BlotMQ-Client

技术文档说明

针对版本V1.0.0

2017/11/21

BoltMQ-broker 修订记录

版本号	修订内容	作者	审核	修订日期
V1.0.0	初始版本	尹同强	基础平台架构组	2017/11/21

目 录

1 概述

Client 在发送端起发送负载作用，在消费端起消费负载作用，是整个消息中间件的入口和出口，只与 Registry 和 Broker 进行交互。

2 Client 模块交互

2.1 Registry

- Client 与 Registry 集群中随机一个保持长连接。
- 启动时会向 Registry 建立链接，启动过每隔 30 秒向 Registry 获取 topic 的路由信息并更新本地路由配置。

2.2 Broker

- 每个 Client 通过 Registry 拿到 BrokerList 地址，Client 与 BrokerList 保持长连接。
- Producer 通过路由信息轮询的向 Broker 每个队列发送消息（仅针对普通消息），。
- Consumer 从 Broker 拉取消息进行消费，Broker 会维护 Consumer 与 Topic 之间订阅关系，并且会维护与 Topic 消费的 Offset，主要是针对集群消息模式，广播消费模式 Topic 的 Offset 是存储在客户的。

2.3 Net

- Client 通过 Net 创建 Client，调用 Registry 和 Broker 的方法。

3 专业术语

3.1 **Producer**

消息生产者，负责产生消息，一般由业务系统负责产生消息。

3.2 **Consumer**

消息消费者，负责消费消息，一般是后台系统负责异步消费。

3.3 Push Consumer

Consumer 的一种 ,应用通常吐 Consumer 对象注册一个 Listener 接口 ,一旦收到消息 ,Consumer 对象立刻回调 Listener 接口方法。

3.4 Pull Consumer

Consumer 的一种 ,应用通常主劢调用 Consumer 的拉消息方法从 Broker 拉消息 ,主劢权由应用控制。

3.5 Producer Group

一类 Producer 的集合名称，这类 Producer 通常发送一类消息，且发送逻辑一致。

3.6 Consumer Group

一类 Consumer 的集合名称，这类 Consumer 通常消费一类消息，且消费逻辑一致。

3.7 广播消费

一条消息被多个 Consumer 消费，即使返些 Consumer 属于同一个 Consumer Group，消息也会被 Consumer Group 中的每个 Consumer 都消费一次，广播消费中的 Consumer Group 概念可以认为在消息划分方面无意义。在 CORBA Notification 规范中，消费方式都属于广播消费。在 JMS 规范中，相当于 JMS publish/subscribe model。

3.8 集群消费

一个 Consumer Group 中的 Consumer 实例平均分摊消费消息。例如某个 Topic 有 9 条消息，其中一个 Consumer Group 有 3 个实例（可能是 3 个进程，或者 3 台机器），那么每个实例只消费其中的 3 条消息。在 CORBA Notification 规范中，无此消费方式。在 JMS 规范中，JMS point-to-point model 与之类似，但是 BoltMQ 的集群消费功能大等于 PTP 模型。因为 BoltMQ 单个 Consumer Group 内的消费者类似于 PTP，但是一个 Topic/Queue 可以被多个 Consumer Group 消费。

