# 消息服务

消息服务是本软件的基础。是第一期主推的内容，如果消息和地图服务完善与否将直接影响用户体验，从而影响用户的口碑，以及软件的推广率。

## 技术与选型：

### 开发语言

我对erlang处理这种消息的转发和对多连接的处理信心十足，除了erlang之外，使用netty的websocket是否也能达到类似的效果却不清楚，两者相比较，netty的稳定性和高效型是毋庸置疑的，并且websocket的长连接模式是消息投递的时效性的保障。erlang语言本身的天然优势是处理以下问题

1：处理大量的socket连接

2：socket创建和销毁非常频繁

3：天生支持消息的定点投递

4：支持热部署和升级

5：非常容易的扩容和集群

6：强大的二进制处理能力

7：内置高效的ets缓存

### 协议的选择

协议可以选择

1：文本协议

2：xmpp协议

3：二进制协议。

在上述协议的选择中最好的，最适合当选择xmpp协议，此协议有对各类消息的处理都有定义，有第三方的软件包支持，有成功的项目案例（ejabberd）。但是此协议复杂，若想应用周期较长。文本协议首先选择以json为载体。选择json很大程度上是因为协议简单。无论在客户端还是服务器端都很好处理。数据处理不存在技术问题。（二进制数据

可以Base64以后进行传输，但是可能会使数据变大）。所有协议均需要自定义。通用性不是太好。二进制协议处理后台应用可能会好些

结论：使用erlang作为消息转发推送模块的服务器 。采用Json协议传递消息。

# 语音处理

用户发送的消息是否需要压缩处理传输？

# 对外接口

作为客户端和服务器之间服务性接口需要具备以下功能

## 用户注册

客户端上操作远远不如PC上，而且现在快节奏的生活。让人们都很浮躁。注册仅需提供邮件地址 和 密码即可，用户进入客户端以后可以通过设置修改自己的相关设置。

|  |  |  |
| --- | --- | --- |
| 用户名 | email | String |
| 密码 | pass | String |

## 用户登陆

同用户注册

|  |  |  |
| --- | --- | --- |
| 用户名 | email | String |
| 密码 | pass | String |

## 信息更新

个人信息以及字段

|  |  |  |
| --- | --- | --- |
| 昵称 | nickname | String |
| 性别 | gender | Int |
| 生日 | birthday | Date |
| 姓名 | Realname | String |
| 备注 | comment | String |
| 个人动态 | trends | String |
| 常走的路线 | haunt | String |
| 注册日期 | Regdate | Date |
| 兴趣爱好 | Hobbit | String |
| 常备工具 | Tools | String |
| 座驾以及型号 | brand | String (Grant-ATX770D) |
| 手机号码 | phone | String |
| qq | qq | String |
| sina微博 | weibo | String |

## 数据上报

用户上报是客户端和服务器交互最直接也最有用的一个功能，因为客户端的情况直接会影响服务器的行为。

|  |  |  |
| --- | --- | --- |
| 客户端名称 | Client\_name | String |
| 客户端版本号 | Client\_version | double |
| 操作系统 | Os\_version | String |
| 分辨率 | Resolution | String |
| 手机号码 | Phone | String |
| 用户名 | Email | String |
| 上报时间 | Update\_time | date |
| 设备标识 | Device\_token | String |
| 推送类型 | Push\_type | int |

## 软件更新

软件更新是对于客户端软件来说最重要的。优先级最高。

|  |  |  |
| --- | --- | --- |
| 用户名 | Email | String |
| 客户端名称 | Client\_name | String |
| 客户端版本号 | Client\_version | double |

## 意见与建议

|  |  |  |
| --- | --- | --- |
| 用户名 | Email | String |
| 意见与建议的类型 | Suggest\_type | Int |
| 建议内容 | Suggest\_content | String |

## 精品线路推荐

# 位置服务

选一个支持android 的位置服务，当前有高德地图和百度地图 ，google 地图，据我所知，高德地图封装的太狠，可塑性不高，我对高德地图感觉不好，我手机里面装了高德地图和百度地图，但是我还是喜欢用百度地图。百度提供官方的API

http://developer.baidu.com/map/geosdk-android.htm

# ERLANG - IPPUSH (方案1)



如上图所示，这个是我的第一个结构设计。

## HTTP接口服务

对客户端提供服务，主要是有用户信息设置，用户登录，用户注销，软件信息上报，软件更新，意见与建议，精品线路推荐等等，这些都是HTTP接口需要提供的支持，需要客户端上行参数，服务器返回JSON格式的数据。类似于

