X-Messenger Servers
概要设计说明书

昆山轩辕软件技术有限公司
研发中心

Revision History

Revision	Date release	Authors	Description
1.0	2010-03-12	Tiger Cheng	Initial version.
2.0	2012-03-25	Tiger Cheng	Re-designed.

目录

内容目录

1. 简介

1.1. 概述

X-Messenger 是 昆山轩辕软件技术有限 公司 推出的即时通信（Instant Messenger）工具，提供文字聊天、文件传输、好友管理等即时通信功能。

本文档将详细描述 X-Messenger Server 各项功能、系统架构、技术架构与流程设计等。

1.2. 系统设计原则

系统（Server）模块设计原则：

1. 高性能：高吞吐量、低延迟，大容量系统不能存在瓶颈；
2. 可伸缩性：随着用户数的增加，可以通过叠加模块扩容；
3. 模块化设计：模块高内聚，模块间低耦合；
4. 模块可重用性强，提高效率；
5. 注重系统的健壮性：设计时首先考虑健壮性，其次是运行效率。

编码原则：

1. 良好的注释风格；
2. 统一命名规范；
3. 详细的操作日志，日志要分等级，可分为 INFO，DEBUG，WARNING，ERROR 等级别，调试过程中，调试日志要详细，运行过程中可关闭调试日志，对错误日志的出错要定位清楚。

1.3. 定义

1. **Instant Messenger**：即时通信
2. **XMMEP**：X-Messenger Message Exchange Protocol（X-Messenger 信息交流协议）

1.4. 参考文献

1. X-Messenger - XMMEP Specification
2. X-Messenger Servers Database Design Specification

2. X-Messenger 系统描述

2.1. X-Messenger 系统网络拓扑

X-Messenger 系统由以下几个部分组成：

注册服务器(Register Servers)、邮件服务器(Mail Servers)、消息路由服务器(Router Servers)、状态服务器(Presence Servers)、管理客户端(Management Client)、Messenger 客户端(X-Messenger Client)。

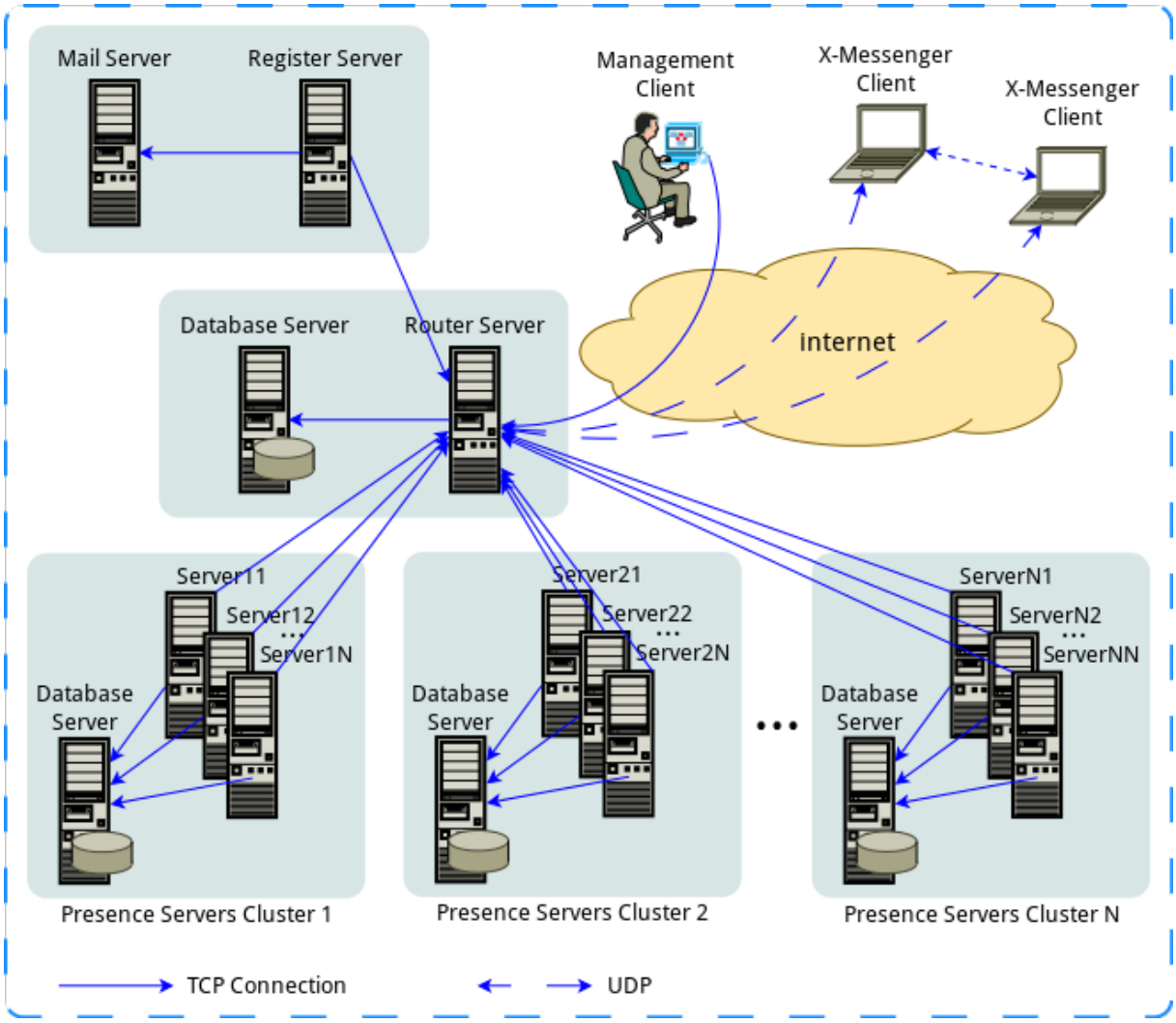

插图 2.1.1: X-Messenger 网络拓扑

2.2. X-Messenger 子系统介绍

2.2.1. Register Server

注册服务器，职责：

1. 负责用户的注册，完成注册数据的生成；
2. 提供用户注册信息修改（如修改登录密码、个性签名等）、更新模块。

2.2.2. Mail Server

邮件服务器，可安装广泛使用的开源 MTA (Mail Transfer Agent)，如 Sendmail、Postfix 等都是功能强大、且安全性、稳定性都久经考验的 MTA。X-Messenger 中只需使用其 SMTP 功能，负责用户激活邮件的发送。

