# 

# 程序文档（实践01）版本

Git链接：<https://github.com/BrunoQin/Open-Reuse.git>

组员：1352958 金敏 1352911 曹琦 1354366 潘舜达

1352961 秦博 1352839 饶伊文

**目录**

[程序文档（实践01）版本 1](#_Toc660)

[1. 引言 2](#_Toc16041)

[1.1 编写目的 2](#_Toc22112)

[1.2 背景 2](#_Toc31624)

[2. 项目设计说明 2](#_Toc5785)

[2.1 项目描述 3](#_Toc30587)

[2.2 功能 3](#_Toc2434)

[1. 客户端登录 3](#_Toc5267)

[2. 注册 4](#_Toc21087)

[3. 客户登录后向服务器发送消息 4](#_Toc21681)

[4. 服务器广播接收到的消息 5](#_Toc20620)

[2.3 数据流说明 5](#_Toc29364)

[2.4 流程逻辑/架构 6](#_Toc12413)

[整体架构 6](#_Toc6675)

[Client 架构 7](#_Toc1395)

[Server 架构 7](#_Toc22557)

[2.5 主要接口及其测试 8](#_Toc14894)

[2.6 可复用构件 10](#_Toc31872)

[IO多路复用 10](#_Toc22505)

[程序模块复用 10](#_Toc2758)

[程序组件复用 11](#_Toc24149)

[3. 开发成果及说明 11](#_Toc11490)

[界面 11](#_Toc22279)

[优势 12](#_Toc14782)

**1. 引言**

**1.1 编写目的**

本报告旨在说明本项目的框架设计，代码实现及测试情况。预期读者为架构师，开发者，测试人员等软件行业专业人员。

**1.2 背景**

本项目的开发形式为 Team Work，成员包括 1352958 金敏，1352911 曹琦，1354366 潘舜达，1352961 秦博，1352839 饶伊文。

本项目开发环境为Java，IDE为Intellij IDEA，使用了gradle构建工具和Hibernate映射框架。数据库使用MySQL。

版本控制工具使用 Github (https://github.com/BrunoQin/Open-Reuse.git )。测试工具使用 Junit 和 Mockito测试框架。

