MongoDB

1.MongoDB简介

MongoDB 是由C++语言编写的，是一个基于分布式文件存储的开源数据库系统。

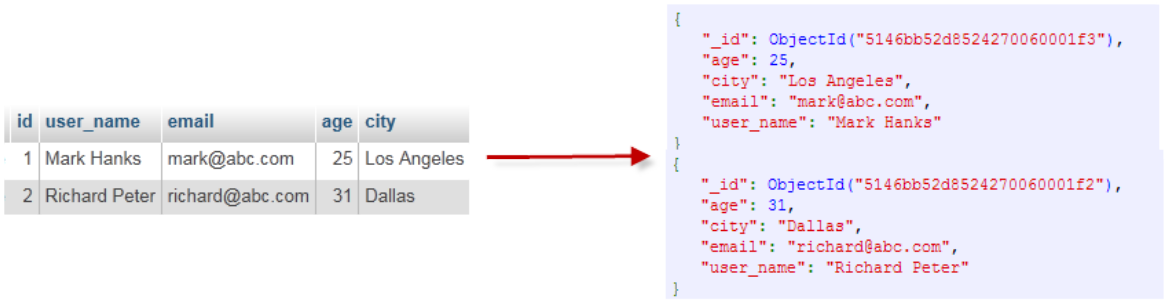
在高负载的情况下，添加更多的节点，可以保证服务器性能。

MongoDB 旨在为WEB应用提供可扩展的高性能数据存储解决方案。

MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。

2.MongoDB中的概念

SQL术语	MongoDB术语/概念	解释/说明
database	database	数据库
table	collection	表/集合
row	document	行/文档
column	field	列/域
index	index	索引
primary key	primary key	MongoDB不支持



3.创建和删除数据库

在命令行输入mongo，进入mongodb shell

- 通过`use 数据库名`创建数据库

```
> use runoob
switched to db runoob
> db
runoob
```

- 通过`show dbs`查看所有数据库

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

- 使用`db.dropDatabase()`删除数据库

```
# 切换到要删除的数据库
> use runoob
switched to db runoob
> db.dropDatabase()
{ "dropped" : "runoob", "ok" : 1 }
```

4.插入和删除集合

- 插入集合

```
db.createCollection(name, options)
```

参数说明：

- name: 要创建的集合名称
- options: 可选参数, 指定有关内存大小及索引的选项

options 可以是如下参数：

字段	类型	描述
capped	布尔	（可选）如果为 true，则创建固定集合。固定集合是指有着固定大小的集合，当达到最大值时，它会自动覆盖最早的文档。 **当该值为 true 时，必须指定 size 参数。**
autoIndexId	布尔	3.2 之后不再支持该参数。（可选）如为 true，自动在 _id 字段创建索引。默认为 false。
size	数值	（可选）为固定集合指定一个最大值，即字节数。 **如果 capped 为 true，也需要指定该字段。**
max	数值	（可选）指定固定集合中包含文档的最大数量。

在插入文档时，MongoDB 首先检查固定集合的 size 字段，然后检查 max 字段。

```
> use test
switched to db test
> db.createCollection("runoob")
{ "ok" : 1 }
> show collections
runoob
system.indexes
# 带参数的方法
> db.createCollection("mycol", { capped : true, autoIndexId : true, size :
   6142800, max : 10000 } )
{ "ok" : 1 }
# 在MongoDB中，插入文档时会自动创建集合，不需要先创建
> db.mycol2.insert({"name" : "菜鸟教程"})
> show collections
mycol2
```

- 删除集合

```
db.collection.drop()
```

```
>db.mycol2.drop()
true
```

5.插入和删除文档

假设一个文档的结构如下：

```
{
    "name":"zhangsan",
    "info":{
        "id":1
        "sex":"male"
    }
}
```

- 插入文档

```
db.COLLECTION_NAME.insert(document)
或
db.COLLECTION_NAME.save(document)
```

- save()：如果 _id 主键存在则更新数据，如果不存在就插入数据。该方法新版本中已废弃，可以使用 **db.collection.insertOne()** 或 **db.collection.replaceOne()** 来代替。
- insert(): 若插入的数据主键已经存在，则会抛 **org.springframework.dao.DuplicateKeyException** 异常，提示主键重复，不保存当前数据。

db.collection.insertOne() 用于向集合插入一个新文档，语法格式如下：

```
db.collection.insertOne(
   <document>,
   {
      writeConcern: <document>
   }
)
```

db.collection.insertMany() 用于向集合插入一个多个文档，语法格式如下：

```
db.collection.insertMany(
   [ <document 1> , <document 2>, ... ],
   {
      writeConcern: <document>,
      ordered: <boolean>
   }
)
```

参数说明：

- document：要写入的文档。
- writeConcern：写入策略，默认为 1，即要求确认写操作，0 是不要求。
- ordered：指定是否按顺序写入，默认 true，按顺序写入。

可以使用数组定义多个文档，一次性插入

```
var people = [
    {"name":"zhangsan", "info":{"id":1,"sex":"male"}},
    {"name":"lisi", "info":{"id":2,"sex":"male"}}
]
db.peo.insert(people)
```

