ACHIEVEIT

软件架构设计说明书

文件状态:	文件标识:	/
[√] 草稿	当前版本:	1.0
[ ] 正式发布	作　　者:	G03
[ ] 正在修改	完成日期:	2020-03-15

版 本 历 史

版本/状态	作者	参与者	起止日期	备注
V0.1	G03 曹威杰	苏美澄 叶姝晴 曹威杰 赵宁 陶明泗 陈弈君	2020.03.04 至 2020.03.07	初版
V1.0	G03 曹威杰	苏美澄 叶姝晴 曹威杰 赵宁 陶明泗 陈弈君	2020.03.11 至 2020.03.15	

目录

1 概述

本文档将简述 AchieveIt 项目管理软件的软件设计，主要从系统逻辑架构设计、对象设计和数据库设计三个角度展开，并针对系统的可用性、高性能、拓展性、安全性要求进行讨论设计。

2 设计目标和约束

本软件旨在为公司内部提供一个高效的可用的项目管理系统，以显著提高软件开发的管理效率。大体上来说，本软件需要在满足兼容公司原有人事系统、软件缺陷系统、Git/文件/邮件系统和资产系统的前提下提供清晰可用的软件开发管理工作流。其次本软件需要满足从公司内部使用的安全性和可靠性，保证权限管理清晰和资产安全完整。最后考虑到公司的发展，本软件应预留一部分性能空间和设计可拓展性。具体上将从业务环境因素、使用环境因素、构建环境因素和技术环境因素四个方面罗列软件设计的约束:

1. 业务环境因素
   a) 项目必须在 2 个月内完成，预计 8 周内需进行产品交付。整个项目开发过程中除去人员成本，软件开发预算需控制在 200 人民币以内。
   b) 业务领域包括项目、工作流、工时、缺陷、风险和设备管理等方面的业务。业务需要替代原有项目管理的工作流，提高项目开发的工作效率。
2. 使用环境因素
   a) 用户目标为公司内部全体员工，出于安全性考虑不提供外部人员访问。具体包括项目经理，各方 Leader 和成员。
   b) 软件需要对公司内部员工进行权限的校验，提高软件的安全性。
   c) 用户的使用习惯将限制在公司内部进行统一规范。
   d) 使用的网络环境需要同时支持内网访问和外网访问。外网访问需提供安全性保障。
   e) 软件需要提供 7*24 小时不间断服务。
   f) 软件需要达到系统峰值时预期 100 人的最大并发用户数，峰值响应时间不超过 5s，服务器系统 CPU 压力最高可保持在 95%以下。
   g) 前后端分开部署，因此需要进行跨域开发。
3. 构建环境因素
   a) 项目人员构成为六人，包括一名产品经理、三名开发和两名测试。开发团队由于是第一次合作且进行线上合作，因此将在开发过程中进行不断磨合。

b) 项目代码管理使用 Github 进行，缺陷管理和进度管理依赖 Github 的 issue 页面进行。

4. 技术环境因素

a) 技术平台前端使用 Vue.js 框架，后端使用 Spring Boot 进行快速开发。Vue 学习成本低且开发灵活，是市场上普遍接受的一套前端解决方案，但是生态环境没有 angular 和 react 来的好。Spring Boot 是目前最流行的中小型后端迅速开发框架，非常适合本项目开发。然而它自动配置的特性可能会降低项目可拓展性。

