**[几款主流 NoSql 数据库的对比](http://www.cnblogs.com/vajoy/p/5471308.html)**

http://www.cnblogs.com/vajoy/p/5471308.html

**HBase**

HBase 是 Apache Hadoop 中的一个子项目，属于 bigtable 的开源版本，所实现的语言为Java*（故依赖 Java SDK）*。HBase 依托于 Hadoop 的 HDFS*（分布式文件系统）*作为最基本存储基础单元。

HBase在列上实现了 BigTable 论文提到的压缩算法、内存操作和布隆过滤器。HBase的表能够作为 [MapReduce](https://zh.wikipedia.org/wiki/MapReduce" \t "_blank)任务的输入和输出，可以通过Java API来访问数据，也可以通过REST、Avro或者Thrift的API来访问。

**1. 特点**

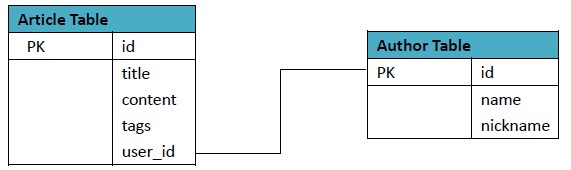
**1.1 数据格式**

HBash 的数据存储是基于列*（ColumnFamily）*的，且非常松散—— 不同于传统的关系型数据库*（RDBMS）*，HBase 允许表下某行某列值为空时不做任何存储*（也不占位）*，减少了空间占用也提高了读性能。

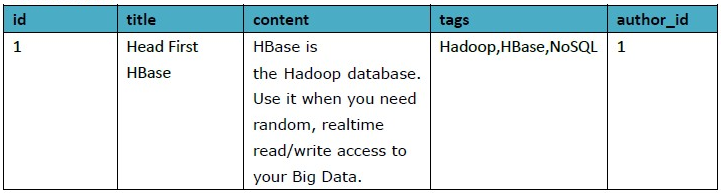
不过鉴于其它NoSql数据库也具有同样灵活的数据存储结构，该优势在本次选型中并不出彩。

我们以一个简单的例子来了解使用 RDBMS 和 HBase 各自的解决方式：

1. RDBMS方案：



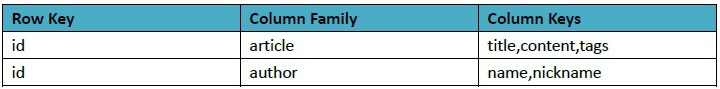
其中Article表格式



Author表格式：

http://images2015.cnblogs.com/blog/561179/201605/561179-20160508182753937-622987437.png

1. 等价的HBase方案



对于前端而言，这里的 Column Keys 和 Column Family 可以看为这样的关系

columId1 = { //id=1的行

article: { //ColumnFamily-article

title: XXX, //ColumnFamily-article下的key之一

content: XXX,

tags: XXX

},

author: { //ColumnFamily-author

name: XXX

nickname: XXX

}

}

**1.2 性能**

HStore存储是HBase存储的核心，它由两部分组成，一部分是MemStore，一部分是StoreFiles。

MemStore 是 Sorted Memory Buffer，用户写入的数据首先会放入MemStore，当MemStore满了以后会Flush成一个StoreFile（底层实现是HFile），当StoreFile文件数量增长到一定阈值，会触发Compact合并操作，将多个StoreFiles合并成一个StoreFile，合并过程中会进行版本合并和数据删除，因此可以看出HBase其实只有增加数据，所有的更新和删除操作都是在后续的compact过程中进行的，这使得用户的写操作只要进入内存中就可以立即返回，保证了HBase I/O的高性能。

**1.3 数据版本**

Hbase 还能直接检索到往昔版本的数据，这意味着我们更新数据时，旧数据并没有即时被清除，而是保留着：

Hbase 中通过 row+columns 所指定的一个存贮单元称为cell。每个 cell都保存着同一份数据的多个版本——版本通过时间戳来索引。

时间戳的类型是 64位整型。时间戳可以由Hbase*(在数据写入时自动 )*赋值，此时时间戳是精确到毫秒的当前系统时间。时间戳也可以由客户显式赋值。如果应用程序要避免数据版本冲突，就必须自己生成具有唯一性的时间戳。每个 cell中，不同版本的数据按照时间倒序排序，即最新的数据排在最前面。

为了避免数据存在过多版本造成的的管理*(包括存贮和索引)*负担，Hbase提供了两种数据版本回收方式。一是保存数据的最后n个版本，二是保存最近一段时间内的版本*(比如最近七天)*。用户可以针对每个列族进行设置。

