Extensible Messaging and Presence Protocol (XMPP): Core

# 1. 介绍

## 1.1. 概述

可扩展消息发送和到场协议（The Extensible Messaging and Presence Protocol，XMPP）是可扩展标记语言（XML）的一种应用方案，使网络上任何两个或多个网络实体之间能进行近乎实时的结构化及扩展的的数据交互。此文档定义XMPP的核心协议方法：建立和解除XML流，通道加密，身份验证，错误处理，和用于发送消息的通信原语，网络可用性(“到场”),和请求响应交互。

## 1.2. 历史

XMPP的基本语法和语义最初由Jabber开源社区开发，主要在1999年。2002年后期，XMPP工作组被许可开发一个基本Jabber协议的适配版，以适应IEFT的与IMP-REQS一致的即时消息发送和出场技术。2004年10月，[RFC3920]和[RFC3921]被发布，表述了XMPP当时的最完整定义。

自2004年以来，互联网社区已经获得了使用XMPP进行开发和部署的扩充的经验，包括在XMPP标准基金会（XSF）支持下进行的正式的互用性（协同工作能力）测试。这个文档综合了来自软件开发者和XMPP服务提供者的有力反馈，包括在附录D下总结的一些反向兼容的修改。因此，此文档大致反映了互联网社区关于XMPP1.0核心功能的一致意见，并弃用RFC3920。

## 1.3. 功能摘要

XMPP的目的是使网络中任意两个（或多个）实体之间，能交换相对小块的数据结构（称为XML段）。XMPP典型的使用分布式client- server架构实现，客户端需要连接服务器以访问网络，进而能许其他实体（与其他服务器关联的）交换XML数据段。处理过程有，客户端连接服务器，交 换XML数据段，和终止连接：

1. 判定用于连接的IP地址和端口，通常基于对全饰域名的解析（3.2节）

2. 打开一个TCP连接

3. 在TCP上打开一个XML流（4.2节）

4. 通道加密优先使用传输层安全（Transport Layer Security，TLS）协议（5节）

5. 身份验证使用一个简单验证和安全层（SASL）机制（6节）

6. 在流上绑定一个资源（7节）

7. 与网络上其他实体交换无限制的XML数据段（8节）

8. 关闭XML流（4.4节）

9. 关闭TCP连接

使用XMPP，一台服务器能可选的连接到另外一台服务器，以进行域间或服务器间的通信。为此，两台服务器之间需要进行协议连接，然后交换XML数据段；处理过程为：

1. 判定连接的IP地址和端口，通常基于对全饰域名的解析（3.2节）

2. 打开一个TCP连接

3. 在TCP上打开一个XML流（4.2节）

4. 优先使用传输层安全（Transport Layer Security，TLS）协议（5节）

5. 身份验证使用一个简单验证和安全层（SASL）机制（6节）\*

6. 直接在服务器之间和间接在关联到各服务的实体（如连接的客户端）之间，交换无限制的XML数据段（8节）

7. 关闭XML流（4.4节）

8. 关闭TCP连接

\* 互用性说明：在写文档的时候，大部分已部署的服务器仍然使用服务器回拨协议（Server Dialback protocol）[XEP-0220] 来提供弱身份验证，而不是使用SASL和PKIX验证来提供强身份验证，尤其在SASL协议不能提供强身份验证的情形下（例如，因为TLS协议不被端服务器代管，或因为端服务器在TLS协议期间提供的PKIK证书是自分配的并未被提前接受）；详细见[XEP-0220].这个文档描述的解决方案提供了一个显著的更强安全水平（见13.6节）。

这个文档定义了客户端如何链接到服务器，并且定义基本的XML数据端语义。然而，此文档没有定义一旦连接成功建立就可能交换的XML数据段的“装载物”，作为替代，这些装载在各种XMPP扩展中定义了。例如，[XMPP-IM]定义了基本即时消息发送和出场功能的扩展。另外，XSF的XEP系列的各种说明[XEP-001]定义了大量应用扩展。

## 1.4. 术语

…...

route:

pass the data to a remote server for direct processing by the remote server or eventual delivery to a client associated with the remote server

deliver:

pass the data to a connected client

ignore:

discard the data without acting upon it or returning an error to the sender

…...

C: = a client

* E: = any XMPP entity
* I: = an initiating entity
* P: = a peer server
* R: = a receiving entity
* S: = a server
* S1: = server1
* S2: = server2

…...