b) 编程语言前后端分别使用 Javascript 和 Java。

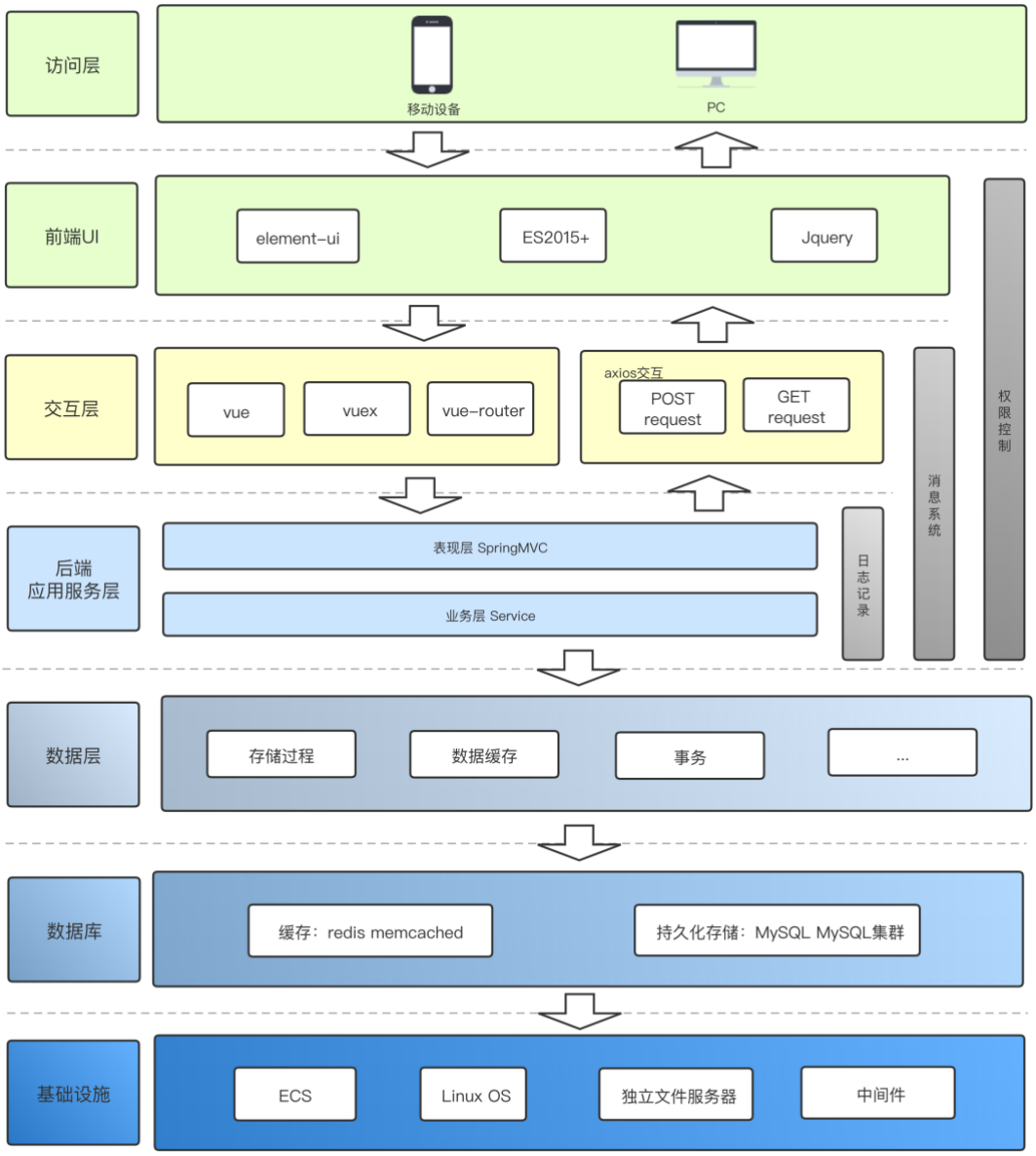
c) 开发工具依赖 IntelliJ Idea 2019 学生版。

d) 由于成本限制，服务器部署将依赖主流云平台（阿里/腾讯云）的 ECS 服务器学生版，服务器配置会受到一定的限制，特别展现在服务带宽方面。

随后的架构设计将说明如何满足以上的约束。

3 架构设计

3.1 总体方案



本软件作为大型动态应用系统平台，主要针对任务管理、权限分配建立的底层系统架构，软件的运行需要一个可靠、安全、可扩展、易维护的应用系统平台做为支撑，以保证应用的平稳运行。

总体架构分为七层，前端的访问层和 UI 层主要提供用户友好型的项目入口，作为权限控制的顶层实现，需要对于用户接口进行封装和美化，对于用户操作实现高容错性。软件运行在 web 端，前端 UI 主要介入 Element-UI，使用 ES2015+ 编译，提供多种设备接入，对于低端访问设备提供低配置版本。

交互层包括 view 和 router 部分的 vue-cli 和 vue-router，使用 vuex 进行状

态管理，采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。实现 axios 来处理网络请求，使用 Promise 管理异步，告别传统 callback 方式，包括 post 请求和 get 请求。