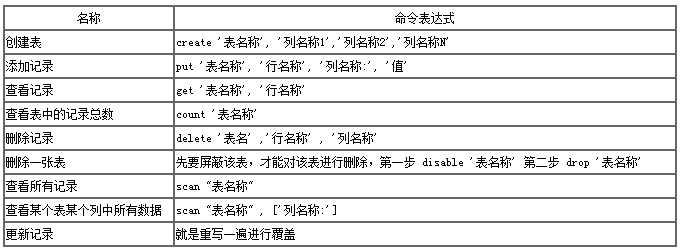
**1.4**[**CAP**](https://zh.wikipedia.org/wiki/CAP%E5%AE%9A%E7%90%86)**类别**

属于CP类型*（*[*了解更多*](http://yangshangchuan.iteye.com/blog/2002544)*）*。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

**2. Node下的使用**

HBase的相关操作可参考下表：



在node环境下，可通过 [node-hbase](https://github.com/wdavidw/node-hbase) 来实现相关访问和操作，注意该工具包依赖于 PHYTHON2.X（3.X不支持）和Coffee。

如果是在 window 系统下还需依赖 .NET framwork2.0，64位系统可能无法直接通过安装包安装。

官方示例：

var assert = require('assert');

var hbase = require('hbase');

hbase({ host: '127.0.0.1', port: 8080 })

.table('my\_table' )

//创建一个Column Family

.create('my\_column\_family', function(err, success){

this.row('my\_row') //定位到指定行

.put('my\_column\_family:my\_column', 'my value', function(err, success){

this.get('my\_column\_family', function(err, cells){

this.exists(function(err, exists){

assert.ok(exists);

});

});

});

});

数据检索：

client

.table('node\_table')

.scan({

startRow: 'my\_row', //起始行

maxVersions: 1 //版本

}, function(err, rows){

console.log(err, rows);

});

另有 [hbase-client](https://github.com/alibaba/node-hbase-client" \t "_blank) 也是一个不错的选择，具体API参照其文档。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

**3. 优缺点**

**优势**

1. 存储容量大，一个表可以容纳上亿行，上百万列；

2. 可通过版本进行检索，能搜到所需的历史版本数据；

3. 负载高时，可通过简单的添加机器来实现水平切分扩展，跟Hadoop的无缝集成保障了其数据可靠性（HDFS）和海量数据分析的高性能（MapReduce）；

4. 在第3点的基础上可有效避免单点故障的发生。

**缺点**

1. 基于Java语言实现及Hadoop架构意味着其API更适用于Java项目；

2. node开发环境下所需依赖项较多、配置麻烦（或不知如何配置，如持久化配置），缺乏文档；

3. 占用内存很大，且鉴于建立在为批量分析而优化的HDFS上，导致读取性能不高；

4. API相比其它 NoSql 的相对笨拙。

**适用场景**

1. bigtable类型的数据存储；

2. 对数据有版本查询需求；

3. 应对超大数据量要求扩展简单的需求。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

**Redis**

Redis 是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。目前由VMware主持开发工作。

**1. 特点**

**1.1 数据格式**

Redis 通常被称为数据结构服务器，因为值（value）可以是 字符串(String), 哈希(Hash/Map), 列表(list), 集合(sets) 和 有序集合(sorted sets)五种类型，操作非常方便。比如，如果你在做好友系统，查看自己的好友关系，如果采用其他的key-value系统，则必须把对应的好友拼接成字符串，然后在提取好友时，再把value进行解析，而redis则相对简单，直接支持list的存储(采用双向链表或者压缩链表的存储方式)。

我们来看下这五种数据类型。

**⑴ String**

* string 是 Redis 最基本的类型，你可以理解成与 Memcached 一模一样的类型，一个key对应一个value。
* string 类型是二进制安全的。意思是 Redis 的 string 可以包含任何数据。比如 jpg 图片或者序列化的对象 。
* string 类型是 Redis 最基本的数据类型，一个键最大能存储512MB。

实例：

redis 127.0.0.1:6379> SET name zfpx

OK

redis 127.0.0.1:6379> GET name

"zfpx"

在以上实例中我们使用了 Redis 的 SET 和 GET 命令。键为 name，对应的值为"zfpx"。 注意：一个键最大能存储512MB。

**⑵ Hash**

* Redis hash 是一个键值对集合。
* Redis hash 是一个 string 类型的 field 和 value 的映射表，hash 特别适合用于存储对象。

