

**软件课程设计报告**

**题目： 在线网络演化可视化系统设计**

**课程名称： 软件课程设计**

**专业班级： 提高1801**

**学 号： U201813464**

**姓 名： 何承民**

**指导教师： 王邦**

**报告日期： 2021.01.16**

**电子信息与通信学院**

目 录

[项目描述 2](#_Toc2235596)

系统描述 [3](#_Toc2235599)

[模型描述 4](#_Toc2235603)

[软件设计 4](#_Toc2235607)

[模块层级图 4](#_Toc2235610)

[技术报告 4](#_Toc2235611)

[总结与建议 4](#_Toc2235612)

[参考资料 6](#_Toc2235613)

[附录1 源码 7](#_Toc2235614)

[附录2 软件说明书 10](#_Toc2235615)

# 项目描述

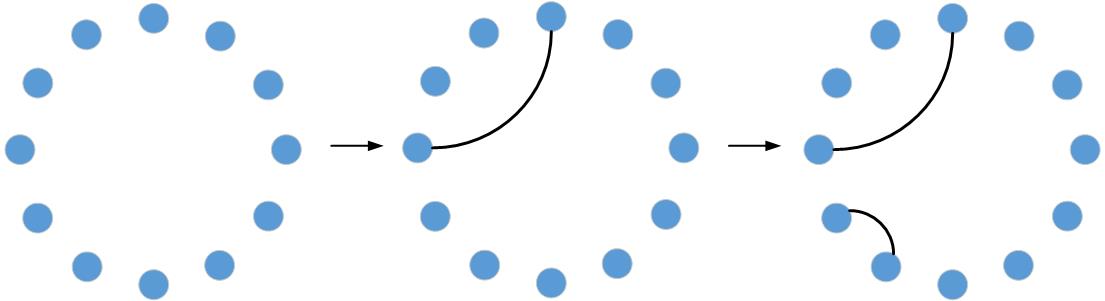
## 设计内容

设计一个可以在网页上对网络的演化过程可视化的工具，并自选编程语言与小组成员合作开发完成设计任务。

## 设计要求

要求具有如下基本功能：

1. 合理设计网页布局，包括：用户交互区、可视化区，指标计算区等
2. 用户能指定网络类型和网络规模。系统实现不同网络类型对应的生成机制，构造指定规模网络（包括：ER网络、WS网络、BA网络），并动态可视化该过程；下图为网络的生成过程某一阶段的拓扑结构变化，要求能动态显示完整过程



1. 用户能指定网络的失效机制。系统实现不同失效机制对应的算法（基本：度（degree）中心性、介数（betweeness）中心性、接近（closeness）中心性、特征向量（eigenvector）中心性、Katz中心性；选做：PageRank），作用在完整网络上，并可视化该过程；下图为网络的失效过程某一阶段的拓扑结构变化，要求能动态显示完整过程；
2. 在网络演化过程可视化的同时，显示对应网络结构的相关统计指标（包括：度分布、平均路径长度、聚类系数）
3. 用户能查看网络演化过程中的任意拓扑结构及其相关统计指标。系统需通过数据库实现信息的存储与交互。

# 系统描述­­

## web系统布局结构描述

本次软件课程设计的web系统界面布局分为“一个主页”+“四个分页”的设计。

主页设计：以一个瀑布流的组图布局为主体布局，外加header部分的导航栏部分。总体以简单干净为主的设计，瀑布流设计分为了三个模块分别介绍了本次设计的三种网络模型及其网络的四种失效网络算法，和我们用到的一些重要工具库和工具包。

子页设计：承继主页简洁干净的特点，header部分的跳转导航栏，主体部分为上面的功能选择区，下面的视图显示区，视图显示区分为左右两个图表。

主页和子页都可以相互的无缝连接，实现用户的交互选择，为了增加系统交互性，每次用户的功能选择都会有相应的弹窗提示，也会用运行进行中显示，来提示系统进行状态，美观简洁的用户交互设计增强系统的用户体验感。

## 系统功能描述

首先用户进入首页，可以选择四个功能模块，分别是ER网路，WS网络，BA网络的系统模拟和文件上传分析这四个模块。

三种网络模块的系统功能类似，用户能指定网络类型和网络规模。系统实现不同网络类型对应的生成机制，构造指定规模网络，然后系统就会生成两张echarts图表，左边是网络演化拓扑图，右边是统计度分布的统计计量图表。与此同时，生成之后用户能指定网络的失效机制。在网络演化过程可视化的同时，显示对应网络结构的相关统计指标（节点id，节点的度，度分布、平均路径长度、聚类系数等），并且也可以改变生成网络图的一些外观样式，比如节点的大小，节点的颜色等。

文件上传功能模块是提供给用户需要对自己的网络模型进行可视化操作提供的功能需求，用户可以上传关于本地的网络节点信息的json文件，对应系统模块就可以在后台分析该json文件，然后可视化出用户想要的任意拓扑结构和统计信息。

## 主要概念描述

1. 什么是ER网络：E-R图又称实体关系图，是一种提供了实体，属性和联系的方法，用来描述现实世界的概念模型。通俗点讲就是，当我们理解了实际问题的需求之后，需要用一种方法来表示这种需求，概念模型就是用来描述这种需求。比如学生生活中的校园卡系统
2. 什么是WS网络：也叫小世界网络，小世界网络是一种数学图,在社交网络中，这种网络属性意味着一些彼此并不相识的人，可以通过一条很短的熟人链条被联系在一起，这也就是小世界现象。
3. 什么是BA网络：前人为了解释幂律的产生机制，提出了无标度网络模型（BA模型）。BA模型具有两个特性，其一是增长性，所谓增长性是指网络规模是在不断的增大的；其二就是优先连接机制，这个特性是指网络当中不断产生的新的节点更倾向于和那些连接度较大的节点相连接。
4. 中心性：度中心性（在无向网络中，我们可以用一个节点的度数来衡量中心性），接近中心性（如果节点到图中其他节点的最短距离都很小，那么它的接近中心性就很高。），中介中心性（如果一个成员位于其他成员的多条最短路径上，那么该成员就是核心成员，就具有较大的中介中心性。）特征向量中心性（一个节点的中心性是相邻节点中心性的函数。也就是说，与你连接的人越重要，你也就越重要。），katz中心性（katz centrality通过测量相邻节点（一级节点）和网络中通过这些相邻节点连接到考虑中节点的所有其他节点的数量，计算网络中节点的相对影响。）

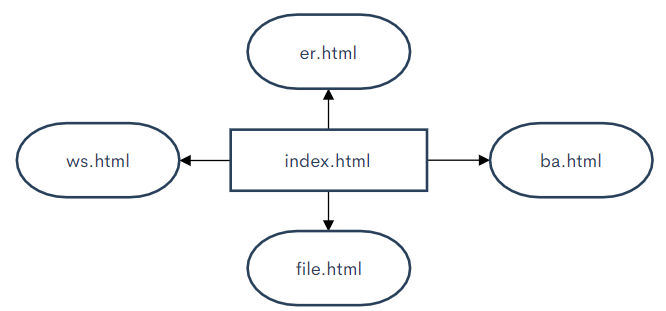
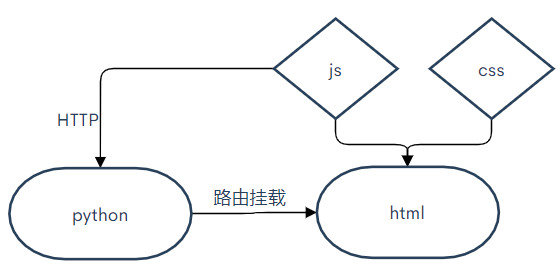
# 模型描述

## 2.1前端模型设计

前端模型设计前文也提及，一个主页面，四个子页面的结构设计，彼此也是通过<a>标签的相对路径相连通。

同后端连接和一系列页面效果通过<script>内部和外部脚本文件实现。

用下面这个框图来表示前端的模型架构。

前端页面在flask框架搭建的本地端口上运行，前端向后端发送数据通过HTTP异步传递，这里具体实现是利用jquery的getJSON()函数来实现HTTP GET数据派发，完成以json文件的形式传递用户输入的操作数。

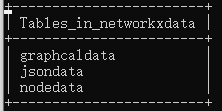
Js方面值得一提的就是echarts这个强大的js图形库，经过js脚本处理将拓扑图信息以对象的形式存储，然后可视化对应的option对象。我们就通过echarts的编写格式编写option对象就可以实现数据可视化操作，非常方便。

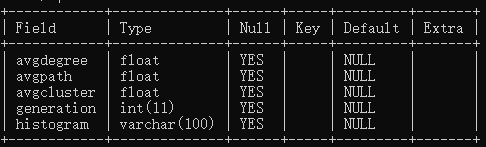
## 2.2数据库模型设计

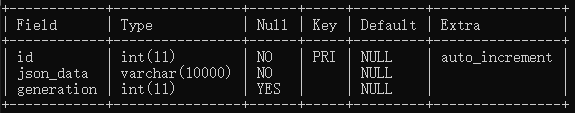
为满足时间旅行，数据库存放每一代图的拓扑结构，节点属性等信息。这里我们选择了非常优秀的mysql数据库。

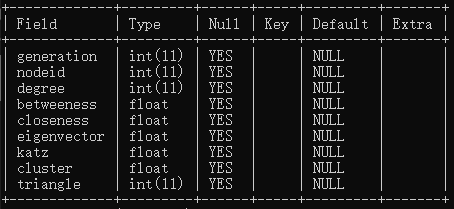
我们建立了networkxdata作为数据库名称，内置三个数据表（jsondata，graphcaldata，nodedata），分别来存放每一代图的拓扑信息，图的属性信息及图中的节点信息。每一个数据表通过其中的generation数据行彼此连接即，主键。

具体信息看下面图示：



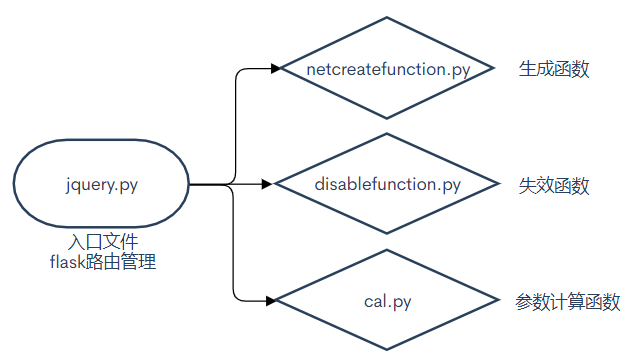




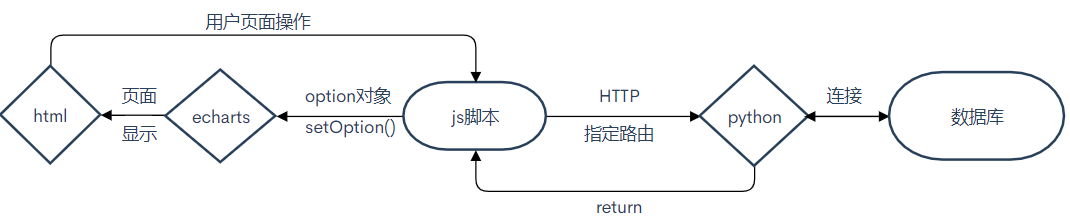


## 2.3后端模型设计

本系统主要功能在于，网络演化生成和失效的可视化模拟，所以我们具体分为netcreatefunction.py，disablefunction.py以及一个参数计算的部分cal.py这三个子模块和一个依托flask框架的入口程序jquery.py。



