MongoDB 使用说明手册

版本记录

日期	版本	原因	撰稿人	审核人
2010-7-15	1.0	初始化	万韬	

目录

第1章　MongoDB 简介

1.1　功能特点

官方网址：http://www.mongodb.org/

MongoDB 是一个基于分布式文件存储的数据库开源项目。由 C++语言编写，旨在为 WEB 应用提供可护展的高性能数据存储解决方案。

它的特点是可扩展，高性能，易使用，模式自由，存储数据非常方便等，主要功能特性有：

- 面向文档存储：(类 JSON 数据模式简单而强大)。
- 高效的传统存储方式：支持二进制数据及大型对象（如照片和视频）。
- 复制及自动故障转移：Mongo 数据库支持服务器之间的数据复制，支持主-从模式及服务器之间的相互复制。
- Auto-Sharding 自动分片支持云级扩展性（处于早期 alpha 阶段）：自动分片功能支持水平的数据库集群，可动态添加额外的机器。
- 动态查询：它支持丰富的查询表达式。查询指令使用 JSON 形式的标记，可轻易查询文档中内嵌的对象及数组。
- 全索引支持：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。
- 支持 RUBY，PYTHON，JAVA，C++，PHP 等多种语言。

1.2 适用范围

适用场景：

- 适合实时的插入，更新与查询，并具备应用程序实时数据存储所需的复制及高度伸缩性。
- 适合作为信息基础设施的持久化缓存层。
- 适合由数十或数百台服务器组成的数据库。因为 Mongo 已经包含对 MapReduce 引擎的内置支持。
- Mongo 的 BSON 数据格式非常适合文档化格式的存储及查询。

不适用场景：

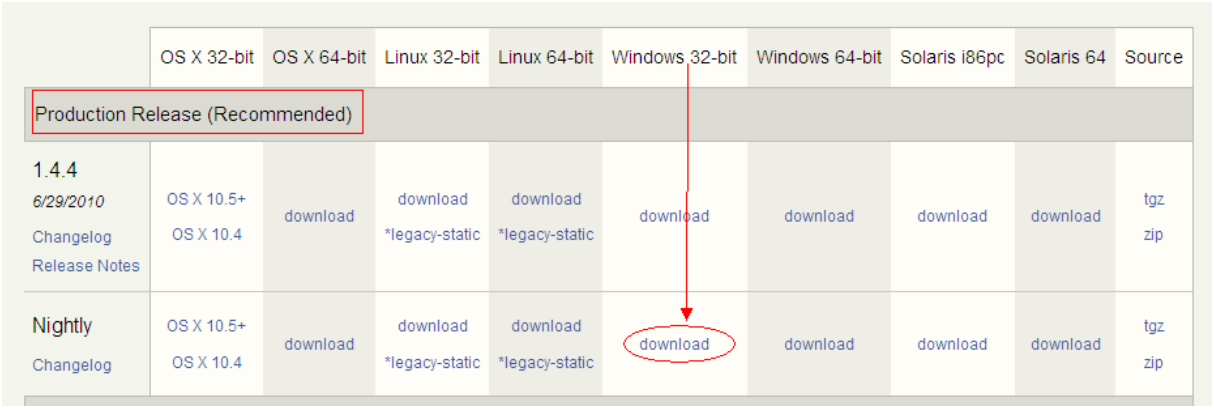
- 高度事务性的系统。
- 传统的商业智能应用。
- 级为复杂的 SQL 查询。

第2章　MongoDB 下载及安装

2.1　下载地址

http://www.mongodb.org/downloads

选择一个稳定的版本 **v1.4.5**，如下图：



	OS X 32-bit	OS X 64-bit	Linux 32-bit	Linux 64-bit	Windows 32-bit	Windows 64-bit	Solaris i86pc	Solaris 64	Source
Production Release (Recommended)									
1.4.4 *6/29/2010* Changelog Release Notes	OS X 10.5+ OS X 10.4	download	download *legacy-static	download *legacy-static	download	download	download	download	tgz zip
Nightly Changelog	OS X 10.5+ OS X 10.4	download	download *legacy-static	download *legacy-static	download	download	download	download	tgz zip