实例：

redis 127.0.0.1:6379> HMSET user:1 username zfpx password 123

OK

redis 127.0.0.1:6379> HGETALL user:1

1) "username"

2) "zfpx"

3) "password"

4) "123"

以上实例中 hash 数据类型存储了包含用户脚本信息的用户对象。 实例中我们使用了 Redis HMSET, HGETALL 命令，user:1 为键值。 每个 hash 可以存储 232 - 1 键值对（40多亿）。

**⑶ List**

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素导列表的头部（左边）或者尾部（右边）。

实例：

redis 127.0.0.1:6379> lpush name zfpx1

(integer) 1

redis 127.0.0.1:6379> lpush name zfpx2

(integer) 2

redis 127.0.0.1:6379> lpush name zfpx3

(integer) 3

redis 127.0.0.1:6379> lrange name 0 -1

1) "zfpx3"

2) "zfpx2"

3) "zfpx1"

列表最多可存储 232 - 1 元素 (4294967295, 每个列表可存储40多亿)。

**⑷ Sets**

Redis的Set是string类型的无序集合。 集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。

添加一个string元素到 key 对应的 set 集合中，成功返回1，如果元素已经在集合中返回0，key对应的set不存在返回错误，指令格式为

sadd key member

实例：

redis 127.0.0.1:6379> sadd school zfpx1

(integer) 1

redis 127.0.0.1:6379> sadd school zfpx1

(integer) 0

redis 127.0.0.1:6379> sadd school zfpx2

(integer) 1

redis 127.0.0.1:6379> sadd school zfpx2

(integer) 0

redis 127.0.0.1:6379> smembers school

1) "zfpx1"

2) "zfpx2"

注意：以上实例中 zfpx1 添加了两次，但根据集合内元素的唯一性，第二次插入的元素将被忽略。 集合中最大的成员数为 232 - 1 (4294967295, 每个集合可存储40多亿个成员)。

**⑸ sorted sets/zset**

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。 不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

zset的成员是唯一的,但分数(score)却可以重复。可以通过 zadd 命令（格式如下） 添加元素到集合，若元素在集合中存在则更新对应score

zadd key score member

实例：

redis 127.0.0.1:6379> zadd school 0 zfpx1

(integer) 1

redis 127.0.0.1:6379> zadd school 2 zfpx2

(integer) 1

redis 127.0.0.1:6379> zadd school 0 zfpx3

(integer) 1

redis 127.0.0.1:6379> zadd school 1 zfpx4

(integer) 0

redis 127.0.0.1:6379> ZRANGEBYSCORE school 0 100

1) "zfpx1"

2) "zfpx3"

3) "zfpx4"

4) "zfpx2"

**1.2 性能**

Redis数据库完全在内存中，因此处理速度非常快，每秒能执行约11万集合，每秒约81000+条记录（测试数据的可参考这篇《[*Redis千万级的数据量的性能测试*](http://www.cnblogs.com/lovecindywang/archive/2011/03/03/1969633.html)》）。

Redis的数据能确保一致性——所有Redis操作是原子性（Atomicity，意味着操作的不可再分，要么执行要么不执行）的，这保证了如果两个客户端同时访问的Redis服务器将获得更新后的值。

**1.3 持久化**

通过定时快照（snapshot）和基于语句的追加（AppendOnlyFile，aof）两种方式，redis可以支持数据持久化——将内存中的数据存储到磁盘上，方便在宕机等突发情况下快速恢复。

**1.4 CAP类别**

属于CP类型（[*了解更多*](https://www.quora.com/What-is-Redis-in-the-context-of-the-CAP-Theorem)）。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

**2. Node下的使用**

node 下可使用 [node\_redis](https://github.com/NodeRedis/node_redis" \t "_blank) 来实现 redis 客户端操作：

var redis = require("redis"),

client = redis.createClient();

// if you'd like to select database 3, instead of 0 (default), call

// client.select(3, function() { /\* ... \*/ });

client.on("error", function (err) {

console.log("Error " + err);

});

client.set("string key", "string val", redis.print);

client.hset("hash key", "hashtest 1", "some value", redis.print);

client.hset(["hash key", "hashtest 2", "some other value"], redis.print);

client.hkeys("hash key", function (err, replies) {

console.log(replies.length + " replies:");

replies.forEach(function (reply, i) {

console.log(" " + i + ": " + reply);

});

client.quit();

});

