2011/1/4

中间件

NOTIFY

NOTIFY 客户端使用说明文档

目录

一、发布消息

所有例子见附录：Notify 客户端例子。NotifyManager 可以通过 spring 注入或直接 new 一个实例使用。该发布消息例子见 notify-example-hsf/com.taobao.notify.comexample/PublisherInstance.java

1、发布消息的方式

（1）同步发送非事务消息

例子见 PublisherInstance.java 的 sendNoTransactionMessage()方法。SendResult 返回消息发送成功与否的信息，isSuccess 判断成功与否。

```
    private static void sendNoTransactionMessage(NotifyManager notifyManager)
{
        StringMessage stringMessage = new StringMessage();

        stringMessage.setBody("NoTranscationTestStringMessage");
        /**
         * 必填属性
         */
        stringMessage.setTopic(TOPIC);
        stringMessage.setMessageType(MESSAGE_TYPE);
        /**
         * 可选属性，使用前，请务必仔细看Notify的User guide的Message章节
         */
        stringMessage.setSendOnceMessage(true);// 默认值为false
        stringMessage.setPostTimeout(10000);// 默认值10*1000毫秒
        stringMessage.setClientPostTimeout(3000);// 默认值3*1000毫秒
        stringMessage.setTimeToLive(7 * 24 * 3600);// 默认值，为-1
        stringMessage.setDLQTime(7 * 24 * 3600);// 默认值，NotifyServer配置文件决
定
        stringMessage.setStringProperty("customHeader", "customValue"); // 该
可选属性是一个系列
        SendResult result = notifyManager.sendMessage(stringMessage);
        if (result.isSuccess()) {
            // 发送成功后处理
            System.out.println("消息发送成功");
        }
        else {
            System.out.println("消息发送失败，原因： " + result.getErrorMessage());
        }
    }
```

（2）异步发送非事务消息

例子见 PublisherInstance.java 的 `asynSendNoTransactionMessage()` *方法。*

```
private static void asynSendNoTransactionMessage(NotifyManager
notifyPublisher) throws Exception {
        AsynSendResultListener asynListener = new SimpleAsynListener();

        StringMessage stringMessage = new StringMessage();

        stringMessage.setBody("NoTranscationTestStringMessage");

        /**
         * 必填属性
         */
        stringMessage.setTopic(TOPIC);
        stringMessage.setMessageType(MESSAGE_TYPE);
        stringMessage.setGroupId(GROUPID);
        /**
         * 可选属性，使用前，请务必仔细看Notify的User guide的Message章节
         */
        stringMessage.setSendOnceMessage(true);// 默认值为false
        stringMessage.setPostTimeout(10000);// 默认值10*1000毫秒
        stringMessage.setClientPostTimeout(3000);// 默认值3*1000毫秒
        stringMessage.setTimeToLive(7 * 24 * 3600);// 默认值，为-1
        stringMessage.setDLQTime(7 * 24 * 3600);// 默认值，NotifyServer配置文件决
定
        stringMessage.setStringProperty("customHeader", "customValue"); // 该
可选属性是一个系列
        // 可以使用Future方式或者异步Listener返回结果，二选一即可，不必同时使用两者

        Future<SendResult> result =
notifyPublisher.asynSendMessage(stringMessage, asynListener);
        if (result.get().isSuccess()) {
            // 发送成功后处理
            System.out.println("消息发送成功");
        }
        else {
            System.out.println("消息发送失败，原因： " +
result.get().getErrorMessage());
        }
    }
```

（3）同步发送事务消息

例子见 PublisherInstance.java 的 `sendTransactionMessage()` *方法。*

```
private static void sendTransactionMessage(NotifyManager notifyManager) {
        // 创建BytesMessage
        final BytesMessage bytesMessage = new BytesMessage();
        // 设置消息内容
        bytesMessage.setBody("TranscationTestBytesMessage".getBytes());
        // 设置消息其他信息

        bytesMessage.setTopic(TOPIC);

        bytesMessage.setMessageType(MESSAGE_TYPE);

        bytesMessage.setStringProperty("customHeader", "customValue");

        // 发送事务消息
        SendResult result = notifyManager.sendMessage(bytesMessage, new
SendMessageCallback() {
            public Object doInTransaction(MessageStatus status) {
                // 这个例子是一个跟业务操作相结合的例子
                // 这个地方抛出异常或者设置status.setRollbackOnly();会回滚half消息。
                // 如果这个还没有成功回滚，而应用或者NotifyServer Down掉，
                // NotifyServer会发送消息来Check这个Half的message
                // 使用和业务操作结合的方式发送消息，一定要注册一个CheckMessageListener
                return null;
            }
        });
        if (result.isSuccess()) {
            // 发送成功后处理
            System.out.println("消息发送成功");
        }
        else {
            System.out.println("消息发送失败，原因：" + result.getErrorMessage());
        }
    }
```

