

MongoDB 概念

感受三点特性

- 三个特性：
 - Powerful – 强大功能，
 - Flexible – 灵活，
 - Scalable – 可伸缩
- RDBMS有的它也有，如第二索引、范围查询和排序。另外，它内建支持MapReduce类型的聚集和地理空间索引
- 容易上手，用起来快乐。
 - 开发人员熟悉的数据模型
 - 管理人员熟悉的配置选项
 - 自然感觉的语言式的API接口和命令行

丰富数据模型

- 为什么采用文档，而抛弃关系型。
 - 因为这样，才能使得Scaling out 容易，文件转移。还有：
 - Row的概念太out了，文档是一个更灵活的模型。通过内嵌文档和数组，面向文档的方法能通过一条记录就可以展示复杂的层次关系。这个十分符合开发人员在思考面向对象语言相关数据的方式
 - 没有Schema，也就是说Key不预先定义或固定。没有了Schema，大数据的迁移基本就不需要了。应用层次来处理新Key或丢失的Key，不需要强迫所有数据都要有相同的形状（列）。

容易伸缩Scale

- 应用的数据大小正在以不可思议的速度膨胀。
传感器，更大带宽，更多有设备的入口
- 两种伸缩方案
 - Scale Up – 用一个更大的机器
 - Scale Out – 用更多的机器给数据做分区。
- MongoDB 为Scale-out 方式设计。
 - 文档型数据模型自动在多台服务器中把数据分开
 - 在一个Cluster中能自动平衡数据分布和负载。
 - 这样，程序员不需要关心Scaling方面的事。

多样的功能

- Indexing
 - 支持第二索引、多种快速查询，提供Unique、复合的及地理空间索引。
- Stored JavaScript
 - 用JavaScript代替存储过程 Stored Procedure
- 聚集
 - 支持MapReduce和其它聚集工具
- 固定大小的集合
 - 对类似Log等很有用。
- 文件存储
 - 支持大文件及文件MetaData

不牺牲速度

- MongoDB使用binary wire 协议（相对应的，HTTP/REST需要额外开销）
- 给文件增加Dynamic padding
- 预分配数据文件以保持稳定Performance
- 在默认存储引擎中使用内存映射文件，使得OS对内存管理更好
- 提供了动态查询优化器，来“记住”最快查询的方式
- 数据库Offload 客户端的处理和逻辑，而由driver或应用代码处理。

基本概念

- Document (文档) = row, 文档更expressive(表现力)
- Collection (集合) = table, a group of documents, 无Schema,
- 1个instance(实例), 可以host多个独立的databases(数据库)。每个database, 有它自己的collections和允许(权限管理)
- MongoDB使用JavaScript来读写数据和管理

文档

- 文档是MongoDB的核心概念：
 - an ordered set of keys with associated values. 有序的键值对
 - 同很多语言中的map, hash, dictionary自然对应
 - {"greeting": "Hello, world!"}, 一个键值对
 - {"greeting": "Hello, world!", "foo": 3}, 两个键值对
 - Python字典、Perl/Ruby中的Hash等不关心次序，但MongoDB是区别次序的。
 - 一个Document中各个键值对数据类型可以不一样。Key，当然只能是字符串，但是唯一的。
 - MongoDB，是类型敏感和大小写敏感的。
 - 每个文档有一个特别的Key "_id"，相当于rowID

文档举例

- {"foo": 3} {"foo": "3"} 不相同
- {"foo": 3} {"Foo": 3} 不相同
- {"greeting": "Hello, world!", "greeting": "Hello, MongoDB!"} 不可以

Collection集合

- 一组文档，相当于Table
- 无Schema，表示在一个集合中，可以有无数不同形状文档。即Key也可以完全不同。

既然任何文档可以被放到一个集合中，为什么我们
还需要多个单独的集合？

- 有什么好处呢？一定要思考
 - 在一个集合中放置不同类型的文档，对开发人员和管理员来说，是恶梦。开发人员希望返回确定类型的文档，或者应用程序能处理不同形状文档。如果我们查询博客，很难去除包含作者数据的文档。
 - 得到一个集合列表要比从一个集合中抽取一个类型的清单要快得多。例如：一个集合中有一个说明文档类型的Key，叫Type。那么在这一个集合中找到三种不同Type的Value，要比通过名字找到三个不同的集合慢得多（也是使用SubCollection子集合的原因）。
 - 把相同文档放在同一个集合中，可以实现数据本地。从一个只包括博客的集合中得到几篇博客会比一个同时从一个包括博客和作者数据的集合中得到同样博客，只需要更少的磁盘访问。
 - 当我们创建索引时，我们就开始暴露文档结构。且是按每个集合定义的。一个集合中只有一个文档类型，能使索引更有效率。
- MongoDB 只是Relax了这种需求，可以让开发人员更有灵活性选择。

