软件技术基础综合课程设计

电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

实验报告

实 验 名 称 互联网+智慧物流质询系统

第 五 组 成 员 及 贡 献 表

姓名	学号	贡献占比
李正阳	2020080301009	20%
张力群	2020080301001	20%
周琨瑜	2020080301010	20%
齐秋逸	2020080301003	20%
刘振强	2020150501026	20%

互联网+智慧物流质询系统

软件技术基础综合课程设计.....1
一、 实验室名称：主楼ｂ１－３０２.....3
二、 实验项目名称：＂互联网＋＂智慧物流质询系统.....3
三、 实验学时：３２.....3
四、 实验原理：.....3
五、 实验目的：.....4
5.1 实验一.....4
5.2 实验二.....4
5.3 实验三.....5
5.4 实验四.....5
六、 实验内容.....5
6.1 实验一：最优物流路线计算实验.....5
6.2 实验二：多进程多用户文件一致性读写访问设计实现.....6
6.3 实验三：SQL 解析器设计实现.....7
6.4 实验四：互联网+智慧物流质询系统设计实现.....9
七、 实验器材（设备，元器件）：.....12
八、 结果与展示.....12
8.1 基本界面及功能介绍.....12
8.2 SQL 语句解析器功能演示.....21
九、 数据结构与程序.....28
9.1 树形结构存储数据.....28
9.2 dijkstra 算法计算最短路径.....30
9.3 实现相应的接口从数据库中读取数据.....33
9.4 对 excel 表格进行预处理.....34
9.5 读取 excel 数据.....36
9.6 excel 操作整合 接口对接.....37
9.7 对 excel 数据表进行保存.....43
9.8 产生多进程访问和互斥操作.....44
9.9 前后端交互接口（增删改查操作）.....46
9.10 SQL 解析器.....55
9.11 网页整体样式.....69
十、 具体贡献.....75
十一、 源代码地址：.....76

一、 实验室名称：主楼 b1-302

二、 实验项目名称：＂互联网＋＂智慧物流质询系统

三、 实验学时：32

四、 实验原理：

4.1 实验一

设计物品信息，通过物品信息计算物流方式以及发货顺序，然后用树这种数据结构存储这些物品信息，用邻接矩阵存储路线信息，并利用 dijstra 算法求最短路。

4.2 实验二

设计 excel 文件系统，我们对于物流物品及不同路径的信息使用上述 excel 表格进行存储，对 excel 进行相应的 io 操作，为了方便展示和处理，每个表格我们只初始化了少量数据，数据具有拓展性，同时为了方便处理，我们对文件系统实现的数据库实现了接口操作，为了直接对接实验一，这部分我们直接在实验一中实现。

设置读、写信号量，利用信号量使得进程间互斥地对文件进行访问，并实现同时写与同时读写的互斥操作，保证文件中数据的一致性。

4.3 实验三

SQL 语句解析器由三部分组成，分别是词法分析器，语法分析器以及词法分析器。

词法分析器是用于识别单词所构筑的一个自动识别程序。其本质是词法分析，由一正规文法或是正规表达式推导出不确定的有穷自动机 NFA，再确定化得到确定的有穷自动机 DFA，最后最小化得到一个最简 DFA，输入符号串，识别单词。

语法分析器是在词法分析器的基础上实现识别一符号串是否符合相关文法，其本质是语法分析，生成语法树而语法分析采用的自底向上，自底而上方法采用的是算符优先分析。

而语义分析就是在语法分析的基础上完成：收集标识符的属性信息 和 对语义的检查（检查合法性）

4.4 实验四

使用 vue 框架和 node.js 以及使用文件系统编写的简易数据库进行前后端的集成。

其中后端的功能主要在 node.js 程序中实现，包括物品信息计算、发货顺序、dijstra 算法求最短路等物流操作，多进程读写访问的实现以及 SQL 语句解释器的功能实现。

其中前端主要通过 vue 框架实现，管理员通过网页进行相关操作。

数据全部储存在用文件系统编写的简易数据库中。

前后端的交互通过 axios 库的 get 和 post 方法实现。

五、 实验目的：

5.1 实验一

(1) 根据物品信息综合计算物流物品的优先级别，根据物流优先级别排序物流物品，根据排序结果对物流物品进行逐个发货。

(2) 根据物流物品的物流条件信息，归类物流物品到物流方案类型，物流方案类型可包括：价格最小物流方案，时间最短物流方案、综合最优方案、航空物流方案等。并运用树型结构存储所有的物流物品到划分的物流方案中。

(3) 根据给定的物流节点信息，计算各类物流方案下的物流最短路径 。

(4) 根据物流最短路径，物流方案和物流优先级发送货物。

掌握数据结构的线性数据结构、树数据结构、图数据结构的运用。令物品为一个类，物流节点图为另一个类，通过在两个类直接定义结点信息、函数实现基本内容。

5.2 实验二

(1) 设计实现数据表的文件存储方式，能在文件中存储多张数据表，每张数据表可存储多条记录。实现指定表中记录的存储、读写和记录的简单查询与索引查询函数。能够实现单用户和进程对文件数据中记录的写入与查询。

(2) 实现多进程对单个文件中某表中的记录的互斥写入与查询访问操作，保证表中记录数据的一致性。

(3) 实现多用户对文件中记录数据的同时写入与查询一致性操作。

利用操作系统中进程并行，互斥和生产消费者问题实现对文件的数据写入和查询访问。

设置读、写信号量。若有进程读取，则申请一个读信号量；若有进程写入，则申请一个写信号量。只有当读写信号量同时可申请时才可以写入，只有当写信号量可申请时才可以读取。进程结束时释放信号量。

5.3 实验三

(1) 构建语法解析器实现部分 SQL 语句，包括 Select 语句，Insert 语句和 Update，创建表语句的语法解析。

(2) 构建语义解析器对 SQL 语句进行语义解析

(3) 将解析的语义对接底层实验 2 中实现的各个数据操作函数

实现部分 SQL 语句，包括 Select 语句，Insert 语句和 Update，创建表语句的解析并对接实验 2 中对应的创建表、写入和查询函数实现数据表创建、写入和查询操作。

5.4 实验四

(1) 结合实验 1,2 构建物流节点信息表，实现物流节点信息的数据库存储与多进程多用户底层操作

(2) 结合实验 1,2，构建物品信息表，实现物品信息的存储与多进程多用户底层操作

(3) 结合实验 3 以 SQL 语句，对物流节点信息进行增删改查

(4) 结合实验 3 以 SQL 语句，对物品信息进行增删改查

(5) 结合实验 1，实现物品的优先级排序和物流方案分类（可采用多种分类方法不一定用树结构存储）

(6) 节点信息会动态变化，因此结合实验 1，每个物品需要动态计算物流最短路径的实现。

(7) 物品的物流状态，用户可以对物件的物流状态进行查询。

六、 实验内容

6.1 实验一：最优物流路线计算实验

首先设计物品信息，我们将选择物流方法的权利交到用户手上，我们设计了四种物品类型，1 代表最短时间消耗方案，2 代表最小价格消耗方案，3 代表最小综合价格消耗以及 4 代表航空直达方案，其中综合价格消耗由时间消耗和价格消耗加权而得。此外还设计了目的地，到达时间，vip,route,发货等表项，其中到达时间我们设计为一个 12 位的数据格式（类似于 200206231003），vip 的值决定了客户是否优先发货，route 为采用 dijkstra 算法计算得出的路径，发货表示对应的货物是否已成功发货，

其次设计物流点信息，我们把物流点看作一个节点并用 Nodes 表进行存储。

最后设计物流路径信息，我们用 edges 表存储物流点之间的边，其中每条边为无向边，每条边的权值有三种分别是时间消耗，价格消耗和综合消耗。

设计 excel 文件系统，我们对于物流物品及不同路径的信息使用上述 excel

表格进行存储，对 excel 进行相应的 io 操作，为了方便展示和处理，每个表格我们只初始化了少量数据，数据具有拓展性，同时为了方便处理，我们对文件系统实现的数据库实现了接口操作。

下面是设计的相应数据

	A	B	C	D	E	F	G
1	id	type	vip	destination	arrival_time	logistics_routes	deliver
2	10001	1	0	西安	202202230020	成都->广州->西安	1
3	11200	1	0	郑州	202206130630	成都->深圳->郑州	1
4	23564	2	1	广州	202201220530	成都->广州	1
5	57468	3	1	上海	202202231830	成都->上海	0
6	99999	4	1	郑州	202205090030	成都->郑州	0

Products 表

	A	B
1	id	name
2	1	成都
3	10	西安
4	11	长沙
5	12	郑州
6	2	北京
7	3	上海
8	4	广州
9	5	深圳
10	6	南京
11	7	武汉
12	8	杭州
13	9	重庆

Nodes 表

	A	B	C	D	E	F
1	id	source	target	time	price	synthesis
2	1	1	2	50	50	25
3	2	1	3	50	16	33
4	3	1	4	30	5	18
5	4	1	5	9	33	21
6	5	2	6	29	9	19
7	6	2	7	28	13	21
8	7	3	8	11	42	27
9	8	3	9	24	37	31
10	9	4	10	1	34	18
11	10	4	11	32	46	39
12	11	5	12	39	10	25
13	12	5	6	41	46	44
14	13	6	7	43	27	35
15	14	7	8	5	2	4
16	15	8	9	47	14	31
17	16	9	10	48	12	30
18	17	1	12	50	50	50

Edges 表

具体代码见第九部分，演示见第八部分。

6.2 实验二：多进程多用户文件一致性读写访问设计实现

1）读写锁包括读取锁和写入锁，多个读线程可以同时访问共享数据；

2）写线程必须等待所有读线程都释放锁以后，才能取得锁；

3）同样的，读线程必须等待写线程释放锁后，才能取得锁；

4）也就是说读写锁要确保的是如下互斥关系：可以同时读，但是读-写，写-写都是互斥的；

读写锁的分配规则：

1.只要没有线程持有某个给定的读写锁用于写，那么任意数目的线程可以持有该读写锁写锁用于读。

2.仅当没有线程持有某个给定的的读写锁用于读或者用于写时，才能分配该读写锁用于写。

通俗点说就是当没有写锁时，就可以加读锁且任意线程可以同时加，而写锁只能有一个，且必须在没有读锁时才能加上，一般来说，写锁优先。

单，多进程的增删改查操作的接口已经在实验一的部分完成，具体结果见第八部分，具体代码见第九部分。

6.3 实验三：SQL 解析器设计实现

6.3.1 词法分析器实现

词法分析器是用于识别单词所构筑的一个自动识别程序。其本质是词法分析，由一正规文法或是正规表达式推导出不确定的有穷自动机 NFA，再确定化得到确定的有穷自动机 DFA，最后最小化得到一个最简 DFA，输入符号串，识别单词。

为了实现语法分析器，我们首先设定了关键词和它所对应的 syn 标号，具体对应关系如下表。

1	select
2	from
3	insert
4	into
5	Values
6	update
7	set
8	where
9	create
10	table
11	=
12	,
13	;
14	(
15	)
16	\’(单引号)
17	*
18	int
19	varchar

20	id
21	num
22	delete

6.3.2 语法分析器实现

语法分析器是在词法分析器的基础上实现识别一符号串是否符合相关文法，其本质是语法分析，生成语法树而语法分析采用的自底向上，自底而上方法采用的是算符优先分析。

算符优先分析步骤：

步骤一：为每个非终结符 A 计算 FIRSTVT(A) 和 LASTVT(A)

（1）Firstvt 集合

找 Firstvt 的三条规则：如果要找 A 的 Firstvt，A 的候选式中出现：

A->a…，即以终结符开头，该终结符入 Firstvt

A->B…，即以非终结符开头，该非终结符的 Firstvt 入 A 的 Firstvt

A->Ba…，即先以非终结符开头，紧跟终结符，则终结符入 Firstvt

（2）Lastvt 集合

找 Lastvt 的三条规则：如果要找 A 的 Lastvt，A 的候选式中出现：

A->…a，即以终结符结尾，该终结符入 Lastvt

A->…B，即以非终结符结尾，该非终结符的 Lastvt 入 A 的 Lastvt

A->…aB，即先以非终结符结尾，前面是终结符，则终结符入 Lastvt

步骤二：逐条扫描文法规则

（1）因存在 E → （E）的规则，则有=

（2）寻找终结符在左边，非终结符在右边的符号对

+T 则 +< FIRSTVT(T)

*F 则 *< FIRSTVT(F)

(E 则 (< FIRSTVT(E)

