非关系型数据存储介质及其应用实验报告

李亦杨[1], 胡泓震[2], and 刁泽皓[3]

[1] 学号：10195101467
[2] 学号：10195101485
[3] 学号：10195101470

目录

1 需求选择

1.1 所有需求

全部需求如下：

1	线路基本信息	11	统计站点数量
2	线路站点信息	12	统计路线类型
3	站点停靠线路	13	查询重复站点
4	起止沿线站点	14	查询线路换乘
5	最短路径	15	统计站台连接
6	直达路线判断	16	统计路线站点
7	线路班次信息	17	统计运行时间
8	站点某时线路	18	计算重复系数
9	站点某时某线	19	线路创建
10	统计停靠路线	20	线路删除更新

1.2 选择实现的需求

基于上述需求，我们最终实现的需求如下：

1	线路基本信息	11(a)	统计站点数量
2	线路站点信息	12	统计路线类型
3	站点停靠线路	13	查询重复站点
4	起止沿线站点	15	统计站台连接
5(a, b)	最短路径	16	统计路线站点
6	直达路线判断	17	统计运行时间
10	统计停靠路线	20(a, b)	线路删除更新

总共完成的需求分为 32 分。

2 数据库设计

2.1 总体设计

数据库分为三种节点：Station, Line, 和 Run。

一个 Station 节点存储一个车站，具有三个属性：english, name 和 id。一个 Line 节点存储大部分的公交信息，包括：

directional, interval, kilometer, name, onewayTime, type, start_time, end_time, departure, destination, direction, route.

一个 Run 节点存储某线路的一个班次的时间表，包括三个属性：line_id, direction, time。除此之外，数据库内还建立了两种关系：Connection 和 BelongTo

一个 Connection 关系从一个 Station 节点甲指向另一个 Station 节点乙，包括 lines 属性，存储从甲开向乙的所有线路。

一个 BelongTo 关系从一个 Run 节点指向一个 Line 节点，表示 Run 节点是 Line 节点的某个班次。

2.2 结点实例

2.2.1 Station 节点实例

```
{
        "identity": 19887,
        "labels": [
                "Station"
        ],
        "properties": {
                "english": "YongTongLu",
                "name": "永通路",
                "id": "41394"
        }
}
```

2.2.2 Line 节点实例

```
{
        "identity": 22330,
        "labels": [
                "Line"
        ],
        "properties": {
```

```
        "onewayTime": 52,
        "destination": "花明公交站",
        "end_time": "23:59",
        "type": "干线",
        "start_time": "6:00",
        "route": [
                "16560",
                "803",
                "98730",
                "14214",
                "761",
                "750",
                "744",
                "730",
                "100201",
                "1104",
                "1148",
                "3654",
                "15343",
                "22007",
                "22011",
                "23048",
                "23058",
                "23084",
                "23114",
                "27327",
                "23133",
                "27760",
                "27810",
                "27789",
                "27711",
                "27732",
                "27698",
                "23351",
                "27676"
        ],
        "directional": true,
```

```
44                  "kilometer": 15.0,
45                  "name": "1",
46                  "interval": 5,
47                  "departure":
48                  "金河客运站",
49                  "direction": "up"
50          }
51  }
```

2.2.3 Run 节点实例

```
1   {
2           "identity": 0,
3           "labels": [
4                   "Run"
5           ],
6           "properties": {
7                   "time": [
8                           "06:26",
9                           "06:28",
10                          "06:30",
11                          "06:34",
12                          "06:35",
13                          "06:37",
14                          "06:39",
15                          "06:41",
16                          "06:43",
17                          "06:45",
18                          "06:47",
19                          "06:49",
20                          "06:51",
21                          "06:53",
22                          "06:55",
23                          "06:57",
24                          "06:59",
25                          "07:01",
26                          "07:03",
27                          "07:05",
```

```
                    "07:07",
                    "07:09",
                    "07:11",
                    "07:13",
                    "07:14",
                    "07:15"
                ],
                "line_id": "10",
                "direction": "down"
            }
}
```


2.3 关系实例

2.3.1 Connection 关系实例


```
[{"name":"金河公园","english":"JinHePark","id":"803"},
{"lines":["1","2","N11","N12","43","72","218"]},
{"english":"PeaceBridge","name":"平桥","id":"98730"}]
```


2.4 关系实例

2.4.1 BelongTo 关系实例


```
[{"time":["06:26","06:28","06:30","06:34","06:35","06:37","06:39","
    06:41","06:43",
"06:45","06:47","06:49","06:51","06:53","06:55","06:57","06:59","
    07:01","07:03",
"07:05","07:07","07:09","07:11","07:13","07:14","07:15"],"line_id":"
    10",
"direction":"down"},
{},
{"onewayTime":49,"destination":"永丰公交站","end_time":"21:30","type"
    :"干线",
"start_time":"6:20","route":["2827","2073","2094","2104","2121","2229
    ","3639","6408",
"6401","6388","6378","6362","62709","62728","59583","62764","62752","
    62778","62544",
"62521","46786","46588","62421","62369","56747","56821"],"directional
    ":true,
```

10 "kilometer":14.0,"name":"10","interval":6,"departure":"科北路","direction":"down"}]

3 需求实现

3.1 需求 1

这是最简单的一个需求，要求根据线路代号（lineId）返回线路基本信息，包括线路首尾站名(route)、线路是否有向（directional)、线路长度（length)、线路代号（lineId)、单程运行时间（oneWayTime)、班次间隔（interval)、线路类型（type）以及线路运行时间（runtime)。

Cypher

```
@Query("""
        match
                (l:Line {name: {line_name}})
        return l limit 1
        """)
Line find_lineId_line(String line_name);
```

匹配到一个 name 是 line_name 的 Line 就返回。

业务层

```
    public JSONObject find_lineId_line(String lineId){
        JSONObject obj = new JSONObject();
        if(linerepository.find_lineId_line(lineId) != null){
            Line line = linerepository.find_lineId_line(lineId);
            obj.put("route",line.getDeparture()+"-"+line.
                getDestination());
            obj.put("directional",line.getDirectional());
            obj.put("length",line.getKilometer());
            obj.put("lineId",line.getName() + "路");
            obj.put("interval",line.getInterval());
            obj.put("oneWayTime",line.getOnewayTime());
            obj.put("type",line.getType());
            obj.put("runtime",line.getStart_time()+"-"+line.
                getEnd_time());
        }
        return obj;
    }
```

这些信息都保存在实体层的 Line 类中，直接调用 LineRepository 层的 find_lineId_line 函数，返回一个 line 对象，将其各个属性转化为 JSONObject 对象并返回。

前端界面测试结果

3.2 需求 2

要求根据线路名称返回途径站点信息，站点信息对应实体层 Station 类中的三个属性：id，name，english。

Cypher

```
@Query("""
        match (l:Line {name: {line_id}, direction: {direction}})
        unwind l.route as k
        match (s:Station {id: k})
        return s""")
ArrayList<Station> find_route_station(String line_id, String
    direction
```