2.2　安装方法

通过 mongod –install 命令把 mongodb 注册成为 window service。

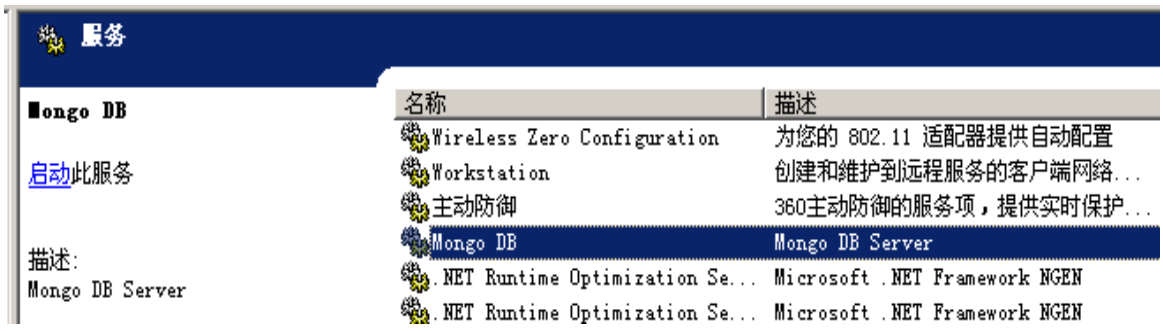
1) 创建数据库存储目录；例如：d:\data\db

2) 通过命令行执行：

```
mongod --bind_ip 127.0.0.1 --logpath d:\data\logs --logappend --dbpath d:\data\db
--directoryperdb –install
```

【注：将 mongodb 安装成服务，装该服务绑定到 IP127.0.0.1，日志文件为 d:\data\logs，以及添加方式记录。数据目录为 d:\data\db。并且每个数据库将储存在一个单独的目录（--directoryperdb）】

安装成功后，如下图：



3) 启动服务后，尝试是否可用，通过命令行进入%MONGODB_HOME%\bin 下执行 mongo.exe 命令后出现如下图所示信息表示连接成功：

```
D:\mongodb\bin>mongo.exe
MongoDB shell version: 1.4.5-pre-
url: test
connecting to: test
type "exit" to exit
type "help" for help
> _
```

第3章　MongoDB 语法

3.1 基本命令

3.1.1. 启动 mongodb

run 直接启动：

例如：mongod run

```
C:\Documents and Settings\ce>d:

D:\>cd mongodb\bin

D:\mongodb\bin>mongod run
Fri Jul 16 10:49:54 Mongo DB : starting : pid = 0 port = 27017 dbpath = /data/db
/ master = 0 slave = 0  32-bit
```

--dbpath 指定存储目录启动：

例如：mongod －dbpath = d:\ db

```
D:\mongodb\bin>mongod --dbpath=d:\DB
Fri Jul 16 10:52:21 Mongo DB : starting : pid = 0 port = 27017 dbpath = d:\DB ma
ster = 0 slave = 0  32-bit
```

--port 指定端口启动：(默认端口是:27017)

例如：mongod --port 12345。

3.1.2. 停止 mongodb

在窗口模式中，可以直接使用 Ctrl+C 停止服务。

3.2 SQL 语法

3.2.1. 基本操作

db.AddUser(username,password) 添加用户

db.auth(usrename,password) 设置数据库连接验证

db.cloneDataBase(fromhost) 从目标服务器克隆一个数据库

db.commandHelp(name) returns the help for the command

db.copyDatabase(fromdb,todb,fromhost) 复制数据库 fromdb---源数据库名称，todb---目标数据库名称，fromhost---源数据库服务器地址

