交大交交——交大二手交易信息平台

软件架构文档

版本 <1.0>

[注：用方括号括起来并以蓝色斜体（样式=InfoBlue）显示的文本，它们用于向作者提供指导，在发布此文档之前应该将其删除。按此样式输入的段落将被自动设置为普通样式（样式=Body Text）。]

[要定制 Microsoft Word 中的自动字段（选中时显示灰色背景），请选择 File>Properties，然后将 Title、Subject 和 Company 等字段替换为此文档的相应信息。关闭该对话框后，通过选择 Edit>Select All（或 Ctrl-A）并按 F9，或只是在字段上单击并按 F9，可以在整个文档中更新自动字段。对于页眉和页脚，这一操作必须单独进行。按 Alt-F9，将在显示字段名称和字段内容之间切换。有关字段处理的详细信息，请参见 Word 帮助。]

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| <日/月/年> | <x.x> | <详细信息> | <姓名> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

1. 简介 3

1.1 目的 3

1.2 参考资料 3

2. 用例视图 3

3. 逻辑视图 3

3.1 概述 3

3.2 在构架方面具有重要意义的设计包 3

4. 进程视图 3

5. 部署视图 3

6. 实现视图 3

7. 数据视图（可选） 3

8. 核心算法设计（可选） 3

软件架构文档 （简化版）

# 简介

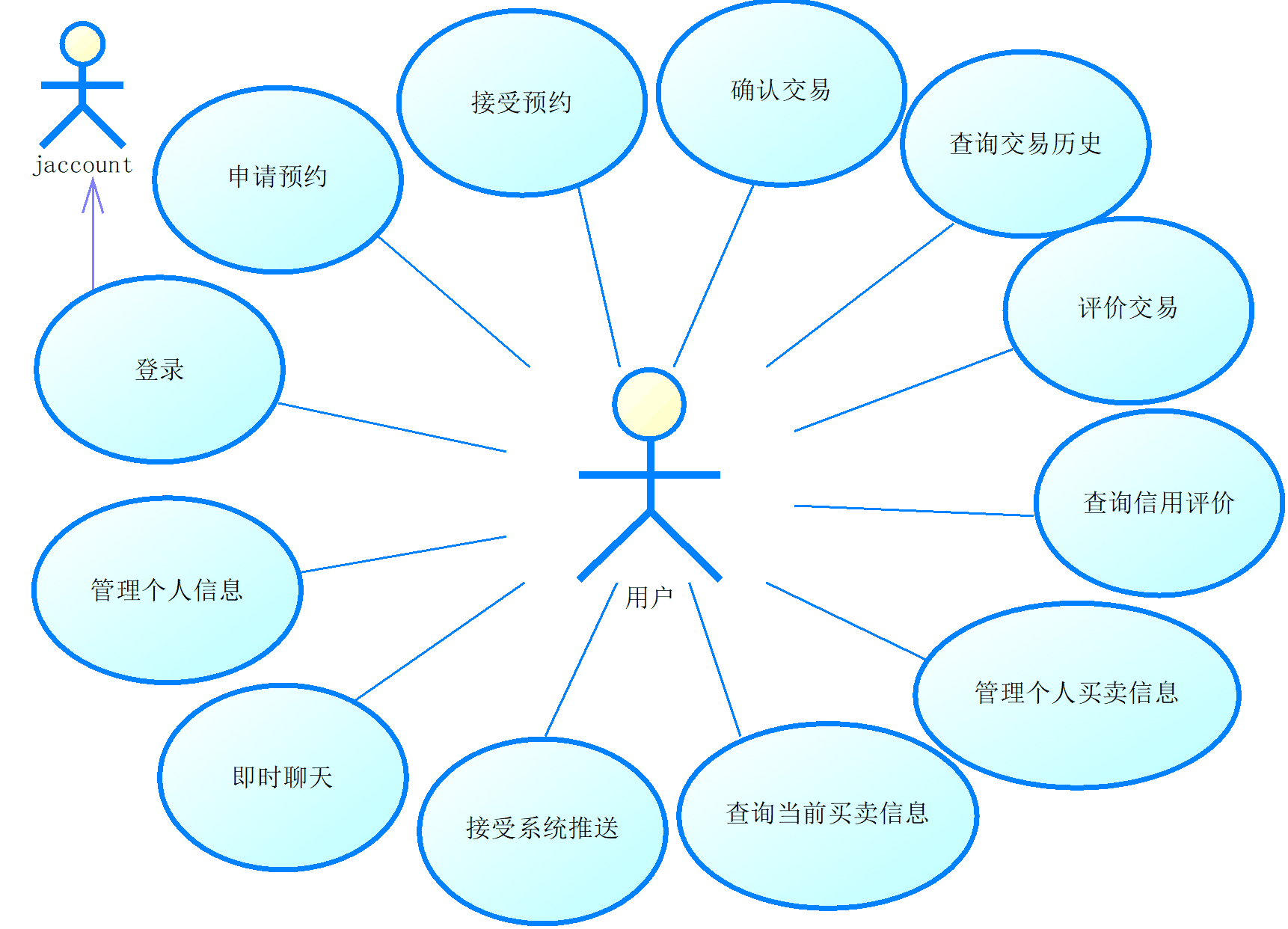
## 目的

本文档将从构架方面对“交大二手交易信息平台”进行综合概述，其中会使用用例视图、逻辑视图、部署视图、实现视图、数据视图来描述系统的各个方面。它用于记录并表述已对系统的构架方面做出的重要决策。

