|  |
| --- |
|  |
| **Androidpn推送框架技术概要** |
| 安卓消息即时推送框架 |
|  |

|  |
| --- |
| 基于开源项目androidpn实现的安卓消息推送框架，文中资料来源于网络 |

目录

[一、Androidpn框架 2](#_Toc502927830)

[1 XMPP协议 2](#_Toc502927831)

[二、服务端 5](#_Toc502927832)

[1服务端实现功能： 5](#_Toc502927833)

[2服务端离线消息推送与回执机制 5](#_Toc502927834)

[3服务端基本原理 5](#_Toc502927835)

[4消息处理流程 7](#_Toc502927836)

[5启动过程： 7](#_Toc502927837)

[三、客户端 7](#_Toc502927838)

[1 客户端概述 7](#_Toc502927839)

[2 客户端主要内容 8](#_Toc502927840)

[3自定义消息 8](#_Toc502927841)

[4 连接、注册、登录 8](#_Toc502927842)

[四、操作 9](#_Toc502927843)

[1 服务端操作界面 9](#_Toc502927844)

[2客户端消息接收 13](#_Toc502927845)

[五、注意事项 14](#_Toc502927846)

[1 服务端 14](#_Toc502927847)

[2客户端 14](#_Toc502927848)

# 一、Androidpn框架

Server

***Socket***

XMPP协议

XMPP协议

Client

## 1 XMPP协议

**（1）定义**

全称:可扩展通讯和表示协议；

XMPP是一种基于[标准通用标记语言](https://baike.baidu.com/item/%E6%A0%87%E5%87%86%E9%80%9A%E7%94%A8%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)的子集[XML](https://baike.baidu.com/item/XML)的协议，它继承了在XML环境中灵活的发展性。因此，基于XMPP的应用具有超强的可扩展性。经过扩展以后的XMPP可以通过发送扩展的信息来处理用户的需求，以及在XMPP的顶端建立如内容发布系统和基于地址的服务等应用程序。而且，XMPP包含了针对服务器端的软件协议，使之能与另一个进行通话，这使得开发者更容易建立客户应用程序或给一个配好系统添加功能

XMPP本质上是一种XML流技术。客户端开始和XMPP服务器会话，会打开一个长时间的TCP连接，然后和服务器协商一个流。一旦你和你的服务器建立了一个XML流，你和你的服务器可以通过流交换三个特殊的XML片段：<message/>，<presence/>，<iq/>.这些片段称为XML节。是XML中最有意义的基本单元，而且一旦你已建立一个XML流，你可以通过流发送无数个节。

**（2）基本网络结构**

XMPP中定义了三个角色，客户端，服务器，[网关](https://baike.baidu.com/item/%E7%BD%91%E5%85%B3)。通信能够在这三者的任意两个之间双向发生。服务器同时承担了客户端信息记录，连接管理和信息的路由功能。网关承担着与异构[即时通信系统](https://baike.baidu.com/item/%E5%8D%B3%E6%97%B6%E9%80%9A%E4%BF%A1%E7%B3%BB%E7%BB%9F)的互联互通，[异构](https://baike.baidu.com/item/%E5%BC%82%E6%9E%84)系统可以包括[SMS](https://baike.baidu.com/item/SMS)（短信），[MSN](https://baike.baidu.com/item/MSN)，[ICQ](https://baike.baidu.com/item/ICQ)等。基本的网络形式是单客户端通过[TCP/IP](https://baike.baidu.com/item/TCP%2FIP)连接到单服务器，然后在之上传输XML

**（3）XMPP消息格式**  
XMPP中定义了3个顶层XML元素: Message、Presence、IQ。

<Message>

用于在两个jabber用户之间发送信息。Jsm(jabber会话管理器)负责满足所有的消息，不管目标用户的状态如何。如果用户在线jsm立即提交;否则jsm就存储。

* **To** :标识消息的接收方。
* **from** : 指发送方的名字或标示(id)o
* **Text**: 此元素包含了要提交给目标用户的信息。

结构：:

