Conceptos claves

Tipos de protocolos jabber/xmpp

A continuacion se describen brebemente los diferentes protocolos que existen en XMPP

* Message protocol

Etiqueta <message >

Tipos de mensajes:

* **normal**: que serían mensajes parecidos a los del correo electrónico.
* **chat**: mensajes persona a persona que serían los mensajes

utilizados en una conversación entre dos personas.

* **groupchat**: mensajes enviados a un grupo de personas
* **headline**: que serían los mensajes de marquesina.
* **error**: para los mensajes de error.

Estos son los errores más comunes del núcleo de Jabber:

* + 400 – Petición errónea.
  + 401 – Desautorizado.
  + 402 – Servicio de pago.
  + 403 – Prohibido.
  + 404 – No encontrado.
  + 405 – No permitido.
  + 406 – No aceptable.
  + 407 – Registro requerido.
  + 408 – Timeout.
  + 409 – Conflicto.
  + 500 – Error interno del servidor.
  + 501 – No implementado.
  + 502 – Error del servidor remoto.
  + 503 – Servicio no disponible.
  + 504 – Timeout de servidor remoto.
* **jabber**:x:oob: para las conexiones directas entre clientes para envío

de archivos. El usuario que va a servir un fichero enviaría un mensaje de este

tipo al otro cliente con la IP y el puerto al que se debe conectar el cliente que va

a descargarse el fichero.Se pueden enviar etiquetas de oob dentro de la

extensión x de un mensaje normal, o empaquetadas dentro de un paquete del

tipo Info/Query.

Ejemplo

Cliente **A** envia al **server**:

<message from="oscar@jabber.es/trabajo" to="iñaki@jabber.es/casa" id="messageid1">

<thread>threadid\_01</thread>

<subject>Título del mensaje</subject>

<body>Cuerpo del mensaje</body>

</message>

Cliente **B** recibe del **Server**:

<message from="oscar@jabber.es/trabajo" to="iñaki@jabber.es">

<body>Hola!</body>

</message>

* Presence protocol

Usado principalmente en dos contextos:

1) **Presence update:** actualización de la presencia debido a un cambio de estado del usuario.

2) **Presence subscription management:** permite a los usuarios subscribirse a las actualizaciones de presencia de otros usuarios y

controlar quien está accediendo a su propia presencia.

En ambos casos el servidor actúa como árbitro entre elemisor de la actualización de presencia y los destinatarios de la misma.

El servidor tiene la obligación de hacer llegar el paquete de actualización de presencia a todos los contactos del cliente que lo generó, pero sólo a los contactos que el cliente emisor ha confirmado que le gustaría que recibieran su

presencia.

El servicio usa un simple mensaje unidireccional del emisor al servidor de su dominio, es el servidor el que tendrá

que copiar y reenviar el mensaje de presencia a todos los clientes del emisor.

El servidor mira la lista de contactos del emisor para conocer quienes son sus contactos, esta lista de contactos en Jabber/XMPP se llama Roster (siempre esta en el server esta lista).

Tipos de presencia:

|  |  |
| --- | --- |
| Tipos | Desc |
| **available(1)** | Indica que el usuario está listo para recibir mensajes. |
| **unavailable(1)** | Usuario no está disponible para recibir mensajes. |
| **subscribe(2)** | El usuario que lo envía desea suscribirse a la presencia del destinatario. |
| **unsubscribe(2)** | cancelación de suscripción de presencia, el usuario que lo envía desea cancelar su suscripción a la presencia del destinatario. |
| **subscribed(2)** | respuesta que recibirá un usuario que ha realizado una petición de suscripción, que indica el estado actual de la suscripción. |
| **unsubscribed(2)** | respuesta que recibirá un usuario que ha realizado una petición de suscripción y le ha sido negada, o una petición de cancelación de la suscripción. |
| **error(2)** | mensaje estándar de error de Jabber/XMPP que indica problemas en la presencia. |
| **probe(2)** | petición Servidor-a-Servidor que envía toda la información de presencia de un servidor a otro. |
|  |  |