（3）寻找非终结符在左边，终结符在右边的符号对

E+ 则 LASTVT(E)>+

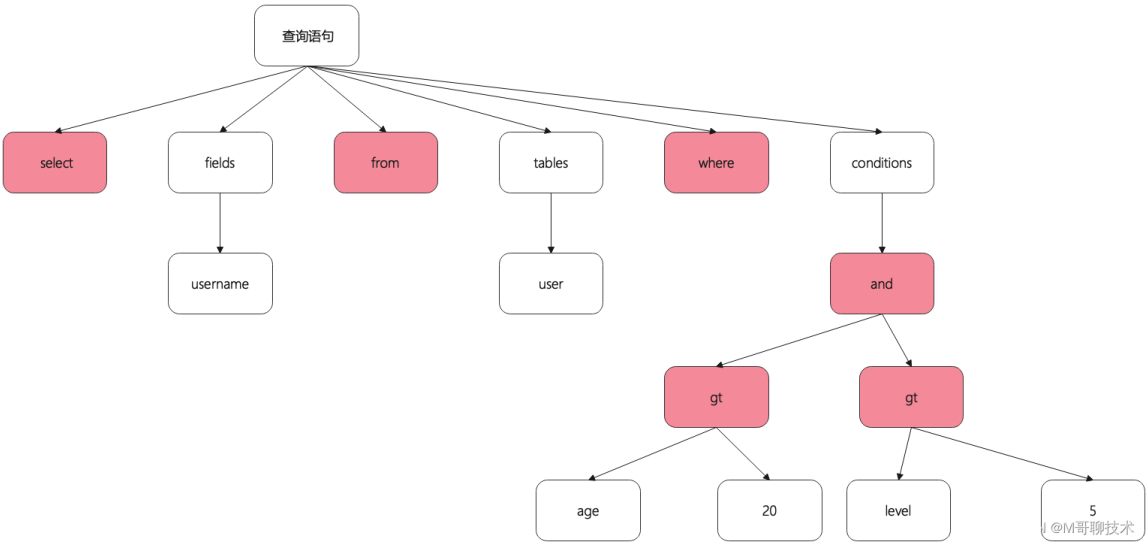
T* 则 LASTVT(T)>*

E) 则 LASTVT(E)>)

步骤三：寻找$与开始符号 E 的关系

（1）$=$

（2）$<FIRSTVT(E)且 LASTVT(E)>$



6.3.3 语义分析器实现

而语义分析就是在语法分析的基础上完成：收集标识符的属性信息 和 对语义的检查（检查合法性）。

为了实现语义分析器我们采用了 AST 树的技术，AST 是 Abstract Syntax Tree 的缩写，也就是抽象语法树。AST 是 parser 输出的结果。这也是语法树的精髓了，sql 解析，本质上就是把 sql 转为 ast 语法树，拿到这个语法树后，我们就能做很多事了，遍历也好，加点，修改也好，都可以在 ast 上完成。

因为语法分析树已经在前面的操作成功建立，因此较好实现，至此 SQL 解析器的原理介绍完毕,具体代码见第九部分，演示见第八部分。

6.4 实验四：互联网+智慧物流质询系统设计实现

6.4.1 系统设计思路及方法

为了将前三个实验集成起来，实现智慧物流质询系统。我们决定设计一个前后端交互的网页。

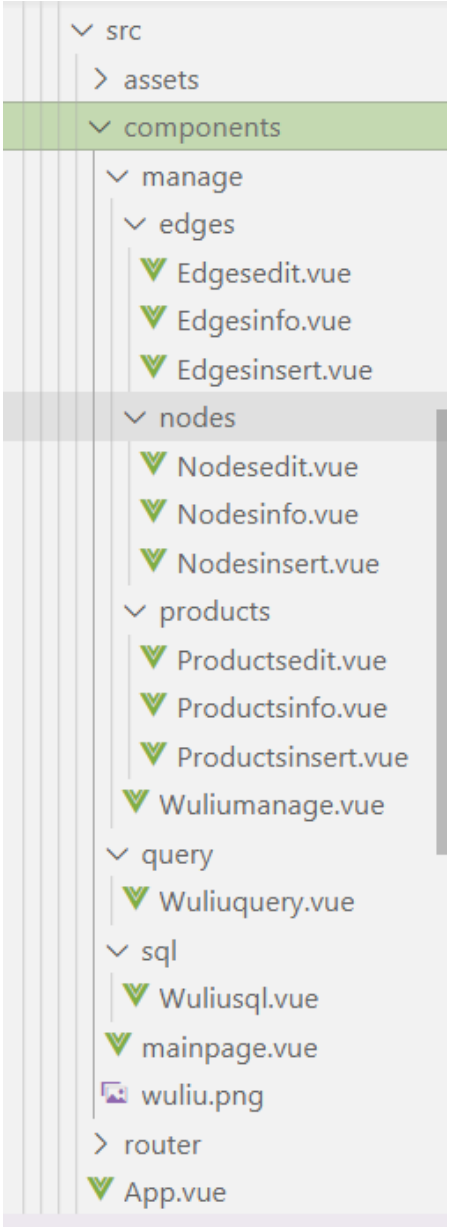
物流系统的管理员在网页端操作，全部计算在后端实现，全部数据存储在用文件系统编写的简易数据库中。其中，网页的前端通过 vue 框架实现，通过编写不同的页面并引入后端的接口，便可将前三个实验的功能集成在前端网页上。

网页的前后端交互通过 axios 库的 get 和 post 方法实现。在后端编写相应的接口，在前端使用，完成前后端的交互以及数据的传输。

我们的智慧物流系统以网页的形式实现，包括 vue 前端、node.js 后端和用文件系统编写的简易数据库。

6.4.2 系统前端页面设计

前端各部分的实现通过 vue 项目框架中不同模块实现；

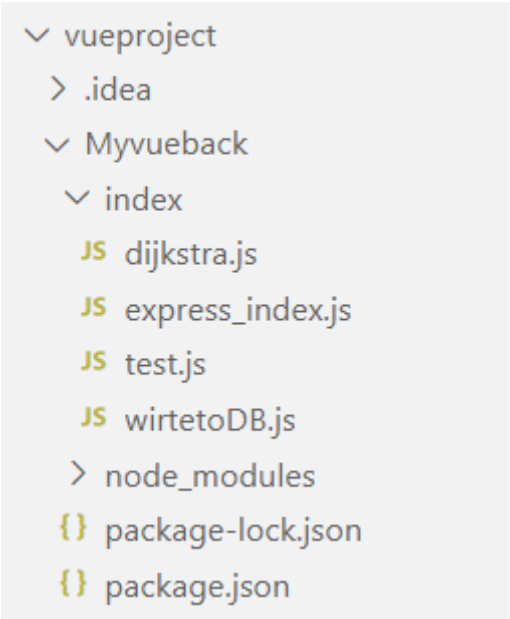


在 manage 文件夹中实现管理功能，其中 edges 是对路径操作，nodes 对节点操作，products 对物品信息操作，在 query 文件夹中实现查询功能，在 sql 文件夹中实现 SQL 语句解释器。在 app.vue 中编写网页端的整体样式。

具体代码见第九部分，演示见第八部分。

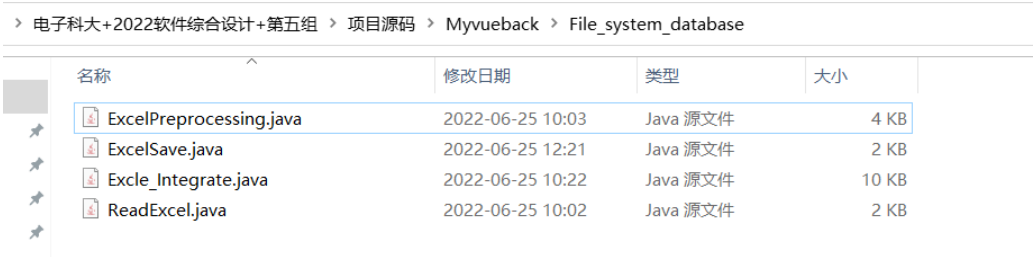
6.4.3 后端实现

后端通过 index 文件夹中的 js 文件实现。



其中 express_index.js 中实现所有前后端交互的接口以及多进程访问，dijkstra.js 中实现计算最短路径的功能，wirtetoDB.js 实现数据的写入。

文件系统由 File_system_database 的函数实现。



6.4.4 简易数据库页面

通过 wuliu 数据库存储数据，由三个表构成，分别为 products（物品信息）、edges（路径信息）、nodes（节点信息）。对应的数据格式如下：

Products：

	A	B	C	D	E	F	G
1	id	type	vip	destination	arrival_time	logistics_routes	deliver
2	10001	1	0	西安	202202230020	成都->广州->西安	1
3	11200	1	0	郑州	202206130630	成都->深圳->郑州	1
4	23564	2	1	广州	202201220530	成都->广州	1
5	57468	3	1	上海	202202231830	成都->上海	0
6	99999	4	1	郑州	202205090030	成都->郑州	0
7							

Edges:

	A	B	C	D	E	F
1	id	source	target	time	price	synthesis
2	1	1	2	50	50	25
3	2	1	3	50	16	33
4	3	1	4	30	5	18
5	4	1	5	9	33	21
6	5	2	6	29	9	19
7	6	2	7	28	13	21
8	7	3	8	11	42	27
9	8	3	9	24	37	31
10	9	4	10	1	34	18
11	10	4	11	32	46	39
12	11	5	12	39	10	25
13	12	5	6	41	46	44
14	13	6	7	43	27	35
15	14	7	8	5	2	4
16	15	8	9	47	14	31
17	16	9	10	48	12	30
18	17	1	12	50	50	50

Nodes:

	A	B
1	id	name
2	1	成都
3	10	西安
4	11	长沙
5	12	郑州
6	2	北京
7	3	上海
8	4	广州
9	5	深圳
10	6	南京
11	7	武汉
12	8	杭州
13	9	重庆

七、 实验器材（设备，元器件）：

硬件和设备：笔记本电脑。

软件和环境：Windows 操作系统，VScode、node.js、用 excel 文件系统编写的简易数据库、Vue 框架。

八、 结果与展示

8.1 基本界面及功能介绍

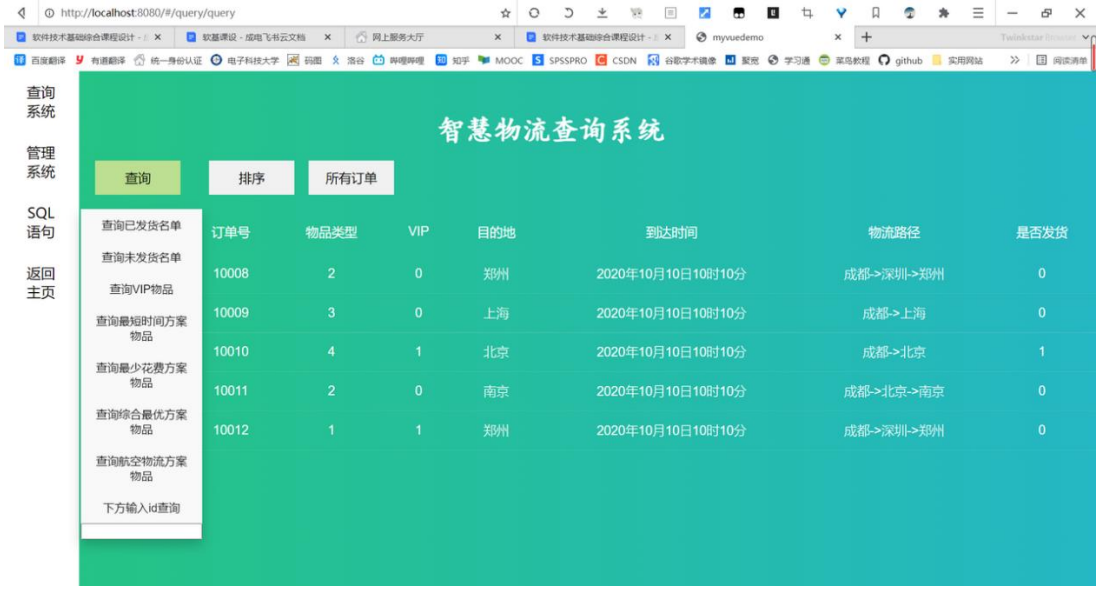
由于基本页面的功能太多，且图片不能反映数据更改后的动态变化，部分功能在附件中的演示视频中有更好的体现。

