迭代二项目设计文档

文档修改记录

日期	操作者	变更说明	版本
2020年3月8日	陈峙宇	迭代一初稿	v1.0
2020年3月9日	陈峙宇	完成迭代一组合视图和业务逻辑层分解	v1.1
2020年3月9日	李泳劭	完成迭代一剩余架构设计和信息视角	v1.2
2020年3月17日	陈峙宇	整合并补充更改迭代一文档	v1.3
2020年4月18日	陈峙宇	在迭代一基础上整合并补充形成迭代二文档	v1.4

1.引言

1.1 编制目的

本报告详细完成对COIN知识图谱系统的概要设计，达到指导详细设计和开发的目的，同时实现和测试人员及用户的沟通。

本报告面向开发人员、测试人员及最终用户而编写，是了解系统的导航。

1.2 词汇表

词汇名称	词汇含义	备注
知识图谱	在图书情报界称为知识域可视化或知识领域映射地图，是显示知识发展进程与结构关系的一系列各种不同的图形，用可视化技术描述知识资源及其载体，挖掘、分析、构建、绘制和显示知识及它们之间的相互联系。	
数据持久化	即把数据（如内存中的对象）保存到可永久保存的存储设备中（如磁盘）。持久化的主要应用是将内存中的对象存储在数据库中，或者存储在磁盘文件中、XML数据文件中等等。	
模块	整个系统中一些相对对独立的程序单元，每个程序单元完成和实现一个相对独立的软件功能	
开发包	具有特定的功能，用来完成特定任务的一个程序或一组程序	
API	Application Programming Interface，应用程序接口	
客户端	与服务器相对应，为客户提供本地服务的程序	
服务端	服务端是为客户端服务的，服务的内容诸如向客户端提供资源，保存客户端数据。	

