The Design and Implementation of Core Function of XMPP-Based Mobile Push System

Zhao Jiantao, Huang Yunyi

Department of Control and Computer Engineering,

North China Electric Power University

Beijing 102206, China

Department of Control and Computer Engineering

North China Electric Power University

Beijing 102206, China

zhaojiantao66@126.com,1203571263@qq.com

Keywords: Push System; XMPP; XML; Java EE ;Android

Abstract. The purpose of this article is to design and implement a XMPP-Based C/S system in which the server pushes information to the client actively. By realizing the client to connect to the server, client registration, the client to login the server and the server to push messages to client 4 core functions through XMPP, this article accomplished the core of a push system. In addition, the use of uninterrupted heartbeat package maintains a long connection, ensuring the client and the server connection. This article has important significance on future mobile push system research.

Introduction

With the rapid development of smart devices, mobile Internet has received a full promotion. *Chinese Internet Network Information Center* (CNNIC) shows that in the thirty-fourth *Statistic Report On Internet Development in China* , in 2014 June, the number of domestic mobile phone users reached 527000000, compared to 2013, increased by 26990000 [1]. The way a user access to data can be classified into two kinds: the client pull and the server push. When data arrival time is unpredictable, the user can only use the polling way to detect whether there is new data coming with the former way. It is inflexible. It not only needs to consume enormous system resources and network traffic but also can't ensure the efficiency and timeliness of the information arrival. While with the way the server pushing data to the client, users can acquire data as soon as possible at lower system resource and network traffic consumption. Therefore, the research of mobile push system is necessary.

Related Knowledge

XMPP is an open Extensible Markup Language (XML) protocol for near-real-time messaging, presence, and request-response services [2]. In XMPP, there are three kinds of roles: server, client and gateway. They can communicate with each other. The server is responsible for message log, message routing and connection management. The client connects directly to a server over a TCP connection and use XMPP to take full advantage of the functionality provided by a server and any associated services. The gateway is the role of interoperability with other instant communication platform.

XMPP Address Format. The communication process of XMPP is actually two network entity communicate with each other, which requires each network entity has an address, and the addresses should be unique to complete the communication smoothly. The XMPP physical address is often referred to as Jabber Identifier or JID because Jabber is the predecessor of XMPP. A valid JID includes a set of elements, including the domain identifier, the node identifier and the resource identifier. A standard JID format as shown below:

jid = [node "@"] domain ["/" resource]
domain = fqdn / address-literal
fqdn = (sub-domain 1*("." sub-domain))
sub-domain = (internationalized domain label)
address-literal = IPv4address / IPv6address

All JID are based on the above structure, it represents an instant messaging entity. The domain part of a JID is unique and represents a server or gateway. Node is always a username the user registered in the server. Resource refers to the way the user login the server; a user can use different ways to login the server (e.g., via your desktop client or your mobile client).

XMPP Message Format. Two fundamental concepts make possible the rapid, asynchronous exchange of relatively small payloads of structured information between presence-aware entities: XML streams and XML stanzas. An XML stream is a container for the exchange of XML elements between any two entities over a network. The start of an XML stream is denoted unambiguously by an opening XML <stream> tag while the end of the XML stream is denoted unambiguously by a closing XML </stream> tag. An XML stanza is a discrete semantic unit of structured information that is sent from one entity to another over an XML stream. XMPP has defined three kinds of XML stanza: Message（<message />）,Presence(<presence />),IQ（<iq />）.

Message(<message/>).The <message/> stanza kind can be seen as a "push" mechanism whereby one entity pushes information to another entity, similar to the communication that occur in a system such as email

Presence(<presence />).Presence advertises the network availability and status messages of entities. It is a specialized publish-subscribe method; people who requested subscription to your presence and authorized by you receive updated presence information when you come online, and go offline, and change your status

IQ(<iq/>).Info/Query, or IQ, is a request-response mechanism, similar in some ways to HTTP. The semantics of IQ enable an entity to make a request of, and receive a response from, another entity.

Achieve Push System Through Xmpp

In order to achieve a message delivery process, the following steps must be included:

1) Establish a connection between client and server
2) The client to register and log in the server
3) The client to receive messages pushed by the server

And to receive real-time messages pushed by the server, the client must maintain a persistent connection with the server. The following is how to achieve the process mentioned above through XMPP.

The following is how to achieve the process mentioned above through XMPP.

Establish a Connection Between Client and Server. To establish a connection, the client needs to send a connection establishment request to the server. The following data should be sent to the server:

<stream:stream to="10.0.2.2" xmlns="jabber:client" xmlns:stream="http://etherx.jabber.org/streams" version="1.0">

In above data, “stream” is an open tag of an XML stream，it says to start transmitting XML stanza, and “to” describes the server's IP address the XML stream needs to be sent to.