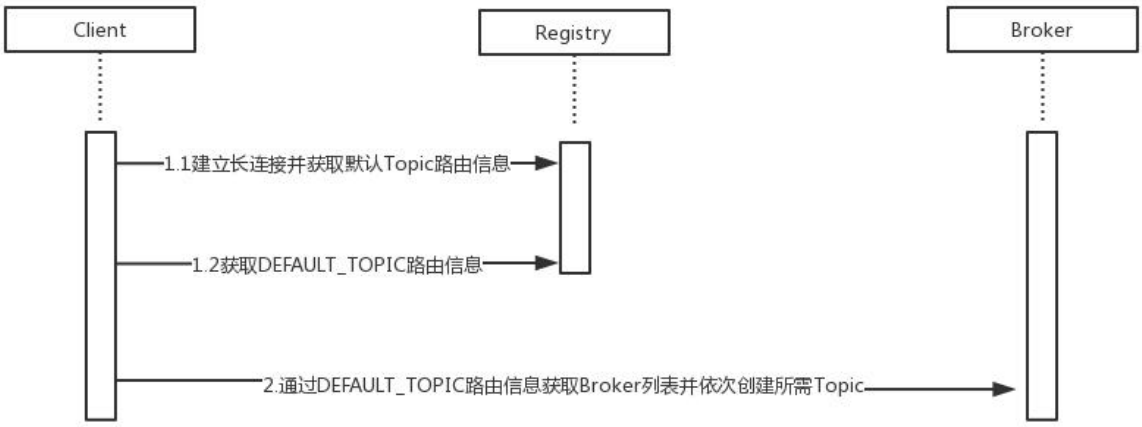
3.9 顺序消息

敬请期待

4 Client 实现原理

4.1 创建 Topic

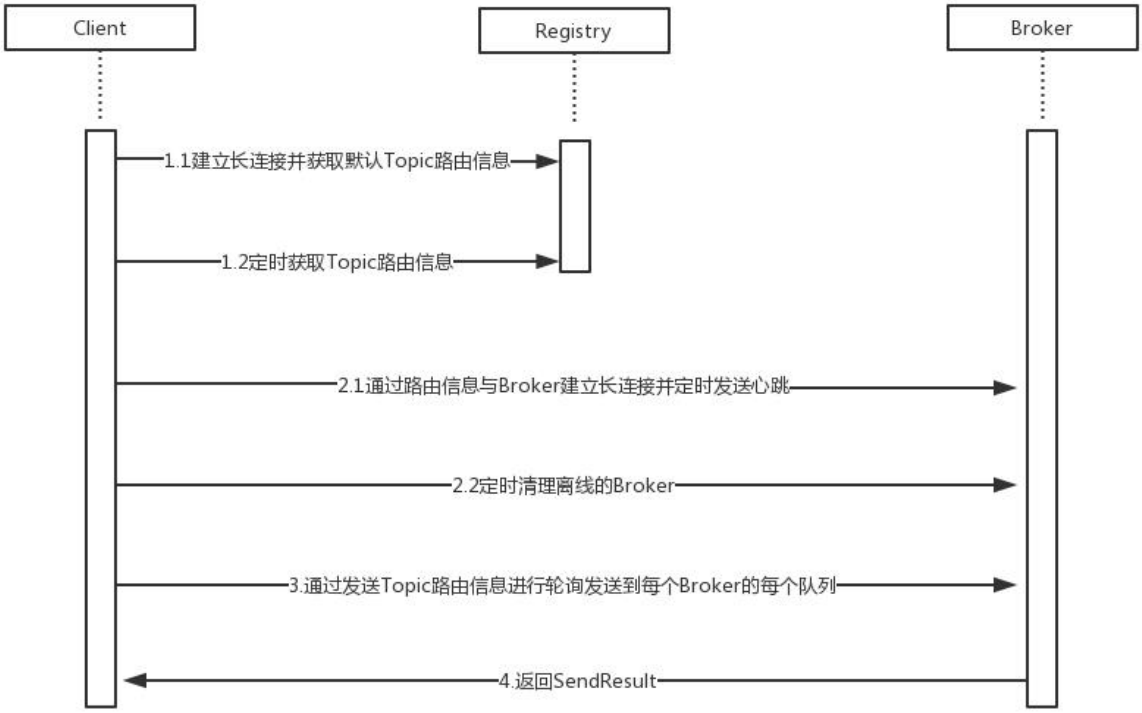
Client 创建 Topic 时序图如下：



每个 Broker 启动时会向 Registry 注册一个 DEFAULT_TOPIC 信息，当客户端创建 Topic 时，通过从 Registry 拿 DEFAULT_TOPIC 路由信息即可拿到集群所有 Broker 列表 ,然后依次调用 Broker 创建 Topic 接口就在 Broker 上创建了该 Topic。

