大连理工大学
DALIAN UNIVERSITY OF TECHNOLOGY

大连理工大学软件学院

2018-2019 学年度

《可视化与可视化分析》大作业

目录

1.题目简介

1.1 题目背景

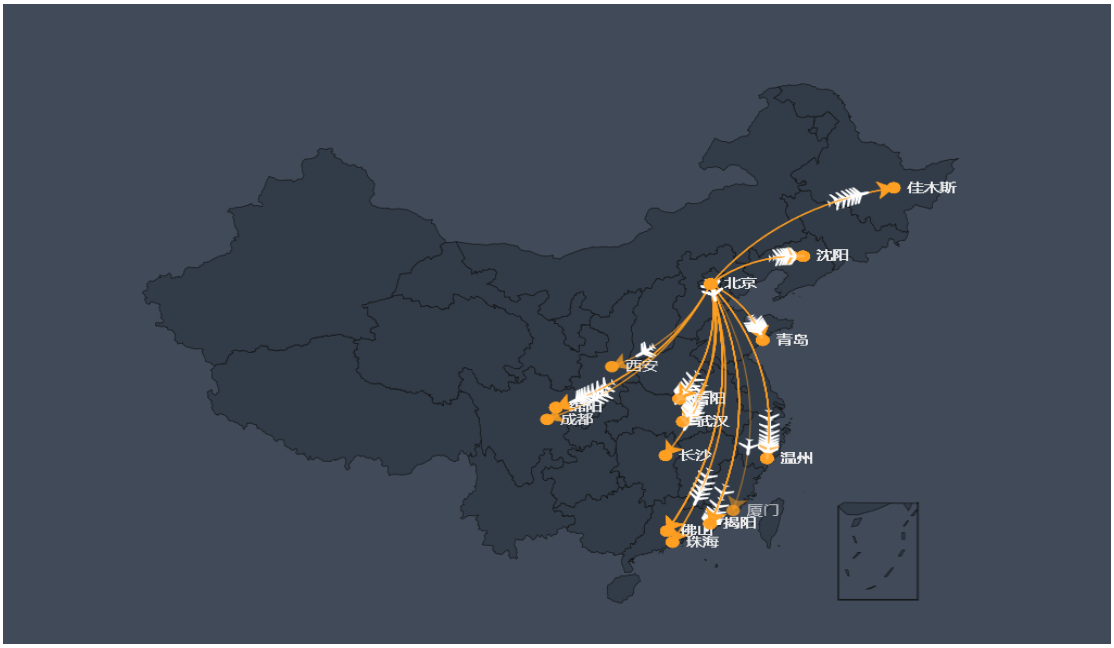
随着新年的临近，学子们和在外务工人员又迎来了一年一度的返乡热潮，因此选择方便快捷的出行方式成为人们急需解决的问题。飞机逐渐成为了人们在交通方面不可缺少的交通工具，乘坐飞机出行的人越来越多，所以飞机的客运量也在迅速增加。同时，乘客对实时查询飞机航班和飞机票剩余量的要求也越来越强，但是现在的飞机票查询系统都是通过查询两点之间的航班信息和通过数字显示机票剩余量，所以我们小组开发了通过 python 爬取机票的相关信息，然后通过可视化的形式动态的展现出来。这个项目对于某城市飞行的便利情况有着现实的研究意义，同时，对于人们的出行也会有很大的帮助，可以在出行前查看航班航线，选择合适的出行时间与出行城市。

1.2 选题介绍

本次项目使用了 PyCharm 开发工具，主要应用到了 echarts-china-cities-pypkg,echarts-china-provinces-pypkg, ,requests,beautifulSoup4, pyecharts 等一系列的 python 库

项目分为两个部分，分别为数据的爬取和数据的可视化。首先通过爬虫爬取机票销售网站上的机票信息，这个过程可以动态的进行爬取，也可以提前爬取然后存在数据库里。

然后就是对爬取到的数据的渲染以及可视化，首先呈现给用户的就是一副中国地图，然后当用户输入查询某起始点的航班信息时，就呈现给用户的就是一组起始点相同，重点散落在中国各个城市的飞机航线，同时在航线上用飞机图标的数量表示机票剩余的数量，当用户鼠标划到相应的航线上，还会显示对应的航班信息。以网页的界面来展示结果方便了其他人的访问，将可视化结果直接在地图上展示，能够清晰的反映出航班数据的分布情况，用动态的飞机来表示航线，也能使可视化的结果清晰明了。