## 2.4模块层次图示



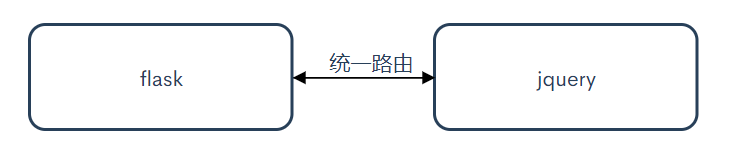
# 软件设计

## 3.1初级设计构想

本系统是一个web端的前后端分离的项目设计，首先应该考虑的是，前端使用什么 去实现可视化，后端怎么去计算来生成和处理数据，当然失效上面也要使用到时间旅行，也就是需要有对应数据库设计来存储一定的拓扑图信息。

其中，我们认为最主要的就是如何实现前端HTTP请求同后端程序实现对接。起初准备使用react作为我们前端的视图层框架，但是研究了许久如果需要使用python库networkx来生成处理数据时，两者不太很好的兼容，需要复杂的中间件，可能本人现前react功底不够，于是我们使用较为简单成熟的前后端结合模式“jquery+flask”。

首先我们测试如何从前端传一个数据，后端准确接手后处理返回结果，前端接受到并显示出来。于是我们测试了简单的“加法器”的实现。前端传递a，b两个参数，后端计算a+b并返回计算结果。



* Jquery：通过在服务器和客户端之间交换 JSON 数据是使得 Web 应用动态化的完美方式。

**注意点：**

1. 我们需要统一路由，

Python: @app.route('/\_add\_numbers')

jquery: $.getJSON($SCRIPT\_ROOT + '/\_add\_numbers',…)

1. URL 的根路径位置

服务器这边，这从来不是一个问题，原因是我们使用的 url\_for() 函数可以帮我们回答这个问题。

使用 jQuery 我们不应该将指向应用的路径硬编码到程序中，而是将它动态化----设定一个全局变量作为一个应用根路径的前缀(具体实现请参考附录代码)

1. 防止浏览器缓存

Ajax请求同时传递一个时间戳，now: new Date().getTime() 防止浏览器缓存的一种小技巧

## 3.2 算法设计实现

实现了前后端交互设计，也应证我们的设计架构是可行的，下面需要完成的就是这个可视化系统的算法分析是如何产生数据，处理数据，显示数据。

1. 生成模块算法设计：算法是基于networkx函数包完成的

# WR\_erdos\_renyi\_graph(n,p) 生成ER图，清除jsondata之前的数据，并写入新的生成数据

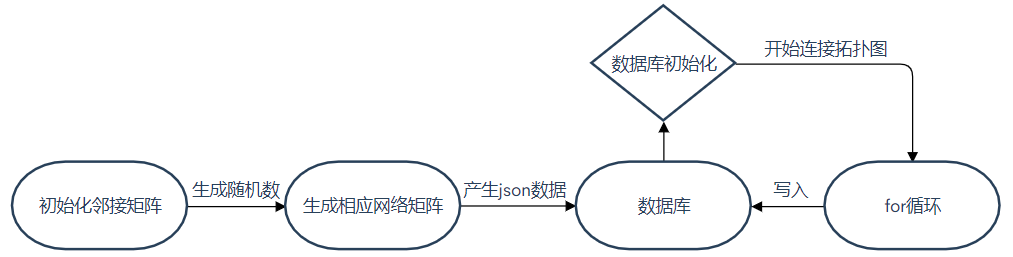
# n:ER网络的顶点个数 p:连边概率

# WR\_watts\_strogatz\_graph(n, k, p) 生成WS图，清除jsondata之前的数据，并写入新的生成数据

# n:WS网络的顶点个数 k:WS网络每个顶点邻居个数 p:重连边概率

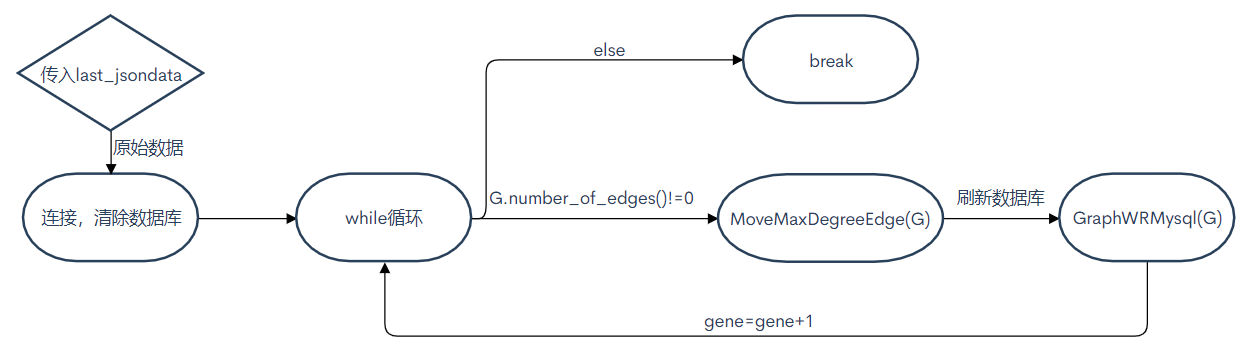
# WR\_barabasi\_albert\_graph(n, m) 生成BA图，清除jsondata之前的数据，并写入新的生成数据

# n:BA网络的顶点个数 m:BA网络每次加入边个数



1. 失效算法设计：五种失效算法分别设计，其中心思想类似，都是找到对应中心性的最大值的节点，然后清除对应节点所有的连边，直到恢复最初的节点图。下面以其中一个为例：

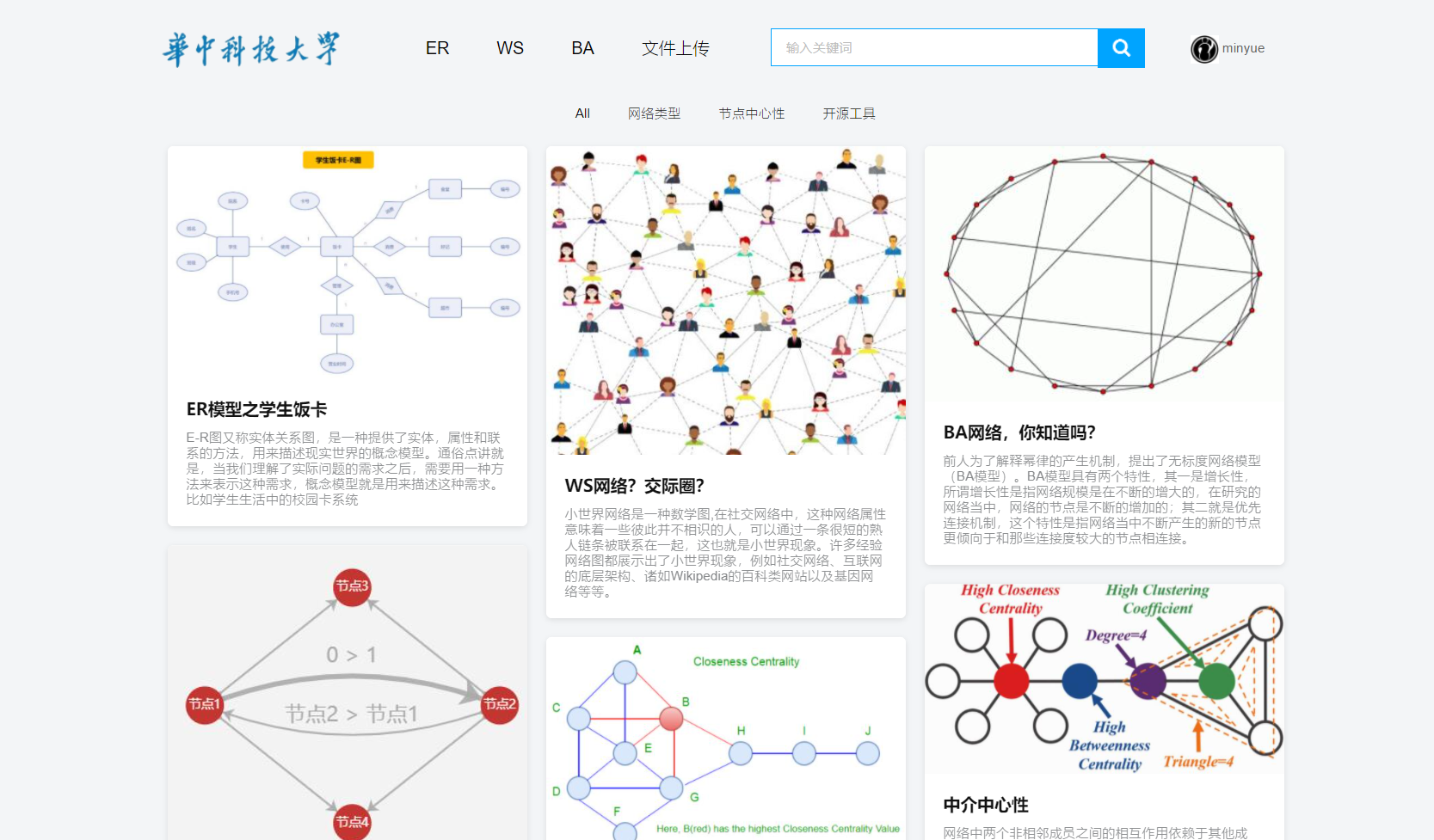
# WR\_degreedisable() 度失效当前数据库jsondata最大generation图，清除jsondata之前的数据，并写入新的失效数据



## 3.3 前端页面设计

前文前端模型设计中对前端页面的布局设计有了一些的介绍，这里我主要介绍一下echarts这个强大的js前端库，以及为什么我们要使用这个库，如何使用？

* Echars：是一个用纯JavaScript编写的一个图表库， 能够很简单便捷的在web网站或是web应用程序添加有交互性的图表，并且免费提供给个人学习、个人网站和非商业用途使用。echarts支持的图表类型有曲线图、区域图、柱状图、饼状图、散状点图和综合图表。
* 为什么选择echarts：现在比较流行的js图形库就是D3和echarts，而且在networkx官网上给的典型实例就是使用D3完成的，为什么不使用D3而使用echarts，原因很简单，D3很不友好！！！，D3默认的解析格式是本地json文件，这就很头疼，HTTP传输的应该是GET请求的json格式数据，而不应该保存在本地，这不是一个web程序了，而echart就很友好使用的是json对象的形式传入实现渲染；还有一点就是echarts是国内的开源项目，有很友好的开源文档很方便去查看，并进行相关参数的修改。
* 如何使用：这里做简单的介绍，官网文档很仔细。首先我们需要规定页面中的DOM部分，哪里去呈现我们的图表，其次，获取DOM，编写option对象文件（关键），最后setOption()即可。是不是非常的方便：)