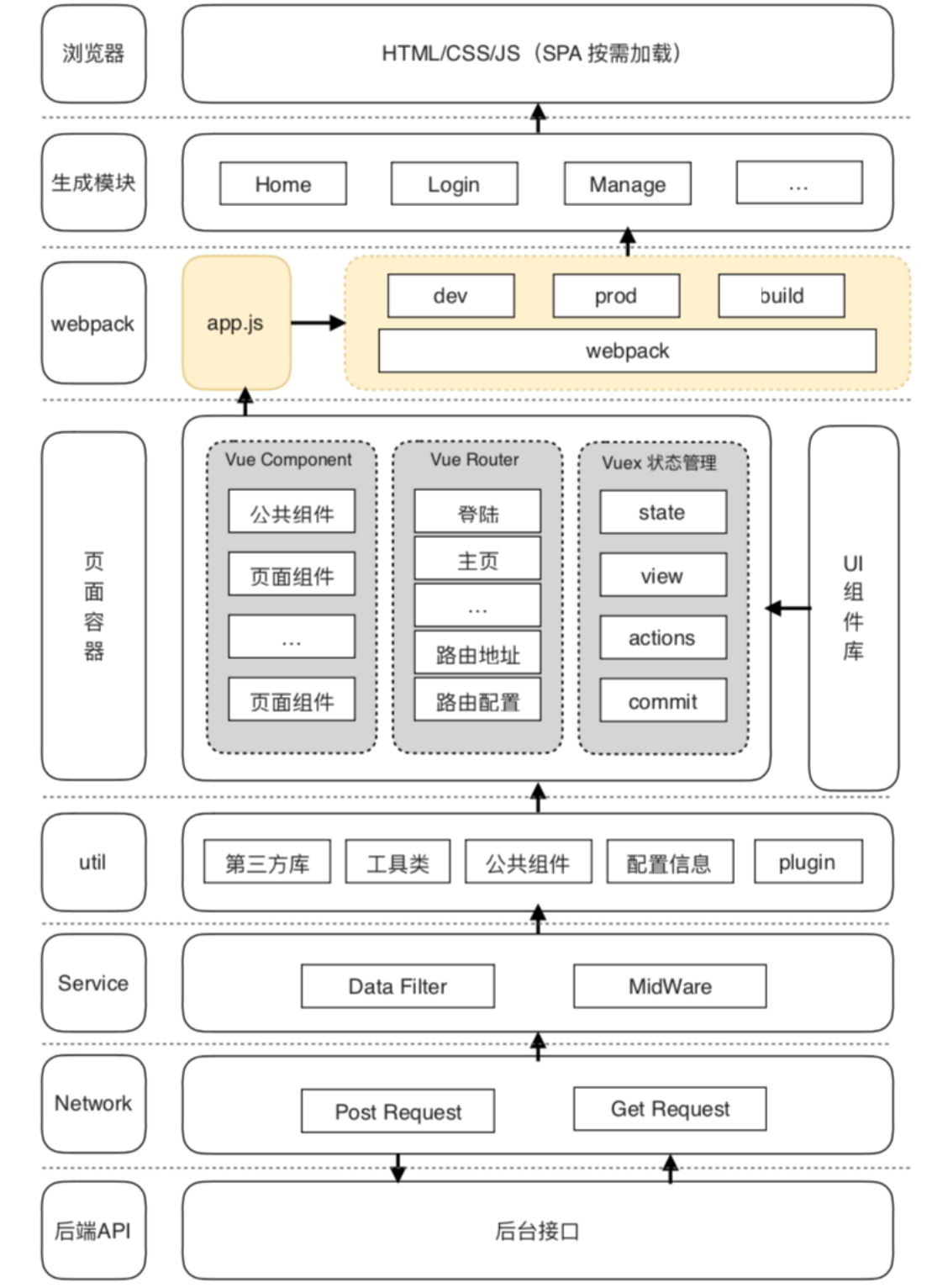
后端的应用服务层位于数据库等后端通用服务层与前台之间，向上接收由前端发而来的客户端访问请求，向下访问由数据库层提供的结构化存储与数据查询服务。应用层实现了 Web 应用的所有业务逻辑，通常要完成大量的计算和数据动态生成任务。其中也包括日志进程，交互层于应用服务层之间通用消息系统。软件设计了数据层来收集、缓存和分析数据，软件将数据发送到数据用于摄取和处理数据的流接口，或是原始数据以及最终转换和得到的增强数据保存到云存储，亦支持经过转换/增强的数据通常被加载到数据仓库中进行分析。

软件利用多个数据库来存储信息。数据库提供了定义数据结构，插入新数据，查找现有数据，更新或删除现有数据，跨数据执行计算等的方法。本软件取消了应用程序服务器与作业服务器直接对话，每个后端服务可能拥有自己的数据库，数据库与应用程序的其余部分隔离。

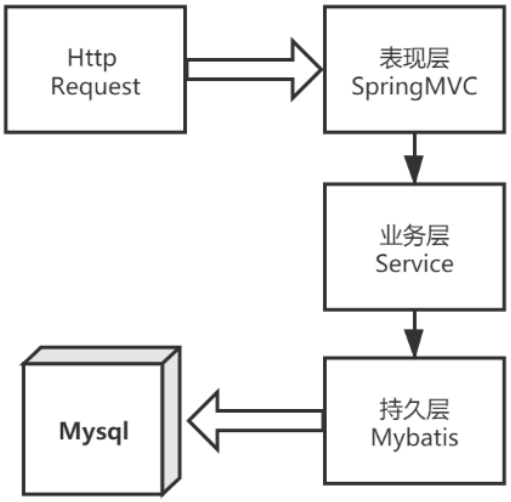
软件的基础设施包括弹性云服务，安装有 linux 操作系统的服务器以及文件服务器等，便于为后端提供坚实的物理基础。

从前端角度来讲：

首先软件在前端网络层向后端发送请求并从后端 API 中取得数据或事件结果，在服务层里进行数据过滤和中间件实现，再往上是工具层，软件会在这一层接入第三方，加载工具类和公共组件（如布局组件），并读入配置信息。最庞大的一层是容器层，在这一层里，软件会分析各类 vue 组件，并链接和匹配路由，通过 vuex 对于前端的对象进行状态管理，加载 UI 组件库并封装组件，在 webpack 层通过 webpack 进行模块打包，从 app.js（默认入口）开始递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 bundle。在顶层生成各个模块，最后在浏览器中渲染出来。