El protocolo de presencia permite además cuatro sub-etiquetas más en el paquete de presencia:

|  |  |
| --- | --- |
| <status> | texto libre para que el usuario explique su estado actual. |
|  |  |
| <priority> | prioridad numérica del mensaje, los números más altos tienen más  prioridad. Sólo se admiten números enteros positivos. |
|  |  |
| <error> | paquete estándar de error de Jabber/XMPP. |
| <show> | Uno de los cuatro estados estándar que los clientes pueden usar para modificar su presencia “available”. Los clientes Jabber/XMPP usarán normalmente el estado show para mostrar iconos de presencia estándar, alertas sonoras y otras cosas. Si el estado show no está indicado, el usuario se encuentra en estado normal o online. Los estados estándar para show son:   * chat: el usuario está intentando hablar con alguien. * away: el usuario está fuera del cliente Jabber/XMPP por un corto periodo de tiempo. * xa (extended away): el usuario está fuera por un periodo prolongado de tiempo. * dnd (do not distrub): el usuario no desea recibir mensajes. |

* Groupchat protocol

Existen cuatro acciones fundamentales:

* **Unirse al grupo:** enviando una presencia de tipo “available” al grupo.
* **Enviar mensajes a todo el grupo**: enviando un mensaje de tipo groupchat al Jabber ID del grupo deseado, al que previamente el usuario se ha debido de unir.
* **Enviar mensajes a un único miembro del grupo** (mensajes privados): enviando un mensaje de tipo groupchat a una persona del grupo,poniendo tras el Jabber ID del grupo el carácter ‘/’ y el nickname de la persona.
* **Abandonar el grupo**: enviando un mensaje “unavailable” de presencia al grupo.

*Como se puede ver, para unirse y abandonar un grupo se usa el protocolo de presencia, y para enviar mensajes, ya sean a todo el grupo, o mensajes privados se usa el protocolo de mensajes, pero con el tipo “groupchat”.*

* Info/Query protocol

Son mensajes de petición respuesta, así que cuando un cliente envía un mensaje IQ siempre deberá recibir una respuesta del destinatario, aunque sea te tipo error.

* Un usuario en Jabber/XMPP representa a una persona.
* Los clientes Jabber/XMPP actúan en representación de un determinado usuario, son llamados “resources”, y se indica al final del Jabber ID del usuario.
* Un usuario puede tener varios clientes activos simultáneamente.
* Protocolo IQ

IQ es un protocolo sencillo y ampliable de petición-respuesta que permite a los usuarios hacer peticiones y almacenar o cambiar datos usando un sencillo protocolo.

IQ es simplemente el conductor de esas peticiones y respuestas, y gestiona qué datos son necesarios de acuerdo con las necesidades particulares de cada servidor.

La mayoría de las peticiones IQ son entre un cliente y un servidor.

Tipo de petición IQ:

|  |  |
| --- | --- |
| set | Ppara modificar datos que posee el destinatario del paquete, |
| get | Para solicitarlos |
| result | Respuesta de una solicitud IQ anterior |
| error | Respuesta de error en el procesamiento de una petición IQ anterior |

*Los dos primeros tipos serían los que utilizaría el cliente que emita la petición IQ y los dos últimos serían las dos posibles respuestas de quien reciba la petición.*

El formato de un paquete IQ sería el siguiente:

<iq type="set|get|result|error" to="jid\_destino" from="jid\_origen" id="unica">

<query xmlns="iq extension namespace">

<query\_field1/>

<query\_field2/>

</query>

</iq>

* Se pueden dar varias peticiones/respuestas <query> dentro de un mismo mensaje iq. Habitual en la respuesta, ya que si por ejemplo un cliente solicita a su servidor la lista de contactos, el cliente enviará una única petición, pero el servidor responderá con un <query> por cada contacto que ese cliente tenga almacenado en el servidor.
* Los paquetes <query> que van dentro de los paquetes <iq> establecen un namespace XML usando el atributo xmlns. Cada namespace establece una extensión IQ con sus propias etiquetas y atributos, y un protocolo para su procesamiento.
* Las extensiones de IQ se organizan según las necesidades de cada aplicación. cualquier desarrollador puede implementar sus propias extensiones según sus necesidades.
* Extensiones IQ más importantes.
* Registration protocol (jabber:iq:register)

Para dejar a los usuarios crear sus cuentas en el servidor Esto no ocurrirá así en un servidor de uso privado.