界面 1：首页



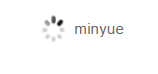
界面 2：功能页

## 3.4 改进汇总

至此，系统的基本设计就完成了，下面从提升用户交互性，页面美观性，拓展功能实现三个方面来介绍改进实现。

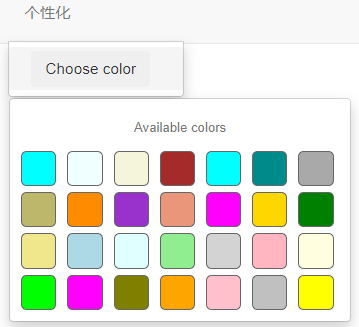
1. 用户交互性设计：完成系统设计之后，我们发现每次生成和失效网络，后台计算并写入数据库需要几秒钟的时间，为了给用户一些提示信息，我们做了两点改进。

* 弹窗设计：在用户功能选择之后，我们都会有一个系统弹窗提示用户当前的进程，无论是开始还是结束都有弹窗设计。
* 加载中标志：生成，销毁后台时间是一个空档时间，怎么样更加直观的显示这个过程呢？于是我们将加载中的图标植入我们的系统（下图所示）

1. 页面美观性改进：首页添加独具特色的瀑布流结构，版心统一，头部和主体分明。
2. 拓展功能实现：拓展功能完成了文件上传，改变节点颜色和大小操作。

* 文件上传：支持用户在本地选择一个已经存在的网络拓扑结构的json格式文件，用户选择上传之后，系统会解析这个json文件，并将拓扑图的可视化图形展示在页面中，并且包含节点信息和各种拓扑信息。
* 改变节点颜色和大小：利用echart天生的优势，直接改变option对象中对节点图的约束信息即可。对应页面中的个性化选项。大小可以通过滚轮进行缩放实现，非常好的用户体验效果。



（关于拓展功能的具体技术实现，在后面技术报告中会进一步讲解）

# 技术报告

前文对项目的系统功能有了完整的描述，对软件设计方案也做了清晰的阐明，下面就技术报告，来梳理一下本次软件课设中所使用到的技术点和技术栈。下面分为四个方面：前端技术栈，后端算法和数据流。

## 4.1 前端技术栈

前端除了老三样（html，css，js）以外，值得一提的就是，jquery库，bootstrap，echarts，flask。

1. Jquery与flask：

Jquery成熟的ajax请求处理方案，连同flask框架搭建起我们的前后端交互的支柱；非常优秀的路由匹配，使我们能过很清楚的处理不同页面应该有的页面效果和页面操作。

其次，jquery除了ajax派发的方面性以外，也给我们提供了很多简便的DOM操作处理方案，其中首页的瀑布流就是基于jquery开发的。

同时，项目的服务器端口是由flask框架来在本地端口开启的，也为整个项目完成完整的路由配发。实现哪个端口处理哪件事情，调用哪些函数，推送哪些界面。

1. Bootstrap：

实现页面组件样式的编写，实现页面的响应式布局。在我们项目中最为突出的运用就是在功能区的选项栏设计和颜色调试器。使我们响应栏的打开和关闭非常的丝滑，这些都得益于bootstrap的加持。

1. Echarts：

前面也有所介绍，echarts这个强大的js库实现了各种各样图示的显示效果，非常适合可视化项目的开发。

1. Iconfont：

值得一提的技术点还有就是，字体图标的使用，使用iconfont的字体图标非常节省网页性能，可以图片被编码为了相应的16进制数字，大小可想而知，这也是各大网站小图标的首推之选。

## 4.2 后端算法和数据流

我们使用python作为我们的后端语言（服务器语言），选择python有两点原因，networkx本身就是python库，使用其中的封装函数更加方便，其次就是前端交互的框架选择的是flask框架也是用python开发的。选择mysql最为我们的数据库，因为mysql相对简单，同时在 WEB 应用方面MySQL是最好的 RDBMS之一。

这里就以代码设计结构和代码设计思想和数据流三个部分阐明

1. 代码结构：

一个入口文件和三个子文件实现本项目的后端程序设计。主程序负责服务器打开，路由管理，对各个其他子文件功能的调用实现项目功能。子程序分为生成函数，失效函数，参数计算三个对应部分，分别完成三种生成方法吧，五种失效方法的设计以及各项参数的测算。

1. 代码设计：

具体的算法设计，根据networkx的封装函数以及其对各种网络图的生成源码，来设计我们需要的程序。

1. 数据流；

这个项目产生的数据量还是很大的，尤其是节点数目的连边概率很大的时候计算量，数据量是很大的。数据流也就是对数据库的设计，如何好的去存放这些数据。Python对mysql数据库有很好的加持，分为三个数据表（jsondata，graphcaldata，nodedata），公共主键generation很重要，表示数据库这一组数据的代数，对生成，失效过程的演示非常重要，同时主键约束也是mysql数据库中使用最为频繁的约束,般情况下，为了便于 DBMS 更快的查找到表中的记录，都会在表中设置一个主键。

# 总结与建议

通过这次课程设计，是真的学到了很多，更加熟练了自己对前端页面的搭构，也让我学到了如何实现前后端的真正交互，还有对数据库的使用。这次课设遇到的一切困难我觉得都非常的有意义，真的是问题越多收获越多，让我不再怕bug，仔细检查，学会如何debug，解决完bug真的是非常的爽！

这次课设也使我意识到，学习的过程中需要同伴，互相监督学习，互相鼓励支持，在团队中我们可以学到他人的长处，多一双眼睛看问题的角度真的宽了很多，希望以后会有更多的项目合作。

最后，希望自己谨记经验教训，坚持前端的道路走下去！！

# 致谢

完成课程设计的期间里，我和我的队员遇到了很多困难，我们从networkx1.0做到networkx5.0进行了5次的更新每一个版本也记录着我和我的队员们埋头的背影和解决问题后成功的笑脸。非常感谢我的队员们，你们都是最棒的，三个人相互督促相互学习，合作的感觉真好，真的谢谢！

# 参考文献

[1] 图形库echarts https://echarts.apache.org/zh/index.html

[2]python库networkx https://networkx.org/

[3] 菜鸟驿站 mysql https://www.runoob.com/mysql/mysql-tutorial.html

[4]css样式 bootstrap中文文档 https://v4.bootcss.com/docs/getting-started/introduction/

[5] CSDN常见网络模型构建 https://blog.csdn.net/un357951/article/details/103514682

[6]jquery-ajax http://docs.jinkan.org/docs/flask/patterns/jquery.html

# 附录A index.html

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>極</title>

<!-- 引入网站图标 -->

<link rel="shortcut icon" href="../static/ig.ico">

<link rel="stylesheet" href="../static/css/style.css">

<link rel="stylesheet" href="../static/css/show.css">

<link rel="stylesheet" href="../static/css/sortable.min.css">

<script src="{{url\_for('static', filename='js/sortable.min.js')}}"></script>

</head>

<body>

<!-- 首页main.html -->

<!-- 头部部分开始 -->

<div class="head w">

<!-- logo部分 -->

<div class="logo">

<img src="../static/image/logo.png">

</div>

<!-- 导航栏 nav -->

<div class="head\_nav">

<ul>

<li><a href="er.html">ER</a></li>

<li><a href="ws.html">WS</a></li>

<li><a href="ba.html">BA</a></li>

<li><a href="file.html">文件上传</a></li>

</ul>

</div>

<!-- 搜索模块 -->

<div class="search">

<input type="text" value="输入关键词">

<button></button>

</div>

<!-- 用户模块 -->

<div class="user">

<img src="../static/image/user.png" alt="" style="vertical-align: middle;">

minyue

</div>

</div>

<!-- 头部部分结束 -->

<!-- 瀑布流展示开始 -->

<main>

<div class="container">

<div class="wrapper">

<ul class="sortable\_\_nav nav">

<li>

<a data-sjslink="all" class="nav\_\_link">

All

</a>

</li>

<li>

<a data-sjslink="net" class="nav\_\_link">

网络类型

</a>

</li>

<li>

<a data-sjslink="node" class="nav\_\_link">

节点中心性

</a>

</li>

<li>

<a data-sjslink="tool" class="nav\_\_link">

开源工具

</a>

</li>

</ul>

<div id="sortable" class="sjs-default">

<div data-sjsel="net">

<div class="card">

<img class="card\_\_picture" src="../static/image/item-1.jpg" alt="">

<div class="card-infos">

<h2 class="card\_\_title">ER模型之学生饭卡</h2>

<p class="card\_\_text">

E-R图又称实体关系图，是一种提供了实体，属性和联系的方法，用来描述现实世界的概念模型。通俗点讲就是，当我们理解了实际问题的需求之后，需要用一种方法来表示这种需求，概念模型就是用来描述这种需求。比如学生生活中的校园卡系统

</p>

</div>

</div>

</div>

<div data-sjsel="net">

<div class="card">

<img class="card\_\_picture" src="../static/image/item-2.jpg" alt="">

<div class="card-infos">

<h2 class="card\_\_title">WS网络？交际圈？</h2>

<p class="card\_\_text">

小世界网络是一种数学图,在社交网络中，这种网络属性意味着一些彼此并不相识的人，可以通过一条很短的熟人链条被联系在一起，这也就是小世界现象。许多经验网络图都展示出了小世界现象，例如社交网络、互联网的底层架构、诸如Wikipedia的百科类网站以及基因网络等等。

</p>

</div>

</div>

</div>

<div data-sjsel="net">

<div class="card">

<img class="card\_\_picture" src="../static/image/item-3.jpg" alt="">

<div class="card-infos">

<h2 class="card\_\_title">BA网络，你知道吗？</h2>

<p class="card\_\_text">

前人为了解释幂律的产生机制，提出了无标度网络模型（BA模型）。BA模型具有两个特性，其一是增长性，所谓增长性是指网络规模是在不断的增大的，在研究的网络当中，网络的节点是不断的增加的；其二就是优先连接机制，这个特性是指网络当中不断产生的新的节点更倾向于和那些连接度较大的节点相连接。

</p>

</div>

</div>

</div>

<div data-sjsel="node">

<div class="card">

<img class="card\_\_picture" src="../static/image/item-4.jpg" alt="">

<div class="card-infos">

<h2 class="card\_\_title">度中心性</h2>

<p class="card\_\_text">

在无向网络中，我们可以用一个节点的度数来衡量中心性。这一指标背后的假设是：重要的节点就是拥有许多连接的节点。你的社会关系越多，你的影响力就越强。

</p>

</div>

</div>

</div>

<div data-sjsel="node">

<div class="card">

<img class="card\_\_picture" src="../static/image/item-5.jpg" alt="">

<div class="card-infos">

<h2 class="card\_\_title">接近中心性</h2>

<p class="card\_\_text">

点度中心性仅仅利用了网络的局部特征，即节点的连接数有多少，但一个人连接数多，并不代表他/她处于网络的核心位置。接近中心性和中介中心性一样，都利用了整个网络的特征，即一个节点在整个结构中所处的位置。如果节点到图中其他节点的最短距离都很小，那么它的接近中心性就很高。相比中介中心性，接近中心性更接近几何上的中心位置。

