Atitit 客服线im计划技术规划

目录

[1. 基本目标 客服线 2](#_Toc3996)

[1.1. GUI客户端 web +pwa+ 跨平台移动hybrid app 2](#_Toc30598)

[1.2. 预计使用人员数量 10w+ 2](#_Toc179)

[1.3. 服务端跨平台部署（ 2](#_Toc24413)

[1.4. 简单易得 快速部署 2](#_Toc24061)

[2. 大略功能 2](#_Toc14001)

[2.1. 二阶段特性 2](#_Toc391)

[3. 技术规化 3](#_Toc27169)

[3.1. 常见开发语言 3](#_Toc15663)

[3.2. 交互 websocket reset ajax 3](#_Toc9445)

[3.3. 数据存储 3](#_Toc29910)

[3.4. 开发框架 rest框架/socket框架 4](#_Toc12983)

[3.5. 该选择什么样的网络通讯技术？ 4](#_Toc26407)

[4. 标准消息规范与模式 5](#_Toc23052)

[4.1. 自定义规范消息格式 5](#_Toc368)

[4.2. XMPP、MQTT gcm 5](#_Toc14589)

[4.3. 协议如何制定？ 5](#_Toc16222)

[4.3.1. 1XMPP 5](#_Toc17040)

[4.3.2. 3MQTT 5](#_Toc23976)

[4.4. 4私有协议 6](#_Toc10056)

[4.4.1. 5结论 6](#_Toc7904)

[5. --------------需求之FURPS模型 6](#_Toc2846)

[6. 技术类功能 6](#_Toc7175)

[6.1. 消息转发事件驱动websocket 6](#_Toc15438)

[6.2. 定时可用 db定时以及os定时机制，加强稳定性 6](#_Toc9887)

[7. 可扩展性 结构简化可维护性 7](#_Toc31146)

[7.1. 简单设计 7](#_Toc12354)

[7.2. 业务逻辑设计简化复杂分支 7](#_Toc3140)

[7.3. 动态数据结构 与扩展性 7](#_Toc3086)

[7.4. 适当的层次结构 7](#_Toc22451)

[8. 稳定性 7](#_Toc16490)

[8.1. 进程保活 cron保活 7](#_Toc23875)

[8.2. 分布式 非集中模式 分平台分片部署 7](#_Toc5099)

[8.3. 基于现有db和os机制更佳可靠性 7](#_Toc20936)

[8.4. 资源自动释放gc vs 永驻内存 7](#_Toc5731)

[8.5. 全面边界检测与异常管理 7](#_Toc8944)

[8.6. 对数据查询限制最大单次条数 7](#_Toc12124)

[8.7. 心跳保活 8](#_Toc28747)

[8.8. 文件中转模式 8](#_Toc17815)

[8.9. 服务器中转方式 8](#_Toc23900)

[9. 未来性能方面 9](#_Toc28478)

[9.1. 负载均衡 nginx （dubbo》》springcloud） 9](#_Toc3445)

[9.2. 分片区存储 分区 分库分表 9](#_Toc22763)

[10. 安全性 9](#_Toc24361)

[11. Ref 9](#_Toc16826)

# 基本目标 客服线

## GUI客户端 web +pwa+ 跨平台移动hybrid app

## 预计使用人员数量 10w+

满足中小型组织使用

人员较少 暂无需特别性能提升

## 服务端跨平台部署（

## 简单易得 快速部署

满足中小型组织使用

# 大略功能

显示客服列表

发送消息 接受消息

在线消息与离线消息

留言消息功能

消息查询（关键词查询法，日期查询 法，

高级查询（消息类型【发送/接受】，消息接收人，日期，日期区段，消息内容 返回条数等）

具备了语音、文本、表情、传图等基础IM业务功能

截屏 文件传输

## 二阶段特性

消息阅后即焚功能，支持数据自定义加解密存储。

导出 导入 备份

消息撤回 删除

音视频支持 **实时音视频 语音留言**

# 技术规化

## 常见开发语言

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 开发效率 | 简单性 | 跨平台 | 需要编译 | 需要部署 | 已知问题 |
| Sql | 高 | 简单 | Y | N | N | 非全能DSL  调试支持不足，for循环语法啰嗦，异常处理啰嗦 |
| php | 中 | 中等 | Y | N | Y | 调试支持没有内建 |
| java | 低 | 繁琐 | Y | Y | Y | 语法啰嗦繁琐 |
| Node.js | 中 | 中等 | Y | N | Y | 单线程影响稳定性 |
| go语言 | ？？ | ？？ | N | Y  区分win linux平台 | Y | 异常处理繁琐，影响稳定性.需要区分平台编译 |
| python | 中 | 中等 | y | N | y | Web使用不多 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