匹配到对应的 Line 节点后，通过 route 属性查询对应站点并返回。

业务层

```
    public JSONArray find_route_station(String line_id,String
        direction){
        JSONArray arr = new JSONArray();
```

```
3           ArrayList<Station> station;
4           station = stationrepository.find_route_station(line_id,
                direction);
5           if(!station.isEmpty()){
6               for(int i = 0; i<station.size();i++)
7               {
8                   JSONObject obj = new JSONObject();
9                   Station s = station.get(i);
10                  obj.put("id",s.getId());
11                  obj.put("name",s.getName());
12                  obj.put("english",s.getEnglish());
13                  arr.add(obj);
14              }
15          }
16          return arr;
17      }
```

调用 StationRepository 中的 find_route_station 函数，返回一个 ArrayList<Station>。由于途径多个站点，返回一个 JSONArray 对象，其中的每个 JSONObject 对应一个 Station 列表中的一项。

前端界面测试结果


展开
收起
线路信息
线路基本信息
线路站点信息
线路站点信息
线路
2
路
线路方向
上行
查询
清空
站点ID
站点名称
站点英语名称
7542
兴义镇(始发站)
XingYiZhen
7527
永盛(始发站)
YongSheng
7504
南华大道
NanHuaDaDao
7485
南华大道南
NanHuaDaDao S
21460
金河客运站
JinHeKeYun
803
金河公园
Jin He Park
98730
平桥
Peace Bridge
李亦杨，刁泽皓，胡泓震


展开
收起
线路信息
14417
双桥子立交北
Shuangqiaozi N
14423
学府路
Xuefulu
97836
地铁东井巷
Dongjinxiang
16772
金飞路一段
Jinfeilu One
2800
蓝天立交北
LanTianLiJiao N
6855
蓝天立交南
LanTianLiJiao S
6901
百花路东
Bai Hua Lu E
6854
百花路西
Bai Hua Lu W
6818
商贸城
Trade City
14495
火车西站公交站(终点站)
Huo Che Zhan
共28个站点
李亦杨，刁泽皓，胡泓震

3.3 需求 3

根据车站名称返回停靠的线路信息（区分上下行），若车站名字存在重复，则按车站 id 进行分组。

Cypher

```
@Query("""
        match (s:Station {name: {station_name}})
        return distinct s.id
        """)
ArrayList<String> find_stationName_routeName_stationId(String
    station_name);

@Query("""
        match (s:Station {id: {station_id}}) -[r]- ()
        match (l:Line) where l.name in r.lines and s.id in l.route
        return distinct l.name + l.direction
        """)
ArrayList<String> find_stationName_routeName_lineId(String station_id
    );
```

第一个函数接受站名返回站 ID。
第二个函数接受站 ID，查询 route 属性中存在该 ID 的 Line 节点。这些节点就是停靠该站的，然后将它们返回。

业务层

```
    public JSONArray find_stationName_routeName(String stationName){
        JSONArray arr = new JSONArray();
        ArrayList<String> res_stationId = stationrepository.
            find_stationName_routeName_stationId(stationName);

        ArrayList<ArrayList<String>> res_lineId = new ArrayList<>();

        if(!res_stationId.isEmpty()){

            for(int i = 0; i < res_stationId.size(); i ++){
                String tmpid = res_stationId.get(i);
                ArrayList<String> tmpres_lineId_t =
                    stationrepository.
                    find_stationName_routeName_lineId(tmpid);

```

```
13                  ArrayList<String> tmpres_lineId = new ArrayList<>();
14
15                  if(!tmpres_lineId_t.isEmpty()){
16                      for(int k = 0; k < tmpres_lineId_t.size(); k ++){
17                          String tmp = tmpres_lineId_t.get(k);
18                          if(tmp.contains("up"))
19                              tmp = tmp.replace("up", "路上行");
20                          else if(tmp.contains("down"))
21                              tmp = tmp.replace("down", "路下行");
22                          else if(tmp.contains("circle"))
23                              tmp = tmp.replace("circle", "路环线");
24
25                          tmpres_lineId.add(tmp);
26                      }
27                  }
28
29                  res_lineId.add(tmpres_lineId);
30              }
31          }
32
33          ArrayList<Demand3> result = new ArrayList<>();
34
35          if(!res_stationId.isEmpty()){
36              for(int i = 0; i < res_stationId.size(); i ++){
37                  Demand3 dem = new Demand3();
38                  dem.stationId = res_stationId.get(i);
39                  dem.lineIds = res_lineId.get(i);
40                  result.add(dem);
41              }
42          }
43
44          if(!result.isEmpty()){
45              for(int i=0;i<result.size();i++)
46              {
47                  JSONObject obj = new JSONObject();
48                  Demand3 demand3 = new Demand3();
49                  demand3 = result.get(i);
```

```
50                  obj.put("id",demand3.stationId);
51                  String str = "";
52                  ArrayList<String> lineIds;
53                  lineIds =demand3.lineIds;
54                  if(!lineIds.isEmpty()){
55                      for(int j = 0 ; j < lineIds.size() ; j++)
56                      {
57                          str += "\"";
58                          str += lineIds.get(j);
59                          str += "\" ";
60                      }
61                  }
62                  obj.put("routes",str);
63                  arr.add(obj);
64              }
65          }
66          return arr;
67      }
```

首先调用 StationRepository 中的 find_stationName_routeName_stationId 函数（函数命名有些复杂因为后期 debug 过程中对函数功能进行了修改），返回 ArrayList<String> res_stationId，里面是输入的车站名称对应的所有车站 id。

然后对 res_stationId 中的每一项分别调用 find_stationName_routeName_lineId 和 find_stationName_rou 函数，前者返回经过该车站的线路 id（是一个 ArrayList<String>），后者返回线路方向（也是 ArrayList<String>）。将 stationId、lineId 和 direction 存入 Demand3 对象列表中，然后将其转换为 JSONArray 返回。在转换过程中还涉及了对线路 id 和方向字符串的拼接。

前端界面测试结果

3.4 需求 4

根据要乘坐的路线代号和起止站点，返回线路运行方向、途径站点、运行时长。

Cypher

```
@Query("""
        match (l:Line {name: {line_id}, direction: {direction}})
        unwind l.route as k
        match (s:Station {id: k})
        return s""")
ArrayList<Station> find_route_station(String line_id, String
    direction);
```

首先通过站名查出站 ID，然后通过 route 属性匹配先经过起点后经过终点的 Line。再通过 Line 查到对应的一个 Run 节点，从 Run 节点中取出到达两站的时间。最后返回站名、方向，到达起点和终点的时刻，以及交给 service 层遍历用的起点和终点在 route 中的索引。
然后，service 通过第二个函数查询起点和终点之间的每个站。

