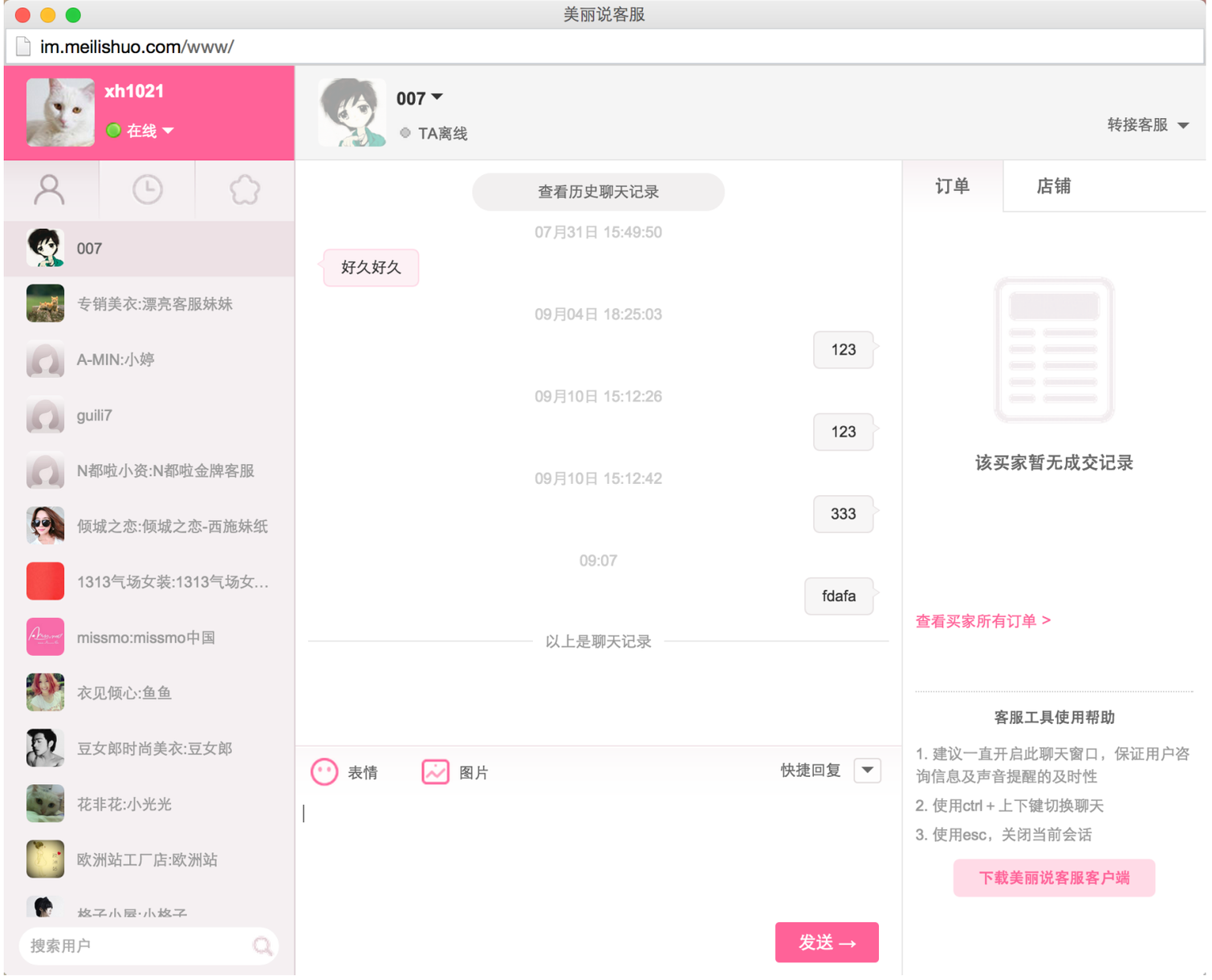
前端在IM中的应用

页面端 及 服务端



页面端：PC页面1

主站商家界面：功能丰富，提供商家操作功能

（在线状态切换、订单店铺信息、添加黑名单、转接客服、快捷回复）

规则：

- from：发消息人，当前登录用户
- to：收消息人，列表中选择的聊天对象

页面端：PC页面2

主站用户界面：商家版本分离出来的简洁版本，只提供用户切换和消息收发

重点：

- 如果多窗体打开，同步切换to对象

页面端：WAP页面

早期嵌入客户端：android，iphone

只提供消息收发功能

重点：

- 某些android不支持websocket，得降级；
- 适应键盘弹出和收起；
- 各手机的form的submit不一致兼容，兼容键盘回车提交和点击发送按钮。

