# Jabber 协议 概述

**Peter Saint-Andre**

[**stpeter@jabber.org**](mailto:stpeter@jabber.org)

1.4版Jabber服务协议的概述

## 1. 介绍

    Jabber是一个由开源社区发起并领导开发的即时消息和在线状态的系统。Jabber系统和其它即时消息（IM）服务的一个功能上的差别在于Jabber拥有开放的XML协议。在保持Jabber1.0版本有关消息核心以及在线状态的协议的基础上，增加了一些必要的扩展。本文档将对Jabber服务器1.4版的Jabber协议进行介绍。

    在XML上下文的数据流中，Jabber开放的XML协议包括三个顶级XML元素：

           <message/>

           <presence/>

           <iq/>(info/query)

每一个元素通过属性和名字空间包含大量的数据，这些属性和名字空间都是Jabber的组成部分（但不包括为特殊应用的名字空间应用）。下面将描述整个Jabber协议。如果您想获得关于Jabber系统结构的信息，请查看名为Jabber技术概述的文档（<http://docs.jabber.org/general/html/overview.html>）。

## 2. XML流

    一个Jabber会话由两个平行的XML流组成，一个从客户端到服务器端，另一个从服务端到客户端。当一个Jabber客户端连接上一个Jabber服务器时，这个客户端将发起一个客户端－服务器的XML流，同时作为响应，服务器也将发起一个服务器－客户端的XML流。所有<message/>，<presence/>，以及<iq/>元素都被放在这些XML流的上下文中。下面就有一个例子：

    SEND:<?xml version=”1.0” encoding=”UTF-8” ?>

    SEND:<stream:stream

SEND:to=’jabber.org’

SEND:xmlns=’jabber:client’

SEND:xmlns:stream=’http://ether.jabber.org/streams’>

RECV:<stream:stream

RECV:xmlns:stream=’http://etherx.jabber.org/streams’

RECV:id=’39ABA7D2’

RECV:xmlns=’jabber:client’

RECV:from=’jabber.org’>

(XML for user session goes here)\*

SEND:</stream:stream>

RECV:</stream:stream>

\*注意：必须等到当一个<iq/>元素（特别指一个’jabber:iq:auth’名字空间下的<query/>元素）发送了认证信息后，服务器才会同意用户会话进行工作。

## 3. <message/>元素

    <message/>元素是Jabber开放XML协议三个顶级元素中的一个。它被用来包含两个Jabber用户间互相发送消息内容，或者两个Jabber IDs之间更一般的消息（因为，通过神奇的传输器，另一个IM系统可以用Jabber ID代替，[如70902454@icq.jabber.org](mailto:如70902454@icq.jabber.org)）。

### 3.1. 消息类型－“type”属性

    Jabber支持几种不同的消息，这些消息通过<message/>的“type”属性来进行区分。下面就是“type”属性的有效值：

#### 3.1.1. [default]（不设置“type”属性）

    表示本消息是一个普通消息。默认的情况下，客户端在没有设置属性时，将消息类型定为普通消息。

    例子：

<message to="romeo@montague.net/orchard">

<body>Wherefore art though Romeo?</body>

</message>

#### 3.1.2.type=”chat”

    表示消息需要被一个接一个的显示在聊天界面上（一般是一条接一条的界面，不过实际界面可以被用户自己定义。）

    例子：

    <message to="romeo@montague.net/orchard" type="chat">

<body>Wherefore art though Romeo?</body>

</message>

#### 3.1.3.type=”error”

    表示消息被描述成一个错误条件。实际的错误在消息中用一个<error/>元素来描述。一个<message/>元素包括一个<body/>和一个<error/>。下面的例子将演示，当Juliet对一个不存在的用户发送一个“hi”的消息时，将收到的包体：

    例子：

<message

to="juliet@capulet.com"

from="flomeo@montague.net"

type="error">

<body>hi</body>

<error code="404">Not Found</error>

</message>

#### 3.1.4. type=”groupchat”

    表示本消息需要在一个聊天室的界面中进行显示。

    例子：

    <message to="cauldron@conference.witches.org" type="groupchat">

<body>Double, double toil and trouble</body>

</message>

### 3.2. <message/>其它属性

    除了设置“type”属性，还可以通过下面的属性修改<message/>元素：

#### 3.2.1. “from”属性

    消息发送者的标志。总得来说，Jabber ID的”from”必须是user@host/resource的格式。从XML的观点来说，这个属性必须包含于所有的<message/>元素中，为了对付一些欺骗，实际应用中服务器添加或置换这个属性。因此，它并不是客户端开发者需要关心的问题。

    例子：

    <message from="juliet@capulet.com/balcony">

<body>Wherefore art thou Romeo?</body>

</message>

#### 3.2.2. “id”属性

    “id”属性是作为消息的一个唯一标志。它由Jabber客户端或客户库（类似WinJab或JabberCOM），生成，客户端用它来为跟踪消息（比如将发送到一个群组聊天室的信息和从这个聊天室收到的消息相关联，或将一个消息与它可能产生的错误相关联）进行标识。“id”属性是可选的，并且在系统的别的方是不可用的。

    例子：

    <message to="romeo@montague.net/orchard" id="JCOM\_85">

<body>Wherefore art thou Romeo?</body>

</message>

#### 3.2.3. “to”属性

    表示消息的接收者。Jabber ID的“to”属性一般是user@host/resource格式，user@host也可以被接受。如果接收者在线，通过user@host/resource方式解决，如果用户不在线，直接发送离线存储器。这个属性是所有<message/>元素必须的。

    例子：

<message to="romeo@montague.net/orchard">

<body>Wherefore art thou Romeo?</body>

</message>

### 3.3. <message/>中的子元素

    下面的子元素是用来定义一个Jabber消息的上下文或数据元。

#### 3.3.1.  <body/>

    这个元素包含消息的主要文本。<body/>元素只存在于一个<message/>元素中，并且只能包含明文。不过，它可以由其它子元素构成（如<error/>、<x/>）。

#### 3.3.2. <error/>

    当消息类型属性被置为“error”时，这个元素将被使用。错误代码被一个”type”属性定义，这个”type”属性包含代码数，通过<error/>元素来包含一个这个错误的文本描述。<error/>元素必须包含在原始消息的<body/>中。下面的例子将演示，当Juliet向一个不存在的用户发送“hi”消息后的包体：

    例子：

<message

to="juliet@capulet.com"

from="flomeo@montague.net"

type="error">

<body>hi</body>

<error code="404">Not Found</error>

</message>

    附文中有非法代码与它们所定义的含义表。

#### 3.3.3. <html/>

    <html/>元素结合XHTML名字空间（http://www.w3.org/1999/xhtml），为一个消息提供一个交替的名字空间，就像在email世界中规定同时发送普通文本与HTML版本的邮件消息一样。这个功能使用的是W3C开发的基本XHTML格式（http://www.w3.org/TR/xhtml-basic/），关于Jabber在消息中的XHTML执行细节，请看Jabber消息的暂定格式规范（http://docs.jabber.org/draft-proto/html/xhtml.html）。所有XHTML的根元素都必须是<html xmlns=”http://www.w3.org/1999/xhmtl”></html>。这个根元素中，支持任何一个基本XHTML格式中定义的元素，但不是每一个元素可以通过即时消息的上下文来进行理解。

#### 3.3.4. <subject/>

    这是一个可选元素，包含消息的标题。

#### 3.3.5. <thread/>

    当一个Jabber客户端发送一个”chat”类型的消息，将生成唯一的线程ID（通常是由发送者的Jabber ID和当前时间生成的一个hash）。消息发送者和接收者在聊天中后续的消息都包含一个<thread/>子元素，这个子元素包含原始线程ID。这使得消息发送者和接收者对回复进行认证并创建一个对话线程。下面的是例子：

<message

from="juliet@capulet.com/balcony"

to="romeo@montague.net/orchard"

type="chat">

<thread>AAC96FEFDE226C2C04BE8F2E19A2151B</thread>

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

</message>

<message

from="romeo@montague.net/orchard"

to="juliet@capulet.com/balcony"

type="chat">

<thread>AAC96FEFDE226C2C04BE8F2E19A2151B</thread>

<body>Neither, fair saint, if either thee dislike.</body>

</message>

#### 3.3.6. <x/>

    <x/>元素作为一个扩展机制来使用，或发送从服务器发送命令给客户端，或从一个客户端发送命令到另一个客户端。每次用到这个元素时，必须指明相关的XML名字空间。单一消息可以包含多个<x/>子元素实例。下列已定义的名字空间可以被<x/>子元素使用：

           jabber:x:autoupdate

           jabber:x:delay

           jabber:x:encrypted

           jabber:x:oob

           jabber:x:roster

这些名字空间在下面的Jabber名字空间中被更完整地描述。

下面是一个使用jabber:x:delay名字空间在加入的群组聊天室的显示上下文的例子：

<message

type=’groupchat’

from=’cauldron@conference.witches.org’>

<body>Thrice the brinded cat hath mew’d.</body>

<x xmlns=’jabber:x:delay’

stamp=’10541031T21:43:40’>Cached In GC History</x>

</message>

## 4. <presence/>元素

    <presence/>元素提供关于一个Jabber实体的可用性信息。任何一个通过一个Jabber ID确认的实体可以与另一个实体进行在线状态信息的通信，这种通信大多以订阅在线状态信息的方式进行。

    所有实体表现出的在线状态不是“可用”就是“不可用”。“可用”状态表示发送者可以立即收到消息。“不可用”状态表示发送者不能在当前时间收到任何数据。默认情况下，所有<presence/>元素除非包含type=”unavailable”属性外，都表示“可用”。“可用”的更多特殊形势通过<status/>和<show/>子元素进行指定。

### 4.1. “type”属性

    <presence/>的“type”属性根据不同目的使用。除了提示其它用户本用户的可用性状态的“普通”用法外，还包括订阅、取消订阅、以及探测在线状态信息。下面是“type”属性的可用值：

#### 4.1.1. type=”available”

    如果没有包含任何“type”属性，在线状态包将被设为type=”available”，用来提供用户在线时的信息。一个type=”available”的在线状态包通常包含一个<show/>元素以详细说明Jabber实体的可用类型，以及一个<status/>元素，该元素包含人能看懂的关于该可用类型的描述。在线状态包还可以包含一个<priority/>元素，来指明该连接的Jabber ID的优先级。

    例子：

    <presence

from="juliet@capulet.com/balcony"

to="romeo@montague.net/orchard"/>

<show>away</show>

<status>Stay but a little, I will come again.</status>

</presence>

    <show/>元素的可用值为：

           away――Jabber用户或实体在线，但不能马上联系上

           chat――Jabber用户或实体在线并有兴趣聊天

           dnd――Jabber用户或实体在线，但不想被打扰（“dnd”表示“do not disturb”）

           xa――Jabber用户或实体在线，但已经离开很长时间了（“xa”表示“extended away”）

<status/>可以包含任意特性数据，这些数据通常包含对于用户状态的描述，如，“开会中”是“away”的一个表现值，或者“忙于编码”可以是“dnd”的一个表现。

#### 4.1.2. type=”error”

    当一个在线状态包发送给一个不存在的JID时，或在发送在线状态请求时发生一个错误时，服务器都将返回一个type=”error”的在线状态包。

    下面是一个例子（注意域名的类型）：

<presence

       type="subscribe"

to="romeo@montague.nett"/>

    回复的例子：

    <presence

       from="romeo@montague.nett"

to="juliet@capulet.com/balcony"

type="error">

<error code=’504’>Remote server timeout. Unable to deliver

packet.</error>

</presence>

#### 4.1.3. type=”probe”

    向一个特定实体（实体不能发送在线状态消息给自己）发出在线状态信息请求。服务器而非客户端，进行一个探测请求。实际上，虽然Jabber客户端也可以发送一个探测一个指定Jabber用户是否在线的请求，但通常都是服务器端发出探测请求。注意，只有在发起请求的用户在请求所对应的用户的花名册上，并且拥有“from”或“both”订阅类型时，服务器才会发出探测请求。在下面的例子中，我们将看到我们的英雄Romeo向Juliet发出一个探测请求，Juliet所在服务器回复一个“状态报告包”（包含jabber:x:delay名字空间中指明的上次在线状态更新的时间戳）

    探测请求的例子：

    <presence

from="romeo@montague.net"

to="juliet@capulet.com"

type="probe"/>

    回复探测请求的例子：

    <presence

from="juliet@capulet.com/."

to="romeo@montague.net">

<status>Stay but a little, I will come again.</status>

<show>away</show>

      <x

xmlns="jabber:x:delay"

from="juliet@capulet.com/."

stamp="15310309T23:47:15"/>

</presence>

#### 4.1.4. type=”subscribe”

    发送一个请求，这个请求是接收者在其在线状态改变时，自动发送在线状态信息给发送者。

    例子：

    <presence

from="romeo@montague.net"

to="juliet@capulet.com"

type="subscribe"/>

#### 4.1.5. type=”subscribed”

    发送一个消息，表明发送者接受了接受者对其再现状态订阅的请求。从现在起，当发送者的在线状态信息改变时，服务器将会把状态信息发送给接收者。

    例子：

    <presence

from="juliet@capulet.com"

to="romeo@montague.net"

type="subscribed"/>

#### 4.1.6. type=”unavailable”

    例子：

    <presence

from="juliet@capulet.com"

to="romeo@montague.net"

type="unavailable">

<status>Disconnnected</status>

</presence>

#### 4.1.7. type=”unsubscribe”

    发送一个请求，表明接收者停止发送在线状态信息给发送者。

    例子：

<presence

from="juliet@capulet.com"

to="romeo@montague.net"

type="unsubscribe"/>

#### 4.1.8. type=”unsubscribed”

    这种类型的在线状态包有两个用途：

1.    发送一个表明发送者接受接收者关于不订阅其在线状态信息的请求的通知。服务器将不再发送发送者的在线状态信息给接收者。

2.    拒绝一个订阅请求。服务器将不再发送发送者的在线状态信息给试图订阅其在线状态信息的用户。

例子：

<presence

from="romeo@montague.net"

to="juliet@capulet.com"

type="unsubscribed"/>

### 4.2. 其它<presence/>属性

    除了设置“type”属性，还可以通过下列属性修改<presence/>元素：

#### 4.2.1. “from”属性

标识在线状态包的发送者。一般Jabber ID再“from”属性中必须写成user@host/resource的格式。从XML的观点来说，这个属性是所有<presence/>元素所必须的，而在实际应用中，都是服务器对该属性进行添加和修改（防止一些欺骗的手段），因此客户端开发者不需要考虑它。

