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

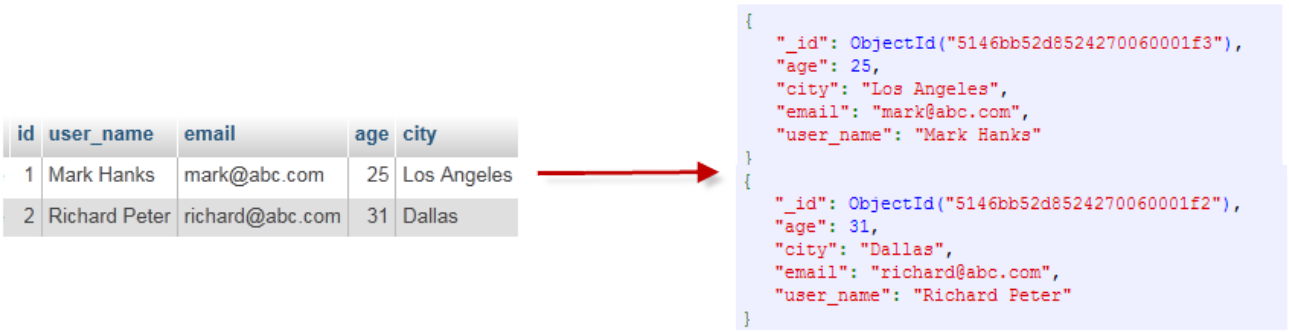
```
#拉取镜像
docker pull mongo:4.0.3

#创建容器
docker create --name mongodb -p 27017:27017 -v /data/mongodb:/data/db mongo:4.0.3

#启动容器
docker start mongodb

#进入容器
docker exec -it mongodb /bin/bash

#使用MongoDB客户端进行操作
mongo
> show dbs  #查询所有的数据库
admin   0.000GB
config  0.000GB
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



1.3.2、数据库以及表的操作

```
#查看所有的数据库
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB

#通过use关键字切换数据库
> use admin
switched to db admin

#创建数据库
#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
> use testdb
switched to db testdb
> show dbs #并没有创建数据库
admin   0.000GB
config  0.000GB
local   0.000GB
> db.user.insert({id:1,name:'zhangsan'})  #插入数据
WriteResult({ "nInserted" : 1 })
> show dbs
```

```
admin   0.000GB
config  0.000GB
local   0.000GB
testdb  0.000GB #数据库自动创建

#查看表
> show tables
user
> show collections
user
>

#删除集合（表）
> db.user.drop()
true  #如果成功删除选定集合，则 drop() 方法返回 true，否则返回 false。

#删除数据库
> use testdb #先切换到要删除的数据中
switched to db testdb
> db.dropDatabase()  #删除数据库
{ "dropped" : "testdb", "ok" : 1 }
> show dbs
admin   0.000GB
config  0.000GB
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）。

```
#插入数据

#语法：db.COLLECTION_NAME.insert(document)
> db.user.insert({id:1,username:'zhangsan',age:20})
WriteResult({ "nInserted" : 1 })
> db.user.save({id:2,username:'lisi',age:25})
WriteResult({ "nInserted" : 1 })
> db.user.find()  #查询数据
{ "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "id" : 1, "username" : "zhangsan",
"age" : 20 }
{ "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi",
"age" : 25 }

```

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：

```
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.user.find()
{ "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "id" : 1, "username" : "zhangsan", "age" : 20 }
{ "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi", "age" : 25 }

> db.user.update({id:1},{$set:{age:22}}) #更新数据

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find()
{ "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "id" : 1, "username" : "zhangsan", "age" : 22 }
{ "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi", "age" : 25 }

#注意：如果这样写，会删除掉其他的字段
> db.user.update({id:1},{age:25})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.user.find()
{ "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "age" : 25 }
{ "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi", "age" : 25 }

#更新不存在的字段，会新增字段
> db.user.update({id:2},{$set:{sex:1}}) #更新数据
> db.user.find()
{ "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "age" : 25 }
{ "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi", "age" : 25, "sex" : 1 }

#更新不存在的数据，默认不会新增数据
```

```
> db.user.update({id:3},{$set:{sex:1}})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
> db.user.find()
{ "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "age" : 25 }
{ "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi", "age" : 25, "sex" : 1 }

#如果设置第一个参数为true，就是新增数据
> db.user.update({id:3},{$set:{sex:1}},true)
WriteResult({
    "nMatched" : 0,
    "nUpserted" : 1,
    "nModified" : 0,
    "_id" : ObjectId("5c08cb281418d073246bc642")
})
> db.user.find()
{ "_id" : ObjectId("5c08c0024b318926e0c1f6dc"), "age" : 25 }
{ "_id" : ObjectId("5c08c0134b318926e0c1f6dd"), "id" : 2, "username" : "lisi", "age" : 25, "sex" : 1 }
{ "_id" : ObjectId("5c08cb281418d073246bc642"), "id" : 3, "sex" : 1 }
```

1.3.5、删除数据

通过remove()方法进行删除数据，语法如下：

```
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。

实例：

```
> db.user.remove({age:25})
WriteResult({ "nRemoved" : 2 })  #删除了2条数据

#插入4条测试数据
db.user.insert({id:1,username:'zhangsan',age:20})
db.user.insert({id:2,username:'lisi',age:21})
db.user.insert({id:3,username:'wangwu',age:22})
db.user.insert({id:4,username:'zhaoliu',age:22})
```

```
> db.user.remove({age:22},true)
WriteResult({ "nRemoved" : 1 })  #删除了1条数据

#删除所有数据
> db.user.remove({})

#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。
db.user.deleteOne({id:1})
db.user.deleteMany({})  #删除所有数据

```

1.3.6、查询数据

MongoDB 查询数据的语法格式如下：

```
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：

```
>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：

```
#插入测试数据
db.user.insert({id:1,username:'zhangsan',age:20})
db.user.insert({id:2,username:'lisi',age:21})
db.user.insert({id:3,username:'wangwu',age:22})
db.user.insert({id:4,username:'zhaoliu',age:22})

db.user.find()  #查询全部数据
db.user.find({},{id:1,username:1})  #只查询id与username字段
db.user.find().count()  #查询数据条数
db.user.find({id:1}) #查询id为1的数据
db.user.find({age:{$lte:21}}) #查询小于等于21的数据
db.user.find({age:{$lte:21}, id:{$gte:2}}) #and查询，age小于等于21并且id大于等于2
db.user.find({$or:[{id:1},{id:2}]}) #查询id=1 or id=2

#分页查询：Skip()跳过几条，limit()查询条数
db.user.find().limit(2).skip(1)  #跳过1条数据，查询2条数据

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，查询可以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

```
#查看索引
> db.user.getIndexes()
[
    {
        "v" : 2,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "testdb.user"
    }
]

#说明：1表示升序创建索引，-1表示降序创建索引。
```

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

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

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

```
#查看索引大小，单位：字节
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。

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