- 删除文档

```
db.collection.remove(
   <query>,
   {
     justOne: <boolean>,
     writeConcern: <document>
   }
)
```

参数说明：

- **query** ：（可选）删除的文档的条件。
- **justOne** ：（可选）如果设为 true 或 1，则只删除一个文档，如果不设置该参数，或使用默认值 false，则删除所有匹配条件的文档。
- **writeConcern** ：（可选）抛出异常的级别。

```
>db.col.remove({'title':'MongoDB 教程'})
WriteResult({ "nRemoved" : 2 })           # 删除了两条数据
>db.col.find()
……                                        # 没有数据
```

6.更新和查询文档

- update() 方法用于更新已存在的文档。语法格式如下：

```
db.collection.update(
   <query>,
   <update>,
   {
     upsert: <boolean>,
     multi: <boolean>,
     writeConcern: <document>
   }
)
```

参数说明：

- **query** : update的查询条件，类似sql update查询内where后面的。
- **update** : update的对象和一些更新的操作符（如,inc...）等，也可以理解为sql update查询内set后面的
- **upsert** : 可选，这个参数的意思是，如果不存在update的记录，是否插入objNew,true为插入，默认是false，不插入。
- **multi** : 可选，mongodb 默认是false,只更新找到的第一条记录，如果这个参数为true,就把按条件查出来多条记录全部更新。
- **writeConcern** :可选，抛出异常的级别。

```
>db.col.update({'title':'MongoDB 教程'},{$set:{'title':'MongoDB'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })   # 输出信息
```

save() 方法通过传入的文档来替换已有文档，_id 主键存在就更新，不存在就插入。语法格式如下：

```
db.collection.save(
   <document>,
   {
     writeConcern: <document>
   }
)
```

参数说明：

- **document** : 文档数据。
- **writeConcern** :可选，抛出异常的级别。

```
>db.col.save({
    "_id" : ObjectId("56064f89ade2f21f36b03136"),
    "title" : "MongoDB",
    "description" : "MongoDB 是一个 Nosql 数据库",
    "by" : "Runoob",
    "url" : "http://www.runoob.com",
    "tags" : [
            "mongodb",
            "NoSQL"
    ],
    "likes" : 110
})
```

MongoDB 查询数据的语法格式如下：

```
db.collection.find(query, projection)
```

- **query** ： 可选，使用查询操作符指定查询条件
- **projection** ： 可选，使用投影操作符指定返回的键。查询时返回文档中所有键值， 只需省略该参数即可（默认省略）。

如果你需要以易读的方式来读取数据，可以使用 pretty() 方法，语法格式如下：

```
>db.col.find().pretty()
```

pretty() 方法以格式化的方式来显示所有文档。

```
> db.col.find().pretty()
{
        "_id" : ObjectId("56063f17ade2f21f36b03133"),
        "title" : "MongoDB 教程",
        "description" : "MongoDB 是一个 Nosql 数据库",
        "by" : "菜鸟教程",
        "url" : "http://www.runoob.com",
        "tags" : [
                "mongodb",
                "database",
                "NoSQL"
        ],
        "likes" : 100
}
```

7.远程连接MongoDB设置

由于MongoDB默认不支持远程连接，需要修改配置文件并关闭虚拟机防火墙。

1. 修改配置文件

   ```
   vim /etc/mongodb.conf
   ```

   将band_ip = 127.0.0.1 改为 band_ip=0.0.0.0

2. 关闭防火墙

   ```
   ufw disable
   ```

3. 重启MongoDB

   ```
   /etc/init.d/mongodb restart
   ```

4. 在maven中添加依赖

   ```
   <!-- mongodb Dirver -->
   <dependency>
       <groupId>org.mongodb</groupId>
       <artifactId>mongodb-driver-legacy</artifactId>
       <version>4.1.1</version>
   </dependency>
   ```

8.JAVA API连接MongoDB

- 创建连接

  连接数据库，你需要指定数据库名称，如果指定的数据库不存在，mongo会自动创建数据库。

  连接数据库的Java代码如下：

```
import com.mongodb.MongoClient;
import com.mongodb.client.MongoDatabase;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
       // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "192.168.6.130" , 27017
);

         // 连接到数据库
         MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");
       System.out.println("Connect to database successfully");

      }catch(Exception e){
        System.err.println( e.getClass().getName() + ": " + e.getMessage()
);
     }
   }
}
```

- 创建集合

  我们可以使用 com.mongodb.client.MongoDatabase 类中的createCollection()来创建集合

  代码片段如下：

```
import com.mongodb.MongoClient;
import com.mongodb.client.MongoDatabase;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
      // 连接到 mongodb 服务
      MongoClient mongoClient = new MongoClient( "192.168.6.130" , 27017 );
      // 连接到数据库
      MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");
      System.out.println("Connect to database successfully");
      mongoDatabase.createCollection("test");
      System.out.println("集合创建成功");

      }catch(Exception e){
        System.err.println( e.getClass().getName() + ": " + e.getMessage()
);
     }
   }
}
```

