基于WebSocket可靠消息设计方案

1.为什么需要设计该方案？

1.1原生websocket存在的问题

1. 原生websocket的心跳基于TCP的keepalive机制。这个选项默认发送心跳检测数据包的时间间隔是 7200 秒（2 小时），这时间间隔实在是太长了，不具有实用性。

虽然可以通过设置 keepalive 相关的三个选项TCP\_KEEPIDLE、TCP\_KEEPINTVL 和 TCP\_KEEPCNT来改变这个时间间隔。但是这需要需要专业人员设置处理。并且这个设置不能够按应用隔离。在这个过程中也不方便注入业务层处理逻辑。

1. 原生websocket的**OnMessage方法是同步方法**。直接在OnMessage中实现业务逻辑。耗时业务会阻塞下一条消息的接收。这会造成客户端假死问题。

3.原生webSocket的send方无法**实现可靠消息发送（应用层无应答机制）**。用户调用send方法不能够确认消息发送成功。

**1.3该设计方案解决什么问题？**

1. 解决原生websocket心跳问题。通过自定义心跳机制可以在秒级感应网络环境变化。断线几秒内可以感应网络断开。同时可以注入相应的事件来处理业务逻辑。网络恢复后也能够在秒级实现重连。
2. 使用线程池解决OnMessage同步问题。网络IO线程与业务线程隔离解决同步网络性能为题。假死问题。
3. 在底层抽象一个消息协议层可以实现可靠消息传输（请求响应模型）、同步发送、异步发送特性。
4. 解决原来我们RPA机器人与服务端通行协议过多导致业务复杂度高，解决原生websocket消息不可靠问题。 有了可靠及时的消息通信机制。RPA机器人和控制台通信就可以全部使用websocket通信从而简化业务逻辑提高可靠性。