2.2.3. Router Server

消息路由服务器，职责：

1. 维护 Presence Server 的 IP 和 Port；
2. 记录 Presence Server 中的用户名，供客户端以及 Presence Server 查询某个用户位于哪个 Presence Server 中；
3. 结合数据库服务器，持久化上述数据；
4. 在 Presence Server 之间转发消息；

2.2.4. Presence Server

状态服务器，或称 用户信息服务器，职责：主要业务逻辑均在 Presence Server 中实现、主要数据在 Presence Server 中管理：

1. 在内存中维护用户相关的信息，如：用户的注册信息；用户的好友信息：好友分组、好友列表、聊天室相关信息；
2. 结合数据库服务器，持久化上述数据；
3. 离线消息的暂存(文件系统中)、发送；
4. 为客户端之间转发各类消息，如：添加好友通知、发送文件请求等。

2.2.5. Database Server

数据库服务器，目前版本使用关系型数据库(RMDBS)，如 Oracle、MySQL 等关系型数据库服务器产品，负责各个子系统中必要的数据持久化，以及少量数据的实时查询。

2.2.6. Management Client

管理客户端，负责监控 (实时但非连续) X-Messenger 所有服务器的状态、与所有服务器交互，如实时改变某台服务器日志级别、实时查看服务器输出的日志、改变服务进程配置项并使其生效、启动/关

闭/重启某台服务器或其服务进程等任务。

2.2.7. X-Messenger Client

X-Messenger 客户端，职责 (略)。

2.3. 子系统之间的交互方式

以下详细列举 X-Messenger 所有子系统之间的关联和消息交互方式：

1. X-Messenger 客户端之间：
   采取 UDP 作为传输层协议，可以实现 P2P 聊天、文件传输等 IM 主要的功能，最大程度上减小服务器的负载。
2. X-Messenger 客户端和服务器端之间：
   采取 UDP 作为传输层协议；
   X-Messenger 客户端只与 Router Server、某个特定的 Presence Server 交互。
3. Router Server 和 Register Server 之间：
   采取 TCP 作为传输层协议；
   Register Server 和 Router Server 之间使用 TCP 长连接，连接需要通过认证，Register Server 为 Client 角色，主动发起连接。
4. Router Server 和所有 Presence Servers 之间：
   采取 TCP 作为传输层协议；
   Router Server 和 Presence Server 之间需保持 TCP 长连接，连接需要通过认证，Presence Server 作为 Client 角色，主动发起连接，并定期发送心跳检测消息。
5. Router Server、Presence Servers 和数据库服务器之间：
   采取 TCP 作为传输层协议；
   保持 TCP 长连接，连接需要通过认证，Router Server、Presence Server 作为 Client 角色，主动发起连接。
6. Management client 和 Router Server 之间：
   采取 TCP 作为传输层协议；
   保持 TCP 长连接，Management client 作为 Client 角色，主动发起连接，并定期发送心跳检测消息。
   Management client 通过 Router Server 获取 Presence Server 列表；
7. Management client 和各 Presence Server 之间：
   采取 TCP 作为传输层协议；
   保持 TCP 长连接，连接需要通过认证，Management client 作为 Client 角色，主动发起连接，并定期发送心跳检测消息。
8. （当前版本中）Presence Server 之间无交互。

3. 服务器端软件的实现要点

X-Messenger 系统建设的目标是：电信级 IM 系统。所以需要保证符合：不间断运行、大容量、高稳定性、可靠性 这几个基本的要求。最终实现的系统必需能够经受以下考验：

1. 承载大量数据的存储/访问；
2. 保障高吞吐量、高并发、低延迟的网络 I/O；
3. 系统的容灾和容错能力。

除上述目标之外，软件架构、设计的合理性以及软件实现的质量也将纳入重点考虑的范围。

3.1. 高负载网络 I/O

客户端与服务器端会有频繁的消息交互，在大用户量的情况下需要考虑如何避免网络 I/O 成为系统瓶颈。可参考的解决方案：

1. 优化业务逻辑、设计合理、高效的应用层通信协议，避免冗余的消息往来和冗余的消息字段；
2. 客户端和服务器端之间的交互使用 UDP 作为传输协议；
3. 服务器端结合硬件能力合理使用线程；
4. 服务器端采用高效的 I/O 模型，如反应式 (Reactor) I/O 多路复用和异步 I/O 等；
5. 把系统的负载分布到各个 Presence Server 上，每个 Presence Server 负责处理一部分用户，维护客户端的状态和好友列表。

3.2. 大规模数据的存储

数据的存储涉及两个方面：1，数据库作为持久化数据的途径；2，为了提高服务器的响应速度，服务器进程需在内存中缓存必要的数据，这些数据需要频繁的被修改和访问。这两处的数据大部分是交集。

一方面，在一个 Presence Server 进程中，可用内存的大小直接决定该 Presence Server 能管理的用户量，所以必须考虑哪些种类的数据需要在内存中缓存，其中有几个注意点：

1. 不常用的数据不缓存；
2. Presence Server 可采取 Lazy load 的方式装载用户信息，比如只在用户登录的时候从数据库装载该用户相关的数据；
3. Presence Server 进程定期清理不活跃用户信息，达到减少有效内存占用的目的。

另外一个方面，数据库服务器中数据库系统为了配合 Presecne Servers Cluster 的布局，需为 Cluster 内的每个 Presence Server 单独创建一个数据库实例，比如 Presence Servers Cluster 1 中有 Presence Server 1 ... Presence Server 20，则数据库实例为 PresenceDB_C1P1 ... PresenceDB_C1P20，后缀 C1P1 代表该数据库实例对应的是 Cluster(C) 1 中的 Presence Server (P) 1，以此类推。