```
#查看执行计划
> db.user.find({age:{$gt:100},id:{$lt:200}}).explain()
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "testdb.user",
        "indexFilterSet" : false,
        "parsedQuery" : {
            "$and" : [
                {
                    "id" : {
                        "$lt" : 200
                    }
                },
                {
                    "age" : {
                        "$gt" : 100
                    }
                }
            ]
```

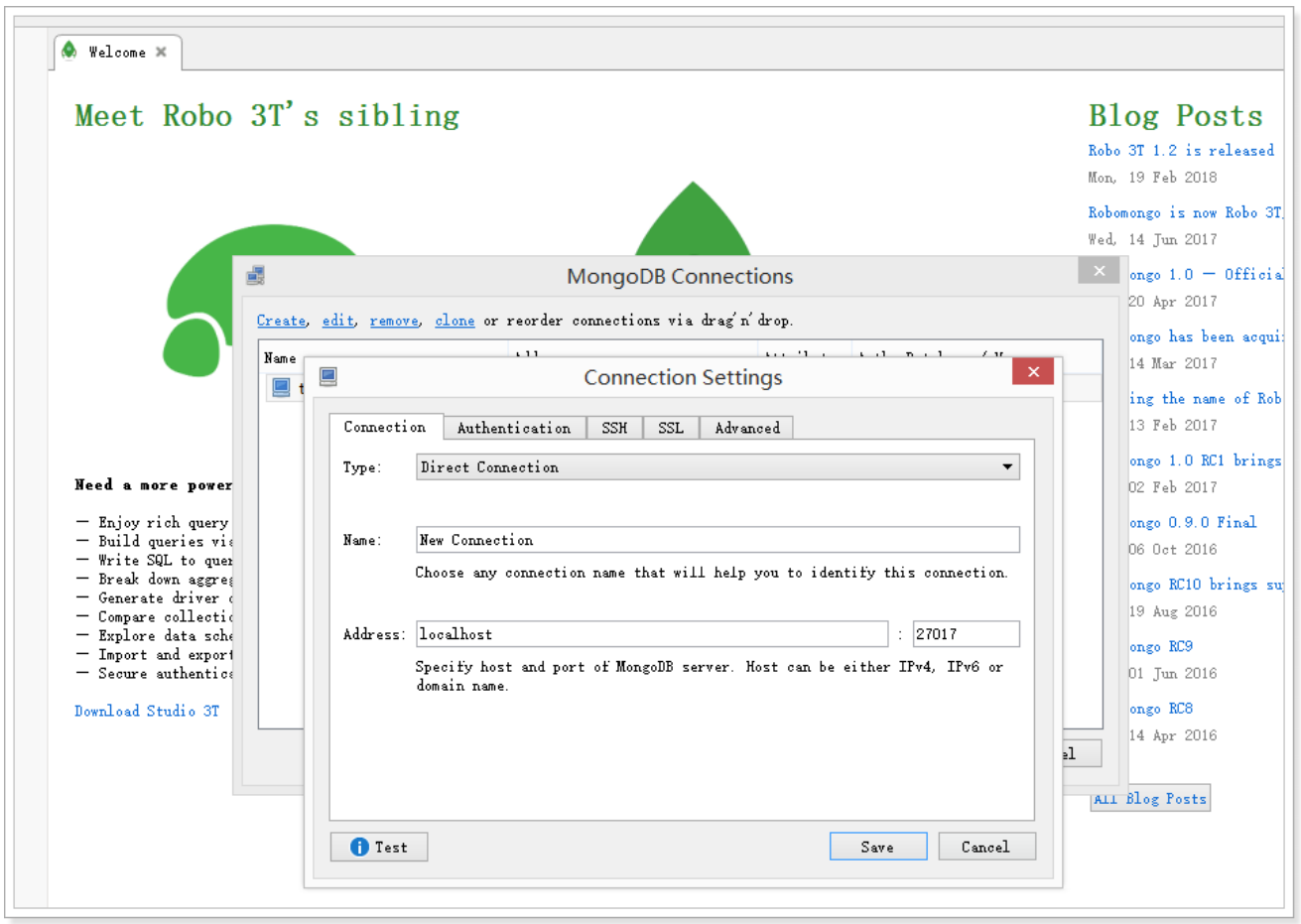
```
        },
        "winningPlan" : {  #最佳执行计划
            "stage" : "FETCH", #查询方式，常见的有COLLSCAN/全表扫描、IXSCAN/索引扫描、FETCH/根据索引去检索文档、SHARD_MERGE/合并分片结果、IDHACK/针对_id进行查询
            "inputStage" : {
                "stage" : "IXSCAN",
                "keyPattern" : {
                    "age" : 1,
                    "id" : -1
                },
                "indexName" : "age_1_id_-1",
                "isMultiKey" : false,
                "multiKeyPaths" : {
                    "age" : [ ],
                    "id" : [ ]
                },
                "isUnique" : false,
                "isSparse" : false,
                "isPartial" : false,
                "indexVersion" : 2,
                "direction" : "forward",
                "indexBounds" : {
                    "age" : [
                        "(100.0, inf.0]"
                    ],
                    "id" : [
                        "(200.0, -inf.0]"
                    ]
                }
            }
        },
        "rejectedPlans" : [ ]
    },
    "serverInfo" : {
        "host" : "c493d5ff750a",
        "port" : 27017,
        "version" : "4.0.3",
        "gitVersion" : "7ea530946fa7880364d88c8d8b6026bbc9ffa48c"
    },
    "ok" : 1
}
```

```
#测试没有使用索引
> db.user.find({username:'zhangsan'}).explain()
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "testdb.user",
        "indexFilterSet" : false,
        "parsedQuery" : {
            "username" : {
                "$eq" : "zhangsan"
```

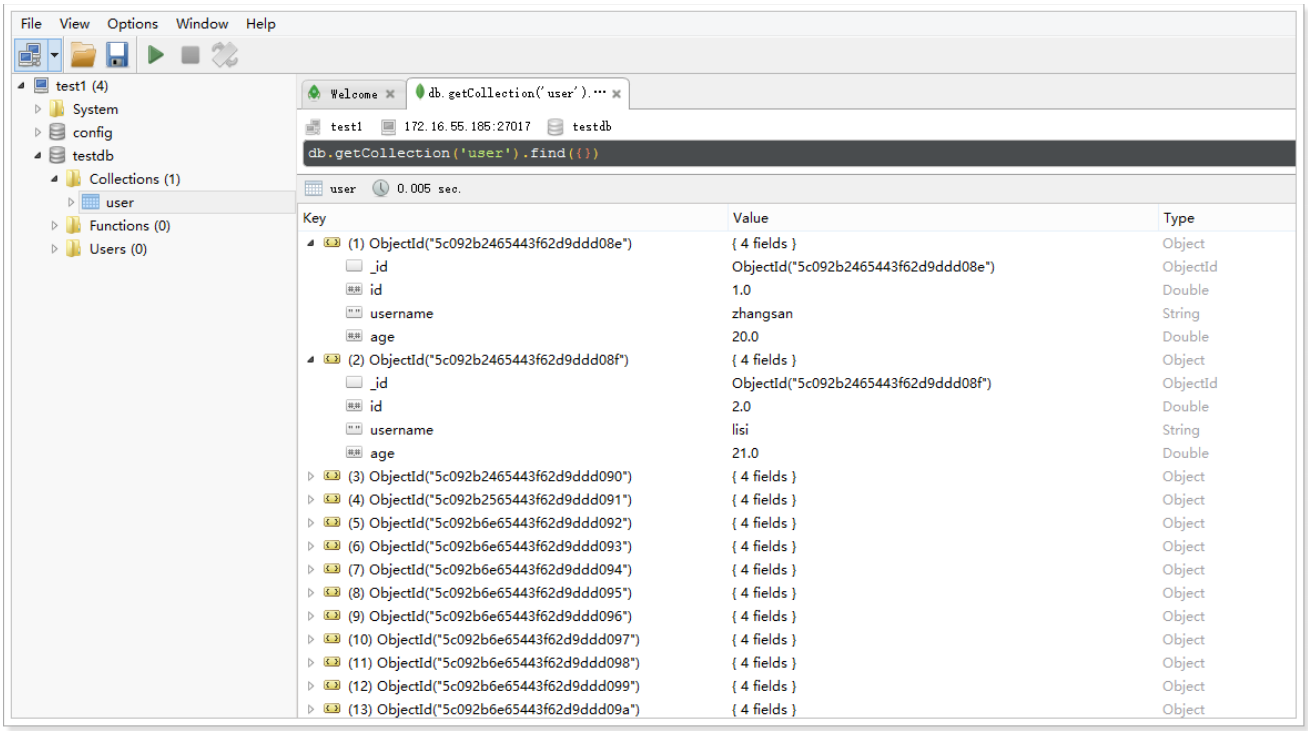
```
            }
        },
        "winningPlan" : {
            "stage" : "COLLSCAN",  #全表扫描
            "filter" : {
                "username" : {
                    "$eq" : "zhangsan"
                }
            },
            "direction" : "forward"
        },
        "rejectedPlans" : [ ]
    },
    "serverInfo" : {
        "host" : "c493d5ff750a",
        "port" : 27017,
        "version" : "4.0.3",
        "gitVersion" : "7ea530946fa7880364d88c8d8b6026bbc9ffa48c"
    },
    "ok" : 1
}
```

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：



2、通过JavaApi操作MongoDB

2.1、创建itcast-mongodb工程

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>cn.itcast.mongodb</groupId>
    <artifactId>itcast-mongodb</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.mongodb</groupId>
            <artifactId>mongodb-driver-sync</artifactId>
            <version>3.9.1</version>
        </dependency>
         <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
```

```
            <artifactId>lombok</artifactId>
            <version>1.18.4</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- java编译插件 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.2</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>

</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。