示例

客户端请求：

POST /b/util/feedback HTTP/1.1

HOST: 218.200.243.234

Accept: \*/\*

Authorization: Basic dXNlcjpwYXNzd29yZA==

Content-Type: application/x-www-form-urlencoded

ver=1.0&text=xxxxxxxxxxxxxxxx

返回响应：

HTTP/1.1 200 OK

Content-Type: text/x-json

Content-Length: 50

{

“result”: 0,

“msg”: ”操作成功”

}

## 消息服务

消息服务主要是处理骑乐汇中即时消息收发。主要用于消息传递。其中有

### 广播消息

广播消息是同时发给在线的很多人的。例如一个车队，车队是很多骑友的集合，我们可以理解为一个组，当车队队长发送公告时，无论车队成员是否在线，均应该能收到公告消息，公告消息就是个广播消息。广播消息需要有个 发件人和收件人，是一个用户列表。如何读取用户列表，需要业务系统去处理。流程如下：



消息格式类似于

{"msg\_id": "server-client-20130226191055503",

"from": "moumou",

"to": "amian,xiaodu",

"subject": "msg",

"type": "public",

"content":{"style":"text",

"length":"11",

"body":"hello world"}

"creationDate": "20130226191055503"

}.

特别需要注意的是：**消息处理模块仅负责消息投递，不负责用户鉴权等功能。**

### 用户的点对点消息

点对点消息与广播消息相同，不同的是收件人为单个用户。

## 结论

如果以上述架构设计服务器端的优缺点如下：

### 优点

消息处理模块结构简单，对于我这个erlang 新手来说。的确是个好消息。协议处理简单，仅仅需要在消息处理模块中，维护客户端连接以及用户名与进程PID对应的关系即可。Erlang 的天生支持的高并发，完全支持我们现在的业务，而且很容易集群。

消息处理模块需要维护2类的连接，一类是数以万计的客户端连接，以及为数不多的业务服务模块的连接。而且协议非常简单，易行。

### 缺点

感觉有些别扭，客户端除了在后台启动一个服务维持与消息服务器的连接之外，还需要定期或者不定期的发起HTTP连接。为什么需要2次连接？本来维护一个长连接就挺麻烦的事儿了。为啥还得有HTTP连接呢？这样这个通道的使用率又太低了。这个问题是这样的，在我的设计中HTTP业务处理接口是处理非即时性的功能，而消息服务，则是处理即时消息的。可是这样增加系统的复杂性以及资源的浪费，会使得客户端很费电。

# NETTY - WEBSOCKET (方案2)



## WEBSOCKET服务模块

WebSocket protocol 是HTML5一种新的协议(protocol)。它是实现了浏览器与服务器全双工通信(full-duplex)。为了实现即时通讯(real-time)，所用的技术都是轮询(polling)。轮询是在特定的时间间隔(time interval)（如每1秒）在 WebSocket API，浏览器和服务器只需要要做一个握手的动作，然后，浏览器和服务器之间就形成了一条快速通道。两者之间就直接可以数据互相传送。在此WebSocket 协议中，为我们实现即时服务带来了两大好处：

1. Header

互相沟通的Header是很小的-大概只有 2 Bytes

2. Server Push

服务器可以主动传送数据给客户端

在实现websocket连线过程中，需要通过浏览器发出websocket连线请求，然后服务器发出回应，这个过程通常称为“握手” (handshaking)。

**浏览器请求**

GET /demo HTTP/1.1

Host: 你的网址.com

Connection: Upgrade

Sec-WebSocket-Key2: 12998 5 Y3 1 .P00

Sec-WebSocket-Protocol: sample

Upgrade: WebSocket

Sec-WebSocket-Key1: 4 @1 46546xW%0l 1 5

Origin: http://你的网址.com

^n:ds[4U

**服务器回应**

HTTP/1.1 101

WebSocket Protocol Handshake

Upgrade: WebSocket

Connection: Upgrade

Sec-WebSocket-Origin: http://你的网址.com

Sec-WebSocket-Location: ws://你的网址.com/demo

Sec-WebSocket-Protocol: sample

8jKS’y:G\*Co,Wxa-

### Android websocket

https://github.com/anismiles/websocket-android-phonegap

为什么选择websocket？这个问题要从我自身出发来说了。毕竟我是搞JAVA 的，erlang 对我来说。我自己非常喜欢这个函数语言，比起scala 我更喜欢erlang 的语法不那么怪异，消息传递机制又是很接近现实，而热替换代码，天生支持集群。让我心动不已，不过当前并不是以自己好恶来做决定的时候，我想我应该先把原型做出来。

