

## 课程介绍

- MongoDB入门
- MongoDB的java api的使用
- SpringBoot整合MongoDB使用
- 搭建微聊系统
- 实现微聊功能
- 分布式WebSocket解决方案分析

## 1、MongoDB入门

### 1.1、MongoDB简介

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

MongoDB是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的，它支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。

MongoDB最大的特点是它支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

官网：<https://www.mongodb.com>

### 1.2、通过docker安装MongoDB

```
1  #拉取镜像
2  docker pull mongo:4.0.3
3
4  #创建容器
5  docker create --name mongodb -p 27017:27017 -v /data/mongodb:/data/db mongo:4.0.3
6
7  #启动容器
8  docker start mongodb
9
10 #进入容器
11 docker exec -it mongodb /bin/bash
12
13 #使用MongoDB客户端进行操作
14 mongo
15 > show dbs  #查询所有的数据库
16 admin    0.000GB
17 config   0.000GB
18 local    0.000GB
```

### 1.3、MongoDB基本操作

#### 1.3.1、基本概念

为了更好的理解，下面与SQL中的概念进行对比：

SQL术语/概念	MongoDB术语/概念	解释/说明
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
column	field	数据字段/域
index	index	索引
table joins		表连接,MongoDB不支持
primary key	primary key	主键,MongoDB自动将_id字段设置为主键

id	user_name	email	age	city
1	Mark Hanks	mark@abc.com	25	Los Angeles
2	Richard Peter	richard@abc.com	31	Dallas



```
{
  "_id": ObjectId("5146bb52d8524270060001f3"),
  "age": 25,
  "city": "Los Angeles",
  "email": "mark@abc.com",
  "user_name": "Mark Hanks"
}
{
  "_id": ObjectId("5146bb52d8524270060001f2"),
  "age": 31,
  "city": "Dallas",
  "email": "richard@abc.com",
  "user_name": "Richard Peter"
}
```

### 1.3.2、数据库以及表的操作

```
1  #查看所有的数据库
2  > show dbs
3  admin   0.000GB
4  config  0.000GB
5  local   0.000GB
6
7  #通过use关键字切换数据库
8  > use admin
9  switched to db admin
10
11 #创建数据库
12 #说明：在MongoDB中，数据库是自动创建的，通过use切换到新数据库中，进行插入数据即可自动创建数据库
13 > use testdb
14 switched to db testdb
15 > show dbs #并没有创建数据库
16 admin   0.000GB
17 config  0.000GB
18 local   0.000GB
19 > db.user.insert({id:1,name:'zhangsang'}) #插入数据
20 writeResult({ "nInserted" : 1 })
21 > show dbs
```



```
22 admin 0.000GB
23 config 0.000GB
24 local 0.000GB
25 testdb 0.000GB #数据库自动创建
26
27 #查看表
28 > show tables
29 user
30 > show collections
31 user
32 >
33
34 #删除集合(表)
35 > db.user.drop()
36 true #如果成功删除选定集合,则 drop() 方法返回 true, 否则返回 false。
37
38 #删除数据库
39 > use testdb #先切换到要删除的数据中
40 switched to db testdb
41 > db.dropDatabase() #删除数据库
42 { "dropped" : "testdb", "ok" : 1 }
43 > show dbs
44 admin 0.000GB
45 config 0.000GB
46 local 0.000GB
```

### 1.3.3、新增数据

在MongoDB中,存储的文档结构是一种类似于json的结构,称之为bson(全称为: Binary JSON)。

```
1 #插入数据
2
3 #语法: db.COLLECTION_NAME.insert(document)
4 > db.user.insert({id:1,username:'zhangsan',age:20})
5 WriteResult({ "nInserted" : 1 })
6 > db.user.save({id:2,username:'lisi',age:25})
7 WriteResult({ "nInserted" : 1 })
8 > db.user.find() #查询数据
9 { "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "id" : 1, "username" : "zhangsan",
  "age" : 20 }
10 { "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi",
  "age" : 25 }
11
```

### 1.3.4、更新数据

update() 方法用于更新已存在的文档。语法格式如下:



```
1 db.collection.update(  
2     <query>,  
3     <update>,  
4     [  
5         upsert: <boolean>,  
6         multi: <boolean>,  
7         writeConcern: <document>  
8     ]  
9 )
```

#### 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。

```
1 > db.user.find()  
2 { "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "id" : 1, "username" : "zhangsan",  
  "age" : 20 }  
3 { "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi",  
  "age" : 25 }  
4  
5 > db.user.update({id:1},{set:{age:22}}) #更新数据  
6  
7 WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
8  
9 > db.user.find()  
10 { "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "id" : 1, "username" : "zhangsan",  
   "age" : 22 }  
11 { "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi",  
   "age" : 25 }  
12  
13 #注意：如果这样写，会删除掉其他的字段  
14 > db.user.update({id:1},{age:25})  
15 WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })  
16 > db.user.find()  
17 { "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "age" : 25 }  
18 { "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi",  
   "age" : 25 }  
19  
20 #更新不存在的字段，会新增字段  
21 > db.user.update({id:2},{set:{sex:1}}) #更新数据  
22 > db.user.find()  
23 { "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "age" : 25 }  
24 { "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi",  
   "age" : 25, "sex" : 1 }  
25  
26 #更新不存在的数据，默认不会新增数据
```



```
27 > db.user.update({id:3},{set:{sex:1}})
28 writeResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
29 > db.user.find()
30 { "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "age" : 25 }
31 { "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi",
  "age" : 25, "sex" : 1 }
32
33 #如果设置第一个参数为true，就是新增数据
34 > db.user.update({id:3},{set:{sex:1}},true)
35 writeResult({
36     "nMatched" : 0,
37     "nUpserted" : 1,
38     "nModified" : 0,
39     "_id" : ObjectId("5c08cb281418d073246bc642")
40 })
41 > db.user.find()
42 { "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "age" : 25 }
43 { "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi",
  "age" : 25, "sex" : 1 }
44 { "_id" : ObjectId("5c08cb281418d073246bc642"), "id" : 3, "sex" : 1 }
45
```

### 1.3.5、删除数据

通过remove()方法进行删除数据，语法如下：

```
1 db.collection.remove(
2     <query>,
3     {
4         justOne: <boolean>,
5         writeConcern: <document>
6     }
7 )
```

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。

实例：

```
1 > db.user.remove({age:25})
2 writeResult({ "nRemoved" : 2 }) #删除了2条数据
3
4 #插入4条测试数据
5 db.user.insert({id:1,username:'zhangsan',age:20})
6 db.user.insert({id:2,username:'lisi',age:21})
7 db.user.insert({id:3,username:'wangwu',age:22})
8 db.user.insert({id:4,username:'zhao Liu',age:22})
9
```

```

10 > db.user.remove({age:22},true)
11 writeResult({ "nRemoved" : 1 }) #删除了1条数据
12
13 #删除所有数据
14 > db.user.remove({})
15
16 #说明：为了简化操作，官方推荐使用deleteOne()与deleteMany()进行删除数据操作。
17 db.user.deleteOne({id:1})
18 db.user.deleteMany({}) #删除所有数据
19
    
```

### 1.3.6、查询数据

MongoDB 查询数据的语法格式如下：

```
1 db.user.find([query],[fields])
```

- **query**：可选，使用查询操作符指定查询条件
- **fields**：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：

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

pretty() 方法以格式化的方式来显示所有文档。

条件查询：

操作	格式	范例	RDBMS中的类似语句
等于	{<key>:<value> }	db.col.find({"by":"黑马程序员"}).pretty()	where by = '黑马程序员'
小于	{<key>:{ \$lt:<value>}}	db.col.find({"likes":{\$lt:50}}).pretty()	where likes < 50
小于或等于	{<key>:{ \$lte:<value>}}	db.col.find({"likes":{\$lte:50}}).pretty()	where likes <= 50
大于	{<key>:{ \$gt:<value>}}	db.col.find({"likes":{\$gt:50}}).pretty()	where likes > 50
大于或等于	{<key>:{ \$gte:<value>}}	db.col.find({"likes":{\$gte:50}}).pretty()	where likes >= 50
不等于	{<key>:{ \$ne:<value>}}	db.col.find({"likes":{\$ne:50}}).pretty()	where likes != 50



实例：

```
1  #插入测试数据
2  db.user.insert({id:1,username:'zhangsan',age:20})
3  db.user.insert({id:2,username:'lisi',age:21})
4  db.user.insert({id:3,username:'wangwu',age:22})
5  db.user.insert({id:4,username:'zhao1iu',age:22})
6
7  db.user.find() #查询全部数据
8  db.user.find({}, {id:1,username:1}) #只查询id与username字段
9  db.user.find().count() #查询数据条数
10 db.user.find({id:1}) #查询id为1的数据
11 db.user.find({age:{$lte:21}}) #查询小于等于21的数据
12 db.user.find({age:{$lte:21}, id:{$gte:2}}) #and查询, age小于等于21并且id大于等于2
13 db.user.find({$or:[{id:1},{id:2}]}) #查询id=1 or id=2
14
15 #分页查询: skip()跳过几条, limit()查询条数
16 db.user.find().limit(2).skip(1) #跳过1条数据, 查询2条数据
17
18 db.user.find().sort({id:-1}) #按照age倒序排序, -1为倒序, 1为正序
```