**2. 项目设计说明**

**2.1 项目描述**

该项目主要完成客户端和服务器的基本通信的设计和实现，包括客户端的登录，注册，收发消息以及服务器的接收和转发消息。

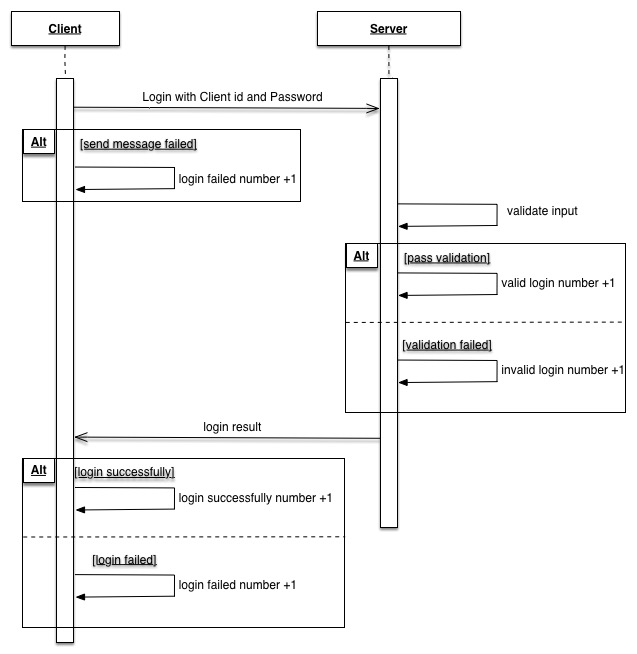
**2.2 功能**

**1. 客户端登录**

客户利用用户名和密码登录已知地址的服务器，在登录之后保持长连接。

输入项：用户名，密码，服务器地址。

流程图：



login

输出项：客户端每分钟登录成功和失败的次数；服务器端每分钟验证登录成功和失败的次数。

**2. 注册**

输入项：用户名和密码。

处理：判断该用户是否存在，若存在则注册失败，若不存在则注册成功。

输出项：注册结果，成功或失败。

**3. 客户登录后向服务器发送消息**

输入项：消息内容

处理：服务器验证接收到的消息并返回response。

验证方式如下：

忽略客户端未登录状态下发送的消息

每秒钟只接受5条消息，超过5条的消息忽略。

用户每次登录只允许发送100条消息，服务器接收到第100条消息则回复 “redo login”，在用户重新登录之前的消息均忽略。在此期间保持服务器和客户端的连接，之前忽略的消息不计入重新登录后的100条记录内。