例子：

<presence from="juliet@capulet.com/balcony"/>

#### 4.2.2. “id”属性

    为在线状态请求包配置唯一的验证符。“id”属性由Jabber客户端或客户库（如Gabber或JabberCOM）生成，客户端用其为在线状态包的轨迹进行验证。“id”属性是可选的，并且不能用于系统其它任何地方。

    例子：

    <presence to="romeo@montague.net/orchard" id="JCOM\_86"/>

#### 4.2.3. “to”属性

    标识在线状态包的接收者。一般Jabber ID在“to”属性中都是user@host/resource的格式，user@host的格式也是可以的，但当接收者在线时，它将转化为user@host/resource。在Jabber的一般用法中，Jabber客户端向服务器发出在线状态包，服务器将这些在线状态包发送到该用户花名册上所有的用户。当一个Jabber客户端发送的<presence/>元素中不含“to”属性时，服务器将将其转化为在线状态包发送给指定的Jabber实体。下面的例子将演示这一过程。

    发送给服务器的在线状态的例子：

    <presence>

<status>Stay but a little, I will come again.</status>

<show>away</show>

</presence>

服务器发给指定用户的在线状态例子：

<presence

to="romeo@montague.net/orchard"

from="juliet@capulet.com/balcony">

<status>Stay but a little, I will come again.</status>

<show>away</show>

</presence>

### 4.3. <presence/>的子元素

    下面的元素用于<presence/>协议。

#### 4.3.1. <priority/>

    在Jabber中，根据Jabber知道的设备数或位置数，一个用户可以建立相同数量的连接。用户可以给每一个资源赋予一个数字优先级（通过对指派给该资源的Jabber客户端进行设置），这个优先级与<presence/>元素的子元素<priority/>方法进行通信。

    当在线状态包发送到服务器后，服务器有能力决定哪一个连接资源需要指派一个单一的Jabber ID，该资源将收到发往该Jabber ID的消息，这是因为数字最高的资源是默认或第一资源。如果最高优先级的资源不可用，消息和其它Jabber通信将发送给次高优先级的资源（当其它资源都不可用时，一个没有从默认数到0的优先级的客户端，也可以成为第一资源）

    举例来说，Julie可以通过两个资源：the balcony 和 the chamber连接她的帐号juliet@capulet.com。如果“balcony”资源的优先级被设置为“2”，而“chamber”资源的优先级被设为“1”，而且两个资源都可以连接，消息将发送到juliet@capulet.com/balcony，而不是juliet@capulet.com/chamber。

#### 4.3.2. <show/>

    可选的<show/>元素告诉Jabber客户端如何显示一个用户的在线状态。<show>标签是一个典型的<status>标签的伴随标签，<status>标签包含更多在线状态改变理由的描述信息。如果<show/>元素不在<presence/>元素中，Jabber接收客户端将指明该用户处于“普通”状态。下面是四个可选项：

|  |  |
| --- | --- |
| 标签 | 含义 |
| <show>chat</show> | 该客户可以马上联系上 |
| <show>away</show> | 该客户在线，但刚刚离开（如吃午饭，开会） |
| <show>xa</show> | 该客户在线，但已经处于非活动状态很长时间了。 |
| <show>dnd<show> | 该用户处于谢绝打扰的模式 |

#### 4.3.3. <status/>

    <status/>元素包含更多关于用户在线状态的描述。大多数Jabber客户端都包含一些<status/>元素的默认设置；另外，它们还允许我们提供富有个性的描述如“我在吃午饭”或者“钓鱼中”。

#### 4.3.4. <x/>

<x/>元素作为一个扩展机制来使用，或发送从服务器发送命令给客户端，或从一个客户端发送命令到另一个客户端。每次用到这个元素时，必须指明相关的XML名字空间。单一消息可以包含多个<x/>子元素实例。下列已定义的名字空间可以被<x/>子元素使用：

           jabber:x:autoupdate

           jabber:x:delay

           jabber:x:encrypted

           jabber:x:oob

           jabber:x:roster

这些名字空间在下面的Jabber名字空间中被更完整地描述。

## 5. <iq/>元素

    信息/查询(IQ)在Jabber中在两个实体间构建一个根本的会话，并且允许实体间来回传送XML格式的请求和响应。信息/查询主要的用处是取得或设置公共的用户信息，比如名字，电子邮件，地址等等。但它的灵活设计使得任何种类的会话都可以发生。任何通过一个Jabber ID标识的实体都能通过一个IQ与其它实体进行会话。

### 5.1. “type”属性

    <iq/>元素的“type”属性用于决定信息/查询是请求还是响应。下面是“type”属性的可用值：

#### 5.1.1. type=”error”

    表示查询失败。实际错误在<iq/>元素的<error/>子元素中描述。

    例子：

    <iq type="error" to="romeo@montague.net">

<error code="403">Forbidden</error>

</iq>

#### 5.1.2. type=”get”

    找回指派给一个查询名字空间的信息。如果没有设置属性，默认情况，这个属性包含在查询中。一个type=”get”的<iq/>元素由一个子元素构成，这个子元素通常是<query>，但也可以是<vcard/>或另一个已有的子元素。不过，指定的子元素必须左空，以便接收一个用户所需提供的信息的空间的清单。一个成功的get查询的结果type=”result”的<iq/>元素，是元素嵌套在包含了所需信息的指定子元素（如<query>或<vcard/>）。

    例子：

    <iq type="get">

<vCard xmlns="vcard-temp"/>

    </iq>

#### 5.1.3. type=”result”

    表示<iq/>包是一个type=”get”或type=”set”的<iq/>查询的成功响应。这个成功查询的结果是一个type=”result”的<iq/>元素，该元素嵌套在一个包含所查询的信息的子元素中（如<query>、<vcard/>）。一个成功的查询或获取查询结果的返回是一个空的type=”result”的<iq/>元素。一个不成功的返回是一个type=”error”的<iq/>元素。<iq type=”result”/>通常与“id”属性的获取、设定查询有关。

例子：

    <iq type="result">

<vCard version="3.0" xmlns="vcard-temp">

*vCard data goes here*

</vCard>

</iq>

#### 5.1.4. type=”set”

    表示<iq/>包是对设值或更改现有数据值。一个type=”set”的<iq/>元素总是包含一个指定的子元素，通常是<query>，也可以是<vcard/>或者其它可以接受的子元素。一个设置命令查询的成功返回是一个空的type=”result”的<iq/>元素。

    例子：

    <iq type="set">

<vCard version="3.0">

*vCard data goes here*

</vCard>

</iq>

### 5.2. 其它<iq/>属性

    除了“type”属性外，IQ元素还可以通过下面的属性进行修改。

#### 5.2.1. “from”属性

    表示iq包的发送者。从XML观点来说，这个属性是所有<iq/>元素所必须的，实际应用中这个属性是服务器来增加和修改的（为防止某些欺骗），因此客户端的开发人员不需要过多关注这个属性。

    例子：

    <iq from="romeo@montague.net"/>

#### 5.2.2. “id”属性

    一个iq包只有唯一的标识。Jabber客户端或客户库生成“id”属性，客户端通过它来标识<iq/>包，以完成对<iq/>包的跟踪（如：一个type=”result”对应一个type=”get”或type=”set”）。“id”属性是可选的，并且不能用于系统的其它地方。

    例子：

    <iq type="get" from="juliet@capulet.com/balcony" id="JCOM\_87"/>

<iq type="result" to="juliet@capulet.com/balcony" id="JCOM\_87"/>

#### 5.2.3 “to”属性

    表示iq包的接收者。在Jabber客户端，对应“to”属性的Jabber ID的格式为user@host/resource。user@host也是可以的，如果用户在线，user@host将被置换成user@host/resource，如果用户不在线，user@host将被直接指向一个离线模块（对于类似服务器端网关这种非用户实体，情况又有所不同，因为它们不拥有资源，并且只是简单通过Jabber ID的一个“server”端口进行标识。Jud.jabber.org——Jabber用户目录就是这样一个例子）。如果<iq/>包没有<to>属性，服务器将对该包进行处理。

    例子：

    <iq to="juliet@capulet.com/balcony">

### 5.3 <iq/>的子元素

    下面介绍<iq/>元素的子元素。

#### 5.3.1. <error/>

    当iq的属性类型被设置为“error”时，将用到本属性。错误代码根据一个含有错误数的“type”属性进行定义，<error/>员司包含该错误的文本描述。比如，下面就是一个坏请求：

    <error code="400">Bad Request</error>

    在附录中，列有错误编码及其对应的错误描述。

#### 5.3.2. <key/>

    <key/>元素为客户端－服务器之间交互提供一层安全保护。它被用户jabber:iq:register和jabber:iq:search名字空间中。

    当一个客户端发起一个与服务其之间的交互时，服务器将发送一个包含一个唯一值的<key>给客户端。客户端在返回<iq type=”set”/>的消息中，包含上诉唯一值到<key/>子元素中。这样，服务器就认为该客户端时接收到原来密码的同一实体。

#### 5.3.3. <query/>

    <iq/>元素中的<query/>子元素定义所执行的查询的类型。特殊情况下，查询可以拥有一个特殊的名字空间，这个名字空间是一个通过“xmlns”属性定义的<query/>子元素。一个<iq/>元素中只能有一个<query/>子元素。

    下面的例子是当我通过我的Linux机器上的Grabber客户端登陆到服务器上，将出现的XML。

<iq type="set">

<query xmlns="jabber:iq:auth">

<username>stpeter</username>

<resource>Gabber</resource>

<digest>f1e881517e9917bb815fed112d81d32b4e4b3aed</digest>

</query>

</iq>

就象你看到的那样，认证查询通过客户端认证名字空间（jabber:iq:auth）向服务器发送认证信息。一般，Jabber核心协议保留jabber:iq:\*这样的名字空间。而开发者可以通过增加jabber:iq:\*名字空间对Jabber核心进行扩展。这些将在下面的名字空间中进行更完整的描述。

一个空<query/>子元素可以与<iq type=”get”>元素一起发送，这样可以找回与<query/>子元素中指定的名字空间相关的一个Jabber实体的信息。比如，找回一个客户端的花名册（“联系人名单”），下面的XML可以被发送：

<iq to="juliet@capulet.com/balcony" type="get">

<query xmlns="jabber:iq:roster"/>

</iq>

## 6. 名字空间

    现面是Jabber中的标准名字空间：

           jabber:iq:agent——代理工具

           jabber:iq:agents——有效的代理组

           jabber:iq:auth——简单用户认证

           jabber:iq:autoupdate——版本更新申请

           jabber:iq:browse——Jabber浏览

           jabber:iq:conference——一般会议

           jabber:iq:gateway——用户地址网关

           jabber:iq:last——上次使用时间

           jabber:iq:oob——绑定数据输出

           jabber:iq:private——私有数据存储

           jabber:iq:register——注册请求

           jabber:iq:roster——花名册（联系人名单）管理

           jabber:iq:search——用户数据库查询

           jabber:iq:time——客户端时间

           jabber:iq:version——客户端版本

           jabber:x:autoupdate——版本通知申请

           jabber:x:conference——会议邀请

           jabber:x:delay——显示的对象

           jabber:x:encrypted——加密的消息

           jabber:x:envelope——消息封套

           jabber:x:event——消息事件

           jabber:x:expire——消息终止

           jabber:x:oob——绑定数据（文件传输）输出

           jabber:x:roster——内部的花名册条目

           jabber:x:signed——标记的在线状态

           vcard-temp——临时的vCard

下面将详细介绍这些名字空间的细节。

### 6.1. jabber:iq:agent——代理工具

    代理工具名字空间包含一个代理的所有工具。jabber:iq:angents查询通常用于注册到一个指定的服务、代理、传输器。

    它也可能检验一个指定代理的所有工具细节。比如，一个客户端可以决定一个开放的注册是否可以进行。

    例子：

    <iq id="wjAgent" type="result" from="Jabber.org">

<query xmlns="jabber:iq:agent">

<agent jid="aim.jabber.org">

<name>AIM Transport</name>

<description>This is the AIM Transport</description>

<transport>AIM/AOL ScreenName</transport>

<service>aim</service>

<register/>

</agent>

</query>

</iq>

### 6.2. jabber:iq:angents——可用的代理列表

    可用代理列表名字空间包含一组实体，这些实体拥有一些特殊工具，并能够为另一个实体提供服务。大多数情况下，可用代理列表是用来显示一台服务器上的可用传输器列表。

    注意：这个名字空间被jabber:iq:browse的支持者所反对（？？）。

    例子：

    <iq id="wjAgents" type="result" from="jabber.org">

<query xmlns="jabber:iq:agents">

<agent jid="users.jabber.org">

<name>User Directory</name>

<description>

You may register and create a public searchable

profile, and search for other registered Jabber

users.