业务层

```
public JSONObject find_lineId_stationName_path(String lineId,String
    stationName1,String stationName2){
        JSONObject obj = new JSONObject();
        Demand4 result = new Demand4();

        String res_lineName = linerepository.
            find_lineId_stationName_path_lineName(lineId, stationName1
            , stationName2);
        String res_direction = linerepository.
            find_lineId_stationName_path_direction(lineId,
            stationName1, stationName2);
        String res_depttime = linerepository.
            find_lineId_stationName_path_departtime(lineId,
            stationName1, stationName2);
        String res_desttime = linerepository.
            find_lineId_stationName_path_desttime(lineId, stationName1
            , stationName2);
        int res_deptind = linerepository.
            find_lineId_stationName_path_departind(lineId,
            stationName1, stationName2);
        int res_destind = linerepository.
            find_lineId_stationName_path_destind(lineId, stationName1,
```

```
            stationName2);
11
12          String res_direct = new String();
13          if(Objects.equals(res_direction, "up"))
14              res_direct = "上行";
15          else if (Objects.equals(res_direction, "down"))
16              res_direct = "下行";
17          else if (Objects.equals(res_direction, "circle"))
18              res_direct = "环线";
19
20          result.lineName = res_lineName;
21          result.direction = res_direction;
22          result.departure_time = res_depttime;
23          result.destination_time = res_desttime;
24          result.departure_index = res_deptind;
25          result.destination_index = res_destind;
26
27          if(result != null){
28              obj.put("lineName",result.lineName + "路" + res_direct);
29              SimpleDateFormat ft = new SimpleDateFormat ("HH:mm");
30              Date t1;
31              long l1;
32              Date t2;
33              long l2;
34              int runtime;
35              if((!(result.departure_time==null))&&(!(result.
                    destination_time==null)))
36              {
37                  try{
38                      t1 = ft.parse(result.destination_time);
39                      l1 = t1.getTime();
40                      t2 = ft.parse(result.departure_time);
41                      l2 = t2.getTime();
42                      runtime = (int)((l1 - l2)/60000);
43                      obj.put("runTime",runtime);
44                  }catch (ParseException e){
45                      System.out.println("Unparseable using " + ft);
```

```
46                  }
47              }
48              JSONArray arr =new JSONArray();
49              int departure_index = result.departure_index;
50              int destination_index = result.destination_index;
51              for(int i = departure_index;i<=destination_index;i++)
52              {
53                  Station sta = stationrepository.
                        find_route_station_by_index(result.lineName,result
                        .direction,i);
54                  JSONObject s = new JSONObject();
55                  s.put("id",sta.getId());
56                  s.put("name",sta.getName());
57                  s.put("english",sta.getEnglish());
58                  arr.add(s);
59              }
60              obj.put("stations",arr);
61          }
62
63          return obj;
64      }
```

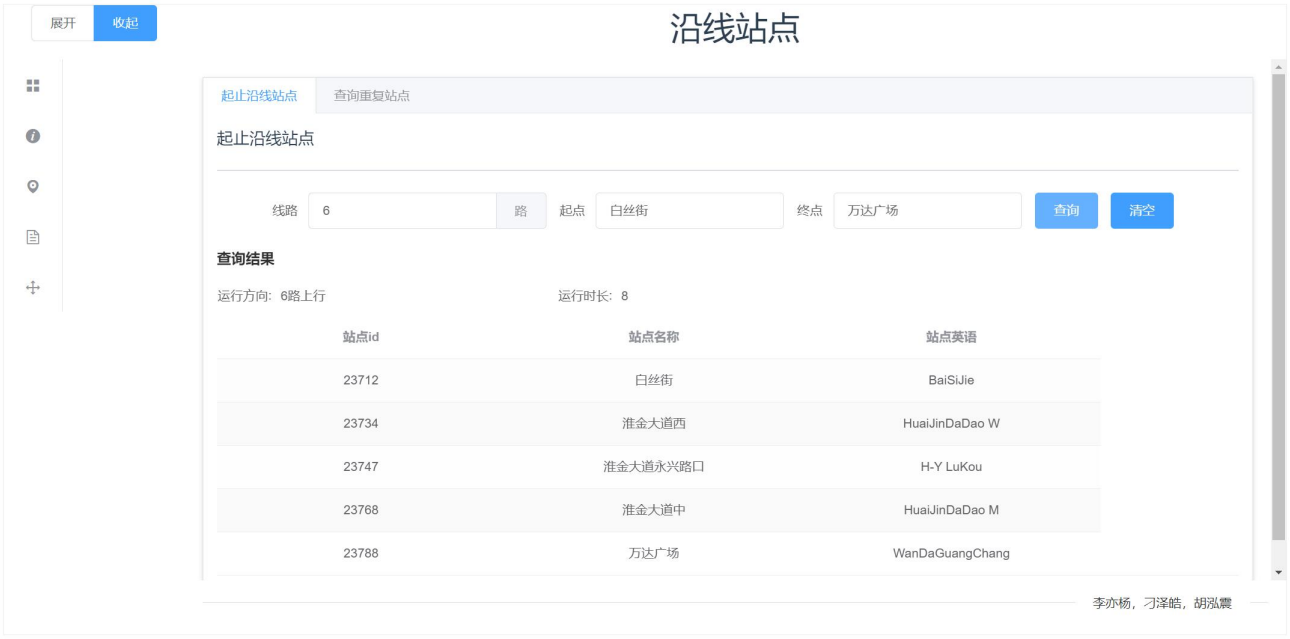
创建 Demand4 类（里面包含 String lineName; String direction; String departure_time; String destination_time; int departure_index; int destination_index）的列表 result。
然后分别调用 LineRepository 层的函数:

- find_lineId_stationName_path_lineName
- find_lineId_stationName_path_direction
- find_lineId_stationName_path_departtime
- find_lineId_stationName_path_desttime
- find_lineId_stationName_path_departind
- find_lineId_stationName_path_destind

传入的参数都是线路 id 和起始终止站点名，分别返回站点间线路的名称、方向、到达起始点的时间、到达终止点的时间、起始点的索引值、终止点的索引值。随后根据两个时间点算出线路运行时间，并根据首尾索引遍历他们中间的所有索引值对应的站点信息，并保存在一个 JSONArray

对象中，最后与运行时间和线路名称整合成一个 JSONObject 对象返回。

前端界面测试结果



3.5 需求 5

3.5.1 a

查询某两个站台之间的最短路径，基于 ID 查询
Cypher

```
1  @Query("""
2          match (ss:Station{id:{station_id1}}), (se:Station{id:{
               station_id2}})
3          unwind nodes(shortestpath((ss)-[*0..15]-> (se))) as res
4          return res.id
5          """)
6  ArrayList<String> shortestpath_by_id_id(String station_id1, String
       station_id2);
7
8  @Query("""
9          match (ss:Station{id:{station_id1}}), (se:Station{id:{
               station_id2}})
10         unwind nodes(shortestpath((ss)-[*0..15]-> (se))) as res
11         return res.name
12         """)
13 ArrayList<String> shortestpath_by_id_name(String station_id1, String
       station_id2);
14
15 @Query("""
16         match (ss:Station{id:{station_id1}}), (se:Station{id:{
               station_id2}})
17         unwind nodes(shortestpath((ss)-[*0..15]-> (se))) as res
18         return res.english
19         """)
20 ArrayList<String> shortestpath_by_id_eng(String station_id1, String
       station_id2);
```