Este protocolo es usado especialmente en los servidores públicos de Jabber/XMPP.

* Authentication protocol (jabber:iq:auth)

Permite a los usuarios demostrar a su servidor que son quienes dicen ser, los clientes no autentificados tienen una serie de permisos, y los clientes autentificados tienen un completo acceso al uso de todos los protocolos de Jabber/XMPP que estén implementados en el servidor del dominio al que pertenecen.

* Roster Protocol (jabber:iq:roster)

Es una extensión del protocolo IQ. Sirven para organizar y administrar las suscripciones de cada usuario. La suscripción de presencia determina los suscriptores que recibirán las actualizaciones de presencia de cada usuario.

Los suscriptores pedirán una suscripción a un usuario, y el usuario aceptará o denegará dicha suscripción. Cada usuario se suscribirá a los usuarios que desee y podrá aceptar que dichos usuarios u otros usuarios se suscriban a los cambios de su presencia.

**Tipos:**

* **Roster get:** usado por los clientes para obtener una copia del roster

almacenado en el servidor.

* **Roster update:** usado por los clientes para actualizar el roster

almacenado en el servidor.

* **Roster push:** actualizaciones asíncronas del roster que el servidor

envía a los clientes.

1. **Ejemplo Roster get**

Permite a los clientes obtener una copia completa del roster almacenado en su servidor Jabber/XMPP. Para ello se debe enviar un mensaje IQ de tipo get vacío especificando el protocolo “jabber:iq:roster”.

<iq type="get" id="roster\_get\_id">

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

</iq>

Y el servidor contestará con una copia completa del roster.

<iq type="result" id="roster\_get\_id">

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

<item jid="sub1ID" name="nickname1" subscription="both">

<group>Personal</group>

<group>Trabajo</group>

</item>

<item jid="sub2ID" name="nickname2" subscription="from">

<group>Trabajo</group>

<group>Familia</group>

<group>Equipo de Futbol</group>

</item>

</query>

</iq>

*Cada paquete <item> representa una suscripción de un contacto, y ese contacto pertenecerá a cero o más grupos, identificados por cero o más paquetes <group>.*

1. **Ejemplo Roster update**

Los clientes pueden modificar el roster enviando un paquete IQ de tipo set

conteniendo dentro del paquete <query> el item a actualizar.

<iq type="set" id="roster\_set\_id">

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

<item jid="sub1ID" name="Pepe">

<group>Trabajo</group>

<group>Familia</group>

<group>Equipo de Baloncesto</group>

</item>

</query>

</iq>

El servidor responderá con un paquete IQ de tipo respuesta indicando que

la actualización se ha realizado correctamente o un paquete IQ de error en

caso contrario

1. **Roster push**

Son los únicos mensajes IQ que no obtienen respuesta, sólo van del servidor a

los clientes y éstos no tienen que responder.

<iq type="set">

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

<item jid="sub1ID" name="Pepe" subscription="both">

<group>Trabajo</group>

<group>Familia</group>

<group>Equipo de Baloncesto</group>

</item>

</query>

</iq>

**Tipos de relaciones** de suscripción tratados por el roster

|  |  |
| --- | --- |
| to | el usuario está interesado en recibir las actualizaciones de presencia del suscriptor. |
| from | el suscriptor está interesado en recibir las actualizaciones de presencia del usuario. |
| both | el usuario y el suscriptor tienen un mutuo interés en recibir las presencias entre ellos. |
| none | ni el usuario ni el suscriptor tienen interés por recibir presencias entre ellos. |

Como cada usuario tendrá su propio roster, si enviamos una actualización de presencia al servidor, éste buscará en nuestro roster todos los contactos que tengan los tipos de suscripción “both” o “form” para reenviarles sólo a ellos nuestra presencia.

Como todo se almacena en el servidor, el cliente comenzará autentificarse pidiendo todo el roster al servidor con un **roster reset.** Mostrará entonces todos los contactos del roster en la aplicación. Si más adelante el usuario desea hacer cambios en algún contacto del roster enviará una roster update al servidor y se quedará esperando un roster push del servidor para confirmar la actualización a todos los clientes abiertos con el mismo Jabber ID,

Los cambios del roster pueden suceder en cualquier momento desde que un usuario se autentifica, por lo tanto los clientes deben estar preparados para recibir del servidor roster push y actualizar los contactos que muestran en ese momento.