</description>

<service>jud</service>

<register/>

<search/>

</agent>

<agent jid="aim.jabber.org">

<name>AIM Transport</name>

<description>This is the AIM Transport</description>

<transport>AIM/AOL ScreenName</transport>

<service>aim</service>

<register/>

</agent>

</query>

</iq>

### 6.3. jabber:iq:auth——简单客户端认证

简单客户认证名字空间是对客户端进行认证的一个简单机制，它为客户端到服务器之间的连接创建一个资源。

认证成功返回一个type=”result”的IQ响应。不成功，就返回一个错误IQ元素。

如果没有发送名字空间或密码，如果服务器允许，服务器将创建一个匿名资源。

例子：

<iq type="set" id="JCOM\_0">

<query xmlns="jabber:iq:auth">

<username>juliet</username>

<password>secret</password>

<resource>balcony</resource>

</query>

</iq>

<iq type="set" id="JCOM\_0">

<query xmlns="jabber:iq:auth">

<username>juliet</username>

<digest>a5e052c48c455bf21d937ccf17225916d9486b09</digest>

<resource>balcony</resource>

</query>

</iq>

<iq type="set" id="JCOM\_0">

<query xmlns="jabber:iq:auth">

<username>juliet</username>

<hash499>secret</hash499>

<resource>balcony</resource>

</query>

</iq>

### 6.4. jabber:iq:autoupdate——版本更新申请

    版本更新申请名字空间允许客户端对所有可用软件更新进行申请请求。

    例子：

<iq type="get" to="jabber.org" id="1">

<query xmlns="jabber:iq:autoupdate"/>

</iq>

### 6.5. jabber:iq:browse——Jabber浏览

    Jabber浏览名字空间的作用是，一是为Jabber世界中已有的Jabber IDs增加结构的一个途径，一是发现为不同Jabber实体提供的服务或应用。一个Jabber浏览的基本概念就是一个jid-type（类似文件的一个模拟类型的概念）。下面是想象中的最高级别的jid-types，不过随着时间的推移，一些其它的逻辑将被增加进来：

           应用

           会议

           标题

           关键字

           回报

           服务

           用户

上面每一个范畴都有一个子类列表，如用户/客户端，或用户/设备（“客户端”或“设备”只是一个“资源”的概念），会议/irc，服务/icq，标题/库存。一般的，通过使用jabber:iq:browse名字空间你将拥有一个与上述某一jid-type范畴相对应的元素，这个元素通常有一个指定子类型的“type”属性。

Jabber:iq:browse的一个用户就是代替jabber:iq:agents名字空间来列出一个服务器上所对应的代理或其它服务。下面是一个浏览服务器的例子：

Jabber客户端发送：

<iq type="get" to="jabber.org">

<query xmlns="jabber:iq:browse"/>

</iq>

Jabber服务器返回：

<iq type="result" from="jabber.org">

<service

type="jabber"

jid="jabber.org"

xmlns="jabber:iq:browse>

<service

type="jud"

jid="users.jabber.org"

name="Jabber User Directory">

<ns>jabber:iq:register</ns>

<ns>jabber:iq:search</ns>

</service>

<service

type="aim"

jid="aim.jabber.org"

name="AIM Transport">

<ns>jabber:iq:register</ns>

<ns>jabber:iq:search</ns>

</service>

<conference

type="private"

jid="conference.jabber.org"

name="Private Chatrooms">

</service>

</service>

</iq>

在会议中的一个例子：

<iq type="set" to="room@server">

<conference xmlns="jabber:iq:browse" name="Room Name">

<user jid="room@server/jidhash" name="nick1"/>

<user jid="room@server/jidhash" name="nick2"/>

</conference>

</iq>

### 6.6. jabber:iq:conference——一般会议

    一般会议名字空间提供一个会议室的机制（如群组聊天室），并提供该会议室的入口。这个名字空间与</iq>元素配套使用来获得一个会议室的信息，包括要求加入这个会议室的请求信息，以及为这个会议室设置信息（通常是为了加入这个房间）。一个普通进入房间的请求将和下面的类似：

1.    想要加入的房间提交一个type=”get”的请求。这个XML将是下面的形式：

<iq type="get" to="room@server">

<query xmlns="jabber:iq:conference"/>

</iq>

2.    接受来自要加入的房间的type=”result”的iq响应，其中有进入房间所需要的参数。这个XML将是下面的形式：

<iq type="result" from="room@server">

<query xmlns="jabber:iq:conference">

<nick/>

<secret/>

<privacy/>

<name>Room Name</name>

</query>

</iq>

这里所有的请求元素的子元素都是可选的。昵称(nickname)元素表示进入房间需要一个昵称。密钥(secret)元素表示进入房间的用户需要提供一个密码。独处(privacy)元素表示如果用户在一个type=”set”（下面可以看到例子）的iq包中提供这样的元素，会议服务器将隐藏用户的真实Jabbere ID。名字（name）元素是房间的名字（用jabber：iq:browse浏览房间时可以看到同样的房间名）。

3.    发送当前状态给要加入的房间。XML如下：

<presence to="room@server"/>

4.    想要加入的房间提交一个type=”set”的iq包以及进入所需信息。XML如下：

<iq type="set" to="room@server">

<query xmlns="jabber:iq:conference">

<nick>mynick</nick>

<nick>mysecondnick</nick>

<secret>thesecret</secret>

</query>

</iq>

注意在用户第一个选择被接受时，用户可以提交多个昵称。如果用户没有提供昵称，服务器将指派一个昵称给用户。

5.    收到你加入的房间的通知，这个iq以下面的格式返回：

<iq type="result" from="room@server">

<query xmlns="jabber:iq:conference">

<id>room@server/uniqueidnumber</id>

<nick>mynick</nick>

</query>

</iq>

这个唯一的ID号是用户Jabber ID的一个哈希。

### 6.7. jabber:iq:gateway——用户地址网关

    一般通过一个网关或传输器（比如AIM传输器）来实现与非Jabber网络用户的通信，每一个非Jabber网络都可能拥有唯一的用户地址，这个地址可能和Jabber IDs并不统一。用户地址网关就是解决这个问题的，它使得Jabber客户端向非Jabber网络用户发出订阅时给出正确Jabber IDs变得很容易。这些通过对用户进行提示和会话，引导用户提供创建一个正确Jabber ID所需信息来实现。下面将演示这种普通的交互：

1.    客户端向网关发出带有jabber:iq:gate名字空间查询的iq get：

<iq type="get" to="aim.jabber.org">

<query xmlns="jabber:iq:gateway"/>

</iq>

2.    服务器返回一个包含请求批准及其描述文本的iq：

<iq type="result" from="aim.jabber.org">

<query xmlns="jabber:iq:gateway">

<desc>

Please enter the AOL Screen Name of the person

you would like to contact.

</desc>

<prompt>Screen Name</prompt>

</query>

</iq>

3.    客户端向网关发送一个包含指定信息的iq set：

<iq type="set" to="aim.jabber.org">

<query xmlns="jabber:iq:gateway">

<prompt>Joe Cool</prompt>

</query>

</iq>

4.    服务器返回一个包含正确Jabber ID的iq result

<iq type="result" from="aim.jabber.org">

<query xmlns="jabber:iq:gateway">

<jid>JoeCool@aim.jabber.org</jid>

</query>

</iq>

### 6.8. jabber:iq:last——最后一次在线时间

最后一次在线时间名字空间提供一个标准的方法，通过这个方法可以查询一个Jabber实体一个或多个服务更新时间，或者一个客户最后一次连接（或活动）到一台服务器上的时间。每一次返回的值都是一个秒的数目。进行查询的实体可以通过下面的法则解释获取的信息：

           user@server/resource——用户最后一次活动的时间

           user@server——用户最后一次连接的时间

           server——服务器最后一次启动的时间

例子（服务器）：

<iq type="get" to="jabber.org">

<query xmlns="jabber:iq:last"/>

</iq>

<iq type="result" to="stpeter@jabber.org/Gabber" from="jabber.org">

<query xmlns="jabber:iq:last" seconds="124933"/>

</iq>

例子（客户端）：

<iq type="get" to="jer@jabber.org">

<query xmlns="jabber:iq:last"/>

</iq>

<iq type="result" to=stpeter@jabber.org/Gabber from="jer@jabber.org">

<query xmlns="jabber:iq:last" seconds="5024">

Disconnected

</query>

</iq>

### 6.9. jabber:iq:oob——绑定数据输出

    绑定数据输出名字空间为客户端－客户端的文件传输提供一个标准方法，通过服务器的传输/代理的传输器实现一个特殊的名字空间：

例子：

<iq type="set" to="horatio@denmark.com" id="file\_1">

<query xmlns="jabber:iq:oob">

<url>http://denmark.com/act4/letter-1.html</url>

<desc>There’s a letter for you sir.</desc>

</query>

</iq>

### 6.10. jabber:iq:private——私有数据存储

    私有数据存储名字空间是Jabber服务器上存储私有数据的一个机制。存储的数据可以是任何合法的XML数据。在服务端存放客户端的当前状态信息是这个名字空间的典型用法。

    例子1（客户端存储私有数据）：

<iq type="set" id="1001">

<query xmlns="jabber:iq:private">

<winjab xmlns="winjab:prefs">

<defaultnick>Hamlet</defaultnick>

</winjab>

</query>

</iq>

例子2（客户端找回私有数据）：

<iq type="get" id="1002">

<query xmlns="jabber:iq:private">

<winjab xmlns="winjab:prefs"/>

</query>

</iq>

另外，对于私有数据存储，1.4版服务器支持服务器上所有有效名字空间（以“jabber”开头的名字空间保留，为Jabber系统进行调用）的XML数据。这个数据存在向服务器端发送type=”set”的iq包的用户的花名册中。

例子1（客户端保存公有数据）：

<iq type="set" id="1003">

<query xmlns="stpeter:public:favorites">

<fav\_things>

<food>Thai</food>

<color>blue</color>

<composer>Bach</composer>

</fav\_things>

</query>

</iq>

例子2（客户端找回公有数据）：

<iq type="get" id="1004">

<query xmlns="stpeter:public:favorites">

<fav\_things/>

</query>

</iq>

### 6.11. jabber:iq:register——注册请求

注册请求名字空间对一个或多个服务进行注册。它也被用来更新或删除一个注册。

例子：

<query xmlns="jabber:iq:register">

<instructions>

Some instructions to be displayed when the

user is filling out the form.

</instructions>

<username/>

<password/>

<email/>

<date/>

<key/>

</query>

### 6.12. jabber:iq:roster——联系人名单管理

联系人名单管理名字空间是客户端用来管理他们的花名册的，该花名册保存在服务器上，这样用户可以在任何地方得到它。花名册是该帐号上所有订阅信息的命令列表，包括用户昵称和联系列表。当用户从任何一个资源登陆上来，服务器将把花名册发送给用户。

花名册是一个条目的列表。每一个条目元素都有描述它的属性。每一个条目元素都包含它所属组的组元素。下面是描述<item/>元素的属性：

           “jid”是条目的Jabber ID

           “subscription”所有和本条目相关的订阅所对应的用户的当前状态。它可以是“none”（不可订阅），“to”（有一个发到该条目的订阅），“from”（从该条目发送一个订阅给我们），“both”（to、from），或者“remove”（从花名册上删除一个条目）。

           “ask”是对这个条目的一个请求的当前状态。Subscribe表示请求对该条目进行订阅，它的值就是，Unsubscribe表示取消订阅。

           “name”表示昵称

通过使用一个<iq type=”set”/>，一个Jabber客户端可以控制用户的JabberID，名称属性，群组元素，并且可以在联系人名单上创建或删除联系人。服务器根据客户端对当前状态订阅请求的响应来管理其它所有属性。

订阅请求的例子：

<iq type="set" id="uniquevalue">

<query xmlns="jabber:iq:roster">

<item jid="romeo@montague.net"

name="Romeo"

subscription="none"

ask="subscribe">

<group>friends</group>

</item>

</query>

</iq>

当一个花名册条目在服务器上改变时，服务器将把这个改变推送给一个用户。这个推送是一个普通的从服务器发送到客户端的<iq type=”set”/>包。下面是一个例子：

<iq type="set">

<query xmlns="jabber:iq:roster">

<item jid="romeo@montague.net" subscription="both"/>

</query>

</iq>

    在上面的例子中，服务器推送给客户端一个花名册，表明该用户对romeo@montague.net的当前状态信息有订阅。一个花名册推送可能发生一个连接的任何时间中，该连接是另一个连接修改资源或服务器（修改订阅属性）的结果。客户端只有使用花名册推送来改变花名册的显示。但不是每次花名册推送都对用户起作用。

