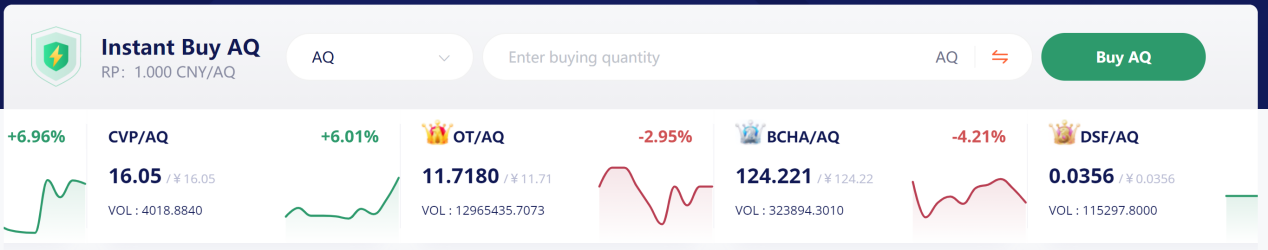
# 推送系统的实现

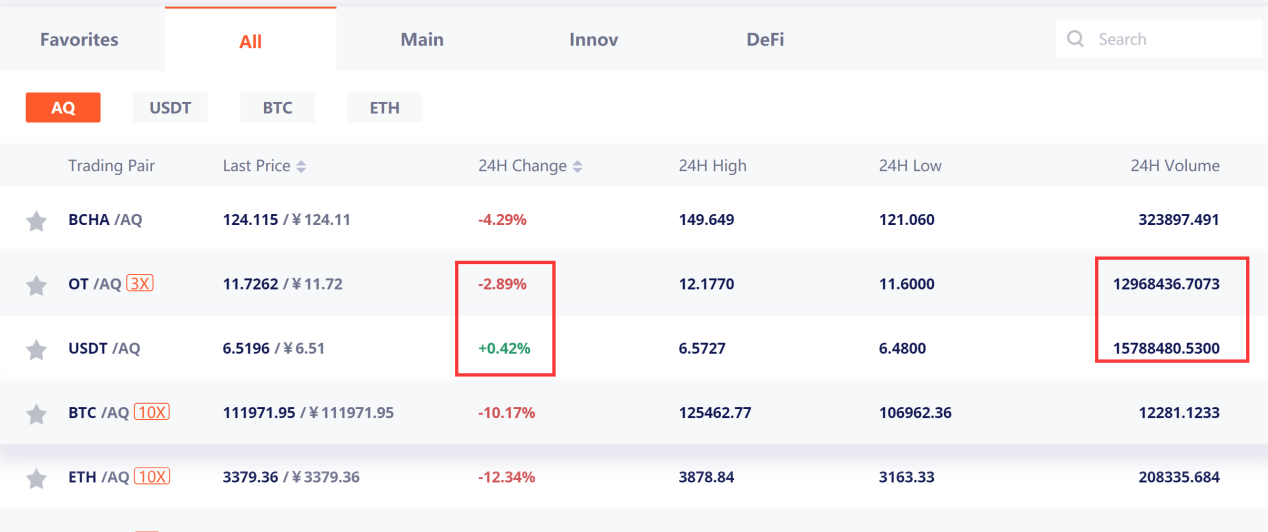
# 推送系统的简介

向前台系统推送市场详情数据\K-Line数据\深度数据\盘口数据等等!

<https://www.aofex.co/#/>

参考这个网站:





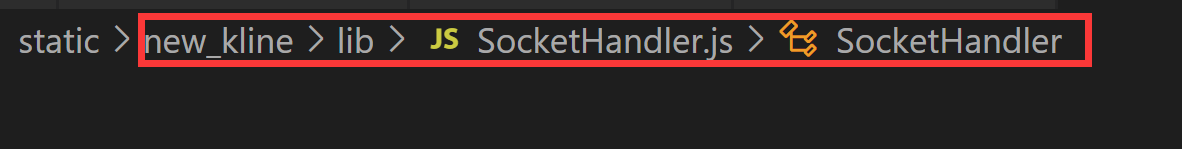




# 前端实现

## 2.1 WebSocket工具类

位于:



|  |
| --- |
| function SocketHandler(url) {  // 基于H5原生api  this.ws = new WebSocket(url);  // 定义一个信号发射塔，用于发送事件  this.tower = document.createElement('div');  }  SocketHandler.prototype = {  // 订阅频道  channels : {},  // 反应堆（用于收集和分发socket的响应）  reactions : {},  // 缓存监听事件  events : [],  // 获取当前时间  nowTime : function() {  return new Date().getTime();  },  // 打开socket连接  open : function(heartbeatTimeout) {  this.ws.onopen = function(){  var heartbeatSendInterval = heartbeatTimeout / 2;  this.lastSubscribeTime = this.nowTime();  this.pingIntervalId = setInterval(function(){  var iv = this.nowTime() - this.lastSubscribeTime;  // 超过一定时间自动与后台ping、pong 单位：秒  if ((heartbeatSendInterval + iv) >= heartbeatTimeout) {  this.send('ping');  }  }.bind(this), heartbeatSendInterval);  }.bind(this);  },  // 创建自定义事件  createEvent : function(event, detail){  var evt = document.createEvent('CustomEvent');  evt.initCustomEvent(event, false, false, detail);  return evt;  },  // 打开socket连接  connect: function(heartbeatTimeout) {  this.open(heartbeatTimeout);  this.message();  this.close();  },  // 订阅消息 ch 为订阅的频道 id 为订阅唯一标识  **subscribe** : function(ch, id, token) {  if(this.ws.readyState == 1) {  var obj = {};  if(ch) {  obj.sub = ch;  obj.id = id;  if(token) obj.authorization = token;  if(this.channels.hasOwnProperty(id)) {  this.unsubscribe.apply(this, Object.values(this.channels[id]));  }  this.channels[id] = obj;  this.reactions[ch] = this.createEvent(id);  this.send(obj);  }  }  },  // 监听订阅结果  on : function(id, callback) {  var handler = function(e) {  if(callback) callback(e.data, e);  };  this.tower.addEventListener(id, handler);  var key = id + '\_' + this.nowTime();  this.events.push({key : key, handler: handler});  },  // 取消订阅  **unsubscribe** : function(ch, id, token) {  if(this.ws.readyState == 1) {  var obj = {};  if(ch) {  obj.cancel = ch;  obj.id = id;  if(token) obj.authorization = token;  if(this.channels.hasOwnProperty(id)) {  delete this.channels[id];  delete this.reactions[ch];  }  this.events = this.events.filter(function(v){  if(v.key.indexOf(id) !== -1) {  // 失效ID解除监听  this.tower.removeEventListener(id, v.handler);  }else{  return v;  }  }.bind(this));  this.send(obj);  }  }  },  send : function(data){  if(typeof data === 'object') {  data = JSON.stringify(data);  }  this.ws.send(data);  },  message : function() {  this.ws.onmessage = function(evt){  var data = evt.data;  this.lastSubscribeTime = this.nowTime();  if(data) {  if(data !== 'pong') data = JSON.parse(data);  if(typeof data === 'object' && data.hasOwnProperty('ch')) {  var e = this.reactions[data.ch];  if(e) {  e.data = data;  this.tower.dispatchEvent(e);  }  }  }  }.bind(this);  },  close : function(callback) {  this.ws.onclose = function(evt){  clearInterval(this.pingIntervalId);  if(callback) callback(evt, this.ws);  }.bind(this);  }  }; |

## 2.2 深度数据的订阅

|  |
| --- |
| subscribeDepths(kline) {  let symbol = this.currentMarket.symbol.toLowerCase();  let mergeDepth = this.chooseMergeDepth;  this.$socket.subscribe(`market.${symbol}.depth.${mergeDepth}`, 'market-depth');  this.$socket.on('market-depth', (data) => {  this.depthsData = data.tick;  if(kline && kline.contentWindow) {  kline.contentWindow.set\_current\_depth(this.depthsData);  }  this.processDepthData()  })  }, |

## 2.3 成交记录的订阅

|  |
| --- |
| subscribeTrades() {  let symbol = this.currentMarket.symbol.toLowerCase();  this.$socket.subscribe(`market.${symbol}.trade.detail`, 'market-trade');  this.$socket.on('market-trade', (data) => {  // console.log("成交记录",data)  this.tradesData = data.data;  })  }, |

## 2.4 历史委任数据的订阅

|  |
| --- |
| //订阅历史委托  subscribeTurnover() {  this.$socket.subscribe(`order.finished.update`, 'order-finished', this.token);  this.$socket.on('order-finished', (data) => {  // console.log("成交记录事件", data)  this.\_serverGetTurnoverOrderList()  })  } |

## 2.5 市场对的订阅

|  |
| --- |
| /\*\*  \* 订阅全部的市场对  \*/  subscribeAllMarkets() {  this.$socket.subscribe("market.ticker", 'all-market-area');  this.$socket.on('all-market-area', (data) => {  // console.log("全部市场订阅", data)  if (data.markets) {  let filteredFavorites = data.markets.filter((item) => {  let itemSymbol = item.symbol.toLowerCase();  return this.userFavorites.indexOf(itemSymbol) !== -1;  })  // console.log("过滤后的", filteredFavorites)  let len = this.marketList.length;  this.marketList[len - 1].markets = filteredFavorites;  }  });  }, |

....

# 接入Tio

https://www.tiocloud.com/2/product/tio.html



## 3.1 添加tio 依赖

在coin-exchange的pom.xml 文件里面添加版本控制

|  |
| --- |
| <tio.version>3.5.5.v20191010-RELEASE</tio.version> |
| <dependencyManagement>  <dependencies>  ...  <dependency>  <groupId>org.t-io</groupId>  <artifactId>tio-websocket-spring-boot-starter</artifactId>  <version>${tio.version}</version>  </dependency>  ...  </dependencies> </dependencyManagement> |

修改chan-service的pom.xml文件,添加以下依赖

|  |
| --- |
| <dependency>  <groupId>org.t-io</groupId>  <artifactId>tio-websocket-spring-boot-starter</artifactId> </dependency> |