- 获取集合

我们可以使用com.mongodb.client.MongoDatabase类的 getCollection() 方法来获取一个集合

代码片段如下：

```
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
       // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "192.168.6.130" , 27017
);

         // 连接到数据库
         MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");
       System.out.println("Connect to database successfully");

       MongoCollection<Document> collection =
mongoDatabase.getCollection("test");
       System.out.println("集合 test 选择成功");
      }catch(Exception e){
        System.err.println( e.getClass().getName() + ": " + e.getMessage()
);
     }
   }
}
```

- 插入文档

  我们可以使用com.mongodb.client.MongoCollection类的 insertMany() 方法来插入一个文档

  代码片段如下：

```
import java.util.ArrayList;
import java.util.List;
import org.bson.Document;

import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
         // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "192.168.6.130" , 27017
);

         // 连接到数据库
         MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");
         System.out.println("Connect to database successfully");

         MongoCollection<Document> collection =
mongoDatabase.getCollection("test");
         System.out.println("集合 test 选择成功");
```

```
         //插入文档
         /**
         * 1. 创建文档 org.bson.Document 参数为key-value的格式
         * 2. 创建文档集合List<Document>
         * 3. 将文档集合插入数据库集合中
mongoCollection.insertMany(List<Document>) 插入单个文档可以用
mongoCollection.insertOne(Document)
         * */
         Document document = new Document("title", "MongoDB").
         append("description", "database").
         append("likes", 100).
         append("by", "Fly");
         List<Document> documents = new ArrayList<Document>();
         documents.add(document);
         collection.insertMany(documents);
         System.out.println("文档插入成功");
      }catch(Exception e){
         System.err.println( e.getClass().getName() + ": " + e.getMessage()
);
      }
   }
}
```

- 检索文档

  我们可以使用 com.mongodb.client.MongoCollection 类中的 find() 方法来获取集合中的所有文档。

  此方法返回一个游标，所以你需要遍历这个游标。

  代码片段如下:

```
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
         // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "192.168.6.130" , 27017
);

         // 连接到数据库
         MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");
         System.out.println("Connect to database successfully");

         MongoCollection<Document> collection =
mongoDatabase.getCollection("test");
         System.out.println("集合 test 选择成功");

         //检索所有文档
         /**
         * 1. 获取迭代器FindIterable<Document>
         * 2. 获取游标MongoCursor<Document>
```

```
         * 3. 通过游标遍历检索出的文档集合
         * */
         FindIterable<Document> findIterable = collection.find();
         MongoCursor<Document> mongoCursor = findIterable.iterator();
         while(mongoCursor.hasNext()){
            System.out.println(mongoCursor.next());
         }

      }catch(Exception e){
         System.err.println( e.getClass().getName() + ": " + e.getMessage()
);
      }
   }
}
```

- 更新文档

  你可以使用 com.mongodb.client.MongoCollection 类中的 updateMany() 方法来更新集合中的文档。

  代码片段如下:

```
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
         // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "192.168.6.130" , 27017
);

         // 连接到数据库
         MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");
         System.out.println("Connect to database successfully");

         MongoCollection<Document> collection =
mongoDatabase.getCollection("test");
         System.out.println("集合 test 选择成功");

         //更新文档   将文档中likes=100的文档修改为likes=200
         collection.updateMany(Filters.eq("likes", 100), new
Document("$set",new Document("likes",200)));
         //检索查看结果
         FindIterable<Document> findIterable = collection.find();
         MongoCursor<Document> mongoCursor = findIterable.iterator();
         while(mongoCursor.hasNext()){
            System.out.println(mongoCursor.next());
         }

      }catch(Exception e){
         System.err.println( e.getClass().getName() + ": " + e.getMessage()
);
```

```
      }
   }
}
```

- 删除第一个文档

  要删除集合中的第一个文档，首先你需要使用com.mongodb.DBCollection类中的 findOne()方法来获取第一个文档，然后使用remove 方法删除。

  代码片段如下：

```
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;

public class MongoDBJDBC{
   public static void main( String args[] ){
      try{
         // 连接到 mongodb 服务
         MongoClient mongoClient = new MongoClient( "192.168.6.130" , 27017
);

         // 连接到数据库
         MongoDatabase mongoDatabase = mongoClient.getDatabase("mycol");
         System.out.println("Connect to database successfully");

         MongoCollection<Document> collection =
mongoDatabase.getCollection("test");
         System.out.println("集合 test 选择成功");

         //删除符合条件的第一个文档
         collection.deleteOne(Filters.eq("likes", 200));
         //删除所有符合条件的文档
         collection.deleteMany (Filters.eq("likes", 200));
         //检索查看结果
         FindIterable<Document> findIterable = collection.find();
         MongoCursor<Document> mongoCursor = findIterable.iterator();
         while(mongoCursor.hasNext()){
           System.out.println(mongoCursor.next());
         }

      }catch(Exception e){
        System.err.println( e.getClass().getName() + ": " + e.getMessage()
);
     }
   }
}
```