用户定义的集合命名

- ""，无效
- \0, 无效，它表示名称结束
- 不可以system.开头。
 - system.users集合，表示数据库用户
 - system.namespaces集合，包含了所有数据库集合的信息
- 不可以包括\$。一般只用于系统产生的集合。

子集合的用处

- 用., 如blog.authors, 只是命名空间, 不表示blog和author是有任何关系。blog可能根本就不存在。
- 用处在于:
 - GridFS, 一个用于存放大文件的协议, 用子集合来存放 metadata.
 - Web console, 把DBTOP中的数据放在一个子集合中。
 - 很多驱动, 也用这种写法。如DBShell中, db.blog, 让你可访问blog集合。db.blog.posts, 让你访问blog.posts集合。
- 子集合是一个有效的组织数据的方法。

数据库

- MongoDB把多个集合整合到一个database中。
- 一个实例可以host多个数据库
- 每个数据库可以有它自己的允许。每个数据库被存放在单独文件中。
- 一个建议规则是：把一个应用的所有数据存放在一个数据库中。当一个服务器上有多个应用或用户组时，应该设置多个单独数据库。

数据库命名规则

- 规则：
 - “'”，无效； \0, 无效，它表示名称结束
 - 应该都小写；最长不超过64个字符。它本质是文件名。
- 有几个保留文件名不能用：
 - **admin**, 是根数据库。如果一个用户被加入**admin**，那他就能自动继承所有数据库的允许。另外，就是服务器端的命令，有些只能从**admin**数据库运行，例如列出所有数据库或**shut down**服务器。
 - **local**。这个数据库不会被复制。可以存放任何对单服务器来说是本地的集合。
 - **config**。当是**sharded setup**，系统用于存放**shards**信息

启动数据库

- 服务器端，要执行mongod,启动数据库。
- 默认调用/data/db目录下的数据库，必须要先有（windows是c:\data\db），否则启动不了。
- db connection的端口是27017
- 还同时启动了一个轻量的HTTP server, 一般端口是+1000（即28017），这样就可以做些管理动作了（通过Web方式）
- 同时注意它的pid(进程ID), master, slave。意思稍后分解。
- Ctrl+C，即停止mongod.

MongoDB Shell命令行

- 执行mongo，它是一个全功能的JavaScript的解释器。也就是说，可以执行任何的JavaScript代码，包括定义函数。请看代码：

```
> function fac(n) {  
  ... if (n<=1) return 1;  
  ... return n* fac(n-1);  
  ...}  
> fac(5); 5*4*3*2*1  
120
```

- 很酷吧！Yes，但还没到重点...

MongoDB Shell

- Shell真正强大不是在于能执行JavaScript，而在于它是一个独立的MongoDB client。
- 启动时，shell自动连接到test数据库

>use foobar ;有点像SQL

switched to db foobar

>db ;db是一个全局变量

foobar

>db.baz ;返回当前数据库的baz集合

CRUD操作 – Insert & Reau

```
>post = {"title" : "My Blog Post",
... "content" : "Here's my blog
  post."},
... "date" : new Date()}
{
"title" : "My Blog Post",
"content" : "Here's my blog post.",
"date" : ISODate("2011-10-
  20T06:37:06.594Z")
}
>db.blog.insert(post)
>db.blog
test.blog
```

```
>test.blog.find() ;会出错误信息
>db.blog.find()
{
  "_id" :
  ObjectId("4b23c3ca7525f35f94
  b60a2d"),
  "title" : "My Blog Post",
  "content" : "Here's my blog
  post.",
  "date" : ISODate("2011-10-
  20T06:37:06.594Z")
}
>db.blog.findOne() ;会出上面find()
同样结果
```



CRUD操作 – 更新 & 删除

- `>post.comments = []`
- `db.blog.update({title: "My Blog Post"}, post)`
- `db.blog.remove({title: "My Blog Post"})`

帮助

>help

>db.help()

>db.blog.update 显示update是如何工作的

Data Types – BSON

- MongoDB 不仅仅支持JavaScript的几个基本数据类型(JSON): null, boolean, numeric, string, array, object. 不能不支持date, 而且numeric中浮点数与整数差异很大, 且有32位和64位差别
- {"x": null}
- {"x": true}, {"y": false}
- 32位整数, 命令行无法表达。因为JavaScript只支持64位浮点数。32位整数会被转为64位浮点数
- 64位整数, 命令行也无法表达。命令行会用一个特别嵌入文档表达。
- 64位浮点数, 命令行中数字都是这种类型 {"x": 3.14}
- string, UTF-8字符都可以表达 {"x": "foobar"}
- symbol, 命令行不支持。如果碰到, 会转为string
- object id, 是一个独特的12字节ID {"x": ObjectId()}
- date, 以毫秒计, 不保存时区 {"x": new Date()}