如果验证通过则返回“OK”。

其他要求如下：

客户端在接收到“redo login”的回复后自动重新登录。

输出：客户端记录发送消息的数量。服务端记录接收消息的数量和忽略消息的数量。

**4. 服务器广播接收到的消息**

输入：服务器接收到的有效的消息

处理：服务器转发给每个在线的客户端。

输出：客户端记录接收到的消息的数量。服务端记录转发的消息的数量，其中每条消息转发给每个客户端均计数＋1。

**2.3 数据流说明**

本项目中的Message采用JSON进行序列化处理和解析。消息格式如下：



**Login Message**

Type: LOGIN\_MESSAGE

From: User Address To: Server Address

Body: Username

服务器接收到该消息进行广播。

**Logout Message**

Type: LOGOUT\_MESSAGE

From: User Address To: Server Address

Body: Username

服务器接收到该消息进行广播。

**Register Message**

Type: REGISTER\_MESSAGE

Body: Username Password

服务器接收到该消息进行单播，为保证安全性，反馈信息的body只有username。

**Redo Login**

Server向Client发送信息，要求其重新登录。

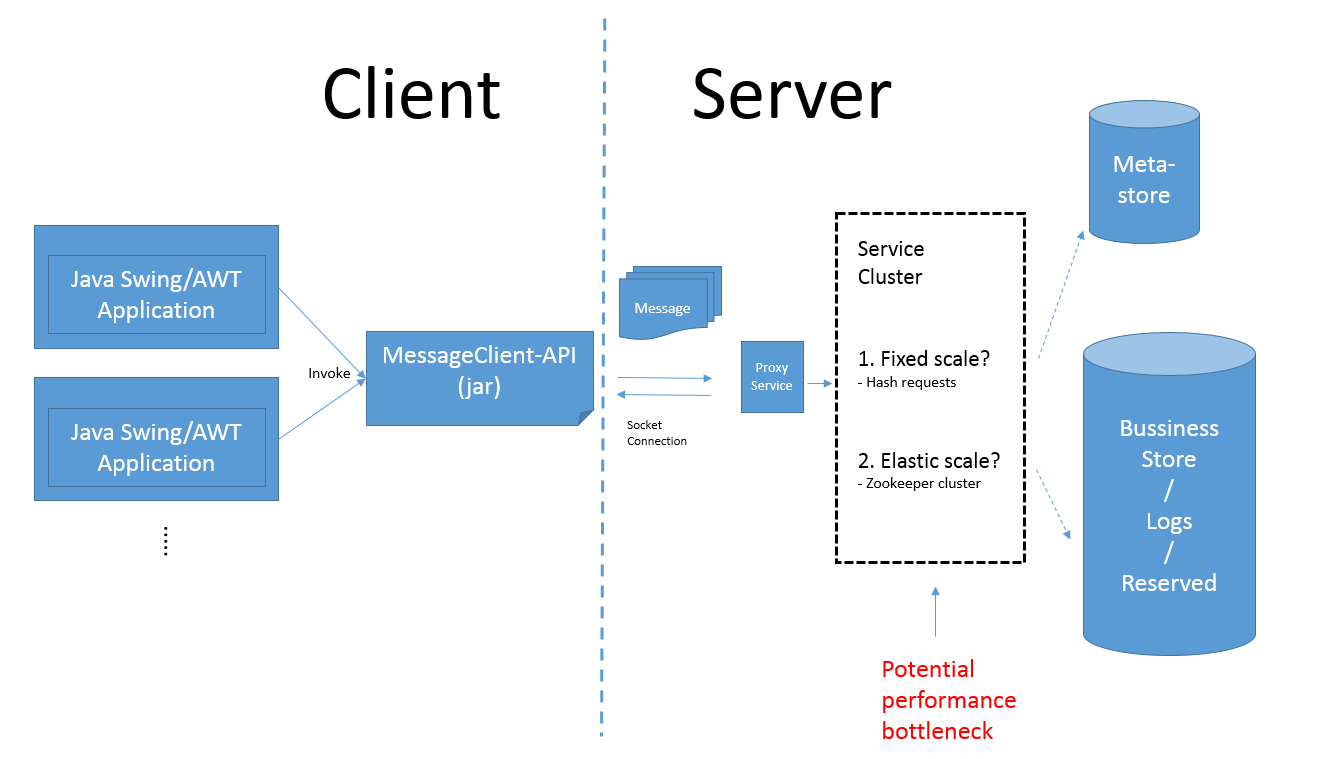
Internal Error

数据爆满信息，唤醒系统进行相应的自救处理。

**2.4 流程逻辑/架构**

**整体架构**

本项目概要设计阶段，将项目划分为三个模块：Client Window, Client API, Server，如下图所示：

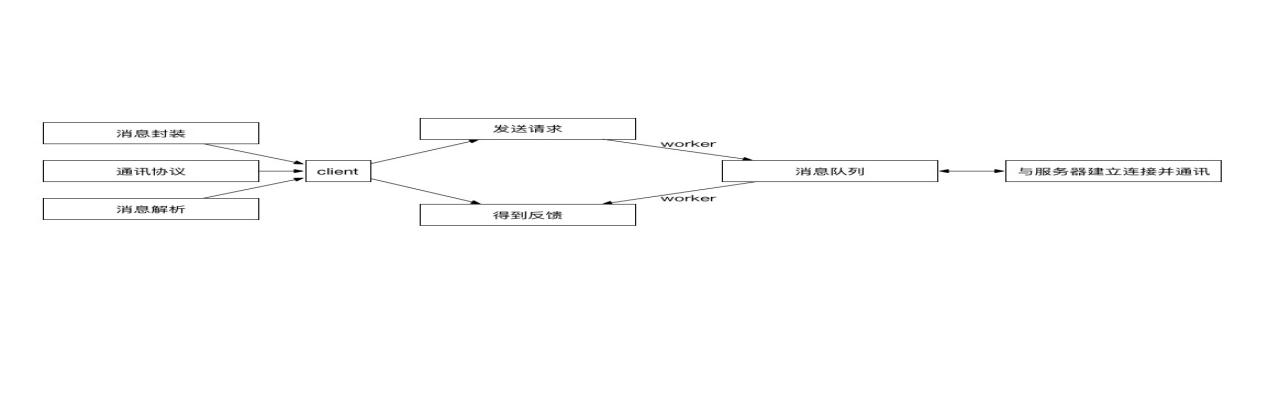


Archtecture

其中Client API 封装客户端逻辑，负责与Server通过Socket Connection连接发送消息，同时为Client Window 提供必要的验证接口。Server端计划利用hash request的方法处理fixed scale，而针对 Elastic scale，比如并发量激增等情况，则采用Zookeeper分布式框架提供集群管理的服务。对于数据和业务的存储，服务端计划采用meta store数据库表结构存储数据。而为了使业务更高效，计划采用日志系统和缓存

