**Cassandra 框架**

Apache Cassandra是一套开源分布式NoSQL数据库系统，采用java编写。它最初由Facebook开发，用于储存收件箱等简单格式数据，集Google BigTable的数据模型与Amazon Dynamo的完全分布式的架构于一身。



Cassandra的类Dynamo特性有以下几点：

对称的，P2P架构

无特殊节点，无单点故障

基于Gossip的分布式管理

通过分布式hash表放置数据

类BigTable特性：

列族数据模型

可配置，2级maps，Super Colum Family

SSTable磁盘存储

Append-only commit log

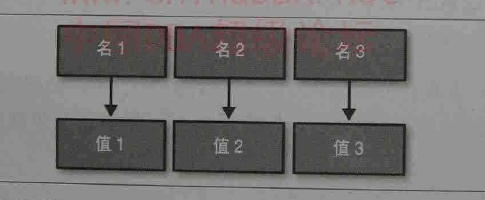
Memtable (buffer and sort)

不可修改的SSTable文件

**二、 Cassandra数据模型**

Keyspace：Cassandra中最顶层的容器，可以看作关系数据库中的数据库。

**Colum / Colum Family, SuperColum / SuperColum Family**



**Colum**

下面是一个用JSON格式表示的column：

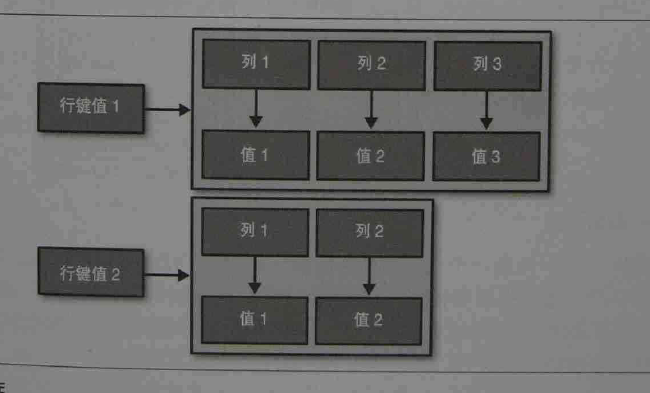
**{  // 这是一个Column**

**name: "emailAddress",**

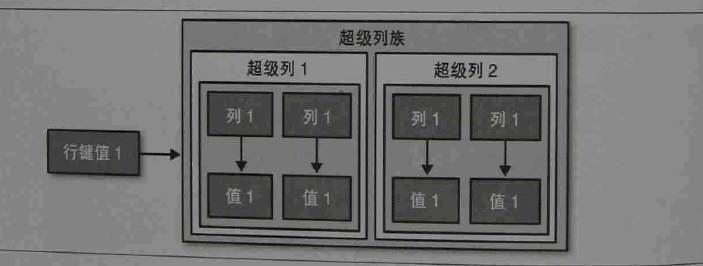
**value: "arin@example.com",**

**timestamp: 123456789**

**}**



**Colum Family**



**SuperColum / SuperColum Family**

**{ // 这是一个SuperColumn**

**name: "homeAddress",**

**// 无限数量的Column**

**value: {**

**street: {name: "street", value: "1234 x street", timestamp: 123456789},**

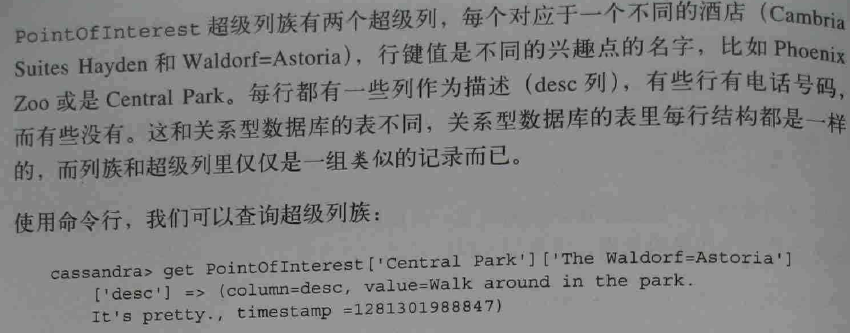
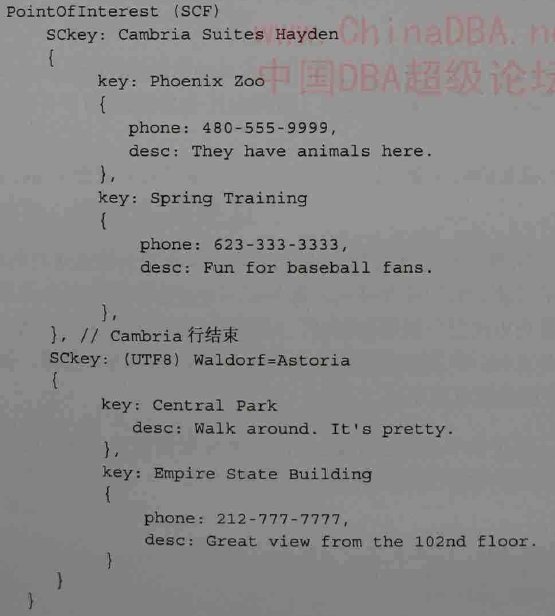
