STOMP Protocol Specification, Version 1.2

STOMP 协议规范 1.2

目录

Abstract 摘要 4

Overview 概述 4

Background 背景 4

Protocol Overview 协议背景 5

Changes in the Protocol 协议变更 8

Design Philosophy 设计思想 9

Conformance 规范性 10

STOMP Frames STOMP 数据帧 11

Value Encoding 编码 13

Body 消息体 14

Standard Headers 标准头 14

Header content-length 15

Header content-type 15

Header receipt 16

Repeated Header Entries 16

Size Limits 大小限制 17

Connection Lingering 延迟 18

Connecting 建立连接 19

CONNECT or STOMP Frame 20

CONNECTED Frame 22

Protocol Negotiation 23

Heart-beating 25

Client Frames 客户端的命令帧 29

SEND 29

SUBSCRIBE 33

SUBSCRIBE id Header 35

SUBSCRIBE ack Header 36

UNSUBSCRIBE 38

ACK 39
NACK 39
BEGIN 40
COMMIT 40
ABORT 41
DISCONNECT 41
Server Frames 服务端的命令帧 42
MESSAGE 42
RECEIPT 44
ERROR 45
Frames and Headers 帧和头信息 47
Augmented BNF 补充 49
License 51

Abstract 摘要

STOMP is a simple interoperable protocol designed for asynchronous message passing between clients via mediating servers. It defines a text based wire-format for messages passed between these clients and servers.

STOMP 是一个简易可交互的协议，可应用于中间服务器和客户端之间进行异步消息传递。它定义了一种在客户端与服务端进行消息传递的文本格式.

STOMP has been in active use for several years and is supported by many message brokers and client libraries. This specification defines the STOMP 1.2 protocol and is an update to **STOMP 1.1**.

STOMP 协议已经应用很多年了,很多消息代理服务或者客户端库也都支持.

Stopm1.2 是对 1.1 版本的更新.

Please send feedback to the stomp-spec@googlegroups.com mailing list.

兄弟要是想给反馈,就给 stomp-spec@googlegroups.com 发邮件.

Overview 概述

Background 背景

STOMP arose from a need to connect to enterprise message brokers from scripting languages such as

Ruby, Python and Perl. In such an environment it is typically logically simple operations that are carried out such as 'reliably send a single message and disconnect' or 'consume all messages on a given destination'.

很多人想用一些脚本语言(Ruby,Python,Perl)连接到企业级的消息服务器,做一些可靠的简单的逻辑操作,比方说"发送单个消息并确定对方能收到","使用对方所有发过来的数据",所以 Stomp 协议应运而生了.

It is an alternative to other open messaging protocols such as AMQP and implementation specific wire protocols used in JMS brokers such as OpenWire. It distinguishes itself by covering a small subset of commonly used messaging operations rather than providing a comprehensive messaging API.

Stomp 是复杂协议(如 AMQP 这种)的替换方案,它不提供全面的消息传递的 API,只是定义了一小部分常用的消息传递操作.

More recently STOMP has matured into a protocol which can be used past these simple use cases in terms of the wire-level features it now offers, but still maintains its core design principles of simplicity and interoperability.

就目前定义的功能来说,Stomp 已是个成熟的协议了,但是 Stomp 依旧保持着简单性和互操作性的核心设计原则.

Protocol Overview 协议背景

STOMP is a frame based protocol, with frames modelled on HTTP. A frame consists of a command, a set of optional headers and an optional body. STOMP is text based but also allows for the transmission of binary messages. The default encoding for STOMP is UTF-8, but it supports the specification of alternative encodings for message bodies.

Stomp 是一个数据帧协议,原型是 HTTP 协议.数据帧包含命令,一组可选的头信息,数据体(可选).Stomp 是基于文本实现的,但也能传二进制数据,默认编码用 UTF-8,消息体也支持别的编码

A STOMP server is modelled as a set of destinations to which messages can be sent. The STOMP protocol treats destinations as opaque string and their syntax is server implementation specific.

Stomp 服务器可以自定义很多个目的地址,来定向实现消息发送.每个地址的功能自己开发.

例如:/topic/state,/topic/publicmessage

都是开发者在服务端自定义的功能地址,第一个实现了状态信息发布,第二个实现了公告信息发布.客户端可以订阅对应的地址,这样订阅这些地址的客户端就可以收到来自这些地址的数据.

通常我的方式是,订阅地址都用一个根目录形式表示,子目录表示具体功能.

再比如:/operation/control_device

这个地址看得出来,就是用来提供给客户端发送操作命令的.

服务端上的地址都是自己定义的,我个人保持比较好的命名规范.自己定义的地址,服务端实现这个地址的功能.客户端开发的时候,按照这个地址订阅,然后按照对应的方式处理接受到的数据.

Additionally STOMP does not define what the delivery semantics of destinations should be. The delivery, or "message exchange", semantics of destinations can vary from server to server and even from destination to destination. This allows servers to be creative with the semantics that they can support with STOMP.

Stomp 没有定义目的地址代表的语义应该是什么.所以服务端可以自由的定义和支持对应的 destination.

例如:

/topic/user:服务端返回的是一个用户的对象

/topic/userList:服务端返回的是用户的列表

每个地址返回的数据类型可以不一样,也可以一样,这个具体开发的时候具体实现.

A STOMP client is a user-agent which can act in two (possibly simultaneous) modes:

Stomp 客户端应该是个消息消费者,也可以是消息生产者,二者也可以同时用.

- as a producer, sending messages to a destination on the server via a SEND frame 作为消息生产者,发送帧消息给服务端

- as a consumer, sending a SUBSCRIBE frame for a given destination and receiving messages from the server as MESSAGE frames.作为消息消费者,订阅服务端提供的目的地址的消息,接受 Stomp 格式的数据

  其实客户端跟服务端建立连接后,需要提交数据到指定的目的地址,那么客户端就是个消息生产者,但是想同步一些服务端的消息,他就是个消息消费者.其实我觉得没必要拎出来.