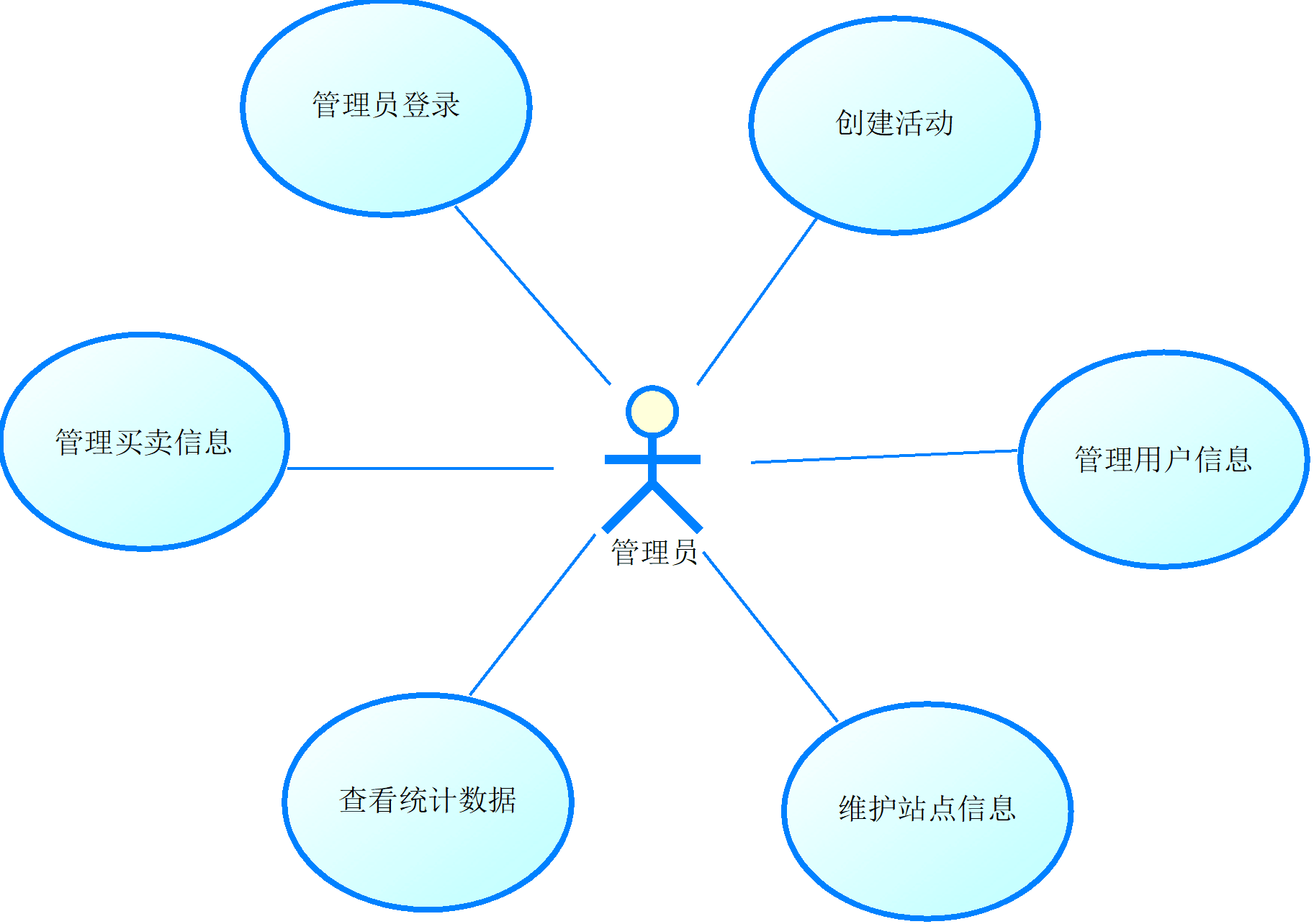
## 参考资料

沈备军, 陈昊鹏, 陈雨亭. 软件工程原理[M]. 高等教育出版社, 2013.