<message to= [‘lily@jabber.org/contact’](mailto:%E2%80%98lily@jabber.org/contact%E2%80%99) type =’chat’>

<body>内容</body>

</message>

 <Presence>

用来表明用户的状态，如：online、away、dnd(请勿打扰)等。当用户离线或改变自己的状态时，就会在stream的上下文中插入一个Presence元素，来表明自身的状态。

结构：

<presence>

From =‘lily @ jabber.com/contact’

To = ‘yaoman @ jabber.com/contact'

<status> Online </status>

</presence>

< IQ >

一种请求／响应机制，从一个实体从发送请求，另外一个实体接受请求，并进行响应．例如，client在stream的上下文中插入一个元素，向Server请求得到自己的好友列表，Server返回一个里面是请求的结果．

* <iq > 主要的属性是type。包括:
* Get :获取当前域值。
* Set :设置或替换get查询的值。
* Result :说明成功的响应了先前的查询。
* Error: 查询和响应中出现的错误。

结构:

<iq id="xMOfN-6" type="set">

<deliverconfirm xmlns="androidpn:iq:deliverconfirm">

<uuid>4e8e8ed7</uuid>

</deliverconfirm>

</iq>

**（4）一个简单XMPP流示例：**

C: <stream:stream>

C: <presence/>

C: <iq type="get">

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

</iq>

S: <iq type="result">

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

<item jid="suke@skh.whu.edu.cn"xs/>

<item jid="gmz@skh.whu.edu.cn"/>

<item jid="beta@skh.whu.edu.cn"/>

</query>

</iq>

C: <message from="suke@skh.whu.edu.cn"

to="beta@skh.whu.edu.cn">

<body>现在几点了？</body>

</message>

S: <message from="lj@skh.whu.edu.cn"

to="cyl@skh.whu.edu.cn ">

<body>昨天的这个时候</body>

</message>

C: <presence type="unavailable"/>

C: </stream:stream>

**注**：C代表客户端，S代表服务端。

**（5）XMPP通信概述**

XMPP的通信中，服务器端和客户端的通信是基于一个session（会话）过程，会话开始，首先客户端会指定端口号，然后把准备的连接信息发送到服务器端，客户端通过XMPP协议做的只有接收消息，其他的管理连接和保存消息等都由服务器负责。消息的传递是以根节点<stream>为起始，以</stream>为结束。

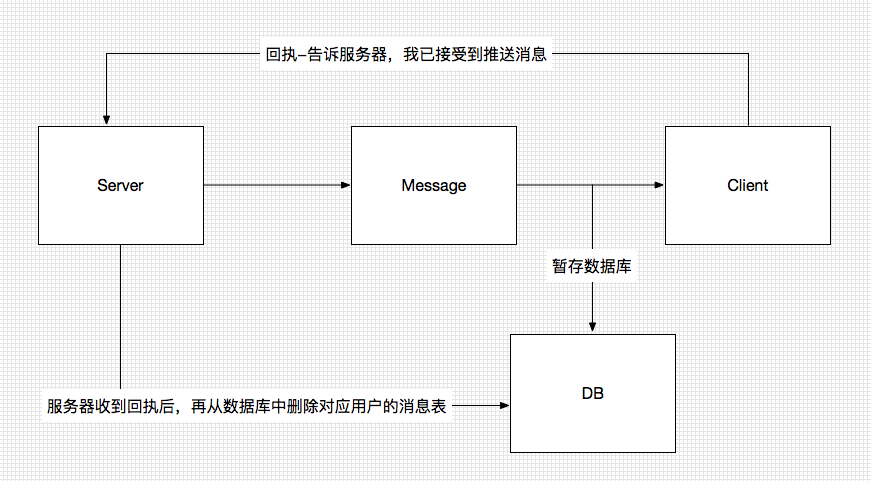
服务器和客户端建立Tcp连接过程如下。首先，建立会话协商，Client 准备connecting to server (127.0.0.1:5222)。其次，服务器询问Client所支持的安全认证。再者，Client发送给服务器要注册的用户。然后，Client根据服务器的要求提交用户信息。最后，Client关闭注册的STREAM元素。

