# RFC3921

## 出自 Jabber/XMPP 中文翻译计划

跳转到: 导航, 搜索

本文的英文原文来自 RFC 3921

网络工作组　　Saint-Andre, Ed.

申请讨论: 3921 Jabber 软件基金会

类别: 标准跟踪 2004 年 10 月

**可扩展的消息和出席信息协议 (XMPP): 即时消息和出席信息**

## 关于本文的说明

本文为互联网社区定义了一个互联网标准跟踪协议，并且申请讨论协议和提出了改进的建议。请参照“互联网官方协议标准”的最新版本（STD 1）获得这个协议的标准化进程和状态。本文可以不受限制的分发。

## 版权声明



## 摘要

本文定义了可扩展消息和出席信息协议（XMPP）的核心功能的扩展和应用，XMPP 提供了 RFC 2779 定义的基本的即时消息和出席信息功能。

# 目录

[隐藏]

# 绪论

## 概览

XMPP 是一个流化 XML[XML]元素的协议，用于准实时的交换消息和出席信息。XMPP 的核心功能定义在 Extensible Messaging and Presence Protocol (XMPP): Core [XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]. 这些功能 -- 主要是

XML 流，使用 TLS 和 SASL,以及流的根元素之下的<message/>, <presence/>, 和 <iq/> 子元素 -- 为各种类型的准实时应用提供了一个构造基础，它可以被放在核心的顶层，使用特定 XML 名字空间[XML-NAMES]发送特定的应用数据. 本文描述 XMPP 核心功能的扩展和应用，XMPP 核心功能提供了 RFC 2779 [IMP-REQS]定义的基本的即时消息和出席信息功能。

## 需求

为了达到本文的目的，基本的即时消息和出席信息应用的需求定义在[IMP-REQS],它是一个高阶的规定,一个用户必须完成以下用例:

- 和其他用户交换消息
- 和其他用户交换出席信息
- 管理和其他用户之间的订阅和被订阅
- 管理联系人列表中的条目(在 XMPP 中这被称为 "roster")
- 屏蔽和特定的其他用户之间的通信（出或入）

这些功能领域的详细定义在[IMP-REQS]中，感兴趣的用户可以直接阅读原文关于需求方面的内容。
[IMP-REQS]也规定出席信息服务必须从即时消息服务中分离；例如，它必须可能用这个协议来提供一个出席信息服务,一个即时消息服务,或同时提供两者. 尽管本文假定实现和部署希望提供统一的即时消息和出席信息服务，但没有要求一个服务必须同时提供出席信息服务和即时消息服务，并且协议也提供了把出席信息服务和即时消息服务分离成为独立服务的可能性.
注意: 虽然基于 XMPP 的即时消息和出席信息符合[IMP-REQS]的要求,但它不是特意为那个协议设计的,因为基础协议是在 RFC 2779 成文之前通过 Jabber 开放源代码社区的一个开放的开发过程发展出来的. 也请注意尽管在 Jabber 社区发展的协议中定义了许多其他方面的功能,但是这些协议不包含在本文之中,因为它们不是[IMP-REQS]所要求的.

## 术语

本文继承了 [XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]定义的术语. 大写关键字 "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", 和 "OPTIONAL" 在本文中的含义定义在 BCP 14, RFC 2119 \[TERMS\].

# XML 节的语法

符合'jabber:client'和'jabber:server'名字空间的 XML 节的基本语义和通用属性已经在[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]中定义了. 无论如何，这些名字空间也定义了 yixie 其他的子元素，比如通用属性'type'的值，对于即时消息和

出席信息应用就是特殊的. 因而, 在选择用于这类应用的特定用例之前, 我们在这里需要先描述一下 XML 节的语法, 用来补充[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]中的讨论.

## 消息语法

符合'jabber:client' or 'jabber:server'名字空间的消息节用于"推" 信息到另一个实体. 在即时消息应用中通常的用法是包含，一个单独的消息,在一个聊天会话中的消息，一个多用户聊天室的上下文中的消息，标题或其他警告和错误的消息,

## 消息的类型

一个消息节的'type' 属性是建议的(RECOMMENDED); 如果包含了它,它指明这个消息的会话上下文,从而提供一个关于表达的线索(例如, 在一个 GUI 中). 如果包含了它, 'type' 属性必须(MUST)是以下的值之一 :

- chat -- 消息是在一对一聊天会话的语境被发送. 一个兼容的客户端应该(SHOULD)在一个允许两个实体进行一对一聊天的界面中显示消息,包括适当的会话历史.

- error -- 发生了一个和上次发送者发送的消息有关的错误(关于节错误语法的详细信息, 参考 [XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]). 一个兼容客户端应该(SHOULD)在一个适当的界面展示它以通知发送者这个错误的种类.

- groupchat -- 消息是在一个多用户聊天环境的语境下发送的(类似\[IRC\]). 一个兼容客户端应该(SHOULD)在允许多对多聊天的界面显示这个消息,包括, 包括这个聊天室的名册和适当的会话历史. 基于 XMPP 的群聊协议的完整定义超出了本文的范围.

- headline -- 一个消息可能是由一个递送或广播内容的自动化服务生成的(新闻, 体育, 市场信息, RSS feeds, 等等.). 这个消息是不需要回复的, 一个兼容客户端应该(SHOULD) 在一个适当的和单独消息,聊天会话,或群聊会话不同的界面显示这个消息(例如, 不给接收者提供回复能力).

- normal -- 这个消息是一个在一对一会话或群聊会话之外的单独消息, 并且它希望接收者能够回复.一个兼容客户端应该(SHOULD)在一个允许接收者回复的界面显示这个消息, 但不需要会话历史.

一个 IM 应用应该(SHOULD)支持所有前述的消息类型;如果一个应用接收了一个没有'type'属性的消息或这个应用不理解'type'属性的值, 它必须(MUST)认为这个消息是一个 "normal" 类型(如,"normal" 是缺省的). "error"类型必须(MUST)仅仅在应答一个和从别的实体接收到的消息有关的错误时生成.

尽管'type'属性是可选的(OPTIONAL), 处于礼貌原因对于消息的任何回复总是和原来的消息同一类型;此外, 一些特殊的应用(例如, 一个多用户聊天服务) 可以(MAY)根据它们的判断强制特定消息类型的使用(例如, type='groupchat').

## 子元素

正如 扩展名字空间 extended namespaces(第二章第四节)所述, 一个消息节可以(MAY)包含任何适当名字空间的子元素.
和缺省名字空间声明一致, 缺省消息节的名字空间是'jabber:client' 或'jabber:server', 定义了某几个允许的消息节的子元素. 如果消息节的类型是 "error", 它必须(MUST)包含一个<error/>子元素; 详细情况, 见[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]. 否则, 消息节可以(MAY)包含以下子元素的任何一种并且无需显式地声明名字空间:

1. <subject/>
2. <body/>
3. <thread/>

## 主题

<subject/> 元素包含了人类可读的 XML 字符数据指明这个消息的主题. <subject/>元素不能(MUST NOT)拥有任何属性, 除了'xml:lang'属性. <subject/> 元素可以(MAY)包含多个实例用于为同一主题提供备用版本, 但是仅在每个实例的拥有的'xml:lang'属性的值互不相同的时候才可以. <subject/> 元素不能(MUST NOT)包含混合的内容(定义在 \[XML\]第三章第二节第二小节).

## 主体

<body/> 元素包含人类可读的 XML 字符数据表达消息的文本内容; 这个子元素通常会有但是是可选的(OPTIONAL). <body/>元素不能(MUST NOT)拥有任何属性, 除非是'xml:lang'属性. <body/> 元素可以(MAY)包含多个实例用于为同一主体提供备用版本, 但是仅在每个实例的拥有的'xml:lang'属性的值互不相同的时候才可以. <body/>元素不能(MUST NOT)包含混合的内容(定义在 \[XML\]第三章第二节第二小节).

## 线索

<thread/> 元素包含非人类可读的 XML 字符数据表达一个标识符用于跟踪两个实体之间的一个会话线索(有时相当于一个"即时消息会话"). <thread/>元素的值是由发送者生成的并且应该(SHOULD)在任何回复中拷贝回来. 如果使用了它, 它必须(MUST)在这个流的会话线索中是唯一的并且必须(MUST)和那个会话相一致(一个从同一个全 JID 但不同线索 ID 接收到消息的客户端必须(MUST)假定这个有问题的消息存在于

已有的会话线索之外. <thread/>元素的使用是可选的(OPTIONAL)并且不是用于标识独立的消息,而是标识会话. 一个消息节不能(MUST NOT)包含超过一个的<thread/>元素. <thread/>元素不能(MUST NOT)拥有任何属性. <thread/>属性的值必须(MUST)被实体处理成不透明的; 不能从它得到任何语义学上的含义,并且只能对它做精确的比较. <thread/>元素不能(MUST NOT)包含混合内容(定义在 [XML]第三章第二节第二小节).

## 出席信息语法

符合'jabber:client' 或 'jabber:server'名字空间的出席信息节用于表达一个实体当前的网络可用性(离线或在线, 包括之后的各种亚状态和可选的用户名义的描述性文本), 并且通知其他实体它的可用性. 出席信息节也用于协商和管理对于其他实体的出席信息的订阅.

## 出席信息的类型

出席信息节的'type'属性是可选的(OPTIONAL). 一个不拥有任何'type'属性的出席信息节用来通知服务器发送者已经在线并且可以进行通信了, 'type' 属性表示缺乏可用性, 请求管理对其他实体的出席信息的订阅, 请求其他实体的当前出席信息, 或发生了和上次发出的出席信息节有关的错误. 如果包含了它, 'type'属性必须(MUST)拥有以下值之一:

- unavailable -- 通知实体将不可通信.
- subscribe -- 发送者希望订阅接收者的出席信息.
- subscribed -- 发送者允许接收者接收他们的出席信息.
- unsubscribe -- 发送者取消订阅另一个实体的出席信息.
- unsubscribed -- 订阅者的请求被拒绝或以前的订阅被取消.
- probe -- 对一个实体当前的出席信息的请求; 只应(SHOULD)由服务器代替一个用户生成.
- error -- 处理或递送之前发送的出席信息节的时候发生了错误.

关于出席信息语义学的详细信息和基于 XMPP 的即时消息和出席信息应用程序的订阅模式,参考 交换出席信息 Exchanging Presence Information(第五章) 和 管理订阅 Managing Subscriptions(第六章).

## 子元素

如 扩展名字空间 extended namespaces(第二章第四节)所述, 一个出席信息节可以(MAY)包含任何适当名字空间的子元素.
和缺省名字空间声明一致, 缺省出席信息节的名字空间是'jabber:client' 或'jabber:server', 定义了某几个允许的出席信息节的子元素. 如果出席信息节的类型是 "error", 它必须(MUST)包含一个<error/>子元素; 详细情况, 见[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]. 如果出席信息节不拥有'type'属性,它可以(MAY)包含以下任何子元素(注意<status/>子元素可以(MAY)在一个类型为"unavailable"或"subscribe"(出于历史原因)的出席信息中被发送):

1. <show/>
2. <status/>
3. <priority/>

## 展示

可选的(OPTIONAL)<show/>元素包含非人类可读的 XML 字符数据表达一个特定的实体或资源的特定的可用性状态. 一个出席信息节不能(MUST NOT)包含多于一个<show/>元素. <show/>元素不能(MUST NOT)拥有任何属性. 如果提供了, 这个 XML 字符数据值必须(MUST)是以下之一(额外的可用性类型可以通过出席信息的适当名字空间来定义):

- away -- 实体或资源临时离开.
- chat -- 实体或资源在聊天中是激活的.
- dnd -- 实体或资源是忙(dnd = "不要打扰").
- xa -- 实体或资源是长时间的离开(xa = "长时间离开").

如果没有提供<show/>元素, 实体被假定是在线和可用的.

## 状态

可选的(OPTIONAL)<status/>元素包含 XML 字符数据表达一个可用性状态的自然语言描述. 它通常用于联合 show 元素以提供可用性状态的详细描述(例如, "会议中"). <status/>元素不能(MUST NOT)拥有任何属性,除了'xml:lang'属性. <status/>元素可以(MAY)包含多个实例但是每个实例的'xml:lang'属性值必须各不相同.

## 优先权

可选的(OPTIONAL)<priority/>元素包含非人类可读的 XML 字符数据指明资源的优先级别. 这个值必须(MUST)是一个介于-128 和+127 之间的数字. 一个出席信息小节不能(MUST NOT)包含超过一个的<priority/>元素. <priority/>元素不能(MUST NOT)拥有任何属性. 如果没有优先权被提供,一个服务器应该(SHOULD)认为优先级是零. 关于即时消息和出席信息系统中节路由的优先级的语义, 参考 处理 XML 节的服务器规则 Server Rules for Handling XML Stanzas(第十一章).

## IQ 语法

IQ 节提供一个结构化的请求-应答机制. 这个机制的基本语义学(例如, 'id'属性是必需的(REQUIRED))定义在[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920], 然而完成特定用例所需要的特定语义的所有案例定义在扩展名字空间 extended namespace(第二章第四节)之中(注意'jabber:client'和'jabber:server'名字空间没有定义除通用的<error/>子元素之外的任何 IQ 节子元素). 本文定义了两个这样的名字空间, 一个用于 名册管理 Roster Management(第七章)而另一个用于 屏蔽通信 Blocking Communication(第十章); 无论如何, 一个 IQ 节可以(MAY)包含符合任何扩展名字空间的结构化信息.

## 扩展名字空间

因为在"jabber:client"或"jabber:server"名字空间中定义的三个 XML 节类型(也包括它们的属性和子元素)提供了一个基本功能级用于消息和出席信息, XMPP 使用 XML 名字空间来扩展节用于提供额外的功能, 所以一个消息或出席信息节可以(MAY)包含一个或更多可选的子元素表达扩展消息含义的内容(例如, 一个 XHTML 格式版本的消息主体), 并且一个 IQ 节可以(MAY)包含一个这样的子元素. 这个子元素可以(MAY)有任何名字并且可以(MUST)拥有一个'xmlns'名字空间声明(不同于"jabber:client", "jabber:server", 或"http://etherx.jabber.org/streams")定义所有包含在子元素中的数据.

对于任何特定的扩展名字空间的支持在任何实现中的一部分是可选的(OPTIONAL)(除了在这里定义的扩展名字空间以外). 如果一个实体不理解这样一个名字空间, 实体被期望的行为依赖于这个实体是(1) 接收者 或 (2) 一个正在路由到接收者的实体

接收者: 如果一个接收者接收了一个包含不理解的子元素的节, 它应该(SHOULD)忽略那个特定的 XML 数据,例如, 它应该(SHOULD)不处理它或不向用户或相关的应用程序(如果有的话)显示它. 具体来说:

- 如果一个实体接收了一个消息或出席信息节包含一个不理解的名字空间, 在节的未知名字空间的这部分应该(SHOULD)被忽略.

- 如果一个实体接收了一个消息节中仅有的一个子元素是不理解的，它必须(MUST)忽略整个节.

- 如果一个实体接收了一个类型"get"或"set"的 IQ 节包含一个不理解的子元素，这个实体应该(SHOULD)返回一个类型为"error"的<service-unavailable/>错误条件的 IQ 节.

路由：如果一个路由实体(通常是一个服务器)处理一个包含它不理解的子元素的节，它应该(SHOULD)原封不动地把它转给接收者而忽略相关的 XML 数据.

# 会话的建立

绝大部分基于 XMPP 的消息和出席信息应用是由一个客户端-服务器体系结构实现的，为了参加期望的即时消息和出席信息活动,需要客户端在服务器上建立一个会话. 无论如何，在客户端能够建立一个即时消息和出席信息会话之前有很多前提必须(MUST)满足. 它们是:

1. 流验证 -- 客户端在尝试建立一个会话或发送任何 XML 节之前必须(MUST)完成[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]中定义的流验证.
2. 资源绑定 -- 完成流验证之后，一个客户端必须(MUST)绑定一个资源到流上，使得客户端的地址符合<user@domain/resource>格式，然后实体以[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]规定的术语来说就是一个 已连接的资源"connected resource".

如果一个服务器支持会话，在完成一个[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]定义的流验证之后它必须(MUST)在它向客户端声明的流特性中包含一个符合'urn:ietf:params:xml:ns:xmpp-session'名字空间的<session/>元素:
服务器向客户端声明会话确定特性:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    id='c2s_345'
    from='example.com'
    version='1.0'>
<stream:features>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
  <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
</stream:features>
```

收到需要会话确立的通知之后(并且是在完成资源绑定之后)，客户端如果想使用即时消息和出席信息功能必须(MUST)建立一个会话；它向服务器发送一个符合'urn:ietf:params:xml:ns:xmpp-session'名字空间的类型为"set"并包含空的<session/>子元素的 IQ 节以完成这一步骤:
步骤 1: 客户端向服务器请求会话:

```
<iq to='example.com'
      type='set'
      id='sess_1'>
  <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
</iq>
```

步骤 2: 服务器通知客户端会话已经建立:

```
<iq from='example.com'
      type='result'
      id='sess_1'/>
```

建立会话之后，一个 已连接的资源([XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]术语)就被称为一个 激活的资源"active resource".

许多错误条件是可能的. 例如，服务器可能遭遇一个内部条件阻碍了它建立会话，用户名或授权身份可能缺乏建立会话的许可，或同一个名字相关的这个资源 ID 已经有一个激活的资源.

如果服务器遭到一个内部条件阻碍了它建立会话，它必须(MUST)返回一个错误.

步骤 2 (替代): 服务器应答一个错误(内部服务器错误):

```
<iq from='example.com' type='error' id='sess_1'>
  <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
  <error type='wait'>
    <internal-server-error
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

如果用户名或资源不被允许建立一个会话，服务器必须(MUST)返回一个错误(例如，被禁止).

步骤 2 (替代): 服务器应答错误(用户名或资源不被允许建立一个会话):

```
<iq from='example.com' type='error' id='sess_1'>
  <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
  <error type='auth'>
    <forbidden
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

如果已经同一名字已经存在一个激活的资源,服务器必须(MUST) (1) 终止这个激活的资源并允许新请求的会话，或者 (2) 不允许新申请的会话并继续激活的资源. 服务器做哪一步取决于具体的实现，尽管建议的(RECOMMENDED)实现 情景 #1. 在情景 #1，服务器应该(SHOULD)发送一个<conflict/>流错误给激活的资源，终止用于这个激活的资源的 XML 流和相关的 TCP 连接，并返回一个类型为"result" 的 IQ 节(表示成功)给新申请的会话. 在 情景 #2，服务器应该(SHOULD)发送一个<conflict/>节错误给新申请的会话但是继续那个连接的 XML 流使得新申请的会话在发送另一个会话建立申请之前有机会协商出一个不冲突的资源 ID.

步骤 2 (替代): 服务器通知现有的激活的资源 资源冲突(情景 #1):

```
<stream:error>
  <conflict xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
```

```
</stream:error>
</stream:stream>
```

步骤 2 (替代): 服务器通知新申请的的会话资源冲突(情景 #2):

```
<iq from='example.com' type='error' id='sess_1'>
  <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
  <error type='cancel'>
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

建立一个会话之后，客户端应该(SHOULD)按以下描述来发送初始化出席信息并请求它的名册，尽管这些动作是可选的(OPTIONAL).

注意: 在允许建立即时消息和出席信息会话之前，一个服务器可能(MAY)需要先提供帐号．可能的提供帐号的方法包括由服务器管理员新建帐号以及使用'jabber:iq:register'名字空间进行带内帐号注册; 后一个方法超出了本文的范围，但是记录在\[JEP-0077\]（译者注：这个协议已改名为 XEP-0077），由 Jabber Software Foundation \[JSF\]发行（译者注：这个组织也已改名为 XSF）.

