**Abstract**

STOMP是一个简单的互操作协议，用于通过中介服务器在客户机之间进行异步消息传递。它为这些客户机和服务器之间传递的消息定义了一种基于文本的连接格式。

STOMP已经活跃使用了好几年，并且得到了许多消息代理和客户机库的支持。本规范定义了STOMP 1.2协议，是对STOMP 1.1的更新。

请将反馈发送至 stomp-spec@googlegroups.com 邮件列表。

## Overview

### Background

STOMP 源于需要从 Ruby、Python 和 Perl 等脚本语言连接到企业消息代理。 在这样的环境中，通常执行逻辑上简单的操作，例如“可靠地发送单个消息并断开连接”或“使用给定目的地上的所有消息”。

它是其他开放消息传递协议（如 AMQP）和 JMS 代理（如 OpenWire）中使用的特定于实现的有线协议的替代方案。 它通过覆盖一小部分常用的消息传递操作而不是提供全面的消息传递 API 来区分自己。

最近，STOMP 已经成熟为一种协议，就它现在提供的线级功能而言，它可以超越这些简单的用例，但仍保持其简单性和互操作性的核心设计原则。

### Protocol Overview

STOMP 是一种基于帧的协议，其帧以 HTTP 为模型。 一个帧由一个命令、一组可选的标题和一个可选的正文组成。 STOMP 是基于文本的，但也允许传输二进制消息。 STOMP 的默认编码是 UTF-8，但它支持消息体的替代编码规范。

STOMP 服务器被建模为一组可以向其发送消息的目的地。 STOMP 协议将目的地视为不透明的字符串，并且它们的语法是特定于服务器实现的。 此外，STOMP 没有定义目的地的交付语义应该是什么。 目的地的传递或“消息交换”语义可能因服务器而异，甚至因目的地而异。 这允许服务器使用 STOMP 可以支持的语义进行创造性。

STOMP 客户端是一个用户代理，它可以在两种（可能同时）模式下运行：

1. 作为生产者，通过 SEND 帧向服务器上的目的地发送消息。
2. 作为消费者，发送给定目的地的 SUBSCRIBE 帧，并从服务器接收消息作为 MESSAGE 帧。

### Changes in the Protocol

STOMP 1.2 主要向后兼容 STOMP 1.1。 只有两个不兼容的更改：

1. 现在可以用回车加换行来结束框架行，而不仅仅是换行。
2. 消息确认已简化，现在使用专用标头。

除此之外，STOMP 1.2 没有引入任何新功能，而是专注于阐明规范的某些领域，例如：

1. 重复的帧头条目
2. 使用 content-length 和 content-type 标头
3. 服务器需要对 STOMP 框架的支持
4. 连接挥之不去
5. 订阅和事务标识符的范围和唯一性
6. RECEIPT 帧相对于先前帧的含义

### Design Philosophy

推动 STOMP 设计的主要理念是简单性和互操作性。

STOMP 被设计成一个轻量级的协议，很容易在客户端和服务器端以多种语言实现。 这尤其意味着对服务器架构没有太多限制，并且目的地命名和可靠性语义等许多功能都是特定于实现的。

在本规范中，我们将注意 STOMP 1.2 未明确定义的服务器特性。 您应该查阅 STOMP 服务器的文档以了解这些功能的具体实现细节。

## Conformance

本文档中的关键词“必须”、“不得”、“要求”、“应”、“不应”、“应该”、“不应”、“推荐”、“可以”和“可选”是 按照 RFC 2119 中的说明进行解释。

实现可能会对不受约束的输入施加特定于实现的限制，例如 防止拒绝服务攻击，防止内存不足或解决特定于平台的限制。

本规范定义的一致性类是 STOMP 客户端和 STOMP 服务器。

## STOMP Frames

STOMP 是一种基于帧的协议，它假定底层有一个reliable 2-way streaming network protocol（例如 TCP）。 客户端和服务器将使用通过流发送的 STOMP 帧进行通信。 框架的结构如下所示：

COMMAND

header1:value1

header2:value2

Body^@

该帧以一个以行尾 (EOL) 结尾的COMMAND字符串开始，该COMMAND字符串由一个可选的回车符 (octet 13) 和一个必需的换行符 (octet 10) 组成。 COMMAND后面是零个或多个 <key>:<value> 格式的标题条目。 每个标头条目都由 EOL 终止。 空行（即额外的 EOL）表示标题的结尾和正文的开头。 正文之后是 NULL 八位字节。 本文档中的示例将使用 ASCII 中的 ^@、control-@ 来表示 NULL 八位字节。 NULL 八位字节后面可以选择跟随多个 EOL。 有关如何解析 STOMP 帧的更多详细信息，请参阅本文档的增强 BNF 部分([**Augmented BNF**](https://stomp.github.io/stomp-specification-1.2.html#Augmented_BNF))。

