迭代一项目设计文档

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	初稿	v1.0
2020年3月9日	陈峙宇	完成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	整合并补充更改文档	v1.3
2020年3月21日	李泳劭	添加jenkins脚本说明	v1.4

1.引言

1.1 编制目的

本报告详细完成对COIN知识图谱系统迭代1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

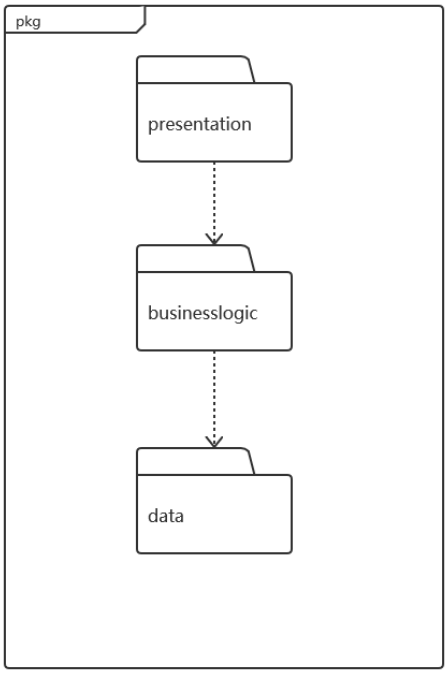
2.COIN知识图谱定义及可视化系统 ——房春荣

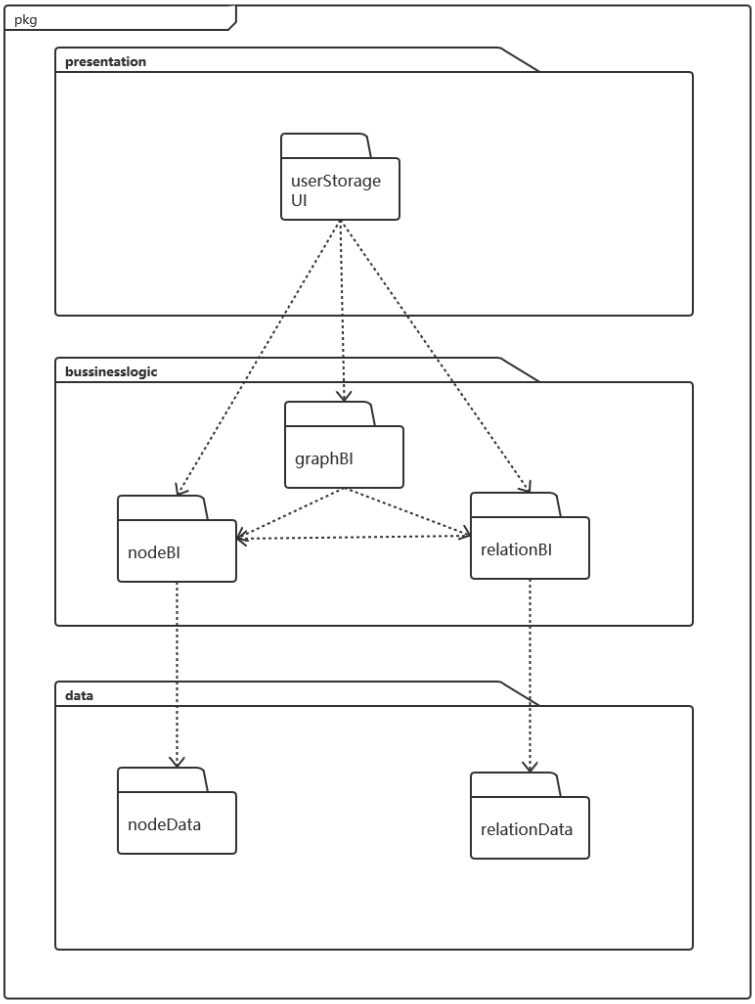
2.产品概述

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

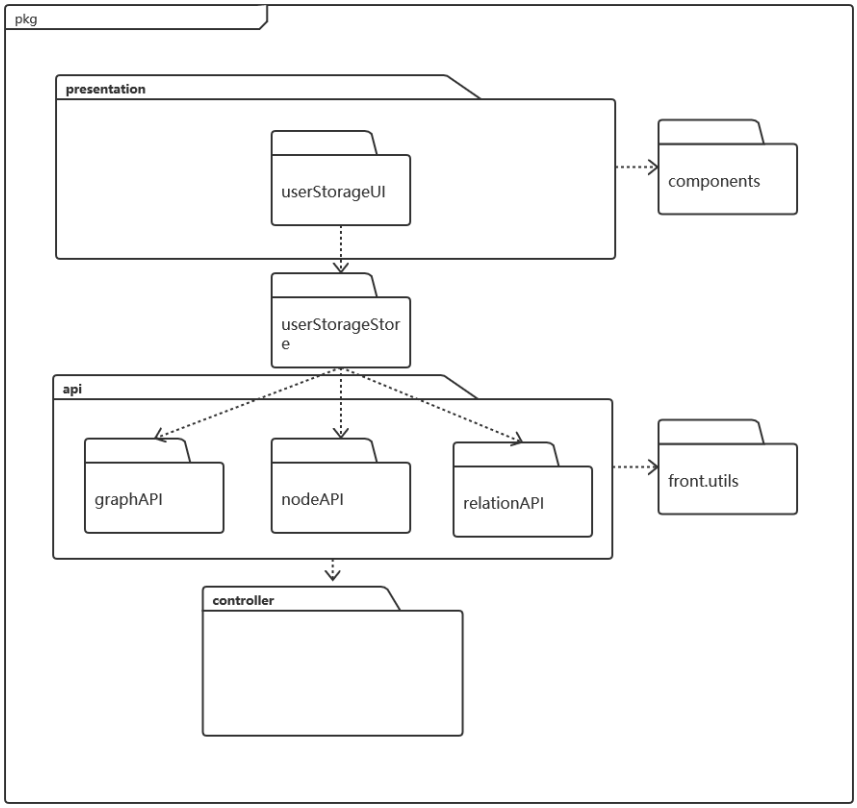
4.1开发包图

- 描述开发包以及相互间的依赖

