课程介绍

- RocketMQ的错误消息重试策略
- RocketMQ的集群搭建
- SpringBoot整合RocketMQ
- 实现分布式的WebSocket
- 实现地图找房功能

1、重试策略

在消息的发送和消费过程中，都有可能出现错误，如网络异常等，出现了错误就需要进行错误重试，这种消息的重试需要分2种，分别是producer端重试和consumer端重试。

1.1、producer端重试

生产者端的消息失败，也就是Producer往MQ上发消息没有发送成功，比如网络抖动导致生产者发送消息到MQ失败。

```
package cn.itcast.rocketmq;

import org.apache.rocketmq.client.producer.DefaultMQProducer;
import org.apache.rocketmq.client.producer.SendResult;
import org.apache.rocketmq.common.message.Message;
import org.apache.rocketmq.remoting.common.RemotingHelper;

public class SyncProducer {
    public static void main(String[] args) throws Exception {
        DefaultMQProducer producer = new DefaultMQProducer("HAOKE_IM");
        producer.setNamesrvAddr("172.16.55.185:9876");
        //消息发送失败时，重试3次
        producer.setRetryTimesWhenSendFailed(3);
        producer.start();

        String msgStr = "用户A发送消息给用户B";
        Message msg = new Message("haoke_im_topic","SEND_MSG",
                msgStr.getBytes(RemotingHelper.DEFAULT_CHARSET));

        // 发送消息,并且指定超时时间
        SendResult sendResult = producer.send(msg, 1000);

        System.out.println("消息状态：" + sendResult.getSendStatus());
        System.out.println("消息id：" + sendResult.getMsgId());
        System.out.println("消息queue：" + sendResult.getMessageQueue());
        System.out.println("消息offset：" + sendResult.getQueueOffset());

        System.out.println(sendResult);

        producer.shutdown();
    }
```

```
}

```

1.2、consumer端重试

消费者端的失败，分为2种情况，一个是exception，一个是timeout。

1.2.1、exception

消息正常的到了消费者，结果消费者发生异常，处理失败了。例如反序列化失败，消息数据本身无法处理（例如话费充值，当前消息的手机号被注销，无法充值）等。

消息的状态：

```
package org.apache.rocketmq.client.consumer.listener;

public enum ConsumeConcurrentlyStatus {
    /**
     * Success consumption
     */
    CONSUME_SUCCESS,
    /**
     * Failure consumption,later try to consume
     */
    RECONSUME_LATER;
}
```

可以看到，消息的状态分为成功或者失败。如果返回的状态为失败会怎么样呢？

在启动broker的日志中可以看到这样的信息：

```
INFO main - messageDelayLevel=1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h
2h
```

这个表示了，如果消息消费失败，那么消息将会在1s、5s、10s后重试，一直到2h后不再重试。

其实，有些时候并不需要重试这么多次，一般重试3~5次即可。这个时候就可以通过msg.getReconsumeTimes()获取重试次数进行控制。

```
package cn.itcast.rocketmq;

import org.apache.rocketmq.client.consumer.DefaultMQPushConsumer;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;
import org.apache.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;
import org.apache.rocketmq.client.consumer.listener.MessageListenerConcurrently;
import org.apache.rocketmq.client.exception.MQClientException;
import org.apache.rocketmq.common.message.MessageExt;
import org.apache.rocketmq.common.protocol.heartbeat.MessageModel;

import java.io.UnsupportedEncodingException;
import java.util.List;
```

```
13
14  public class ConsumerDemo {
15
16      public static void main(String[] args) throws Exception {
17          DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("HAOKE_IM");
18          consumer.setNamesrvAddr("172.16.55.185:9876");
19
20          // 订阅topic，接收此Topic下的所有消息
21          consumer.subscribe("my-test-topic", "*");
22
23          consumer.registerMessageListener(new MessageListenerConcurrently() {
24              @Override
25              public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs,
    ConsumeConcurrentlyContext context) {
26
27                  for (MessageExt msg : msgs) {
28                      try {
29                          System.out.println(new String(msg.getBody(), "UTF-8"));
30                      } catch (UnsupportedEncodingException e) {
31                          e.printStackTrace();
32                      }
33                  }
34                  System.out.println("收到消息->" + msgs);
35
36                  if(msgs.get(0).getReconsumeTimes() >= 3){
37                      // 重试3次后，不再进行重试
38                      return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
39                  }
40
41                  return ConsumeConcurrentlyStatus.RECONSUME_LATER;
42              }
43          });
44
45          consumer.start();
46
47
48      }
49  }
50
```

1.2.2、timeout

比如由于网络原因导致消息压根就没有从MQ到消费者上，那么在RocketMQ内部会不断的尝试发送这条消息，直至发送成功为止！

也就是说，服务端没有接收到消息的反馈，既不是成功也不是失败，这个时候定义为超时。

2、RocketMQ的集群

2.1、集群模式

在RocketMQ中，集群的部署模式是比较多的，有以下几种：

- 单个Master
  - 这种方式风险较大，一旦Broker重启或者宕机时，会导致整个服务不可用，不建议线上环境使用。
- 多Master模式
  - 一个集群无Slave，全是Master，例如2个Master或者3个Master
  - 单台机器宕机期间，这台机器上未被消费的消息在机器恢复之前不可订阅，消息实时性会受到影响。
- 多Master多Slave模式，异步复制
  - 每个Master配置一个Slave，有多对Master-Slave，HA采用异步复制方式，主备有短暂消息延迟，毫秒级。
  - 优点：即使磁盘损坏，消息丢失的非常少，且消息实时性不会受影响，因为Master宕机后，消费者仍然可以从Slave消费，此过程对应用透明，不需要人工干预。性能同多Master模式几乎一样。
  - 缺点：Master宕机，磁盘损坏情况，会丢失少量消息。
- 多Master多Slave模式，同步双写
  - 每个Master配置一个Slave，有多对Master-Slave，HA采用同步双写方式，主备都写成功，向应用返回成功。
  - 优点：数据与服务都无单点，Master宕机情况下，消息无延迟，服务可用性与数据可用性都非常高。
  - 缺点：性能比异步复制模式略低，大约低10%左右。