**3. 优缺点**

**优势**

1. 非常丰富的数据结构；

2. Redis提供了事务的功能，可以保证一串 命令的原子性，中间不会被任何操作打断；

3. 数据存在内存中，读写非常的高速，可以达到10w/s的频率。

**缺点**

1. Redis3.0后才出来官方的集群方案，但仍存在一些架构上的问题（[*出处*](http://sunxiang0918.cn/2015/10/03/Redis%E9%9B%86%E7%BE%A4%E9%83%A8%E7%BD%B2/)）；

2. 持久化功能体验不佳——通过快照方法实现的话，需要每隔一段时间将整个数据库的数据写到磁盘上，代价非常高；而aof方法只追踪变化的数据，类似于mysql的binlog方法，但追加log可能过大，同时所有操作均要重新执行一遍，恢复速度慢；

3. 由于是内存数据库，所以，单台机器，存储的数据量，跟机器本身的内存大小。虽然redis本身有key过期策略，但是还是需要提前预估和节约内存。如果内存增长过快，需要定期删除数据。

**适用场景**

适用于数据变化快且数据库大小可遇见（适合内存容量）的应用程序。更具体的可参照这篇《[Redis 的 5 个常见使用场景](http://blog.jobbole.com/88383/)》译文。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

**MongoDB**

MongoDB 是一个高性能，开源，无模式的文档型数据库，开发语言是C++。它在许多场景下可用于替代传统的关系型数据库或键/值存储方式。

**1.特点**

**1.1 数据格式**

在 MongoDB 中，文档是对数据的抽象，它的表现形式就是我们常说的 BSON（Binary JSON ）。

BSON 是一个轻量级的二进制数据格式。MongoDB 能够使用 BSON，并将 BSON 作为数据的存储存放在磁盘中。

BSON 是为效率而设计的，它只需要使用很少的空间，同时其编码和解码都是非常快速的。即使在最坏的情况下，BSON格式也比JSON格式再最好的情况下存储效率高。

对于前端开发者来说，一个“文档”就相当于一个对象：

{“name":"mengxiangyue","sex":"nan"}

对于文档是有一些限制的：有序、区分大小写的，所以下面的两个文档是与上面不同的：

{”sex“:"nan","name":"mengxiangyue"}

{"Name":"mengxiangyue","sex":"nan"}

另外，对于文档的字段 MongoDB 有如下的限制：

\_id必须存在，如果你插入的文档中没有该字段，那么 MongoDB 会为该文档创建一个ObjectId作为其值。\_id的值必须在本集合中是唯一的。

多个文档则组合为一个“集合”。在 MongoDB 中的集合是无模式的，也就是说集合中存储的文档的结构可以是不同的，比如下面的两个文档可以同时存入到一个集合中：

{"name":"mengxiangyue"}

{"Name":"mengxiangyue","sex":"nan"}

**1.2 性能**

MongoDB 目前支持的存储引擎为内存映射引擎。当 MongoDB 启动的时候，会将所有的数据文件映射到内存中，然后操作系统会托管所有的磁盘操作。这种存储引擎有以下几种特点：

\* MongoDB 中关于内存管理的代码非常精简，毕竟相关的工作已经有操作系统进行托管。

\* MongoDB 服务器使用的虚拟内存将非常巨大，并将超过整个数据文件的大小。不用担心，操作系统会去处理这一切。

在《[Mongodb亿级数据量的性能测试](http://www.cnblogs.com/lovecindywang/archive/2011/03/02/1969324.html)》一文中，MongoDB 展现了强劲的大数据处理性能（数据甚至比[*Redis的*](http://www.cnblogs.com/lovecindywang/archive/2011/03/03/1969633.html)漂亮的多）。

另外，MongoDB 提供了[全索引支持](http://www.cnblogs.com/yangecnu/archive/2011/07/19/2110989.html)：包括文档内嵌对象及数组。Mongo的查询优化器会分析查询表达式，并生成一个高效的查询计划。通常能够极大的提高查询的效率。

**1.3 持久化**

MongoDB 在1.8版本之后开始支持 journal，就是我们常说的 redo log，用于故障恢复和持久化。

当系统启动时，MongoDB 会将数据文件映射到一块内存区域，称之为Shared view，在不开启 journal 的系统中，数据直接写入shared view，然后返回，系统每60s刷新这块内存到磁盘，这样，如果断电或down机，就会丢失很多内存中未持久化的数据。

当系统开启了 journal 功能，系统会再映射一块内存区域供 journal 使用，称之为 private view，MongoDB 默认每100ms刷新 privateView 到 journal，也就是说，断电或宕机，有可能丢失这100ms数据，一般都是可以忍受的，如果不能忍受，那就用程序写log吧（但开启journal后使用的虚拟内存是之前的两倍）。

