hmsg: 分布式消息系统，由Java编写，使用netty处理底层网络通信，使用protobuf实现自定义协议和消息序列化，地址：<https://github.com/HitTheFloor2/hmsg>

1. 概述

hmsg构建了一个多节点集群，集群中的每一个节点独立，节点之间对等。集群中的通信基于socket，如不同物理主机之间通过运行hmsg进程实现通信、同一物理主机运行不同的hmsg进程实现通信等。

hmsg使用简单，通过集成hmsg.jar依赖，可以将其作为第三方通信工具来使用。在开发分布式系统中有着一定的便利。

1. 系统架构
2. 网络连接模型（大圈代表节点的server，小圈代表节点的client）

被连接

S

S

主动连接

S

S

在实现对等节点的过程中，hmsg将每一个节点划分出两个功能模块：Server和Client。类似于传统的C/S模型，Server作为一个监听的socket地址（如192.168.1.1:8080），可以由其他hmsg节点的Client端找到并连接。Client端负责主动连接其他hmsg节点的Server端，并且管理、监控整个节点的网络感知状况。

举例说明：先运行A节点，A节点未发现其他的hmsg节点，则A节点的Client暂时不记录；之后运行B节点（并且通过配置等方法使其知道A节点地址），B节点的Client端建立和A节点的连接，并发送消息；A节点接到B节点的消息，在A节点自己的Client端添加和B节点的连接。

在这个模型中，一对节点如果能实现全双工通信，那么需要建立两条网络连接。如上述例子中，Server端不负责记录连接到自己的连接，同时也无法发送消息，只能监听。Client端负责记录相连节点和发送消息。值得注意的是，每一条连接是记录两端的地址的，当Client端连接其他hmsg节点的Server端时，会自动选择一个空闲的地址作为新建连接的起始端地址。

1. 网络集群模型

在hmsg节点组成的集群中，每一个节点都是对等的，他们使用相同的hmsg代码进行节点的初始化。随着网络的运行，每一个节点在网络中的角色或者所知的网络信息可能不同。

当hmsg节点初始化时，需要有外部的操作使其感知到其他hmsg节点或集群的存在。初始化的hmsg节点需要被配置一些“入口”节点的地址（这些“入口”节点至少有一个需要可用），通过和“入口”节点交换网络信息，初始化的节点可以快速的得知整个网络的情况。

在hmsg节点运行时，通过守护线程定时执行任务：如广播询问网络信息，连接保活等，更新自身对集群的感知。

1. 消息收发模型

在hmsg节点的设计中，尽量在保持并发性能和执行速度上，加入一定的可拓展性。MessageManager是hmsg中处理消息收发的模块，使用Client端提供的消息发送功能和Server提供的消息接收功能，实现了如下接口：

**public interface IMessageManager {**

**void sendMsg(Object msg,String receiver\_address);**

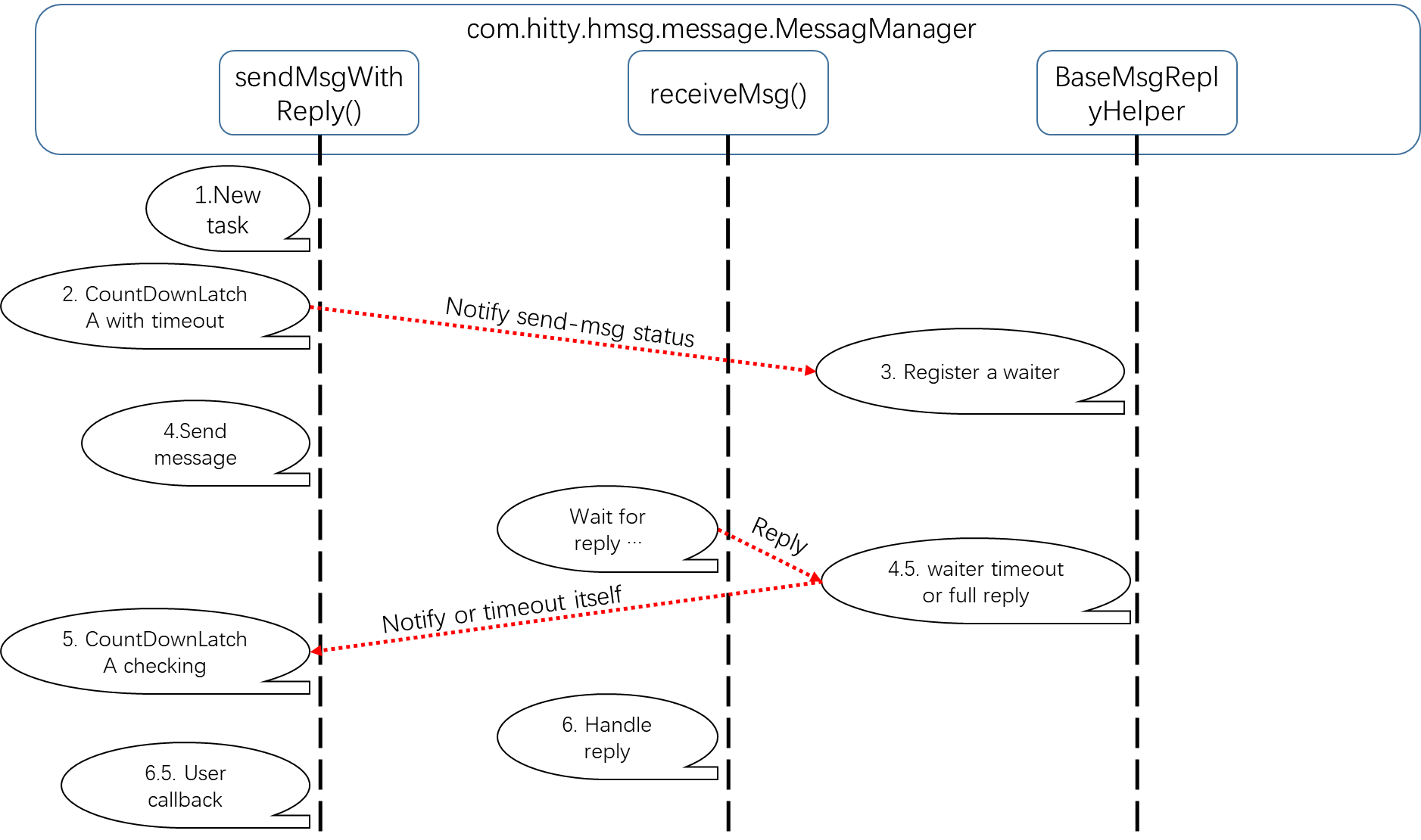
**void sendMsgWithReply(Object msg, String receiver\_address, int timeout, Object baseMsgCallBack);**

