分布式消息中间件之Activemq

技术部. 甘建新.2017.08.02

1. JMS
2. Activemq
3. Pfs消息处理机制

Java Message Service（JMS）

Java消息服务（JMS）API是一种消息传递标准，允许基于Java平台企业版（Java EE）的应用程序组件创建，发送，接收和读取消息。它实现了松散耦合，可靠和异步的分布式通信。

实现高性能，高可用，可伸缩和最终一致性架构。

JMS基本构件

- 连接工厂
- 连接
- 会话
- 目的地
- 消息生产者
- 消息消费者
- 消息

- 连接工厂

连接工厂是客户用来创建连接的对象，例如ActiveMQ 提供的ActiveMQConnection Factory。

- 连接

JMS Connection 封装了客户与JMS 提供者之间的一个虚拟的连接。

- 会话

JMS Session 是生产和消费消息的一个单线程上下文。会话用于创建消息生产者（producer）、消息消费者（consumer）和消息 （message）等。会话提供了一个事务性的上下文，在这个上下文中，一组发送和接收被组合到了一个原子操作中。

- 目的地

- 目的地是客户用来指定它生产的消息的目标和它消费的消息的来源的对象。

JMS1.0.2 规范中定义了两种消息传递域：点对点 （PTP）消息传递域和发布/订阅消息传递域。

队列（Queue）和主题（Topic）是JMS支持的两种消息传递模型：

类型	Topic	Queue
概要	Publish Subscribe messaging 发布订阅消息	Point-to-Point 点对点
有无状态	topic数据默认不落地，是无状态的。	Queue数据默认会在mq服务器上以文件形式保存，比如Active MQ一般保存在$AMQ_HOME\data\kr-store\data下面。也可以配置成DB存储。
完整性保障	并不保证publisher发布的每条数据，Subscriber都能接受到。	Queue保证每条数据都能被receiver接收。
消息是否会丢失	一般来说publisher发布消息到某一个topic时，只有正在监听该topic地址的sub能够接收到消息；如果没有sub在监听，该topic就丢失了。**（除持久订阅）**	Sender发送消息到目标Queue，receiver可以异步接收这个Queue上的消息。Queue上的消息如果暂时没有receiver来取，也不会丢失。
消息发布接收策略	一对多的消息发布接收策略，监听同一个topic地址的多个sub都能收到publisher发送的消息。Sub接收完通知mq服务器	一对一的消息发布接收策略，一个sender发送的消息，只能有一个receiver接收。receiver接收完后，通知mq服务器已接收，mq服务器对queue里的消息采取删除或其他操作。

➢ 消息生产者

消息生产者是由会话创建的一个对象，用于把消息发送到一个目的地。

➢ 消息消费者

消息消费者是由会话创建的一个对象，它用于接收发送到目的地的消息。消息的消费可以采用以下两种方法之一：

同步消费。通过调用 消费者的receive 方法从目的地中显式提取消息。receive 方法可以一直阻塞到消息到达。

异步消费。客户可以为消费者注册一个消息监听器，以定义在消息到达时所采取的动作。

➢ 消息

JMS 消息由以下三部分组成：

消息头。每个消息头字段都有相应的getter 和setter 方法。

消息属性。如果需要除消息头字段以外的值，那么可以使用消息属性。

消息体。JMS 定义的消息类型有TextMessage、MapMessage、BytesMessage、StreamMessage 和 ObjectMessage。

JMS可靠性机制

- 确认
- 持久性
- 优先级
- 消息过期
- 临时目的地
- 持久订阅
- 本地事务

➢ 确认

JMS 消息只有在被确认之后，才认为已经被成功地消费了。消息的成功消费通常包含三个阶段：客户接收消息、客户处理消息和消息被确认。在事务性会话中，当一个事务被提交的时候，确认自动发生。在非事务性会话中，消息何时被确认取决于创建会话时的应答模式（acknowledgement mode）。

该参数有以下三个可选值：

Session.AUTO_ACKNOWLEDGE。当客户成功的从receive 方法返回的时候，或者从MessageListener.onMessage 方法成功返回的时候，会话自动确认客户收到的消息。

