

Lethe Project

架构设计

|  |  |
| --- | --- |
| 组长 | 15211105 于涌溢 |
| 成员 | 14211050 蒋松儒 15211105 于涌溢  15211092 闫涛 15211082 吴易泽  15211087 侯传嘉 14005025 黄邦靖  15211097 苟义钦 15211117 陈少华  15211123 吴鹏 15211076 练钊荣  15211035 刘希阳 15211093 何宇阳 |

2018年5月5日

目录

[简介 3](#_Toc513304285)

[目的 3](#_Toc513304286)

[架构概览 4](#_Toc513304287)

[客户端服务器架构 4](#_Toc513304288)

[逻辑表示 5](#_Toc513304289)

[实现方案 5](#_Toc513304290)

[架构表示 6](#_Toc513304291)

[组件视图 7](#_Toc513304292)

[数据视图 9](#_Toc513304293)

[存储方式 9](#_Toc513304294)

[使用方式 10](#_Toc513304295)

[部署视图 10](#_Toc513304296)

# 简介

## 目的

此文档的目的在于，为最终应用系统提供一个综合的架构概览，简洁地捕捉和传达至关重要的架构决策，使每一位小组成员能够充满信心地进行接下来的开发工作。

在这份文档中，采取了三个不同的架构视图来描述该系统的不同方面，这样的决策是基于这样的一个想法：由于我们实际工程经验不足，所以我们不打算照搬现成的架构设计模板，而是在参考模板的同时，进行大刀阔斧的裁剪，使其满足我们的需要。

# 架构概览

## 客户端服务器架构

我们采取客户端服务器架构，系统的实现逻辑非常简单，以两个人聊天为例，用户使用客服端发送消息，消息被发往服务器，服务器知道你的目标用户，于是对消息进行转发。

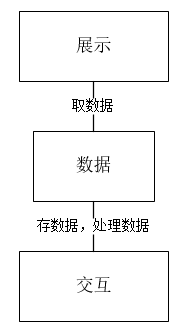
我们避免了采用传统的http服务器模式，没有轮询，而是采用web socket协议，服务器也可以主动地发送消息给客户端。

具体来说就是，服务器端由node.js实现，采用socket.io库，实现实时的，双向的，基于事件的通信。整个流程简要的说来就是：

1. 用户点击发送按钮之后，客户端“调用（引发事件）”服务器端的“聊天”方法，将聊天内容与目标对象作为参数传递。
2. 服务器端处理“聊天”方法，“调用”目标对象客户端的“聊天”方法，将聊天内容与消息来源作为参数传递。

这样就完成了一次简单的消息传递。

## 逻辑表示

整个系统逻辑上来说是三个部分，数据是这个系统的核心，展示和交互都围绕数据进行。

1. 表示：可类比网页开发中的前端。例如，当进入聊天页面的时候，需要读取个人信息，将昵称，头像等显示在页面上；还需要读取数据库中的聊天信息，构建聊天列表。
2. 交互：可类比网页开发中的后端，主要处理业务逻辑。例如，当用户发送消息，消息应该如何处理，如何转发。
3. 数据：专注于与数据库的交互，进行数据的存取。

## 实现方案

服务器端我们采用node.js，客户端存在两种选择，一种是桌面端，另一种是web端。

桌面端的实现，只需要使用Electron，便可以打包生成Linux，Windows，Mac三个平台的可执行程序。在具体编码的时候，还可以使用node.js。web端的实现，和一般的网页开发差别不大。

综合考虑后，我们决定编写客户端的时候，只使用javascript，而不使用node.js，这样最后既可以用Electron打包生成桌面端应用程序，也可以在浏览器环境下直接使用本系统。

## 架构表示

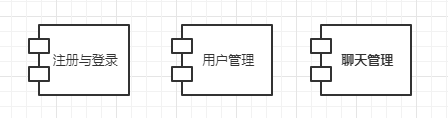
此文档将架构表示为一系列的视图：组件视图，数据视图，部署视图。

此类型的开发和一般开发的不同之处在于：小组各成员对所要采取的技术并不熟悉。因此，和一般开发的架构设计文档不同，这份文档不能过于抽象，正式。所以，在架构的表示中，总体来说是基于UML，但会灵活地使用我们自己创造的沟通表达方式，也会更多地关注具体细节。一切都围绕着增进大家对将要开发的系统的理解，而不是一份循规蹈矩的正式文档。