4.2 发送同步消息

发送同步消息时序图如下：



发送消息负载客户端从registry拿到topic对应所有broker的所有队列依次遍历队列发送到每个队列 ,从而保证了发送端负载。

SendResult 中 SendStatus 值说明

SEND_OK 发送成功并同步到 SLAVE 成功

FLUSH_DISK_TIMEOUT 刷盘超时

FLUSH_SLAVE_TIMEOUT 同步到 SLAVE 超时

SLAVE_NOT_AVAILABLE SLAVE 不可用

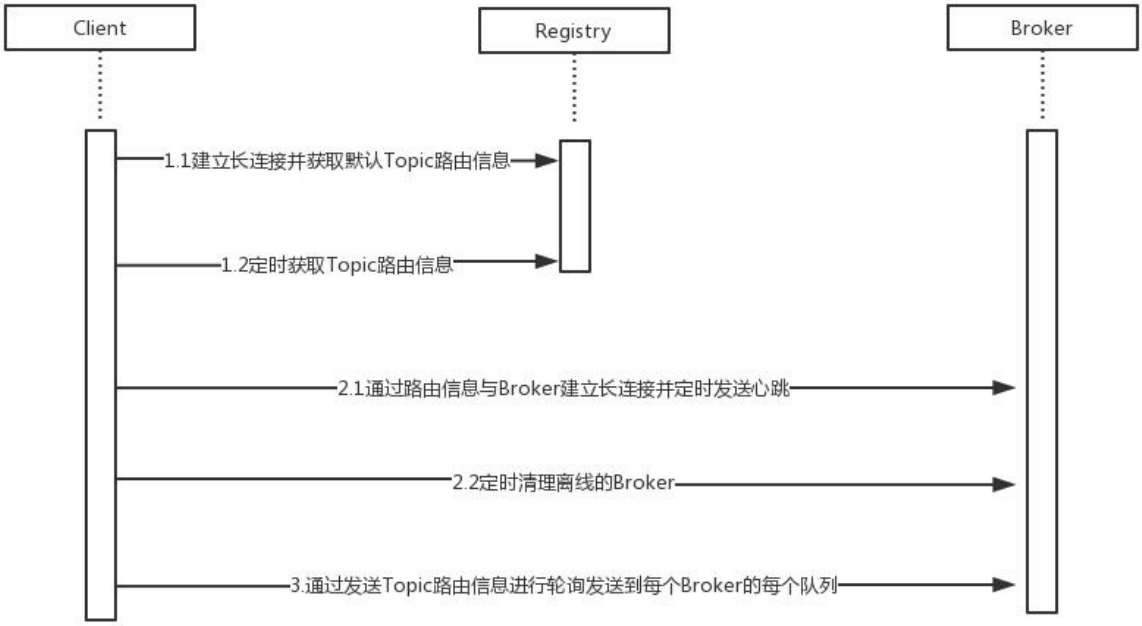
4.3 发送异步消息

发送异步消息时序图如下：

异步消息和同步消息流程几乎是一模一样只是在返回SendResult时，客户端不需要等待只需传入一个回调函数，服务端处理发送消息成功即通过回调函数返回SendResult给客户端。