db.createCollection(name,{size:3333,capped:333,max:88888}) 创建一个数据集，相当于一个表

db.currentOp() 取消当前库的当前操作

db.dropDataBase() 删除当前数据库

db.eval(func,args) run code server-side

db.getCollection(cname) 取得一个数据集合，同用法：db['cname'] or

db.getCollenctionNames() 取得所有数据集合的名称列表

db.getLastError() 返回最后一个错误的提示消息

db.getLastErrorObj() 返回最后一个错误的对象

db.getMongo() 取得当前服务器的连接对象 get the server

db.getMondo().setSlaveOk() allow this connection to read from then nonmaster membr of a replica pair

db.getName() 返回当操作数据库的名称

db.getPrevError() 返回上一个错误对象

db.getProfilingLevel()

db.getReplicationInfo() 获得重复的数据

db.getSisterDB(name) get the db at the same server as this onew

db.killOp() 停止（杀死）在当前库的当前操作

db.printCollectionStats() 返回当前库的数据集状态

db.printReplicationInfo()

db.printSlaveReplicationInfo()

db.printShardingStatus()　　返回当前数据库是否为共享数据库

db.removeUser(username)　　删除用户

db.repairDatabase()　　修复当前数据库

db.resetError()

db.runCommand(cmdObj)　　run a database command. if cmdObj is a string, turns it into {cmdObj:1}

db.setProfilingLevel(level)　　0=off,1=slow,2=all

db.shutdownServer()　　关闭当前服务程序

db.version()　　返回当前程序的版本信息

3.2.2. 数据集(表)操作

db.test.find({id:10})　　返回 test 数据集 ID=10 的数据集

db.test.find({id:10}).count()　返回 test 数据集 ID=10 的数据总数

db.test.find({id:10}).limit(2) 返回 test 数据集 ID=10 的数据集从第二条开始的数据集

db.test.find({id:10}).skip(8)　返回 test 数据集 ID=10 的数据集从 0 到第八条的数据集

db.test.find({id:10}).limit(2).skip(8)　返回 test 数据集 ID=1=的数据集从第二条到第八条的数据

db.test.find({id:10}).sort()　返回 test 数据集 ID=10 的排序数据集

db.test.findOne([query])　　返回符合条件的一条数据

db.test.getDB()　　返回此数据集所属的数据库名称

db.test.getIndexes()　　返回些数据集的索引信息

db.test.group({key:...,initial:...,reduce:...[,cond:...]})

db.test.mapReduce(mayFunction,reduceFunction,<optional params>)

db.test.remove(query)　　在数据集中删除一条数据

db.test.renameCollection(newName)　　重命名些数据集名称

db.test.save(obj)　　往数据集中插入一条数据

db.test.stats()　　返回此数据集的状态

db.test.storageSize()　　返回此数据集的存储大小

db.test.totalIndexSize()　　返回此数据集的索引文件大小

db.test.totalSize()　　返回些数据集的总大小

db.test.update(query,object[,upsert_bool]) 在此数据集中更新一条数据

db.test.validate()　　验证此数据集

db.test.getShardVersion()　　返回数据集共享版本号

MongoDB 语法与现有关系型数据库 SQL 语法比较

MongoDB 语法　　MySql 语法

db.test.find({'name':'foobar'}) <==> select * from test where name='foobar'

db.test.find()　　<==> select * from test

db.test.find({'ID':10}).count() <==> select count(*) from test where ID=10

db.test.find().skip(10).limit(20) <==> select * from test limit 10,20

db.test.find({'ID':{$in:[25,35,45]}}) <==> select * from test where ID in (25,35,45)

db.test.find().sort({'ID':-1}) <==> select * from test order by ID desc

db.test.distinct('name',{'ID':{$lt:20}}) <==> select distinct(name) from test where ID<20

db.test.group({key:{'name':true},cond:{'name':'foo'},reduce:function(obj,prev){prev.msum+= obj.marks;},initial:{msum:0}}) <==> select name,sum(marks) from test group by name

db.test.find('this.ID<20',{name:1}) <==> select name from test where ID<20

db.test.insert({'name':'foobar','age':25})<==>insert into test ('name','age') values('foobar',25)

