一、 NoSQL 与 MongoDB

1. NoSQL

NoSQL(NoSQL = Not Only SQL)，意即反 SQL 运动，是一项全新的数据库革命性运动，早期就有人提出，发展至 2009 年趋势越发高涨。NoSQL 的拥护者们提倡运用非关系型的数据存储，相对于目前铺天盖地的关系型数据库运用，这一概念无疑是一种全新的思维的注入。

1. 为什么使用 NoSQL：
   对数据库高并发读写。
   对海量数据的高效率存储和访问。
   对数据库的高可扩展性和高可用性。
2. 弱点：
   数据库事务一致性需求
   数据库的写实时性和读实时性需求
   对复杂的 SQL 查询，特别是多表关联查询的需求
3. NoSQL 分类：
   a) key-value 存储

Examples	Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB
典型应用场景	内容缓存，主要用于处理大量数据的高访问负载，也用于一些日志系统等等。
数据模型	Key 指向 Value 的键值对，通常用 hash table 来实现
强项	查找速度快
弱项	数据无结构化，通常只被当作字符串或者二进制数据

   b) 列式数据库

Examples	Cassandra, HBase, Riak
典型应用场景	分布式的文件系统
数据模型	以列簇式存储，将同一列数据存在一起
强项	查找速度快，可扩展性强，更容易进行分布式扩展
弱项	功能相对局限

c) 文档型数据库

Examples	CouchDB, MongoDb
典型应用场景	Web 应用（与 Key-Value 类似，Value 是结构化的，不同的是数据库能够了解 Value 的内容）
数据模型	Key-Value 对应的键值对，Value 为结构化数据
强项	数据结构要求不严格，表结构可变，不需要像关系型数据库一样需要预先定义表结构
弱项	查询性能不高，而且缺乏统一的查询语法。

d) 图结构数据库

Examples	Neo4J, InfoGrid, Infinite Graph
典型应用场景	社交网络，推荐系统等。专注于构建关系图谱
数据模型	图结构
强项	利用图结构相关算法。比如最短路径寻址，N 度关系查找等
弱项	很多时候需要对整个图做计算才能得出需要的信息，而且这种结构不太好做分布式的集群方案。

2. MongoDB

1. 简介

Mongo 是一个高性能，开源，无模式的文档型数据库，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。MongoDB 使用 C++开发。不支持 SQL，但有自己功能强大的查询语法。MongoDB 使用 BSON 作为数据存储和传输的格式。BSON 是一种类似 JSON 的二进制序列化文档，支持嵌套对象和数组。MongoDB 很像 MySQL，document 对应 MySQL 的 row，collection 对应 MySQL 的 table。

2. 特点

高性能、易部署、易使用，存储数据非常方便。

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 Python，PHP，Ruby，Java，C，C#，Javascript，Perl 及 C++语言的驱动程序,

社区中也提供了对 Erlang 及.NET 等平台的驱动程序。

文件存储格式为 BSON（一种 JSON 的扩展）

可通过网络访问

3. 功能

面向集合的存储：适合存储对象及 JSON 形式的数据

动态查询：MongoDB 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。

完整的索引支持：包括文档内嵌对象及数组。MongoDB 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。

查询监视：MongoDB 包含一系列监视工具用于分析数据库操作的性能。

复制及自动故障转移：MongoDB 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。复制的主要目标是提供冗余及自动故障转移。

高效的传统存储方式：支持二进制数据及大型对象（如照片或图片）

自动分片以支持云级别的伸缩性：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

4. 适用场合

网站数据：Mongo 非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。

缓存：由于性能很高，Mongo 也适合作为信息基础设施的缓存层。在系统重启之后，由 Mongo 搭建的持久化缓存层可以避免下层的数据源 过载。

大尺寸，低价值的数据：使用传统的关系型数据库存储一些数据时可能会比较昂贵，在此之前，很多时候程序员往往会选择传统的文件进行存储。

高伸缩性的场景：Mongo 非常适合由数十或数百台服务器组成的数据库。Mongo 的路线图中已经包含对 MapReduce 引擎的内置支持。

用于对象及 JSON 数据的存储：Mongo 的 BSON 数据格式非常适合文档化格式的存储及查询。

5. 不适用场合

高度事务性的系统：例如银行或会计系统。传统的关系型数据库目前还是更适用于需要大量原子性复杂事务的应用程序。

传统的商业智能应用：针对特定问题的 BI 数据库会对产生高度优化的查询方式。对于此类应用，数据仓库可能是更合适的选择。

二、 安装部署

1. 下载

在 MongoDB 的官方网站 http://www.mongodb.org/downloads 下载，建议下载最新版本 2.2.2，注意最新的 32 位版本对 windows 的支持的最低版本是 vista，就是说不支持 xp 了。

2. 部署启动

拷贝下载的版本到某个目录，如 D:\MongoDB，解压。
最简单的启动：切换到 bin 目录，执行命令：mongod 即可。一般最起码要指定一下数据库文件的存放位置的，如：mongod --dbpath D:\MongoDb\data。