El servidor enviará un roster push cuando:

* Una actualización del roster cambie los atributos del mismo
* Cuando por algún cambio en el tipo de suscripción de presencia se deba crear o borrar un contacto.

El protocolo de suscripción de presencia tiene efectos sobre el roster.

Ej: una actualización del tipo de suscripción puede hacer desaparecer de la pantalla del usuario un contacto, ya que si el contacto no desea ser visto por el usuario y envía una cancelación de la suscripción de presencia poniéndola a “none” : Ejemplo Messenger cuando bloquea o hace q alguien no lo pueda ver

Otras extensiones estándares del protocolo IQ

* XEP-0012 Last Activity: jabber:iq:last

Sirve para conocer la ultima actividad asociada con una entidad. Determinar la ultima vez que un cliente accedio a un server.

Ejemplo:

* Last Activity **Query:** El solicitante envia la siguiente stanza tipo get con namespase jabber:iq:last

<iq from="romeo@montague.net/orchard" id="last1" to="juliet@capulet.com" type="get">

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

</iq>

* Last Activity **Response** La entidad destino DEBE retornar un Iq tipo return o error como se muestra en la siguiente su stanza:

<iq from="juliet@capulet.com" id="last1" to="romeo@montague.net/orchard" type="result">

<query xmlns="jabber:iq:last" **seconds**="903"/>

</iq>

* XEP-0030: Service Discovery

# <http://xmpp.org/extensions/xep-0030.html>

## Discovering Information About a Jabber Entity disco#info

Descubre información sobre las entidades sobre la red jabber. Tal info incluye características o protocolos soportados por las entidades.-

**Potocolo básico:**

La entiad solicoitante (request ) quiere descubrir algo acerca de otra entidad. La inf. deseada en general es **Identificación de la entidad solicitada**: Categorías pueden ser *server, client, gateway, directory, etc.* y su tipo particular dentro de esa categoría: *M server, phone vs. handheld client, MSN gateway vs. AIM gateway, user directory vs. chatroom directory, etc*.

Esta info le ayuda a la entidad q solicitante determinar el grupo en el que le es más propicio incluirse.

Una entidad PUEDE tener varias identidades.

Para descubrir toda esta info, la entidad debe enviar una stanza del tipo: **get** conteniendo una <query/> vacía y namespase http://jabber.org/protocol/disco#info

Ejemplo:

* **Envió de consulta de información (get)**

<iq type="get" from="romeo@montague.net/orchard" to="plays.shakespeare.lit" id="info1">

<query xmlns="http://jabber.org/protocol/disco#info"/>

</iq>

* **Recepcion de petición (Result-set for information request)**

<iq type="result" from="plays.shakespeare.lit" to="romeo@montague.net/orchard" id="info1">

<query xmlns="http://jabber.org/protocol/disco#info">

<identity category="conference" type="text" name="Play-Specific Chatrooms"/>

<identity category="directory" type="chatroom" name="Play-Specific Chatrooms"/>

<feature var="http://jabber.org/protocol/disco#info"/>

<feature var="http://jabber.org/protocol/disco#items"/>

<feature var="http://jabber.org/protocol/muc"/>

<feature var="jabber:iq:register"/>

<feature var="jabber:iq:search"/>

<feature var="jabber:iq:time"/>

<feature var="jabber:iq:version"/>

</query>

</iq>

Siempre se debe retornar si no es error una etiqueta query <query xmlns="http://jabber.org/protocol/disco#info">

feature var: debe tener el atributo var que espesifica el protocol u otra característica q ofrezca la entidad.-

Ejemplos:

* **Query a una sala de conferencia (Conference room)**

<iq type="get" from="juliet@capulet.com/balcony" to="**balconyscene**@plays.shakespeare.lit" id="info3">

<query xmlns="http://jabber.org/protocol/disco#info"/>

</iq>

<iq type="result" from="**balconyscene**@plays.shakespeare.lit" to="juliet@capulet.com/balcony" id="info3">