本文档中引用的所有commands和header 名称均区分大小写。

### Value Encoding

commands和headers以 UTF-8 编码。 除了 CONNECT 和 CONNECTED 帧之外的所有帧也将转义在生成的 UTF-8 编码标头中找到的任何回车、换行或冒号。

需要转义以允许header keys and values包含那些帧标题分隔八位字节作为值。 CONNECT 和 CONNECTED 帧不会转义回车、换行或冒号八位字节，以保持与 STOMP 1.0 的向后兼容。

C 风格的字符串文字转义用于编码在 UTF-8 编码的标头中找到的任何回车、换行或冒号。 解码帧头时，必须应用以下转换：

1. \r（八位字节 92 和 114）转换为回车（八位字节 13）
2. \n（八位字节 92 和 110）转换为换行符（八位字节 10）
3. \c（八位字节 92 和 99）转换为：（八位字节 58）
4. \\（八位字节 92 和 92）转换为 \（八位字节 92）

未定义的转义序列，例如 \t（八位字节 92 和 116）必须被视为致命的协议错误。 相反，当编码帧头时，必须应用逆变换。

STOMP 1.0 规范包括许多在标头中带有填充的示例帧，并且实现了许多服务器和客户端来修剪或填充标头值。 如果应用程序想要发送不应该被修剪的标头，这会导致问题。 在 STOMP 1.2 中，客户端和服务器绝不能用空格修剪或填充标题。

### Body

只有 SEND、MESSAGE 和 ERROR 帧可以有Body。 所有其他帧不得有Body。

### Standard Headers

大多数帧都可以使用一些headers，并且具有特殊含义。

#### Header content-length

所有的帧都可以包含一个content-length header。 此header是消息正文长度的八位字节计数。 如果包含content-length标头，则必须读取此字节数，无论正文中是否有 NULL 字节。 该帧仍需要以 NULL 八位字节终止。

如果帧存在body，SEND, MESSAGE和ERROR帧应该包含一个content-length的header，以便于帧解析。如果帧体包含NULL字节，则该帧必须包含一个content-length的header。

#### Header content-type

如果帧存在body，则 SEND、MESSAGE 和 ERROR 帧应该包含一个content-type的header，以帮助帧的接收者解释其**body**。 如果设置了 content-type的header，它的值必须是描述正文格式的 MIME 类型。 否则，接收者应该将主体视为二进制 blob。

以 text/ 开头的 MIME 类型的隐含文本编码是 UTF-8。 如果您使用具有不同编码的基于文本的 MIME 类型，那么您应该将 ;charset=<encoding> 附加到 MIME 类型。 例如，如果您要发送 UTF-16 编码的 HTML 正文，则应该使用 text/html;charset=utf-16。 ;charset=<encoding> 也应该附加到任何可以解释为文本的非文本/MIME 类型。 一个很好的例子是 UTF-8 编码的 XML。 它的内容类型应该设置为 application/xml;charset=utf-8。

所有 STOMP 客户端和服务器必须支持 UTF-8 编码和解码。 因此，为了在异构计算环境中实现最大互操作性，建议使用 UTF-8 对基于文本的内容进行编码。

#### Header receipt

除 CONNECT 之外的任何客户端发送的帧都可以包含任意值的receipt header。 这将导致服务器使用 RECEIPT 帧确认客户端帧的处理（有关更多详细信息，请参阅 [**RECEIPT**](https://stomp.github.io/stomp-specification-1.2.html#RECEIPT)  帧）。

SEND

destination:/queue/a

receipt:message-12345

hello queue a^@

### Repeated Header Entries

由于消息传递系统可以按照存储和转发拓扑进行组织，类似于 SMTP，因此消息在到达消费者之前可能会经过多个消息传递服务器。 STOMP 服务器可以通过在消息中添加header或在消息中就地修改header来“更新”header值。

如果客户端或服务器接收到重复的帧header键值对，则只有第一个键值对应该用作header键的值。 后续值仅用于维护标头状态更改的历史记录，可以忽略。

For example, if the client receives:

MESSAGE

foo:World

foo:Hello

^@

The value of the foo header is just World.