（4）异步发送事务消息

例子见 PublisherInstance.java 的 asynSendTransactionMessage() 方法。

```
private static void asynSendTransactionMessage(NotifyManager notifyPublisher)
throws Exception {
        AsynSendResultListener asynListener = new SimpleAsynListener();

        // 创建BytesMessage
        final BytesMessage bytesMessage = new BytesMessage();
        // 设置消息内容
        bytesMessage.setBody("TranscationTestBytesMessage".getBytes());
        // 设置消息其他信息

        bytesMessage.setTopic(TOPIC);
        bytesMessage.setMessageType(MESSAGE_TYPE);
        bytesMessage.setGroupId(GROUPID);

        bytesMessage.setStringProperty("customHeader", "customValue");

        // 发送事务，消息可以使用Future方式或者异步Listener返回结果，二选一即可，不必同
时使用两者
        Future<SendResult> result =
notifyPublisher.asynSendMessage(bytesMessage, new SendMessageCallback() {
            public Object doInTransaction(MessageStatus status) {
                // 这个例子是一个跟业务操作相结合的例子
                // 这个地方抛出异常或者设置status.setRollbackOnly();会回滚half消息。
                // 如果这个还没有成功回滚，而应用或者NotifyServer Down掉，
                // NotifyServer会发送消息来Check这个Half的message
                // 使用和业务操作结合的方式发送消息，一定要注册一个CheckMessageListener
                return null;
            }
        }, asynListener);
        if (result.get().isSuccess()) {
            // 发送成功后处理
            System.out.println("消息发送成功");
        }
        else {
            System.out.println("消息发送失败，原因: " +
result.get().getErrorMessage());
        }
    }
```

（5）可靠异步发送非事务消息

例子见 PublisherInstance.java 的 `reliableAsynSendNoTransactionMessage()` *方法。*

```
private static void reliableAsynSendNoTransactionMessage(NotifyManager
notifyPublisher) {
        StringMessage stringMessage = new StringMessage();

        stringMessage.setBody("NoTranscationTestStringMessage");

        /**
         * 必填属性
         */
        stringMessage.setTopic(TOPIC);
        stringMessage.setMessageType(MESSAGE_TYPE);
        stringMessage.setGroupId(GROUPID);

        /**
         * 可选属性，使用前，请务必仔细看Notify的User guide的Message章节
         */
        stringMessage.setSendOnceMessage(true);// 默认值为false
        stringMessage.setPostTimeout(10000);// 默认值10*1000毫秒
        stringMessage.setClientPostTimeout(3000);// 默认值3*1000毫秒
        stringMessage.setTimeToLive(7 * 24 * 3600);// 默认值，为-1
        stringMessage.setDLQTime(7 * 24 * 3600);// 默认值，NotifyServer配置文件决
定

        stringMessage.setStringProperty("customHeader", "customValue"); // 该
可选属性是一个系列

        SendResult result =
notifyPublisher.reliableAsynSendMessage(stringMessage);

        if (result.isSuccess()) {
            // 发送成功后处理
            System.out.println("消息发送成功");
        }
        else {
            System.out.println("消息发送失败，原因： " + result.getErrorMessage());
        }
    }
```

（6）可靠异步发送事务消息

例子见 PublisherInstance.java 的 `reliableAsynSendTransactionMessage()` *方法。*

```
private static void reliableAsynSendTransactionMessage(NotifyManager
notifyPublisher) {
        // 创建BytesMessage
        final BytesMessage bytesMessage = new BytesMessage();
        // 设置消息内容
        bytesMessage.setBody("TranscationTestBytesMessage".getBytes());
        // 设置消息其他信息

        bytesMessage.setTopic(TOPIC);
        bytesMessage.setMessageType(MESSAGE_TYPE);
        bytesMessage.setGroupId(GROUPID);

        bytesMessage.setStringProperty("customHeader", "customValue");

        // 发送事务消息
        SendResult result =
notifyPublisher.reliableAsynSendMessage(bytesMessage, new
SendMessageCallback() {
            public Object doInTransaction(MessageStatus status) {
                // 这个例子是一个跟业务操作相结合的例子
                // 这个地方抛出异常或者设置status.setRollbackOnly();会回滚half消息。
                // 如果这个还没有成功回滚，而应用或者NotifyServer Down掉，
                // NotifyServer会发送消息来Check这个Half的message
                // 使用和业务操作结合的方式发送消息，一定要注册一个CheckMessageListener
                return null;
            }
        });
        if (result.isSuccess()) {
            // 发送成功后处理
            System.out.println("消息发送成功");
        }
        else {
            System.out.println("消息发送失败，原因：" + result.getErrorMessage());
        }
    }
```

可靠异步发送消息详细配置请见 notify-example-hsf 例子下

com.taobao.notify.reliableAsynExample/ReliableAsynSendExample.java

2、发布各种类型的消息

所有例子见附录：Notify 客户端例子。NotifyManager 可以通过 spring 注入或直接 new 一个实例使用。该发布消息例子见 notify-example-hsf/com.taobao.notify.comexample/PublisherInstance.java。各种类型的消息都是通过上面列过的某一种方式发送出去的。

（1）发送 Bytes 类型的消息

见 PublisherInstance.java 的 sendBytesMessage()方法。

```
public static void sendBytesMessage(NotifyManager notifyManager){
    // 创建BytesMessage
        BytesMessage bytesMessage = new BytesMessage();
        // 设置消息内容
        bytesMessage.setBody("TranscationTestBytesMessage".getBytes());
        // 设置消息其他信息

        bytesMessage.setTopic(TOPIC);

        bytesMessage.setMessageType(MESSAGE_TYPE);

        bytesMessage.setStringProperty("customHeader", "customValue");

        SendResult result = notifyManager.sendMessage(bytesMessage);

        if (result.isSuccess()) {
            // 发送成功后处理
            System.out.println("消息发送成功");
        }
        else {
            System.out.println("消息发送失败，原因：" + result.getErrorMessage());
        }
    }
```

