



EBook Gratis

APRENDIZAJE

xmpp

Free unaffiliated eBook created from
Stack Overflow contributors.

#xmpp

Tabla de contenido

Acerca de.....	1
Capítulo 1: Empezando con xmpp.....	2
Observaciones.....	2
Versiones.....	2
Examples.....	2
Conectando y enviando un mensaje.....	2
SleekXMPP (Python).....	2
Smack (Java / Android).....	3
Creando una sesión de chat y enviando un mensaje.....	3
Crear conexión de cliente Xmpp utilizando la biblioteca agsxmpp.....	4
Enviar un mensaje utilizando la biblioteca agsxmpp.....	4
Capítulo 2: Arquitectura.....	6
Observaciones.....	6
Direccionabilidad.....	6
Corrientes de estado.....	6
Enrutamiento.....	6
Servidores.....	6
Examples.....	7
Visualizando la red XMPP como un gráfico.....	7
Capítulo 3: Direcciones XMPP alias. JIDs (Identificadores Jabber).....	10
Sintaxis.....	10
Parámetros.....	10
Observaciones.....	10
Examples.....	10
Dividiendo un JID (genérico).....	10
Tipos de JID.....	11
Validando un JID (genérico).....	11
Dividiendo un JID (Ir).....	12
Dividiendo un JID (Rust).....	13
Capítulo 4: Negociación de flujo.....	14

Observaciones.....	14
Examples.....	14
Cerrando un arroyo.....	14
Comenzando una corriente.....	14
Cierre la conexión XMPP usando la biblioteca agsxmpp.....	15
Creditos.....	16

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [xmpp](#)

It is an unofficial and free xmpp ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official xmpp.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con xmpp

Observaciones

El protocolo extensible de mensajería y presencia (XMPP) es un protocolo de red que utiliza XML para intercambiar datos estructurados entre dos o más entidades conectadas a la red casi en tiempo real. XMPP fue creado para satisfacer las pautas de IETF para los protocolos de presencia y mensajería instantánea ([RFC 2779](#)), pero su propósito va mucho más allá de la mensajería instantánea. También se utiliza como middleware orientado a mensajes, para la comunicación máquina a máquina (M2M) y para Internet de las cosas (IoT).

El protocolo básico ligero de XMPP proporciona a los usuarios

- autenticación fuerte
- direcciones globales
- Formato estructurado y extensible para el intercambio de datos.

El enfoque extensible hace posible crear protocolos personalizados sobre el núcleo de XMPP.

El protocolo XMPP central está definido en [RFC 6120](#) y es administrado por el Grupo de trabajo de ingeniería de Internet (XMPP). Las extensiones de mensajería instantánea se definen en [RFC 6121](#) , y un tercer documento ([RFC 7622](#)) define el formato de las direcciones XMPP, también denominadas "Identificadores Jabber" (JID). La funcionalidad adicional se especifica en la forma de [Protocolos de Extensión XMPP](#) (XEP), que son creados por la comunidad y mantenidos por la [Fundación de Estándares XMPP](#) (XSF).

Versiones

Versión	Notas	Fecha de lanzamiento
1.0	Núcleo: RFC 6120 , IM: RFC 6121 , Dirección: RFC 7622	2011-03-01
0.9	Núcleo: RFC 3920 , IM: RFC 3921 , Dirección: RFC 6122	2004-10-01

Examples

Conectando y enviando un mensaje.

SleekXMPP (Python)

```
import sleekxmpp

client = sleekxmpp.Client("address@example.net", "password")
client.connect()
client.process(blocking=False)
client.send_message(mto="remote@example.net", mbody=self.msg)
```

Smack (Java / Android)

```
XMPPTCPConnection connection = new XMPPTCPConnection("user", "password", "example.org")
connection.connect().login();
Message message = new Message("otheruser@example.net", "Hi, how are you?");
connection.sendStanza(message);
connection.disconnect();
```

Creando una sesión de chat y enviando un mensaje.