## 1.4、索引

索引通常能够极大的提高查询的效率，如果没有索引，MongoDB在读取数据时必须扫描集合中的每个文件并选取那些符合查询条件的记录。

这种扫描全集合的查询效率是非常低的，特别在处理大量的数据时，查询可能要花费几十秒甚至几分钟，这对网站的性能是非常致命的。

索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，索引是对数据库表中一列或多列的值进行排序的一种结构

```
1  #查看索引
2  > db.user.getIndexes()
3  [
4    {
5      "v" : 2,
6      "key" : {
7        "_id" : 1
8      },
9      "name" : "_id_",
10     "ns" : "testdb.user"
11   }
12 ]
13
14 #说明：1表示升序创建索引，-1表示降序创建索引。
```



```
1 #创建索引
2 > db.user.createIndex({'age':1})
3 {
4   "createdCollectionAutomatically" : false,
5   "numIndexesBefore" : 1,
6   "numIndexesAfter" : 2,
7   "ok" : 1
8 }
```

```
1 #删除索引
2 db.user.dropIndex("age_1")
3 #或者，删除除了_id之外的索引
4 db.user.dropIndexes()
```

```
1 #创建联合索引
2 db.user.createIndex({'age':1, 'id':-1})
```

```
1 #查看索引大小，单位：字节
2 db.user.totalIndexSize()
```

## 1.5、执行计划

MongoDB 查询分析可以确保我们建议的索引是否有效，是查询语句性能分析的重要工具。

```
1 #插入1000条数据
2 for(var i=1;i<1000;i++)db.user.insert({id:100+i,username:'name_'+i,age:10+i})
```

```
1 #查看执行计划
2 > db.user.find({age:{$gt:100},id:{$lt:200}}).explain()
3 {
4   "queryPlanner" : {
5     "plannerVersion" : 1,
6     "namespace" : "testdb.user",
7     "indexFilterSet" : false,
8     "parsedQuery" : {
9       "$and" : [
10        {
11          "id" : {
12            "$lt" : 200
13          }
14        },
15        {
16          "age" : {
17            "$gt" : 100
18          }
19        }
20      ]
21    }
22  }
```





```
21     },
22     "winningPlan" : { #最佳执行计划
23         "stage" : "FETCH", #查询方式，常见的有COLLSCAN/全表扫描、IXSCAN/索引扫描、
        FETCH/根据索引去检索文档、SHARD_MERGE/合并分片结果、IDHACK/针对_id进行查询
24         "inputStage" : {
25             "stage" : "IXSCAN",
26             "keyPattern" : {
27                 "age" : 1,
28                 "id" : -1
29             },
30             "indexName" : "age_1_id_-1",
31             "isMultikey" : false,
32             "multikeyPaths" : {
33                 "age" : [ ],
34                 "id" : [ ]
35             },
36             "isUnique" : false,
37             "isSparse" : false,
38             "isPartial" : false,
39             "indexVersion" : 2,
40             "direction" : "forward",
41             "indexBounds" : {
42                 "age" : [
43                     "(100.0, inf.0)"
44                 ],
45                 "id" : [
46                     "(200.0, -inf.0)"
47                 ]
48             }
49         },
50     },
51     "rejectedPlans" : [ ]
52 },
53 "serverInfo" : {
54     "host" : "c493d5ff750a",
55     "port" : 27017,
56     "version" : "4.0.3",
57     "gitVersion" : "7ea530946fa7880364d88c8d8b6026bbc9ffa48c"
58 },
59 "ok" : 1
60 }
61
```

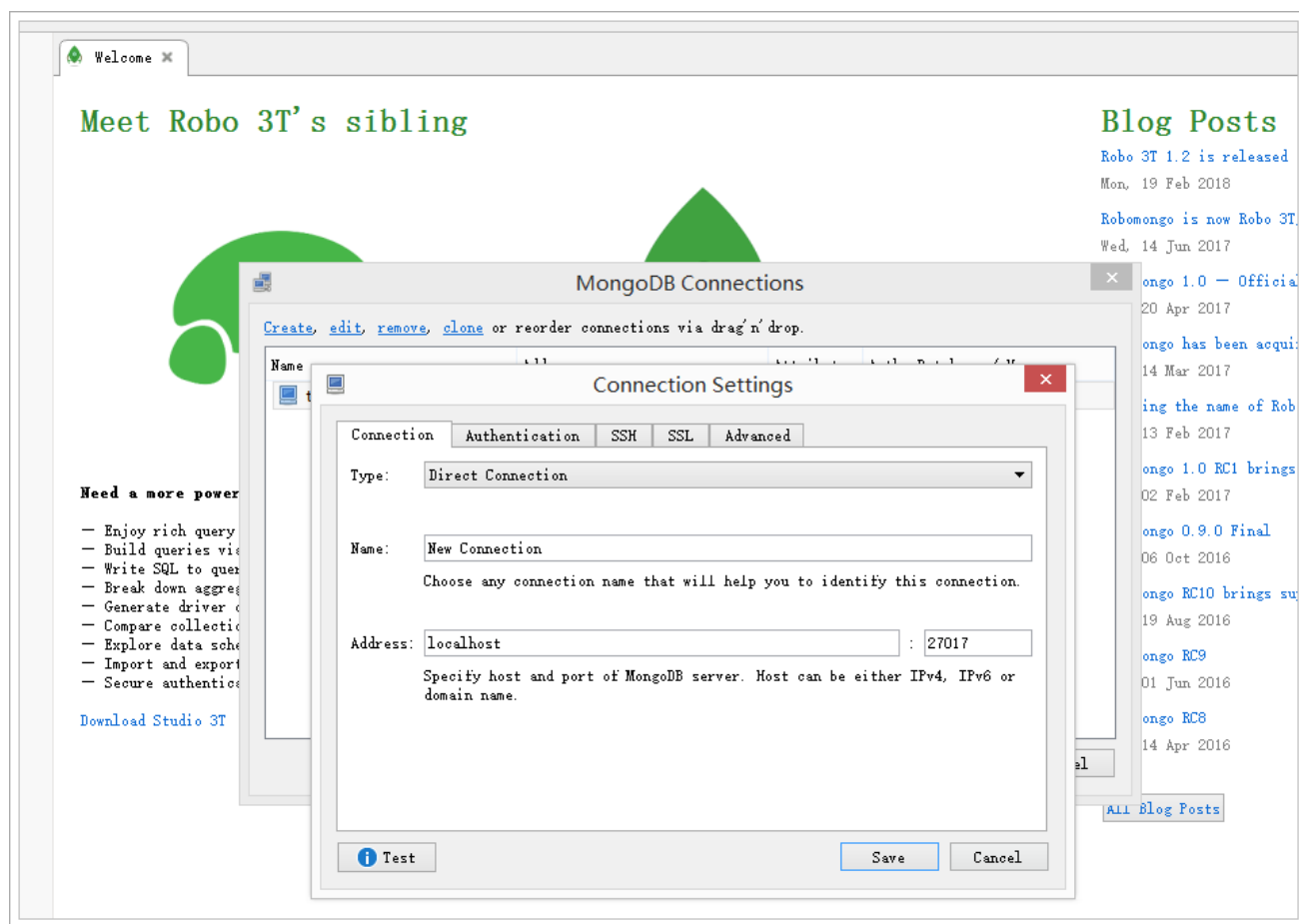
```
1 #测试没有使用索引
2 > db.user.find({username: 'zhangsan'}).explain()
3 {
4     "queryPlanner" : {
5         "plannerVersion" : 1,
6         "namespace" : "testdb.user",
7         "indexFilterSet" : false,
8         "parsedQuery" : {
9             "username" : {
10                 "$eq" : "zhangsan"

```

```
11     }
12     },
13     "winningPlan" : {
14         "stage" : "COLLSCAN", #全表扫描
15         "filter" : {
16             "username" : {
17                 "$eq" : "zhangsan"
18             }
19         },
20         "direction" : "forward"
21     },
22     "rejectedPlans" : [ ]
23 },
24 "serverInfo" : {
25     "host" : "c493d5ff750a",
26     "port" : 27017,
27     "version" : "4.0.3",
28     "gitVersion" : "7ea530946fa7880364d88c8d8b6026bbc9ffa48c"
29 },
30 "ok" : 1
31 }
```

## 1.6、UI客户端工具

Robo 3T是MongoDB的客户端工具，我们可以使用它来操作MongoDB。



查看数据：



The screenshot shows the MongoDB Compass application. On the left, a tree view shows the database structure: test1 (4) > System, config, testdb > Collections (1) > user. The main window displays the 'user' collection with 13 documents. The documents are listed in a table with columns: Key, Value, and Type. The documents contain fields like \_id, id, username, and age.