  通常我们使用 Stomp,只是想搞点数据交互,完了使用 stomp 订阅功能同步下消息.数据体用 json.多客户端的情况下,想同步个状态,这个用数据订阅就很合适.想执行操作就用带 send 命令帧消息体.

Changes in the Protocol 协议变更

STOMP 1.2 is mostly backwards compatible with STOMP 1.1. There are only two incompatible changes:

1.2 版本是向后兼容 1.1 的,有两个不兼容的地方:

- it is now possible to end frame lines with carriage return plus line feed instead of only line feed 现在帧内结构之间的间隔,不只是用换行符,需要回车符加换行符表示
- message acknowledgment has been simplified and now uses a dedicated header 确认帧简化并且使用专用的确认头

Apart from these, STOMP 1.2 introduces no new features but focuses on clarifying some areas of the specification such as:

除此之外，1.2 版本没有引入新功能，但是特意对某些问题做出了澄清：

- repeated frame header entries 重复的帧头信息
- use of the content-length and content-type headers(content-length/content-type 的使用)
- required support of the STOMP frame by servers(要求服务端对 Stomp 的支持,应该是关于版本的型号)
- connection lingering(连接延迟)
- scope and uniqueness of subscription and transaction identifiers(订阅和事务的规范和统一)
- meaning of the RECEIPT frame with regard to previous frames(RECEIPT 帧相对于之前版本的含义)

Design Philosophy 设计思想

The main philosophies driving the design of STOMP are simplicity and interoperability.

Stomp 的主题思想遵循简单性和互操作性

STOMP is designed to be a lightweight protocol that is easy to implement both on the client and server side in a wide range of languages. This implies, in particular, that there are not many constraints on the architecture of servers and many features such

as destination naming and reliability semantics are implementation specific.

Stomp 协议是一个轻量级的协议,能够被各类开发语言支持和实现服务端和客户端.

也就是说服务器是啥结构或者啥特性(命名规范,语法)都没有啥限制.

In this specification we will note features of servers which are not explicitly defined by STOMP 1.2. You should consult your STOMP server's documentation for the implementation specific details of these features.

1.2 中没有定义服务器的特性,具体实现,你应该参考你是用的 stomp 服务的文档

Stomp 其实就是一个简单协议,只是保证了简单性和互操作性.如果你用了别人做好的服务端的框架啥的,具体还是参考你框架提供的文档.

Conformance 规范性

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **RFC 2119**.

文档中使用的这些关键词的解释在这里 **RFC 2119**

```
MUST
MUST NOT
REQUIRED
SHALL
SHALL NOT
SHOULD
SHOULD NOT
```

```
RECOMMENDED
MAY
OPTIONAL
```

Implementations may impose implementation-specific limits on unconstrained inputs, e.g. to prevent denial of service attacks, to guard against running out of memory, or to work around platform-specific limitations.

遵循上面这些关键词的限制,可以防止拒绝式服务攻击，防止内存耗尽，或解决特定于平台的限制。

The conformance classes defined by this specification are STOMP clients and STOMP servers.

本协议规范了 Stomp 协议的服务端和客户端的一致性

STOMP Frames STOMP 数据帧

STOMP is a frame based protocol which assumes a reliable 2-way streaming network protocol (such as TCP) underneath. The client and server will communicate using STOMP frames sent over the stream. A frame's structure looks like:

STOMP 是一种基于数据帧的协议,就像 TCP 协议一样.它假设底层是双向的流数据的可靠交换.客户端和服务端的连接采用 STOMP 数据包交互.数据包的格式如下所示:

```
COMMAND
```

```
header1:value1

header2:value2



Body^@
```

The frame starts with a command string terminated by an end-of-line (EOL), which consists of an OPTIONAL carriage return (octet 13) followed by a REQUIRED line feed (octet 10). Following the command are zero or more header entries in <key>:<value> format. Each header entry is terminated by an EOL. A blank line (i.e. an extra EOL) indicates the end of the headers and the beginning of the body. The body is then followed by the NULL octet. The examples in this document will use ^@, control-@ in ASCII, to represent the NULL octet. The NULL octet can be optionally followed by multiple EOLs. For more details, on how to parse STOMP frames, see the**Augmented BNF** section of this document.

1 数据帧以 COMMAND 命令开始,以 end-of-line (EOL)结束

2 end-of-line(EOL)由一个回车(可选)和一个换行(必须有)组成

名称	英文	描述	ASCII 值	C# 表示
回车符	Carriage Return	回到一行开头	13	\r
换行符	New Line	下一行开头位置	10	\n

3COMMAND 命令那行下面紧跟着的是 header 头信息(头信息可以是 0 或者多个)

4 header 头信息是键值对

5 header 头信息完毕后,下面跟一个 EOL,区分头部和消息体 Body.

6 空白行(例如 EOL)是头部和消息体 Body 的分隔标识

7 消息体 Body 后面跟一个 NULL.

```
public const byte EndOfFrameByte = 0; //  '\0'
```

8 本文中用^@来表示 NULL.

9 NULL 标记后面可以跟多个 EOL

10 更多解析 Stomp 消息体的信息,请看后面附录部分

11 关于解析的代码,我建议参考此处

All commands and header names referenced in this document are case sensitive.

协议中所有 commands 和 header 头信息都大小写敏感.

Value Encoding 编码

The commands and headers are encoded in UTF-8. All frames except the CONNECT and CONNECTED frames will also escape any carriage return, line feed or colon found in the resulting UTF-8 encoded headers.

command 和 header 使用 UTF-8 编码.

除了 CONNECT 和 CONNECTED 数据帧,Stomp 帧的 header 头信息避免使用回车符,换行符和冒号.

Escaping is needed to allow header keys and values to contain those frame header delimiting octets as values. The CONNECT and CONNECTED frames do not escape the carriage return, line feed or colon octets in order to remain backward compatible with STOMP 1.0.