8.1.1 主页

网站的主页导航栏有三个功能，分别是查询、管理和 SQL 语句解析器。

8.1.2 查询系统

查询系统对应有查询和排序的下拉菜单，可进行查询和排序。

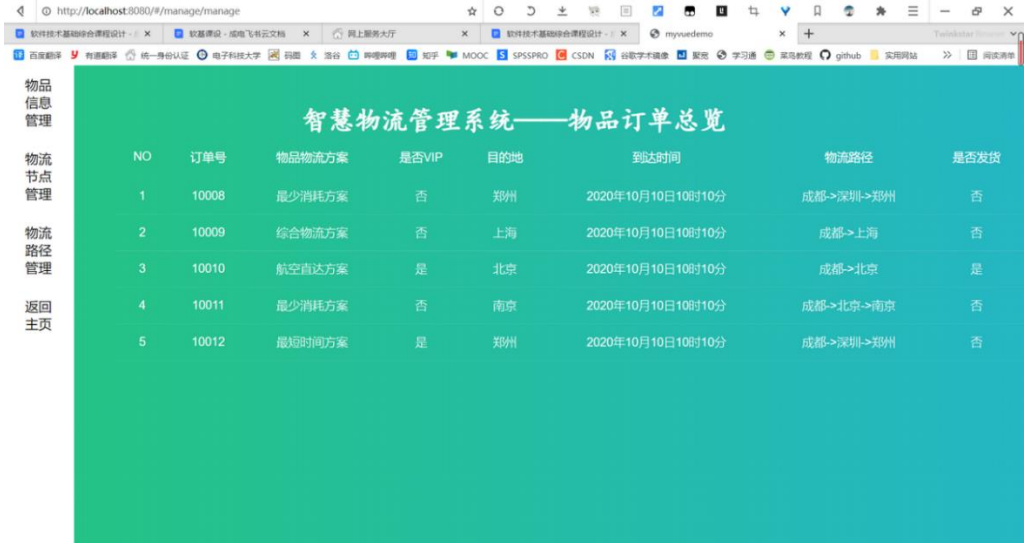


以查询最短时间方案为例：

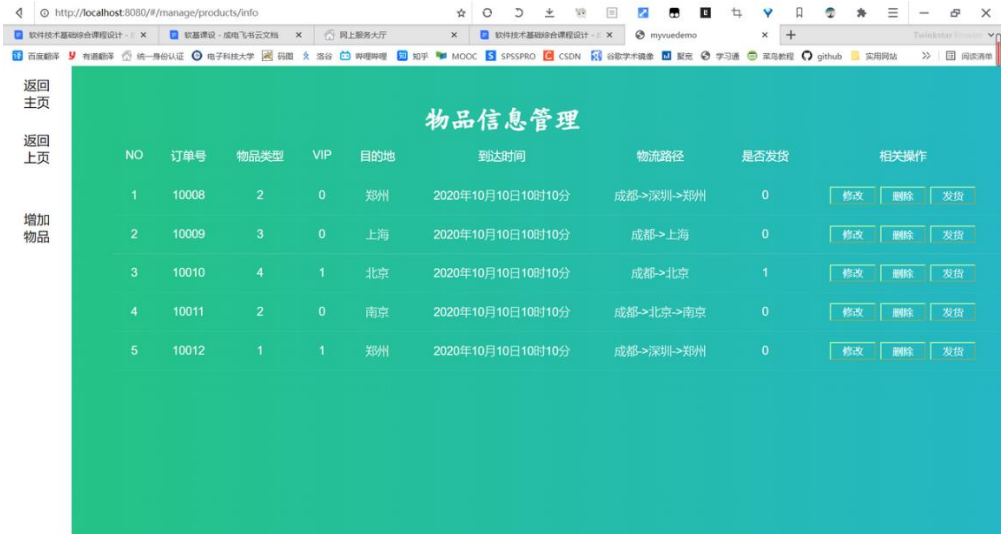
按照订单号升序排序：

8.1.3 管理系统

管理系统分为物品信息管理、物流节点管理、物流路径管理三个子页面。



物品信息管理



物品信息管理系统有发货，增加物品、修改以及删除的功能。

对 10008 号订单发货

物品信息管理

NO	订单号	物品类型	VIP	目的地	到达时间	物流路径	是否发货	相关操作
1	10008	2	0	郑州	2020年10月10日10时10分	成都->深圳->郑州	1	修改 删除 发货
2	10009	3	0	上海	2020年10月10日10时10分	成都->上海	0	修改 删除 发货
3	10010	4	1	北京	2020年10月10日10时10分	成都->北京	1	修改 删除 发货
4	10011	2	0	南京	2020年10月10日10时10分	成都->北京->南京	0	修改 删除 发货
5	10012	1	1	郑州	2020年10月10日10时10分	成都->深圳->郑州	0	修改 删除 发货

删除 10008 号

物品信息管理

NO	订单号	物品类型	VIP	目的地	到达时间	物流路径	是否发货	相关操作
1	10009	3	0	上海	2020年10月10日10时10分	成都->上海	0	修改 删除 发货
2	10010	4	1	北京	2020年10月10日10时10分	成都->北京	1	修改 删除 发货
3	10011	2	0	南京	2020年10月10日10时10分	成都->北京->南京	0	修改 删除 发货
4	10012	1	1	郑州	2020年10月10日10时10分	成都->深圳->郑州	0	修改 删除 发货

增加 10003 号

物流信息增加表

返回上页

订单号	10003
物品类型	4
VIP	0
目的地	杭州
到达时间	202205130830

添加并保存

增加后：

物品信息管理

NO	订单号	物品类型	VIP	目的地	到达时间	物流路径	是否发货	相关操作
1	10001	1	0	西安	2022年02月23日08点20分	成都->广州->西安	1	修改 删除 发货
2	10003	4	0	杭州	2022年05月13日08点30分	成都->杭州	0	修改 删除 发货
3	11200	1	0	郑州	2022年06月13日06点30分	成都->深圳->郑州	1	修改 删除 发货
4	23564	2	1	广州	2022年01月22日05点30分	成都->广州	1	修改 删除 发货
5	57468	3	1	上海	2022年02月23日18点30分	成都->上海	0	修改 删除 发货
6	99999	4	1	郑州	2022年05月09日00点30分	成都->郑州	0	修改 删除 发货