2.方案提供的特性

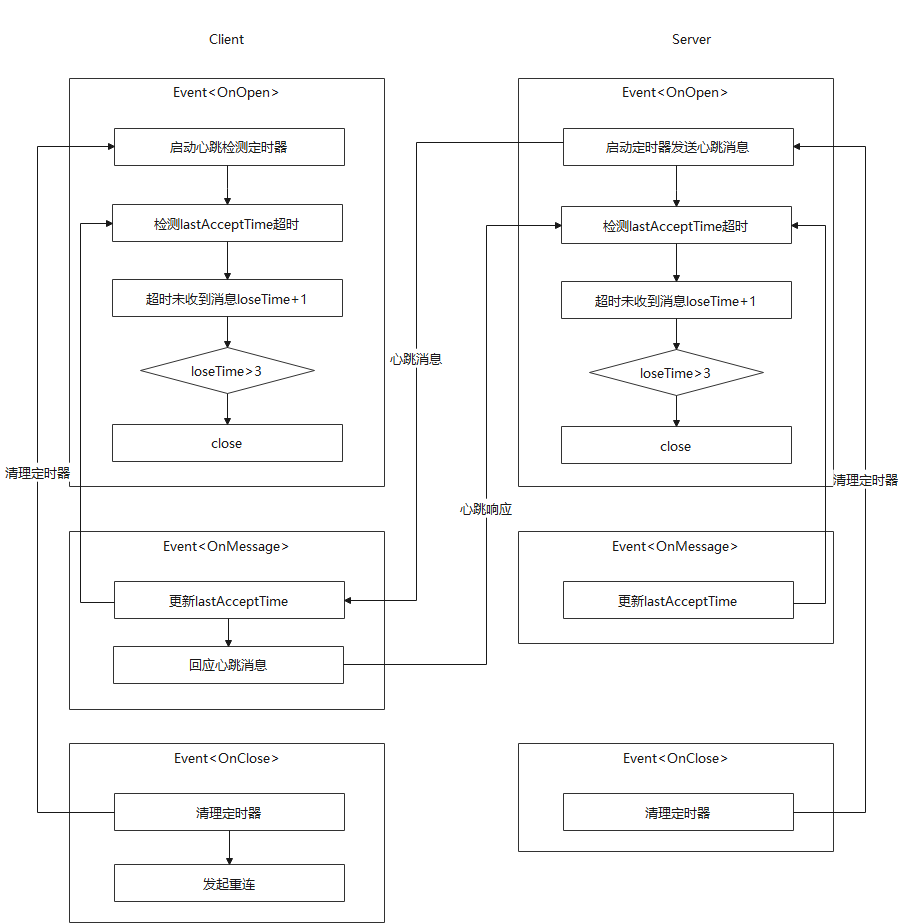
心跳检测、断线重连、异步非阻塞、同步消息、异步消息、请求响应模型

1. 具体实现方案

可靠消息的核心思想是消息确认。发送消息需要加上协议头数据。接受消息解析协议头。基于JSON数据格式实现。简化消息编码解码逻辑。线程池实现异步非阻塞。

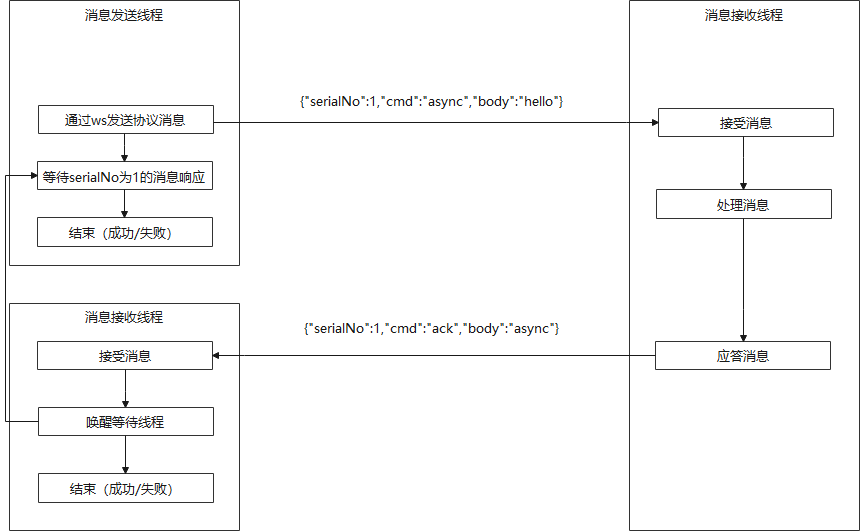
3.1心跳方案

设计流程图

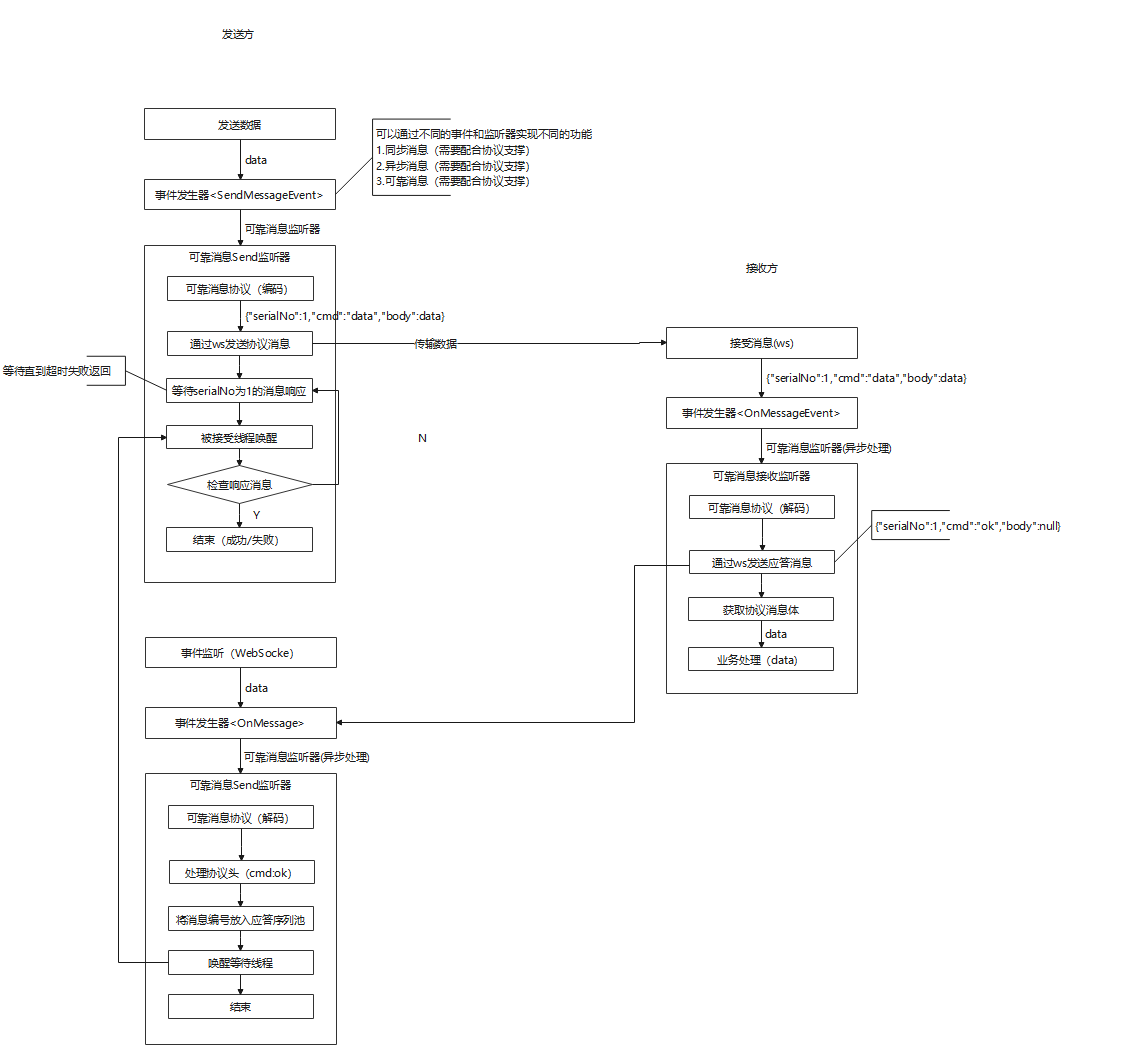


3.2同步消息方案

简化流程图



详细流程图

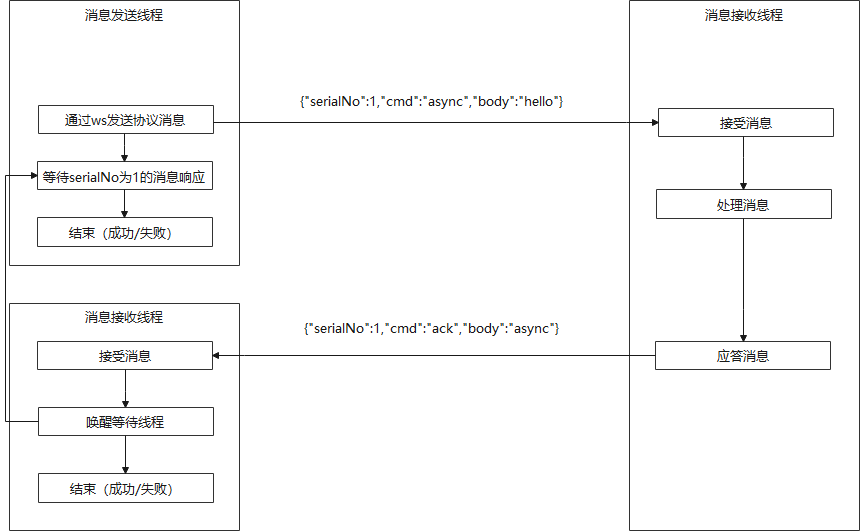


3.3消息协议设计

1. 数据结构

|  |  |  |
| --- | --- | --- |
| 字段 | 说明 | 备注 |
| serialNo | 消息编号 | 发送方各自维护一套 |
| protocol | 协议类型 |  |
| body | 消息体 |  |

同步可靠消息流程



3.3请求响应消息方案

与同步消息逻辑一致。同步可靠消息body返回的是协议类容、这里返回的是对应业务返回的数据。

3.4异步消息方案

略 跟原生websocket一致只管发送数据

同步数据交换示例

|  |  |  |
| --- | --- | --- |
| 发送方数据 | 接受方响应数据 | 说明 |
| {"serialNo":1,"cmd":"async","body":"hello"} | {"serialNo":1,"cmd":"ack","body":"async"} |  |
| {"serialNo":2,"cmd":"async","body":"hello"} | {"serialNo":2,"cmd":"ack","body":"async"} |  |