1.3 参考资料

1.COIN知识图谱系统需求规格说明文档

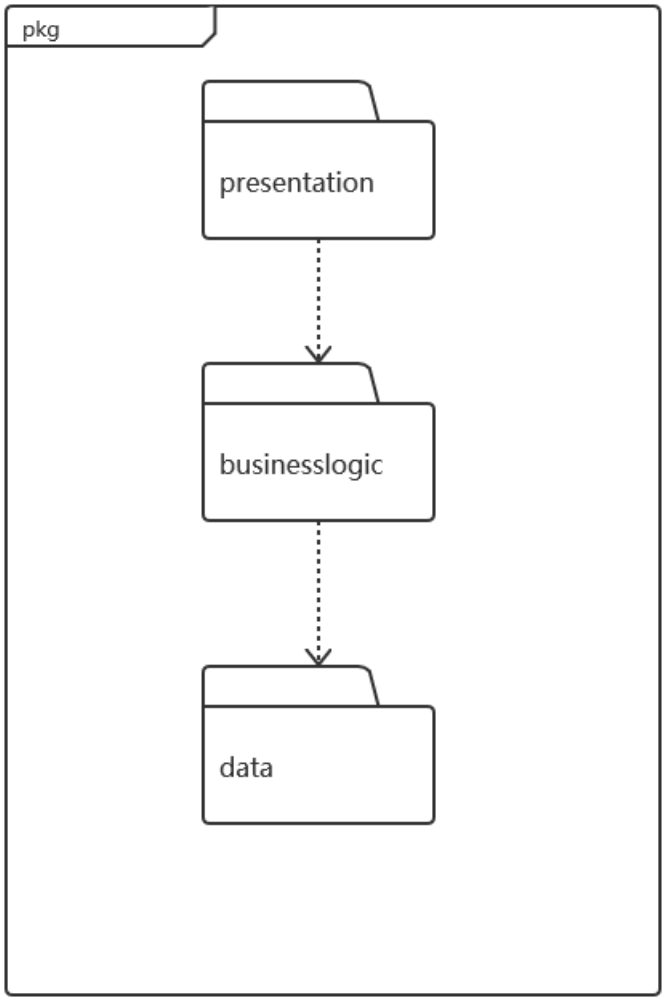
2.COIN知识图谱定义及可视化系统 ——房春荣

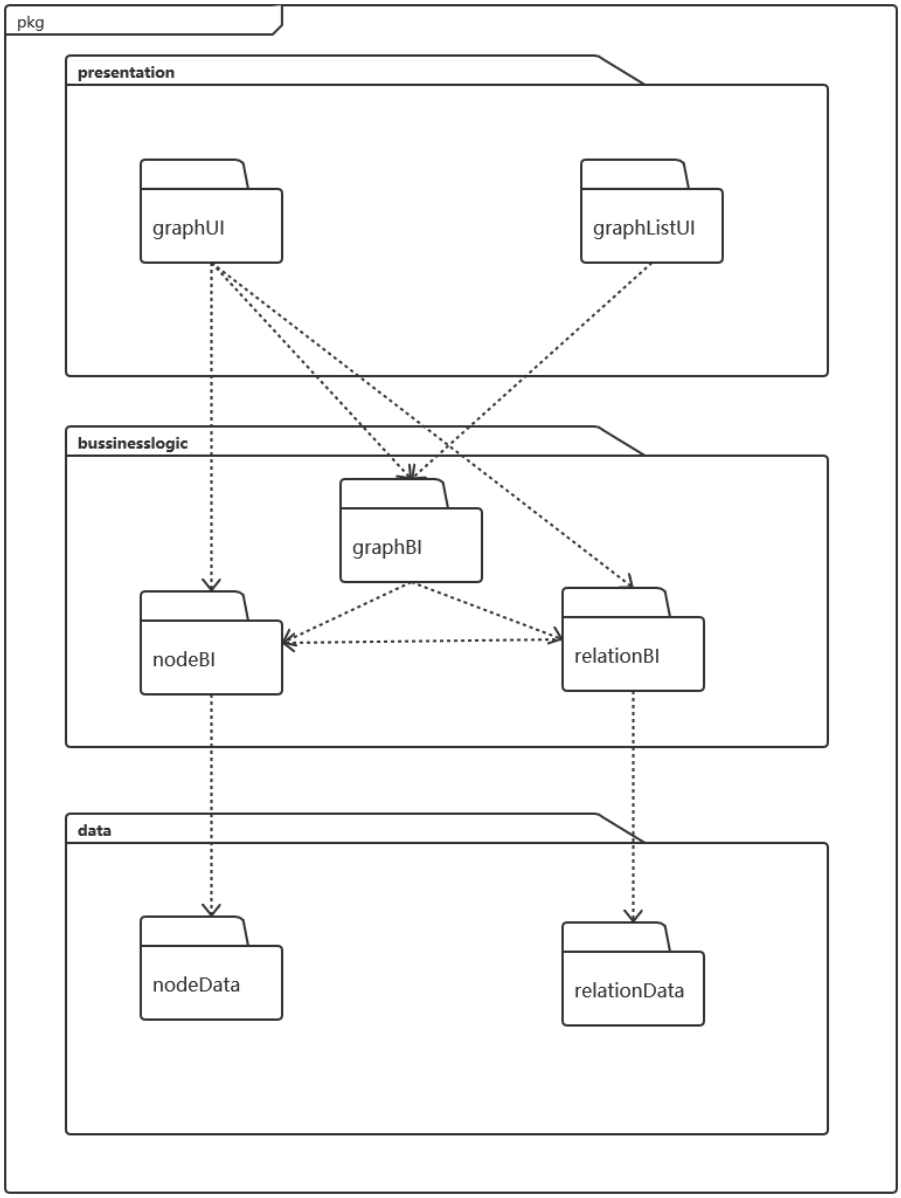
2.产品概述

参考COIN知识图谱系统需求规格说明文档中对产品的概括描述。

3.逻辑视图

COIN知识图谱系统中，选择了分层体系结构风格，将系统分为三层(展示层，业务逻辑层，数据层)能够很好的示意整个高层抽象。展示层包含网页页面的实现，业务逻辑层包含业务逻辑处理的实现，数据层负责数据的持久化和访问。分层体系结构的逻辑视角和逻辑设计方案见下图。