添加日期的工具类

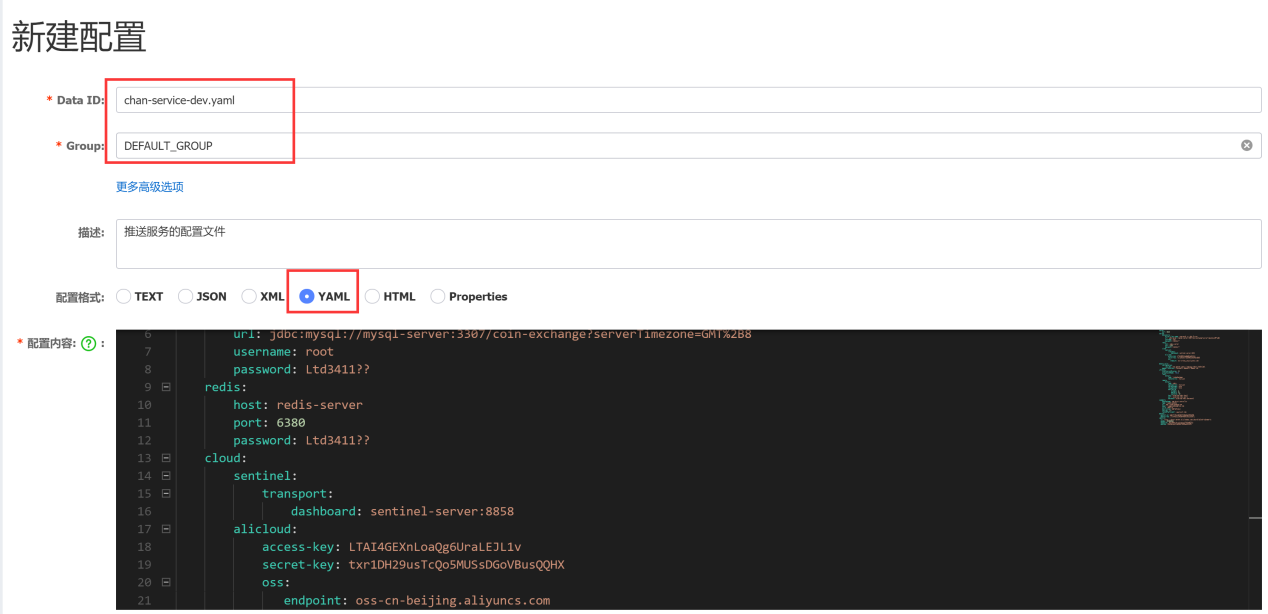
|  |
| --- |
| <dependency>  <groupId>joda-time</groupId>  <artifactId>joda-time</artifactId> </dependency> |

## 3.2 添加配置文件

### 3.2.1 本地 bootstrap.yml

|  |
| --- |
| spring:  application:  name: chan-service  cloud: *# nacos地址* nacos:  server-addr: nacos-server:8848  config:  file-extension: yaml  profiles:  active: dev *# 拉取的配置文件的dataID = chan-service-dev.yaml  # tio的配置文件* tio:  websocket:  server:  port: 8987 *# 监听的websocket端口* heartbeat-timeout: 600000 *# 心跳时间* cluster: *# 开启集群* enabled: true  redis: *# 集群基于Redis Pus/Sub* ip: redis-server  port: 6380  password: Ltd3411?? |

### 3.2.2 nacos-config的配置文件



内容为:

|  |
| --- |
| server:  port: 8040  spring:  datasource:  driver-class-name: com.mysql.cj.jdbc.Driver  url: jdbc:mysql://mysql-server:3307/coin-exchange?serverTimezone=GMT%2B8  username: root  password: Ltd3411??  redis:  host: redis-server  port: 6380  password: Ltd3411??  cloud:  sentinel:  transport:  dashboard: sentinel-server:8858  alicloud:  access-key: LTAI4GEXnLoaQg6UraLEJL1v  secret-key: txr1DH29usTcQo5MUSsDGoVBusQQHX  oss:  endpoint: oss-cn-beijing.aliyuncs.com  mybatis-plus:  configuration:  log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  mapper-locations: classpath:/mappers/\*Mapper.xml  jetcache:  statIntervalMinutes: 15  areaInCacheName: false  local:  default:  type: linkedhashmap  keyConvertor: fastjson  remote:  default:  type: redis  keyConvertor: fastjson  valueEncoder: kryo  valueDecoder: kryo  poolConfig:  minIdle: 5  maxIdle: 20  maxTotal: 50  host: ${spring.redis.host}  port: ${spring.redis.port}  password: ${spring.redis.password}  swagger2:  basePackage: com.bjsxt.controller  name: liangtiandong  url: www.liangtiandong.com  email: liangtiandong@live.com  title: 推送API接口  description: 推送系统API接口演示  version: 1.0  termsOfServiceUrl: www.bjsxt.com  geetest:  geetest-id: 3a01ffc01c1d63b37c3dbe8ee9555290  geetest-key: 27c7b4a18124d5d649b9c58ca1830871  identify:  url: https://idcert.market.alicloudapi.com/idcard?idCard=%s&name=%s  appKey: 203866940  appSecret: KF6LRDKejSejx2vfyLqiLP7bVvWOZYCp  appCode: 39e89acacec14a9d80782d8aa2893295 |

## 3.3 添加启动类

|  |
| --- |
| @SpringBootApplication @EnableTioWebSocketServer *// 开启tio的websocket* @EnableScheduling public class ChanServiceApplication {   @Autowired  private TioWebSocketServerBootstrap bootstrap ;   public static void main(String[] args) {  SpringApplication.*run*(ChanServiceApplication.class ,args) ;  }  } |

