项目架构设计文档

团队名称：OwO

创建者：何文兵

文档更新记录：2021.4.16

修改日期	作者	版本描述	版本
2021.3.14	何文兵	文档结构初始化	V1.0
2021.3.21	何文兵	完成迭代一项目结构设计文档	V2.0
2021.4.16	赵睿豪	完成迭代二项目结构设计文档	V3.0

1 总体介绍

1.1 编写目的

本文档提供知识图谱可视化系统的软件架构概览，采用若干架构试图描述系统的不同方面，以便表示构造系统所需要的重要架构决策。

1.2 对象和范围

本文档的读者是OwO团队内部的开发和管理人员，参考了RUP的《软件架构文档模板》，用于指导下一次迭代的代码开发和测试工作。

1.3 参考文献

- 《需求规格说明书》，OwO团队；
- 《软件架构文档模板》，Rational Software Corporation, 2002;
- The Object Management Group(OMG), The Unified Modeling Language Specification v1.4, 2003
- 《软件工程与计算——团队与软件开发实践》，骆斌、刘嘉、张瑾玉、黄蕾;

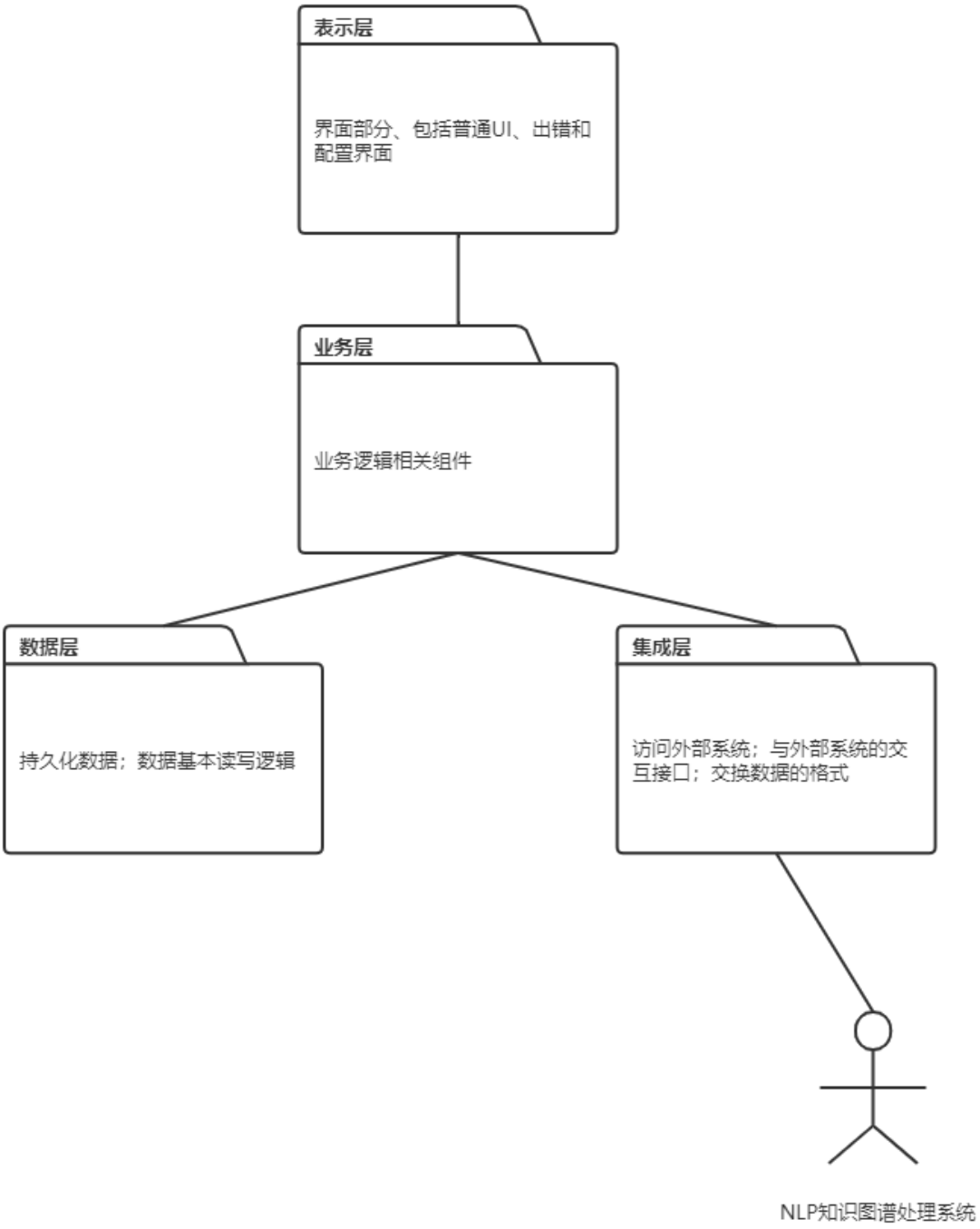
1.4 名词与术语

词汇名称	词汇含义	备注
MySQL	一个小型关系型数据管理系统，开发者为瑞典MySQL AB公司，属于开源软件	
JSP	Java Server Pages，是一种动态网页技术标准	
IDEA	后端开发工具	
WebStorm	前端开发工具	
VUE	一套自底向上型用于构建用户界面的渐进式JavaScript框架	
Spring Boot	一个开源的简化配置过程的轻量级框架	
Element-UI	一套为开发者、设计师和产品经理准备的基于 VUE 2.0 的组件库	
axios	一个基于 promise 的 HTTP 库	