Key	Value	Type
(1) ObjectId("5c092b2465443f62d9ddd08e")	{ 4 fields }	Object
_id	ObjectId("5c092b2465443f62d9ddd08e")	ObjectId
id	1.0	Double
username	zhangsan	String
age	20.0	Double
(2) ObjectId("5c092b2465443f62d9ddd08f")	{ 4 fields }	Object
_id	ObjectId("5c092b2465443f62d9ddd08f")	ObjectId
id	2.0	Double
username	lisi	String
age	21.0	Double
(3) ObjectId("5c092b2465443f62d9ddd090")	{ 4 fields }	Object
(4) ObjectId("5c092b2565443f62d9ddd091")	{ 4 fields }	Object
(5) ObjectId("5c092b6e65443f62d9ddd092")	{ 4 fields }	Object
(6) ObjectId("5c092b6e65443f62d9ddd093")	{ 4 fields }	Object
(7) ObjectId("5c092b6e65443f62d9ddd094")	{ 4 fields }	Object
(8) ObjectId("5c092b6e65443f62d9ddd095")	{ 4 fields }	Object
(9) ObjectId("5c092b6e65443f62d9ddd096")	{ 4 fields }	Object
(10) ObjectId("5c092b6e65443f62d9ddd097")	{ 4 fields }	Object
(11) ObjectId("5c092b6e65443f62d9ddd098")	{ 4 fields }	Object
(12) ObjectId("5c092b6e65443f62d9ddd099")	{ 4 fields }	Object
(13) ObjectId("5c092b6e65443f62d9ddd09a")	{ 4 fields }	Object

## 2、通过JavaApi操作MongoDB

### 2.1、创建itcast-mongodb工程

pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6
7     <modelVersion>4.0.0</modelVersion>
8
9     <groupId>cn.itcast.mongodb</groupId>
10    <artifactId>itcast-mongodb</artifactId>
11    <version>1.0-SNAPSHOT</version>
12
13    <dependencies>
14        <dependency>
15            <groupId>org.mongodb</groupId>
16            <artifactId>mongodb-driver-sync</artifactId>
17            <version>3.9.1</version>
18        </dependency>
19        <dependency>
20            <groupId>junit</groupId>
21            <artifactId>junit</artifactId>
22            <version>4.12</version>
23            <scope>test</scope>
24        </dependency>
25        <dependency>
26            <groupId>org.projectlombok</groupId>
```



```
25         <artifactId>lombok</artifactId>
26         <version>1.18.4</version>
27     </dependency>
28 </dependencies>
29
30 <build>
31     <plugins>
32         <!-- java编译插件 -->
33         <plugin>
34             <groupId>org.apache.maven.plugins</groupId>
35             <artifactId>maven-compiler-plugin</artifactId>
36             <version>3.2</version>
37             <configuration>
38                 <source>1.8</source>
39                 <target>1.8</target>
40                 <encoding>UTF-8</encoding>
41             </configuration>
42         </plugin>
43     </plugins>
44 </build>
45
46 </project>
```

## 2.2、编写Demo

该demo中演示了，如何连接到MongoDB，如何选择数据库、表，进行查询的操作。

```
1 package cn.itcast.mongodb;
2
3 import com.mongodb.client.*;
4 import org.bson.Document;
5
6 import java.util.function.Consumer;
7
8 public class MongoDBDemo {
9
10     public static void main(String[] args) {
11         // 建立连接
12         MongoClient mongoClient =
13             MongoClient.create("mongodb://172.16.55.185:27017");
14
15         // 选择数据库
16         MongoDBDatabase mongoDatabase = mongoClient.getDatabase("testdb");
17
18         // 选择表
19         MongoCollection<Document> userCollection =
20             mongoDatabase.getCollection("user");
21
22         // 查询数据
23         userCollection.find().limit(10).forEach((Consumer<? super Document>)
24             document -> {
25                 System.out.println(document.toJson());
26             });
27     }
28 }
```



```
25
26      // 查询数据
27      //      userCollection.find().limit(10).forEach(new Consumer<Document>() {
28      //          @Override
29      //          public void accept(Document document) {
30      //              System.out.println(document.toJson());
31      //          }
32      //      });
33
34      // 关闭连接
35      mongoClient.close();
36
37  }
38
39  }
40
```

## 2.3、CURD操作

```
1  package cn.itcast.mongodb;
2
3  import com.mongodb.client.MongoClient;
4  import com.mongodb.client.MongoClients;
5  import com.mongodb.client.MongoCollection;
6  import com.mongodb.client.MongoDatabase;
7  import com.mongodb.client.model.Projections;
8  import com.mongodb.client.model.Sorts;
9  import com.mongodb.client.model.Updates;
10 import com.mongodb.client.result.DeleteResult;
11 import com.mongodb.client.result.UpdateResult;
12 import org.bson.Document;
13 import org.junit.Before;
14 import org.junit.Test;
15
16 import java.util.function.Consumer;
17
18 import static com.mongodb.client.model.Filters.*;
19
20 public class TestCRUD {
21
22     private MongoCollection<Document> mongoCollection;
23
24     @Before
25     public void init() {
26         // 建立连接
27         MongoClient mongoClient =
28             MongoClients.create("mongodb://172.16.55.185:27017");
29
30         // 选择数据库
31         MongoDatabase mongoDatabase = mongoClient.getDatabase("testdb");
32
33         // 选择表
```



```
34     this.mongoCollection = mongoDatabase.getCollection("user");
35 }
36
37 // 查询age<=50并且id>=100的用户信息，并且按照id倒序排序，只返回id，age字段，不返回_id字段
38 @Test
39 public void testQuery() {
40     this.mongoCollection.find(
41         and(
42             lte("age", 50),
43             gte("id", 100)
44         )
45     )
46     .sort(Sorts.descending("id"))
47     .projection(
48         Projections.fields(
49             Projections.include("id", "age"),
50             Projections.excludeId()
51         )
52     )
53     .forEach((Consumer<? super Document>) document -> {
54         System.out.println(document.toJson());
55     });
56 ;
57 }
58
59 @Test
60 public void testInsert(){
61     Document document = new Document("id",10001)
62         .append("name", "张三")
63         .append("age", 30);
64     this.mongoCollection.insertOne(document);
65     System.out.println("插入数据成功!");
66
67     this.mongoCollection.find(eq("id", 10001))
68         .forEach((Consumer<? super Document>) doc->{
69             System.out.println(doc.toJson());
70         });
71 }
72
73 @Test
74 public void testUpdate(){
75     UpdateResult updateResult = this.mongoCollection
76         .updateOne(eq("id", 10001), Updates.set("age", 25));
77     System.out.println(updateResult);
78
79     this.mongoCollection.find(eq("id", 10001))
80         .forEach((Consumer<? super Document>) doc->{
81             System.out.println(doc.toJson());
82         });
83 }
84
85 @Test
86 public void testDelete(){
```



```
87         DeleteResult deleteResult = this.mongoCollection.deleteOne(eq("id",
88         10001));
89         System.out.println(deleteResult);
90     }
91 }
92
```

## 2.4、面向对象操作

前面对MongoDB的操作都是基于Document对象操作，操作略显繁琐，下面我们通过面向对象的方式进行操作。

创建Person、Address对象：

```
1 package cn.itcast.mongodb;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6 import org.bson.types.ObjectId;
7
8 @Data
9 @AllArgsConstructor
10 @NoArgsConstructor
11 public class Person {
12
13     private ObjectId id;
14     private String name;
15     private int age;
16     private Address address;
17
18 }
19
```

```
1 package cn.itcast.mongodb;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 @Data
8 @AllArgsConstructor
9 @NoArgsConstructor
10 public class Address {
11
12     private String street;
13     private String city;
14     private String zip;
15 }
16
```

编写测试用例：



```
1 package cn.itcast.mongodb;
2
3 import com.mongodb.MongoClientSettings;
4 import com.mongodb.client.MongoClient;
5 import com.mongodb.client.MongoClients;
6 import com.mongodb.client.MongoCollection;
7 import com.mongodb.client.MongoDatabase;
8 import com.mongodb.client.model.Filters;
9 import com.mongodb.client.model.Updates;
10 import com.mongodb.client.result.DeleteResult;
11 import com.mongodb.client.result.UpdateResult;
12 import org.bson.codecs.configuration.CodecRegistries;
13 import org.bson.codecs.configuration.CodecRegistry;
14 import org.bson.codecs.pojo.PojoCodecProvider;
15 import org.bson.types.ObjectId;
16 import org.junit.Before;
17 import org.junit.Test;
18
19 import java.util.Arrays;
20 import java.util.List;
21 import java.util.function.Consumer;
22
23 public class TestPerson {
24
25     MongoCollection<Person> personCollection;
26
27     @Before
28     public void init() {
29
30         //定义对象的解码注册器
31         CodecRegistry pojoCodecRegistry = CodecRegistries.
32             fromRegistries(MongoClientSettings.getDefaultCodecRegistry(),
33
34             CodecRegistries.fromProviders(PojoCodecProvider.builder().automatic(true).build())
35         );
36
37         // 建立连接
38         MongoClient mongoClient =
39             MongoClients.create("mongodb://172.16.55.185:27017");
40
41         // 选择数据库 并且 注册解码器
42         MongoDatabase mongoDatabase = mongoClient.getDatabase("testdb")
43             .withCodecRegistry(pojoCodecRegistry);
44
45         // 选择表
46         this.personCollection = mongoDatabase
47             .getCollection("person", Person.class);
48
49     }
50
51     @Test
52     public void testInsert() {
53         Person person = new Person(ObjectId.get(), "张三", 20,
```