# 交换消息

交换消息是 XMPP 的一个基本用途并且随之而来的是一个用户生成一个发给另一个实体的消息节. 正如 用于处理 XML 节的服务器规则 Server Rules for Handling XML Stanzas(第十一章)中所定义的 ，发送者的服务器负责递送消息给预定的接收者(如果接收者在同一个服务器上)或路由消息给接收者的服务器(如果接收者在不同的服务器上).

关于消息节的语法和它们已定义的属性和子元素信息，参考 消息语法 Message Syntax(第二章第一节).

## 指明一个预定的接收者

一个即时消息客户端应该(SHOULD)通过提供一个 JID 或<message/>节中不同于发送者的'to'属性来指定一个消息的预定接收者. 如果这个消息是在回复之前接收到的消息,而接收到的消息是从 JID 格式为<user@domain/resource>(例如,在一个聊天会话的上下文中)实体发来的，这个回复消息的'to'地址的值应该(SHOULD)是<user@domain/resource>而不是<user@domain>,除非发送者知道(通过出席信息)预定的接收者的资源将不再可用. 如果消息是在任何现存的聊天会话或接收到的消息之外被发送的,'to'地址的值应该(SHOULD)格式为<user@domain>而不是<user@domain/resource>.

## 指定一个消息类型

大家知道，对于一个消息节来说拥有'type'属性(它的值代表了消息的会话上下文(参见 Type(第二章第一节第一小节)))是建议的(RECOMMENDED).

以下例子展示一个'type'属性的合法值:

例子: 一个已定义类型的消息:

```
<message
    to='romeo@example.net'
    from='juliet@example.com/balcony'
    type='chat'
    xml:lang='en'>
  <body>Wherefore art thou, Romeo?</body>
</message>
```

## 指定一个消息主体

一个消息节可以(MAY)(并且经常会)包含一个<body/>子元素,它的 XML 字符数据表达消息的主要含义(见 Body(第二章第一节第二小节第二小小节)).

例子: 一个带主体的消息:

```
<message
    to='romeo@example.net'
    from='juliet@example.com/balcony'
    type='chat'
    xml:lang='en'>
  <body>Wherefore art thou, Romeo?</body>
  <body xml:lang='cz'>Pro&#x010D;e&#x017D; jsi ty, Romeo?</body>
</message>
```

## 指定一个消息主题

一个消息节可以(MAY)包含一个或多个<subject/>子元素指明消息的主题(见 Subject(第二章第一节第二小节第一小小节)).

例子: 一个带主题的消息:

```
<message
    to='romeo@example.net'
    from='juliet@example.com/balcony'
    type='chat'
    xml:lang='en'>
  <subject>I implore you!</subject>
  <subject
      xml:lang='cz'>&#x00DA;p&#x011B;nliv&#x011B; prosim!</subject>
  <body>Wherefore art thou, Romeo?</body>
  <body xml:lang='cz'>Pro&#x010D;e&#x017D; jsi ty, Romeo?</body>
</message>
```

## 指定一个会话线索

一个消息可以(MAY)包含一个<thread/>子元素指定消息处于哪个会话线索，用于跟踪会话(见 Thread(第二章第一节第二小节第三小小节)).

例子: 一个带线索的会话:

```
<message
      to='romeo@example.net/orchard'
      from='juliet@example.com/balcony'
      type='chat'
      xml:lang='en'>
   <body>Art thou not Romeo, and a Montague?</body>
   <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
</message>
<message
      to='juliet@example.com/balcony'
      from='romeo@example.net/orchard'
      type='chat'
      xml:lang='en'>
   <body>Neither, fair saint, if either thee dislike.</body>
   <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
</message>
<message
      to='romeo@example.net/orchard'
      from='juliet@example.com/balcony'
      type='chat'
      xml:lang='en'>
   <body>How cam'st thou hither, tell me, and wherefore?</body>
   <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
</message>
```

# 交换出席信息

交换出席信息通过使用出席信息节直接和 XMPP 相关. 无论如何，我们看到在这里和消息处理形成一个对比:尽管一个客户端可以(MAY)通过包含一个'to'地址直接给另一个实体发送出席信息，通常出席信息通知(例如,不包含'type'的或类型为"unavailable"的并且没有'to'地址的出席信息节) 被客户端发送给它的服务器然后由服务器广播给任何订阅了发送实体的出席信息的实体(在 RFC 2778 \[IMP-MODEL\] 术语中，这些实体称为订阅者). 这个广播模式不适用于和订阅相关的节或类型为"error"的出席信息，而仅适用于以上定义的出席信息通知. (注意: 虽然出席信息可以(MAY)由一个自动化服务代替用户提供，通常它还是由用户的客户端提供.)

关于出席信息节的语法以及它们的已定义的属性和子元素的信息，参考[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920].

## 客户端和服务器出席信息职责

## 初始化出席信息

建立起一个会话之后，一个客户端应该(SHOULD)发送初始化出席信息给服务器来通知它的通信可用性.如这里定义的，初始化出席信息节 (1) 必须(MUST) 不拥有'to'地址(这表示它是由服务器代替客户端发送的广播) 并且 (2) 必须(MUST) 不拥有'type'属性(者表示拥护的可用性). 在发送初始化出席信息之后，一个激活的资源被称为可用的资源"available resource".
从一个客户端接收到初始化出席信息之后，如果这个用户没有一个或更多的已存在的可用资源(如果这个用户已经有一个或更多可用的资源，服务器明显不需要发送出席信息探测，因为它已经拥有需要的信息),用户的服务器必须(MUST)做以下的步骤:

1. 从用户的全 JID(例如,<user@example.com/resource>)发送出席信息探针(例如，'type'属性值为'probe'的出席信息节)给已被这个用户订阅了的所有联系人以确定它们是否可用; 这些联系人就是那些显示在用户的名册中的 JID 并且'subscription'属性值为"to"或"both"(注意: 用户的服务器不能(MUST NOT)发送出席信息探针给用户已经屏蔽入站出席信息通知的联系人，具体的描述在 屏蔽入站出席信息通知 Blocking Inbound Presence Notifications(第十章第十节).)
2. 从用户的全 JID (e.g.,<user@example.com/resource>)广播初始化出席信息给所有订阅了该用户的出席信息的联系人; 这些联系人就是那些显示在用户的名册中的 JID 并且'subscription'属性值为"from"或"both"(注意: 用户的服务器不能(MUST NOT)发送出席信息探针给用户已经屏蔽出站出席信息通知的联系人，具体的描述在 屏蔽出站出席信息通知 Blocking Outbound Presence Notifications(第十章第十一节).)

另外，用户的服务器必须(MUST)从用户的新的可用的资源向用户任何现存的可用的资源(如果有的话)广播初始化出席信息.
从用户接收到初始化出席信息之后，联系人的服务器必须(MUST)递送这个用户的出席信息节给所有联系人的可用资源相应的全 JID(<contact@example.org/resource>), 但是仅适用于用户在联系人名册中并且订阅状态为"to"或"both"并且联系人的纯 JID 或全 JID 没有被屏蔽入站出席信息通知(定义在 屏蔽入站出席信息通知 Blocking Inbound Presence Notifications(第十章第十节)).
如果用户的服务器接收到一个类型为"error"的出席信息节,而这个节是用来回复服务器代替用户向联系人发送的初始化出席信息，它不应该(SHOULD NOT)发送更多的出席信息更新给那个联系人(直到并且除非它从这个联系人接收到一个出席信息节).

## 出席信息广播

发送初始化出席信息之后，用户可以(MAY)在任何时候更新它的出席信息,方法是在会话期间发送一个没有'to'地址也没有'type'属性的出席信息节或'type'属性值为"unavailable"的出席信息节.(注意:一个用户的客户端不应该(SHOULD NOT)发送一个出席信息更新来自行广播用户出席信息和可用性的改变信息.)
如果出席信息节缺乏'type'属性(例如，表达可用性)，用户的服务器必须(MUST)广播

那个出席信息节的全 XML 给所有联系人(满足以下三点) (1) 它们在用户的联系人名册中并且订阅类型是"from"或"both", (2) 用户对于这些联系人没有屏蔽出站出席信息通知, 并且 (3) 服务器在用户的会话期间没有从它们那里接收到出席信息错误(同样适用于这个用户的其他可用的资源).
如果出席信息节的'type'属性值是"unavailable", 用户的服务器必须(MUST)广播那个出席信息节的全 XML 给所有符合以上描述的实体, 也适用于用户曾经在会话过程中直接发送了可用出席信息的任何实体(如果用户还没来得及直接发送不可用出席信息给那个实体).

## 出席信息调查

从用户接收到一个出席信息调查之后, 联系人的服务器应该(SHOULD)应答如下:

1. 如果用户联系人的名册中的状态不是 "From", "From + Pending Out", 或 "Both" (定义在 订阅状态 Subscription States(第九章)), 联系人的服务器必须(MUST)返回一个类型为"error"的出席信息节应答这个出席信息调查 (无论如何, 如果一个服务器从这个服务器的主机名的子域或其他信任的服务接收到一个出席信息调查, 它可以(MAY)提供这个用户的出席信息给那个实体). 具体来说:
    1. 如果用户在联系人的名册中的订阅状态是 "None", "None + Pending Out", 或 "To" (或根本不在联系人的名册中), 联系人的服务器必须(MUST)返回一个<forbidden/>节错误应答这个出席信息调查.
    2. 如果用户在联系人的名册中的订阅状态是 "None + Pending In", "None + Pending Out/In", 或 "To + Pending In", 联系人的服务器必须(MUST)返回一个<not-authorized/>节错误应答这个出席信息调查.
2. 其次, 如果联系人对这个用户的纯 JID 或全 JID 屏蔽了出席信息通知(使用缺省列表或激活列表,定义在 屏蔽出站出席信息通知 Blocking Outbound Presence Notifications (第十章第十一节)), 服务器不能(MUST NOT)应答这个出席信息调查.
3. 然后, 如果联系人没有可用的资源, 服务器必须(MUST) 要么 (1) 应答这个出席信息调查, 向这个用户发送服务器从联系人接收到的最后的类型为"unavailable"的出席信息节的全 XML, 或 (2) 不应答.
4. 最后, 如果联系人至少有一个可用的资源, 服务器必须(MUST)应答这个出席信息调查, 向这个用户发送服务器从联系人的每一个可用的资源收到的最后的没有'to'属性的出席信息节的全 XML (再一次的,对于每一个会话都要强制服从隐私列表).

## 直接出席信息

一个用户可以(MAY)直接发送出席信息给另一个实体 (例如, 一个出席信息节,包含'to'属性并且值为另一个实体的 JID 并且没有'type'属性或'type'属性值为"unavailable"). 可能出现三种情形:

1. 如果用户在已经发送过初始化出席信息广播之后,发送不可用信息广播之前,直接发送出席信息给它的名册中一个订阅状态为"from" 或 "both"的联系人,这个用户的服务器必须(MUST)路由或递送这个出席信息节的全 XML(服从隐私列表)但是不应该(SHOULD NOT) 根据出席信息广播修改联系人的状态(例如, 它应该(SHOULD)在任何接下来的由用户初始化的出席信息广播包含这个联系人的 JID).
2. 如果用户在已经发送过初始化出席信息广播之后,发送不可用信息广播之前,直接发送出席信息给一个不在用户名册中的实体并且其订阅状态为"from" 或 "both", 这个用户的服务器必须(MUST)路由或递送这个出席信息节的全 XML(服从隐私列表)但是不能(MUST NOT) 根据可用性的出席信息广播来修改这个联系人的状态(例如, 它不能(MUST NOT)在任何接下来的由用户初始化的可用性的出席信息广播中包含这个联系人的 JID); 无论如何, 如果无法从用户的可用的资源直接发送出席信息, 用户的服务器必须( MUST)广播不可用出席信息给那个实体(如果这个用户还没有直接发送不可用出席信息给那个实体).
3. 如果用户不是在已经发送过初始化出席信息广播之后,或在发送不可用信息广播之前,直接发送出席信息(例如, 资源激活了但是还不可用), 用户的服务器必须(MUST)认为用户向其直接发送出席信息的这个实体视为上述第二种情形中的那个实体,采用相同的处理方式.

## 不可用出席信息

在和一个服务器结束它的会话之前, 客户端应该(SHOULD)雅致地成为不可用的,发送一个最后的没有'to'属性并且'type'属性值为"unavailable"的出席信息节(可选的, 最后的出席信息节可以(MAY)包含一个或多个<status/>元素以指明为什么用户不再可用). 无论如何, 用户的服务器不能(MUST NOT)依赖于从一个可用的资源接收最后的出席信息, 因为资源可能意外的变成不可用或可能被服务器判定超时. 如果用户的资源之一因为任何原因成为不可用的(包括雅致的或粗鲁的), 用户的服务器必须(MUST)广播不可用出席信息给所有如下的联系人 (1) 在用户的名册中并且订阅类型为"from" 或 "both", (2) 用户没有对它们屏蔽出站出席信息的联系人, 以及 (3) 用户会话期间,服务器没有从它们那里收到出席信息错误的联系人; 用户的服务器也必须(MUST)发送不可用出席信息节给这个用户的任何其他可用的资源, 以及任何用户的资源曾经在会话期间直接向其发送过出席信息的实体(如果用户还没有直接发送不可用出席信息给那个实体). 在直接发送或广播不可用出席信息之后发送的任何没有'type'属性也没有'to'属性的出席信息节必须(MUST)由服务器广播给所有订阅者.

## 出席信息订阅

一个订阅请求就是一个'type'属性值为"subscribe"的出席信息节. 如果订阅请求被发送给一个即时消息联系人, 在'to'属性中提供的 JID 的格式应该(SHOULD)是<contact@example.org>而不是<contact@example.org/resource>, 因为用户期望的结果通常是从联系人的所有资源接收到出席信息, 而不仅是'to'属性中的特定资源.
一个用户的服务器不能(MUST NOT)代替用户自动批准订阅请求. 所有订阅申请必须

(MUST)直接发给用户的客户端，具体来说就是这一用户的一个或多个可用的资源. 如果当订阅申请被用户的服务器接收到的时候没有这个用户的可用资源，用户的服务器必须(MUST)保持这个订阅申请的记录并且在用户下次建立一个可用的资源时递送这个订阅申请，直到这个用户批准或拒绝这个请求. 如果如果当订阅申请被用户的服务器接收到的时候这个用户有多于一个的可用资源，用户的服务器必须(MUST)广播这个订阅申请给所有可用的资源(根据 处理 XML 节的服务器规则 Server Rules for Handling XML Stanzas(第十一章)). (注意：如果一个激活的资源还没有提供初始化出席信息，服务器不能(MUST NOT)认为它是可用的并且因而不能(MUST NOT)发送订阅申请给它.) 无论如何，如果用户从一个它已授权可以看到用户的出席信息的联系人那里收到一个类型为"subscribe"的出席信息节(例如，当一个联系人重新同步订阅状态的时候),用户的服务器应该(SHOULD)代替用户自动应答. 另外，用户的服务器可以(MAY)基于一个特定实现的法则(例如，无论何时当用户的一个新的资源可用的时候，或在一段特定长度的时间过去之后)选择重新发送一个未批准的未决订阅申请给这个联系人；这有助于恢复可能和原始订阅申请有关的瞬间的,无声的错误.

## 指明可用性状态

一个客户端可以(MAY)使用<show/>元素提供关于可用性状态的更多信息(参见 Show (第二章第二节第二小节第一小小节)).

例子：可用性状态:

```
<presence>

  <show>dnd</show>

</presence>
```

## 指明详细的可用性状态信息

通过联合<show/>元素，客户端使用<status/>元素可以(MAY)提供详细的可用性状态信息(参见 Status (第二章第二节第二小节第二小小节)).

例子：详细的状态信息:

```
<presence xml:lang='en'>
  <show>dnd</show>
  <status>Wooing Juliet</status>
  <status xml:lang='cz'>Ja dvo&#x0159;&#x00ED;m Juliet</status>
</presence>
```

## 指明出席信息优先级

客户端可以(MAY)使用<priority/>元素为它的资源提供优先级(参见 Priority (第二章第二节第二小节第三小小节)).

例子：出席信息优先级:

```
<presence xml:lang='en'>
  <show>dnd</show>
  <status>Wooing Juliet</status>
  <status xml:lang='cz'>Ja dvo&#x0159;&#x00ED;m Juliet</status>
  <priority>1</priority>
</presence>
```

## 出席信息例子

本章的例子用于阐明上述和出席信息相关的协议. 用户是 romeo@example.net, 他有一个可用的资源, 资源 ID 为 "orchard", 并且他的名册中有以下这些人:

- juliet@example.com (subscription="both" 并且她有两个可用的资源, 一个资源名为"chamber" 而另一个资源名为 "balcony")
- benvolio@example.org (subscription="to")
- mercutio@example.org (subscription="from")

例子 1: 用户发送初始化出席信息:

```
<presence/>
```

例子 2: 用户的服务器代替用户发送出席信息调查给 subscription="to" 和 subscription="both" 的联系人的可用资源:

```
<presence
    type='probe'
    from='romeo@example.net/orchard'
    to='juliet@example.com'/>
<presence
    type='probe'
    from='romeo@example.net/orchard'
    to='benvolio@example.org'/>
```

例子 3: 用户的服务器代替用户发送初始化出席信息给 subscription="from" 和 subscription="both"的联系人的可用资源:

```
<presence
    from='romeo@example.net/orchard'
    to='juliet@example.com'/>
<presence
    from='romeo@example.net/orchard'
    to='mercutio@example.org'/>
```

例子 4: 联系人的服务器代替所有可用的资源应答出席信息调查:

```
<presence
    from='juliet@example.com/balcony'
    to='romeo@example.net/orchard'
    xml:lang='en'>
```

```
  <show>away</show>
  <status>be right back</status>
  <priority>0</priority>
</presence>
<presence
    from='juliet@example.com/chamber'
    to='romeo@example.net/orchard'>
  <priority>1</priority>
</presence>
<presence
    from='benvolio@example.org/pda'
    to='romeo@example.net/orchard'
    xml:lang='en'>
  <show>dnd</show>
  <status>gallivanting</status>
</presence>
```

例子 5: 联系人的服务器递送用户的初始化出席信息给所有可用的资源或返回错误给用户:

```
<presence
    from='romeo@example.net/orchard'
    to='juliet@example.com/chamber'/>
<presence
    from='romeo@example.net/orchard'
    to='juliet@example.com/balcony'/>
<presence
    type='error'
    from='mercutio@example.org'
    to='romeo@example.net/orchard'>
  <error type='cancel'>
    <gone xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</presence>
```

例子 6: 用户直接发送出席信息给另一个不在他的名册中的用户:

```
<presence
    from='romeo@example.net/orchard'
    to='nurse@example.com'
    xml:lang='en'>
  <show>dnd</show>
  <status>courting Juliet</status>
  <priority>0</priority>
</presence>
```

例子 7: 用户发送更新的可用出席信息用于广播:

```
<presence xml:lang='en'>
  <show>away</show>
```

```
  <status>I shall return!</status>
  <priority>1</priority>
</presence>
```

例子 8: 用户的服务器仅向一个联系人广播更新的出席信息 (不是那些返回错误的联系人,也不是那些用户直接向其发送出席信息的联系人):

```
<presence
    from='romeo@example.net/orchard'
    to='juliet@example.com'
    xml:lang='en'>
  <show>away</show>
  <status>I shall return!</status>
  <priority>1</priority>
</presence>
```

例子 9: 联系人的服务器递送更新的出席信息给联系人所有可用的资源:

```
[to "balcony" resource...]
<presence
    from='romeo@example.net/orchard'
    to='juliet@example.com'
    xml:lang='en'>
  <show>away</show>
  <status>I shall return!</status>
  <priority>1</priority>
</presence>
[to "chamber" resource...]
<presence
    from='romeo@example.net/orchard'
    to='juliet@example.com'
    xml:lang='en'>
  <show>away</show>
  <status>I shall return!</status>
  <priority>1</priority>
</presence>
```

例子 10: 联系人的资源之一广播最后出席信息:

```
<presence from='juliet@example.com/balcony' type='unavailable'/>
```

例子 11: 联系人的服务器发送不可用出席信息给用户:

```
<presence
    type='unavailable'
    from='juliet@example.com/balcony'
    to='romeo@example.net/orchard'/>
```

例子 12: 用户发送最后出席信息:

```
<presence from='romeo@example.net/orchard'
          type='unavailable'
          xml:lang='en'>
  <status>gone home</status>
```

```
</presence>
```

例子 13: 用户的服务器广播不可用出席信息给联系人,包括用户直接向其发送出席信息的那个人:

```
<presence
    type='unavailable'
    from='romeo@example.net/orchard'
    to='juliet@example.com'
    xml:lang='en'>
  <status>gone home</status>
</presence>
<presence
    from='romeo@example.net/orchard'
    to='nurse@example.com'
    xml:lang='en'>
  <status>gone home</status>
</presence>
```

# 管理订阅

为了保护即时消息用户和任何其他实体的隐私，出席信息和可用性信息仅向用户已批准的其他实体披露. 当一个用户同意其他用户可以看到它的出席信息，这个实体被称为对于用户的出席信息有一个订阅. 订阅超越了会话；实际上，它一直存在直到订阅者取消订阅或被订阅者取消曾经授权的订阅为止. 在 XMPP 中订阅是通过发送包含特定属性的出席信息节来管理的.