# 2. 架构

XMPP提供技术用于，在全局可地址化、到场可知的客户端和服务器组成的一个分布式网络中，通过直接、持续的XML流进行异步的，端对端（end- to-end）的结构化数据交换。因为此架构风格包括网络可用性的广泛知识，和理论上给定客户端-服务器，服务器-服务器场景下的无限制数目的并发信息传 输，我们为它打上标签“并发传输可用性”（”Availability for Concurrent Transactions”，ACT），以区分万维网中相似的“可表达的状态传移”（”Representational State Transfer” REST）架构风格。虽然XMPP的架构在一些重要方式上与与email（EMAIL-ARCH）相似，但介绍了若干改动以促进通信到几乎实时。此 ACTive架构风格的显著特征如下。

## 2.1 全局地址

如email，XMPP使用全局唯一的地址（基于域名系统DNS）以在网络上选择路径并分发消息。所有XMPP实体在网络中都是可地址化的，尤其是 客户端和服务器，也包括客户端和服务器可以访问的额外服务。通常，服务器地址形式是<domainpart>（例 如，<im.example.com>）, 服务器上的账号的地址形式是<localpart@domainpart>（例 如，<juliet@im.example.com>,称为裸JID，”bare JID”），而一个账号当前验证通过用于交互的特定连接的设备或资源的地址形式 是<localpart@domainpart/resourcepart>（如，<juliet@im.example.com /balcony，称为全JID，”full JID”）。出于历史原因，XMPP地址通常称为JabberIDs 或 JIDs. 因为XMPP地址的正式定义依赖于写文档时不断变化的国际化技术，其格式在[XMMP-ADDR](RFC 6122)中定义。术语”localpart”, “domainpart”, 和”reourcepart”更正式地定义在[XMMP-ADDR]。

## 2.2 到场（Presence）

XMPP包含这样的功能，用于一个实体向其他实体通告其网络可用或“到场”。在XMPP中，这种通信可用性由端对端信号告知，通过一个专用的通信原 语：<presence/>段。虽然知道网络可用性对XMPP信息交换不是严格必要的，但有利于实时通信，因为消息发起者能在发起通信前知道 目标接收者在线并可达。端对端(end-to-end)到场被定义在[XMPP-IM].

## 2.3 持久流（Persistent Streams）

通过长生命期的TCP连接上的持久XML流，通信的可用性也被构建到每个点对点(point-to-point)”hop”。这些”永远在线” (“always-on”)的客户端到服务器端和服务器端到服务器端流，使各方在任何时候都能向其他方即时推送数据。XML流被定义在第4节下。

## 2.4 结构化数据

XMPP中基本协议的数据单元不是一个XML流（其只是简单地提供传输用于点对点通信）而是一个XML”段”，本质上是在流中发送的一个XML片 段。段的顶级元素包括路由属性（如，”from”和”to”地址），段的孩子节点元素包含用于发送到目标接收者的装载物。XML段定义在第8节。

## 2.5 客户端和服务器的分布式网络

实际上，XMPP包含一个客户端和服务器间相互通信的网络（然而，任何两个给定服务器之间的通信严格的自由并取决于本地服务策略）。因此，例如，服 务器<im.example.com>上的用户<juliet@im.example.com>，能与服务 器<example.net>上的用户<romeo@example.net>交换消息、到场、和其他数据结构。这种模式与使用 全局地址的消息协议相似，如email网络(见[SMTP]和[EMAIL-ARCH]). 因此，XMPP中的端对端(end-to-end)通信逻辑上是点对点(point-to-point)的，但是物理上是client-to- server-to-server-to-client的，如下图说明的。

  example.net <-------------->im.example.com

     ^                                ^

     |                                |

     v                                v

romeo@example.net           juliet@im.example.com

# 3. TCP绑定

## 3.1. 范围

如这份文档定义的XMPP，一个发起实体（客户端或服务器）在与接收实体发起XML流协议前，

必须打开一个到接收实体的TCP链接。然后参与方在使用XML流期间 维护这个TCP链接。下面章节定义了应用到TCP绑定的规则。

说明：XML流与TCP的绑定不是必要，其他传输也是可能的。例如，两个实体可以通过HTTP相互连接，如[XEP-0124]和[XEP-0206]中定义的。但这份文档只定义XMPP到TCP的绑定。

## 3.2. 解析全饰域名

因为XML流在TCP上发送，发起实体在能尝试打开一个XML流之前，需要判定接收实体的IPv4或IPv6地址（和端口）。这通常通过解析接收实体的全饰域名（fully qualified domain name，FQDN，见[DNS-CONCEPTS]）来完成。

#### 3.2.1. 首选处理：SRV查找

FQDN首选处理是使用[DNS-SRV]记录如下：

…...