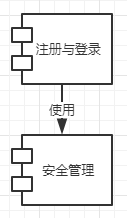
三个视图的内容简要介绍如下：

* 组件视图：这里说的组件，更多的是从功能上对整个系统的划分，每一个部分都相对独立，可作为单独的工作任务进行分配。对这一部分的理解，可以回答“我们做什么？”这个问题。
* 数据视图：这部分主要关注这个系统中数据的存储方式（静态的，偏向于数据库的）与使用方式（动态的，偏向于业务逻辑）。存储方式回答这样的问题，比如，我们知道会有用户聊天，那么聊天消息怎么存，用户信息怎么存？使用方式回答这样的问题，比如，聊天消息，用户信息，在系统的使用过程中，是怎样被使用的？文本消息和文字表情消息有什么区别？
* 部署视图：这部分主要关注系统如何使用，比如，我们需要怎样的服务器，以及如何连接并使用该系统。

# 组件视图



服务器端主要由注册与登陆，用户管理，聊天管理三部分组成。这三部分是相互独立的。

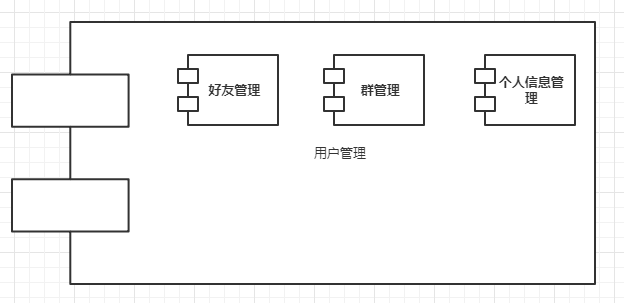
* 注册与登陆：

这一部分的业务逻辑主要是：用户提供昵称和密码进行注册，系统返回用户id（类比qq号码），提示注册成功，跳转登陆页面。用户输入id和密码进行登陆，成功后跳转聊天页面。

安全管理部分需要提供的服务是，确保用户账号密码安全，比如存入数据库时候需要进行加密；控制访问权限，比如，作为web端实现的时候，未经登陆，对聊天页面的访问应该是禁止的。

* 用户管理：

首先应该说明的是，群组的实现是可选的，所以在目前描述设计的时候，我们将群组的实现一起描述，最终是否实现，视情况而定。



用户管理包括独立的三个部分：用户管理，群管理，个人信息管理。这一部分的业务逻辑主要是：对好友和群的增删查，对个人信息的改。

* 聊天管理：

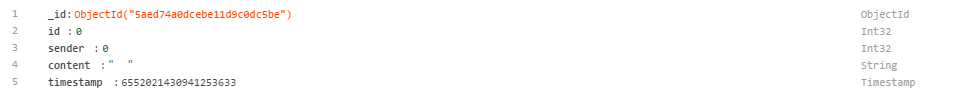
这一部分的主要业务逻辑是：历史消息的拉取，即使消息的发送与接受。其中消息分为文字消息，文字表情消息和图片消息。

# 数据视图

## 存储方式

本系统使用MongoDB，各个重要的数据对象设计如下：（由于UML数据库表示一般用于关系型数据库，MongoDB又是NoSQL数据库，所以我们直接使用截图。）

用户：

消息：

聊天实例：

群：

## 使用方式

* 用户与群：用户（群）信息将用于聊天界面与聊天列表，比如聊天的时候显示互相的头像，在聊天列表显示各自的昵称与id。
* 消息：消息将直接展示在聊天界面。
* 聊天实例：聊天实例主要用于服务器端对消息进行转发，实例作为持久化的对象，包含了聊天记录和聊天对象。

# 部署视图

部署方式和典型的web应用没有什么差别，在此不多叙述。