```
package cn.itcast.mongodb;

import com.mongodb.client.*;
import org.bson.Document;

import java.util.function.Consumer;

public class MongoDBDemo {

    public static void main(String[] args) {
        // 建立连接
        MongoClient mongoClient =
                MongoClients.create("mongodb://172.16.55.185:27017");

        // 选择数据库
        MongoDatabase mongoDatabase = mongoClient.getDatabase("testdb");

        // 选择表
        MongoCollection<Document> userCollection =
mongoDatabase.getCollection("user");

        // 查询数据
        userCollection.find().limit(10).forEach((Consumer<? super Document>)
document -> {
            System.out.println(document.toJson());
        });
```

```
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
```

```
        this.mongoCollection = mongoDatabase.getCollection("user");
    }

    // 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
    @Test
    public void testQuery() {
        this.mongoCollection.find(
                and(
                        lte("age", 50),
                        gte("id", 100)
                )
        )
                .sort(Sorts.descending("id"))
                .projection(
                        Projections.fields(
                                Projections.include("id","age"),
                                Projections.excludeId()
                        )
                )
                .forEach((Consumer<? super Document>) document -> {
                    System.out.println(document.toJson());
                });
        ;
    }

    @Test
    public void testInsert(){
        Document document = new Document("id",10001)
                .append("name", "张三")
                .append("age", 30);
        this.mongoCollection.insertOne(document);
        System.out.println("插入数据成功！");

        this.mongoCollection.find(eq("id", 10001))
                .forEach((Consumer<? super Document>) doc->{
                    System.out.println(doc.toJson());
        });
    }

    @Test
    public void testUpdate(){
        UpdateResult updateResult = this.mongoCollection
                .updateOne(eq("id", 10001), Updates.set("age", 25));
        System.out.println(updateResult);

        this.mongoCollection.find(eq("id", 10001))
                .forEach((Consumer<? super Document>) doc->{
                    System.out.println(doc.toJson());
                });
    }

    @Test
    public void testDelete(){
```

```
        DeleteResult deleteResult = this.mongoCollection.deleteOne(eq("id",
10001));
        System.out.println(deleteResult);
    }

}

```

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：

```
package cn.itcast.mongodb;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.bson.types.ObjectId;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Person {

    private ObjectId id;
    private String name;
    private int age;
    private Address address;

}

```

```
package cn.itcast.mongodb;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Address {

    private String street;
    private String city;
    private String zip;
}

```

编写测试用例：

```java
package cn.itcast.mongodb;

import com.mongodb.MongoClientSettings;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates;
import com.mongodb.client.result.DeleteResult;
import com.mongodb.client.result.UpdateResult;
import org.bson.codecs.configuration.CodecRegistries;
import org.bson.codecs.configuration.CodecRegistry;
import org.bson.codecs.pojo.PojoCodecProvider;
import org.bson.types.ObjectId;
import org.junit.Before;
import org.junit.Test;

import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;

public class TestPerson {

    MongoCollection<Person> personCollection;

    @Before
    public void init() {

        //定义对象的解码注册器
        CodecRegistry pojoCodecRegistry = CodecRegistries.
                fromRegistries(MongoClientSettings.getDefaultCodecRegistry(),
 CodecRegistries.fromProviders(PojoCodecProvider.builder().automatic(true).build())
);

        // 建立连接
        MongoClient mongoClient =
                MongoClients.create("mongodb://172.16.55.185:27017");

        // 选择数据库 并且 注册解码器
        MongoDatabase mongoDatabase = mongoClient.getDatabase("testdb")
                .withCodecRegistry(pojoCodecRegistry);

        // 选择表
        this.personCollection = mongoDatabase
                .getCollection("person", Person.class);

    }

    @Test
    public void testInsert() {
        Person person = new Person(ObjectId.get(), "张三", 20,
```

```
                new Address("人民路", "上海市", "666666"));
        this.personCollection.insertOne(person);
        System.out.println("插入数据成功");
    }

    @Test
    public void testInserts() {
        List<Person> personList = Arrays.asList(new Person(ObjectId.get(), "张三",
20, new Address("人民路", "上海市", "666666")),
                new Person(ObjectId.get(), "李四", 21, new Address("北京西路", "上海
市", "666666")),
                new Person(ObjectId.get(), "王五", 22, new Address("南京东路", "上海
市", "666666")),
                new Person(ObjectId.get(), "赵六", 23, new Address("陕西南路", "上海
市", "666666")),
                new Person(ObjectId.get(), "孙七", 24, new Address("南京西路", "上海
市", "666666")));
        this.personCollection.insertMany(personList);
        System.out.println("插入数据成功");
    }

    @Test
    public void testQuery() {
        this.personCollection.find(Filters.eq("name", "张三"))
                .forEach((Consumer<? super Person>) person -> {
                    System.out.println(person);
                });
    }

    @Test
    public void testUpdate() {
        UpdateResult updateResult =
this.personCollection.updateMany(Filters.eq("name", "张三"), Updates.set("age",
22));
        System.out.println(updateResult);
    }

    @Test
    public void testDelete() {
        DeleteResult deleteResult =
this.personCollection.deleteOne(Filters.eq("name", "张三"));
        System.out.println(deleteResult);
    }

}

```

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：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.0.RELEASE</version>
</parent>

<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

第二步，编写application.properties配置文件

```
# Spring boot application
spring.application.name = itcast-mongodb

spring.data.mongodb.uri=mongodb://172.16.55.185:27017/testdb

```

第三步，编写PersonDao

```
package cn.itcast.mongodb.dao;

import cn.itcast.mongodb.Person;
import com.mongodb.client.result.DeleteResult;
import com.mongodb.client.result.UpdateResult;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.core.query.Update;
import org.springframework.stereotype.Component;

import java.util.List;

@Component
public class PersonDao {

    @Autowired
    private MongoTemplate mongoTemplate;

    public void savePerson(Person person) {
        this.mongoTemplate.save(person);
    }

```

```
    public List<Person> queryPersonListByName(String name) {
        Query query = Query.query(Criteria.where("name").is(name));
        return this.mongoTemplate.find(query, Person.class);
    }

    public List<Person> queryPersonListByName(Integer page, Integer rows) {
        Query query = new Query().limit(rows).skip((page - 1) * rows);
        return this.mongoTemplate.find(query, Person.class);
    }

    public UpdateResult update(Person person) {
        Query query = Query.query(Criteria.where("id").is(person.getId()));
        Update update = Update.update("age", person.getAge());
        return this.mongoTemplate.updateFirst(query, update, Person.class);
    }

    public DeleteResult deleteById(String id) {
        Query query = Query.query(Criteria.where("id").is(id));
        return this.mongoTemplate.remove(query, Person.class);
    }
}

```

第四步，编写启动类