4.组合视图

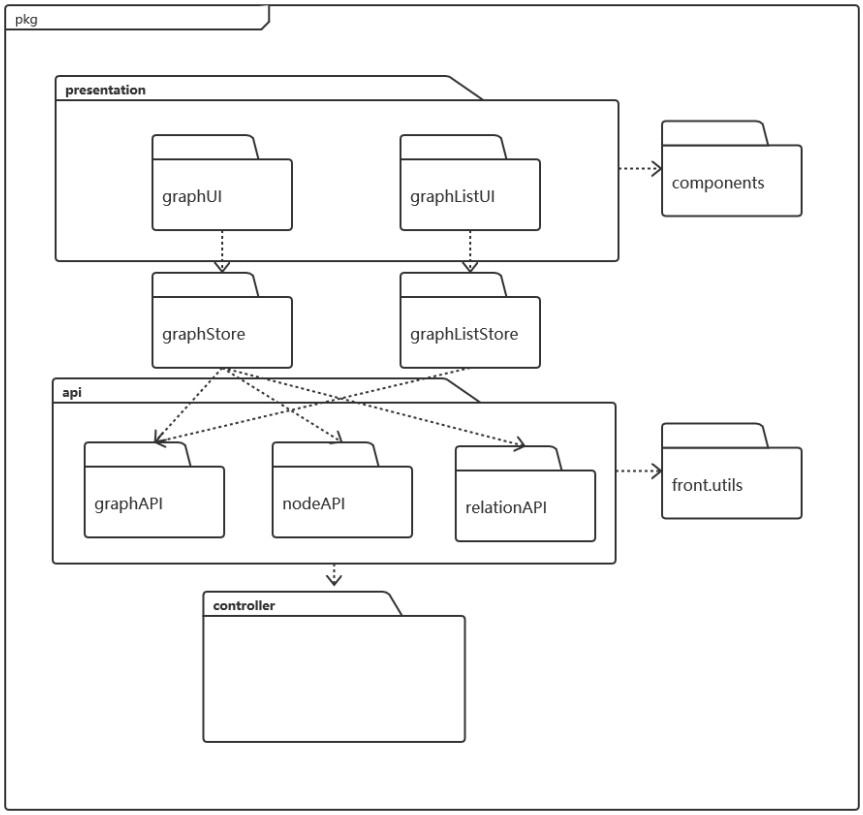
表示软件组件在开发时环境中的静态组织

4.1开发包图

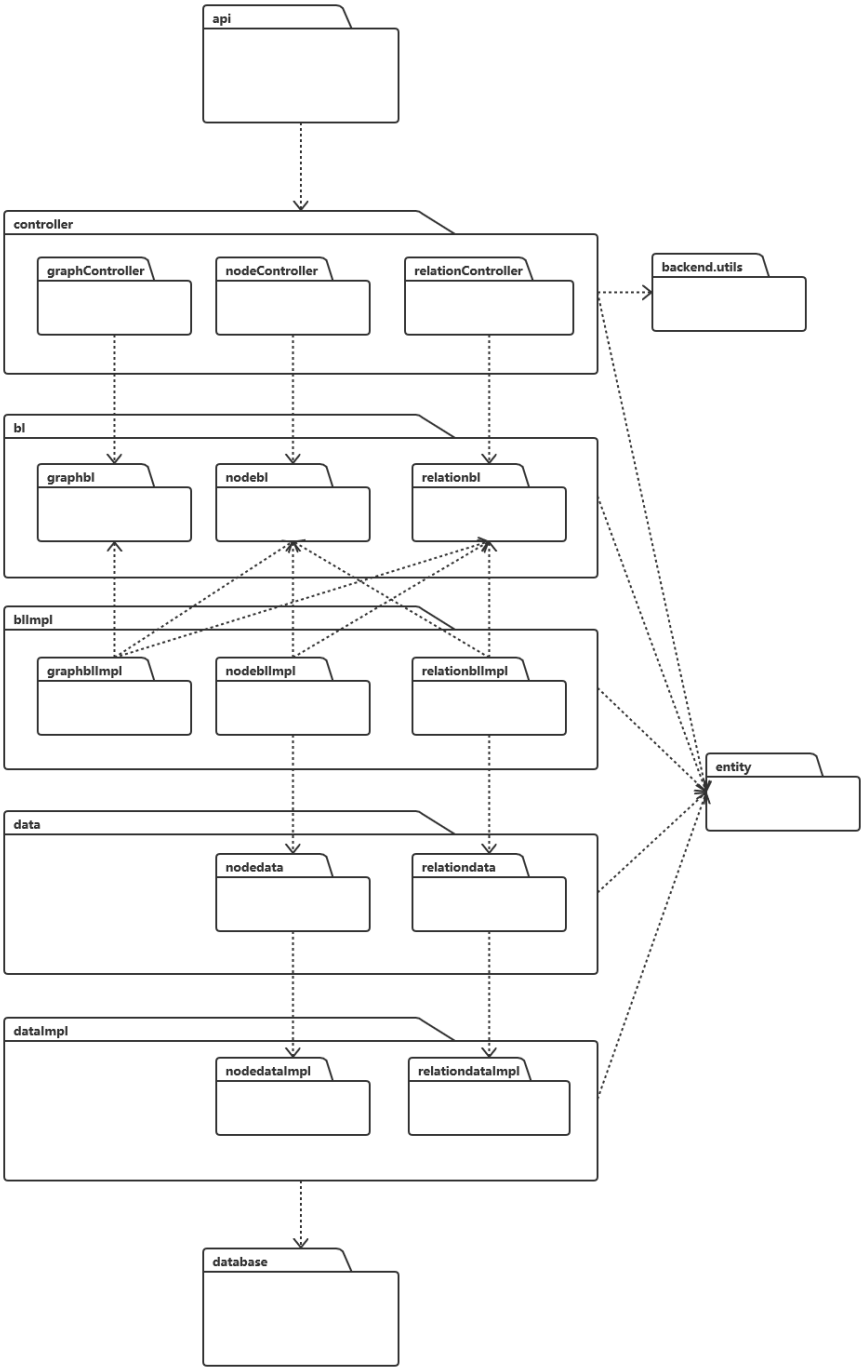
- 描述开发包以及相互间的依赖

开发包	依赖的其他开发包
controller.graph	bl.graph,entity,backend.utils
controller.node	bl.node,entity,backend.utils
controller.relation	bl.relation,entity,backend.utils
bl.graph	entity
bl.node	entity
bl.relation	entity
blImpl.graph	bl.graph,bl.node.bl.relation,entity
blImpl.node	bl.node,data.node,bl.relation,entity
blImpl.relation	bl.relation,bl.node,data.relation,entity
data.node	entity
data.relation	entity
dataImpl.node	data.node,entity
dataImpl.relation	data.relation,entity
entity	
backend.utils	
api.graph	controller.graph,front.util
api.node	controller.node,front.util
api.relation	controller.relation,front.util
store.graph	api.graph,api.node,api.relation
store.graphList	api.graph
presentation.graph	store.graph,components
presentation.graphList	store.graphList,components
front.util	
components	

- 绘制开发包
-


pkg
presentation
graphUI
graphListUI
components
graphStore
graphListStore
api
graphAPI
nodeAPI
relationAPI
front.utils
controller

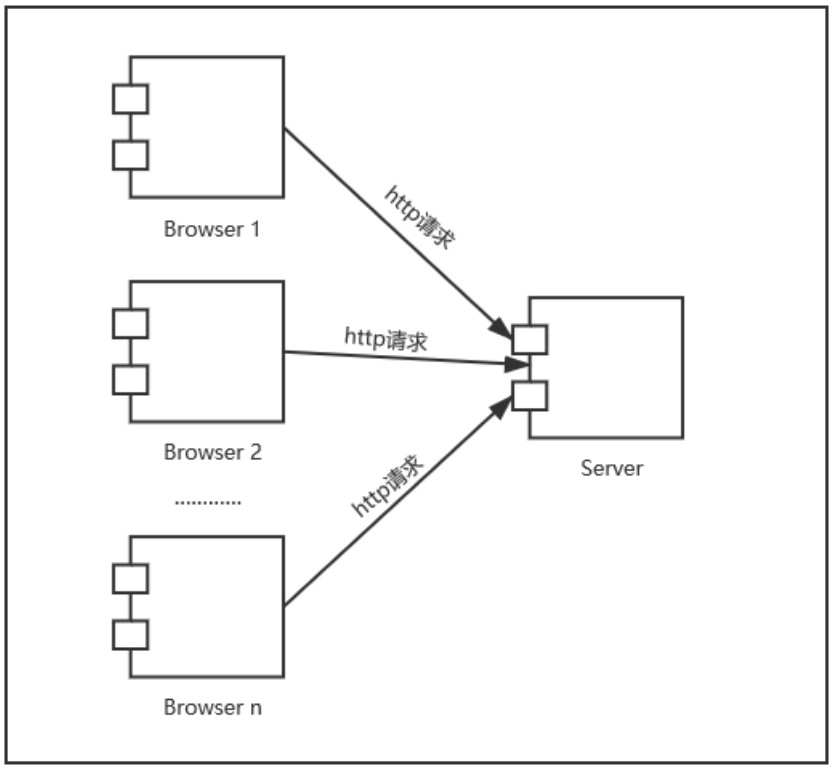
客户端开发包图


api
controller
graphController
nodeController
relationController
backend.utils
bl
graphbl
nodebl
relationbl
blImpl
graphblImpl
nodeblImpl
relationblImpl
entity
data
nodedata
relationdata
dataImpl
nodedataImpl
relationdataImpl
database