目前版本的系统采用的方式是：Cluster 内通过逐步增加 Presence Server 的数量、当 Cluster 内的 Presence Server 达到规定的数量后，再通过逐步增加 Presence Servers Cluster 的数量来逐步扩大整个系统的容量。

（上述方案的问题是数据的分割，服务器之间彼此无关联，造成信息孤岛。）

3.3. 系统的容灾和容错措施

我们必须认识到，软、硬件的故障并非异常，而是属于常态。所以保证不间断服务、提供可靠服务需要我们提供有效的系统容灾和容错的措施。

当前版本的系统采取的策略是：

1. Presence Server 机器采取 N+1 冗余的方式：在每个 Presence Servers Cluster 中，多出一台机器作为备用机，一旦某台 Presence Server 机器故障或软件出问题导致无法正常提供服务，即启用备用机。
2. Router Server、Register Server、Mail Server 均采取 Master-Slave 形式的双机热备方式：即上述服务器各自都有两台主机，一台为主服务器，一台为从服务器，两台同时运行，但只有主服务器对外提供服务，且实时（或定时）将数据同步到从服务器。一旦主服务器故障而不能提供服务时，即安排从服务器接管，对外提供服务。
3. 针对上述 1，2 锁描述的方式，在软件实现上：
   A，每个 Register Server 和 Presence Server 均保持 2 个 Router Server（主、从服务器）的 TCP 长连接，如果主 Router Server 故障，则使用从 Router Server。
   B，每个 Register Server 保持 2 个 Mail Server 的 TCP 长连接。
4. Database Servers 采取双机热备的方式。

上述方案的问题集中在 Presence Server 的冗余上，由于不是双机热备方式，所以一旦某台 Presence Server 故障，虽然备用机能及时补位，但故障机进程的内存数据中与数据库交集以外的数据（如用户的在线状态，已登录、忙碌、离开等状态数据不会持久化到数据库）将丢失。针对这个问题，在线用户的状态等数据可以在 Router Server 进程的内存中统一冗余缓存，这样就可以在 Presence Server 故障后得以恢复，但相应的增加 Router Server 的数据存储压力。

（X-Messenger 系统的下一个版本将实现真正意义上的分布式系统，上述诸多问题将迎刃而解。）

4. 服务器端子系统设计

4.1. 系统架构

4.1.1. Router Server 和 Presence Server 架构

Router Server 和 Presence Server 的工作方式基本一致：接收来自客户端和其它服务器端的 XMMEP 消息 --> 解析并处理消息 --> 如有必要，则更新数据 --> 如有必要，则回复对端。

由此可以将服务器端应用（不包括 Register Server）划分出几个核心模块：

1. I/O Demultiplexer，I/O 事件的多路复用组件和 I/O 事件处理的分派；
2. Message Handlers，消息处理器。由于 X-Messenger 各子系统之间的交流依赖于 XMMEP 消息，每中不同的消息都对应着不同的事件和针对该事件的处理流程，所以需要一系列的消息（或称事件）处理器来完成这些流程。
3. Message Handlers Map，消息（事件）处理器仓库，key 为消息的 code，value 则是消息处理器的指针，在处理消息的时候，可以通过该模块获取消息处理器。
4. Data Management，数据管理。数据管理两种：一是内存中缓存的用户数据，二是需要持久化而存储在数据库（或文件系统）中的数据；

以下架构图针对 Router Server 和 Presence Server，下图以 Router Server 为例，Presence Server 架构与 Router Server 相同，差别在于交互的对象（其它子系统）不同。