Websocket 可以说是一门新的技术，HTML5是最近很流行的，而且websocket 是在HTTP基础上的，该技术很成熟，而且WEBSOCKET又支持长连接，协议有专门人制定，在持续的发展，而且NETTY恰好有这个协议的实现，我对NETTY 很熟悉。已经用NETTY 成功做了2个项目 了。而且成功的再线上运行。基本上我对NETTY 的运行机制都不陌生，甚至公司内的AOI项目也是用netty 实现的，经过测试单机400多W 并发连接数最高(528W)，不过AOI走的是移动的协议，也是基于IPPUSH 的。所以，对我来说在一个可以预见的未来，netty 确实比erlang 的不确定性少很多。更容易出货。

不夸张的说，使用netty 我可以再一周之内完成消息转发模块。而用erlang 我可就不敢说了。 而且使用netty ，工作全变成JAVA 的了。有很多优秀的，并且我熟悉的框架可以借鉴并使用，使得消息处理模块不那么独立，消息处理模块可以无障碍的与MEMECACHED 交互(有现成的代码和库)。什么spring ，maven ，等等都可以使用。更加得心应手。所以我打算使用websocket 作为我们第一版消息转发模块的实现。

## 消息与处理

客户端与服务器的交互分为同步与异步两类，其中同步交互可以人为，需要服务器立即响应的请求，比如：用户登陆，修改设置，用户上报。等等这类消息需要客户端根据不同的返回结果进行不同的处理。异步处理主要是处理点对点的对话消息，广播消息，此类消息的返回结果不会影响客户端的处理流程。

### 消息处理流程



### 消息格式

{"msg\_id": "server-client-20130226191055503",

"from": "moumou@leadtone.com",

"to": "server|OTHER",

"subject": "login",

"content":{"style":"text",

"length":"11",

"body":"hello world"}

"creationDate": "20130226191055503"

"status":"0"

}.

上述消息中蓝色的部分由 客户端提供的 service 处理，客户端service 提供2个功能

1：保持客户端与服务器的连接

客户端在后台启动这个service，定期向服务端发送PING的请求用于保持客户端与服务器之间的连接。

2：负责客户端和服务器端的协议封装和协议分配

Service 收到消息之后会立即转化消息，转化成 RiderMessage 的对象，并且将CONTENT 中的信息交给不同的 Activity。



### 同步消息处理

### 异步消息处理

## 协议,业务等处理模块

协议处理模块，其实主要是处理JSON的数据格式。有请求，响应，等等。其实主要是处理响应。个人认为可以作为打成一个单独的JAR 这样，服务器的消息处理模块与协议处理模块完全分的开。

可以这么理解，整个服务器有很多模块，rider-server，rider-protocol，rider-business，rider-sns ，rider-sms，rider-cache，等等。

# 通用模块

## 客户端

## 缓存服务

首先看一个缓存的设计图



其次是流程图



## Lbs服务

Lbs 服务当前能有2个选择，一个是高德地图另外一个是百度，这2个LBS服务均提供客户端API可以实现定位，路线搜索等功能。

### 高德地图

1：最早的地图供应商

2：有专门的测绘人员

3：客户端定位更为省电(有人做过测试)

4：POI信息使用国标

5：阿里注资前景不差

6：小葛部门正在研究，有人趟路

### 百度地图

1：比较广泛的LBS应用，有很好的群众基础

2：API使用更简单

3：客户端定位比较费电

4：POI信息使用厂标(百度自己的标准)

5：百度应用丰富

6：橘子在百度的LBS部门，看是否能获取最新的资料

# 数据结构设计

# 数据存储

数据存储这块脑子比较混乱主要是通过如下图例分别描述一下数据在整个系统中如何存储，分别存储的是什么数据

需要检索的缓存数据采用 mongodb ，主要用于存储 如下数据

1：实时的地图数据

2：用户的离线消息数据

# 日志与历史数据

# 车队

# 活动

# 求救系统

# 精品线路推荐

# 协议

{

"msg\_id":"1vafd",

"from":"999991",

"to":"AAAAAA",

"subject":"reg" ,

"content":{"nick":"lvqi","email":"lvqi","password":"1234"},

"creationDate":"20130227121212"

}

# 其他