# 用例视图



**图2.1 用户用例图**

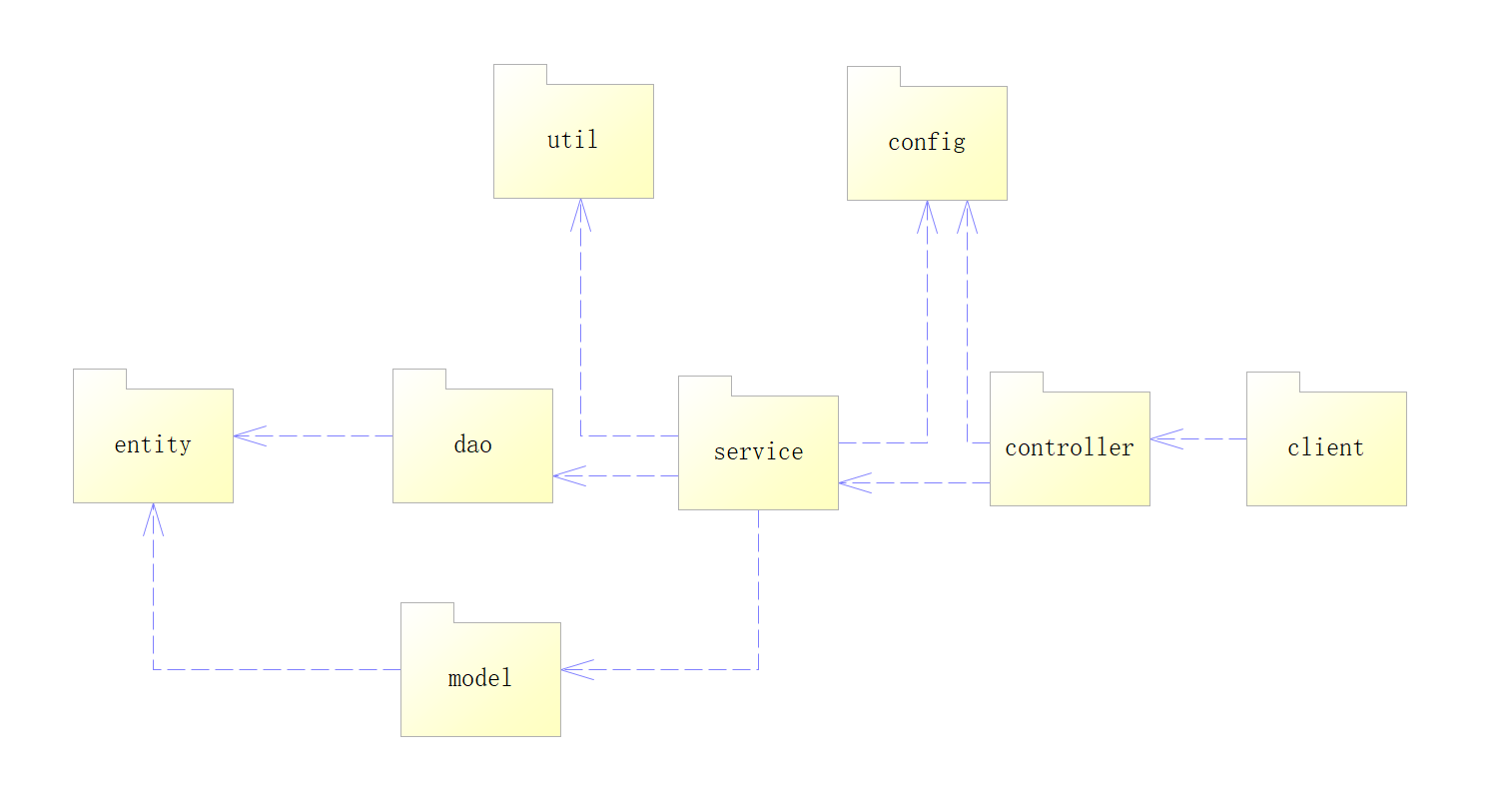


**图2.2 管理员用例图**

# 逻辑视图

## 概述

系统可以分解为客户端与服务端，服务端可进一步分解为接口层、业务层、数据访问层、实体层等。接口层即controller包，服务层即service包，数据层即dao包与model包，实体层即entity包。util包与config包作为辅助工具存在。



**图3.1.1 逻辑视图**

## 在构架方面具有重要意义的设计包

* + 1. **config包**

config包储存系统配置，包括视图解析配置、跨域配置、redis相关配置、系统拦截器配置等。

* + 1. **util包**

util包由常用工具类组成，包括日期转换工具、班次信息转换工具等。

* + 1. **entity包**

entity包由OR映射的entity类

* + 1. **model包**

model包由数据传输dto类、返回类型Response类及其子类组成。

* + 1. **dao包**

dao包数据库访问层，负责与数据库交互。每一个OR映射类，均有相应的repository类进行访问。

* + 1. **service包**

由系统功能抽象而得的service类组成，实现业务逻辑。包括账户服务、内容服务、上传服务等。

* + 1. **controller包**