Neo4j 内置了查询最短路径的函数 shortestpath，因而只需通过站点 id 查询作为出发和目的站点的站点，随后利用 shortestpath 查询最短路径，并输出结点即可。
由于有的 Station 关系结点不包含进入方向的关系，有的不包含出方向的关系，因而若直接返回 Station 实体类会发生映射问题，解决方式是拆分查询语句，并使之返回多个字符串。

业务层

```
1     public JSONArray find_shortestRoute_id(String station1, String
          station2){
2         JSONArray arr = new JSONArray();
3         ArrayList<String> station_id;
4         station_id = linerepository.shortestpath_by_id_id(station1,
              station2);
5         ArrayList<String> station_name;
6         station_name = linerepository.shortestpath_by_id_name(
              station1, station2);
7         ArrayList<String> station_eng;
8         station_eng = linerepository.shortestpath_by_id_eng(station1,
              station2);
9         if(!station_id.isEmpty()){
10            for(int i = 0; i<station_id.size();i++)
11            {
12                JSONObject obj = new JSONObject();
13                String tmpid = station_id.get(i);
14                String tmpname = station_name.get(i);
15                String tmpeng = station_eng.get(i);
16                obj.put("id",tmpid);
17                obj.put("name",tmpname);
18                obj.put("english",tmpeng);
19                arr.add(obj);
20            }
21        }
22        return arr;
23    }
```

逻辑较为简单,与需求6类似,只需要调用三个函数:shortestpath_by_id_id, shortestpath_by_id_name, shortestpath_by_id_eng，再将返回值输出到 JSON 数组中，最终返回 JSON 对象即可。

前端界面测试结果

3.5.2 b

查询某两个站台之间的最短路径，基于 ID 查询

Cypher 名字

```
1 @Query("""
2         match (s:Station{name:{station_name}})
3         return s.id
4         """)
5 ArrayList<String> get_all_station_ids_by_name(String station_name);
```

考虑到查询最短路径本质上还是根据站点 Id 进行查询，因而 Cypher 层提供一个利用站点名查询所有同名站点 Id 的函数，其余操作交给业务层完成。

业务层

```
1     public JSONArray find_shortestRoute_name(String station1, String
          station2){
2         JSONArray arr = new JSONArray();
3         ArrayList<String> station1_id = linerepository.
              get_all_station_ids_by_name(station1);
4         ArrayList<String> station2_id = linerepository.
              get_all_station_ids_by_name(station2);
5         ArrayList<Station> routes = new ArrayList<>();
6
7         int count = Integer.MAX_VALUE;
```

```
8
9           if(!station1_id.isEmpty()){
10              if(!station2_id.isEmpty()){
11                  for(int i = 0; i < station1_id.size(); i ++){
12                      for(int j = 0; j < station2_id.size(); j ++){
13                          ArrayList<String> tmpids = linerepository.
                                shortestpath_by_id_id(station1_id.get(i),
                                station2_id.get(j));
14                          ArrayList<String> tmpnames = linerepository.
                                shortestpath_by_id_name(station1_id.get(i)
                                , station2_id.get(j));
15                          ArrayList<String> tmpengs = linerepository.
                                shortestpath_by_id_eng(station1_id.get(i),
                                 station2_id.get(j));
16
17                          ArrayList<Station> tmproutes = new ArrayList
                                <>();
18
19                          for(int k = 0; k < tmpids.size(); k ++){
20                              Station tmps = new Station();
21                              tmps.setId(tmpids.get(k));
22                              tmps.setName(tmpnames.get(k));
23                              tmps.setEnglish(tmpengs.get(k));
24
25                              tmproutes.add(tmps);
26                          }
27
28                          if(tmproutes.size() != 0 && tmproutes.size()
                                < count){
29                              count = tmproutes.size();
30                              routes = tmproutes;
31                          }
32                      }
33                  }
34
35
36              }
```

```
37          }
38
39          if(!routes.isEmpty()){
40              for(int i = 0; i<routes.size();i++)
41              {
42                  JSONObject obj = new JSONObject();
43                  Station s = routes.get(i);
44                  obj.put("id",s.getId());
45                  obj.put("name",s.getName());
46                  obj.put("english",s.getEnglish());
47                  arr.add(obj);
48              }
49          }
50          return arr;
51      }
```

我们对需求的理解是，对同名站点 A, B，以及另一站点 C 而言，倘若 A C 间存在路径 ac，B C 间存在路径 bc，则选择 ac 与 bc 中较短的输出，换言之总是保证结果只有一条路径。

总体上来说，业务层维护三个 ArrayList，routes 用来存储最终答案，而另两个则用来存储起点终点站的全部同名站点 Id。随后利用两次 for 循环，为每一组（起点站，终点站）进行最短路线查询。同时设置一个 int 型变量，初始值位 Integer.MAX_VALUE，用于辅助检测最短路径，倘若在一次循环中所查出的路径所含站点数比该变量小，就将该变量重新赋值，并将此次循环查处的路径存储到 routes。由此便可以得到最短路径。

最后将 routes 内站点封装进入 JSON 对象，并返回。

前端界面测试结果

3.6 需求 6

根据起止站点名称查询是否存在直达线路，如果有返回线路名称及方向，如果没有返回提示信息。

Cypher

```
@Query("""
        match (s1:Station{name:{station1}}), (s2:Station{name:{
            station2}})
        with s1.id as departure, s2.id as destination
        match
                (l:Line) where apoc.coll.indexOf(l.route, departure)
                    > 0 and apoc.coll.indexOf(l.route, departure) <
                    apoc.coll.indexOf(l.route, destination)
        return l.name + l.direction
        """)
ArrayList<String> find_directRoute(String station1, String station2);
```

根据起点和终点站名查出站 ID，然后根据站 ID 查出 Line 的 route 中起点索引小于终点的节点，即表示该线路从起点运行到终点，之后返回所有匹配的线路。

业务层

```
    public JSONArray find_directRoute(String station1, String
        station2){
```

```
2           JSONArray arr = new JSONArray();
3           ArrayList<String> route;
4           route = linerepository.find_directRoute(station1, station2);
5           String s = "";
6           if(!route.isEmpty()){
7               for(int i = 0 ; i<route.size(); i++)
8               {
9                   String tmp1 = route.get(i);
10                  if(tmp1.contains("up"))
11                      tmp1 = tmp1.replace("up", "路上行");
12                  else if(tmp1.contains("down"))
13                      tmp1 = tmp1.replace("down", "路下行");
14                  else if(tmp1.contains("circle"))
15                      tmp1 = tmp1.replace("circle", "路环线");
16                  s += tmp1;
17                  JSONObject obj = new JSONObject();
18                  obj.put("route",s);
19                  arr.add(obj);
20              }
21          }
22          return arr;
23      }
```

调用 LineRepository 中的 find_directRoute 函数，返回 ArrayList<String> route，里面保存着所有直达线路名称，之后只要将里面每一个对象转化为一个 JSONObject 然后拼接成一个 JSONArray 返回。在转化过程中对 route 的每一项进行判断，将字符串中英文转换为中文。