1. 启动参数：

使用 mongod --help 可以查看所有的启动参数，都有相应的简要说明。主要启动参数：

名称	说明
dbpath	数据文件存放路径，每个数据库会在其中创建一个子目录，用于防止同一个实例多次运行的 mongod.lock 也保存在此目录中
logpath	日志文件
logappend	日志采用追加模式（默认是覆写模式）
bind_ip	对外服务的绑定 ip，一般设置为空，及绑定在本机所有可用 ip 上
port	对外服务端口。Web 管理端口在这个 port 的基础上+1000
fork	以后台 Daemon 形式运行服务
journal	开启日志功能，通过保存操作日志来降低单机故障的恢复时间
syncdelay	系统同步刷新磁盘的时间，单位为秒，默认是 60 秒
directoryperdb	每个 db 存放在单独的目录中，建议设置该参数
maxConns	最大连接数
repairpath	执行 repair 时的临时目录。在如果没有开启 journal，异常 down 机后重启，必须执行 repair 操作
nohttpinterface	关闭管理接口
auth	开启安全认证检查

2. 启动参数文件：

使用—config 参数可以指定 MongoDB 的启动参数文件，具体操作步骤如下：

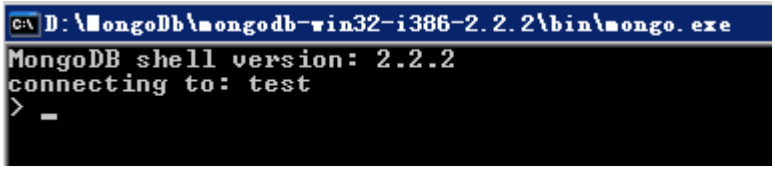
1) 在 MongoDB 的 bin 目录，创建配置文件，如 mongdb.conf，写入配置信息，参数信息以键值对的形式存放。
   dbpath = D:\MongoDb\data
2) 在命令行启动，执行命令：mongod --config mongodb.conf

上面的操作与直接执行命令 mongod --dbpath D:\MongoDb\data 效果一致。

3. 简单验证

1. Shell 验证

双击直接启动 mongo.exe，可以连接到本机的默认端口的数据库，效果如下：



输入 db 命令，可以查看当前的数据库，还可以把这个 shell 当成计算器来使用，如下：

2. 管理端口验证

MongoDB 的管理接口占用的端口比 MongoDB 本身的端口（就是上面可以用—port 指定的端口）大 1000，默认就是 28107。我们可以在浏览器中打开，查看 MongoDB 的状态：

4. 安全认证

MongoDB 官方建议把 MongoDB 部署在局域网内，不允许外部访问，另外推荐使用操作系统提供的防火墙等限制，MongoDB 也提供了简单的用户名/密码认证。默认情况下 MongoDB 是关闭用户名验证的。

MongoDB 的权限是按照数据库划分的，每个数据库有两种权限：全部权限和只读权限。另外 MongoDB 有一个比较特殊的数据库 admin。在 admin 数据库中的用户，切换到其它数据库就拥有类似 admin 库中的权限，即：admin 库的全部权限用户拥有其它数据库的全部权限，admin 库的只读权限拥有其它库的只读权限。

注意：如果 MongoDB 只能在严格的局域网防卫内访问，不建议开启安全认证，因为这样可以简化分片及主备等生产环境下的操作！

开启认证步骤：

1. 在 MongoDB 没有认证的模式下登录，创建 admin 库的全权限管理员

MongoDB shell version: 2.2.2

```
connecting to: test
> use admin
switched to db admin
> db.addUser("admin","ecss20");
{
        "user" : "admin",
        "readOnly" : false,
        "pwd" : "eeb25cea37b23907e64ad26323c6ed62",
        "_id" : ObjectId("50dbedd97d783328564f48b4")
}
>
```

2. 以认证模式登录 MongoDB
   关闭 MongoDB，重新启动，增加参数—auth。如下：
   ```
   mongod --dbpath D:\MongoDb\data --auth
   ```
3. 登录，切换到其它数据库，创建用户

```
MongoDB shell version: 2.2.2
connecting to: test
> use admin
switched to db admin
> db.auth("admin","ecss20");
1
> use test
switched to db test
> db.addUser("testUser","1");
{
        "user" : "testUser",
        "readOnly" : false,
        "pwd" : "ae898013f74412e87bdfe3a570bff7b9",
        "_id" : ObjectId("50dbf445a5f655ecd879923b")
}
> db.addUser("readonly","1",true);
{
        "user" : "readonly",
        "readOnly" : true,
        "pwd" : "27750e2df7b152d01c2531e61918f276",
        "_id" : ObjectId("50dbf460a5f655ecd879923c")
}
>
```

4. 权限验证
   a) 以 test 库的 testUser 用户登录

```
MongoDB shell version: 2.2.2
connecting to: test
> db.test.find();
error: {
```

```
        "$err" : "unauthorized db:test ns:test.test lock type:0 client:127.0.0.1
",
        "code" : 10057
}'
> db.auth("testUser","1");
1
> db.test.find();
> db.test.insert({"x":2});
> db.test.find();
{ "_id" : ObjectId("50dbf684a97bc7a2429d9109"), "x" : 2 }
>
```

b) 以 test 库的 readonly 用户登录

```
MongoDB shell version: 2.2.2
connecting to: test
> db.auth("readonly","1");
1
> db.test.find();
{ "_id" : ObjectId("50dbf684a97bc7a2429d9109"), "x" : 2 }
> db.test.insert({"x":3});
unauthorized
>
```

在 shell 中直接使用用户名，密码登录某个数据库的命令格式如下：

```
mongo -u readonly -p 1 192.168.85.136:27017/test
```