（2）发送 String 类型的消息

见 PublisherInstance.java 的 sendStringMessage()方法。

```
StringMessage stringMessage = new StringMessage();

        stringMessage.setBody("NoTranscationTestStringMessage");

        /**
         * 必填属性
         */
        stringMessage.setTopic(TOPIC);
        stringMessage.setMessageType(MESSAGE_TYPE);
        /**
         * 可选属性，使用前，请务必仔细看Notify的User guide的Message章节
         */
        stringMessage.setSendOnceMessage(true);// 默认值为false
        stringMessage.setPostTimeout(10000);// 默认值10*1000毫秒
        stringMessage.setClientPostTimeout(3000);// 默认值3*1000毫秒
        stringMessage.setTimeToLive(7 * 24 * 3600);// 默认值，为-1
        stringMessage.setDLQTime(7 * 24 * 3600);// 默认值，NotifyServer配置文件决
定

        stringMessage.setStringProperty("customHeader", "customValue"); // 该可
选属性是一个系列

        SendResult result = notifyManager.sendMessage(stringMessage);

        if (result.isSuccess()) {
            // 发送成功后处理
            System.out.println("消息发送成功");
        }
        else {
            System.out.println("消息发送失败，原因: " + result.getErrorMessage());
        }
```

（3）发送 Stream 类型消息

例子见 PublisherInstance.java 的 sendStreamMessage() 方法。发送 Stream 类型的消息可以结合上面例子以非事务或者事务的方式发送消息。

```
private static void sendStreamMessage(NotifyManager notifyPublisher) {
        StreamMessage streamMessage = new StreamMessage();
        streamMessage.writeString("NoTranscationTestStringMessage");
        streamMessage.writeInt(999);
        streamMessage.writeFloat(888.88f);
        streamMessage.writeDouble(777.777);
        streamMessage.writeLong(66666666);
        streamMessage.writeShort((short) 55);
        streamMessage.writeByte((byte) 1);
        /**
         * 必填属性
         */
        streamMessage.setTopic(TOPIC);
        streamMessage.setMessageType(MESSAGE_TYPE);
        streamMessage.setGroupId(GROUPID);
        /**
         * 可选属性，使用前，请务必仔细看Notify的User guide的Message章节
         */
        streamMessage.setSendOnceMessage(true);// 默认值为false
        streamMessage.setPostTimeout(10000);// 默认值10*1000毫秒
        streamMessage.setClientPostTimeout(3000);// 默认值3*1000毫秒
        streamMessage.setTimeToLive(7 * 24 * 3600);// 默认值，为-1
        streamMessage.setDLQTime(7 * 24 * 3600);// 默认值，NotifyServer配置文件决
定

        streamMessage.setStringProperty("customHeader", "customValue"); // 该
可选属性是一个系列

        SendResult result = notifyPublisher.sendMessage(streamMessage);

        if (result.isSuccess()) {
            // 发送成功后处理
            System.out.println("消息发送成功");
        }
        else {
            System.out.println("消息发送失败，原因： " + result.getErrorMessage());
        }
    }
```

（4）发送 Object 类型消息

例子见 PublisherInstance.java 的 sendObjectMessage() 方法。发送 Object 类型的消息可以结合上面例子以非事务或者事务的方式发送消息。

```
private static void sendObjectMessage(NotifyManager notifyPublisher) {
        ObjectMessage objectMessage = new ObjectMessage();
        //可序列化对象
        TradeRefund obj = new TradeRefund("refund-orderId001", "refund-desc");

        objectMessage.setObject(obj);

        /**
         * 必填属性
         */
        objectMessage.setTopic(TOPIC);
        objectMessage.setMessageType(MESSAGE_TYPE);
        objectMessage.setGroupId(GROUPID);

        /**
         * 可选属性，使用前，请务必仔细看Notify的User guide的Message章节
         */
        objectMessage.setSendOnceMessage(true);// 默认值为false
        objectMessage.setPostTimeout(10000);// 默认值10*1000毫秒
        objectMessage.setClientPostTimeout(3000);// 默认值3*1000毫秒
        objectMessage.setTimeToLive(7 * 24 * 3600);// 默认值，为-1
        objectMessage.setDLQTime(7 * 24 * 3600);// 默认值，NotifyServer配置文件决
定

        objectMessage.setStringProperty("customHeader", "customValue"); // 该
可选属性是一个系列

        SendResult result = notifyPublisher.sendMessage(objectMessage);

        if (result.isSuccess()) {
            // 发送成功后处理
            System.out.println("消息发送成功");
        }
        else {
            System.out.println("消息发送失败，原因： " + result.getErrorMessage());
        }
```