从后端角度来说，本软件的后端将基于 Spring Boot 开发。因此整体框架会采用 SSM (SpringMVC + Service + MyBatis) 系统架构，使用 MSCM(Mapper/Service/Controller/Model)层级设计完成后端的设计开发。系统架构如下图所示：

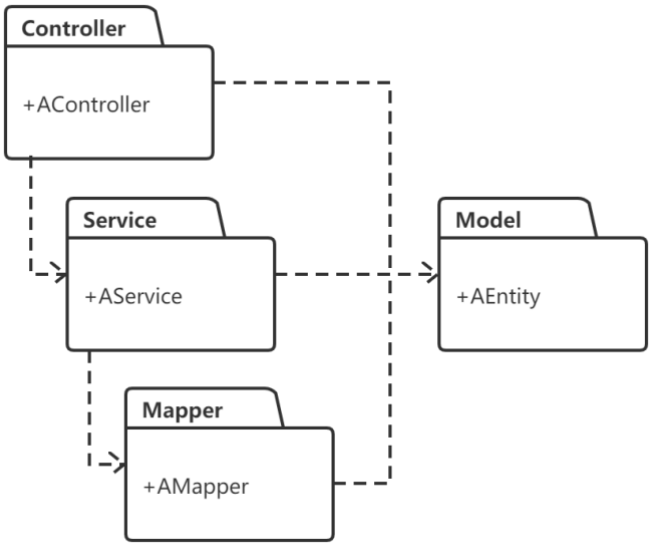


在 SSM 系统架构中，系统通过 Spring 将各层进行整合，分别管理持久层、业务层和进行事务控制。而通过 Spring 管理的表现层将作为软件的接口为前端提供接口，与业务层交互完成整个业务。业务层负责将复杂的业务拆分成多个细粒度的 Dao 操作，调用 Dao 获取数据进行封装成为结果。持久层使用 Mybatis 框架，完成业务层提交的数据库操作。在整个系统框架中，Spring 将持有各层依赖的对象的实例，通过依赖注入的方式注入到需要的地方、降低系统分层之间的耦合度。

3.2 架构说明

3.2.1 架构图及说明

在软件架构设计方面，将系统分为 MSCM(Mapper/Service/Controller/Model)四个层级进行抽象，抽象层级结构如下面给出的包图所示：



在 MSCM 设计中，Controller 对应 SSM 框架中的表现层，Service 对应框架中的业务层，Mapper 对应框架的中的持久层。最后 Model 层中存放软件的实体类，与数据库中的属性值保持一致。

3.2.2 架构设计关键点

MSCM 设计的关键点在于职责的划分，按照 MSCM 的方式分层可以进一步细化软件架构的划分，使各个组件的职责清晰。具体的职责划分如下：

1. Controller 层中类主要的职责为：表单校验、控制跳转、调用服务和异常处理。它实质上是软件的 API，负责响应控制 Http 请求调用服务完成业务。
2. Service 层中类的主要职责为：校验查询条件、调用持久层、业务逻辑处理、数据封装、异常处理、事务处理。
3. Mapper 层中类的主要职责为：执行数据库操作、事务控制、持久化数据、连接数据库。
4. Model 层中主要存放与 MySQL 数据库对应的实体类。

3.2.3 高可用性设计

为了满足系统高可用性，需要完成三方面的设计，分别是 API 抽象、对象设计和数据库设计。首先 API 抽象如下 Mind Map 所示：

API Mind Map 罗列了需求中抽象出来的接口，然后根据责任划分为 Controller 层分配多个 Controller 类，依赖对应 Service 层中提供的逻辑服务，通过 Mapper 来进行持久层的数据交互完成整个业务。

重要的前后端接口和接口数据格式：

1. 用户模块

本系统中使用用户 id 与用户 token 配对的方式进行鉴权。

- 用户登录

Request 结构	Response 结构
eid	Employee DAO
password	token
	msg
	code

- 用户登出

Request 结构	Response 结构
eid	msg
	code

- 获取用户基本信息

Request 结构	Response 结构
eid	Employee DAO
uid	msg
token	code

- 我的 Dashboard

Request 结构	Response 结构
uid	Project DAOs
token	Requirement DAOs
	Defect DAOs
	Manhour DAOs

2. 项目管理模块

   所有模块操作都需要利用 uid 与 token 进行鉴权。

   工时模块:

- 提交工时单

Request 结构	Response 结构
uid	Manhour DAOs
token	msg
project_id	code
activity_id	

- 审核工时单

Request 结构	Response 结构
uid	Manhour DAOs

token	msg
manhour_id	code
status	

项目成员

- 获取全部成员

Request 结构	**Response 结构**
uid	Employee DAOs
token	msg
project_id	code

- 添加新成员

Request 结构	**Response 结构**
uid	Employee DAO
token	msg
project_id	code
eid	
project_role	