**city: {name: "city", value: "san francisco", timestamp: 123456789},**

**zip: {name: "zip", value: "94107", timestamp: 123456789},**

**}**

**}**

**可以把cassandra看作是一个5维的hash表。**



**Colum排序**

不同于数据库可以通过Order by定义排序规则，Cassandra取出的数据顺序是总是一定的，数据保存时已经按照定义的规则存放，所以取出来的顺序已经确定了。另外，Cassandra按照column name而不是column value来进行排序。

Cassandra可以通过Colum Family的CompareWith属性配置Colume值的排序，在SuperColum中，则是通过SuperColum Family的CompareSubcolumnsWith属性配置Colum的排序。

Cassandra提供了以下一些选：BytesType，UTF8Type，LexicalUUIDType，TimeUUIDType，AsciiType， Column name

**三、分区策略**



**1. Murmur3Partitioner（默认）：基于MurmurHash哈希算法**

**2 .RandomPartitioner：基于MD5哈希算法**

**3.** **ByteOrderedPartitioner：根据partition key的bytes进行有序分区**

1. 负载均衡更困

2. 顺序的分区容易造成热点

## 四、副本存储

Cassandra不像HBase是基于HDFS的分布式存储，它的数据是存在每个节点的本地文件系统中。

Cassandra有三种副本配置策略：

**1) SimpleStrategy （RackUnawareStrategy）：**

副本不考虑机架的因素，按照Token放置在连续下几个节点。如图3所示，假如副本数为3，属于A节点的数据在B.C两个节点中也放置副本。

**2) OldNetworkTopologyStrategy （RackAwareStrategy）:**

考虑机架的因素，除了基本的数据外，先找一个处于不同数据中心的点放置一个副本，其余N-2个副本放置在同一数据中心的不同机架中。

**3)  NetworkTopologyStrategy （DatacenterShardStrategy）：**

将M个副本放置到其他的数据中心，将N-M-1的副本放置在同一数据中心的不同机架中

## 六、故障检测

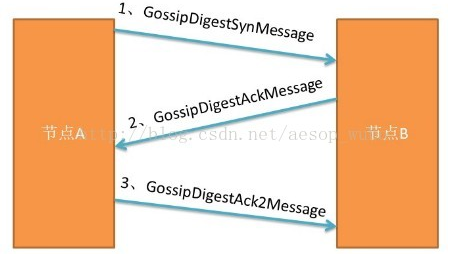
## Gossip。

**种子节点（节点的代表）：**

当节点第一次启动时，它会去查配置文件**cassandra.yaml**从而得到它属于的集群名称，但是它如何获得集群中其他节点的信息呢？就是通过种子节点（seed node).记住，**同一集群中所有的节点的cassandra.yaml中必须有相同的种子节点列表。**

