大学生创业平台的开发与系统优化策略研究

武斌

电子系 中国海洋大学

Department electronic engineering
Ocean University Of China

2017 年 5 月 20 日

概述

1. 选题背景
2. 研究内容
3. 研究方案
4. 进展情况

选题背景

课题来源

- 创业过程中负责创游记平台的系统开发、维护和 WEB 开发
- 海信实习过程中参与小微系统开发、环境部署与优化的工作

选题背景

课题来源

- 创业过程中负责创游记平台的系统开发、维护和 WEB 开发
- 海信实习过程中参与小微系统开发、环境部署与优化的工作

选题依据和背景情况

- [5] 进行创业实践的大学生不断增多，开发一个大学生创新创业的平台，提供创业指导、成果展示、投融资推荐和就业机会的推荐等服务，针对平台用户的创业行为进行分析与研究
- WEB 产品不断增加，很多产品在系统优化方面做的比较少，无论是系统的持续集成还是系统负载的平衡以及性能的优化等方面，本课题将以平台为案例进行系统优化等方面的研究

选题背景

课题研究目的

- 基于 WEB 技术开发一款服务于大学生创新创业的网络平台
- 研究系统开发过程中的代码、持续集成、负载等方面的优化策略

选题背景

课题研究目的

- 基于 WEB 技术开发一款服务于大学生创新创业的网络平台
- 研究系统开发过程中的代码、持续集成、负载等方面的优化策略

理论意义和研究价值

- 从平台功能而言，希望可以帮助大学生进行创新创业方面的尝试和实践，帮助投资机构寻找具有商业价值的创业项目或者创业者
- 从研究角度而言，希望可以通过开发和使用更好的技术来提升系统在使用过程中的代码、部署、测试以及负载等方面的性能，研究一体化的优化策略

研究内容

1. 创游记平台的开发
2. 系统优化策略研究

创游记平台的开发

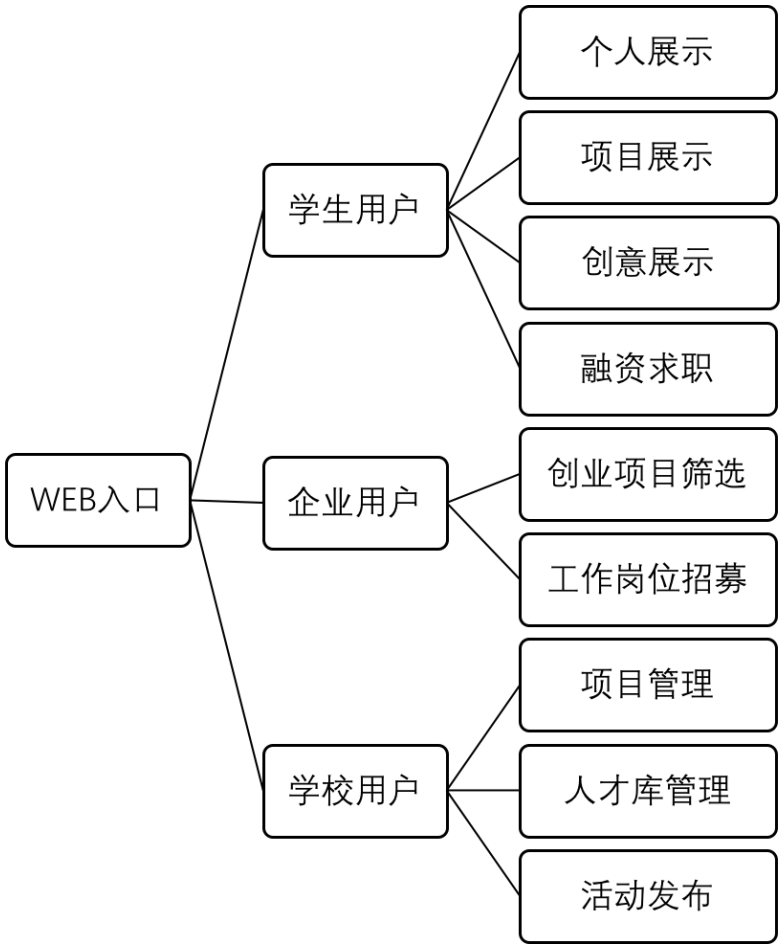
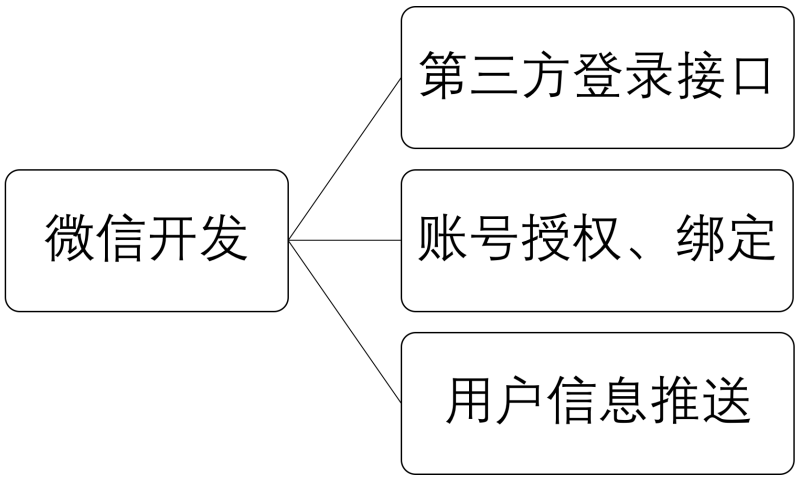