**1.4 CAP类别**

MongoDB 比较灵活，可以设置成 strong consistent （CP类型）或者 eventual consistent（AP类型）。

但其默认是 CP 类型（[*了解更多*](http://stackoverflow.com/a/11297667/5100984)）。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

**2. Node下的使用**

MongoDB 在 node 环境下的驱动引擎是 [node-mongodb-native](https://github.com/mongodb/node-mongodb-native) ，作为依赖封装到 mongodb 包里，我们直接安装即可：

npm install mongodb

实例：

var mongodb = require('mongodb');

var mongodbServer = new mongodb.Server('localhost', 27017, { auto\_reconnect: true, poolSize: 10 });

var db = new mongodb.Db('mydb', mongodbServer);

/\* open db \*/

db.open(function() {

/\* Select 'contact' collection \*/

db.collection('contact', function(err, collection) {

/\* Insert a data \*/

collection.insert({

name: 'Fred Chien',

email: 'cfsghost@gmail.com',

tel: [

'0926xxx5xx',

'0912xx11xx'

]

}, function(err, data) {

if (data) {

console.log('Successfully Insert');

} else {

console.log('Failed to Insert');

}

});

/\* Querying \*/

collection.find({ name: 'Fred Chien' }, function(err, data) {

/\* Found this People \*/

if (data) {

console.log('Name: ' + data.name + ', email: ' + data.email);

} else {

console.log('Cannot found');

}

});

});

});

另外我们也可以使用MongoDB的ODM（面向对象数据库管理器） —— [mongoose](http://mongoosejs.com/docs/index.html) 来做数据库管理，具体参照其API文档。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

**3. 优缺点**

**优势**

1. 强大的自动化 shading 功能（更多[*戳这里*](http://xiezhenye.com/2012/12/mongodb-sharding-%E6%9C%BA%E5%88%B6%E5%88%86%E6%9E%90.html)）；

2. 全索引支持，查询非常高效；

3. 面向文档（BSON）存储，数据模式简单而强大。

4. 支持动态查询，查询指令也使用JSON形式的标记，可轻易查询文档中内嵌的对象及数组。

5. 支持 javascript 表达式查询，可在服务器端执行任意的 javascript函数。

**缺点**

1. 单个文档大小限制为16M，32位系统上，不支持大于2.5G的数据；

2. 对内存要求比较大，至少要保证热数据（索引，数据及系统其它开销）都能装进内存；

3. 非事务机制，无法保证事件的原子性。

**适用场景**

1. 适用于实时的插入、更新与查询的需求，并具备应用程序实时数据存储所需的复制及高度伸缩性；

2. 非常适合文档化格式的存储及查询；

3. 高伸缩性的场景：MongoDB 非常适合由数十或者数百台服务器组成的数据库。

4. 对性能的关注超过对功能的要求。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

**Couchbase**

本文之所以没有介绍 CouchDB 或 Membase，是因为它们合并了。合并之后的公司基于 Membase 与 CouchDB 开发了一款新产品，新产品的名字叫做 Couchbase。

Couchbase 可以说是集合众家之长，目前应该是最先进的Cache系统，其开发语言是 C/C++。

Couchbase Server 是个面向文档的数据库（其所用的技术来自于Apache CouchDB项目），能够实现水平伸缩，并且对于数据的读写来说都能提供低延迟的访问（这要归功于Membase技术）。

**1.特点**

**1.1 数据格式**

Couchbase 跟 MongoDB 一样都是面向文档的数据库，不过在往 Couchbase 插入数据前，需要先建立 bucket —— 可以把它理解为“库”或“表”。

因为 Couchbase 数据基于 Bucket 而导致缺乏表结构的逻辑，故如果需要查询数据，得先建立 view（跟RDBMS的视图不同，view是将数据转换为特定格式结构的数据形式如JSON）来执行。

Bucket的意义 —— 在于将数据进行分隔，比如：任何 view 就是基于一个 Bucket 的，仅对 Bucket 内的数据进行处理。一个server上可以有多个Bucket，每个Bucket的存储类型、内容占用、数据复制数量等，都需要分别指定。从这个意义上看，每个Bucket都相当于一个独立的实例。在集群状态下，我们需要对server进行集群设置，Bucket只侧重数据的保管。

每当views建立时， 就会建立indexes， index的更新和以往的数据库索引更新区别很大。 比如现在有1W数据，更新了200条，索引只需要更新200条，而不需要更新所有数据，map/reduce功能基于index的懒更新行为，大大得益。