## 3.4 添加消息处理的Handler

|  |
| --- |
| @Component @Slf4j public class WebSocketMessageHandler implements IWsMsgHandler {   */\*\*  \* 握手时走这个方法，业务可以在这里获取cookie，request参数等  \*/* @Override  public HttpResponse handshake(HttpRequest request, HttpResponse httpResponse, ChannelContext channelContext) throws Exception {  String clientip = request.getClientIp();  *log*.info("收到来自{}的ws握手包\r\n{}", clientip, request.toString());  return httpResponse;  }   */\*\*  \* 握手成功后走的方法  \*  \* @param httpRequest  \* @param httpResponse  \* @param channelContext  \* @throws Exception  \*/* @Override  public void onAfterHandshaked(HttpRequest httpRequest, HttpResponse httpResponse, ChannelContext channelContext) throws Exception {  }   */\*\*  \* 字节消息（binaryType = arraybuffer）过来后会走这个方法  \*/* @Override  public Object onBytes(WsRequest wsRequest, byte[] bytes, ChannelContext channelContext) throws Exception {  return null;  }   */\*\*  \* 当客户端发close flag时，会走这个方法  \*/* @Override  public Object onClose(WsRequest wsRequest, byte[] bytes, ChannelContext channelContext) throws Exception {  Tio.*remove*(channelContext, "receive close flag");  return null;  }   */\*  \* 字符消息（binaryType = blob）过来后会走这个方法  \*/* @Override  public Object onText(WsRequest wsRequest, String text, ChannelContext channelContext) throws Exception {  WsSessionContext wsSessionContext = (WsSessionContext) channelContext.get();  HttpRequest httpRequest = wsSessionContext.getHandshakeRequest();*//获取websocket握手包  /\* if (log.isDebugEnabled()) {  log.debug("握手包:{}", httpRequest);  }\*/  //log.info("收到ws消息:{}", text);* if (Objects.*equals*("ping", text)) {  return "pong";  }  *// 订阅消息 ch 为订阅的频道 id 为订阅唯一标识* JSONObject payload = JSONObject.*parseObject*(text);  String sub = payload.getString("sub");  String req = payload.getString("req");  String cancel = payload.getString("cancel");  String id = payload.getString("id");   *//如果用户的authorization 数据* String authorization = payload.getString("authorization");  if (StringUtils.*hasText*(authorization)) {  JSONObject jsonObject = JSON.*parseObject*(JwtHelper.*decode*(authorization).getClaims());  String userId = "";*// jsonObject.get() ;* Tio.*bindUser*(channelContext, userId);   }    if (!Strings.*isNullOrEmpty*(sub)) {  *//绑定到群组，后面会有群发* Tio.*bindGroup*(channelContext, sub); *// 绑定群  //返回值是要发送给客户端的内容，一般都是返回null* return new ResponseEntity()  .setId(id)  .setSubbed(sub)  .setStatus("ok")  .build();  } else if (!Strings.*isNullOrEmpty*(req)) {   } else if (!Strings.*isNullOrEmpty*(cancel)) {  *//取消订阅通道* Tio.*unbindGroup*(cancel, channelContext);  return new ResponseEntity()  .setId(id)  .setCanceled(cancel)  .setStatus("ok")  .build();  }  return null;  } } |

## 3.5 启动测试

打开: <http://www.websocket-test.com/>



## 3.4 推送消息测试

### 3.4.1 添加定时任务

打开启动类:ChanServiceApplication:



|  |
| --- |
| */\*\*  \* 每5 s 给前端主动推送数据  \*/* @Scheduled(fixedRate = 5000) public void pushData(){   long time = new Date().getTime();  *// 使用o 推送数据非常简单* Tio.*sendToGroup*(bootstrap.getServerTioConfig(),"test", WsResponse.*fromText*("现在是:"+time , "utf-8")); } |

### 3.4.2 重启观察效果



# 接入Rocketmq

## 4.1 添加依赖

|  |
| --- |
| <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-stream-binder-rocketmq</artifactId> </dependency> |

## 4.2 添加配置文件

|  |
| --- |
| spring:  application:  name: chan-service  cloud: *# nacos地址* nacos:  server-addr: nacos-server:8848  config:  file-extension: yaml   **stream:  bindings:  tio\_group: {destination: tio\_group, content-type: text/plain, group: order-group, consumer.maxAttempts: 1}  rocketmq:  binder:  name-server: rocket-server:9876**  profiles:  active: dev *# 拉取的配置文件的dataID = chan-service-dev.yaml  # tio的配置文件* tio:  websocket:  server:  port: 8987 *# 监听的websocket端口* heartbeat-timeout: 600000 *# 心跳时间* cluster: *# 开启集群* enabled: true  redis: *# 集群基于Redis Pus/Sub* ip: redis-server  port: 6380  password: Ltd3411?? |

## 4.3 添加Sink

|  |
| --- |
| public interface Sink {   @Input("tio\_group")  MessageChannel messageGroupChannel() ; } |