# 二、服务端

## 1服务端实现功能：

1. 展示已注册用户信息（状态、注册名、创建时间）。
2. 当前sessions信息展示。
3. 推送消息（通知所有用户、指定用户、指定用户组）。
4. 离线消息推送。

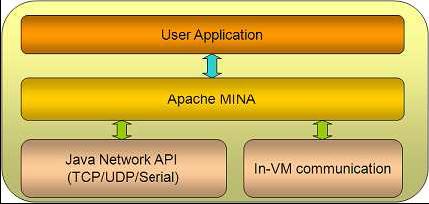
## 2服务端离线消息推送与回执机制



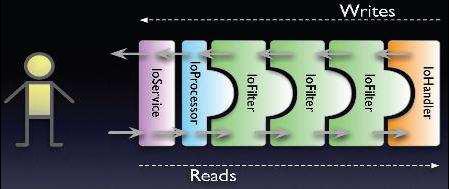
整个机制在图中描述很清晰，不再赘述。

## 3服务端基本原理

  在Androidpn服务端的底层主要采用的两大框架mina和openfire两大框架，其中mina主要为底层数据传输的Socket框架。  
    Apache Mina Server 是一个网络通信应用框架，它主要是对基于TCP/IP、UDP/IP协议栈的通信框架(也可以提供JAVA 对象的序列化服务、虚拟机管道通信服务等),Mina 同时提供了网络通信的Server 端、Client 端的封装，无论是哪端，Mina 在整个网通通信结构中都处于如下的位置：



Mina 的API 将真正的网络通信与我们的应用程序隔离开来，你只需要关心你要发送、接收的数据以及你的业务逻辑即可。同样的，无论是哪端，Mina 的执行流程如下所示：



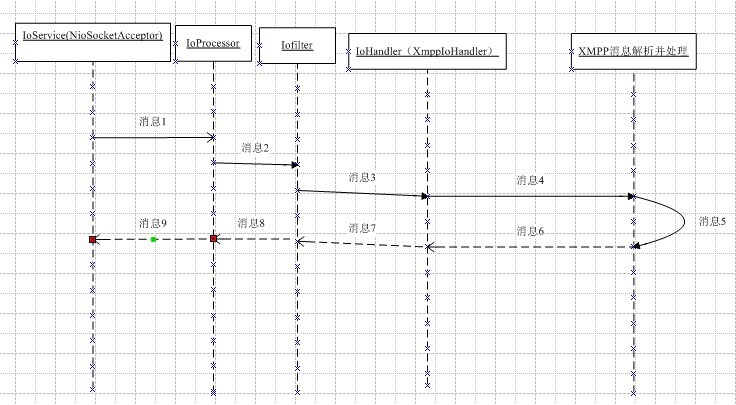
**(1). IoService**：这个接口在一个线程上负责套接字的建立，拥有自己的Selector，监听是否连接被建立。

**(2). IoProcessor**：这个接口在另一个线程上负责检查是否有数据在通道上读写，也就是说它也拥有自己的Selector，这是与我们使用JAVA NIO 编码时的一个不同之处，通常在JAVA NIO 编码中，我们都是使用一个Selector，也就是不区分IoService与IoProcessor 两个功能接口。另外，IoProcessor 负责调用注册在IoService 上的过滤器，并在过滤器链之后调用IoHandler。

**(3). IoFilter**：这个接口定义一组拦截器，这些拦截器可以包括日志输出、黑名单过滤、数据的编码（write 方向）与解码（read 方向）等功能，其中数据的encode 与decode是最为重要、也是你在使用Mina 时最主要关注的地方。

**(4). IoHandler**：这个接口负责编写业务逻辑，也就是接收、发送数据的地方。

## **4消息处理流程**



## **5启动过程：**

1.**spring**初始化并启动过程，调用NioSocketAcceptor。

2.**NioSocketAcceptor**开始执行调用IoProcessor.