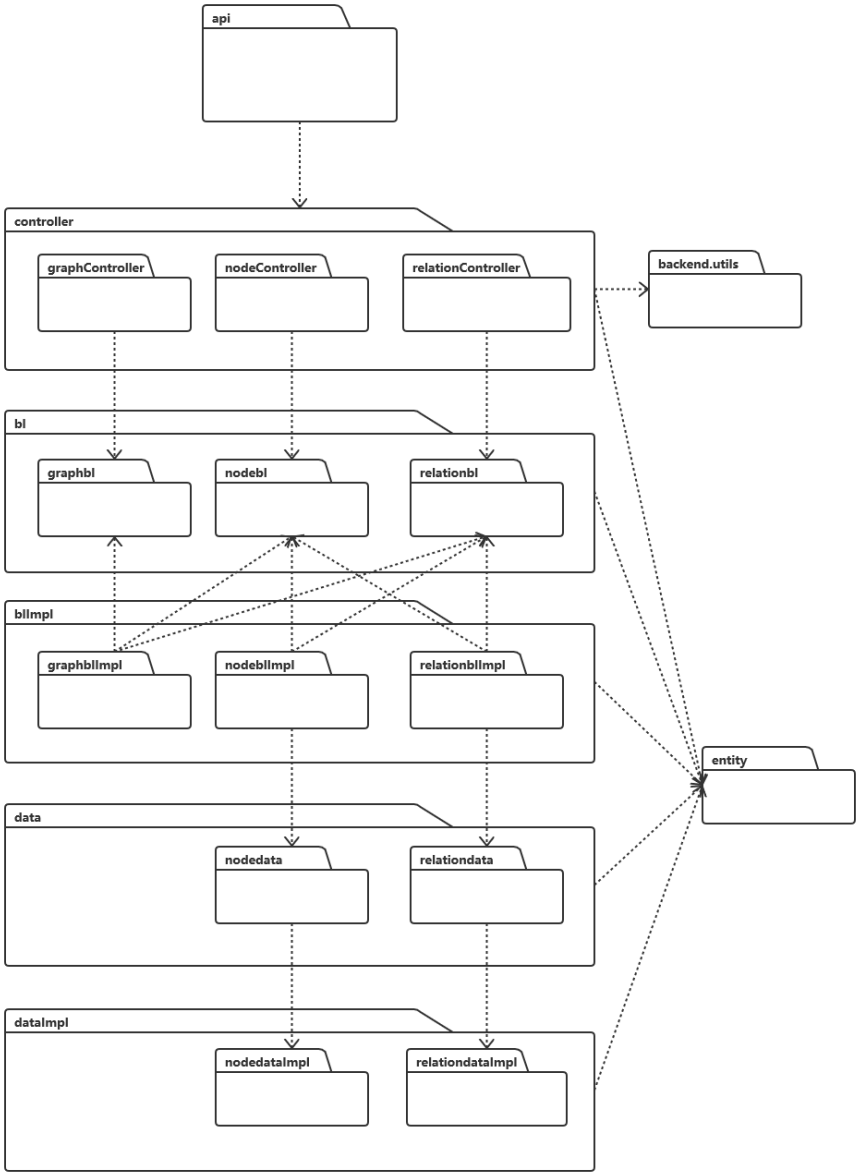
开发包	依赖的其他开发包

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.userStorage	api.graph,api.node,api.relation
presentation.userStorage	store.graph,components
front.util	
components	

- 绘制开发包



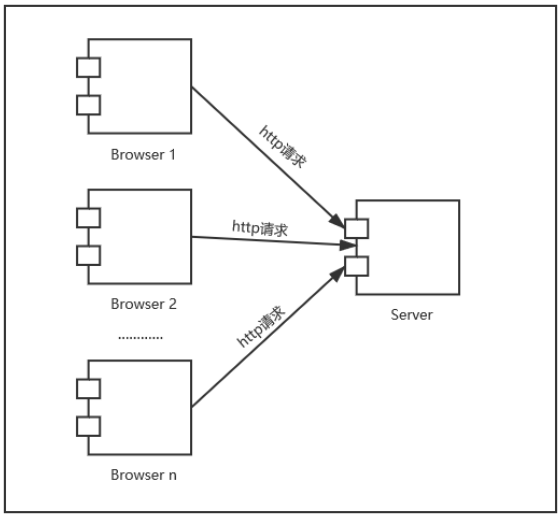
客户端开发包图



服务端开发包图

4.2 运行时进程

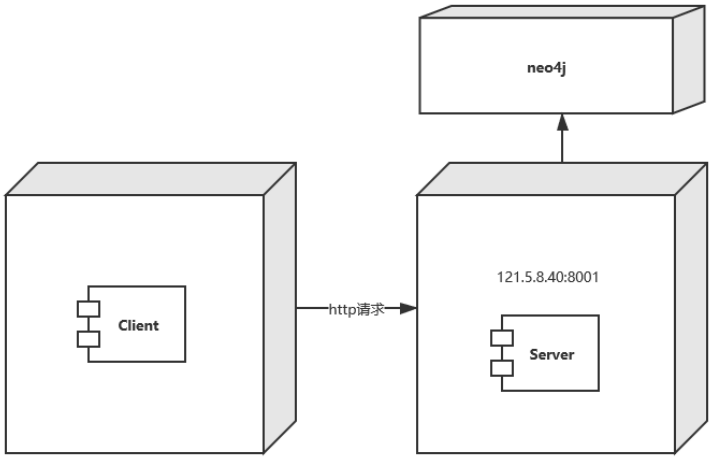
在COIN知识图谱系统中，会有多个浏览器进程和一个服务端进程，其进程图如下



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

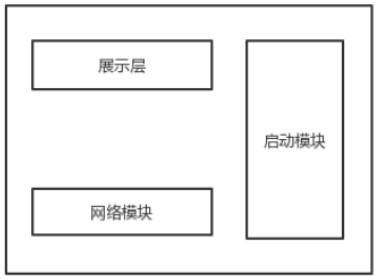


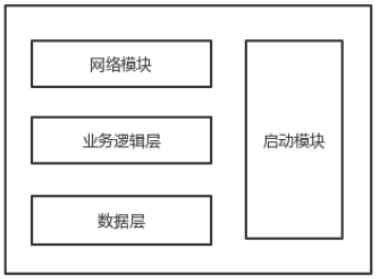
5.架构设计

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

根据需求，系统仅存在一个主界面 `Index` ，主界面上方为导航栏，左侧区域预留加载信息栏，中部为图谱可视化区域，右侧为操作栏，暂无页面跳转。

5.2.1 职责

模块	职责
index	负责导入、编辑、导出知识图谱

5.2.2 接口规范

index接口规范

提供的服务（供接口）		
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
需要的服务(需接口）		
服务名	服务	
GraphService.importGraphFromFile	构建并向数据库中插入从文件导入的Graph对象	
NodeService.retrieveAll	从数据库中获取完整图谱数据	
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.deleteByName	在数据库中删除对应名的关系	
RelationshipService.updateRelation	在数据库中更新对应关系	

