地铁票价信息表生成系统

学 号\_\_\_18072002\_\_\_

姓 名\_\_\_邱时\_\_\_

指导教师\_\_\_杜永萍\_\_

2020年12月

[年]

目录

[1. 需求分析 2](#_Toc59797770)

[1.1 背景 2](#_Toc59797771)

[1.2 用户需求 3](#_Toc59797772)

[1.2.1 文字输入查询 3](#_Toc59797773)

[1.2.2 图形界面查询 3](#_Toc59797774)

[1.3 系统需求 3](#_Toc59797775)

[1.3.1 地铁信息数据结构 3](#_Toc59797776)

[1.3.2 最短路径查询算法 3](#_Toc59797777)

[1.3.3 TXT文件读入地铁信息 3](#_Toc59797778)

[1.3.4 前端图形界面 3](#_Toc59797779)

[1.3.5 前后端交互 4](#_Toc59797780)

[2. 数据结构设计 4](#_Toc59797781)

[2.1 Station类 4](#_Toc59797782)

[2.1.1需求和思想 4](#_Toc59797783)

[2.1.2 代码实现 4](#_Toc59797784)

[2.2 Edge类 5](#_Toc59797785)

[2.2.1需求和思想 5](#_Toc59797786)

[2.2.2 代码实现 5](#_Toc59797787)

[2.3 StationGraph类 5](#_Toc59797788)

[2.3.1需求和思想 5](#_Toc59797789)

[2.3.2 代码实现 5](#_Toc59797790)

[2.4 SubwayData类 7](#_Toc59797791)

[2.4.1需求和思想 7](#_Toc59797792)

[2.4.2 代码实现 7](#_Toc59797793)

[3. 算法设计 8](#_Toc59797794)

[3.1最短路径算法 8](#_Toc59797795)

[3.2票价计算算法 11](#_Toc59797796)

[3.3查询算法 11](#_Toc59797797)

[3.4 文件读入算法 11](#_Toc59797798)

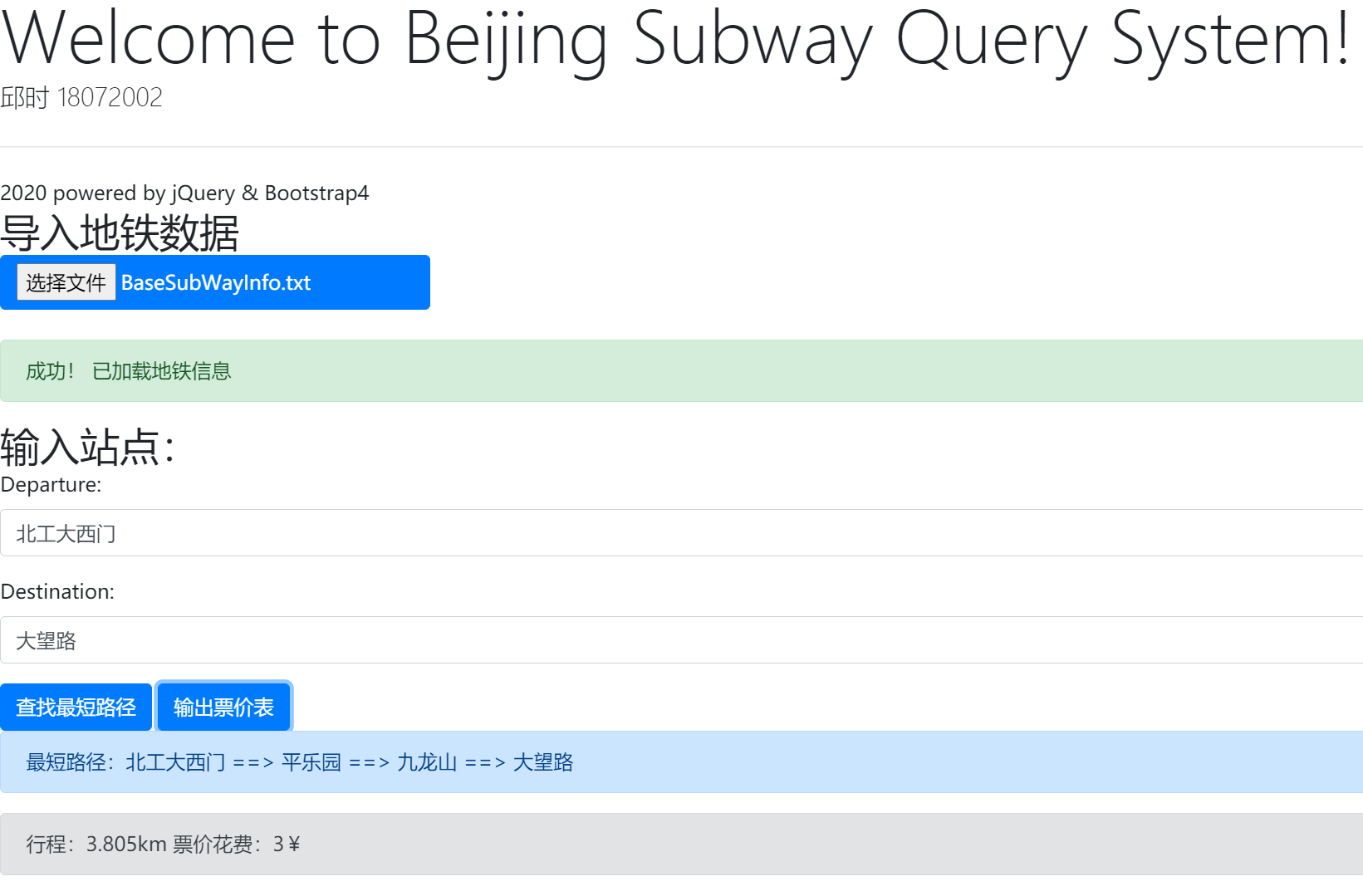
[4. 前端界面设计 12](#_Toc59797799)