controller包由MVC模式中controller类组成，响应用户请求。针对App客户端、Web端各有相应API接口。

* + 1. **client子系统**

client子系统包括App客户端、Web后台管理端。

# 进程视图

# 



**图4.1 进程视图**

如图所示，App客户端进程、Web后台管理客户端与Web服务器进程的通信，是通过HTTP协议、符合Restful风格的API进行远程通信。而Web服务器与三种数据库管理进程的通信，是通过一台机器的本地TCP协议进行端口间的相互通信部署视图

请求的产生进程是用户访问系统使用的App客户端以及管理员管理系统时所用的后台管理Web客户端，它们分别运行在与服务器不同的手机或计算机上，因此需要单独进程。

请求的接收响应进程是由应用服务器进行的，考虑到性能，整个接收响应进程可以拆分为三部分任务，并且三者均通过本地的接口进行交互。

最后是两个数据库管理系统进程，分别为MySQL服务进程、MongoDB服务进程，负责管理存储在数据库中的数据。

# 部署视图



**图5.1 部署视图**

App客户端运行在手机设备上，Web后台管理系统运行在桌面PC浏览器上。应用服务器上部署了Web服务器、MySQL数据库、MongoDB数据库。后台管理系统、App客户端与服务端通过互联网连接，应用服务器内部通过本地网络通信（注：第三方服务作为外部系统未标注在部署视图中）。