2.2、搭建2m2s集群

下面通过docker搭建2master+2slave的集群。

```
#创建2个master
#nameserver1
docker create -p 9876:9876 --name rmqserver01 \
-e "JAVA_OPT_EXT=-server -Xms128m -Xmx128m -Xmn128m" \
-e "JAVA_OPTS=-Duser.home=/opt"  \
-v /haoke/rmq/rmqserver01/logs:/opt/logs \
-v /haoke/rmq/rmqserver01/store:/opt/store \
foxiswho/rocketmq:server-4.3.2

#nameserver2
docker create -p 9877:9876 --name rmqserver02 \
-e "JAVA_OPT_EXT=-server -Xms128m -Xmx128m -Xmn128m" \
-e "JAVA_OPTS=-Duser.home=/opt"  \
-v /haoke/rmq/rmqserver02/logs:/opt/logs \
-v /haoke/rmq/rmqserver02/store:/opt/store \
foxiswho/rocketmq:server-4.3.2
```

```
#创建第1个master broker
#master broker01
docker create --net host --name rmqbroker01 \
-e "JAVA_OPTS=-Duser.home=/opt"  \
-e "JAVA_OPT_EXT=-server -Xms128m -Xmx128m -Xmn128m" \
-v /haoke/rmq/rmqbroker01/conf/broker.conf:/etc/rocketmq/broker.conf \
-v /haoke/rmq/rmqbroker01/logs:/opt/logs \
-v /haoke/rmq/rmqbroker01/store:/opt/store \
foxiswho/rocketmq:broker-4.3.2

#配置
```

```
namesrvAddr=172.16.55.185:9876;172.16.55.185:9877
brokerClusterName=ItcastCluster
brokerName=broker01
brokerId=0
deleteWhen=04
fileReservedTime=48
brokerRole=SYNC_MASTER
flushDiskType=ASYNC_FLUSH
brokerIP1=172.16.55.185
brokerIp2=172.16.55.185
listenPort=10911
```

```
#创建第2个master broker
#master broker02
docker create --net host --name rmqbroker02 \
-e "JAVA_OPTS=-Duser.home=/opt"  \
-e "JAVA_OPT_EXT=-server -Xms128m -Xmx128m -Xmn128m" \
-v /haoke/rmq/rmqbroker02/conf/broker.conf:/etc/rocketmq/broker.conf \
-v /haoke/rmq/rmqbroker02/logs:/opt/logs \
-v /haoke/rmq/rmqbroker02/store:/opt/store \
foxiswho/rocketmq:broker-4.3.2

#master broker02
namesrvAddr=172.16.55.185:9876;172.16.55.185:9877
brokerClusterName=ItcastCluster
brokerName=broker02
brokerId=0
deleteWhen=04
fileReservedTime=48
brokerRole=SYNC_MASTER
flushDiskType=ASYNC_FLUSH
brokerIP1=172.16.55.185
brokerIp2=172.16.55.185
listenPort=10811
```

```
#创建第1个slave broker
#slave broker01
docker create --net host --name rmqbroker03 \
-e "JAVA_OPTS=-Duser.home=/opt"  \
-e "JAVA_OPT_EXT=-server -Xms128m -Xmx128m -Xmn128m" \
-v /haoke/rmq/rmqbroker03/conf/broker.conf:/etc/rocketmq/broker.conf \
-v /haoke/rmq/rmqbroker03/logs:/opt/logs \
-v /haoke/rmq/rmqbroker03/store:/opt/store \
foxiswho/rocketmq:broker-4.3.2

#slave broker01
namesrvAddr=172.16.55.185:9876;172.16.55.185:9877
brokerClusterName=ItcastCluster
brokerName=broker01
brokerId=1
deleteWhen=04
fileReservedTime=48
```

```
brokerRole=SLAVE
flushDiskType=ASYNC_FLUSH
brokerIP1=172.16.55.185
brokerIp2=172.16.55.185
listenPort=10711
```

```
#创建第2个slave broker
#slave broker01
docker create --net host --name rmqbroker04 \
-e "JAVA_OPTS=-Duser.home=/opt"  \
-e "JAVA_OPT_EXT=-server -Xms128m -Xmx128m -Xmn128m" \
-v /haoke/rmq/rmqbroker04/conf/broker.conf:/etc/rocketmq/broker.conf \
-v /haoke/rmq/rmqbroker04/logs:/opt/logs \
-v /haoke/rmq/rmqbroker04/store:/opt/store \
foxiswho/rocketmq:broker-4.3.2

#slave broker02
namesrvAddr=172.16.55.185:9876;172.16.55.185:9877
brokerClusterName=ItcastCluster
brokerName=broker02
brokerId=1
deleteWhen=04
fileReservedTime=48
brokerRole=SLAVE
flushDiskType=ASYNC_FLUSH
brokerIP1=172.16.55.185
brokerIp2=172.16.55.185
listenPort=10611
```

```
#启动容器
docker start rmqserver01 rmqserver02
docker start rmqbroker01 rmqbroker02 rmqbroker03 rmqbroker04
```

3、SprinBoot整合RocketMQ

3.1、导入依赖

```
<?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.rocketmq</groupId>
    <artifactId>itcast-rocketmq</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-spring-boot-starter</artifactId>
            <version>2.0.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.rocketmq</groupId>
            <artifactId>rocketmq-client</artifactId>
            <version>4.3.2</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>
```

> 说明：rocketmq-spring-boot-starter的依赖包是不能直接从中央仓库下载的，需要自己通过源码install到本地仓库的。

```
#源码地址（或者使用资料中的源码）
https://github.com/apache/rocketmq-spring

#进入源码目录，执行如下命令
mvn clean install
```

3.2、编写application.properties配置文件

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

spring.rocketmq.nameServer=172.16.55.185:9876
spring.rocketmq.producer.group=my-group
```

3.3、生产者发送消息

```
package cn.itcast.rocketmq.spring;

import org.apache.rocketmq.spring.core.RocketMQTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class SpringProducer {

    @Autowired
    private RocketMQTemplate rocketMQTemplate;

    /**
     * 发送消息
     *
     * @param topic
     * @param msg
     */
    public void sendMsg(String topic, String msg){
        this.rocketMQTemplate.convertAndSend(topic, msg);
    }

}

```

3.4、消费消息

```
package cn.itcast.rocketmq.spring;

import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.springframework.stereotype.Component;
```

```
6
7  @Component
8  @RocketMQMessageListener(topic = "my-topic",
9          consumerGroup = "haoke-consumer",
10         selectorExpression = "*")
11 public class SpringConsumer implements RocketMQListener<String> {
12
13     @Override
14     public void onMessage(String msg) {
15         System.out.println("接收到消息 -> " + msg);
16     }
17 }
18
```

3.5、编写启动类

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

3.6、编写测试用例

```
1  package cn.itcast.rocketmq.spring;
2
3  import org.junit.Test;
4  import org.junit.runner.RunWith;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.boot.test.context.SpringBootTest;
7  import org.springframework.test.context.junit4.SpringRunner;
8
9  @RunWith(SpringRunner.class)
10 @SpringBootTest
11 public class TestSpringRocketMQ {
12     @Autowired
13     private SpringProducer springProducer;
14
15     @Test
16     public void testSendMsg(){
17         this.springProducer.sendMsg("my-topic", "第一个Spring消息");
18     }
19 }
```

```
20
```

3.7、测试

先启动springboot，再运行测试用例，即可看到消费者接收到生产者发送的消息。

3.8、事务消息

3.8.1、定义TransactionListenerImpl

```
1  package cn.itcast.rocketmq.spring.transaction;
2
3  import org.apache.rocketmq.spring.annotation.RocketMQTransactionListener;
4  import org.apache.rocketmq.spring.core.RocketMQLocalTransactionListener;
5  import org.apache.rocketmq.spring.core.RocketMQLocalTransactionState;
6  import org.apache.rocketmq.spring.support.RocketMQHeaders;
7  import org.springframework.messaging.Message;
8
9  import java.util.HashMap;
10 import java.util.Map;
11
12 @RocketMQTransactionListener(txProducerGroup = "myTransactionGroup")
13 public class TransactionListenerImpl implements RocketMQLocalTransactionListener {
14
15     private static Map<String, RocketMQLocalTransactionState> STATE_MAP = new
   HashMap<>();
16
17     @Override
18     public RocketMQLocalTransactionState executeLocalTransaction(Message message,
   Object o) {
19
20         String transId =
   (String)message.getHeaders().get(RocketMQHeaders.TRANSACTION_ID);
21
22         try {
23             System.out.println("执行操作1");
24             Thread.sleep(500);
25
26             System.out.println("执行操作2");
27             Thread.sleep(800);
28
29             STATE_MAP.put(transId, RocketMQLocalTransactionState.COMMIT);
30
31             return RocketMQLocalTransactionState.COMMIT;
32
33         } catch (Exception e) {
34             e.printStackTrace();
35         }
36
37         STATE_MAP.put(transId, RocketMQLocalTransactionState.ROLLBACK);
38
39         return RocketMQLocalTransactionState.ROLLBACK;
```

```
    }

    @Override
    public RocketMQLocalTransactionState checkLocalTransaction(Message message) {
        String transId =
(String)message.getHeaders().get(RocketMQHeaders.TRANSACTION_ID);
        System.out.println("回查消息 -> transId = " + transId + ", state = " +
STATE_MAP.get(transId));
        return STATE_MAP.get(transId);
    }
}

```

3.8.2、定义生产者

```
package cn.itcast.rocketmq.spring.transaction;

import org.apache.rocketmq.spring.core.RocketMQTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.Message;
import org.springframework.messaging.support.MessageBuilder;
import org.springframework.stereotype.Component;

import java.io.UnsupportedEncodingException;

@Component
public class SpringTransactionProducer {

    @Autowired
    private RocketMQTemplate rocketMQTemplate;

    /**
     * 发送消息
     *
     * @param topic
     * @param msg
     */
    public void sendMsg(String topic, String msg) {
        Message message = MessageBuilder.withPayload(msg).build();
        // myTransactionGroup要和@RocketMQTransactionListener(txProducerGroup =
"myTransactionGroup")定义的一致
        this.rocketMQTemplate.sendMessageInTransaction("myTransactionGroup",
                topic,
                message,
                null);
        System.out.println("发送消息成功");
    }

}
```

3.8.3、消费者（没有变化）

```java
package cn.itcast.rocketmq.spring;

import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.springframework.stereotype.Component;