## 交互 websocket reset ajax

业务普通走rest通道，消息及时性要求高的走websocket通道

## 数据存储

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Sql支持dsl支持 | 复杂业务支持 |
| mysql |  | 支持 | 高 |
| mongodb |  | Nosql 但有dsl | 中 |
| es |  | Dsl功能较薄弱 | 低 |
| redis |  | 啥都没 | 最低 |
|  |  |  |  |
|  |  |  |  |

## 开发框架 rest框架/socket框架

前端直接支持websocket api，，后端使用各自语言的websocket模块类库

Rest api前端可使用featch api，后端使用各自语言的rest模块或类库

## 该选择什么样的网络通讯技术？

****IM主流网络通讯技术有两种:****

* 基于TCP的长连接；
* 基于HTTP短连接PULL的方式。

基于TCP长连接则能够更好地支持大批量用户，问题是客户端和服务器的实现比较复杂。当然也还有一些变种，如下行使用MQTT进行服务器通知/消息的下发，上行使用HTTP短连接进行指令和消息的上传。这种方式能够保证下行消息/指令的及时性，但是在弱网络下上行慢的问题还是比较严重。早期的来往就是基于这种方式。

短连接PULL常见于WEB IM系统(当然现在很多WEB IM都是基于WebSocket实现)，它的优点是实现简单，方便开发上手， ，比较适合小型的IM系统

# 标准消息规范与模式

## 自定义规范消息格式

## **XMPP、MQTT gcm**

简介：基于XML协议的通讯协议，前身是Jabber，目前已由IETF国际标准化组织完成了标准化工作。  
优点：协议成熟、强大、可扩展性强、目前主要应用于许多聊天系统中，且已有开源的Java版的开发实例androidpn。  
缺点：协议较复杂、冗余（基于XML）、费流量、费电，部署硬件成本高。

****使用HTTP轮循方式****

简介：定时向HTTP服务端接口（Web Service API）获取最新消息。  
优点：实现简单、可控性强，部署硬件成本低。  
缺点：实时性差

## 协议如何制定？

IM协议选择原则一般是：易于拓展，方便覆盖各种业务逻辑，同时又比较节约流量。后一点的需求在移动端IM上尤其重要。常见的协议有：XMPP、SIP、MQTT、私有协议。（更多关于即时通讯应用的协议选择，请参见《如何选择即时通讯应用的数据传输格式》：[http://www.52im.net/thread-276-1-1.html](http://www.52im.net/thread-276-1-1.html" \t "http://www.52im.net/_blank)）

### 1XMPP

****优点：****协议开源，可拓展性强，在各个端(包括服务器)有各种语言的实现，开发者接入方便；  
****缺点：****缺点也是不少，XML表现力弱、有太多冗余信息、流量大，实际使用时有大量天坑。

2SIP

SIP协议多用于VOIP相关的模块，是一种文本协议，由于我并没有实际用过，所以不做评论，但从它是文本协议这一点几乎可以断定它的流量不会小。

### 3MQTT

****优点：****协议简单，流量少；  
****缺点：****它并不是一个专门为IM设计的协议，多使用于推送。

## 4私有协议

市面上几乎所有主流IM APP都是是使用私有协议，一个被良好设计的私有协议优点非常明显。  
  
****优点：****高效，节约流量(一般使用二进制协议)，安全性高，难以破解；  
****缺点：****在开发初期没有现有样列可以参考，对于设计者的要求比较高。

### 5结论

一个好的协议需要满足如下条件:高效，简洁，可读性好，节约流量，易于拓展，同时又能够匹配当前团队的技术堆栈。基于如上原则，我们可以得出: 如果团队小，团队技术在IM上积累不够可以考虑使用XMPP或者MQTT+HTTP短连接的实现。反之可以考虑自己设计和实现私有协议。