前端界面测试结果

3.7 需求 10

统计停靠线路最多的站点前 15 位，结果显示为 4 列：站点 id，站点名，线路条数，线路名称。

Cypher

```
@Query("""
        match
                (l:Line) where {station_id} in l.route
        return l.name + l.direction
        """)
ArrayList<String> get_lines_in_a_station(String station_id);
```

第一个函数简单返回所有的站 ID。
第二个函数简单返回所有经过该站的线路。

业务层

```
    public JSONArray most_line_station(){
        JSONArray arr = new JSONArray();
        ArrayList<String> stations;
        stations = stationrepository.get_all_station_id();
        ArrayList<StationLines> sta_lin = new ArrayList<>();
        for(int i = 0;i < stations.size();i++)
        {
            StationLines a = new StationLines();
            a.stationId = stations.get(i);
            a.station = stationrepository.get_station_name_by_id(a.
                stationId);
            ArrayList<String> line_f;
            line_f = linerepository.get_lines_in_a_station(a.
                stationId);

            ArrayList<String> line = new ArrayList<>();

            if(!line_f.isEmpty()){
                for(int k = 0; k < line_f.size(); k ++){
                    String tmp = line_f.get(k);
                    if(tmp.contains("up"))
                        tmp = tmp.replace("up", "路上行");
                    else if(tmp.contains("down"))
                        tmp = tmp.replace("down", "路下行");
                    else if(tmp.contains("circle"))
```

```
24                          tmp = tmp.replace("circle", "路环线");
25
26                      line.add(tmp);
27                  }
28              }
29
30              String s = "";
31              if(!line.isEmpty()){
32                  for(int j = 0;j<line.size();j++)
33                  {
34                      s += line.get(j);
35                      if(j < line.size() - 1)
36                          s+=",";
37                  }
38              }
39              a.route = s;
40              a.num = line.size();
41              sta_lin.add(a);
42          }
43          Collections.sort(sta_lin,new SortByNum());
44          if(!sta_lin.isEmpty()){
45              for(int i = 0;i<15;i++)
46              {
47                  JSONObject obj = new JSONObject();
48                  StationLines a = new StationLines();
49                  a = sta_lin.get(i);
50                  obj.put("stationId",a.stationId);
51                  obj.put("station",a.station);
52                  obj.put("num",a.num);
53                  obj.put("route",a.route);
54                  arr.add(obj);
55              }
56          }
57          return arr;
58      }
```

```
1  class SortByNum implements Comparator {
2      public int compare(Object o1,Object o2){
```

```
3            if(((StationLines)o1).num < ((StationLines)o2).num)
4                return 1;
5            if(((StationLines)o1).num > ((StationLines)o2).num)
6                return -1;
7            return 0;
8        }
9    }
```

首先建立了一个新类 StationLines，里面包含四个属性：String stationId；String station；int num；String route。并初始化一个该类的列表 sta_lin，这对应着结果的 4 个属性。先调用 StationRepository 中的 get_all_station_id()，返回所有站点 id 的列表 stations，然后遍历 stations，对每个线路 id 将其加入 sta_lin 中的 stationId 属性，然后对每个 id 调用 get_station_name_by_id 得到 stationName，将其加入 sta_lin 的 station 属性，再调用 get_lines_in_a_station 函数得到经过该站点的线路名的列表，将其变为字符串后加入 sta_lin 的 route 属性，将其元素个数加入 num 属性。然后调用 StationLines 的排序类对其按 num 属性排序，随后转为 JSONArray 对象输出。

前端界面测试结果

3.8 需求 11

3.8.1 a

统计地铁站、起点站、终点站数量，并返回站点名。

Cypher

```
@Query("""
        match (s:Station) where s.name starts with "地铁"
        return distinct s.name
        """)
ArrayList<String> count_subway_station();

@Query("""
        match (s:Station) where s.name ends with "(始发站)" or not ()
            --> (s)
        return distinct s.name
        """)
ArrayList<String> count_start_station();

@Query("""
```

```
14          match (s:Station) where s.name ends with "(终点站)" or not (s
                ) --> ()
15          return distinct s.name
16          """)
17  ArrayList<String> count_end_station();
```

使用三个类似的 cypher 语句分别进行查询，利用 match with 关键字来寻找符号条件的全部站点，并返回其中文名称。

业务层

```
1       public JSONObject special_station(){
2           JSONObject obj = new JSONObject();
3           ArrayList<String> subway;
4           subway = stationrepository.count_subway_station();
5           ArrayList<String> start;
6           start = stationrepository.count_start_station();
7           ArrayList<String> end;
8           end = stationrepository.count_end_station();
9           if(!subway.isEmpty())
10          {
11              JSONObject obj1 = new JSONObject();
12              obj1.put("type","地铁站");
13              obj1.put("amount",subway.size());
14              JSONArray arr =new JSONArray();
15              String s = "";
16              for(int i =0;i<subway.size();i++)
17              {
18                  JSONObject t = new JSONObject();
19                  s = subway.get(i);
20                  t.put("station", s);
21                  arr.add(t);
22              }
23              obj1.put("stations",arr);
24              obj.put("subway",obj1);
25          }
26          if(!start.isEmpty())
27          {
```

```
28              JSONObject obj1 = new JSONObject();
29              obj1.put("type","始发站");
30              obj1.put("amount",start.size());
31              JSONArray arr =new JSONArray();
32              String s = "";
33              for(int i =0;i<start.size();i++)
34              {
35                  JSONObject t = new JSONObject();
36                  s = start.get(i);
37                  t.put("station", s);
38                  arr.add(t);
39              }
40              obj1.put("stations",arr);
41              obj.put("start",obj1);
42          }
43          if(!end.isEmpty())
44          {
45              JSONObject obj1 = new JSONObject();
46              obj1.put("type","终点站");
47              obj1.put("amount",end.size());
48              JSONArray arr =new JSONArray();
49              String s = "";
50              for(int i =0;i<end.size();i++)
51              {
52                  JSONObject t = new JSONObject();
53                  s = end.get(i);
54                  t.put("station", s);
55                  arr.add(t);
56              }
57              obj1.put("stations",arr);
58              obj.put("end",obj1);
59          }
60
61          return obj;
62      }
```

直接调用 dao 层上述三个函数即可。

为了确保前端界面的美观程度，将地铁站、始发站和终点站分别整合为一个 JSON 对象，并将

这三个 JSON 对象整合为一个 JSON 对象返回。

前端界面测试结果

3.9 需求 12

分组统计常规公交 (包括干线、支线、城乡线、驳接线、社区线)、快速公交 (K 字开头)、高峰公交 (G 字开头)、夜班公交 (N 字开头)。返回四类名称及对应数量。

Cypher

```
@Query("""
        match (l:Line) where l.name =~ "^[0-9]+"
        return count(l)
        """)
Integer count_type_l();

@Query("""
        match (k:Line) where k.name starts with "K"
        return count(k)
        """)
Integer count_type_k();

@Query("""
        match (g:Line) where g.name starts with "G"
        return count(g)
        """)
Integer count_type_g();