[4.1 所使用的web开发工具 12](#_Toc59797800)

[4.2 核心流程 13](#_Toc59797801)

[4.3 Amap 辅助实现图形化查询 15](#_Toc59797802)

[5. 测试 16](#_Toc59797803)

[ 16](#_Toc59797804)

[6. 总结&收获 17](#_Toc59797805)

# 需求分析

## 背景

地铁已经成为大多数人出行的首选,北京地铁有多条运营线路, 截至 2019 年 12 月，北京市轨道交通路网运营线路达 23 条、总里程 699.3 公里、车站 405 座。2019 年，北京地铁年乘客量达到 45.3 亿人次，日均客流为 1241.1 万人次，单日客运量最高达 1327.46 万人次。由于采用浮动票价，乘客在乘坐地铁时需要知道出发站和目的站所需低票价以避免不必要的浪费，这就需要在乘客购票前够告知此次乘车的票价。因此，每个车站，均要提供从该站出发，到其他所有地铁站的最少票价信息表供乘客购票前查看。如果有新的地铁线路的加入，就会导致价格表的变更，因此需要使用计算机软件自动生成该票价表。

请设计一个地铁票价信息表生成软件，当输入任意起始站后，能够自动计算出以该站为起始点到其他所有各地铁站点（仅限地铁出行）的票价信息表。

## 1.2 用户需求

### 1.2.1 文字输入查询

路径查询：用户输入起始站、终点站，点击相应按钮，系统查找输出里程最短的路径。

票价表查询：用户输入起始站，点击相应按钮，系统查找并输出以相应站点为起始站，到地铁线路中每一站的票价、里程等信息。

### 1.2.2 图形界面查询

显示完整的北京地铁线路图，用户点击选取自己旅程的起点、终点站，点击相应查询按钮，系统在地铁书上突出显示最短路径。

## 1.3 系统需求

### 1.3.1 地铁信息数据结构

每个站点的名称、线路等信息；站点之间的连接、距离等关系；最短路查询算法中用到的标记信息等等，都需要被储存在易于查找、更改的数据结构中。

### 1.3.2 最短路径查询算法

最短路径问题是图论研究中的一个经典算法问题，旨在寻找图（由结点和路径组成的）中两结点之间的最短路径。

在本任务中，输入一个站时，需要能查找到整个图中其他所有站点到他的最短路径及其长度。

### 1.3.3 TXT文件读入地铁信息

北京地铁线路基础信息数据通过一个名为“BaseSubWayInfo.txt”的文本文件读入。需要有算法能够读取并切割提取该文本文档中的内容，储存在相应的地铁信息数据结构中。