</p>

</div>

</div>

</div>

<div data-sjsel="node">

<div class="card">

<img class="card\_\_picture" src="../static/image/item-6.jpg" alt="">

<div class="card-infos">

<h2 class="card\_\_title">中介中心性</h2>

<p class="card\_\_text">

网络中两个非相邻成员之间的相互作用依赖于其他成员，特别是两成员之间路径上的那些成员。他们对两个非相邻成员之间的相互作用具有控制和制约作用。中介中心性的思想是：如果一个成员位于其他成员的多条最短路径上，那么该成员就是核心成员，就具有较大的中介中心性。

</p>

</div>

</div>

</div>

<div data-sjsel="tool">

<div class="card">

<img class="card\_\_picture" src="../static/image/item-7.jpg" alt="">

<div class="card-infos">

<h2 class="card\_\_title">echarts</h2>

<p class="card\_\_text">

echarts 是一个用纯JavaScript编写的一个图表库， 能够很简单便捷的在web网站或是web应用程序添加有交互性的图表，并且免费提供给个人学习、个人网站和非商业用途使用。echarts支持的图表类型有曲线图、区域图、柱状图、饼状图、散状点图和综合图表。

</p>

</div>

</div>

</div>

<div data-sjsel="node">

<div class="card">

<img class="card\_\_picture" src="../static/image/item-8.jpg" alt="">

<div class="card-infos">

<h2 class="card\_\_title">特征向量中心性</h2>

<p class="card\_\_text">

特征向量中心性的基本思想是，一个节点的中心性是相邻节点中心性的函数。也就是说，与你连接的人越重要，你也就越重要。

</p>

</div>

</div>

</div>

<div data-sjsel="node">

<div class="card">

<img class="card\_\_picture" src="../static/image/item-9.jpg" alt="">

<div class="card-infos">

<h2 class="card\_\_title">katz中心性</h2>

<p class="card\_\_text">

katz centrality通过测量相邻节点（一级节点）和网络中通过这些相邻节点连接到考虑中节点的所有其他节点的数量，计算网络中节点的相对影响。额外的权重可以通过参数$beta$提供给直接邻居。然而，为了正确计算Katz中心度，衰减因子$alpha$应严格小于邻接矩阵的最大逆特征值

</p>

</div>

</div>

</div>

<div data-sjsel="tool">

<div class="card">

<img class="card\_\_picture" src="../static/image/item-10.jpg" alt="">

<div class="card-infos">

<h2 class="card\_\_title">networkx</h2>

<p class="card\_\_text">

用python语言编写的软件包，便于用户对复杂网络进行创建、操作和学习。利用networkx可以以标准化和非标准化的数据格式存储网络、生成多种随机网络和经典网络、分析网络结构、建立网络模型、设计新的网络算法、进行网络绘制等。

</p>

</div>

</div>

</div>

</div>

</div>

</div>

</main>

</div>

<script type="text/javascript">

document.querySelector('#sortable').sortablejs()

</script>

<!-- 瀑布流展示结束 -->

</html>

# 附录B er.html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>ER网络演化模型</title>

<!-- 引入网站图标 -->

<link rel="shortcut icon" href="../static/ig.ico">

<!-- 插入jquery -->

<script src="../static/jquery.js"></script>

<!-- 插入echarts -->

<script src="../static/echarts.min.js"></script>

<!-- 设定一个全局变量作为一个应用根路径的前缀 -->

<script type=text/javascript>

var $SCRIPT\_ROOT = {{request.script\_root|tojson|safe}};

</script>

<!-- 异步函数引入 -->

<script src="{{url\_for('static', filename='js/er\_submit.js')}}"></script>

<script src="{{url\_for('static', filename='js/update.js')}}"></script>

<script src="{{url\_for('static', filename='js/last.js')}}"></script>

<script src="{{url\_for('static', filename='js/disable/degree.js')}}"></script>

<script src="{{url\_for('static', filename='js/disable/between.js')}}"></script>

<script src="{{url\_for('static', filename='js/disable/katz.js')}}"></script>

<script src="{{url\_for('static', filename='js/disable/close.js')}}"></script>

<script src="{{url\_for('static', filename='js/disable/vector.js')}}"></script>

<script src="{{url\_for('static', filename='js/show\_remove.js')}}"></script>

<!-- css样式引入 -->

<link rel="stylesheet" href="../static/css/style.css">

<link rel="stylesheet" href="../static/font/iconfont.css">

<link

href="https://cdn.bootcss.com/twitter-bootstrap/3.3.7/css/bootstrap.min.css"

rel="stylesheet"

/>

</head>

<body>

<!-- 头部部分开始 -->

<div class="head w">

<!-- logo部分 -->

<a class="logo" href="/">

<img src="../static/image/logo.png">

</a>

<!-- 导航栏 nav -->

<div class="head\_nav">

<ul>

<li><a href="#" style="color: blue;">ER</a></li>

<li><a href="ws.html">WS</a></li>

<li><a href="ba.html">BA</a></li>

</ul>

</div>

<!-- 搜索模块 -->

<div class="search">

<input type="text" value="输入关键词">

<button></button>

</div>

<!-- 用户模块 -->

<div class="user">

<img src="../static/image/user.png" id="ico" alt="" style="vertical-align: middle;">

minyue

</div>

</div>

<!-- 头部部分结束 -->

<!-- 主体部分 -->

<div class="w">

<div style="padding-left: 14px">

<p>

ER网络的顶点个数：<input type="text" size="5" name="er\_nodes" value="20" />

连边概率：<input type="text" size="5" name="er\_p" value="0.1" />

</p>

</div>

<div>

<nav class="navbar navbar-default navbar-mobile bootsnav">

<div class="collapse navbar-collapse" id="navbar-menu">

<ul

style="float: left;"

class="nav navbar-nav"

data-in="fadeInDown"

data-out="fadeOutUp"

>

<li class="active">

<a href="#"

><span class="iconfont">&#xe691;</span></span

></a>

</li>

<li>

<a href="#" id="er\_submit"

>create </a>

</li>

<li class="dropdown">

<a

href="#"

class="dropdown-toggle"

>show</a>

<ul class="dropdown-menu">

<li><a href="#" id="update">update</a></li>

<li><a href="#" id="last">show last</a></li>

</ul>

</li>

<li class="dropdown">

<a

href="#"

class="dropdown-toggle"

>remove methods</span

></a>

<ul class="dropdown-menu">

<li><a href="#" id="between">介数中心性失效</a></li>

<li><a href="#" id="close">接近中心性</a></li>

<li><a href="#" id="degree">度中心性</a></li>

<li><a href="#" id="vector">特征向量中心性</a></li>

<li><a href="#" id="katz">katz中心性</a></li>

</ul>

</li>

<li>

<a href="#" id="show\_remove"

>remove </a>

</li>

<li class="dropdown">

<a

href="#"

class="dropdown-toggle"

>个性化</a>

<ul class="dropdown-menu">

<li><a href="#" id="colorpalettediv" value="rgb(92,123,217)" >color</a></li>

</ul>

</li>

</ul>

</div>

</nav>

</div>

<div style="padding-left: 14px">

平均度：<span id="avgdegree">?</span>

平均路径：<span id="avgpath">?</span>

平均聚类系数：<span id="avgcluster">?</span>

</div>

<div style="overflow: hidden;">

<div id="er\_pic" style="width: 590px; height: 400px; float: left;"></div>

<div id="er\_degree" style="width: 590px; height: 400px; float: right;"></div>

</div>

</div>

<!-- 导航栏样式文件 -->

<script src="{{url\_for('static', filename='js/bootsnav.js')}}"/></script>

<script

src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.0/umd/popper.min.js"

integrity="sha384-cs/chFZiN24E4KMATLdqdvsezGxaGsi4hLGOzlXwp5UZB1LY//20VyM2taTB4QvJ"

crossorigin="anonymous"

></script>

<script src="https://cdn.bootcss.com/bootstrap/4.1.0/js/bootstrap.min.js"></script>

<script src="{{url\_for('static', filename='js/colorpalettepicker.js')}}"></script>

<script type="text/javascript">

$("#colorpalettediv").colorPalettePicker({

lines: 4,

onSelected: function (color) {

$("#colorpalettediv").attr("value", color);

},

});

</script>

</body>

</html>

# 附录C jquery.py

# -\*- coding: utf-8 -\*-

from flask import Flask, jsonify, render\_template, request

import json

import time

import random

import networkx as nx

from networkx.readwrite import json\_graph

import pymysql

import numpy as np

import cal

from disablefuction import \*

from netcreatefuction import \*

#client = MongoClient()

#db = client['test-database']

disablecurrent\_gen=0

current\_gen=0

last\_jsondata=0

last\_graphdata=0

last\_nodedata=0

app = Flask(\_\_name\_\_)

@app.route("/")

def index():

# 主页面

return render\_template("index.html")

@app.route("/er.html")

def er\_index():

return render\_template('er.html')

@app.route("/ws.html")

def ws\_index():

return render\_template('ws.html')

@app.route("/ba.html")

def ba\_index():

return render\_template('ba.html')

@app.route("/file.html")

def file\_index():

return render\_template('file.html')

@app.route("/readFile")

def readFile():

jsonFile = request.args.get('a')

#jsonFile = json.dumps(jsonFile)

#jsonFile = str(jsonFile)

print(type(jsonFile))

strtojson = json.loads(jsonFile)

G = json\_graph.node\_link\_graph(strtojson)

histogram=cal.calhistogram(G)

avgdegree = cal.calavgdegree(G)

avgpath = cal.calavgpath(G)

avgcluster = cal.calavgcluster(G)

#strtojson = json.loads(jsonFile)

jsonFile = '{"series":[{"type":"graph","layout":"force","roam":true,"force":{"repulsion":200},"draggable":true,"focusNodeAdjacency":true,"lineStyle":{"width":3,"color":"#000"},'+jsonFile[1:]

jsonFile = jsonFile+']}'

strtojson = json.loads(jsonFile)

num\_node=G.number\_of\_nodes()

bc=nx.betweenness\_centrality(G)

cc=nx.closeness\_centrality(G)

try:

ec=nx.eigenvector\_centrality(G,max\_iter=1000)

except:

ec={}

for i in range(num\_node):

ec[i]=0

print('bbbbbug')

print(gene)

kc=nx.katz\_centrality(G)

for n in range(G.number\_of\_nodes()):

node\_id=n

id\_degree=G.degree(n)

id\_betweeness=bc[n]

id\_closeness=cc[n]

id\_eigenvector=ec[n]

id\_katz=kc[n]

id\_cluster= nx.clustering(G,n)

id\_triangle= nx.triangles(G,n)

fuck1={}

fuck2={}

fuck1['formatter']='id: %d\n------------------------\ndegreee: %d\nbetweeness: %f\ncloseness: %f\neigenvector: %f\nkatz: %f\ncluster: %f\ntriangle: %d'%(n,id\_degree,id\_betweeness,id\_closeness,id\_eigenvector,id\_katz,id\_cluster,id\_triangle)

fuck1['align']='left'

fuck1['position']='right'

fuck1['color']= "rgba(255, 255, 255, 1)"

fuck1['backgroundColor']= "rgba(0, 0, 0, 0.7)"

fuck1['fontSize']= "14"

fuck1['padding']= 5

fuck1['borderRadius']= 10

fuck2['label']=fuck1

strtojson['series'][0]['nodes'][n]['emphasis']=fuck2

return jsonify(avg\_degree=avgdegree,avg\_path=avgpath,avg\_cluster=avgcluster,json\_str=strtojson,histogramtostr=histogram)

@app.route('/er\_submit')

def er\_submit():

global current\_gen

global disablecurrent\_gen

current\_gen =0

disablecurrent\_gen=0

NETWORK\_SIZE = request.args.get('a', 0, type=int)

PROBABILITY\_OF\_EAGE = request.args.get('b', 0, type=float)

WR\_erdos\_renyi\_graph(NETWORK\_SIZE,PROBABILITY\_OF\_EAGE)

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from jsondata order by id desc limit 1')

mydata = cur.fetchall()

maxgene = mydata[0][2]

i=1

while(i<=maxgene):

cal.avggene(i)

cal.WRnode(i)

i=i+1

# conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

# cur = conn.cursor()

# cur.execute('select \* from jsondata order by id desc limit 1')

# data1 = cur.fetchall()

# maxgen = data1[0][2]

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from graphcaldata order by generation desc limit 1')

graphdata = cur.fetchall()

cur.execute('select \* from nodedata where generation=%d'%maxgene)

nodedata = cur.fetchall()

global last\_jsondata

last\_jsondata=mydata

global last\_graphdata

last\_graphdata=graphdata

global last\_nodedata

last\_nodedata=nodedata

return jsonify('submit success !')