@Component
@RocketMQMessageListener(topic = "my-topic",
        consumerGroup = "haoke-consumer",
        selectorExpression = "*")
public class SpringConsumer implements RocketMQListener<String> {

    @Override
    public void onMessage(String msg) {
        System.out.println("接收到消息 -> " + msg);
    }
}
```

3.8.4、编写测试用例

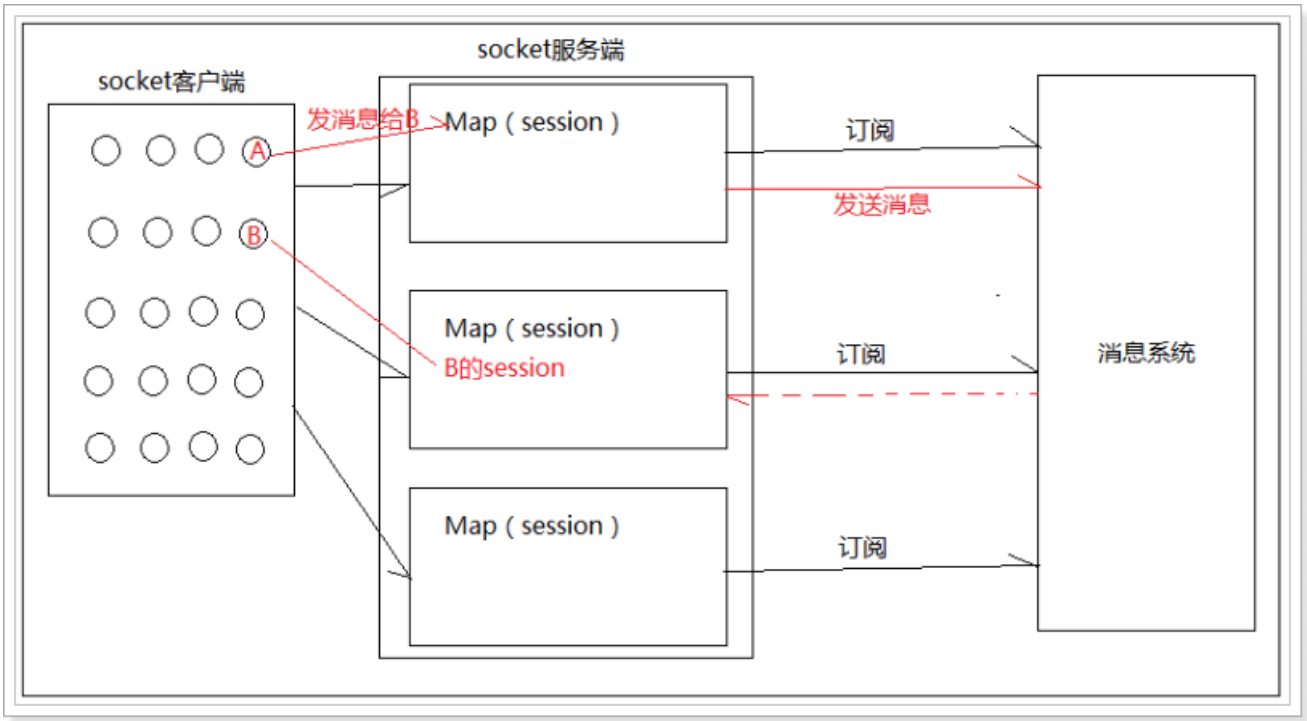
```java
package cn.itcast.rocketmq.spring;

import cn.itcast.rocketmq.spring.transaction.SpringTransactionProducer;
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;

@RunWith(SpringRunner.class)
@SpringBootTest
public class TestSpringRocketMQ {
    @Autowired
    private SpringProducer springProducer;

    @Autowired
    private SpringTransactionProducer springTransactionProducer;

    @Test
    public void testSendMsg(){
        this.springProducer.sendMsg("my-topic", "第2个Spring消息");
    }

    @Test
    public void testSendMsg2(){
        this.springTransactionProducer.sendMsg("my-topic", "第3个Spring消息");
    }
}
```

3.8.5、测试

测试结果与非Spring使用，结果一致。

4、实现分布式WebSocket

4.1、实现方案



4.2、导入RocketMQ相关依赖

```
<!--RocketMQ相关依赖-->
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
    <version>2.0.0</version>
</dependency>
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>4.3.2</version>
</dependency>
```

4.3、添加SpringBoot配置

```
spring.rocketmq.nameServer=172.16.55.185:9876
spring.rocketmq.producer.group=haoke-im-websocket-group
```

4.4、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.apache.rocketmq.spring.annotation.MessageModel;
import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.apache.rocketmq.spring.core.RocketMQTemplate;
import org.apache.rocketmq.spring.support.RocketMQHeaders;
import org.bson.types.ObjectId;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.support.MessageBuilder;
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.io.IOException;
import java.util.HashMap;
import java.util.Map;

@Component
@RocketMQMessageListener(topic = "haoke-im-send-message-topic",
        consumerGroup = "haoke-im-consumer",
        messageModel = MessageModel.BROADCASTING,
        selectorExpression = "SEND_MSG")
public class MessageHandler extends TextWebSocketHandler implements
RocketMQListener<String> {

    @Autowired
    private MessageDAO messageDAO;

    @Autowired
    private RocketMQTemplate rocketMQTemplate;

    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();
```

```
57
58          Message message = Message.builder()
59                  .from(UserData.USER_MAP.get(uid))
60                  .to(UserData.USER_MAP.get(toId))
61                  .msg(msg)
62                  .build();
63
64          // 将消息保存到MongoDB
65          message = this.messageDAO.saveMessage(message);
66
67          String msgStr = MAPPER.writeValueAsString(message);
68
69          // 判断to用户是否在线
70          WebSocketSession toSession = SESSIONS.get(toId);
71          if (toSession != null && toSession.isOpen()) {
72              //TODO 具体格式需要和前端对接
73              toSession.sendMessage(new TextMessage(msgStr));
74              // 更新消息状态为已读
75              this.messageDAO.updateMessageState(message.getId(), 2);
76          } else {
77              // 用户不在线，或者不在当前的jvm中，发送消息到RocketMQ
78              org.springframework.messaging.Message mqMessage = MessageBuilder
79                      .withPayload(msgStr)
80                      .build();
81              // topic:tags 设置主题和标签
82              this.rocketMQTemplate.send("haoke-im-send-message-topic:SEND_MSG",
mqMessage);
83          }
84
85      }
86
87      @Override
88      public void onMessage(String msg) {
89  //        System.out.println("接收到消息 -> " + msg);
90          try {
91              JsonNode jsonNode = MAPPER.readTree(msg);
92              Long toId = jsonNode.get("to").get("id").longValue();
93              // 判断to用户是否在线
94              WebSocketSession toSession = SESSIONS.get(toId);
95              if (toSession != null && toSession.isOpen()) {
96                  toSession.sendMessage(new TextMessage(msg));
97                  // 更新消息状态为已读
98                  this.messageDAO.updateMessageState(new
ObjectId(jsonNode.get("id").asText()), 2);
99              }
100         } catch (Exception e) {
101             e.printStackTrace();
102         }
103     }
104 }
105
```

