<项目名称>

软件架构文档

版本 <1.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| <13日/12月/2022年> | <1.0> | <第一版软件架构文档> | <第二组全体成员> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 4

1.2 参考资料 4

2. 用例视图 4

3. 逻辑视图 4

3.1 概述 4

3.2 在构架方面具有重要意义的设计包 4

4. 进程视图 4

5. 部署视图 4

6. 实现视图 5

7. 技术视图 5

8. 数据视图 5

9. 核心算法设计 5

10. 质量属性的设计 5

软件架构文档

# 简介

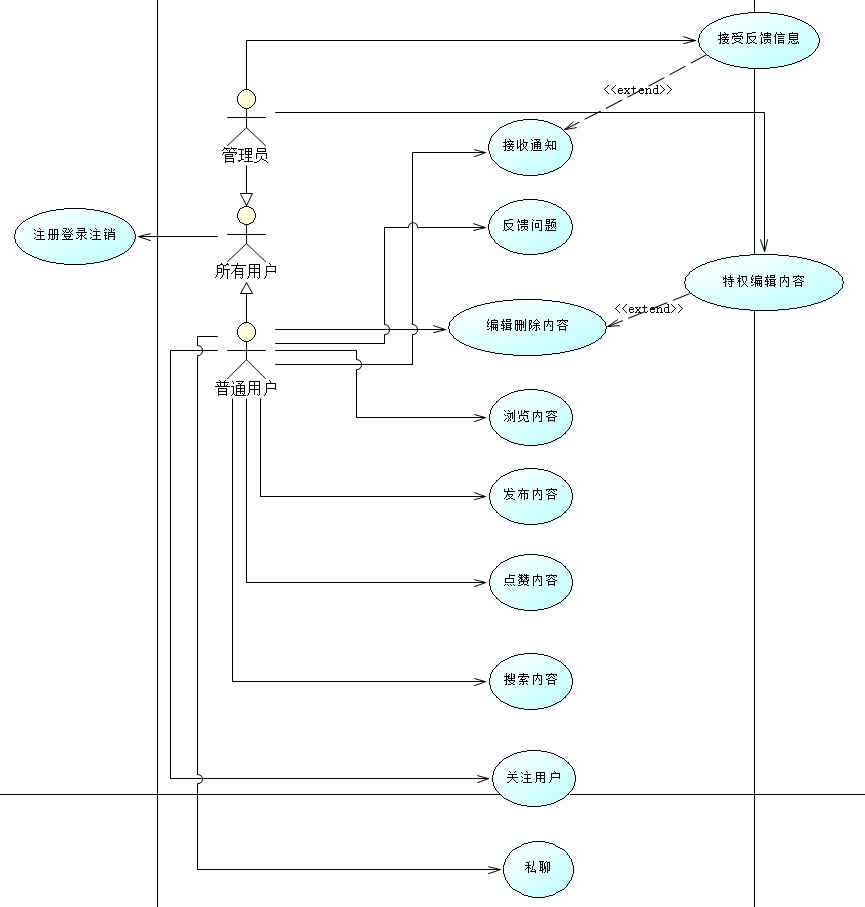
## 目的

本文档将从构架方面对“Hobbitat(校园兴趣分享社区)”项目进行综合概述，其中会使用用例视图、逻辑视图、部署视图构等来描述软件的架构设计。

## 参考资料

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

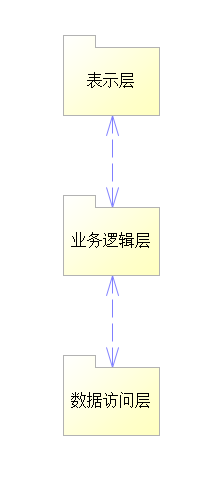