Smack (Java)

- Utilizando Smack 4.1
- Se recomienda incluir la dependencia de Smack as Maven en su proyecto (por ejemplo, utilizando gradle o Maven).
- Los demás artefactos / tarros de Smack deben agregarse manualmente a la ruta de clase: smack-core, smack-extensions, smack-experimental, smack-im, smnack-tcp, smack-java7

```
import org.jivesoftware.smack.ConnectionConfiguration.SecurityMode;
import org.jivesoftware.smack.SmackException;
import org.jivesoftware.smack.XMPPException;
import org.jivesoftware.smack.chat.Chat;
import org.jivesoftware.smack.chat.ChatManager;
import org.jivesoftware.smack.chat.ChatMessageListener;
import org.jivesoftware.smack.packet.Message;
import org.jivesoftware.smack.packet.Presence;
import org.jivesoftware.smack.tcp.XMPPTCPConnection;
import org.jivesoftware.smack.tcp.XMPPTCPConnectionConfiguration;

public void sendMessage() {

    XMPPTCPConnectionConfiguration config =
        XMPPTCPConnectionConfiguration.builder()
            .setServiceName("mydomain.local")
            .setHost("127.0.0.1")
            .setPort(5222)
            .build();

    XMPPTCPConnection connection = new XMPPTCPConnection(config);

    connection.connect();
    connection.login("test1", "test1pwd");

    ChatManager chatManager = ChatManager.getInstanceFor(connection);
    String test2JID = "test2@domain.example";
    Chat chat = chatManager.createChat(test2JID);
    chat.sendMessage("Hello, how are you?");
```

```
connection.disconnect();
}
```

Crear conexión de cliente Xmpp utilizando la biblioteca agsxmpp

```
public void OpenXmppConnection(int port, bool useSsl, string serverJid, string userName,
string password)
{
    try
    {
        _xmppClientConnection.AutoResolveConnectServer = true;
        _xmppClientConnection.Port = port;
        _xmppClientConnection.UseSSL = useSsl;
        _xmppClientConnection.Server = serverJid;
        _xmppClientConnection.Username = userName;
        _xmppClientConnection.Password = password;
        _xmppClientConnection.Resource = "web";

        //authenticate and open connection with server
        _xmppClientConnection.Open();
    }
    catch (Exception ex)
    {
    }
}
```

Enviar un mensaje utilizando la biblioteca agsxmpp.

```
public class ConversationManager
{
    #region ClassMemeber

    private XmppClientConnection _xmppClientConnection = null;

    public ConversationManager(XmppClientConnection con)
    {
        _xmppClientConnection = con;
    }

    public void SendMessage(string message, string to, string guid, string type)
    {
        try
        {
            if (_xmppClientConnection != null)
            {
                Jid jidTo = new Jid(to);
                agsXMPP.protocol.client.Message mesg = new
                agsXMPP.protocol.client.Message(jidTo, _ConnectionWrapper.MyJid,
                agsXMPP.protocol.client.MessageType.chat,
                message);

                mesg.Id = guid;
                mesg.AddChild(new
```

```
agsXMPP.protocol.extensions.msgreceipts.Request()); //request delievery
        _xmppClientConnection.Send(msg);
    }
}
catch (Exception ex)
{
}
}
}
```

Lea Empezando con xmpp en línea: <https://riptutorial.com/es/xmpp/topic/2451/empezando-con-xmpp>

Capítulo 2: Arquitectura

Observaciones

XMPP permite el intercambio dúplex completo de datos estructurados y el procesamiento simultáneo de solicitudes entre clientes y servidores de la red globalmente accesibles. A diferencia de HTTP y la arquitectura "Representational State Transfer" (REST) ampliamente implementada en la web, las conexiones XMPP son de estado y concurrentes, y puede ocurrir un número ilimitado de transacciones en el contexto de una sola sesión. Esta arquitectura a veces también se denomina "Disponibilidad para transacciones concurrentes" (ACT).

Direccionabilidad

Para facilitar el enrutamiento a través de la red, todas las direcciones XMPP son direccionables globalmente. Al igual que el correo electrónico, esto se logra con DNS y una arquitectura de cliente / servidor federada. Las direcciones son de la forma `localpart@domainpart/resourcepart` donde la parte local es opcional y corresponde a un usuario de la red, el dominio es obligatorio y corresponde a un servidor, y la opción de recurso es opcional y se refiere a un cliente conectado específico para ese usuario (en Los usuarios de XMPP pueden iniciar sesión desde muchas ubicaciones diferentes, por ejemplo, un teléfono y una computadora portátil en el caso de mensajería instantánea, o muchos sensores que usan una cuenta en el caso de dispositivos habilitados para Internet de las cosas). XMPP también proporciona instalaciones para descubrir la presencia (disponibilidad) de otras direcciones en la red.