要留意的是，对于所有文件，couchbase 都会建立一个额外的 56byte 的 metadata，这个 metadata 功能之一就是表明数据状态，是否活动在内存中。同时文件的 key 也作为标识符和 metadata 一起长期活动在内存中。

**1.2 性能**

couchbase 的精髓就在于依赖内存最大化降低硬盘I/O对吞吐量的负面影响，所以其读写速度非常快，可以达到亚毫秒级的响应。

couchbase在对数据进行增删时会先体现在内存中，而不会立刻体现在硬盘上，从内存的修改到硬盘的修改这一步骤是由 couchbase 自动完成，等待执行的硬盘操作会以write queue的形式排队等待执行，也正是通过这个方法，硬盘的I/O效率在 write queue 满之前是不会影响 couchbase 的吞吐效率的。

鉴于内存资源肯定远远少于硬盘资源，所以如果数据量小，那么全部数据都放在内存上自然是最优选择，这时候couchbase的效率也是异常高。

但是数据量大的时候过多的数据就会被放在硬盘之中。当然，最终所有数据都会写入硬盘，不过有些频繁使用的数据提前放在内存中自然会提高效率。

**1.3 持久化**

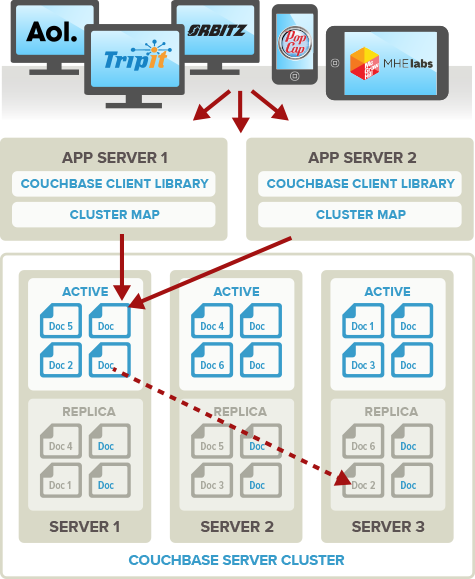
其前身之一 memcached 是完全不支持持久化的，而 Couchbase 添加了对异步持久化的支持：

Couchbase提供两种核心类型的buckets —— Couchbase 类型和 Memcached 类型。其中 Couchbase 类型提供了高可用和动态重配置的分布式数据存储，提供持久化存储和复制服务。

Couchbase bucket 具有持久性 —— 数据单元异步从内存写往磁盘，防范服务重启或较小的故障发生时数据丢失。持久性属性是在 bucket 级设置的。

**1.4 CAP类型**

Couchbase 群集所有点都是对等的，只是在创建群或者加入集群时需要指定一个主节点，一旦结点成功加入集群，所有的结点对等。



对等网的优点是，集群中的任何节点失效，集群对外提供服务完全不会中断，只是集群的容量受影响。

由于 couchbase 是对等网集群，所有的节点都可以同时对客户端提供服务，这就需要有方法把集群的节点信息暴露给客户端，couchbase 提供了一套机制，客户端可以获取所有节点的状态以及节点的变动，由客户端根据集群的当前状态计算 key 所在的位置。

就上述的介绍，Couchbase 明显属于 CP 类型。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

**2. Node下的使用**

Couchbase 对 Node SDK 提供了官方文档：<http://docs.couchbase.com/couchbase-sdk-node-1.2/index.html>

实例：

var couchbase = require("couchbase");

var bucket = new couchbase.Connection({

'bucket':'beer-sample',

'host':'127.0.0.1:8091'

}, function(err) {

if (err) {

// Failed to make a connection to the Couchbase cluster.

throw err;

}

// 获取文档

bucket.get('aass\_brewery-juleol', function(err, result) {

if (err) {

// Failed to retrieve key

throw err;

}

var doc = result.value;

console.log(doc.name + ', ABV: ' + doc.abv);

doc.comment = "Random beer from Norway";

bucket.replace('aass\_brewery-juleol', doc, function(err, result) {

if (err) {

// Failed to replace key

throw err;

}

console.log(result);

// Success!

process.exit(0);

});

});

});