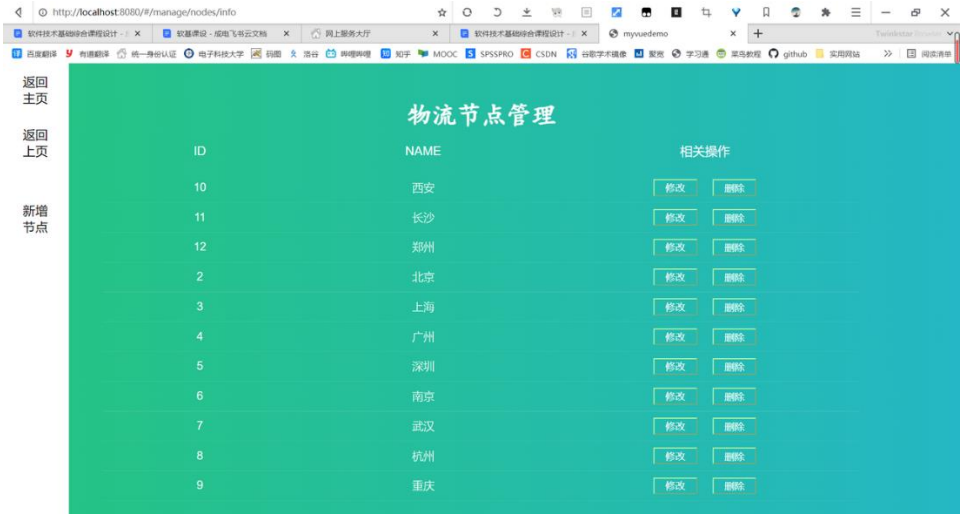
修改 1003 号

修改前：

修改后：

物流节点管理

物流节点管理系统有新增节点、修改和删除的操作。



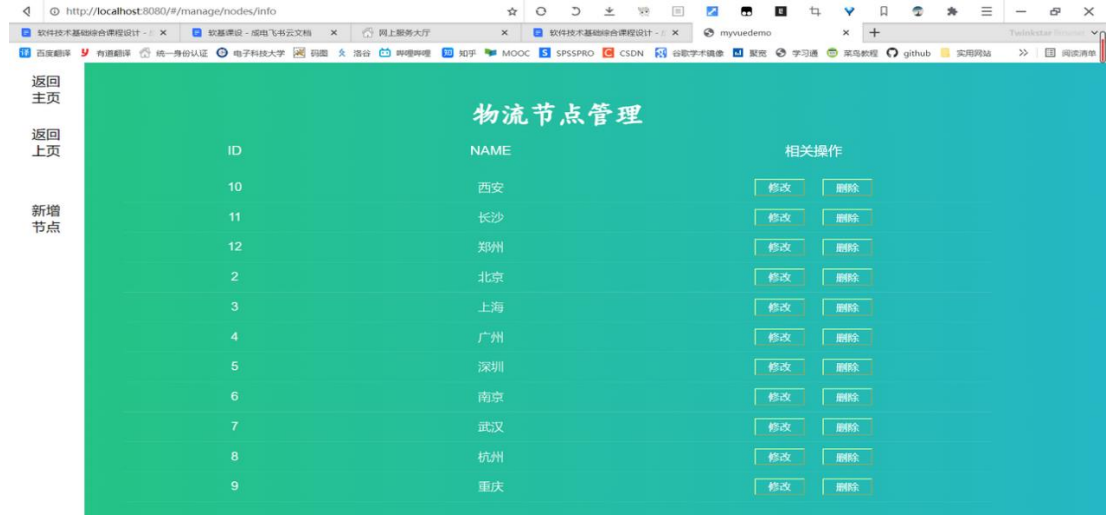
新增节点 13 名称为南昌

加入节点编号为 13 的南昌节点

修改 13 号节点为绵阳

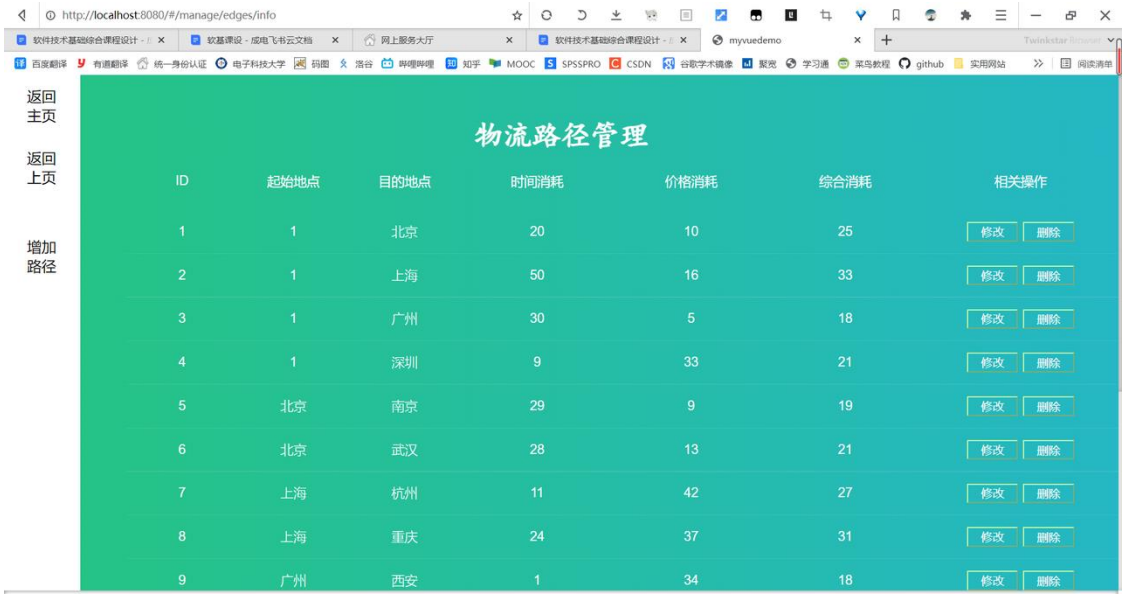
修改后的结果 南昌已经变为绵阳

删除南昌节点

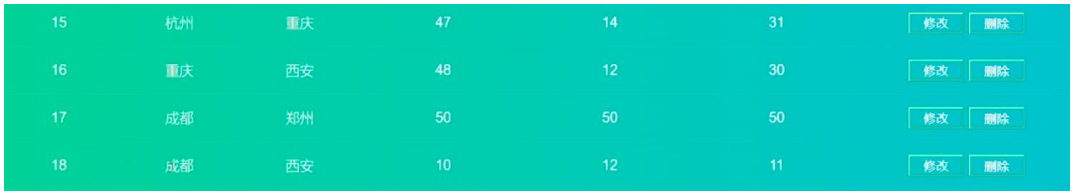


物流路径管理

物流路径管理系统有增加路径、修改和删除的操作。



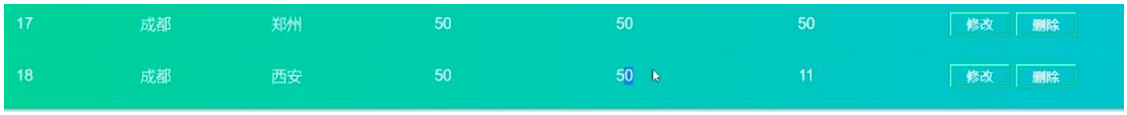
增加第 18 条路径



修改第 18 条路径
修改前：

修改后：

删除第 18 条路径
删除前：

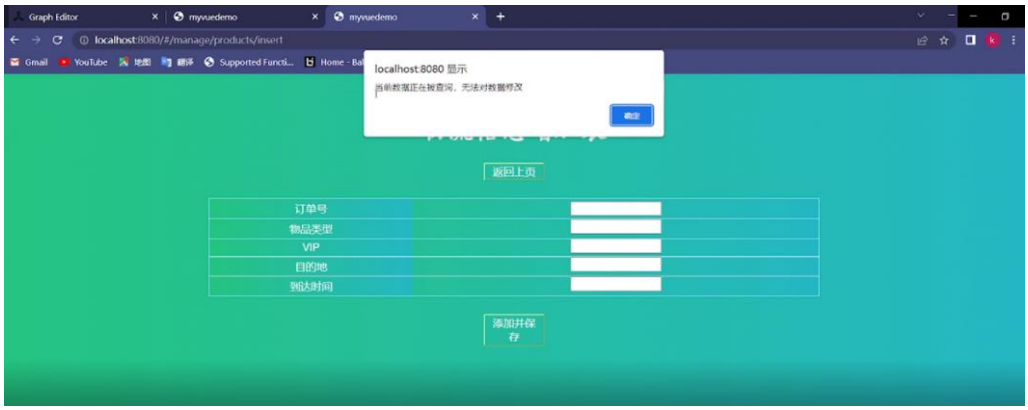


删除后：

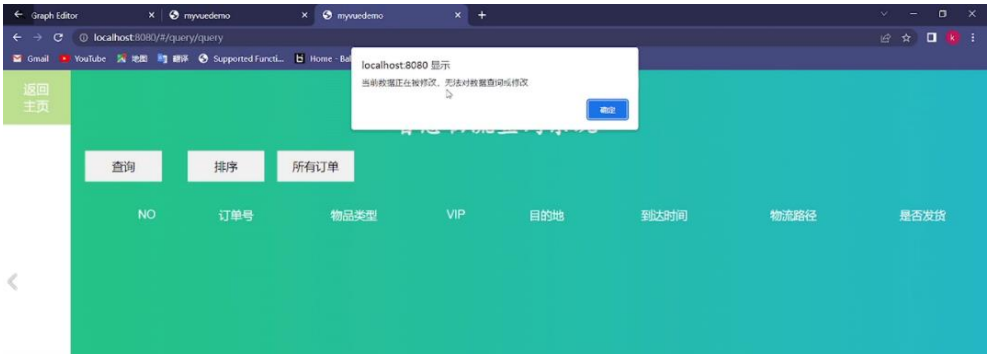
8.2 多进程模块介绍

根据多进程访问的三条规则，读时不能写，写时不能读和写。我们通过同时打开两个页面来演示这一功能。

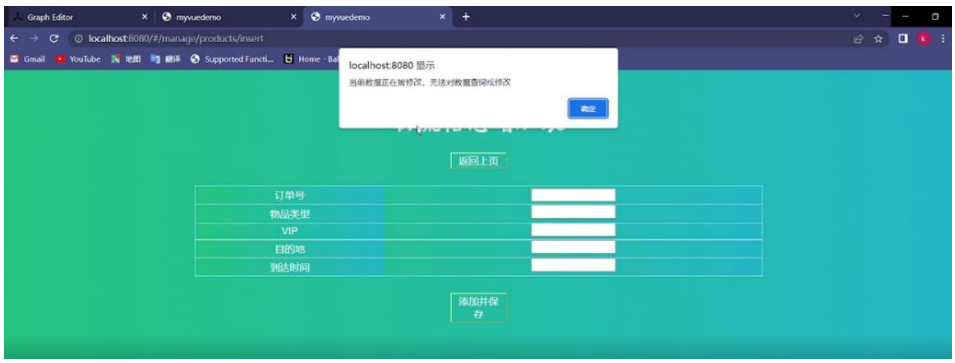
当一个页面正在读时，另一个页面不能写。



当一个页面正在写时，另一个页面不能读。



在一个页面写时，另一个页面不能写。

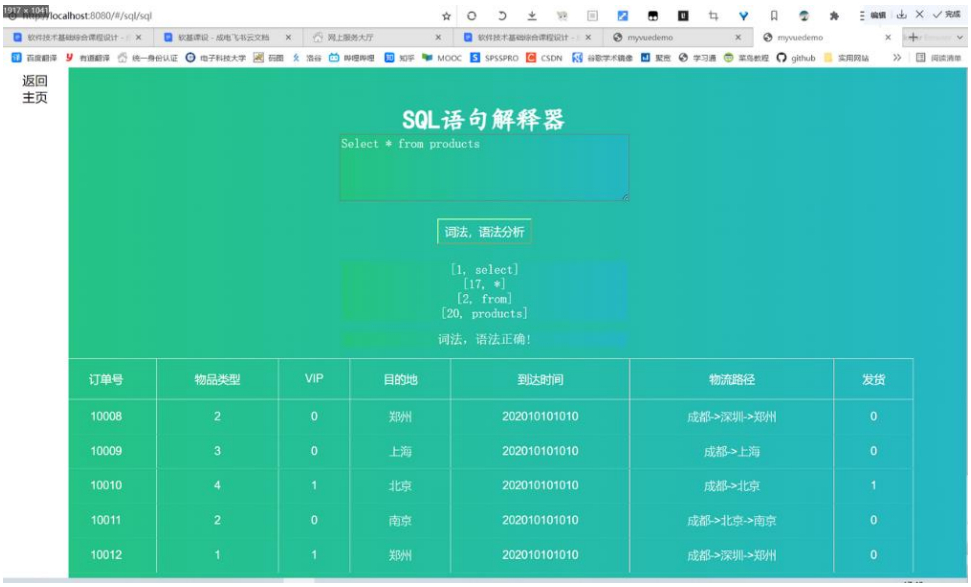


具体的详细演示部分请观看视频。

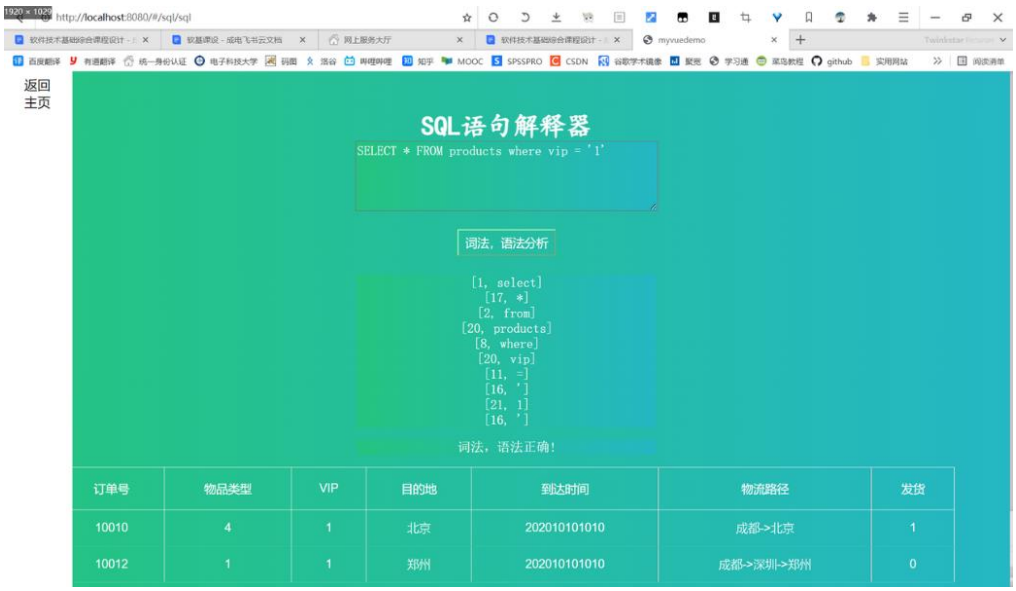
8.3 SQL 语句解析器功能演示

1.Select * from products

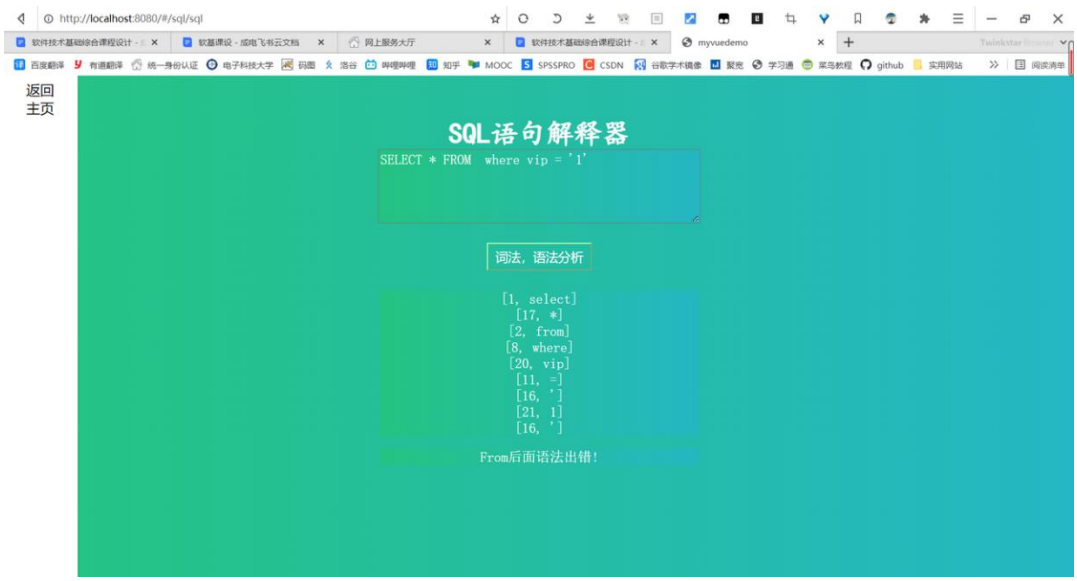
SQL 页面分为 4 个部分，第一部分 SQL 语句输入框，第二部分是 SQL 词法分析输出框，第三部分是 SQL 语法分析结果的输入框，第四部分是根据对应的 SQL 语句动态的输入页面渲染出动态的表格



2.SELECT * FROM products where vip = '1'



3.SELECT * FROM where vip = '1'

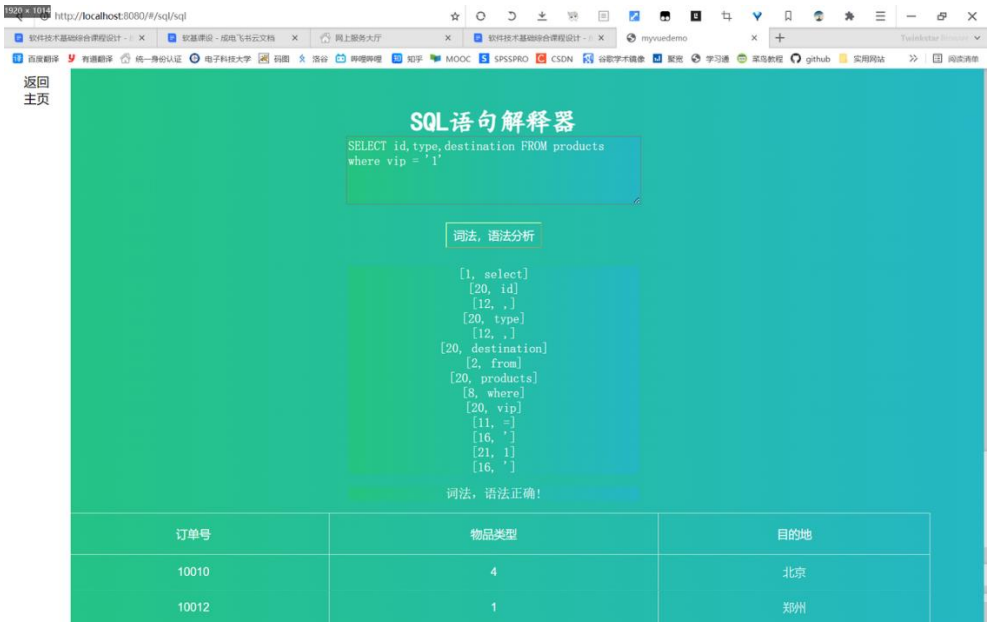


4.SELECT * FROM products where vip = '1

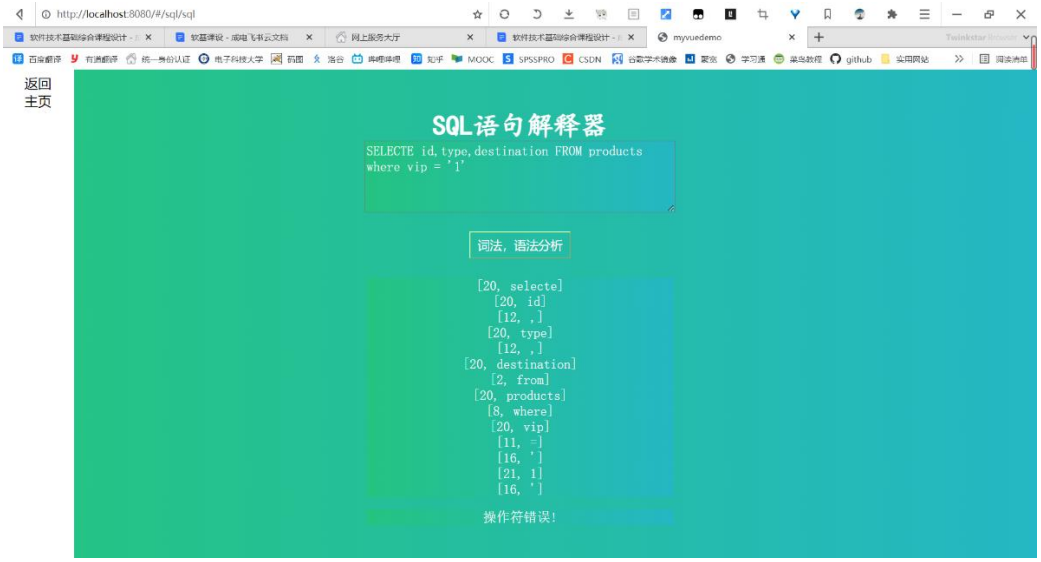
5.SELECT * FROM products where vip '1'