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。

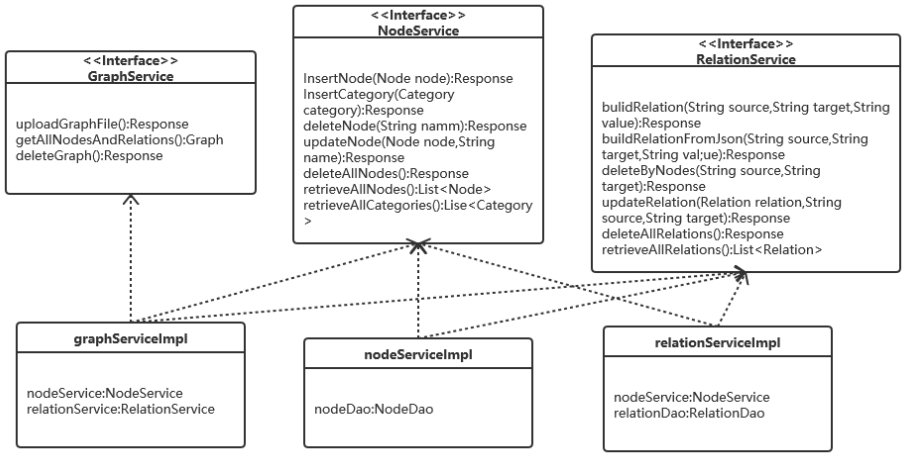
5.3 业务逻辑层的分解

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.uploadGraphFile()	语法	public Response uploadGraphFile()
	前置条件	用户上传图谱json文件
	后置条件	返回添加结果
GraphService.getAllNodesAndRelations	语法	public Response getAllNodesAndRelations()
	前置条件	用户载入界面
	后置条件	返回所有关系和节点
GraphService.deleteGraph	语法	public Response deleteGraph()
	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.retrieveAllNodes()	获取所有持久化Node对象	
nodeService.deleteAllNodes()	删除所有持久化Node对象	
RelationService.buildRelationFromJson(Relation relation)	持久化Relation对象	
RelationService.retrieveAllRelations()	获取所有持久化Relation对象	
RelationService.deleteAllRelations()	删除所有持久化Relation对象	

NodeService接口规范

提供的服务（供接口）		
NodeService.InsertNode	语法	public Response InsertNode(Node node)
	前置条件	用户创建新节点
	后置条件	返回添加结果
NodeService.InsertCategory	语法	public Response InsertCategory(Category category)
	前置条件	用户创建新节点时设置未存在的类目
	后置条件	返回添加结果
NodeService.deleteNodeByName	语法	public Response deleteNodeByName(String name)
	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 Response updateNode(Node node,String name)
	前置条件	用户更新节点信息
	后置条件	返回更新结果
NodeService.retrieveAllNodes	语法	public List retrieveAllNodes()
	前置条件	用户载入界面
	后置条件	返回节点列表
NodeService.deleteAllNodes	语法	public Response deleteAllNodes()
	前置条件	用户删除图表
	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.retrieveAllNodes()	从数据库中获取所有Node对象
NodeDao.retrieveAllCategories()	从数据库中获取所有Category
NodeDao.insert(Node node)	往数据库中插入Node对象
NodeDao.insert(Category category)	往数据库中插入category对象
NodeDao.deleteNodeByName(String name)	从数据库中删除指定Node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 Response buildRelation(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.buildRelationFromJson	语法	public Response buildRelationFromJson(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 Response 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 Response 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.deleteAllRelations()	语法	public Response deleteAllRelations()
	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()
	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)	往数据库中插入Relation对象	
RelationDao.buildRelationFromJson(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.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()	从数据库中获取所有Relation对象	
RelationDao.deleteAll()	从数据库中获取所有Relation对象	
NodeService.updateNode(Node node,String name)	更新数据库中指定Node对象	