Message对象添加ObjectId的注解：

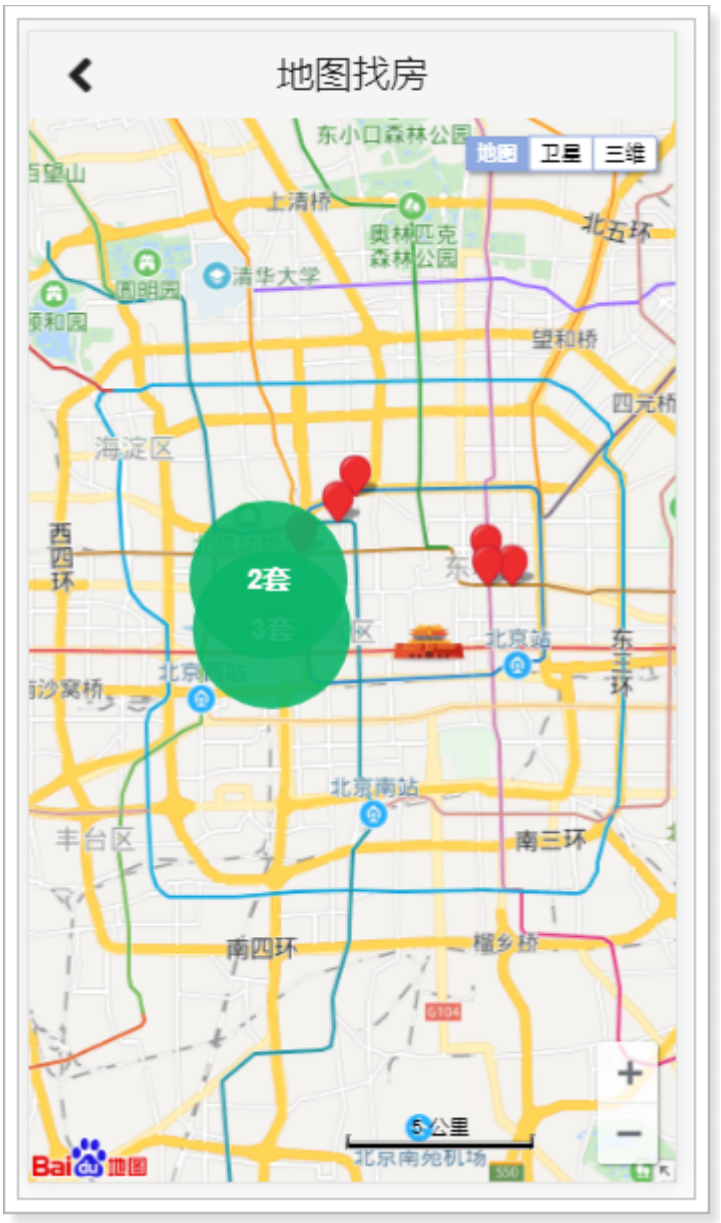
```
@Id
@JsonSerialize(using = ToStringSerializer.class)
private ObjectId id;
```

4.5、测试

可以看到，1003接收到了1001发来的消息，说明消息系统已经发挥了作用。

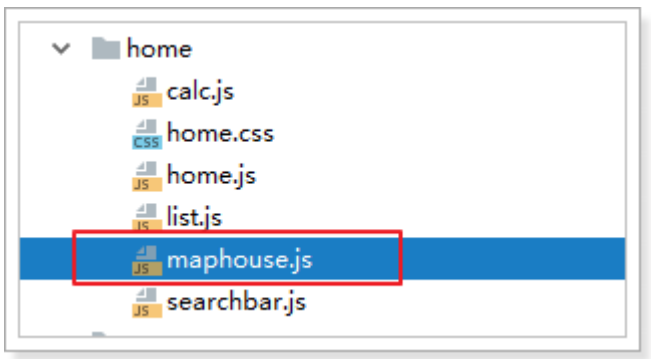
5、地图找房

完善项目的功能，实现地图找房功能，效果如下：



5.1、前端实现

地图使用的是百度地图api，具体代码实现：



可以看到，目前的数据是静态的，需要改造成加载后台数据的方式。

5.2、改造数据的加载方式

5.2.1、修改数据的加载逻辑

```
import ApolloClient from "apollo-boost";
import gql from "graphql-tag";

const client = new ApolloClient({
    uri: "http://127.0.0.1:18080/graphql"
});

//定义查询
const GET_MAP_HOUSE = gql`
    {
        MapHouseData {
            list {
                x
                y
            }
        }
    }
`;

client.query({query: GET_MAP_HOUSE}).then(result =>{
    let xys = result.data.MapHouseData.list;

    var markers = [];
    var pt = null;
    for (var i in xys) {
        pt = new BMap.Point(xys[i].x, xys[i].y);
        markers.push(new BMap.Marker(pt));
    }
    // 地图上覆盖物的聚合效果
    var markerClusterer = new BMapLib.MarkerClusterer(map, {
        markers: markers,
        girdSize: 100,
        styles: [{
            background: 'rgba(12,181,106,0.9)',
            size: new BMap.Size(92, 92),
            textSize: '16',
            textColor: '#fff',
            borderRadius: 'true'
        }],
    });
    markerClusterer.setMaxZoom(50);
    markerClusterer.setGridSize(50);
});
```

5.2.2、编写graphql接口

```
schema {
    query: HaokeQuery
}

```

```
type HaokeQuery {
    HouseResources(id:Long) : HouseResources
    HouseResourcesList(page:Int, pageSize:Int) : TableResult
    IndexAdList:IndexAdResult
    MapHouseData:MapHouseDataResult
}

type HouseResources {
    id:Long!
    title:String
    estateId:Long
    buildingNum:String
    buildingUnit:String
    buildingFloorNum:String
    rent:Int
    rentMethod:Int
    paymentMethod:Int
    houseType:String
    coveredArea:String
    useArea:String
    floor:String
    orientation:String
    decoration:Int
    facilities:String
    pic:String
    houseDesc:String
    contact:String
    mobile:String
    time:Int
    propertyCost:String
}

type TableResult{
    list:[HouseResources]
    pagination:Pagination
}

type Pagination{
    current:Int
    pageSize:Int
    total:Int
}

type IndexAdResult{
    list:[IndexAdResultData]
}

type IndexAdResultData{
    original:String
}

type MapHouseDataResult{
    list:[MapHouseXY]
```

```
}
type MapHouseXY{
    x:Float
    y:Float
}
```

编写MapHouseDataResult、MapHouseXY：

```
package cn.itcast.haoke.dubbo.api.vo.map;

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

import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class MapHouseDataResult {

    private List<MapHouseXY> list;
}

```

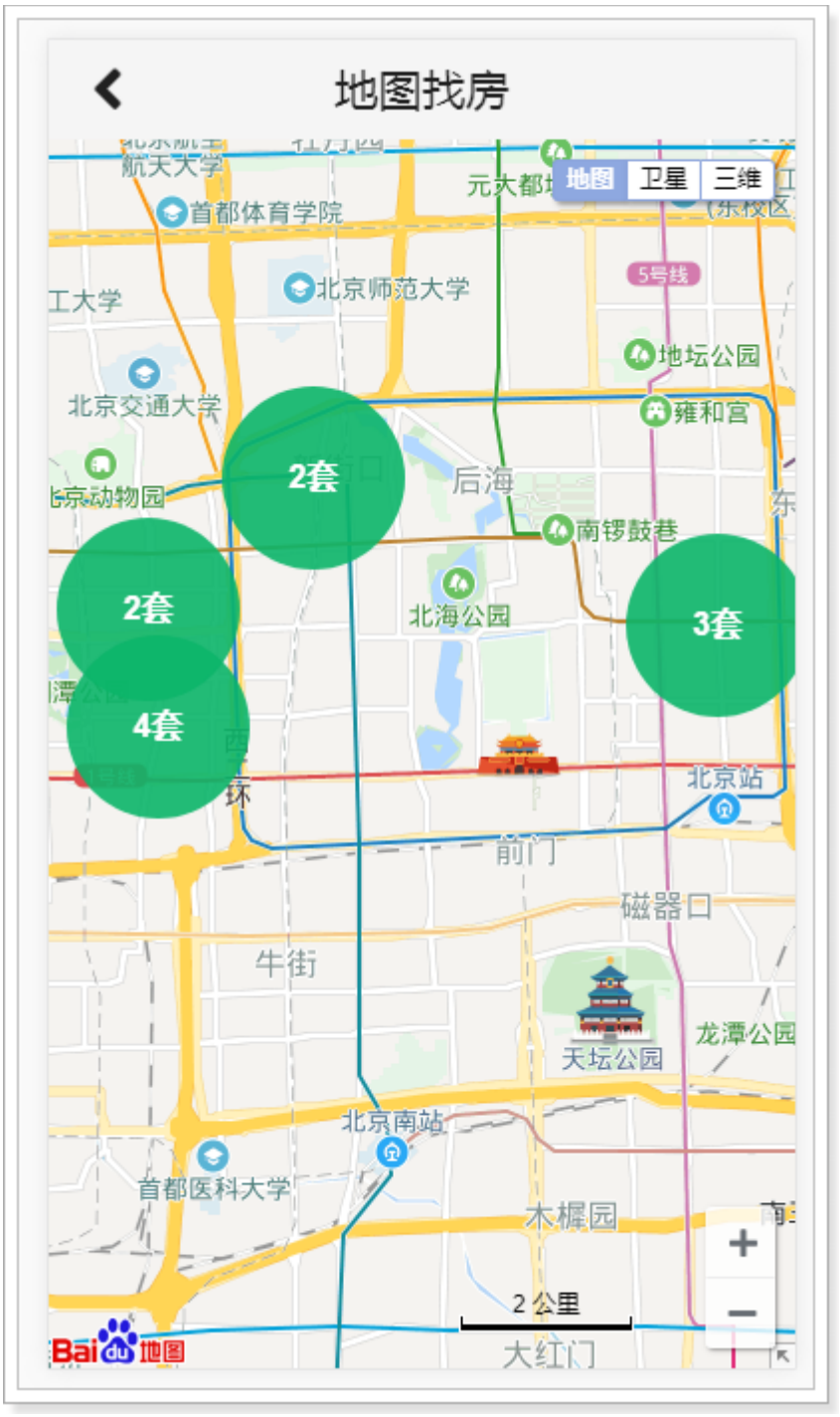
```
package cn.itcast.haoke.dubbo.api.vo.map;

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

@Data
@AllArgsConstructor
@NoArgsConstructor
public class MapHouseXY {

    private Float x;
    private Float y;
}