3.**IoProcessor**开始调用FilterChain。FilterChain调用相关的IoFilter的。其中ProtocolCodecFilter的过滤器调用了org.androidpn.server.xmpp.codec.XmppCodecFactory进行编码。

4.**XmppIoHandler**实现自IoHanlder并调用通过openfire 的XMLLightweightParser解析相关的业务逻辑。

5.解析信息并调用**xmpp**并处理信息，最后交给路由类router来转发到相应的Handler处理。

# 三、客户端

## 1 客户端概述

客户端采用基于java的XMPP协议包asmack（该包依赖于openfire下的开源项目smack）。通过该协议包提供的XMPPConnetcion类与服务器建立持久连接，并通过该连接进行用户注册和登录认证，以及接受服务器消息。

Client包含有消息的收发，解析以及持久连接的发起，重连等功能呢，十分强大，我们开发时完全不用管底层的连接，也不用担心断线，可以专注于业务部分的开发。

## 2 客户端主要内容

* 控制器

XmppManager是Client的主控制器，主要用来管理连接信息，包括XMPP端口、IP地址、登录的用户名和密码，以及对连接的维护。

* 消息解析处理

NotificationIQ、NotificationPackerListener以及NotificationIQProvider三个类负责对收到的Notification格式的消息进行解析和处理。

* 手机状态监听

PersistentConnetcionListener、PhoneStateChangeListener以及ReconnectionThread三个类负责监听手机状态和断线重连。

## 3自定义消息

自定义消息时需要定义3个类：在NotificationIQ中定义消息的实体，在NotificationIQProvider中将消息转化为NotificationIQ实体，在NotificationPacketListener中对实体进行处理，在定义这些类之后，还需要在XmppManager中将这3个类中注册到connection中。

需要注意的是，注册NotificationIQProvider时，传入的namespace需要和服务端组装消息时使用的namespace一致，才能正确的收到。

## 4 连接、注册、登录

* **连接**

通过创建asmack库提供的XMPPConnection类的一个实例，然后指定主机及端口后即可创建连接。

* **注册**

Client在与服务器建立connection后，首先会提交注册任务，在Androidpn项目中，用户注册是采用UUID（通用唯一标识码）来实现的，其每次登录都会利用这个UUID.randomUUID()来产生一个用户名和密码，而且不会重复。

注册方法根据Server的格式要求组装注册消息，发送到服务器，同时使用在connection中添加监听，来获取服务器返回的消息packet。Client会根据消息的类型来判断是否注册成功。

服务端收到Client的注册请求，会通过路由类router来转发到相应的Handler处理，router首先会读取Client发来包Packet的Namespace部分，根据XMPP RFC协议注册packet的namespace为jabber:iq:register，授权packet的namespace为jabber:iq:auth，由此转发到IQRegisterHandler处理注册请求。

如果注册成功，Client会把相应的用户名、密码通过SharePerence保存在Client的共享文件AndroidpnClient中。此外，AndroidpnClient还保存了当前应用的包名、类名、服务器地址、端口、客户端版本、设备ID等。

* **登录**

Client的登录是在注册之后进行的，在Client的XmppManager类中，建立连接的时候首先会提交登录任务，然而在提交登录任务的时候都有一个submitRegisterTask()方法，可知，每次登录都会首先提交一个注册任务。

登录的时候，要先通过isAuthenticated()方法判断是否授权，授权处理同样会通过connection发送到服务端，服务端会根据packet的namespace部分进行解析，然后路由类router会根据namespace转发到IQAuthHandler处理。

Client收到服务端返回的消息packet，会使用connection的监听器异步接收消息，并通过XmppManager的context的sendBroadcast(Intentintent)以广播发送出去，同时还会回执一条消息给服务端，以知晓其发送消息已收到。