**void receiveMsg(Object msg);**

**}**

其中sendMsgWithReply是一个重要的规范，第一个参数是发送的消息实体，第二个是发送目标节点的地址，第三个是超时时间（毫秒），第四个是自定义操作。在hmsg提供的默认MessageManager中，sendMsgWithReply的实现如下：

|  |  |  |
| --- | --- | --- |
| 步骤 | 操作 | 具体描述 |
| 0 | 在MessageManager的send任务线程池中注册一个新的send任务 | 使得sendMsgWithReply成为一个异步任务，不会在执行中被阻塞。因为sendMsgWithReply需要等待其他的消息进行回复，然而回复所需的时间是不可控的。 |
| 1 | 注册超时管理器A | 超时管理器A负责sendMsgWithReply任务的超时。当A发生超时或者被解除时，停止等待。 |
| 2 | 注册该消息的等待器 | 等待器集合被MsgReplyHelper模块管理，当Server端收到任何回复消息时，MsgReplyHelper模块都会检查该消息是否为任何等待器的回复 |
| 3 | 发送消息 | 发送消息也是异步的，不阻塞 |
| 4 | 开启超时管理器A的等待 | 防止sendMsgWithReply操作超时 |
| … | 等待回复 | 在这一段时间中，等待器开启一个定时线程，其超时时间为发送消息内定义的超时时间（比如发送一次投票消息，在6秒内有效，6秒后收到的回复作废），将收到的回复消息存储至等待器中。超时后，解除步骤1中的超时管理器A |
| 5 | 检查超时管理器A | 等待器里面存储了收到的回复消息 |
| 6 | 执行baseMsgCallback | 执行自定义逻辑 |



1. 消息自定义与序列化

由于在消息的传递中，有许多不可控情况发生（如网络异常、消息过大），这里hmsg通过protobuf进行了自定义的消息体实现。具体内容可以在/protobuf/BaseMsg.proto 中找到。

|  |  |
| --- | --- |
| 字段定义 | 内容描述 |
| int32 msg\_type = 1; | 消息类型 |
| int64 msg\_sequence\_number = 2; | 消息序列号，每一个hmsg节点都会按照顺序记录自己发出的消息的序列号，当前为自增。 |
| int64 msg\_uuid = 3; | 消息唯一标识。其中按高位到低位顺序：0-15为发送者地址hash值，16-31为时间戳的后16位，32-64是序列号 |
| int64 msg\_reply\_uuid = 4; | 如果这个消息是用来回复某个消息，那么这个值代表了被回复消息的uuid |
| string server\_sender\_address = 5; | 消息发送者的地址 |
| string server\_single\_receiver\_address = 6; | 如果是单点发送，消息接收者的地址 |
| repeated string server\_multi\_receiver\_address = 7; | 如果是组播信息，消息接收者的地址列表 |
| int32 timeout = 8; | 如果需要回复，那么回复的超时时间 |
| int64 timestamp = 9; | 消息生成时的时间戳 |
| int32 need\_reply\_num = 10; | 如果需要回复，那么需要被回复的个数 |
| string content = 11; | 自定义内容，String类型。 |

其中msg\_type有着如下的定义：

// 询问集群信息

public static final int CLUSTER = 0;

public static final int CLUSTER\_REPLY = 1;

// 定时消息与恢复，用于保活，强制要求设置timeout

public static final int ECHO = 2;

public static final int ECHO\_REPLY = 3;

// 投票，等待回复

public static final int VOTE = 4;

public static final int VOTE\_REPLY = 5;

// 加入集群

public static final int JOIN = 6;

public static final int JOIN\_REPLY = 7;

……

1. 未完成的工作

需要继续完成不同消息下的发送和接收动作。并且需要完善消息的分类：消息应该被分别hmsg的系统级消息和用户级消息，系统级消息屏蔽用户，用户级消息可以使用户自定义处理动作。

由于整体机构和区块链系统有一定的吻合程度。在完成系统后，通过集成共识算法、加密算法和持久化，可以初步完成一个区块链系统。