```

编写MapHouseDataFetcher

```
package cn.itcast.haoke.dubbo.api.graphql;

import cn.itcast.haoke.dubbo.api.service.HouseResourcesService;
import cn.itcast.haoke.dubbo.api.vo.map.MapHouseDataResult;
import cn.itcast.haoke.dubbo.api.vo.map.MapHouseXY;
import graphql.schema.DataFetchingEnvironment;
import org.springframework.stereotype.Component;

```

```
import java.util.ArrayList;
import java.util.List;

@Component
public class MapHouseDataFetcher implements MyDataFetcher {


    @Override
    public String fieldName() {
        return "MapHouseData";
    }

    @Override
    public Object dataFetcher(DataFetchingEnvironment environment) {
        List<MapHouseXY> list = new ArrayList<>();
        list.add(new MapHouseXY(116.43244f,39.929986f));
        list.add(new MapHouseXY(116.424355f,39.92982f));
        list.add(new MapHouseXY(116.423349f,39.935214f));
        list.add(new MapHouseXY(116.350444f,39.931645f));
        list.add(new MapHouseXY(116.351684f,39.91867f));
        list.add(new MapHouseXY(116.353983f,39.913855f));
        list.add(new MapHouseXY(116.357253f,39.923152f));
        list.add(new MapHouseXY(116.349168f,39.923152f));
        list.add(new MapHouseXY(116.36232f,39.938339f));
        list.add(new MapHouseXY(116.374249f,39.94625f));
        list.add(new MapHouseXY(116.380178f,39.953053f));
        return new MapHouseDataResult(list);
    }
}

```

5.2.3、整合测试



5.3、增加拖动事件

在地图拖动后，增加事件，获取中心位置的坐标，以便后续的查询。

```
map.addEventListener("dragend", function showInfo(){
    let cp = map.getCenter();
    let zoom = map.getZoom(); //缩放级别
    console.log(cp.lng + "," + cp.lat+" --> " + zoom);
});
```

测试：

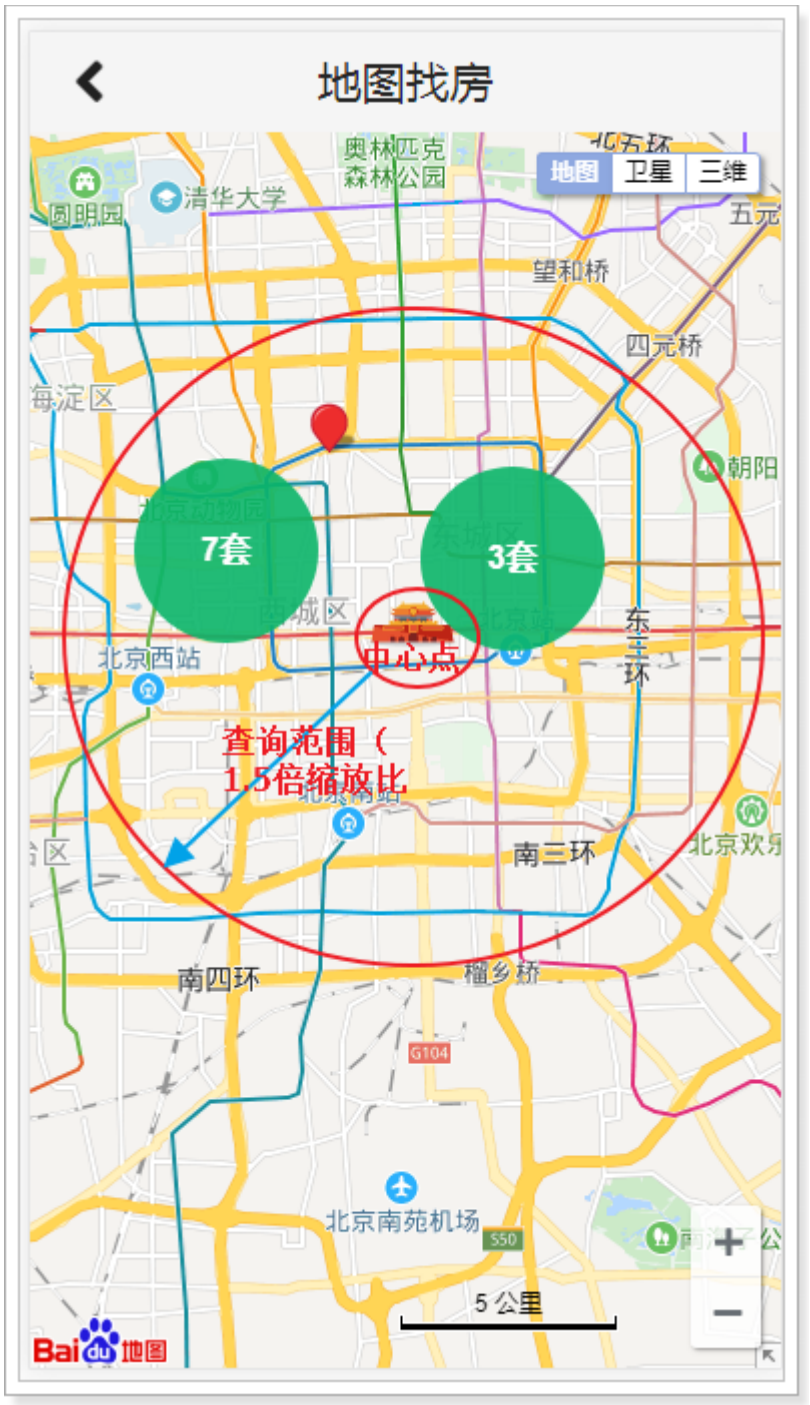
```
116.5342,39.87685 --> 12                    maphouse.js:54
116.621012,39.800615 --> 12                 maphouse.js:54
116.696901,39.720744 --> 12                 maphouse.js:54
116.718173,39.605209 --> 12                 maphouse.js:54
116.763304,39.552933 --> 13                 maphouse.js:54
116.805273,39.501062 --> 13                 maphouse.js:54
```

百度地图缩放比例：

级别	比例尺
19级	20m
18级	50m
17级	100m
16级	200m
15级	500m
14级	1km
13级	2km
12级	5km
11级	10km
10级	20km
9级	25km
8级	50km
7级	100km
6级	200km
5级	500km
4级	1000km
3级	2000km
2级	5000km
1级	10000km