三、 Shell 基本操作

1. DB shell 数据库操作

shell 命令操作语法和 JavaScript 很类似，其实控制台底层的查询语句都是用 JavaScript 脚本完成操作的。

数据库

1、 Help 查看命令提示

```
help
db.help();
db.yourColl.help();
db.youColl.find().help();
rs.help();
```

2、 切换/创建数据库

```
>use yourDB;
```

当创建一个集合(table)的时候会自动创建当前数据库

3、 查询所有数据库
show dbs;

4、 删除当前使用数据库
db.dropDatabase();

5、 从指定主机上克隆数据库
db.cloneDatabase(“127.0.0.1”);
将指定机器上的数据库的数据克隆到当前数据库

6、 从指定的机器上复制指定数据库数据到某个数据库
db.copyDatabase("mydb", "temp", "127.0.0.1");
将本机的 mydb 的数据复制到 temp 数据库中

7、 修复当前数据库
db.repairDatabase();

8、 查看当前使用的数据库
db.getName();
db;
db 和 getName 方法是一样的效果，都可以查询当前使用的数据库

9、 显示当前 db 状态
db.stats();

10、 当前 db 版本
db.version();

11、 查看当前 db 的链接机器地址
db.getMongo();

Collection 聚集集合

1、 创建一个聚集集合（table）
db.createCollection(“collName”, {size: 20, capped: 5, max: 100});

2、 得到指定名称的聚集集合（table）
db.getCollection("account");

3、 得到当前 db 的所有聚集集合
db.getCollectionNames();

4、 显示当前 db 所有聚集索引的状态
db.printCollectionStats();

用户相关

1、 添加一个用户
db.addUser("name");
db.addUser("userName", "pwd123", true);
添加用户、设置密码、是否只读

2、 数据库认证、安全模式
db.auth("userName", "123123");

3、 显示当前所有用户
show users;

4、 删除用户
db.removeUser("userName");

其他

1、 查询之前的错误信息
db.getPrevError();

2、 清除错误记录
db.resetError();

2. Collection 聚集集合操作

查看聚集集合基本信息

1、 查看帮助
db.yourColl.help();

2、 查询当前集合的数据条数
db.yourColl.count();

3、 查看数据空间大小
db.userInfo.dataSize();

4、 得到当前聚集集合所在的 db

db.userInfo.getDB();

5、 得到当前聚集的状态
db.userInfo.stats();

6、 得到聚集集合总大小
db.userInfo.totalSize();

7、 聚集集合储存空间大小
db.userInfo.storageSize();

8、 Shard 版本信息
db.userInfo.getShardVersion()

9、 聚集集合重命名
db.userInfo.renameCollection("users");
将 userInfo 重命名为 users

10、 删除当前聚集集合
db.userInfo.drop();

聚集集合查询

1、 查询所有记录
db.userInfo.find();
相当于：select * from userInfo;
默认每页显示 20 条记录，当显示不下的情况下，可以用 it 迭代命令查询下一页数据。注意：键入 it 命令不能带“;”
但是你可以设置每页显示数据的大小，用 DBQuery.shellBatchSize = 50;这样每页就显示 50 条记录了。

2、 查询去掉后的当前聚集集合中的某列的重复数据
db.userInfo.distinct("name");
会过滤掉 name 中的相同数据
相当于：select distict name from userInfo;

3、 查询 age = 22 的记录
db.userInfo.find({"age": 22});
相当于： select * from userInfo where age = 22;

4、 查询 age > 22 的记录
db.userInfo.find({age: {$gt: 22}});
相当于：select * from userInfo where age > 22;

5、 查询 age < 22 的记录
db.userInfo.find({age: {$lt: 22}});
相当于：select * from userInfo where age < 22;

6、 查询 age >= 25 的记录
db.userInfo.find({age: {$gte: 25}});
相当于：select * from userInfo where age >= 25;

7、 查询 age <= 25 的记录
db.userInfo.find({age: {$lte: 25}});

8、 查询 age >= 23 并且 age <= 26
db.userInfo.find({age: {$gte: 23, $lte: 26}});

9、 查询 name 中包含 mongo 的数据
db.userInfo.find({name: /mongo/});
//相当于%%
select * from userInfo where name like '%mongo%';

10、 查询 name 中以 mongo 开头的
db.userInfo.find({name: /^mongo/});
select * from userInfo where name like 'mongo%';

11、 查询指定列 name、age 数据
db.userInfo.find({}, {name: 1, age: 1});
相当于：select name, age from userInfo;
当然 name 也可以用 true 或 false,当用 ture 的情况下河 name:1 效果一样，如果用 false 就是排除 name，显示 name 以外的列信息。

12、 查询指定列 name、age 数据, age > 25
db.userInfo.find({age: {$gt: 25}}, {name: 1, age: 1});
相当于：select name, age from userInfo where age > 25;

13、 按照年龄排序
升序：db.userInfo.find().sort({age: 1});
降序：db.userInfo.find().sort({age: -1});

14、 查询 name = zhangsan, age = 22 的数据
db.userInfo.find({name: 'zhangsan', age: 22});
相当于：select * from userInfo where name = 'zhangsan' and age = '22';

15、 查询前 5 条数据
db.userInfo.find().limit(5);
相当于：select top 5 * from userInfo;

16、 查询 10 条以后的数据

```
db.userInfo.find().skip(10);
```

相当于：select * from userInfo where id not in (

```
select top 10 * from userInfo
);
```