服务端开发包图

4.2 运行时进程

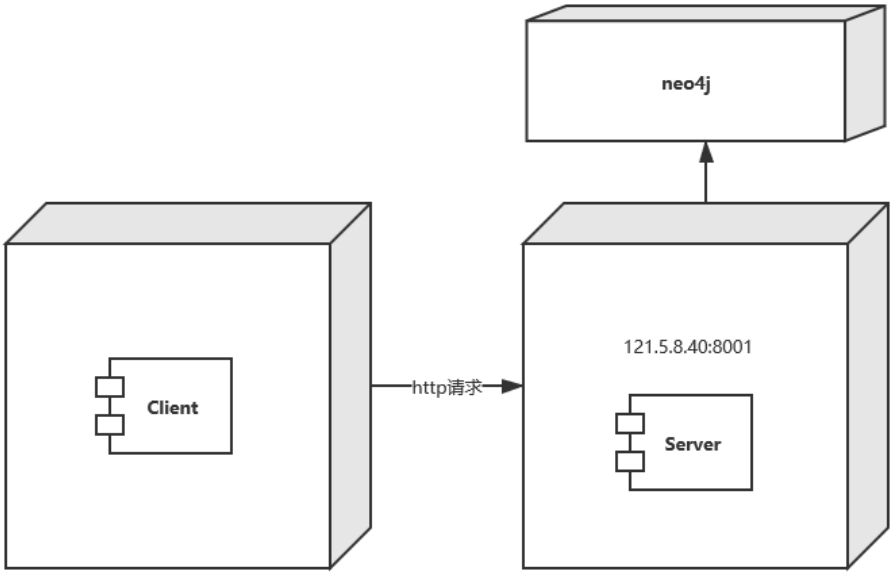
在COIN知识图谱系统中，会有多个浏览器进程和一个服务端进程，其进程图如下



4.3 物理部署

本项目部署在服务器上，用户通过了浏览器访问网站获取服务

开发过程用Jenkins，从Gitlab拉取代码，并编译、测试、自动部署

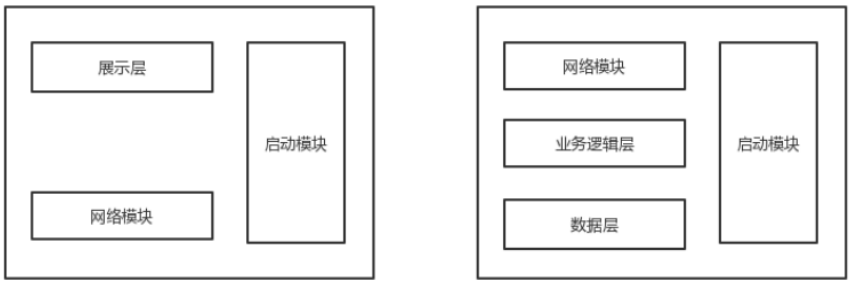


5.架构设计

本COIN系统采用web三层架构设计，整个系统划分为展示层、业务逻辑层、数据层。每一层只使用下方直接接触的层，层与层之间的交互是通过接口的调用来完成的。系统采用B/S结构，展示层在浏览器上，系统功能的实现集中在服务器。

5.1 模块职责

- **模块视图**



- **各层职责**

  客户端各层职责

层	职责
启动模块	初始化网络通信机制，加载、渲染用户界面
客户端展示层	基于web的COIN界面，并通过restAPI调用业务逻辑层以响应用户操作
网络模块	使用http与后端进行通信

  服务端各层职责

层	职责
启动模块	通过SpringApplication.run()实现自动初始化
数据层	负责数据导入、初始化以及提供数据访问、修改接口
业务逻辑层	根据业务逻辑处理数据并反馈处理结果
网络模块	接受到客户端request后，通过RestController，根据客户端的请求，调用业务逻辑层的服务并将处理结果反馈给客户端

- **层间调用接口**

接口	服务调用方	服务提供方
GraphService NodeService RelationshipService	展示层	业务逻辑层
NodeDao RelationshipDao	业务逻辑层	数据层

5.2 用户界面层分解

系统存在两个界面，一个为图谱仓库界面，负责展示图谱，点击具体图谱后可跳转至具体图谱界面，左侧为图谱信息栏，中部为图谱可视化区域，右侧为操作栏。

5.2.1 职责

模块	职责
graph	负责展示、编辑、导出知识图谱
graphList	负责展示、编辑图谱仓库

5.2.2 接口规范

graph接口规范

提供的服务（供接口）		
saveLayout	语法	saveLayout()
	前置条件	用户选择保存布局
	后置条件	保存图谱布局
resetMonitor	语法	resetMonitor()
	前置条件	用户选择重置视角
	后置条件	重置图谱视角中心和放缩大小
commonChange	语法	commonChange()
	前置条件	用户拖动视图更改中的力引导斥力大小或标签字体大小
	后置条件	改变力引导斥力大小或标签字体大小
symbolSizeChange	语法	symbolSizeChange()
	前置条件	用户拖动视图更改中的节点大小
	后置条件	改变节点大小
showLabelChange	语法	showLabelChange()
	前置条件	用户拖动视图更改中的显示标签阈值
	后置条件	改变显示标签阈值
updatePosition	语法	updatePosition(event)
	前置条件	用户退出拖拽模式
	后置条件	使节点变得不可拖拽并保存所有更新节点位置
enableDraggable	语法	enableDraggable()
	前置条件	用户进入拖拽模式
	后置条件	使节点变得可拖拽
changeMode	语法	changeMode(mode)