# 用例视图



# 逻辑视图

## 概述

本应用的逻辑架构采用3 Tiers风格，分为应用层（表示层）、业务逻辑层、数据访问层三层。



**应用层**

负责显示用户界面，与用户进行交互，将用户请求发送给业务逻辑层，处理并展示业务逻辑层返回的结果。在本项目中包括登录注册，基础页面展示，发布内容，个人中心等页面。

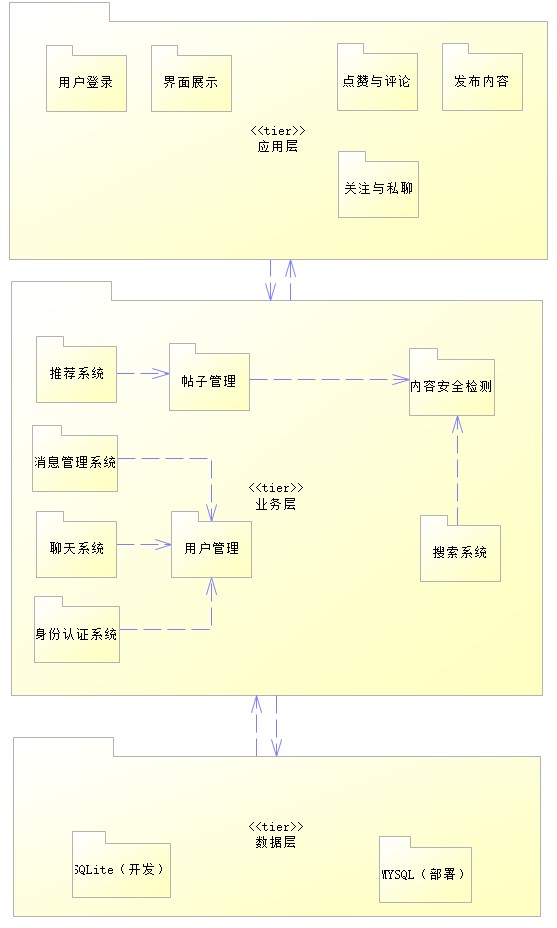
**业务逻辑层**

在该层中，采用业务逻辑（一组特定的业务规则）处理从应用层收集到的信息， 有时也会处理数据层中的其他信息 。 业务逻辑层还可以添加、删除或修改数据层中的数据。

**数据层：**

存储和管理应用程序所处理的信息。 在本项目中主要用到SQLite 和 MySQL。

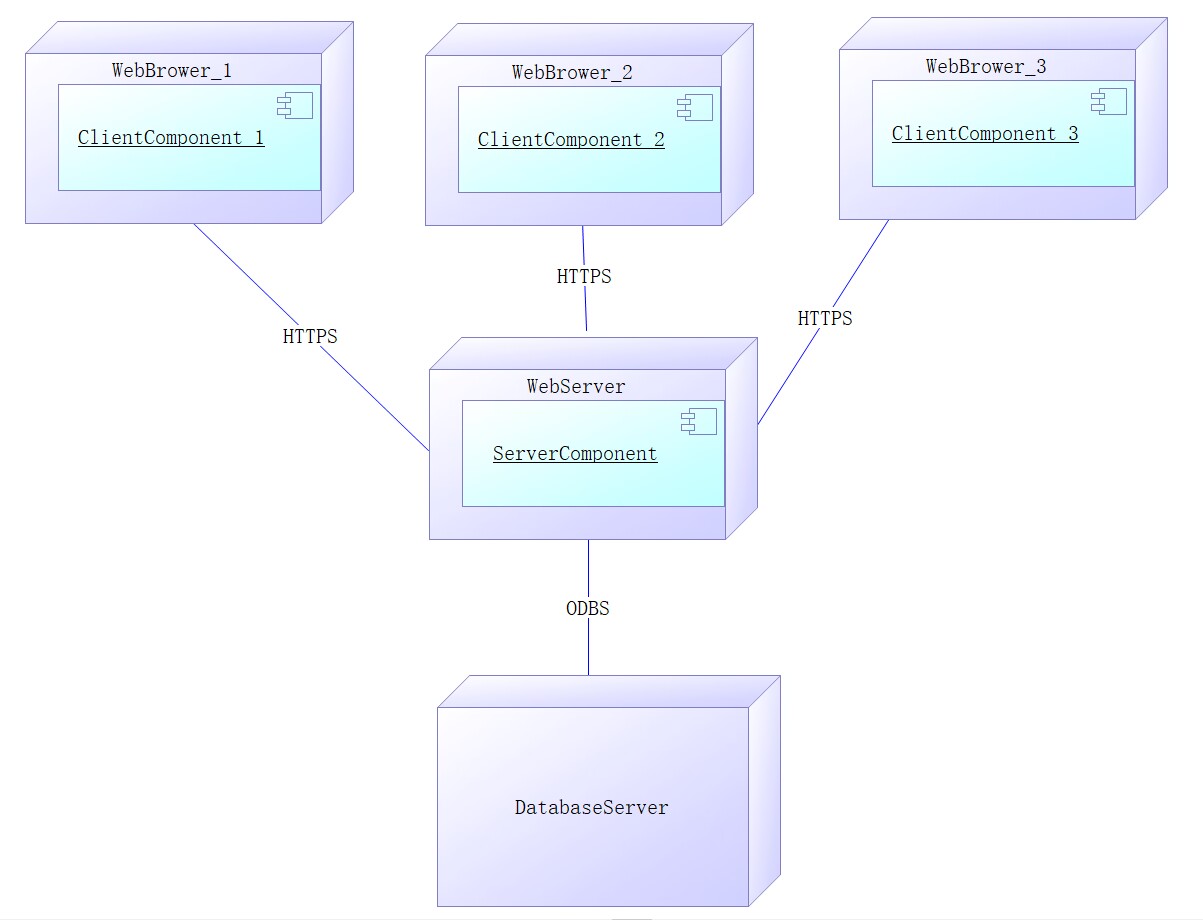
**下图为逻辑视图概览**



# 进程视图

本项目使用Flask 框架进行进程的组织和通信，借助该框架更容易地实现了进程之间的主要通信模糊，例如消息传递，中断和会和。

# 部署视图



5.1. Web Browser

用户通过Web浏览器访问网站应用，浏览器（客户端）通过互联网使用https协议向Web服务器发送任务请求并将得到的结果进行显示。

5.2. Web Server

Web服务器负责接受和处理Web浏览器发送的任务请求并分配任务给数据库服务器处理，接收数据库服务器对任务的处理结果并将响应结果返回给Web浏览器。

在本项目中使用Flask 框架来接受和处理请求，发送响应。

5.3. Data Server

数据库服务器是用来储存用户各种信息，它通过ODBC协议来和Web服务器交互信息。在本项目中开发中使用SQLite, 在部署时使用MySQL。

# 实现视图

本项目的具体实现分为如下几个主要模块

1. 数据处理模块

- data.db 存储项目用到的数据

- dataloader 加载项目使用的所有数据

- model 管理和实现项目的所有数据库模型

2. 核心算法模块

-blueprints

- chat 处理用户之间的聊天逻辑

- posts 处理与发帖，浏览，点赞，评论等事务

- user 处理用户相关的模块

- ...

- utils 实现推荐系统等算法

- abstract\_factory 抽象工厂

3. 页面处理和渲染模块

- static 存储前端相应的代码

-css

-js

- ...