项目权限

- 角色查询

Request 结构	**Response 结构**
uid	Employee DAO
token	msg
project_id	code
eid	

- 新增角色

Request 结构	**Response 结构**
uid	Employee DAO
token	msg
project_id	code
eid	
role	

- 设定权限

Request 结构	Response 结构
uid	Employee DAO
token	msg
project_id	code
eid	
right_id	

项目状态

- 新建项目

Request 结构	Response 结构
uid	Project DAO
token	msg
Project DAO	code

- 查询需要审核的项目

Request 结构	Response 结构
uid	Project DAO
token	msg
project_id	code

- 批准项目

Request 结构	Response 结构
uid	Project DAO
token	msg
project_id	code
status_ok	

- 更新项目配置

Request 结构	Response 结构
uid	Project DAO
token	msg
Project DAO	code

- 上传归档文件

Request 结构	Response 结构
uid	Project DAO
token	msg
file_name	code
file_payload	File DAO
project_id	

- 归档文件审批

Request 结构	Response 结构
uid	Project DAO
token	msg
project_id	code

项目检索与更新

- 项目查询

Request 结构	Response 结构
uid	Project DAO
token	msg
key	code
value	

- 新建里程碑

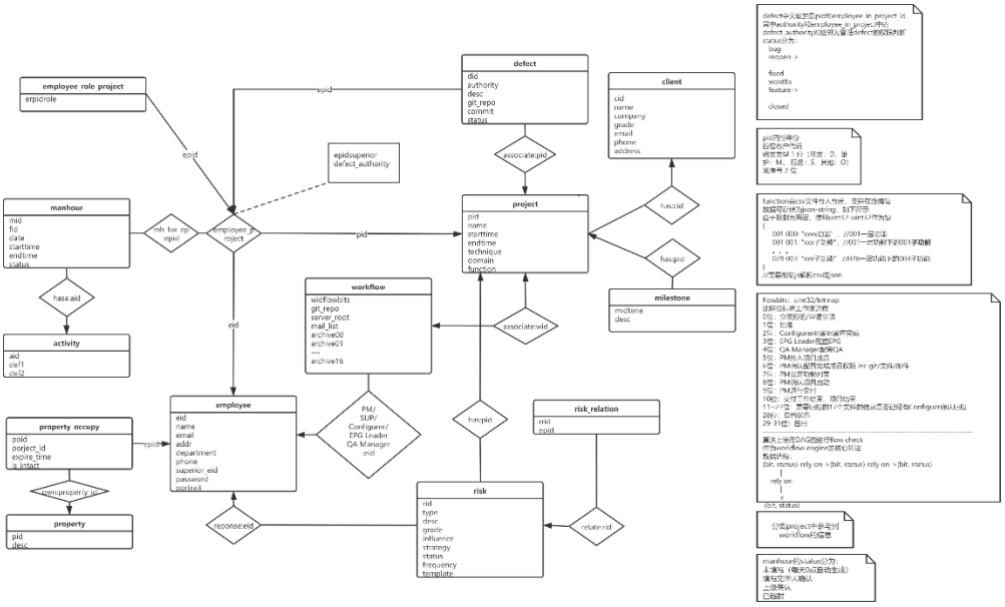
Request 结构	Response 结构
uid	Project DAO
token	msg
Milestone DAO	code
project_id	Milestone DAO

- 查询某个项目里的里程碑

Request 结构	Response 结构
uid	Milestone DAO
token	msg
milestone_id	code

project_id	

为了高可用性，相应的数据库设计也是必须的。数据库使用 MySQL 进行开发，对需求进行抽象和分解保证数据实体具有清晰的依赖关系并不重叠。具体的数据库关系 ER 图如下所示：



3.2.4 高性能设计

系统的高性能要求软件需达到系统峰值时预期 100 人的最大并发用户数，峰值响应时间不超过 5s，服务器系统 CPU 压力最高需保持在 95%以下。为满足这一需求，可提供三种解决方案，分别是使用高性能服务器、压缩 Response 数据和分布式设计。

- 高性能服务器：
  原设计是将系统部署在在 1Core/2GB/1Mb 带宽服务器上，但发现无法满足基本的用户需求。为了满足峰值条件下 100 并发数小于等于 5s 的响应时间，需要提高一定的服务器带宽来解决性能瓶颈，故选择了阿里云的应用级服务器学生版，具体服务器配置参数请参考 4 配置方案。
- 压缩 Response 数据：
  在项目开发的初级阶段，可采用减小 Response 的体积在经济可行范围内提升带宽来完成需求。该项目可利用 Spring Boot 的 Gzip 压缩实现，此方案可显著减少 json 串大小及传输时间；最小压缩 response 的数值需要视服务涉及接口返回数据的大小而定。保证响应 JSON 最小化，也是压缩 response 数据的有效手段。此外，在页面直接加载几百 KB 甚

