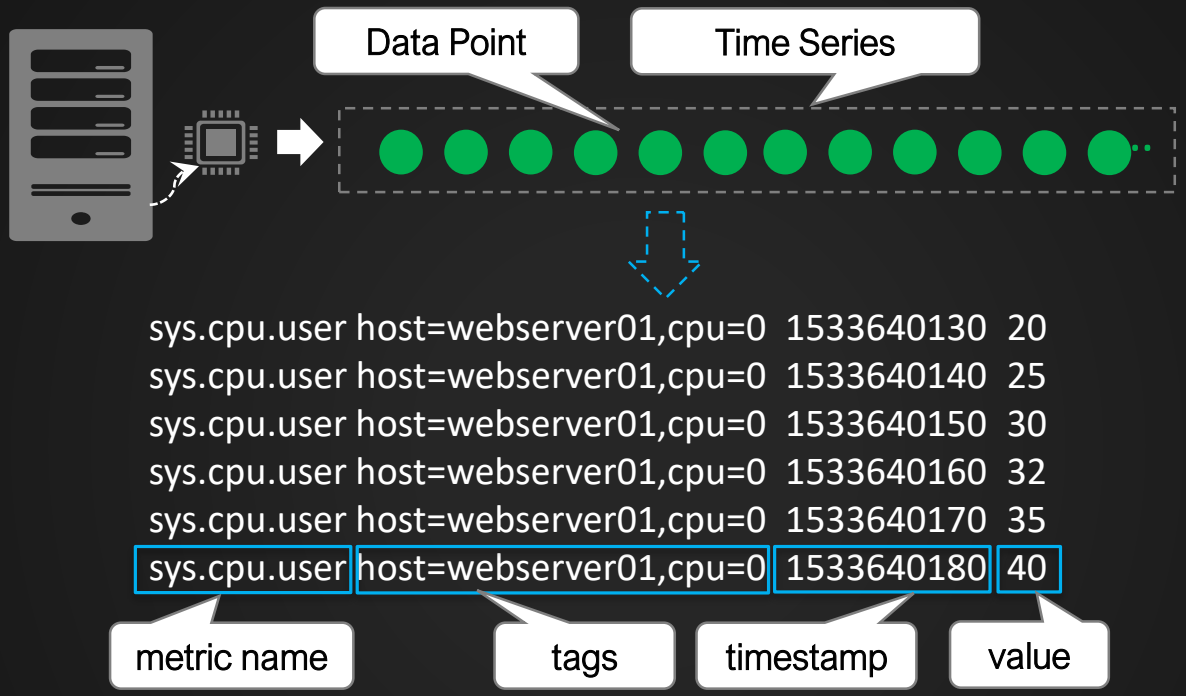
Elasticsearch/Solr是专业的索引引擎，但不适合作为数据主存系统，因此，将核心数据存放在HBase中，而通过ES/Solr构建全文索引能力，是一种常见的组合应用场景