Session.CLIENT_ACKNOWLEDGE。客户通过消息的acknowledge 方法确认消息。需要注意的是，在这种模式中，确认是在会话层上进行：确认一个被消费的消息将自动确认所有已被会话消费的消息。例如，如果一个消息消费者消费了10 个消息，然后确认第5 个消息，那么所有10 个消息都被确认。

Session.DUPS_ACKNOWLEDGE。该选择只是会话迟钝的确认消息的提交。如果JMS provider 失败，那么可能会导致一些重复的消息。如果是重复的消息，那么JMS provider 必须把消息头的JMSRedelivered 字段设置为true。

➢ 持久性

JMS 支持以下两种消息提交模式：

PERSISTENT。指示JMS provider 持久保存消息，以保证消息不会因为JMS provider 的失败而丢失。

NON_PERSISTENT。不要求JMS provider 持久保存消息。

➢ 优先级

可以使用消息优先级来指示JMS provider 首先提交紧急的消息。优先级分10个级别，从0（最低）到9（最高）。如果不指定优先级，默认级别是4。需要注意的是，JMS provider 并不一定保证按照优先级的顺序提交消息。

➢ 消息过期

可以设置消息在一定时间后过期，默认是永不过期。

➢ 临时目的地

可以通过会话上的createTemporaryQueue 方法和createTemporaryTopic 方法来创建临时目的地。它们的存在时间只限于创建它们的连接所保持的时间。只有创建该临时目的地的连接上的消息消费者才能够从临时目的地中提取消息。

➢ 持久订阅

首先消息生产者必须使用PERSISTENT 提交消息。客户可以通过会话上的createDurableSubscriber 方法来创建一个持久订阅，该方法的第一个参数必须是一个topic。第二个参数是订阅的名称。

JMS provider 会存储发布到持久订阅对应的topic 上的消息。如果最初创建持久订阅的客户或者任何其它客户使用相同的连接工厂和连接的客户ID、相同的主题和相同的订阅名再次调用会话上的createDurableSubscriber 方法，那么该持久订阅就会被激活。JMS provider 会向客户发送客户处于非激活状态时所发布的消息。

持久订阅在某个时刻只能有一个激活的订阅者。持久订阅在创建之后会一直保留，直到应用程序调用会话上的unsubscribe 方法。

➢ 本地事务

在一个JMS 客户端，可以使用本地事务来组合消息的发送和接收。JMS Session接口提供了commit 和rollback 方法。事务提交意味着生产的所有消息被发送，消费的所有消息被确认；事务回滚意味着生产的所有消息被销毁，消费的所有消息被恢复并重新提交，除非它们已经过期。

事务性的会话总是牵涉到事务处理中，commit 或rollback 方法一旦被调用，一个事务就结束了，而另一个事务被开始。关闭事务性会话将回滚其中的事务。需要注意的是，如果使用请求/回复机制，即发送一个消息，同时希望在同一个事务中等待接收该消息的回复，那么程序将被挂起，因为知道事务提交，发送操作才会真正执行。

需要注意的还有一个，消息的生产和消费不能包含在同一个事务中。

Activemq

Apache ActiveMQ™是最流行和最强大的开源消息传递和集成模式服务器。

支持许多跨语言客户端和协议，具有易于使用的企业集成模式和许多高级功能，同时完全支持JMS 1.1和J2EE 1.4。

特征

- 支持Java，C，C ++，C＃，Ruby，Perl，Python，PHP中的各种跨语言客户端和协议；
- OpenWire for Java，C，C ++，C＃中的高性能客户端；

  Stomp支持，以便C，Ruby，Perl，Python，PHP，ActionScript / Flash，Smalltalk中的客户端可以轻松地与ActiveMQ以及任何其他流行的Message Broker；

  AMQP v1.0支持；

  MQTT v3.1支持允许在IoT环境中进行连接。
- 完全支持JMS客户端和Message Broker中的企业集成模式；
- 支持许多高级功能，如消息组，虚拟目的地，通配符和复合目的地；
- 完全支持JMS 1.1和J2EE 1.4，支持瞬态，持久，事务和XA消息传递；
- Spring支持，使ActiveMQ可以轻松嵌入到Spring应用程序中，并使用Spring的XML配置机制进行配置；
- 受欢迎的J2EE服务器（如TomEE，Geronimo，JBoss，GlassFish和WebLogic）进行了测试；