至几十 MB 的图片也极为占用带宽，影响网站加载速度，因此根据前端页面所需求图片的尺寸生成动态缩略图可大大压缩 response 数据，提高响应速度。

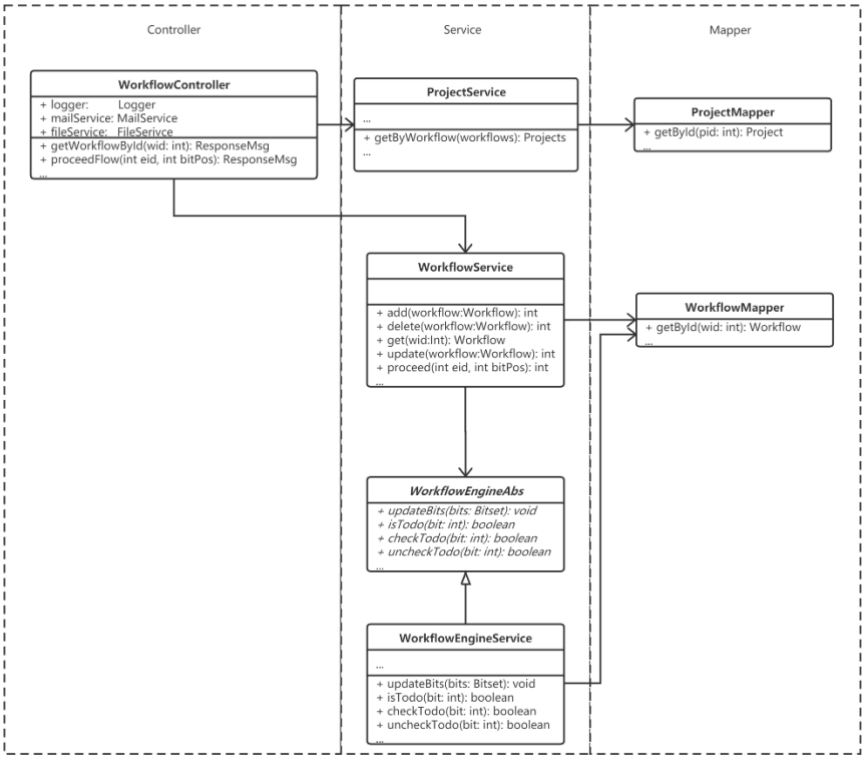
- 分布式设计：

  当前设计考虑到成本问题，故将数据库、Web Application 和前端组件部署在同一台服务器上。在项目后期，为保证满足用户当前及未来 1 到 3 年的发展需求，考虑分布式部署，使用 nginx 等负载均衡中间件，将请求分布到不同的机器上，避免单个应用持续的处理引起宕机。常用的分布式策略有如下几种：1）分布式应用和服务：将分层和分隔后的应用和服务模块分布式部署，可以改善网站性能和并发性、加快开发和发布速度、减少数据库连接资源消耗。2）分布式静态资源：网页的静态资源如 JS、CSS、图片等资源独立分布式部署，即动静分离。静态资源分布式部署可以减轻应用服务器的负载压力。3）分布式数据和存储：项目后期可能需要处理以 P 为单位的海量数据，这些数据库需要分布式存储。

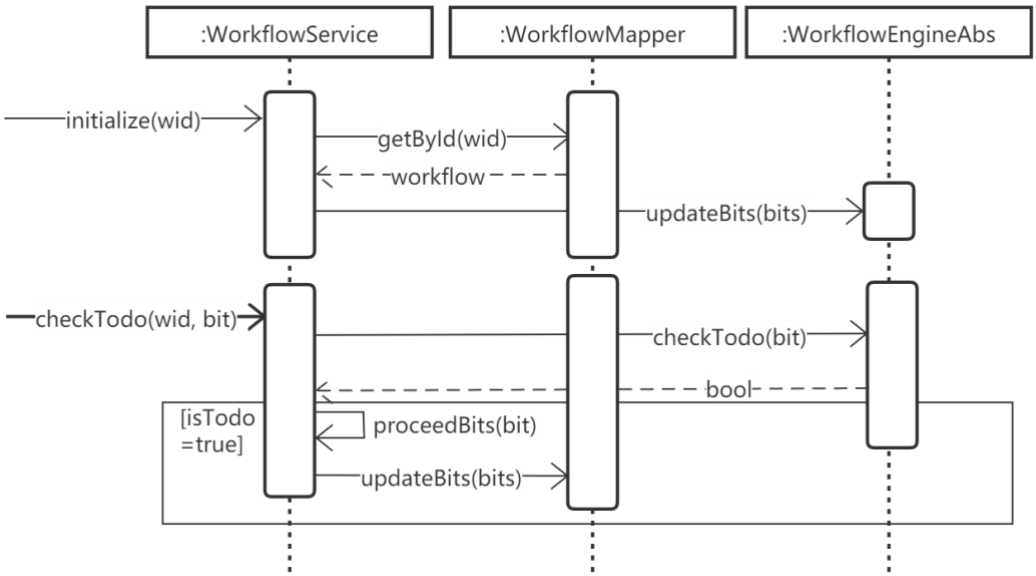
3.2.5 可扩展性设计

- 业务可拓展性：

本软件核心业务是工作流业务，其中工作流引擎是保证工作流相关的业务正常高效进行的核心。因此工作流引擎服务使用了面向对象的 Strategy 模式来提供核心的工作流服务算法，保证软件需求演变过程中的可拓展性。类图如下所示：