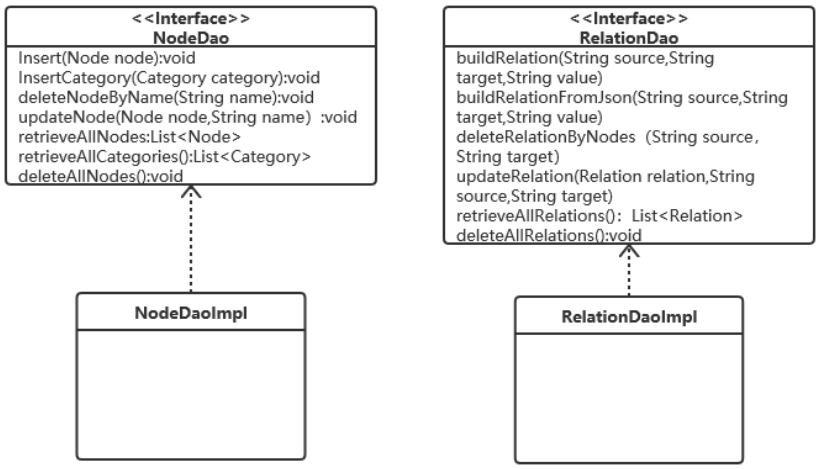
5.4 数据层分解

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.insertCategory	语法	public void insertCategory(Category category)
	前置条件	实体属于一个尚未创建的category
	后置条件	新建category并持久化该类目
NodeDao.deleteNodeByName	语法	public void deleteNodeByName(String name)
	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)
	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.retrieveAllNodes	语法	public List retrieveAllNodes()
	前置条件	
	后置条件	获取所有实体的持久化数据
NodeDao.retrieveAllCategories	语法	public List retrieveAllCategories()
	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.retrieveAll	语法	public Graph retrieveAll()
	前置条件	
	后置条件	返回包含所有持久化数据的图谱结构

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)
	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.buildRelationFromJson	语法	public void buildRelationFromJson(String source, String target, String value)
	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)
	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)
	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()
	前置条件	
	后置条件	获取所有关系的持久化数据
RelationshipDao.delete	语法	public List deleteAllRelations()
	前置条件	
	后置条件	删除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 {
    private List<Node> nodes;
    private List<Relation> links;
    private List<Category> categories;
    public Graph(){
        this.nodes = new LinkedList<>();
        this.links = new LinkedList<>();
        this.categories = new LinkedList<>();
    }
    /**

    *向graph中插入节点，节点不允许为null
    *@param node
    */

public boolean addNodes(Node node){
```

```
            return this.nodes.add(node);
        }
        public boolean addRelations(Relation relation){
            return this.links.add(relation);
        }
        public List<Node> getNodes() {
            return nodes;
        }
        public void setNodes(List<Node> nodes) {
            this.nodes = 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;
        }
    }
```

类型：

1. nodes:List
2. links:List
3. categories:List

- **Node类**

  Node类包含描述实体的名字name，值value，坐标x、y，大小symbolSize，类目category，定义如下：

```
    public class Node {
        private int category;
        private String name;
        private double symbolSize;
        private double x;
        private double y;
        private long value;
        public Node(){}
        public Node(String name, long value){
            this.category = 0;
            this.name = name;
            this.value = value;
            symbolSize = (10.0 + Double.parseDouble(String.valueOf(value))) / 2;
        }
        public Node(String name, int category){
            this.value = 1;
            this.category = category;
            this.name = name;
            symbolSize = (10.0 + Double.parseDouble(String.valueOf(value))) / 2;
        }
```

```
21      public Node(String name, double symbolSize, double x, double y, long
    value, int category){
22          this.name = name;
23          this.symbolSize = symbolSize;
24          this.x = x;
25          this.y = y;
26          this.value = value;
27          this.category = category;
28      }
29      @Override
30
31   public boolean equals(Object obj){
32          if(this == obj)
33              return true;
34          if(obj == null)
35              return false;
36          if(obj instanceof Node) {
37              Node temp = (Node) obj;
38              return this.getName().equals(temp.getName());
39          }
40          return false;
41      }
42      @Override
43      public int hashCode(){
44          int result = 17;
45          result = 31 * result + (this.name == null ? 0 : name.hashCode());
46          return result;
47      }
48      public String getName() {
49          return name;
50      }
51      public void setName(String name) {
52          this.name = name;
53      }
54      public int getCategory() {
55          return category;
56      }
57      public void setCategory(int category) {
58          this.category = category;
59      }
60      public double getSymbolSize() {
61          return symbolSize;
62
63  }
64      public void setSymbolSize(double symbolSize) {
65          this.symbolSize = symbolSize;
66      }
67      public long getValue() {
68          return value;
69      }
70      public void setValue(long value) {
71          this.value = value;
72      }
73      public double getX() {
74          return x;
75      }
76      public void setX(double x) {
77          this.x = x;
```