**assandra故障探测：**

Gossip 进程（Gossiper)通过每个节点的心跳（heartbeat)来感知每个节点是否还存活。它会考虑到网络状况，负载等因素综合考虑来计算节点心跳时间的临界值。gossip过程中，每个节点都维护着其他节点gossip消息的内部到达次数。在Cassandra中，如果配置phi\_convict\_threshold，可以调节失败探测的敏感度，从而适应相对不可靠的网络环境,超过这个时间（秒）则被cassandra认为这个节点down了。



#### Cassandra的实现(交换包含token，路由用途。)

当一个节点启动时，获取配置文件（storage-conf.xml）中的seeds配置，从而知道集群中所有的seed节点。

Cassandra内部有一个Gossiper，每隔一秒运行一次（在Gossiper.java的start方法中），按照以下规则向其他节点发 送同步消息：

1. 随机取一个当前活着的节点，并向它发送同步请求
2. 向随机一台不可达的机器发送同步请求
3. 如果第一步中所选择的节点不是seed，或者当前活着的节点数少于seed数，则向随意一台seed发送同步请求

这里假设192.168.1.1（源节点）决定和192.168.1.2（目标节点）同步，首先源节点向目标节点发送 GossipDigestSynMessage包，这个包包括本机维护的所有节点的状态信息的最新版本摘要，摘要只包含key和version，不包含具 体的value，这样可以减小同步的带宽消耗。

当目标节点收到GossipDigestSynMessage包时，它需要做两件事：

1. 找出收到的消息中比本地版本新的状态，按照版本号差异大小排序，将这些**状态的摘要** 放入 GossipDigestAckMessage中
2. 找出本地比源节点版本更新的状态，将这些**状态** 放入GossipDigestAckMessage中

当GossipDigestAckMessage构建完成后，会被发送给源节点

这里按照版本号差异大小排序的原因是每个Message允许发送的状态数量是有限的（参见Gossip.java中的 MAX\_GOSSIP\_PACKET\_SIZE定义），这样可以保证比较老的状态（版本号差异大的）可以优先得到更新。

源机器接收到GossipDigestAckMessage后，使用发送过来的目标节点更新的状态更新本地的状态，这样源节点就获取到了目标节点上 比自己更新的状态。

同时源节点把包含在GossipDigestAckMessage中摘要对应的状态通过GossipDigestAck2Message发送到目标 服务器，目标服务器更新本地的状态，这样目标服务器也获取到了源节点上比自己更新的状态。

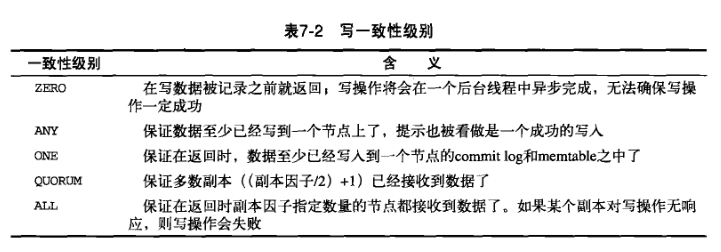
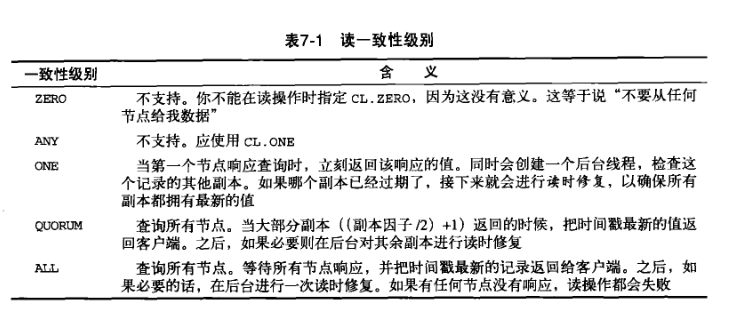
完成这样一次同步后，源节点和目标节点上的状态都得到了同步。这中工作方式还是比较优的。