After the server receives the XML data, it needs to response to the client:

<?xml version=’1.0’ encoding=’UTF-8’?>

<stream:stream xmlns:stream=”http://etherx.jabber.org/streams” xmlns=”jabber:client” from=”127.0.0.1” id=”f78f7da0” xml:lang=”en” version=”1.0”>

In above data, the “from” is the IP address of the client, the “ID” is created by the server to identify a session between a client and server . In addition to send the above data to the client shows that data has been received, the server also need to negotiate with clients on some connection details

and the expanded namespaces needed to be used. It means the server needs to send following more data to the client:

<stream:features>
<starttls xmlns="urn:ietf:params:xml:ns:xmpp-tls"></starttls>
<auth xmlns="http://jabber.org/features/iq-auth"/>
<register xmlns="http://jabber.org/features/iq-register"/>
</stream:features>

In above data, the "<starttls>" stanza is a consultation with the client whether to use the TLS protocol, and the "<auth>" and "<register>" stanza is the expanded namespace needed to be used later. The push system does not need encryption, so the "<starttls>" stanza can be ignored directly. At this point, the connection between the client and the server is established.

The Client to Register And Login The Server. In order to register and login the server, the client need to send username, encrypted password and resource name to the serves:

<iq id="SB3y1-1" type="set">
<query xmlns="jabber:iq:auth">
<username>sam</username>
<digest>bd24d2072faa41a6268da28eb728932280aa7c78</digest>
<resource>AndroidpnClient</resource>
</query>
</iq>

The above data used the expanded namespace mentioned before- "auth", the server and the client both need to provide a realization of this namespace. The server should check the username and password sent by the client in the database. If the username and password are in the database, the server should send the following data to notify the client that the login is successful [3]:

<iq type="result" id="SB3y1-4" to="sam@127.0.0.1/AndroidpnClient"/>

If there is no corresponding data in the database, the server needs to send the following data to notify the client the failed login:

<iq type="error" id="SB3y1-1" to="127.0.0.1/f78f7da0">
<query xmlns="jabber:iq:auth">
<username>sam</username>
<digest>bd24d2072faa41a6268da28eb728932280aa7c78</digest>
<resource>AndroidpnClient</resource>
</query>
<error code="401" type="auth">
<not-authorized xmlns="urn:ietf:params:xml:ns:xmpp-stanzas"/>
</error>
</iq>

The data tells the client there is an “authorized” error, which means the user is not registered. After receives this error, the client needs to send the registration request to the server [4]:

<iq id="SB3y1-2" type="set">
<query xmlns="jabber:iq:register">
<password>d12f6403cd094721867fa750f079f828</password>
<username>sam</username>
</query>
</iq>

Above XML stanza used the second expanded name space "register" mentioned before to send the username and password to the server. Then the server will insert the user into the database, thus the user registration is completed. After completed the registration, the client can log in the server.

The Client to Receive Messages Pushed by The Server. When the client has already login the server, the server can easily push messages to the client. The server sends the following data to strike a push:

<iq type="set" id="88-0" to="sam@127.0.0.1/AndroidpnClient">

```
<notification xmlns="androidpn:iq:notification">
<id>ffe31f20</id>
<title>Hello World</title>
<message>hello world </message>
<uri></uri>
</notification>
</iq>
```

The “to = sam@127.0.0.1/AndroidpnClient” in above data is the destination of the message, namely JID(Jabber Identifier) of the client. And child element < notification > encapsulates the content of the message, "id" is the identity of the message, "title" is the title of the message, "message" is the contents of the message and “uri” is the address of the other resources included by the message. “uri” property can be used to realize multimedia message delivery. The server can add the address of pictures or movies it wants to push into the message to the “uri”.

Maintain a Persistent connection to The Server. To maintain a persistent connection to the server, the client needs to send a heartbeat packet to the server regularly. The following code in the client can maintain a persistent connection from client to server [5]:

```
private class KeepAliveTask implements Runnable {
while (!done && keepAliveThread == thread) {
    synchronized (writer) {
      if (System.currentTimeMillis() - lastActive >= delay) {
            try {
                  writer.write(" ");
                  writer.flush();
            } catch (Exception e) {
            }
        }
}
```

System Implementation

Based on XMPP realization above, this article uses Java code to complete a simple push system; the following is some screenshot of the push system.

Server-side Function Page. Figure 1 is a notification sending interface of the server; users can send messages to clients through this interface.

Fig1 Notification sending interface

Client-side Function Page. Figure 2 is the messages viewing interface; through this interface, users can view the notice details.

Fig2 Notification details interface

References

[1] Chinese Internet Network Information Center (CNNIC). The thirty-fourth Statistic Report On Internet Development in Chin [J]. Internet world, 2014, (7).

[2] Ozturk O. Introduction to XMPP protocol and developing online collaboration applications using open source software and libraries[C]. //International Symposium on Collaborative Technologies and Systems. IEEE, 2010:21 - 25.

[3] Lu X, Lei W, Zhang W. The Design and Implementation of XMPP-Based SMS Gateway[C]. //International Conference on Computational Intelligence, Communication Systems and Networks. IEEE, 2012:145 - 148.

[4] P. Saint-Andre, Extensible Messaging and Presence Protocol (XMPP): Core, IETF RFC 3920, October 2004; www.rfc-editor.org/rfc/rfc3920.txt.

[5] http://wangqinghua123.iteye.com/blog/1341473