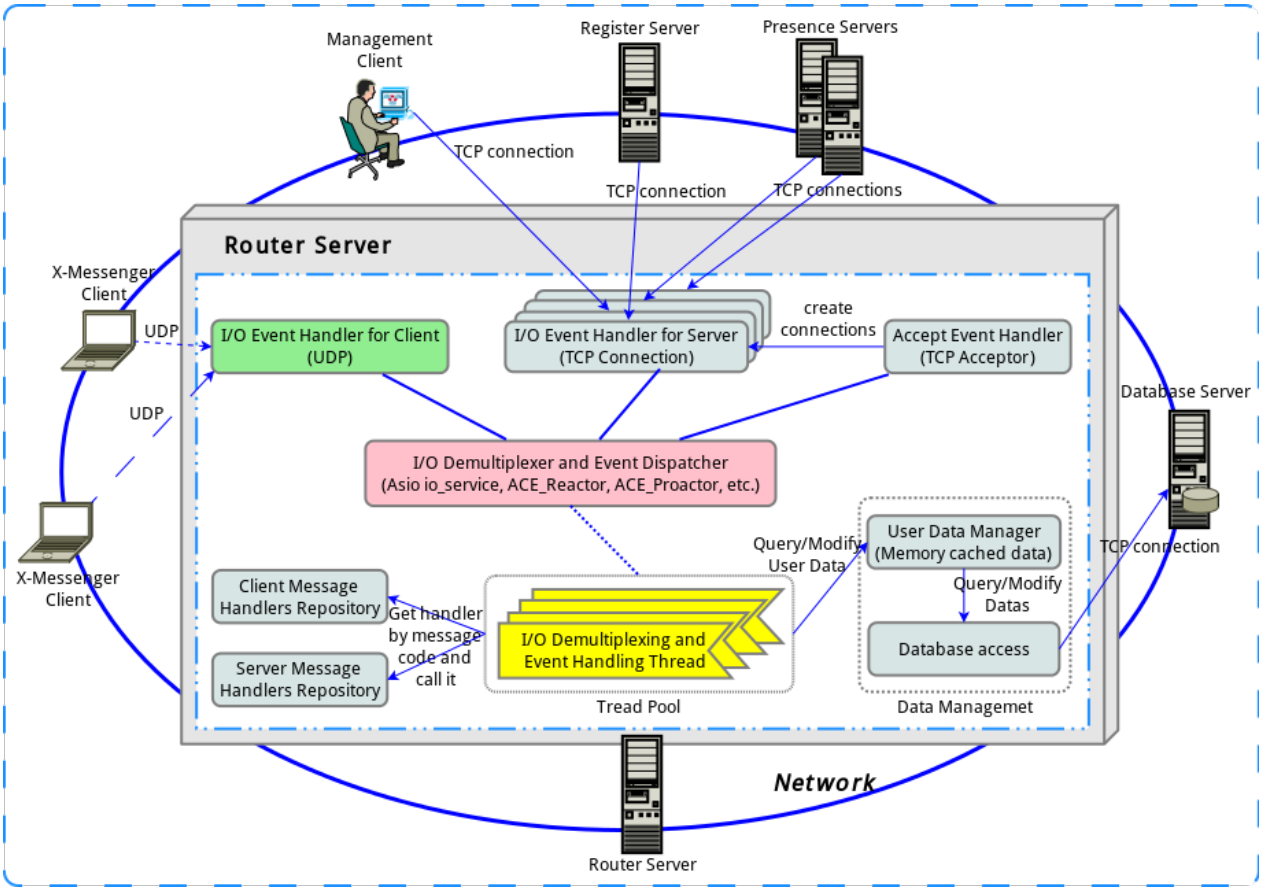

插图 4.1.1: 服务器端通用架构

4.1.2. Register Server 架构

Register Server 基于 Java Web 技术（架构描述略）。

4.2. 子系统实现技术

（以下仅讨论 Router Server 和 Presence Server。）

Router Server 和 Presence Server 使用 C++语言实现，严格使用标准 C++语言特性（可参考的 C++标准：C++03、C++11），优先使用 C++标准库。在此基础上，需要的第三方库的选择上以下面列举的库为准，如非必要则避免引入其它第三方库：

4.2.1. C++ Boost Library

C++ Boost 的库数量十分庞大，涉及范围很广泛，在 X-Messenger Servers 的实现过程中，可能涉及的库：

1. Boost.Asio：高性能网络 I/O 库，实现了 Proactor 框架，封装了各个操作系统中优秀的 I/O Demultiplexer，如 Linux 2.6 下的 Event Poll(epoll)、Windows 下 IOCP、UNIX 下的 Select 等；除 I/O Demultiplexer 外，还封装了 Event Dispatcher、Workers Queue 等必要的设施，

以及散读/聚写等机制，定时器等等；

2. Boost.Threads：可移植的线程库；
3. Boost.Containers：如 unordered_map、unordered_set、array 等；
4. Boost.Bind：通用函数对象适配器；
5. Boost.Function：函数对象包装器，可以替代函数指针，特别适合在需要回调的地方；
6. Boost.Property_Tree：可以作为读/写配置文件的工具
7. Boost.Date_Time，...

4.2.2. C++ ACE

与 boost 相比，ACE 也是一个庞大的 C++类库，很多库的功能上跟 Boost 的某些库重叠，但 ACE 更偏向网络编程。ACE 提供了基本的网络、线程、进程等操作系统原生 API 的薄封装，同时在更高层次上提供了一系列良好设计的框架，这些框架使得编写高性能的网络应用更为便捷、高效和安全。

在 X-Messenger Server 的实现过程中，可能会涉及的框架或类库：

1. ACE_Reactor 框架：I/O Demultiplexer 和 Event Dispatcher；
2. ACE_Task 框架：结合 ACE_Reactor 可实现半同步/半异步的线程模型；
3. ACE_Proactor：异步 I/O 以及 Event Dispatcher；
   上述 3 者可以结合使用，如同 Client 交互的部分可以使用 ACE_Proactor，同其它 Server 交互的部分可以使用 ACE_Reactor 等；
4. ACE_Tread_Manager：线程管理；
5. ACE_Log_Msg：ACE 的日志组件；
6. ACE_Configuration_Heap：配置文件的读写；
7. 其它需要用得上的组件。

注：在第三方库的选择上，Boost 和 ACE 是二者选其一，不要将二者混合使用。

4.2.3. C++ OTL

由于 X-Messenger 当前版本需要通过关系数据库(RDBMS)——MySQL 数据库做用户相关的数据持久化，所以需要通过 C++连接、操作数据库。

当前版本规定使用最新版本的 OTL，通过 ODBC 的方式连接 MySQL 数据库服务器，所以除 OTL 外，还需安装 libmyodbc 的适当版本。

4.3. 子系统部署

各个子系统的部署原则是，软/硬件稳定、性价比高、容易部署和维护。

4.3.1. 硬件配置

（略）

4.3.2. 软件环境

4.3.2.1. Router Server 和 Presence Server

1. 操作系统：Linux 2.6.18+，建议安装 CentOS 5.x、CentOS 6.x、Debian 5.x 或之一各自更新的版本；
2. 数据库：MySQL 5.1.x、MySQL 5.5.x 之一；
3. 最新版本的 SSH Server；

4.3.2.2. Register Server

1. 操作系统：Linux 2.6.18+，建议安装 CentOS 5.x、CentOS 6.x、Debian 5.x 或之一各自更新的版本；
2. Java SE：Sun Java SE 6 update 20 或更新版本；
3. Java Web Server：Apache Tomcat 6.x；
4. 最新版本的 SSH Server；

4.3.2.3. Mail Server

1. 操作系统：Linux 2.6.18+，建议安装 CentOS 5.x、CentOS 6.x、Debian 5.x 或之一各自更新的版本；
2. MTA：Sendmail 最新版本；
3. 最新版本的 SSH Server；

5. 消息处理流程

X-Messenger 各个关联的子系统之间是通过 XMMEP 协议中的消息进行交流，每个消息均对应不同的事件，以下为消息交互的流程描述。

5.1. 用户注册

用户注册分两个阶段：用户注册和用户激活。用户成功注册后，将收到用户激活邮件，通过激活邮件激活之前注册的用户，用户只有在成功激活后才能通过 X-Messenger 客户端登录到服务器。