#### 3.2.1. 备选处理

备选处理是用通常的“A”或“AAAA”地址记录解决方案来判定源域的IPv4或IPv6地址，使用”xmpp-client”端口5222用于client-t—server连接，或”xmpp-server”端口5269用于server-to-server连接（这些是使用IANA注册的默认端口，见14.7节）。

如果TCP连接不成功，发起实体可能尝试查找并使用代替的链接方法，如HTTP连接。

#### 3.2.3. 什么时候不使用SRV

如果发起实体被明确的配置成与接收实体域关联特定的FQDN（硬编码...）。

#### 3.2.4. 使用SRV和附加服务

…...

### 3.3. 重连接

### 3.4. 可靠性

# 4. XML流

## 4.1. 流基础

两个基本概念使得XMPP实体之间相对较小载荷的结构化信息能迅速，异步交换：XML流和XML段。这两个术语在下面定义。

XMLStream的定义：

  一个XML流是一个容器，用于网络上任意两个实体之间交换XML元素。一个XML流的开启明确地由一个”stream header”（即，XML<stream>标签及适当的属性和名字空间）指示，而流的结束明确地由XML关闭标 签</stream>指示。在流的生命周期中，初始化它的实体可以在流上发送无限制数目的XML元素，不管用来流协商的元素（如，完成TSL 协商或SASL协商）或XML段。发起流(”initial stream”)从发起它的实体（通常是一个客户端或服务器）到接收实体（通常是服务器）进行建立，可以看做发起实体”连接到”或”会话到”接收实体。初 始流能进行从发起实体到接收实体单向通信，为了能从接收实体到发起实体能进行数据段交换，接收实体必需发起一个反向的流（响应流）。

XMLStanza的定义：

  一个XML段是XMPP中的基本含义单元。一个段是一个顶层元素（深度为1），元素名是”message”, “presence”, 或”iq”， 并且修饰的命名空间(namespace)是’jabber:client’或 ‘jabber:server’。相比之下，任何其他命名空间修饰的顶层元素都不是XML段（stream errors, stream features, TLS-related elements, SASL-relatedelements, etc.）,由’jabber:client’或’jabber:server’命名空间修饰但不在顶层出现的<message/>, <presence/>, 或<iq/>元素（如报告用扩展元素内的</message>元素）不是，由”jabber:client” 或”jabber:server”之外的命名空间修饰的<message/>,<presence/>或<iq/> 也不是。XML段通常包括一个或多个孩子节点（含属性，元素和XML字符数据）以传递必要的信息，它们可能由XML命名空间修饰（XML-NAMES 8.4节）。

有3种类型的段：message，presence，和IQ（Info/Query）.这些段类型提供3种不同的通讯原语：一个”push”机制用 于生成的消息，一个发布-订阅(”publish-subscribe”)机制用于广播关于网络可用性的信息，和一个请求-响应(“request- response”)机制用于更结构化的数据交换（类似于HTTP）。进一步的解释在8.2.1, 8.2.2和8.2.3节中提供。

考虑一个客户端连接服务器的例子。客户端通过发送一个stream头初始化一个到服务器的XML流，最好前面有一个XML声明指定XML的版本和支 持的字符编码（见11.5节和11.6节）。根据本地策略和服务供应，服务器然后以第二个XML流回应客户端，同样最好前面有一个XML声明。一旦客户端 完成了SASL协商和资源绑定，客户端能在流上发送无限制数目的XML段。当客户端想要关闭流，只要简单的向服务器发送一 个</stream>关闭标签，在4.4节进一步描述了。

实质上，一个XML流作为会话期间发送的XML段的封装，另一个XML流作为会话期间接收的XML段的封装。我们可以一个简单的方式来如下表示：

|  |  |
| --- | --- |
| INITIAL STREAM 初始流 | RESPONSE STREAM 响应流 |
| <stream> |  |
|  | <stream> |
| <presence>    <show/>  </presence> |  |
| <message to=’foo’> <body/> </message> |  |
| <iq to=’chat’ type=’get’> <query/> </iq> |  |
|  | <iq from=’bar’ type=’result’><query/></iq> |
| […] |  |
|  | […] |
| </stream> |  |
|  | </stream> |