（5）发送 package 类型的消息

参见 *例子见 PublisherInstance.java 的* sendPackageMessage 方法。

```
/**
     * 使用package发送消息的好处是一次性可以发送多份消息，这些消息
     * 可以放在一个事务里面，要不全部成功，要不全部失败。
     * 一个package里面可以存放多个消息。 注意：这些消息的topic必须一样。
     */
    private static void sendPackageMessage(NotifyManager
notifyPublisher){

        BytesMessage msg1 = new BytesMessage();
        msg1.setBody("TranscationTestBytesMessage".getBytes());
        msg1.setTopic(TOPIC);
        msg1.setMessageType(MESSAGE_TYPE);
        msg1.setGroupId(GROUPID);

        BytesMessage msg2 = new BytesMessage();
        msg2.setBody("TranscationTestBytesMessage".getBytes());
        msg2.setTopic(TOPIC);
        msg2.setMessageType(MESSAGE_TYPE);
        msg2.setGroupId(GROUPID);

        PackagedMessage packageMsg = new PackagedMessage();
        packageMsg.addMessage(msg1);
        packageMsg.addMessage(msg2);
        notifyPublisher.sendMessage(packageMsg);
    }
```

（6）使用 Packet 序列化方式发送消息

例子见 PublisherInstance.java 的 `sendMessageUsePacketSerialization()` *方法。Packet 系列化的消息类似的可以以事务和非事务方式发送。*

Package 只包含一个消息的情况：

```
private static void sendOnePackageMessage(NotifyManager notifyPublisher){
    ExamplePacket packet = new ExamplePacket();
    packet.setValue("orderId", 123456L);
    packet.setValue("address", "创业大厦六楼");
    packet.setValue("content", "Nike 6 篮球鞋6双");
    BytesMessage msg = new BytesMessage();
    msg.setTopic(TOPIC);
    msg.setMessageType(MESSAGE_TYPE);
    msg.setGroupId(GROUPID);
    msg.setBody(packet.toByteArray());
    SendResult result = notifyPublisher.sendMessage(msg);

    if (result.isSuccess()) {
        // 发送成功后处理
        System.out.println("Package 消息发送成功!");
        System.out.println(result.getMessageId());
    }
    else {
        System.out.println("消息发送失败，原因：" + result.getErrorMessage());
    }
}
```

例子见 PublisherInstance.java 的 `sendMessageUsePacketSerializationEx()` *方法。*

使用 Packet 序列化多个消息（list）的情况：

```
private static void sendManyPackageMessage(NotifyManager notifyPublisher){
    List<ExamplePacket> packetList = new LinkedList<ExamplePacket>();
    ExamplePacket packetOne = new ExamplePacket();
    packetOne.setValue("orderId", 123456L);
    packetOne.setValue("address", "创业大厦六楼");
    packetOne.setValue("content", "Nike 6 篮球鞋6双");
    ExamplePacket packetTwo = new ExamplePacket();
    packetTwo.setValue("orderId", 1234567L);
    packetTwo.setValue("address", "创业大厦六楼");
    packetTwo.setValue("content", "Nike 7 篮球鞋7双");
    ExamplePacket packetThree = new ExamplePacket();
    packetThree.setValue("orderId", 12345678L);
    packetThree.setValue("address", "创业大厦六楼");
    packetThree.setValue("content", "Nike 8 篮球鞋8双");
    packetList.add(packetOne);
    packetList.add(packetTwo);
    packetList.add(packetThree);
    ExampleListPacket packets = new ExampleListPacket();
    packets.setValue("packetList", packetList);
    BytesMessage msg = new BytesMessage();
    msg.setTopic(TOPIC);
    msg.setMessageType(MESSAGE_TYPE);
    msg.setGroupId(GROUPID);
    msg.setBody(packets.toByteArray());
    SendResult result = notifyPublisher.sendMessage(msg);
    if (result.isSuccess()) {
            // 发送成功后处理
            System.out.println("Package 消息发送成功!");
            System.out.println(result.getMessageId());
        }
        else {
            System.out.println("消息发送失败，原因：" + result.getErrorMessage());
        }
    }
}
```

3、发布消息注意事项

（1）groupId 是不需要申请的，发送消息方的 groupId 和订阅消息方的 groupId 不要一样，否则会造成出现“没有设置 MessageListener”导致消息处理不了的情况。

（2）发布事务消息时要向 NotifyManager 注册 CheckMessageListener。

（3）通过“new” 方式获得 NotifyManager 实例发送消息时，一定要记得调用 addPublishTopic()或 setPublishTopics ()方法设置需要发布消息的 topic，否则会造成发消息失败。如果通过 spring 注入的话需要配置属性 publishTopics 的值。

（4）发布消息的 groupId 是在 NotifyManager 里面直接设置的，而订阅方的 groupId 是在构建订阅关系的时候设置的。

4、发布消息方式和消息类型的场景的选择

（1）同步发送非事务消息

使用场景：发送消息后需要同步等待消息发送结果。适合于需要知道消息发送结果才进行下一步动作的业务场景。

（2）异步发送非事务消息

使用场景：消息发送后无需等待结果返回，可进行下一步动作，最后再根据消息发送结果处理业务的场景。

（3）同步发送事务消息

使用场景：需要发送事务消息，比如消息的入库和发送需要同时成功或者同时失败。发送结果需要同步等待，结果返回后进行下一步动作。

（4）异步发送事务消息

使用场景：需要发送事务消息，比如消息的入库和发送需要同时成功或者同时失败。不需要同步等待消息发送结果的返回，消息发送后线程可以处理其他业务，最后再根据消息发送结果进行下一步动作。