数据类型 - BSON

- regular 表达式, 支持 `{"x": /foobar/i}`
- code, 表示Javascript代码 `{"x": function() {.....}}`
- binary data, 是随意字节的字符串, 命令行无法操作
- maximum value, BSON有这样一个值, 但命令行没有这个类型
- minimum value, BSON有这样一个值, 但命令行没有这个类型
- undefined。Javascript区别null和undefined。
`{"x":undefined}`
- array 列表或一套值可以以array表示 `{"x": ["a", "b", "c"]}`
- embedded document `{"x":{"foo": "bar"}}`

数据类型 - Number

- JavaScript只支持一个“number”类型。但MongoDB有3个number类型（4字节整数，8字节整数以及8字节浮点数）。
- 从命令行，如果把从数据库取出的4字节整数，写回到数据库，那么这个数将被当做浮点数对待。因此，不要从命令行改写整个文档。
- 另一个问题，8字节整数不能被8字节浮点数正确表达。

```
doc = db.numbers.findOne()
```

```
{  
  "_id" : ObjectId  
    ("4c0beecfd096a2580fe6fa08"),  
  "myInteger" : {  
    "floatApprox" : 3  
  }  
}
```

```
> db.numbers.findOne()
```

```
{  
  "_id" : ObjectId  
    ("4c0beecfd096a2580fe6fa09"),  
  "myInteger" : {  
    "floatApprox" :  
      9223372036854776000,  
    "top" : 2147483647,  
    "bottom" : 4294967295  
  }  
}
```

数据类型 - Dates

- 当创建时，要用`new Date(...)`，而不是`Date(...)`
- 调用`Date(...)` (构造器)，会返回一个代表日期的字符串，而不是一个`Date`对象。这是JavaScript的工作机制。
- `Shell`中的`Dates`会使用本地时区设置。数据库中没
有时区信息，只有代表时间的毫秒数。

数据类型 – 数组

- 数组，可以有序操作 (如lists, stacks, or queues)，也可以是无序 (就是sets).
- {"things": ["pie", 3.14]}, 一个数组可以包含不同数据类型的值。
- MongoDB对数组的支持十分好。“reach inside”十分容易。

数据类型 – 嵌入式文档

- 嵌入式文档改变了我们工作的方式。
- 好处：原先是两个表 "people" 和 "address"；现在把地址文档嵌入到人员文档就好了。这更自然，也更高效
- 坏处：修改一批有相同地址的人员的地址。关系型数据库只要改一个地址表，MongoDB中要改这N个人员中的地址文档。

- 地址文档

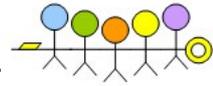
```
{  
  "name" : "John Doe",  
  "address" : {  
    "street" : "123 Park Street",  
    "city" : "Anytown",  
    "state" : "NY"  
  }  
}
```

_id和ObjectIds

- 每个文档都有一个“_id” key. “_id” key的值可以是任何类型，但默认是ObjectId。
- 在每个集合中，每个文档是唯一的，就是通过“_id”。
- ObjectIds，使用12个字节来保存，一般是用16进制字符串来表示。因此，24个字符长度，实际是12个字节。
 - Byte: 0~3, 4个字节, 是时间戳。因此，独特性和索引容易了。
 - Byte: 4~6, 3个字节, 代表机器
 - Byte: 7~8, 2个字节, 代表PID, 进程ID
 - Byte: 9~11, 3个字节, 是递增数。256*256*256个编号

_id和ObjectId

- “_id”的自动产生
- 如果插入时没有“_id”，MongoDB会自动处理加一个。但更一般的做法，是由客户端驱动的处理。原因：
 - 始终有overhead(开销)。MongoDB的哲学是尽量把服务器端的工作推倒客户端的驱动去。应用层面的Scale out比数据库层的更容易些。毕竟，把工作移到客户端，可以减轻数据库层的Scale
 - 通过客户端产生的ObjectId，驱动能提供丰富的APIs. 例如，一个驱动会有insert方法，会返回它产生的ObjectId，或者在插入文档时把它注入该文档。如果服务器端产生ObjectId,只必须要有单独的一个查询来得到ObjectId.



创建、更新、删除文档

- 单个Insert:
 - `db.foo.insert({"bar": "bar"})`
- Batch Insert –
 - 插一批数据更快，传一个文档数组到一个数据库的一个集合中。它只是一个TCP请求。overhead省了，每个消息的header处理也省了，不用像每个单独请求都要处理。使用场合：如感应数据，Log处理（用户访问痕迹，系统Log）等。
 - 如果是导入数据，可用MongoDB中的mongoimport。当然，一般到导入前都要调整改动数据，用batch insert没有问题。
 - 最大Message不能超过16MB
- Insert: internal and 隐含
- BSON结构