图: 平台功能

创游记平台的开发



开发目标

1. 针对不同用户完成不同功能模块的开发
2. 实现微信端的浏览、访问、授权
3. 实现针对定向用户的个性化推荐

针对平台的优化策略研究

代码优化

- 整合平台开发过程中过多代码复用的功能到公共函数库，实现模块化的开发
- 将特定功能写成系统插件或者框架，提升代码运行效率

推荐系统开发

- 探索基于平台用户数据的推荐系统开发，实现学生用户的融资求职机会推荐和企业用户的创业项目推荐

针对系统的优化策略研究

持续集成环境构建

- 探索在本地环境搭建一个持续集成环境，实现一个自动构建过程，包括自动编译、分发、部署和测试等功能

针对系统的优化策略研究

持续集成环境构建

- 探索在本地环境搭建一个持续集成环境，实现一个自动构建过程，包括自动编译、分发、部署和测试等功能

系统缓存开发

- 探索系统缓存对于 WEB 系统性能的影响
- 基于本系统研究系统的缓存机制和策略
- 基于本系统进行缓存机制的开发和性能测试

针对系统的优化策略研究

持续集成环境构建

- 探索在本地环境搭建一个持续集成环境，实现一个自动构建过程，包括自动编译、分发、部署和测试等功能

系统缓存开发

- 探索系统缓存对于 WEB 系统性能的影响
- 基于本系统研究系统的缓存机制和策略
- 基于本系统进行缓存机制的开发和性能测试

搜索优化

- 研究在保证搜索的实时、稳定、可靠和快速基础上，降低系统负载的方式