### 1.3.4 前端图形界面

需要生成可通按钮、输入框等形式与用户进行交互的GUI（Graphical User Interface），通过点击按钮、上传文件、输入站点、提交表单等等方式完成用户的查询需求。

### 1.3.5 前后端交互

前端的图形界面要与后端的数据、算法进行有效、可信的交互。把输入的信息提取出来，作为参数传递给后端相应的算法，获得结果再传回前端图形界面，使得用户的操作能够输出正确的结果。这是本任务中最不好入门，也是我在上面耗费时间最长的部分。

# 数据结构设计

由于本项目的数据结构由JavaScript语言写就。

JavaScript 是 Web 的编程语言。所有现代的 HTML 页面都使用 JavaScript。

## Station类

### 2.1.1需求和思想

Name[]数组储存站名，

### 2.1.2 代码实现

1. **class** Station  {
2. constructor (name) {
3. **this**.name = name;
4. **this**.lines = [];
5. //this.location = [0];
6. **this**.toEdge = undefined;
7. //this.isTransfer = false;
8. //用于求最短路径
9. **this**.preStation = undefined;
10. **this**.isVisited = **false**;
11. **this**.distance = Number.MAX\_SAFE\_INTEGER;
12. **this**.fare = Number.MAX\_SAFE\_INTEGER;
14. }
15. FareGet() {
16. **if** (**this**.distance < 6) **return** 3;
17. **else** **if** (**this**.distance < 12) **return** 4;
18. **else** **if** (**this**.distance < 22) **return** 5;
19. **else** **if** (**this**.distance < 32) **return** 6;
20. **else** **if** (**this**.distance < 52) **return** 7;
21. **else** **if** (**this**.distance < 72) **return** 8;
22. **else** **return** 9;
23. }
25. }

## 2.2 Edge类

### 2.2.1需求和思想

### 2.2.2 代码实现

1. **class** Edge {
2. constructor(from, to, weight) {
3. **this**.nextEdge = undefined; //指向邻接表中的下一条边
4. **this**.from = from;
5. **this**.to = to;
6. **this**.weight = weight;
7. }
8. }