```
52         new Address("人民路", "上海市", "666666"));
53         this.personCollection.insertOne(person);
54         System.out.println("插入数据成功");
55     }
56
57     @Test
58     public void testInserts() {
59         List<Person> personList = Arrays.asList(new Person(ObjectId.get(), "张三",
60 20, new Address("人民路", "上海市", "666666")),
61         new Person(ObjectId.get(), "李四", 21, new Address("北京西路", "上海
62 市", "666666")),
63         new Person(ObjectId.get(), "王五", 22, new Address("南京东路", "上海
64 市", "666666")),
65         new Person(ObjectId.get(), "赵六", 23, new Address("陕西南路", "上海
66 市", "666666")),
67         new Person(ObjectId.get(), "孙七", 24, new Address("南京西路", "上海
68 市", "666666")));
69         this.personCollection.insertMany(personList);
70         System.out.println("插入数据成功");
71     }
72
73     @Test
74     public void testQuery() {
75         this.personCollection.find(Filters.eq("name", "张三"))
76             .forEach((Consumer<? super Person>) person -> {
77                 System.out.println(person);
78             });
79     }
80
81     @Test
82     public void testUpdate() {
83         UpdateResult updateResult =
84 this.personCollection.updateMany(Filters.eq("name", "张三"), Updates.set("age",
85 22));
86         System.out.println(updateResult);
87     }
88
89     @Test
90     public void testDelete() {
91         DeleteResult deleteResult =
92 this.personCollection.deleteOne(Filters.eq("name", "张三"));
93         System.out.println(deleteResult);
94     }
95 }
```

### 3、SpringBoot整合MongoDB

spring-data对MongoDB做了支持，使用spring-data-mongodb可以简化MongoDB的操作。

地址：<https://spring.io/projects/spring-data-mongodb>



第一步，导入依赖：

```
1 <parent>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-parent</artifactId>
4     <version>2.1.0.RELEASE</version>
5 </parent>
6
7 <dependency>
8     <groupId>org.springframework.boot</groupId>
9     <artifactId>spring-boot-starter-data-mongodb</artifactId>
10 </dependency>
11 <dependency>
12     <groupId>org.springframework.boot</groupId>
13     <artifactId>spring-boot-starter-test</artifactId>
14     <scope>test</scope>
15 </dependency>
```

第二步，编写application.properties配置文件

```
1 # Spring boot application
2 spring.application.name = itcast-mongodb
3
4 spring.data.mongodb.uri=mongodb://172.16.55.185:27017/testdb
5
```

第三步，编写PersonDao

```
1 package cn.itcast.mongodb.dao;
2
3 import cn.itcast.mongodb.Person;
4 import com.mongodb.client.result.DeleteResult;
5 import com.mongodb.client.result.UpdateResult;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.data.mongodb.core.MongoTemplate;
8 import org.springframework.data.mongodb.core.query.Criteria;
9 import org.springframework.data.mongodb.core.query.Query;
10 import org.springframework.data.mongodb.core.query.Update;
11 import org.springframework.stereotype.Component;
12
13 import java.util.List;
14
15 @Component
16 public class PersonDao {
17
18     @Autowired
19     private MongoTemplate mongoTemplate;
20
21     public void savePerson(Person person) {
22         this.mongoTemplate.save(person);
23     }
24 }
```



```
25     public List<Person> queryPersonListByName(String name) {
26         Query query = Query.query(Criteria.where("name").is(name));
27         return this.mongoTemplate.find(query, Person.class);
28     }
29
30     public List<Person> queryPersonListByName(Integer page, Integer rows) {
31         Query query = new Query().limit(rows).skip((page - 1) * rows);
32         return this.mongoTemplate.find(query, Person.class);
33     }
34
35     public UpdateResult update(Person person) {
36         Query query = Query.query(Criteria.where("id").is(person.getId()));
37         Update update = Update.update("age", person.getAge());
38         return this.mongoTemplate.updateFirst(query, update, Person.class);
39     }
40
41     public DeleteResult deleteById(String id) {
42         Query query = Query.query(Criteria.where("id").is(id));
43         return this.mongoTemplate.remove(query, Person.class);
44     }
45 }
46
```

#### 第四步，编写启动类

```
1 package cn.itcast.mongodb;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class MongoApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(MongoApplication.class, args);
11     }
12 }
13
```

#### 第五步，编写单元测试

```
1 package cn.itcast.mongodb;
2
3 import cn.itcast.mongodb.dao.PersonDao;
4 import org.bson.types.ObjectId;
5 import org.junit.Test;
6 import org.junit.runner.RunWith;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.boot.test.context.SpringBootTest;
9 import org.springframework.test.context.junit4.SpringRunner;
10
11 import java.util.List;
```



```
12
13 @RunWith(SpringRunner.class)
14 @SpringBootTest
15 public class TestPersonDao {
16
17     @Autowired
18     private PersonDao personDao;
19
20     @Test
21     public void testSave() {
22         Person person = new Person(ObjectId.get(), "张三", 20,
23             new Address("人民路", "上海市", "666666"));
24         this.personDao.savePerson(person);
25     }
26
27     @Test
28     public void testQuery() {
29         List<Person> personList = this.personDao.queryPersonListByName("张三");
30         for (Person person : personList) {
31             System.out.println(person);
32         }
33     }
34
35     @Test
36     public void testQuery2() {
37         List<Person> personList = this.personDao.queryPersonListByName(2, 2);
38         for (Person person : personList) {
39             System.out.println(person);
40         }
41     }
42
43     @Test
44     public void testUpdate() {
45         Person person = new Person();
46         person.setId(new ObjectId("5c0956ce235e192520086736"));
47         person.setAge(30);
48         this.personDao.update(person);
49     }
50
51     @Test
52     public void testDelete() {
53         this.personDao.deleteById("5c09ca05235e192d8887a389");
54     }
55
56 }
57
```

## 4、搭建微聊系统

下面我们开发微聊系统，实现好客租房项目的即时通讯功能。

使用到的技术：

- Spring WebSocket
- Spring-data-MongoDB

## 4.1、创建工程itcast-haoke-im

pom.xml :

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <parent>
8         <groupId>org.springframework.boot</groupId>
9         <artifactId>spring-boot-starter-parent</artifactId>
10        <version>2.1.0.RELEASE</version>
11    </parent>
12
13    <groupId>cn.itcast.haoke.im</groupId>
14    <artifactId>itcast-haoke-im</artifactId>
15    <version>1.0-SNAPSHOT</version>
16
17    <dependencies>
18        <dependency>
19            <groupId>org.springframework.boot</groupId>
20            <artifactId>spring-boot-starter-web</artifactId>
21        </dependency>
22        <dependency>
23            <groupId>org.springframework.boot</groupId>
24            <artifactId>spring-boot-starter-data-mongodb</artifactId>
25        </dependency>
26        <dependency>
27            <groupId>org.springframework.boot</groupId>
28            <artifactId>spring-boot-starter-websocket</artifactId>
29        </dependency>
30        <dependency>
31            <groupId>org.springframework.boot</groupId>
32            <artifactId>spring-boot-starter-test</artifactId>
33            <scope>test</scope>
34        </dependency>
35        <dependency>
36            <groupId>org.mongodb</groupId>
37            <artifactId>mongodb-driver-sync</artifactId>
38            <version>3.9.1</version>
39        </dependency>
40        <dependency>
41            <groupId>junit</groupId>
42            <artifactId>junit</artifactId>
43            <version>4.12</version>
44            <scope>test</scope>
45        </dependency>
```



```
46     <dependency>
47         <groupId>org.projectlombok</groupId>
48         <artifactId>lombok</artifactId>
49         <version>1.18.4</version>
50     </dependency>
51     <dependency>
52         <groupId>org.apache.commons</groupId>
53         <artifactId>commons-lang3</artifactId>
54     </dependency>
55 </dependencies>
56
57 <build>
58     <plugins>
59         <!-- java编译插件 -->
60         <plugin>
61             <groupId>org.apache.maven.plugins</groupId>
62             <artifactId>maven-compiler-plugin</artifactId>
63             <version>3.2</version>
64             <configuration>
65                 <source>1.8</source>
66                 <target>1.8</target>
67                 <encoding>UTF-8</encoding>
68             </configuration>
69         </plugin>
70     </plugins>
71 </build>
72
73 </project>
```