  包括用于入站和出站邮件的JCA 1.5资源适配器，以便ActiveMQ在任何符合J2EE 1.4标准的服务器中自动部署；
- 支持可插拔传输协议，如VM，TCP，SSL，NIO，UDP，多播，JGroups和JXTA传输；
- 支持使用JDBC和高性能日志的非常快速的持久性；
- 专为高性能集群，客户端服务器，对等通信而设计；
- REST API提供技术不可知和语言中性的基于Web的API到消息传递；
- Ajax支持使用纯DHTML的Web浏览器进行Web流媒体支持，允许Web浏览器成为消息传递结构的一部分；
- CXF和Axis支持，使ActiveMQ可以轻松地放入这些Web服务堆栈中以提供可靠的消息传递；
- 可用作内存JMS提供程序，是单元测试JMS的理想选择；

硬件：
约60 MB的可用磁盘空间为ActiveMQ 5.x二进制分配。（您需要额外的磁盘空间来存储永久消息到磁盘）
约300 MB的可用磁盘空间为ActiveMQ 5.x源或开发人员的发行版。

操作系统：
Windows：Windows XP SP2，Windows 2000，Windows Vista，Windows 7。
Unix：Ubuntu Linux，Powerdog Linux，MacOS，AIX，HP-UX，Solaris或任何支持Java的Unix平台。

环境：
Java运行时环境（JRE）JRE 1.7（版本<= 5.10.0为1.6）？（如果您计划重新编译源代码，则需要JDK）
必须将JAVA_HOME环境变量设置为安装JRE的目录？Maven 3.0.0 build system?将要使用的JAR必须添加到类路径中。

启动

Windows系统:

```
cd [activemq_install_dir]
bin\activemq start
```

Linux系统:

```
cd [activemq_install_dir]/bin
./activemq console
```

作为守护进程启动：

```
cd [activemq_install_dir]/bin
./activemq start
```

停止

```
cd [activemq_install_dir]/bin
./activemq stop
```

管理平台

URL: http://127.0.0.1:8161/admin/

Login: admin

Passwort: admin

日志输出

[activemq_install_dir]/data/activemq.log

监听端口

Windows:

netstat -an|find "61616"

Linux:

netstat -nl|grep 61616

传输协议

- AUTO Transport

从5.13.0开始，ActiveMQ支持通过TCP，SSL，NIO和NIO SSL进行自动线路协议检测。 支持OpenWire，STOMP，AMQP和MQTT。

- VM Transport
- AMQP Transport
- MQTT Transport
- TCP Transport
- NIO Transport
- SSL Transport
- NIO SSL Transport
- Peer Transport
- UDP Transport
- Multicast Transport
- HTTP and HTTPS Transport
- WebSockets Transport
- Stomp Transport

持久化

- Kaha Peristence

是专门针对消息持久性编写的存储解决方案，是ActiveMQ项目的一部分。 它被调整为为典型的消息使用模式提供最佳性能，其中涉及写入/读取和丢弃持续非常快速的消息。

```
<persistenceAdapter>
        <kahaDB directory="${activemq.data}/kahadb"/>
</persistenceAdapter>
```

- JDBC Persistence

目前支持的数据库有Apache Derby, Axion, DB2, HSQL, Informix, MaxDB,MySQL, Oracle, Postgresql, SQLServer, Sybase。

集群

- Queue consumer clusters
- Broker clusters
- Discovery of brokers
- Networks of brokers
- Master Slave
- Replicated Message Stores

特性

- **Exclusive Consumer**

ActiveMQ 从4.x 版本起开始支持Exclusive Consumer （或者说ExclusiveQueues）。Broker会从多个consumers 中挑选一个consumer 来处理queue 中所有的消息，从而保证了消息的有序处理。如果这个consumer 失效，那么broker会自动切换到其它的consumer。

- **Message Groups**

。Message Groups 特性保证所有具有相同JMSXGroupID 的消息会被分发到相同的consumer（只要这个consumer 保持active）。另外一方面，Message Groups 特性也是一种负载均衡的机制。