db.test.remove({})　　<==> delete * from test

db.test.remove({'age':20})　　<==> delete test where age=20

db.test.remove({'age':{$lt:20}}) <==> elete test where age<20

db.test.remove({'age':{$lte:20}}) <==> delete test where age<=20

db.test.remove({'age':{$gt:20}}) <==> delete test where age>20

db.test.remove({'age':{$gte:20}}) <==> delete test where age>=20

db.test.remove({'age':{$ne:20}}) <==> delete test where age!=20

db.test.update({'name':'foobar'},{$set:{'age':36}}) <==> update test set age=36 where name='foobar'

db.test.update({'name':'foobar'},{$inc:{'age':3}}) <==> update test set age=age+3 where

```
name='foobar'
```

第4章　JAVA 操作 MongoDB

4.1.1. MongoDB 入门

文章分类:数据库

有关于 MongoDB 的资料现在较少，且大多为英文网站，以上内容大多由笔者翻译自官网，请翻译或理解错误之处请指证。之后笔者会继续关注 MongoDB，并翻译“Developer Zone”和“Admin Zone”的相关内容，敬请期待下期内容。

MongoDB 是一个基于分布式文件存储的数据库开源项目。由 C++语言编写。旨在为 WEB 应用提供可护展的高性能数据存储解决方案。

它的特点是高性能、易部署、易使用，存储数据非常方便。主要功能特性有：

***面向集合存储，易存储对象类型的数据。**

***模式自由。**

***支持动态查询。**

***支持完全索引，包含内部对象。**

***支持查询。**

***支持复制和故障恢复。**

***使用高效的二进制数据存储，包括大型对象（如视频等）。**

***自动处理碎片，以支持云计算层次的扩展性**

***支持 RUBY，PYTHON，JAVA，C++，PHP 等多种语言。**

***文件存储格式为 BSON（一种 JSON 的扩展）**

***可通过网络访问**

所谓“面向集合”（Collenction-Orented），意思是数据被分组存储在数据集中，被称为一个集合（Collenction)。每个集合在数据库中都有一个唯一的标识名，并且可以包含无限数目的文档。集合的概念类似关系型数据库（RDBMS）里的表（table），不

同的是它不需要定义任何模式（schema)。

模式自由（schema-free)，意味着对于存储在 mongodb 数据库中的文件，我们不需要知道它的任何结构定义。如果需要的话，你完全可以把不同结构的文件存储在同一个数据库里。

存储在集合中的文档，被存储为键-值对的形式。键用于唯一标识一个文档，为字符串类型，而值则可以是各中复杂的文件类型。我们称这种存储形式为 BSON（Binary Serialized dOcument Format）。

MongoDB 服务端可运行在 Linux、Windows 或 OS X 平台，支持 32 位和 64 位应用，默认端口为 27017。推荐运行在 64 位平台，因为 MongoDB

在 32 位模式运行时支持的最大文件尺寸为 2GB。

MongoDB 把数据存储在文件中（默认路径为：/data/db），为提高效率使用内存映射文件进行管理。

安装：

Linux/OS X 下:

1 建立数据目录

```
mkdir -p /data/db
```

2 下载压缩包

```
curl -O http://downloads.mongodb.org/linux/mongodb-linux-i686-latest.tgz
```

3 解压缩文件

```
tar xzf mongodb-linux-i386-latest.tgz
```

4 启动服务

```
bin/mongod run &
```

5 使用自带客户端连接

```
/bin/mongo
```

6 测试

```
db.foo.save( { a : 1 } )
db.foo.findOne()
```

windows 下：

1 建立数据目录 c:\data\db

2 下载压缩包，解压文件

3 启动服务

bin\mongod.exe run

4 自带客户端

bin\mongon.exe

在 LINUX 和 WINDOWS 系统下的使用大同小异，不同的地方主要是默认的数据存储目录。LINUX 类系统下存放在/data/db 下，而 WINDOWS

会存放在 C:\data\db 下。可以在启动时使用--dbpath 参数指定存储目录并启动。如：

bin\mongod.exe --dbpath d:\data\mongo

常用启动参数：

run 直接启动。例：./mongod run

--dbpath 指定特定存储目录启动，若目录不存在则创建。例：./mongod --dbpath /var/data/mongo

--port 指定端口启动。例：./mongod --port 12345

停止 MONGO 服务：

方法 1：服务端停止，可使用 Ctrl+C

方法 2：在客户端停止，可先连接客户端

./mongo

并使用命令

db.shutdownerver()