- templates 存储主要页面的对应模板

- chat

- errors

- posts

- user

4. 软件配置模块

- setting 处理软件的相关配置

# 技术视图

7.1.编程语言： Python

7.2.开发工具： Visual Studio Code

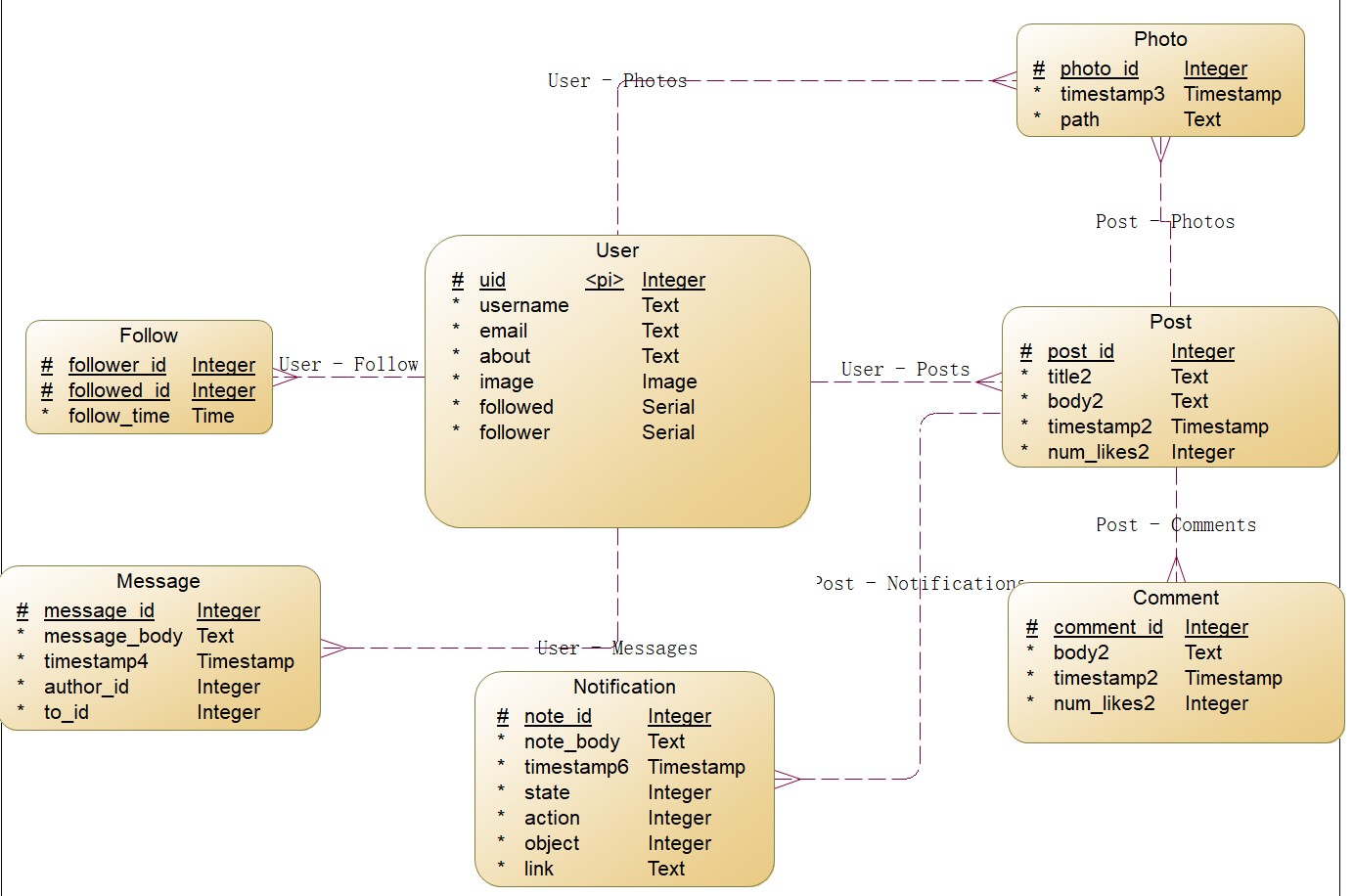
7.3.框架： Flask

7.4.数据库： SQLite, MYSQL

# 数据视图

8.1.持久化机制的选择：关系数据库SQLite、MYSQL

8.2.数据视图的实例：



# 核心算法设计

* **推荐算法**：

考虑到项目目前积累的数据量的不足，难以使用深度学习等相关技术来实现推荐系统。因此本项目中使用了协同过滤的推荐算法。

该算法的优点在于对于数据量和计算量的需求不大，并且推荐功能易于实现，可解释性高。

通过基于用户和基于产品的协同过滤，项目中实现了对关注用户的推荐，以及对相应帖子的推荐

* **Bridge模式的通知系统**

将“动作”视为抽象，而“作用对象”视为实现。Abstract Action是我们操作通知等动作的对象

Abstract Action：根据动作的不同，设置不同的数据更新方式和通知格式。相应信息通过调用Object的统一接口来获取

Object:根据作用对象的不同，提供相应的数据库更新位置、提示文字、跳转链接等信息

# 质量属性的设计

* **可扩展性**

可扩展性的主要实现方式包括垂直扩展与水平扩展，垂直扩展指在同一个逻辑单位添加资源以增加容量。开始的设置非常基础，可能就是一台web服务器和一台数据库服务器。当机器性能不足时，用更大的机器替换它。新机器能力不足时，用另外一台机器替换它。这台机器能力也不足时，就买一台更加强大的机器。如此反复；水平扩展不同于垂直扩展的地方在于，增长时不需要超级强劲的机器，而只需要很多常规的机器。从一台常规的机器开始，其能力不足后添加第二台。然后第三台，第四台等。增加多个逻辑单元资源并且使他们作为一个整体在工作。大多数的集群解决方案，比如分布式文件系统，负载均衡都是通过横向扩展技术来进行的。

由于扩展设备的数量（如实现分布式软件）不太现实，本组对软件性能的扩展选择垂直扩展，在实现一个具有基本功能的的Web服务器以及数据库服务器的基础上，一步步增加新的功能以及优化现有功能，逐渐搭建出一款功能完备的社区软件。同时将功能接口化，便于之后进一步做垂直拓展。