工作流算法的设计思路基础为：使用比特位标志工作流进程，依赖 DAG 图进行 Flow 进度的检查，完成工作流的推进。这样，可以为工作流业务提供高效且可拓展的服务，在需要拓展时仅需重新配置比特位依赖关系即可生成全新的工作流。在这个组件中，WorkflowEngineAbs 作为接口提供了前后交互的中间件服务，具体的顺序图如下所示：



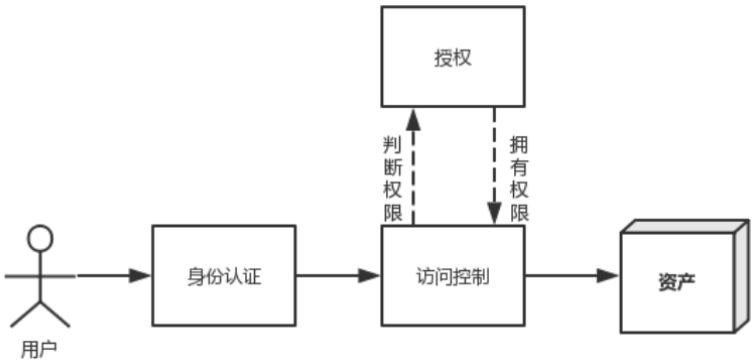
- 框架可拓展性：

  此项目为了迅速开发，项目组件完全依赖 Spring Boot 现有的封装，例如

Spring Mail、Mybatis、Hikari、Spring dev tools、swagger UI 等。因此框架的可拓展性较差。鉴于公司的发展速度和业务的增量情况，考虑在后期实现 Spring MVC 框架的搭建以支持更好的组件依赖拓展。

3.2.6 安全性设计

本软件将采用如下访问资产的流程来保障数据的安全：

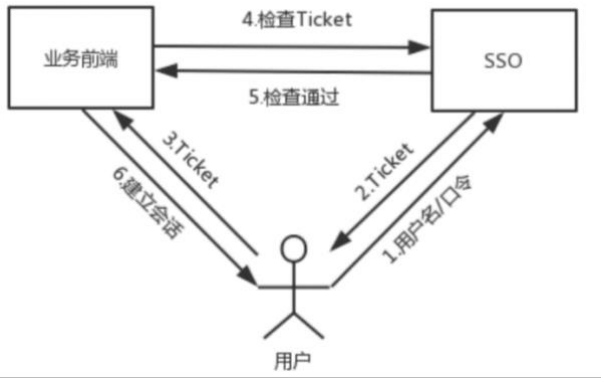


首先用户需要通过身份认证，而且用户必须有访问该资产的权限访问控制模块才能放行用户的请求，并且需要对资产采取一定的保护措施。上述步骤均需要留下日志记录，以提供可审计性。

软件具体的安全性设计方案如下：

1) 身份识别：

在用户登陆时跳转到 SSO 系统登陆，采集用户名与口令，调用相应 Dao 操作验证登陆信息，识别用户身份，完成登陆后跳回业务前端。在传输的过程中避免明文储存用户口令，对口令执行加盐散列操作，并选用 HTTPS 传输，来保障用户信息的安全。流程如下：



2) 权限控制：

对于页面模块与功能的展示而言，选择基于用户角色的授权作为策略方案。系统预置设定项目负责人、项目经理、项目成员等角色，导入员工信息的同时完成对每个成员的角色绑定，从而让用户拥有该角色所对应的权限。系统在对用户进行身份认证时获取用户的角色信息，进而决定前端展示的模块与功能，达到权