	前置条件	用户调整图谱布局
	后置条件	调整图谱布局并刷新图谱
confirmDeleteGraph	语法	confirmDeleteGraph()
	前置条件	用户请求删除关系图谱
	后置条件	删除图谱并刷新页面
editInfo	语法	editInfo()
	前置条件	用户提交编辑图谱内容
	后置条件	判断提交内容并更新图谱
delNode	语法	delNode()
	前置条件	用户删除节点
	后置条件	删除节点并刷新图谱
addEdge	语法	addEdge(data)
	前置条件	用户提交新增边请求
	后置条件	新增边并刷新图谱
delEdge	语法	delEdge()
	前置条件	用户请求删除关系
	后置条件	删除关系并刷新页面
setChart	语法	setChart()
	前置条件	图谱数据变更
	后置条件	刷新图谱渲染
exportImg	语法	exportImg()
	前置条件	用户选择导出图片

	后置条件	导出图片
exportJSON	语法	exportJSON()
	前置条件	用户选择导出Json文件
	后置条件	导出Json文件
exportXML	语法	exportXML()
	前置条件	用户选择导出XML文件
	后置条件	导出XML文件

需要的服务(需接口)	
服务名	服务
GraphService.retrieveGraphById	从数据库中返回指定图谱
GraphService.deleteGraphById	从数据库中删除指定图谱
GraphService.importGraphFromFile	构建并向数据库中插入从文件导入的Graph对象
NodeService.retrieveAllEntitiesByGraph	从数据库中返回指定图谱所有节点
NodeService.retrieveAllCategories	从数据库中返回指定图谱所有类目
NodeService.insert	向数据库中插入实体节点
NodeService.deleteNodeByName	根据实体名从数据库中删除
NodeService.deleteAll	删除数据库中所有数据
NodeService.updateNode	更新数据库中的目标实体数据
NodeService.getByName	根据实体名获取实体的详细数据
RelationshipService.buildRelationIncValue	在数据库中新增对应关系
RelationshipService.deleteRelationsByValue	在数据库中删除对应值的关系
RelationshipService.deleteRelationsByNodes	在数据库中删除对应源节点和目标节点的关系
RelationshipService.getAllRelations	在数据库中获取指定图谱所有关系
RelationshipService.updateRelation	在数据库中更新对应关系

graphList接口规范

提供的服务(供接口)		
loadGraph	语法	exportJSON()
	前置条件	用户进入图谱仓库
	后置条件	列表显示用户所有图谱
uploadGraphFile	语法	uploadGraphFile(graph)
	前置条件	用户选择上传图谱
	后置条件	上传图谱文件
deleteGraphById	语法	deleteGraphById(graphId)
	前置条件	用户选择删除图谱
	后置条件	删除选定图谱
需要的服务(需接口)		
服务名	服务	
GraphService.deleteGraphById	从数据库中删除指定图谱	
GraphService.getGraphList	从数据库中获取用户所有图谱	
GraphService.importGraphFromFile	用图谱文件新建图谱对象	

5.2.3 用户界面模块设计原理

用户界面基于node.js，使用Vue框架实现，页面设计UI框架使用Vuetify，图谱可视化部分使用echarts部分组件。

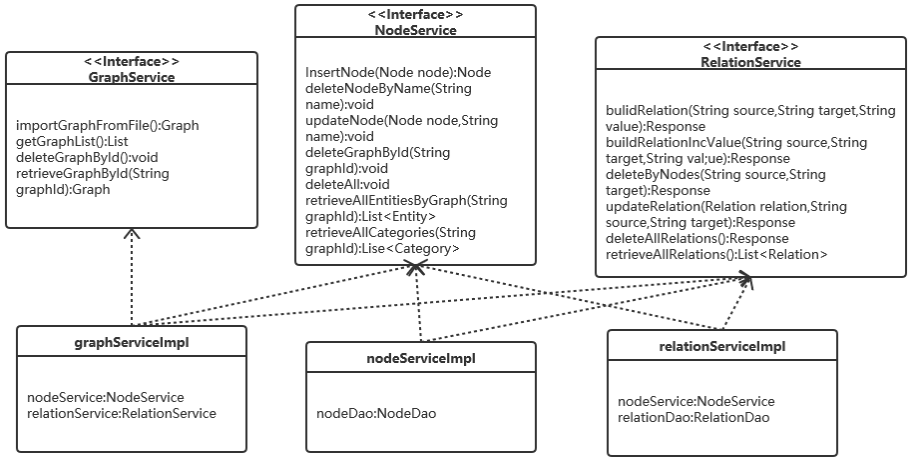
5.3 业务逻辑层的分解

业务逻辑层包括多个针对界面的业务逻辑处理的类及其实现

5.3.1 职责

模块	**职责**
NodeServiceImpl	负责实现节点的增删改查
RelationServiceImpl	负责实现关系的增删改查
GraphServiceImpl	负责实现整张图的创建、删除和获取

类图



5.3.2接口规范

GraphService接口规范

提供的服务（供接口）		
GraphService.importGraphFromFile()	语法	public Graph importGraphFromFile()
	前置条件	用户上传图谱文件
	后置条件	返回添加结果
GraphService.getGraphList	语法	public List getGraphList()
	前置条件	用户进入图谱仓库
	后置条件	返回用户所有图谱
GraphService.retrieveGraphById	语法	public Graph retrieveGraphById(String graphId)
	前置条件	用户进入图谱
	后置条件	返回图对象
GraphService.deleteGraphById	语法	public void deleteGraphById(String graphId)
	前置条件	用户点击并确认删除图谱
	后置条件	返回删除结果
需要的服务(需接口)		
服务名	服务	
nodeService.InsertNode(Node node)	持久化Node对象	
nodeService.InsertCategory(Category category)	持久化category对象	
nodeService.retireveAllGraphs()	获取所有持久化Graph对象	
nodeService.deleteGraphById(String graphId)	删除指定持久化Graph对象	
RelationService.buildRelation(Relation relation)	持久化Relation对象	

NodeService接口规范

提供的服务（供接口）		
NodeService.InsertNode	语法	public Node Insert(Node node)
	前置条件	用户创建新节点
	后置条件	返回添加结果
NodeService.deleteNodeByName	语法	public void deleteNodeByName(String name,String graphId)
	前置条件	用户删除节点
	后置条件	返回删除结果
NodeService.updateNode	语法	public void updateNode(Node node,String name,String graphId)
	前置条件	用户更新节点信息
	后置条件	返回更新结果
NodeService.retrieveAllEntitiesByGraph	语法	public List retrieveAllEntitiesByGraph(String graphId)
	前置条件	用户载入界面
	后置条件	返回节点列表