## 4.2、编写Message对象

```
1  package cn.itcast.haoke.im.pojo;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Builder;
5  import lombok.Data;
6  import lombok.NoArgsConstructor;
7  import org.bson.types.ObjectId;
8  import org.springframework.data.annotation.Id;
9  import org.springframework.data.mongodb.core.index.Indexed;
10 import org.springframework.data.mongodb.core.mapping.Document;
11 import org.springframework.data.mongodb.core.mapping.Field;
12
13 import java.util.Date;
14
15 @Data
16 @AllArgsConstructor
17 @NoArgsConstructor
18 @Document(collection = "message")
19 @Builder
20 public class Message {
21
```



```
22     @Id
23     private ObjectId id;
24     private String msg;
25     /**
26      * 消息状态, 1-未读, 2-已读
27      */
28     @Indexed
29     private Integer status;
30     @Field("send_date")
31     @Indexed
32     private Date sendDate;
33     @Field("read_date")
34     private Date readDate;
35     @Indexed
36     private User from;
37     @Indexed
38     private User to;
39
40 }
41
```

```
1 package cn.itcast.haoke.im.pojo;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Builder;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7
8 @Data
9 @AllArgsConstructor
10 @NoArgsConstructor
11 @Builder
12 public class User {
13
14     private Long id;
15     private String username;
16
17 }
```

构造用户数据：

```
1 package cn.itcast.haoke.im.pojo;
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6 public class UserData {
7
8     public static final Map<Long, User> USER_MAP = new HashMap<>();
9
10     static {
11         USER_MAP.put(1001L, User.builder().id(1001L).username("zhangsan").build());
12     }
13 }
```



```
12     USER_MAP.put(1002L, User.builder().id(1002L).username("lisi").build());
13     USER_MAP.put(1003L, User.builder().id(1003L).username("wangwu").build());
14     USER_MAP.put(1004L, User.builder().id(1004L).username("zhaoliu").build());
15     USER_MAP.put(1005L, User.builder().id(1005L).username("sunqi").build());
16 }
17 }
```

### 4.3、编写MessageDAO

定义MessageDAO接口：

```
1 package cn.itcast.haoke.im.dao;
2
3 import cn.itcast.haoke.im.pojo.Message;
4 import com.mongodb.client.result.DeleteResult;
5 import com.mongodb.client.result.UpdateResult;
6 import org.bson.types.ObjectId;
7
8 import java.util.List;
9
10 public interface MessageDAO {
11
12     /**
13      * 查询点对点聊天记录
14      *
15      * @param fromId
16      * @param toId
17      * @param page
18      * @param rows
19      * @return
20      */
21     List<Message> findListByFromAndTo(Long fromId, Long toId, Integer page, Integer rows);
22
23     /**
24      * 根据id查询数据
25      *
26      * @param id
27      * @return
28      */
29     Message findMessageById(String id);
30
31     /**
32      * 更新消息状态
33      *
34      * @param id
35      * @param status
36      * @return
37      */
38     UpdateResult updateMessageState(ObjectId id, Integer status);
39
40     /**
41      * 新增消息数据
```





```
42      *
43      * @param message
44      * @return
45      */
46      Message saveMessage(Message message);
47
48      /**
49      * 根据消息id删除数据
50      *
51      * @param id
52      * @return
53      */
54      DeleteResult deleteMessage(String id);
55  }
```

编写实现类：

```
1  package cn.itcast.haoke.im.dao.impl;
2
3  import cn.itcast.haoke.im.dao.MessageDAO;
4  import cn.itcast.haoke.im.pojo.Message;
5  import cn.itcast.haoke.im.pojo.User;
6  import com.mongodb.client.result.DeleteResult;
7  import com.mongodb.client.result.UpdateResult;
8  import org.bson.types.ObjectId;
9  import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.data.domain.PageRequest;
11 import org.springframework.data.domain.Sort;
12 import org.springframework.data.mongodb.core.MongoTemplate;
13 import org.springframework.data.mongodb.core.query.Criteria;
14 import org.springframework.data.mongodb.core.query.Query;
15 import org.springframework.data.mongodb.core.query.Update;
16 import org.springframework.stereotype.Component;
17
18 import java.util.Date;
19 import java.util.List;
20
21 @Component
22 public class MessageDAOImpl implements MessageDAO {
23
24     @Autowired
25     private MongoTemplate mongoTemplate;
26
27     @Override
28     public List<Message> findListByFromAndTo(Long fromId, Long toId, Integer page,
29 Integer rows) {
30
31         Criteria fromList =
32         Criteria.where("from.id").is(fromId).and("to.id").is(toId);
33         Criteria toList =
34         Criteria.where("from.id").is(toId).and("to.id").is(fromId);
35
36         Criteria criteria = new Criteria().orOperator(fromList, toList);
```



```
34
35     PageRequest pageRequest = PageRequest.of(page - 1, rows,
Sort.by(Sort.Direction.ASC, "send_date"));
36     Query query = new Query(criteria).with(pageRequest);
37
38     System.out.println(query);
39
40     return this.mongoTemplate.find(query, Message.class);
41 }
42
43 @Override
44 public Message findMessageById(String id) {
45     return this.mongoTemplate.findById(new ObjectId(id), Message.class);
46 }
47
48 @Override
49 public UpdateResult updateMessageState(ObjectId id, Integer status) {
50     Query query = Query.query(Criteria.where("id").is(id));
51
52     Update update = Update.update("status", status);
53     if (status.intValue() == 1) {
54         update.set("send_date", new Date());
55     } else if (status.intValue() == 2) {
56         update.set("read_date", new Date());
57     }
58     return this.mongoTemplate.updateFirst(query, update, Message.class);
59 }
60
61 @Override
62 public Message saveMessage(Message message) {
63     message.setId(ObjectId.get());
64     message.setSendDate(new Date());
65     message.setStatus(1);
66     return this.mongoTemplate.save(message);
67 }
68
69 @Override
70 public DeleteResult deleteMessage(String id) {
71     Query query = Query.query(Criteria.where("id").is(id));
72     return this.mongoTemplate.remove(query, Message.class);
73 }
74 }
```

## 4.4、编写单元测试

```
1 package cn.itcast.haoke.im.dao;
2
3 import cn.itcast.haoke.im.pojo.Message;
4 import cn.itcast.haoke.im.pojo.User;
5 import org.bson.types.ObjectId;
6 import org.junit.Test;
7 import org.junit.runner.RunWith;
```



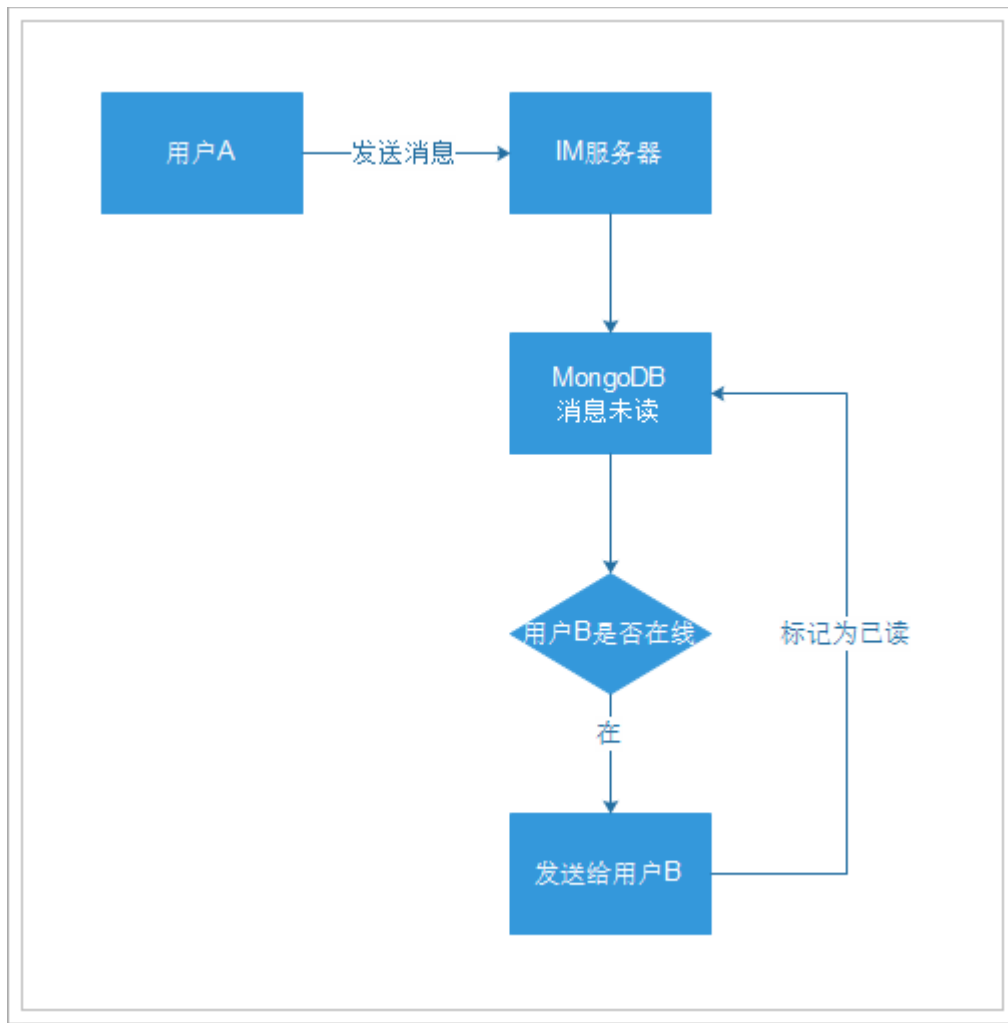
```
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.test.context.junit4.SpringRunner;
11
12 import java.util.Date;
13 import java.util.List;
14
15 @RunWith(SpringRunner.class)
16 @SpringBootTest
17 public class TestMessageDAO {
18
19     @Autowired
20     private MessageDAO messageDAO;
21
22     @Test
23     public void testSave(){
24         Message message = Message.builder()
25             .id(ObjectId.get())
26             .msg("你好")
27             .sendDate(new Date())
28             .status(1)
29             .from(new User(1001L, "zhangsan"))
30             .to(new User(1002L, "lisi"))
31             .build();
32         this.messageDAO.saveMessage(message);
33
34         message = Message.builder()
35             .id(ObjectId.get())
36             .msg("你也好")
37             .sendDate(new Date())
38             .status(1)
39             .to(new User(1001L, "zhangsan"))
40             .from(new User(1002L, "lisi"))
41             .build();
42         this.messageDAO.saveMessage(message);
43
44         message = Message.builder()
45             .id(ObjectId.get())
46             .msg("我在学习开发IM")
47             .sendDate(new Date())
48             .status(1)
49             .from(new User(1001L, "zhangsan"))
50             .to(new User(1002L, "lisi"))
51             .build();
52         this.messageDAO.saveMessage(message);
53
54         message = Message.builder()
55             .id(ObjectId.get())
56             .msg("那很好啊!")
57             .sendDate(new Date())
58             .status(1)
59             .to(new User(1001L, "zhangsan"))
60             .from(new User(1002L, "lisi"))
```