限控制的目的。

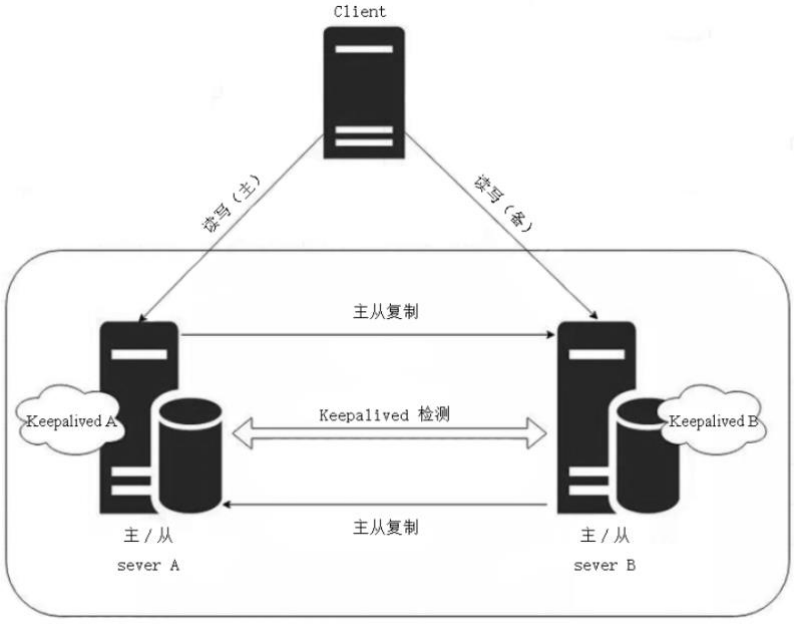
对于具体项目中成员的不同权限如访问 Git、文件系统相关信息、收取相关邮件的权限等则采取基于 ACL（Access Control List）的授权策略。人员具体的权限由项目经理进行分配，系统需要记录权限申请流程的每个审批环节的时间、IP 地址、用户 ID、审批者 ID、通过或驳回权限申请的动作，并将当前的权限信息与对应用户信息写入数据库相应表中。

3) 访问控制：

系统识别用户的权限信息，对用户的相应操作进行访问控制，执行的结果是放行或驳回。系统应记录所有的驳回动作，以及对敏感资产的每一个请求及动作，便于追溯。

4) 资产保护：

系统记录用户访问的资产及操作（查询、添加、修改、删除等），提供可审计性。采取互为主从的架构来对应用数据进行双向复制与备份，搭建两台服务器，在作为主机的同时也作为对方的从机，两台服务器均提供完整的读写服务，无需切换，前端在调用时随机选一台主机即可，当其中一台出现故障另一台依然可以继续提供服务，提高系统从故障中快速恢复的能力，保障资产的完整性与可用性。对应的架构视图如下所示：



3.2.7 其他设计

前端使用 vue.js 框架开发，渐进式框架和自底向上的特性使其便于视图层搭建。对于一个只有两人负责前端的团队来说，轻量级的 Vue 可以帮助项目的快速成型。

后端基于 Spring Boot2.2.4 开发，凭借其出色的功能包装来进行迅速的微服

务开发迭代，适合本项目后端的开发。

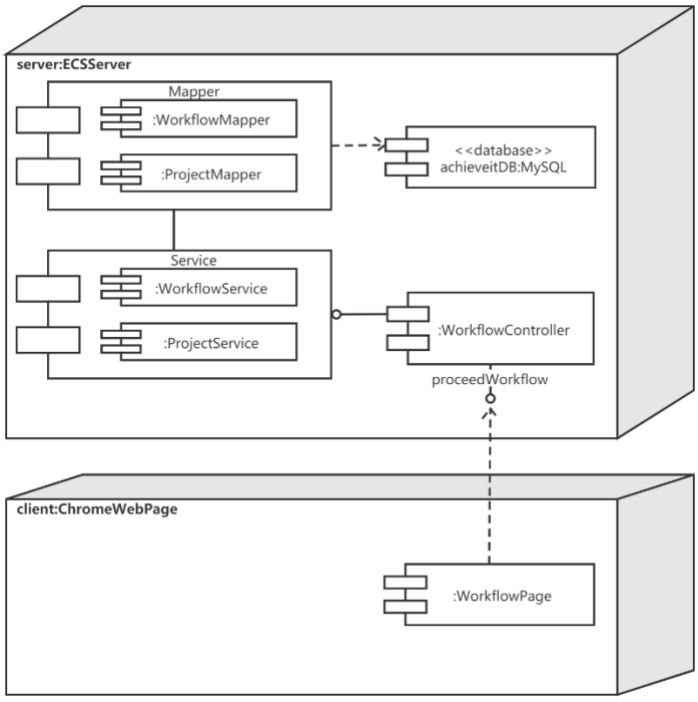
数据库依赖 MySQL 5.7，使用关系型数据库即可满足开发和业务需求。

4 部署方案

为了满足峰值条件下 100 并发数小于等于 5s 的响应时间，需要提高一定的服务器带宽来解决性能瓶颈。因此选择了阿里云的应用级服务器学生版，比一般的学生版服务器多出 4Mbps 的上下行带宽。具体的部署方案如下表所示：

云服务器组件	主要配置	具体描述
CPU	1 Core	/
内存	2GB	/
带宽	峰值 5Mbps	上行加下行总和
MySQL	/	5.7
Java	/	JDK 11 LTS

为了满足前后端分离开发的约束，将在同一个服务器的不同端口部署前后端组件，使用串联的方式组网。具体来说，后端的组织方式和应用的 MSCM 架构大体一致，对应的部署视图如下所示：



考虑到成本问题，在服务器端，不仅需要包含 DB 和 Web Application 组件，前端组件也需部署在同一服务器上。