两个流的简单视图

那些习惯于以文档为中心的方式来思考XML的人，可能会发现下面的类比很有用：

l  两个XML流就像两个“文档”，由XML段堆积而成。

l  顶端的<stream/>元素就像每个“文档”的“文档实体”。

l  流中发送的XML段就像”文档”的”片段”。

然而，这些仅仅是类比，因为XMPP不是以文档和片段处理，而是以流和段。

这章剩下的部分定义了关于XML流的以下方面（与相关主题一起）：

\* 如何打开一个流（4.2节）

\* 流协商流程（4.3节）

\* 如何关闭一个流（4.4节）

\* XML流的方向性（4.5节）

\* 如何处理沉默端（4.6节）

\* 流的XML属性（4.7节）

\* 流的XML名字空间（4.8节）

\* XML流的错误处理（4.9节）

## 4.2 打开一个流

在连接到接收实体的合适的IP地址和端口后，发起实体通过发送一个流头（initial stream header）到接收实体来打开一个流。

I: <?xml version='1.0'?>

   <stream:stream

       from='juliet@im.example.com'

       to='im.example.com'

       version='1.0'

       xml:lang='en'

       xmlns='jabber:client'

       xmlns:stream='http://etherx.jabber.org/streams'>

接收实体然后通过发送一个它自己的流头（response stream header）到发起实体来回应。

R: <?xml version='1.0'?>

   <stream:stream

       from='im.example.com'

       id='++TR84Sm6A3hnt3Q065SnAbbk3Y='

       to='juliet@im.example.com'

       version='1.0'

       xml:lang='en'

       xmlns='jabber:client'

       xmlns:stream='http://etherx.jabber.org/streams'>

## 4.3 流商议（Stream Negotiation）

### 4.3.1 基本概念

因为流的接收实体对它服务的域的作用像一个看门者，它强加特定的条件，用于连接为客户端或服务器，发起实体在被允许发送数据段到接受实体前，需要与接收实体进行验证（对client-to-server而言意味着使用SASL，在6节描述）。然而，接收实体可以考虑验证之外的条件以强制委托，如用第5节描述的TLS加密。接收实体通过“流特征”(stream features)通信告知发起实体这些条件: 在接收实体从发起实体接受XML段前发起实体需要完成的特定协议交互集合，及任何自愿但可能改进XML流处理的协议交互（如，[XEP-0138]描述的建立应用层压缩）。

The order of layers (TCP, then TLS, then SASL, then XMPP）

### 4.3.2. Stream Features 格式

## 4.4 关闭流

## 4.5 方向性

一个XML流总是单向的，即XML数据段只能在一个流上单向发送（不管从发起实体到接收实体，还是从接收实体到发起实体）。

## 4.6. 处理静默端

## 4.7. 流属性

根元素<stream/>的属性

### 4.7.1 from

定义发送流元素的实体标识。

### 4.7.2. to

发起实体在c-to-s和s-to-st通信中，发起实体都**必须**包含'to'属性，并且**必须**设置它的值为接收实体服务的domainpart.

### 4.7.3 id

“Stream ID” 由接收实体生成，并在接收实体内唯一。

### 4.7.4 xml:lang

首选或默认语言

### 4.7.5. version

### 4.7.6. 流属性摘要

+----------+--------------------------+-------------------------+

| | initiating to receiving | receiving to initiating |

+----------+--------------------------+-------------------------+

| to | JID of receiver | JID of initiator |

| from | JID of initiator | JID of receiver |

| id | ignored | stream identifier |

| xml:lang | default language | default language |

| version | XMPP 1.0+ supported | XMPP 1.0+ supported |

+----------+--------------------------+-------------------------+

## 4.8. XML名字空间

### 4.8.3. XMPP内容名字空间

client-to-server communication for 'jabber:client' (from, to, 是可选的)

server-to-server communication for 'jabber:server' (from, to, 是必须的)

# 8. XML Stanzas （XML段）

客户端和服务器（或两服务器之间）完成流协议后，两方都可以发送XML段。3种XML段被定义了：<message/>,<presence/>和<iq/>. 另外这3中XML段5种共通属性。

## 8.1. 共通属性

### 8.1.1. to

指定目标接收者的JID

<message to='romeo@example.net'>

<body>Art thou not Romeo, and a Montague?</body>

</message>

#### 8.1.1.1 Client-to-Server 流

#### 8.1.1.1 Server-to-Server流

### 8.1.2 from

指定发送方的JID

<message from='juliet@im.example.com/balcony'

to='romeo@example.net'>

<body>Art thou not Romeo, and a Montague?</body>

</message>

### 8.1.3. id

用于发起实体追踪来自其他实体的响应。

### 8.1.4. type

指定message,presence,或IQ段内容的目的。

### 8.1.5. xml:lang

## 8.2. 基本语义

### 8.2.1. Message

Push 机制

### 8.2.1. Presence

“广播”或“发布-订阅”机制

### 8.2.1. IQ

Info/Query, “请求-响应”机制，类似HTTP.