**3. 优缺点**

**优势**

1. 高并发性，高灵活性，高拓展性，容错性好；

2. 以 vBucket 的概念实现更理想化的自动分片以及动态扩容（[了解更多](http://jolestar.com/couchbase/)）；

**缺点**

1. Couchbase 的存储方式为 Key/Value，但 Value 的类型很为单一，不支持数组。另外也不会自动创建doc id，需要为每一文档指定一个用于存储的 Document Indentifer；

2. 各种组件拼接而成，都是c++实现，导致复杂度过高，遇到奇怪的性能问题排查比较困难，（中文）文档比较欠缺；

3. 采用缓存全部key的策略，需要大量内存。节点宕机时 failover 过程有不可用时间，并且有部分数据丢失的可能，在高负载系统上有假死现象；

4. 逐渐倾向于闭源，社区版本（免费，但不提供官方维护升级）和商业版本之间差距比较大。

**适用场景**

1. 适合对读写速度要求较高，但服务器负荷和内存花销可遇见的需求；

2. 需要支持 memcached 协议的需求。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

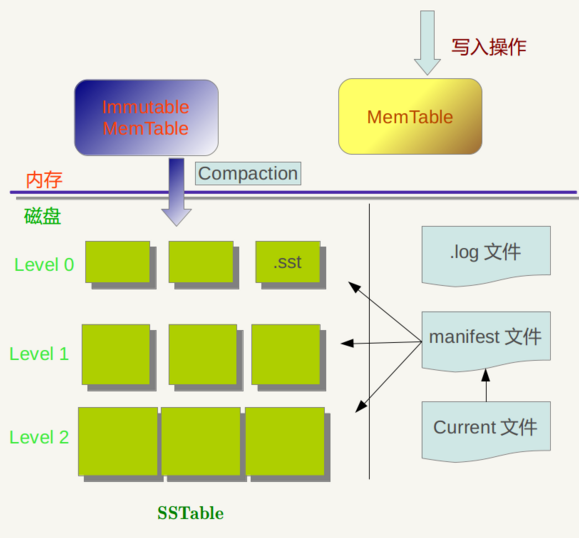
**LevelDB**

LevelDB 是由谷歌重量级工程师（Jeff Dean 和 Sanjay Ghemawat）开发的开源项目，它是能处理十亿级别规模 key-value 型数据持久性存储的程序库，开发语言是C++。

除了持久性存储，LevelDB 还有一个特点是 —— 写性能远高于读性能（当然读性能也不差）。

**1.特点**

LevelDB 作为存储系统，数据记录的存储介质包括内存以及磁盘文件，当LevelDB运行了一段时间，此时我们给LevelDb进行透视拍照，那么您会看到如下一番景象：



LevelDB 所写入的数据会先插入到内存的 Mem Table 中，再由 Mem Table 合并到只读且键值有序的 Disk Table（SSTable） 中，再由后台线程不时的对 Disk Table 进行归并。

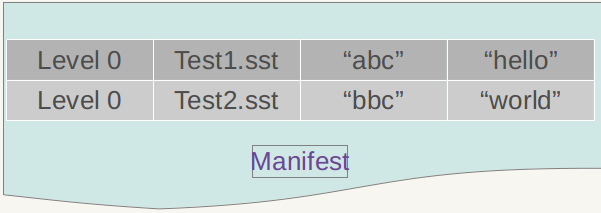
内存中存在两个 Mem Table —— 一个是可以往里面写数据的table A，另一个是正在合并到硬盘的 table B。

Mem Table 用 [skiplist](http://blog.csdn.net/ict2014/article/details/17394259" \t "_blank) 实现，写数据时，先写日志（.log），再往A插入，因为一次写入操作只涉及一次磁盘顺序写和一次内存写入，所以这是为何说LevelDb写入速度极快的主要原因。如果当B还没完成合并，而A已经写满时，写操作必须等待。

DiskTable（SSTable，格式为.sst）是分层的（leveldb的名称起源），每一个大小不超过2M。最先 dump 到硬盘的 SSTable 的层级为0，层级为0的 SSTable 的键值范围可能有重叠。如果这样的 SSTable 太多，那么每次都需要从多个 SSTable 读取数据，所以LevelDB 会在适当的时候对 SSTable 进行 Compaction，使得新生成的 SSTable 的键值范围互不重叠。

进行对层级为 level 的 SSTable 做 Compaction 的时候，取出层级为 level+1 的且键值空间与之重叠的 Table，以顺序扫描的方式进行合并。level 为0的 SSTable 做 Compaction 有些特殊：会取出 level 0 所有重叠的Table与下一层做 Compaction，这样做保证了对于大于0的层级，每一层里 SSTable 的键值空间是互不重叠的。