<query xmlns="http://jabber.org/protocol/disco#info">

<identity category="conference" type="text" name="Romeo and Juliet, Act II, Scene II"/>

<feature var="http://jabber.org/protocol/disco#info"/>

<feature var="http://jabber.org/protocol/muc"/>

<feature var="http://jabber.org/protocol/feature-neg"/>

<feature var="muc-password"/>

<feature var="muc-hidden"/>

<feature var="muc-temporary"/>

<feature var="muc-open"/>

<feature var="muc-unmoderated"/>

<feature var="muc-nonanonymous"/>

</query>

</iq>

* **Querying a connected resource for further information Query a un recurso conectado por information suplementaria**

<iq type="get" from="juliet@capulet.com/balcony" to="romeo@montague.net/orchard" id="info4">

<query xmlns="http://jabber.org/protocol/disco#info"/>

</iq>

<iq type="result" from="romeo@montague.net/orchard" to="juliet@capulet.com/balcony" id="info4">

<query xmlns="http://jabber.org/protocol/disco#info">

<identity category="client" type="pc" name="Gabber"/>

<feature var="jabber:iq:time"/>

<feature var="jabber:iq:version"/>

</query>

</iq>

## Info Nodes

Una query disco#info también puede ser dirigida a un nodo especifico asociado a un JID, a pesar de que el principal uso del nodo es Item-Nodes menos que Info-Nodes

Ejemplo:

* **Querying a specific JID and node combination**

<iq type="get" from="romeo@montague.net/orchard" to="mim.shakespeare.lit" id="info3">

<query xmlns="http://jabber.org/protocol/disco#info" node="http://jabber.org/protocol/commands"/>

</iq>

Si el solicitante incluye un atributo **“nodo”** , el que responde debe reflejar un atributo “nodo” también.-

* **JID+node result**

<iq type="result" from="mim.shakespeare.lit" to="romeo@montague.net/orchard" id="info3">

<query xmlns="http://jabber.org/protocol/disco#info" node="http://jabber.org/protocol/commands">

<identity category="automation" type="command-list"/>

</query>

</iq>

## Descubrir los Items asociados a un Jabber Entity: disco#items

**Protocolo básico**

El que solicita envía una stanza tipo get a la entidad destino conteniendo una <query/> vacía y namespase <http://jabber.org/protocol/disco#items>

* **Requesting all ítems**

<iq type="get" from="romeo@montague.net/orchard" to="shakespeare.lit" id="items1">

<query xmlns="http://jabber.org/protocol/disco#items"/>

</iq>

* **Result-set for all ítems**

<iq type="result" from="shakespeare.lit" to="romeo@montague.net/orchard" id="items1">

<query xmlns="http://jabber.org/protocol/disco#items">

<item jid="people.shakespeare.lit" name="Directory of Characters"/>

<item jid="plays.shakespeare.lit" name="Play-Specific Chatrooms"/>

<item jid="mim.shakespeare.lit" name="Gateway to Marlowe IM"/>

<item jid="words.shakespeare.lit" name="Shakespearean Lexicon"/>

<item jid="globe.shakespeare.lit" name="Calendar of Performances"/>

<item jid="headlines.shakespeare.lit" name="Latest Shakespearean News"/>

<item jid="catalog.shakespeare.lit" name="Buy Shakespeare Stuff!"/>

<item jid="en2fr.shakespeare.lit" name="French Translation Service"/>

</query>

</iq>

Si no hay items asociados a la entidad destino. Se retorna:

* **Empty result-set**

<iq type="result" from="shakespeare.lit" to="romeo@montague.net/orchard" id="items1">

<query xmlns="http://jabber.org/protocol/disco#items"/>

</iq>

**Bare JID**

Igual que disco#info, cuando una entidad envía una request disco#items a un bare JID (JID puro o simple) ej: account@domain.tld

hosteado por el server, el server mismo debe responder a favor de la cuenta hosteada.-

Si el acceso a la info no está denegada, el server responde los ítems asociados incluyendo coconected o available recursos.-+

* **Requesting items from a bare JID**

<iq type="get" from="shakespeare.lit" to="juliet@capulet.com" id="items2">

<query xmlns="http://jabber.org/protocol/disco#items"/>

</iq>

* **Server replies on behalf of bare JID**

<iq type="result" from="catalog.shakespeare.lit" to="romeo@montague.net/orchard" id="items2">