6.SELECT id type,destination FROM products where vip = '1'

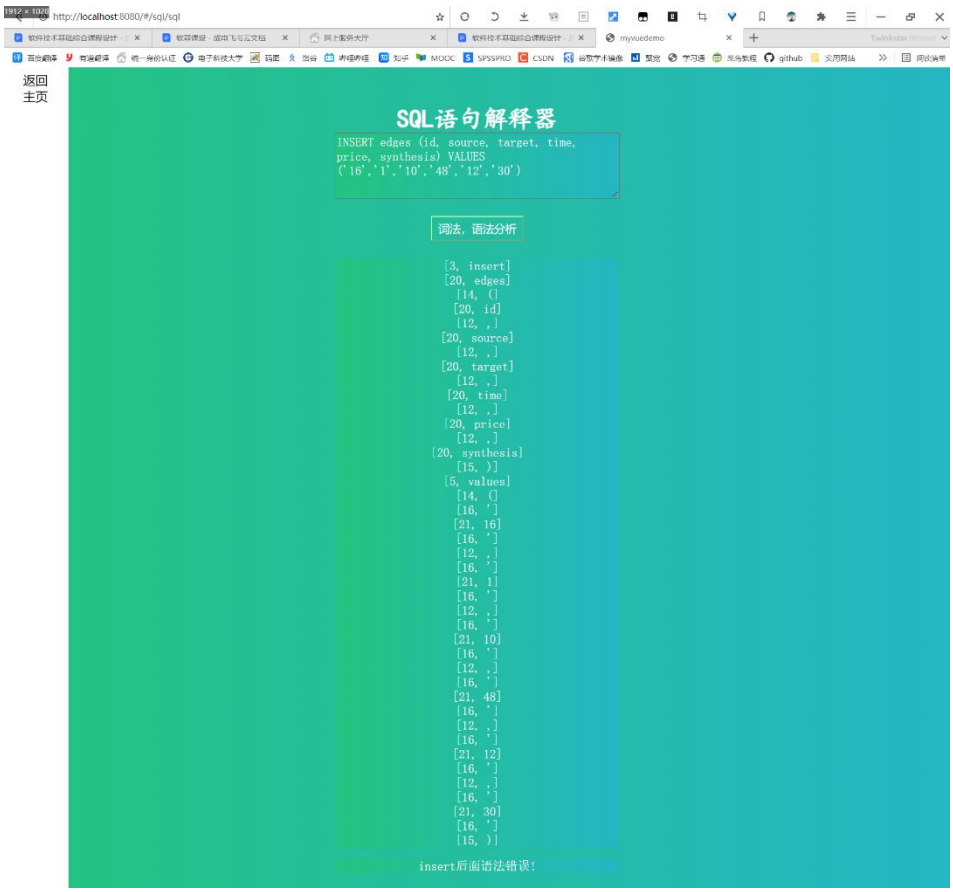
7.SELECT id,type,destination FROM products where vip = '1'



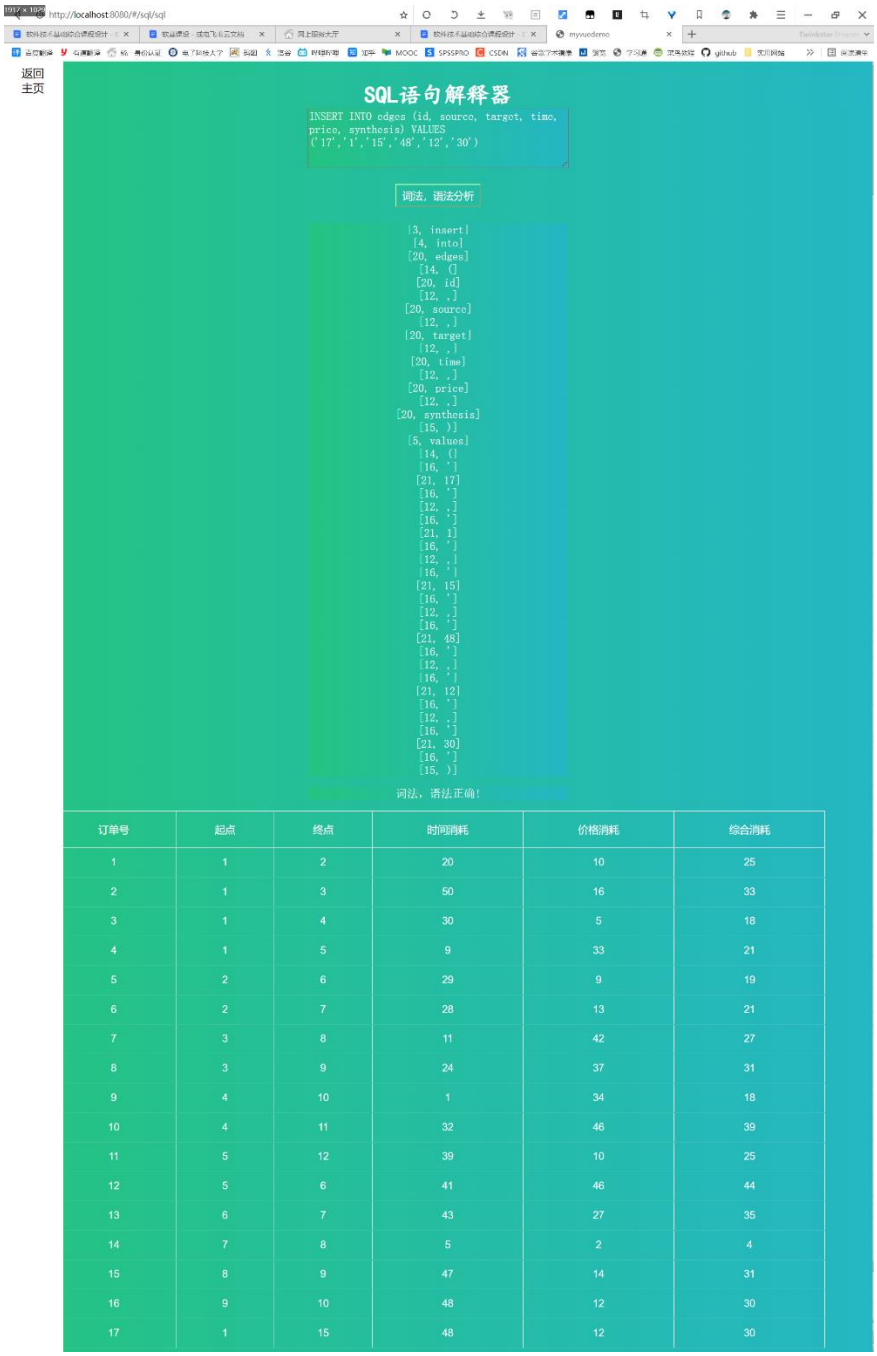
8.SELECTE id,type,destination FROM products where vip = '1'



9.INSERT edges (id, source, target, time, price, synthesis) VALUES ('16','1','10','48','12','30')

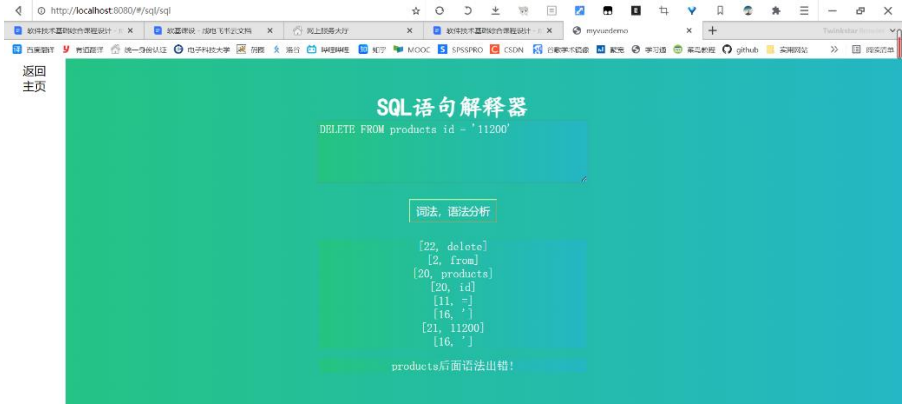


10.INSERT INTO edges (id, source, target, time, price, synthesis) VALUES ('17','1','15','48','12','30')

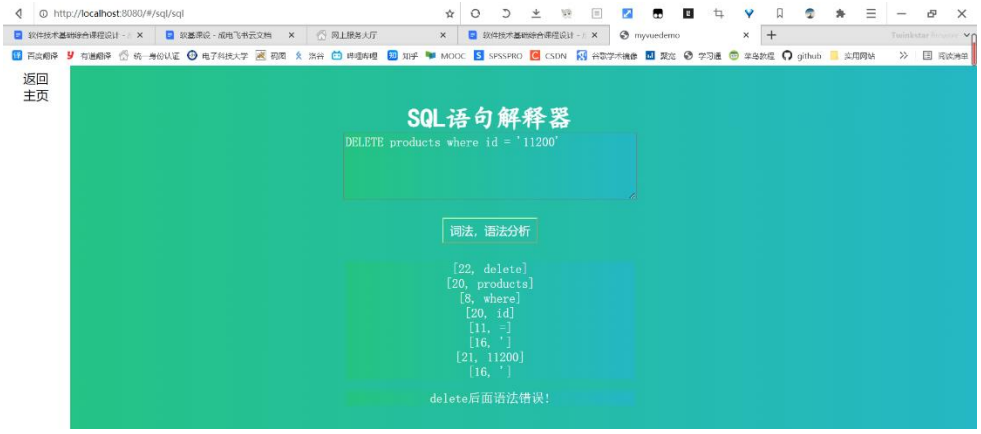


订单号	起点	终点	时间消耗	价格消耗	综合消耗
1	1	2	20	10	25
2	1	3	50	16	33
3	1	4	30	5	18
4	1	5	9	33	21
5	2	6	29	9	19
6	2	7	28	13	21
7	3	8	11	42	27
8	3	9	24	37	31
9	4	10	1	34	18
10	4	11	32	46	39
11	5	12	39	10	25
12	5	6	41	46	44
13	6	7	43	27	35
14	7	8	5	2	4
15	8	9	47	14	31
16	9	10	48	12	30
17	1	15	48	12	30

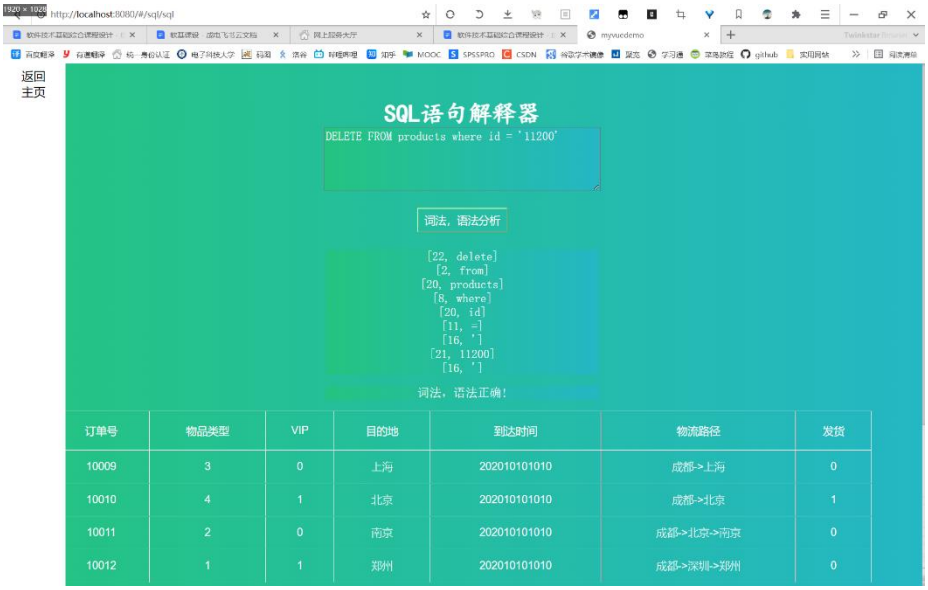
11.DELETE FROM products id = '11200'