## 2.3 StationGraph类

### 2.3.1需求和思想

### 2.3.2 代码实现

1. **class** StationsGraph {
2. //无向图
3. constructor() {
4. //顶点
5. **this**.stationArray = [];
6. }
7. StationAdd(stationName, lineNumber) {
8. **for** (let element of **this**.stationArray) {
9. **if** (element.name == stationName) {
10. element.lines.push(lineNumber);
11. element.isTransfer = **true**;
12. **return** element;
13. }
14. };
16. let stationNew = **new** Station(stationName);
17. stationNew.lines.push(lineNumber);
18. **this**.stationArray.push(stationNew);
19. **return** stationNew;
20. };
21. StationGet(stationName) {
22. **for** (**const** iterator of **this**.stationArray) {
23. **if** (iterator.name == stationName) **return** iterator;
25. }
26. **return** undefined;
28. };
29. StationGetIndex(stationName) {
30. **for** (let i = 0; i < **this**.stationArray.length; i++) {
31. **if** (**this**.stationArray[i].name == stationName)
32. **return** i;
33. }
34. **return** -1;
35. };
36. EdgeWeightGet(from, to) {
37. let index = **this**.StationGetIndex(from);
38. //找到该顶点
39. **if** (index != -1) {
40. let stationFrom = **this**.stationArray[index];
41. let edge = stationFrom.toEdge;
42. //该顶点无边
43. **if** (edge == **null**)
44. **return** Number.MAX\_SAFE\_INTEGER;
45. //有边
46. **else** {
47. **while** ((edge === **null** || edge === **void** 0 ? **void** 0 : edge.nextEdge) != **null**) {
48. **if** (edge.from == from && edge.to == to)
49. **return** edge.weight;
50. **else**
51. edge = edge === **null** || edge === **void** 0 ? **void** 0 : edge.nextEdge;
52. }
53. }
54. //但是找不到输入的那条
55. **return** Number.MAX\_SAFE\_INTEGER;
56. }
57. //没这个点
58. **else**
59. **return** Number.MAX\_SAFE\_INTEGER;
60. };
61. EdgeAdd(from, to, weight) {
62. let index = **this**.StationGetIndex(from);
63. //找到该顶点
64. **if** (index != -1) {
65. let stationFrom = **this**.stationArray[index];
66. let edge = stationFrom.toEdge;
67. //该顶点无边
68. **if** (edge == undefined) {
69. stationFrom.toEdge = **new** Edge(from, to, weight);
70. **return** **true**;
71. }
72. //该顶点已经有边
73. **else** {
74. **while** ((edge === **null** || edge === **void** 0 ? **void** 0 : edge.nextEdge) != undefined) {
75. edge = edge === **null** || edge === **void** 0 ? **void** 0 : edge.nextEdge;
76. }
77. edge.nextEdge = **new** Edge(from, to, weight);
78. **return** **true**;
79. }
80. }
81. //找不到该顶点
82. **else**
83. **return** **false**;
84. };
86. Dijkstra(from) { //最短路径算法，暂且不表
87. }

90. };

## 2.4 SubwayData类

### 2.4.1需求和思想

### 2.4.2 代码实现

1. **class** SubwayData  {
3. constructor (){
4. **this**.graph = **new** StationsGraph();
5. **this**.allLines = [];
6. **this**.lineInfo = [];
7. };
8. ReadLines(reader){
9. **var** subwayInfoArray = reader.result.split("\n");
11. **var** lineAmount = subwayInfoArray[0];
12. //var subwayInfoArray = subwayInfoArray.slice(1).toString();
14. //切割并读入每一条线
15. **for** (let i = 1; i <= lineAmount; i++) {
16. //读入lineInfo
17. let lineInfoArray = subwayInfoArray[i].toString().split(",");
18. **this**.lineInfo.push([lineInfoArray[0], lineInfoArray[1]], lineInfoArray[2]); //编号 线路名称 线站点数
19. //读入Stations
20. **for** (let j = 3; j < lineInfoArray.length; j += 2) {
21. let stationNew = **this**.graph.StationAdd(lineInfoArray[j], lineInfoArray[1]);//自带检测
22. **if**(j > 3){  //若当前添加的站点不是最后一站，建立与上一站的双向边
23. **this**.graph.EdgeAdd(lineInfoArray[j-2], lineInfoArray[j], lineInfoArray[j-1]);
24. **this**.graph.EdgeAdd(lineInfoArray[j], lineInfoArray[j-2], lineInfoArray[j-1]);
25. **debugger**;
26. }
28. }
29. }
30. }
31. }