17、 查询在 5-10 之间的数据

```
db.userInfo.find().limit(10).skip(5);
```

可用于分页，limit 是 pageSize，skip 是第几页*pageSize

18、 or 与 查询

```
db.userInfo.find({$or: [{age: 22}, {age: 25}]});
```

相当于：select * from userInfo where age = 22 or age = 25;

19、 查询第一条数据

```
db.userInfo.findOne();
```

相当于：select top 1 * from userInfo;

```
db.userInfo.find().limit(1);
```

20、 查询某个结果集的记录条数

```
db.userInfo.find({age: {$gte: 25}}).count();
```

相当于：select count(*) from userInfo where age >= 20;

21、 按照某列进行排序

```
db.userInfo.find({sex: {$exists: true}}).count();
```

相当于：select count(sex) from userInfo;

索引

1、 创建索引

```
db.userInfo.ensureIndex({name: 1});
db.userInfo.ensureIndex({name: 1, ts: -1});
```

2、 查询当前聚集集合所有索引

```
db.userInfo.getIndexes();
```

3、 查看总索引记录大小

```
db.userInfo.totalIndexSize();
```

4、 读取当前集合的所有 index 信息

```
db.users.reIndex();
```

5、 删除指定索引

db.users.dropIndex("name_1");

6、 删除所有索引索引
db.users.dropIndexes();

修改、添加、删除集合数据

1、 添加
db.users.save({name: ‘zhangsan’, age: 25, sex: true});
添加的数据的数据列，没有固定，根据添加的数据为准

2、 修改
db.users.update({age: 25}, {$set: {name: 'changeName'}}, false, true);
相当于：update users set name = ‘changeName’ where age = 25;

db.users.update({name: 'Lisi'}, {$inc: {age: 50}}, false, true);
相当于：update users set age = age + 50 where name = ‘Lisi’;

db.users.update({name: 'Lisi'}, {$inc: {age: 50}, $set: {name: 'hoho'}}, false, true);
相当于：update users set age = age + 50, name = ‘hoho’ where name = ‘Lisi’;

3、 删除
db.users.remove({age: 132});

4、 查询修改删除
```
db.users.findAndModify({
    query: {age: {$gte: 25}},
    sort: {age: -1},
    update: {$set: {name: 'a2'}, $inc: {age: 2}},
    remove: true
});

db.runCommand({ findandmodify : "users",
    query: {age: {$gte: 25}},
    sort: {age: -1},
    update: {$set: {name: 'a2'}, $inc: {age: 2}},
    remove: true
});
```

参数	详解	默认值
query	查询过滤条件	{}
sort	如果多个文档符合查询过滤条件，将以该参数指定的排列方式选择出排在首位的对象，该对象将被操作	{}
remove	若为 true，被选中对象将在返回前被删除	N/A
update	一个 修改器对象	N/A

new	若为 true，将返回修改后的对象而不是原始对象。在删除操作中，该参数被忽略。	false
fields	参见 Retrieving a Subset of Fields (1.5.0+)	All fields
upsert	创建新对象若查询结果为空。 示例 (1.5.4+)	false

语句块操作

1、 简单 Hello World
print("Hello World!");
这种写法调用了 print 函数，和直接写入"Hello World!"的效果是一样的；

2、 将一个对象转换成 json
tojson(new Object());
tojson(new Object('a'));

3、 循环添加数据
> for (var i = 0; i < 30; i++) {
... db.users.save({name: "u_" + i, age: 22 + i, sex: i % 2});
... };
这样就循环添加了 30 条数据，同样也可以省略括号的写法
> for (var i = 0; i < 30; i++) db.users.save({name: "u_" + i, age: 22 + i, sex: i % 2});
也是可以的，当你用 db.users.find()查询的时候，显示多条数据而无法一页显示的情况下，可以用 it 查看下一页的信息；

4、 find 游标查询
>var cursor = db.users.find();
> while (cursor.hasNext()) {
printjson(cursor.next());
}
这样就查询所有的 users 信息，同样可以这样写
var cursor = db.users.find();
while (cursor.hasNext()) { printjson(cursor.next); }
同样可以省略{}号

5、 forEach 迭代循环
db.users.find().forEach(printjson);
forEach 中必须传递一个函数来处理每条迭代的数据信息

6、 将 find 游标当数组处理
var cursor = db.users.find();
cursor[4];
取得下标索引为 4 的那条数据
既然可以当做数组处理，那么就可以获得它的长度：cursor.length();或者 cursor.count();

那样我们也可以用循环显示数据
for (var i = 0, len = c.length(); i < len; i++) printjson(c[i]);

7、 将 find 游标转换成数组
> var arr = db.users.find().toArray();
> printjson(arr[2]);
用 toArray 方法将其转换为数组

8、 定制我们自己的查询结果
只显示 age <= 28 的并且只显示 age 这列数据
db.users.find({age: {$lte: 28}}, {age: 1}).forEach(printjson);
db.users.find({age: {$lte: 28}}, {age: true}).forEach(printjson);
排除 age 的列
db.users.find({age: {$lte: 28}}, {age: false}).forEach(printjson);

9、 forEach 传递函数显示信息
db.things.find({x:4}).forEach(function(x) {print(tojson(x));});
上面介绍过 forEach 需要传递一个函数，函数会接受一个参数，就是当前循环的对象，然后在函数体重处理传入的参数信息。

四、 Java 基本操作

1. 准备工作

添加 MongoDB 的 java 驱动包
建议使用 Maven 项目，直接在 pom 中添加 MongoDB 的驱动包即可

```
        <dependency>
            <groupId>org.mongodb</groupId>
            <artifactId>mongo-java-driver</artifactId>
            <version>2.9.1</version>
        </dependency>
```