C style string literal escapes are used to encode any carriage return, line feed or colon that are found within the UTF-8 encoded headers. When decoding frame headers, the following transformations MUST be applied:

- \r (octet 92 and 114) translates to carriage return (octet 13)
- \n (octet 92 and 110) translates to line feed (octet 10)
- \c (octet 92 and 99) translates to : (octet 58)
- \\ (octet 92 and 92) translates to \ (octet 92)

Undefined escape sequences such as \t (octet 92 and 116) MUST be treated as a fatal protocol error. Conversely when encoding frame headers, the reverse transformation MUST be applied.

The STOMP 1.0 specification included many example frames with padding in the headers and many servers and clients were implemented to trim or pad header values. This causes problems if applications want to send headers that SHOULD not get trimmed. In STOMP 1.2, clients and servers MUST never trim or pad headers with spaces.

Body 消息体

Only the SEND, MESSAGE, and ERROR frames MAY have a body. All other frames MUST NOT have a body.

Standard Headers 标准头

Some headers MAY be used, and have special meaning, with most frames.

Header content-length

All frames MAY include a content-length header. This header is an octet count for the length of the message body. If a content-length header is included, this number of octets MUST be read, regardless of whether or not there are NULL octets in the body. The frame still needs to be terminated with a NULL octet.

理论上所有的帧都应该包含 content-length 头,值就是整个 body 的字节数.如果存在 content-length 头,所指的 body 体必须是可读的,不论 body 是不是个空字节.帧依旧需要一个 NULL 字节来表示结束.

C#代码直接解释吧:

```
if(Body!=null&&Body!="")

Encoding.UTF8.GetBytes(Body).Length.ToString();
```

If a frame body is present,
the SEND, MESSAGE and ERROR frames SHOULD include a content-length header to ease frame parsing. If the frame body contains NULL octets, the frame MUST include a content-length header.

如果 body 存在,SEND,MESSAGE,ERROR 包都应该包含 content-length,以便帧的解析.

如果 Body 中存在 NULL 字节,头信息必须加上 content-length.

Header content-type

If a frame body is present,
the SEND, MESSAGE and ERROR frames SHOULD include a content-typeheader to help the receiver of the frame interpret its body. If the content-type header is set, its value MUST be a MIME type which

describes the format of the body. Otherwise, the receiver SHOULD consider the body to be a binary blob.

The implied text encoding for MIME types starting with `text/` is UTF-8. If you are using a text based MIME type with a different encoding then you SHOULD append `;charset=<encoding>` to the MIME type. For example, `text/html;charset=utf-16` SHOULD be used if you're sending an HTML body in UTF-16 encoding. The `;charset=<encoding>` SHOULD also get appended to any non `text/` MIME types which can be interpreted as text. A good example of this would be a UTF-8 encoded XML. Its `content-type` SHOULD get set to `application/xml;charset=utf-8`

All STOMP clients and servers MUST support UTF-8 encoding and decoding. Therefore, for maximum interoperability in a heterogeneous computing environment, it is RECOMMENDED that text based content be encoded with UTF-8.

Header receipt

Any client frame other than `CONNECT` MAY specify a `receipt` header with an arbitrary value. This will cause the server to acknowledge the processing of the client frame with a `RECEIPT` frame (see the **RECEIPT** frame for more details).

```
SEND

destination:/queue/a

receipt:message-12345



hello queue a^@
```

Repeated Header Entries

Since messaging systems can be organized in store and forward topologies, similar to SMTP, a message may traverse several messaging servers before reaching a consumer. A STOMP server MAY 'update' header values by either prepending headers to the message or modifying a header in-place in the message.

If a client or a server receives repeated frame header entries, only the first header entry SHOULD be used as the value of header entry. Subsequent values are only used to maintain a history of state changes of the header and MAY be ignored.

For example, if the client receives:

```
MESSAGE

foo:World

foo:Hello



^@
```

The value of the `foo` header is just `World`.

Size Limits 大小限制

To prevent malicious clients from exploiting memory allocation in a server, servers MAY place maximum limits on:

为了防止客户端恶意分配服务端的内存资源,服务端可以在一下几个方面做限制:

- the number of frame headers allowed in a single frame

  header 数量的最多个数

- the maximum length of header lines

  header 的最大长度

- the maximum size of a frame body

  body 的最大长度

If these limits are exceeded the server SHOULD send the client an ERROR frame and then close the connection.

如果超出了限制,服务端应该发送一个 ERROR 包,然后关闭连接

Connection Lingering 延迟

STOMP servers must be able to support clients which rapidly connect and disconnect.

STOMP 服务端必须支持客户端的快速连接和断开连接.

This implies a server will likely only allow closed connections to linger for short time before the connection is reset.

意味着已关闭的连接在重置前,服务端只允许存在一小点延迟.

As a consequence, a client may not receive the last frame sent by the server (for instance an ERRORframe or the RECEIPT frame in reply to a DISCONNECT frame) before the socket is reset.

因此,在连接重置前,客户端可能无法收到服务端发来的最后一个包

(ERROR/RECEIPT) (我个人认为应该在 close 的时候,sleep(一小段时间))

Connecting 建立连接

A STOMP client initiates the stream or TCP connection to the server by sending the CONNECT frame:

客户端通过初始化流数据或者 TCP 连接,发送一个 CONNECT 包:

```
CONNECT

accept-version:1.2

host:stomp.github.org



^@
```

If the server accepts the connection attempt it will respond with a CONNECTED frame:

服务端要是接受连接,就回应一个 CONNECTED 包

```
CONNECTED

version:1.2



^@
```

The server can reject any connection attempt. The server SHOULD respond back with an ERROR frame explaining why the connection was rejected and then close the connection.

服务端可以拒绝任何连接请求,但是得回应一个 ERROR 包说明拒绝原因,

然后关闭连接.

CONNECT or STOMP Frame

STOMP servers MUST handle a STOMP frame in the same manner as a CONNECT frame. STOMP 1.2 clients SHOULD continue to use the CONNECT command to remain backward compatible with STOMP 1.0 servers.

1.2 的 Stomp 服务端**必须**根据 CONNECT 包做出对应的回应.