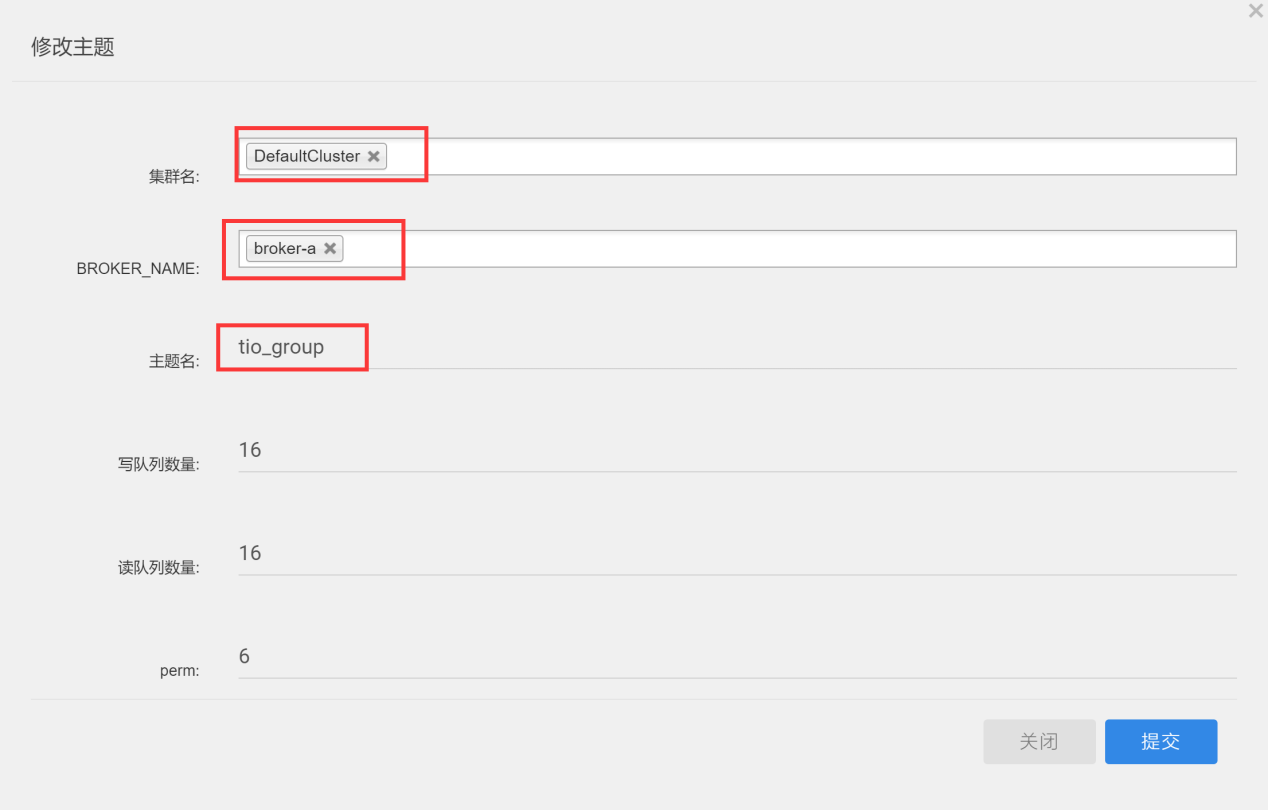
## 4.4 添加监听器

|  |
| --- |
| @Component @Slf4j public class RocketMessageListener {   @Autowired  private TioWebSocketServerBootstrap tioWebSocketServerBootstrap ;   @StreamListener("tio\_group")  public void handlerMessage(String message){  *log*.info("接收到rocketmq的消息========>{}", JSON.*toJSONString*(message));  *// 推送给前端的用户就可以了* Tio.*sendToGroup*(tioWebSocketServerBootstrap.getServerTioConfig(),"test", WsResponse.*fromText*("从rocketmq里面得到了数据:"+message,"UTF-8"));  } } |