# 实现视图



**图6.1 实现视图**

# 数据视图（可选）

1. 数据库设计采用Maria关系型数据库和Mongo非关系型数据库的结合。其中Maria关系型数据库中存储结构化的数据，例如用户、信息发布、商品、标签、交易、评分，包含了主要的全部数据存储；Mongo非关系型数据库中存储非结构化数据，包括聊天记录、评论、多媒体信息、申诉。
2. Maria关系型数据库

如图所示：



**图7.1 CDM图**



**图7.2 PDM图**

SQL建表语句见附录1。

1. Mongo非关系型数据库

* 聊天记录：MariaDB中负责存储了产生聊天记录的两个用户，但真实的记录存储在MongoDB中，从MariaDB中找到chat\_log表中的chat\_content\_id，从而在mongoDB中检索相关的聊天记录信息
* 评论：MariaDB中只存储了发生评论的用户和相应交易，具体的评论信息都存储在MongoDB中，同样使用comment表中的comment\_content\_id检索其他的所有评论。
* 多媒体信息：MariaDB中存储的content\_id，通过content\_id访问MongoDB中的多媒体信息，包括图片视频等多种类型，使用GridFS分片存储。
* 申诉：MariaDB中负责存储了产生申诉的主体用户和客体用户，但复杂的申诉信息存储在MongoDB中，从MariaDB中找到complaint表中的complaint\_content\_id，从而在mongoDB中检索相关的申诉信息

## 附录一

/\*==============================================================\*/

/\* DBMS name: MySQL 5.0 \*/

/\* Created on: 2019/7/2 10:57:32 \*/

/\*==============================================================\*/

drop table if exists Good;

drop table if exists Relationship\_5;

drop table if exists chat\_log;

drop table if exists comment;

drop table if exists comment\_picture;

drop table if exists complaint;

drop table if exists evaluate;

drop table if exists good\_picture;

drop table if exists "release";

drop table if exists tag;

drop table if exists transaction;

drop table if exists user;

/\*==============================================================\*/

/\* Table: Good \*/

/\*==============================================================\*/

create table Good

(

release\_id int not null,

good\_name varchar(64) not null,

good\_description varchar(1024) not null,

primary key (release\_id)

);

/\*==============================================================\*/

/\* Table: Relationship\_5 \*/

/\*==============================================================\*/

create table Relationship\_5

(

release\_id int not null,

tag\_id int not null,

primary key (release\_id, tag\_id)

);

/\*==============================================================\*/

/\* Table: chat\_log \*/

/\*==============================================================\*/

create table chat\_log

(

chat\_log\_id int not null,

user\_id int not null,

use\_user\_id int not null,

mongo\_chat\_id varchar(1024) not null,

primary key (chat\_log\_id)

);

/\*==============================================================\*/

/\* Table: comment \*/

/\*==============================================================\*/

create table comment

(

comment\_id varchar(1024) not null,

user\_id int not null,

release\_id int not null,

content varchar(1024) not null,

mongo\_comment\_id varchar(1024),

primary key (comment\_id)

);

/\*==============================================================\*/

/\* Table: comment\_picture \*/

/\*==============================================================\*/

create table comment\_picture

(

comment\_picture\_id int not null,

comment\_id varchar(1024) not null,

comment\_picture\_url varchar(1024) not null,

primary key (comment\_picture\_id)

);

/\*==============================================================\*/

/\* Table: complaint \*/

/\*==============================================================\*/

create table complaint

(

complaint\_id int not null,

user\_id int,

use\_user\_id int,

content varchar(1024) not null,

primary key (complaint\_id)

);

/\*==============================================================\*/

/\* Table: evaluate \*/

/\*==============================================================\*/

create table evaluate

(

evaluate\_id int not null,

user\_id int not null,

score int not null,

primary key (evaluate\_id)