```
package cn.itcast.mongodb;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MongoApplication {

    public static void main(String[] args) {
        SpringApplication.run(MongoApplication.class, args);
    }
}

```

第五步，编写单元测试

```
package cn.itcast.mongodb;

import cn.itcast.mongodb.dao.PersonDao;
import org.bson.types.ObjectId;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

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：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.0.RELEASE</version>
    </parent>

    <groupId>cn.itcast.haoke.im</groupId>
    <artifactId>itcast-haoke-im</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-websocket</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mongodb</groupId>
            <artifactId>mongodb-driver-sync</artifactId>
            <version>3.9.1</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
```

```
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.4</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- java编译插件 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.2</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

4.2、编写Message对象

```
package cn.itcast.haoke.im.pojo;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.bson.types.ObjectId;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.data.mongodb.core.mapping.Field;

import java.util.Date;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Document(collection = "message")
@Builder
public class Message {

```

```
    @Id
    private ObjectId id;
    private String msg;
    /**
     * 消息状态，1-未读，2-已读
     */
    @Indexed
    private Integer status;
    @Field("send_date")
    @Indexed
    private Date sendDate;
    @Field("read_date")
    private Date readDate;
    @Indexed
    private User from;
    @Indexed
    private User to;

}

```

```
package cn.itcast.haoke.im.pojo;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class User {

    private Long id;
    private String username;

}
```

构造用户数据：

```
package cn.itcast.haoke.im.pojo;

import java.util.HashMap;
import java.util.Map;

public class UserData {

    public static final Map<Long,User> USER_MAP = new HashMap<>();

    static {
        USER_MAP.put(1001L, User.builder().id(1001L).username("zhangsan").build());
```

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

4.3、编写MessageDAO

定义MessageDAO接口：

```
package cn.itcast.haoke.im.dao;

import cn.itcast.haoke.im.pojo.Message;
import com.mongodb.client.result.DeleteResult;
import com.mongodb.client.result.UpdateResult;
import org.bson.types.ObjectId;

import java.util.List;

public interface MessageDAO {

    /**
     * 查询点对点聊天记录
     *
     * @param fromId
     * @param toId
     * @param page
     * @param rows
     * @return
     */
    List<Message> findListByFromAndTo(Long fromId, Long toId, Integer page, Integer
rows);

    /**
     * 根据id查询数据
     *
     * @param id
     * @return
     */
    Message findMessageById(String id);

    /**
     * 更新消息状态
     *
     * @param id
     * @param status
     * @return
     */
    UpdateResult updateMessageState(ObjectId id, Integer status);

    /**
     * 新增消息数据
```

```
     *
     * @param message
     * @return
     */
    Message saveMessage(Message message);

    /**
     * 根据消息id删除数据
     *
     * @param id
     * @return
     */
    DeleteResult deleteMessage(String id);
}
```

编写实现类：

```
package cn.itcast.haoke.im.dao.impl;

import cn.itcast.haoke.im.dao.MessageDAO;
import cn.itcast.haoke.im.pojo.Message;
import cn.itcast.haoke.im.pojo.User;
import com.mongodb.client.result.DeleteResult;
import com.mongodb.client.result.UpdateResult;
import org.bson.types.ObjectId;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.core.query.Update;
import org.springframework.stereotype.Component;

import java.util.Date;
import java.util.List;

@Component
public class MessageDAOImpl implements MessageDAO {

    @Autowired
    private MongoTemplate mongoTemplate;

    @Override
    public List<Message> findListByFromAndTo(Long fromId, Long toId, Integer page,
Integer rows) {

        Criteria fromList =
Criteria.where("from.id").is(fromId).and("to.id").is(toId);
        Criteria toList =
Criteria.where("from.id").is(toId).and("to.id").is(fromId);

        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

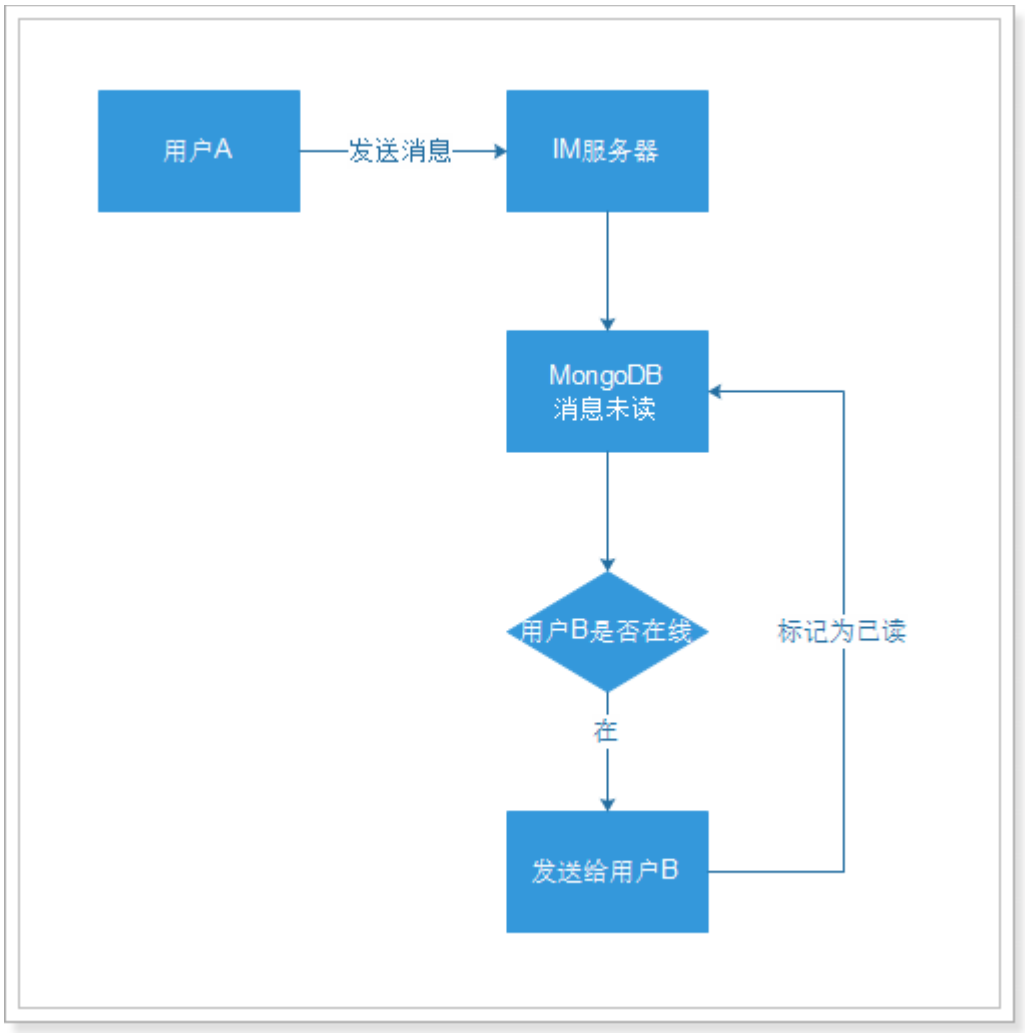
```
1  package cn.itcast.haoke.im.dao;
2
3  import cn.itcast.haoke.im.pojo.Message;
4  import cn.itcast.haoke.im.pojo.User;
5  import org.bson.types.ObjectId;
6  import org.junit.Test;
7  import org.junit.runner.RunWith;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import java.util.Date;
import java.util.List;

@RunWith(SpringRunner.class)
@SpringBootTest
public class TestMessageDAO {

    @Autowired
    private MessageDAO messageDAO;