# 算法设计

## 3.1最短路径算法

1. Dijkstra(from) {
3. //init
4. **this**.stationArray.forEach((element) => {
5. **if** (element.name == from) {
6. element.distance = 0;
7. }
8. **else** {
9. element.distance = Number.MAX\_SAFE\_INTEGER;
11. }
12. element.isVisited = **false**;
13. element.preStation = undefined;
15. });
16. let curSrc = **this**.StationGet(from);
17. //理论上可以更新所有, 只要有未被访问的点，就不停下循环
19. **do** {
20. **if** (curSrc.isVisited == **true**) **continue**;
21. curSrc.isVisited = **true**;
22. let edge = curSrc.toEdge;
24. **while** (edge != undefined) {
25. let curDest = **this**.StationGet(edge.to);
26. **if** ((curDest.isVisited) == **false** && curDest.distance > curSrc.distance + edge.weight) {
27. //更新最短距离
28. curDest.distance = (curSrc.distance + edge.weight).toFixed(3); //保留三位小数
29. curDest.distance = Number(curDest.distance);
30. curDest.preStation = curSrc.name;
32. }
33. edge = edge.nextEdge;
34. }
35. //更新cursrc,找到distance最小，且isvisited = false的最近邻接点
36. **var** nextSrc = **new** Station("default");   //default 标志结束循环
37. nextSrc.distance = Number.MAX\_SAFE\_INTEGER;
38. **for** (**const** iter of **this**.stationArray) {
39. **if** (iter.isVisited == **false**)
40. nextSrc = nextSrc.distance > iter.distance ? iter : nextSrc;
41. }
43. //更新当前遍历的节点
44. curSrc.fare = curSrc.FareGet();
45. curSrc = nextSrc;
47. }
48. **while** (curSrc.name != "default");//default 标志结束循环
49. }
50. Dijkstra(from) {
52. //init
53. **this**.stationArray.forEach((element) => {
54. **if** (element.name == from) {
55. element.distance = 0;
56. }
57. **else** {
58. element.distance = Number.MAX\_SAFE\_INTEGER;
60. }
61. element.isVisited = **false**;
62. element.preStation = undefined;
64. });
65. let curSrc = **this**.StationGet(from);
66. //理论上可以更新所有, 只要有未被访问的点，就不停下循环
68. **do** {
69. **if** (curSrc.isVisited == **true**) **continue**;
70. curSrc.isVisited = **true**;
71. let edge = curSrc.toEdge;
73. **while** (edge != undefined) {
74. let curDest = **this**.StationGet(edge.to);
75. **if** ((curDest.isVisited) == **false** && curDest.distance > curSrc.distance + edge.weight) {
76. //更新最短距离
77. curDest.distance = (curSrc.distance + edge.weight).toFixed(3); //保留三位小数
78. curDest.distance = Number(curDest.distance);
79. curDest.preStation = curSrc.name;
81. }
82. edge = edge.nextEdge;
83. }
84. //更新cursrc,找到distance最小，且isvisited = false的最近邻接点
85. **var** nextSrc = **new** Station("default");   //default 标志结束循环
86. nextSrc.distance = Number.MAX\_SAFE\_INTEGER;
87. **for** (**const** iter of **this**.stationArray) {
88. **if** (iter.isVisited == **false**)
89. nextSrc = nextSrc.distance > iter.distance ? iter : nextSrc;
90. }
92. //更新当前遍历的节点
93. curSrc.fare = curSrc.FareGet();
94. curSrc = nextSrc;
96. }
97. **while** (curSrc.name != "default");//default 标志结束循环
98. }

## 3.2票价计算算法

1. FareGet() {
2. **if** (**this**.distance < 6) **return** 3;
3. **else** **if** (**this**.distance < 12) **return** 4;
4. **else** **if** (**this**.distance < 22) **return** 5;
5. **else** **if** (**this**.distance < 32) **return** 6;
6. **else** **if** (**this**.distance < 52) **return** 7;
7. **else** **if** (**this**.distance < 72) **return** 8;
8. **else** **return** 9;
9. }

## 3.3查询算法

1. Query(from, to) {
3. let pathArray = **new** Array();
4. let curStation = **this**.graph.StationGet(to);
5. **this**.graph.Dijkstra(from);
6. **do** {
7. pathArray.push(curStation?.name);
8. curStation = **this**.graph.StationGet(curStation.preStation);
9. //debugger;
10. }
11. **while** (curStation.preStation != undefined);
12. pathArray.push(curStation.name); //加入始发站
14. **return** pathArray.reverse();
15. }