# --------------需求之FURPS模型

**FURPS**是功能（function）、易用性（usability）、可靠度（reliability）、性能（performance）及可支援性（supportability）五个词英文字首的缩写，是一种识别软体品质属性的模型。

# 技术类功能

## 消息转发事件驱动websocket

## 定时可用 db定时以及os定时机制，加强稳定性

# 可扩展性 结构简化可维护性

## 简单设计

## 业务逻辑设计简化复杂分支

## 动态数据结构 与扩展性

## 适当的层次结构

# 稳定性

## 进程保活 cron保活

**保活处理**

## 分布式 非集中模式 分平台分片部署

## 基于现有db和os机制更佳可靠性

## 资源自动释放gc vs 永驻内存

## 全面边界检测与异常管理

## 对数据查询限制最大单次条数

防止无响应。。熔断拒绝机制

## 心跳保活

心跳保活机制的实现方案参考

从上面我们可以得出结论，目前而言，应用层心跳的确是检测连接有效性，双方是否存活的最佳实践，那么剩下的问题就是怎么实现。  
  
最简单粗暴做法当然是定时心跳，如每隔 30 秒心跳一次，15 秒内没有收到心跳回包则认为当前连接已失效，断开连接并进行重连。这种做法最直接，实现也简单。唯一的问题是比较耗电和耗流量。以一个协议包 5 个字节计算，一天收发 2880 个心跳包，一个月就是 5 \* 2 \* 2880 \* 30 = 0.8 M 的流量，如果手机上多装几个 IM 软件，每个月光心跳就好几兆流量没了，更不用说频繁的心跳带来的电量损耗。  
  
既然频繁心跳会带来耗电和耗流量的弊端，改进的方向自然是减少心跳频率，但也不能过于影响连接检测的实时性。基于这个需求，一般可以将心跳间隔根据程序状态进行调整，当程序在后台时(这里主要考虑安卓)，尽量拉长心跳间隔，5 分钟、甚至 10 分钟都可以。  
  
而当 App 在前台时则按照原来规则操作。连接可靠性的判断也可以放宽，避免一次心跳超时就认为连接无效的情况，使用错误积累，只在心跳超时 n 次后才判定当前连接不可用。当然还有一些小 trick 比如从收到的最后一个指令包进行心跳包周期计时而不是固定时间，这样也能够一定程度减少心跳次数。

## 文件中转模式

移动网络因为网络不稳定的客观因素存在，不适合实时发送较大的2进制文件（像电脑上的实时文件发送的那种），因为这涉及3方：客户端A、服务端、客户端B，任何两方的通讯因网络的不稳定而导致的重传等，都是个很不好处理的事情。  
  
现在多数情况下都是通过http先上传到中转文件服务器，成功后再通知接收方，这种情况下，因上传的过程只限于客户端A和服务端，只涉及两方，网络的不稳定，也只影响了发送者（而不涉及接收者，因为对方还不知道你正在发送文件呢），所以无论是从可靠性、复杂性，还是用户体验的处理上，这种方式都要简单的多。而这么多年的移动应用也证明，这种云中转暂存的方式是比较适合于当前的移动网络和移动应用体验的

## 服务器中转方式

几乎所有互联网IM产品都采用服务器中转这种方式进行消息传输，相对于P2P的方式，它有如下的优点:

* 能够支持更多P2P无法支持或支持不好的业务，如离线消息，群组，聊天室服务；
* 方便业务逻辑的拓展和新旧版本的兼容。

当然它也有自己的问题:服务器架构复杂，并发要求高。

# 未来性能方面

## 负载均衡 nginx （dubbo》》springcloud）

## 分片区存储 分区 分库分表

分区 分片机制即可。。。

可以按照年份稳定扩容

存储与消息中心都可以按照片区运行

# 安全性

防止信息泄露

远程命令安全性

性能安全

数据安全之片区隔离机制

# Ref

移动端IM开发需要面对的技术问题-IM开发/专项技术区 - 即时通讯开发者社区!