### 6.13. jabber:iq:search——用户数据库查询

    任何一个代理都可以成为一个查询代理。例如，JUD查询Jabber用户，ICQ传输器查询ICQ用户。

    用户发送一个进行可查询域的Get请求：

<iq

type="get"

id="1001"

to="users.jabber.org"

from="romeo@montague.net/orchard">

<query xmlns="jabber:iq:search"/>

</iq>

    查询代理返回一个可查域：

<iq type="result" id="1001" from="users.jabber.org">

<query xmlns="jabber:iq:search">

<instructions>Fill in a field to search for any

matching Jabber User

</instructions>

<name/>

<first/>

<last/>

<nick/>

<email/>

<key>067941fd96a6a2752a21abcb6d737130dd51dd50</key>

</query>

</iq>

    注意这些域是以提示的形式返回。为了传输的安全还包含一个密钥（下面会看到）。用户现在可以在允许的域内进行标准的查询了。用户发回一个Set请求给代理进行实际查询：

    <iq

type="set"

id="1002"

to="users.jabber.org"

from="romeo@montague.net/orchard">

<query xmlns="jabber:iq:search">

<last>Capulet</last>

<key>11b830e604215c3a2a24652c69fd4efa2a7a5746</key>

</query>

</iq>

    服务器根据查询返回结果：

<iq type="result" id="1002" from="users.jabber.org">

<query xmlns="jabber:iq:search">

<item jid="juliet@capulet.com">

<name>Juliet Capulet</name>

<first>Juliet</first>

<last>Capulet</last>

<nick>juliC</nick>

<email></email>

</item>

<item jid="sampson@capulet.com>

<name>Sampson Capulet</name>

<first>Sampson</first>

<last>Capulet</last>

<nick>sammy</nick>

<email></email>

</item>

</query>

</iq>

    注意有两套items标记含有指定信息。这是因为通过代理发送返回有两种方法：

           单一返回标签

           多个“pushed”到用户的标签，类似花名册推送，如在一个时间上的一个记录。

发送的返回的样式是查询代理的一个工具。比如：

<iq type="set" from="icq.jabber.org" id="1003">

<query xmlns="jabber:iq:search">

<item jid="7090245@icq.jabber.org">

<email>juliet@capulet.com</email>

<nick>juliC</nick>

<given>Juliet</given>

<family>Capulet</family>

</item>

</query>

</iq>

多个查询结果通过服务器进行推送。当所有数据发送完毕，服务器将发送下面的返回结果：

<iq type="result" from="icq.jabber.org" id="1003">

<query xmlns="jabber:iq:search"/>

</iq>

客户端接收到多个“sets”，每一个都被记录，最后收到一个表示“数据结尾”的“返回”。在每一个<item>标签中，JID属性都被强制带上。

例子：

<iq

type="get"

id="1001"

to="users.jabber.org"

from="juliet@capulet.com">

<query xmlns="jabber:iq:search"/>

</iq>

### 6.14. jabber:iq:time——客户端时间

    客户端时间名字空间是客户端改变当前时间的一个标准方法。

    例子：

<iq type="result" from="stpeter@jabber.org">

<query xmlns="jabber:iq:time">

<utc>20010214T18:55:06</utc>

<display>2/15/00 4:55:06 PM</display>

</query>

</iq>

### 6.15. jabber:iq:version——客户端版本

    客户端版本名字空间是得到另一个用户的客户端的版本信息的一个标准方法。

    例子：

<iq type="result" from="stpeter@jabber.org/Gabber">

<query xmlns="jabber:iq:version">

<name>Gabber</name>

<version>0.8.1 (Powered by: jabberoo 1.0.0</version>

<os>Linux 2.2.16-22 i686</os>

</query>

</iq>

### 6.16. jabber:x:autoupdate——应用程序版本通知

    版本更改申请名字空间使服务器可以在一个应用的新版本可用时通知客户端。

    例子：

<message from="winjab.sourceforge.net">

<x xmlns="jabber:x:autoupdate">A new version of WinJab

is available, see http://winjab.sourceforge.net/

</x>

</message>

### 6.17. jabber:x:conference——会议邀请

    会议邀请名字空间使用户可以邀请其它Jabber用户到指定会议室。相关联信息——该会议室的Jabber ID，内嵌在一个消息的<x/>元素中。

    例子：

<message to="romeo@montague.net" from="juliet@capulet.com">

<body>Wherefore art thou Romeo?</body>

<x xmlns="jabber:x:conference" jid="balcony@conference.capulet.com/>

</message>

### 6.18. jabber:x:delay——标记对象延时

    标记对象延时名字空间是消息和当前状态信息的时间戳信息，其中，当一个Jabber实体不在线时，当前状态信息可以保存并在稍后发送。当实体上线时，包含的信息标记对象延时名字空间使Jabber客户端显示出该包体最早产生的时间，而不是它发送给Jabber实体的时间。

    例子：

<message

type=’groupchat’

from=’cauldron@conference.witches.org’>

<body>Thrice the brinded cat hath mew’d.</body>

<x xmlns=’jabber:x:delay’

stamp=’10541031T21:43:40’>Cached In GC History</x>

</message>

### 6.19. jabber:x:encrypted——加密消息

    加密消息名字空间用来支持使用公共密钥（一般通过客户端使用PGP或GPG，服务端用同样的名字空间进行加密来实现）加密的消息交互。一个相关的名字空间，jabber:x:signed，用来进行当前状态信息的加密。

    例子：

<message from=’juliet@capulet.com/balcony’ to=’romeo@montague.net/orchard’>

<body>This Message is Encrypted</body>

<x xmlns=’jabber:x:encrypted’>

hQEOA7ucqu53AhlPEAP/ZbU6oPnRAbIcUxMK1XRVnkgZ/Agtq1tcTQuEZxbpZLl4

C/4psQGLgBU5h5Y3/khxtJTPXKn1izyc+DRZ8hqn2p4mwC8ioKTBJ6P6dfEpQEyt

a4bimM5fqdeh4gRkMhwThRSJxBCJBVVWFEViu+0KlHHB5AeeL4qwRGb2dhGjIgQD

/R9x0D0qtgBGwf/TVnRGZYRX7epxCNuNDEYKZSs4LEoTPL8CVsAWOzS2QgS0GBqt

tFDKId6XaNu36dB2D8VHdxQnI8RtHo9pfTYDaXWB3dMGTt896iEO/sTuucYObf3s

K5Kygg0uWpBpvQPj8Y041FhnUBz8DRGCnuFLQxCI6ch4ybauXfOKNOGDQwmsnJZm

6OaeVFUwdsedI3c6VdQtodnWVutckR5BOjnn0VqnhrVTu7cp6VXrrRK4g9atPEe6

C4R/MilBjzIJBcET0jhWuAyiBo3gN/6IqYRZNSXL9ZqGPJwNTlYim1EHN3qBqiUw

zUMamEoRzcusn80Z7kylve5ujIeSD/pVwoawHHvLp92kO2hd0yGD0UrWSfKU1o6y

EY8yhZ5P1v02pIKigAUI2c6LTDxt/KhSxQ==

=fijN

</x>

</message>

### 6.20. jabber:x:envelope——消息信封

    消息信封名字空间是表明消息有更多的寻址方式，如联合email进行的寻址。下面是一些所支持的名字空间：

           to

           cc

           replyto

           from

           forwardedby

上述每一个元素都带有一个”jid”属性来标识Jabber实体是发送、中转、还是收到消息。

例子：

<message from=’juliet@capulet.com/tomb’ to=’romeo@montague.net/tomb’>

<body>And there I am. Where is my Romeo?</body>

<x xmlns=’jabber:x:envelope’>

<to jid=’romeo@heaven.org>Romeo</to>

<forwardedby jid=’root@heaven.org>God</forwardedby>

</x>

</message>

### 6.21. jabber:x:event——消息事件

    消息时间名字空间是标识一条消息的状态的一个机制。现在，事件与一条消息如下般相关联：

           <composing/>——个用户正在对消息进行回复

           <delivered/>——发送给指定接收者的消息

           <displayed/>——显示给指定接收者的消息

           <offline/>——为离线进行存储的消息

对于客户端来说，这些消息事件的支持是可选的，而且，只有在另一用户在聊天中发出请求，这些消息事件才会被发送。不同的Jabber客户端将对当前状态消息事件进行不同的显示。

对消息通知的请求的例子：

<message

from="juliet@capulet.com/balcony"

to="romeo@montague.net/orchard"

id="1001">

<body>By whose direction found’st thou out this place?</body>

<x xmlns="jabber:x:event">

<composing/>

</x>

</message>

发送消息通知的例子：

<message from="romeo@montague.net/orchard" to="juliet@capulet.com/balcony">

<body>By whose direction found’st thou out this place?</body>

<x xmlns="jabber:x:event">

<composing/>

<id>1001<id/>

</x>

</message>

    可以同时请求多个消息事件。

### 6.22. jabber:x:expire——消息到期

    消息到期名字空间是说明一条消息拥有一个有限的存活事件的一个简单扩展。如果消息被离线存储，而到了到期时间，服务器将不再发送该消息。如果一条消息为预览就进行发送，Jabber客户端可以选择不显示该消息。“secondes”属性定义消息发送的事件。

    例子：

<message from="sailor@denmark.com" to="horatio@denmark.com">

<body>There’s a letter for you sir</body>

<x xmlns="jabber:x:expire" seconds="3600"/>

</message>

### 6.23. jabber:x:oob——绑定数据输出

    绑定数据输出名字空间使用户可以通过交换一个标准的URL来实现文件传输的目电。使用jabber:x:oob的URLs交换可以包含任一消息（在一个<x/>子元素内），感觉就像email里的附件一样。多个附件可以包含在同一个消息中。

    例子：

<message from="sailor@denmark.com" to="horatio@denmark.com">

<body>URL Attached.</body>

<x xmlns="jabber:x:oob">

<url>http://denmark.com/act4/letter-1.html</url>

<desc>There’s a letter for you sir</desc>

</x>

</message>

### 6.24. jabber:x:roster——内置的花名册条目

    内置的花名册条目使用户可以在一个消息中包含花名册条目，这样很容易在用户之间发送联系方式。每一个花名册条目都包含在一个<x/>的一个<item/>子元素中。这个<x/>元素一般用在一个<message/>元素中，但这不是必须的。

    例子：

<message to="hamlet@denmark.com">

<body>

Here are some new Jabber users

to add to your contact list!

</body>

<x xmlns="jabber:x:roster">

<item jid="claudius@denmark.com" name="King">

<group>Royalty</group>

</item>

<item jid="horatio@denmark.com" name="Horatio">

<group>Friends</group>

</item>

<item jid="fortinbras@norway" name="Prince Fortinbras"/>

</x>

</message>

### 6.25. jabber:x:signed——有符号的当前状态

    有符号的当前状态名字空间用来支持交换使用公共密钥（客户端使用PGP或GPG，服务端使用同样的名字空间进行加密）加密的当前状态信息。一个相关的名字空间，jabber:x:encrypted用来支持加密消息。

    例子：

<presence from=’juliet@capulet.com/balcony’ to=’romeo@montague.net/orchard’>

<show>away</show>

<status>be back later</status>

<x xmlns=’jabber:x:signed’>

iD8DBQA6kasv0xpc2/POfPkRAnz0AJ9UEYYWWSReddIKk3AYMfTFtqQDJwCfbcLd

JcSUOR0FlS+rDFjAPaSMgSM+iNaNm

</x>

</presence>

### 6.26. vcard-temp——临时vCard

    vCard格式是一个“电子商务卡的标准格式，通过使用通过互联网进行个人数据的直接交换，如同在非互联网下的环境一样”。由于XML的vCard的格式还没有标准化，Jabber.org项目在XML的vCard标准建立之前，暂时设置了这样一个临时名字文件。由于这几年在vCard的XML的官方标准的指定上没有任何进步，Jabber.org开发者专门为这个项目创建了一个项目吸引Jabber社区外其它开发者的注意。因此，最后的XML的vCard格式的最后DTD可能会在下面的URL中找到：

<http://www.vcard-xml.org/>

    例子1（对vCard的客户端请求）：

<iq type="get">

<vCard xmlns="vcard-temp"/>

</iq>

    例子2（客户端收到vCard数据）：

<iq type="result">

<vCard version="3.0" xmlns="vcard-temp">

*vCard data goes here*

</vCard>

</iq>

    例子3（客户端向服务端发送vCard）：

<iq type="set">

<vCard version="3.0">

*vCard data goes here*

</vCard>

</iq>

## 7. 使用用例

    这一部分提供一些在Jabber协议上略有不同的观点，通过用例来阐述。下面每一个例子都展示一个Jabber用户完成一个完整的任务的消息流程，该流程包括接收和发送，如注册到一台服务器，登陆，改变当前状态，或者发送一条消息。如果时间允许，我将把这部分引申的更远。

### 7.1. Jabber用户注册

    本用例同时邪显示Jabber用户向服务器开启一个socket的连接，以及服务器的响应(如：<stream:stream>标签)。

1.    Jabber用户通过开启一个从客户端到服务器端的XML流，来申请一个在服务器上的socket连接。

SEND: <stream:stream

to=’capulet.com’

xmlns=’jabber:client’

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

2.    服务器通过开启一个从服务器到客户端的XML流进行回复。

RECV: <stream:stream

from=’capulet.com’

id=’180763465’

xmlns=’jabber:client’

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

3.    Jabber用户提供一个需要注册一个帐号（理论上，这需要一个不同的顺序：首先询问服务器需要什么信息，然后服务器器告知客户端需要什么样的信息；但在实际中，假定需要的信息是：用户名，资源，密码）的信息。

SEND:   <iq id=’1’ type=’set’>

           <query xmlns=’jabber:iq:register’>

               <username>Juliet</username>

               <resource>balcony</resource>

           </query>

        </iq>

4.    服务器响应一个空的类型为”result”的iq元素，表示注册已成功。

RECV:   <iq id=’1’ type=’result’/>

### 7.2. Jabber用戶登陆

1.     Jabber用户询问服务器，登陆所需要提供的信息。

SEND:  <iq id=’2’ type=’get’>

           <query xmlns=’jabber:iq:auth’>

              <username>Juliet</username>

           </query>

       </iq>

2.     服务器提示Jabber用户登陆所需要的信息。

RECV:  <iq id=’2’ type=’result’>

           <query xmlns=’jabber:iq:auth’>

              <username>Juliet</username>

              <password/>

              <digest/>

              <sequence>500<sequence/>

              <token>3B905BFD</token>

              <resource/>

           </query>

       </iq>

3.     Jabber用户提供所需的信息——在本例中，是一个<hash/>元素，它是对信息进行一个零度知识认证的一个哈希。（详情请见<http://docs.jabber.org/draft-proto/html/zerok.html>）。

SEND:  <iq id=’3’ type=’set’>

           <query xmlns=’jabber:iq:auth’>

              <username>Juliet</username>

              <resource>balcony</resource>

                          <hash>77d7eacde5e56b9622d0a075cb88361b110f

b9d7</hash>

                     </query>

                  </iq>

4.     服务器响应一个空的类型为”result”的iq元素，表明登陆成功。

RECV:  <iq id=’3’ type=’result’/>

5.     Jabber用户发送当前状态给服务器，表明其在线。

SEND:  <presence>

           <status>Online</status>

       </presence>

### 7.3. Jabber用户增加一个联系人

1.     Jabber客户端在花名册上增加一个联系人（只是预备的操作）。

SEND:  <iq type=’set’>

           <query xmlns=’jabber:iq:roster’>

              <item jid=’romeo@montague.net’