### Size Limits

为了防止恶意客户端利用服务器中的内存分配，服务器可以对以下内容设置最大限制：

1. 单个帧中允许的帧header数
2. Header line的最大长度
3. 帧body的最大长度

如果超过这些限制，服务器应该向客户端发送一个错误帧，然后关闭连接。

### Connection Lingering

STOMP 服务器必须能够支持快速连接和断开连接（翻译为短连接）的客户端。

这意味着服务器可能只允许关闭的连接在连接重置之前逗留一小段时间。

因此，在套接字重置之前，客户端可能无法接收到服务器发送的最后一个帧（例如，错误帧或回复 DISCONNECT 帧的 RECEIPT 帧）。

## Connecting

STOMP 客户端通过发送 CONNECT 帧来启动到服务器的流或 TCP 连接：

CONNECT

accept-version:1.2

host:stomp.github.org

^@

如果服务器接受连接尝试，它将以 CONNECTED 帧响应：

CONNECTED

version:1.2

^@

服务器可以拒绝任何连接尝试。 服务器应该返回一个错误帧，解释连接被拒绝的原因，然后关闭连接。

### CONNECT or STOMP Frame

STOMP 服务器必须以与 CONNECT 帧相同的方式处理 STOMP 帧。 STOMP 1.2 客户端应该继续使用 CONNECT 命令来保持向后兼容 STOMP 1.0 服务器。

使用 STOMP 帧而不是 CONNECT 帧的客户端将只能连接到 STOMP 1.2 服务器（以及一些 STOMP 1.1 服务器），但优点是协议嗅探器/鉴别器将能够区分 STOMP 连接和 HTTP 连接。

STOMP 1.2 clients MUST set the following headers:

1. accept-version ：客户端支持的 STOMP 协议的版本。 有关详细信息，请参阅协议协商（ [**Protocol Negotiation**](https://stomp.github.io/stomp-specification-1.2.html#Protocol_Negotiation)）。
2. host ：客户端希望连接的虚拟主机的名称。 建议客户端将此设置为建立套接字的主机名，或他们选择的任何名称。 如果此标头与已知虚拟主机不匹配，支持虚拟主机的服务器可以选择默认虚拟主机或拒绝连接。

STOMP 1.2 clients MAY set the following headers:

1. login ：用于针对安全 STOMP 服务器进行身份验证的用户标识符。
2. passcode ：用于对安全 STOMP 服务器进行身份验证的密码。
3. heart-beat : The [**Heart-beating**](https://stomp.github.io/stomp-specification-1.2.html#Heart-beating) settings.

### CONNECTED Frame

STOMP 1.2 服务器必须设置以下header：

* version ：会话将使用的 STOMP 协议的版本。 有关详细信息，请参阅协议协商。

STOMP 1.2 服务器可以设置以下标头：

* heart-beat : The [**Heart-beating**](https://stomp.github.io/stomp-specification-1.2.html#Heart-beating) settings.
* session ：唯一标识会话的会话标识符。
* server ：包含有关 STOMP 服务器信息的字段。 该字段必须包含一个服务器名称字段，并且可以后跟由空格八位字节分隔的可选注释字段。

server-name 字段由一个名称标记和一个可选的版本号标记组成。

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

Example:

server:Apache/1.3.9

### Protocol Negotiation

从 STOMP 1.1 起，CONNECT 帧必须包含 accept-version 标头。 它应该设置为客户端支持的递增 STOMP 协议版本的逗号分隔列表。 如果缺少accept-version 标头，则表示客户端仅支持1.0 版本的协议。

将被采用的协议将是客户端和服务器共有的最高协议版本。

例如，如果客户端发送：

CONNECT

accept-version:1.0,1.1,2.0

host:stomp.github.org

^@

服务器将使用它与客户端共有的协议的最高版本进行响应：

CONNECTED

version:1.1

^@

如果客户端和服务器不共享任何共同的协议版本，那么服务器必须以类似于以下的 ERROR 帧进行响应，然后关闭连接：

ERROR

version:1.2,2.1

content-type:text/plain

Supported protocol versions are 1.2 2.1^@

### Heart-beating

可以选择使用心跳来测试底层 TCP 连接的健康状况，并确保远程端处于活动状态并且正在运行。

为了实现心跳，每一方都必须声明它可以做什么以及希望对方做什么。 这发生在 STOMP 会话的最开始，通过向 CONNECT 和 CONNECTED 帧添加心跳header。