目录

案例剖析

- OpenTSDB设计分析
- JanusGraph设计分析
- GeoMesa设计分析

OpenTSDB – 数据模型



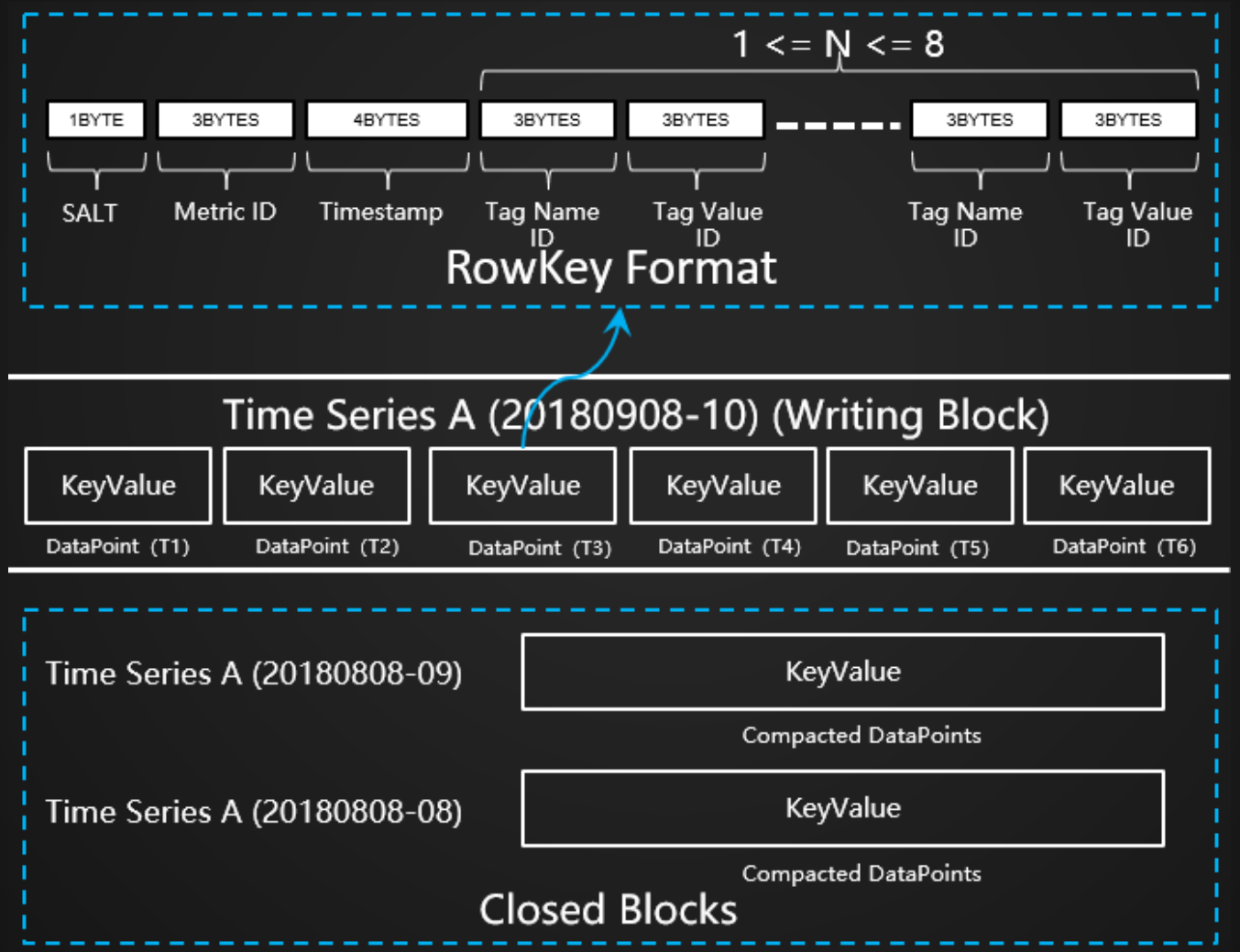
一个Time Series可以理解成是一个数据源的一个指标按时间产生的指标数据序列，每一个指标称之为一个Data Point。OpenTSDB使用一个Metric Name以及一组Tags信息来唯一确定一个Time Series。

OpenTSDB – 典型场景分析

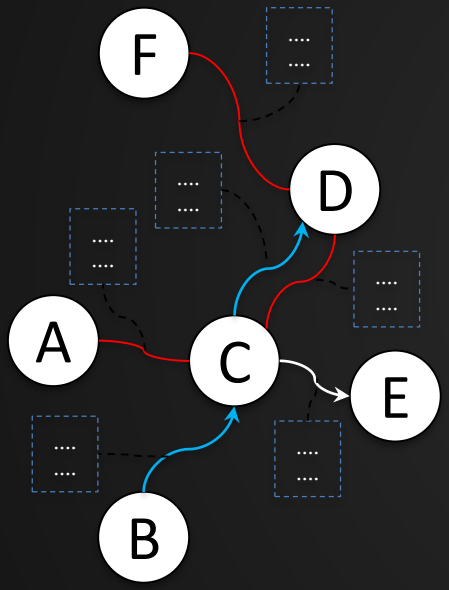
- 给定Metric Name以及一组Tags信息，查询某时间范围的所有的Data Points
- 给定Metric Name以及一组Tags信息，查询某时间范围的聚合结果
- 给定Metric Name，查询所有相关Time Series在某时间范围的统计信息

OpenTSDB – RowKey设计

OpenTSDB Version : 2.3



JanusGraph – 数据模型



主要包含两类数据:

- **顶点(Vertex)**

Vertex包含属性信息。

- **边(Edge)**

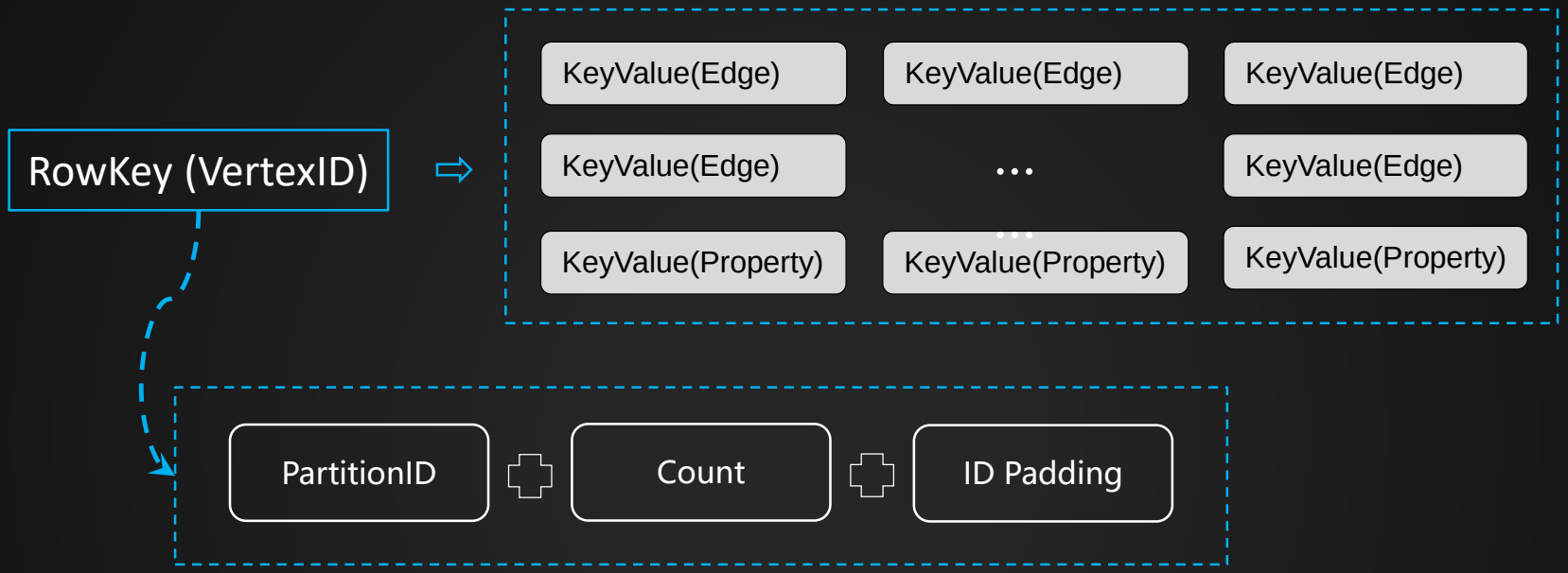
边拥有EdgeLabel信息，EdgeLabel拥有Multiplicity属性，用来定义任意两个顶点之间具有同一EdgeLabel的边的数量信息，以及定义任意一个顶点的出入度信息(入边数量与出边数量)。
边也可以包含属性信息。
边分为单向边与无向边(双向边)。

JanusGraph – 典型查询场景

- 简单关系查询：“A的同事有哪些人？”
- 多层关系查询(扩线查询)：“A的朋友的朋友的朋友？”
- 关系（多层）查询+属性条件过滤：“A的朋友的朋友中，有哪些拥有本科学历并且在深圳工作？”
- 基于属性的查询：“姓名为XXX，学历为本科，居住地在深圳龙岗区，年龄在30~40岁左右的人？”