2 系统的分层架构

2.1 系统的逻辑层次

系统的分层架构：



系统划分为以下4个逻辑层次。

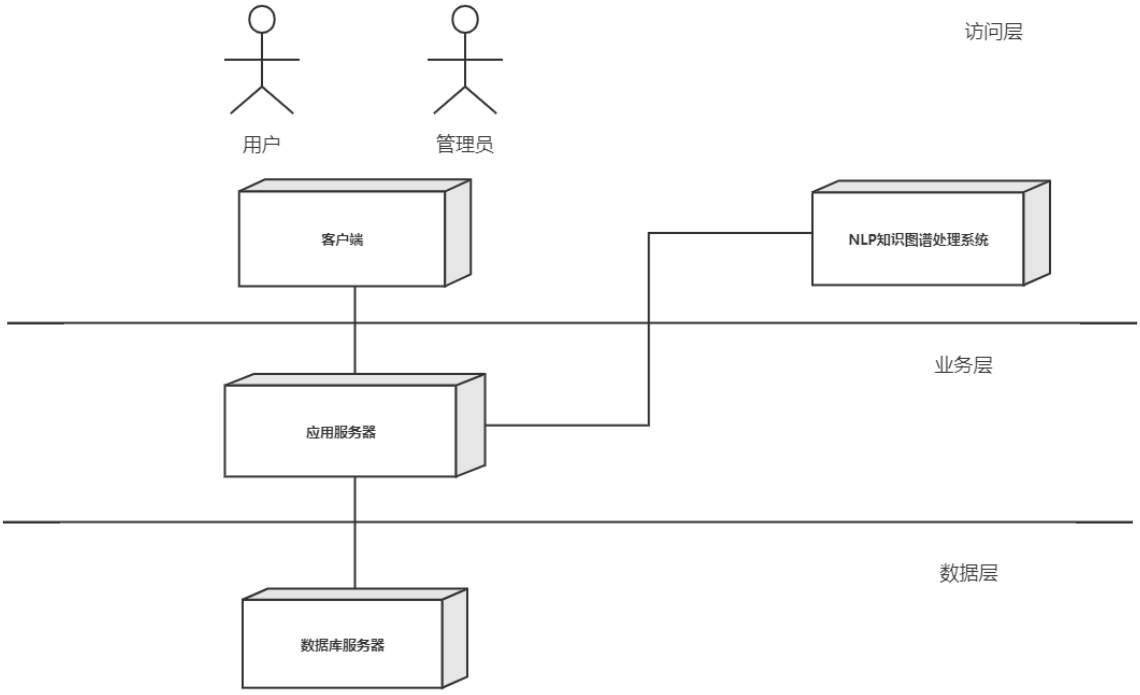
（1）表示层：用于前台界面展示和配置的层次。

（2）业务层：包含业务控制和逻辑的层次。

（3）数据层：定义和存储系统中相关数据的层次。

（4）集成层：定义和集成与外部系统交互接口的层次。

2.2 系统的物理层次



系统可以部署在以下3个物理层次。

（1）访问层：用于用户访问系统的层次。

（2）业务层：部署业务控制和逻辑的层次。

（3）数据层：部署和存储系统中相关数据的层次。

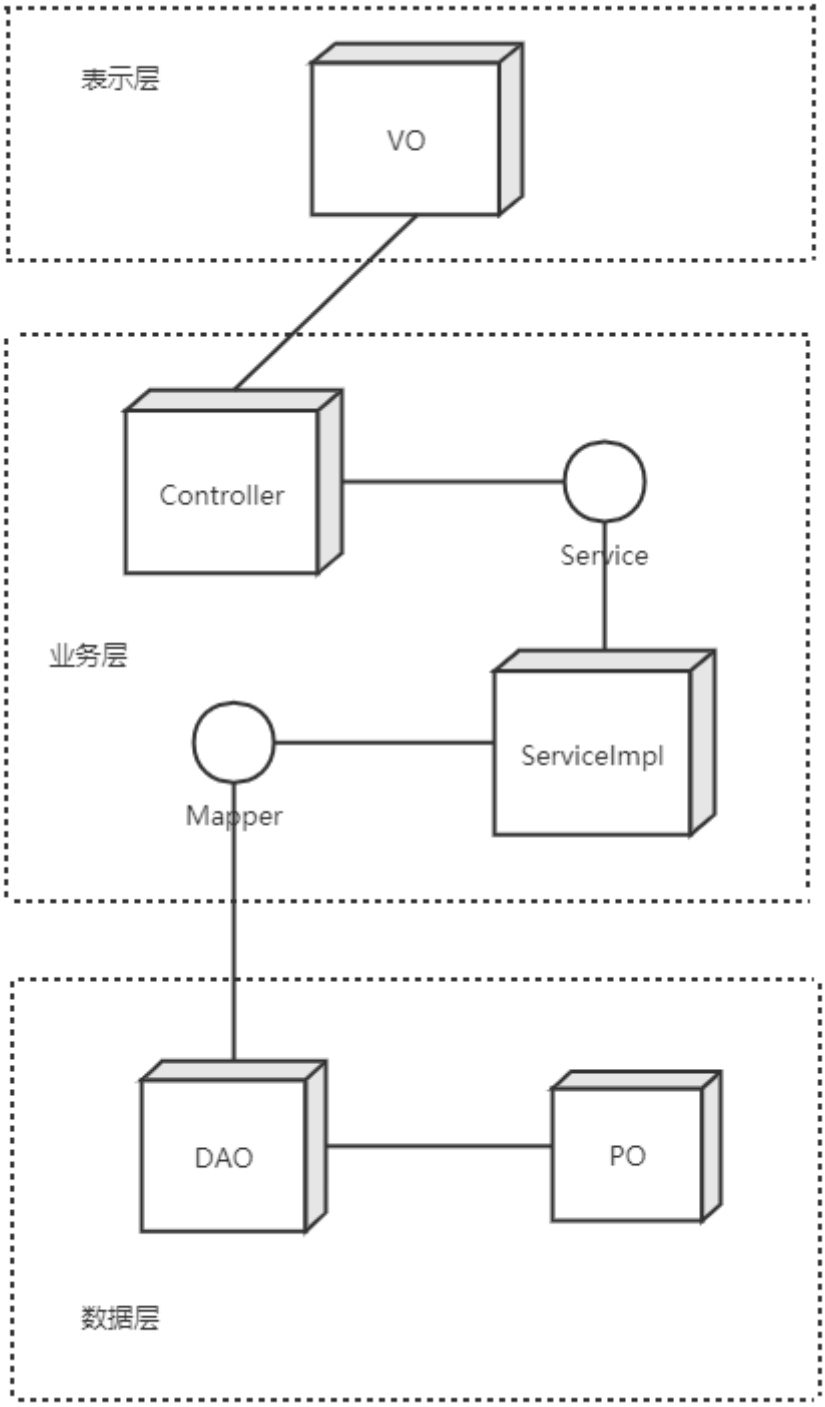
2.3 系统的架构设计

系统的架构设计如下。

系统架构中的对象分为6类：

1. VO对象，负责处理系统数据的展现和用户的交互。
2. Controller对象，控制器负责获取用户输入，并调用Service模块的服务。
3. Service对象，负责提供平台业务服务的抽象接口。
4. ServiceImpl对象，负责对于抽象接口的实现模块。
5. Mapper对象，负责提供获取数据对象的接口。
6. PO对象，数据持久化，用于将数据库中的数据封装成数据实体。
7. DAO对象，负责与数据库实体交互，获取数据，是Mapper抽象接口的实现模块。

2.4 系统的组件和组件接口


表示层
VO
Controller
Service
业务层
Mapper
ServiceImpl
DAO
PO
数据层

接口ID	连接组件		接口信息
I1	连接VO和Controller	语法	Return(ResponseVO) Interface(Request)
		前置条件	用户的输入正确
		后置条件	处理控制组件请求并且响应
I2	连接Controller和Service	语法	Return(result) Interface()
		前置条件	无
		后置条件	对应的Service执行对应的业务逻辑