使用时，心跳header必须包含两个用逗号分隔的正整数。

第一个数字代表帧的发送者可以做什么（传出的心跳）：

* 0 表示不能发送心跳
* 否则是它可以保证的心跳之间的最小毫秒数

第二个数字代表帧的发送者想要得到什么（传入的心跳）：

* 0 表示它不想接收心跳
* 否则它是心跳之间所需的毫秒数

心跳header是可选的。 丢失的心跳header必须以与“heart-beat:0,0”header相同的方式处理，即：一方不能发送也不想接收心跳。

心跳header提供了足够的信息，以便每一方都可以找出是否可以使用心跳、在哪个方向以及以何种频率使用。

更正式地说，初始帧如下所示：

CONNECT

heart-beat:<cx>,<cy>

CONNECTED

heart-beat:<sx>,<sy>

对于从客户端到服务器的心跳：

* 如果 <cx> 为 0（客户端无法发送心跳）或 <sy> 为 0（服务器不想接收心跳），则不会有心跳，在某一方向上。
* 否则，每隔 MAX(<cx>,<sy>) 毫秒会有一次心跳

在另一个方向上，<sx> 和 <cy> 的使用方式相同。

关于心跳本身，通过网络连接接收到的任何新数据都表明远程端处于活动状态。 在给定方向上，如果预期每 <n> 毫秒会有一次心跳：

* 发送者必须至少每 <n> 毫秒通过网络连接发送新数据
* 如果发送者没有真正的 STOMP 帧要发送，它必须发送一个行尾（EOL）
* 如果在至少 <n> 毫秒的时间窗口内，接收方没有收到任何新数据，它可能认为连接已死
* 由于时间不准确，接收者应该容忍并考虑误差范围

## Client Frames

客户端可以发送不在此列表中的帧，但是对于不在列表中的帧，STOMP 1.2 服务器可以用错误帧响应，然后关闭连接。

* [**SEND**](https://stomp.github.io/stomp-specification-1.2.html#SEND)
* [**SUBSCRIBE**](https://stomp.github.io/stomp-specification-1.2.html#SUBSCRIBE)
* [**UNSUBSCRIBE**](https://stomp.github.io/stomp-specification-1.2.html#UNSUBSCRIBE)
* [**BEGIN**](https://stomp.github.io/stomp-specification-1.2.html#BEGIN)
* [**COMMIT**](https://stomp.github.io/stomp-specification-1.2.html#COMMIT)
* [**ABORT**](https://stomp.github.io/stomp-specification-1.2.html#ABORT)
* [**ACK**](https://stomp.github.io/stomp-specification-1.2.html#ACK)
* [**NACK**](https://stomp.github.io/stomp-specification-1.2.html#NACK)
* [**DISCONNECT**](https://stomp.github.io/stomp-specification-1.2.html#DISCONNECT)

### SEND

SEND 帧发送消息到目的地消息系统。 它有一个必须的 header，destination，它指示将消息发送到哪里。 SEND 帧的主体是要发送的消息。 例如：

SEND

destination:/queue/a

content-type:text/plain

hello queue a

^@

这会将消息发送到名为 /queue/a 的目的地。 请注意，STOMP 将此目的地视为不透明的字符串，并且目的地名称不假定传递语义。 您应该查阅 STOMP 服务器的文档以了解如何构建目的地名称，从而为您提供应用程序所需的交付语义。

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.

SEND supports a transaction header which allows for transactional sends.

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

应用程序可以将任意用户定义的headers添加到 SEND 帧。 用户定义的headers通常用于允许消费者使用 SUBSCRIBE 帧上的选择器根据应用程序定义的标头过滤消息。 用户header的标头**必须**在 MESSAGE 帧中传递。

如果服务器由于任何原因无法成功处理 SEND 帧，则服务器必须向客户端发送 ERROR 帧，然后关闭连接。

### SUBSCRIBE

SUBSCRIBE 帧用于注册以监听给定的目的地。 与 SEND 帧一样，SUBSCRIBE 帧需要一个destination header，指示客户端想要订阅的目的地。 此后，在订阅目的地收到的任何消息都将作为 MESSAGE 帧从服务器传送到客户端。 ack 头控制消息的确认模式。

例子：

SUBSCRIBE

id:0

destination:/queue/foo

ack:client

^@

如果服务器无法成功创建订阅，服务器必须向客户端发送一个错误帧，然后关闭连接。

STOMP 服务器可以支持额外的服务器特定header来自定义订阅的交付语义。 有关详细信息，请参阅服务器的文档。

#### SUBSCRIBE id Header

由于单个连接可以与服务器有多个打开的订阅，因此帧中**必须**包含 id header以唯一标识订阅。 id 标头允许客户端和服务器将后续的 MESSAGE 或 UNSUBSCRIBE 帧与原始订阅相关联。