在一个消息被分发到consumer 之前，broker 首先检查消息JMSXGroupID 属性。
如果存在，那么broker 会检查是否有某个consumer 拥有这个message group。
如果没有，那么broker 会选择一个consumer，并将它关联到这个message group。
此后，这个consumer 会接收这个message group 的所有消息，直到：

Consumer 被关闭。

Message group 被关闭。通过发送一个消息，并设置这个消息的JMSXGroupSeq 为0。

特性

- Pending Message Limit Strategy

ActiveMQ 通过prefetch 机制来提高性能，这意味这客户端的内存里可能会缓存一定数量的消息。缓存消息的数量由prefetch limit 来控制。当某个consumer 的prefetch buffer 已经达到上限，那么broker 不会再向consumer 分发消息，直到consumer 向broker 发送消息的确认。可以通过在 ActiveMQConnectionFactory 或者ActiveMQConnection 上设 ActiveMQPrefetchPolicy 对象来配置prefetch policy。也可以通过connection options 或者destination options 来配置。

例如：

```
tcp://localhost:61616?jms.prefetchPolicy.all=50
tcp://localhost:61616?jms.prefetchPolicy.queuePrefetch=1
queue = new ActiveMQQueue("TEST.QUEUE?consumer.prefetchSize=10");
```