    @Test
    public void testSave(){
        Message message = Message.builder()
                .id(ObjectId.get())
                .msg("你好")
                .sendDate(new Date())
                .status(1)
                .from(new User(1001L, "zhangsan"))
                .to(new User(1002L,"lisi"))
                .build();
        this.messageDAO.saveMessage(message);

        message = Message.builder()
                .id(ObjectId.get())
                .msg("你也好")
                .sendDate(new Date())
                .status(1)
                .to(new User(1001L, "zhangsan"))
                .from(new User(1002L,"lisi"))
                .build();
        this.messageDAO.saveMessage(message);

        message = Message.builder()
                .id(ObjectId.get())
                .msg("我在学习开发IM")
                .sendDate(new Date())
                .status(1)
                .from(new User(1001L, "zhangsan"))
                .to(new User(1002L,"lisi"))
                .build();
        this.messageDAO.saveMessage(message);

        message = Message.builder()
                .id(ObjectId.get())
                .msg("那很好啊！")
                .sendDate(new Date())
                .status(1)
                .to(new User(1001L, "zhangsan"))
                .from(new User(1002L,"lisi"))
```

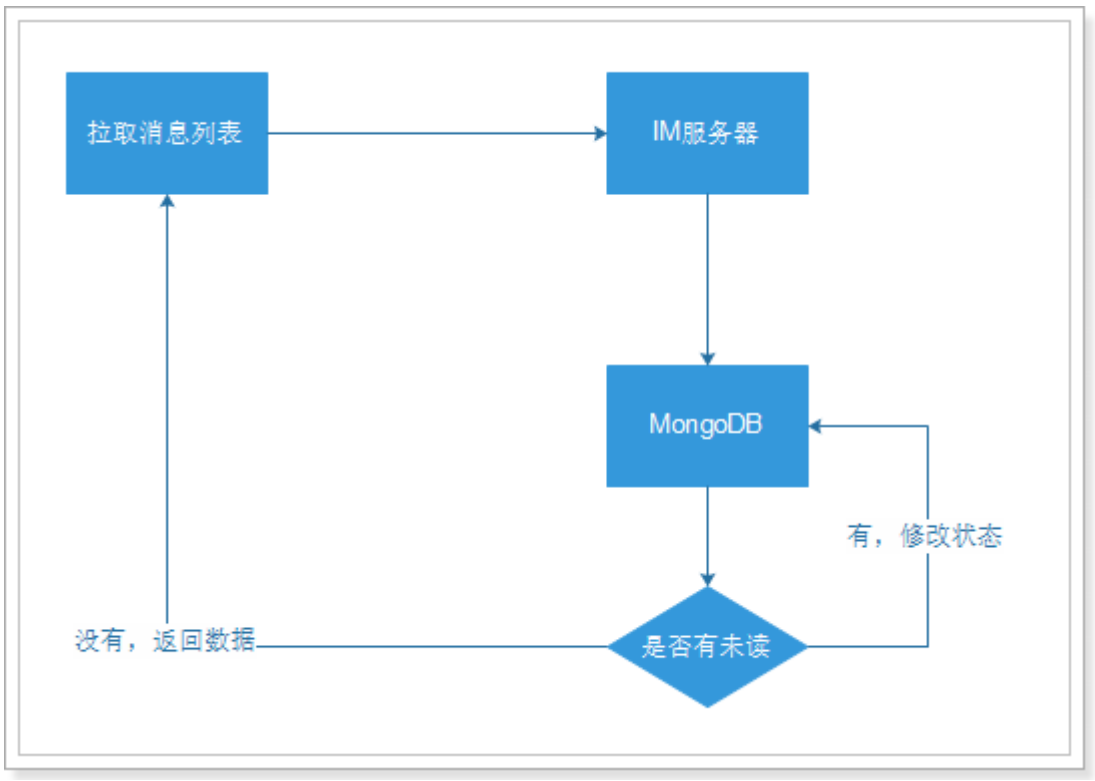
```
61                  .build();
62          this.messageDAO.saveMessage(message);
63
64          System.out.println("ok");
65      }
66
67      @Test
68      public void testQueryById(){
69          Message message =
this.messageDAO.findMessageById("5c0a9109235e193bd4873b92");
70          System.out.println(message);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

4.5、编写websocket

4.5.1、发送消息流程


用户A
发送消息
IM服务器
MongoDB
消息未读
用户B是否在线
标记为已读
在
发送给用户B

4.5.2、接收消息流程


拉取消息列表
IM服务器
MongoDB
有，修改状态
没有，返回数据
是否有未读

4.5.3、实现

```
package cn.itcast.haoke.im.websocket;

import cn.itcast.haoke.im.dao.MessageDAO;
import cn.itcast.haoke.im.pojo.Message;
import cn.itcast.haoke.im.pojo.UserData;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.web.socket.TextMessage;
import org.springframework.web.socket.WebSocketSession;
import org.springframework.web.socket.handler.TextWebSocketHandler;

import java.util.HashMap;
import java.util.Map;

@Component
public class MessageHandler extends TextWebSocketHandler {

    @Autowired
    private MessageDAO messageDAO;

    private static final ObjectMapper MAPPER = new ObjectMapper();

    private static final Map<Long,WebSocketSession> SESSIONS = new HashMap<>();

    @Override
    public void afterConnectionEstablished(WebSocketSession session) throws
            Exception {
        Long uid = (Long)session.getAttributes().get("uid");
        // 将当前用户的session放置到map中，后面会使用相应的session通信
        SESSIONS.put(uid, session);
    }


    @Override
    protected void handleTextMessage(WebSocketSession session, TextMessage
textMessage) throws Exception {
        Long uid = (Long)session.getAttributes().get("uid");

        JsonNode jsonNode = MAPPER.readTree(textMessage.getPayload());
        Long toId = jsonNode.get("toId").asLong();
        String msg = jsonNode.get("msg").asText();

        Message message = Message.builder()
                .from(UserData.USER_MAP.get(uid))
                .to(UserData.USER_MAP.get(toId))
                .msg(msg)
                .build();

        // 将消息保存到MongoDB
```

```
        message = this.messageDAO.saveMessage(message);

        // 判断to用户是否在线
        WebSocketSession toSession = SESSIONS.get(toId);
        if(toSession != null && toSession.isOpen()){
            //TODO 具体格式需要和前端对接
            toSession.sendMessage(new
TextMessage(MAPPER.writeValueAsString(message)));

            // 更新消息状态为已读
            this.messageDAO.updateMessageState(message.getId(), 2);
        }

    }
}
```