## 4.5 测试数据的流动

### 4.5.1 在Rocketmq 里面新建主题



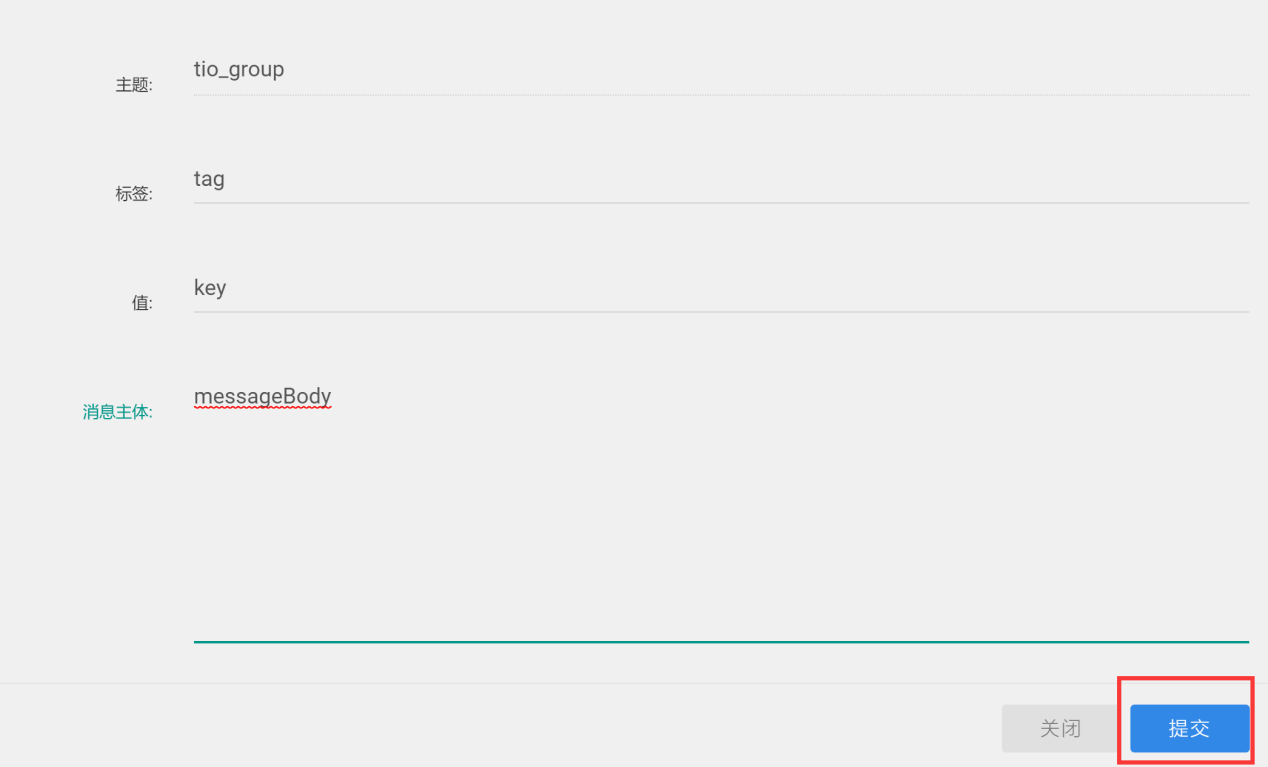


### 4.5.2 在websocket测试网页建立连接

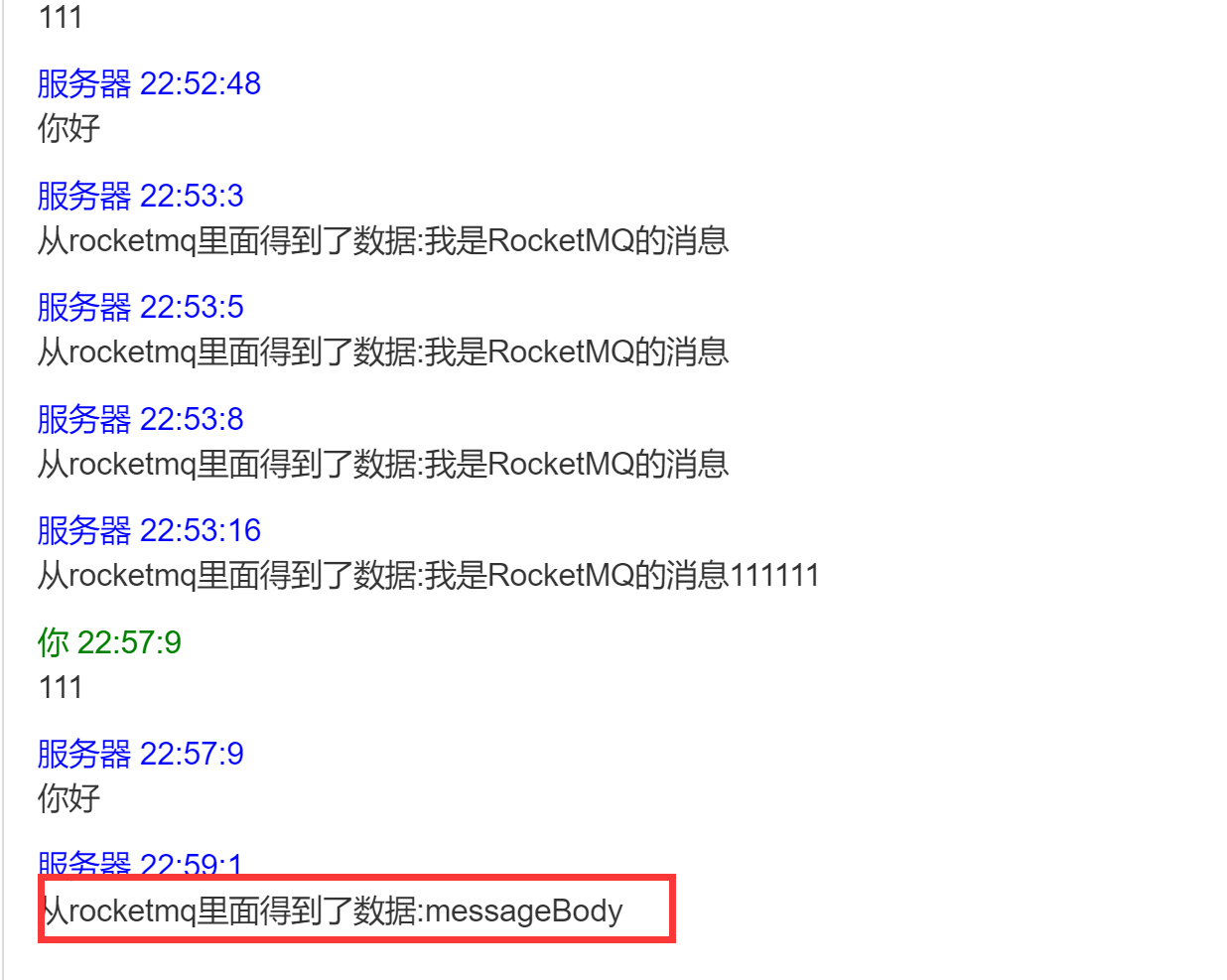


### 4.5.3 在mq的队列里面发送消息





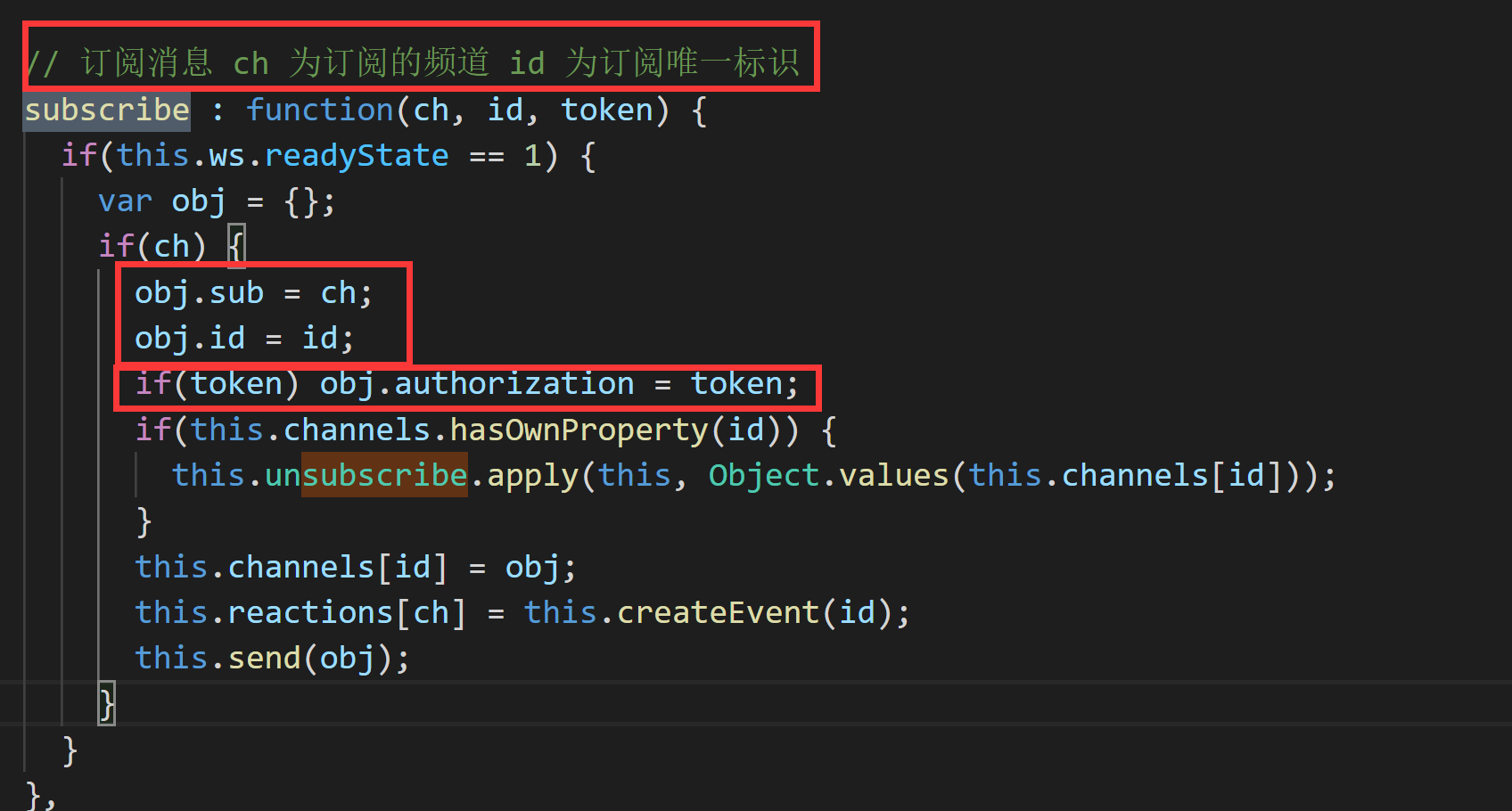
### 4.5.4 观察前端



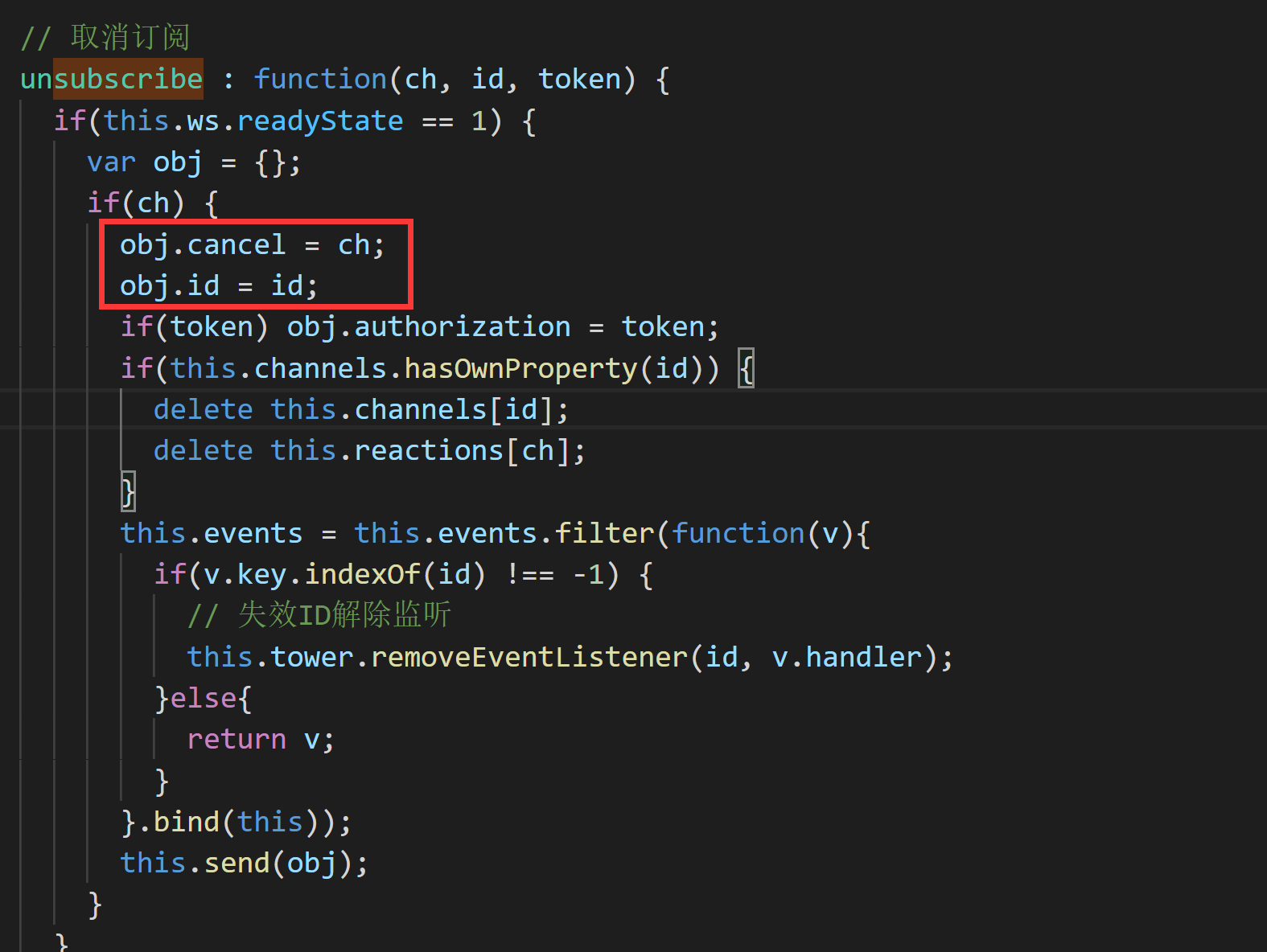
# 适配前端的订阅请求

## 5.1 前端的订阅请求数据

订阅:



退出订阅:



数据模型为:

|  |
| --- |
| {  "sub": 订阅的组  "req": 请求(预留的)  "cancel": 取消订阅的组  "id": 订阅的唯一标识(无用)  "authorization": 用户的token  } |

## 5.2 响应给前端的数据

|  |
| --- |
| {  "id" 推送的id,  "ch": 推送的channel,  "status":状态,  "subbed": 订阅的组名 ,  "canceled": 取消订阅的组名,  "event": 发送的事件,  "ts": 时间戳,  "result": 响应的结果,  } |

将他封装成一个对象:

|  |
| --- |
| @Data @Accessors(chain = true) @AllArgsConstructor @NoArgsConstructor public class ResponseEntity {   private String subbed;   private String canceled;   private String event;   private String id;   private String ch;   private String status;   public Long getTs() {  return new DateTime().getMillis();  }   private Long ts;   private Map<String, Object> extend = new LinkedHashMap<>();   public WsResponse build() {  extend.put("id", this.getId());  extend.put("ch", this.getCh());  extend.put("status", this.getStatus());  extend.put("subbed", this.getSubbed());  extend.put("canceled", this.getCanceled());  extend.put("event", this.getEvent());  extend.put("ts", this.getTs());  return WsResponse.*fromText*(JSONObject.*toJSONString*(extend), "utf-8");  }   public ResponseEntity put(String key, Object value) {  extend.put(key, value);  return this;  }   public ResponseEntity putAll(Map<String, Object> m) {  extend.putAll(m);  return this;  } } |