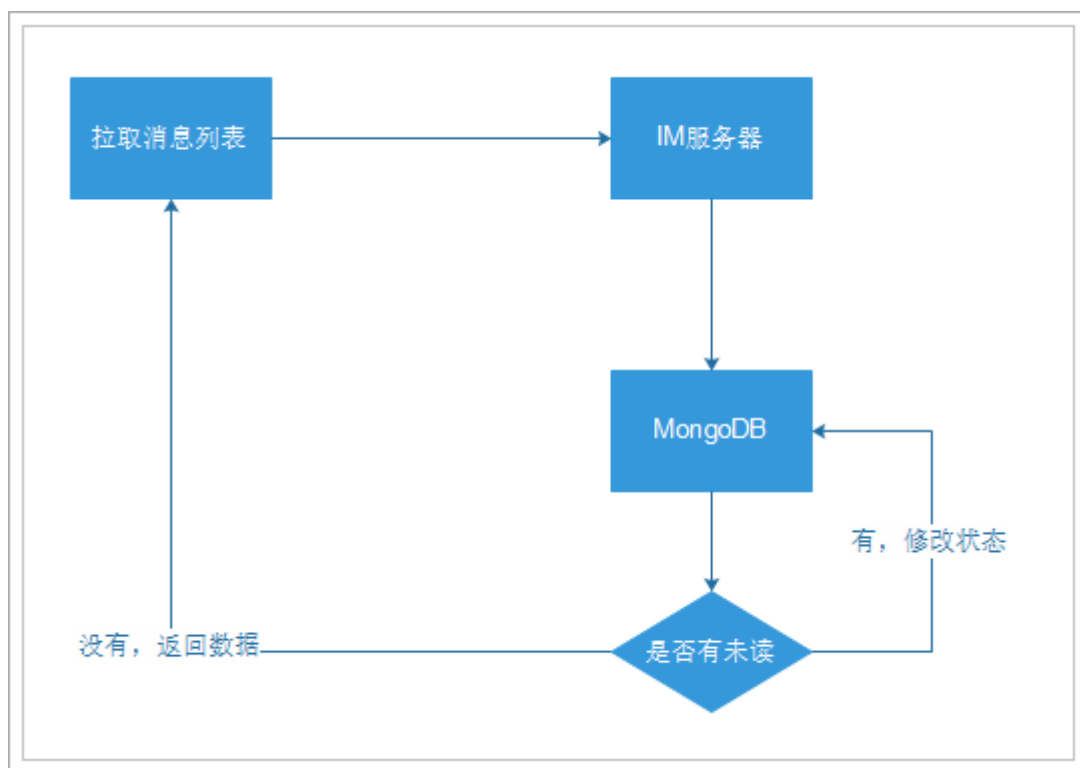
```
61         .build();
62         this.messageDAO.saveMessage(message);
63
64         System.out.println("ok");
65     }
66
67     @Test
68     public void testQueryById(){
69         Message message =
70         this.messageDAO.findMessageById("5c0a9109235e193bd4873b92");
71         System.out.println(message);
72     }
73
74     @Test
75     public void testQueryList(){
76         List<Message> list = this.messageDAO.findListByFromAndTo(1001L, 1002L, 2,
77         1);
78         for (Message message : list) {
79             System.out.println(message);
80         }
81     }
82 }
```

## 4.5、编写websocket

### 4.5.1、发送消息流程



#### 4.5.2、接收消息流程





### 4.5.3、实现

```
1 package cn.itcast.haoke.im.websocket;
2
3 import cn.itcast.haoke.im.dao.MessageDAO;
4 import cn.itcast.haoke.im.pojo.Message;
5 import cn.itcast.haoke.im.pojo.UserData;
6 import com.fasterxml.jackson.databind.JsonNode;
7 import com.fasterxml.jackson.databind.ObjectMapper;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.stereotype.Component;
10 import org.springframework.web.socket.TextMessage;
11 import org.springframework.web.socket.WebSocketSession;
12 import org.springframework.web.socket.handler.TextWebSocketHandler;
13
14 import java.util.HashMap;
15 import java.util.Map;
16
17 @Component
18 public class MessageHandler extends TextWebSocketHandler {
19
20     @Autowired
21     private MessageDAO messageDAO;
22
23     private static final ObjectMapper MAPPER = new ObjectMapper();
24
25     private static final Map<Long, WebSocketSession> SESSIONS = new HashMap<>();
26
27     @Override
28     public void afterConnectionEstablished(WebSocketSession session) throws
29         Exception {
30         Long uid = (Long)session.getAttributes().get("uid");
31         // 将当前用户的session放置到map中，后面会使用相应的session通信
32         SESSIONS.put(uid, session);
33     }
34
35
36     @Override
37     protected void handleTextMessage(WebSocketSession session, TextMessage
38 textMessage) throws Exception {
39         Long uid = (Long)session.getAttributes().get("uid");
40
41         JsonNode jsonNode = MAPPER.readTree(textMessage.getPayload());
42         Long toId = jsonNode.get("toId").asLong();
43         String msg = jsonNode.get("msg").asText();
44
45         Message message = Message.builder()
46             .from(UserData.USER_MAP.get(uid))
47             .to(UserData.USER_MAP.get(toId))
48             .msg(msg)
49             .build();
50
51         // 将消息保存到MongoDB
```



```
51     message = this.messageDAO.saveMessage(message);
52
53     // 判断to用户是否在线
54     WebSocketSession toSession = SESSIONS.get(toId);
55     if(toSession != null && toSession.isOpen()){
56         //TODO 具体格式需要和前端对接
57         toSession.sendMessage(new
58             TextMessage(MAPPER.writeValueAsString(message)));
59
60         // 更新消息状态为已读
61         this.messageDAO.updateMessageState(message.getId(), 2);
62     }
63 }
64 }
```

```
1  package cn.itcast.haoke.im.websocket;
2
3  import org.apache.commons.lang3.StringUtils;
4  import org.springframework.http.server.ServerHttpRequest;
5  import org.springframework.http.server.ServerHttpResponse;
6  import org.springframework.stereotype.Component;
7  import org.springframework.web.socket.WebSocketHandler;
8  import org.springframework.web.socket.server.HandshakeInterceptor;
9
10 import java.util.Map;
11
12 @Component
13 public class MessageHandshakeInterceptor implements HandshakeInterceptor {
14
15     @Override
16     public boolean beforeHandshake(ServerHttpRequest request, ServerHttpResponse
17         response, WebSocketHandler wsHandler, Map<String, Object> attributes) throws
18         Exception {
19         String path = request.getURI().getPath();
20         String[] ss = StringUtils.split(path, '/');
21         if(ss.length != 2){
22             return false;
23         }
24         if(!StringUtils.isNumeric(ss[1])){
25             return false;
26         }
27         attributes.put("uid", Long.valueOf(ss[1]));
28         return true;
29     }
30
31     @Override
32     public void afterHandshake(ServerHttpRequest request, ServerHttpResponse
33         response, WebSocketHandler wsHandler, Exception exception) {
34     }
35 }
```



```
1 package cn.itcast.haoke.im.websocket;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.web.socket.config.annotation.EnableWebSocket;
6 import org.springframework.web.socket.config.annotation.WebSocketConfigurer;
7 import org.springframework.web.socket.config.annotation.WebSocketHandlerRegistry;
8
9 @Configuration
10 @EnableWebSocket
11 public class WebSocketConfig implements WebSocketConfigurer {
12
13     @Autowired
14     private MessageHandler messageHandler;
15
16     @Autowired
17     private MessageHandshakeInterceptor messageHandshakeInterceptor;
18
19     @Override
20     public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
21         registry.addHandler(this.messageHandler, "/ws/{uid}")
22             .setAllowedOrigins("*")
23             .addInterceptors(this.messageHandshakeInterceptor);
24     }
25 }
```