2. Java 操作 MongoDB 示例

在本示例之前你需要启动 mongod.exe 的服务，启动后，下面的程序才能顺利执行；

简单的 mongoDB 数据库操作

Mongo mongo = new Mongo();
这样就创建了一个 MongoDB 的数据库连接对象，它默认连接到当前机器的 localhost 地址，端口是 27017。

DB db = mongo.getDB(“test”);
这样就获得了一个 test 的数据库，如果 mongoDB 中没有创建这个数据库也是可以正常运行的。如果你读过上一篇文章就知道，mongoDB 可以在没有创建这个数据库的情况下，完成数据的添加操作。当添加的时候，没有这个库，mongoDB 会自动创建当前数据库。
如果已经添加了安全检查，就要先进行安全验证才能继续下一步：
boolean b=db.authenticate("testUser", new char[]{'1'});
得到了 db，下一步我们要获取一个“聚集集合 DBCollection”，通过 db 对象的 getCollection 方法来完成。
DBCollection users = db.getCollection("users");
这样就获得了一个 DBCollection，它相当于我们数据库的“表”。
查询所有数据
DBCursor cur = users.find();
while (cur.hasNext()) {
System.out.println(cur.next());
}

完整源码
```
package com.hoo.test;

import java.net.UnknownHostException;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.Mongo;
import com.mongodb.MongoException;
import com.mongodb.util.JSON;

/**
 * <b>function:</b>MongoDB 简单示例
 * @author hoojo
 * @createDate 2011-5-24 下午 02:42:29
 * @file SimpleTest.java
 * @package com.hoo.test
 * @project MongoDB
 * @blog http://blog.csdn.net/IBM_hoojo
 * @email hoojo_@126.com
 * @version 1.0
 */
public class SimpleTest {

    public static void main(String[] args) throws UnknownHostException, MongoException {
        Mongo mg = new Mongo();
        DB db = mg.getDB("test");
        db.authenticate("testUser", new char[]{'1'});
```

```
        //查询所有的聚集集合
        for (String name : db.getCollectionNames()) {
            System.out.println("collectionName: " + name);
        }

        DBCollection users = db.getCollection("users");

        //查询所有的数据
        DBCursor cur = users.find();
        while (cur.hasNext()) {
            System.out.println(cur.next());
        }
        System.out.println(cur.count());
        System.out.println(cur.getCursorId());
        System.out.println(JSON.serialize(cur));
    }
}
```

完成 CRUD 操作

首先建立一个 MongoDB4CRUDTest.java，基本测试代码如下：

```
package com.hoo.test;

import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.List;
import org.bson.types.ObjectId;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import com.mongodb.BasicDBObject;
import com.mongodb.Bytes;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.Mongo;
import com.mongodb.MongoException;
import com.mongodb.QueryOperators;
import com.mongodb.util.JSON;

/**
 * <b>function:</b>实现 MongoDB 的 CRUD 操作
```

```
 * @author hoojo
 * @createDate 2011-6-2 下午 03:21:23
 * @file MongoDB4CRUDTest.java
 * @package com.hoo.test
 * @project MongoDB
 * @blog http://blog.csdn.net/IBM_hoojo
 * @email hoojo_@126.com
 * @version 1.0
 */
public class MongoDB4CRUDTest {

    private Mongo mg = null;
    private DB db;
    private DBCollection users;

    @Before
    public void init() {
        try {
            mg = new Mongo();
            //mg = new Mongo("localhost", 27017);
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (MongoException e) {
            e.printStackTrace();
        }
        //获取 temp DB；如果默认没有创建，mongodb 会自动创建
        db = mg.getDB("temp");
        //获取 users DBCollection；如果默认没有创建，mongodb 会自动创建
        users = db.getCollection("users");
    }

    @After
    public void destory() {
        if (mg != null)
            mg.close();
        mg = null;
        db = null;
        users = null;
        System.gc();
    }

    public void print(Object o) {
        System.out.println(o);
    }
```

```
}
```

添加操作

在添加操作之前，我们需要写个查询方法，来查询所有的数据。代码如下：

```
/**
 * <b>function:</b> 查询所有数据
 * @author hoojo
 * @createDate 2011-6-2 下午 03:22:40
 */
private void queryAll() {
	print("查询 users 的所有数据：");
	//db 游标
	DBCursor cur = users.find();
	while (cur.hasNext()) {
		print(cur.next());
	}
}

@Test
public void add() {
	//先查询所有数据
	queryAll();
	print("count: " + users.count());

	DBObject user = new BasicDBObject();
	user.put("name", "hoojo");
	user.put("age", 24);
	//users.save(user)保存，getN()获取影响行数
	//print(users.save(user).getN());

	//扩展字段，随意添加字段，不影响现有数据
	user.put("sex", "男");
	print(users.save(user).getN());

	//添加多条数据，传递 Array 对象
	print(users.insert(user, new BasicDBObject("name", "tom")).getN());

	//添加 List 集合
	List<DBObject> list = new ArrayList<DBObject>();
	list.add(user);
	DBObject user2 = new BasicDBObject("name", "lucy");
	user.put("age", 22);
```

```
        list.add(user2);
        //添加 List 集合
        print(users.insert(list).getN());

        //查询下数据，看看是否添加成功
        print("count: " + users.count());
        queryAll();
}
```