在同一个连接中，不同的订阅必须使用不同的订阅标识符。

#### SUBSCRIBE ack Header

ack header的有效值为 auto、client 或 client-individual。 如果未设置ack header，则默认为auto。

当 ack 模式为 auto 时，客户端不需要为它收到的消息发送服务器 ACK 帧。 服务器将假定将消息发送给客户端就已收到该消息。 这种确认模式可能会导致发送到客户端的消息被丢弃（丢失）。

当 ack 模式是client时，客户端**必须**为它处理的消息向服务器发送ACK 帧。 如果在客户端为消息发送 ACK 帧之前连接失败，服务器将假定该消息尚未被处理，并且可以将消息重新传递给另一个客户端。 客户端发送的 ACK 帧将被视为累积确认。 这意味着确认对 ACK 帧中指定的消息以及在 此ACK 之前发送到订阅的所有消息都被接受到。

如果客户端没有处理某些消息，它应该发送 NACK 帧来告诉服务器它没有使用这些消息。

当 ack 模式为 client-individual 时，确认消息的操作与 client 模式一样，只是客户端发送的 ACK 或 NACK 帧不是累积的。 这意味着后续消息的 ACK 或 NACK 帧不得导致先前的消息得到确认。

### UNSUBSCRIBE

UNSUBSCRIBE 框架用于删除现有订阅。 删除订阅后，STOMP 连接将不再接收来自该订阅的消息。

由于单个连接可以与服务器有多个打开的订阅，因此帧中必须包含一个 id header以唯一标识要删除的订阅。 此header必须与现有订阅的订阅标识符匹配。

例如：

UNSUBSCRIBE

id:0

^@

### ACK

ACK 被用来确认来自订阅消息被消费，使用client-individual或者client模式。 在通过 ACK 确认消息之前，从此类订阅接收到的任何消息都不会被视为已被消费。

ACK 帧必须包含一个与被确认的 MESSAGE 的 ack 头相匹配的 id 头。 可选地，**可以**指定transaction header，指示消息确认**应该**是事务的一部分。

ACK

id:12345

transaction:tx1

^@

### NACK

NACK 是 ACK 的反义词。 它用于告诉服务器客户端没有消费该消息。 然后，服务器可以将消息发送到不同的客户端，将其丢弃，或将其放入死信队列。 确切的行为是特定于服务器的。

NACK 采用与 ACK 相同的标头：id（必需）和transaction（可选）。

NACK 适用于单个消息（如果订阅的确认模式是client-individual）或之前发送但尚未确认或确认的所有消息（如果订阅的确认模式是client）。

### BEGIN

BEGIN 用于启动事务。 在这种情况下，事务适用于sending and acknowledging模式 - 在事务期间发送或确认的任何消息都将根据事务进行原子处理。

BEGIN

transaction:tx1

^@

transaction header是**必需**的，transaction identifier将用于 SEND、COMMIT、ABORT、ACK 和 NACK 帧，以将它们绑定到相同事务。 在同一个连接中，不同的事务必须使用不同的transaction identifiers。

如果客户端发送 DISCONNECT 帧或 TCP 连接因任何原因失败，则任何尚未提交的已启动事务都将被隐式中止。

### COMMIT

COMMIT 用于提交正在进行的事务。

COMMIT

transaction:tx1

^@

transaction header是必需的，并且必须指定要提交的事务的标识符。

### ABORT

ABORT 用于回滚正在进行的事务。

ABORT

transaction:tx1

^@

transaction header是必需的，并且必须指定要中止的事务的标识符。

### DISCONNECT

客户端可以通过关闭套接字随时断开与服务器的连接，但不能保证服务器已经接收到先前发送的帧。 要进行优雅的关闭，客户端可以确保所有之前的帧都已被服务器接收，客户端应该：

1. 发送带有receipt header set的 DISCONNECT 帧。 例子：

DISCONNECT

receipt:77

^@

2. 等待 RECEIPT 帧对 DISCONNECT 的响应。 例子：

RECEIPT

receipt-id:77

^@

3. close the socket.

请注意，如果服务器关闭其套接字末端的速度太快，客户端可能永远不会收到预期的 RECEIPT 帧。 有关详细信息，请参阅连接延迟部分。（[**Connection Lingering**](https://stomp.github.io/stomp-specification-1.2.html#Connection_Lingering)）

发送 DISCONNECT 帧后，客户端不得再发送任何帧。

## Server Frames