@app.route('/ws\_submit')

def ws\_submit():

global current\_gen

global disablecurrent\_gen

current\_gen =0

disablecurrent\_gen=0

n = request.args.get('a', 0, type=int)

k = request.args.get('b', 0, type=int)

p = request.args.get('c', 0, type=float)

WR\_watts\_strogatz\_graph(n,k,p)

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from jsondata order by id desc limit 1')

mydata = cur.fetchall()

maxgene = mydata[0][2]

i=1

while(i<=maxgene):

cal.avggene(i)

cal.WRnode(i)

i=i+1

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from graphcaldata order by generation desc limit 1')

graphdata = cur.fetchall()

cur.execute('select \* from nodedata where generation=%d'%maxgene)

nodedata = cur.fetchall()

global last\_jsondata

last\_jsondata=mydata

global last\_graphdata

last\_graphdata=graphdata

global last\_nodedata

last\_nodedata=nodedata

return jsonify('submit success !')

@app.route('/ba\_submit')

def ba\_submit():

global current\_gen

global disablecurrent\_gen

current\_gen =0

disablecurrent\_gen=0

n = request.args.get('a', 0, type=int)

m = request.args.get('b', 0, type=int)

WR\_barabasi\_albert\_graph(n, m)

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from jsondata order by id desc limit 1')

mydata = cur.fetchall()

maxgene = mydata[0][2]

i=1

while(i<=maxgene):

cal.avggene(i)

cal.WRnode(i)

i=i+1

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from graphcaldata order by generation desc limit 1')

graphdata = cur.fetchall()

cur.execute('select \* from nodedata where generation=%d'%maxgene)

nodedata = cur.fetchall()

global last\_jsondata

last\_jsondata=mydata

global last\_graphdata

last\_graphdata=graphdata

global last\_nodedata

last\_nodedata=nodedata

return jsonify('submit success !')

@app.route('/update')

def update():

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from jsondata order by id desc limit 1')

data1 = cur.fetchall()

maxgen = data1[0][2]

lastdata = data1[0][1]

lastdata = '{"series":[{"type":"graph","layout":"force","force":{"repulsion":200},'+lastdata[1:]

lastdata = lastdata+']}'

cur.execute('select \* from graphcaldata order by generation desc limit 1')

graphdata = cur.fetchall()

cur.execute('select \* from nodedata where generation=%d'%maxgen)

nodedata = cur.fetchall()

id\_num=len(nodedata)

last\_avgdegree= graphdata[0][0]

last\_avgpath= graphdata[0][1]

last\_avgcluster= graphdata[0][2]

last\_histogram= graphdata[0][4]

last\_strtojson=json.loads(lastdata)

for i in range(id\_num):

fuck1={}

fuck2={}

fuck1['formatter']='id: %d\n------------------------\ndegreee: %d\nbetweeness: %f\ncloseness: %f\neigenvector: %f\nkatz: %f\ncluster: %f\ntriangle: %d'%(nodedata[i][1],nodedata[i][2],nodedata[i][3],nodedata[i][4],nodedata[i][5],nodedata[i][6],nodedata[i][7],nodedata[i][8])

fuck1['align']='left'

fuck1['position']='right'

fuck1['color']= "rgba(255, 255, 255, 1)"

fuck1['backgroundColor']= "rgba(0, 0, 0, 0.7)"

fuck1['fontSize']= "14"

fuck1['borderRadius']= 10

fuck1['padding']= 5

fuck2['label']=fuck1

last\_strtojson['series'][0]['nodes'][i]['emphasis']=fuck2

global current\_gen

current\_gen +=1

global last\_jsondata

last\_jsondata=data1

global last\_graphdata

last\_graphdata=graphdata

global last\_nodedata

last\_nodedata=nodedata

if(current\_gen<=maxgen):

cur = conn.cursor()

cur.execute('select \* from jsondata where generation=%d'%(current\_gen))

mydata = cur.fetchall()

mydata = mydata[0][1]

mydata = '{"series":[{"type":"graph","layout":"circular","roam":true,"focusNodeAdjacency":true,"lineStyle":{"width":3,"color":"#000"},'+mydata[1:]

mydata = mydata+']}'

cur.execute('select \* from graphcaldata where generation=%d'%(current\_gen))

graphdata = cur.fetchall()

cur.execute('select \* from nodedata where generation=%d'%(current\_gen))

nodedata = cur.fetchall()

id\_num2=len(nodedata)

print(id\_num2)

avgdegree= graphdata[0][0]

avgpath= graphdata[0][1]

avgcluster= graphdata[0][2]

histogram= graphdata[0][4]

strtojson=json.loads(mydata)

for i in range(id\_num2):

fuck1={}

fuck2={}

fuck1['formatter']='id: %d\n------------------------\ndegreee: %d\nbetweeness: %f\ncloseness: %f\neigenvector: %f\nkatz: %f\ncluster: %f\ntriangle: %d'%(nodedata[i][1],nodedata[i][2],nodedata[i][3],nodedata[i][4],nodedata[i][5],nodedata[i][6],nodedata[i][7],nodedata[i][8])

fuck1['align']='left'

fuck1['position']='right'

fuck1['color']= "rgba(255, 255, 255, 1)"

fuck1['backgroundColor']= "rgba(0, 0, 0, 0.7)"

fuck1['fontSize']= "14"

fuck1['padding']= 5

fuck1['borderRadius']= 10

fuck2['label']=fuck1

strtojson['series'][0]['nodes'][i]['emphasis']=fuck2

print(i)

return jsonify(avg\_degree=avgdegree,avg\_path=avgpath,avg\_cluster=avgcluster,json\_str=strtojson,histogramtostr=histogram)

else:

return jsonify(avg\_degree=last\_avgdegree,avg\_path=last\_avgpath,avg\_cluster=last\_avgcluster,json\_str=last\_strtojson,histogramtostr=last\_histogram)

@app.route('/last')

def er\_last():

global last\_graphdata

global last\_jsondata

global last\_nodedata

print(last\_graphdata)

return cal.last\_json(last\_jsondata,last\_graphdata,last\_nodedata)

@app.route('/degree')

def er\_remove():

global current\_gen

global disablecurrent\_gen

global last\_jsondata

current\_gen =0

disablecurrent\_gen=0

WR\_degreedisable(last\_jsondata)

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from jsondata order by id desc limit 1')

mydata = cur.fetchall()

maxgene = mydata[0][2]

i=1

while(i<=maxgene):

cal.avggene(i)

cal.WRnode(i)

i=i+1

return jsonify('submit success')

@app.route('/between')

def between():

global current\_gen

global disablecurrent\_gen

global last\_jsondata

current\_gen =0

disablecurrent\_gen=0

WR\_betweenessdisable(last\_jsondata)

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from jsondata order by id desc limit 1')

mydata = cur.fetchall()

maxgene = mydata[0][2]

i=1

while(i<=maxgene):

cal.avggene(i)

cal.WRnode(i)

i=i+1

return jsonify('submit success')

@app.route('/close')

def close():

global current\_gen

global disablecurrent\_gen

global last\_jsondata

current\_gen =0

disablecurrent\_gen=0

WR\_closenessdisable(last\_jsondata)

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from jsondata order by id desc limit 1')

mydata = cur.fetchall()

maxgene = mydata[0][2]

i=1

while(i<=maxgene):

cal.avggene(i)

cal.WRnode(i)

i=i+1

return jsonify('submit success')

@app.route('/vector')

def vector():

global current\_gen

global disablecurrent\_gen

global last\_jsondata

current\_gen =0

disablecurrent\_gen=0

WR\_eigenvectordisable(last\_jsondata)

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from jsondata order by id desc limit 1')

mydata = cur.fetchall()

maxgene = mydata[0][2]

i=1

while(i<=maxgene):

cal.avggene(i)

cal.WRnode(i)

i=i+1

return jsonify('submit success')

@app.route('/katz')

def katz():

global current\_gen

global disablecurrent\_gen

global last\_jsondata

current\_gen =0

disablecurrent\_gen=0

WR\_katzdisable(last\_jsondata)

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from jsondata order by id desc limit 1')

mydata = cur.fetchall()

maxgene = mydata[0][2]

i=1

while(i<=maxgene):

cal.avggene(i)

cal.WRnode(i)

i=i+1

return jsonify('submit success')

@app.route('/show\_remove')

def er\_show\_remove():