删除数据

```
@Test
public void remove() {
        queryAll();
        print("删除 id = 4de73f7acd812d61b4626a77：" + users.remove(new BasicDBObject("_id", new ObjectId("4de73f7acd812d61b4626a77"))).getN());
        print("remove age >= 24: " + users.remove(new BasicDBObject("age", new BasicDBObject("$gte", 24))).getN());
}
```

修改数据

```
@Test
public void modify() {
        print("修改：" + users.update(new BasicDBObject("_id", new ObjectId("4dde25d06be7c53ffbd70906")), new BasicDBObject("age", 99)).getN());
        print("修改：" + users.update(
                        new BasicDBObject("_id", new ObjectId("4dde2b06feb038463ff09042")),
                        new BasicDBObject("age", 121),
                        true,//如果数据库不存在，是否添加
                        false//多条修改
                        ).getN());
        print("修改：" + users.update(
                        new BasicDBObject("name", "haha"),
                        new BasicDBObject("name", "dingding"),
                        true,//如果数据库不存在，是否添加
                        true//false 只修改第一天，true 如果有多条就不修改
                        ).getN());

        //当数据库不存在就不修改、不添加数据，当多条数据就不修改
        //print("修改多条：" + coll.updateMulti(new BasicDBObject("_id", new ObjectId("4dde23616be7c19df07db42c")), new BasicDBObject("name", "199")));
```

```
}
```

查询数据

```
@Test
public void query() {
    //查询所有
    //queryAll();

    //查询 id = 4de73f7acd812d61b4626a77
    print("find id = 4de73f7acd812d61b4626a77: " + users.find(new BasicDBObject("_id", new ObjectId("4de73f7acd812d61b4626a77"))).toArray());

    //查询 age = 24
    print("find age = 24: " + users.find(new BasicDBObject("age", 24)).toArray());

    //查询 age >= 24
    print("find age >= 24: " + users.find(new BasicDBObject("age", new BasicDBObject("$gte", 24))).toArray());
    print("find age <= 24: " + users.find(new BasicDBObject("age", new BasicDBObject("$lte", 24))).toArray());

    print("查询 age!=25：" + users.find(new BasicDBObject("age", new BasicDBObject("$ne", 25))).toArray());
    print("查询 age in 25/26/27：" + users.find(new BasicDBObject("age", new BasicDBObject(QueryOperators.IN, new int[] { 25, 26, 27 }))).toArray());
    print("查询 age not in 25/26/27：" + users.find(new BasicDBObject("age", new BasicDBObject(QueryOperators.NIN, new int[] { 25, 26, 27 }))).toArray());
    print("查询 age exists 排序：" + users.find(new BasicDBObject("age", new BasicDBObject(QueryOperators.EXISTS, true))).toArray());

    print("只查询 age 属性：" + users.find(null, new BasicDBObject("age", true)).toArray());
    print("只查属性：" + users.find(null, new BasicDBObject("age", true), 0, 2).toArray());
    print("只查属性：" + users.find(null, new BasicDBObject("age", true), 0, 2, Bytes.QUERYOPTION_NOTIMEOUT).toArray());

    //只查询一条数据，多条去第一条
    print("findOne: " + users.findOne());
    print("findOne: " + users.findOne(new BasicDBObject("age", 26)));
    print("findOne: " + users.findOne(new BasicDBObject("age", 26), new BasicDBObject("name", true)));

    //查询修改、删除
```

```
    print("findAndRemove 查询 age=25 的数据，并且删除: " + users.findAndRemove(new BasicDBObject("age", 25)));

    //查询 age=26 的数据，并且修改 name 的值为 Abc
    print("findAndModify: " + users.findAndModify(new BasicDBObject("age", 26), new BasicDBObject("name", "Abc")));
    print("findAndModify: " + users.findAndModify(
        new BasicDBObject("age", 28), //查询 age=28 的数据
        new BasicDBObject("name", true), //查询 name 属性
        new BasicDBObject("age", true), //按照 age 排序
        false, //是否删除，true 表示删除
        new BasicDBObject("name", "Abc"), //修改的值，将 name 修改成 Abc
        true,
        true));

    queryAll();
}
```

mongoDB 不支持联合查询、子查询，这需要我们自己在程序中完成。将查询的结果集在 Java 查询中进行需要的过滤即可。

其他操作

```
public void testOthers() {
    DBObject user = new BasicDBObject();
    user.put("name", "hoojo");
    user.put("age", 24);

    //JSON 对象转换
    print("serialize: " + JSON.serialize(user));
    //反序列化
    print("parse: " + JSON.parse("{ \"name\" : \"hoojo\" , \"age\" : 24}"));

    print("判断 temp Collection 是否存在: " + db.collectionExists("temp"));

    //如果不存在就创建
    if (!db.collectionExists("temp")) {
        DBObject options = new BasicDBObject();
        options.put("size", 20);
        options.put("capped", 20);
        options.put("max", 20);
        print(db.createCollection("account", options));
    }
```

```
        //设置 db 为只读
        db.setReadOnly(true);

        //只读不能写入数据
        db.getCollection("test").save(user);
}
```