I3	连接Service和Mapper	语法	Return(data) Interface(command)
		前置条件	无
		后置条件	对应的Mapper调用相应的实现方法获取数据集合
I4	连接Mapper和PO	语法	Return(PO) Interface(criteria)
		前置条件	数据库连接正常
		后置条件	PO对象写入数据库或从数据库中返回PO对象
I5	连接api和response	语法	Return response.data
		前置条件	前后端服务器运行正常
		后置条件	将response.data返回
I6	连接api和store	语法	Return axios()
		前置条件	response.res为success，即应答成功
		后置条件	处理返回数据

3 Jenkins配置

3.1 pipeline脚本

```
node {
    // 拉取代码
    stage('Git Checkout') {
        checkout([$class: 'GitSCM', branches: [[name: '${branch}']],
doGenerateSubmoduleConfigurations: false,
        extensions: [], submoduleCfg: [], userRemoteConfigs: [[credentialsId:
        'adf9f1d0-a09d-42e6-aa2c-c6a5dacc8ebc', url:
'http://212.129.149.40/181250043_owo/backend-owo.git']]])
    }
    // 代码编译
```

```
    stage('Maven Build') {
        sh '''
        export JAVA_HOME=/usr/local/jdk
        /usr/local/maven/bin/mvn clean package -Dmaven.test.skip=true
        '''
    }
    // 项目打包到镜像并推送到镜像仓库
    stage('Build and Push Image') {
        sh '''
        REPOSITORY=212.129.149.40/181250043_owo/backend-owo:${branch}
        cat > Dockerfile << EOF
        FROM 212.129.19.40/ceshi/tomcat:v1
        MAINTAINER wfy
        RUN rm -rf /usr/local/tomcat/webapps/
        ADD target/.war /usr/local/tomcat/webapps/ROOT.war
        EOF
        docker build -t $REPOSITORY .
        docker login 212.219.149.40 -u 181250043 -p 181250043
        docker push $REPOSITORY
        '''
    }
    // 部署到Docker主机
    stage('Deploy to Docker') {
        sh '''
        REPOSITORY=212.129.149.40/181250043_owo/backend-owo:${branch}
        docker rm -f tomcat-java-demo |true
        docker pull $REPOSITORY
        docker container run -d --name tomcat-java-demo -p 88:8080 $REPOSITORY
        '''
    }
}
```