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from jsondata order by id desc limit 1')

data1 = cur.fetchall()

maxgen = data1[0][2]

lastdata = data1[0][1]

lastdata = '{"series":[{"type":"graph","layout":"circular","roam":true,"focusNodeAdjacency":true,"lineStyle":{"width":3,"color":"#000"},'+lastdata[1:]

lastdata = lastdata+']}'

cur.execute('select \* from graphcaldata order by generation desc limit 1')

graphdata = cur.fetchall()

last\_avgdegree= graphdata[0][0]

last\_avgpath= graphdata[0][1]

last\_avgcluster= graphdata[0][2]

last\_histogram= graphdata[0][4]

last\_strtojson=json.loads(lastdata)

cur.execute('select \* from nodedata where generation=%d'%(maxgen))

nodedata = cur.fetchall()

id\_num=len(nodedata)

for i in range(id\_num):

fuck1={}

fuck2={}

fuck1['formatter']='id: %d\n------------------------\ndegreee: %d\nbetweeness: %f\ncloseness: %f\neigenvector: %f\nkatz: %f\ncluster: %f\ntriangle: %d'%(nodedata[i][1],nodedata[i][2],nodedata[i][3],nodedata[i][4],nodedata[i][5],nodedata[i][6],nodedata[i][7],nodedata[i][8])

fuck1['align']='left'

fuck1['position']='right'

fuck1['color']= "rgba(255, 255, 255, 1)"

fuck1['backgroundColor']= "rgba(0, 0, 0, 0.7)"

fuck1['fontSize']= "14"

fuck1['padding']= 5

fuck1['borderRadius']= 10

fuck2['label']=fuck1

last\_strtojson['series'][0]['nodes'][i]['emphasis']=fuck2

#lastdata = json.loads(lastdata)

global disablecurrent\_gen

disablecurrent\_gen +=1

if(disablecurrent\_gen<=maxgen):

cur = conn.cursor()

cur.execute('select \* from jsondata where generation=%d'%(disablecurrent\_gen))

mydata = cur.fetchall()

mydata = mydata[0][1]

mydata = '{"series":[{"type":"graph","layout":"circular","roam":true,"focusNodeAdjacency":true,"lineStyle":{"width":3,"color":"#000"},'+mydata[1:]

mydata = mydata+']}'

cur.execute('select \* from graphcaldata where generation=%d'%(disablecurrent\_gen))

graphdata = cur.fetchall()

avgdegree= graphdata[0][0]

avgpath= graphdata[0][1]

avgcluster= graphdata[0][2]

histogram= graphdata[0][4]

strtojson=json.loads(mydata)

cur.execute('select \* from nodedata where generation=%d'%(disablecurrent\_gen))

nodedata = cur.fetchall()

id\_num=len(nodedata)

for i in range(id\_num):

fuck1={}

fuck2={}

fuck1['formatter']='id: %d\n------------------------\ndegreee: %d\nbetweeness: %f\ncloseness: %f\neigenvector: %f\nkatz: %f\ncluster: %f\ntriangle: %d'%(nodedata[i][1],nodedata[i][2],nodedata[i][3],nodedata[i][4],nodedata[i][5],nodedata[i][6],nodedata[i][7],nodedata[i][8])

fuck1['align']='left'

fuck1['position']='right'

fuck1['color']= "rgba(255, 255, 255, 1)"

fuck1['backgroundColor']= "rgba(0, 0, 0, 0.7)"

fuck1['fontSize']= "14"

fuck1['padding']= 5

fuck1['borderRadius']= 10

fuck2['label']=fuck1

strtojson['series'][0]['nodes'][i]['emphasis']=fuck2

return jsonify(avg\_degree=avgdegree,avg\_path=avgpath,avg\_cluster=avgcluster,json\_str=strtojson,histogramtostr=histogram)

else:

return jsonify(avg\_degree=last\_avgdegree,avg\_path=last\_avgpath,avg\_cluster=last\_avgcluster,json\_str=last\_strtojson,histogramtostr=last\_histogram)

if \_\_name\_\_=="\_\_main\_\_":

app.run(host = "0.0.0.0",port = 5000, debug = True)

# 附录D netcreatefunction.py

# -\*- coding: utf-8 -\*-

"""

Created on 2020/12/24

"""

import networkx as nx

import matplotlib.pyplot as plt

import numpy as np

import random

import time

import json

import pymysql

import flask

from networkx.readwrite import json\_graph

def WR\_erdos\_renyi\_graph(NETWORK\_SIZE,PROBABILITY\_OF\_EAGE):

    adjacentMatrix=np.zeros((NETWORK\_SIZE,NETWORK\_SIZE),dtype=int)#初始化邻接矩阵

    random.seed(time.time())#'random.random()#生成[0,1)之间的随机数

    #生成ER网络矩阵

    count=0

    probability=0.0

    for i in range(NETWORK\_SIZE):

        for j in range(i+1,NETWORK\_SIZE):

            probability=random.random()

            if probability<PROBABILITY\_OF\_EAGE:

                count =count+1

                adjacentMatrix[i][j]=adjacentMatrix[j][i]=1

    print('您所构造的ER网络边数为：'+str(count))

    #产生json数据并写入数据库

    G = nx.random\_graphs.random\_regular\_graph(0,NETWORK\_SIZE)  #生成包含NETWORK\_SIZE个节点规则图G

    # so add a name to each node

    for n in G:

        G.nodes[n]["name"] = n

    #图的初始状态

    d = json\_graph.node\_link\_data(G)  # node-link format to serialize

    d\_json = json.dumps(d)

    conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

    cur = conn.cursor()

    #将之前的数据删除

    cur.execute("truncate jsondata")

    cur.execute("truncate graphcaldata")

    cur.execute("truncate nodedata")

    conn.commit()

    gene=1

    tsql  =  "insert into jsondata(json\_data) values('{json}')"

    sql = tsql.format(json=pymysql.escape\_string(d\_json))

    cur.execute(sql)

    cur.execute('select \* from jsondata order by id desc limit 1')

    iddata = cur.fetchall()

    iddata = iddata[0][0]

    #print(iddata)

    gene=1

    tsql2 = "update jsondata set generation=%d where id=%d" %(gene,iddata)

    cur.execute(tsql2)

    conn.commit()

    #开始连接

    for i in range(len(adjacentMatrix)):

        for j in range(i+1,len(adjacentMatrix)):

            if adjacentMatrix[i][j]==1:#如果不加这句将生成完全图，ER网络的邻接矩阵将不其作用

                G.add\_edge(i,j)

                d = json\_graph.node\_link\_data(G)  # node-link format to serialize

                d\_json = json.dumps(d)

                #conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123', db='networkxdata')

                #cur = conn.cursor()

                #tsql  =  "insert into jsondata(data) values('{json}')"

                sql = tsql.format(json=pymysql.escape\_string(d\_json))

                cur.execute(sql)

                cur.execute('select \* from jsondata order by id desc limit 1')

                iddata = cur.fetchall()

                iddata = iddata[0][0]

                #print(iddata)

                gene=gene+1

                tsql2 = "update jsondata set generation=%d where id=%d" %(gene,iddata)

                cur.execute(tsql2)

                conn.commit()

    #conn.commit()

    conn.close()

def WR\_watts\_strogatz\_graph(n, k, p):

    G = nx.Graph()

    nodes = list(range(n))  # nodes are labeled 0 to n-1

    # connect each node to k/2 neighbors

    for j in range(1, k // 2 + 1):

        targets = nodes[j:] + nodes[0:j]  # first j nodes are now last in list

        G.add\_edges\_from(zip(nodes, targets))

    random.seed(time.time())#'random.random()#生成[0,1)之间的随机数

    conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

    cur = conn.cursor()

    #将之前的数据删除

    cur.execute("truncate jsondata")

    cur.execute("truncate graphcaldata")

    cur.execute("truncate nodedata")

    conn.commit()

    # so add a name to each node

    for n in G:

        G.nodes[n]["name"] = n

    #图的初始状态

    #写初始数据库json\_data

    d = json\_graph.node\_link\_data(G)  # node-link format to serialize

    d\_json = json.dumps(d)

    tsql  =  "insert into jsondata(json\_data) values('{json}')"

    sql = tsql.format(json=pymysql.escape\_string(d\_json))

    cur.execute(sql)

    #写初始数据库generation

    cur.execute('select \* from jsondata order by id desc limit 1')

    iddata = cur.fetchall()

    iddata = iddata[0][0]

    gene=1

    tsql2 = "update jsondata set generation=%d where id=%d" %(gene,iddata)

    cur.execute(tsql2)

    conn.commit()

    for j in range(1, k // 2 + 1):  # outer loop is neighbors

        targets = nodes[j:] + nodes[0:j]  # first j nodes are now last in list

        # inner loop in node order

        for u, v in zip(nodes, targets):

            if random.random() < p:

                w = random.choice(nodes)

                # Enforce no self-loops or multiple edges

                while w == u or G.has\_edge(u, w):

                    w = random.choice(nodes)

                    if G.degree(u) >= n - 1:

                        break  # skip this rewiring

                else:

                    #remove

                    G.remove\_edge(u, v)

                    #写数据库json\_data

                    d = json\_graph.node\_link\_data(G)  # node-link format to serialize

                    d\_json = json.dumps(d)

                    tsql  =  "insert into jsondata(json\_data) values('{json}')"

                    sql = tsql.format(json=pymysql.escape\_string(d\_json))

                    cur.execute(sql)

                    #写数据库generation

                    cur.execute('select \* from jsondata order by id desc limit 1')

                    iddata = cur.fetchall()

                    iddata = iddata[0][0]

                    gene=gene+1

                    tsql2 = "update jsondata set generation=%d where id=%d" %(gene,iddata)

                    cur.execute(tsql2)

                    conn.commit()

                    print("remove(%d,%d)"%(u,v))

                    #add

                    G.add\_edge(u, w)

                    d = json\_graph.node\_link\_data(G)  # node-link format to serialize

                    d\_json = json.dumps(d)

                    tsql  =  "insert into jsondata(json\_data) values('{json}')"

                    sql = tsql.format(json=pymysql.escape\_string(d\_json))

                    cur.execute(sql)

                    #写数据库generation

                    cur.execute('select \* from jsondata order by id desc limit 1')

                    iddata = cur.fetchall()

                    iddata = iddata[0][0]

                    gene=gene+1

                    tsql2 = "update jsondata set generation=%d where id=%d" %(gene,iddata)

                    cur.execute(tsql2)

                    conn.commit()

                    print("add(%d,%d)"%(u,w))

    print("finish")

def WR\_barabasi\_albert\_graph(n, m):

    random.seed(time.time())#'random.random()#生成[0,1)之间的随机数

    seed=random

    def \_random\_subset(seq, m, rng):

        """ Return m unique elements from seq.

        This differs from random.sample which can return repeated

        elements if seq holds repeated elements.

        Note: rng is a random.Random or numpy.random.RandomState instance.