Name=’remeo’/>

                     </query>

                  </iq>

2.     Jabber用户发送一个对该联系人的订阅请求。

SEND:  <presence to’remeo@montague.net’ type=’subscribe’>

           <status>Wherefore are thou?</status>

       </presence>

3.     服务器发送一个花名单推送给用户一个新条目和一个类型为”none”的订阅（早已有了该订阅）。

RECV:  <iq type=’set’>

           <query xmlns=’jabber:iq:roster’>

              <item jid=’romeo@montague.net’

name=’romeo’

subscription=’none’/>

                     </query>

                  </iq>

4.     服务器发送一个类型为”result”的iq包，表示花名册推送成功（让人有点疑惑的是，这个包是从Jabber用户发送到Jabber用户的！）

RECV:  <iq

from=’juliet@capulet.com/balcony’

to=’julie@capulet.com/balcony’

type=’result’>

5.     服务器发送另一个花名单推送，这次是ask=’subscribe’属性，表示订阅的状态未定。

RECV:  <iq type=’set’>

           <query xmlns=’jabber:iq:roster’>

              <item

ask=’subscribe’

jid=’romeo@montague.net’

name=’romeo’

subscription=’none’/>

                     </query>

                  </iq>

6.     Romeo对订阅请求的下一个响应会是什么呢？现在我们假定订阅以Romeo接受订阅请求为“结束”的。

RECV:  <presence

from=’romeo@montague.net’

to=’juliet@capulet.com’

type=’subscribed’/>

7.     服务器再次发送一个花名单推送给客户端，这次subscription=’to’，表示订阅请求被接受了（Juliet现在可以订阅到Romeo的当前状态了）。

RECV:  <iq type=’set’>

           <query xmlns=’jabber:iq:roster’>

              <item

jid=’romeo@montague.net’

name=’romeo’

subscription=’to’/>

                     </query>

                  </iq>

### 7.4. Jabber用户获得花名册

           SEND:  <iq type=’get’>

                     <query xmlns=’jabber:iq:roster’/>

                  </iq>

RECV:<iq from=’juliet@capulet.com/balcony’ type=’result’>

                     <query xmlns=’jabber:iq:roster’>

                         <item

jid=’romeo@montague.net’

name=’romeo’

subscription=’both’/>

                     </query>

                  </iq>

### 7.5. Jabber用户发送一条消息

           SEND:  <message to’romeo@montague.net’>

                     <body>Wherefore are thou?</body>

                  </message>

    注意：服务器会根据Jabber用户的会话信息加上一个源地址，这样接收者收到消息时，消息中已经包含了源地址。

### 7.6. Jabber用户改变当前状态

       SEND:  <presence>

                  <status>stepped away…</status>

                  <show>away</show>

              </presence>

### 7.7. Jabber用户登出

    登出时很容易的，只需要关闭<stream>……

       SEND:  </stream:stream>

       RECV:  </stream:stream>

## 8. 参考文献

    本协议参考以下文献

           Jabber开发指南（<http://docs.jabber.org/jpg/>）

           Jabber协议――标准（<http://docs.jabber.org/proto/>）

           Jabber协议――草案（<http://docs.jabber.org/draft-proto/>）

           Romeo and Juliet

(<http://tech-two.mit.edu/Shakespeare/Tragedy/romoandjuliet/full.html>)

## 9. 结束语

    本文档提供了一个关于Jabber的XML协议的一个详细的怪数。如果你对本文档有任何问题，请通过email或Jabber的方式自由地与作者联系（Peter Saint-Andre），他的帐号时stpeter@jabber.org。

## 10. 版权信息

This document is copyright 2001 by Peter Saint-Andre.

Permission is granted to copy, distribute and/or modify this document under the terms of the *GNU Free Documentation License*(http://www.gnu.org/copyleft/fdl.html),

Version 1.1 or any later version published by the Free Software Foundation, with no

Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. You may obtain a

copy of the GNU Free Documentation License from the Free Software Foundation by

visiting *http://www.fsf.org/*or by writing to:

The Free Software Foundation, Inc.

59 Temple Place - Suite 330

Boston, MA 02111-1307

USA

# 附录 A 标准错误代码

    下面是Jabber中错误代码的一些简要描述。Jabber服务器在碰到不同的错误条件下生成这些错误代码。一般Jabber错误代码以HTTP规格的RFC 2616（http://www.ietf.org/rfc/rfc2616.txt）为基础。但Jabber没有使用所有的HTTP的错误代码，并且Jabber错误代码与HTTP错误代码相对应时，通常只有Jabber自己的含义。注意本附录只包含Jabber服务器生成的错误代码，而不包括服务器端组件如网关发到外部消息系统。

|  |  |  |
| --- | --- | --- |
| 错误代码 | 描述 | 注意 |
| 302 | 重定向 | 尽管HTTP规定中包含八种不同代码来表示重定向，Jabber只用了其中一个（用来代替所有的重定向错误）。不过Jabber代码302是为以后的功能预留的，目前还没有用到 |
| 400 | 坏请求 | Jabber代码400用来通知Jabber客户端，一个请求因为其糟糕的语法不能被识别。例如，当一个Jabber客户端发送一个的订阅请求给它自己活发送一条没有包含“to”属性的消息，Jabber代码400就会产生。 |
| 401 | 未授权的 | Jabber代码401用来通知Jabber客户端它们提供的是错误的认证信息，如，在登陆一个Jabber服务器时使用一个错误的密码，或未知的用户名。 |
| 402 | 所需的费用 | Jabber代码402为未来使用进行保留，目前还不用到。 |
| 403 | 禁止 | Jabber代码403被Jabber服务器用来通知Jabber客户端该客户端的请求可以识别，但服务器拒绝执行。目前只用在注册过程中的密码存储失败。 |
| 404 | 没有找到 | Jabber代码404用来表明Jabber服务器找不到任何与JabberID匹配的内容，该JabberID是一个Jabber客户端发送消息的目的地。如，一个用户打算向一个不存在的JabberID发送一条消息。如果接受者的Jabber服务器无法到达，将发送一个来自500级数的错误代码。 |
| 405 | 不允许的 | Jabber代码405用在不允许操作被’from’地址标识的JabberID。例如，它可能产生在，一个非管理员用户试图在服务器上发送一条管理员级别的消息，或者一个用户试图发送一台Jabber服务器的时间或版本，或者发送一个不同的JabberID的vCard。 |
| 406 | 不被接受的 | Jabber代码406用于服务器因为某些理由不接受一个包。例如，这个可能发生在，一个Jabber客户端试图使用jabber:iq:private在服务器上存储信息，但当前的用于存储的名字空间用”jabber:”开头（在Jabber里是一个被存的XML开头）。另一种可能产生406错误的情况是当一个Jabber客户端试图用一个空密码注册到一台Jabber服务器上。 |
| 407 | 必须注册 | Jabber代码407当前不被使用 |
| 408 | 注册超时 | 当一个Jabber客户端不能在服务器准备好的时间内发起一个请求时，Jabber服务器生成Jabber代码408。这个代码当前只用于Jabber会话管理器使用的零度认证模式中。 |
| 409 | 冲突 | 略 |
| 500 | 服务器内部错误 | 当一台Jabber服务器遇到一种预期外的条件，该条件阻止服务器处理来自Jabber客户端的包，这是将用到Jabber代码500。现在，唯一会引发500错误代码的时间是当一个Jabber客户端试图通过服务器认证，而该认证因为某些原因没有被处理（如无法保存密码）。 |
| 501 | 不可执行 | 当服务器不支持Jabber客户端请求的功能，使用Jabber代码501。例如，该代码只当Jabber客户端发送一个认证请求，而该认证请求不包含服务器配置中定义的任何一种认证方式时，服务器发送Jabber代码501。这个代码还被用于，当一个Jabber客户端试图注册一个不允许注册的服务器。 |
| 502 | 远程服务器错误 | 当因为无法到达远程服务器导致转发一个包失败时，使用Jabber代码502。该代码发送的特殊例子包括一个远程服务器的连接的失败，无法获取远程服务器的主机名，以及远程服务器错误导致的外部时间过期。 |
| 503 | 服务无法获得 | 当一个Jabber客户端请求一个服务，而Jabber服务器通常由于一些临时原因无法提供该服务时，使用Jabber代码503。例如，一个Jabber客户端试图发送一条消息给另一个用户，该用户不在线，但它的服务器不提供离线存储服务，服务器将返回一个503错误代码给发送消息的JabberID。当为vcard-temp和jabber:iq:private名字空间设置信息时，出现通过xdb进行数据存储的写入错误，也使用该代码。 |
| 504 | 远程服务器超时 | Jabber代码504用于下列情况:试图连接一台服务器发生超时，错误的服务器名。 |
| 510 | 连接失败 | Jabber代码510目前还没有使用。 |

Overview

[XMPP](http://www.xmpp.org/)由[Jabber软件基金会](http://www.jabber.org/)开发，最早在Jabber上实现。Jabber项目由Jeremie Miller在1998年开始的一个免费、开源的项目，用于提供给MSN、Yahoo！的IM服务。由于XMPP是一种基于XML架构的开放式协议，在IM通讯中被广泛采用，已经得到了互联网工程任务组(IETF)的批准（[RFC 3920](http://www.ietf.org/rfc/rfc3920.txt)|[RFC 3921](http://www.ietf.org/rfc/rfc3921.txt)）。

### IQ

即Information Query，是XMPP协议中一个类似HTTP的GET、POST的动词。比如说修改Jabber密码：  
  
<iq type='set' to='jabber.org'>  
<query xmlns='jabber:iq:register'>  
<username>username</username>  
<password>newpassword</password>  
</query>  
</iq>

### JID

即Jabber ID，格式为：  
[node@]domain[/resource]  
node是用户名，domain是域；如果是Google Talk的话域就是 gmail.com；resource表示工作状态，Jabber允许在不同的地方同时登陆一个Jabber账号，用resource来表示不同的状态，可以自定义，GTalk目前版本定义的resource就是 Talk.v104C0F37955  
例如：estbot@googlemail.com/AtWork

### Disco

Service Discovery，一个Jabber服务器可能不止提供了IM服务，还提供了和其它IM互通网关、群聊等特性。

### Stanza

一个XML片段

### Roster

联系人名单

### Presence

即隐身功能，把你的在线/离线状态只告诉一部分联系人

### Subscription

订阅一个联系人的在线状态，通俗的讲就是添加好友。A给服务器发送一个presence消息，要subscribe B，那么相当于A要添加B为好友。

### show

忙碌状态  
away――在线，但不能马上联系上  
chat――在线并有兴趣聊天  
dnd――在线，但不想被打扰（“dnd”表示“do not disturb”）  
xa――在线，但已经离开很长时间了（“xa”表示“extended away”）  
GTalk里，绿色图标表示<show />，没有任何show的值，如果是红色图标则是<show>dnd</show>，只有这两种状态

### status

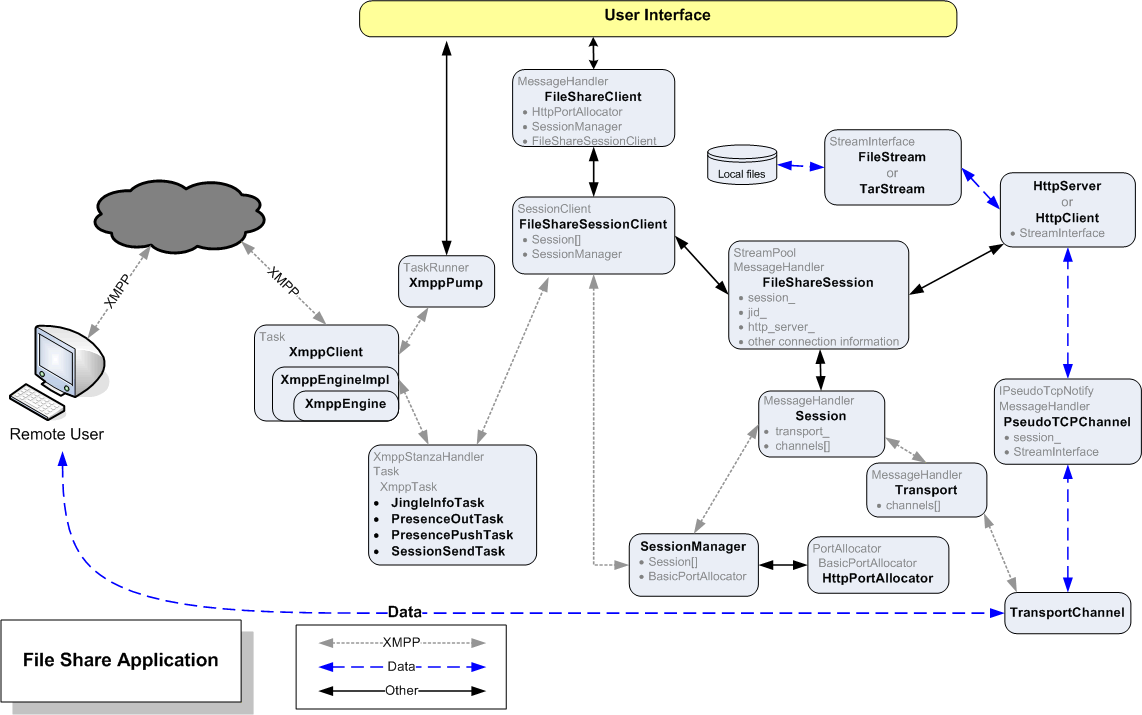
好友的个性签名。好友的头像是通过X标签来获得的

### vCard

联系人名片，见 [RFC 2426](http://rfc.net/rfc2426.html)

### Jingle

Google定义的Jabber扩展，包含了p2p、语聊、文件传输、穿防火墙/NAT/Proxy等高级功能。例如[File Transfer模块](http://code.google.com/apis/talk/libjingle/file_share.html)：



# Subscriptions 运行机制

**Peter Saint-Andre**

**Jabber, Inc.**

## 1、介绍

Subscriptions是Jabber中最复杂的部分。服务端和客户端做了大量的工作，以保证用户在不同状态下接收或发送正确的subscription请求，接受或拒绝其他用户的请求，取消订阅，删除花名册条目，等等。本文档着重解释subscription在两层结构中是如何工作的：（1）XML协议要求对特定动作进行响应；（2）user:xml文件的返回状态包括Jabber User（如 订阅者等等）以及联系人（如被订阅者）。