<query xmlns="http://jabber.org/protocol/disco#items">

<item jid="catalog.shakespeare.lit" node="books" name="Books by and about Shakespeare"/>

<item jid="catalog.shakespeare.lit" node="clothing" name="Wear your literary taste with pride"/>

<item jid="catalog.shakespeare.lit" node="music" name="Music from the time of Shakespeare"/>

</query>

</iq>

**Further nodes**

Son nodos asociados al primer nivel. Nodos asociados al nodo de la query anterior.- Pueden ser categorías asociadas que tengan ítems asociados

* **Requesting further nodes**

<iq type="get" from="romeo@montague.net/orchard" to="catalog.shakespeare.lit" id="items3">

<query xmlns="http://jabber.org/protocol/disco#items" node="**music**"/>

</iq>

* **Service returns further nodes**

<iq type="result" from="catalog.shakespeare.lit" to="romeo@montague.net/orchard" id="items3">

<query xmlns="http://jabber.org/protocol/disco#items" node="music">

<item jid="catalog.shakespeare.lit" node="music/A"/>

<item jid="catalog.shakespeare.lit" node="music/B"/>

<item jid="catalog.shakespeare.lit" node="music/C"/>

<item jid="catalog.shakespeare.lit" node="music/D"/>

</query>

</iq>

* **Otro ejemplo**

<iq type="get" from="romeo@montague.net/orchard" to="catalog.shakespeare.lit" id="items3">

<query xmlns="http://jabber.org/protocol/disco#items" node="**music/D**"/>

</iq>

* **Resultado seria:**

<iq type="result" from="catalog.shakespeare.lit" to="romeo@montague.net/orchard" id="items4">

<query xmlns="http://jabber.org/protocol/disco#items" node="music/D">

<item jid="catalog.shakespeare.lit" node="music/D/dowland-firstbooke"

name="John Dowland - First Booke of Songes or Ayres"/>

<item jid="catalog.shakespeare.lit" node="music/D/dowland-solace"

name="John Dowland - A Pilgrimes Solace"/>

</query>

</iq>

**XEP-0094: Agent Information**

jabber:**i**q:agents

<http://xmpp.org/extensions/xep-0094.html>

The 'jabber:iq:agents' namespace was used to obtain a list of entities associated with another Jabber Entity; most commonly, the list of trusted services associated with a specific host.

When 'jabber:iq:agents' is used, information about available agents properties is contained within a <query/> element that is scoped by the 'jabber:iq:agents' namespace. The reply to a request of type "get" in the 'jabber:iq:agents' namespace contains zero or more <agent/> elements. The <agent/> element has a required 'jid' attribute that contains the Jabber Identifier of each agent. The <agent/> element in turn may contain the following children:

* name - a natural-language name for the service
* description - a short phrase describing the service
* transport - inclusion of this empty element signals that the service is a gateway to a non-Jabber instant messaging system
* groupchat - inclusion of this empty element signals that the service is multi-user chat service
* service - the CDATA of this element specifies the type of gateway (aim, icq, irc, msn, yahoo), the type of conferencing service (public or private), or user directory (jud); these values were never standardized and are not registered with the XMPP Registrar
* register - inclusion of this empty element signals that the service supports registration
* search - inclusion of this empty element signals that the service supports searching

## 3. Examples

Example 1. Request for Agents List

<iq id='agents1' type='get' to='shakespeare.lit'>  
  <query xmlns='jabber:iq:agents'/>  
</iq>

Example 2. Reply Containing Agents List

<iq  
    to='juliet@capulet.com/balcony'  
    from='shakespeare.lit'  
    type='result'  
    id='agents1'>  
  <query xmlns='jabber:iq:agents'>  
    <agent jid='users.shakespeare.lit'>  
      <name>Jabber User Directory</name>  
      <service>jud</service>  
      <search/>  
      <register/>  
    </agent>  
    <agent jid='chat.shakespeare.lit'>  
      <name>Conferencing Service</name>  
      <service>public</service>  
      <groupchat/>  
    </agent>  
  </query>  
</iq>