在详细设计阶段，我们具体设计了Client架构和Server的架构。

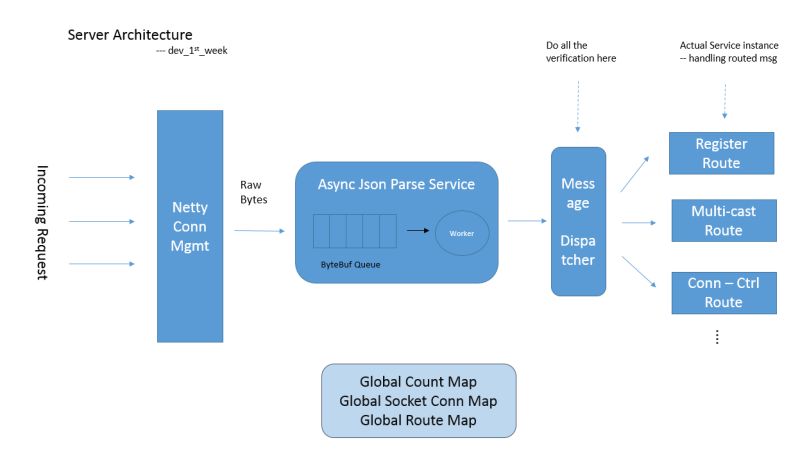
**Client 架构**



Archtecture

Client端的主要工作包括消息的封装，解析和传递，通过Socket协议与服务器连接。具体而言，Client将Window 传递而来的消息序列化封装之后发送至一个延迟队列，Server端通过多线程（worker）从延迟队列中取出消息，进行必要的数据处理，之后将反馈发送至客户端，客户端将消息解析后通过回调机制传递给Client Window。

**Server 架构**



Archtecture

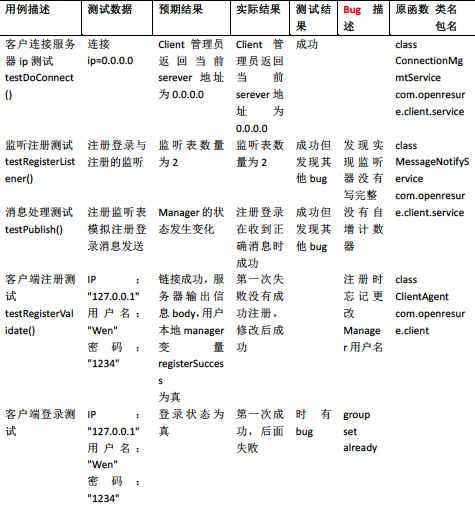
整体采用了Netty架构。对于incoming request，服务端将MESSAGE解析成相应的格式（Raw Bytes），并且将其添加到一个延迟数组中。就在此时，分配给这个延迟数组的三个worker（线程）就开始工作。worker根据一定的时间从延迟数组中取得数据，并且当其中一个获取数据时，另外的worker不能重复获取，这个worker将message进一步解析，并且得到其type将message分配到专门的处理路径（route）中。在这些route里有DAO等工具对数据进行处理。当得到结果后，根据message中的请求（单播或者广播）来进行message的反馈。同时Server端维护了三个map：Global Count Map（记录全局的登录注册消息数量）, Global Socket Conn Map（管理连接到server的客户端的数量）, Global Rout Map（控制message被分配到某个route）.

**2.5 主要接口及其测试**

**服务端**



**客户端**



需求实现分析：

实现了对消息数目（消息队列大小限制）的限制

实现了对登录注册收发消息操作计数并输出到文件

因为时间较紧，起初实现监听器没有写完整（缺少对再登录，发信息等状态的监听），后期补充后完整

**2.6 可复用构件**

**IO多路复用**

为了避免过度复杂的Java原生Nio套接字开发，我们使用了Netty通讯框架作为客户端和服务端相互传输消息的基础。Netty框架采用Reactor模式对套接字进行复用，和传统的BIO的套接字相比减少了高并发场景中上下文频繁切换导致的性能低下。并且使用Netty优化的线程池代替Java Concurrent库中提供的线程池，更有针对性，对性能的提升更加显著。

**程序模块复用**

我们将整个项目划分为四个模块，其中open-reuse-common是被server端和client端复用的模块，包括定了消息结构与格式，消息枚举类型，其中消息的构造使用了Builder模式，课程要求统计数据的转储方法。open-reuse-client-api是共window模块复用的，旨在提供一个开放可供多个窗口调用的健壮API，其内部实现也包括对客户端套接字对应的channel的复用，这也是Netty长连接的保证。