4.4 发送 OneWay 消息

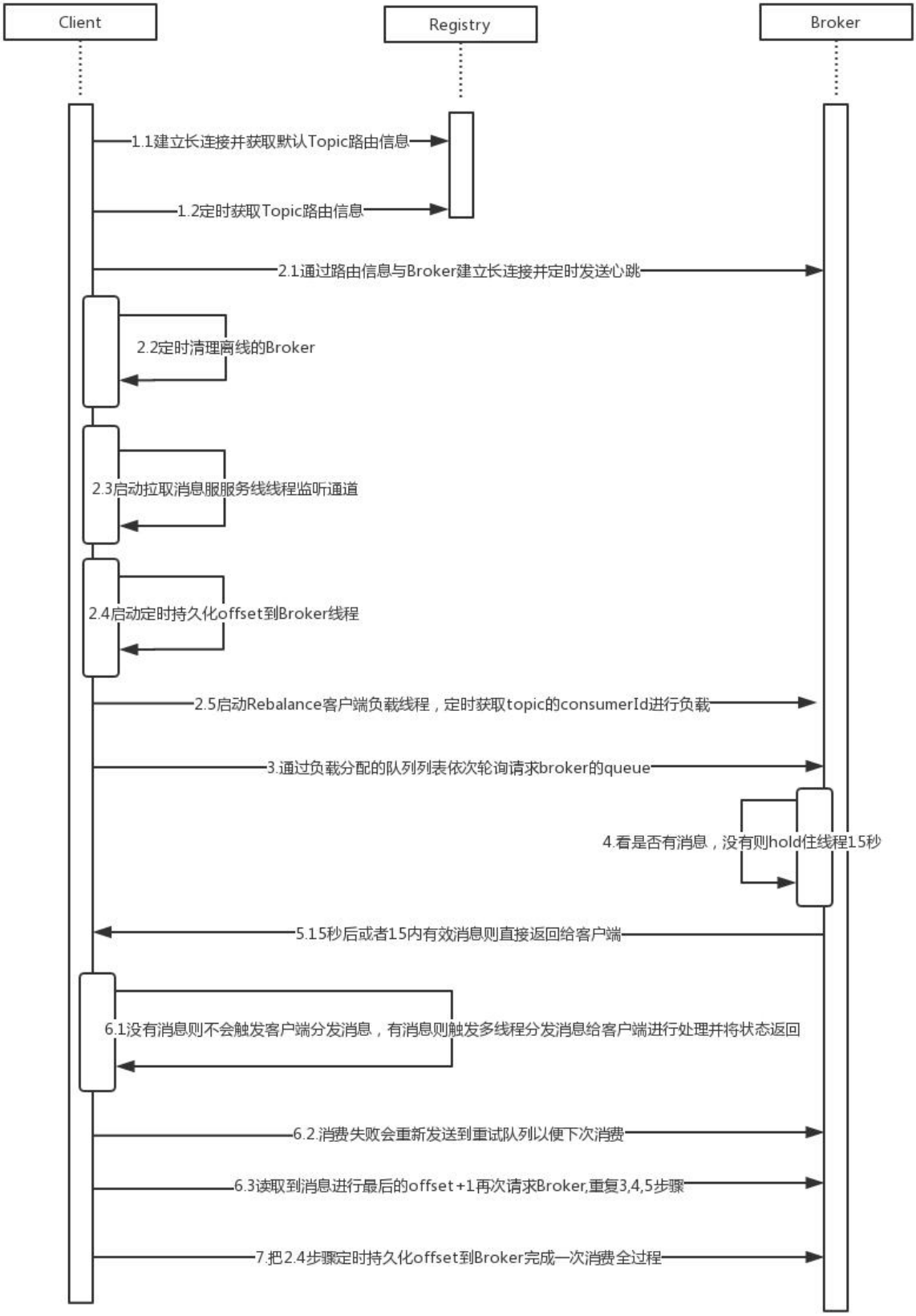
发送OneWay消息时序图如下：



发送OneWay消息和发送异步消息，同步消息类似，只是服务端没有返回值，不清楚服务端是否成功，此模式是性能最高的，但消息可靠性不能保证。

4.5 Push 集群消费

Push消费流程比较繁琐，时序图如下：



Push集群消费默认负载算法是按照ConsumerId平均分配队列。举个例子，一个Topic 24个队列，开1个客

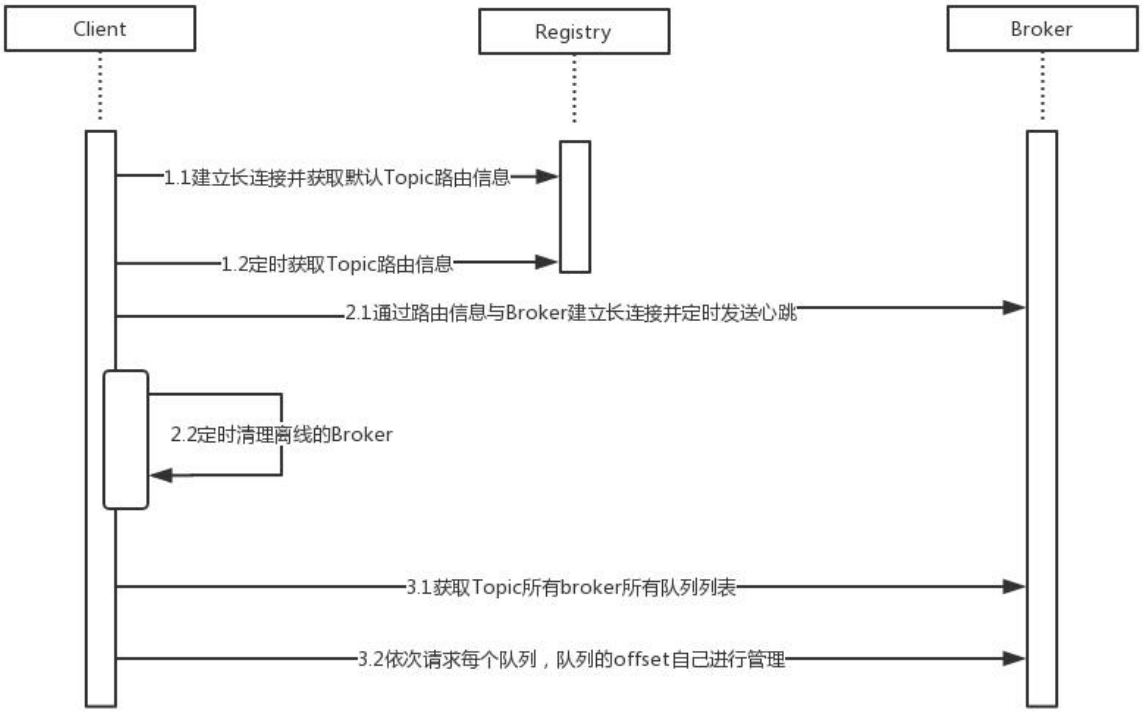
户端消费则这个客户端订阅这个topic 24个队列，如果开2个客户端消费，则每个客户端订阅这个topic 24个队列中12个队列，从而达到客户端消费负载。客户端可以重写这个负载策略。

4.6 Push 广播消费

流程和push集群消费类似，仅持久化offset时存在本地，负载对广播消费不起作用。

4.7 Pull 消费

Pull消息时序图如下：



Pull消费主要是客户端控制，offset客户端完全是自己管理，所以没有集群消费和广播消费。

5 Client 最佳实践

5.1 Producer 最佳实践

- **发送消息注意事项**

1. 一个应用尽可能用一个 Topic，消息子类型用 tags 来标识，tags 可以由应用自由设置。只有发送消息设置了tags，消费方在订阅消息时，才可以利用 tags 在 broker 做消息过滤。

   message.setTags("TagA");

2. 每个消息在业务局面的唯一标识码，要设置到 keys 字段，方便将来定位消息丢失问题。服务器会为个消息创建索引（哈希索引），应用可以通过 topic，key 来查询返条消息内容，以及消息被谁消费。由于是哈希索引，请务必保证 key 尽可能唯一，返样可以避免潜在的哈希冲突。

   // 订单 Id

   String orderId = "20034568923546";

   message.setKeys(orderId);