1.3 应用相关

开发语言：Python，一种开源的、解析性的，面向对象的语言编程。源代码和解释器遵循 GPL 协议。Python 语法简洁清晰，特色之一是强制用空白符作为语句缩进。Python 语言有很大的优势，它具有丰富和强大的库，常被称为胶水语言，能够把用其他语言制作的各种模块（尤其是 C/C++）很轻松地联结在一起。但它的性能不如 Java、C、C++这类编译性语言强大，因此常见的一种应用情形是，使用 Python 快速生成程序的原型，然后对其中有特别要求的部分，用更合适的语言改写。

Pytecharts[1]库：一个用于生成 Echarts 图标类的类库，echarts 是一个开源的数据可视化 JS 库，使用 Echart 生成的图可视化效果非常好，pycharts 是为了与 Python 进行对接，方便在 python 中直接使用数据生成图。使用 pycharts 可以生成独立的网页，也可以在 flash，django 中集成生成。本项目主要应用到 pyecharts 中的 echarts-china 中的一系列库。

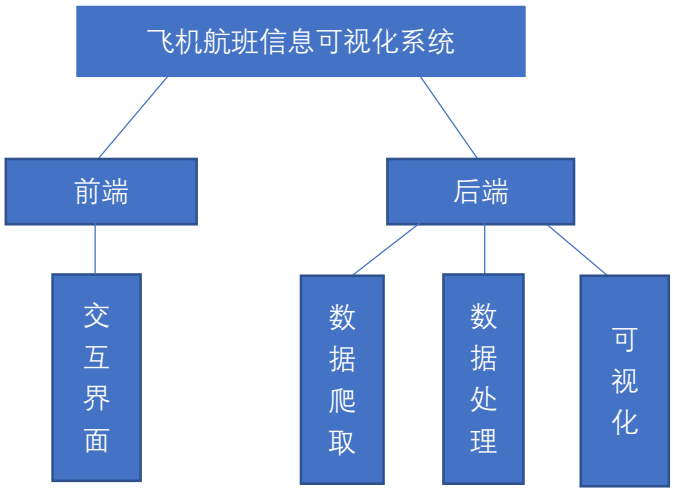
echarts-china 库：商业级数据图表，一个使用 JavaScript 实现的开源可视化库，可以流畅的运行在 PC 和移动设备上，兼容当前绝大部分浏览器（IE8/9/10/11，Chrome，Firefox，Safari 等），底层依赖轻量级的矢量图形库 ZRender 提供直观，交互丰富，可高度个性化定制的数据可视化图表。本项目中我们使用该库生成中国地图。

Beautifulsoup4，requests：使用爬虫需要使用到的库

2. 技术实现

2.1 总体设计方案

本方案采用前后端分离的设计思想，前端负责设计交互界面，便于用户的使用，后端负责对航班数据的爬取，处理与可视化。



2.2 前端部分

前端部分主要是做一个网页版的与用户之间的交互界面，方便用户使用。交互页面主要实现是使用户可以选择城市，从而显示该城市各时间段的航班信息。

图示为选择太原时，在 1 时从太原出发的航班信息：

前端基于 Python 的 Flask 框架搭建，通过 render_template 模块实现交互，前端发起请求（选择目标城市，点击下一时刻按钮）服务器响应之后再后端动态生成所需要的 HTML 文件，在用户点击播放后动态加载这些 HTML 文件。

代码实现

```
# Flask
app=Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/query',methods=['GET','POST'])
def query():
    if request.method=="POST":
        from_city=request.form['station']
        print(from_city)
        while (True):
```

```
                if plane.checkFromCity(from_city):
                    break
            plane.getRouteHTML()

        return render_template("index.html")
```

后端返回的航班数据按照每小时划分，被渲染成 24 个不同的 HTML 文件。前端部分可以实现与用户的交互，让用户可以每个时间段依次进行查看，也可以自动播放与暂停。