## 3.4 文件读入算法

1. ReadLines(reader) {
2. **var** subwayInfoArray = reader.result.split("\n");
3. **var** lineAmount = subwayInfoArray[0];
4. //var subwayInfoArray = subwayInfoArray.slice(1).toString();
6. //切割并读入每一条线
7. **for** (let i = 1; i <= lineAmount; i++) {
8. //读入lineInfo
9. let lineInfoArray = subwayInfoArray[i].toString().split(",");
10. **this**.lineInfo.push([lineInfoArray[0], lineInfoArray[1]], lineInfoArray[2]); //编号 线路名称 线站点数
11. //读入Stations
12. **for** (let j = 3; j < lineInfoArray.length; j += 2) {
13. let stationNew = **this**.graph.StationAdd(lineInfoArray[j], lineInfoArray[1]);//自带检测
14. **if** (j > 3) {  //若当前添加的站点不是最后一站，建立与上一站的双向边
15. **this**.graph.EdgeAdd(lineInfoArray[j - 2], lineInfoArray[j], lineInfoArray[j - 1]);
16. **this**.graph.EdgeAdd(lineInfoArray[j], lineInfoArray[j - 2], lineInfoArray[j - 1]);
17. **debugger**;
18. }
20. }
21. }
22. }

# 前端界面设计

## 所使用的web开发工具

HTML + jQuery + Bootstrap 4 实现主页面和文字输入查询超文本标记语言（英语：HyperText Markup Language，简称：HTML）是一种用于创建网页的标准标记语言。可以使用 HTML 来建立 WEB 站点，HTML 运行在浏览器上，由浏览器来解析。

jQuery是一个快速，小型且功能丰富的JavaScript库。 通过易于使用的API（可在多种浏览器中使用），它使HTML文档的遍历和操作，事件处理，动画和Ajax等事情变得更加简单。 兼具多功能性和可扩展性，jQuery改变了数百万人编写JavaScript的方式。

Bootstrap 是一个用于快速开发 Web 应用程序和网站的前端框架。Bootstrap 是基于 HTML、CSS、JAVASCRIPT 的。Bootstrap 是由 *Twitter* 的 *Mark Otto* 和 *Jacob Thornton* 开发的。Bootstrap 是 2011 年八月在 GitHub 上发布的开源产品

## 核心流程

1. <script type="text/javascript">
2. **class** SubwayData {
3. }
5. //初始化总数据结构
6. let subwayData = **new** SubwayData();
7. let pathsData = **new** Array();
8. let tableColumns = [
10. {field: 'id', title: '序号', sortable: **true**},
11. {field: 'name', title: '目的地', sortable: **true**},
12. {field: 'distance', title: '最短距离', sortable: **true**},
13. {field: 'fare', title: '最低票价', sortable: **true**},
14. ];
15. $('#table1').bootstrapTable({//表格初始化
16. columns: tableColumns,  //表头
17. data: undefined, //通过ajax返回的数据
18. width:300,
19. height:268,
20. method: 'get',
21. pageSize: 3,
22. pageNumber: 1,
23. pageList: [],
24. cache: **false**,
25. striped: **true**,
26. pagination: **true**,
27. sidePagination: 'client',
28. search: **false**,
29. showRefresh: **false**,
30. showExport: **false**,
31. showFooter: **true**,
32. // exportTypes: ['csv', 'txt', 'xml'],
33. clickToSelect: **false**,
34. });
35. //读取txt
36. **var** openFile = () => {
37. **var** input = event.target;
38. **var** reader = **new** FileReader();
40. reader.onload = () => {
41. **if** (reader.result) {
42. //读入数据txt，数据都存在subwaydata
43. subwayData.ReadLines(reader);
44. //subwayData.Query()
45. }
46. $("#success").show(100);
47. //$("#output").html(reader.result);
48. }
49. reader.readAsText(input.files[0]);
50. };
51. //query
52. $("#submit").click(() => {
54. **var** departure = $("#departure").val();
55. **var** destination = $("#destination").val();
56. **var** pathArray = subwayData.Query(departure, destination);
57. **var** destStation= subwayData.graph.StationGet(destination);
59. $("#dijpath").html(() => {
60. **return** pathArray.join(" ==> ");
61. });
62. $("#path").show(100);
64. $("#dist").html(destStation.distance + "km");
65. $("#fee").html(destStation.fare + "￥");
66. $("#fare").show(100);