## 4.6、编写启动类

```
1 package cn.itcast.haoke.im;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class ImApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(ImApplication.class, args);
11     }
12
13 }
14 }
```

## 4.7、测试





Server Location

URL:

Status: **OPENED**

模拟用户1001登录，发送消息

Request

```
{
  "toId": 1002,
  "msg": "你好，1002，在吗"
}
```

[Shortcut] Ctrl + Enter

Message Log

```
{
  "toId": 1002,
  "msg": "你好，1002，在吗"
}
{"id":{"timestamp":1544244142,"machineIdentifier":2317849,"processIdentifier":16728,"counter":14405583,"date":1544244142000,"
```

服务器配置 状态: 连接成功

消息记录 [清空消息](#)

服务地址

☐ 收包清空记录 ☐ 收包JSON解码 ☐ 暂停接收

发包设置

每隔  秒发送内容

```
{
  "toId": 1001,
  "msg": "你好，1001，我在的"
}
```

☐ 发包清空输入

调试消息

12:41:03 => 初始化完成

12:41:08 => OPENED => 127.0.0.1:18081/ws/1002

收到消息 12:41:57

```
{
  "id": {
    "timestamp": 1544244116,
    "machineIdentifier": 2317849,
    "processIdentifier": 16728,
    "counter": 14405582,
    "date": 1544244116000,
    "time": 1544244116000,
    "timeSecond": 1544244116
  },
  "msg": "你好，1002，在吗",
  "status": 1,
  "sendDate": 1544244116880,
  "readDate": null,
  "from": {
    "id": 1001,
    "username": "zhangsan"
  },
  "to": {
    "id": 1002,
    "username": "lisi"
  }
}
```

发送消息 12:42:20

```
{
  "toId": 1001,
  "msg": "你好，1001，我在的"
}
```

模拟用户1002登录，发送消息

## 4.8、提供查询历史消息的服务

```
1 package cn.itcast.haoke.im.controller;
2
3 import cn.itcast.haoke.im.pojo.Message;
4 import cn.itcast.haoke.im.service.MessageService;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.*;
7
8 import java.util.List;
```

北京市昌平区建材城西路金燕龙办公楼一层

电话：400-618-9090



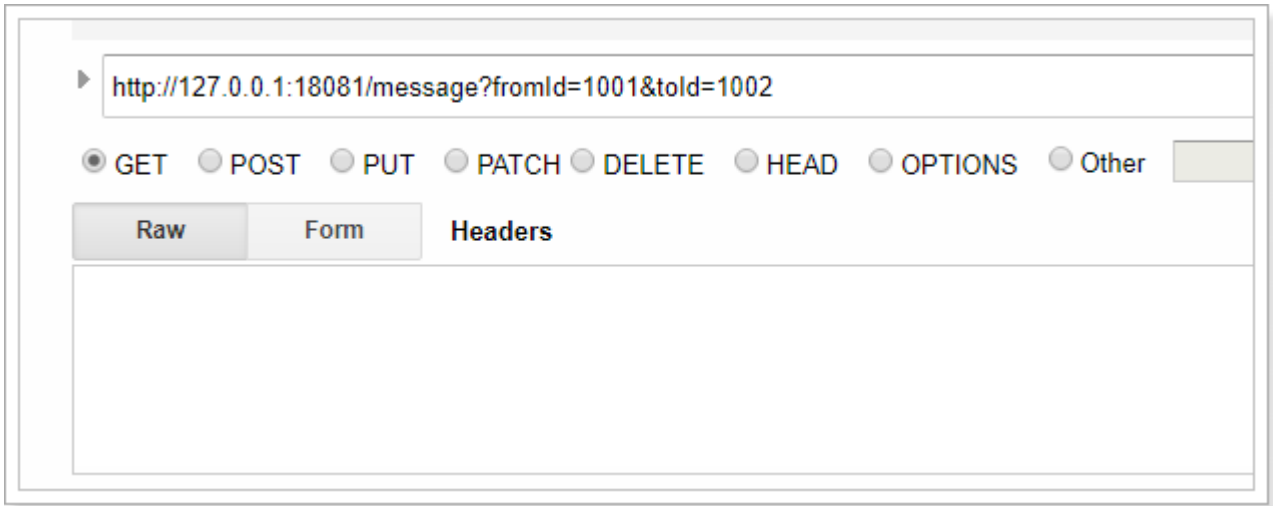
```
9
10 @RestController
11 @RequestMapping("message")
12 @CrossOrigin
13 public class MessageController {
14
15     @Autowired
16     private MessageService messageService;
17
18
19     /**
20      * 拉取消息列表
21      *
22      * @param fromId
23      * @param toId
24      * @param page
25      * @param rows
26      * @return
27      */
28     @GetMapping
29     public List<Message> queryMessageList(@RequestParam("fromId") Long fromId,
30                                           @RequestParam("toId") Long toId,
31                                           @RequestParam(value = "page",
32                                                         defaultValue = "1") Integer page,
33                                           @RequestParam(value = "rows",
34                                                         defaultValue = "10") Integer rows) {
35         return this.messageService.queryMessageList(fromId, toId, page, rows);
36     }
37 }
```

```
1 package cn.itcast.haoke.im.service;
2
3 import cn.itcast.haoke.im.dao.MessageDAO;
4 import cn.itcast.haoke.im.pojo.Message;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import java.util.List;
9
10 @Service
11 public class MessageService {
12
13     @Autowired
14     private MessageDAO messageDAO;
15
16     public List<Message> queryMessageList(Long fromId, Long toId, Integer page,
17                                           Integer rows) {
18         List<Message> list = this.messageDAO.findListByFromAndTo(fromId, toId,
19                                                                     page, rows);
20         for (Message message : list) {
21             if(message.getStatus().intValue() == 1){
22                 // 修改消息状态为已读
23             }
24         }
25     }
26 }
```



```
21         this.messageDAO.updateMessageState(message.getId(), 2);
22     }
23 }
24 return list;
25 }
26 }
27 }
```

测试：





```
Raw JSON Response
Copy to clipboard Save as file
[3]
-0: {
  -id: {
    timestamp: 1544202529
    machineIdentifier: 2317849
    processIdentifier: 16540
    counter: 14301597
    date: "2018-12-07T17:08:49.000+0000"
    time: 1544202529000
    timeSecond: 1544202529
  }
  msg: "你好"
  status: 1
  sendDate: "2018-12-07T17:08:49.046+0000"
  readDate: null
  -from: {
    id: 1001
    username: "zhangsan"
  }
  -to: {
    id: 1002
    username: "lisi"
  }
}
-1: {
  -id: {
    timestamp: 1544202856
    machineIdentifier: 2317849
    processIdentifier: 16540
```