（5）可靠异步发送非事务消息

使用场景：消息发送后如果客户端连接 Notify 服务器不正常，消息也不会丢掉，会暂时存在本地，当客户端连接 Notify 服务器正常后暂存本地的消息会自动发给 Notify 服务器。适用于消息需要同步发送并且高可靠的场景。

（6）可靠异步发送事务消息

使用场景：消息发送后如果客户端连接 Notify 服务器不正常，消息也不会丢掉，会暂时存在本地，当客户端连接 Notify 服务器正常后暂存本地的消息会自动发给 Notify 服务器。适用于需要发送事务消息（比如数据入库和消息发送需要在一个事物里面，同时成功或者失败），并且需要高可靠和可以异步处理的场景。

（7）发送 Bytes 类型的消息

使用场景：发送的消息内容是 Byte 数组格式的。

（8）发送 String 类型的消息

使用场景：发送的消息内容是 String 字符串格式的。

（9）发送 Stream 类型消息

使用场景：发送的消息的内容是二进制流格式。

（10）发送 Object 类型消息

使用场景：发送消息的内容是可序列化的对象，例如发布方序列化一个对象发给 Notify 服务器，订阅方反序列化后获取这个对象的内容。

（11）发送 Package 类型的消息

使用场景：多个消息包装在一个 package 里面然后发送，消息有几个或者几十个，这些消息可以放在一个事务里面，要不全部成功，要不全部失败。

二、订阅消息

所有例子见附录：Notify 客户端例子。NotifyManager 可以通过 spring 注入或直接 new 一个实例使用。该发布消息例子见 notify-example-hsf/com.taobao.notify.comexample/SubscriberInstance.java

1、订阅消息方式

（1）Direct 直接订阅

例子请见 SubscriberInstance.java 的 `subscribeByDirect()` 方法。

```
    /**
     * 直接订阅，Notify1.7.0以下版本（老版本）只有直接订阅这种
     * 方式，直接订阅订阅方式也是兼容了老的订阅方式。
     * @param subscriber
     */
    public static void subscribeByDirect(NotifyManager subscriber){
     subscriber.subscribe(Binding.direct("_NOTIFYTEST", "test-trade-refund",
"S-NT-aoqiong-test", -1, true));
    }
```

（2）Pattern 订阅

例子请见 SubscriberInstance.java 的 `subscribeByPattern()` 方法。

```
   /**
     * MessageType支持正则表达式的Pattern订阅方式，比如你想要
     * 订阅交易多有订单成功的消息只需要把messageType写成
     * *.-trade-success的方式。
     * @param subscriber
     */
    public static void subscribeByPattern(NotifyManager subscriber){
     subscriber.subscribe(Binding.pattern("TRADE", "*.-trade-success",
"S-NT-aoqiong-test", -1, true));
    }
```

（3）Header 订阅

例子请见 SubscriberInstance.java 的 `subscribeByHeader()`方法。

```
/**
     * Header是用于根据消息的属性订阅的，该订阅方式适合于订阅特定的
     * 消息，过滤掉一些不符合要求的消息。如你想订阅自己命名的
     * property.boolean == false的消息。该属性通过Message的
     * setBooleanProperty()方法设置。
     * @param subscriber
     */
    public static void subscribeByHeader(NotifyManager subscriber){
    /**
         * Header类型匹配支持的属性列表如下：<br>
         * 字段名 含义<br>
         * GMTCreate 消息的创建时间<br>
         * GMTLastDelivery 消息的上次投递时间<br>
         * timeToLive 消息的存活时间<br>
         * sendOnceMessage 消息是否是一次投递，true为是<br>
         * deliverCount 消息的投递次数<br>
         * messageType 消息的MessageType<br>
         * groupId 消息的发送分组名<br>
         * Committed 是否为half消息，true为否<br>
         * bornTIme 消息在发送端的创建时间<br>
         * dlqTIme 消息的DLQTime<br>
         * priority 消息的优先级<br>
         * property.name 用户自定义属性.名称<br>
         */
       subscriber.subscribe(Binding.header("_NOTIFYTEST",
               "sendOnceMessage == true || (priority == 1 && property.boolean
== false)", "S-NT-aoqiong-test", -1, true));
    }
```

（4）Fanout 订阅

例子请见 SubscriberInstance.java 的 `subscribByFanout()`方法。

```
    /**
     * Fanout订阅方式适合于一个topic下面所有的消息，如果你想订阅交易所有的
     * 消息就可以通过这种方式。
     * @param subscriber
     */
    public static void subscribByFanout(NotifyManager subscriber){

    subscriber.subscribe(Binding.fanout("_NOTIFYTEST", "S-NT-aoqiong-test",
-1, true));
    }
```

（5）使用 SubscriptionHelper 订阅

例子见 notify-example-hsf 包下的：
com.taobao.notify.helperexample/`HelperSubscriberInstance.java`

```
public static void helperSubscription(NotifyManager notifyManager){
        SubscriptionHelper helper = new SubscriptionHelper(notifyManager);
        //Direct直接订阅
        helper.addSubscription("_NOTIFYTEST", "test-trade-refund", true, -1);
        //Pattern正则表达式订阅
        helper.addPatternSubscription("_NOTIFYTEST","*.-trade-refund", true,
-1);
        //Header方式订阅
        helper.addHeaderSubscription("_NOTIFYTEST", "property.boolean ==
true",true,-1);
        //fanout方式订阅
        helper.addFanoutSubscription("_NOTIFYTEST", true, -1);
        try {
            helper.init();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

（6）广播方式订阅消息

广播方式订阅消息即是每台订阅机器都会收到一份一模一样的消息，非广播订阅都是同一个组下面的机器只会有一台机器收到消息。详细例子见
notify-example-hsf 包下的
com.taobao.notify.syncexample/SyncSubscriberInstance.java