然后退出客户端

exit

使用 JAVA 语言操作 MONGODB 非常简单，只要将驱动文件加入到 CLASSPATH 中

就可以使用。

1 建立连接

要建立 **MongoDB** 的连接，你只要指定要连接到的数据库就可以。这个数据库不一定存在，如果不存在，**MongoDB** 会先为你建立这个

库。同时，在连接时你也可以具体指定要连接到的网络地址和端口。下面的是连接本机数据库的一些例子：

```
import com.mongodb.Mongo;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.mongodb.DBCursor;
import com.mongodb.MongoAdmin;

Mongo db = new Mongo("mydb");
Mongo db = new Mongo("localhost", "mydb");
Mongo db = new Mongo("localhost", 27017, "mydb");
```

2 安全验证（非必选）

MongoDB 服务可以在安全模式运行，此时任何客户端要连接数据库时需使用用户名和密码。在 **JAVA** 中可使用如下方法连接：

```
boolean auth = db.authenticate(userName, password);
```

如果用户名密码验证通过，返回值为 **true**，否则为 **false**

3 获取集合列表

每个数据库都存在零个或多个集合，需要时你可以获得他们的列表：

```
Set<String> colls = db.getCollectionNames();
```

```
for(String s : colls){
System.out.println(s);
}
```

4 获得一个集合

要获得某个特定集合，你可以指定集合的名字，并使用 **getCollection()**方法：

```
DBCollection coll = db.getCollection("testCollection");
```

当你获取了这个集合对象，你就可以对数据进行增删查改之类的操作。

5 插入文档

当你获得了一个集合对象，你就可以把文档插入到这个对象中。例如，存在一个 **JSON** 式的小文档：

```
{
"name" : "MongoDB",
"type" : "database",
"count" : 1,
"info" : {
  x : 203,
  y : 102
 }
}
```

请注意，这个文档包含一个内部文档。我们可以使用 **BasicDBObject** 类来创建这个文档，并且使用 **insert()**方法方便地将它插入到集

合中。

```
BasicDBObject doc = new BasicDBObject();
doc.put("name", "MongoDB");
doc.put("type", "database");
doc.put("count", 1);
```

```
BasicDBObject info = new BasicDBObject();
info.put("x", 203);
info.put("y", 102);

doc.put("info", info);

coll.insert(doc);
```

6 使用 **findOne()**查找集合中第一个文档

要查找我们上一步插入的那个文档，可以简单地使用 **findOne()**操作来获取集合中第一个文档。这个方法返回一个单一文档（这是相对于使用 **DBCursor** 的 **find()**操作的返回），这对于只有一个文档或我们刚插入第一个文档时很有用，因为此时并不需要使用光标。

```
DBObject myDoc = coll.findOne();
System.out.println(myDoc);
```

返回类似：

```
{
"_id" : "ac907a1f5b9d5e4a233ed300" ,
"name" : "MongoDB" ,
"type" : 1 ,
"info" : {
  "x" : 203 ,
  "y" : 102} ,
"_ns" : "testCollection"
}
```

注意_id 和_ns 元素是由 **MongoDB** 自动加入你的文档。记住：**MongoDB** 内部存储使用的元素名是以“_”做为开始。

7 加入多种文档

为了做更多有趣的查询试验，让我们向集合中加入多种文档类型，象：

```
{
"i" : value
}
```

可以通过循环来实现

```
for(int i = 0; i < 100; i++){
coll.insert(new BasicDBObject().append("i", i));
}
```