```
    }
    public double getY() {
        return y;
    }
    public void setY(double y) {
        this.y = y;
    }
}
```

类型：

1. name:String
2. value:long
3. x:double
4. y:double
5. symbolSize:double
6. category:int

- **Relation类**

Relation类包含描述关系的源节点名source，目标节点名target，关系类型value，定义如下：

```
public class Relation {
    private String source;
    private String target;
    private String value;
    public Relation(){}
    public Relation(String source, String target, String value){
        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;
    }

}
```

类型：

1. source:String

2. target:String
3. value:String

- **Category类**

  Category类包含描述类目的名字name，定义如下：

```
public class Category {
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Category(){
        this.name = "类目0";
    }
    public Category(String name){
        this.name = name;
    }
}
```

  类型：

1. name:String

7. 持续集成

本项目采用Docker+Jenkins创建PipelineJob，Hook到Gitlab的Master分支变更，实现自动化的前、后端持续集成。

7.1 后端脚本

- **Dokerfile**

```
FROM java:8
ADD ./target/nekocoin-0.0.1-SNAPSHOT.jar /app/nekocoin-0.0.1-SNAPSHOT.jar
ADD ./target/site/jacoco/index.html /app/
ADD runboot.sh /app/
WORKDIR /app
RUN chmod a+x runboot.sh
CMD /app/runboot.sh
```

- **Jenkinsfile**

```
pipeline {
    agent{
        label 'master'
    }

    stages {
        stage('Maven Build and Test') {
            agent{
                docker {
                    image 'maven:latest'
                    args '-v /root/.m2:/root/.m2'
```

```
                }
            }
            steps{
                echo 'Maven Build Stage'
                // sh 'mvn -DskipTests=true package '
                sh 'mvn clean'
                sh 'mvn test jacoco:report'
                sh 'mvn package'

            }
        }
        stage('Image Build'){
            agent{
                label 'master'
            }
            steps{
                echo 'Image Build Stage'
                sh "docker build . -t nekocoin:${BUILD_ID}"

            }
        }

        stage('deploy'){
            agent{
                label 'master'
            }
            steps{
                sh "if (docker ps -a |grep nekocoin) then (docker stop nekocoin && docker rm nekocoin) fi"
                sh "docker run -p 8001:8001 --name nekocoin -v /log:/log -d nekocoin:${BUILD_ID}"


            }
        }
    }
}
```

7.2 前端脚本

- **Dockerfile**

```
FROM nginx
RUN mkdir /feapp
COPY ./dist /feapp
COPY nginx.conf /etc/nginx/nginx.conf
```

- **Jenkinsfile**

```
pipeline{
  agent any
  tools { nodejs "nodejs" }
  stages{
    stage('Image Clear'){
      steps{
        echo 'Image Clear Stage'
```

```
        sh "if (docker ps|grep nekocoin-fe) then (docker container stop nekocoin-fe && docker container rm nekocoin-fe) fi"
        sh "if (docker images|grep nekocoin-fe) then (docker rmi \$(docker images nekocoin-fe -q)) fi"
      }
    }
    stage('Build'){
      steps{
        echo 'Build Stage'
        sh "npm install -g cnpm --registry https://registry.npm.taobao.org"
        sh "cnpm install"
        sh "npm run build"
        sh "docker build -t nekocoin-fe:${BUILD_ID} . "
      }
    }
    stage('Deploy'){
      steps{
        sh "docker run -p 8000:80 --name nekocoin-fe -v /log:/log -d nekocoin-fe:${BUILD_ID}"
      }
    }
  }
}
```

- **Nginx.conf**

```
user  nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;
events {
  worker_connections  1024;
}
http {
  proxy_read_timeout 300;
  include       /etc/nginx/mime.types;
  default_type  application/octet-stream;
  log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
  '$status $body_bytes_sent "$http_referer" '
  '"$http_user_agent" "$http_x_forwarded_for"';
  access_log  /var/log/nginx/access.log  main;
  sendfile        on;
  keepalive_timeout  65;
  server {
    listen       80;
    server_name  http://121.5.8.40;  # 域名
    location / {
      root   /feapp;  # 指向目录
      index  index.html;
      try_files $uri $uri/ /index.html;
    }
    location /api{
      rewrite ^/api/(.*)$ /$1 break;
      proxy_pass http://121.5.8.40:8001;
    }
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
      root   /usr/share/nginx/html;
```

```
33          }
34        }
35      }
36
```