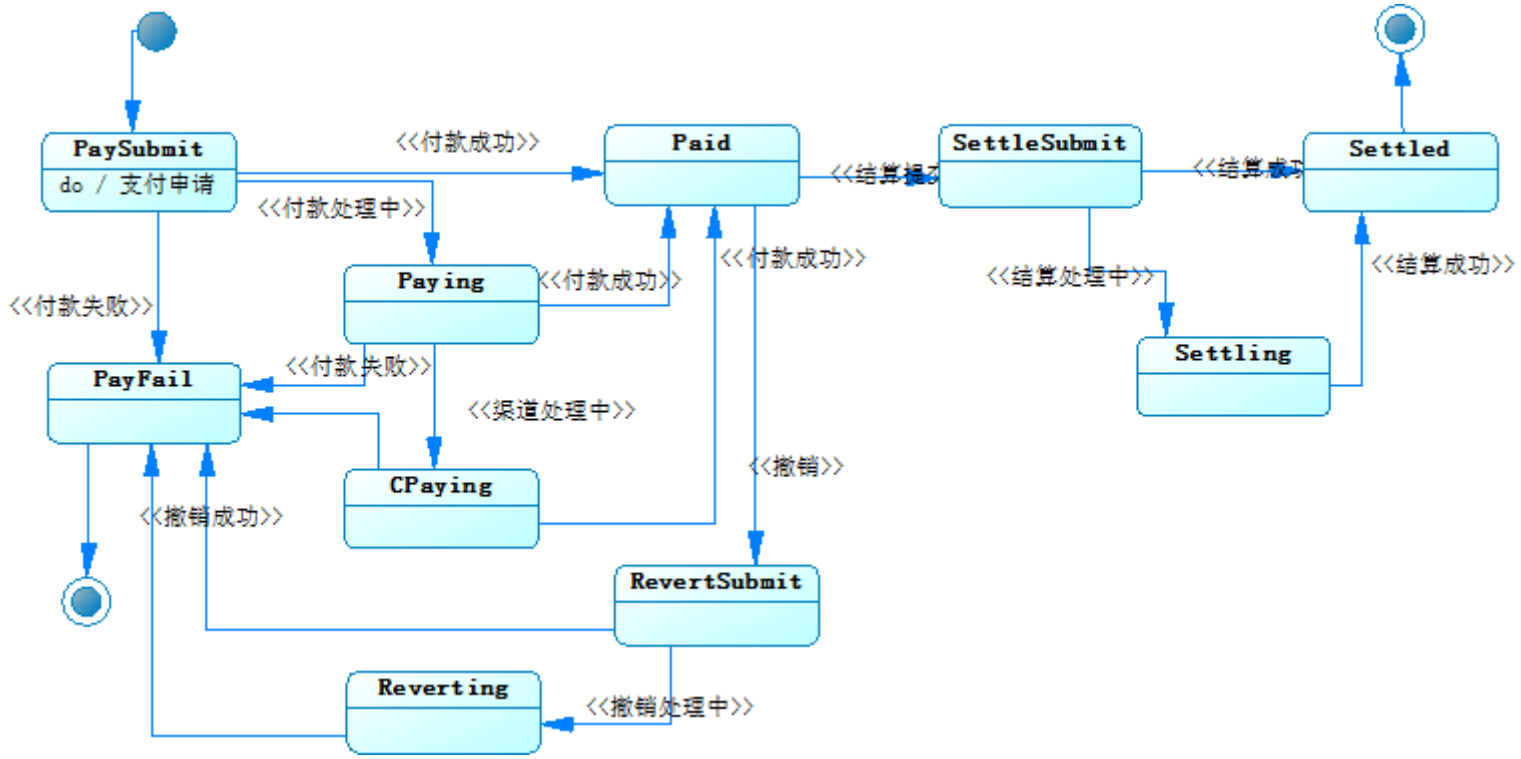
- **Async Sends**

ActiveMQ 缺省使用异步传输方式。但是按照JMS 规范，当在事务外发送持久化消息的时候，ActiveMQ 会强制使用同步发送方式。

其他特性请参阅官网。

Pfs消息处理机制：入款

在支付流程中会有很多状态的流转，在状态流转过程中，需要对外发送消息，以下是入款的状态流转图

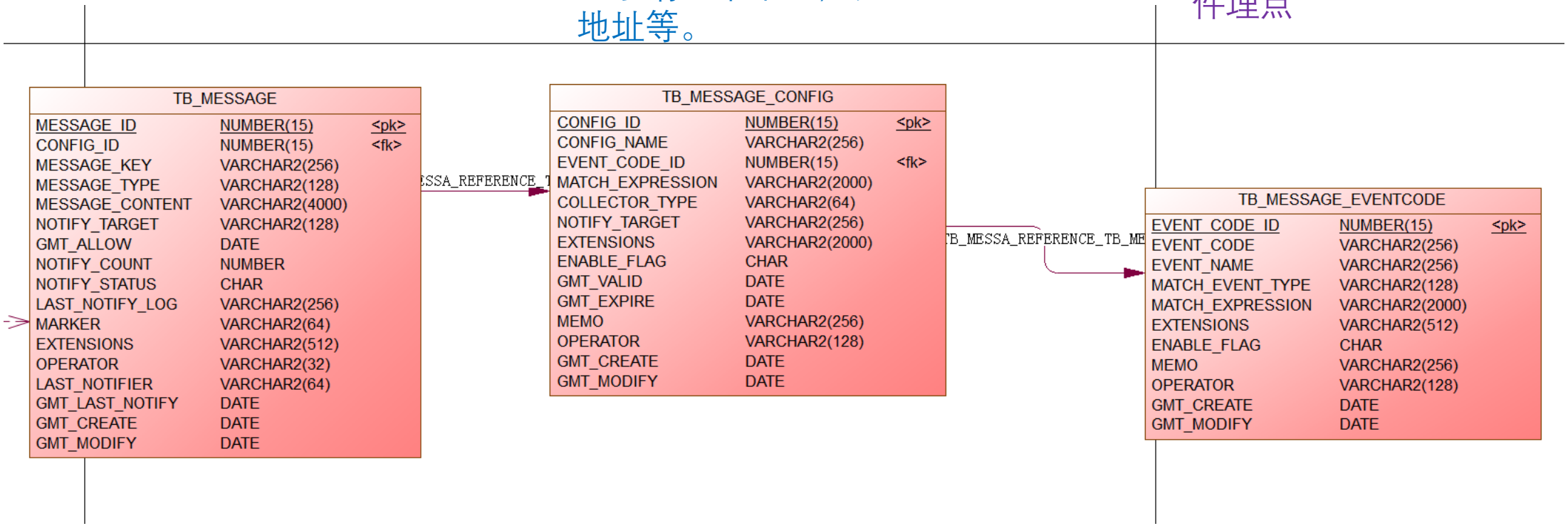


Pfs消息处理机制：表结构

消息表：存放发送的消息

消息配置表：多项配置对应一个事件。该表配置了消息采集器，发送地址等。

消息事件表：在状态流转时的事件埋点



Pfs消息处理机制：发送消息

在状态流转时，会判断流程状态是否调用通知事件

```
if ((result.getProcessFlag() == StateProcessFlag.ASYNC)
    || (result.getProcessFlag() == StateProcessFlag.CHANGED_STOP)
    || (result.getProcessFlag() == StateProcessFlag.SYNC)) {
    notifyManager.notifyEvent(new BizOrderEvent(context, result));
    for (PaymentOrderV2 paymentOrder : context.getBizPaymentOrder().getPaymentOrderList()) {
        notifyManager.notifyEvent(new PayOrderEvent(context, paymentOrder));
    }
}
```