```

```
@Query("""
        match (n:Line) where n.name starts with "N"
        return count(n)
        """)
Integer count_type_n();
```

每个函数利用正则匹配对应类型站点名，然后返回匹配的站数。

业务层

```
    public JSONArray count_type(){
        JSONArray arr = new JSONArray();
        int[] count = new int[4];
        count[0] = linerepository.count_type_l();
        count[1] = linerepository.count_type_k();
        count[2] = linerepository.count_type_g();
        count[3] = linerepository.count_type_n();
        ArrayList<String> type = new ArrayList<>();
        type.add("常规公交");
        type.add("快速公交");
        type.add("高峰公交");
        type.add("夜班公交");
        for(int i = 0 ; i < 4; i++)
        {
            JSONObject obj = new JSONObject();
            int a = count[i];
            obj.put("type",type.get(i));
            obj.put("num",a);
            arr.add(obj);
        }
        return arr;
    }
```

分别调用 count_type_l, count_type_k, count_type_g, count_type_n 四个函数返回四种线路类型的数量，然后将其整合为 JSONArray 对象返回。

事实上，原本设想通过建立 Demand 系列里虚构类来对非实体对象进行抽象，从而便于通过同一条 Cypher 语句来返回获得的数据，然而由于使用的 SDN6 移除了 @QueryMapper 等命令，而选择通过 Neo4jDriver 与 Driver 的形式来完成了 Query 与非实体类之间的映射的实现，考虑

到可能用到的非实体类较少，且该种形式过于复杂，因而我们选择了拆分 Query 查询语句的方式进行实现。之后需求倘若有涉及到 Demand 类的使用的，在数据库查询层面都利用拆分的方式进行了实现。

前端界面测试结果

3.10 需求 13

查询两条线路重复的站点名，返回相同站点的 id，名称、英文名。

Cypher

由 Service 层调用 find_route_station 两次。

业务层

```
1       public JSONArray find_sameStations(String id1,String direction1,
            String id2,String direction2){
2           JSONArray arr = new JSONArray();
3           ArrayList<Station> result1;
4           result1 = stationrepository.find_route_station(id1,direction1
                );
```

```
5           ArrayList<Station> result2;
6           result2 = stationrepository.find_route_station(id2,direction2
                );
7           Station sta;
8           //result1.retainAll(result2);
9
10
11
12          ArrayList<Station> to_delete = new ArrayList<>();
13          if(!result1.isEmpty()){
14              for(int i = 0; i < result1.size(); i ++){
15                  boolean has = false;
16                  if(!result2.isEmpty()){
17                      for(int j = 0; j < result2.size(); j ++){
18                          if(Objects.equals(result1.get(i).getId(),
                                result2.get(j).getId())){
19                              has = true;
20                              break;
21                          }
22                      }
23                  }
24                  if(!has){
25                      to_delete.add(result1.get(i));
26                  }
27              }
28          }
29
30          for(int i = 0; i < to_delete.size(); i ++)
31              result1.remove(to_delete.get(i));
32
33          if(!result1.isEmpty()){
34              for(int i = 0 ; i < result1.size() ; i++)
35              {
36                  sta = result1.get(i);
37                  JSONObject obj = new JSONObject();
38                  obj.put("station_id",sta.getId());
39                  obj.put("station_name",sta.getName());
```

```
40                  obj.put("station_english",sta.getEnglish());
41                  arr.add(obj);
42              }
43          }
44          return arr;
45      }
```

对两个线路名分别调用 find_route_station 函数，返回两个线路对应的站点列表，然后对两个列表求交集，将结果转化为 JSONArray 对象返回。

前端界面测试结果

3.11 需求 15

查询两个相邻站点间线路数量并排序，输出连接数量排序前 15 的两站台名和对应线路数量（考虑方向性）。

Cypher

```
@Query("""
        match(a:Station) -[r]-> (b:Station)
        with a, b, length(r.lines) as cnt
        order by cnt desc
        return a.name limit 15
        """)
ArrayList<String> most_connection_in();

@Query("""
        match(a:Station) -[r]-> (b:Station)
        with a, b, length(r.lines) as cnt
        order by cnt desc
        return b.name limit 15
        """)
```

```
15 ArrayList<String> most_connection_out();
16
17 @Query("""
18         match(a:Station) -[r]-> (b:Station)
19         with a, b, length(r.lines) as cnt
20         order by cnt desc
21         return cnt limit 15
22         """)
23 ArrayList<Integer> most_connection_count();
```

匹配所有 Connection 关系，然后根据 lines 的元素格式降序排序，输出前 15 个。

业务层

```
1     public JSONArray most_connections(){
2         JSONArray arr = new JSONArray();
3         ArrayList<String> res_in = stationrepository.
              most_connection_in();
4         ArrayList<String> res_out = stationrepository.
              most_connection_out();
5         ArrayList<Integer> res_cnt = stationrepository.
              most_connection_count();
6         ArrayList<Demand15> result = new ArrayList<>();
7         if(!res_cnt.isEmpty()){
8             for(int i = 0; i < res_cnt.size(); i++){
9                 Demand15 dem = new Demand15();
10                dem.name_in = res_in.get(i);
11                dem.name_out = res_out.get(i);
12                dem.count = res_cnt.get(i);
13                result.add(dem);
14            }
15
16            Collections.sort(result,new SortByCount());
17            Demand15 a = new Demand15();
18
19            for(int i = 0 ; i < 15 ;i++)
20            {
21                a = result.get(i);
```

```
22                  JSONObject obj = new JSONObject();
23                  obj.put("station1",a.name_in);
24                  obj.put("station2",a.name_out);
25                  obj.put("num",a.count);
26                  arr.add(obj);
27              }
28          }
29          return arr;
30      }
```

```
1  public class SortByCount implements Comparator {
2      public int compare(Object o1,Object o2){
3          if(((Demand15)o1).count < ((Demand15)o2).count)
4              return 1;
5          if(((Demand15)o1).count > ((Demand15)o2).count)
6              return -1;
7          return 0;
8      }
9  }
```

分别调用 most_connection_in, most_connection_out, most_connection_count 函数，返回两个站名列表和一个整数列表，对应每对相邻站点的名字和之间的线路数。将其整合为一个 Demand15 列表，对其按 count 属性降序排序，然后将其前 15 位转为 JSONArray 对象输出。

前端界面测试结果

3.12 需求 16

根据站点数量对线路（含方向）排序显示前 15，返回线路名和对应站点数量。

Cypher

```
@Query("""
        match (a:Line)
        with a, length(a.route) as cnt
        order by cnt desc
        return a.name limit 15
        """)
ArrayList<String> most_station_name();

@Query("""
        match (a:Line)
        with a, length(a.route) as cnt
        order by cnt desc
        return a.direction limit 15
        """)
ArrayList<String> most_station_direction();

@Query("""
        match (a:Line)
        with a, length(a.route) as cnt
        order by cnt desc
        return cnt limit 15
        """)
ArrayList<Integer> most_station_count();
```