## 证明是logN

## 六、一致性

在一致性上，Cassandra采用了最终一致性，可以根据具体情况来选择一个最佳的折衷，来满足特定操作的需求。Cassandra可以让用户指定读/插入/删除操作的一致性级别，一致性级别有多种，如图5所示。

注：一致性级别是由副本数决定，而不是集群的节点数目决定。



**Quorum NRW**

## N: 复制的节点数量，即副本数

## R: 成功读操作的最小节点数

## W: 成功写操作的最小节点数

Quorum协议中，R 代表一次成功的读取操作中最小参与节点数量，W 代表一次成功的写操作中最小参与节点数量。R + W>N ，则会产生类似quorum 的效果。该模型中的读(写)延迟由最慢的 R(W)复制决定，为得到比较小的延迟，R和W有的时候的和比N小。

Quorum协议中，只需W + R > N，就可以保证强一致性。因为读取数据的节点和被同步写入的节点是有重叠的。在一个RDBMS的复制模型中（Master/salve)，假如N=2,那么W=2,R=1此时是一种强一致性,但是这样造成的问题就是可用性的减低，因为要想写操作成功，必须要等 2个节点的写操作都完成以后才可以。

在分布式系统中，一般都要有容错性，因此N一般大于3的，此时根据CAP理论，我们就需要在一致性和分区容错性之间做一平衡，如果要高的一致性，那么就配置N=W，R=1,这个时候可用性就会大大降低。如果想要高的可用性，那么此时就需要放松一致性的要求，此时可以配置W=1，这样使得写操作延迟最低，同时通过异步的机制更新剩余的N-W个节点。

当存储系统保证最终一致性时，存储系统的配置一般是W+R<=N，此时读取和写入操作是不重叠的，不一致性的窗口就依赖于存储系统的异步实现方式，不一致性的窗口大小也就等于从更新开始到所有的节点都异步更新完成之间的时间。

一般来说，Quorum中比较典型的NRW为（3,2,2）。

**维护最终一致性**

Cassandra 通过4个技术来维护数据的最终一致性，分别为逆熵（Anti-Entropy），读修复（Read Repair），提示移交（Hinted Handoff）和分布式删除。

**1)       逆熵**

这是一种备份之间的同步机制。节点之间定期互相检查数据对象的一致性，这里采用的检查不一致的方法是 Merkle Tree；（讲一下实现原理。）

**2)       读修复**

客户端读取某个对象的时候，触发对该对象的一致性检查：

读取Key A的数据时，系统会读取Key A的所有数据副本，如果发现有不一致，则进行一致性修复。

如果读一致性要求为ONE，会立即返回离客户端最近的一份数据副本。然后会在后台执行Read Repair。这意味着第一次读取到的数据可能不是最新的数据；如果读一致性要求为QUORUM，则会在读取超过半数的一致性的副本后返回一份副本给客户端，剩余节点的一致性检查和修复则在后台执行；如果读一致性要求高(ALL)，则只有Read Repair完成后才能返回一致性的一份数据副本给客户端。可见，该机制有利于减少最终一致的时间窗口。

**3)       提示移交**

对写操作，如果其中一个目标节点不在线，先将该对象中继到另一个节点上，中继节点等目标节点上线再把对象给它：

Key A按照规则首要写入节点为N1，然后复制到N2。假如N1宕机，如果写入N2能满足ConsistencyLevel要求，则Key A对应的RowMutation将封装一个带hint信息的头部（包含了目标为N1的信息），然后随机写入一个节点N3，此副本不可读。同时正常复制一份数据到N2，此副本可以提供读。如果写N2不满足写一致性要求，则写会失败。 等到N1恢复后，原本应该写入N1的带hint头的信息将重新写回N1。