服务端

- nodejs，socket.io，redis，nginx

- imserver，socket.io，站内使用。
  为什么用socket.io？快，页面需要兼容（websocket，polling）。

- huaserver、pushserver，ws，美丽说主app、花花（android、iphone、windows）使用。
  为什么用ws？不需要兼容直接websocket，改动小，稳定。

- 为什么用redis？数据库和pub/sub，线上集群，后端熟悉运维成熟。

- huaserver、pushserver为什么分开部署？压力不同，连接数不同。

浏览器 和 服务器

socket.io封装event。

例子：建立连接

浏览器引入socket.io.js，全局有io。

```
var socket = io.connect(Meilishuo.config.im_url);
```

浏览器建立和服务器的连接，connection，得到socket

```
socket.on('connect', function (data) {
    console.log('连接成功')
})
```

```
socket.emit('changeUser', data)
```

浏览器发送changeUser事件

```
socket.on('changeUserPush', function(data){
```

浏览器监听changeUserPush事件，进行相关操作

```
1 var Server = require('socket.io')
var io = new Server(ws_config)
```

服务端引入socket.io，new io

```
io.sockets.on('connection', function(socket){
```

监听connection事件，得到socket

确认浏览器和服务器连接成功，发送connect事件（socket.io封装）

```
socket.on('changeUser', function(data){
```

服务端监听changeUser事件，进行相关操作

```
socket.emit('changeUserPush',data)
```

服务端返回changeUserPush事件

websocket

```
{
    "pingInterval":25000
    ,"pingTimeout":10000
}
```

ping间隔：维持通信
超时时间：断网情况下重要

```
var t = setInterval(function(){
    if (1 == socket.readyState)
        socket.ping()
}, opts.pingInterval)
```

base/io.js

socket.io 和 base/io.js 都对原本的h5 websocket进行了封装，基本用法都是emit，on。

关于redis

例子：同步切换用户

引入redis模块，符合站内im的封装。base/redis.js

```
redis = require('../base/redis')
```

各环境的redis config集合

```
➜  config  vi redis_
redis_devlab3.json      redis_newlab.json       redis_push_online.json
redis_local.json        redis_online.json       redis_test.json
```

```
[
    {"host":"172.16.8.79", "port":"6479"}
    ,{"host":"172.16.8.81", "port":"6579"}
    ,{"host":"172.16.4.69", "port":"6379"}
    ,{"host":"172.16.8.109", "port":"6379"}
]
```

根据：“标记 + 用户来源 + 用户id” 的组合生成key，再根据config的数量，得到redis编号

```
exports.getNo = function(key,num){
    return (hexdec((md5(key)).substr(0, 2)) % num)
}
```

config和算法同后端（php）一致。

nodejs + redis

例子：同步切换用户

引入redis模块，符合站内im的封装。base/redis.js

```
redis = require('../base/redis')
```

每个用户一成功连入server，就生成一个chanel。

```
channel = base.getKey('ImMsgPub', userData)
```

```
var getKey = function(key, userData){
    return key + ':' + (userData.source||'') + userData.fromid
}
```

建立了一个redis Sub

```
msgSub = new redis.Subsocket(channel, function(data){
```

需要同步消息，就pub这个chanel。

```
redis.pub(channel, {
    'type' : 'syncChangeUser'
    , 'uid': userData.toid
    , 'socketid': userData.socketid
    , 'changeUserData' : data
})
```

监听了这个chanel的server就会sub到sync信息。

```
else if(data.type == 'syncChangeUser'){
    if(userData.socketid != data.socketid){
        userData.toid = data.uid;
        socket.emit('syncChangeUserPush', {
            'toid':data.uid
            , 'socketid' : userData.socketid
        })
    }
}
```

整体

浏览器 + nodejs + redis

其它

1. 浏览器

- io状态监听（上下线，断线重连）：app/im/init.js，fml.vars.im.reconnect
- 全局快捷键：app/im/key.js
- 用户相关：userList.js，userStatus.js，userSearch.js等
- 触发打开im：page/im/open.js
- huaserver的io：app/im/io.js，对websocket的封装，用法和socket.io相同

2. 服务器

- huaserver的io：base/io.js，用法和socket.io相同，没有向polling的兼容

http://redmine.meilishuo.com/projects/im_/wiki/Imserver%E6%96%87%E6%A1%A3

学习资源

- socket.io： https://github.com/Automattic/socket.io
- engine.io#methods-1： https://github.com/Automattic/engine.io#methods-1
- node_redis： https://github.com/mranney/node_redis
- ws： https://github.com/einaros/ws
- emit： http://nodejs.org/docs/latest/api/util.html#util_util_inherits_constructor_superconstructor
- redis： http://redis.readthedocs.org/en/latest/index.html

Thanx

Q&A