5.1.1. 用户注册

消息流程图：

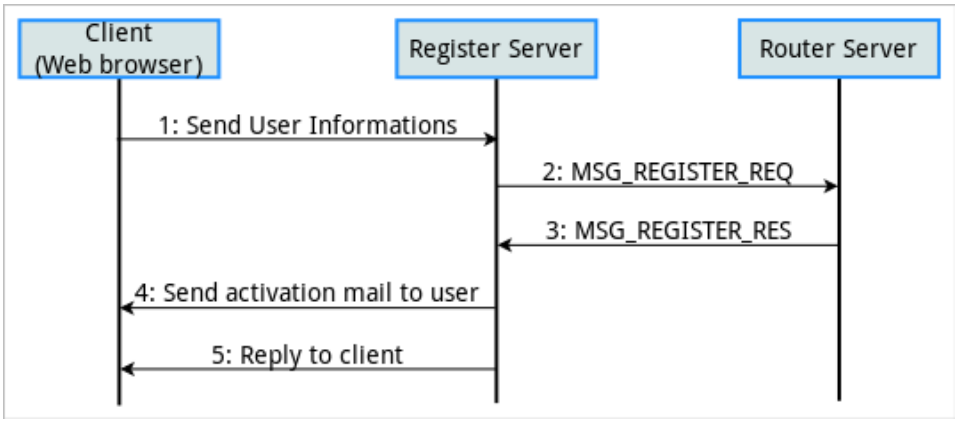

插图 5.1.1: 用户注册流程

流程说明：

1. Client 向 Register Server 发送用户注册信息，具体信息字段见 MSG_REGISTER_REQ 消息规定；
2. Register Server 将 Client 发送的用户信息组装成 MSG_REGISTER_REQ 消息，发送至 Router Server；
3. Router Server 验证用户信息，包括用户名、email 是否已经存在，将结果通过 MSG_REGISTER_RES 消息发送给 Register Server；
4. Register Server 根据 MSG_REGISTER_RES 消息的结果，如果注册成功，则发送用户激活消息至用户注册时填写的 email 地址；
5. Register Server 根据 MSG_REGISTER_RES 消息的结果，回复 Client。

5.1.2. 用户激活

消息流程图：

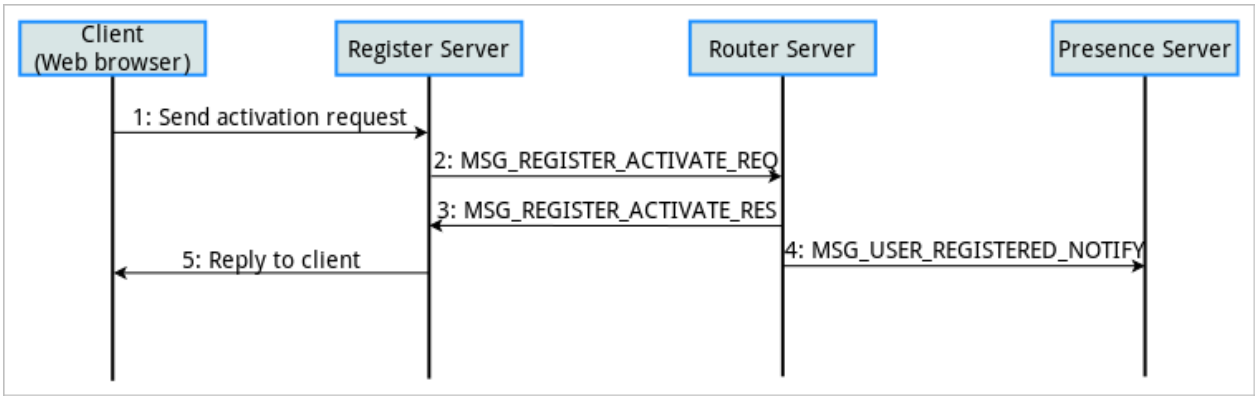

插图 5.1.2: 用户激活流程

流程说明：

1. 用户收到激活邮件后，点击激活的链接，通过 Web browser 向 Register Server 发送激活请求；
2. Register Server 收到 Client 激活请求后，向 Router Server 发送 MSG_REGISTER_ACTIVATE_REQ 消息；
3. Router Server 验证用户信息，包括用户名是否存在、用户名和验证码是否匹配，将结果通过 MSG_REGISTER_ACTIVATE_RES 消息返回给 Register Server；
4. 如果步骤 3 检查结果为 Success，Router Server 根据一定的算法，将用户分配到已有的某个 Presence Server，并向该 Presence Server 发送 MSG_USER_REGISTERED_NOTIFY 消息，表明添加了新用户，（Presence Server 则需存储该新用户的信息）。
5. Register Server 根据 MSG_REGISTER_ACTIVATE_RES 消息的结果，回复 Client；

5.1.3. 用户注册信息更新

用户注册信息的修改可以通过两个渠道完成：1，通过 X-Messenger Client 直接发送信息到 Presence Server 修改，2，使用 Web 浏览器，通过 Register Server 完成修改。