```
<bean id="notifyManager" class="com.taobao.hsf.notify.client.SyncManagerBean"
    init-method="init">
    <property name="subscribeMessages">
        <map>
            <entry key="_NOTIFYTEST">
                <map>
                    <entry key="test-trade-refund">
                        <bean class="com.taobao.hsf.notify.client.SubscriptMsgDetailInfo">
                            <property name="persistence" value="true" />
                            <property name="waterMark" value="-1" />
                        </bean>
                    </entry>
                </map>
            </entry>
        </map>
    </property>
    <property name="groupId" value="S-NT-fenghan-test" />
    <property name="name" value="notifyManager" />
    <property name="description" value="notifyManager" />
    <property name="messageListener" ref="messageListener" />
</bean>

<bean id="messageListener" class="com.taobao.notify.comexample.SimpleMessageListener" />
```

注意：一般的为NotifyManagerBean，而这里为SyncManagerBean

2、打开关闭订阅

订阅关系的临时打开和关闭可以控制消息的暂停和发送，消息暂停接收时会存放在数据库里面，打开订阅时这些消息都会投递过来。代码例子见
notify-example-hsf/com.taobao.notify.comexample/SubscriberInstance.java

（1）打开订阅

详细例子见 SubscriberInstance.java 的 `openSubscription()`方法。

```
/**
     * 打开订阅，临时关闭的消息会投递过来，比如某个时间段把订阅
     * 关系暂时关闭了，过段时间再打开，在关闭订阅的这段时间的
     * 消息不会丢掉,这时候就会投递过来。
     * @param subscriber
     */
    public static void openSubscription(NotifyManager subscriber){
     subscriber.openSubscription(Binding.direct("_NOTIFYTEST",
"test-trade-refund", "S-NT-aoqiong-test", -1, true));
    }
```

（2）关闭订阅

详细例子见 SubscriberInstance.java 的 `closeSubscription()`方法。

```
/**
     * 关闭订阅，这时消息就暂时不会往客户端发送了，这些消息会暂存
     * 在数据库里面，当订阅关系打开时，这些消息就会投递过来。
     * @param subscriber
     */
    public static void closeSubscription(NotifyManager subscriber){
     subscriber.closeSubscription(Binding.direct("_NOTIFYTEST",
"test-trade-refund", "S-NT-aoqiong-test", -1, true));
    }
```

3、订阅消息注意事项

（1）订阅消息用的组名与发布消息用的组名要不同，否则订阅消息时会出现问题，导致有时候接收到消息，而有时候又接收不到消息的情况。

（2）使用“new”方式使用 NotifyManager 时注意先要订阅消息，在初始化 init()。推荐使用 spring 注入方式。

（3）订阅消息时使用的 groupId 先要到对应的 Notify console（日常，预发，线上）上面查询一下需要订阅的 topic 下面有没有其他人到使用了这个组名，如果没有使用（查询结果为空），就可以使用这个组名订阅消息，注意组名是不需要经过申请批准的。只有订阅线上交易的消息才需要申请组订阅权限。

（4）订阅消息的时候，订阅端注册的 MessageListener 收到消息后一定要判断是哪种类型的消息（byte 或者 string）再做处理，否则直接强制转换会出错。

4、使用各种订阅方式场景的选择

（1）Direct 直接订阅

使用场景：需要精确的订阅每一条消息，topic，messageType 和 groupId 需要一一对应。目前业务线使用这种订阅方式是最多的。

（2）Pattern 订阅

使用场景：messageType 支持正则表达式的订阅，比如想订阅所有交易成功的消息，messageType 可以写成“*.-trade-success”。则所有的 100-trade-success，200-trade-success 等等以“-trade-success”结尾的消息都会订阅上。

（3）Header 订阅

使用场景：支持选择性的订阅具有某种特征的消息，具体情况请看 header 方式订阅的例子。

（4）Fanout 订阅

使用场景：订阅某个 topic 下面所有的消息。

（5）使用 SubscriptionHelper 订阅

使用场景：用于 NotifyManager 初始化后订阅的场景。

（6）广播方式订阅消息

使用场景：广播方式的订阅即是每台使用同一个 groupId 的订阅方都会收到一份一模一样的消息。

三、客户端直连

所有例子见附录：Notify 客户端例子。NotifyManager 可以通过 spring 注入或直接 new 一个实例使用。该客户端直连的例子见 notify-example-hsf/com.taobao.notify.directWiredExample 下的例子。

（1）直连发布端

客户端直连的发布端例子见 notify-example-hsf/com.taobao.notify.directWiredExample/ DirectWiredTestPublisher.java。发布端直连方式与一般正常使用方式相比，需要多配置一些参数。直连发布端与直连订阅端的

