# 柜门关闭

接收到mq门关了的消息**public** **boolean** process(**byte**[] bytes, String deviceId, String eventName) {

**try** {

**if**("DOOR.CLOSED".equals(eventName)) {

DoorClosedMessage doorClosedMessage=objectMapper.readValue(bytes, **new** TypeReference<DoorClosedMessage>() {});

deviceBlh.doorClosed(doorClosedMessage);

**return** **true**;

}

调用public void doorClosed(DoorClosedMessage doorClosedMessage) {

获取li设备id

String liIds = doorClosedMessage.getDeviceId();

通过li去查eqpad

Device eqPad = deviceService.findEqpadByLiId(liIds);

调用getRedisKey（）会将rediskey变成eqPad- openDoor

String redisKey = getRedisKey(eqPad.getIdentification(), "openDoor");

判断redis里面是否有当前开柜的柜子的rediskey

Boolean scanable = false;

**if** (redisTemplate.hasKey(redisKey)) {

如果有，遍历rediskey（eqPad- openDoor）因为主辅柜原因，redis里存的数据应该是（eqPad- openDoor ：key：value1，eqPad- openDoor ：key：value2…这种形式），key为lis和lisClone（猜测，lis是多个li标识拼接起来的字符串，lisClone为多个lisClone拼接起来的字符串）

Map<Object, Object> map = redisTemplate.opsForHash().entries(redisKey);

将map里面存放的key为lis的value（应该是li1li2…拼接起来的字符串）取出并通过，将value分割为数组然后转为集合，redis里存储的全是已经开柜的柜子的li标识

List<String> lis = **new** ArrayList<>(Arrays.*asList*(((String) map.get("lis")).split(",")));

如果集合不为空并且包含当前liIds（柜子关门传过来的li标识），就移除当前liIds，表示该柜子已关闭

**if** (**null** != lis && lis.contains(liIds)) {

lis.remove(liIds);

}

如果集合为空表示没有柜子的门是开着的，删除rediskey

**if** (lis.isEmpty()) {

// 如果所有开过的门都已关闭，则开始启动reader扫描

scanable = **true**;

liIds = (String)map.get("lisClone");

redisTemplate.delete(redisKey);

} 如果不为空就往map里存数据，key为lis，value为用，分割开的li标识，此时的map是删除传过来的li标识后的集合

**else** {

map.put("lis", String.*join*(",", lis));

redisTemplate.opsForHash().putAll(redisKey, map);

}

} **else** {

scanable = **true**;

}

如果全部关闭调用doscan方法，并发送socket消息

**if** (scanable) {

// 告知前端此次领耗材打开过的柜子的Li模块identification，以便reScan（重新扫描）

redisTemplate.opsForValue().set("socketio-event", "browser:scan-door-state");

PushMessageDto pushMessageDto = **new** PushMessageDto();

pushMessageDto.setClientId(eqPad.getIdentification());

pushMessageDto.setEvent("browser:scan-door-state");

pushMessageDto.setData(liIds);

socketClient.sendMessageRest(pushMessageDto);

// 启动所有打开并关闭了的柜子的reader扫描

List<Device> readers = deviceService.findMultiCabinetReadersByLiId(**new** ArrayList<>(Arrays.*asList*((liIds.split(",")))));

doScan(eqPad.getIdentification(), **null**, readers);

} **else** {

PushMessageDto pushMessageDto = **new** PushMessageDto();

pushMessageDto.setClientId(eqPad.getIdentification());

pushMessageDto.setEvent("browser:server-notify");

SocketMessageVo socketMessageVo = **new** SocketMessageVo();

socketMessageVo.setMessage("您当前还有柜门未关闭，必须关闭所有柜门才能启动扫描");

socketMessageVo.setType(SocketMessageVo.***TYPE\_INFO***);

pushMessageDto.setData(socketMessageVo);

socketClient.sendMessageRest(pushMessageDto);

}

}

# 扫描

**private** **boolean** doScan(String eqId, String storeCode, List<Device> readers) {

如果reader集合不为空

**if**(**null** != readers && !readers.isEmpty()) {

// 调用readers扫描

List<String> readerIds = readers.stream().map(reader -> reader.getIdentification()).collect(Collectors.*toList*());

*logger*.info("doScan:readers["+readerIds.toString()+"]正在扫描...");

调用硬件扫描

ResponseModel responseModel = deviceClient.MutilRealtimeInventory(readerIds, scanTimes.byteValue(), eqId);

**if** (ErrorInfo.***SUCCESS*** == responseModel.getCode()) {

// 将本次启动的readers的identification记录到redis，等待rabbiMQ返回扫描数据后在scanResult方法中处理扫描结果

Map<String, Object> map = **new** HashMap<String, Object>() {

**private** **static** **final** **long** ***serialVersionUID*** = 7774095146996163048L;

{put("readers", String.*join*(",", readerIds));}

};

**if** (StringUtils.*isNotBlank*(storeCode)) {

map.put("storeCode", storeCode);

}

String redisKey = getRedisKey(eqId, responseModel.getRequestId());

redisTemplate.opsForHash().putAll(redisKey, map);

*logger*.info("doScan:redisTemplate存入key=["+redisKey+"]的数据");

// 发送延时消息，如果超过设置的时间，就告诉前端超时了

MessagePostProcessor processor = **new** MessagePostProcessor() {

@Override

**public** Message postProcessMessage(Message message) **throws** AmqpException {

message.getMessageProperties().setExpiration("5000");

**return** message;

}

};

rabbitTemplate.convertAndSend(exchangeName, "delay", redisKey.getBytes(), processor);

}**else** {

*logger*.error("doScan:通知硬件控制器扫描的时候出错,错误信息:"+responseModel.getMessage());

**return** **false**;

}

}**else** {

*logger*.error("doScan:没找到对应reader");

**return** **false**;

}

**return** **true**;

}