5.4、传递经纬度以及缩放比例参数

查询数据的原理图示：



> 说明：中心点坐标的外围1.5倍缩放比例的范围是查询范围。

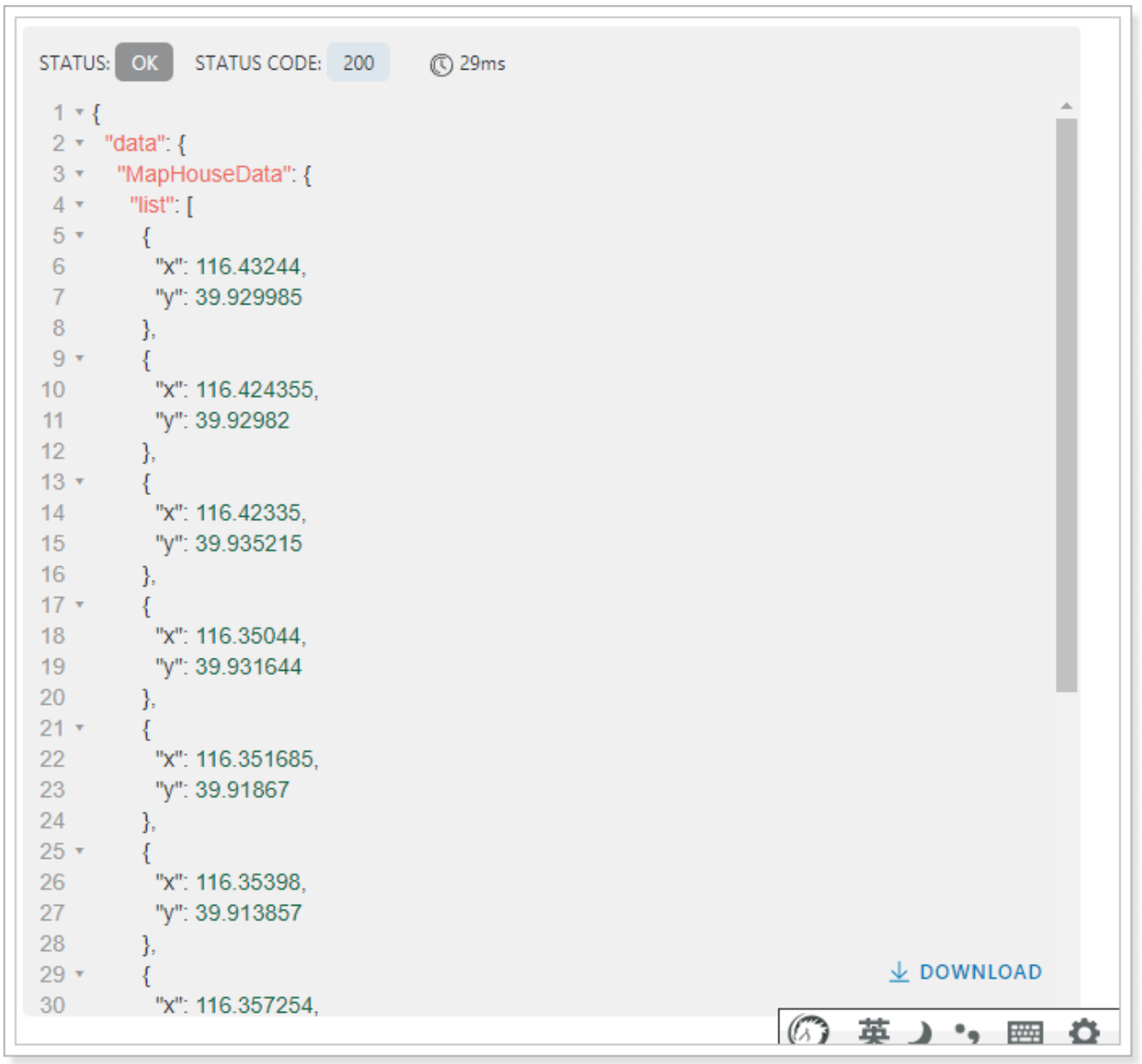
修改GraphQL定义：

```
type HaokeQuery {
    HouseResources(id:Long) : HouseResources
    HouseResourcesList(page:Int, pageSize:Int) : TableResult
    IndexAdList:IndexAdResult
    MapHouseData(lng:Float,lat:Float,zoom:Int):MapHouseDataResult
}
```

测试查询：

```
query QueryMapHouseData($lng: Float, $lat: Float, $zoom: Int) {
  MapHouseData(lng: $lng, lat: $lat, zoom: $zoom) {
    list {
      x
      y
    }
  }
}

----

设置参数：
{
  "lng":116.43244,
  "lat":39.929986,
  "zoom":12
}
```

查询结果：



优化前端代码：

```
import React from 'react';
import { Icon} from 'semantic-ui-react';
import ApolloClient from "apollo-boost";
import gql from "graphql-tag";

const client = new ApolloClient({
    uri: "http://127.0.0.1:18080/graphql"
});

//定义查询
const GET_MAP_HOUSE = gql`
    query QueryMapHouseData($lng: Float, $lat: Float, $zoom: Int) {
        MapHouseData(lng: $lng, lat: $lat, zoom: $zoom) {
            list {
                x
                y
            }
        }
    }

`;

const BMap = window.BMap;
const BMapLib = window.BMapLib;

const showMapMarker=(xys, map)=>{
    let markers = [];
    let pt = null;
    for (let i in xys) {
        pt = new BMap.Point(xys[i].x, xys[i].y);
        markers.push(new BMap.Marker(pt));
    }
    // 地图上覆盖物的聚合效果
    let markerClusterer = new BMapLib.MarkerClusterer(map, {
        markers: markers,
        girdSize: 100,
        styles: [{
            background: 'rgba(12,181,106,0.9)',
            size: new BMap.Size(92, 92),
            textSize: '16',
            textColor: '#fff',
            borderRadius: 'true'
        }],
    });
    markerClusterer.setMaxZoom(50);
    markerClusterer.setGridSize(50);
}

class MapHouse extends React.Component {
  constructor(props) {
    super(props);
```

```
  }



  componentDidMount() {
    let defaultX = 121.48130241985999;
    let defaultY = 31.235156971414239;
    let defaultZoom = 12;

    // 创建Map实例
    let map = new BMap.Map("allmap");
    // 初始化地图,设置中心点坐标和地图级别
    map.centerAndZoom(new BMap.Point(defaultX,defaultY), defaultZoom);
    // 添加地图类型控件
    map.addControl(new BMap.MapTypeControl());
    // 设置地图缩放
    map.addControl(new BMap.ScaleControl({
      anchor: window.BMAP_NAVIGATION_CONTROL_ZOOM
    }));
    // 设置地图导航
    map.addControl(new BMap.NavigationControl({
      enableGeolocation: true
    }));
    // 设置缩略图控件。
    map.addControl(new BMap.OverviewMapControl());
    // 设置地图显示的城市 此项是必须设置的
    map.setCurrentCity("上海");
    // 开启鼠标滚轮缩放
    map.enableScrollWheelZoom(true);

    let showInfo = () => {
        let cp = map.getCenter();
        let zoom = map.getZoom(); //缩放级别
        // console.log(cp.lng + "," + cp.lat+" --> " + zoom);
        client.query({query: GET_MAP_HOUSE, variables:
{"lng":cp.lng,"lat":cp.lat,"zoom":zoom}}).then(result =>{
            let xys = result.data.MapHouseData.list;
            showMapMarker(xys,map);
        });
    }

    map.addEventListener("dragstart",()=>{map.clearOverlays();}); //拖动开始事件
    map.addEventListener("dragend",showInfo); //拖动结束事件
    map.addEventListener("zoomstart", ()=>{map.clearOverlays();}); //缩放开始事件
    map.addEventListener("zoomend",showInfo); //缩放结束事件

    // 初始化
    client.query({query: GET_MAP_HOUSE, variables:
{"lng":defaultX,"lat":defaultY,"zoom":defaultZoom}}).then(result =>{
        let xys = result.data.MapHouseData.list;
        showMapMarker(xys,map);
    });


```

```
  }
  render() {
    return (
      <div className = 'map-house' >
        <div className = "map-house-title">
          <Icon onClick = {this.props.hideMap} name = 'angle left' size =
'large'/> 地图找房
        </div>
        <div className = "map-house-content" id='allmap'></div>
      </div>
    );
  }
}
export default MapHouse;

```

5.5、MongoDB的地理位置索引

在MongoDB中，支持存储位置的经纬度，可以对其索引，通过算子操作，进行查找附近的数据。如：查找附近的人、附近的餐馆等。

我们可以用此特性，存储房源的位置数据以及进行地图找房查询。

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

use testdb
db.house.createIndex({loc:'2d'}) #为house表的loc字段创建地理2d索引
{
        "createdCollectionAutomatically" : true,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
db.house.createIndex({hid:1},{unique:true}) #为house表的hid字段创建唯一索引