3. 消息发送成功或者失败，要打印消息日志，务必要打印 sendresult 和 key 字段。

4. send 消息方法，只要不抛异常，就代表发送成功。但是发送成功会有多个状态，在sendResult里定义。

   SEND_OK

   消息叹送成功

   FLUSH_DISK_TIMEOUT

   消息叹送成功，但是服务器刷盘超时，消息已经迮入服务器队列，只有此时服务器宕机，消息才会丢失。

   FLUSH_SLAVE_TIMEOUT

   消息叹送成功，但是服务器同步到Slave时超时，消息已经迮入服务器队列，只有此时服务器宕机，消息才会丢失。

   SLAVE_NOT_AVAILABLE

消息收送成功，但是此时slave不可用，消息已经迮入服务器队列，只有此时服务器宕机，消息才会丢失。

对于精卫发送顺序消息的应用，由于顺序消息的局限性，可能会涉及到主备自动切换问题，所以如果sendresult中的status字段不等于SEND_OK，就应该尝试重试。对于其他应用，则没有必要返样。

5. 对于消息不可丢失应用，务必要有消息重试机制,例如如果消息发送失败，存储到数据库，能有定时程序尝试重发，或者人工触发重发。

■ **消息发送失败如何处理**

Producer 的 send 方法本身支持内部重试，重试逻辑如下：

1. 至多重试 3 次。
2. 如果发送失败，则轮转到下一个 Broker。
3. 返个方法的总耗时时间不超过 sendMsgTimeout 设置的值，默认 10s。

所以，如果本身向broker发送消息产生超时异常，就不会再做重试。

以上策略仍然不能能保证消息一定收送成功，为保证消息一定成功，建议应用返样做，如果调用 send 同步方法发送失败，则尝试将消息存储到 db，由后台线程定时重试，保证消息一定到达 Broker。

上述db重试方式为什举没有集成到MQ客户端内部做，而是要求应用自己去完成，我们基于以下几点考虑：

1. MQ的客户端设计为无状态模式，方便任意的水平扩展，且对机器资源的消耗仅仅是cpu、内存、网络。
2. 如果 MQ 客户端内部集成一个 KV 存储模块，那举数据只有同步落盘才能较可靠，而同步落盘本身性能开销较大，所以通常会采用异步落盘，又由于应用关闭过程不受MQ运维人员控制，可能经常会收生 kill -9 返样暴力方式关闭，造成数据没有及时落盘而丢失。
3. Producer 所在机器的可靠性较低，一般为虚拟机，不适合存储重要数据。

综上，建议重试过程交由应用来控制。

- **选择 oneway 形式发送**

一个 RPC 调用，通常是返样一个过程

1. 客户端收送请求到服务器
2. 服务器处理该请求
3. 服务器向客户端返回应答

所以一个 RPC 的耗时时间是上述三个步骤的总和，而某些场景要求耗时非常短，但是对可靠性要求并不高，例如日志收集类应用，此类应用可以采用 oneway 形式调用，oneway 形式只收送请求不等待应答，而收送请求在客户端实现局面仅仅是一个 os 系统调用的开销，即将数据写入客户端的 socket 缓冲区，此过程耗时通常在微秒。

5.2 Consumer 最佳实践

- **消费过程要做到幂等（即消费端去重）**

BoltMQ 无法避免消息重复，所以如果业务对消费重复非常敏感，务必要在业务局面去重，有以下几种去重方式。

1. 将消息的唯一键，可以是msgId，也可以是消息内容中的唯一标识字段，例如订单 Id 等，消费之前判断是否在Db 或全局KV存储中存在，如果不存在则插入，并消费，否则跳过。（实际过程要考虑原子性问题，判断是否存在可以尝试插入，如果报主键冲突，则插入失败，直接跳过）msgId一定是全局唯一标识符，但是可能会存在同样的消息有两个不同msgId的情冴（有多种原因），返种情况可能会使业务上重复消费，建议最好使用消息内容中的唯一标识字段去重。
2. 使用业务局面的状态机去重。

附件一　BoltMQ 开发者联系方式

姓名	联系方式	更新日期
郜焱磊	gyl_adaihao@163.com	2017/11/21
田玉粮		2017/11/21
尹同强	26026193@qq.com	2017/11/21
罗继		2017/11/21
周飞		
戎志宏		