**程序组件复用**

程序实现细节上一些组件的复用包括：

server端解耦消息的传输和业务处理，所有消息在server内部的输入输出通路被完全复用。

client端channel的复用，多个窗体共用维护一个channel。

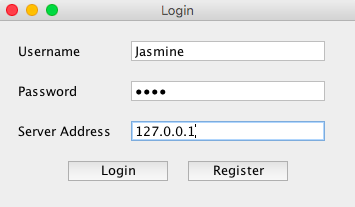
server处理分派逻辑的复用，所有消息都是通过消息分派组件从而route到与之相对应的具体实现方法（单例服务）。

**3. 开发成果及说明**

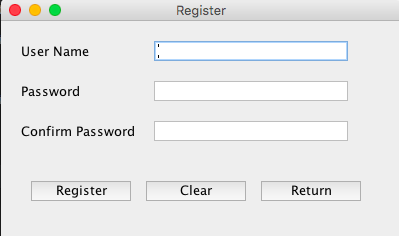
经过两周的设计和开发，主要框架基本完成，设计阶段的主要部分都得到了实现，整个项目组织结构清晰，能满足功能性的需求，同时兼顾性能的维护。

**界面**

登录：

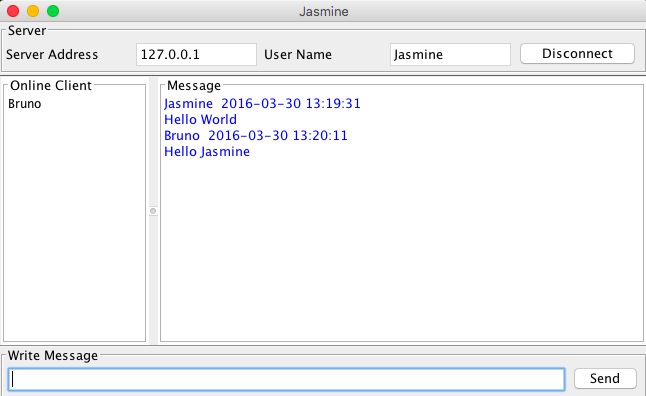


注册：

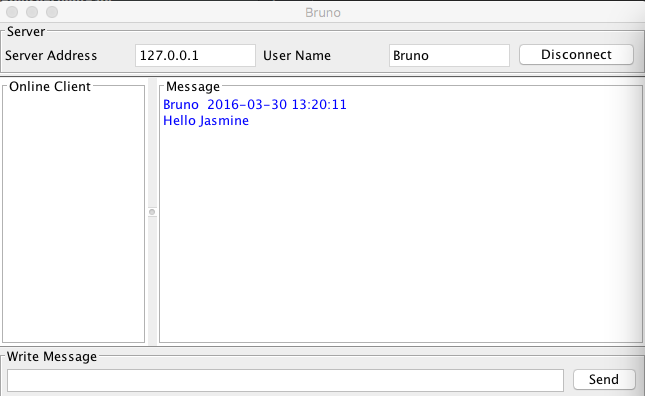


发送消息：

Client-1 Jasmine



Client-2 Bruno



**优势**

未依赖现有的网络通信中间件，独立设计和完善通信机制，项目架构清晰，代码低耦合，具有良好的可扩展性。

在设计和实现过程中，考虑功能性需求的同时考虑速度的保证和性能的维护。比如对登录次数等数据的统计在核心task结束后利用回调机制统计，将其从核心逻辑中解藕，以此保证核心task的工作效率。

数据库采用Hibernate映射方案，使数据库的操作更为方便高效。

改进之处

为了提高效率，可以在服务端添加三级缓存，这也是我们下一步工作的重点，即尽可能提高效率。

由于时间有限，虽然设计了对Elabric Scale的处理，即zookeeper，但并未实现。