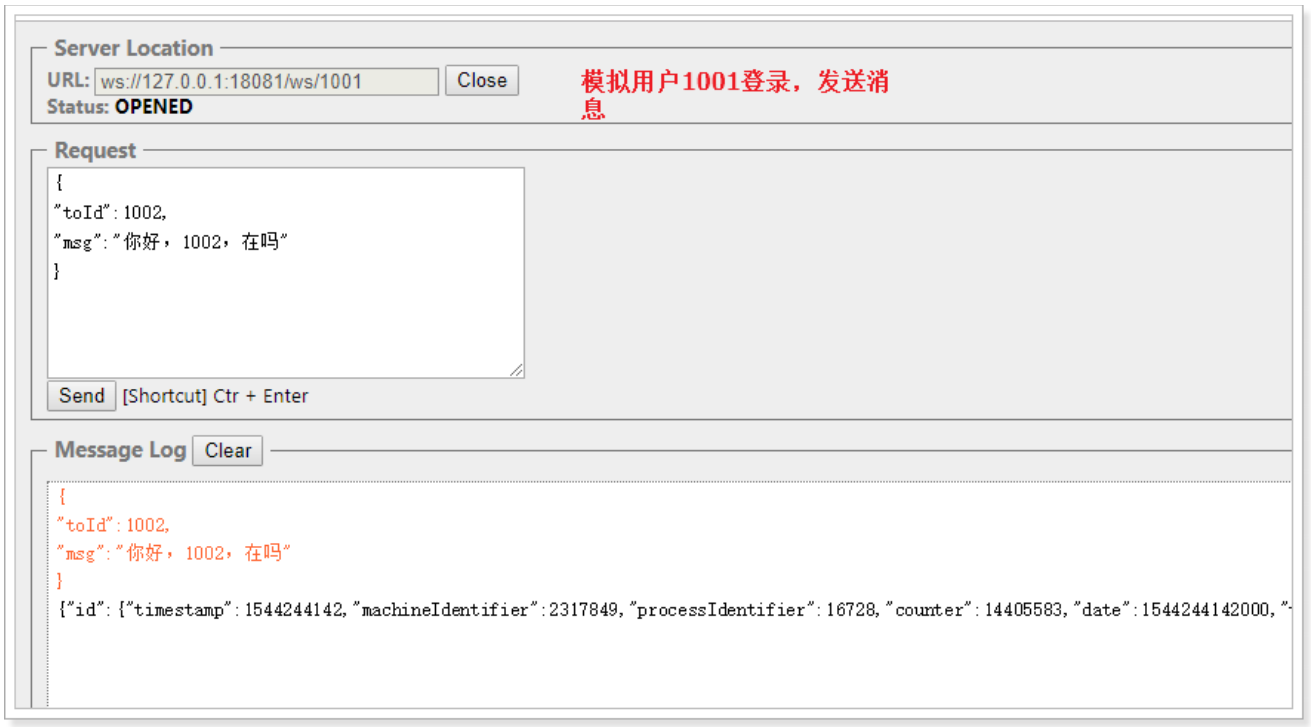
```
package cn.itcast.haoke.im.websocket;

import org.apache.commons.lang3.StringUtils;
import org.springframework.http.server.ServerHttpRequest;
import org.springframework.http.server.ServerHttpResponse;
import org.springframework.stereotype.Component;
import org.springframework.web.socket.WebSocketHandler;
import org.springframework.web.socket.server.HandshakeInterceptor;

import java.util.Map;

@Component
public class MessageHandshakeInterceptor implements HandshakeInterceptor {

    @Override
    public boolean beforeHandshake(ServerHttpRequest request, ServerHttpResponse
response, WebSocketHandler wsHandler, Map<String, Object> attributes) throws
Exception {
        String path = request.getURI().getPath();
        String[] ss = StringUtils.split(path, '/');
        if(ss.length != 2){
            return false;
        }
        if(!StringUtils.isNumeric(ss[1])){
            return false;
        }
        attributes.put("uid", Long.valueOf(ss[1]));
        return true;
    }

    @Override
    public void afterHandshake(ServerHttpRequest request, ServerHttpResponse
response, WebSocketHandler wsHandler, Exception exception) {

    }

}
```

```java
package cn.itcast.haoke.im.websocket;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.socket.config.annotation.EnableWebSocket;
import org.springframework.web.socket.config.annotation.WebSocketConfigurer;
import org.springframework.web.socket.config.annotation.WebSocketHandlerRegistry;

@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

    @Autowired
    private MessageHandler messageHandler;

    @Autowired
    private MessageHandshakeInterceptor messageHandshakeInterceptor;

    @Override
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
        registry.addHandler(this.messageHandler, "/ws/{uid}")
                .setAllowedOrigins("*")
                .addInterceptors(this.messageHandshakeInterceptor);
    }
}
```

4.6、编写启动类

```java
package cn.itcast.haoke.im;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ImApplication {

    public static void main(String[] args) {
        SpringApplication.run(ImApplication.class, args);
    }

}

```

4.7、测试



4.8、提供查询历史消息的服务

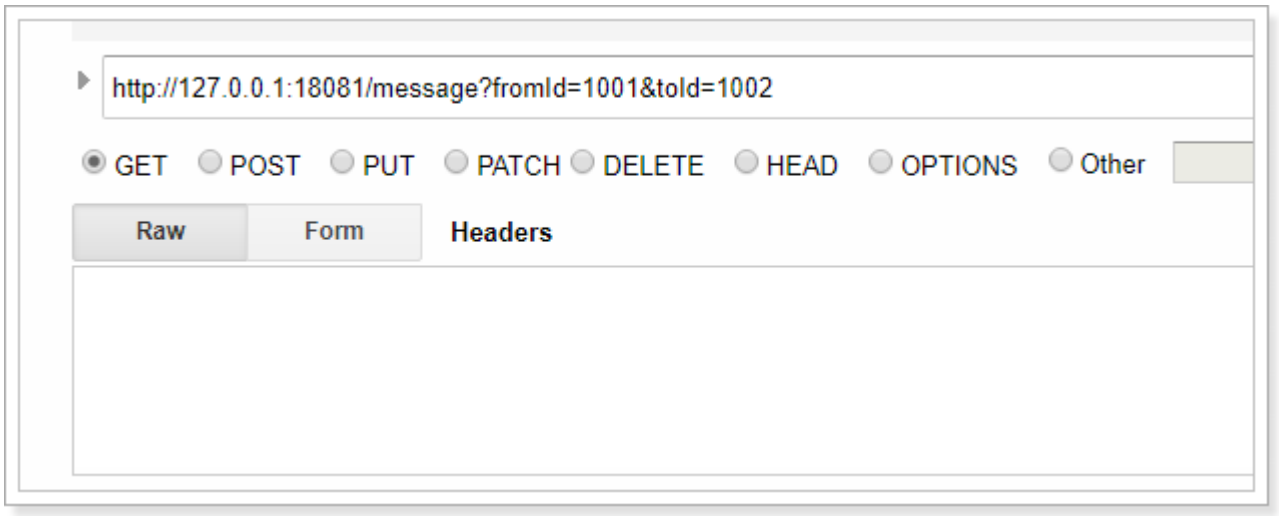
```
1  package cn.itcast.haoke.im.controller;
2
3  import cn.itcast.haoke.im.pojo.Message;
4  import cn.itcast.haoke.im.service.MessageService;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.web.bind.annotation.*;
7
8  import java.util.List;
```

```
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",
    defaultValue = "1") Integer page,
32                                            @RequestParam(value = "rows",
    defaultValue = "10") Integer rows) {
33          return this.messageService.queryMessageList(fromId, toId, page, rows);
34      }
35
36  }
```

```
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,
    Integer rows) {
17          List<Message> list = this.messageDAO.findListByFromAndTo(fromId, toId,
    page, rows);
18          for (Message message : list) {
19              if(message.getStatus().intValue() == 1){
20                  // 修改消息状态为已读
```