NodeService.retrieveAllCategories	语法	public List retrieveAllCategories(String graphId)
	前置条件	用户载入界面
	后置条件	返回类目列表
NodeService.deleteGraphById	语法	public void deleteGraphById(String graphId)
	前置条件	用户删除图表
	后置条件	返回删除结果
NodeService.deleteAll	语法	public void deleteAll()
	前置条件	用户清空图谱仓库
	后置条件	返回删除结果

需要的服务(需接口)	
服务名	服务
NodeDao.updateNode(Node node,String name)	更新数据库中指定Node对象
NodeDao.insert(Node node)	往数据库中插入Node对象
NodeDao.deleteNodeByName(String name)	从数据库中删除指定Node对象
NodeDao.retireveAllEntitiesByGraph(String graphId)	从数据库中返回图所有Node对象
NodeDao.retireveAllCategoriesByGraph(String graphId)	从数据库中返回图所有Category对象

NodeDao.deleteGraphById(String graphId)	删除指定图谱对象
NodeDao.deleteAll()	删除所有Node对象
RelationService.deleteByNodes(String source,String target)	删除指定持久化relation对象

RelationService接口规范

提供的服务（供接口）		
RelationService.buildRelation	语法	public void buildRelationIncValue(String source,String target,String value)
	前置条件	用户创建新关系
	后置条件	返回添加结果
RelationService.buildRelationIncValue	语法	public void buildRelation(String source,String target,String value)
	前置条件	用户导入图谱
	后置条件	返回添加结果
RelationService.deleteByNodes	语法	public void deleteByNodes(String source,String target)
	前置条件	用户删除关系
	后置条件	返回删除结果
RelationService.updateRelation	语法	public boolean RelationService.updateRelation(Relation relation,String source,String target)
	前置条件	用户更新关系信息
	后置条件	返回更新信息

RelationService.deleteAllRelationsByGraph	语法	public void deleteAllRelationsByGraph(String graphId)
	前置条件	用户删除图表
	后置条件	返回删除结果
RelationService.retrieveAllRelations	语法	public List retrieveAllRelations(String graphId)
	前置条件	用户载入界面
	后置条件	返回关系列表
需要的服务(需接口)		
服务名	服务	
RelationDao.buildRelation(String source,String target,String value)	用于在根据文件创建图谱时的插入，不会增加节点的value	
RelationDao.buildRelationIncValue(String source,String target,String value)	往数据库中插入Relation对象	
RelationDao.deleteRelationByNodes(String source,String target	从数据库中删除指定Relation对象	
RelationDao.updateRelation(Relation relation,String source,String target)	更新数据库中指定Relation对象	
RelationDao.retrieveAllRelations(String graphId)	从数据库中获取指定图谱所有Relation对象	
RelationDao.deleteAllRelationsByGraphId(String graphId)	从数据库中删除指定图谱所有Relation对象	
NodeService.updateNode(Node node,String name)	更新数据库中指定Node对象	

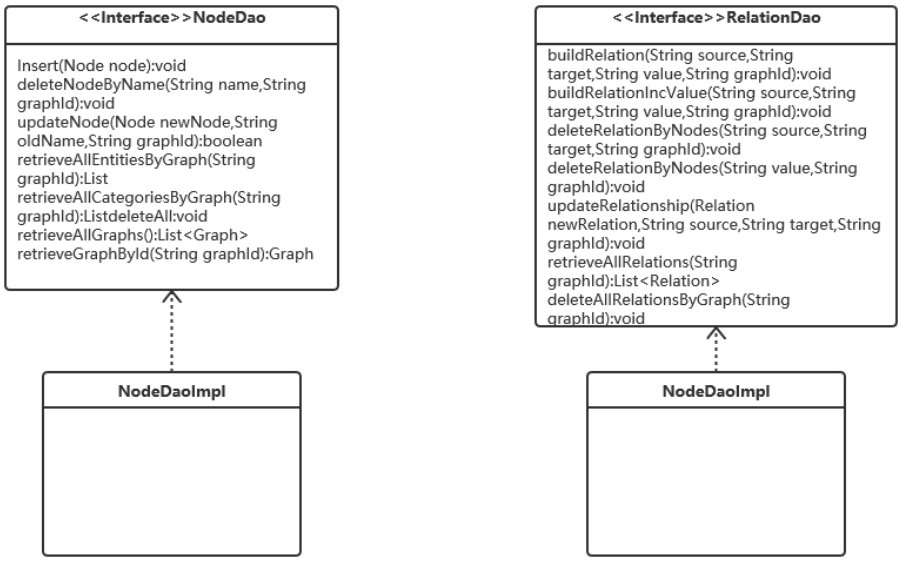
5.4 数据层分解

数据层主要给业务逻辑层提供数据访问服务，包括对于持久化数据的增删改查。

5.4.1职责

模块	职责
NodeDao	持久化Node对象的接口，提供涉及图谱中的实体的增删改查等操作
RelationshipDao	持久化Relation对象的接口，提供设计图谱中的关系的增删改查等操作

类图



5.4.2 接口规范

NodeDao接口规范

提供的服务（供接口）		
NodeDao.insert	语法	public void insert(Node node)
	前置条件	
	后置条件	在数据库中持久化node对象
NodeDao.deleteNodeByName	语法	public void deleteNodeByName(String name,String graphId)
	前置条件	
	后置条件	根据name查找指定实体所有持久化对象node并删除
NodeDao.updateNode	语法	public boolean updateNode(Node newNode, String oldName,String graphId)
	前置条件	原名对应的实体node存在
	后置条件	根据oldName获取持久化对象，用newNode替换对应数据
NodeDao.retrieveAllEntitiesByGraph	语法	public List retrieveAllEntitiesByGraph(String graphId)
	前置条件	
	后置条件	获取所有实体的持久化数据