**4)       分布式删除**

单机删除非常简单，只需要把数据直接从磁盘上去掉即可，而对于分布式，则不同，分布式删除的难点在于：如果某对象的一个备份节点 A 当前不在线，而其他备份节点删除了该对象，那么等 A 再次上线时，它并不知道该数据已被删除，所以会尝试恢复其他备份节点上的这个对象，这使得删除操作无效。Cassandra 的解决方案是：本地并不立即删除一个数据对象，而是给该对象标记一个hint，定期对标记了hint的对象进行垃圾回收。在垃圾回收之前，hint一直存在，这使得其他节点可以有机会由其他几个一致性保证机制得到这个hint。Cassandra 通过将删除操作转化为一个插入操作，巧妙地解决了这个问题。

## 七、存储机制

Cassandra的存储机制借鉴了Bigtable的设计，采用Memtable和SSTable的方式。

**CommitLog**

和HBase一样，Cassandra在写数据之前，也需要先记录日志，称之为Commit Log，然后数据才会写入到Column Family对应的MemTable中，且MemTable中的数据是按照key排序好的。SSTable一旦完成写入，就不可变更，只能读取。下一次Memtable需要刷新到一个新的SSTable文件中。所以对于Cassandra来说，可以认为只有顺序写，没有随机写操作。

**MenTable**

MemTable是一种内存结构，当数据量达到块大小时，将批量flush到磁盘上，存储为SSTable。这种机制，相当于缓存写回机制(Write-back Cache)，优势在于将随机IO写变成顺序IO写，降低大量的写操作对于存储系统的压力。所以我们可以认为Cassandra中只有顺序写操作，没有随机写操作。

**SSTable**

SSTable是Read Only的，且一般情况下，一个CF会对应多个SSTable，当用户检索数据时，Cassandra使用了Bloom Filter，即通过多个hash函数将key映射到一个位图中，来快速判断这个key属于哪个SSTable。

为了减少大量SSTable带来的开销，Cassandra会定期进行compaction，简单的说，compaction就是将同一个CF的多个SSTable合并成一个SSTable。在Cassandra中，compaction主要完成的任务是：

1）垃圾回收： cassandra并不直接删除数据，因此磁盘空间会消耗得越来越多，compaction 会把标记为删除的数据真正删除；

2）合并SSTable：compaction 将多个 SSTable 合并为一个（合并的文件包括索引文件，数据文件，bloom filter文件），以提高读操作的效率；

3）生成 MerkleTree：在合并的过程中会生成关于这个 CF 中数据的 MerkleTree，用于与其他存储节点对比以及修复数据。

**http://wenku.baidu.com/link?url=tgqXyMQv9wO9sAytcF8Y1w1LV4pbUwZj9ATUzIal6blmr8gLWZFIGbAdFNrlioovW4ax54Sbw3jgyTvC5ydsuyYAChvnkUSsUltWsHLeB6e**

[**https://www.ibm.com/developerworks/cn/opensource/os-cn-cassandraxu2/**](https://www.ibm.com/developerworks/cn/opensource/os-cn-cassandraxu2/)