五、 索引

六、 集群与分片

1. 集群

MongoDB 的集群主要作用：故障切换，数据集成，读扩展，热备份，离线批处理数据源等。

主从复制

注：主从复制主要用于早起的 MongoDB 版本，现在官网也推荐使用副本集集群来替代主从复制。（所以本小节建议略过）

下面介绍搭建主从复制的步骤：

1. 准备两套 MongoDB 环境

   首先，复制两份 MongoDB 软件；然后，创建两个数据库文件的存放目录。比如说：D:\MongoDb\data 和 D:\MongoDb\data2

2. 启动主服务

   选择一个作为主服务，启动命令如下：

   mongod --dbpath D:\MongoDb\data --port 27107 --master

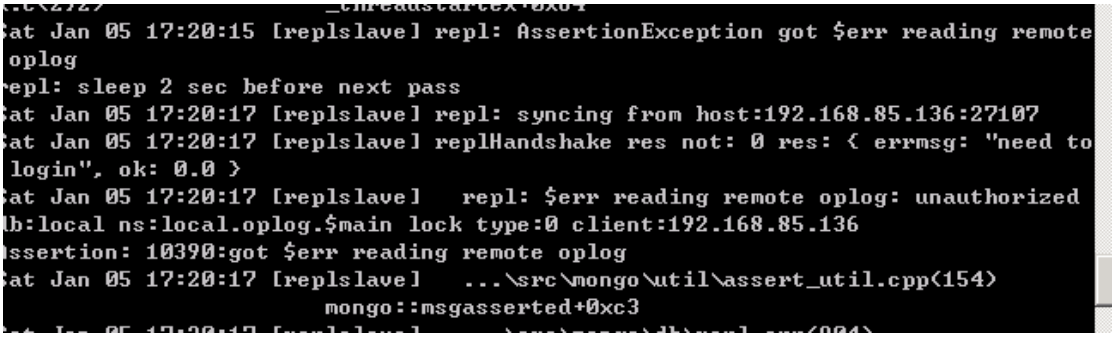
3. 启动从服务

   启动另外一个，作为从服务，启动命令如下，其中—source 指定的是刚启动的主服务的 IP 及端口。

   mongod --dbpath D:\MongoDb\data2 --port 27108 --slave --source 192.168.85.136:27107

4. 需安全认证的处理

   如果主服务启用了安全认证，并且指定了用户名密码，从服务里没有用户名和密码的话，会出现如下提示：



解决办法是，给主从服务都在 local 库中创建一个 repl 用户即可

分别在主从服务中执行语句：

use local

db.addUser("repl","repl")

然后在主从服务启动的时候都要加上--auth 参数

故障切换：

如果主服务挂了（或者因故需要切换），操作步骤

1. 关掉主服务 A
2. 关掉从服务 B
3. 移除从服务 B 数据文件目录下的以 local 开头的文件（不可恢复）
4. 启动从服务 B，以 master 的身份启动

副本集集群

搭建步骤

准备 3 个环境，至少 3 个，因为 2 个的话有一个挂了另外一个不能投票了：

1. 准备 3 个环境，包括程序及数据库文件存放目录
2. 启动 3 个数据库

   mongod --dbpath D:\MongoDb\data_rep1 --port 27107 --replSet myrepl

   mongod --dbpath D:\MongoDb\data_rep2 --port 27108 --replSet myrepl

   mongod --dbpath D:\MongoDb\data_rep3 --port 27109 --replSet myrepl
3. 配置副本集集群

   登录其中一个 Mongo 服务。如：mongo 192.168.85.136:27107

   配置如下：

```
var myrepl={
    _id:'myrepl',
    members:[
        {_id:0, host:'192.168.85.136:27107'},
        {_id:1, host:'192.168.85.136:27108'},
        {_id:2, host:'192.168.85.136:27109'}]
}

rs.initiate(myrepl);
```

4. 验证

随便关掉一个试试就行了

副本集管理

登录集群中其中一个副本后，输入 rs.help()，会显示详细的管理命令。
主要有
rs.conf() 查看配置信息
rs.status() 查看集群状态
rs.initiate() 默认初始化
rs.initiate(cfg) 按照摸个配置初始化，详见上面步骤
rs.slaveOk() shorthand for db.getMongo().setSlaveOk()
db.isMaster() check who is primary

集群功能扩展

读扩展
用从节点做数据处理

集群工作机制

参考 mongodb 权威指南

2. 分片

3. 副本集及分片

七、 备份恢复

八、 深入解析

1. BSON

BSON 简介

BSON 是一种类 json 的一种二进制形式的存储格式，简称 Binary JSON，它和 JSON 一样，支持内嵌的文档对象和数组对象，但是 BSON 有 JSON 没有的一些数据类型，如 Date 和 BinData 类型。

BSON 可以做为网络数据交换的一种存储形式，这个有点类似于 Google 的 Protocol Buffer，但是 BSON 是一种 schema-less 的存储形式，它的优点是灵活性高，但它的缺点是空间利用率不是很理想，

BSON 有三个特点：轻量性、可遍历性、高效性

{“hello":"world"} 这是一个 BSON 的例子，其中"hello"是 key name，它一般是 cstring 类型，字节表示是 cstring::= (byte*) "/x00" ,其中*表示零个或多个 byte 字节，/x00 表示结束符;后面的"world"是 value 值，它的类型一般是 string,double,array,binarydata 等类型。

MongoDB 与 BSON

MongoDB 的文档是个抽象概念,其具体的呈现形式取决与使用的驱动和编程语言.因为 MongoDB 中的通信大量依赖于文档,所以需要一种所有驱动,工具和进程都能共享的文档表达方式.这种表达方式就是 Binary JSON(BSON)

BSON 是轻量级的二进制格式,能将 MongoDB 的所有文档表示为自己字符串.数据库能理解 BSON,存在磁盘上的文档也是这种格式.当驱动要插入文档,或者将文档作为查询条件,驱动会将文档转化成 BSON,然后再发往服务器.同样,返回到客户端的文档也是 BSON 格式的字符串.驱动需要将这些数据解码,编程本机的文档表示,最后返回给客户端.