NodeDao.retrieveAllCategoriesByGraph	语法	public List retrieveAllCategoriesByGraph(String graphId)
	前置条件	
	后置条件	获取所有类目的持久化数据
NodeDao.deleteAll	语法	public void deleteAll()
	前置条件	
	后置条件	删除所有持久化数据
NodeDao.retrieveGraphById	语法	public Graph retrieveGraphById(String graphId)
	前置条件	
	后置条件	返回指定图谱
NodeDao.getGraphList	语法	public List retrieveAllGraphs()
	前置条件	
	后置条件	返回用户所有持久化的图谱结构

RelationshipDao接口规范

提供的服务（供接口）		
RelationshipDao.buildRelation	语法	public void buildRelation(String source, String target, String value,String graphId)
	前置条件	导入的数据格式正确
	后置条件	在数据库中持久化Relation对象
RelationshipDao.buildRelationIncValue	语法	public void buildRelationIncValue(String source, String target, String value,String graphId)
	前置条件	
	后置条件	新建关系，更新关系所关联的实体，将结果持久化保存
RelationshipDao.deleteRelationByNodes	语法	public void deleteRelationByNodes(String source, String target,String graphId)
	前置条件	关联的节点与边在持久化数据中存在
	后置条件	根据source和target查找对应的关系，在持久化数据中删除
RelationshipDao.deleteRelationByValue	语法	public void deleteRelationByNodes(String value, String graphId)
	前置条件	关联的节点与边在持久化数据中存在

	后置条件	根据source和target查找对应的关系，在持久化数据中删除
RelationshipDao.updateRelationship	语法	public boolean updateRelation(Relation newRelation, String source, String target, String graphId)
	前置条件	source,target和newRelation关联的实体在持久化数据中存在
	后置条件	根据source和target查找关系，使用newRelation替换并完成持久化
RelationshipDao.retrieveAllRelations	语法	public List retrieveAllRelations(String graphId)
	前置条件	
	后置条件	获取所有关系的持久化数据
RelationshipDao.deleteAllRelationsByGraph	语法	public void deleteAllRelationsByGraph(String graphId)
	前置条件	
	后置条件	删除指定图谱所有关系的持久化数据

6.信息视角

- **Graph类**

Graph类包含描述关系图的实体Nodes，关系Links，类目Categories，定义如下：

```
public class Graph extends Node{




    private List<Entity> entities;
    private List<Relation> links;
    private List<Category> categories;

    private Map<String, Object> graphProperties;


    public Graph(){
        this.graphProperties = new HashMap<>();
        this.entities = new LinkedList<>();
        this.links = new LinkedList<>();
        this.categories = new LinkedList<>();
//        this.setLabel("graph");
    }

    public Graph(Map<String, Object> properties){
        this.graphProperties = new HashMap<>();
        this.entities = new LinkedList<>();
        this.links = new LinkedList<>();
        this.categories = new LinkedList<>();
        for(Map.Entry<String, Object> entry: properties.entrySet()){
            if("name".equals(entry.getKey())){
                super.setName(entry.getValue().toString());
            }
            else if("graphId".equals(entry.getKey())){
                super.setGraphId(entry.getValue().toString());
            }
            else{
                graphProperties.put(entry.getKey(), entry.getValue());
            }
        }
    }
    @Override
    public String getPropertiesAsString(String variable) {
        StringBuilder myProperties = new StringBuilder();
        Map<String, Object> viewProperties = new HashMap<>();
        viewProperties.put("name", this.getName());
        viewProperties.put("graphId", this.getGraphId());
        transProperties2Str(variable,myProperties, viewProperties);
        transProperties2Str(variable,myProperties, graphProperties);
        return deleteEndingComma(myProperties);
    }
    public void setGraphBasicInfo(Map<String, Object> graphBasicInfo){
        for(Map.Entry<String, Object> entry: graphBasicInfo.entrySet()){

            if("name".equals(entry.getKey())){
                super.setName(entry.getValue().toString());
            }
            else if("graphId".equals(entry.getKey())){
                super.setGraphId(entry.getValue().toString());
            }
            else{
```

```
                this.graphProperties.put(entry.getKey(), entry.getValue());
            }
        }
    }
    /**
     * 向graph中插入节点，节点不允许为null
     * @param node
     */
    public boolean addEntities(Entity node){
        return this.entities.add(node);
    }
    public boolean addRelations(Relation relation){
        return this.links.add(relation);
    }

    public List<Entity> getEntities() {
        return entities;
    }

    public void setEntities(List<Entity> nodes) {
        this.entities = nodes;
    }

    public List<Relation> getLinks() {
        return links;
    }

    public void setLinks(List<Relation> links) {
        this.links = links;
    }

    public List<Category> getCategories() {
        return categories;
    }

    public void setCategories(List<Category> categories) {
        this.categories = categories;
    }


    public Map<String, Object> getGraphProperties() {
        return graphProperties;
    }

    public void setGraphProperties(Map<String, Object> graphProperties) {
        this.graphProperties = graphProperties;
    }

}
```

类型:

1. nodes:List
2. links:List
3. categories:List
4. graphId:String

- **Node类**

Node类为节点抽象类，提取了Graph,Category,Entity三种节点类型的共同属性和方法定义如下：

```
public abstract class Node {

    private String name;
    private String graphId;


    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public abstract String  getPropertiesAsString(String variable);


    public String getGraphId() {
        return graphId;
    }

    public void setGraphId(String graphId) {
        this.graphId = graphId;
    }
}
```