Corrientes de estado

Las conexiones XMPP son conexiones TCP de larga duración que transportan secuencias XML desde un cliente a un servidor (c2s) o desde un servidor a un servidor (s2s). Tener estas sesiones de larga duración y con estado permite a los nodos de la red transmitir datos en cualquier momento y enviarlos o entregarlos de inmediato.

Enrutamiento

Los flujos forman un enlace directo en la red entre un cliente y un servidor o un servidor y un servidor. Si un cliente desea comunicarse con un cliente remoto en la red, primero envía la información a su servidor que forma una conexión de servidor a servidor con el servidor remoto que luego entrega la información a su cliente.

Servidores

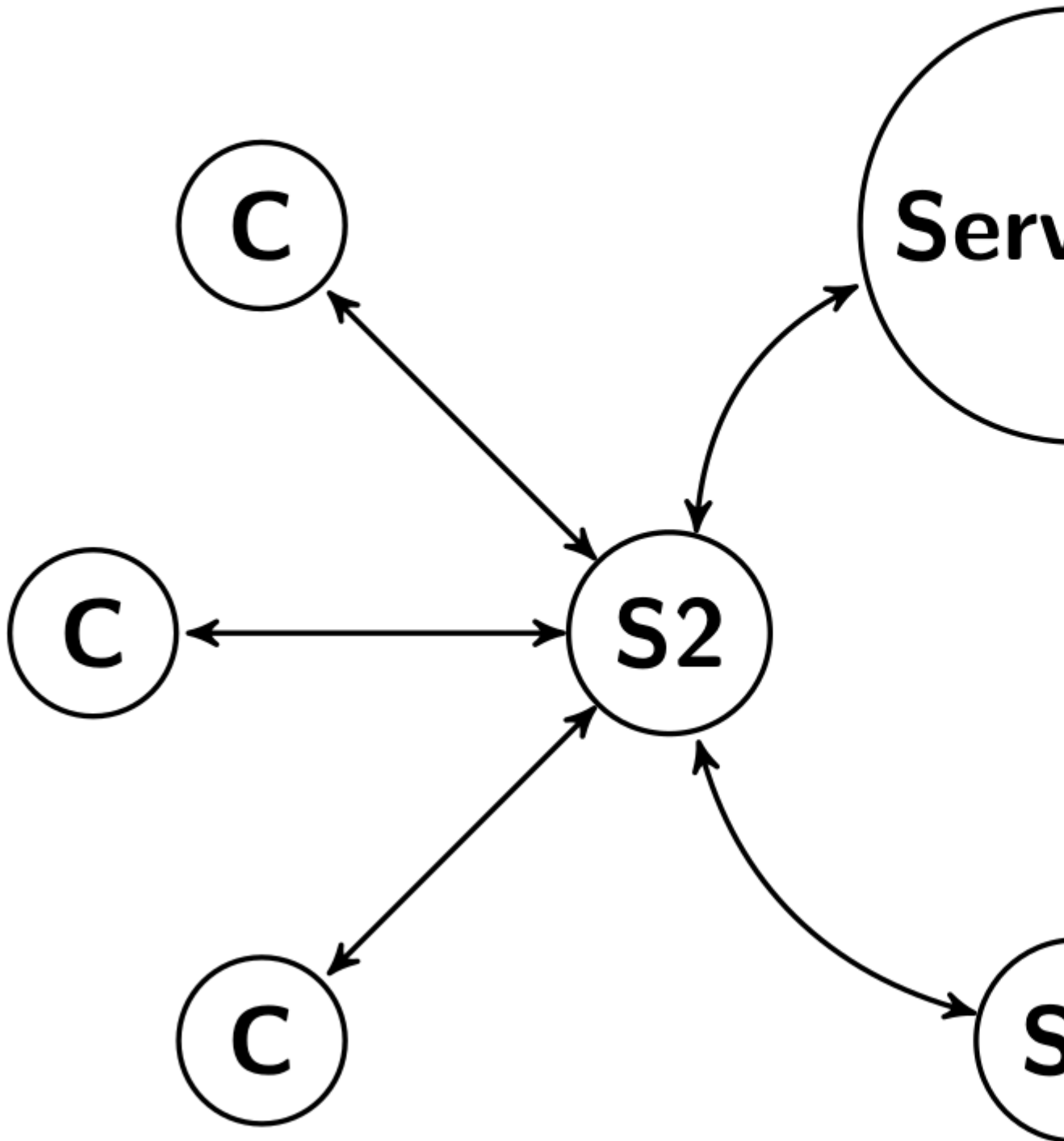
Los servidores en la red XMPP enrutan datos, pero también tienen una serie de otras responsabilidades, como mantener el estado de la sesión, almacenar datos del cliente (historial de chat, archivos, mensajes enviados cuando no había ningún cliente en línea para recibirlos,

listas de contactos, etc.). Allí es donde reside la mayor parte de la lógica empresarial del manejo de una conexión XMPP. Esto permite a los clientes permanecer tan "tontos" como sea posible (que contienen muy poca lógica).