对所有 Line 节点，统计 route 内的元素个数，然后按个数降序返回前 15 个。
由于 springboot 不支持返回自定义类的集合，因此拆成三个函数。

业务层

```
        public JSONArray most_stations(){
        JSONArray arr = new JSONArray();

        ArrayList<String> res_name = stationrepository.
            most_station_name();
        ArrayList<String> res_direction = stationrepository.
            most_station_direction();
        ArrayList<Integer> res_cnt = stationrepository.
```

```
            most_station_count();
7           ArrayList<Demand16> result = new ArrayList<>();
8
9           if(!res_cnt.isEmpty()){
10              for(int i = 0; i < res_cnt.size(); i ++){
11                  Demand16 dem = new Demand16();
12                  dem.name = res_name.get(i);
13                  dem.direction = res_direction.get(i);
14                  dem.count = res_cnt.get(i);
15                  result.add(dem);
16              }
17          }
18
19          Demand16 res = new Demand16();
20          if(!result.isEmpty()){
21              for(int i = 0; i < result.size(); i ++){
22                  JSONObject obj = new JSONObject();
23                  res = result.get(i);
24                  if(Objects.equals(res.direction, "up"))
25                      res.direction = "上行";
26                  else if(Objects.equals(res.direction, "down"))
27                      res.direction = "下行";
28                  else if(Objects.equals(res.direction, "circle"))
29                      res.direction = "环线";
30                  obj.put("route", res.name + "路" + res.direction);
31                  obj.put("num", res.count);
32                  arr.add(obj);
33              }
34          }
35          return arr;
36      }
```

分别调用 most_station_name, most_station_direction 和 most_station_count 函数，返回两个字符串列表 res_name(线路名字), res_direction(线路方向), 和一个整数列表 res_cnt(线路中站点数量)，列表的长度为 15，是站点数前 15 位。将其整合为一个 Demand16 列表 result。将其转化为 JSONArray 输出。

前端界面测试结果


展开
收起
主页
最多站点线路
线路
站点数量
1 736路上行 47
2 736路下行 46
3 735路上行 44
4 735路下行 44
5 727路下行 39
6 727路上行 37
7 82路上行 35
8 322路下行 35
9 17路上行 34
10 17路下行 34
11 334路下行 34
李亦杨，刁泽皓，胡泓震


展开
收起
最多站点线路
4 735路下行 44
5 727路下行 39
6 727路上行 37
7 82路上行 35
8 322路下行 35
9 17路上行 34
10 17路下行 34
11 334路下行 34
12 322路上行 34
13 716路下行 33
14 82路下行 33
15 716路上行 33
李亦杨，刁泽皓，胡泓震

3.13 需求 17

由班次数据计算出每班单程运行时间，对时间降序排列输出前 15 条。

Cypher

```
@Query("""
        match
                (r:Run{line_id:{line_name}, direction: {line_direct
                    }})
        return r.time[0] limit 1
        """)
String get_start_time_in_one_run(String line_name, String line_direct
    );

@Query("""
        match
                (r:Run{line_id:{line_name}, direction: {line_direct
                    }})
        return r.time[-1] limit 1
        """)
String get_end_time_in_one_run(String line_name, String line_direct);
```

第一个函数返回所有 Line 节点的名字，供 service 层调用剩下两个函数。
第二个函数匹配指定名字和方向的 Line 节点，然后返回线路开始运行的时间。
第三个函数匹配指定名字和方向的 Line 节点，然后返回线路结束运行的时间。

业务层

```
    public JSONArray longest_time(){
        JSONArray arr = new JSONArray();
        ArrayList<Line> linenames;
        linenames = linerepository.get_all_line_names();

        ArrayList<Demand17> result = new ArrayList<>();

        if(!linenames.isEmpty()){
            for(int i = 0 ; i < linenames.size() ; i++)
            {
```

```
11                  String nam = linenames.get(i).getName();
12                  String direct = linenames.get(i).getDirection();
13                  String start_time = linerepository.
                        get_start_time_in_one_run(nam, direct);
14                  String end_time = linerepository.
                        get_end_time_in_one_run(nam, direct);
15                  SimpleDateFormat ft = new SimpleDateFormat ("HH:mm");
16                  Date t1;
17                  long l1;
18                  Date t2;
19                  long l2;
20                  int runtime;
21
22                  String dir = new String();
23                  if(Objects.equals(direct, "up"))
24                      dir = "路上行";
25                  else if(Objects.equals(direct, "down"))
26                      dir = "路下行";
27                  else if(Objects.equals(direct, "circle"))
28                      dir = "路环线";
29
30                  nam += dir;
31
32                  Demand17 dem = new Demand17();
33                  dem.name = nam;
34
35                  try {
36                      t1 = ft.parse(start_time);
37                      l1 = t1.getTime();
38                      t2 = ft.parse(end_time);
39                      l2 = t2.getTime();
40                      runtime = (int)((l2 - l1)/60000);
41                      dem.time = runtime;
42                  } catch (ParseException e) {
43                      System.out.println("Unparseable using " + ft);
44                  }
45
```

```
46                  result.add(dem);
47              }
48          }
49
50          Collections.sort(result,new SortDemand17ByTime());
51
52          ArrayList<Demand17> res = new ArrayList<>();
53
54          for(int i = 0; i < 15; i ++){
55              res.add(result.get(i));
56          }
57
58          if(!res.isEmpty()){
59              for(int i = 0; i < res.size(); i ++){
60                  Demand17 tmp_dem = res.get(i);
61
62                  JSONObject obj = new JSONObject();
63                  obj.put("route", tmp_dem.name);
64                  obj.put("time", tmp_dem.time);
65                  arr.add(obj);
66              }
67          }
68          return arr;
69      }
70  }
```

```
1   class SortDemand17ByTime implements Comparator {
2       public int compare(Object o1,Object o2){
3           if(((Demand17)o1).time < ((Demand17)o2).time)
4               return 1;
5           if(((Demand17)o1).time > ((Demand17)o2).time)
6               return -1;
7           return 0;
8       }
9   }
```

首先调用 get_all_line_names 函数返回所有线路名称的列表 linenames。然后创建一个 Demand17（包含线路名称和运行时间两个属性）的列表 result 保存最终结果，对 linenames 中每

一项调用 get_start_time_in_one_run 和 get_end_time_in_one_run 函数，返回其某一班次的起始和终止时间（字符串保存），将其通过 Date 类型转换为 int，存入 result 列表中，对其进行排序。转化 JSONArray 对象输出。

在本需求中，一开始我们将两个 Query 语句写在了一起，并输出到一个 ArrayList<String> 形的容器当中，然而一直报错。经过 Debug 我们发现，Query 语句的映射也是有方向性的。倘若返回一组数据，会将一组数据的每一项都映射为 ArrayList 中的一项，而对每一项数据，考虑其为结构体或者说类的形式，Query 语句中 return 后所跟的返回值的个数，应当与 ArrayList 中每一项数据内的数据种类相匹配。