代码实现

```
MAX=24;
MIN=1;
var num=MIN;

//load
var loop=null;
var runTime=0;
function search(){
    document.getElementById("load_dis").style.cssText='display:flex;';
    loop=setInterval(function () {
        document.getElementById("load_dis").style.cssText='display:none;';
        document.getElementById("change").src="/static/html/t"+num+".html";
        runTime+=1;
        if(runTime>0){
            clearInterval(loop);
        }
    },30000)
}

//next
function next(){
    num+=1;
    if(num>MAX){
        num=MIN;
    }
    document.getElementById("change").src="/static/html/t"+num+".html";
}

//pre
function pre(){
    num-=1;
    if(num<MIN){
```

```
            num=MAX;
        }
        document.getElementById("change").src="/static/html/t"+num+".html";
}

//start
function start(){
        clearInterval(loop);
        console.log("start success");
        loop=setInterval(function () {
            console.log(num);
            num+=1;
            if(num>MAX){
                num=MIN;
            }
            document.getElementById("change").src="/static/html/t"+num+".html";
        },5000);
}

//pause
function pause(){
        clearInterval(loop);
        console.log("pause");
}
```

2.3 后端部分

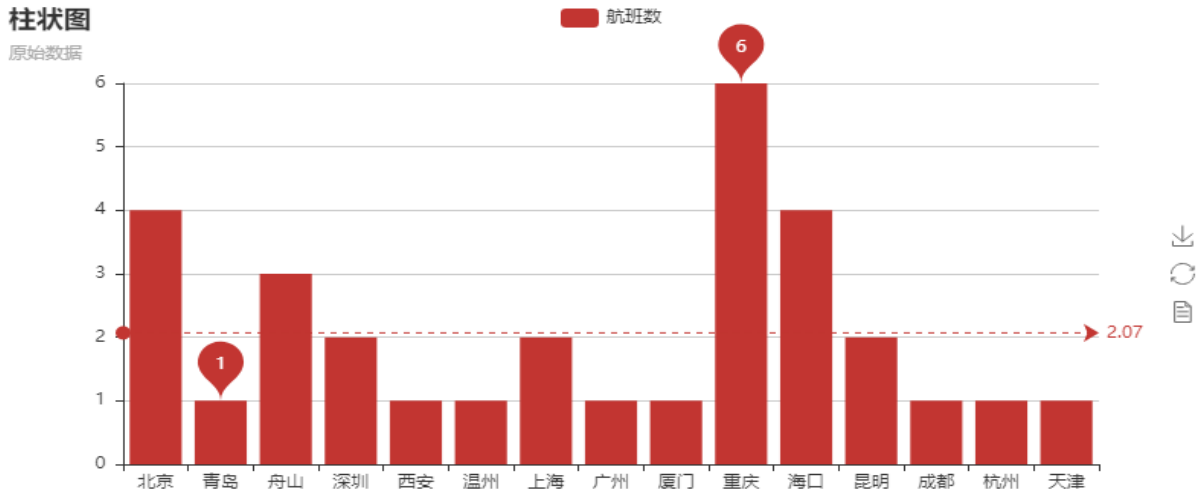
2.3.1 数据扒取

数据爬取采用的方法是基于 requests，beautiful soup4 和线程池的并行数据爬取，给定出发城市扒取相应航班信息，通过线程池分配线程进一步扒取各班次出发时间，通过 beautifulsoup4 处理 HTML 信息获得航班出发时间及目标城市。对于某些目标城市航班数目较多的还有二级页面则需要进一步扒取。

以阜阳为例，爬取数据如下：

```
从阜阳到北京的时间有：
['20:55', '20:55', '21:25', '21:25']
从阜阳到青岛的时间有：
['11:05']
从阜阳到舟山的时间有：
['09:55', '09:55', '09:55']
从阜阳到深圳的时间有：
['15:00', '15:00']
从阜阳到西安的时间有：
['15:35']
从阜阳到温州的时间有：
['10:35']
从阜阳到上海的时间有：
['09:30', '09:30']
从阜阳到广州的时间有：
['13:40']
从阜阳到厦门的时间有：
['10:40']
从阜阳到重庆的时间有：
['14:50', '14:50', '14:50', '14:50', '14:50', '14:50']
从阜阳到海口的时间有：
['19:10', '19:10', '21:10', '21:10']
从阜阳到昆明的时间有：
['21:15', '21:20']
从阜阳到成都的时间有：
['07:50']
从阜阳到杭州的时间有：
['14:40']
从阜阳到天津的时间有：
```