- 探索基于分布式搜索的系统接口开发和性能测试

研究方案

1. 创游记平台的开发
2. 平台代码优化
3. 推荐系统研究与开发
4. 自动化测试部署方案研究
5. 系统缓存和优化方案研究

创游记平台的开发

基本环境搭建

- 搭建基于 Linux+Apache+MySQL+PHP/Python 的 WEB 开发环境
- 搭建本地的代码版本控制工具 GitLab

创游记平台的开发

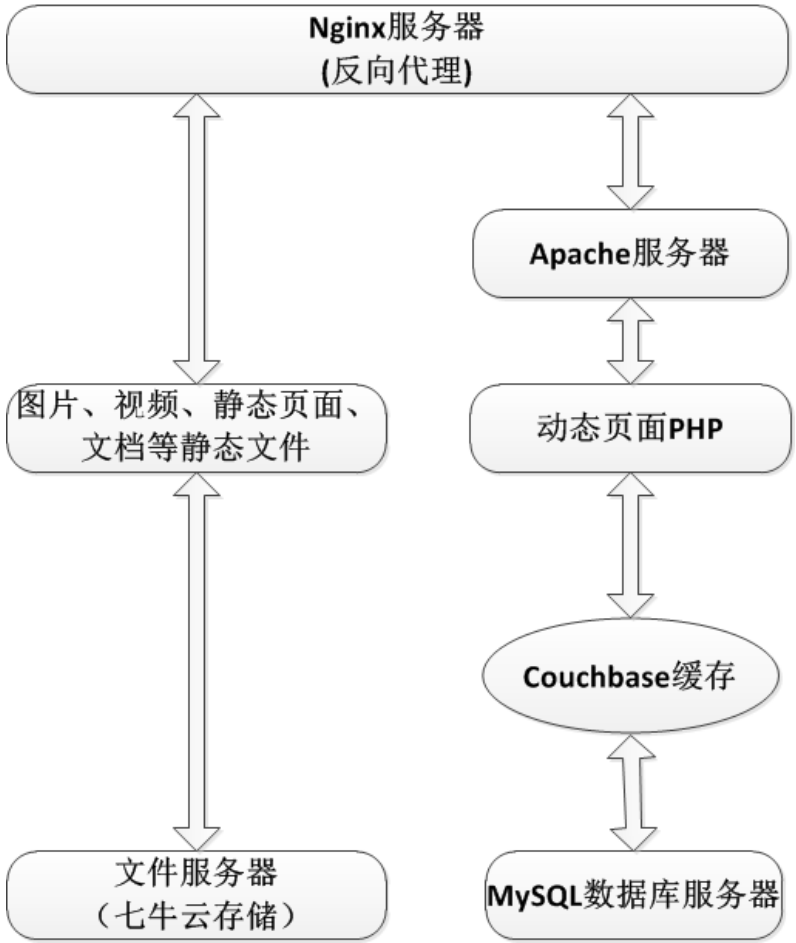
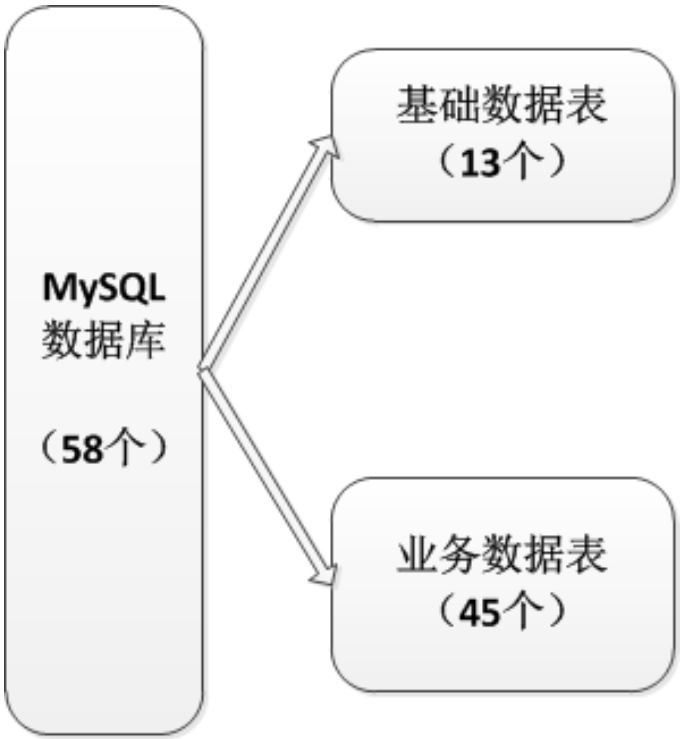


图: 平台基本框架

创游记平台的开发



数据表设计和维护

- 基础数据表主要包括系统的基本环境配置、插件配置、文档模型配置等
- 业务数据表主要包含用户、活动、项目、创意等数据

创游记平台的开发

平台代码优化

- 模块化表单功能开发，通过开发 FormBuilder 和 ListBuilder 两个模块以及在数据库中配置界面字典和物理字典，实现表单的统一配置和显示，避免了在每一个页面都复写 form 的复杂操作
- 开发 JS 提交插件 AJAXPost 和 JAJXGet 实现数据提交和获取，避免在每一个前端页面和后台控制器中都配置数据变量和返回格式的问题
- 开发 JS 上传插件 Uploader，实现文件的上传、图片的上传和修改等功能以及七牛云存储的保存