## 2、Subscription 状态

总共有四种subscription状态：

1、  None——Jabber用户和联系人互相都不订阅对方的presence

2、  To——Jabber用户被联系人的presence订阅，但联系人并不被Jabber用户的presence订阅。

3、  From——联系人接受来自Jabber用户的subscription，但联系人不被Jabber用户的presence所订阅（这样的结果就是Jabber用户在联系人的花名册中“不可见”）。

4、  Both——Jabber用户和联系人被彼此的presence订阅。

当Jabber用户将联系人加入他的花名册中（通常是通过对联系人发送一个subscriptiong请求实现），名义上还存在第五种状态，就是Jabber用户和联系人甚至不知道对方的存在（至少通过每一个用户的花名册显示）。不过，我们对这种状态没有命名。从本文档出发，我将这种状态定义为初始状态。

## 3、订阅一个联系人

就象上面提到的，subscriptions和以及它们相关联的花名册状态是相当复杂的。所以这里我将把一个Jabber用户与一个联系人之间的交互进行极其详尽的描述，以便你能搞清楚当一个Jabber用户订阅一个联系人的presence，结果引起该Jabber用户对联系人产生了一个subcription这样的过程中，究竟发生了些什么。我们将从Jabber用户对联系人发送一个subscription请求开始。

       1、为了让联系人在Jabber用户的花名册上可见，Jabber用户客户端

(1)    发送一个<iq/>包，其jabber:iq:roster名字空间中包含一个type=’set’；

(2)    成功的情况下，从服务器接收到一个<iq/>包，其type=’result’。

(3)    从服务器接收到一个“roster push”，产生一个新的roster条目，其subscription状态置为“none”：

JABBER USER SENDS:

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’contact@host’

name=’contact’/>

</query>

</iq>

JABBER USER RECEIVES:

<iq

type=’result’

from=’jabberuser@host/work’

to=’jabberuser@host/work’/>

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’contact@host’

subscription=’none’

name=’contact’/>

</query>

</iq>

              2、Jabber用户的客户端接着发送一个type=’subscribe’的<presence/>的包给联系人：

                     JABBER USER SENDS:

<presence

to=’contact@host’

type=’subscribe’>

<status>I would like to add you to my roster.</status>

</presence>

3、Jabber用户客户端接着从服务器收到第二个包含联系人待定子状态的’none’订阅状态的“roster push”；这个待定子状态在其花名册条目中包含一个ask=’subscribe’属性：

                     JABBER USER RECEIVES:

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’contact@host’

subscription=’none’

name=’contact’

ask=’subscribe’/>

</query>

</iq>

4、一旦Jabber用户客户端发送type=’subscribe’的<presence/>包，该包被发送给联系人（我们假定该联系人在线）：

                     CONTACT RECEIVES:

<presence

to=’contact@host’

type=’subscribe’

from=’jabberuser@host’>

<status>I would like to add you to my roster.</status>

</presence>

5、  联系人下一步要做的就是决定是否接受订阅请求。这里我们假定为”happy path”，即联系人接受订阅，这样联系人的Jabber客户端：

（1）       发送一个type=’subscribed’的<presence/>的包给Jabber用户

（2）       从服务器接收一个“roster push”包含Jabber用户的条目，其中subscription状态置为“from”：

CONTACT SENDS:

<presence to=’jabberuser@host’ type=’subscribed’/>

CONTACT RECEIVES:

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’jabberuser@host’

subscription=’from’/>

</query>

</iq>

6、  联系人接受订阅请求的结果是：Jabber用户客户端接收到

（1）       一个从联系人发来的type=’subscribed’的<presence/>的包

（2）       从服务器发来的一个“roster push”，其中包含一个更新的条目，用来更新联系人的subscription为“to”；

JABBER USER RECEIVES:

<presence

to=’jabberuser@host’

type=’subscribed’

from=’contact@host’/>

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’contact@host’

subscription=’to’

name=’contact’/>

</query>

</iq>

                     现在Jabber用户已经订阅了联系人。每个user.xml的花名册条目将如下所示：

JABBER USER’S ROSTER:

<item

jid=’contact@host’

subscription=’to’

name=’contact’/>

           CONTACT’S ROSTER:

<item

jid=’jabberuser@host’

subscription=’from’/>

（注意：如果在这里Jabber用户发送一个subscription请求给联系人，Jabber用户的Jabber服务器将“吞没”该请求，而不再将其发送给联系人。）

## 4、从一个联系人获取一个Subscription

       如我们在前面所见，需要有一个Jabber用户和联系人之间的双向的subscription。在这个过程结束部分，我们将研究Jabber用户与联系人互相订阅的情况。

## 5、创建一个相互的Subscription

       要建立一个相互的Subscription（如 a subscription=’both’），联系必须发送一个subscription请求给Jabber用户，并且Jabber用户必须接受该请求。让我们看看XML传回了什么，弄清楚其中发生了什么，以及每个人的花名册文件发生了什么变化。

1、  首先，联系人发送一个订阅请求给Jabber用户：

CONTACT SENDS:

<presence to=’jabberuser@host’ type=’subscribe’>

                     <status>I would like to add you to my roster.</status>

</presence>

2、  联系人客户端从服务器接收到一个“roster push”，其中包含Jabber用户仍然是‘from’的subscription状态，但同时一个待定’to’的subscription通过花名册条目中的ask=’subscribe’的属性产生了：

CONTACT RECEIVES:

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’jabberuser@host’

subscription=’from’

name=’jabberuser’

ask=’subscribe’/>

</query>

</iq>

此行为的一个结果是，联系人和Jabber用户的花名册文件的状态如下所示：

CONTACT’S ROSTER:

<item

jid=’jabberuser@host’

subscription=’from’

name=’jabberuser’

ask=’subscribe’/>

JABBER USER’S ROSTER:

<item

jid=’contact@host’

subscription=’to’

name=’contact’

subscribe=’I would like to add you to my roster.’/>

3、  一旦联系人客户端发送一个包含type=’subscribe’的<presence/>包，该包将直接发送给Jabber用户（我们假定此时Jabber用户在线）：

JABBER USER RECEIVES:

<presence

to=’jabberuser@host’

type=’subscribe’

from=’contact@host’>

<status>I would like to add you to my roster.</status>

</presence>

4、Jabber用户下一步要做的就是决定是否接受订阅请求。这里我们假定为”happy path”，即联系人接受订阅，这样Jabberyonghu的Jabber客户端：

（1）       发送一个type=’subscribed’的<presence/>的包给联系人

（2）       从服务器接收一个“roster push”包含联系人的条目，其中subscription状态置为“both”：

JABBER USER SENDS:

<presence to=’contact@host’ type=’subscribed’/>

JABBER USER RECEIVES:

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’contact@host’

subscription=’both’/>

</query>

</iq>

5、Jabber用户接受订阅请求的结果是：联系人客户端接收到

（1）       一个从Jabber用户发来的type=’subscribed’的<presence/>的包

（2）       从服务器发来的一个“roster push”，其中包含一个更新的条目，用来更新Jabber用户的subscription为“to”；

CONTACT RECEIVES:

<presence

to=’contact@host’

type=’subscribed’

from=’jabberuser@host’/>

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’jabberuser@host’

subscription=’both’

name=’jabberuser’/>

</query>

</iq>

此行为的一个结果是，联系人和Jabber用户的花名册文件的状态如下所示：

       CONTACT’S ROSTER:

<item

jid=’jabberuser@host’

subscription=’both’

name=’jabberuser’/>

JABBER USER’S ROSTER:

<item

jid=’contact@host’

subscription=’both’

name=’contact’/>

Jabber用户和联系人现在有了一个相互交互的subscription——其subscription的类型为“both”。（注意：如果在这里Jabber用户发送一个subscription请求给联系人，或者联系人发送一个subscription请求给Jabber用户，发送方的Jabber服务器将“吞没”该请求，而不再将其发送给联系人。）

## 6、取消一个Subscription

一旦在Jabber中你订阅了某人的presence，在某些情况下你就有可能需要取消订阅对方的presence。

尽管你发送的XML都一样，Jabber用户和联系人的花名册文件根据取消订阅包发过来时它们的花名册文件的状态的不同而表现不同。

### 6.1、例#1:Subscription Type ‘to’

       我们的第一个例子中，我们假定Jabber用户有一个到联系人的subscription，而联系人没有到Jabber用户的subscription。在这个例子中，每个人的花名册文件的状态如下：

JABBER USER’S ROSTER:

<item

jid=’contact@host’

subscription=’to’

name=’contact’/>

CONTACT’S ROSTER:

<item

jid=’jabberuser@host’

subscription=’from’

name=’jabberuser’/>

1、  为了取消订阅联系人的presence，Jabber用户发送下面的XML：

JABBER USER SENDS:

<presence to=’contact@host’ type=’unsubscribe’/>

2、  发送presence包的结果，Jabber用户客户端从Jabber服务器上接收到一个“roster push”，其中包含‘ask’属性来取消订阅，指示取消订阅是待定的：

JABBER USER RECEIVES:

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’contact@host’

subscription=’to’

name=’contact’

ask=’unsubscribe’/>

</query>

</iq>

3、  现有Jabber服务器功能将会代替联系人的行为自动回复出一个type为unsubscribe的presence包（注意：这意味着联系人在不在线都会一样进行处理）。这就导致Jabber用户接收到以下的XML：

JABBER USER RECEIVES:

<presence

type=’unsubscribed’

to=’jabberuser@host’

from=’contact@host’>

<status>Autoreply</status>

</presence>

4、  Jabber用户还将收到一个来自Jabber服务器的“roster push”，其联系人subscription被再一次设为‘none’：

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’contact@host’

subscription=’none’

name=’contact’/>

</query>

</iq>

5、  最后使得联系人和Jabber用户的花名册文件如下所示：

JABBER USER’S ROSTER:

<item

jid=’contact@host’

subscription=’none’

name=’contact’/>

CONTACT’S ROSTER:

no <item/> for Jabber User

### 6.2、例＃2：Subscription Type ‘both’

       第二个例子，我们假定Jabber用户有一个到联系人的subscription，联系人也有一个到Jabber用户的subscription。在这个例子中，各人的花名册的状态如下：

JABBER USER’S ROSTER:

<item

jid=’contact@host’

subscription=’both’

name=’contact’/>

CONTACT’S ROSTER:

<item

jid=’jabberuser@host’

subscription=’both’

name=’jabberuser’/>

1、  为了取消来自联系人的presence的订阅，Jabber用户发送以下XML：

JABBER USER SENDS:

<presence to=’contact@host’ type=’unsubscribe’/>

2、  发送此presence包后，Jabber用户客户端马上接收到来自Jabber服务器的一个“roster push”，其中’ask’属性设为unsubscribe，以指示此取消订阅待定：

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’contact@host’

subscription=’both’

name=’contact’

ask=’unsubscribe’/>

</query>

</iq>

3、现有的Jabber系统将自动代替联系人回复一个unsubscripe的presence包（注意：这意味着无论联系人是否在线，对于subscription请求是被待定还是不予处理，结果都一样，尽管“roster push”在请求被待定时有少许不同）。Jabber用户将接收到下列XML：

JABBER USER RECEIVES:

<presence

type=’unsubscribed’

to=’jabberuser@host’

from=’contact@host’>

<status>Autoreply</status>

</presence>

4、  Jabber用户将收到一个来自Jabber服务器的“roster push”，其subscription将联系人改成’from’：（联系人继续订阅Jabber用户的presence）

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’contact@host’

subscription=’from’

name=’contact’/>

</query>

</iq>

5、  联系人将收到一个来自Jabber服务器的“roster push”，其subscription将Jabber用户改成‘to’（从此Jabber用户不再订阅联系人的presence）

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’jabberuser@host’

subscription=’to’

name=’jabberuser’/>

</query>

</iq>

6、  最后，联系人与Jabber用户的花名册文件的状态如下：

JABBER USER’S ROSTER:

<item

jid=’contact@host’

subscription=’from’

name=’contact’/>

CONTACT’S ROSTER:

<item

jid=’jabberuser@host’

subscription=’to’

name=’jabberuser’/>

## 7、删除一个好友条目

       除了取消订阅来自某人的presence，你也可以从你的花名册中删除对方的对应条目。你可以发送一个包含特殊subscription为’remove’的<iq/>包。通常，发送一个删除请求将开始一个从你的花名册删除某人的过程，同时服务器来执行取消订阅的细节。下面是其工作流程……

### 7.1、例＃1：Subscription Type ‘to’

       我们第一个例子，我们将假定Jabber用户有一个到联系人的subscription，而联系人却没有到Jabber用户的subscription。在这个例子中，各人的花名册文件的状态如下：

JABBER USER’S ROSTER:

<item

jid=’contact@host’

subscription=’to’

name=’contact’/>

CONTACT’S ROSTER:

<item

jid=’jabberuser@host’

subscription=’from’

name=’jabberuser’/>

1、  第一步是Jabber用户发送取消请求：

JABBER USER SENDS:

<iq type=’set’>

<query

xmlns=’jabber:iq:roster’><item

jid=’contact@host’

subscription=’remove’/>

</query>

</iq>

2、  发送取消请求后，Jabber用户接收到：

（1）       一个对联系人的“roster push”

（2）       一个type=’result’的<iq/>包，指示开始进行请求：

JABBER USER RECEIVES:

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item jid=’contact@host’ subscription=’remove’/>

</query>

</iq>

<iq

type=’result’

from=’jabberuser@host/work’

to=’jabberuser@host/work’/>

3、  另外，联系人接收到：

（1）       一个’unsubscribe’的presence包

（2）       一个“roster push”，指示Jabber用户的花名册条目的subscription为‘none’：

CONTACT RECEIVES:

<presence

type=’unsubscribe’

to=’contact@host’

from=’jabberuser@host’/>

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’jabberuser@host’

subscription=’none’

name=’jabberuser’/>

</query>

</iq>

### 7.2、例＃2：Subscription Type ‘both’

       第二例子，我们假定Jabber用户有一个到联系人的subscription，联系人也有一个到Jabber用户的subscription。在这个例子中，各人的花名册文件状态如下：