可视化之后的数据为



代码实现

1、从初始链接获得链接列表

```
# 正则匹配所有到达目标城市
to_city_pattern = re.compile(r'[\u4e00-\u9fa5]+')  # 匹配汉字方式
# 匹配结果为 大连-北京 的样子
to_city_list = to_city_pattern.findall(str(self.flight_tag))
```

```
# 目标城市的子链接
to_city_url_pattern = re.compile(r'\bhttp\S*?html\b')
to_city_url_list = to_city_url_pattern.findall(str(self.flight_tag))

# 统计一共需要爬取的目标城市的个数
for i in range(0, len(to_city_list)):
    if to_city_list[i*2] != self.from_city:
        self.total_link = i
        break

# 为到达城市的数据结构（类）添加数据,包括达到城市
# 初始化线程池
pool = ThreadPoolExecutor(self.total_link)

for i in range(0, self.total_link):
    done = pool.submit(self.request, to_city_url_list[i], to_city_list[i*2 + 1])
    done.add_done_callback(self.getDepartureTimeList)
pool.shutdown(wait=True)
```

2、根据链接列表扒取数据

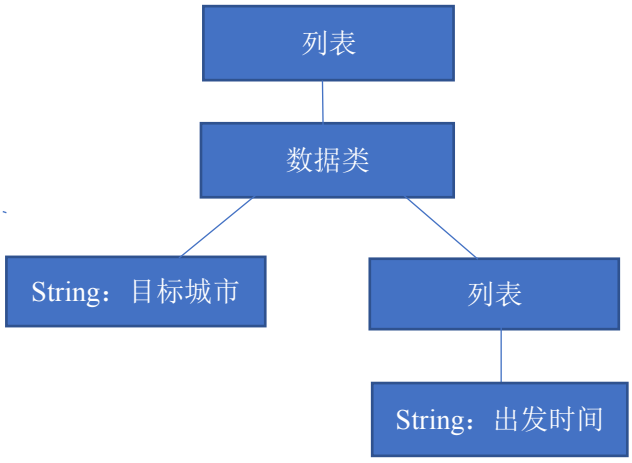
```
# 从链接爬取航班开始时间
def getDepartureTimeList(self, future):

    flight_time = FlightTime()
    if future.result == None:
        return
    r, name = future.result()

    # 设置到达城市的名字
    flight_time.setToCity(name)
    departure_time_list = []

    soup = BeautifulSoup(r.text, 'html.parser')

    # html 标签树匹配规则
    flight_time_html = soup.find_all('td', align='right')
    # 正则匹配规则
    pattern = re.compile(r'\d{1,2}:\d{1,2}')  # 匹配时间
    departure_time_list = pattern.findall(str(flight_time_html))

    # 检查是否有扩展链接
```

```
new_page = soup.find_all('div', class_='schedule_page_list clearfix')
if len(new_page) != 0:
    try:
        url_pattern = re.compile(r'\bhttp\S*?html\b')
        other_url_list = url_pattern.findall(str(new_page))

        for i in range(1, len(other_url_list)):
            other_r = requests.get(other_url_list[i], headers=self.headers, timeout=10)
            other_soup = BeautifulSoup(other_r.text, 'html.parser')
            other_flight_time_html = other_soup.find_all('td', align='right')
            departure_time_list.extend(pattern.findall(str(other_flight_time_html)))
    except Exception as e:
        pass


# 设置到达时间的列表
flight_time.setDate(departure_time_list)
# 每个到达城市类组成一个集合
self.flight_time_list.append(flight_time)
```

2.3.2 数据处理

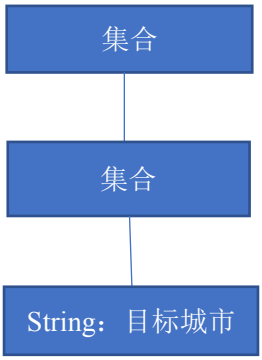
数据处理的目的是改变数据的数据结构，使数据能够更加便于使用。我们的初始数据结构是这样的：