注意: 在订阅和名册之间有重要的交互；这些定义在 名册条目和出席信息订阅的集成 Integration of Roster Items and Presence Subscriptions (第八章)，而且读者必须参考那一章才能完整地理解出席信息订阅.

## 请求一个订阅

对另一个实体的出席信息的订阅请求是由发送一个类型为"subscribe"的出席信息节来开始的.

例子: 发送一个订阅请求:

```
<presence to='juliet@example.com' type='subscribe'/>
```

关于客户端和服务器在订阅请求中的职责，参考 出席信息订阅 Presence Subscriptions(第五章第一节第六小节).

## 处理一个订阅请求

当一个客户端从另一个实体接收到一个订阅请求，它必须(MUST)批准这个请求(发送一个类型为"subscribed"的出席信息节)或拒绝这个请求(发送一个类型为"unsubscribed"的出席信息节).

例子: 批准一个订阅请求:

```
<presence to='romeo@example.net' type='subscribed'/>
```

例子: 拒绝一个出席信息订阅的请求:

```
<presence to='romeo@example.net' type='unsubscribed'/>
```

## 从另一个实体取消一个订阅

如果一个用户想取消一个曾经允许的订阅请求, 它发送一个类型为"unsubscribed"的出席信息节.

例子: 取消一个曾经允许的订阅请求:

```
<presence to='romeo@example.net' type='unsubscribed'/>
```

## 取消对于另一个实体的出席信息的订阅

如果用户想取消对于另一个实体的出席信息的订阅, 它发送一个类型为"unsubscribe"的出席信息节.

例子: 取消对一个实体的出席信息的订阅:

```
<presence to='juliet@example.com' type='unsubscribe'/>
```

# 名册管理

在 XMPP 中, 一个人的联系人列表被称为名册(roster), 它包括任意数量的特定名册条目, 每个名册条目被一个唯一的 JID(通常格式是<contact@domain>)所标识. 一个用户的名册由用户的服务器代替用户储存从而这个用户可以从任何资源访问名册信息.

注意: 在名册和订阅之间有重要的交互; 这些定义在 名册条目和出席信息订阅的集成 Integration of Roster Items and Presence Subscriptions (第八章), 而且读者必须参考那一章才能完整地理解名册管理.

## 语法和语义

名册使用 IQ 节来管理, 具体来说就是符合'jabber:iq:roster'名字空间的<query/>子元素的含义.这个<query/>元素可以(MAY)包含一个或更多<item/>子元素, 每个描述一个唯一的名册条目或曰 联系人"contact".

每个名册条目的"key"或者说唯一标识符就是一个 JID,封装在<item/>元素的'jid'属性(它是必需的(REQUIRED))之中. 如果这个条目是和另一个(人类)即时消息用户相关的,'jid'属性的值的格式应该(SHOULD)是<user@domain>.

和一个名册条目相关的出席信息订阅的状态从<item/>元素的'subscription'属性可以得到.这个属性允许的值包括:

- "none" -- 这个用户没有对这个联系人出席信息的订阅，这个联系人也没有订阅用户的出席信息
- "to" -- 这个用户订阅了这个联系人的出席信息，但是这个联系人没有订阅用户的出席信息
- "from" -- 这个联系人订阅了用户的出席信息，但是这个用户没有订阅这个联系人的出席信息
- "both" -- 用户和联系人互相订阅了对方的出席信息

每个<item/>条目可以(MAY)包含一个'name'属性，它设置和这个 JID 相关的"nickname"，取决于用户(而不是联系人). 'name'属性的值是不透明的.
每个<item/>条目可以(MAY)包含一个或多个<group/>子元素,用于把名册条目收集到多个类别之中. <group/>子元素的 XML 字符数据是不透明的.

## 商业规则

在一个名册"set"中一个服务器必须(MUST)忽略任何'to'地址，并且必须(MUST)认为任何名册"set"是应用于发送者的. 为了更多的安全性，一个客户端应该(SHOULD)检查"roster push"(包含一个名册条目的类型为"set"的输入 IQ)的"from"地址以保证它来自一个信任的源；具体的，这个节必须(MUST)没有 'from'属性(例如，从服务器隐含的) 或'from'属性的值匹配用户的纯 JID(格式为<user@domain>)或全 JID(格式为<user@domain/resource>)；否则，客户端应该(SHOULD)忽略这个"roster push".

## 登录时接收一个人的名册

在连接到服务器并成为一个激活的资源之后，一个客户端应该(SHOULD)在发送初始化出席信息之前请求名册(无论如何，因为可能不是所有的资源都想接收名册，例如，一个带宽受限的连接，客户端对于名册的请求是可选的(OPTIONAL)). 如果一个可用的资源在一个会话期间没有请求名册，服务器不能(MUST NOT)向它发送出席信息订阅以及相关的名册更新.

例子: 客户端向服务器请求当前的名册:

```
<iq from='juliet@example.com/balcony' type='get' id='roster_1'>
  <query xmlns='jabber:iq:roster'/>
</iq>
```

例子: 客户端从服务器收到名册:

```
<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>
```

## 增加一个名册条目

任何时候，一个用户可以(MAY)增加一个条目到他或她的名册.

例子：客户端添加一个新的条目：

```
  <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>
```

服务器必须(MUST)在持久信息存储机构中更新名册信息,并且也要向这个用户的所有已请求名册的可用资源推送这一改变. 这个"名册推送"包括一个类型为"set"的 IQ 节,从服务器发送给客户端,使用户的所有可用资源保持和基于服务器的名册信息的同步.

例子：服务器 (1) 推送更新的名册信息给所有已请求名册的可用资源 并且 (2) 以一个 IO 结果应答发送的资源:

```
  <iq to='juliet@example.com/balcony'
      type='set'
      id='a78b4q6ha463'>
    <query xmlns='jabber:iq:roster'>
      <item jid='nurse@example.com'
            name='Nurse'
            subscription='none'>
        <group>Servants</group>
      </item>
    </query>
  </iq>
  <iq to='juliet@example.com/chamber'
```

```
      type='set'
      id='a78b4q6ha464'>
    <query xmlns='jabber:iq:roster'>
      <item jid='nurse@example.com'
            name='Nurse'
            subscription='none'>
        <group>Servants</group>
      </item>
    </query>
  </iq>
  <iq to='juliet@example.com/balcony' type='result' id='roster_2'/>
```

正如 IQ 节类型(定义在[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920])的语义所要求的,每个接收到了名册推送的资源必须(MUST)应答一个类型为"result"(或 "error")的 IQ 节.

例子: 资源应答一个 IQ 结果给服务器:

```
<iq from='juliet@example.com/balcony'
      to='example.com'
      type='result'
      id='a78b4q6ha463'/>
<iq from='juliet@example.com/chamber'
      to='example.com'
      type='result'
      id='a78b4q6ha464'/>
```

## 更新名册条目

更新一个已有的名册条目(例如, 改变组) 的方法和增加一个新的名册条目是一样的, 换言之, 在 IQ set 节中发送名册条目给服务器.

例子: 用户更新名册条目(增加组):

```
<iq from='juliet@example.com/chamber' type='set' id='roster_3'>
  <query xmlns='jabber:iq:roster'>
    <item jid='romeo@example.net'
          name='Romeo'
          subscription='both'>
      <group>Friends</group>
      <group>Lovers</group>
    </item>
  </query>
</iq>
```

正如增加一个名册条目, 当更新一个名册条目时服务器必须(MUST)在持久信息存储机构中更新名册信息, 并且也要初始化一个名册推送给这个用户的所有已请求名册的可用资源.

## 删除一个名册条目

任何时候，用户可以(MAY)从他或她的名册中删除一个条目,只要发送一个 IQ set 给服务器并确保其'subscription'属性值为"remove" (如果从一个客户端接收到'subscription'属性的任何其他值,一个兼容的服务器必须(MUST)忽略它).

例子：客户端移除一个条目：

```
<iq from='juliet@example.com/balcony' type='set' id='roster_4'>
  <query xmlns='jabber:iq:roster'>
    <item jid='nurse@example.com' subscription='remove'/>
  </query>
</iq>
```

和增加一个名册条目一样，删除一个名册条目的时候服务器必须(MUST)在持久信息存储机构中更新名册信息,初始化一个名册推送给这个用户的所有已请求名册的可用资源(伴随着把'subscription'属性值设为"remove"),并且发送一个 IQ result 给初始化资源.

关于这个命令的含义的更多信息，见 移除一个名册条目并取消所有订阅 Removing a Roster Item and Cancelling All Subscriptions (第八章第六节).

# 名册条目和出席信息订阅的集成

## 概览

关于用户从或向别的联系人订阅出席信息,一个即时消息用户通常希望在名册条目和出席信息订阅之间有某些层次的集成. 本章描述了在 XMPP 即时消息应用中必须(MUST)支持的那些层次的集成.

有四种主要的订阅状态:

- None -- 这个用户没有对这个联系人出席信息的订阅，这个联系人也没有订阅用户的出席信息
- To -- 这个用户订阅了这个联系人的出席信息，但是这个联系人没有订阅用户的出席信息
- From -- 这个联系人订阅了用户的出席信息，但是这个用户没有订阅这个联系人的出席信息
- Both -- 用户和联系人互相订阅了对方的出席信息(例如，联合'from' 和 'to')

这些状态的每一个都被反射到用户和联系人双方的名册中，从而导致持久的订阅状态.

在以下的子章节中将叙述这些订阅状态如何为了完成特定的已定义的用例而进行交互. 关于服务器和客队户端处理所有订阅状态的细节 (包括处于以上所列的状态之

外的未决状态)在 订阅状态 Subscription States(第九章).
服务器不能(MUST NOT)发送出席信息订阅请求或名册推送给不可用的资源，也不能给没有已请求的名册的可用资源.
在名册推送中'from'和'to'地址是可选的(OPTIONAL)；如果包含了，它们的值应该(SHOULD)是那个会话的资源的全 JID. 一个客户端必须(MUST)以一个类型为"result"的 IQ 节来承认每个名册推送(为了暂时的原因，这些节不显示在以下的例子中但是按[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]定义的 IQ 语义学的规定它们是必需的).

## 用户向联系人订阅

以下描述一个用户向一个联系人订阅的过程，包括名册条目和订阅状态之间的互动.
1. 为了能够在用户的客户端界面处理联系人以及在服务器跟踪订阅，用户的客户端应该(SHOULD)为新的名册条目执行一个"roster set". 这个请求包括发送一个类型为'set'的 IQ 节并拥有符合'jabber:iq:roster'名字空间的<query/>子元素，它(<query/>元素)再包含一个<item/>子元素来定义新的名册条目；这个<item/>元素必须(MUST)拥有一个'jid'属性，可以(MAY)拥有一个'name'属性，不能(MUST NOT)拥有一个'subscription'属性，并且可以(MAY)包含一个或多个<group/>子元素:

```
<iq type='set' id='set1'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='contact@example.org'
        name='MyContact'>
      <group>MyBuddies</group>
    </item>
  </query>
</iq>
```

2. 作为结果，这个用户的服务器 (1) 必须(MUST)为这个新的名册条目初始化一个名册推送给这个用户的所有已经请求名册的可用资源，其'subscription'属性的值为"none"；并且 (2) 必须(MUST)以一个 IQ result 应答发送的资源表明名册设置成功了:

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='contact@example.org'
        subscription='none'
        name='MyContact'>
      <group>MyBuddies</group>
    </item>
  </query>
</iq>
<iq type='result' id='set1'/>
```

3. 如果用户想向这个联系人请求出席信息的订阅，用户的客户端必须(MUST)发送一个类型为'subscribe'的出席信息节给联系人:

```
<presence to='contact@example.org' type='subscribe'/>
```

4. 作为结果，用户的服务器必须(MUST)初始化第二个名册推送给这个用户的所有已经请求名册的可用资源,把这个联系人设置成'none'订阅状态的未决子状态；这个未决子状态是由名册条目中包含的 ask='subscribe'属性所指示的:

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='contact@example.org'
        subscription='none'
        ask='subscribe'
        name='MyContact'>
      <group>MyBuddies</group>
    </item>
  </query>
</iq>
```

注意：如果用户在发送订阅请求之前没有新建一个名册条目，服务器必须(MUST)现在代替用户新建一个,然后发送一个名册推送给这个用户的所有已经请求名册的可用资源，不含以上所示的'name'属性和<group/>子元素.

5. 用户的服务器也必须(MUST)把这个类型为"subscribe"的出席信息节的'from'地址设置为用户的纯JID(例如, <user@example.com>)(如果用户提供了设置为用户的全JID的'from'地址，服务器应该(SHOULD)移除资源 ID). 如果联系人和用户在不同的主机上，用户的服务器必须(MUST)路由这个出席信息节到联系人的服务器来递送到这个联系人(这种情形的假定贯穿本文；无论如何，如果联系人在同一台主机，那么服务器可以简单地直接递送出席信息节):

```
<presence
    from='user@example.com'
    to='contact@example.org'
    type='subscribe'/>
```

注意:如果用户的服务器从联系人的服务器收到了一个类型为"error"的出席信息节，它必须(MUST)这个错误节给用户，用户的客户端可以(MAY)确定那个错误是否对于上次用户发出的"subscribe"类型的出席信息节(例如，通过跟踪'id'属性)的应答,然后选择重新发送"subscribe"请求还是发送一个"unsubscribe"类型的出席信息节给联系人以恢复到它的上一个状态.

6. 接收到指向联系人的"subscribe"类型的出席信息节之后，这个联系人的服务器必须(MUST)决定是否至少有一个已请求名册的联系人的可用资源. 如果是，它必须(MUST)递送这个订阅请求给这个联系人(如果不是，联系人的服务器必须(MUST)离线存储这个订阅请求用于递送 when this condition is next met；通常这是通过增加一个关于这个联系人的名册条目到用户名册中来实现的，伴随着一个 "None + Pending In"的状态(定义在 订阅状态 Subscription States (第九章))，无论如何一个服务器不应该(SHOULD NOT)在那种状态下推送或递送名册条目给联系人). 不论何时订阅请求被递送到了，联系人必须决定是否批准它(根据联系人的配置选项，联系人的客户端可以(MAY）批准或拒绝订阅请求而无需向联系人显示). 这里我们假定这个 "happy path"，即联系人批准了订阅请求(替代的拒绝订阅请求的流程定义在第八章第二节第一小节). 在这种情形下，这个联系人的客户端 (1) 应该(SHOULD) 执行一个 roster

set 为这个用户指明期望的昵称和组(如果有的话); 并且 (2) 必须(MUST)发送一个"subscribed"类型的出席信息节给这个用户以批准这个订阅请求.

```
<iq type='set' id='set2'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='user@example.com'
        name='SomeUser'>
      <group>SomeGroup</group>
    </item>
  </query>
</iq>
<presence to='user@example.com' type='subscribed'/>
```

7. 作为结果, 联系人的服务器 (1) 必须(MUST) 初始化一个名册推送给所有联系人已请求名册的可用资源, 包含一个关于那个用户的名册条目,并且其订阅状态为'from'(甚至联系人不执行 roster set,服务器也必须(MUST)发送它); (2) 必须(MUST)返回一个 IQ result 给发送的资源表示名册设置(roster set)成功了; (3) 必须(MUST)路由这个"subscribed"类型的出席信息节给用户, 首先把'from'地址设为联系人的纯JID(<contact@example.org>); 然后 (4) 必须(MUST)从所有联系人的可用资源向用户发送可用的出席信息:

```
<iq type='set' to='contact@example.org/resource'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='user@example.com'
        subscription='from'
        name='SomeUser'>
      <group>SomeGroup</group>
    </item>
  </query>
</iq>
<iq type='result' to='contact@example.org/resource' id='set2'/>
<presence
    from='contact@example.org'
    to='user@example.com'
    type='subscribed'/>
<presence
    from='contact@example.org/resource'
    to='user@example.com'/>
```

注意: 如果联系人的服务器从用户的服务器收到一个"error"类型的出席信息节, 它必须(MUST)递送这个错误节给联系人, 联系人的客户端可以(MAY)确定那个错误是否对于上次联系人发出的"subscribe"类型的出席信息节(例如, 通过跟踪'id'属性)的应答,然后选择重新发送"subscribe"请求还是发送一个"unsubscribe"类型的出席信息节给用户以恢复到它的上一个状态.

8. 接收到一个指向用户的"subscribed"类型的出席信息节之后, 用户的服务器必须(MUST)首先检查在用户名册中的这个联系人的状态是: (a) subscription='none' and

ask='subscribe' 还是 (b) subscription='from' and ask='subscribe'. 如果联系人不是以上述的状态在用户的名册中,用户的服务器必须(MUST)安静的忽略这个"subscribed"类型的出席信息节(例如, 服务器不能(MUST NOT)路由它到用户, 修改用户的名册, 或生成一个名册推送到用户的可用资源). 如果联系人以上述任何一种状态存在于用户的名册中, 用户的服务器 (1) 必须(MUST)从联系人向用户递送这个"subscribed"类型的出席信息节; (2)必须(MUST)初始化一个名册推送给所有已请求名册的这个用户的可用资源,包含一个关于这个联系人的更新的名册条目,同时其'subscription'属性值设置为"to"; 并且 (3) 必须(MUST)从每一个联系人的可用资源向每一个用户的可用资源递送服务器接收到的可用的出席信息节:

```
<presence
    to='user@example.com'
    from='contact@example.org'
    type='subscribed'/>
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='contact@example.org'
        subscription='to'
        name='MyContact'>
      <group>MyBuddies</group>
    </item>
  </query>
</iq>
<presence
    from='contact@example.org/resource'
    to='user@example.com/resource'/>
```

9. 接收到"subscribed"类型的出席信息节之后, 用户应该(SHOULD)承认接收到了订阅状态通知,

要么发送一个"subscribe"类型的出席信息节给联系人证实它, 要么发送一个"unsubscribe"类型的出席信息节给联系人否认它;这个步骤不一定影响订阅状态(见 订阅状态 Subscription States(第九章)的细节), 但是会让用户用户的服务器知道它必须(MUST)不再发送订阅状态改变通知给用户(见第九章第四节).