例如:

version:客户端问咱们用哪个版本的协议,我这边支持哪些版本(accept-version).然后服务端看了下支持的协议列表,最后定了,那就求同存异,使用能够支持的最高版本协议

heart-beat:心跳包要不要,间隔多长时间要,服务端说不要了,太麻烦

…

1.2 版本的客户端也应该继续使用 CONNECT 命令保持兼容之前的 1.0 的服务端.

Clients that use the STOMP frame instead of the CONNECT frame will only be able to connect to STOMP 1.2 servers (as well as some STOMP 1.1 servers) but the advantage is that a protocol sniffer/discriminator will be able to differentiate the STOMP connection from an HTTP connection.

客户端使用 1.2 版本的协议,就只能使用 1.2 版本的协议与服务端交互(1.1 的也一样).这样的好处就是:协议嗅探/协议鉴别能够从 Http 连接中区分出是 Stomp 的连接.

(我自己看半天就按照自己理解写了,如果翻译不好,请看原文理解下,也可以通知我修改)

STOMP 1.2 clients MUST set the following headers:

1.2 版本的客户端必须遵守下面的 header 头约定,也就是必须带下面的头信息.

- accept-version : The versions of the STOMP protocol the client supports. See **Protocol Negotiation** for more details.(客户端支持哪些版本)
- host : The name of a virtual host that the client wishes to connect to. It is recommended clients set this to the host name that the socket was established against, or to any name of their choosing. If this header does not match a known virtual host, servers supporting virtual hosting MAY select a default virtual host or reject the connection.(客户端想要连接的虚拟 stomp 服务主机名,客户端希望 socket 连接到哪个主机.如果服务端这边没有匹配的,服务端可以选择一个默认的或者不接受连接)

STOMP 1.2 clients MAY set the following headers:

客户端也可以设置下面的头信息

- login : The user identifier used to authenticate against a secured STOMP server.(安全验证的服务端需要的用户名)

- passcode : The password used to authenticate against a secured STOMP server. (安全验证的服务端需要的密码)
- heart-beat : The **Heart-beating** settings.(心跳设置)

CONNECTED Frame

STOMP 1.2 servers MUST set the following headers:

1.2 服务端必须回应的头信息

- version : The version of the STOMP protocol the session will be using. See **Protocol Negotiation** for more details.(确定用哪个版本的协议)

STOMP 1.2 servers MAY set the following headers:

1.2 服务端可以携带的头信息(不是一定要带)

- heart-beat : The **Heart-beating** settings.(回应客户端用什么间隔的心跳时间)
- session : A session identifier that uniquely identifies the session.(给客户端分配的 id)
- server : A field that contains information about the STOMP server. The field MUST contain a server-name field and MAY be followed by optional comment fields delimited by a space character.(服务端的信息,必须包括服务端的名称后面可以跟一个注释信息,中间用空格字符分隔)

The server-name field consists of a name token followed by an optional version number token.

服务端名字还可以用反斜杠带一个版本

server = name ["/" version] *(comment)

Example:

例如可以这样写:

Server:hostname/1.0.0 空格 thisIsAComment

server:Apache/1.3.9

Protocol Negotiation

From STOMP 1.1 and onwards, the CONNECT frame MUST include the accept-version header. It SHOULD be set to a comma separated list of incrementing STOMP protocol versions that the client supports. If the accept-version header is missing, it means that the client only supports version 1.0 of the protocol.

1.1 及以后的版本, CONNECT 包里的头信息必须设置 accept-version,表示客户端自己支持的版本.版本逐步自增,中间使用英文逗号分隔开

要是不加这个头信息,就表示默认支持 1.0 版本的协议

The protocol that will be used for the rest of the session will be the highest protocol version that both the client and server have in common.

协议协商最后采用双方都支持的最高的版本.

For example, if the client sends:

例如下面,客户端发送了一个 CONNECT 包,里面包含自己支持的协议版本.

```
CONNECT

accept-version:1.0,1.1,2.0

host:stomp.github.org



^@
```

The server will respond back with the highest version of the protocol that it has in common with the client:

服务端这边最高支持到 1.1,所以就返回了一个 1.1,表示后面就采用 1.1 版本的协议来做数据交互

```
CONNECTED

version:1.1



^@
```

If the client and server do not share any common protocol versions, then the server MUST respond with an ERROR frame similar to the following and then close the connection:

如果客户端和服务端没有共同支持的版本,服务端就必须回复一个

ERROR 包,大概就像下面的一样,然后服务端就断开连接.

```
ERROR

version:1.2,2.1

content-type:text/plain



Supported protocol versions are 1.2 2.1^@
```

Heart-beating

Heart-beating can optionally be used to test the healthiness of the underlying TCP connection and to make sure that the remote end is alive and kicking.

心跳包是可选项,目的是为了测试底层 TCP 连接的正常,确保远程机器

还活着

In order to enable heart-beating, each party has to declare what it can do and what it would like the other party to do. This happens at the very beginning of the STOMP session, by adding a heart-beat header to the CONNECT and CONNECTED frames.

为了保持心跳,双方都得明确自己能做啥,希望对方做啥,在建立连接

之初就确定好了,其实就发生在 CONNECT and CONNECTED 交互过程中.

When used, the heart-beat header MUST contain two positive integers separated by a comma.

如果确定要保持心跳,header 的 heart-beat 部分必须包含两个英文逗号分隔开的的正整数.

The first number represents what the sender of the frame can do (outgoing heart-beats):

第一个数字表示,发送者可以做啥

- 0 means it cannot send heart-beats(数字 0 表示发送者不发送心跳包)
- otherwise it is the smallest number of milliseconds between heart-beats that it can guarantee(其他数值表示发送者发送心跳的最小时间间隔)

The second number represents what the sender of the frame would like to get (incoming heart-beats):

逗号后面的数字,也就是第二个数字,希望对方做啥

- 0 means it does not want to receive heart-beats(发送者不希望对方发送心跳包)
- otherwise it is the desired number of milliseconds between heart-beats(发送者希望对方间隔某毫秒发送一个心跳包)

The heart-beat header is OPTIONAL. A missing heart-beat header MUST be treated the same way as a "heart-beat:0,0" header, that is: the party cannot send and does not want to receive heart-beats.