列表中的元素为数据类，每个数据类中包含一个 string 类型的目标城市和一个列表，每个列表中包含了该城市各班次 string 类型的出发时间



处理后的数据结构如下：

集合的下标代表着出发的时间段（一个小时为间隔），集合中的元素为一个个集合，每个集合包含了在同一时间段内出发的各班次的所有目标城市，



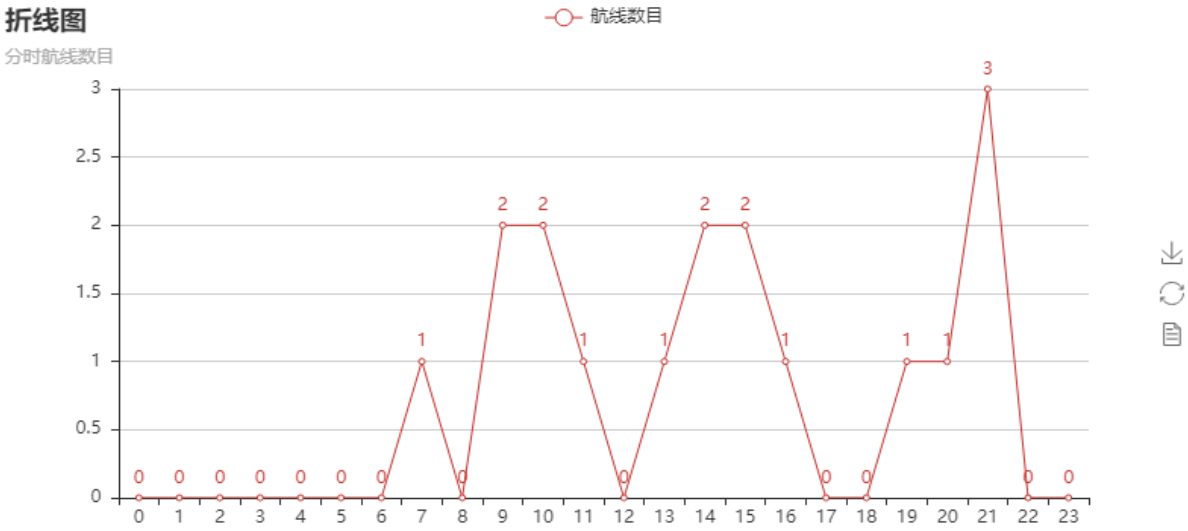
代码实现

```
    # 将 城市-时间 的数据结构转化为 时间-城市
    def sortTime(self):
        self.flight_data = [set() for i in range(24)]   # 借助集合排除重复名字
        for item in self.flight_time_list:
            for date in item.getDate():
                self.flight_data[int(date[0:2])].add(item.getToCity())

        # 测试输出
        for i in range(len(self.flight_data)):
            if len(self.flight_data[i]) != 0:
                print('在' + str(i) + '时起飞的飞机有:', end='')
                print(self.flight_data[i])
            else:
                print('在' + str(i) + '时起飞的飞机有 0 架')

        columns = []
        data = []
        for i in range(len(self.flight_data)):
            columns.append(str(i))
            data.append(len(self.flight_data[i]))
```

以阜阳为例，图示如下：

```
在0时起飞的飞机有0架
在1时起飞的飞机有0架
在2时起飞的飞机有0架
在3时起飞的飞机有0架
在4时起飞的飞机有0架
在5时起飞的飞机有0架
在6时起飞的飞机有0架
在7时起飞的飞机有:{'成都'}
在8时起飞的飞机有0架
在9时起飞的飞机有:{'上海', '舟山'}
在10时起飞的飞机有:{'温州', '厦门'}
在11时起飞的飞机有:{'青岛'}
在12时起飞的飞机有0架
在13时起飞的飞机有:{'广州'}
在14时起飞的飞机有:{'重庆', '杭州'}
在15时起飞的飞机有:{'西安', '深圳'}
在16时起飞的飞机有:{'天津'}
在17时起飞的飞机有0架
在18时起飞的飞机有0架
在19时起飞的飞机有:{'海口'}
在20时起飞的飞机有:{'北京'}
在21时起飞的飞机有:{'海口', '北京', '昆明'}
在22时起飞的飞机有0架
在23时起飞的飞机有0架
```

可视化



2.3.3 生成 HTML 文件

我们借助了 pyecharts 库实现了数据的可视化，展示了每小时内所选城市通往全国各地的航班信息，显示了航线和航班数目

图示为某时间段内从北京出发的航班各班次信息可视化结果

代码实现

```
# 读取数据并在地图上绘制
def drawRoute(self):
    for i in range(len(self.flight_data)):
        html_data = []
        if len(self.flight_data[i]) != 0:
            self.renderHtml(i, html_data)
        else:
            self.NoneHtml(i)

# 必须用一个新的方法防止在同一张图上叠加生成
def renderHtml(self, leave_time, html_data):
    plane_map = GeoLines("航线分时展示", **self.style.init_style)  # 相当于设置背景

    for city_name in self.flight_data[leave_time]:

        html_data.append([self.from_city, city_name])
        map_name = '在' + str(leave_time) + '时从' + self.from_city + '起飞的飞机:'
        try:
            plane_map.add(map_name, html_data,
                          tooltip_formatter="{a} : {c}", **self.style_geo)
        except Exception as e:
            pass  # 忽略无法标识出来的城市
```

```
plane_map.render(self.local + '/t' + str(leave_time+1) + '.html')
```