12. DELETE products where id = '11200'

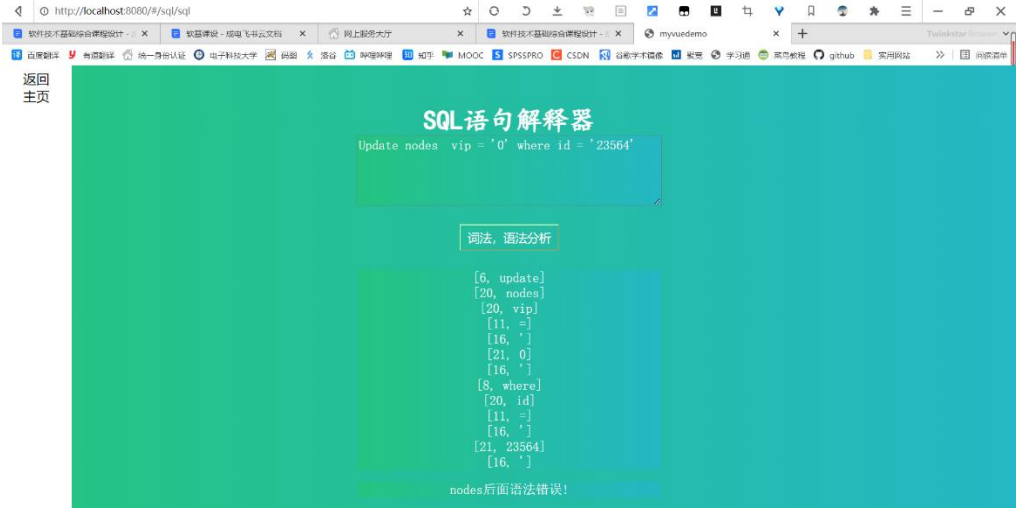


13.DELETE FROM products where id = '11200'

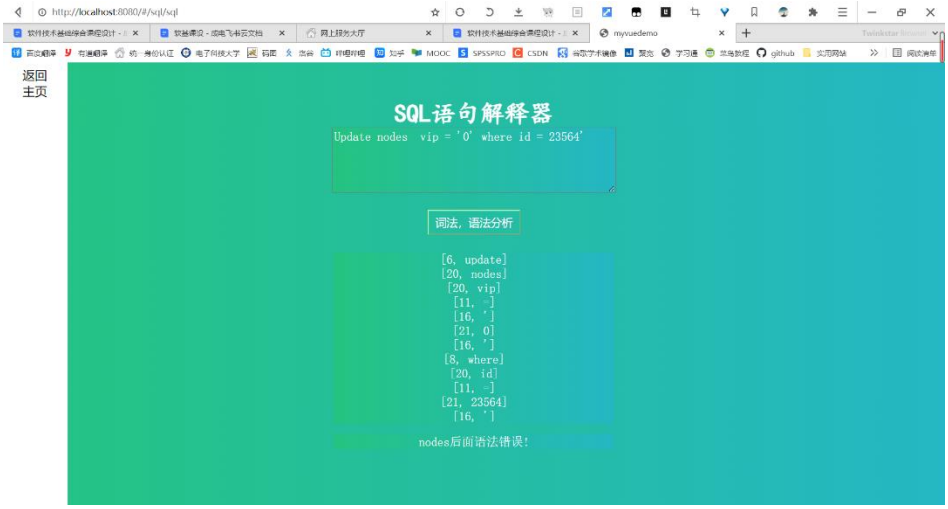


订单号	物品类型	VIP	目的地	到达时间	物流路径	发货
10009	3	0	上海	202010101010	成都->上海	0
10010	4	1	北京	202010101010	成都->北京	1
10011	2	0	南京	202010101010	成都->北京->南京	0
10012	1	1	郑州	202010101010	成都->深圳->郑州	0

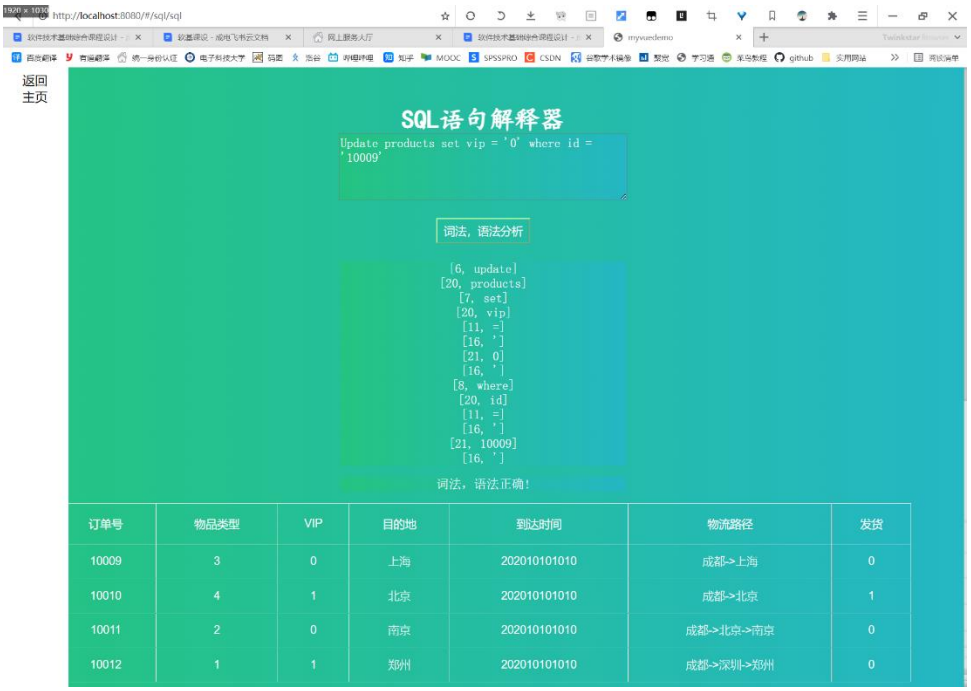
14.Update nodes vip = '0' where id = '23564'



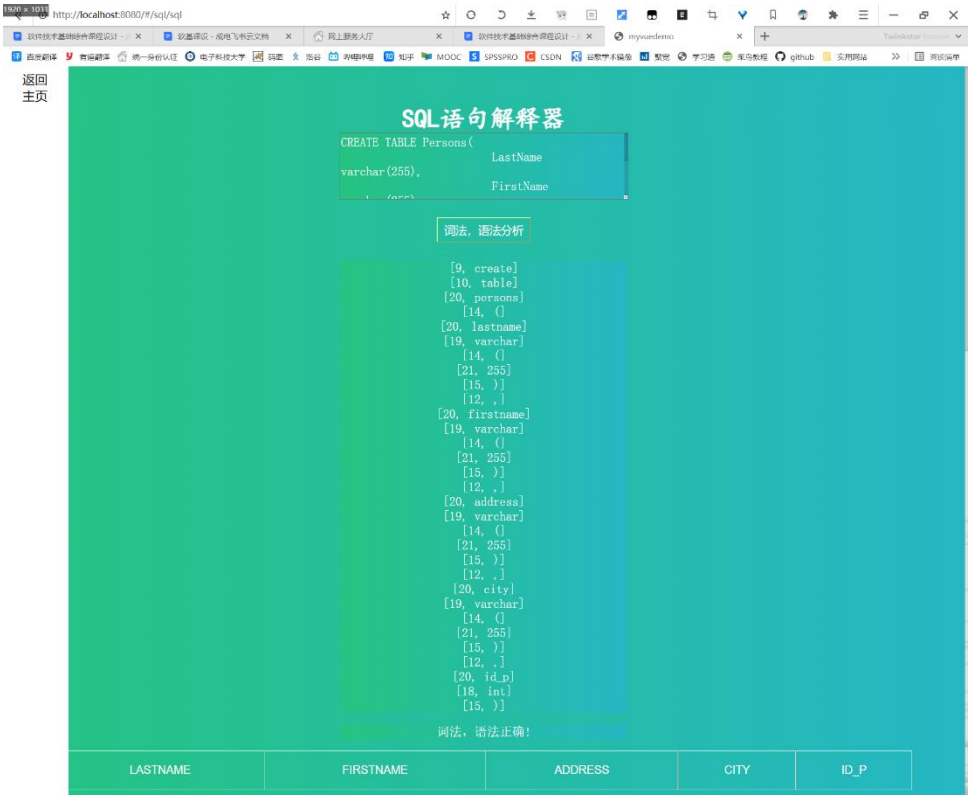
15.Update nodes vip = '0' where id = 23564'



16.Update nodes set vip = '0' where id = '23564'



17.CREATE TABLE Persons(LastName varchar(255),FirstName varchar(255),Address varchar(255),City varchar(255),Id_P int)



九、 数据结构与程序

9.1 树形结构存储数据

```
public class NodeTree {
    Goods data; //根节点数据
    NodeTree left; //左子树
    NodeTree right; //右子树

    public NodeTree() {
        super();
    }

    public NodeTree(Goods data) { //实例化二叉树
        super();
        this.data = data;
        left = null;
        right = null;
    }

    public void insert(NodeTree root, Goods data) {
        if (data.ID > root.data.ID) { //如果插入的节点大于跟节点
```

```
                if (root.right == null) {         //如果右子树为空，就插入，如果不为空就再创建一个节点
                    root.right = new NodeTree(data); //就把插入的节点放在右边
                } else {
                    this.insert(root.right, data);
                }
            } else {  //如果插入的节点小于根节点
                if (root.left == null) { //如果左子树为空，就插入，如果不为空就再创建一个节点
                    root.left = new NodeTree(data); //就把插入的节点放在左边边
                } else {
                    this.insert(root.left, data);
                }
            }
        }

        public static void preOrder(NodeTree root) { // 先根遍历
            if (root != null) {
                System.out.println(root.data + "-");
                System.out.println();
                preOrder(root.left);
                preOrder(root.right);
            }
        }

        public static void inOrder(NodeTree root) { // 中根遍历

            if (root != null) {
                inOrder(root.left);
                System.out.print(root.data + "--");
                inOrder(root.right);
            }
        }

        public static void postOrder(NodeTree root) { // 后根遍历

            if (root != null) {
                postOrder(root.left);
                postOrder(root.right);
                System.out.print(root.data + "---");
            }
        }
    }
```

9.2 dijkstra 算法计算最短路径

```
var express = require("express");
var app = express();
const BodyParser = require('body-parser')

// <-------------------------------------------dijkstra 算法-------------------
------------------------->
async function dijkstra(type, destination) {
    let nodes = [];
    let edges = [];
    let nodes_json = await pool.query("select * from nodes");
    let edges_json = await pool.query("select * from edges");
    for (let i = 0; i < nodes_json[0].length; i++) {
        nodes[i] = {
            id: nodes_json[0][i].id,
            name: nodes_json[0][i].name,
        }
    }
    for (let i = 0; i < edges_json[0].length; i++) {
        edges[i] = {
            source: edges_json[0][i].source,
            target: edges_json[0][i].target,
            time: edges_json[0][i].time,
            price: edges_json[0][i].price,
            synthesis: edges_json[0][i].synthesis,
        }
    }
    // await pool.end();

    // 构建时间属性的邻接矩阵，如果没有边的话是无穷
    let timeMatrix = new Array(100)
    for (let i = 1; i <= nodes.length; i++) {
        timeMatrix[i] = new Array(100)
        for (let j = 1; j <= nodes.length; j++) {
            timeMatrix[i][j] = 65535
        }
    }
    for (let i = 0; i < edges.length; i++) {
        timeMatrix[edges[i].source][edges[i].target] = edges[i].time
    }
```

```
    // 构建价格属性的邻接矩阵
    let priceMatrix = new Array(100)
    for (let i = 1; i <= nodes.length; i++) {
        priceMatrix[i] = new Array(100)
        for (let j = 1; j <= nodes.length; j++) {
            priceMatrix[i][j] = 65535
        }
    }
    for (let i = 0; i < edges.length; i++) {
        priceMatrix[edges[i].source][edges[i].target] = edges[i].price
    }

    // 构建综合属性的邻接矩阵
    let synthesisMatrix = new Array(100)
    for (let i = 1; i <= nodes.length; i++) {
        synthesisMatrix[i] = new Array(100)
        for (let j = 1; j <= nodes.length; j++) {
            synthesisMatrix[i][j] = 65535
        }
    }
    for (let i = 0; i < edges.length; i++) {
        synthesisMatrix[edges[i].source][edges[i].target] = edges[i].synthesis
    }

    // 如果 type = 1，则使用时间矩阵，如果 type = 2，则使用价格矩阵，如果 type = 3，则使用综合矩阵
    let matrix = new Array(nodes.length)
    if (type == 1) {
        matrix = timeMatrix
    }
    else if (type == 2) {
        matrix = priceMatrix
    }
    else if (type == 3) {
        matrix = synthesisMatrix
    }

    // 初始化一个距离数组
    let distance = new Array(nodes.length + 10)
    for (let i = 1; i <= nodes.length; i++) {
        distance[i] = 65535
    }