```
<bean id="directWiredNotifyManager" class="com.taobao.hsf.notify.client.NotifyManagerBean" init-method="ini
    <property name="publishTopics">
        <list>
            <value>_NOTIFYTEST</value>
        </list>
    </property>
    <!-- 直连测试的连接格式 TOPIC:订阅者GroupId:订阅者IP:订阅者端口 -->
    <property name="debugRemoteSubscribers">
        <set>
            <value>_NOTIFYTEST:S-MOXING-TEST:localhost:19999</value>
        </set>
    </property>
    <!-- debug 是否为直连测试 -->
    <property name="debug" value="true"/>
    <property name="debugLocalPort" value="19998" />
    <property name="groupId" value="P-MOXING-TEST"/>
    <property name="description" value="DirectWiredNotifyManager"/>
    <property name="name" value="DirectWiredNotifyManager"/>
    <property name="checkMessageListener" ref="myCheckMessageListener"/>
</bean>
<bean id="myCheckMessageListener" class="com.taobao.notify.comexample.SimpleCheckMessage"/>
```

需要配置直连测试连接信息，topic，groupId以及ip等等。

设置NotifyManager为debug模式

端口

说明：直连的客户端发布端一定要设置对 debugRemoteSubscribers 的参数，直连方式只允许一个发布端对应一个订阅端，并且只有一个 topic。debugRemoteSubscribers 的参数方式为：TOPIC:订阅者 GroupId:订阅者 IP:订阅者端口。如：_NOTIFYTEST:S-MOXING-TEST:localhost:19999

（2）直连订阅端

客户端直连的订阅端例子见 notify-example-hsf/com.taobao.notify.directWiredExample/SimpleWiredTestSuscriber.java。订阅端直连方式与一般正常使用方式相比，在发布端配置好后，只需要设置 debug 模式和端口即可。

```
<beans default-autowire="byName">
    <bean id="notifyManager" class="com.taobao.hsf.notify.client.NotifyManagerBean" init-method="init" >

        <property name="debug" value="true"/>
        <property name="debugLocalPort" value="19999" />
        <property name="groupId" value="S-MOXING-TEST"/>
        <property name="description" value="SubscriberNotifyManager"/>
        <property name="name" value="SubscriberNotifyManager"/>
        <property name="messageListener" ref="simpleMessageListener"/>
    </bean>