3. 总结

这个项目的主要是选择一个城市查看该城市一天的时间内的在不同的时间段其航线情况。首先使用 python 的 requests 扒取当天某城市的所有航线(目标城市即出发时间),获取数据,这数据是与网站中实时获取,因此具有实时的效应,是一个动态的时间,而非一个静态的可视化,对于某城市飞行的便利情况有着现实的研究意义,同时,对于人们的出行也会有很大的帮助,可以在出行前查看航班航线,选择合适的出行时间与出行城市。这个算法的使用对于这个程序的实用性以及现实性有所提高,但是,这算法增加了电脑的开销,在数据量比较大的时候运行的时间过长。

基于 flask 网页以一小时为间隔动态展示,以网页的界面来展示结果方便了其他人的访问,将可视化结果直接在地图上展示,能够清晰的反映出航班数据的分布情况,用动态的飞机来表示航线,也能使可视化的结果清晰明了。但是用网页加载数据会严重增加运行的时间,我认为可以将可视化做成简单的数据模型,如疏密图,云图,饼形图等,这样可以增加运行的效率,并且具有准确简洁的效果,而且不需要更多的解释,更加具有实际意义。

在数据处理方便使用了转换数据结构,集合:下标表示出发时间段(一个小时为间隔)元素:改时间段到达的目标城市的集合,通过这种方式能够将预存在代码程序中的城市地址能够分开的提取出来,同时,基于着这种方式能够将网络中爬取的大量数据进行降维处理,将数据按照时间顺序提取和整理出来能够帮助可视化处理时的构图与使用,不需要再可视化时再次对于数据进行处理。这方法能够帮助我们筛去大量无作用和不需要的数据,对于效率有着显著的提高。

可视化在这个项目中起到了很重要的作用,当我们使用大量的表格数据来处理和研究航线时,对于航线的感官是不够强烈与清晰的,而且还有冗余的数据掺杂其中,对于研究是非常不利的,可视化处理之后,我们可以一眼看出这是中国某城市的航线图,去哪块地区的航线最多,这对于航空公司来说很有借鉴的,它可以在一些航线少的地区新增航班,可以在某时间段少的时候新增航班等等。

我们认为更好的方式是将可视化做成 3D 的模型,这既是现在时代的需求,同时,在对人的感官刺激方面有着更好的作用,3D 模型还能看出不同的飞行在不同的高度上,对于航线的宏观规划也有很积极的现实意义。

我理想中的可视化就是将大量的数据处理成简单明了的图案,可以是 3D 的模型,也可以是二维的图表,在将数据可视化之后我们可以直观明了的对于数据有清晰的认识,比如我么在项目中对于城市航线的可视化,当看到北京密密麻麻的航线图时,才了解到北京不愧是首都,他的飞行便利程度是前所未有的,你在其中甚至可以看到凌晨的时候仍然灯火不息的飞行。我认为可视化的意义就在与数据处理之后让人更清晰的认识数据,无论其怎么变化,使什么算法与方法,可视化的意义就在于此。

4. 参考文献

[1] pyecharts 的文档, pyecharts 官网[EB/OL].http://pyecharts.org

[2] 杨天克 缪剑 徐水平 杜丹,航线设计与设计结果的可视化仿真[J],测绘信息与工程,2007,(01),16-18

[3] 杨国志 江业峰,基于 python 的聚焦网络爬虫数据采集系统设计与实现[J],科学技术创新,2018,(27),73-74

[4] 王华 马亮 顾明,线程池技术研究与应用[J],计算机应用研究,2005,(11),141-142+145

[5] 洪敏,吴红亚,杨保华.基于 HTML 的 ECharts 的动态数据显示前端设计[J].计算机时代,2018(08):27-28+32.

[6] Maneesh Agrawala.*Visualizing Route Maps*, *Ph.D. Dissertation*, Stanford University, January 2002.