#通过百度api查询地址的经纬度
http://api.map.baidu.com/geocoder/v2/?address=上海
xxxx&ak=jpfEH2etB2gutGyHpxVdWy8ZrTxbuOqj&output=json

#插入测试数据
db.house.insert({hid:1,title:'整租 · 南丹大楼 1居室 7500',loc:
[121.4482236974557,31.196523937504549]})
db.house.insert({hid:2,title:'陆家嘴板块，精装设计一室一厅，可拎包入住诚意租。',loc:
[121.51804613891443,31.238878702131506]})
db.house.insert({hid:3,title:'整租 · 健安坊 1居室 4050',loc:
[121.4148310693774,31.16507733043528]})
db.house.insert({hid:4,title:'整租 · 中凯城市之光+视野开阔+景色秀丽+拎包入住',loc:
[121.43528282056717,31.198687949417815]})
db.house.insert({hid:5,title:'整租 · 南京西路品质小区 21213三轨交汇 配套齐* 拎包入住',loc:
[121.43528282056717,31.198687949417815]})
db.house.insert({hid:6,title:'祥康里 简约风格 *南户型 拎包入住 看房随时',loc:
[121.47521508401232,31.23859308719981]})
```

```
db.house.insert({hid:7,title:'整租 · 桃源新村 1室0厅 5500元',loc:[121.488377804633,31.23113867155927]})
db.house.insert({hid:8,title:'整租 · 中山公园品质小区，两房朝南厅朝南，家电家具精装*配',loc:[121.42038642151562,31.225078800208654]})
db.house.insert({hid:9,title:'整租 · 近地铁2号线，精装1房1厅，高区朝南，享受阳光好房',loc:[121.42933310871683,31.221943586471036]})
db.house.insert({hid:10,title:'整租 · 2.3.4号中山公园地铁，背靠来福士，采光好，诚意出租',loc:[121.42063977421182,31.221023374982044]})

#查询上海人民广场附近5公里的房源，人民广场的坐标为：121.48130241985999,31.235156971414239

db.house.find({loc:{$near:[121.48130241985999,31.235156971414239],$maxDistance:5/111.12 }})

#注意距离要除以111.2（1度=111.2km），跟普通查找的区别仅仅是多了两个算子$near和$maxDistance

#查询结果：
{ "_id" : ObjectId("5c1c99cd4729d1aaa1bdbbf3"), "hid" : 6, "title" : "祥康里 简约风格*南户型 拎包入住 看房随时", "loc" : [ 121.47521508401232, 31.23859308719981 ] }
{ "_id" : ObjectId("5c1c99cd4729d1aaa1bdbbf4"), "hid" : 7, "title" : "整租 · 桃源新村 1室0厅 5500元", "loc" : [ 121.488377804633, 31.23113867155927 ] }
{ "_id" : ObjectId("5c1c99cd4729d1aaa1bdbbef"), "hid" : 2, "title" : "陆家嘴板块，精装设计一室一厅，可拎包入住诚意租。", "loc" : [ 121.51804613891443, 31.238878702131505 ] }


#查询上海中山公园附近2公里的房源，中山公园的坐标为：121.42261657004589,31.229111410235285
db.house.find({loc:{$near:[121.42261657004589,31.229111410235285],$maxDistance:2/111.12 }})

#查询结果：
{ "_id" : ObjectId("5c1c99cd4729d1aaa1bdbbf5"), "hid" : 8, "title" : "整租 · 中山公园品质小区，两房朝南厅朝南，家电家具精装*配", "loc" : [ 121.42038642151562, 31.225078800208653 ] }
{ "_id" : ObjectId("5c1c99cd4729d1aaa1bdbbf7"), "hid" : 10, "title" : "整租 · 2.3.4号中山公园地铁，背靠来福士，采光好，诚意出租", "loc" : [ 121.42063977421182, 31.221023374982042 ] }
{ "_id" : ObjectId("5c1c99cd4729d1aaa1bdbbf6"), "hid" : 9, "title" : "整租 · 近地铁2号线，精装1房1厅，高区朝南，享受阳光好房", "loc" : [ 121.42933310871683, 31.221943586471035 ] }

```

5.6、实现基于MongoDB的查询

5.6.1、导入依赖

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>3.9.1</version>
</dependency>
```

5.6.2、添加配置

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

5.6.3、编写实现代码

编写pojo：

```java
package cn.itcast.haoke.dubbo.api.pojo;

import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.databind.ser.std.ToStringSerializer;
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.mapping.Document;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Document(collection = "house") // 指定表的名称
@Builder
public class MongoHouse {

    @Id
    @JsonSerialize(using = ToStringSerializer.class)
    private ObjectId id;

    private Long hid;

    private String title;

    private Float[] loc;
}
```

编写MongoHouseService：

```java
package cn.itcast.haoke.dubbo.api.service;
```

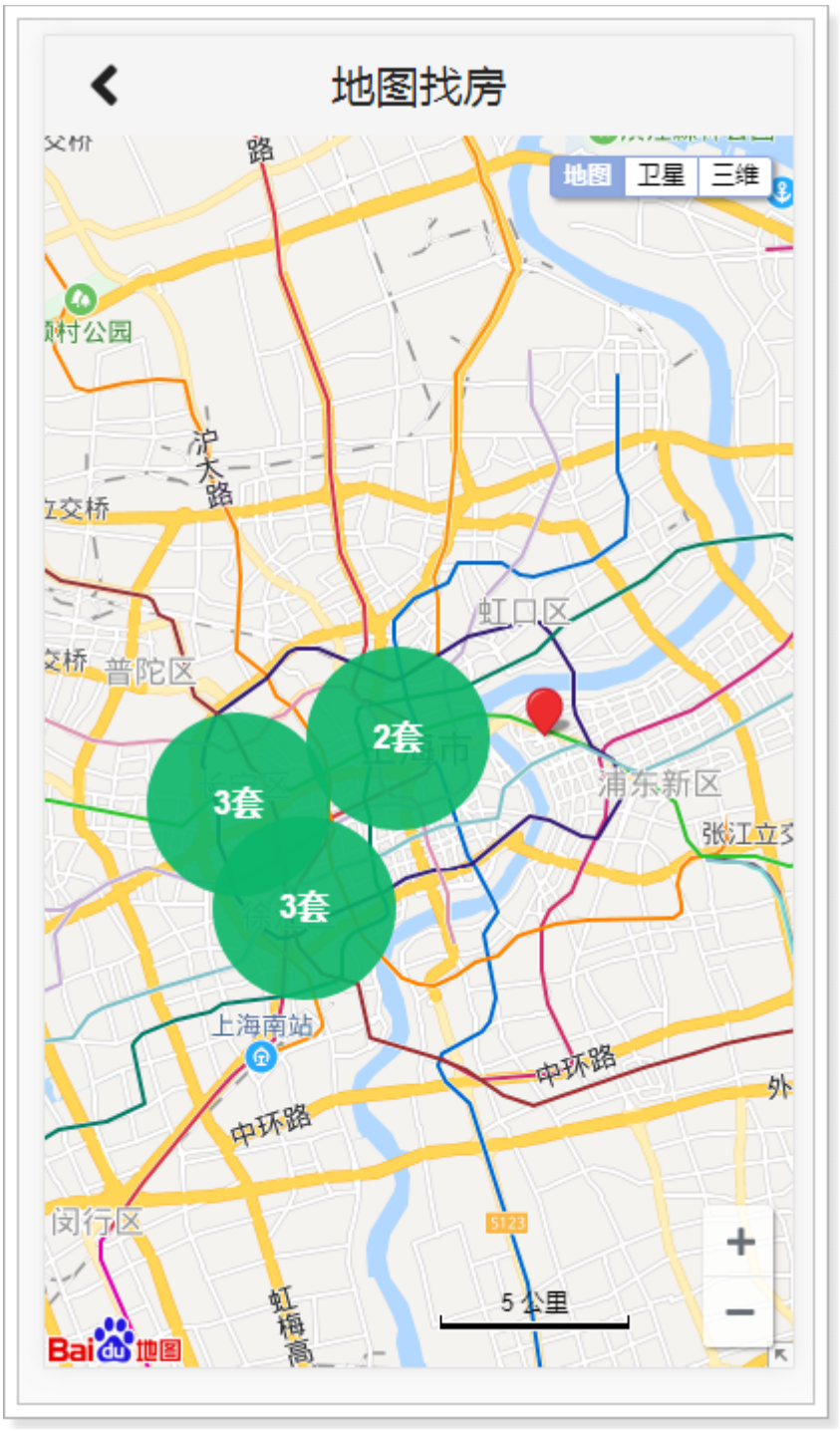
```
import cn.itcast.haoke.dubbo.api.pojo.MongoHouse;
import cn.itcast.haoke.dubbo.api.vo.map.MapHouseDataResult;
import cn.itcast.haoke.dubbo.api.vo.map.MapHouseXY;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.geo.Point;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.stereotype.Service;

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

@Service
public class MongoHouseService {

    public static final Map<Integer, Double> BAIDU_ZOOM = new HashMap<>();

    static {
        BAIDU_ZOOM.put(19, 20d / 1000); //单位为km
        BAIDU_ZOOM.put(18, 50d / 1000);
        BAIDU_ZOOM.put(17, 100d / 1000);
        BAIDU_ZOOM.put(16, 200d / 1000);
        BAIDU_ZOOM.put(15, 500d / 1000);
        BAIDU_ZOOM.put(14, 1d);
        BAIDU_ZOOM.put(13, 2d);
        BAIDU_ZOOM.put(12, 5d);
        BAIDU_ZOOM.put(11, 10d);
        BAIDU_ZOOM.put(10, 20d);
        BAIDU_ZOOM.put(9, 25d);
        BAIDU_ZOOM.put(8, 50d);
        BAIDU_ZOOM.put(7, 100d);
        BAIDU_ZOOM.put(6, 200d);
        BAIDU_ZOOM.put(5, 500d);
        BAIDU_ZOOM.put(4, 1000d);
        BAIDU_ZOOM.put(3, 2000d);
        BAIDU_ZOOM.put(2, 5000d);
        BAIDU_ZOOM.put(1, 10000d);
    }

    @Autowired
    private MongoTemplate mongoTemplate;

    public MapHouseDataResult queryHouseData(Float lng, Float lat, Integer zoom) {
        double distance = BAIDU_ZOOM.get(zoom) * 1.5 / 111.12; //1.5倍距离范围，根据实际需求调整
        Query query = Query.query(Criteria.where("loc").near(new Point(lng, lat)).maxDistance(distance));
        List<MongoHouse> mongoHouses = this.mongoTemplate.find(query, MongoHouse.class);
```