心跳头信息是可选项.客户端没有设置,那么就不当成不想发送心跳包,也不想接受心跳包.

The heart-beat header provides enough information so that each party can find out if heart-beats can be used, in which direction, and with which frequency.

More formally, the initial frames look like:

心跳头已经提供了足够的信息明确了双方间怎么交互:发送的方向,发送的频率,接受的方向和接受的频率

.

```
CONNECT

heart-beat:<cx>,<cy>



CONNECTED

heart-beat:<sx>,<sy>
```

For heart-beats from the client to the server:

如果客户端给服务端发送心跳

- if <cx> is 0 (the client cannot send heart-beats) or <sy> is 0 (the server does not want to receive heart-beats) then there will be none

  cx=0,客户端不想发或者不支持心跳

  sy=0,服务端不想接受心跳包

  那么双方就互相不发心跳了

- otherwise, there will be heart-beats every MAX(<cx>,<sy>) milliseconds

  MAX(<cx>,<sy>)这里是个函数,求最大值.表示在这个最大值的

  毫秒时间发送一次心跳.

In the other direction, <sx> and <cy> are used the same way.

服务端发送给客户端的方向上也是同理

Regarding the heart-beats themselves, any new data received over the network connection is an indication that the remote end is alive. In a given direction, if heart-beats are expected every <n>milliseconds:

对于心跳机制来说,接收到从网络来的任何新数据都表示连接是正常

的.

在给定的方向上,希望每隔 n 毫秒发送一次心跳:

- the sender MUST send new data over the network connection at least every <n> milliseconds

  在已经建立连接的基础上,发送者必须至少每隔 n 毫秒发送一个

  新数据

- if the sender has no real STOMP frame to send, it MUST send an end-of-line (EOL)

  如果发送者没有 stomp 数据包要发,就必须发一个 EOL 包

- if, inside a time window of at least <n> milliseconds, the receiver did not

receive any new data, it MAY consider the connection as dead

如果在等待的 n 毫秒内,接收者没有受到任何新数据,就认为连接断开了

- because of timing inaccuracies, the receiver SHOULD be tolerant and take into account an error margin

因为定时不精准,接收者也要有点容错时间.

Client Frames 客户端的命令帧

A client MAY send a frame not in this list, but for such a frame a STOMP 1.2 server MAY respond with an ERROR frame and then close the connection.

- **SEND**
- **SUBSCRIBE**
- **UNSUBSCRIBE**
- **BEGIN**
- **COMMIT**
- **ABORT**
- **ACK**
- **NACK**
- **DISCONNECT**

SEND

The SEND frame sends a message to a destination in the messaging system. It has one REQUIRED header, destination, which indicates where to send the message. The body of the SEND frame is the message to be sent. For example:

SEND 包是一个带有远程地址(destination)头,body 带有信息的 Stomp 格式的包.所以 header 头里面必须有"destination"这个头信息,表示发给 server 的这个地址.Body 里面是要发送的数据.

例如下面这个 SEND 包

```
SEND

destination:/queue/a

content-type:text/plain



hello queue a

^@
```

This sends a message to a destination named /queue/a. Note that STOMP treats this destination as an opaque string and no delivery semantics are assumed by the name of a destination. You should consult your STOMP server's documentation to find out how to construct a destination name which gives you the delivery semantics that your application needs.

上面这个例子,目的地址是: /queue/a

STOMP 协议是不理解这个地址表示什么的,只会认为是一个字符串.

所以在对接服务端的时候,需要跟服务端开发那边提供的地址,还有服务端那边解析什么格式的数据,你就需要把这个数据填充为什么格式.

举例一个服务端提供的接口

Springboot websocketController 中的一个地址实现

```
1 @MessageMapping(“/send-command”)
2 @SendTo(“/topic/sync-command-result”)
3 public Map<String, ResultX> controlDevices(Map<String, CommandX> map) {
```

```
4     return controlService.SendCommand(map);
5 }
```

在这个框架中:第 1 行,表示服务端接受的地址是”/send-command”,

那么客户端的 destination: /send-command

然后看第 3 行函数的参数 Map<String, CommandX> map

是说服务端接受一个 map,

客户端的 body 就需要将这个 map 放进去.

所以在具体跟服务端对接的时候,需要具体看服务端那边提供了什么地址,要接受什么样类型的数据.Stomp 协议是不管这些的.

_____(稍微再发散一点,不理解也没关系)

在这个实际的例子中,map 对象其实是被转成了 json 字符串.服务端接受的数据会把这个字符串转换成相应的对象.

第 2 行,是服务端处理完毕后,发送给客户端的.到时候 server->client 的 MESSAGE 包的头里就有这样的字段

destination: /topic/sync-command-result

这个地址其实是个订阅地址,所有订阅了这个地址的客户端都将收到执行后的数据,目的是同步信息.后面会有 SUBSCRIBE 包的具体解释

第 3 行的函数返回依旧返回一个 map,所以这个 MESSAGE 包的 Body 部分就是第 4 行 return 的对象的 Json 字符串.

The reliability semantics of the message are also server specific and will depend on the destination value being used and the other message headers such as the transaction header or other server specific message headers.

Body 部分的数据也是服务端定义的,要看服务端提供的 destination 地址接口需要什么样的数据,或者看 transaction 接口,或者服务端那边自定义头要什么样的数据.

就像上面哪个 java 的接口一样,函数需要的是个 Map 对象(jsonString),客户端就得传一个 Map 对象(tojson).

SEND supports a transaction header which allows for transactional sends.

SEND 包的 header 支持一个 transaction 头,允许发送事务信息.

SEND frames SHOULD include a **content-length** header and a **content-type** header if a body is present.