异步数据交换示例（不可靠方案）

|  |  |  |
| --- | --- | --- |
| 发送方数据 | 接受方响应数据 | 说明 |
| {"serialNo":null,"cmd":"sync","body":"hello"} | {"serialNo":null,"cmd":"ack","body":"sync"} | 接受方响应不是必须的 |
| {"serialNo":null,"cmd":"sync","body":"hello"} | {"serialNo":null,"cmd":"ack","body":"sync"} |

请求响应类型数据交换示例

|  |  |  |
| --- | --- | --- |
| 发送方数据 | 接受方响应数据 | 说明 |
| {"serialNo":1,"cmd":"req-resp","body":"hello"} | {"serialNo":1,"cmd":"ack","body":"message"} | Message为业务数据类似HTTP请求 |
| {"serialNo":2,"cmd":"req-resp","body":"hello"} | {"serialNo":2,"cmd":"ack","body":"message"} |

4.使用成本评估

方案的实现与业务分离、业务不需要关心底层消息是怎么传输的。只需要结合业务需求使用不同的底层API。接入成本低，只需要替换原有业务中关于websocket接受发送相关API即可。Java版的客户端于服务端已经实现。其他语言可以根据这个逻辑来自己实现。

5.基于该方案已经实现的语言

Java

<https://github.com/boyalearn/broheim>

C#

开发中