消息流程图：

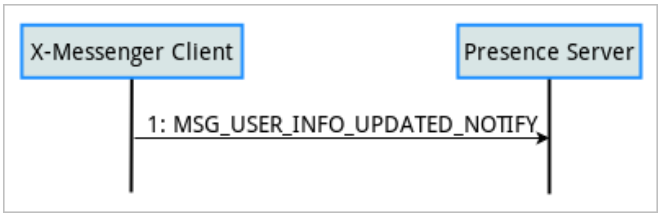

插图 5.1.3: 用户信息更新流程 1 - 通过 X-Messenger Client 更新

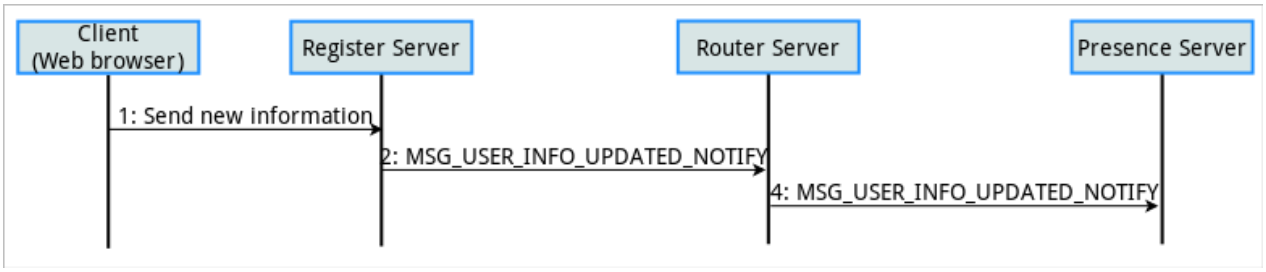

插图 5.1.4: 用户信息更新流程 2 - 通过 Register Server 更新

5.2. 用户登录流程

5.2.1. 用户登录

用户登录，对于服务器端而言，除了要回应登录用户外，还涉及该用户的好友和加该用户为好友的用户。

消息流程图：

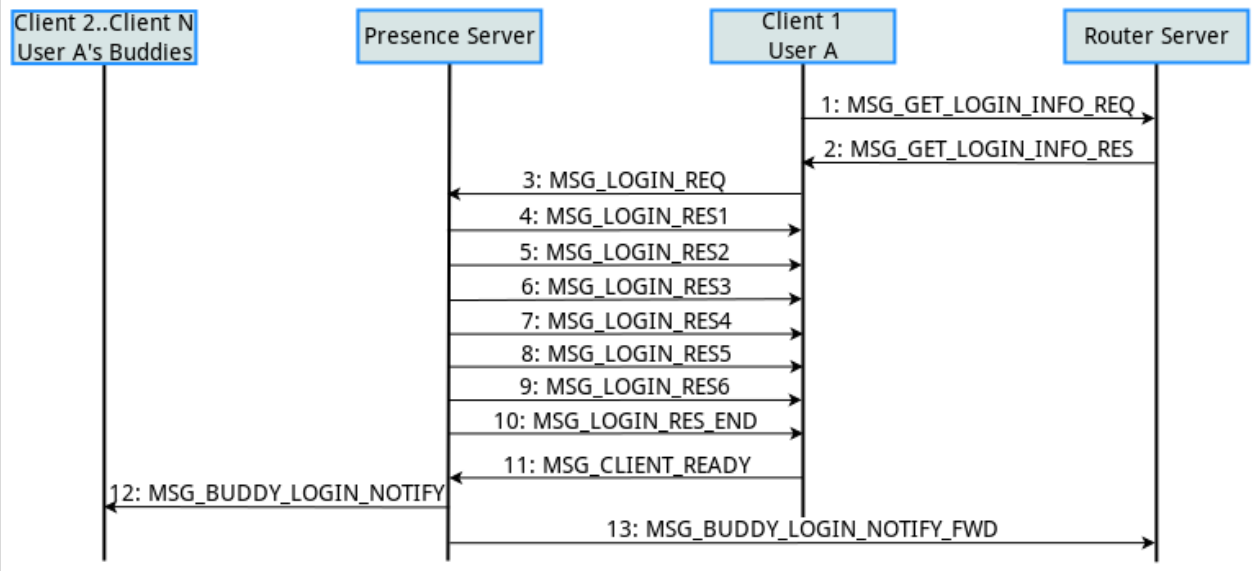

插图 5.2.1: 用户登录流程

流程说明：

1. 由于 Client 1 并不知道 User A 位于哪一个 Presnece Server，所以在登录到 Presence Server 之前需从 Router Server 查询，也即发送 MSG_GET_LOGIN_INFO_REQ 消息到 Router Server；
2. Router Server 发送 MSG_GET_LOGIN_INFO_RES 回应 Client 1，告知相应的 Presence Server 的 IP 和 Port 信息；
3. Client 1 收到 Router Server 的 MSG_GET_LOGIN_INFO_RES 消息后，向 Presence Server 发送登录请求消息 MSG_LOGIN_REQ；
4. Presence Server 回应 MSG_LOGIN_RES1，消息内容为登录结果、User A 的个人信息；
5. 如果 MSG_LOGIN_RES1 的登录结果为 Success，且 User A 的好友组集合不为空，则 Presence Server 继续发送 MSG_LOGIN_RES2；
6. 如果 User A 的好友列表不为空，Presence Server 继续发送 MSG_LOGIN_RES3；
7. 如果 User A 的黑名单不为空，Presence Server 继续发送 MSG_LOGIN_RES4；
8. 如果 User A 有参与的或其创建的 chatroom，Presence Server 继续发送 MSG_LOGIN_RES5；
9. 如果 User A 的在线好友和加 User A 为好友的在线用户的并集不为空，Presence Server 继续发送 MSG_LOGIN_RES6；
10. 至此 Presence Server 发送 MSG_LOGIN_RES_END，标志登录回应结束；

11. Client 1 在接收到 MSG_LOGIN_RES_END 消息后，发送 MSG_CLIENT_READY 至 Presence Server 标志 Client 1 已准备就绪；
12. Presence Server 在收到 MSG_CLIENT_READY 后，向 User A 的在线好友和加 User A 为好友且目前在线的用户们发送好友上线通知，即 MSG_BUDDY_LOGIN_NOTIFY 消息；
13. 同时，Presence Server 发送 MSG_BUDDY_LOGIN_NOTIFY_FWD 消息通知 Router Server，User A 登录成功。

注意：A) 上述流程中，如果第 2 步骤的结果不为 Success，则以后步骤不存在；B) 同样，如果第 4 步骤结果不为 Success，则其下步骤不存在；C) MSG_LOGIN_RES2 ~ MSG_LOGIN_RES6 消息可能都不需发送，或部分消息不需发送。