SEND 包里的 body 有数据,则应该加上 **content-length 和 content-type** 头信息

An application MAY add any arbitrary user defined headers to the SEND frame. User defined headers are typically used to allow consumers to filter messages based on the application defined headers

using a selector on a SUBSCRIBE frame. The user defined headers MUST be passed through in the MESSAGE frame.

Header 头里允许加入自定义的头信息.

通常是服务端已经定义好可允许的 header 头信息,然后客户端通过 SUBSCRIBE 帧来订阅这些头信息,从里面选择.

用户自定义的头信息,必须使用 MESSAGE 帧来传递.(这里应该是服务端定义好可选的头信息列表,完了客户端订阅头信息,服务端将这些头信息放在 MESSAGE 帧里面发送给客户端.因为 MESSAGE 帧只有服务端给客户端发)

If the server cannot successfully process the SEND frame for any reason, the server MUST send the client an ERROR frame and then close the connection.

如果服务端接收到了客户端发过来的 SEND 包,但是解析失败,服务端就返回一个 ERROR 包,然后关闭这个连接.(想攻击?协议不允许!)

SUBSCRIBE

The SUBSCRIBE frame is used to register to listen to a given destination. Like the SEND frame, the SUBSCRIBE frame requires a destination header indicating the destination to which the client wants to subscribe. Any messages received on the subscribed destination will henceforth be delivered as MESSAGE frames from the server to the client. The ack header controls the message acknowledgment mode.

SUBSCRIBE 包其实就在服务端注册一下,用来监听服务端某 destination 地址的数据.跟 SEND 包一样,必须包含 destination 头信息,用来订阅这个地址的数据.

服务端上这个地址产生的任何数据,都将发送给每个订阅这个地址的客户端.服务端发过去的包格式是一个 MESSAGE 包.

ack 头信息控制客户端对这个包怎么确认.

Example:

```
SUBSCRIBE

id:0

destination:/queue/foo

ack:client



^@
```

If the server cannot successfully create the subscription, the server MUST send the client an ERROR frame and then close the connection.

如果服务端无法完成客户端请求的订阅,那么服务端必须随手扔给客户端一个 ERROR 包, 然后立马关闭连接

STOMP servers MAY support additional server specific headers to customize the delivery semantics of the subscription. Consult your server's documentation for details.

STOMP 服务端的订阅功能方面,可能支持自定义添加特殊头信息,这些头信息表示别的语义,具体看 Stomp 服务端那边的文档来实现(好好扯皮).

SUBSCRIBE id Header

Since a single connection can have multiple open subscriptions with a server, an id header MUST be included in the frame to uniquely identify the subscription. The id header allows the client and server to relate
subsequent MESSAGE or UNSUBSCRIBE frames to the original subscription.

一个连接中,可以订阅服务端多个地址的数据,订阅包里的 id 头就必须是唯一的,这个订阅 id 将用在 MESSAGE 和取消订阅的操作中.

意思就是说,同一个连接中,如果某客户端订阅了服务端 /pathA,/pathB,/pathC 三个地址的数据,每个订阅的 subid 都不可以相同.

这样,服务端生成数据发送 MESSAGE 包中,就会带上这个客户端订阅的 subid,客户端收到包后,就可以认识来自哪个地址的数据了.

所以 ID 必须唯一.

Within the same connection, different subscriptions MUST use different subscription identifiers.

同一个连接中,订阅不同的地址,所使用的 subid 不可以一样.

SUBSCRIBE ack Header

The valid values for the ack header are auto, client, or client-individual. If the header is not set, it defaults to auto.

SUBSCRIBE 包的头里面,ack 项的值有:

auto, client, client-individual

客户端如果没有设置 ack,缺省值为:auto.

When the ack mode is auto, then the client does not need to send the server ACK frames for the messages it receives. The server will assume the client has received the message as soon as it sends it to the client. This acknowledgment mode can cause messages being transmitted to the client to get dropped.

ack: auto;//客户端收到 MESSSAGE 包,不必回复 ACK 帧

服务端会认为,一旦自己发出了 MESSAGE 包,客户端肯定收到了.但是这个确认机制,有可能发生服务端发送出去了,但是客户端没有收到包,MESSAGE 包可能被丢掉了.

When the ack mode is client, then the client MUST send the server ACK frames for the messages it processes. If the connection fails before a client sends an ACK frame for the message the server will assume the message has not been processed and MAY redeliver the message to another client.

ack: client;//客户端收到 MESSSAGE 包,一定回复确认帧

在客户端返回 ACK 包确认之前,连接中断了,那么服务端认为这个消息没有被处理, 会将这个信息再次发送给另一个客户端(emm…你发的消息没有被确认,完了你发给另一个人干嘛,对不对)

The ACK frames sent by the client will be treated as a cumulative acknowledgment. This means the acknowledgment operates on the message specified in the ACK frame and all messages sent to the subscription before the ACK'ed message.

客户端们回复的 ACK 信息,将会被累加处理.这就是说,信息确定这个操作,不仅确认了单个 ACK 信息,群发信息的 ACK 都被批量确认掉了.

(累加机制回复了上面为啥随便给别人发没被确认的包,这样其实是为了早点清空服务端这边的缓存信息,防止内存撑爆,在实现这个功能的时候,可以考虑信息不要占用内存时间过长,及时清理)

In case the client did not process some messages, it SHOULD send NACK frames to tell the server it did not consume these messages.

为了防止客户端收到了信息,但是没有处理这个信息,客户端应该返还一个 NACK 包给服务端,表示东西我收到了,但是没有处理这个数据包.

When the ack mode is client-individual, the acknowledgment operates just like the client acknowledgment mode except that the ACK or NACK frames sent by the client are not cumulative. This means that an ACK or NACK frame for a subsequent message MUST NOT cause a previous message to get acknowledged.

```
ack: client-individual;//客户端要自己确认掉 MESSAGE
```

确认操作跟上面的确认模式(ack: client;)一样的,但是 ACK 和 NACK 不进行累加.也就是说发给谁的,谁来确认,服务端不考虑所有客户端都收到了信息,并且将消息处理完毕.