    <bean id="simpleMessageListener" class="com.taobao.notify.comexample.SimpleMessageListener"/>
</beans>
```

说明：各种订阅方式（direct，pattern，header，fanout）请看 notify-example-hsf/com.taobao.notify.directWiredExample/DirectWiredTestSubscriber.java

四、消息参数详细说明

客户端能发送的基本消息类型有 BytesMessage，StringMessage，StreamMessage 和 ObjectMessage。而 Packet 类型的消息是 BytesMessage 的一种形式。

1、messageId：消息唯一识别号，客户端在消息发送前自动生成，对于 Notify 来说不同的 messageId 意味着不同的消息。

2、committed：判断消息是否为 half 消息。false 为 half 消息；true 为 commit 消息。

3、groupId:发送消息方的 groupId。

4、topic：消息主题。消息主题标识着某一类业务服务，这一类业务下面某个具体的业务由 messageType 表示。

5、messageType：某个具体的消息类型，对应某个具体的业务服务类型。

6、gmtCreate：消息的创建时间，消息在到达 Notify 服务器时被设置。

7、gmtLastDelivery：消息上一次投递完后的时间，每轮投递完都会更新这个值。

8、deliveryCount：投递次数，表示到目前为止该消息被重投了多少次。

9、timeToLive：消息存活时间，表示消息一直失败后保存在 Notify 服务器上的时间，目前保存时间上限是 7 天。

10、dlqTime：消息进入 dlq 的时间，与 timeToLive 对应，dlqTime 结束后消息进入 dlq。注意：如果已经设置了 timeToLive，则设置的 dlqTime 无效。

11、publisherHostName：发布消息方的 hostName，无需客户端设置，自动生成。

12、bornTime：消息在发送端的生成的时间，自动生成。

13、postTimeOut：消息投递超时时间，为消息到达 Notify 服务器上后，Notify 服务器向客户端投递时的超时时间，目前上限为 30s。

14、clientPostTimeOut：客户端向 Notify 服务器发送消息时的超时时间。默认 3s。

15、priority：消息优先级，1 至 10，默认为 5。值越大，优先级越高。

16、sendOnceMessage：标识消息是否只投递一次，如果为 true，Notify 服务器向订阅者投递消息时不管成功还是失败，只投递一次，不会重投。

17、userDefinedProperties：用户可以自定义的一些属性。比如setBooleanProperty()，setStringProperty()等等。客户端通过NotifyManager可设置。

五、NotifyManager常用方法说明

（1）addPublishTopic(topic)
说明：发布一个topic，该方法用于发布消息时验证Notify是否提供此topic服务，发布消息时为必填属性。

（2）asynSendMessage(Message, AsynSendResultListener)
说明：异步方式发送一个非事务消息，须实现AsynSendResultListener接口回调。

（3）asynSendMessage(Message, SendMessageCallback ,AsynSendResultListener)
说明：异步方式发送一个非事务消息，需要实现事务回调接口SendMessageCallback和异步发送回调监听器AsynSendResultListener。

（4）closeSubscription(Binding)
说明：临时关闭订阅关系，客户端暂时不会收到消息。关闭操作是幂等的。

（5）openSubscription(Binding)
说明：打开临时关闭的订阅关系，客户端重新接收被关闭掉的消息，之前关闭的消息都会收到，不会丢掉，打开操作是幂等的。

（6）reliableAsynSendMessage（Message）
说明:可靠方式发送非事务消息,如果此时客户端连接Notify服务器失败也不会丢掉消息，消息会暂时存在本地，当客户端重新连接好服务器后，暂存本地的消息会自动投给 Notify服务器。

（7）reliableAsynSendMessage(Message, SendMessageCallback)
说明：可靠方式发送事务消息。

（8）resetPublishTopics(Collection<topics>);
说明：重新设置需要发布的topic。

（9）sendMessage(Message)
说明：发送一条消息。

（10）sendMessage(Message, SendMessageCallback)
说明：发送一条事务消息。

（11）setBindingList(List<Binding>)
说明：一次性订阅多个消息

（12）setCheckMessageListener(CheckMessageListener)
说明：发布事务消息时必须要注册这个 CheckMessageListener，当 half 消息发送到服务器之后服务器会发送 check 消息给客户端检查这个消息状态。

（13）setCheckmessageTPCorePoolSize(int)
说明：设置消息核心线程池大小，默认为 10。

（14）setCheckmessageTPMaxPoolSize(int)
说明：设置消息线程池的最大值，所有订阅消息共用这个线程池。默认为 20。

（15）setCheckMessageTPMaxQueueSize(int)
说明：设置消息线程池队列大小，默认 10000。

（16）subscriberAfterInited(List<Binding>);
说明：在 NotifyManager.init()之后订阅一组消息。

（17）subscribeMessages（Map<String, Map<String,SubscriptMsgDetailInfo>>);
说明：以 Map<topic,Map<MessageType, SubscriptMsgDetailInfo>>方式订阅一组消息.

（18）subscribe(Binding binding);
说明：订阅一个消息，这种订阅方式在 Notify1.7.0 以后才能用。Binding 有四种方式，Direct,Header,Pattern 和 Fanout。

（19）subscribe(List<Binding>);
说明：订阅一组消息

（20）subscribeMessage(topic,messageType, persistSub, waterMark);
说明：直接根据topic，messageType等参数订阅消息，不需要构建Binding或Message。

六、术语表

（1）发布消息

发布消息是指Notify客户端把业务数据包装成Notify自身定义的消息后向Notify服务器发送消息的过程。

（2）投递消息

投递消息是指 Notify 服务器将消息投递给订阅这些消息的客户端的过程。

（3）订阅消息

订阅消息是指 Notify 客户端向 Notify 服务器请求接收消息的过程

（4）同步发送

同步发送是指 Notify 客户端向 Notify 发送消息过程中需要等待直到发送结果返回，才能进行下一步动作的过程。

（5）异步发送

异步发送是指Notify客户端向Notify服务器发送消息过程中不需要同步等待发送结果的返回，线程可以继续进行下一步动作的过程。

（6）可靠发送

可靠发送是指Notify客户端向Notify服务器发送消息过程中即使客户端连接Notify服务器有异常，消息也能正常发送，不会丢掉的过程。

（7）客户端直连

客户端直连是指发布消息的客户端和订阅消息的客户端直接通信，不需要走 Notify 服务器的这个过程。

（8）广播方式订阅

广播方式订阅是指当客户端向 Notify 服务器请求接收消息时，Notify 服务器向将这条消息向这个组（groupId）下的所有连接的客户端广播，所有的这个组下的客户端都会接收到这条消息的过程。

（9）订阅关系

订阅关系是指某个客户端的订阅组名（groupId）所订阅的一系列的 topic 和这些 topic 下面对应的 messageType 的映射，这种映射关系会持久保存。

当客户端连接 Notify 订阅消息时，首先这个组下面的订阅映射会持久保存并且这些订阅关系映射会更新到 Notify 服务器；其次客户端的元信息如 groupId 和连接信息等会缓存在 Notify 服务器，因此服务端有一个订阅方组名（groupId）到客户端连接地址（connection）的映射的缓存；最后当消息投递时，Notify 服务器根据消息的 topic 和 messageType 从订阅关系映射中检索出所有的对应的 groupId，并根据这些 groupId 获取订阅方的连接将消息投递出去。

订阅关系参数说明：

Topic：订阅消息的 topic

MessageType：消息类型，为 topic 下面的某一种消息类型。

groupId：订阅方的 groupId，是订阅方的唯一识别 Id，Notify 只根据 groupId 识别订阅者，而不是 ip 地址。

Persistent：true 或者 false，代表订阅消息是否需要持久可靠化。如果为 true，则订阅者下线后消息不丢失；如果为 false，则订阅者不在线时消息丢失。

附录：

Notify 网站：所有例子与资源可以在：http://arch.taobao.ali.com/notify/index.html 找到。

Notify 申请：http://arch.taobao.ali.com/notify/UserGuide/ApplyForm.html

Notify FAQ：http://arch.taobao.ali.com/notify/UserGuide/FAQ.html

Notify 客户端例子：http://arch.taobao.ali.com/notify/QuickStart.html

Notify maven 依赖：http://arch.taobao.ali.com/notify/QuickStart.html

Notify Console 地址：

日常：http://nconsole.taobao.net
预发：http://ops.jm.taobao.org:9999/notify-console-pre/index.jsp
线上：http://ops.jm.taobao.org:9999/notify-console/index.jsp