注意我们可以在一个集合中插入不同类型的文档，这就是我们所说的“模式自由”（schema-free）。

8 统计文档数量

使用 **getCount()**方法

```
System.out.println(coll.getCount());
```

9 使用光标（**cursor**）来获取全部文档

为了获取集合中的所有文档，我们可以使用 **find()**方法。这个方法返回一上 **DBCursor** 对象，来允许我们将符合查询条件的文档迭代

出来。

```
DBCursor cur = coll.find();
while(cur.hasNext()){
System.out.println(cur.next());
}
```

10 在查询中获取单一文档

我们可以创建一个查询，并传递给 **find()**方法来获取集合中所有文档的一个子集。例如，我们想要查询域名为"i"，并且值为 **71** 的文档：

```
BasicDBObject query = new BasicDBObject();
query.put("i", 71);
cur = coll.find(query);
while(cur.hasNext()){
System.out.println(cur.next());
}
```

11 使用条件查询获取集合

例如，我们想要查询所有 **i>50** 的文档：

```
BasicDBObject query = new BasicDBObject();
query.put("i", new BasicDBObject("$gt", 50));
cur = coll.find(query);
while(cur.hasNext()){
System.out.println(cur.next());
}
```

当然，我们也可以做 **20 < i <= 30** 的查询

```
BasicDBObject query = new BasicDBObject();
query.put("i", new BasicDBObject("$gt", 20).append("$lte", 30));
cur = coll.find(query);
while(cur.hasNext()){
System.out.println(cur.next());
}
```

12 创建索引

MongoDB 支持索引，而且很容易在集合上增加索引。要创建索引，只需要指定要加索引的属性，并且指定升序（**1**）或降序即可（**-1**）。

```
coll.createIndex(new BasicDBObject("i", 1));
```

13 获取索引列表

```
List<DBObject> list = coll.getIndexInfo();
for(DBObject o : list){
System.out.println(o);
}
```

14 MongoDB 管理函数

管理函数在 **com.mongodb.MongoAdmin** 类中定义。

例 **A**：获取数据库列表

```
MongoAdmin admin = new MongoAdmin();
for(String s : admin.getDatabaseNames()){
  System.out.println(s);
}
```

例 **B**：获取数据库对象

```
Mongo m = admin.getDB("mydb");
```

例 **C**：删除数据库

```
admin.dropDatabase("mydb");
```

15 用 DBObject 存储 JAVA 对象

MongoDB for JAVA 驱动中提供了用于向数据库中存储普通对象的接口 **DBObject**

例如，存在一个需要存储的对象类 **Tweet**

```
public class Tweet implements DBObject{
/*...*/
}
```

可以使用如下代码：

```
Tweet myTweet = new Tweet();
myTweet.put("user", userId);
myTweet.put("message", message);
```

myTweet.put("date", new Date());

collection.insert(myTweet);

当一个文档从 **MongoDB** 中取出时，它会自动把文档转换成 **DBObject** 接口类型，要将它实例化为你的对象，需使用

DBCollection.setObjectClass()。

collection.setObjectClass(Tweet);

Tweet myTweet = (Tweet)collection.findOne();

16 JAVA 驱动的并发性

JAVA 的 **MongoDB** 驱动是线程安全的。如果你将它用在 **WEB** 服务中，可以创建它的一个单例，并在所有请求中使用它。

然而，如果你需要在一个会话（例如 **HTTP** 请求）中保证事务一致性，也许你会希望在这个会话中对驱动使用同一个端口。这仅仅在

请求量非常大的环境中，例如你经常会读取刚写入的数据。

为了这一点，你需要使用如下代码：

Mongo m;

m.restartStart();

// code.....

m.requestDone();

以上介绍了简单的 **mongoDB** 使用，更多信息请查阅 **MongoDB API for Java**。

官方主页：**http://www.mongodb.org/display/DOCS/Home**

第5章　其它

5.1　正在整理中……