        """

        targets = set()

        while len(targets) < m:

            x = rng.choice(seq)

            targets.add(x)

        return targets

    # Add m initial nodes (m0 in barabasi-speak)

    G = nx.random\_graphs.random\_regular\_graph(0,m)

    #G = empty\_graph(m)

    # so add a name to each node

    for i in G:

        G.nodes[i]["name"] = i

    # Target nodes for new edges

    targets = list(range(m))

    # List of existing nodes, with nodes repeated once for each adjacent edge

    repeated\_nodes = []

    # Start adding the other n-m nodes. The first node is m.

    source = m

    conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

    cur = conn.cursor()

    #将之前的数据删除

    cur.execute("truncate jsondata")

    cur.execute("truncate graphcaldata")

    cur.execute("truncate nodedata")

    conn.commit()

    d = json\_graph.node\_link\_data(G)  # node-link format to serialize

    d\_json = json.dumps(d)

    tsql  =  "insert into jsondata(json\_data) values('{json}')"

    sql = tsql.format(json=pymysql.escape\_string(d\_json))

    cur.execute(sql)

    #写初始数据库generation

    cur.execute('select \* from jsondata order by id desc limit 1')

    iddata = cur.fetchall()

    iddata = iddata[0][0]

    gene=1

    tsql2 = "update jsondata set generation=%d where id=%d" %(gene,iddata)

    cur.execute(tsql2)

    conn.commit()

    while source < n:

        # Add edges to m nodes from the source.

        G.add\_edges\_from(zip([source] \* m, targets))

        # Add one node to the list for each new edge just created.

        repeated\_nodes.extend(targets)

        # And the new node "source" has m edges to add to the list.

        repeated\_nodes.extend([source] \* m)

        # Now choose m unique nodes from the existing nodes

        # Pick uniformly from repeated\_nodes (preferential attachment)

        targets = \_random\_subset(repeated\_nodes, m, seed)

        source += 1

        for i in G:

            G.nodes[i]["name"] = i

        #写数据库json\_data

        d = json\_graph.node\_link\_data(G)  # node-link format to serialize

        d\_json = json.dumps(d)

        tsql  =  "insert into jsondata(json\_data) values('{json}')"

        sql = tsql.format(json=pymysql.escape\_string(d\_json))

        cur.execute(sql)

        #写数据库generation

        cur.execute('select \* from jsondata order by id desc limit 1')

        iddata = cur.fetchall()

        iddata = iddata[0][0]

        gene=gene+1

        tsql2 = "update jsondata set generation=%d where id=%d" %(gene,iddata)

        cur.execute(tsql2)

        conn.commit()

# 附录E disablefunction.py

# -\*- coding: utf-8 -\*-

"""

Created on 2020/12/24

"""

import networkx as nx

import matplotlib.pyplot as plt

import numpy as np

import random

import time

import json

import pymysql

import flask

from networkx.readwrite import json\_graph

#函数功能说明

# WR\_degreedisable() 度失效当前数据库jsondata最大generation图，清除jsondata之前的数据，并写入新的失效数据

# WR\_betweenessdisable 中介中心性失效当前数据库jsondata最大generation图，清除jsondata之前的数据，并写入新的失效数据

# WR\_closenessdisable() 接近中心性失效当前数据库jsondata最大generation图，清除jsondata之前的数据，并写入新的失效数据

# WR\_eigenvectordisable() 特征向量中心性失效当前数据库jsondata最大generation图，清除jsondata之前的数据，并写入新的失效数据

# WR\_katzdisable() katz中心性失效当前数据库jsondata最大generation图，清除jsondata之前的数据，并写入新的失效数据

def WR\_degreedisable(last\_jsondata):

    def GraphWRMysql(G):

        d = json\_graph.node\_link\_data(G)  # node-link format to serialize

        d\_json = json.dumps(d)

        conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

        cur = conn.cursor()

        tsql  =  "insert into jsondata(json\_data) values('{json}')"

        sql = tsql.format(json=pymysql.escape\_string(d\_json))

        cur.execute(sql)

        cur.execute('select \* from jsondata order by id desc limit 1')

        iddata = cur.fetchall()

        iddata = iddata[0][0]

        #print(iddata)

        tsql2 = "update jsondata set generation=%d where id=%d" %(gene,iddata)

        cur.execute(tsql2)

        conn.commit()

    def MoveMaxDegreeEdge(G):

        NETWORK\_SIZE=G.number\_of\_nodes()

        degreeMatrix=np.zeros((NETWORK\_SIZE),dtype=int)#初始化度矩阵

        for i in range(NETWORK\_SIZE):

            degreeMatrix[i]=G.degree(i)

        maxdegree=0

        maxdegreenode=0

        for i in range(NETWORK\_SIZE):

            if degreeMatrix[i]>maxdegree:

                maxdegree=degreeMatrix[i]

                maxdegreenode=i

        print("最大度节点"+str(maxdegreenode))

        neighbor=list(G.neighbors(maxdegreenode))

        neighbor\_len=len(neighbor)

        for i in range(neighbor\_len):

            G.remove\_edge(maxdegreenode,neighbor[i])

        return G

    conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

    cur = conn.cursor()

    # cur.execute('select \* from jsondata order by id desc limit 1')

    # mydata = cur.fetchall()

    mydata=last\_jsondata[0][1]

    #mydata = mydata[0][1]

    mydata = json.loads(mydata)

    G = json\_graph.node\_link\_graph(mydata)

    #读取后删除数据

    cur.execute("truncate jsondata")

    cur.execute("truncate  graphcaldata")

    cur.execute("truncate  nodedata")

    conn.commit()

    gene=1

    while(1):

        if G.number\_of\_edges()==0:

            print("finish")

            break

        else:

            print("还剩余的边数"+str(G.number\_of\_edges()))

            G=MoveMaxDegreeEdge(G)

            GraphWRMysql(G)

            gene=gene+1

def WR\_betweenessdisable(last\_jsondata):

    def GraphWRMysql(G):

        d = json\_graph.node\_link\_data(G)  # node-link format to serialize

        d\_json = json.dumps(d)

        conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

        cur = conn.cursor()

        tsql  =  "insert into jsondata(json\_data) values('{json}')"

        sql = tsql.format(json=pymysql.escape\_string(d\_json))

        cur.execute(sql)

        cur.execute('select \* from jsondata order by id desc limit 1')

        iddata = cur.fetchall()

        iddata = iddata[0][0]

        #print(iddata)

        tsql2 = "update jsondata set generation=%d where id=%d" %(gene,iddata)

        cur.execute(tsql2)

        conn.commit()

    def MoveMaxBetweennessEdge(G):

        bc=nx.degree\_centrality(G)

        bc=sorted(bc.items(), key=lambda k: k[1], reverse=True)

        maxbetweennessnode=bc[0][0]

        print("最大中介中心性节点"+str(maxbetweennessnode))

        neighbor=list(G.neighbors(maxbetweennessnode))

        neighbor\_len=len(neighbor)

        for i in range(neighbor\_len):

            G.remove\_edge(maxbetweennessnode,neighbor[i])

        return G

    conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

    cur = conn.cursor()

    # cur.execute('select \* from jsondata order by id desc limit 1')

    # mydata = cur.fetchall()

    # mydata = mydata[0][1]

    mydata=last\_jsondata[0][1]

    mydata = json.loads(mydata)

    G = json\_graph.node\_link\_graph(mydata)

    #读取后删除数据

    cur.execute("truncate jsondata")

    cur.execute("truncate graphcaldata")

    cur.execute("truncate nodedata")

    conn.commit()

    gene=1

    while(1):

        if G.number\_of\_edges()==0:

            print("finish")

            break

        else:

            print("还剩余的边数"+str(G.number\_of\_edges()))

            G=MoveMaxBetweennessEdge(G)

            GraphWRMysql(G)

            gene=gene+1

def WR\_closenessdisable(last\_jsondata):

    def GraphWRMysql(G):

        d = json\_graph.node\_link\_data(G)  # node-link format to serialize

        d\_json = json.dumps(d)

        conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

        cur = conn.cursor()

        tsql  =  "insert into jsondata(json\_data) values('{json}')"

        sql = tsql.format(json=pymysql.escape\_string(d\_json))

        cur.execute(sql)

        cur.execute('select \* from jsondata order by id desc limit 1')

        iddata = cur.fetchall()

        iddata = iddata[0][0]

        #print(iddata)

        tsql2 = "update jsondata set generation=%d where id=%d" %(gene,iddata)

        cur.execute(tsql2)

        conn.commit()

    def MoveMaxClosenessEdge(G):

        bc=nx.closeness\_centrality(G)

        bc=sorted(bc.items(), key=lambda k: k[1], reverse=True)

        maxclosenessnode=bc[0][0]

        print("最大紧密中心性节点"+str(maxclosenessnode))

        neighbor=list(G.neighbors(maxclosenessnode))

        neighbor\_len=len(neighbor)

        for i in range(neighbor\_len):

            G.remove\_edge(maxclosenessnode,neighbor[i])

        return G

    conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

    cur = conn.cursor()

    # cur.execute('select \* from jsondata order by id desc limit 1')

    # mydata = cur.fetchall()

    # mydata = mydata[0][1]

    mydata=last\_jsondata[0][1]

    mydata = json.loads(mydata)

    G = json\_graph.node\_link\_graph(mydata)

    #读取后删除数据

    cur.execute("truncate jsondata")

    cur.execute("truncate graphcaldata")

    cur.execute("truncate nodedata")

    conn.commit()

    gene=1

    while(1):

        if G.number\_of\_edges()==0:

            print("finish")

            break

        else:

            print("还剩余的边数"+str(G.number\_of\_edges()))

            G=MoveMaxClosenessEdge(G)

            GraphWRMysql(G)

            gene=gene+1

def WR\_eigenvectordisable(last\_jsondata):

    def GraphWRMysql(G):

        d = json\_graph.node\_link\_data(G)  # node-link format to serialize

        d\_json = json.dumps(d)

        conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

        cur = conn.cursor()

        tsql  =  "insert into jsondata(json\_data) values('{json}')"

        sql = tsql.format(json=pymysql.escape\_string(d\_json))

        cur.execute(sql)

        cur.execute('select \* from jsondata order by id desc limit 1')

        iddata = cur.fetchall()

        iddata = iddata[0][0]

        #print(iddata)

        tsql2 = "update jsondata set generation=%d where id=%d" %(gene,iddata)

        cur.execute(tsql2)

        conn.commit()

    def MoveMaxEigenvectorEdge(G):

        #bc=nx.eigenvector\_centrality(G)

        try:

            bc=nx.eigenvector\_centrality(G,max\_iter=1000)

        except:

            bc={}

            for i in range(num\_node):

                ec[i]=0

            print('bbbbbug')

        bc=sorted(bc.items(), key=lambda k: k[1], reverse=True)

        maxeigenvectornode=bc[0][0]

        print("最大特征向量中心性节点"+str(maxeigenvectornode))

        neighbor=list(G.neighbors(maxeigenvectornode))

        neighbor\_len=len(neighbor)

        for i in range(neighbor\_len):

            G.remove\_edge(maxeigenvectornode,neighbor[i])

        return G

    conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

    cur = conn.cursor()

    # cur.execute('select \* from jsondata order by id desc limit 1')

    # mydata = cur.fetchall()

    # mydata = mydata[0][1]

    mydata=last\_jsondata[0][1]

    mydata = json.loads(mydata)

    G = json\_graph.node\_link\_graph(mydata)

    #读取后删除数据

    cur.execute("truncate jsondata")

    cur.execute("truncate graphcaldata")

    cur.execute("truncate nodedata")

    conn.commit()

    gene=1

    while(1):

        if G.number\_of\_edges()==0:

            print("finish")

            break

        else:

            print("还剩余的边数"+str(G.number\_of\_edges()))

            G=MoveMaxEigenvectorEdge(G)

            GraphWRMysql(G)

            gene=gene+1

def WR\_katzdisable(last\_jsondata):

    def GraphWRMysql(G):

        d = json\_graph.node\_link\_data(G)  # node-link format to serialize

        d\_json = json.dumps(d)

        conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

        cur = conn.cursor()

        tsql  =  "insert into jsondata(json\_data) values('{json}')"

        sql = tsql.format(json=pymysql.escape\_string(d\_json))

        cur.execute(sql)

        cur.execute('select \* from jsondata order by id desc limit 1')

        iddata = cur.fetchall()

        iddata = iddata[0][0]