扩展方式：

* 业务维度扩展：

在开发初期，对业务的发展方向进行及时有效地预判，对于预判中可能被调用很多次的功能进行包装处理，并设置方便的接口，从而简化开发的流程，加强可扩展性。需要注意的是，要警惕过度设计；不能每个设计点都考虑可扩展性；所有的预测都存在不正确的可能。

* 技术维度扩展

采取的方案是将“变化”封装在一个“变化层”，将不变的部分封装在一个独立的“稳定层”。

* **可靠性**

在《GB/T 11457-2006 信息技术 软件工程术语》中，对软件可靠性的定义如下：

a)在规定条件下、在规定时间内，软件不引起系统失效的概率。该概率是系统输入和系统使用的函数，也是软件中存在的缺陷的函数。系统输入将确定是否会遇到已存在的缺陷（如果有缺陷存在的话）；

b)在规定的时间周期内所述条件下程序执行所要求的功能的能力。

我们在软件开发过程中比较看重软件的实时性，对功能的实现都不断优化消耗时间，从而达到及时性的指标。同时保证了数据传输的安全性和可靠性，没有检测到错误数据的产生。

同时我们设置了一些exception case，当我们的网站遇到一些错误输入的信息时，例如一些会导致404/500等错误的的时候，我们制作的404等error网页会提示用户做出了什么错误行为，并且让用户修改。

* **易用性**

一种简单的定义易用性的方式是，易用性是指消费方使用系统提供的资源（能力或服务）的便利程度。

易用性关注的是对用户来说完成某个期望任务的容易程度和系统所提供的用户支持的种类。可以将易用性分为如下几个方面：易理解，易操作，易学习，效率性，出错率，满意度，易分享等。以上的7点可以概括为：让软件系统易见、易学、易用。以技术为基础，以用户的体验为中心，提升软件易用性和界面友好性。

我们的项目在设计上很关注软件的易用性：

* 插入了很多图形化的icon方便用户理解每个功能，可以在没有预先经过指导的情况下去使用。
* 在用户浏览过程中，我们会在前端及时收集用户的各种行为信息，同时利用一个session记录用户的个人信息，使系统及时对用户的信息作出反馈，而不用临时收集需要的信息从而导致一些时间上的浪费以及对用户的使用体验的破坏。
* 在搜索功能上，我们提供了分类搜索这种搜索方式，使用户面对多数据源搜索问题时，有更好的使用体验，更加的方便快捷。
* 我们落地了基于协同过滤算法的推荐系统，为用户提供与用户有相似行为的用户以及用户可能感兴趣的话题，为用户在社区更好地探索创造了更方便的条件。