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

测试：



4.9、提供查询用户列表服务（mock实现）

数据结构：

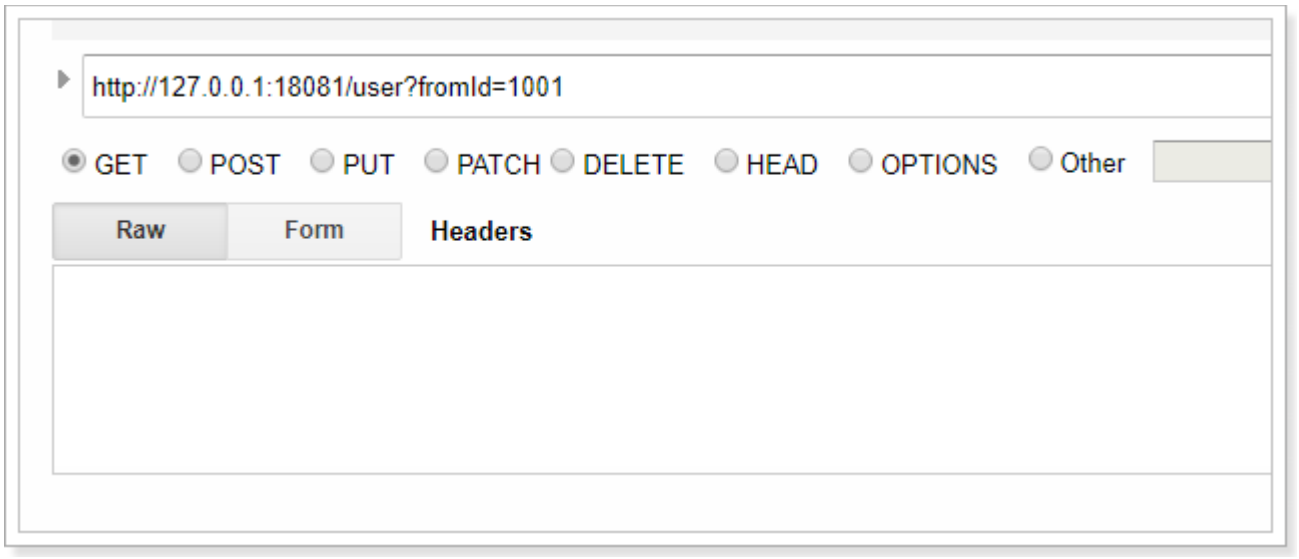
```
{
    "data": {
        "list": [
            {
                "id": 3,
                "from_user": 1,
                "to_user": 3,
                "chat_time": 1531045313395,
                "chat_msg": "12213123",
                "info_type": null,
                "username": "spike",
                "avatar": "public/icon.png"
            },
```

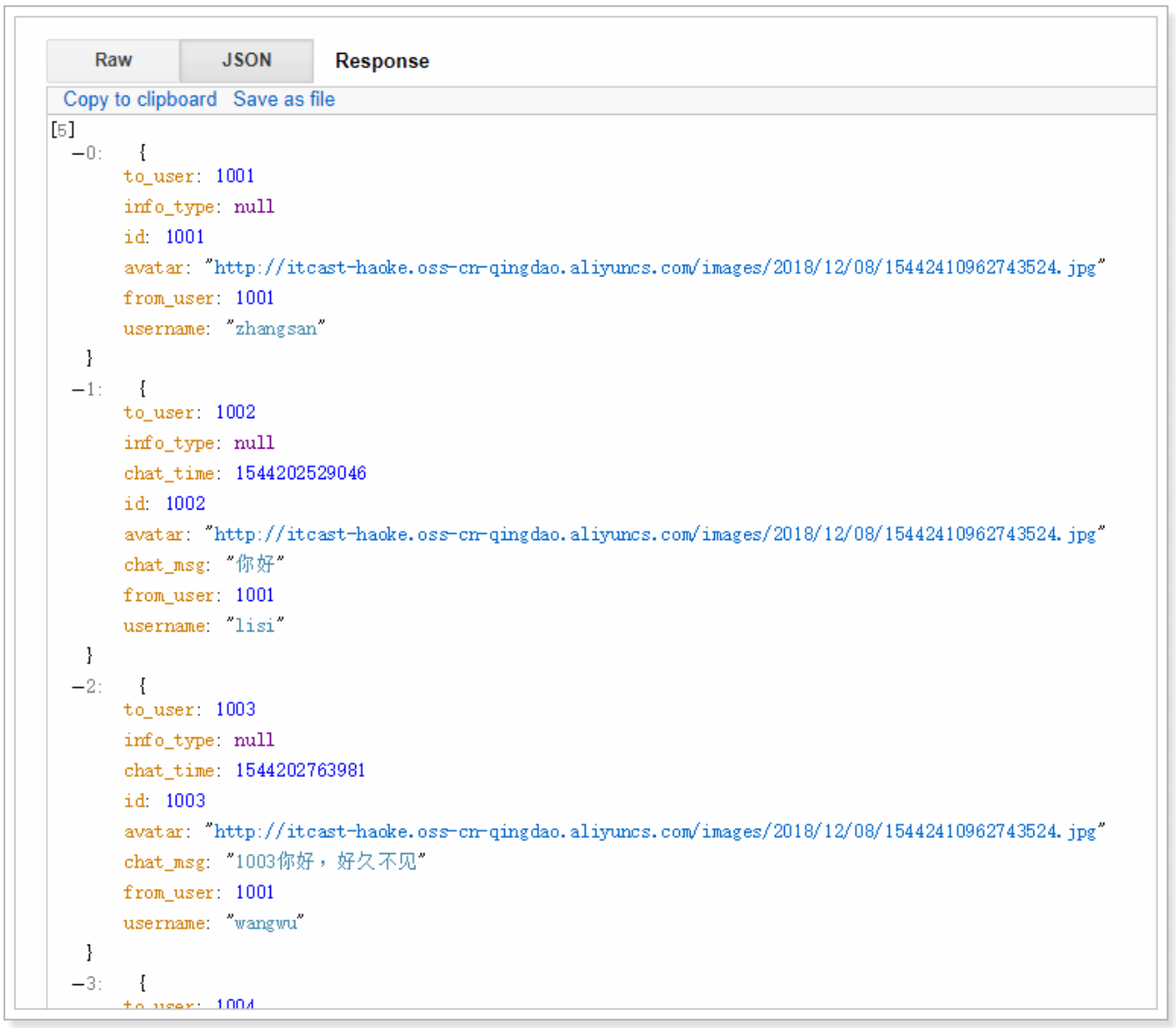
```
            {
                "id": 4,
                "from_user": 1,
                "to_user": 3,
                "chat_time": 1531045313395,
                "chat_msg": "12",
                "info_type": null,
                "username": "spike",
                "avatar": "public/icon.png"
            },
            {
                "id": 5,
                "from_user": 1,
                "to_user": 3,
                "chat_time": 1531045313395,
                "chat_msg": "12",
                "info_type": null,
                "username": "spike",
                "avatar": "public/icon.png"
            },
            {
                "id": 6,
                "from_user": 1,
                "to_user": 3,
                "chat_time": 1531045313395,
                "chat_msg": "1212",
                "info_type": null,
                "username": "spike",
                "avatar": "public/icon.png"
            }
        ]
    },
    "meta": {
        "status": 200,
        "msg": "用户数据"
    }
}
```

```
package cn.itcast.haoke.im.controller;

import cn.itcast.haoke.im.pojo.Message;
import cn.itcast.haoke.im.pojo.User;
import cn.itcast.haoke.im.pojo.UserData;
import cn.itcast.haoke.im.service.MessageService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
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：





5、整合前端实现微聊功能

5.1、查询好友列表

chat.js：

```
componentDidMount = () => {
  axios.get('http://127.0.0.1:18081/user?fromId=1001').then((data)=>{
    this.setState({
      list: data,
      isLoading: true
    })
  })
}

--------------------
if (isLoading) {
    list = this.state.list.map(item => {
      return (
        <li key={item.id} onClick={(e) => this.toChat(e,{item})}>
          <div className="avarter">
            <img src={item.avatar} alt="avarter"/>
            <span className="name">{item.username}</span>
            <span className="info">{item.chat_msg}</span>
            <span className="time">{item.ctime}</span>
          </div>
        </li>
      )
    })
  }
```

效果：

5.2、好友单聊

chat-window.js文件：

```
componentDidMount = () => {
    let {to_user,from_user} = this.props.chatInfo;
    axios.get('http://127.0.0.1:18081/message',{params:{
        toId: to_user,
        fromId: from_user
    }}).then(data=>{
      this.setState({
        infos: data,
        isLoading: true,
```