        #print(iddata)

        tsql2 = "update jsondata set generation=%d where id=%d" %(gene,iddata)

        cur.execute(tsql2)

        conn.commit()

    def MoveMaxKatzEdge(G):

        bc=nx.katz\_centrality(G)

        bc=sorted(bc.items(), key=lambda k: k[1], reverse=True)

        maxkatznode=bc[0][0]

        print("最大katz中心性节点"+str(maxkatznode))

        neighbor=list(G.neighbors(maxkatznode))

        neighbor\_len=len(neighbor)

        for i in range(neighbor\_len):

            G.remove\_edge(maxkatznode,neighbor[i])

        return G

    conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

    cur = conn.cursor()

    # cur.execute('select \* from jsondata order by id desc limit 1')

    # mydata = cur.fetchall()

    # mydata = mydata[0][1]

    mydata=last\_jsondata[0][1]

    mydata = json.loads(mydata)

    G = json\_graph.node\_link\_graph(mydata)

    #读取后删除数据

    cur.execute("truncate jsondata")

    cur.execute("truncate graphcaldata")

    cur.execute("truncate nodedata")

    conn.commit()

    gene=1

    while(1):

        if G.number\_of\_edges()==0:

            print("finish")

            break

        else:

            print("还剩余的边数"+str(G.number\_of\_edges()))

            G=MoveMaxKatzEdge(G)

            GraphWRMysql(G)

            gene=gene+1

附录F cal.py

import json

import time

import random

import networkx as nx

from networkx.readwrite import json\_graph

import pymysql

from flask import jsonify,request

def avggene(gene):

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from jsondata where generation=%d' %gene)

mydata = cur.fetchall()

mydata = mydata[0][1]

mydata = json.loads(mydata)

G = json\_graph.node\_link\_graph(mydata)

avg\_degree=calavgdegree(G)

avg\_path=calavgpath(G)

avg\_cluster=calavgcluster(G)

histogramtostr=calhistogram(G)

tsql = "insert into graphcaldata(avgdegree,avgpath,avgcluster,generation) values(%f,%f,%f,%d)" %(avg\_degree,avg\_path,avg\_cluster,gene)

cur.execute(tsql)

tsql2 = "update graphcaldata set histogram='"+histogramtostr+"' where generation=%d" %(gene)

cur.execute(tsql2)

conn.commit()

def calhistogram(G):

degree\_list=list(nx.degree\_histogram(G))

histogramtostr= str(degree\_list)

#print(histogramtostr)

return histogramtostr

def calavgdegree(G):

degree\_list=list(nx.degree\_histogram(G))

#print(degree\_list)

avg\_degree=0

for i in range(len(degree\_list)):

avg\_degree += i\*(degree\_list[i])

avg\_degree=avg\_degree / G.number\_of\_nodes()

return avg\_degree

def calavgpath(G):

degree\_list=list(nx.degree\_histogram(G))

if (degree\_list[0]==0):

try:

avg\_path=nx.average\_shortest\_path\_length(G)

except:

print('bug')

avg\_path= 999

else:

avg\_path= 999

return avg\_path

def calavgcluster(G):

avg\_cluster=nx.average\_clustering(G)

return avg\_cluster

def last\_json(last\_jsondata,last\_graphdata,last\_nodedata):

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

# cur.execute('select \* from jsondata order by id desc limit 1')

# data1 = cur.fetchall()

# maxgen = data1[0][2]

# lastdata = data1[0][1]

#maxgen =last\_jsondata[0][2]

lastdata =last\_jsondata[0][1]

lastdata = '{"series":[{"type":"graph","layout":"force","roam":true,"force":{"repulsion":200},"draggable":true,'+lastdata[1:]

lastdata = lastdata+']}'

#cur.execute('select \* from graphcaldata order by generation desc limit 1')

#graphdata = cur.fetchall()

graphdata=last\_graphdata

last\_avgdegree= graphdata[0][0]

last\_avgpath= graphdata[0][1]

last\_avgcluster= graphdata[0][2]

last\_histogram= graphdata[0][4]

last\_strtojson=json.loads(lastdata)

#cur.execute('select \* from nodedata where generation=%d'%(maxgen))

#nodedata = cur.fetchall()

nodedata = last\_nodedata

id\_num=len(nodedata)

for i in range(id\_num):

fuck1={}

fuck2={}

fuck1['formatter']='id: %d\n------------------------\ndegreee: %d\nbetweeness: %f\ncloseness: %f\neigenvector: %f\nkatz: %f\ncluster: %f\ntriangle: %d'%(nodedata[i][1],nodedata[i][2],nodedata[i][3],nodedata[i][4],nodedata[i][5],nodedata[i][6],nodedata[i][7],nodedata[i][8])

fuck1['align']='left'

fuck1['position']='right'

fuck1['color']= "rgba(255, 255, 255, 1)"

fuck1['backgroundColor']= "rgba(0, 0, 0, 0.7)"

fuck1['fontSize']= "14"

fuck1['padding']= 5

fuck1['borderRadius']= 10

fuck2['label']=fuck1

last\_strtojson['series'][0]['nodes'][i]['emphasis']=fuck2

return jsonify(avg\_degree=last\_avgdegree,avg\_path=last\_avgpath,avg\_cluster=last\_avgcluster,json\_str=last\_strtojson,histogramtostr=last\_histogram)

def WRnode(gene):

conn = pymysql.connect(host='localhost', port=3306, user='root', passwd='123456', db='networkxdata')

cur = conn.cursor()

cur.execute('select \* from jsondata where generation=%d' %gene)

mydata = cur.fetchall()

mydata = mydata[0][1]

mydata = json.loads(mydata)

G = json\_graph.node\_link\_graph(mydata)

num\_node=G.number\_of\_nodes()

bc=nx.betweenness\_centrality(G)

cc=nx.closeness\_centrality(G)

try:

ec=nx.eigenvector\_centrality(G,max\_iter=1000)

except:

ec={}

for i in range(num\_node):

ec[i]=0

print('bbbbbug')

print(gene)

kc=nx.katz\_centrality(G)

for n in range(G.number\_of\_nodes()):

node\_id=n

id\_degree=G.degree(n)

id\_betweeness=bc[n]

id\_closeness=cc[n]

id\_eigenvector=ec[n]

id\_katz=kc[n]

id\_cluster= nx.clustering(G,n)

id\_triangle= nx.triangles(G,n)

tsql = "insert into nodedata(generation,nodeid,degree,betweeness,closeness,eigenvector,katz,cluster,triangle) values(%d,%d,%d,%f,%f,%f,%f,%f,%d)"%(gene,node\_id,id\_degree,id\_betweeness,id\_closeness,id\_eigenvector,id\_katz,id\_cluster,id\_triangle)

cur.execute(tsql)

conn.commit()

附录G update.js

（Js脚本文件示例）

$(function () {

  function submit\_form(e) {

    $.getJSON(

      $SCRIPT\_ROOT + "/update",

      {

        now: new Date().getTime(),

      },

      function (data) {

        var color\_rgba= $("#colorpalettediv").attr("value");

        var myChart = echarts.init(document.getElementById("er\_pic"));

        myjson=data.json\_str;

        var fuc = {

          color: color\_rgba

        }

        myjson.series[0].itemStyle = fuc;

        console.log(myjson.series[0].itemStyle.color);

        myChart.setOption(data.json\_str);

        var myChart2 = echarts.init(document.getElementById("er\_degree"));

        var xdata = data.histogramtostr.slice(

          1,

          data.histogramtostr.length - 1

        );

        xdata = xdata.split(", ");

        var data2 = JSON.parse(JSON.stringify(xdata));

        //xdata.forEach((value,index)=>{value=index});

        for (let i = 0; i < xdata.length; i++) {

          xdata[i] = i;

        }

        option = {

          xAxis: {

            type: "category",

            nameLocation: "end",

            name: "度",

            data: xdata,

            nameTextStyle: {

              color: "#000",

              fontWeight: "bolder",

              fontSize: 16,

            },

            axisLabel: {

              fontWeight: "bold",

              fontSize: 12,

            },

          },

          yAxis: {

            type: "value",

            nameLocation: "end",

            name: "节点个数",

            nameTextStyle: {

              color: "#000",

              fontWeight: "bolder",

              fontSize: 16,

            },

            axisLabel: {

              fontWeight: "bold",

              fontSize: 12,

            },

          },

          series: [

            {

              data: data2,

              type: "bar",

            },

          ],

        };

        myChart2.setOption(option);

        $('#avgdegree').text(data.avg\_degree);

        $('#avgpath').text(data.avg\_path==999? "无穷大" : data.avg\_path);

        $('#avgcluster').text(data.avg\_cluster);

      }

    );

  }

  // 绑定click事件

  $("#update").bind("click", submit\_form);

});

附录H 软件说明书

* 具体源码和相关说明书本人已经挂载在[github](https://github.com/minyue-hcm/my_work/tree/master/)

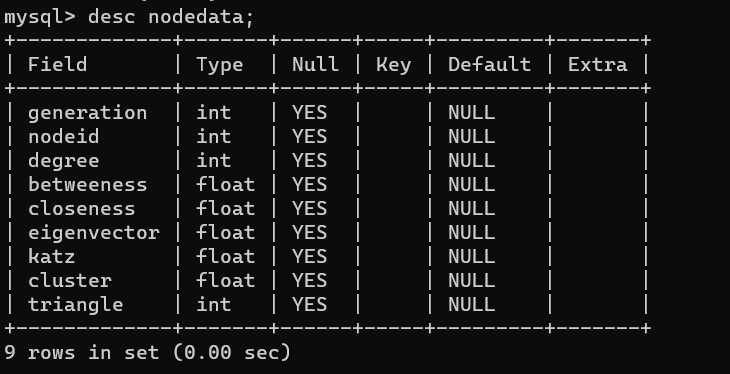
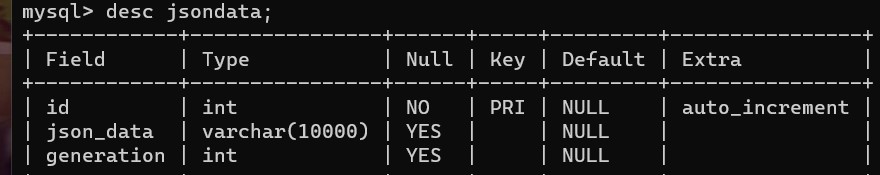
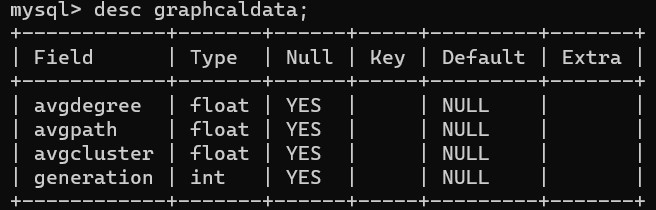
## 使用说明

项目依赖python-flask，jquery ，mysql 前后端结合实现网络演化可视化系统。 ---使用前请安装必要的包文件，配置好对应的环境。相关包文件请参考jquery.py头部包文件引入语句

## 使用流程

1. 先加载数据库mysql

前提完成数据库结构的搭建实例如下：



1. 运行对应jquery.py主程序

在对应localhost:5000 本地服务器端口下可以实现测试