Examples

Visualizando la red XMPP como un gráfico

La red XMPP puede considerarse como un gráfico bidireccional con servidores (S) que operan en una malla, clientes (C) agrupados en torno a su servidor local y flujos representados por bordes extravertidos:



Cuando un cliente desea enviar datos (por ejemplo, un mensaje o información de presencia) a través de la red a otro cliente, el mensaje siempre se enruta a lo largo de la ruta más corta posible

(de un cliente a su servidor, luego al cliente remoto si están en el mismo servidor o al servidor de clientes remotos y luego al cliente si el cliente remoto está en un servidor diferente).

Lea Arquitectura en línea: <https://riptutorial.com/es/xmpp/topic/3038/arquitectura>

Capítulo 3: Direcciones XMPP alias. JIDs (Identificadores Jabber)

Sintaxis

- [localpart "@" domainpart ["/" resourcepart]

Parámetros

Parte	Uso común
Localpart	Identifica una entidad XMPP (opcional)
Parte de dominio	Identifica el servicio XMPP
Resourcepart	Identifica una sesión de una entidad XMPP (opcional)

Observaciones

Las direcciones XMPP, más conocidas como JID (Jabber Identifiers) se definen en [RFC 7622](#) y actúan como direcciones en la red XMPP. Se ven como una dirección de correo electrónico, pero a veces tienen un "resourcepart" opcional al final que identifica a un cliente en particular que inició sesión como la cuenta representada por el resto de la dirección (ya que XMPP puede tener varios clientes conectados por cuenta). Un ejemplo de una dirección XMPP con el recurso de recurso (un cliente) `xyz` es:

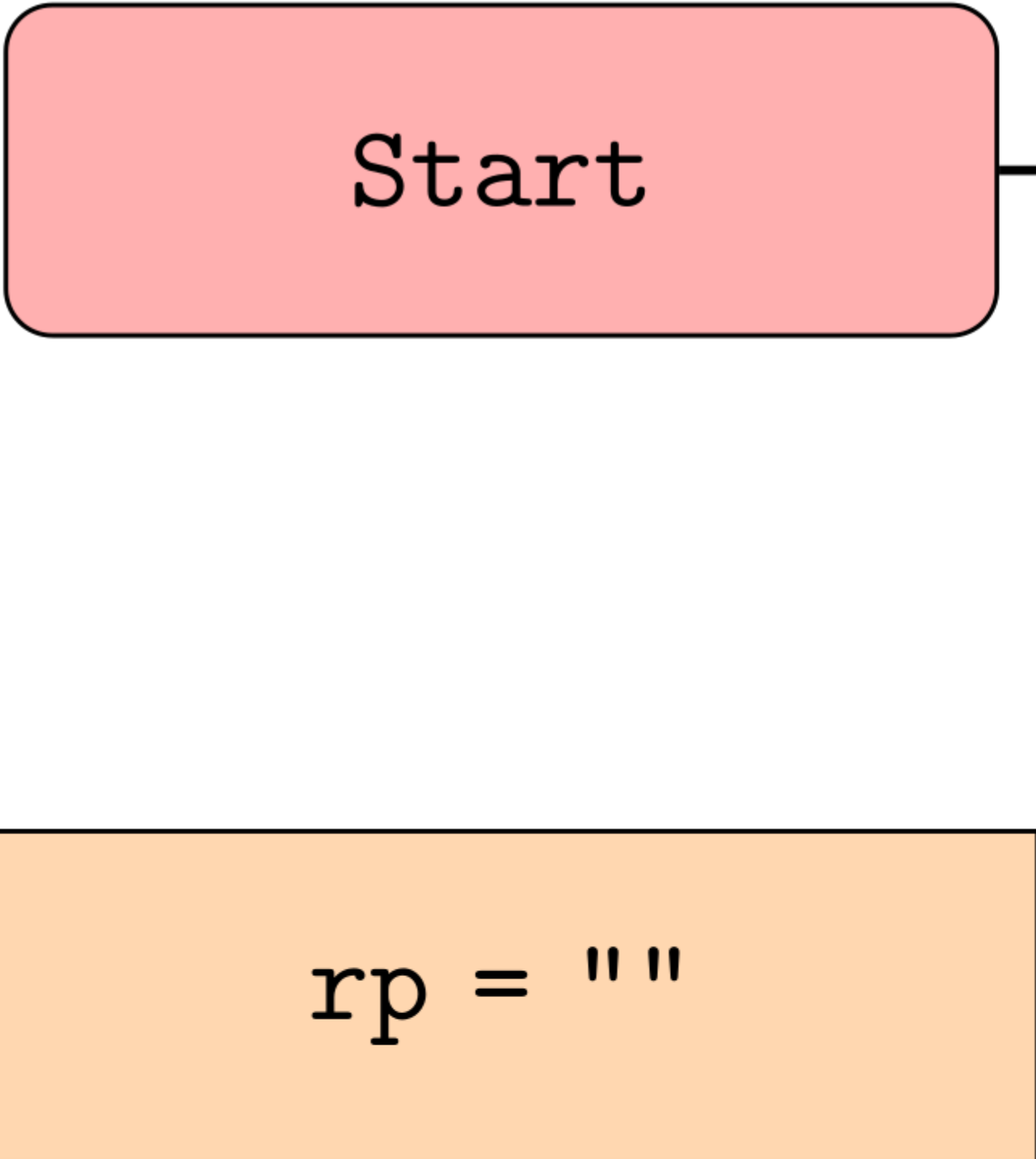
```
romeo@example.net/xyz
```

Examples

Dividiendo un JID (genérico)

Para dividir un JID en sus partes componentes (localpart, domainpart y resourcepart), se debe usar el siguiente algoritmo (donde localpart está representado por `lp`, resourcepart por `rp`, y domainpart por `dp` y `∈` se usa para verificar si el carácter dado está incluido en la cadena):

Start



rp = ""

- Compruebe que la parte local no contenga ninguno de "&'/:<>@"
- Compruebe que el resourcepart es inferior a 1024 bytes
- Compruebe que el dominio sea mayor que cero bytes y menor que 1024 bytes (y posiblemente valide que las partes individuales del dominio se ajusten a los requisitos de DNS)
- Si el dominio es una dirección IPv6 válida, asegúrese de que utiliza la notación entre corchetes (por ejemplo, [::1] lugar de ::1)

Dividiendo un JID (lr)

El paquete [mellium.im/xmpp/jid](https://github.com/mellium/im-xmpp/jid) implementa operaciones en JIDs. Para dividir una cadena JID en sus partes componentes, se puede usar la función `SplitString`:

```
lp, dp, rp, err := SplitString("romeo@example.net")
```

La función no realiza ninguna validación y no se garantiza que las partes sean válidas.

Para dividir manualmente una cadena sin depender del paquete `jid`, el código subyacente se ve así:

```
// SplitString splits out the localpart, domainpart, and resourcepart from a
// string representation of a JID. The parts are not guaranteed to be valid, and
// each part must be 1023 bytes or less.
func SplitString(s string) (localpart, domainpart, resourcepart string, err error) {

    // RFC 7622 §3.1. Fundamentals:
    //
    // Implementation Note: When dividing a JID into its component parts,
    // an implementation needs to match the separator characters '@' and
    // '/' before applying any transformation algorithms, which might
    // decompose certain Unicode code points to the separator characters.
    //
    // so let's do that now. First we'll parse the domainpart using the rules
    // defined in §3.2:
    //
    // The domainpart of a JID is the portion that remains once the
    // following parsing steps are taken:
    //
    // 1. Remove any portion from the first '/' character to the end of the
    // string (if there is a '/' character present).
    sep := strings.Index(s, "/")

    if sep == -1 {
        sep = len(s)
        resourcepart = ""
    } else {
        // If the resource part exists, make sure it isn't empty.
        if sep == len(s)-1 {
            err = errors.New("The resourcepart must be larger than 0 bytes")
            return
        }
        resourcepart = s[sep+1:]
        s = s[:sep]
    }
}
```

```

// 2. Remove any portion from the beginning of the string to the first
// '@' character (if there is an '@' character present).

sep = strings.Index(s, "@")

switch sep {
case -1:
    // There is no @ sign, and therefore no localpart.
    localpart = ""
    domainpart = s
case 0:
    // The JID starts with an @ sign (invalid empty localpart)
    err = errors.New("The localpart must be larger than 0 bytes")
    return
default:
    domainpart = s[sep+1:]
    localpart = s[:sep]
}

// We'll throw out any trailing dots on domainparts, since they're ignored:
//
// If the domainpart includes a final character considered to be a label
// separator (dot) by [RFC1034], this character MUST be stripped from
// the domainpart before the JID of which it is a part is used for the
// purpose of routing an XML stanza, comparing against another JID, or
// constructing an XMPP URI or IRI [RFC5122]. In particular, such a
// character MUST be stripped before any other canonicalization steps
// are taken.

domainpart = strings.TrimSuffix(domainpart, ".")

return
}