(这里貌似给服务端增加了点压力,也就是说给每个客户端都准备了消息,客户端不确认收到,这个消息就在这个过程中保留着,服务端可能有自己的机制来处理超时的消息,可能会选择再发一次,也可能就这么保留着等待销毁,或者撑爆内存.但是这样的好处就是确保信息能够到达客户端,并且处理掉了消息)

UNSUBSCRIBE

The UNSUBSCRIBE frame is used to remove an existing subscription. Once the subscription is removed the STOMP connections will no longer receive messages from that subscription.

UNSUBSCRIBE 用来移出服务端那边已经订阅的地址.一旦执行了该操作,客户端就不再收到来自那个订阅地址的数据了

Since a single connection can have multiple open subscriptions with a server, an id header MUST be included in the frame to uniquely identify the subscription to remove. This header MUST match the subscription identifier of an existing subscription.

一个客户端到服务端的连接中,客户端订阅了很多地址,id 头信息是唯一作为移出订阅的标志,id 头信息的值必须是原来订阅时候的那个值.

Example:

```
UNSUBSCRIBE

id:0



^@
```

ACK

ACK is used to acknowledge consumption of a message from a subscription using client or client-individual acknowledgment. Any messages received from such a subscription will not be considered to have been consumed until the message has been acknowledged via an ACK.

The ACK frame MUST include an id header matching the ack header of the MESSAGE being acknowledged. Optionally, a transaction header MAY be specified, indicating that the message acknowledgment SHOULD be part of the named transaction.

```
ACK

id:12345

transaction:tx1



^@
```

NACK

NACK is the opposite of ACK. It is used to tell the server that the client did not consume the message. The server can then either send the message to a different client, discard it, or put it in a dead letter queue. The exact behavior is server specific.

NACK takes the same headers as ACK: id (REQUIRED) and transaction (OPTIONAL).

NACK applies either to one single message (if the subscription's ack mode is client-individual) or to all messages sent before and not yet ACK'ed or NACK'ed (if the subscription's ack mode is client).

BEGIN

BEGIN is used to start a transaction. Transactions in this case apply to sending and acknowledging - any messages sent or acknowledged during a transaction will be processed atomically based on the transaction.

```
BEGIN

transaction:tx1



^@
```

The transaction header is REQUIRED, and the transaction identifier will be used for SEND, COMMIT, ABORT, ACK, and NACK frames to bind them to the named transaction. Within the same connection, different transactions MUST use different transaction identifiers.

Any started transactions which have not been committed will be implicitly aborted if the client sends a DISCONNECT frame or if the TCP connection fails for any reason.

COMMIT

COMMIT is used to commit a transaction in progress.

```
COMMIT

transaction:tx1
```

```
^@
```

The transaction header is REQUIRED and MUST specify the identifier of the transaction to commit.

ABORT

ABORT is used to roll back a transaction in progress.

```
ABORT

transaction:tx1



^@
```

The transaction header is REQUIRED and MUST specify the identifier of the transaction to abort.

DISCONNECT

A client can disconnect from the server at anytime by closing the socket but there is no guarantee that the previously sent frames have been received by the server. To do a graceful shutdown, where the client is assured that all previous frames have been received by the server, the client SHOULD:

1. send a DISCONNECT frame with a receipt header set. Example:

```
2. DISCONNECT

3. receipt:77
```

```
^@
```

4. wait for the RECEIPT frame response to the DISCONNECT. Example:

```
5. RECEIPT

6. receipt-id:77
```

```
^@
```

7. close the socket.

Note that, if the server closes its end of the socket too quickly, the client might never receive the expected RECEIPT frame. See the **Connection Lingering** section for more information.

Clients MUST NOT send any more frames after the DISCONNECT frame is sent.

Server Frames 服务端的命令帧

The server will, on occasion, send frames to the client (in addition to the initial CONNECTED frame). These frames MAY be one of:

- **MESSAGE**
- **RECEIPT**
- **ERROR**

MESSAGE

MESSAGE frames are used to convey messages from subscriptions to the client.

MESSAGE 包是发送订阅数据给客户端的.(客户端的 SEND 包也算订阅,相当于一对一的订阅)

The MESSAGE frame MUST include a destination header indicating the destination the message was sent to. If the message has been sent using STOMP,

this destination header SHOULD be identical to the one used in the corresponding SEND frame.

MESSAGE 包头里面必须包含 destination 头,说明是哪个订阅地址发送的.

客户端如果发送了一个 SEND 包过来,那么 MESSAGE 包回去的时候,头信息中的 destination 的值是一样的.(其实就是客户端 SEND 给服务端的哪个地址,服务端返回 message 的时候就说这个 MESSAGE 是谁发过来的)

The MESSAGE frame MUST also contain a message-id header with a unique identifier for that message and a subscription header matching the identifier of the subscription that is receiving the message.

MESSAGE 包头信息也必须带一个 message-id 头信息,这个 id 必须是唯一的,在这个订阅地址(destination)生产的所有消息中,这个信息是唯一的.

If the message is received from a subscription that requires explicit acknowledgment (either client or client-individual mode) then the MESSAGE frame MUST also contain an ack header with an arbitrary value. This header will be used to relate the message to a subsequent ACK or NACKframe.

如果这个 MESSAGE 是当初订阅时候要求要确认的(ack: client/client-individual),那么 MESSAGE 包的头信息里,必须包含一个 ack 头:ack:任意值,这个头信息将被客户端在后面回复 ACK 或者 NACK 包确认掉.

The frame body contains the contents of the message:

```
MESSAGE

subscription:0

message-id:007

destination:/queue/a

content-type:text/plain



hello queue a^@
```

MESSAGE frames SHOULD include a **content-length** header and a **content-type** header if a body is present.

MESSAGE 包应该加上 **content-length** 和 **content-type** 这两个头信息.

MESSAGE frames will also include all user defined headers that were present when the message was sent to the destination in addition to the server specific headers that MAY get added to the frame. Consult your server's documentation to find out the server specific headers that it adds to messages.