SSTable 中的某个文件属于特定层级，而且其存储的记录是 key 有序的，那么必然有文件中的最小 key 和最大 key，这是非常重要的信息，LevelDB 应该记下这些信息 —— Manifest 就是干这个的，它记载了 SSTable 各个文件的管理信息，比如属于哪个Level，文件名称叫啥，最小 key 和最大 key 各自是多少。下图是 Manifest 所存储内容的示意：



图中只显示了两个文件（Manifest 会记载所有 SSTable 文件的这些信息），即 Level0 的 Test1.sst 和 Test2.sst 文件，同时记载了这些文件各自对应的 key 范围，比如 Test1.sstt 的 key 范围是“an”到 “banana”，而文件 Test2.sst 的 key 范围是“baby”到“samecity”，可以看出两者的 key 范围是有重叠的。

那么上方图1中的 Current 文件是干什么的呢？这个文件的内容只有一个信息，就是记载当前的 Manifest 文件名。因为在 LevleDB 的运行过程中，随着 Compaction 的进行，SSTable 文件会发生变化，会有新的文件产生，老的文件被废弃，Manifest 也会跟着反映这种变化，此时往往会新生成 Manifest 文件来记载这种变化，而 Current 则用来指出哪个 Manifest 文件才是我们关心的那个 Manifest 文件。

注意，鉴于 LevelDB 不属于分布式数据库，故CAP法则在此处不适用。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

**2. Node下的使用**

Node 下可以使用 [LevelUP](https://github.com/Level/levelup" \t "_blank) 来操作 LevelDB 数据库

var levelup = require('levelup')

// 1) Create our database, supply location and options.

// This will create or open the underlying LevelDB store.

var db = levelup('./mydb')

// 2) put a key & value

db.put('name', 'LevelUP', function (err) {

if (err) return console.log('Ooops!', err) // some kind of I/O error

// 3) fetch by key

db.get('name', function (err, value) {

if (err) return console.log('Ooops!', err) // likely the key was not found

// ta da!

console.log('name=' + value)

})

})

LevelUp 的API非常简洁实用，具体可参考官方文档。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

**3. 优缺点**

**优势**

1. 操作接口简单，基本操作包括写记录，读记录和删除记录，也支持针对多条操作的原子批量操作；

2. 写入性能远强于读取性能，

3. 数据量增大后，读写性能下降趋平缓。

**缺点**

1. 随机读性能一般;

2. 对分布式事务的支持还不成熟。而且机器资源浪费率高。

**适应场景**

适用于对写入需求远大于读取需求的场景*（大部分场景其实都是这样）*。

http://images.cnblogs.com/cnblogs_com/vajoy/558869/o_div.jpg

**References**

hbase快速入门 —— http://wangmored.iteye.com/blog/1727731

8种 NoSQL 数据库系统对比 —— http://blog.jobbole.com/1344/

node-hbase —— https://github.com/wdavidw/node-hbase

HBase 超详细介绍 —— http://blog.csdn.net/frankiewang008/article/details/41965543

HBase 将主导 NoSQL 吗 —— http://www.oschina.net/translate/big-data-debate-will-hbase-dominate-nosq

memcached,redis,mongodb的区别以及优缺点 —— http://blog.csdn.net/senssic/article/details/30511543

redis的优点和缺点 —— http://joezhengjinhong.blog.51cto.com/7791846/1565754

redis入门 （一）认识redis*（该文章有部分示例不正确，引用时做了修正）* —— http://www.zhufengpeixun.cn/jishuziliao/Nodejishuziliao/2015-11-23/410.html

MongoDB文档、集合、数据库概念 —— http://blog.csdn.net/mengxiangyue/article/details/9879925

mongodb 持久化(5) —— http://ju.outofmemory.cn/entry/81554

关于Mongodb的全面总结 —— http://blog.csdn.net/shellching/article/details/7651979

Couchbase的简单介绍 —— http://bbs.byr.cn/#!article/Database/8365

Memcache升级版：CouchBase（一）安装篇 —— http://blog.hackroad.com/operations-engineer/linux\_server/8380.html

Couchbase介绍，更好的Cache系统 —— http://zhang.hu/couchbase/

Couchbase第一印象（架构特性） —— http://www.bubuko.com/infodetail-550423.html

NoSQL选型详解 —— http://www.thebigdata.cn/JieJueFangAn/6476.html

数据分析与处理之二（Leveldb 实现原理）—— http://www.cnblogs.com/haippy/archive/2011/12/04/2276064.html

消息中间件剖析 —— http://blog.lday.me/?p=170