JanusGraph – RowKey设计

JanusGraph Version : 0.2

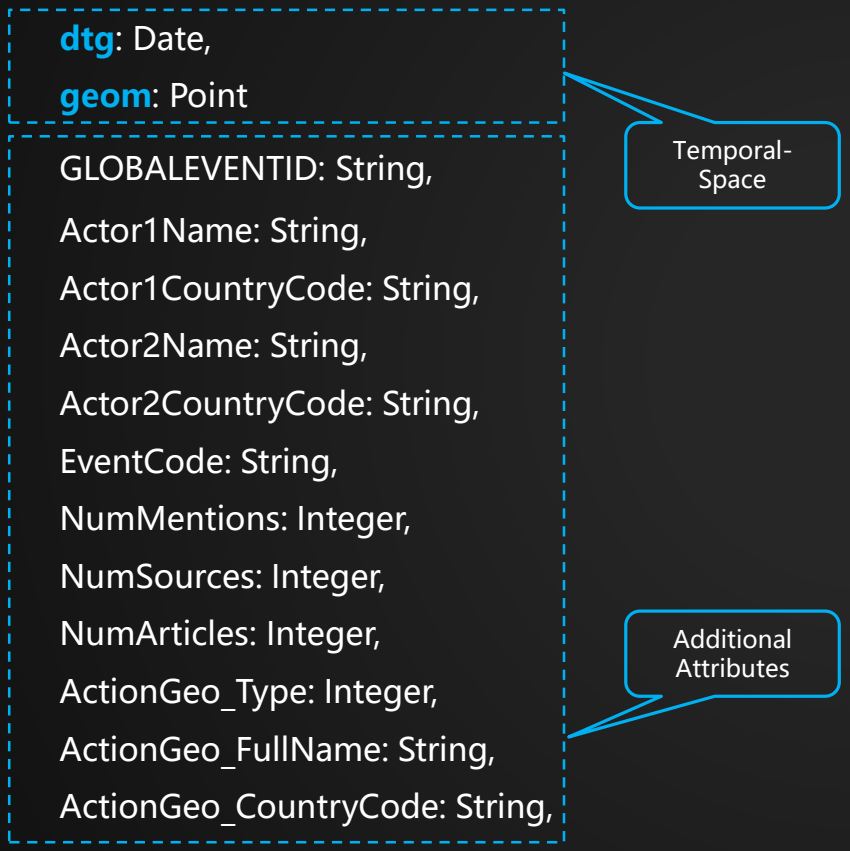


在JanusGraph中，无论是用户数据还是元数据信息，都以Vertex形式存在，因此，JanusGraph包含了各色各样的Vertex类型，这些类型在ID Padding部分体现出来。如：

000: Normal Vertices; *010: Partitioned Vertices;* *100: Unmodifiable vertices*

101: Schema Type Vertices; *000101: User Property Key;* *100101: System Property Key*

GeoMesa – 数据模型



GeoMesa is an Apache licensed open source suite of tools that enables large-scale geospatial analytics on cloud and distributed computing systems, letting you manage and analyze the huge spatio-temporal datasets that IoT, social media, tracking, and mobile phone applications seek to take advantage of today.

GeoMesa的一条数据，称之为一个SimpleFeature， SimpleFeature中主要包含如下数据：

1. 时间信息
2. 空间信息
3. 其它属性

GeoMesa – 典型查询场景

- 查询某一个地理区域在某个时间范围发生的关键事件
- 一个任意区域的流量信息
- 给出2015年受血吸虫病影响的区域
- 查找最近10分钟进入飓风区域的汽车?
- 查找某一个点附近所有的酒店
- 查找某嫌疑人在2018年9月的移动轨迹
- ……..

时空查询：区域 + 时间区间

空间查询：区域信息

时序查询：轨迹查看

属性查询：主题属性信息

GeoMesa – RowKey设计

GeoMesa Version : 2.11

- 针对Point的时间+空间三维索引(Z3)

  ShardKey(1byte) + Epoch Week(2bytes) + Z3(x, y, t) (8bytes)+ FeatureID

- 针对Point的空间索引(Z2)

  ShardKey(1byte) + Z2(x, y) (2bytes) + FeatureID

- 针对复杂空间对象(如Polygon)的时间+空间三维索引(XZ3)

  ShardKey(1byte) + Epoch Week(2bytes) + XZ2(minX, minY, maxX, maxY)(8bytes)
  + FeatureID

- 针对复杂空间对象(如Polygon)的空间二维索引 (XZ2)

  ShardKey(1byte) + XZ2(minX, minY, maxX, maxY)(8bytes) + FeatureID

- Attribute索引

  IdxBytes(2bytes) + ShardKey(1byte) + AttrValue + SplitByte(1byte)
  + SecondaryIndex(Z3/XZ3/Z2/XZ2) + FeatureID

- ID索引

  FeatureID

“知识点备注：Z2/Z3的核心思想是基于Z-Order空间填充曲线将二维/三维信息映射成一维信息，在这个一维信息中能够很好的保持时空距离信息。而XZ2/XZ3的核心思想是XZ-Order空间填充曲线，XZ-Order是基于Z-Order做的扩展，从而能够支持将Polygon/Rectangle等复杂空间对象映射成一维信息。

内容总结

- HBase基础概念/读写流程回顾，探讨RowKey的关键作用
- RowKey与索引设计前需求调研的几个关键维度
- RowKey与索引设计的技巧与原则
- 围绕OpenTSDB/JanusGraph/GeoMesa的数据模型与查询场景，简单探讨了它们的RowKey结构设计

加入我们

我们正在招聘大数据/AI相关开发工程师，欢迎感兴趣的同学加入我们。简历投递地址：yuanchunfeng@huawei.com

华为云表格存储服务

公有云HBase服务

微信公众号

NoSQL漫谈

参考信息

1. http://hbase.apache.org/book.html
2. 李华植[韩] 海量数据库解决方案
3. https://docs.janusgraph.org/latest/
4. https://www.geomesa.org/documentation/user/index.html
5. http://docs.geotools.org/latest/userguide/library/opengis/
6. XZ-Ordering: A Space-Filling Curve for Objects with Spatial Extension
7. Megastore: Providing Scalable, Highly Available Storage for Interactive Services
8. http://opentsdb.net/docs/build/html/index.html
9. http://www.nosqlnotes.com
10. https://github.com/locationtech/sfcurve

THANK YOU

LEADING NEW ICT
THE ROAD TO
DIGITAL TRANSFORMATION