前端界面测试结果

3.14 需求 20

3.14.1 a

删除某条线路及其独占的站点

Cypher

```
@Query("""
        match
                (n:Line {name:{line_id}})
        detach delete n
        match
                (r:Run{line_id:n.name})
        detach delete r
        match
                (a:Station) -[r]-> (b:Station) where n.name in r.
                    lines and size(r.lines) = 1
        delete r
        match
                (a:Station) where not (a) -- ()
        delete a
```

```
14          return n limit 1
15          """)
16  Line delete_line(@Param("lien_id") String line_id);
```

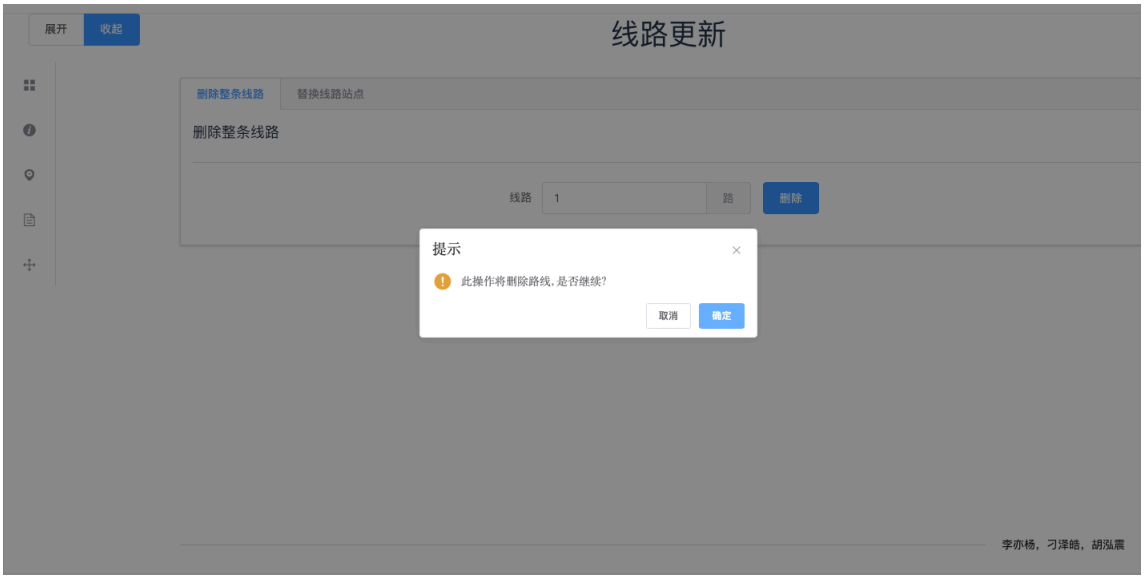
先查出要删除的线路，删除它和到它的所有 BelongTo 关系。
再查出所有该线路的班次删除。
接着查出所有 Connection 关系中 lines 里含有该线路的，从 lines 中删除，如果 lines 里没有其它线路，删除该 Connection。
最后查出所有和其它站点都没有关系的站点，删除。

业务层

```
1       public void delete_line(String lineId){
2           linerepository.delete_line(lineId);
3       }
```

直接调用 LineRepository 中的 delete_line 函数，传入线路 id 即可。

前端界面测试结果



3.14.2 b

替换一条线路中某站点，返回更改后的线路中所有站点信息

Cypher

首先根据线路名和旧站点查出对应的线路，通过 route 属性反查出旧站点对应的索引，将索引给新站点的 ID。

之后查出并删除前驱和后继站点到旧站点的 Connection 关系中 lines 属性里的对应线路，如果 lines 里因此没有元素，删除这个关系。

更新新站点到前驱和后继的关系。

如果旧站点因此不是任何关系的一端，则删除旧站点。

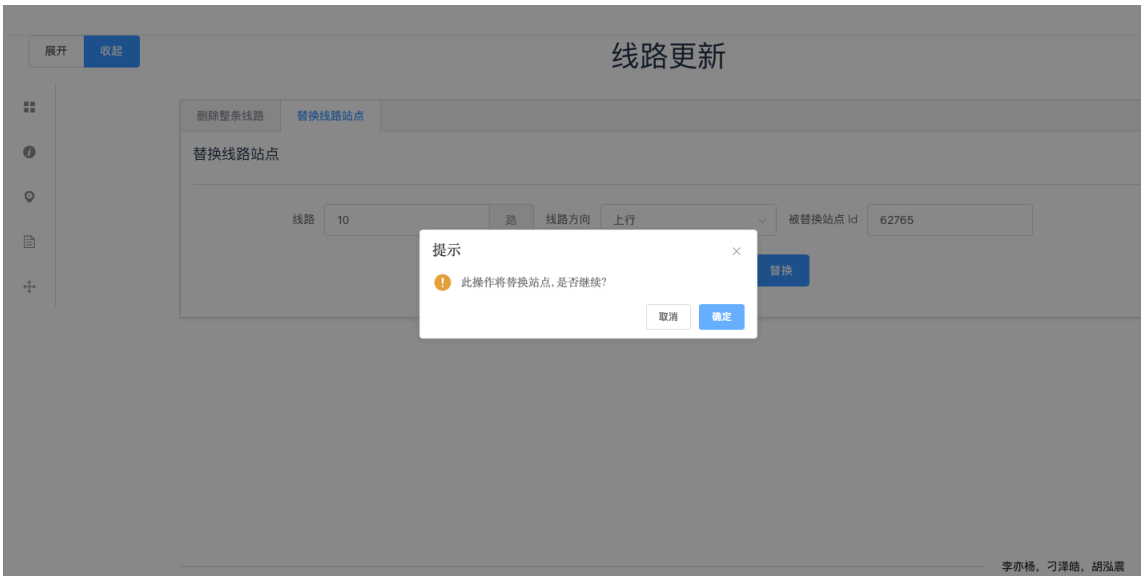
业务层

```
1       public JSONArray change_line(String lineId,String direction,
            String stationId,String newStationId){
2           linerepository.change_line(lineId, stationId, newStationId);
3           JSONArray arr = new JSONArray();
4           ArrayList<Station> station;
5           station = stationrepository.find_route_station(lineId,
                direction);
6           if(!station.isEmpty()){
7               for(int i = 0; i<station.size();i++)
8               {
```

```
9                   JSONObject obj = new JSONObject();
10                  Station s = station.get(i);
11                  obj.put("id",s.getId());
12                  obj.put("name",s.getName());
13                  obj.put("english",s.getEnglish());
14                  arr.add(obj);
15              }
16          }
17          return arr;
18      }
```

调用 LineRepository 中的 change_line 函数，传入 lineId，stationId 和 newStationId 参数。然后和需求二中完全相同，将更改后的线路站点信息返回。

前端界面测试结果



4 项目代码

由于小组成员分工较为明确，基本不会有同一行或一个文件的代码上的编辑冲突，为了方便小组成员修改后项目的版本控制，我们组建了一个 Github 仓库。考虑到项目报告并不需要提交整个项目的代码，因而在此附上我们项目的 Github 仓库链接，便于老师查阅。

项目 Github 仓库如下：

https://github.com/pikapikapikaori/NoSQL