JABBER USER’S ROSTER:

<item

jid=’contact@host’

subscription=’both’

name=’contact’/>

CONTACT’S ROSTER:

<item

jid=’jabberuser@host’

subscription=’both’

name=’jabberuser’/>

1、  第一步是Jabber用户发送删除请求：

JABBER USER SENDS:

<iq type=’set’>

<query

xmlns=’jabber:iq:roster’><item

jid=’contact@host’

subscription=’remove’/>

</query>

</iq>

2、发送请求后，Jabber用户接收到：

（1）       一个对联系人的“roster push”包

（2）       一个type=’result’的<iq/>包来开始请求：

JABBER USER RECEIVES:

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item jid=’contact@host’ subscription=’remove’/>

</query>

</iq>

<iq

type=’result’

from=’jabberuser@host/work’

to=’jabberuser@host/work’/>

3、同时，联系人接收到：

（1）       一个type为’unsubscribe’的presence包

（2）       一个“roster push”包，改变Jabber用户的花名册条目的subscription为‘to’

（3）       一个type为’unsubscribed’的presence包

（4）       一个“rost push”包，改编Jabber用户的花名册条目的subscription为“none”；

CONTACT RECEIVES:

<presence

type=’unsubscribe’

to=’contact@host’

from=’jabberuser@host’/>

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’jabberuser@host’

subscription=’to’

name=’jabberuser’/>

</query>

</iq>

<presence

type=’unsubscribed’

to=’contact@host’

from=’jabberuser@host’/>

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’jabberuser@host’

subscription=’none’

name=’jabberuser’/>

</query>

</iq>

## 8、客户端开发注意事项

       从上面的论述中，必须要清楚，详尽的XML片断并不一定指得是同一个事物，也不一定拥有相同的含义——其含义根据实际情况中subscription的上下文以及花名册条目的状态决定。Jabber客户端的开发人员在写代码时，需要牢记这一点。一般来说，最好不要修改客户端一方关于花名册地描述，该花名册以你发送到服务器的外部XML为基础。在大多数情况下，你只需要等待“roster push”并使你的客户端根据其进行改变。

## 9、引用

       本文引用了下列材料：

         ***Jabber Programmers Guide***（<http://docs.jabber.org/jpg/>）

         ***Jabber Protocol Overview***（<http://docs.jabber.org/general/html/protocol.html>）

         ***Jabber protocol－Standard***（<http://docs.jabber.org/proto/>）

## 注意

1、  如果联系人不在线，下列XML保存到联系人的花名册文件中：

<item

jid=’jabberuser@host’

subscription=’none’

subscribe=’I would like to add you to my roster.’/>

       联系人下次登录时，服务器将提示’subscribe’属性，并产生一个来自Jabber用户的type=’subscribe’<presence/>包，发送给联系人。

2、  如果联系人拒绝了订阅请求，联系人的客户端发送下列XML给Jabber用户:

<presence to=’jabberuser@host’ type=’unsubscribed’/>

Jabber用户收到这个presence包，同时收到一个来自服务器的“roster push”

<presence

to=’jabberuser@host’

type=’unsubscribed’

from=’contact@host’/>

       <iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’contact@host’

subscription=’none’

name=’contact’/>

</query>

</iq>

       这个操作后，Jabber用户和联系人的花名册状态如下：

JABBER USER’S ROSTER:

<item

jid=’contact@host’

subscription=’none’

name=’contact’/>

       CONTACT’S ROSTER:

no <item/> for Jabber User

       你肯定想知道为什么对联系人的<item/>没有直接从Jabber用户的花名册中删除掉（那样返回的花名册我叫它“初始状态”），二是在花名册中保存了一个type为“none”的subscription。好问题，但我想说的时现有的Jabber系统机制是以这个方式运行的。（注意如果在这种状态Jabber用户向联系人发送一个订阅请求，其表现就联系人不在线的表现完全相同，Jabber用户客户端不必从服务器上获取初始的花名册是的这种情况除外。）

3、  如果Jabber用户在线，下面的XML保存仔Jabber用户的花名册文件中：

<item

jid=’contact@host’

subscription=’to’

subscribe=’I would like to add you to my roster.’/>

下次Jabber用户登录时，服务器注意到’subscribe’属性，然后生成一个从联系人到Jabber用户的的type=’subscribe’的<presence/>包。

4、  如果Jabber用户拒绝订阅请求，Jabber用户的客户端发送下列XML给联系人：

<presence to=’contact@host’ type=’unsubscribed’/>

       联系人也将从服务器上收到一个“roster push”，改变Jabber用户的订阅状态为“from”：

<presence

to=’contact@host’

type=’unsubscribed’

from=’jabberuser@host’/>

<iq type=’set’>

<query xmlns=’jabber:iq:roster’>

<item

jid=’jabberuser@host’

subscription=’from’

name=’jabberuser’/>

</query>

</iq>

       结果，联系人和Jabber用户的花名册名单的状态如下：

       CONTACT’S ROSTER:

<item

jid=’jabberuser@host’

subscription=’from’

name=’jabberuser’/>

       JABBER USER’S ROSTER:

<item

jid=’contact@host’

subscription=’to’

name=’contact’/>

2010年05月9日 8:35 下午

# [XMPP 协议笔记 一 基础篇](http://chloerei.com/2010/05/09/172)

**1. XMPP 优缺点**

XMPP (Extensible Messaging and Presence Protocol) (前称Jabber) 是一种以 XML 为基础的开放式即时通讯协议，是经由互联网工程工作小组 (IETF) 通过的互联网标准。[1]

**1.1 XMPP 协议的优点**

**1.1.1 可扩展性**

XMPP 的数据传输基于 XML 格式，可扩展性强。XMPP 的核心协议栈 (Core Stack) 部分只定义了基础的 Presence，Message，Iq 等最主要数据格式和传输逻辑，更多的功能则通过定义扩展 (Extensions) 实现。

**1.1.2 受 IETF 组织规范**

Internet Engineering Task Force  (IETF) 在2002年开始规范 XMPP 协议，使其协议的修订和扩展的添加都经过严格的流程审核，防止 XMPP 协议因缺乏标准而分裂。并且这也保证了 XMPP 协议是完全开放的。

**1.1.3 应用广泛**

XMPP 协议的应用比其他开放即时通讯协议更为广泛。较有名的使用 XMPP 协议的聊天服务有 Google Gtalk 和 Facebook Chat 等。此外，XMPP 在各平台下都有若干服务端、客户端和程序库的实现，二次开发时成本较低。

XMPP 协议的可扩展性和开放性是该协议被广泛应用的保证。

**1.2 XMPP 协议的缺点**

**1.2.1 不内置支持二进制数据的传输**

XMPP 的核心部分没有包含对二进制数据传输的支持，这使得 XMPP 的基本数据限定在文本文件范围内。XMPP 社区认为，XMPP 应该用于传输 meta 信息，辅助其他应用进行协议握手，XMPP 本身不应负担海量信息的传输。

从当前流行的轻量化观点来看，XMPP 把二进制数据传输的协议移入核心栈，是符合了最小核心的需求。但同时却为实际应用中 XMPP 客户端传输二进制数据增加了开发扩展协议的负担。

**1.2.2 缺乏旗舰应用**

XMPP 是开放的，任何个人和组织都可以使用 XMPP。但同时产生的副作用是每个组织使用 XMPP 的目的不同，侧重点不同，导致XMPP 所开发的应用实际上导致了各个厂商各自为政，比如 Cisco 将 XMPP 用于设备通信，游戏厂商用于游戏内的简易聊天。即时通讯中只有 Google Gtalk 和 Fackbook Chat 较出名，但都没有作为这两家企业的核心产品作为推广。 XMPP 的应用中并没有旗舰应用。

XMPP 的缺点归根结底是因为其已经成为开放标准，制定和修改要顾及多方的利益。其核心栈只能包括各种应用的交集部分。各厂商对 XMPP 的利用多会建立一套新的扩展协议以扩展功能，如 Google 用于文件和语音流传输的 Jingle 协议和完全和其他 XMPP 应用不流通的 Google Wave。

但总的来说，XMPP 的问题是一个开放即时通讯协议不可避免遇到的问题。研究 XMPP 协议的实用效果，对将来开放更好的开放协议有重要的参考意义。

**2. XMPP 基础**

**2.1 网络层次和数据包**

XMPP 使用 TCP 连接，并支持安全传输 (TSL/SASL)。

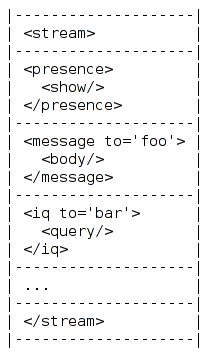
XMPP 的层次结构如下

XMPP  
SASL  
TSL  
TCP

XMPP 层中传输的数据包采用 XML 格式，称为 XML stanzas，XMPP 节点间 XML stanzas 的传输构成的数据流称为 XML stream。

一个 XML stream 的概览[2]如下

图一 XML Stream

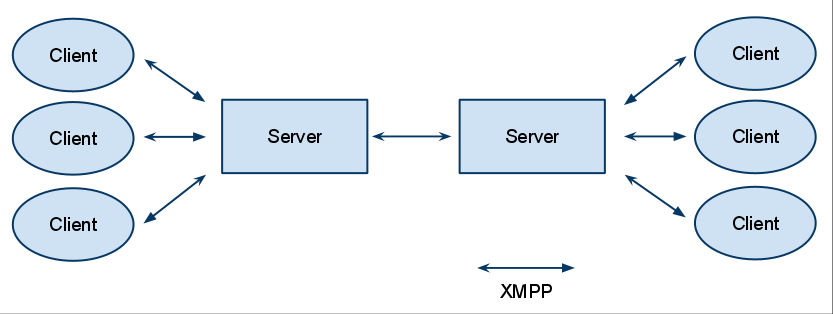


XMPP 核心栈中，XML stanzas 包括 Presence、Message 和 Iq stanzas。Presence 用于传输节点状态，Message 用于传输信息内容，而 Iq 用于传输更复杂的应答。实际应用中，Presence 多限定用于简单的状态传输，而扩展协议多通过扩展 Iq stanzas 元素实现。

**2.2 XMPP 的节点与路由**

XMPP 中的节点大致有两种，一种是服务器，一种是客户端(暂不讨论各种代理)。客户端需要连接服务器，服务器为客户端提供数据包 (XML stanzas) 的路由和转发。

图二 XMPP Server-Client 节点



客户端与客户端的通信需要通过服务器中转。每个到达服务器的数据包，由服务器分析后发往别的服务器或客户端。与 Email 的通信机制不同的是，XMPP 服务器之间不设立中转点，而是直接连接，以提高即时性和安全性。

**2.3 地址标识**

每个客户端需要拥有一个地址标识用于定位，XMPP 中称之为 JID (Jabber ID)。地址标识的格式如下

[ node "@" ] domain [ "/" resource ]

例如

username@gmail.com/pidgin

格式与 Email 地址格式类似，但增添了 resource 项，用于支持同一账号的多客户端登录。上述例子可以解释为：在 gmail.com 服务器注册的 username 用户，且使用 pidgin 客户端登录。当一个 JID 不包含 resource 部分时，该 JID 一般称为 BareJID。

用户地址标识的认证由提供 XMPP 服务的服务器执行。例如，注册于 gmail 服务器的账号由 gmail 服务器进行验证。其他服务器发往 gmail.com 域名的数据包均通过域名查询与服务间验证后发往 gmail 服务器，而不用考虑 gmail 服务器与下属账号间的通信。

2010年05月10日 9:43 下午

# [XMPP 协议笔记 二 XML数据包](http://chloerei.com/2010/05/10/265)

by [Rei](http://chloerei.com/author/rei/)    in [技术](http://chloerei.com/category/%e6%8a%80%e6%9c%af)

### 3. XMPP 核心数据包

XMPP 的核心数据包类型有Precense，Message，Iq ，此外加上初始化 stream 用到的 Stream 数据包。这些数据包是 XMPP 信息传输的载体，被用于 XMPP 核心功能和扩展功能的实现。

该部分仅对 XMPP 中使用的数据包进行概览，用于感受基于 XML 的数据包与其他非 XML 数据包协议的差别，不能替代 IETF 关于 XMPP 协议的 RFC 文档 [[3920](http://tools.ietf.org/html/rfc3920)] [[3921](http://tools.ietf.org/html/rfc3921)][3][4]，以及 XMPP 的扩展协议文档 [[extensions](http://xmpp.org/extensions/)][5] 中描述。

#### 3.1 公有属性

在 XML stream 中，每个数据包都是 XML 格式纯文本。而每个 XML 数据包有以下公有属性：

* to 数据包要发送的目的地址
* from 数据包发送的源地址
* id 数据包标示符

此三项属性在 XML stanza 中最为常见。

to 和 from 属性用于服务器决定该数据包的路由规则。某些情况下，to 和 from 属性可以只有一个，例如：客户端向服务端发送设置配置的 Iq 包只含有 to (不向外路由)，客户端向联系人发送 Message 只含有 to (from 属性总是被改写为客户端的地址)。

id 用于节点间判断请求和应答数据包的对应状况，大多数情况可以不处理。

#### 3.2 初始化 XML stream，身份验证

在客户端与服务器产生 TCP 连接后，需要与服务器初始化 XML stream，以及进行身份验证。

初始化时，客户端发送 stream 头部 XML：

**<?xml** version='1.0'**?>**

**<stream:stream**

   to='example.com'

   xmlns='jabber:client'

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

   version='1.0'**>**

服务器在收到客户端的 stream 头后，回应一个 stream 头：

**<?xml** version='1.0'**?>**

**<stream:stream**

   from='example.com'

   id='someid'

   xmlns='jabber:client'