```

```
    // 另起始数组，成都为 0
    distance[1] = 0

    // 初始化一个路径数组
    let path = new Array(nodes.length + 10)
    for (let i = 1; i <= nodes.length; i++) {
        path[i] = -1
    }

    // 初始化一个已经访问过的节点数组
    let visited = new Array(nodes.length + 10)
    for (let i = 1; i <= nodes.length; i++) {
        visited[i] = false
    }

    // 从 0 到 n 开始枚举
    for (let i = 0; i < nodes.length; i++) {
        // 寻找最小距离的节点
        let min = 65535
        let minIndex = -1
        for (let j = 1; j <= nodes.length; j++) {
            if (visited[j] == false && (minIndex == -1 || distance[j] < min)) {
                min = distance[j]
                minIndex = j
            }
        }
        // 将最小距离的节点加入已经访问过的节点数组
        visited[minIndex] = true
        // 对最小距离的节点进行更新
        for (let k = 1; k <= nodes.length; k++) {
            if (distance[minIndex] + matrix[minIndex][k] < distance[k]) {
                distance[k] = distance[minIndex] + matrix[minIndex][k]
                path[k] = minIndex
            }
        }
    }
    // 输出最短路径
    let result = []
    let index = -1
    // index 为目的地 destination 对应的编号
    for (let i = 0; i < nodes.length; i++) {
        if (nodes[i].name === destination) {
            index = nodes[i].id
```

```
        }
    }
    while (index != -1) {
        result.push(index)
        index = path[index]
    }
    result.reverse()
    //输出最短路径中每个节点在 nodes 中的 name
    let resultName = []
    for (let i = 0; i < result.length; i++) {
        for (let j = 0; j < nodes.length; j++) {
            if (nodes[j].id == result[i]) {
                resultName.push(nodes[j].name)
            }
        }
    }
    //将 resultName 转换为形如 a->b->c 的字符串
    let resultString = ''
    for (let i = 0; i < resultName.length; i++) {
        resultString += resultName[i]
        if (i != resultName.length - 1) {
            resultString += "->"
        }
    }
    console.log(resultString)
    await pool.end();
    return resultString
}
module.exports = dijkstra;
// dijkstra(1, '郑州')
```

9.3 实现相应的接口从数据库中读取数据

```
var express = require("express");
var app = express();

let nodes = new Array();

// 从 wuliu 数据库的 nodes 表中读取数据，读取到的数据写入到 nodes 数组中
function readNodes() {
    connection.query("SELECT * FROM stuscore", function (err, result) {
        if (err) {
```

```
            console.log(err);
        } else {
            if (result.length == 0) {
                console.log("nodes 表中没有数据");
            } else {
                for (let i = 0; i < result.length; i++) {
                    nodes.push({
                        id: result[i].id,
                        name: result[i].name,
                    });
                }
            }
        }
    }
    );
}
readNodes();
```

9.4 对 excel 表格进行预处理

```
import org.apache.commons.io.FileUtils;
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.ss.usermodel.CellType;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class ExcelPreprocessing {
    public static void main(String[] args) {
        // 需要解析的 excel 文件
        File file = new File("src/main/resources/input.xls");
        try {
            // 创建 excel，读取文件
            HSSFWorkbook workbook = new HSSFWorkbook(FileUtils.openInputStream(file));

            // 获取第一个工作表 workbook.getSheet("Sheet0")
//            HSSFSheet sheet = workbook.getSheet("Sheet0");
```

```
22            // 读取默认第一个工作表 sheet
23            HSSFSheet sheet = workbook.getSheetAt(0);
24
25            // 创建 excel，用于存储数据
26            HSSFWorkbook workbook2 = new HSSFWorkbook();
27            // 创建一个工作表 sheet
28            HSSFSheet sheet2 = workbook2.createSheet();
29
30            int lastRowNum = sheet.getLastRowNum();
31            boolean trigger = true;
32            for (int i = 0; i <= lastRowNum; i++) {
33                HSSFRow row = sheet.getRow(i);
34                // 获取当前行最后单元格列号
35                int lastCellNum = row.getLastCellNum();
36
37                // 所属地
38                String belonging = null;
39                HSSFRow row2 = sheet2.createRow(i);
40                for (int j = 0; j < lastCellNum; j++) {
41                    HSSFCell cell = row.getCell(j);
42                    // 转化为 String 类型
43                    cell.setCellType(CellType.STRING);
44                    String value = cell.getStringCellValue();
45                    if (j == 2) {
46                        belonging = value;
47                    } else if (j == 3 && belonging == value) {
48                        if (trigger == true) {
49                            value = "H";
50                        } else {
51                            belonging = "H";
52                        }
53                        HSSFCell cell2 = row2.createCell(j - 1);
54                        cell2.setCellValue(belonging);
55                        trigger = !trigger;
56                    }
57
58                    // 创建处理过后的 excel
59                    HSSFCell cell2 = row2.createCell(j);
60                    cell2.setCellValue(value);
61
62                    System.out.print(value + " ");
63                }
64                System.out.println();
```

```
            }

            File file2 = new File("src/main/resources/products.xls");
            try {
                System.out.println("save");
                file2.createNewFile();
                // 将 excel 存盘
                FileOutputStream stream = FileUtils.openOutputStream(file2);
                workbook2.write(stream);
                stream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }

        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

9.5 读取 excel 数据

```
import org.apache.commons.io.FileUtils;
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import java.io.File;
import java.io.IOException;

public class ReadExcel {
    /**
     * Poi excel 文件解析
     */
    public static void main(String[] args) {
        // 需要解析的 excel 文件
        File file = new File("src/main/resources/input.xls");
        try {
            // 创建 excel，读取文件
```

```
            HSSFWorkbook workbook = new HSSFWorkbook(FileUtils.openInputStream(file));
            // 获取第一个工作表 workbook.getSheet("Sheet0")
//            HSSFSheet sheet = workbook.getSheet("Sheet0");
            // 读取默认第一个工作表 sheet
            HSSFSheet sheet = workbook.getSheetAt(0);
            int firstRowNum = 0;
            int lastRowNum = sheet.getLastRowNum();
            for (int i = 0; i < lastRowNum; i++) {
                HSSFRow row = sheet.getRow(i);
                // 获取当前行最后单元格列号
                int lastCellNum = row.getLastCellNum();
                for (int j = 0; j < lastCellNum; j++) {
                    HSSFCell cell = row.getCell(j);
                    String value = cell.getStringCellValue();
                    System.out.print(value + " ");
                }
                System.out.println();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

9.6 excel 操作整合 接口对接

```
import org.apache.commons.io.FileUtils;
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import java.io.File;
import java.io.IOException;
import java.util.*;

public class Main {
    private static final int goodsIdIndex = 0;
    private static final int goodsNameIndex = 1;
    private static final int belongingRegionIndex = 2;
```

```
    private static final int sendingRegionIndex = 3;
    private static final int goodsTypeIndex = 4;
    private static final int customerLevelIndex = 5;
    private static final int receiveDateIndex = 6;
    private static List<String> belongingRegionList = new ArrayList<>();
    private static ArrayList<Goods> goodsList = new ArrayList<>();

    public static void main(String[] args) {

        // 需要解析的 excel 文件
        File file = new File("src/main/resources/products.xls");
        try {
            readDataFromFile(file);

            while (true) {

                for (int i = 0; i < belongingRegionList.size(); i++) {
                    System.out.println(i + 1 + "." + belongingRegionList.get(i));
                }

                try {
                    Scanner input = new Scanner(System.in);
                    int inputValue = input.nextInt();
                    if (inputValue == 0) {
                        return;
                    } else if (inputValue < belongingRegionList.size()) {
                        regionSelector(inputValue - 1);
                    } else {
                        System.out.println("输入错误！请重新输入！");
                    }
                } catch (Exception e) {
                    System.out.println("输入错误！请重新输入！");
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void readDataFromFile(File file) throws IOException {
        HSSFWorkbook workbook = new HSSFWorkbook(FileUtils.openInputStream(file));
        // 读取默认第一个工作表 sheet
```

```
        HSSFSheet sheet = workbook.getSheetAt(0);
        int lastRowNum = sheet.getLastRowNum();
        Set<String> belongingRegionSet = new HashSet<>();
        for (int i = 1; i <= lastRowNum; i++) {
            HSSFRow row = sheet.getRow(i);
            HSSFCell cell0 = row.getCell(goodsIdIndex);
            HSSFCell cell1 = row.getCell(goodsNameIndex);
            HSSFCell cell2 = row.getCell(belongingRegionIndex);
            HSSFCell cell3 = row.getCell(sendingRegionIndex);
            HSSFCell cell4 = row.getCell(goodsTypeIndex);
            HSSFCell cell5 = row.getCell(customerLevelIndex);
            HSSFCell cell6 = row.getCell(receiveDateIndex);
            int goodsId = Integer.parseInt(cell0.getStringCellValue());
            String goodsName = cell1.getStringCellValue();
            String belongingRegion = cell2.getStringCellValue();
            belongingRegionSet.add(belongingRegion);
            String sendingRegion = cell3.getStringCellValue();
            int goodsType = Integer.parseInt(cell4.getStringCellValue());
            int clientGrade = Integer.parseInt(cell5.getStringCellValue());
            String receiveDate = cell6.getStringCellValue();

            Goods goods = new Goods(goodsId, goodsName, belongingRegion, sendingRegion, goodsType, clientGrade,
                    receiveDate);
            goodsList.add(goods);
        }
        belongingRegionList.addAll(belongingRegionSet);
    }

    // 依据不同的发货方案对数据进行相应的操作
    private static void regionSelector(int belongingRegionIndex) {
        String belongRegion = belongingRegionList.get(belongingRegionIndex);
        while (true) {
            Scanner input = new Scanner(System.in);
            int inputValue;
            try {
                inputValue = input.nextInt();
                switch (inputValue) {
                    case 0:
                        return;
                    case 1:
                        prioritySort(belongRegion);
                        break;
```

```
                    case 2:
                        projectSeletor(belongRegion, 1);
                        break;
                    case 3:
                        projectSeletor(belongRegion, 2);
                        break;
                    case 4:
                        projectSeletor(belongRegion, 3);
                        break;
                    case 5:
                        projectSeletor(belongRegion, 4);
                        break;
                    default:
                        System.out.println("输入错误！请重新输入！");
                        break;

                }
            } catch (Exception e) {
                e.printStackTrace();
                System.out.println("输入错误！请重新输入！");
            }
        }
    }

    // 对数据库中相关内容进行排序
    private static void projectSeletor(String belongingRegion, int projectId) {
        while (true) {
            Scanner input = new Scanner(System.in);
            int inputValue;
            try {
                inputValue = input.nextInt();
                switch (inputValue) {
                    case 0:
                        return;
                    case 1:
                        printGoodsOfThisProject(belongingRegion, projectId);
                        break;
                    case 2:
                        minStep(belongingRegionList.indexOf(belongingRegion), -1, projectId);
                        break;
                    case 3:
                        sendGoods(belongingRegion, projectId);
```

```
                    break;
                default:
                    System.out.println("输入错误！请重新输入！");
                    break;

            }
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("输入错误！请重新输入！");
        }
    }
}

// 发货操作
private static void sendGoods(String belongingRegion, int projectId) {
    while (true) {
        Scanner input = new Scanner(System.in);
        int inputValue;
        try {
            inputValue = input.nextInt();
            switch (inputValue) {
                case 0:
                    return;
                default:
                    goodsList.stream()
                            .filter(goods -> goods.belongingArea.equals(belongingRegion))
                            .filter(goods -> goods.type == projectId)
                            .filter(goods -> goods.ID == inputValue)
                            .forEach(el -> {
                                System.out.println(el);
                                minStep(belongingRegionList.indexOf(belongingRegion),
                                        belongingRegionList.indexOf(el.sendingArea), projectId);
                            });
                    break;
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("输入错误！请重新输入！");
        }
    }
```

```java
    }

    // if end == -1, then print all. else print specified route
    private static void minStep(int start, int end, int projectId) {
        File file = new File("src/main/resources/regionDistance.xls");
        try {
            int[][] Graph = new int[belongingRegionList.size()][belongingRegionList.size()];
            HSSFWorkbook workbook = new HSSFWorkbook(FileUtils.openInputStream(file));
            // 读取默认第一个工作表 sheet
            HSSFSheet sheet = workbook.getSheetAt(projectId - 1);
            for (int i = 1; i < belongingRegionList.size() + 1; i++) {
                HSSFRow row = sheet.getRow(i);
                // 获取当前行最后单元格列号
                for (int j = 1; j < belongingRegionList.size() + 1; j++) {
                    HSSFCell cell = row.getCell(j);
                    int value = (int) cell.getNumericCellValue();
                    Graph[i - 1][j - 1] = value;
                }
            }
            Dijkstra.minStep(Graph, start, end, belongingRegionList);
            System.out.println();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // 采用树形结构保存物品节点
    private static void printGoodsOfThisProject(String belongingRegion, int goodsType) {
        // NodeTree root = new NodeTree(goodsList.get(0)); 用树形结构存储，用根节点
        goodsList.stream()
                .filter(goods -> goods.belongingArea.equals(belongingRegion))
                .filter(goods -> goods.type == goodsType)
                .sorted((o1, o2) -> {
                    double diff = o1.priority - o2.priority;
                    if (diff == 0) {
                        return 0;
                    }
                    return diff > 0 ? -1 : 1;
                })
                .forEach(System.out::println);
```

```
        // .forEach(el-> {
        // root.insert(root, el); //插入元素
        // });
        System.out.println();
        // root.preOrder(root); //后向遍历
    }

    // 依据不同的指令对数据进行排序
    private static void prioritySort(String belongingRegion) {
        goodsList.stream()
                .filter(goods -> goods.belongingArea.equals(belongingRegion))
                .sorted((o1, o2) -> {
                    double diff = o1.priority - o2.priority;
                    if (diff == 0) {
                        return 0;
                    }
                    return diff > 0 ? -1 : 1;
                })
                .forEach(System.out::println);
        System.out.println();
    }
}
```