);

/\*==============================================================\*/

/\* Table: good\_picture \*/

/\*==============================================================\*/

create table good\_picture

(

good\_picture\_id int not null,

release\_id int,

good\_picture\_url varchar(1024) not null,

primary key (good\_picture\_id)

);

/\*==============================================================\*/

/\* Table: "release" \*/

/\*==============================================================\*/

create table "release"

(

release\_id int not null,

user\_id int not null,

Goo\_release\_id int,

tra\_release\_id int,

is\_saled bool not null,

release\_time datetime not null,

valid\_time datetime not null,

is\_appointed bool not null,

is\_finished bool not null,

primary key (release\_id)

);

/\*==============================================================\*/

/\* Table: tag \*/

/\*==============================================================\*/

create table tag

(

tag\_id int not null,

tag\_name varchar(16) not null,

primary key (tag\_id)

);

/\*==============================================================\*/

/\* Table: transaction \*/

/\*==============================================================\*/

create table transaction

(

release\_id int not null,

user\_id int not null,

create\_time datetime not null,

primary key (release\_id)

);

/\*==============================================================\*/

/\* Table: user \*/

/\*==============================================================\*/

create table user

(

user\_id int not null,

user\_name varchar(16) not null,

password varchar(16) not null,

portrait\_url varchar(1024) not null,

telephone char(11) not null,

primary key (user\_id)

);

alter table Good add constraint FK\_Relationship\_6 foreign key (release\_id)

references "release" (release\_id) on delete restrict on update restrict;

alter table Relationship\_5 add constraint FK\_Relationship\_7 foreign key (release\_id)

references Good (release\_id) on delete restrict on update restrict;

alter table Relationship\_5 add constraint FK\_Relationship\_8 foreign key (tag\_id)

references tag (tag\_id) on delete restrict on update restrict;

alter table chat\_log add constraint FK\_Relationship\_14 foreign key (user\_id)

references user (user\_id) on delete restrict on update restrict;

alter table chat\_log add constraint FK\_Relationship\_15 foreign key (use\_user\_id)

references user (user\_id) on delete restrict on update restrict;

alter table comment add constraint FK\_Relationship\_12 foreign key (user\_id)

references user (user\_id) on delete restrict on update restrict;

alter table comment add constraint FK\_Relationship\_13 foreign key (release\_id)

references transaction (release\_id) on delete restrict on update restrict;

alter table comment\_picture add constraint FK\_Relationship\_11 foreign key (comment\_id)

references comment (comment\_id) on delete restrict on update restrict;

alter table complaint add constraint FK\_Relationship\_17 foreign key (user\_id)

references user (user\_id) on delete restrict on update restrict;

alter table complaint add constraint FK\_Relationship\_18 foreign key (use\_user\_id)

references user (user\_id) on delete restrict on update restrict;

alter table evaluate add constraint FK\_Relationship\_16 foreign key (user\_id)

references user (user\_id) on delete restrict on update restrict;

alter table good\_picture add constraint FK\_Relationship\_10 foreign key (release\_id)

references Good (release\_id) on delete restrict on update restrict;

alter table "release" add constraint FK\_Relationship\_4 foreign key (tra\_release\_id)

references transaction (release\_id) on delete restrict on update restrict;

alter table "release" add constraint FK\_Relationship\_5 foreign key (Goo\_release\_id)

references Good (release\_id) on delete restrict on update restrict;

alter table "release" add constraint FK\_Relationship\_9 foreign key (user\_id)

references user (user\_id) on delete restrict on update restrict;

alter table transaction add constraint FK\_Relationship\_1 foreign key (user\_id)

references user (user\_id) on delete restrict on update restrict;

alter table transaction add constraint FK\_Relationship\_3 foreign key (release\_id)

references "release" (release\_id) on delete restrict on update restrict;

# 核心算法设计（可选）

暂无