从用户这方面看, 现在存在一个向联系人的出席信息的订阅; 从联系人的方面看, 现在存在一个从用户的来的订阅.

## 替代流程: 联系人拒绝订阅请求

以上活动流程展示了关于用户向联系人的订阅请求的 "happy path" . 如果联系人拒绝用户的订阅请求,那么主要的替代流程如下所述.

1. 如果联系人想拒绝这个请求, 联系人的客户端必须(MUST)发送一个"unsubscribed"类型的出席信息节给用户(取代第八章第二节中步骤 6 发送的"subscribed"类型的出席信息节):

```
<presence to='user@example.com' type='unsubscribed'/>
```

2. 作为结果，联系人的服务器必须(MUST)路由这个"unsubscribed"类型的出席信息节给用户,首先把'from'地址设为联系人的纯 JID(<contact@example.org>):

```
<presence
    from='contact@example.org'
    to='user@example.com'
    type='unsubscribed'/>
```

注意：如果联系人的服务器之前把用户添加到了联系人的名册中用来跟踪，这时它必须(MUST)移除这个相关的条目.

3. 接收到指向用户的"unsubscribed"类型出席信息节之后，用户的服务器 (1) 必须(MUST)地送那个出席信息节给用户 并且 (2) 必须(MUST) 初始化一个名册推送给这个用户的所有已请求名册的可用资源，包含一个关于这个联系人的一个更新条目，其'subscription'属性设为"none"并且没有'ask'属性:

```
<presence
    from='contact@example.org'
    to='user@example.com'
    type='unsubscribed'/>
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='contact@example.org'
        subscription='none'
        name='MyContact'>
      <group>MyBuddies</group>
    </item>
  </query>
</iq>
```

4. 接收到类型为"unsubscribed"出席信息节之后，用户应该(SHOULD)承认收到订阅状态通知，要么发送一个"unsubscribe"类型的出席信息节给联系人证实它，要么发送一个"subscribe"类型的出席信息节给联系人否认它；这一步骤不影响订阅状态(见订阅状态 Subscription States(第九章)的细节)，但是让用户的服务器知道它必须(MUST)不再发送订阅状态改变的通知给用户(见第九章第四节).

作为这一行为的结果，联系人现在在用户的名册中，状态为"none",而用户根本不在联系人的名册中.

## 建立一个相互的订阅

用户和联系人可以在前述"happy path"的基础上建立一个相互的订阅(例如，一个"both"的订阅类型). 流程如下.

1. 如果联系人想建立一个相互的订阅，联系人必须(MUST)发送一个订阅请求给用户(视联系人的配置选项而定，联系人的客户端可以(MAY)自动发送它):

```
<presence to='user@example.com' type='subscribe'/>
```

2. 作为结果，联系人的服务器 (1) 必须(MUST)初始化一个名册推送给联系人的所有已请求名册的可用资源，伴随着用户仍在'from'订阅状态但同时有一个未决的'to'

订阅状态(通过在名册条目中包含一个 ask='subscribe'的属性来指示); 并且 (2) 必须(MUST)路由这个"subscribe"类型的出席信息节给用户(先把'from'地址设为联系人的纯 JID(<contact@example.org>)):

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='user@example.com'
        subscription='from'
        ask='subscribe'
        name='SomeUser'>
      <group>SomeGroup</group>
    </item>
  </query>
</iq>
<presence
    from='contact@example.org'
    to='user@example.com'
    type='subscribe'/>
```

注意: 如果联系人的服务器从用户的服务器收到一个"error"类型的出席信息节, 它必须(MUST)递送这个错误节给联系人, 它的客户端可以(MAY)确定这个错误是用来应答上次发送的"subscribe"类型的出席信息节(换言之, 通过跟踪'id'属性) 并且选择重发这个"subscribe"请求还是发送一个"unsubscribe"类型的出席信息节给用户以把名册恢复到它的前一个状态.

3. 接收到指向用户的"subscribe"类型出席信息节之后, 用户的服务器必须确定是否至少有一个已请求名册可用资源. 如果是, 用户的服务器必须(MUST)递送这个订阅请求给用户(如果不是, 它必须(MUST)离线存储这个订阅请求等这种情形再次发生时递送). 无论何时订阅请求被递送了, 用户必须决定是否批准它(视用户的配置选项而定, 用户的客户端可以(MAY)批准或拒绝这个订阅请求而不需要向用户显示). 这里我们假定这是"happy path",用户批准了订阅请求(替代的拒绝订阅请求的流程定义在第八章第三节第一小节). 在这种情形下, 用户的客户端必须(MUST)发送一个"subscribed"类型的出席信息节给联系人表示批准了订阅请求.

```
<presence to='contact@example.org' type='subscribed'/>
```

4. 作为结果, 用户的服务器 (1) 必须(MUST)初始化一个名册推送给用户的所有已请求名册的可用资源, 包含一个关于联系人的名册条目,其'subscription'属性设为"both"; (2) 必须(MUST)路由这个"subscribed"类型的出席信息节给联系人(先把'from'地址设为用户的纯 JID<user@example.com>)); 并且 (3) 必须(MUST)向联系人发送它从用户的每个可用资源收到的最近一次出席信息节的全 XML(不带'to'属性)(强制每个会话遵守隐私列表):

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='contact@example.org'
        subscription='both'
        name='MyContact'>
```

```
        <group>MyBuddies</group>
      </item>
    </query>
</iq>
<presence
      from='user@example.com'
      to='contact@example.org'
      type='subscribed'/>
<presence
      from='user@example.com/resource'
      to='contact@example.org'/>
```

注意：如果用户的服务器从联系人的服务器接收到一个"error"类型的出席信息节，它必须(MUST)递送这个错误节给用户，它客户端可以(MAY)确定这个错误是用来应答上次发出去的"subscribed"类型的出席信息节(换言之，通过跟踪'id'属性) 并且选择重发这个订阅请求还是发送一个"unsubscribed"类型的出席信息节给联系人以把名册恢复到上次的状态.

5. 接收到指向联系人的"subscribed"类型的出席信息节之后，联系人的服务器必须(MUST)首先检查用户在联系人的名册中的状态是否以下状态之一：(a) subscription='none' and ask='subscribe' 或 (b) subscription='from' and ask='subscribe'. 如果用户不是以上述两种状态之一存在于联系人的名册中，联系人的服务器必须(MUST)安静地忽略这个"subscribed"类型的出席信息节(例如，它不能(MUST NOT)路由它给联系人，修改联系人的名册，或生成一个名册推送给联系人的可用资源). 如果用户以上述两种状态之一存在于联系人的名册中，联系人的服务器 (1) 必须(MUST)从用户向联系人递送这个"subscribed"类型的出席信息节；(2) 必须(MUST)初始化一个名册推送给这个联系人的所有已请求名册的可用资源，包含一个关于这个用户的更新的名册条目,其'subscription'属性值设为"both"；并且 (3) 必须(MUST)向这个联系人的每个可用资源递送它从这个用户的每个资源收到的可用出席信息节:

```
<presence
      from='user@example.com'
      to='contact@example.org'
      type='subscribed'/>
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='user@example.com'
        subscription='both'
        name='SomeUser'>
      <group>SomeGroup</group>
    </item>
  </query>
</iq>
<presence
      from='user@example.com/resource'
      to='contact@example.org/resource'/>
```

6. 收到"subscribed"类型的出席信息节之后，联系人应该(SHOULD)承认收到订阅请求通知,要么发送一个"subscribe"的出席信息节给用户证实它,要么发送一个"unsubscribe"类型的出席信息节给用户否认它；这一步骤不影响订阅状态(细节见订阅状态 Subscription States(第九章))，但是让联系人的服务器知道它必须(MUST)不再发送订阅状态变更通知给联系人(见第九章第四节).
用户和联系人现在有了对双方的出席信息的一个相互订阅 \-\- 换言之，这个订阅类型为 "both".

## 替代流程：用户拒绝订阅请求

以上活动流程展示了关于联系人对用户的订阅请求的 "happy path". 如果用户拒绝了联系人的订阅请求,其主要流程如下.
1. 如果用户想拒绝请求，用户的客户端必须(MUST)发送一个"unsubscribed"类型的出席信息节给联系人(替代第八章第三节中的第三步中所发送的"subscribed"类型出席信息节):

```
<presence to='contact@example.org' type='unsubscribed'/>
```

2. 作为结果，用户的服务器必须(MUST)路由这个"unsubscribed"类型的出席信息节给联系人(首先把'from'地址设为用户的纯 JID(<user@example.com>)):

```
<presence
    from='user@example.com'
    to='contact@example.org'
    type='unsubscribed'/>
```

3. 接收到指向联系人的"unsubscribed"类型的出席信息节之后，联系人的服务器 (1) 必须(MUST)递送这个出席信息节给联系人；并且 (2) 必须(MUST)初始化一个名册推送给这个联系人的所有已请求名册的可用资源，包含关于这个用户的更新的名册条目,其'subscription'属性的值设为"from"并且没有'ask'属性:

```
<presence
    from='user@example.com'
    to='contact@example.org'
    type='unsubscribed'/>
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='user@example.com'
        subscription='from'
        name='SomeUser'>
      <group>SomeGroup</group>
    </item>
  </query>
</iq>
```

4. 接收到"unsubscribed"类型的出席信息节之后，联系人应该(SHOULD)承认收到那个订阅状态通知,要么向用户发送一个"unsubscribe"类型的出席信息节以证实它,要么向用户发送一个"subscribe"类型的出席信息以否认它；这个步骤不会影响订阅状

态(详见 订阅状态 Subscription States(第九章)),但是让联系人的服务器知道它必须(MUST)不再发送订阅状态变更通知给联系人(见第九章第四节).
作为这一活动的结果，订阅状态没有任何改变；换言之，联系人在用户的名册中的订阅状态为"to"并且用户在联系人的名册中的订阅状态为"from".

## 取消订阅

在订阅了一个联系人的出席信息之后的任何时候，一个用户可以(MAY)取消订阅. 在所有实例中用户发送来执行这一动作的 XML 是相同的，接下来的订阅状态根据发出取消订阅命令时获得的订阅状态的情况而不同. 两种可能的情节描述如下.

## 情形 #1: 当订阅不是相互的时候取消订阅

在第一种情形，用户有一个向联系人的出席信息的订阅但是联系人没有对用户的出席信息的订阅（换言之，订阅不是相互的).
1. 如果用户想取消对联系人的出席信息的订阅，用户必须(MUST)发送一个"unsubscribe"类型的出席信息节给联系人:

```
<presence to='contact@example.org' type='unsubscribe'/>
```

2. 作为一个结果，用户的服务器 (1) 必须(MUST) 发送一个名册推送给这个用户的所有已请求名册的可用资源,包含一个关于这个联系人的更新名册条目,其'subscription'属性设为"none"；并且 (2) 必须(MUST)路由这个"unsubscribe"类型的出席信息节给联系人(首先把'from'地址设为用户的纯 JID(<user@example.com>)):

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='contact@example.org'
        subscription='none'
        name='MyContact'>
      <group>MyBuddies</group>
    </item>
  </query>
</iq>
<presence
    from='user@example.com'
    to='contact@example.org'
    type='unsubscribe'/>
```

3. 接收到指向联系人的"unsubscribe"类型出席信息节之后，联系人的服务器 (1) 必须(MUST)初始化一个名册推送给这个联系人的所有已请求名册的可用资源，包含一个关于这个用户的名册条目,其'subscription'属性值设为"none" (如果联系人不可用或未曾请求名册，联系人的服务器必须(MUST)修改名册条目并在下次联系人请求名册时发送那个已修改的条目)；并且 (2) 必须(MUST)递送这个"unsubscribe"状态改变通知给联系人:

```
<iq type='set'>
```

```
    <query xmlns='jabber:iq:roster'>
      <item
          jid='user@example.com'
          subscription='none'
          name='SomeUser'>
        <group>SomeGroup</group>
      </item>
    </query>
</iq>
<presence
    from='user@example.com'
    to='contact@example.org'
    type='unsubscribe'/>
```

4. 接收到"unsubscribe"类型的出席信息节之后，联系人应该(SHOULD)承认收到那个订阅状态通知,要么发送一个"unsubscribed"类型的出席信息节给用户以证实它,要么发送一个"subscribed"类型的出席信息节给用户否认它；这个步骤不影响订阅状态(详见 订阅状态 Subscription States (第九章))，但是让联系人的服务器知道它必须(MUST)不再发送订阅状态变更通知给联系人(见第九章第四节).

5. 联系人的服务器接着 (1) 必须(MUST)发送一个"unsubscribed"类型的出席信息节给用户;并且 (2) 应该(SHOULD)向用户发送从这个联系人的所有可用资源收到的不可用出席信息:

```
<presence
    from='contact@example.org'
    to='user@example.com'
    type='unsubscribed'/>
<presence
    from='contact@example.org/resource'
    to='user@example.com'
    type='unavailable'/>
```

6. 当用户的服务器收到类型为"unsubscribed" 和 "unavailable"的出席信息节，它必须(MUST)递送它们给用户:

```
<presence
    from='contact@example.org'
    to='user@example.com'
    type='unsubscribed'/>
<presence
    from='contact@example.org/resource'
    to='user@example.com'
    type='unavailable'/>
```

7. 接收到"unsubscribed"类型的出席信息节之后，用户应该(SHOULD)承认收到那个订阅状态变更通知,要么向联系人发送一个"unsubscribe"类型的出席信息节以证实它,要么向联系人发送一个"subscribe"的出席信息节以否认它;这步骤不影响订阅状态(详见 订阅状态 Subscription States(第九章))，但是让用户的服务器知道它必须(MUST)不在发送订阅状态变更通知给用户(见第九章第四节).

## 情形 #2: 当订阅是相互的时候取消订阅

在第二种情形下, 用户有一个向联系人的出席信息的订阅并且联系人也有一个向用户的出席信息的订阅(换言之, 订阅是相互的).

1. 如果用户想从联系人的出席信息取消订阅, 用户必须(MUST)发送一个"unsubscribe"类型的出席信息节给联系人:

```
<presence to='contact@example.org' type='unsubscribe'/>
```

2. 作为一个结果, 用户的服务器 (1) 必须(MUST)发送一个名册推送给这个用户的所有已请求名册的可用资源,包含一个关于这个联系人的更新名册条目,其'subscription'属性值设为"from"; 并且 (2) 必须(MUST)路由这个"unsubscribe"类型的出席信息节给这个联系人( 首先把'from'地址设为这个用户的纯JID(<user@example.com>):

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='contact@example.org'
        subscription='from'
        name='MyContact'>
      <group>MyBuddies</group>
    </item>
  </query>
</iq>
<presence
    from='user@example.com'
    to='contact@example.org'
    type='unsubscribe'/>
```

3. 接收到指向联系人的"unsubscribe"类型的出席信息节之后, 联系人的服务器 (1) 必须(MUST)初始化一个名册推送给这个联系人的所有已请求名册的可用资源, 包含一个关于这个用户的名册条目,其'subscription'属性值设为"to" (如果联系人不可用或未曾请求名册, 联系人的服务去必须(MUST)修改这个名册条目并且等下次联系人请求名册的时候再发送这个修改过的名册条目); 并且 (2) 必须(MUST)递送这个"unsubscribe"状态变更通知给联系人:

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='user@example.com'
        subscription='to'
        name='SomeUser'>
      <group>SomeGroup</group>
    </item>
  </query>
</iq>
<presence
```

```
    from='user@example.com'
    to='contact@example.org'
    type='unsubscribe'/>
```

4. 接收到这个"unsubscribe"类型的出席信息节之后，联系人应该(SHOULD)承认收到了那个订阅状态通知,要么向用户发送一个"unsubscribed"类型的出席信息节以证实它,要么向用户发送一个"subscribed"类型的出席信息节以否认它；这个步骤不影响订阅状态(详见 订阅状态 Subscription States(第九章))，但是让联系人的服务器知道它必须(MUST)不再发送订阅状态变更通知给联系人(见第九章第四节).

5. 联系人的服务器然后 (1) 必须(MUST)发送一个"unsubscribed"类型的出席信息节给用户；并且 (2) 应该(SHOULD)向用户发送它从联系人的所有可用资源收到的不可用出席信息:

```
<presence
    from='contact@example.org'
    to='user@example.com'
    type='unsubscribed'/>
<presence
    from='contact@example.org/resource'
    to='user@example.com'
    type='unavailable'/>
```

6. 当用户的服务器收到"unsubscribed"和"unavailable"类型的出席信息节，它必须(MUST)递送它们给用户:

```
<presence
    from='contact@example.org'
    to='user@example.com'
    type='unsubscribed'/>
<presence
    from='contact@example.org/resource'
    to='user@example.com'
    type='unavailable'/>
```

7. 接收到"unsubscribed"类型的出席信息节之后，用户应该(SHOULD)承认收到了那个订阅状态的通知,要么向联系人发送一个"unsubscribe"类型的出席信息节以证实它,要么向联系人发送一个"subscribe"类型的出席信息节以否认它;这个步骤不影响订阅状态(详见 订阅状态 Subscription States(第九章))，但是让用户的服务器知道它必须(MUST)不在发送订阅状态变更通知给用户(见第九章第四节).

注意：显然这不会导致名册条目从用户的名册移除，并且联系人仍然有一个对用户的出席信息的订阅.为了完全取消双向的订阅并完全从用户的名册中移除名册条目，用户应该(SHOULD)使用 subscription='remove'(定义在 移除一个名册条目并取消所有订阅项 Removing a Roster Item and Cancelling All Subscriptions (第八章第六节))更新名册条目.

## 取消一个订阅项

在批准来自一个用户的任何订阅请求之后的任何时候，一个联系人可以(MAY)取消那个订阅项. 联系人在所有实例中执行这个动作中发送的 XML 是相同的，接下来的

订阅状态根据取消命令发出当时所获得的订阅状态而有所不同. 所有可能的情节描述如下.

## 情形 #1: 当订阅不是相互的时候取消订阅项

在第一种情形下, 用户有一个对联系人的出席信息的订阅但是联系人没有对于用户的出席信息的订阅(换言之, 订阅还不是相互的).

1. 如果联系人想取消用户的订阅项, 联系人必须(MUST)发送一个"unsubscribed"类型的出席信息节给用户:

```
<presence to='user@example.com' type='unsubscribed'/>
```

2. 作为一个结果, 联系人的服务器 (1) 必须(MUST)发送一个名册推送给这个联系人的所有已请求名册的可用资源, 包含一个关于这个用户的更新的名册条目,其'subscription'属性值设为"none"; (2) 必须(MUST)路由这个"unsubscribed"类型的出席信息节给用户(首先把'from'地址设为联系人的纯 JID(<contact@example.org>)); 并且 (3) 应该(SHOULD)向用户发送它从联系人的所有可用资源收到的不可用出席信息:

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='user@example.com'
        subscription='none'
        name='SomeUser'>
      <group>SomeGroup</group>
    </item>
  </query>
</iq>
<presence
    from='contact@example.org'
    to='user@example.com'
    type='unsubscribed'/>
<presence
    from='contact@example.org/resource'
    to='user@example.com'
    type='unavailable'/>
```

3. 接收到指向用户的"unsubscribed"类型的出席信息节之后, 用户的服务器 (1) 必须(MUST)初始化一个名册推送给这个用户的所有已请求名册的可用资源, 包含一个关于这个联系人的名册条目更新,其'subscription'属性值设为"none"(如果用户不可用或未曾请求名册, 用户的服务器必须(MUST)修改这个名册条目并且等下次用户请求名册的时候发送修改过的名册条目); (2) 必须(MUST)递送这个"unsubscribed"状态改变通知给这个用户的所有可用资源; 并且 (3) 必须(MUST)向这个用户的所有可用资源递送不可用出席信息:

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
```

```
            jid='contact@example.org'
            subscription='none'
            name='MyContact'>
          <group>MyBuddies</group>
        </item>
      </query>
    </iq>
    <presence
        from='contact@example.org'
        to='user@example.com'
        type='unsubscribed'/>
    <presence
        from='contact@example.org/resource'
        to='user@example.com'
        type='unavailable'/>
```

4. 接收到"unsubscribed"类型的出席信息节之后，用户应该(SHOULD)承认收到了那个订阅状态通知,要么向联系人发送一个"unsubscribe"出席信息节以证实它,要么向联系人发送一个"subscribe"类型的出席信息节以否认它;这个步骤不影响订阅状态(详见 订阅状态 Subscription States(第九章))，但是让服务器知道它必须(MUST)不再发送订阅状态变更通知给用户(见第九章第四节).

## 情形 #2: 当订阅项是相互的时候取消

在这种情形下，用户有一个对联系人的出席信息的订阅并且联系人也有一个对用户的出席信息的订阅(换言之，订阅是相互的).

1. 如果联系人想取消用户的订阅，联系人必须(MUST)发送一个"unsubscribed"类型的出席信息节给用户:

```
<presence to='user@example.com' type='unsubscribed'/>
```

2. 作为结果，联系人的服务器 (1) 必须(MUST)发送一个名册推送给这个联系人的所有已请求名册的可用资源，包含关于这个用户的一个更新的名册条目,其'subscription'属性值设为"to"; (2) 必须(MUST)路由这个"unsubscribed"类型的出席信息节给用户(首先把'from'地址设为联系人的纯 JID(<contact@example.org>)); 并且 (3) 应该(SHOULD)向这个用户的所有可用资源发送它从联系人的所有可用资源收到的不可用出席信息:

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='user@example.com'
        subscription='to'
        name='SomeUser'>
      <group>SomeGroup</group>
    </item>
  </query>
```

```
</iq>
<presence
      from='contact@example.org'
      to='user@example.com'
      type='unsubscribed'/>
<presence
      from='contact@example.org/resource'
      to='user@example.com'
      type='unavailable'/>
```

3. 接收到指向用户的"unsubscribed"类型出席信息节之后，用户的服务器 (1) 必须(MUST)初始化一个名册推送给这个用户的所有已请求名册的可用资源，包含关于这个联系人的更新的名册条目,其'subscription'属性值设为"from"(如果这个用户不可用或未曾请求名册，用户的服务器必须(MUST)修改这个名册条目并且等下次用户请求名册的时候发送修改过的条目给它)；并且(2) 必须(MUST)递送这个"unsubscribed"状态变更通知给用户的所有可用资源；并且 (3) 必须(MUST)向这个用户的所有可用资源递送这个不可用出席信息:

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='contact@example.org'
        subscription='from'
        name='MyContact'>
      <group>MyBuddies</group>
    </item>
  </query>
</iq>
<presence
      from='contact@example.org'
      to='user@example.com'
      type='unsubscribed'/>
<presence
      from='contact@example.org/resource'
      to='user@example.com'
      type='unavailable'/>
```

4. 接收到这个"unsubscribed"类型的出席信息节之后，用户应该(SHOULD)承认收到了那个订阅状态通知,要么向联系人发送一个"unsubscribe"类型的出席信息节以证实它,要么向联系人发送一个"subscribe"类型的出席信息节以否认它；这一步骤不影响订阅状态(详见 订阅状态 Subscription States (第九章))，但是让用户的服务器知道它必须(MUST)不再发送订阅状态变更通知给用户(见第九章第四节).

注意：显然这不会使得名册条目从联系人的名册中移除，并且联系人仍然有一个对用户的出席信息的订阅. 为了完全双向的取消一个相互的订阅并且从联系人的名册中完全移除这个名册条目，联系人应该以 subscription='remove'(定义在 移除一个名册条目并取消所有订阅项 Removing a Roster Item and Cancelling All Subscriptions (第八章第六节))更新名册条目.

## 移除一个名册条目并取消所有订阅项

因为在双向完整移除一个名册条目和取消所有订阅的过程中可能有很多步骤，名册管理协议包含一个"shortcut"方法来做这件事．无论当前的订阅状态是什么，这个过程可以通过发送一个 roster set(包含一个用于这个联系人的条目,其'subscription'属性值设为"remove")来初始化:

```
<iq type='set' id='remove1'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='contact@example.org'
        subscription='remove'/>
  </query>
</iq>
```

当用户从他或她的名册中移除一个联系人(通过把'subscription'属性值设为"remove"),用户的服务器 (1) 必须(MUST)自动取消用户和联系人之间的任何现存的出席信息订阅项(包括相应的'to'和'from'); (2) 必须(MUST)从用户的名册移除这个名册条目并且通知这个用户的所有已请求名册的可用资源这个名册条目被移除了; (3) 必须(MUST)通知初始化的资源移除成功了; 并且 (4) 应该(SHOULD)向联系人发送它从这个用户的所有可用资源收到的不可用出席信息:

```
<presence
    from='user@example.com'
    to='contact@example.org'
    type='unsubscribe'/>
<presence
    from='user@example.com'
    to='contact@example.org'
    type='unsubscribed'/>
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='contact@example.org'
        subscription='remove'/>
  </query>
</iq>
<iq type='result' id='remove1'/>
<presence
    from='user@example.com/resource'
    to='contact@example.org'
    type='unavailable'/>
```

收到"unsubscribe"类型的出席信息后，联系人的服务器 (1) 必须(MUST)初始化一个名册推送给这个联系人的所有已请求名册的可用资源,包含关于这个用户的一个更新的名册条目,其'subscription'属性值设为"to"(如果这个联系人不可用或未曾请求名册，联系人的服务器必须(MUST)修改这个名册条目并且等下次联系人请求名册的时

候发送这个修改过的条目给它); 并且 (2) 也必须(MUST)递送这个"unsubscribe"状态变更通知给这个联系人的所有可用资源:

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='user@example.com'
        subscription='to'
        name='SomeUser'>
      <group>SomeGroup</group>
    </item>
  </query>
</iq>
<presence
    from='user@example.com'
    to='contact@example.org'
    type='unsubscribe'/>
```

收到这个"unsubscribed"类型的出席信息节之后, 联系人的服务器 (1) 必须(MUST)初始化一个名册推送给这个联系人的所有已请求名册的可用资源,包含一个关于这个用户的更新的名册条目,其'subscription'属性值设为"none"(如果这个联系人不可用或未曾请求名册, 联系人的服务器必须(MUST)修改名册条目并且等下次联系人请求名册的时候把修改过的条目发送给它); 并且 (2) 也必须(MUST)递送这个"unsubscribe"状态改变通知给这个联系人的所有可用资源:

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
    <item
        jid='user@example.com'
        subscription='none'
        name='SomeUser'>
      <group>SomeGroup</group>
    </item>
  </query>
</iq>
<presence
    from='user@example.com'
    to='contact@example.org'
    type='unsubscribed'/>
```

接收到指向联系人的"unavailable"出席信息节之后, 联系人的服务器必须(MUST)递送这个不可用出席信息给这个用户的所有可用资源:

```
<presence
    from='user@example.com/resource'
    to='contact@example.org'
    type='unavailable'/>
```

注意: 当用户从用户的名册中移除联系人的时候, 这个联系人的名册最后状态是用户仍然在联系人名册中但是订阅状态为"none"; 为了完全移除关于这个用户的名册

条目, 联系人也需要发送一个名册移除请求.

# 订阅状态

本章提供关于订阅状态以及和订阅相关的出席信息节(换言之,类型为"subscribe", "subscribed", "unsubscribe",和 "unsubscribed"的出席信息节)的服务器处理过程的详细信息.

## 已定义的状态

有九种可能的订阅状态, 从用户的(不是联系人的)角度描述如下:

1. "None" = 联系人和用户互相没有被对方订阅, 并且也都没有从对方那里请求一个订阅
2. "None + Pending Out" = 联系人和用户互相没有被对方订阅, 用户已经向联系人发送了一个订阅请求但还没有收到回复
3. "None + Pending In" = 联系人和用户互相没有被对方订阅, 联系人已经向用户发送了一个订阅请求但还没有收到回复(注意: 在这种状态下联系人的服务器不应该(SHOULD NOT)推送或递送名册条目, 但是应该(SHOULD)等待,直到联系人的订阅请求已经从用户那里得到批准)
4. "None + Pending Out/In" = 联系人和用户互相没有被对方订阅, 联系人已经向用户发送了一个订阅请求但还没有收到回复, 用户已经向联系人发送了一个订阅请求但还没有收到回复
5. "To" = 用户已订阅联系人(单向)
6. "To + Pending In" = 用户已订阅联系人, 联系人已经向用户发送了一个订阅请求但还没有收到回复
7. "From" = 联系人已订阅用户(单向)
8. "From + Pending Out" = 联系人已订阅用户(单向), 用户已经向联系人发送了一个订阅请求但还没有收到回复
9. "Both" = 用户和联系人互相被对方订阅了(双向)

## 出站出席信息订阅节的服务器处理过程

出站出席信息订阅节使用户能管理他或她对联系人的出席信息的订阅(通过"subscribe"和"unsubscribe"类型), 并且管理联系人对用户的出席信息的访问(通过"subscribed"和"unsubscribed"类型).
因为用户的服务器和联系人的服务器有可能失去对于订阅状态的同步, 用户的服务器必须(MUST)毫无例外地路由所有"subscribe"或"unsubscribe"类型的出站出席信息节给联系人,使用户能在需要的时候重新同步他或她的对联系人的出席信息的订阅.
如果从用户的角度来看,一个"subscribed"或"unsubscribed"类型的出席信息节不会导致一个订阅状态的变更,用户的服务器不应该(SHOULD NOT)路由这个节到联系人那里,并且不能(MUST NOT)做出一个状态变更. 如果这个节导致一个订阅状态的变更,

用户的服务器必须(MUST)路由这个节到联系人,并且必须(MUST)做出相应的状态变更. 这些规则总结如下这些表.

表 1: 推荐的出站"subscribed"节的处理

| 当前状态 | 路由? | 新状态 |
|---|---|---|
| "None" | 否 | 状态不变 |
| "None + Pending Out" | 否 | 状态不变 |
| "None + Pending In" | 是 | "From" |
| "None + Pending Out/In" | 是 | "From + Pending Out" |
| "To" | 否 | 状态不变 |
| "To + Pending In" | 是 | "Both" |
| "From" | 否 | 状态不变 |
| "From + Pending Out" | 否 | 状态不变 |
| "Both" | 否 | 状态不变 |

表 2: 推荐的出站"unsubscribed"节处理

| 当前状态 | 路由? | 新状态 |
|---|---|---|
| "None" | 否 | 状态不变 |
| "None + Pending Out" | 否 | 状态不变 |
| "None + Pending In" | 是 | "None" |
| "None + Pending Out/In" | 是 | "None + Pending Out" |
| "To" | 否 | 状态不变 |
| "To + Pending In" | 是 | "To" |
| "From" | 是 | "None" |
| "From \+ Pending Out" | 是 | "None \+ Pending Out" |
| "Both" | 是 | "To" |

## 入站出席信息订阅节的服务器处理过程

入站出席信息订阅节从用户请求一个订阅相关的动作(通过"subscribe"类型), 通知用户由联系人所做的订阅状态相关的动作(通过"unsubscribe"类型),或使联系人能够管理用户对联系人的出席信息的访问(通过"subscribed"和"unsubscribed"类型).

当用户的服务器为用户从联系人那里接收到一个订阅请求(换言之, 一个"subscribe"类型的出席信息节), 如果用户未曾允许联系人访问用户的出席信息或者没有未决的入站订阅请求, 它必须(MUST)递送那个请求给用户;无论如何, 如果有一个未决的入站订阅请求, 用户的服务器不应该(SHOULD NOT)递送这个新的请求, 因为上一个订阅请求可能已经被记录下来了. 如果用户已经允许联系人访问用户的出席信息,用户的服务器应该(SHOULD)对一个从联系人发来的"subscribe"类型的入站出席信息节自动回复(通过代替用户向联系人发送一个"subscribed"类型的出席信息节); 这个规则使得联系人可以在需要的时候重新同步订阅状态. 这些规则总结如下面这些表.

表 3: 推荐的入站"subscribe"节处理

| 当前状态 | 递送? | 新状态 |
|---|---|---|
| "None" | 是 | "None + Pending In" |
| "None + Pending Out" | 是 | "None + Pending Out/In" |
| "None + Pending In" | 否 | 状态不变 |
| "None + Pending Out/In" | 否 | 状态不变 |
| "To" | 是 | "To + Pending In" |
| "To + Pending In" | 否 | 状态不变 |
| "From" | 否 * | 状态不变 |
| "From + Pending Out" | 否 * | 状态不变 |
| "Both" | 否 * | 状态不变 |

- 服务器应该(SHOULD)以"subscribed"节自动回复

当用户的服务器为用户从联系人那里收到一个"unsubscribe"类型的出席信息节，如果从用户的角度看这个节会导致一个订阅状态变更,那么用户的服务器应该(SHOULD)代替用户自动应答(发送一个"unsubscribed"类型的出席信息节给联系人)，必须(MUST)递送这个"unsubscribe"节给用户,并且必须(MUST)改变状态. 如果不会导致订阅状态变更，用户的服务器不应该(SHOULD NOT)递送这个节并且不能(MUST NOT)改变状态. 这些规则总结如下表.

表 4: 推荐的入站"unsubscribe"节处理

| 当前状态 | 递送? | 新状态 |
|---|---|---|
| "None" | 否 | 状态不变 |
| "None + Pending Out" | 否 | 状态不变 |
| "None + Pending In" | 是 * | "None" |
| "None + Pending Out/In" | 是 * | "None \+ Pending Out" |
| "To" | 否 | 状态不变 |
| "To + Pending In" | 是 * | "To" |
| "From" | 是 * | "None" |
| "From + Pending Out" | 是 * | "None \+ Pending Out |
| "Both" | 是 * | "To" |

- 服务器应该(SHOULD)以"unsubscribed"节自动应答

当用户的服务器为用户从联系人那里收到一个"subscribed"类型的出席信息节，如果没有一个为访问联系人的出席信息的未决的出站请求,它不能(MUST NOT)递送这个节给用户并且不能(MUST NOT)改变订阅状态. 如果有一个为了访问联系人的出席信息的未决的出站请求并且这个"subscribed"类型的入站出席信息请求会导致一个订阅状态的改变,用户的服务器必须(MUST)递送这个节给用户并且必须(MUST)改变订阅状态. 如果用户已经有授权可以访问联系人的出席信息，这个"subscribed"类型的入站出席信息节不导致一个订阅状态的变更;从而用户的服务器不应该(SHOULD NOT)递送这个节给用户并且不能(MUST NOT)改变订阅状态. 这些规则总结如下表.

表 5: 推荐的入站"subscribed"节处理

| 当前状态 | 递送? | 新状态 |
|---|---|---|
| "None" | 否 | 状态不变 |
| "None + Pending Out" | 是 | "To" |
| "None + Pending In" | 否 | 状态不变 |
| "None + Pending Out/In" | 是 | "To \+ Pending In" |
| "To" | 否 | 状态不变 |
| "To + Pending In" | 否 | 状态不变 |
| "From" | 否 | 状态不变 |
| "From + Pending Out" | 是 | "Both" |
| "Both" | 否 | 状态不变 |

当用户的服务器为用户从联系人那里收到了一个"unsubscribed"类型的出席信息节,如果有一个为了访问联系人的出席信息的未决的出站请求或者用户当前已经有授权可以访问联系人的出席信息,它必须(MUST)递送这个节给用户并且必须(MUST)改变订阅状态. 否则, 用户的服务器不应该(SHOULD NOT)递送这个节并且不能(MUST NOT)改变订阅状态. 这些规则总结如下表.

表 6: 推荐的入站"unsubscribed"节处理

| 当前状态 | 递送? | 新状态 |
|---|---|---|
| "None" | 否 | 状态不变 |
| "None + Pending Out" | 是 | "None" |
| "None + Pending In" | 否 | 状态不变 |
| "None + Pending Out/In" | 是 | "None \+ Pending In" |
| "To" | 是 | "None" |
| "To + Pending In" | 是 | "None \+ Pending In" |
| "From" | 否 | 状态不变 |
| "From + Pending Out" | 是 | "From" |
| "Both" | 是 | "From" |

## 服务器递送和客户端承认订阅请求以及状态变更通知

当一个服务器收到一个"subscribe"类型的入站出席信息节(换言之, 一个订阅请求)或"subscribed"类型,"unsubscribe"类型, 或"unsubscribed"类型(换言之, 一个订阅状态变更通知), 除了发送适当的名册推送(或当下次名册被一个可用资源请求时发送更新的名册), 它必须(MUST)递送这个请求或通知给预定的接收者至少一次. 一个服务器可以(MAY)要求接收者的回执以承认接收到了所有状态变更通知(并且必须(MUST)要求承认订阅请求的情形, 换言之,类型的出席信息节"subscribe"). 为了要求回执, 一个服务器应该(SHOULD)在每次接收者登陆的时候发送这个请求或通知给它, 直到这个接收者承认收到这个通知(通过证实"affirming"或禁止"denying"这个通知),如下表:

表 7: 订阅状态变更通知的承认

| 节类型 | 接受 | 禁止 |
|---|---|---|
| subscribe | subscribed | unsubscribed |
| subscribed | subscribe | unsubscribe |
| unsubscribe | unsubscribed | subscribed |
| unsubscribed | unsubscribe | subscribe |

显然, 根据前述的订阅状态图表, 一些回执节将被路由到联系人并且导致状态的变更, 而其他的则不会. 无论如何, 任何这样的节必须(MUST)导致服务器不再发送订阅状态变更通知给用户.

因为在接收到 roster set(其'subscription'属性值设为"remove"(见 移除一个名册条目并且取消所有订阅项 Removing a Roster Item and Cancelling All Subscriptions (第八章第六节)))之后,用户的服务器必须(MUST)自动生成"unsubscribe"和"unsubscribed"类型的出站出席信息节,服务器必须(MUST)把一个名册移除请求视为发送所有这些出席信息节,以决定是否继续向用户发送"subscribe"或"subscribed"类型的订阅状态变更通知.

# 屏蔽通信

大多数即时消息系统已发现有必要实现一些方法来为用户屏蔽来自某些特定的其他用户的通信(这在\[IMP-REQS\]的第五章第一节第五小节, 第五章第一节第十五小节, 第五章第三节第二小节, 和第五章第四节第十小节中也有要求). 在 XMPP 中这是由管理某人的隐私列表来实现的(使用'jabber:iq:privacy'名字空间).

服务器端的隐私列表使得以下用例能够完成:

- 接收某人的隐私列表.
- 增加, 移除, 和 编辑某人的隐私列表.
- 设置, 改变, 或 取消 激活的列表.
- 设置, 改变, 或 取消 缺省的列表 (换言之, 缺省激活的那个列表).
- 基于 JID, group, 或 subscription 类型(或全局的) 允许或屏蔽消息.
- 允许或屏蔽入站出席信息通知,基于 JID, group, 或 subscription 类型 (或全局的).
- 允许或屏蔽出站出席信息通知,基于 JID, group, 或 subscription 类型 (或全局的).
- 允许或屏蔽 IQ 节, 基于 JID, group,或 subscription 类型(或全局的).
- 允许或屏蔽所有通信, 基于 JID, group, 或 subscription 类型(或全局的).

注意: 出席信息通知不包括出席信息订阅,只包括广播给那些已订阅了用户出席信息的实体的出席信息. 因而这包括没有'type'属性或只包含 type='unavailable'的的出席信息节.

## 语法和语义

一个用户可以(MAY)定义一个或更多的隐私列表, 它们由用户的服务器保存. 每个<list/>元素包含一个或多个格式为<item/>元素的规则, 并且每个<item/>元素使用属性来定义一个隐私规则类型, 一个适用于规则的特定值, 相应的动作, 和处理顺序相应的条目位置.
语法如下:

```
<iq>
  <query xmlns='jabber:iq:privacy'>
    <list name='foo'>
      <item
          type='[jid|group|subscription]'
          value='bar'
          action='[allow|deny]'
            order='unsignedInt'>
          [<message/>]
          [<presence-in/>]
          [<presence-out/>]
          [<iq/>]
      </item>
    </list>
  </query>
</iq>
```

如果类型是"jid", 那么'value'属性必须(MUST)包含一个合法的 Jabber ID. JIDs 应该(SHOULD)满足以下顺序:

1. <user@domain/resource> (仅为匹配的资源)
2. <user@domain> (任何匹配的资源)
3. <domain/resource> (仅匹配的资源)
4. <domain> (匹配这个域本身, 正如任何 user@domain, domain/resource, 或 包含一个子域的地址)

如果类型为"group", 那么'value'属性应该(SHOULD)包含组在用户的名册中的名字. (如果一个客户端尝试更新, 新建, 或删除一个不在用户名册中的组的列表条目, 服务器应该(SHOULD)返回给客户端一个<item-not-found/>节错误.)
如果类型是"subscription", 那么'value'属性必须(MUST)是"both", "to", "from", 或 "none" (定义在 名册语法和语义 Roster Syntax and Semantics (第七章第一节))中的一个人, 在这里 "none" 包括对于用户来说完全未知和根本不在用户名册中的实体.
如果没有包含'type'属性, 这个规则提供 失败"fall-through" 情景.
'action'属性必须(MUST)被包含并且它的值必须(MUST)是 允许"allow"或 禁止"deny".

'order'属性必须(MUST)被包含并且它的值必须(MUST)是一个在列表的所有条目中具有唯一性的非负整数. (如果一个客户端尝试以一个非唯一的 order 值建立或更新一个列表, 服务器必须(MUST)返回给客户端一个<bad-request/>节错误.)
<item/>元素可以(MAY)包含一个或更多子元素,使得一个实体可以指明更多的细微控制包括屏蔽哪些种类的节(换言之, 不只是简单地屏蔽所有节). 允许的子元素包括:

- <message/> \-\- 屏蔽引入消息节
- <iq/> \-\- 屏蔽引入 IQ 节
- <presence-in/> \-\- 屏蔽引入出席信息通知
- <presence-out/> \-\- 屏蔽外出出席信息通知

在"jabber:iq:privacy'名字空间之内, 一个"set"类型的 IQ 节的<query/>子元素不能(MUST NOT)包含超过一个子元素(换言之, 这个节必须(MUST)只包含一个<active/>元素, 一个<default/>元素, 或一个<list/>元素); 如果一个发送中的实体违反了这个规则, 接收中的实体必须(MUST)返回一个 return a <bad-request/>节错误.
当一个客户端增加或更新一个隐私列表, <list/>元素应该(SHOULD)包含至少一个<item/>子元素; 当一个客户端移除一个隐私列表的时候, <list/>元素不能(MUST NOT)包含任何<item/>子元素.
当一个客户端更新一个隐私列表的时候, 它必须包含所有想得到的条目(换言之, 不是一个"delta").

## 商业规则

1. 如果有一个为某会话设置的激活的列表, 它只影响为其激活的那个会话, 并且只在那一个会话的持续期间有效; 服务器必须(MUST)只应用激活列表,并且不能(MUST NOT)应用缺省列表(换言之, 列表没有层次"layering").
2. 缺省列表总体适用于用户, 并且如果没有为一个节指向的目标 session/resource 设置激活列表,或用户当前没有会话,它会被处理.
3. 如果没有为一个会话设置激活列表(或用户当前没有会话), 并且没有缺省列表, 那么所有节应该被(SHOULD BE)接受或由服务器代替用户做适当的处理(遵守用于处理 XML 节的服务器规则 Server Rules for Handling XML Stanzas (第十一章)).
4. 隐私列表必须(MUST)是第一个由服务器应用的递送规则, 替代 (1) 定义在 用于处理 XML 节的服务器规则 Server Rules for Handling XML Stanzas (第十一章)的路由和递送规则, 以及 (2)和订阅相关的出席信息节的处理(和相应的名册推送的生成) 定义在 名册条目和出席信息订阅的集成 Integration of Roster Items and Presence Subscriptions (第八章).
5. 服务器处理隐私列表条目的顺序是很重要的. 列表条目必须(MUST)按照每个<item/>的'order'属性的整数值的升序来处理.
6. 一旦节和一个隐私列表规则匹配, 服务器必须(MUST)按照这个规则适当地处理这个节,然后终止处理.
7. 如果在一个列表中没有提供一个 fall-through 的条目, fall-through 动作被假定为 允许"allow".

8. 如果一个用户为一个激活列表更新定义，之后的基于那个激活列表的操作必须(MUST)使用更新的定义(为了那些激活列表正应用的所有资源).
9. 如果在用户的会话期间,在激活的或缺省的列表中定义的名册条目中的订阅状态改变了,或名册组改变了,接下来基于那个列表的处理必须(MUST)考虑计入这个已改变的状态或组(对那个列表当前应用的所有资源).
10. 当一个规则的定义修改了的时候，服务器必须(MUST)发送一个类型为"set"的 IQ 节给所有已连接的资源，包括一个只有一个<list/>子元素的<query/>元素，其'name'属性设为已修改的隐私列表的名字. 这些 隐私列表推送("privacy list pushes")遵守和用于名册管理的名册推送("roster pushes")同样的语义 ，除了被推送给已连接的资源的列表名字本身(不是完整的列表定义或那个"delta"). 是否接受这个修改了的列表定义最终由接收中的资源来决定，尽管如果这个列表正在应用于一个已连接的资源,它应该(SHOULD)这样做.
11. 当一个已连接的资源尝试移除一个列表或指定一个新的缺省列表,而那个列表应用于一个已连接的资源而不是正在发送的资源，服务器必须(MUST)返回一个<conflict/>错误给发送中的资源并且不能(MUST NOT)执行这个请求的改变.

## 接收某人的隐私列表

例子: 客户端向服务器请求隐私列表的名字:

```
<iq from='romeo@example.net/orchard' type='get' id='getlist1'>
  <query xmlns='jabber:iq:privacy'/>
</iq>
```

例子: 服务器发送隐私列表的名字给客户端，激活列表和缺省列表放在前面:

```
<iq type='result' id='getlist1' to='romeo@example.net/orchard'>
  <query xmlns='jabber:iq:privacy'>
    <active name='private'/>
    <default name='public'/>
    <list name='public'/>
    <list name='private'/>
    <list name='special'/>
  </query>
</iq>
```

例子: 客户端向服务器请求一个隐私列表:

```
<iq from='romeo@example.net/orchard' type='get' id='getlist2'>
  <query xmlns='jabber:iq:privacy'>
    <list name='public'/>
  </query>
</iq>
```

例子: 服务器发送一个隐私列表给客户端:

```
<iq type='result' id='getlist2' to='romeo@example.net/orchard'>
  <query xmlns='jabber:iq:privacy'>
    <list name='public'>
```

```
        <item type='jid'
              value='tybalt@example.com'
              action='deny'
              order='1'/>
        <item action='allow' order='2'/>
      </list>
    </query>
  </iq>
```

例子：客户端向服务器请求另一个隐私列表：

```
  <iq from='romeo@example.net/orchard' type='get' id='getlist3'>
    <query xmlns='jabber:iq:privacy'>
      <list name='private'/>
    </query>
  </iq>
```

例子：服务器发送另一个隐私列表给客户端：

```
  <iq type='result' id='getlist3' to='romeo@example.net/orchard'>
    <query xmlns='jabber:iq:privacy'>
      <list name='private'>
        <item type='subscription'
              value='both'
              action='allow'
              order='10'/>
        <item action='deny' order='15'/>
      </list>
    </query>
  </iq>
```

例子：客户端再向服务器请求另一个隐私列表：

```
  <iq from='romeo@example.net/orchard' type='get' id='getlist4'>
    <query xmlns='jabber:iq:privacy'>
      <list name='special'/>
    </query>
  </iq>
```

例子：服务器再发送另一个隐私列表给客户端：

```
  <iq type='result' id='getlist4' to='romeo@example.net/orchard'>
    <query xmlns='jabber:iq:privacy'>
      <list name='special'>
        <item type='jid'
              value='juliet@example.com'
              action='allow'
              order='6'/>
        <item type='jid'
              value='benvolio@example.org'
              action='allow'
              order='7'/>
```

```
        <item type='jid'
              value='mercutio@example.org'
              action='allow'
              order='42'/>
        <item action='deny' order='666'/>
      </list>
    </query>
  </iq>
```

在这个例子中，用户有三个列表：(1) 'public'，它允许所有人的通信,除了一个指定的实体(这是缺省列表)；(2) 'private'，它只允许和这个用户有双向订阅的联系人的通信(这是激活的列表)；还有 (3) 'special'，它只允许三个指定的实体通信.

如果用户尝试接收一个列表但是这个列表的名字不存在，服务器必须(MUST)返回一个<item-not-found/>节错误给用户：

例子：客户端尝试接收不存在的列表：

```
<iq to='romeo@example.net/orchard' type='error' id='getlist5'>
  <query xmlns='jabber:iq:privacy'>
    <list name='The Empty Set'/>
  </query>
  <error type='cancel'>
    <item-not-found
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

用户在一次只被允许接收一个列表. 如果用户尝试同一个请求中接收超过一个列表，服务器必须(MUST)返回一个<bad request/>节错误给用户：

例子：客户端尝试接收多于一个的列表：

```
<iq to='romeo@example.net/orchard' type='error' id='getlist6'>
  <query xmlns='jabber:iq:privacy'>
    <list name='public'/>
    <list name='private'/>
    <list name='special'/>
  </query>
  <error type='modify'>
    <bad-request
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

## 管理激活列表

为了设置或改变服务器当前应用的激活列表，用户必须(MUST)发送一个类型为"set"的 IQ 节,包含一个符合'jabber:iq:privacy'名字空间的<query/>元素,这个元素<query/>又包含一个空的<active/>子元素,而这个<active/>拥有一个'name'属性,'name'属性值则设为期望的列表名.

例子: 客户端请求激活列表变更:

```
<iq from='romeo@example.net/orchard' type='set' id='active1'>
  <query xmlns='jabber:iq:privacy'>
    <active name='special'/>
  </query>
</iq>
```

服务器必须(MUST)在返回结果给客户端之前激活并应用这个已请求的列表.

例子: 服务器承认激活列表变更成功:

```
<iq type='result' id='active1' to='romeo@example.net/orchard'/>
```

如果用户尝试设置一个激活列表但是列表名不存在, 服务器必须(MUST)返回一个<item-not-found/>节错误给用户:

例子: 客户端尝试设置一个不存在的列表作为激活列表:

```
<iq to='romeo@example.net/orchard' type='error' id='active2'>
  <query xmlns='jabber:iq:privacy'>
    <active name='The Empty Set'/>
  </query>
  <error type='cancel'>
    <item-not-found
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

为了取消使用任何激活列表, 已连接的资源必须(MUST)发送一个空的<active/>元素, 并且不带'name'属性.

例子: 客户端取消使用激活列表:

```
<iq from='romeo@example.net/orchard' type='set' id='active3'>
  <query xmlns='jabber:iq:privacy'>
    <active/>
  </query>
</iq>
```

例子: 服务器承认取消任何激活列表成功:

```
<iq type='result' id='active3' to='romeo@example.net/orchard'/>
```

## 管理缺省列表

为了改变它的缺省列表(它对用户来说是全局应用的, 不只是发送中的资源), 用户必须(MUST)发送一个类型为"set"的 IQ 节, 它包含一个符合'jabber:iq:privacy'名字空间的<query/>元素,这个<query/>元素包含一个空的<default/>子元素,这个<default/>子元素拥有一个'name'属性,这个'name'属性的值设为期望的列表名.

例子: 用户请求变更缺省列表:

```
<iq from='romeo@example.net/orchard' type='set' id='default1'>
  <query xmlns='jabber:iq:privacy'>
    <default name='special'/>
  </query>
</iq>
```

例子: 服务器承认缺省列表变更成功:

```
<iq type='result' id='default1' to='romeo@example.net/orchard'/>
```

如果用户尝试变更一个缺省列表但是这个缺省列表正在由至少一个已连接的但不是当前发送中的这个资源使用着,服务器必须(MUST)返回一个<conflict/>节错误给发送中的资源:

例子: 客户端尝试改变一个缺省列表但是这个列表正在被另一个资源使用:

```
<iq to='romeo@example.net/orchard' type='error' id='default1'>
  <query xmlns='jabber:iq:privacy'>
    <default name='special'/>
  </query>
  <error type='cancel'>
    <conflict
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

如果用户尝试设置一个缺省列表但是这个列表的名字不存在, 服务器必须(MUST)返回一个<item-not-found/>节错误给用户:

例子: 客户端尝试设置一个不存在的列表作为缺省列表:

```
<iq to='romeo@example.net/orchard' type='error' id='default1'>
  <query xmlns='jabber:iq:privacy'>
    <default name='The Empty Set'/>
  </query>
  <error type='cancel'>
    <item-not-found
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

为了取消使用缺省列表(换言之, 任何时候都使用域的节路由规则), 用户必须(MUST)发送一个空的不带'name'属性的<default/>元素.

例子: 客户端取消使用缺省列表:

```
<iq from='romeo@example.net/orchard' type='set' id='default2'>
  <query xmlns='jabber:iq:privacy'>
    <default/>
  </query>
</iq>
```

例子: 服务器承认成功地取消了任何缺省列表:

```
<iq type='result' id='default2' to='romeo@example.net/orchard'/>
```

如果一个已连接的资源尝试取消一个用户全局的缺省列表但是这个缺省列表正在应用于另一个已连接的资源,服务器必须(MUST)返回一个<conflict/>错误给发送中的资源:

例子: 客户端尝试取消一个缺省列表但是这个列表正在被另一个资源使用:

```
<iq to='romeo@example.net/orchard' type='error' id='default3'>
  <query xmlns='jabber:iq:privacy'>
    <default/>
```

```
    </query>
    <error type='cancel'>
      <conflict
          xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    </error>
  </iq>
```

## 编辑一个隐私列表

为了编辑一个隐私列表，用户必须(MUST)一个类型为"set"的 IQ 节,包含一个符合'jabber:iq:privacy'名字空间的<query/>元素,这个<query/>元素包含一个拥有一个<list/>子元素,这个<list/>子元素拥有一个'name'属性,这个'name'属性的值设为用户想编辑的列表名. 这个<list/>元素必须(MUST)包含一个或多个<item/>元素，它们包含了列表中的所有元素以指明用户期望的对列表的变更(不是 the "delta").

例子: 客户端编辑隐私列表:

```
<iq from='romeo@example.net/orchard' type='set' id='edit1'>
  <query xmlns='jabber:iq:privacy'>
    <list name='public'>
      <item type='jid'
            value='tybalt@example.com'
            action='deny'
            order='3'/>
      <item type='jid'
            value='paris@example.org'
            action='deny'
            order='5'/>
      <item action='allow' order='68'/>
    </list>
  </query>
</iq>
```

例子: 服务器承认列表编辑成功:

```
<iq type='result' id='edit1' to='romeo@example.net/orchard'/>
```

注意: 任何给定的条目的'order'属性值不是固定的. 因而在前述的例子中如果用户想在"tybalt@example.com"条目和"paris@example.org"条目之间增加 4 个条目，用户的客户端必须(MUST)在向服务器提交列表之前对相关的条目重新编号.

服务器必须(MUST)现在发送一个 隐私列表推送"privacy list push"给所有已连接的资源:

例子: 基于列表编辑的隐私列表推送:

```
<iq to='romeo@example.net/orchard' type='set' id='push1'>
  <query xmlns='jabber:iq:privacy'>
    <list name='public'/>
  </query>
</iq>
<iq to='romeo@example.net/home' type='set' id='push2'>
```

```
  <query xmlns='jabber:iq:privacy'>
    <list name='public'/>
  </query>
</iq>
```

按照定义在[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]的 IQ 节语义，每个已连接的子元素必须(MUST)返回一个如下的 IQ result 给服务器:

例子: 承认收到一个隐私列表推送:

```
<iq from='romeo@example.net/orchard'
    type='result'
    id='push1'/>
<iq from='romeo@example.net/home'
    type='result'
    id='push2'/>
```

## 增加一个新的隐私列表

增加一个新的列表和编辑一个现有的列表使用的协议是相同的. 如果列表名和现有的列表名吻合，这个增加新列表的请求将复写那个旧的列表. 正如编辑列表一样，服务器也必须(MUST)发送一个 隐私列表推送"privacy list push" 给所有已连接的资源.

## 移除一个隐私列表

为了移除一个隐私列表，用户必须(MUST)发送一个类型为"set"的 IQ 节,包含一个符合'jabber:iq:privacy'名字空间的<query/>元素,这个<query/>元素包含一个空的<list/>子元素,这个<list/>子元素拥有一个'name'属性,这个'name'属性的值设为用户想移除的列表名.

例子: 客户端移除一个隐私列表:

```
<iq from='romeo@example.net/orchard' type='set' id='remove1'>
  <query xmlns='jabber:iq:privacy'>
    <list name='private'/>
  </query>
</iq>
```

例子: 服务器承认成功地移除列表:

```
<iq type='result' id='remove1' to='romeo@example.net/orchard'/>
```

如果一个用户尝试移除一个列表而这个列表正在被应用于至少一个和发送中的资源不同的已连接的资源，服务器必须(MUST)返回一个<conflict/>节错误给用户; 换言之，用户在尝试移除它之前必须(MUST)先设置另一个列表成为激活或缺省列表. 如果用户尝试移除一个列表但是列表名字不存在，服务器必须(MUST)返回一个<item-not-found/>节错误给用户. 如果用户尝试在同一个请求中移除超过一个的列表，服务器必须(MUST)反回一个<bad request/>节错误给用户.

## 屏蔽消息

服务器端的隐私列表使得一个用户可以基于实体的JID,名册组,或订阅状态(或全局地)来屏蔽从其他实体引入的消息. 以下例子阐明这个协议. (注意: 为了精简, "result"类型的 IQ 节没有在以下例子中显示, 隐私列表推送也没有显示.)

例子: 基于 JID 的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='msg1'>
  <query xmlns='jabber:iq:privacy'>
    <list name='message-jid-example'>
      <item type='jid'
            value='tybalt@example.com'
            action='deny'
            order='3'>
        <message/>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果, 用户将不会接收到从特定 JID 发来的消息.

例子: 基于名册组的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='msg2'>
  <query xmlns='jabber:iq:privacy'>
    <list name='message-group-example'>
      <item type='group'
            value='Enemies'
            action='deny'
            order='4'>
        <message/>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果, 用户将不会收到从指定名册组中的任何实体发来的消息.

例子: 基于订阅状态的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='msg3'>
  <query xmlns='jabber:iq:privacy'>
    <list name='message-sub-example'>
      <item type='subscription'
            value='none'
            action='deny'
            order='5'>
        <message/>
```

```
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果，用户将不会收到从任何指定订阅状态的实体发来的消息.

例子：全局的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='msg4'>
  <query xmlns='jabber:iq:privacy'>
    <list name='message-global-example'>
      <item action='deny' order='6'>
        <message/>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果，用户将不会收到从任何其他用户发来的消息.