## 5.3 后台解析请求订阅和取消订阅

|  |
| --- |
| */\*\*  \* 当前端发送文本过来  \*  \* @param wsRequest  \* @param text  \* @param channelContext  \* @return  \* @throws Exception  \*/* @Override public Object onText(WsRequest wsRequest, String text, ChannelContext channelContext) throws Exception {   if (Objects.*equals*("ping", text)) {  return "pong";  }   JSONObject payload = JSONObject.*parseObject*(text);  String sub = payload.getString("sub");  String req = payload.getString("req");  String cancel = payload.getString("cancel");  String id = payload.getString("id");   *//如果用户已登录，同时绑定用户* String authorization = payload.getString("authorization");  if (!Strings.*isNullOrEmpty*(authorization) && authorization.startsWith("bearer ")) {  String accessToken = authorization.replaceAll("bearer ", "");  *// 2 查询我们的菜单数据* Jwt jwt = JwtHelper.*decode*(accessToken);  String jwtJsonStr = jwt.getClaims();  JSONObject jwtJson = JSON.*parseObject*(jwtJsonStr);  String userId = jwtJson.getString("user\_name");  Tio.*bindUser*(channelContext, userId); *// 绑定用户* }   if (!Strings.*isNullOrEmpty*(sub)) {  *//绑定到群组，后面会有群发* Tio.*bindGroup*(channelContext, sub);  *//返回值是要发送给客户端的内容，一般都是返回null* return new ResponseEntity()  .setId(id)  .setSubbed(sub)  .setStatus("ok")  .build();  } else if (!Strings.