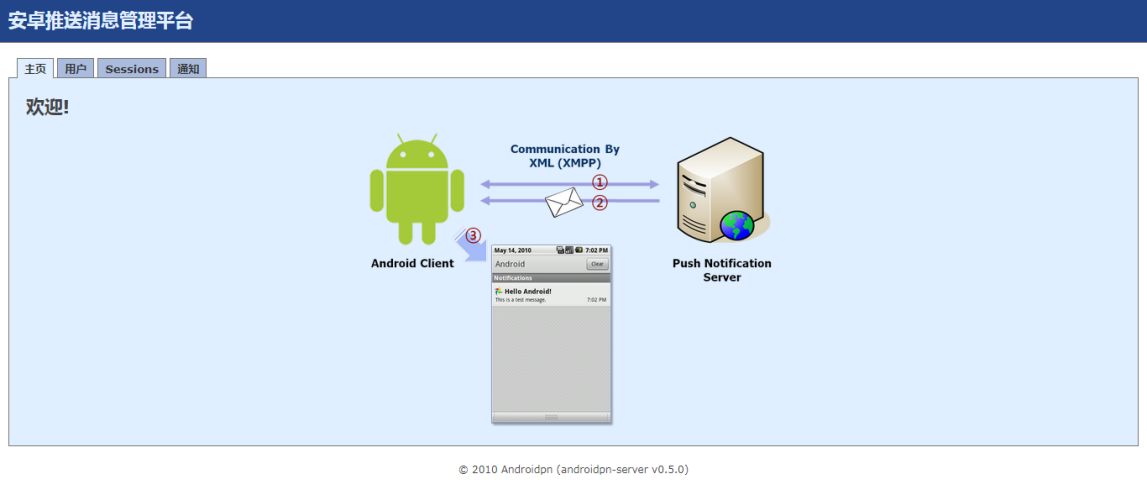
* **重连开启时机**

当连接、注册、登录过程中发生失败都会取消当前任务组并开启重连机制

# 四、操作

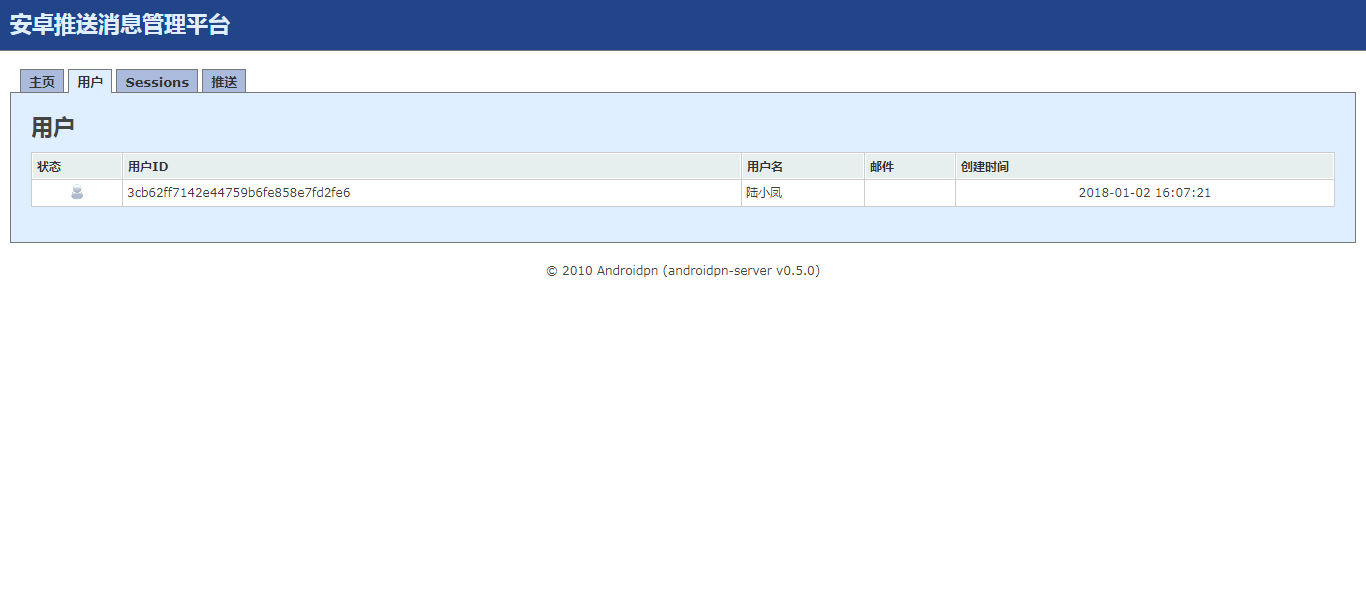
## 1 服务端操作界面

1. 主页



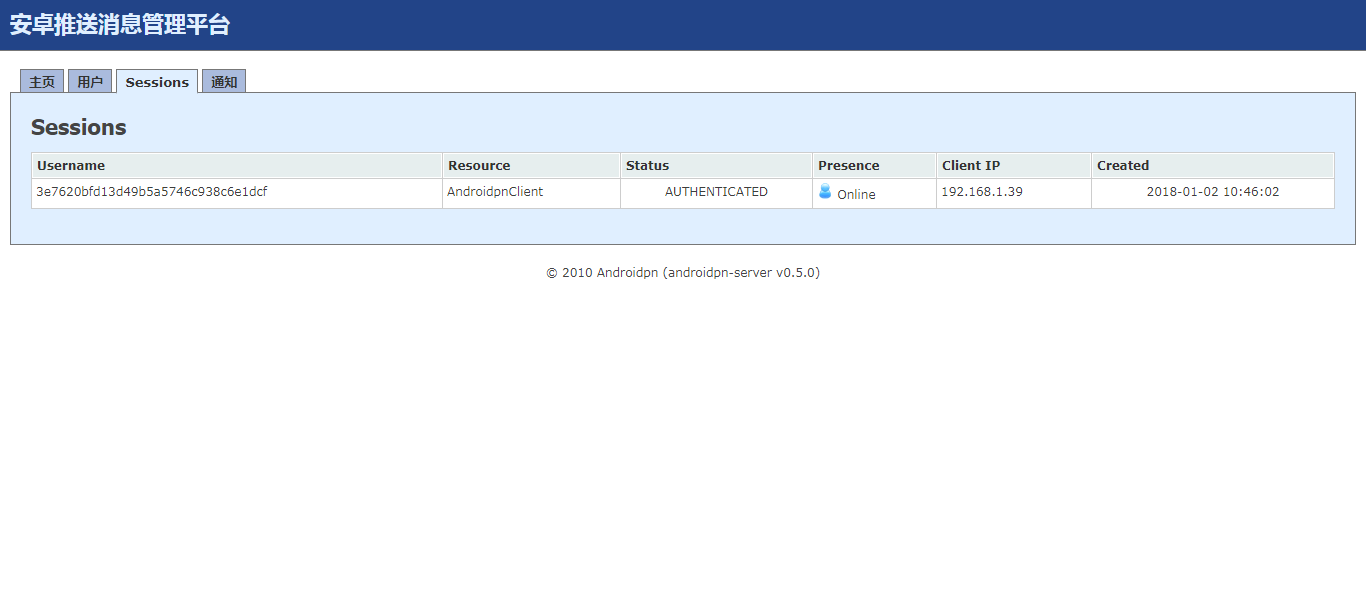
控制面板主页。

1. 用户



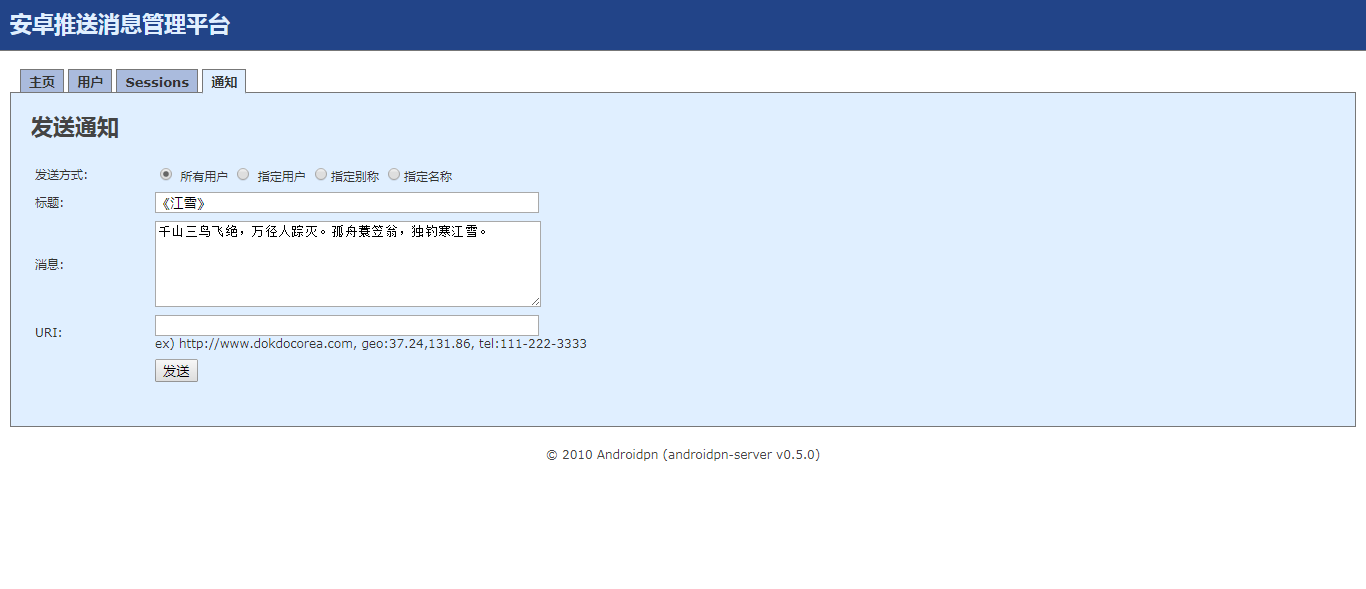
展示所有已注册用户信息。

（3） sessions



展示当前所有有效的会话信息。

1. 通知--所有用户



发送给所有已注册用户，无论是否在线，如果用户未在线，消息会先存入数据库，当用户变为在线时再发送消息。

1. 通知—指定用户ID



根据指定用户注册时提供的ID（见“用户”页面列表）将消息推送给特定用户，此ID是惟一的。

1. 通知—指定别称