[6] 推荐系统研究与开发

常用的推荐系统算法

1. 基于内容的推荐 – 推荐和用户过去喜欢项的内容相似的项
2. 协同过滤推荐 – 通过在用户的一系列行为中寻找特定模式来产生用户特殊推荐
3. 混合推荐 – 综合以上两种算法的优点同时抵消各自的缺点
4. 高级非传统的推荐 – 深度学习、上下文感知推荐等

[6] 推荐系统研究与开发

常用的推荐系统算法

1. 基于内容的推荐 – 推荐和用户过去喜欢项的内容相似的项
2. 协同过滤推荐 – 通过在用户的一系列行为中寻找特定模式来产生用户特殊推荐
3. 混合推荐 – 综合以上两种算法的优点同时抵消各自的缺点
4. 高级非传统的推荐 – 深度学习、上下文感知推荐等

本课题使用的推荐算法和框架

- ⊛ 本课题主要使用协同过滤推荐算法中的基于用户的协同过滤算法
- ⊛ [8] 本课题使用的框架是基于 Python 的 Crab 推荐系统引擎

推荐系统研究与开发

协同过滤推荐算法 (Collaborative Filtering Recommendation)

- 基于用户的协同过滤 (User-Based CF)
  找到和目标用户兴趣相似的用户集合，然后给目标用户推荐这个集合的用户喜欢的物品
- 基于物品的协同过滤 (Item-Based CF)
  给目标用户推荐与他喜欢的物品相似度较高高的物品

推荐系统研究与开发

协同过滤推荐算法 (Collaborative Filtering Recommendation)

- 基于用户的协同过滤 (User-Based CF)
  找到和目标用户兴趣相似的用户集合，然后给目标用户推荐这个集合的用户喜欢的物品
- 基于物品的协同过滤 (Item-Based CF)
  给目标用户推荐与他喜欢的物品相似度较高高的物品

选择基于用户的协同过滤的原因

- 适合用户较少的场合，否则用户相似度矩阵计算代价很大
- 适合时效性较强，用户个性化兴趣不太明显的领域

基于用户的协同过滤算法的主要公式和步骤

- [1] 基于用户的协同过滤的关键在于计算用户与用户之间的兴趣相似度，主要使用余弦相似度来计算：

$$\omega_{\mu\nu} = \frac{|\mathrm{N}(\mu) \cap \mathrm{N}(\nu)|}{\sqrt{|\mathrm{N}(\mu) \parallel \mathrm{N}(\nu)|}}$$

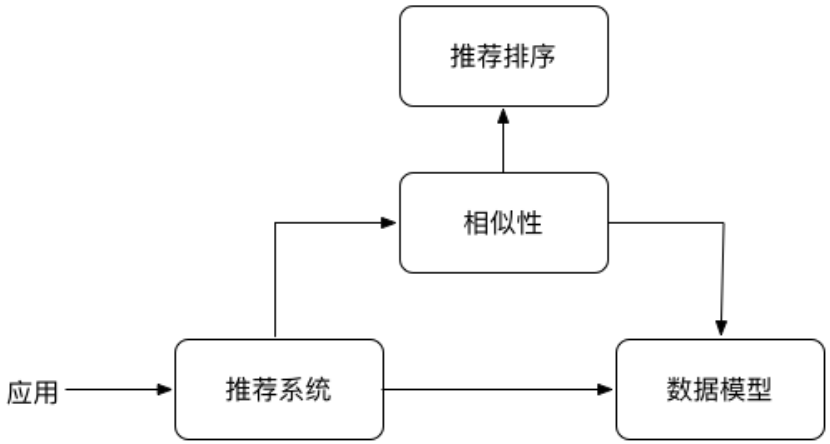
$\omega_{\mu\nu}$ 代表用户 μ 与 ν 之间的兴趣相似度，$\mathrm{N}(\mu)$ 表示用户 μ 曾经喜欢过的物品集合，$\mathrm{N}(\nu)$ 表示用户 ν 曾经喜欢过的物品集合