## 4.9、提供查询用户列表服务 ( mock实现 )

数据结构：

```
1 {
2   "data": {
3     "list": [
4       {
5         "id": 3,
6         "from_user": 1,
7         "to_user": 3,
8         "chat_time": 1531045313395,
9         "chat_msg": "12213123",
10        "info_type": null,
11        "username": "spike",
12        "avatar": "public/icon.png"
13      },
```



```
14         {
15             "id": 4,
16             "from_user": 1,
17             "to_user": 3,
18             "chat_time": 1531045313395,
19             "chat_msg": "12",
20             "info_type": null,
21             "username": "spike",
22             "avatar": "public/icon.png"
23         },
24         {
25             "id": 5,
26             "from_user": 1,
27             "to_user": 3,
28             "chat_time": 1531045313395,
29             "chat_msg": "12",
30             "info_type": null,
31             "username": "spike",
32             "avatar": "public/icon.png"
33         },
34         {
35             "id": 6,
36             "from_user": 1,
37             "to_user": 3,
38             "chat_time": 1531045313395,
39             "chat_msg": "1212",
40             "info_type": null,
41             "username": "spike",
42             "avatar": "public/icon.png"
43         }
44     ],
45     "meta": {
46         "status": 200,
47         "msg": "用户数据"
48     }
49 }
50 }
```

```
1 package cn.itcast.haoke.im.controller;
2
3 import cn.itcast.haoke.im.pojo.Message;
4 import cn.itcast.haoke.im.pojo.User;
5 import cn.itcast.haoke.im.pojo.UserData;
6 import cn.itcast.haoke.im.service.MessageService;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.web.bind.annotation.*;
9
10 import java.util.ArrayList;
11 import java.util.HashMap;
12 import java.util.List;
13 import java.util.Map;
```



```
14
15 @RestController
16 @CrossOrigin
17 @RequestMapping("user")
18 public class UserController {
19
20     @Autowired
21     private MessageService messageService;
22
23     //拉取用户列表（模拟实现）
24     @GetMapping
25     public List<Map<String, Object>> queryUserList(@RequestParam("fromId") Long
fromId) {
26         List<Map<String, Object>> result = new ArrayList<>();
27
28         for (Map.Entry<Long, User> userEntry : UserData.USER_MAP.entrySet()) {
29             Map<String, Object> map = new HashMap<>();
30             map.put("id", userEntry.getValue().getId());
31             map.put("avatar", "http://itcast-haoke.oss-cn-
qingdao.aliyuncs.com/images/2018/12/08/15442410962743524.jpg");
32
33             map.put("from_user", fromId);
34             map.put("info_type", null);
35             map.put("to_user", map.get("id"));
36             map.put("username", userEntry.getValue().getUsername());
37
38             // 获取最后一条消息
39             List<Message> messages = this.messageService.queryMessageList(fromId,
userEntry.getValue().getId(), 1, 1);
40             if (messages != null && !messages.isEmpty()) {
41                 Message message = messages.get(0);
42                 map.put("chat_msg", message.getMsg());
43                 map.put("chat_time", message.getSendDate().getTime());
44             }
45
46             result.add(map);
47         }
48
49         return result;
50     }
51
52 }
```

测试：



▶

☒ GET ☐ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD ☐ OPTIONS ☐ Other

Raw

Form

Headers

Raw JSON Response

[Copy to clipboard](#) [Save as file](#)

```
[5]
-0: {
  to_user: 1001
  info_type: null
  id: 1001
  avatar: "http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/12/08/15442410962743524.jpg"
  from_user: 1001
  username: "zhangsan"
}
-1: {
  to_user: 1002
  info_type: null
  chat_time: 1544202529046
  id: 1002
  avatar: "http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/12/08/15442410962743524.jpg"
  chat_msg: "你好"
  from_user: 1001
  username: "lisi"
}
-2: {
  to_user: 1003
  info_type: null
  chat_time: 1544202763981
  id: 1003
  avatar: "http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/12/08/15442410962743524.jpg"
  chat_msg: "1003你好，好久不见"
  from_user: 1001
  username: "wangwu"
}
-3: {
  to_user: 1004
```

## 5、整合前端实现微聊功能

### 5.1、查询好友列表



chat.js :

```
1  componentDidMount = () => {
2    axios.get('http://127.0.0.1:18081/user?fromId=1001').then((data) => {
3      this.setState({
4        list: data,
5        isLoading: true
6      })
7    })
8  }
9
10 -----
11 if (isLoading) {
12   list = this.state.list.map(item => {
13     return (
14       <li key={item.id} onClick={(e) => this.toChat(e, {item})}>
15         <div className="avarter">
16           <img src={item.avatar} alt="avarter"/>
17           <span className="name">{item.username}</span>
18           <span className="info">{item.chat_msg}</span>
19           <span className="time">{item.ctime}</span>
20         </div>
21       </li>
22     )
23   })
24 }
```

效果：





## 5.2、好友单聊

chat-window.js文件：

```
1  componentDidMount = () => {  
2    let {to_user,from_user} = this.props.chatInfo;  
3    axios.get('http://127.0.0.1:18081/message',{params:{  
4      toId: to_user,  
5      fromId: from_user  
6    }}).then(data=>{  
7      this.setState({  
8        infos: data,  
9        isLoading: true,
```



```
10     fromId: from_user,
11     client: handle(localStorage.getItem('uid'),(data)=>{
12         let newList = [...this.state.infos];
13         newList.push(JSON.parse(data.content));
14         this.setState({
15             infos: newList
16         })
17     })
18 });
19 })
20 }
21
22 if(this.state.isLoading) {
23     // let currentUser = parseInt(localStorage.getItem('uid'),10);
24     let currentUser = parseInt(this.state.fromId,10);
25     infoList = this.state.infos.map(item=>{
26         return (
27             <li key={item.id} className={currentUser===item.from.id? 'chat-info-
right': 'chat-info-left'}>
28                 
29                 <span>{item.msg}</span>
30             </li>
31         )
32     })
33 }
```

效果：



### 5.3、发送消息

chat-window.js :

```
1  sendMsg = () => {  
2    let {to_user,from_user,avatar} = this.props.chatInfo;  
3    let pdata = {  
4      id: this.guid(),  
5      // from_user: from_user,  
6      toId: to_user,  
7      from:{  
8        id:this.state.fromId  
9      },
```



```
10      // avatar: avatar,  
11      msg: this.state.msgContent  
12    }  
13    let newList = [...this.state.infos];  
14    newList.push(pdata);  
15    this.setState({  
16      infos: newList  
17    })  
18  
19    this.state.client.emitEvent(IMEvent.MSG_TEXT_SEND, JSON.stringify(pdata));  
20  }
```

修改IMClient.js，不包装，直接发送数据：

```
1  // 向服务器发送数据包  
2  sendDataPacket(dataPacket) {  
3    // if (this._isOpened) {  
4    //   this._socket.send(dataPacket.rawMessage);  
5    // } else {  
6    //   this._DataPacketQueue.push(dataPacket);  
7    // }  
8    // 直接发送，不包装  
9    this._socket.send(dataPacket);  
10 }
```

测试：

```
{id: "328a9e94-d160-82f7-8aae-d785060f4f2e", toId: 1002, from: {id: 1001}, msg:  
  from: {id: 1001}  
  id: "328a9e94-d160-82f7-8aae-d785060f4f2e"  
  msg: "hello"  
  toId: 1002}
```

## 5.4、接收消息

chat-window.js中，注册接收消息后的处理逻辑：

```
1  componentDidMount = () => {  
2    let {to_user, from_user} = this.props.chatInfo;  
3    axios.get('http://127.0.0.1:18081/message', {params: {  
4      toId: to_user,  
5      fromId: from_user  
6    }}).then(data => {  
7      this.setState({  
8        infos: data,  
9        isLoading: true,  
10       fromId: from_user,  
11       client: handle(from_user, (data) => {  
12         let newList = [...this.state.infos];
```



```
13     newList.push(JSON.parse(data));
14     this.setState({
15         infos: newList
16     })
17 })
18 });
19 })
20 }
```

修改wsmain.js文件：

```
1 const client = new IMClient(config.wsBaseUrl + "/" + currentUser, handleMessage);
```

修改IMClient.js文件：

```
1 constructor(url, onMyMessage) {
2     this._url = url;
3     this._autoConnect = true;
4     this._handlers = {};
5     this._DataPacketQueue = [];
6     this._isOpened = false;
7     this.onMyMessage = onMyMessage;
8
9     this.addEventListener(IMEvent.CONNECTED, () => {
10         this.serverOnConnected();
11     })
12
13     // this.addEventListener(IMEvent.CONNECTED, () => {
14     //     this.clearMsgQueue();
15     // })
16
17     this.addEventListener(IMEvent.DISCONNECTED, () => {
18         this.serverOnDisconnected();
19     })
20 }
21
```

```
1 connect() {
2     if (!this._socket) {
3         this._socket = new WebSocket(this._url);
4
5         this._socket.onmessage = (evt) => {
6             this.onMessage(evt.data);
7             if(this.onMyMessage){
8                 this.onMyMessage(evt.data);
9             }
10        }
11        this._socket.onopen = (ws) => {
12            this.onOpen(ws);
13        }
14        this._socket.onclose = ws => {
```



```
15     this.onClose(ws);  
16   }  
17   this._socket.onerror = ws => {  
18     this.onError(ws);  
19   };  
20 }  
21 }
```

测试：

发送消息：

服务器配置 状态：连接成功

服务地址

ws://127.0.0.1:18081/ws/1002

关闭连接

发包设置

每隔

1

秒发送内容

PING

开始发送

```
{  
  "toId":1001,  
  "msg":"我这里下雪了"  
}
```

☐ 发包清空输入

发送到服务端

调试消息

12:41:03 => 初始化完成

12:41:08 => OPENED => 127.0.0.1:18081/ws/1002

接收到消息：



可以看到，已经接收到了消息。

## 6、分布式Socket解决方案分析

问题：前面的实现中，将Session对象放到全局的Map中，当连接变得非常多时，这将成为系统瓶颈，因为不能进行分布式部署。

解决方案：采用消息系统进行解决。