根据指定用户连接时提供的一个别称将消息推送给特定用户，这个别称是临时存放在服务端的（也就是说靠这个别称推送是不靠谱的，这只是一种技术上的参考）。

1. 通知—指定名称



这个名称可能会有相同，因此这里可能是发给一个或多个用户。根据业务需求可以使用该字段对用户分组。

## 2客户端消息接收



当客户端收到消息后会在通知栏收到提示。

# 五、注意事项

## 1 服务端

* 离线消息应该加入健全的清理机制，否则，数据库中会积累大量过时无用消息。
* 服务端session的readerIdleTime设置需要根据实际网络环境进行调整，并且需要与客户端心跳相关参数相适配，以达到连接最佳与资源占用最少间的平衡。

## 2客户端

* **遗留问题**

当前客户在注册是使用UUID随机生成的作为 username 和 password ，为了避免导致同一设备在服务端数据库库注册多不同条数据，在客户端采用SharedPreferences 保存注册后的 username 和 password，当注册时对SharedPreferences 进行检测，如果有对应数据代表注册过，不向服务器添加该信息。

但是，当把客户端的数据清除(或卸载后重新安装)，SharedPreferences 里的数据会被清除，这会导致同一设备在服务端数据库库注册多不同条数据！

* **参考解决思路：**

**1、**使用设备ID或 MAC替换客户端源码中使用UUID随机生成的作为 username 和 password ，但是，如果把客户端随机生成的UUID代码，替换为设备的ID（SharedPreferences 已经保存有）或者MAC作为用户名，服务器端会出现重复插入的错误。

**2、**把客户端的数据清除(或卸载后重新安装)，那么 SharedPreferences 里的数据也会被清除，然而服务器端又有我们手机的设备 ID，这时客户端启动程序从首选项中读取不到 username 和 password 会重新拿着相同的设备 ID 提交给服务器进行注册，这时服务器端就会出现重复插入的问题。但是，只要在服务器端保存用户信息的时候，检查数据库中是否存在该用户，不存在就插入该用户，问题迎刃而解。（这个简单问题留给使用者练手）。