MESSAGE 包也应该加上用户自定义的包头,主要看服务端开发那边的定义.

RECEIPT

A RECEIPT frame is sent from the server to the client once a server has successfully processed a client frame that requests a receipt.

RECEIPT:收条;回执;收货;收货地

RECEIPT 包是服务端发给客户端的.客户端的包携带了需要回执的请求,服务端成功处理了这个包的内容后,就返回一个 RECEIPT 包.

A RECEIPT frame MUST include the header receipt-id, where the value is the value of the receipt header in the frame which this is a receipt for.

RECEIPT 包头里面必须包含 receipt-id 头信息.值就是客户端发过来的包里当初带的那个值,表示服务端让客户端知道自己处理了那个包.

```
RECEIPT

receipt-id:message-12345



^@
```

A RECEIPT frame is an acknowledgment that the corresponding client frame has been *processed* by the server. Since STOMP is stream based, the receipt is also a cumulative acknowledgment that all the previous frames have been *received* by the server. However, these previous frames may not yet be fully *processed*. If the client disconnects, previously received frames SHOULD continue to get processed by the server.

ERROR

The server MAY send ERROR frames if something goes wrong. In this case, it MUST then close the connection just after sending the ERROR frame. See the next section about **connection lingering**.

如果服务端执行过程中出错,服务端可能发送一个 ERROR 包.

只要出现 ETRROR 包,那么服务端将这个 ERROR 包发送完后,就关闭这个客户端的连接.

The ERROR frame SHOULD contain a message header with a short description of the error, and the body MAY contain more detailed information (or MAY be empty).

ERROR 包的 header 中应该包含一些简单的错误描述,body 部分可以包含更多的错误解释信息,也可以为空.

```
ERROR

receipt-id:message-12345

content-type:text/plain

content-length:171

message:malformed frame received


The message:

-----

MESSAGE

destined:/queue/a

receipt:message-12345


Hello queue a!

-----

Did not contain a destination header, which is REQUIRED

for message propagation.
```

```
^@
```

If the error is related to a specific frame sent from the client, the server SHOULD add additional headers to help identify the original frame that caused the error. For example, if the frame included a receipt header, the ERROR frame SHOULD set the receipt-id header to match the value of the receipt header of the frame which the error is related to.

如果 ERROR 包是关于客户端发过来的一个特定的包,服务端应该添加上一些帮助头信息,帮助识别错误帧.举例来说,如果源帧有 receipt 头,ERROR 帧里就应该设置 receipt-id,值就源帧 receipt 的值.

ERROR frames SHOULD include a **content-length** header and a **content-type** header if a body is present.

ERROR 包的 body 不为空的话,头信息里面也应该加上 **content-length** 和 **content-type** .

Frames and Headers 帧和头信息

In addition to the **standard headers** described above (content-length, content-type and receipt), here are all the headers defined in this specification that each frame MUST or MAY use:

本协议对 header 头做了扩充,加上之前的标准的头信息,在这里定义了所有头信息的必须和非必须:

- CONNECT or STOMP
    - 必须: accept-version, host
    - 可选: login, passcode, heart-beat

- CONNECTED
  - 必须: version
  - 可选: session, server, heart-beat
- SEND
  - 必须: destination
  - 可选: transaction
- SUBSCRIBE
  - 必须: destination, id
  - 可选: ack
- UNSUBSCRIBE
  - 必须: id
  - 可选: none
- ACK or NACK
  - 必须: id
  - 可选: transaction
- BEGIN or COMMIT or ABORT
  - 必须: transaction
  - 可选: none
- DISCONNECT
  - 必须: none
  - 可选: receipt
- MESSAGE
  - 必须: destination, message-id, subscription
  - 可选: ack
- RECEIPT
  - 必须: receipt-id
  - 可选: none

- ERROR
    - 必须：none
    - 可选：message

In addition, the SEND and MESSAGE frames MAY include arbitrary user defined headers that SHOULD be considered as being part of the carried message. Also, the ERROR frame SHOULD include additional headers to help identify the original frame that caused the error.

在扩展头信息中,SEND 帧和 MESSAGE 帧可以包含任意自定义的头信息,也会被当做帧信息的一部分.同样,ERROR 包也带上错误头信息,帮助客户端定位出帧信息头的错误的地方.

Finally, STOMP servers MAY use additional headers to give access to features like persistency or expiration. Consult your server's documentation for details.

最后,STOMP 服务端可以用附加头信息的过期或者永久性特征做访问控制.具体参考你所使用的服务端的文档.

(其实在实际开发过程中,有时候会自定义一些头信息,因为你不想把数据都放在 body 中,但是信息又不能不放,server 端就可以考虑在 header 里取,客户端到时候把小参数填写上,这样就比较好处理了)

Augmented BNF 补充

A STOMP session can be more formally described using the Backus-Naur Form (BNF) grammar used in HTTP/1.1 **RFC 2616**.

```
NULL              = <US-ASCII null (octet 0)>

LF                = <US-ASCII line feed (aka newline) (octet
10)>

CR                = <US-ASCII carriage return (octet 13)>

EOL               = [CR] LF

OCTET             = <any 8-bit sequence of data>


frame-stream       = 1*frame


frame             = command EOL

                    *( header EOL )

                    EOL

                    *OCTET

                    NULL

                    *( EOL )


command           = client-command | server-command


client-command     = "SEND"

                    | "SUBSCRIBE"

                    | "UNSUBSCRIBE"

                    | "BEGIN"

                    | "COMMIT"

                    | "ABORT"

                    | "ACK"
```

```
                    | "NACK"

                    | "DISCONNECT"

                    | "CONNECT"

                    | "STOMP"


server-command     = "CONNECTED"

                    | "MESSAGE"

                    | "RECEIPT"

                    | "ERROR"


header             = header-name ":" header-value

header-name        = 1*<any OCTET except CR or LF or ":">

header-value       = *<any OCTET except CR or LF or ":">
```

License