5.2.2. 会话状态报告

X-Messenger 目前版本规定同一个用户同一时刻只能有一个会话，如 User A 已经登录，而后再在其它客户端中登录，这样将造成先登录的会话终止。该事件由用户登录流程触发。

消息流程图：

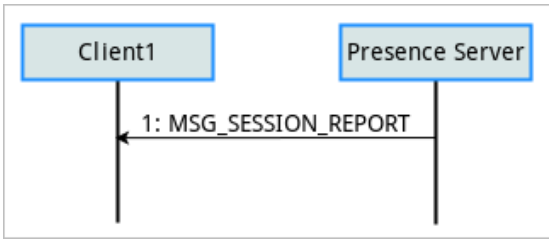

插图 5.2.2: 会话状态报告

流程说明：

1. （当相同的用户在其它的客户端中登录成功后），Presence Server 发送 MSG_SESSION_REPORT 消息通知原客户端 Client1；

5.3. 用户状态管理

对于用户状态，客户端关注的是好友的状态，而服务器端关注的是用户本身的状态变化。用户状态管理是 IM 服务器端软件的主要功能之一。

5.3.1. 用户状态改变

消息流程图：

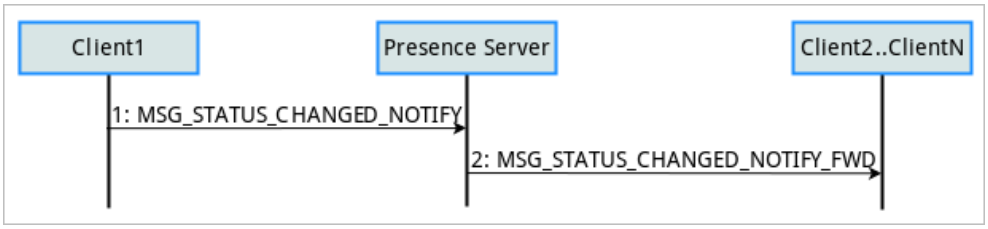

插图 5.3.1: 用户状态改变

流程说明：

1. Client1 改变 User A 的状态后，发送 MSG_STATUS_CHANGED_NOTIFY 至 Presence Server；
2. Presence Server 将该消息转发到 User A 的所有在线好友、以及所有加 User A 为好友的在线用户的并集。

注：状态改变通知，不包括用户登录后引起的状态改变。

5.4. 好友管理

5.4.1. 添加好友

消息流程图：

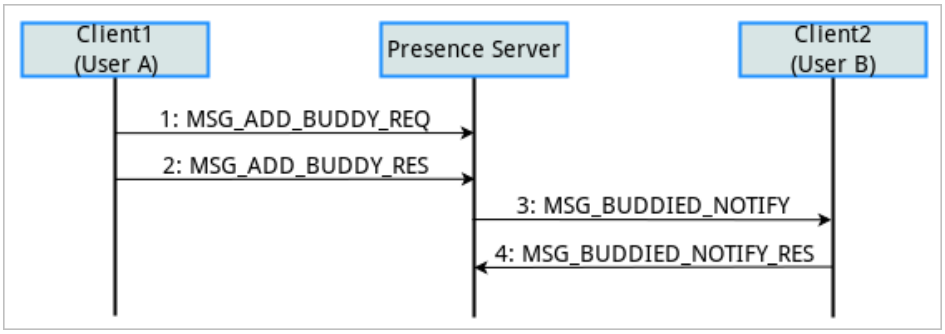

插图 5.4.1: 添加好友

流程说明：

1. Client 1(User A)要添加 User B 为好友，则发送 MSG_ADD_BUDDY_REQ 到 Presence Server；
2. Presence Server 回应消息 MSG_ADD_BUDDY_RESClient 1；
   如果 MSG_ADD_BUDDY_RES 的结果为 Success，并且 User B 的好友列表中没有 User A，才继续执行以下步骤：
3. Presence Server 发送 MSG_BUDDIED_NOTIFY 消息到 Client 2(User B)，告知其被 User A 加为好友；
4. Client 2 发送 MSG_BUDDIED_NOTIFY_RES 至 Presence Server 回应是否要加 User A 为好友。

5.4.2. 删除好友

消息流程图：

插图 5.4.2: 删除好友

5.4.3. 修改好友信息

消息流程图：

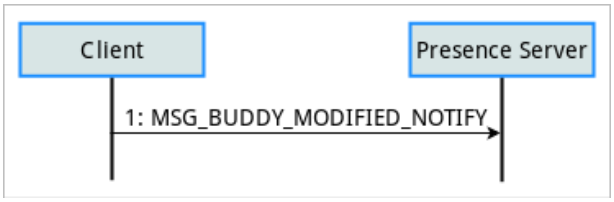

插图 5.4.3: 修改好友信息

5.4.4. 阻止用户(加入黑名单)

消息流程图：

插图 5.4.4: 阻止用户

5.4.5. 解除阻止用户(从黑名单中删除)