```
            fromId: from_user,
          client: handle(localStorage.getItem('uid'),(data)=>{
            let newList = [...this.state.infos];
            newList.push(JSON.parse(data.content));
            this.setState({
              infos: newList
            })
          })
        });
      })
    }

if(this.state.isLoading) {
      // let currentUser = parseInt(localStorage.getItem('uid'),10);
      let currentUser = parseInt(this.state.fromId,10);
      infoList = this.state.infos.map(item=>{
        return (
          <li key={item.id} className={currentUser===item.from.id? 'chat-info-right':'chat-info-left'}>
            <img src="http://itcast-haoke.oss-cn-qingdao.aliyuncs.com/images/2018/12/08/15442410962743524.jpg" alt=""/>
            <span>{item.msg}</span>
          </li>
        )
      })
    }
```

效果：

5.3、发送消息

chat-window.js：

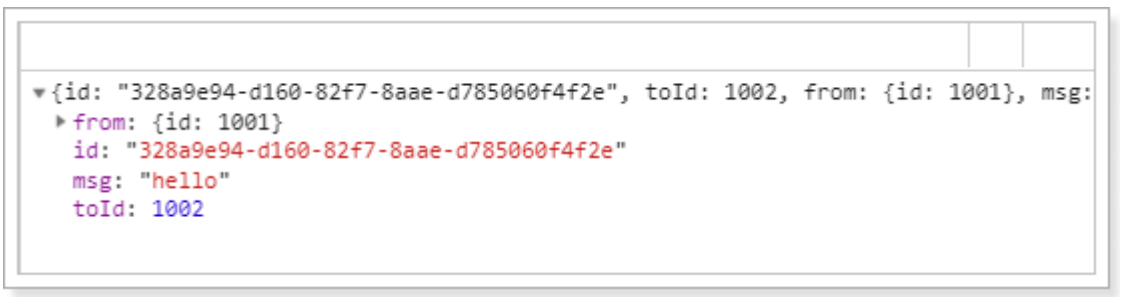
```
sendMsg = () => {
  let {to_user,from_user,avatar} = this.props.chatInfo;
  let pdata = {
    id: this.guid(),
    // from_user: from_user,
    toId: to_user,
    from:{
      id:this.state.fromId
    },
```

```
      // avatar: avatar,
      msg: this.state.msgContent
    }
    let newList = [...this.state.infos];
    newList.push(pdata);
    this.setState({
      infos: newList
    })

    this.state.client.emitEvent(IMEvent.MSG_TEXT_SEND,JSON.stringify(pdata));
  }
```

修改IMClient.js，不包装，直接发送数据：

```
// 向服务器发送数据包
sendDataPacket(dataPacket) {
  // if (this._isOpened) {
  //   this._socket.send(dataPacket.rawMessage);
  // } else {
  //   this._DataPacketQueue.push(dataPacket);
  // }
    // 直接发送，不包装
    this._socket.send(dataPacket);
}
```

测试：



5.4、接收消息

chat-window.js中，注册接收消息后的处理逻辑：

```
componentDidMount = () => {
  let {to_user,from_user} = this.props.chatInfo;
  axios.get('http://127.0.0.1:18081/message',{params:{
      toId: to_user,
      fromId: from_user
  }}).then(data=>{
    this.setState({
      infos: data,
      isLoading: true,
        fromId: from_user,
      client: handle(from_user,(data)=>{
        let newList = [...this.state.infos];
```

```
        newList.push(JSON.parse(data));
        this.setState({
          infos: newList
        })
      })
    });
  })
}
```

修改wsmain.js文件：

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

修改IMClient.js文件：

```
constructor(url, onMyMessage) {
    this._url = url;
    this._autoConnect = true;
    this._handlers = {};
    this._DataPacketQueue = [];
    this._isOpened = false;
    this.onMyMessage = onMyMessage;

    this.addEventListener(IMEvent.CONNECTED, () => {
      this.serverOnConnected();
    })

    // this.addEventListener(IMEvent.CONNECTED, () => {
    //   this.clearMsgQueue();
    // })

    this.addEventListener(IMEvent.DISCONNECTED, () => {
      this.serverOnDisconnected();
    })
  }

```

```
connect() {
  if (!this._socket) {
    this._socket = new WebSocket(this._url);

    this._socket.onmessage = (evt) => {
      this.onMessage(evt.data);
      if(this.onMyMessage){
          this.onMyMessage(evt.data);
      }
    }
    this._socket.onopen = (ws) => {
      this.onOpen(ws);
    }
    this._socket.onclose = ws => {
```

```
      this.onClose(ws);
    }
    this._socket.onerror = ws => {
      this.onError(ws);
    };
  }
}
```

测试：

发送消息：

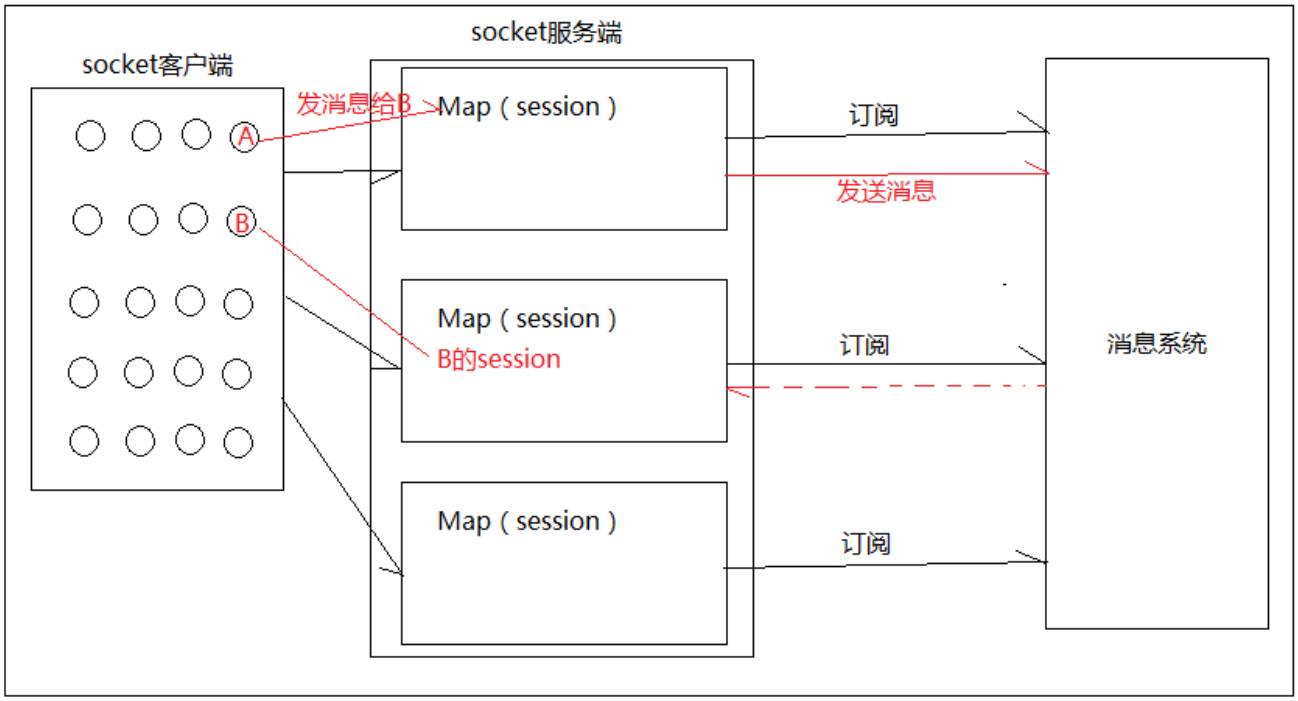
接收到消息：

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，这将成为了系统瓶颈，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。


socket服务端
socket客户端
发消息给B
Map (session)
订阅
发送消息
A
B
Map (session)
B的session
订阅
消息系统
Map (session)
订阅