payment.tb_message_eventcode
BizType 对应业务支付订单
PayOrder 对应支付指令订单
从设计上，一笔业务支付订单对应多笔支付指令订单；

Match_event_type 对应的事件类型
Match_expression velocity表达式

payment.tb_message_config
event_code_id 对应message_eventcode中的id
match_expression velocity表达式,判断是否为[MATCH]
collect_type 内容采集器类型，对应具体代码的内容采集器
notify_target 通知地址

消息去重

messageKey: ContentCollector. collectMessageKey

Payment.tb_message_config的colltor_type内容对应到具体的内容采集器

如果在tb_message中存在则剔除，不通知；

```
@Override
public String collectMessageKey(NotifyEvent event, MessageConfig config) {
    return String.format("%s_%d", ((BizOrderEvent) event).getBizPaymentOrder()
        .getBizPaymentSeqNo(), config.getConfigId());
}
```

消息构建

- 根据message_config配置的内容采集器构建消息内容，默认从数据库取，如果没有，根据内容采集器collectNotifyTarget获取。
- 通知地址Notify_target
- 消息类型：INIT
- 消息校验

设置校验原文和校验码
Hash类型：从数据库字段extentions中获取，默认ParamKeyEnum.
HASH_TYPE　MD5
hash消息
盐值：ParamKeyEnum. HASH_SALT

保存消息tb_message

保存通知，如果保存失败，剔除出通知列表

异步线程池发送消息

重试： 默认衰减策略

重试策略： RETRY_STRATEGY；
重试策略：衰减策略（默认）、定时策略；

- 参数

最大重试次数：RETRY_MAX_TIMES；默认99
通知重试最大时长：RETRY_MAX_DURATION；72小时

- 定时重试：

重试间隔：RETRY_TIMER_DURATION；默认60秒

- 衰减重试：

时间间隔：RETRY_ATTEN_INTERVAL；默认10S
衰减因子：RETRY_ATTEN_FACTOR；默认2；
示例：20s->40s->80s->…

消息：通知结果

对通知结果进行处理：

```
public void onNotifyResult(NotifyResult result, String subLogPrefix) {
    Message message = result.getNotifyMessage();
    setMessageResult(message, result);
    switch (result.getNotifyStatus()) {
        case SUCCESS:
            if (logger.isInfoEnabled()) {
                logger.info("{}，通知成功更新", subLogPrefix);
            }
            messageManager.updateNotify(message);
            break;
        case FAIL:
        case ERROR:
            retryManager.runRetryStrategy(message, subLogPrefix);
            if (logger.isInfoEnabled()) {
                logger.info("{}，通知失败更新", subLogPrefix);
            }
            messageManager.updateNotify(message);
            break;
        default:
            throw new RuntimeException(subLogPrefix + "，通知结果状态未知，" + message);
    }
}
```

消息补发机制：

定时任务：三分钟
范围：初始、通知失败、通知异常
批次：每次10条

消息：刷新缓存配置

本地缓存

超时时间600秒

```
LocalUCache<List<EventCode>> listCache = new LocalUCache<List<EventCode>>();

@Override
public List<EventCode> getEnabled() {
    String key = CACHE_NAMESPACE_PREFIX + "eventcode.enabled";
    List<EventCode> ret = listCache.get(key);
    if (ret == null) {
        ret = eventCodeManager.getEnabled();
        listCache.add(key, ret, CACHE_EXPIRE_SECONDS);
    }
    return ret;
}

@Override
public void invalidateAll() {
    listCache.flush();
}
```

参考资料

http://activemq.apache.org

http://192.168.180.67/svn/src/basis/pfs

Thank you !

海尔金控
Haier Financial Holdings