- 根据上述描述，可以有如下算法步骤
  1. 建立物品-用户的倒排表 (Inverted Index)
  2. 计算用户与用户之间的共享矩阵 $\mathrm{C}[\mu][\nu]$，表示用户 μ 与 ν 喜欢相同物品的个数
  3. 计算用户与用户之间的相似度矩阵 $\omega[\mu][\nu]$，根据上述相似度计算公式计算
  4. 用上面的相似度矩阵来给用户推荐和他兴趣相似的用户喜欢的物品

推荐系统研究与开发

Crab 推荐系统引擎

- Crab 是基于 Python 开发的开源推荐软件，其中实现的方法有 item 和 user 的协同过滤，其他的一些算法在开发中
- 基于 scikit-learn 库，scikit-learn 库是 Python 下的一个机器学习库，建立在 NumPy，SciPy 和 matplotlib 模块之上，可以实现很多机器学习的算法，为用户提供各种机器学习算法接口，让用户简单、高效地进行数据挖掘和数据分析



持续集成方案研究

关于持续集成

持续集成是一种软件开发实践，通过频繁地（一天多次）将代码集成到主干。每次集成都通过自动化的构建（包括编译，发布，自动化测试）来验证，从而尽早地发现集成错误[2]。

持续集成方案研究

关于持续集成

持续集成是一种软件开发实践，通过频繁地（一天多次）将代码集成到主干。每次集成都通过自动化的构建（包括编译，发布，自动化测试）来验证，从而尽早地发现集成错误[2]。

持续集成的好处[4]

1. 快速发现错误。每完成一点更新，就集成到主干，可以快速发现错误，定位错误也比较容易。
2. 防止分支大幅偏离主干。如果不是经常集成，主干又在不断更新，会导致以后集成的难度变大，甚至难以集成。

持续集成方案研究

关于持续集成

持续集成是一种软件开发实践，通过频繁地（一天多次）将代码集成到主干。每次集成都通过自动化的构建（包括编译，发布，自动化测试）来验证，从而尽早地发现集成错误[2]。

持续集成的好处[4]

1. 快速发现错误。每完成一点更新，就集成到主干，可以快速发现错误，定位错误也比较容易。
2. 防止分支大幅偏离主干。如果不是经常集成，主干又在不断更新，会导致以后集成的难度变大，甚至难以集成。