**看一下最新的论文。**

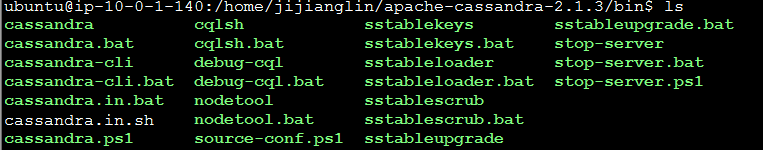
**（CAP结合起来）**

### 安装和应用

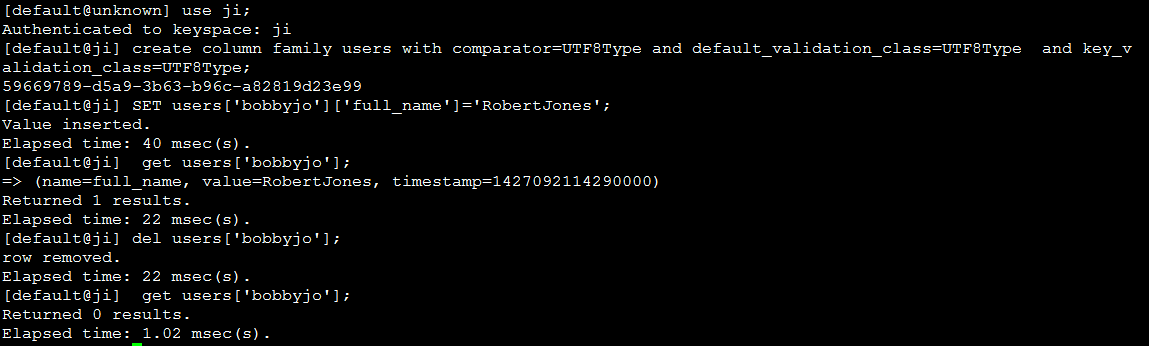
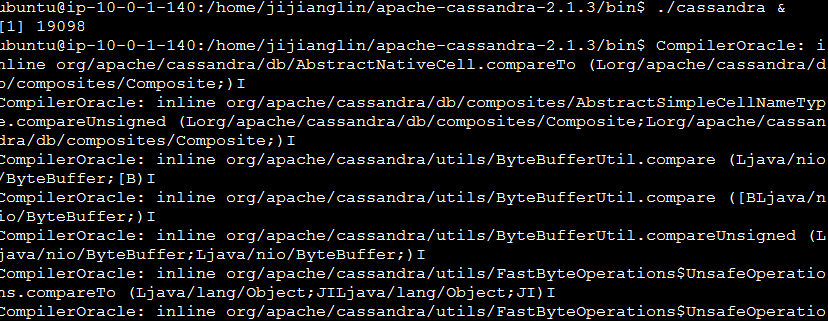
**Start Cassandra**

1. **安装java jdk1.7**

**2.**



**3. ./Cassandra &（后台执行，或者使用-f 在前台执行）**



## 3. Using cqlsh

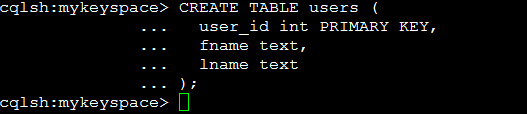
First, create a keyspace -- a namespace of tables.



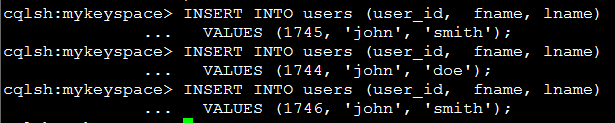
Second, authenticate to the new keyspace:



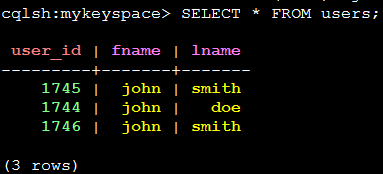
Third, create a users table:



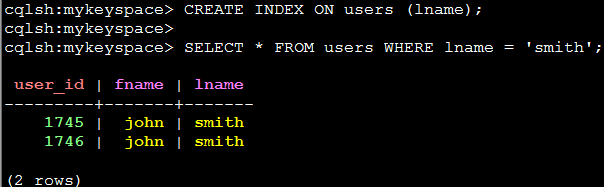
Now you can store data into users:



Now let's fetch the data you inserted:



You can retrieve data about users whose last name is smith by creating an index, then querying the table as follows:



4.5、通过JAVA操作Cassandra

Hector是一个比较好的选择，完全开源，这个是GitHub的源码地址：<https://github.com/rantav/hector>，以下是一个基于Hector的CRUB的示例，依赖的包在Cassandra的lib目录下面就可以找到。

<http://teddymaef.github.io/learncassandra/cn/>

<http://www.cnblogs.com/dyf6372/p/3537250.html>

<http://www.cnblogs.com/dyf6372/p/3537250.html>

cql