## 屏蔽入站出席信息通知

服务器端的隐私列表使得用户可以基于实体的 JID,名册组,或订阅状态(或全局地)屏蔽来自其他实体的入站出席信息通知. 以下例子阐明了这个协议.

注意：出席信息通知不包括出席信息订阅,只是把出席信息广播给当前已订阅某个联系人的出席信息的用户. 所以它只包括没有'type'属性的或 type='unavailable'的出席信息节.

例子：基于 JID 的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='presin1'>
  <query xmlns='jabber:iq:privacy'>
    <list name='presin-jid-example'>
      <item type='jid'
            value='tybalt@example.com'
            action='deny'
            order='7'>
        <presence-in/>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果，用户将不会收到从指定 JID 发来的出席信息通知.

例子：基于名册组的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='presin2'>
  <query xmlns='jabber:iq:privacy'>
    <list name='presin-group-example'>
      <item type='group'
```

```
                value='Enemies'
                action='deny'
                order='8'>
            <presence-in/>
          </item>
        </list>
      </query>
    </iq>
```

作为建立和应用前述列表的结果，用户将不会从指定的名册组中的任何实体收到出席信息通知.

例子：基于订阅状态的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='presin3'>
  <query xmlns='jabber:iq:privacy'>
    <list name='presin-sub-example'>
      <item type='subscription'
                value='to'
                action='deny'
                order='9'>
          <presence-in/>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果，用户将不会从指定订阅状态的任何实体收到出席信息通知.

例子：全局的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='presin4'>
  <query xmlns='jabber:iq:privacy'>
    <list name='presin-global-example'>
      <item action='deny' order='11'>
        <presence-in/>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果，用户将不会从任何其他实体收到出席信息通知.

## 评比出站出席信息通知

服务器端的隐私列表使用户能够屏蔽发出到其他实体的出席信息通知(基于实体的 JID，名册组，或订阅状态 (或全局的)). 以下例子阐明了这个协议.

注意：出席信息通知不包括出席信息订阅,只把出席信息广播给已订阅了用户的出席信息的联系人.所以 只包括没有'type'属性或 type='unavailable'的出席信息节.

例子：基于 JID 的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='presout1'>
  <query xmlns='jabber:iq:privacy'>
    <list name='presout-jid-example'>
      <item type='jid'
            value='tybalt@example.com'
            action='deny'
            order='13'>
        <presence-out/>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果，用户将不会给指定 JID 发送出席信息通知.

例子：基于名册组的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='presout2'>
  <query xmlns='jabber:iq:privacy'>
    <list name='presout-group-example'>
      <item type='group'
            value='Enemies'
            action='deny'
            order='15'>
        <presence-out/>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果，用户将不会向指定名册组的任何实体发送出席信息通知.

例子：基于订阅状态的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='presout3'>
  <query xmlns='jabber:iq:privacy'>
    <list name='presout-sub-example'>
      <item type='subscription'
            value='from'
            action='deny'
            order='17'>
        <presence-out/>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果，用户将不会向指定订阅状态的任何实体发送出席信息通知.

例子：全局的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='presout4'>
  <query xmlns='jabber:iq:privacy'>
    <list name='presout-global-example'>
      <item action='deny' order='23'>
        <presence-out/>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果，用户将不会向任何其他用户发送出席信息通知.

## 屏蔽 IQ 节

服务器端的隐私列表使用户能够屏蔽从其他实体进来的 IQ 节(基于实体的 JID,名册组，或订阅状态(或全局地)). 以下例子阐明了这个协议.

例子: 基于 JID 的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='iq1'>
  <query xmlns='jabber:iq:privacy'>
    <list name='iq-jid-example'>
      <item type='jid'
            value='tybalt@example.com'
            action='deny'
            order='29'>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用这个列表的结果，用户将不会收到从指定 JID 发来的 IQ 节.

例子: 基于名册组的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='iq2'>
  <query xmlns='jabber:iq:privacy'>
    <list name='iq-group-example'>
      <item type='group'
            value='Enemies'
            action='deny'
            order='31'>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果，用户将不会收到从指定名册组的任何实体发来的 IQ 节.

例子: 基于订阅状态的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='iq3'>
```

```
  <query xmlns='jabber:iq:privacy'>
    <list name='iq-sub-example'>
      <item type='subscription'
            value='none'
            action='deny'
            order='17'>
       </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果，用户将不会收到从指定订阅状态的任何实体发来的 IQ 节.

例子：全局的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='iq4'>
  <query xmlns='jabber:iq:privacy'>
    <list name='iq-global-example'>
      <item action='deny' order='1'>
      </item>
    </list>
  </query>
</iq>
```

作为建立和应用前述列表的结果，用户将不会收到从任何其他用户发来的 IQ 节.

## 屏蔽所有通信

服务器端的隐私列表使用户能够基于其他实体的 JID,名册组,或订阅状态(或全局的)屏蔽所有进来和出去的节. 注意那部包括订阅相关的出席信息节，它们被排除在外(定义在 屏蔽入站出席信息通知 Blocking Inbound Presence Notifications (第十章第十节)). 以下例子阐明了这个协议.

例子：基于 JID 的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='all1'>
  <query xmlns='jabber:iq:privacy'>
    <list name='all-jid-example'>
      <item type='jid'
            value='tybalt@example.com'
            action='deny'
            order='23'/>
    </list>
  </query>
</iq>
```

作为建立和应用这个列表的结果，用户将不会收到和发送任何通信给指定 JID.

例子：基于名册组的用户屏蔽:

```
<iq from='romeo@example.net/orchard' type='set' id='all2'>
  <query xmlns='jabber:iq:privacy'>
```

```
      <list name='all-group-example'>
        <item type='group'
              value='Enemies'
              action='deny'
              order='13'/>
      </list>
    </query>
  </iq>
```

作为建立和应用前述列表的结果，用户将不会收到和发送和指定名册组的任何实体的通信.

例子：基于订阅状态的用户屏蔽:

```
  <iq from='romeo@example.net/orchard' type='set' id='all3'>
    <query xmlns='jabber:iq:privacy'>
      <list name='all-sub-example'>
        <item type='subscription'
              value='none'
              action='deny'
              order='11'/>
      </list>
    </query>
  </iq>
```

作为建立和应用前述列表的结果，用户将不会收到和发送和指定订阅状态的任何实体的通信.

例子：全局的用户屏蔽:

```
  <iq from='romeo@example.net/orchard' type='set' id='all4'>
    <query xmlns='jabber:iq:privacy'>
      <list name='all-global-example'>
        <item action='deny' order='7'/>
      </list>
    </query>
  </iq>
```

作为建立和应用前述列表的结果，用户将不会收到和发送和任何其他用户的通信.

## 已被屏蔽的实体尝试和用户通信

如果一个已被屏蔽的实体尝试发送消息或出席信息给用户，用户的服务器应该(SHOULD)安静的丢掉这个节并且不能(MUST NOT)返回一个错误给发送的实体.

如果一个已被屏蔽的实体尝试发送一个类型为"get"或"set"的 IQ 节给用户，用户的服务器必须(MUST)给发送的实体一个<service-unavailable/>节错误，因为这是一个客户端不理解 IQ get 或 set 的名字空间的时候所发送的标准错误码. 其他类型的 IQ 节应该(SHOULD)被服务器安静的丢弃.

例子：已被屏蔽的实体尝试发送 IQ get:

```
  <iq type='get'
      to='romeo@example.net'
```

```
      from='tybalt@example.com/pda'
      id='probing1'>
    <query xmlns='jabber:iq:version'/>
  </iq>
```

例子: 服务器返回一个错误给已被屏蔽的实体:

```
  <iq type='error'
      from='romeo@example.net'
      to='tybalt@example.com/pda'
      id='probing1'>
    <query xmlns='jabber:iq:version'/>
    <error type='cancel'>
      <service-unavailable
          xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    </error>
  </iq>
```

## 高级启发

当建立一个高级隐私启发的表达式的时候，客户端应该(SHOULD)使用尽可能简单的表达式.

例如，启发 "屏蔽不在我名册中的任何用户的通信" 可以使用以下任何一种方式来构造:

- 允许任何来自我的名册中的 JID 的通信 (换言之，列出每个 JID 成为单独的列表条目)，但是屏蔽和其他任何人的通信
- 允许任何来自我的名册的某个组中的用户的通信(换言之，列出每个组作为单独的条目)，但是屏蔽和任何其他人的通信
- 允许任何我的他(她)之间的订阅状态为'both'或'to'或'from'的用户的通信(换言之，单独列出每个订阅状态值)，但是屏蔽和任何其他人的通信
- 屏蔽和任何订阅状态为'none'的用户的通信

最后一个表达式是最简单的并且应该(SHOULD)被使用；这种情形下将被发送的 XML 如下:

```
  <iq type='set' id='heuristic1'>
    <query xmlns='jabber:iq:privacy'>
      <list name='heuristic-example'>
        <item type='subscription'
              value='none'
              action='deny'
              order='437'/>
      </list>
```

```
    </query>
  </iq>
```

# 服务器处理 XML 节的规则

用于服务器的基本路由和递送规则定义在[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]中. 本章定义附加的用于 XMPP 兼容的即时消息和出席信息服务器规则.

## 入站节

如果一个入站的节的'to'属性的 JID 中的域标识符部分的主机名和服务器自身的主机名相同并且'to'属性的 JID 的格式是 <user@example.com> 或 <user@example.com/resource>, 服务器必须(MUST)首先强制应用任何隐私列表(第十章),然后服从以下定义的规则:

1. 如果 JID 的格式是<user@domain/resource>并且有一个可用的资源和这个全 JID 吻合, 接受这得服务器必须(MUST)递送这个节给那个资源.
2. 然后如果 JID 的格式是<user@domain>或格式是<user@domain/resource>,并且和用户相关的帐号不存在, 接收者的服务器 (a) 如果它是一个出席信息节, 应该(SHOULD) 安静的忽略这个节(换言之,既不递送它也不返回一个错误), (b) 如果它是一个 IQ 节,必须(MUST)返回一个<service-unavailable/>节错误给发送者, 并且 (c) 如果它是一个消息节,应该(SHOULD)返回一个<service-unavailable/>节错误给发送者.
3. 然后如果 JID 的格式是<user@domain/resource>并且没有可用的资源和它的全 JID 匹配, 接收者的服务器 (a) 如果它是一个出席信息节,应该(SHOULD)安静地忽略这个节(换言之, 既不递送它也不返回一个错误), (b) 如果它是一个 IQ 节,必须(MUST)返回一个<service-unavailable/>节错误给发送者, 并且 (c) 如果它是一个消息节,应该(SHOULD)把这个节视为发往<user@domain>.
4. 然后如果 JID 的格式是<user@domain>并且这个用户至少有一个可用的资源, 接收者的服务器必须(MUST)遵守以下规则:
    1. 对于消息节, 服务器应该(SHOULD)递送这个节给高优先级的可用资源(如果这个资源没有提供<priority/>元素的值, 服务器应该(SHOULD)认为它提供的值为零). 如果两个或更多的可用资源有相同的优先级, 服务器可以(MAY)使用一些其他的规则(例如, 最近的连接时间, 最近的活动时间, 或由一些<show/>值的层次所定义的最高的可用性)来从它们中间选择,或可以(MAY)递送这个消息到所有这些资源. 无论如何, 服务器不能(MUST NOT)这个节到一个优先级为负数的可用资源; 如果唯一的一个可用资源的优先级是负数, 服务器应该(SHOULD)当成没有可用资源一样处理这个消息(定义在后面). 另外, 服务器不能(MUST NOT)重写'to'属性(换言之, 它必须(MUST)让它保持<user@domain>而不是改成<user@domain/resource>).

2. 对于类型不是"probe"的出席信息节，服务器必须(MUST)递送这个节给所有可用的资源;对于出席信息调查，服务器应该(SHOULD)基于定义在 出席信息调查 Presence Probes (第五章第一节第三小节)的规则来应答. 另外，服务器不能(MUST NOT)重写'to'属性(换言之，它必须(MUST)保持<user@domain>而不是改为<user@domain/resource>).
3. 对于 IQ 节，服务器本身必须(MUST)代替用户应答一个 IQ result 或一个 IQ error，并且不能(MUST NOT)递送这个 IQ 节给任何可用的资源. 具体来说，如果合格的名字空间的语义定义了一个服务器可以提供的应答，服务器必须(MUST)代替用户应答这个节；如果没有，服务器必须(MUST)应答一个<service-unavailable/>节错误.

5. 然后如果 JID 的格式为<user@domain>并且没有这个用户的可用资源，这个节如何处理依赖于节的类型:
    1. 对于类型为"subscribe", "subscribed", "unsubscribe"，和"unsubscribed"的出席信息节，服务器必须(MUST)维持这个节的一个记并且至少递送这个节一次(也就是，当这个用户下次建立一个可用的资源的时候)；另外，服务器必须(MUST)递送类型为"subscribe"的出席信息节直到用户批准或拒绝这个订阅请求为止(参见 出席信息订阅 Presence Subscriptions (第五章第一节第六小节)).
    2. 对于所有其他的出席信息节，服务器应该(SHOULD)安静的忽略这个节,既不存储它用于以后递送也不代替用户应答它.
    3. 对于消息节，服务器可以(MAY)选择代替用户存储这个节并且当用户下次可用的时候递送给他，或通过一些其他的手段转发这个消息给用户(例如，给用户的邮箱). 无论如何，如果离线消息存储或消息转发没有激活，服务器必须(MUST)返回发送者一个<service-unavailable/>节错误. (注意:离线信息存储和消息转发没有定义在 XMPP，因为严格来说它们是实现和服务提供的问题.)
    4. 对于 IQ 节，服务器本身必须(MUST)代替用户应答一个 IQ result 或一个 IQ error. 具体来说,如果合法的名字空间的语义定义了一个服务器可以提供的应答，服务器必须(MUST)代替用户应答这个节；如果没有，服务器必须(MUST)应答一个<service-unavailable/>节错误.

## 出站节

如果出站节的'to'属性的地址的域标识符部分的主机名和服务器自身的一个主机名吻合，服务器必须(MUST) 根据 入站节 Inbound Stanzas(第十一章第一节)的规则递送这个节给一个本地实体.
如果出站节的'to'属性的地址的域标识符部分的主机名不和服务器自身的一个主机名吻合，服务器必须(MUST)尝试路由这个节到外部域. 推荐的动作顺序定义如下:

1. 首先尝试用一个"xmpp-server"服务和"tcp"协议的\[SRV\]服务解析这个外部的主机名,结果得到的资源记录格式如"_xmpp-server._tcp.example.com.", 定义在[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920].
2. 如果"xmpp-server"地址记录解析失败，尝试解析"_im"或"_pres"\[SRV\]服务(定义在\[IMP-SRV\])，使用"_im"服务用语<message/>节,使用"_pres"服务用语

<presence/>节(如何处理<iq/>节取决于具体实现). 这样得到的结果是一个或多个格式为"_im.<proto>.example.com."或 "_pres.<proto>.example.com."的记录, 这里"<proto>"是一个注册在 即时消息 SRV 协议标签注册表 Instant Messaging SRV Protocol Label registry 中的一个标签,或者是 出席信息 SRV 协议标签注册表 Presence SRV Protocol Label registry 中的标签: 要么是"_xmpp", 用于 XMPP-aware 的域,要么是一些 IANA 注册的标签 IANA-registered label (例如,"_simple") 用于 non-XMPP-aware 的域.

3. 如果这两种 SRV 地址记录解析都失败了, 尝试执行一个通用的 IPv4/IPv6 地址记录解析来决定 IP 地址,使用"xmpp-server"端口号 5269(已在 IANA 注册, 定义在[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]).

强烈鼓励部署服务器的管理员们保持 _im._xmpp, _pres._xmpp, 和 _xmpp._tcp SRV 记录正确同步, 因为不同的实现可能在"xmpp-server"查找之前执行"_im"和"_pres"查找.

# 即时消息和出席信息兼容性需求

本章总结了即时消息和出席信息服务器和客户端为了保证兼容的实现而必须(MUST)支持的部分 XMPP 协议. 所有这些应用必须(MUST)遵守定义在[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]的要求. 本章的文字定义了附加的用于即时消息和出席信息的兼容性需求; 注意定义在这里的要求补充而不是替代核心需求. 也要注意一个服务器或客户端可以(MAY)仅支持出席信息或即时消息, 如果只想支持出席信息服务或即时消息服务,可以不必同时支持两个.

## 服务器

除了核心服务器兼容需求之外, 一个即时消息和出席信息服务器必须(MUST)还要支持以下协议:

- 定义在本文中的所有服务器相关的即时消息和出席信息语法和语义, 包括代替客户端广播出席信息, 出席信息订阅, 名册存储和处理, 隐私列表, 以及 IM-specific 路由和递送规则

## 客户端

除了核心的客户端兼容性需求之外, 一个即时消息和出席信息客户端还必须(MUST)支持以下协议:

- 生成和处理由 XML 规划定义的 XML 节的 IM-specific 语义, 包括消息和出席信息节以及它们的子元素的的'type'属性

- 所有本文定义的客户端相关的即时消息语法和语义，包括出席信息订阅，名册管理，和隐私列表

- 端到端的对象加密(定义在 XMPP 中的端到端对象加密 End-to-End Object Encryption in the Extensible Messaging and Presence Protocol (XMPP) \[XMPP-E2E\])

一个客户端也必须(MUST)处理编码为"im:" URIs 的地址(定义在\[CPIM\])，并且可以(MAY)移除"im:"scheme 并把地址解析委托给服务器(定义在 出站节 Outbound Stanzas(第十一章第二节).

# 国际化事项

关于国际化的考虑，参考[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]相关章节.

# 安全性事项

XMPP 的核心安全性事项定义在[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]相关章节.
附加的事项仅适用于分散定义在本文许多地方的 XMPP 即时消息和出席信息应用;特别是:

- 当一个服务器处理一个任何类型的入站节,这个节的预定接收者是和服务器的主机名相关的一个用户,服务器必须(MUST)首先强制应用任何隐私列表(第十章),见 处理 XML 节的服务器规则 Server Rules for Handling XML Stanzas(第十一章)).

- 当一个服务器处理一个类型为"probe"的入站出席信息节,这个节的预定接收者是和服务器的主机名相关的一个用户，如果这个发送者是一个由出席信息订阅决定的未被授权接收那个信息的实体,服务器不能(MUST NOT)揭露这个用户的出席信息(见 客户端和服务器出席信息职责 Client and Server Presence Responsibilities (第五章第一节)).

- 当一个服务器处理一个任何类型的出站出席信息节,这个节没有 type 属性或 type 属性值为"unavailable"，为了确保这个出席信息不被广播给那些未被授权知道这个信息的实体，它必须(MUST)服从客户端和服务器出席信息职责 Client and Server Presence Responsibilities (第五章第一节) 定义的规则 .

- 当一个服务器生成一个错误节作为不存在的用户接收到的一个节的应答的时候，使用<service-unavailable/>错误条件有助于防止著名的字典攻击，因为这个错误和条件和其他一些错误条件相同,例如，一个 IQ 子元素的名字空间不被理解，或离线存储或消息转发不被一个域允许.

# IANA 事项

很多相关的 IANA 事项，参考[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920]相关章节.

## 会话数据的 XML 名字空间名

以下为 XMPP 中会话相关的数据定义了一个 URN 子名字空间. (这个名字空间名的格式遵循 IETF XML Registry \[XML-REG\].)
URI: urn:ietf:params:xml:ns:xmpp-session
Specification: RFC 3921
Description: This is the XML namespace name for session-related data in the Extensible Messaging and Presence Protocol (XMPP) as defined by RFC 3921.
Registrant Contact: IETF, XMPP Working Group, <xmppwg@jabber.org>

## 即时消息 SRV 协议标签注册

确定即时消息和出席信息地址[IMP-SRV],为那些能提供遵守"_im"SRV 服务标签的服务定义了一个即时消息 SRV 协议标签注册表. 因为 XMPP 是其中一个协议, IANA 在适当的注册项中注册了"_xmpp"协议标签,如下:
Protocol label: _xmpp
Specification: RFC 3921
Description: Instant messaging protocol label for the Extensible Messaging and Presence Protocol (XMPP) as defined by RFC 3921.
Registrant Contact: IETF, XMPP Working Group, <xmppwg@jabber.org>

## 出席信息 SRV 协议标签注册

确定即时消息和出席信息地址\[IMP-SRV\],为那些能够提供遵守"_pres"SRV 服务标签的服务定义了一个出席信息 SRV 协议标签注册项. 因为 XMPP 是其中一个协议, IANA 在适当的注册项中注册了"_xmpp"协议标签,如下:
Protocol label: _xmpp
Specification: RFC 3921
Description: Presence protocol label for the Extensible Messaging and Presence Protocol (XMPP) as defined by RFC 3921.
Registrant Contact: IETF, XMPP Working Group, <xmppwg@jabber.org>

# 参考

## 标准参考

[CPIM] Peterson, J., "Common Profile for Instant Messaging (CPIM)", RFC 3860, August 2004.
[IMP-REQS] Day, M., Aggarwal, S., Mohr, G., and J. Vincent, "Instant Messaging/Presence Protocol Requirements", RFC 2779, February 2000.
[IMP-SRV] Peterson, J., "Address Resolution for Instant Messaging and Presence", RFC 3861, August 2004.
[SRV] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
[TERMS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., and E. Maler, "Extensible Markup Language (XML) 1.0 (2nd ed)", W3C REC-xml, October 2000, <http://www.w3.org/TR/REC-xml>.
[XML-NAMES] Bray, T., Hollander, D., and A. Layman, "Namespaces in XML", W3C REC-xml-names, January 1999, <http://www.w3.org/TR/REC-xml-names>.
[XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.
[XMPP-E2E] Saint-Andre, P., "End-to-End Object Encryption in the Extensible Messaging and Presence Protocol (XMPP)", RFC 3923, October 2004.

## 信息参考

[IMP-MODEL] Day, M., Rosenberg, J., and H. Sugano, "A Model for Presence and Instant Messaging", RFC 2778, February 2000.
[IRC] Oikarinen, J. and D. Reed, "Internet Relay Chat Protocol", RFC 1459, May 1993.
[JEP-0054] Saint-Andre, P., "vcard-temp", JSF JEP 0054, March 2003.
[JEP-0077] Saint-Andre, P., "In-Band Registration", JSF JEP 0077, August 2004.
[JEP-0078] Saint-Andre, P., "Non-SASL Authentication", JSF JEP 0078, July 2004.
[JSF] Jabber Software Foundation, "Jabber Software Foundation", <http://www.jabber.org/>.
[VCARD] Dawson, F. and T. Howes, "vCard MIME Directory Profile", RFC 2426, September 1998.
[XML-REG] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

# 附录 A. vCards

[IMP-REQS]的第三章第一节第三小节和第四章第一节第四小节要求可能为其他用户接收带外的联系人信息(例如,电话号码或电子邮件地址). 在 Jabber 社区中通常使用 RFC 2426 [VCARD]中 vCard 规范的 XML 来表达这类信息,但这超出了 XMPP 的范围(这个协议的文档包含在[JEP-0054], 由[JSF]发行).

译者注: [JSF]已改名为[XSF],[JEP-0054]已改名为[XEP-0054]

# 附录 B. XML 规划

接下来的 XML 规划是描述性的，不是标准化的. 规划定义在 XMPP 的核心特性，参考 [XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920].

## B.1 jabber:client

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='jabber:client'
    xmlns='jabber:client'
    elementFormDefault='qualified'>

  <xs:import namespace='urn:ietf:params:xml:ns:xmpp-stanzas'/>

  <xs:element name='message'>
    <xs:complexType>
      <xs:sequence>
        <xs:choice minOccurs='0' maxOccurs='unbounded'>
          <xs:element ref='subject'/>
          <xs:element ref='body'/>
          <xs:element ref='thread'/>
        </xs:choice>
        <xs:any       namespace='##other'
                      minOccurs='0'
                      maxOccurs='unbounded'/>
        <xs:element ref='error'
                      minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='from'
                      type='xs:string'
                      use='optional'/>
      <xs:attribute name='id'
                      type='xs:NMTOKEN'
                      use='optional'/>
      <xs:attribute name='to'
                      type='xs:string'
                      use='optional'/>
      <xs:attribute name='type' use='optional' default='normal'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
```

```
                    <xs:enumeration value='chat'/>
                    <xs:enumeration value='error'/>
                    <xs:enumeration value='groupchat'/>
                    <xs:enumeration value='headline'/>
                    <xs:enumeration value='normal'/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
              <xs:attribute ref='xml:lang' use='optional'/>
            </xs:complexType>
          </xs:element>

          <xs:element name='body'>
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base='xs:string'>
                  <xs:attribute ref='xml:lang' use='optional'/>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>

          <xs:element name='subject'>
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base='xs:string'>
                  <xs:attribute ref='xml:lang' use='optional'/>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>

          <xs:element name='thread' type='xs:NMTOKEN'/>

          <xs:element name='presence'>
            <xs:complexType>
              <xs:sequence>
                <xs:choice minOccurs='0' maxOccurs='unbounded'>
                  <xs:element ref='show'/>
                  <xs:element ref='status'/>
                  <xs:element ref='priority'/>
                </xs:choice>
                <xs:any         namespace='##other'
                                minOccurs='0'
```

```
                        maxOccurs='unbounded'/>
      <xs:element ref='error'
                        minOccurs='0'/>
    </xs:sequence>
    <xs:attribute name='from'
                        type='xs:string'
                        use='optional'/>
    <xs:attribute name='id'
                        type='xs:NMTOKEN'
                        use='optional'/>
    <xs:attribute name='to'
                        type='xs:string'
                        use='optional'/>
    <xs:attribute name='type' use='optional'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='error'/>
          <xs:enumeration value='probe'/>
          <xs:enumeration value='subscribe'/>
          <xs:enumeration value='subscribed'/>
          <xs:enumeration value='unavailable'/>
          <xs:enumeration value='unsubscribe'/>
          <xs:enumeration value='unsubscribed'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute ref='xml:lang' use='optional'/>
  </xs:complexType>
</xs:element>

<xs:element name='show'>
  <xs:simpleType>
    <xs:restriction base='xs:NCName'>
      <xs:enumeration value='away'/>
      <xs:enumeration value='chat'/>
      <xs:enumeration value='dnd'/>
      <xs:enumeration value='xa'/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name='status'>
  <xs:complexType>
    <xs:simpleContent>
```

```
        <xs:extension base='xs:string'>
          <xs:attribute ref='xml:lang' use='optional'/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name='priority' type='xs:byte'/>

  <xs:element name='iq'>
    <xs:complexType>
      <xs:sequence>
        <xs:any         namespace='##other'
                        minOccurs='0'/>
        <xs:element ref='error'
                        minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='from'
                        type='xs:string'
                        use='optional'/>
      <xs:attribute name='id'
                        type='xs:NMTOKEN'
                        use='required'/>
      <xs:attribute name='to'
                        type='xs:string'
                        use='optional'/>
      <xs:attribute name='type' use='required'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='error'/>
            <xs:enumeration value='get'/>
            <xs:enumeration value='result'/>
            <xs:enumeration value='set'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute ref='xml:lang' use='optional'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='error'>
    <xs:complexType>
      <xs:sequence    xmlns:err='urn:ietf:params:xml:ns:xmpp-stanzas'>
        <xs:group     ref='err:stanzaErrorGroup'/>
```

```
      <xs:element ref='err:text'
                  minOccurs='0'/>
    </xs:sequence>
    <xs:attribute name='code' type='xs:byte' use='optional'/>
    <xs:attribute name='type' use='required'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='auth'/>
          <xs:enumeration value='cancel'/>
          <xs:enumeration value='continue'/>
          <xs:enumeration value='modify'/>
          <xs:enumeration value='wait'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

</xs:schema>
```