用 BSON 的三个主要原因:

a.效率

BSON 设计用来更有效的表示数据,暂用更好的空间.最差的情况下,BSON 比 JSON 效率略低,最好的情况下(比如存放二进制数据或着大多数),BSON 要比 JSON 要高效的多.

b.可遍历行

有些时候 BSON 牺牲了空间效率,换取更容易遍历的格式.如,在字符串前面加入其长度,而不是在结尾处使用一个终结符，这对 MongoDB 的内嵌文档很有用.

c.性能

BSON 编码和解码速度都很快.它用 C 风格的表现方式表示类型,在大多数编程语言中都很快.

2. Mongo 传输协议

驱动在 TCP/IP 协议的基础上简单封装了 MongoDB 传输协议,用来与 MongoDB 交互.MongoDB 的传输协议基本上是对一个简单封装的 BSON 数据,如,插入消息会有 20 字节的头部数据(包括告知服务器执行写入操作的代码,以及消息长度,要插入的集合名,要插入的 BSON 文档列表)。

3. 数据文件

MongoDB 的数据目录中,每个数据库都有几个独立的文件.每个数据有一个.ns 文件和若干数据文件,数据文件以递增的数字结尾,所以,数据库 foo 会被存放在 foo.ns,foo.0,foo.1 等文件中.

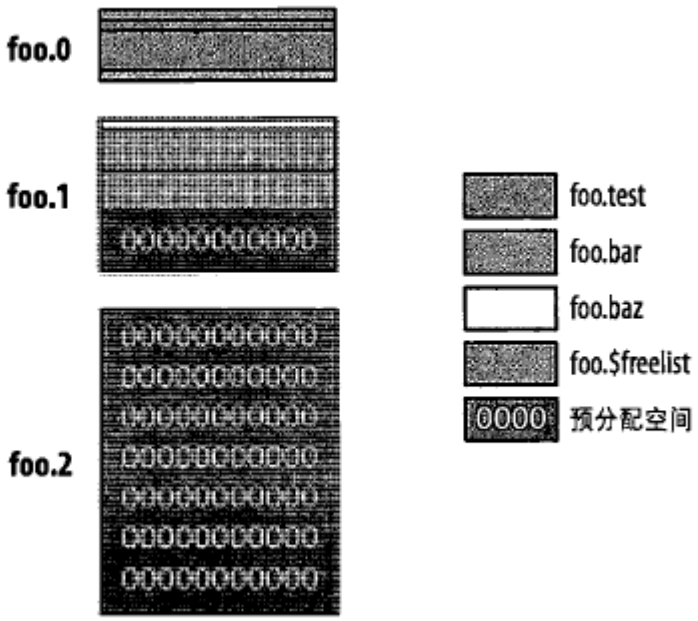
每个新的以数字结尾的数据文件大小会加倍,直到达到最大值 2GB.这是为了让小数据库不浪费太多的磁盘空间,同时让大数据库使用磁盘上连续的空间.

MongoDB 为了保证性能还会预分配数据文件(可以用--norealloc 关闭这一功能).预分配在后台完成,有数据文件被填满时就会自动启动.这意味着 MongoDB 服务器总是视图在每一个数据库保留一个额外的空数据文件来避免文件分配产生的阻塞.

4. 命名空间和数据域

在数据文件内部,每个数据库都是按照 命名空间 组织的,一种类别的数据与其它类别的分开存放.每个集合的文档都有自己的命名空间,索引也是.命名空间的元数据存放在数据库的.ns 文件中.

每个命名空间的数据都被分成若干组,放到数据文件的某一个区域内,这个区域成为数据域.如图:



数据库 foo 有 3 个数据文件,其中第 3 个是预分配的空文件.前两个数据文件被分成几个数据域,属于几个不同的命名空间

每个命名空间可以有几个不同的数据域,在磁盘上不必连续.类似于数据库的数据文件,每次新分配的命名空间的数据域大小也会增加.这是为了平衡命名空间浪费的控件和尽量让一个命名空间的数据连续做出的这种.图中还有个特殊的命名空间$freelist,存放这不再使用的数据域(如删除集合或索引产生的数据域).当命名空间分配新的数据域时,系统会先查找空闲列表,看看有没有适合大小的数据域可用

5. 内存映射存储引擎

MongoDB 默认的存储引擎是内存映射引擎,当服务器启动后,将所有数据文件映射到内存,然后由操作系统来负责将缓冲数据写入磁盘并将数据调入调出内存页面

这样的移情有若干重要的特性:

a) MongoDB 管理内存的代码非常精炼,原因是将大部分工作交给了操作系统.

b) MongoDB 服务器进程的虚拟大小通常会非常大,超过整个数据集的大小,这没关系,因为操作系统会处理让那些数据常驻内存.

c) MongoDB 不能控制数据写入到磁盘的顺序,也就不能用预写日志提供单机的持久性.

d) 32 位的 MongoDB 服务器有个限制,每个 mongod 最多处理 2GB 数据,这是因为所有数据必须能用 32 位地址访问到.