消息流程图：

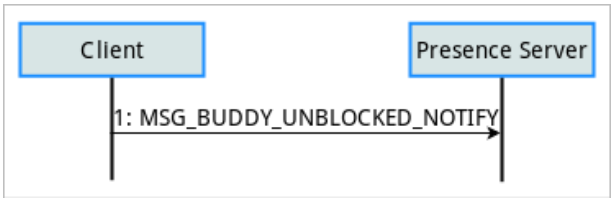

插图 5.4.5: 解除阻止用户

5.4.6. 添加组

消息流程图：

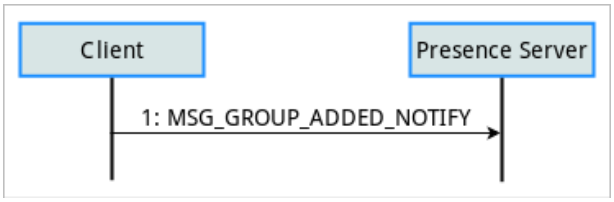

插图 5.4.6: 添加组

5.4.7. 删除组

消息流程图：

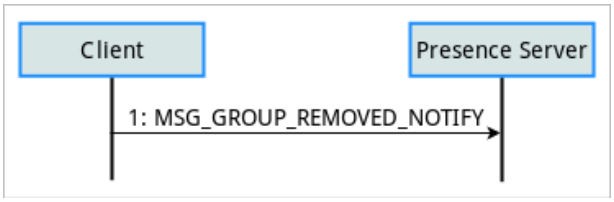

插图 5.4.7: 删除组

5.4.8. 组重命名

消息流程图：

插图 5.4.8: 组重命名

5.5. 聊天消息

5.5.1. 即时消息

消息流程图：

插图 5.5.1: 即时消息

5.5.2. 离线消息

消息流程图：

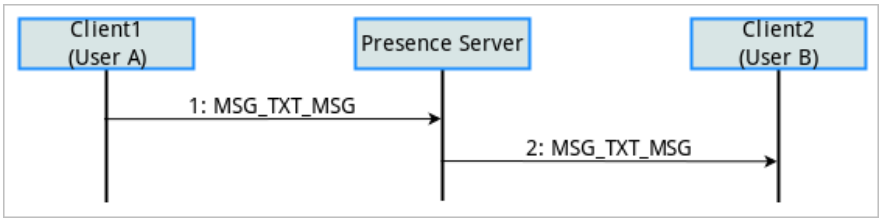

插图 5.5.2: 离线消息

流程说明：

1. User A 向 User B 发送聊天消息，如果 User B 离线，则 Client 1 发送 MSG_TXT_MSG 至 Presence Server；
   Presence Server 将消息持久化至文件系统，暂存；
2. 当 User B 登录后，Presence Server 从文件系统读取离线消息，发送至 Client 2。

5.6. 文件传输

X-Messenger 文件传输实现 P2P 方式，即文件的发送、接收只在客户端之间直接进行，为了穿越 NAT，需要借助服务器来实现。

5.6.1. 在线文件传输流程

消息流程图：

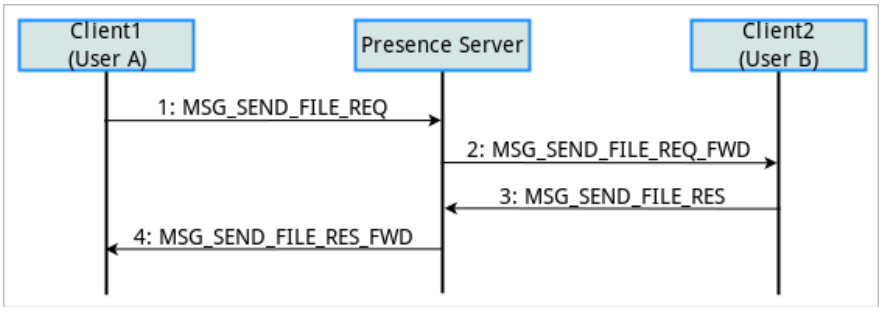

插图 5.6.1: 在线文件传输

流程说明：假设 Client 1 的用户为 User A，Client 2 的用户为 User B

1. User A（通过新建的 Socket，Socket_NA）发送消息 MSG_SEND_FILE_REQ 至 Presence Server；
2. Presence Server 将接收到的 MSG_SEND_FILE_REQ 加入 Client1 的 IP 和 Port，重新包装为 MSG_SEND_FILE_REQ_FWD 消息，发送至 Client2；
3. Client 2 分两种情况回应：
   A) 如果愿意接收文件，则（通过新建的 Socket，Socket_NB）发送 MSG_SEND_FILE_RES 至 Presence Server；
   B) 如果不愿意接收文件，则使用原有的 Socket 发送 MSG_SEND_FILE_RES 至 Presence Server；
4. Presence Server 将接收到的 MSG_SEND_FILE_RES 加入 Client2 的 IP 和 Port，重新包装为 MSG_SEND_FILE_RES_FWD 消息，回应 Client1；
   （以下为客户端之间的操作）

5. 如果在回应消息 MSG_SEND_FILE_RES_FWD 中，User B 愿意接收文件，则 Client1（通过 Socket_NA）向 Client2（Socket_NB）发送一个字符（任意字符）；
6. 同样，Client2（通过 Socket_NB）向 Client1（Socket_NA）发送一个字符（任意字符）；（上面 5、6 两个步骤是 NAT 穿越过程中的必要步骤）
7. NAT 穿越之后，Client1 和 Client2 通过 Socket_NA、Socket_NB 发送、接收文件内容。

5.6.2. 离线文件传输

（当前版本暂不实现）

5.7. 好友信息查询

5.7.1. 获取好友信息

消息流程图：

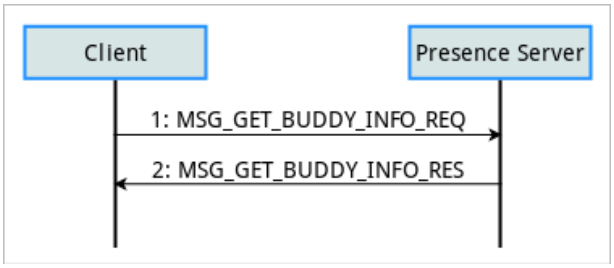

插图 5.7.1: 获取好友信息

5.7.2. 好友搜索

消息流程图：

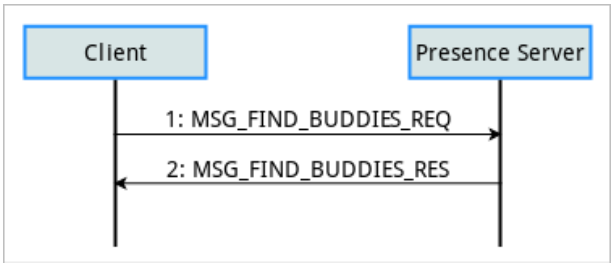

插图 5.7.2: 搜索好友