## B.2 jabber:server

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='jabber:server'
    xmlns='jabber:server'
    elementFormDefault='qualified'>

  <xs:import namespace='urn:ietf:params:xml:ns:xmpp-stanzas'/>

  <xs:element name='message'>
    <xs:complexType>
      <xs:sequence>
        <xs:choice minOccurs='0' maxOccurs='unbounded'>
          <xs:element ref='subject'/>
          <xs:element ref='body'/>
          <xs:element ref='thread'/>
        </xs:choice>
        <xs:any       namespace='##other'
                      minOccurs='0'
                      maxOccurs='unbounded'/>
        <xs:element ref='error'
```

```
                        minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='from'
                        type='xs:string'
                        use='required'/>
      <xs:attribute name='id'
                        type='xs:NMTOKEN'
                        use='optional'/>
      <xs:attribute name='to'
                        type='xs:string'
                        use='required'/>
      <xs:attribute name='type' use='optional' default='normal'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='chat'/>
            <xs:enumeration value='error'/>
            <xs:enumeration value='groupchat'/>
            <xs:enumeration value='headline'/>
            <xs:enumeration value='normal'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute ref='xml:lang' use='optional'/>
    </xs:complexType>
</xs:element>

<xs:element name='body'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute ref='xml:lang' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='subject'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute ref='xml:lang' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
```

```
</xs:element>

<xs:element name='thread' type='xs:NMTOKEN'/>

<xs:element name='presence'>
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs='0' maxOccurs='unbounded'>
        <xs:element ref='show'/>
        <xs:element ref='status'/>
        <xs:element ref='priority'/>
      </xs:choice>
      <xs:any         namespace='##other'
                      minOccurs='0'
                      maxOccurs='unbounded'/>
      <xs:element ref='error'
                      minOccurs='0'/>
    </xs:sequence>
    <xs:attribute name='from'
                      type='xs:string'
                      use='required'/>
    <xs:attribute name='id'
                      type='xs:NMTOKEN'
                      use='optional'/>
    <xs:attribute name='to'
                      type='xs:string'
                      use='required'/>
    <xs:attribute name='type' use='optional'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='error'/>
          <xs:enumeration value='probe'/>
          <xs:enumeration value='subscribe'/>
          <xs:enumeration value='subscribed'/>
          <xs:enumeration value='unavailable'/>
          <xs:enumeration value='unsubscribe'/>
          <xs:enumeration value='unsubscribed'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute ref='xml:lang' use='optional'/>
  </xs:complexType>
</xs:element>
```

```
<xs:element name='show'>
  <xs:simpleType>
    <xs:restriction base='xs:NCName'>
      <xs:enumeration value='away'/>
      <xs:enumeration value='chat'/>
      <xs:enumeration value='dnd'/>
      <xs:enumeration value='xa'/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name='status'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute ref='xml:lang' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='priority' type='xs:byte'/>

<xs:element name='iq'>
  <xs:complexType>
    <xs:sequence>
      <xs:any         namespace='##other'
                      minOccurs='0'/>
      <xs:element ref='error'
                      minOccurs='0'/>
    </xs:sequence>
    <xs:attribute name='from'
                      type='xs:string'
                      use='required'/>
    <xs:attribute name='id'
                      type='xs:NMTOKEN'
                      use='required'/>
    <xs:attribute name='to'
                      type='xs:string'
                      use='required'/>
    <xs:attribute name='type' use='required'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='error'/>
```

```
                <xs:enumeration value='get'/>
                <xs:enumeration value='result'/>
                <xs:enumeration value='set'/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute ref='xml:lang' use='optional'/>
        </xs:complexType>
      </xs:element>

      <xs:element name='error'>
        <xs:complexType>
          <xs:sequence    xmlns:err='urn:ietf:params:xml:ns:xmpp-stanzas'>
            <xs:group     ref='err:stanzaErrorGroup'/>
            <xs:element ref='err:text'
                        minOccurs='0'/>
          </xs:sequence>
          <xs:attribute name='code' type='xs:byte' use='optional'/>
          <xs:attribute name='type' use='required'>
            <xs:simpleType>
              <xs:restriction base='xs:NCName'>
                <xs:enumeration value='auth'/>
                <xs:enumeration value='cancel'/>
                <xs:enumeration value='continue'/>
                <xs:enumeration value='modify'/>
                <xs:enumeration value='wait'/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:complexType>
      </xs:element>

    </xs:schema>
```

## B.3 session

```
    <?xml version='1.0' encoding='UTF-8'?>

    <xs:schema
        xmlns:xs='http://www.w3.org/2001/XMLSchema'
        targetNamespace='urn:ietf:params:xml:ns:xmpp-session'
        xmlns='urn:ietf:params:xml:ns:xmpp-session'
        elementFormDefault='qualified'>
```

```
    <xs:element name='session' type='empty'/>

    <xs:simpleType name='empty'>
      <xs:restriction base='xs:string'>
        <xs:enumeration value=''/>
      </xs:restriction>
    </xs:simpleType>

  </xs:schema>
```

## B.4 jabber:iq:privacy

```
  <?xml version='1.0' encoding='UTF-8'?>

  <xs:schema
      xmlns:xs='http://www.w3.org/2001/XMLSchema'
      targetNamespace='jabber:iq:privacy'
      xmlns='jabber:iq:privacy'
      elementFormDefault='qualified'>

    <xs:element name='query'>
      <xs:complexType>
        <xs:sequence>
          <xs:element ref='active'
                      minOccurs='0'/>
          <xs:element ref='default'
                      minOccurs='0'/>
          <xs:element ref='list'
                      minOccurs='0'
                      maxOccurs='unbounded'/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name='active'>
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base='xs:NMTOKEN'>
            <xs:attribute name='name'
                          type='xs:string'
                          use='optional'/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
```

```
</xs:element>

<xs:element name='default'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:NMTOKEN'>
        <xs:attribute name='name'
                      type='xs:string'
                      use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='list'>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='item'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
    </xs:sequence>
    <xs:attribute name='name'
                  type='xs:string'
                  use='required'/>
  </xs:complexType>
</xs:element>

<xs:element name='item'>
  <xs:complexType>
    <xs:sequence>
      <xs:element name='iq'
                  minOccurs='0'
                  type='empty'/>
      <xs:element name='message'
                  minOccurs='0'
                  type='empty'/>
      <xs:element name='presence-in'
                  minOccurs='0'
                  type='empty'/>
      <xs:element name='presence-out'
                  minOccurs='0'
                  type='empty'/>
    </xs:sequence>
    <xs:attribute name='action' use='required'>
```

```
            <xs:simpleType>
              <xs:restriction base='xs:NCName'>
                <xs:enumeration value='allow'/>
                <xs:enumeration value='deny'/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name='order'
                        type='xs:unsignedInt'
                        use='required'/>
          <xs:attribute name='type' use='optional'>
            <xs:simpleType>
              <xs:restriction base='xs:NCName'>
                <xs:enumeration value='group'/>
                <xs:enumeration value='jid'/>
                <xs:enumeration value='subscription'/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name='value'
                        type='xs:string'
                        use='optional'/>
        </xs:complexType>
      </xs:element>

      <xs:simpleType name='empty'>
        <xs:restriction base='xs:string'>
          <xs:enumeration value=''/>
        </xs:restriction>
      </xs:simpleType>

    </xs:schema>
```

## B.5 jabber:iq:roster

```
    <?xml version='1.0' encoding='UTF-8'?>

    <xs:schema
        xmlns:xs='http://www.w3.org/2001/XMLSchema'
        targetNamespace='jabber:iq:roster'
        xmlns='jabber:iq:roster'
        elementFormDefault='qualified'>

      <xs:element name='query'>
```

```
      <xs:complexType>
        <xs:sequence>
          <xs:element ref='item'
                      minOccurs='0'
                      maxOccurs='unbounded'/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name='item'>
      <xs:complexType>
        <xs:sequence>
          <xs:element ref='group'
                      minOccurs='0'
                      maxOccurs='unbounded'/>
        </xs:sequence>
        <xs:attribute name='ask' use='optional'>
          <xs:simpleType>
            <xs:restriction base='xs:NCName'>
              <xs:enumeration value='subscribe'/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name='jid' type='xs:string' use='required'/>
        <xs:attribute name='name' type='xs:string' use='optional'/>
        <xs:attribute name='subscription' use='optional'>
          <xs:simpleType>
            <xs:restriction base='xs:NCName'>
              <xs:enumeration value='both'/>
              <xs:enumeration value='from'/>
              <xs:enumeration value='none'/>
              <xs:enumeration value='remove'/>
              <xs:enumeration value='to'/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>

    <xs:element name='group' type='xs:string'/>

  </xs:schema>
```

# 附录 C. Jabber IM Presence 协议和 XMPP 之间的不同

本章是非标准的.
译者注：附录 D 对于新接触 XMPP 的人没有什么意义，就不翻译了，免得浪费时间。因为现在 RFC 公布已经很久了，以前的 Jabber 实现很多都进化到 XMPP 了。
XMPP has been adapted from the protocols originally developed in the Jabber open-source community, which can be thought of as "XMPP 0.9". Because there exists a large installed base of Jabber implementations and deployments, it may be helpful to specify the key differences between the relevant Jabber protocols and XMPP in order to expedite and encourage upgrades of those implementations and deployments to XMPP. This section summarizes the differences that relate specifically to instant messaging and presence applications, while the corresponding section of [XMPP-CORE|XMPP 文档列表/XMPP 正式 RFC 标准/RFC3920] summarizes the differences that relate to all XMPP applications.

## C.1 Session Establishment

The client-to-server authentication protocol developed in the Jabber community assumed that every client is an IM client and therefore initiated an IM session upon successful authentication and resource binding, which are performed simultaneously (documentation of this protocol is contained in \[JEP-0078\], published by the Jabber Software Foundation \[JSF\]). XMPP maintains a stricter separation between core functionality and IM functionality; therefore, an IM session is not created until the client specifically requests one using the protocol defined under Session Establishment (Section 3).

## C.2 Privacy Lists

The Jabber community began to define a protocol for communications blocking (privacy lists) in late 2001, but that effort was deprecated once the XMPP Working Group was formed. Therefore the protocol defined under Blocking Communication (Section 10) is the only such protocol defined for use in the Jabber community.

# 贡献者

XMPP 的大部分核心方面是由 1999 年开始的 Jabber 开源社区开发的. 这个社区是由 Jeremie Miller 建立的, 他于 1999 年 1 月发布了最初版的 jabber server 源代码. 主要的基础协议的早期贡献者还包括 Ryan Eatmon, Peter Millard, Thomas Muldowney,和 Dave Smith. XMPP 工作组在即时消息和出席信息方面的工作主要集中在 IM 会话建立和通信屏蔽(隐私列表); 会话建立协议主要是由 Rob Norris 和 Joe Hildebrand 开发的, 隐私列表协议最初是由 Peter Millard.贡献的

# 致谢

感谢许多在贡献者名单之外的人们. 尽管很难提供一个完整的名单, 以下个人对于定义协议或评论标准提供了很多帮助:
Thomas Charron, Richard Dobson, Schuyler Heath, Jonathan Hogg, Craig Kaes, Jacek Konieczny, Lisa Dusseault, Alexey Melnikov, Keith Minkler, Julian Missig, Pete Resnick, Marshall Rose, Jean-Louis Seguineau, Alexey Shchepin, Iain Shigeoka, and David Waite.
也感谢 XMPP 工作组的成员和 IETF 社区在本文的成文过程中一直提供的评论和反馈。

# 作者地址

Peter Saint-Andre (编辑)
Jabber Software Foundation
EMail: stpeter@jabber.org

# 完整的版权声明



# 知识产权

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

# 感谢

目前为 RFC 编辑活动提供资金的 Internet Society.