类型：

1. name:String
2. graphId:String

- **Relation类**

Relation类定义如下：

```
public class Relation {


    private String graphId;

    private String source;

    private String target;

    private String value;

    private Map<String, Object> styles;
```

```
public Relation(){}
public Relation(String source, String target, String value){
    this.source = source;
    this.target = target;
    this.value = value;
}
public Relation(String source, String target, String value, String graphId){
    this.graphId = graphId;
    this.source = source;
    this.target = target;
    this.value = value;
}


public String getSource() {
    return source;
}

public void setSource(String source) {
    this.source = source;
}

public String getTarget() {
    return target;
}

public void setTarget(String target) {
    this.target = target;
}

public String getValue() {
    if(this.value == null) {
        return "null";
    }
    else {
        return this.value;
    }
}

public void setValue(String value) {
    this.value = value;
}

public String getGraphId() {
    return graphId;
}

public void setGraphId(String graphId) {
    this.graphId = graphId;
}


public Map<String, Object> getStyles() {
    return styles;
}

public void setStyles(Map<String, Object> styles) {
```

```
        this.styles = styles;
    }
}
```

类型：

1. source:String
2. target:String
3. value:String
4. graphId:String
5. styles:Map<String, Object>

- **Category类**

Category类定义如下：

```
public class Category extends Node{

    private Map<String, Object> categoryProperties;


    public Category(){
        this.setName("未命名");
    }
    public Category(Map<String, Object> properties){
        this.categoryProperties = new HashMap<>();
        for(Map.Entry<String, Object> entry: properties.entrySet()){
            if("name".equals(entry.getKey())){
                super.setName(entry.getValue().toString());
            }
            else if("graphId".equals(entry.getKey())){
                super.setGraphId(entry.getValue().toString());
            }
        }
    }
    public Category(String name, String graphId){
        this.categoryProperties = new HashMap<>();
//        this.setLabel("category");
        this.setName(name);
        this.setGraphId(graphId);
    }
    @Override
    public String getPropertiesAsString(String variable) {
        StringBuilder myProperties = new StringBuilder();
        Map<String, Object> viewProperties = new HashMap<>();
        viewProperties.put("name", this.getName());
        viewProperties.put("graphId", this.getGraphId());
        transProperties2Str(variable,myProperties, viewProperties);
        transProperties2Str(variable,myProperties, categoryProperties);
        return deleteEndingComma(myProperties);
    }


    public Map<String, Object> getCategoryProperties() {
```

```
        return categoryProperties;
    }

    public void setCategoryProperties(Map<String, Object> categoryProperties) {
        this.categoryProperties = categoryProperties;
    }


}
```

- **类型**:

    1. name:String
    2. private Map<String, Object> categoryProperties
    3. graphId:String

- **Entity类**

```
public class Entity extends Node {


    private Map<String, Object> entityProperties;
    private Map<String, Object> contextProperties;
    private static String contextLabel = "entity";



    public Entity(){}
    public Entity(String name, int category){
        this.setEntitiesProperties("value", 1);
        this.setEntitiesProperties("category", category);
        this.setEntitiesProperties("symbol","circle");
        super.setName(name);
        this.setEntitiesProperties("symbolSize",
                10.0 +
Double.parseDouble(String.valueOf(this.getEntitiesPropertiesByName("value"))) / 2
);

    }
//    public Entity(String name, double symbolSize, double x, double y, long
value, int category){
//        super.setName(name);
//        this.setEntitiesProperties("symbolSize", symbolSize);
//        this.setEntitiesProperties("x", x);
//        this.setEntitiesProperties("y", y);
//        this.setEntitiesProperties("value", value);
//        this.setEntitiesProperties("category", category);
//        this.setEntitiesProperties("symbol","circle");
//    }
    public Entity(Map<String, Object> properties){
        this.entityProperties = new HashMap<>();
        this.contextProperties = new HashMap<>();
        for(Map.Entry<String, Object> entry: properties.entrySet()){
            if(EntityProperties.isNodeStyle(entry.getKey())){
                this.entityProperties.put(entry.getKey(), entry.getValue());
```

```
            }
            else if("name".equals(entry.getKey())){
                super.setName(entry.getValue().toString());
            }
            else if("graphId".equals(entry.getKey())){
                super.setGraphId(entry.getValue().toString());
            }
            else{
                this.contextProperties.put(entry.getKey(), entry.getValue());
            }
        }
    }
    @Override
    public String getPropertiesAsString(String variable){
        StringBuilder myProperties = new StringBuilder();
        Map<String, Object> viewProperties = new HashMap<>();
        viewProperties.put("name", this.getName());
        viewProperties.put("graphId", this.getGraphId());

        transProperties2Str(variable,myProperties,viewProperties);
        transProperties2Str(variable, myProperties, contextProperties);
        transProperties2Str(variable, myProperties, entityProperties);
        return deleteEndingComma(myProperties);
    }



    @Override
    public boolean equals(Object obj){
        if(this == obj) {
            return true;
        }
        if(obj == null) {
            return false;
        }
        if(obj instanceof Entity) {
            Entity temp = (Entity) obj;
            return this.getName().equals(temp.getName()) &&
this.getGraphId().equals(temp.getGraphId());
        }
        return false;
    }
    @Override
    public int hashCode(){
        int result = 17;
        result = 31 * result + (this.getName() == null ? 0 :
this.getName().hashCode());
        result = 31 * result + (this.getGraphId() == null ? 0 :
this.getGraphId().hashCode());
        return result;
    }



    public Object getEntitiesPropertiesByName(String name){
        return this.entityProperties.get(name);
    }
    public void setEntitiesProperties(String key, Object value){
```

```
        this.entityProperties.put(key, value);
    }
    public Map<String, Object> getentityProperties() {
        return entityProperties;
    }

    public void setentityProperties(Map<String, Object> entityProperties) {
        this.entityProperties = entityProperties;
    }

    public Map<String, Object> getContextProperties() {
        return contextProperties;
    }

    public void setContextProperties(Map<String, Object> contextProperties) {
        this.contextProperties = contextProperties;
    }
}
```

- **类型**:

1. entityProperties: Map<String, Object>
2. contextProperties: Map<String, Object>
3. contextLabel:String
4. graphId:String