*isNullOrEmpty*(req)) {   } else if (!Strings.*isNullOrEmpty*(cancel)) {  *//取消订阅通道* Tio.*unbindGroup*(cancel, channelContext);  return new ResponseEntity()  .setId(id)  .setCanceled(cancel)  .setStatus("ok")  .build();  }  return null; } |

## 5.4 使用对象接收MQ里面的消息

|  |
| --- |
| @Data @AllArgsConstructor @NoArgsConstructor public class MessagePayload {   */\*\*  \* 用户的id  \*/* private String userId;    */\*\*  \* 订阅的组名称  \*/* @NonNull  private String channel;   */\*\*  \* 消息的内容  \*/* @NonNull  private String body; } |

## 5.5 修改为MessagePayload 监听

|  |
| --- |
| @Component @Slf4j public class RocketMessageListener {   @Autowired  private TioWebSocketServerBootstrap tioWebSocketServerBootstrap;   @StreamListener("tio\_group")  public void handlerMessage(**MessagePayload message**) {  *log*.info("接收到rocketmq的消息========>{}", JSON.*toJSONString*(message));  ResponseEntity responseEntity = new ResponseEntity();  responseEntity.setSubbed(message.getChannel());  responseEntity.put("result", message.getBody());  *// 推送给前端的用户就可以了* if (StringUtils.*hasText*(message.getUserId())) {  Tio.*sendToUser*(tioWebSocketServerBootstrap.getServerTioConfig(), message.getUserId(), responseEntity.build());  return;  }  @NonNull String group = message.getChannel();  Tio.*sendToGroup*(tioWebSocketServerBootstrap.getServerTioConfig(), group, responseEntity.build());  } } |