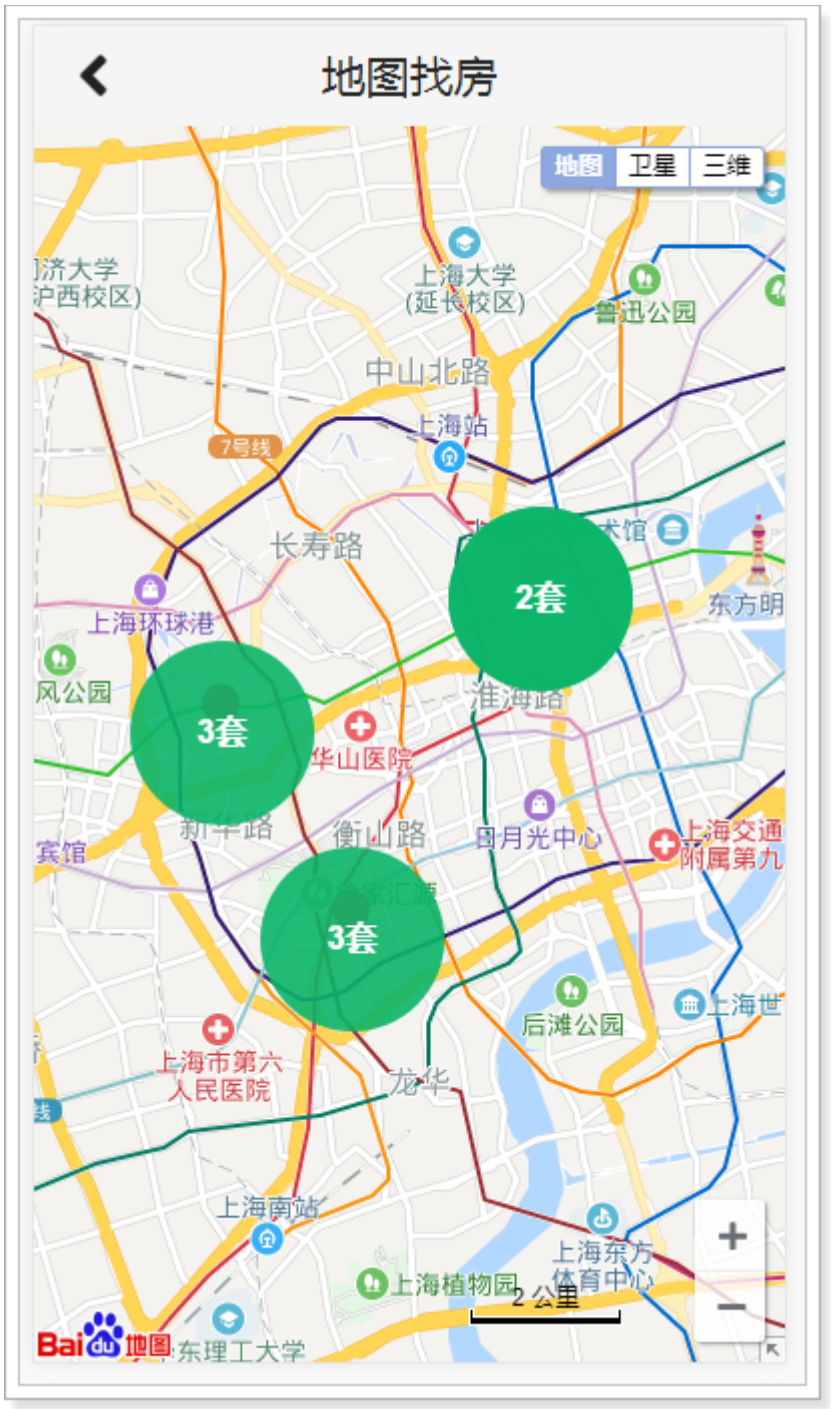
```
        List<MapHouseXY> list = new ArrayList<>();
        for (MongoHouse mongoHouse : mongoHouses) {
            list.add(new MapHouseXY(mongoHouse.getLoc()[0], mongoHouse.getLoc()
[1]));
        }
        return new MapHouseDataResult(list);
    }
}
```

修改MapHouseDataFetcher：

```
package cn.itcast.haoke.dubbo.api.graphql;

import cn.itcast.haoke.dubbo.api.service.HouseResourcesService;
import cn.itcast.haoke.dubbo.api.service.MongoHouseService;
import cn.itcast.haoke.dubbo.api.vo.map.MapHouseDataResult;
import cn.itcast.haoke.dubbo.api.vo.map.MapHouseXY;
import graphql.schema.DataFetchingEnvironment;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;

@Component
public class MapHouseDataFetcher implements MyDataFetcher {

    @Autowired
    private MongoHouseService mongoHouseService;

    @Override
    public String fieldName() {
        return "MapHouseData";
    }

    @Override
    public Object dataFetcher(DataFetchingEnvironment environment) {
        Float lng = ((Double)environment.getArgument("lng")).floatValue();
        Float lat = ((Double)environment.getArgument("lat")).floatValue();
        Integer zoom = environment.getArgument("zoom");

        System.out.println("lng->" + lng + ",lat->" + lat + ",zoom->" + zoom);

        return this.mongoHouseService.queryHouseData(lng,lat,zoom);
    }
}

```

5.7、测试


地图找房
地图
卫星
三维
虹口区
普陀区
浦东新区
2套
3套
3套
上海南站
中环路
中环路
闵行区
5公里
Bai du 地图


地图找房
地图
卫星
三维
上海大学
(延长校区)
鲁迅公园
中山北路
上海站
7号线
长寿路
2套
上海环球港
东方明
3套
华山医院
淮海路
衡山路
国月光中心
上海交通
附属第九
3套
后滩公园
上海市第六
人民医院
龙华
上海南站
上海东方
体育中心
上海植物园
2 公里
东理工大学
Baidu 地图