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

   version='1.0'**>**

接着服务器向客户端发送服务端支持的身份验证方式列表，常见的方式有基于安全传输 SASL 的 BASE64 编码账户密码验证。身份验证的种类多样，且过程较为繁琐，可以参考 《XMPP: The Definitive Guide》[第12章](http://books.google.com/books?id=SG3jayrd41cC&lpg=PP1&pg=PA165#v=onepage&q&f=false)的介绍。

在对话结束时，客户端和服务端要先后发送 stream 尾部 XML，以使整个 XMP stream 闭合。(如果 TCP 异常中断，则服务端直接中断对话)

客户端：

**</stream:stream>**

服务端：

**</stream:stream>**

#### 3.3 Roster 获取联系人列表

在即时聊天 (IM) 应用中，客户端登录服务器后做的第一个操作通常是获取联系人列表。获取联系人列表需要发送 get 类型的 Iq 数据包。(Iq数据包将会在3.6节解释)

客户端：

**<iq** from='juliet@example.com/balcony' type='get' id='roster\_1'**>**

**<query** xmlns='jabber:iq:roster'**/>**

**</iq>**

该请求的意义为：名为 juliet 的用户 (登录资源为 balcony) 向 example.com 服务器请求获得 (get) roster 表。

服务器收到请求后，返回 roster 表。

服务端：

**<iq** to='juliet@example.com/balcony' type='result' id='roster\_1'**>**

**<query** xmlns='jabber:iq:roster'**>**

**<item** jid='romeo@example.net'

name='Romeo'

subscription='both'**>**

**<group>**Friends**</group>**

**</item>**

**<item** jid='mercutio@example.org'

name='Mercutio'

subscription='from'**>**

**<group>**Friends**</group>**

**</item>**

**<item** jid='benvolio@example.org'

name='Benvolio'

subscription='both'**>**

**<group>**Friends**</group>**

**</item>**

**</query>**

**</iq>**

可以看到，juliet 的 roster 表内有3个联系人，分别名为 Romeo，Mercutio，Benvolio，都属于 Friends 分组。Roster 列表中的 JID 信息将会用在稍候客户端发送信息包的目的地址中。

Item 中的 subscription 关系到联系人状态信息的传输，有 none，both，from，to 四种。详细的 subscription 操作在 RFC 3921 [Managing Subscriptions](http://tools.ietf.org/html/rfc3921#page-26) 章节[7]中定义。

#### 3.4 Presence 状态数据包

Presence 数据包被设计用来发送轻量级的节点状态信息，例如 IM 中用来发送用户的在线、离开、忙碌、离线等简单的状态消息。

一个 presence 数据包范例如下：

客户端：

**<presence>**

**<show>**away**</show>**

**</presence>**

该 presence 向服务端表明，客户端用户处于离线状态。服务端收到 presence 后，会自行填充 from 和 to 属性，发送到订阅 (subscribeb) 了该用户状态信息的联系人服务器上。

Presence 数据包也可以在发送时填上 to 属性，用于指定 presence 的接收方。例如：

客户端：

**<presence** to="romeo@example.net"**>**

**<show>**away**</show>**

**</presence>**

该 Presence 只会被转发至 romeo@example.net。

#### 3.5 Message 信息数据包

Message 是即时聊天应用中最常用的数据包，其功能是发送用户聊天信息。一个 Message 例子如下：

客户端：

**<message**

    to='romeo@example.net'

    from='juliet@example.com/balcony'

    type='chat'

    xml:lang='en'**>**

**<body>**Wherefore art thou, Romeo?**</body>**

**</message>**

该 message 包将会被服务器转发至 example.net 服务器，随后转交给 romeo 已登录的客户端上(如果该用户没有登录，message 信息会储存在服务端直至用户上线)。

其中，body 标签中包含用户要传输的聊天信息。

要传输格式化的富文本信息，可以通过支持扩展 [XEP-0071](http://xmpp.org/extensions/xep-0071.html)[8]，引入 html 标签。

3.6 Iq 数据包

Iq 是一个为其他操作提供 get，result，set，error 动作的数据包，本身并没有限定用途的范围。核心协议中就有大量需要用到 Iq 数据包的操作，例如添加 roster 联系人，需要用到 set 类型的 Iq 数据包：

客户端：

**<iq** from='juliet@example.com/balcony' type='set' id='roster\_2'**>**

**<query** xmlns='jabber:iq:roster'**>**

**<item** jid='nurse@example.com'

       name='Nurse'**>**

**<group>**Servants**</group>**

**</item>**

**</query>**

**</iq>**

服务器用 result 类型的 Iq 数据包返回操作结果：

服务端：

**<iq** to='juliet@example.com/balcony' type='result' id='roster\_2'**/>**

XMPP 扩展协议大都通过扩展 Iq 数据包的元素来添加操作。Iq 数据包的 type 属性就像 HTTP 协议中的 Method 项一样，提供增删改查 (CURD) 动作，具体实现何种操作并不限制。

2010年05月12日 3:56 下午

# [XMPP 协议笔记 三 扩展](http://chloerei.com/2010/05/12/305)

by [Rei](http://chloerei.com/author/rei/)    in [技术](http://chloerei.com/category/%e6%8a%80%e6%9c%af)

### 4. XMPP 扩展

XMPP 的核心部分是一个轻量级的协议，不足以满足一般即时聊天应用的需求。 XMPP 的社区在核心协议的基础上定义了众多的扩展协议，用以实现电子名片、二进制传输等功能。

#### 4.1 通过 vcard-temp 获取电子名片

电子名片 (个人资料) 是聊天程序中常见的功能。通过电子名片，用户可以查看联系人除了 JID 地址外的其他信息，如：昵称，全名，个人网站 URL 等。

XMPP 的扩展之一 [[XEP-0054](http://xmpp.org/extensions/xep-0054.html)] vcard-temp [9] 实现了电子名片。要在聊天程序中支持电子名片，需要客户端和服务端都实现 vcard-temp 扩展。知名的 XMPP 服务器大都支持 vcard-temp 扩展，如 jabber.org， gtalk 等。

##### 4.1.1 请求电子名片

vcard-temp 扩展规定的电子名片访问规则如下：

客户端：

**<iq** from='stpeter@jabber.org/roundabout'

   id='v3'

   to='jer@jabber.org'

   type='get'**>**

**<vCard** xmlns='vcard-temp'**/>**

**</iq>**

该查询使用 get 类型的 Iq 包请求 jabber.org 服务器返回 jer 用户的 vcard 数据。

接着 jabber.org 服务器返回 jer 用户的 vcard 数据：

服务器：

**<iq** from='jer@jabber.org'

   to='stpeter@jabber.org/roundabout'

   type='result'

   id='v3'**>**

**<vCard** xmlns='vcard-temp'**>**

**<FN>**JeremieMiller**</FN>**

**<N>**

**<GIVEN>**Jeremie**</GIVEN>**

**<FAMILY>**Miller**</FAMILY>**

**<MIDDLE/>**

**</N>**

**<NICKNAME>**jer**</NICKNAME>**

**<EMAIL><INTERNET/><PREF/><USERID>**jeremie@jabber.org**</USERID></EMAIL>**

**<JABBERID>**jer@jabber.org**</JABBERID>**

**</vCard>**

**</iq>**

vcard-temp 中定义了大量数据项，以上的示例只返回了用户已设置的部分数据项。

##### 4.1.2 隐私问题

目前为止，vcard-temp 扩展被定义为无须经过授权访问，即非好友用户也可以获取任何一个 XMPP 用户的所有 vcard 信息。因此，vcard-temp 扩展建议用户慎重考虑是否要填入个人地址等隐私信息。

#### 4.2 通过 In-Band Bytestreams 传输二进制数据

XMPP 核心部分没有提供二进制数据传输的支持，而 XMPP 扩展中有若干提供二进制传输的协议。应用较广的有 [[XEP-0047](http://xmpp.org/extensions/xep-0047.html)] In-Band Bytestreams [10] 和 [[XEP-0065](http://xmpp.org/extensions/xep-0065.html)] SOCKS5 Bytestreams [11]。由于实际上被广为应用二进制传输协议就只有这两种，所以客户端能做的选择不多。In-Band Bytestreams 和 SOCKS5 Bytestreams 可被整合为 [[XEP-0096](http://xmpp.org/extensions/xep-0096.html)] SI File Transfer [12]，这个协议允许接收端选择传输方式。

此外还有 Google 提出的 Jingle 传输协议，Jingle 支持的网络连接方式更多，目前被应用于 Gtalk 客户端上，用于支持二进制文件传输与语音通信的实现，但并未普及。

##### 4.2.1 In-Band Bytestreams 原理

所谓 In-Band ，即为 “在 XMPP 流内”的意思，其完整意义为：将二进制数据置入 XMPP 流内传输。但 XMPP 流的内容是纯文本，如何传输二进制数据呢？In-Band Bytestreams 的方案是把二进制数据使用 Base64 编码转换成可打印的纯文本，然后当作纯文本塞入 Iq 或者 Message 数据包中传输。

##### 4.2.2 In-Band Bytestreams 示例

In-Band Bytestreams 的流程包括发起、接受/拒绝，分包传输，结束应答这几个过程，详细的流程示例参照 [XEP-0047] [10] 中的定义。

一个发送 In-Band Bytestreams 的 Iq 数据包示例如下：

**<iq** from='romeo@montague.net/orchard'

    id='kr91n475'

    to='juliet@capulet.com/balcony'

    type='set'**>**

**<data** xmlns='http://jabber.org/protocol/ibb' seq='0' sid='i781hf64'**>**

    qANQR1DBwU4DX7jmYZnncmUQB/9KuKBddzQH+tZ1ZywKK0yHKnq57kWq+RFtQdCJ

    WpdWpR0uQsuJe7+vh3NWn59/gTc5MDlX8dS9p0ovStmNcyLhxVgmqS8ZKhsblVeu

    IpQ0JgavABqibJolc3BKrVtVV1igKiX/N7Pi8RtY1K18toaMDhdEfhBRzO/XB0+P

    AQhYlRjNacGcslkhXqNjK5Va4tuOAPy2n1Q8UUrHbUd0g+xJ9Bm0G0LZXyvCWyKH

    kuNEHFQiLuCY6Iv0myq6iX6tjuHehZlFSh80b5BVV9tNLwNR5Eqz1klxMhoghJOA

**</data>**

**</iq>**

其中，data 段内包含的字符串是经过 Base64 编码的二进制数据。可见，该二进制数据经过编码后被当作一般纯文本来传输。data 段的 seq 属性标明了此段数据在所有数据分包中的编号。

##### 4.2.3 In-Band Bytestreams 缺陷

In-Band Bytestreams 借助 XMPP 流进行二进制传输，优点是可以稳定交付，只要双方客户端仍然连接各自的服务器就能实现二进制传输。但其重大缺陷是速率过慢，每段二进制数据都要经过分包和多次封包解包，并且每个数据包都通过两个服务器的交付处理，会产生很大的延迟，效率很低。后文有对 In-Band Bytestreams 的简单速率测试。\*未完成\*

#### 4.3 通过 SOCKS5 Bytestreams 传输二进制数据

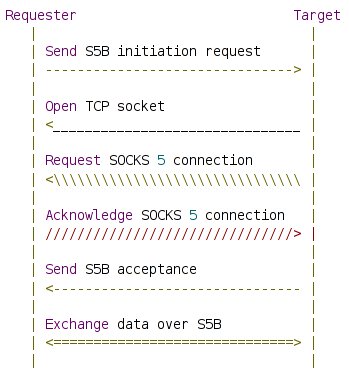
4.3.1 SOCKS5 Bytestreams 原理

有别于 In-Band Bytestreams 流内传输，SOCKS5 Bytestreams 方案使用双方客户端的 socks5 直连传输，这样可以使传输速率达到双方主机直连的速度。

SOCKS5 Bytestreams 在传输过程的起效部分仅是让双方客户端决定传输方案，交换 IP 地址，端口地址等信息。当双方客户端决定可以使用的 IP 地址和端口地址后，就发起 socks5 连接，此时 XMPP 流不再参与数据传输，不增加额外的传输延迟。

整个 SOCKS5 Bytestreams (S5B) 流程如图三所示：

图三 XMPP 发起 SOCKS5 Bytestreams 流程



##### 4.3.2 SOCKS5 Bytestreams 示例

一个发起 SOCKS5 Bytestreams 的 Iq 数据包示例如下：

**<iq** from='requester@example.com/foo'

    id='hu3vax16'

    to='target@example.org/bar'

    type='set'**>**

**<query** xmlns='http://jabber.org/protocol/bytestreams'

         sid='vxf9n471bn46'**>**

**<streamhost**

        jid='requester@example.com/foo'

        host='192.168.4.1'

        port='5086'**/>**

**</query>**

**</iq>**

其中 streamhost 包含了发起人可用的 IP 地址，这个 IP 地址可以是主机地址，也可以是 socks5 代理主机地址 (用于突破防火墙和 NAT 网络的限制)。streamhost 项可以有多个，最终由接收方选择可用的地址，然后发起连接。

##### 4.3.4 SOCKS5 Bytestreams 局限

由于互联网连接的复杂性，发起传输请求的 XMPP 客户端可能只拥有内网 IP ，并且没有可用的 socks5 代理服务器可用。这种情况下发起人将无法实现二进制文件的传输。

并且即使有代理服务器，由于经过中转，传输速率可能会大大降低。对此，除了假设更好的 socks5 代理服务器外，没有更好的方法。

#### 4.4 扩展机制的缺点

XMPP 的扩展协议相当多，详细列表可以在 http://xmpp.org/extensions/ 找到。

但是通过扩展协议实现的功能，通常需要双方客户端均实现才能使用。有些还需要服务器做额外支持 (如 vcard-temp)，而有名的 XMPP 服务器通常只会选择性实现扩展协议，所以要在现实中实现所需的 XMPP 扩展，就需要搭建全套的服务端、客户端方案，开发成本大。并且造成各方 XMPP 服务器实现不一致，出现功能不同的 XMPP “部落”，只能使用核心协议中已定义的部分进行通信。