```

Dividiendo un JID (Rust)

En Rust, la `xmpp-addr` ([docs](#)) se puede utilizar para manipular JID. Para dividir un JID en sus partes componentes (sin validar que esas partes son válidas), se puede usar la función `Jid::split`:

```

let (lp, dp, rp) = Jid::split("feste@example.net");
assert_eq!(lp, Some("feste"));
assert_eq!(dp, "example.net");
assert_eq!(rp, None);

```

Lea Direcciones XMPP alias. JIDs (Identificadores Jabber) en línea:

<https://riptutorial.com/es/xmpp/topic/3036/direcciones-xmpp-alias--jids--identificadores-jabber->

Capítulo 4: Negociación de flujo

Observaciones

Las conexiones XMPP comprenden dos flujos XML: uno para ingreso y otro para egreso. Estas secuencias generalmente se envían a través de la misma conexión TCP (aunque a veces se pueden usar múltiples conexiones, especialmente para las conexiones de servidor a servidor) y comparten ciertas características para las cuales se requiere negociación (por ejemplo, autenticación con SASL).

Examples

Cerrando un arroyo

Un flujo se cierra enviando una etiqueta de cierre `</stream>`. Después de que se envía la etiqueta de flujo de cierre, no se deben enviar más datos en el flujo (incluso en respuesta a los datos recibidos de la otra parte). Antes de cerrar la conexión, la entidad emisora debe esperar una etiqueta de respuesta `</stream>` para dar tiempo a la otra parte para enviar los datos pendientes y debe expirar (y terminar la conexión TCP subyacente [s]) si una etiqueta de flujo de cierre es no recibido dentro de un período de tiempo elegido.

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

Si la transmisión está encriptada con TLS, las partes deben finalizar TLS de forma limpia mediante el envío de una alerta TLS `close_notify` y recibir una en respuesta. Tu biblioteca TLS probablemente hace esto por ti.

Comenzando una corriente

Una vez que se establece una conexión TCP, la entidad iniciadora envía el encabezado de la secuencia inicial. De manera similar, siempre que se requiera un reinicio de flujo (por ejemplo, después de negociar una capa de seguridad como TLS), también se debe enviar un encabezado de flujo:

```
<?xml version='1.0'?>
<stream:stream
  from='juliet@im.example.com'
  to='im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
```

El encabezado XML es opcional, pero si existe, no debe especificar nada más que la versión 1.0 de XML con codificación UTF-8.

En respuesta, la entidad receptora enviará su propia etiqueta de secuencia de apertura que contiene un ID de sesión único:

```
<?xml version='1.0'?>
<stream:stream
  from='im.example.com'
  id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
  to='juliet@im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
```

Cierre la conexión XMPP usando la biblioteca agsxmpp

```
public class ConnectionManager
{
    private XmppClientConnection _xmppClientConnection = null;

    public ConnectionManager()
    {
        if (_xmppClientConnection == null)
        {
            _xmppClientConnection = new XmppClientConnection();
        }
    }

    public void CloseXmppConnection()
    {
        try
        {
            if (_xmppClientConnection != null)
            {
                //Close xmpp Client Connection
                _xmppClientConnection.Close();
            }
        }
        catch (Exception ex)
        {
        }
    }
}
```

Lea Negociación de flujo en línea: <https://riptutorial.com/es/xmpp/topic/4248/negociacion-de-flujo>

Creditos

S. No	Capítulos	Contributors
1	Empezando con xmpp	Afsheen126 , bilal , Community , Flow , ge0rg , khan , Sam Whited
2	Arquitectura	Sam Whited
3	Direcciones XMPP alias. JIDs (Identificadores Jabber)	Flow , ge0rg , Sam Whited
4	Negociación de flujo	khan , Sam Whited