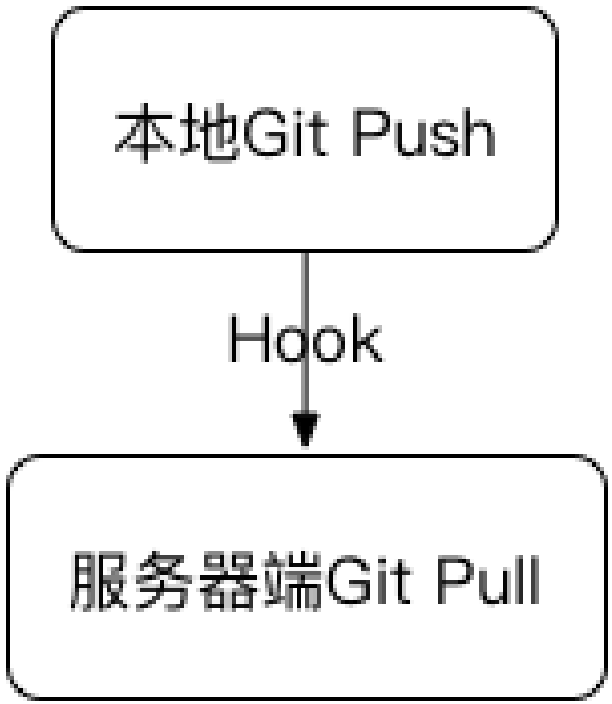
持续集成的目的

让产品可以快速迭代，同时还能保持高质量。它的核心措施是，代码集成到主干之前，必须通过自动化测试。只要有一个测试用例失败，就不能集成。

持续集成方案研究

本课题搭建持续集成的目的和研究

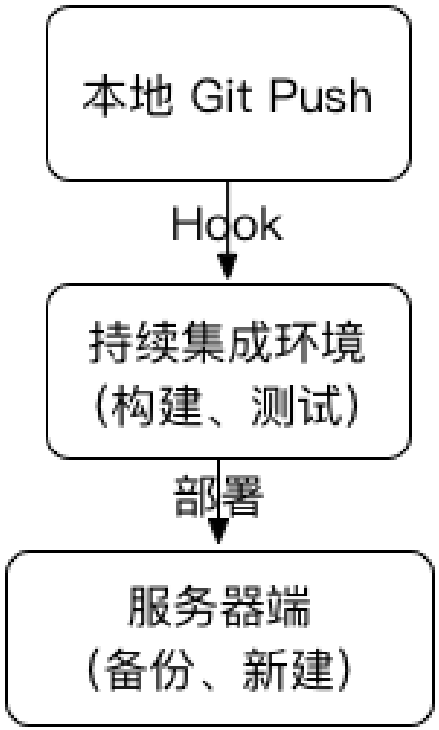
- 在使用持续集成方案之前，系统的部署流程是直接将代码通过 GitLab 的钩子 (hook) 将代码同步到服务器端的 WEB 目录下，没有进行自动化的构建、测试和部署。



持续集成方案研究

本课题搭建持续集成的目的和研究

- 使用持续集成方案之后，通过 GitLab 的钩子 (hook) 触发系统的持续集成工具，对代码进行合并、构建、测试和部署，出现重大问题时可以回滚。



持续集成方案研究

集成环境的搭建和配置

- 本地搭建 Jenkins 持续集成服务器，并且安装针对 PHP 的构建、测试插件
- 针对关键功能开发自动化测试脚本，在 Jenkins 集成时调用
- 开发 Hook 触发脚本和持续部署脚本，实现服务器端的的代码更新和文件备份

Jenkins 简介

- Jenkins 是一个开源项目，提供一种易于使用的持续集成系统，使开发者从繁杂的集成中解脱出来，专注于更为重要的业务逻辑实现上。
- 同时 Jenkins 能实时监控集成中存在的错误，提供详细的日志文件和提醒功能，还能用图表的形式形象地展示项目构建的趋势和稳定性。