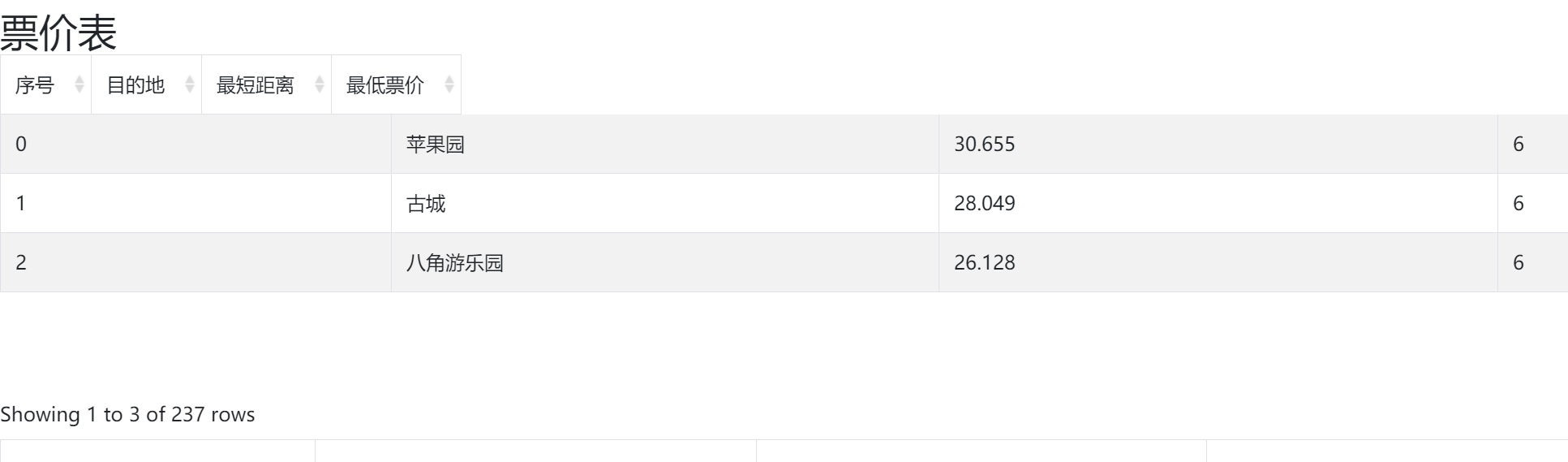
69. let index = 0;
70. **for**(**const** iter of subwayData.graph.stationArray)
71. {
72. **if**(iter.name == departure) **continue**;
73. let data = {}
74. data["id"] = index++;
75. data["name"] = iter.name;
76. data["distance"] = iter.distance;
77. data["fare"] = iter.fare;
78. pathsData.push(data);
79. }
80. $('#table1').bootstrapTable("load", pathsData);
82. pathsData = undefined;
84. });
85. $("#faresheet").click(
86. () => {$('#faretable').show(100);}
87. );
89. </script>

## Amap 辅助实现图形化查询

1. html**>**
2. **<head>**
3. **<meta** charset="UTF-8"**>**
4. <!--重要meta, 必须!-->
5. **<meta** name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0,shrink-to-fit=no"**/>**
6. **<title>**SUBWAY**</title>**
7. **</head>**
8. **<body>**
9. **<div** id="mysubway"**></div>**
11. **<script** src="https://webapi.amap.com/subway?v=1.0&key=5f9f571c1e5291573744a37de7128d46&callback=cbk"**></script>**
12. **<script** type="text/javascript"**>**
13. window.cbk = function(){
14. var mysubway = subway("mysubway", {
15. easy: 1
16. });
17. };
18. **</script>**
19. **</body>**
20. **</html>**

# 测试

# 



# 总结&收获

通过本次课设活动，我真正第一次从头到尾整了一个可以用的前端项目。虽然并没有用到什么特别复杂的算法或者框架，但是收获的工程经历、经验是非常宝贵的。完成项目的过程中精力的无数心酸、崩溃、难过并没能压倒我，相反做成了这一次项目之后，我心就有底气了。回过头来看，当时感觉根本不可能解决的那些问题都一步一步迈过去了，根本没有必要惊慌，没有必要自乱阵脚。天塌不下来！好好干活，会好起来的！