系统缓存和优化方案研究

- 系统之前版本和现在版本使用的缓存机制对比

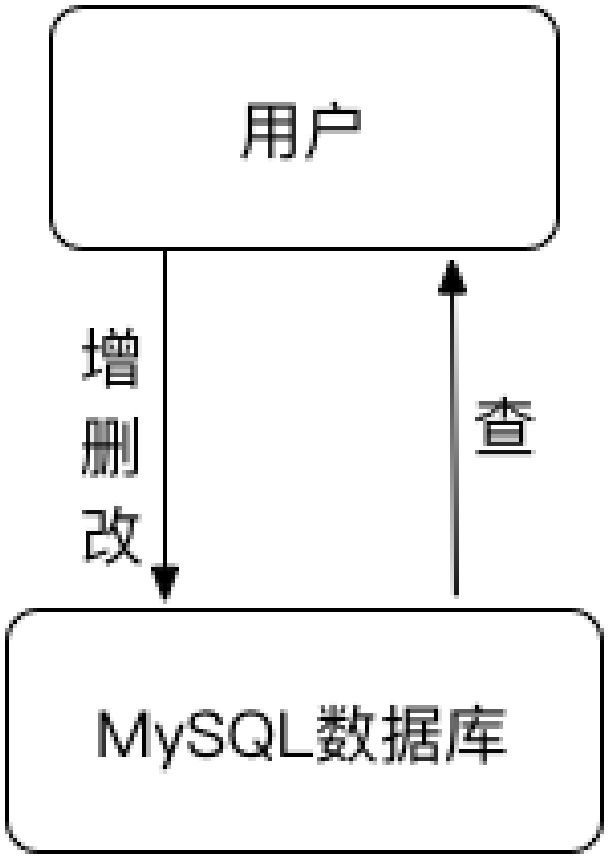


图: 之前数据操作模型

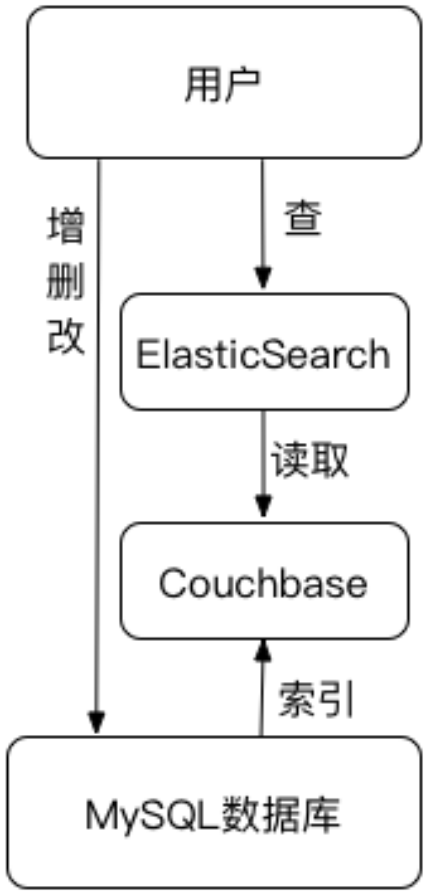


图: 当前数据操作模型

系统缓存和优化方案研究

Couchbase 缓存[7]

- Couchbase 是一个具有高性能、可扩展性和可用性强的 NoSQL 数据库引擎
- 使用 Couchbase 将系统的业务数据在系统启动时加载到缓存中，有效降低系统的负载并加快数据读取速度
- 使用 Couchbase 缓存机制可以降低系统的丢包率，增强系统的稳定性

ElasticSearch 搜索引擎[3]

- Elasticsearch 是一个基于 Apache Lucene(TM) 的开源搜索引擎
- 实时分析搜索引擎，每个字段都被索引并可被搜索
- RESTful 搜索引擎, 能够达到搜索实时、稳定、可靠和快速，支持通过 HTTP 请求，使用 JSON 进行数据索引[9]

系统缓存和优化方案研究

七牛云存储

- 将平台的图片和视频等媒体数据上传到七牛云存储
- 通过七牛的 CND 加速加快系统图片和视频的加载速度

WEB 缓存的意义和作用

- 减少网络带宽消耗
- 降低服务器压力
- 减少网络延迟，加快页面打开速度

进展情况

已完成

- 平台功能开发
- Jenkins 自动化测试、部署工具的配置和部署脚本开发
- Couchbase 缓存机制的开发
- ElasticSearch 分布式全文搜索引擎的部署和测试

进展情况

已完成

- 平台功能开发
- Jenkins 自动化测试、部署工具的配置和部署脚本开发
- Couchbase 缓存机制的开发
- ElasticSearch 分布式全文搜索引擎的部署和测试

进行中

- 缓存和搜索接口的开发
- 推荐系统的开发

References I

[1] Jark. 推荐系统学习：协同过滤实现.

[2] 卞孟春. 基于 jenkins 的持续集成方案设计与实现. Master's thesis, 中国科学院大学, 2014.

[3] 唐志贤 王超姜康, 冯钧. 基于 elasticsearch 的元数据搜索与共享平台. 计算机与现代化, 2:117–121, 2015.

[4] 朱二东. 基于 jenkins 的 web 应用自动化测试的设计与实施. Master's thesis, 北京邮电大学, 2015.

[5] 陈文娟 朱永跃杨道建, 赵喜仓. 大学生创业培养环境、创业品质和创业能力关系的实证研究. 科技管理研究, (20):130–136, 2014.

[6] 刘贺平王国霞. 个性化推荐系统综述. 计算机工程与应用, 07, 2012.

[7] 王旭铭. 负载均衡集群中的会话保持研究. Master's thesis, 中山大学, 2013.

[8] 胡新明. 基于商品属性的电子商务推荐系统研究. Master's thesis, 华中科技大学, 2012.

[9] 许大宏. Elasticsearch 在车牌识别系统中的应用研究. 计算机时代, 12, 2014.