9.7 对 excel 数据表进行保存

```
import org.apache.commons.io.FileUtils;
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class ExcelSave {
    /**
     * 生成 excel 文件
     */
    public static void main(String[] args) {

        String[] title = { "id", "name" };
```

```

        // 创建 excel 工作簿
        HSSFWorkbook workbook = new HSSFWorkbook();
        // 创建一个工作表 sheet
        HSSFSheet sheet = workbook.createSheet();
        // 创建第一行
        HSSFRow row = sheet.createRow(0);
        HSSFCell cell = null;
        // 在 excel 插入对应的数据
        for (let i = 0; i < title.length; i++) {
            cell = row.createCell(i);
            cell.setCellValue(title[i]);
        }
        // 在 excel 中插入其他信息
        for (let i = 1; i <= 10; i++) {
            HSSFRow nextrow = sheet.createRow(i);
            HSSFCell cell2 = nextrow.createCell(0);
            cell2.setCellValue("a" + i);
            cell2 = nextrow.createCell(1);
            cell2.setCellValue("user" + i);
            cell2 = nextrow.createCell(2);
        }

        // 创建一个文件
        File file = new File("src/main/resources/products.xls");
        try {
            file.createNewFile();
            // 将 excel 存盘
            FileOutputStream stream = FileUtils.openOutputStream(file);
            workbook.write(stream);
            stream.close();
        } catch (IOException e) {
            e.prletStackTrace();
        }
    }
}
```

9.8 产生多进程访问和互斥操作

```
// <-------------------------------------------信号量设置与操作-----------------
----------------------------->
let isReading = false;
```

```
let isWriting = false;

app.post("/write_signal", function(req, res) {
    console.log('write_signal 被调用')
    if((!isReading)&&(!isWriting)){
        isWriting = true
        console.log("isWriting: ", isWriting)
    }
    else{
        if(isReading){
            console.log('接口被占用，无法写(isReading)')
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(
                'isReading'
            );
        }
        if(isWriting){
            console.log('接口被占用，无法写(isWriting)')
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(
                'isWriting'
            );
        }
    }
});

app.post("/read_signal", function(req, res) {
    if(!isWriting){
        isReading = true
        console.log("isReading", isReading)
    }else{
        //无法查询页面
        console.log('接口被占用，无法读')
        res.writeHead(200, { 'Content-Type': 'application/json' });
        res.end(
            'isWriting'
        );
    }
});

app.post("/back_signal_read", function(req, res) {
    isReading = false
    console.log("isReading", isReading)
```

```
});

app.post("/back_signal", function(req, res) {
    isWriting = false
    console.log("isWriting: ", isWriting)
});
```

9.9 前后端交互接口（增删改查操作）

```
// <----------------------------------简易数据库 SELECT 操作----------------------
------------------------>
// query_products 接口：查询整张表的物品信息数据(manage 页面中，不设置信号量)
app.get("/query_products_manage", function (req, res) {
    // console.log("主页 GET 请求,前端请求数据库数据：");
    connection.query("SELECT * FROM products", function (error, results, fields)
 {
        if (error) throw error;
        res.writeHead(200, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify(results));
    });
});

// query_products 接口：查询整张表的物品信息数据(wuliuquery 页面中，需设置信号量)
app.get("/query_products", function (req, res) {
    if(!isWriting){
        connection.query("SELECT * FROM products", function (error, results, fiel
ds) {
            if (error) throw error;
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify(results));
        });
    }
    else{
        // console.log("接口被占用")
    }
});

//query_product 接口:查询特定 id 物品信息，便于展示到前端并修改.
app.get("/query_product", function (req, res) {
    editid = req.query.id
    console.log("id:", editid)
```

```
    connection.query(`SELECT * FROM products where id=${editid}`, function (error, results, fields) {
        if (error) throw error;
        else {
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify(results));
        }
    });
})

// query_nodes 接口：查询物流节点信息数据
app.get("/query_nodes", function (req, res) {
    // console.log("主页 GET 请求,前端请求数据库数据：");
    if(!isWriting){
        connection.query("SELECT * FROM nodes", function (error, results, fields) {
            if (error) throw error;
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify(results));
        });
    }
    else{
        console.log("接口被占用")
    }
});

// query_node 接口：查询特定 id 物流节点信息，便于展示到前端并修改.
app.get("/query_node", function (req, res) {
    editid = req.query.id
    console.log("id:", editid)
    connection.query(`SELECT * FROM nodes where id=${editid}`, function (error, results, fields) {
        if (error) throw error;
        else {
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify(results));
        }
    });
})

// query_edges 接口：查询物流路径信息数据
app.get("/query_edges", function (req, res) {
    // console.log("主页 GET 请求,前端请求数据库数据：");
```

```
    if(!isWriting){
        connection.query("SELECT * FROM edges", function (error, results, fields) {
            if (error) throw error;
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify(results));
        });
    }
    else{
        console.log("接口被占用")
    }
});

// query_edge 接口：查询特定 id 物流路径信息，便于展示到前端并修改.
app.get("/query_edge", function (req, res) {
    editid = req.query.id
    console.log("id:", editid)
    connection.query(`SELECT * FROM edges where id=${editid}`, function (error, results, fields) {
        if (error) throw error;
        else {
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify(results));
        }
    });
})

// <-------------------------------------------数据库 DELETE 操作-------------------------------------------->
// del_product 接口：删除物品信息(通过 id)
app.post("/del_product", function (req, res) {

    // var del_id = req.body.id
    connection.query(`DELETE  FROM products where id = ${req.body.id}`,
        function (error, results, fields) {
            if (error) console.log(error);
            else {
                console.log("编号为 ", req.body.id, "的节点信息删除成功。")
                // console.log("The result is: ", results);
            }
        });
    console.log("主页 POST 请求,收到前端信息，删除的节点编号为：");

```

```
});

//del_node 接口：删除物流节点信息(通过 id)
app.post("/del_node", function (req, res) {
    // var del_id = req.body.id
    connection.query(`DELETE  FROM nodes where id = ${req.body.id}`,
        function (error, results, fields) {
            if (error) console.log(error);
            else {
                console.log("编号为 ", req.body.id, "的节点信息删除成功。")
                // console.log("The result is: ", results);
            }
        });
    console.log("主页 POST 请求,收到前端信息，删除的节点编号为：");
    console.log(req.body.id)
});

// del_edge 接口：删除物流路径节点信息(通过 id)
app.post("/del_edge", function (req, res) {
    // var del_id = req.body.id
    connection.query(`DELETE  FROM edges where id = ${req.body.id}`,
        function (error, results, fields) {
            if (error) console.log(error);
            else {
                console.log("编号为 ", req.body.id, "的节点信息删除成功。")
                // console.log("The result is: ", results);
            }
        });
    console.log("主页 POST 请求,收到前端信息，删除的节点编号为：");
    console.log(req.body.id)
});

// <-------------------------------------------数据库 UPADATE 操作--------------
------------------------------>
// edit_product 接口：修改物品信息(通过 id)
app.post("/edit_product", function (req, res) {
    if((!isReading)&&(!isWriting)){
        isWriting = false
        console.log("req.body:", req.body)
        let editid = req.body.id
        let edit_obj = req.body
        console.log("主页 POST 请求,收到前端要修改的物流节点的 id:");
        console.log(editid)
```

```
        edit_obj.deliver = 0

        let sql = `UPDATE products SET ? where id = ${editid}`;
        //在这里说明是一个参数，通过下边的 edit_obj 代替
        connection.query(sql, edit_obj, (err, result) => {
            if (err) {
                console.log(err)
            } else {
                console.log(result);
                res.send(`update ${editid} success......`)
            }
        })
        console.log(isWriting)
    }else{
        //无法打开 insert 和 edit 页面
        console.log('无法写入')
    }
});

//deliverproduct 接口：通过 id 在数据库中修改物品发货信息
app.post('/deliverproduct', function (req, res) {
    let editid = req.body.id
    let edit_obj = req.body
    edit_obj.deliver = 1
    console.log("主页 POST 请求,收到前端要发货的物品的 id:");
    console.log(editid)
    let sql = `UPDATE products SET ? where id = ${editid}`;
    //在这里说明是一个参数，通过下边的 edit_obj 代替
    connection.query(sql, edit_obj, (err, result) => {
        if (err) {
            console.log(err)
        } else {
            console.log(result);
            res.send(`update ${editid} success......`)
        }
    })
});

// edit_node 接口：修改物流节点信息(通过 id)
app.post("/edit_node", function (req, res) {
    if((!isReading)&&(!isWriting)){
        isWriting = false
        console.log("req.body:", req.body)
```

```
        let editid = req.body.id
        let edit_obj = req.body
        console.log("主页 POST 请求,收到前端要修改的物流节点的 id: ");
        console.log(editid)

        let sql = `UPDATE nodes SET ? where id = ${editid}`;
        //在这里说明是一个参数，通过下边的 edit_obj 代替
        connection.query(sql, edit_obj, (err, result) => {
            if (err) {
                console.log(err)
            } else {
                console.log(result);
                res.send(`update ${editid} success......`)
            }
        })
    }else{
        //无法打开 insert 和 edit 页面
        console.log('无法写入')
    }
});

//edit_edge 接口：修改物流路径信息(通过 id)
app.post("/edit_edge", function (req, res) {
    if((!isReading)&&(!isWriting)){
        isWriting = false
        console.log("req.body:", req.body)
        let editid = req.body.id
        let edit_obj = req.body
        console.log("主页 POST 请求,收到前端要修改的物流节点的 id: ");
        console.log(editid)

        let sql = `UPDATE edges SET ? where id = ${editid}`;
        //在这里说明是一个参数，通过下边的 edit_obj 代替
        connection.query(sql, edit_obj, (err, result) => {
            if (err) {
                console.log(err)
            } else {
                console.log(result);
                res.send(`update ${editid} success......`)
            }
        })
    }else{
        //无法打开 insert 和 edit 页面
```

```
        console.log('无法写入')
    }
});

// <-------------------------------------------数据库 INSERT 操作----------------------------------------->
// insert_product 接口：新建物品信息
app.post("/insert_product", function (req, res) {
    if((!isReading)&&(!isWriting)){
        isWriting = false
        var insert_obj = req.body
        console.log("主页 POST 请求,收到前端表单提交的数据：");
        console.log(insert_obj)
        // 默认没有发货
        insert_obj.deliver = 0

        //根据类型利用迪杰斯特拉算法计算最短路径，如果类型为 1 用 edges 里的时间计算，如果类型为 2 用 edges 里的价格计算，如果类型为 3 用 edges 里的 synthesis 计算
        if (insert_obj.type != 4) {
            dijkstra(insert_obj.type, insert_obj.destination).then(res => {
                insert_obj.logistics_routes = res
                //将数据插入数据库
                connection.query('INSERT INTO products SET ?', insert_obj, (err, result) => {
                    if (err) {
                        console.log(err)
                    } else {
                        console.log(result);
                    }
                })
            }).catch(err => {
                console.log(err)
            })
        }

        //如果类型为 4，则物流路径为直达
        else {
            insert_obj.logistics_routes = '成都->' + insert_obj.destination
            //将数据插入数据库
            connection.query('INSERT INTO products SET ?', insert_obj, (err, result) => {
                if (err) {
                    console.log(err)
```

```
                } else {
                    console.log(result);
                }
            }
            )
        }
    }else{

        //无法打开 insert 和 edit 页面
        console.log('无法写入')
    }

});

// insert_node 接口：新建物流节点信息
app.post("/insert_node", function (req, res) {
    if((!isReading)&&(!isWriting)){
        isWriting = false
        var insert_obj = req.body
        console.log("主页 POST 请求,收到前端表单提交的数据：");
        console.log(insert_obj)

        //将数据插入数据库
        connection.query('INSERT INTO nodes SET ?', insert_obj, (err, result) =>
{
            if (err) {
                console.log(err)
            } else {
                console.log(result);
                res.send('添加成功')
            }
        })
    }else{
        //无法打开 insert 和 edit 页面
        console.log('无法写入')
    }
});

// insert_edge 接口：新建物流边信息
app.post("/insert_edge", function (req, res) {
    if((!isReading)&&(!isWriting)){
        isWriting = false
        var insert_obj = req.body
```

```
        console.log("主页 POST 请求,收到前端表单提交的数据: ");
        console.log(insert_obj)

        //计算综合价值
        insert_obj.synthesis = insert_obj.time * 0.5 + insert_obj.price * 0.5

        //将数据插入数据库
        connection.query('INSERT INTO edges SET ?', insert_obj, (err, result) => {
            if (err) {
                console.log(err)
            } else {
                console.log(result);
                res.send('添加成功')
            }
        })
    }else{
        //无法打开 insert 和 edit 页面
        console.log('无法写入')
    }
});

// <-----------------------------------------sql 解析器页面 Wuliusql 数据库操作------------------------------------------>
// sql 接口:将解析 sql 后的结果展示到前端
app.post("/sql", function (req, res) {
    // console.log("主页 GET 请求,前端请求数据库数据: ");
    // console.log(req)
    let table_name = req.body.table_name
    let sql = req.body.sql
    let start_sql = sql.slice(0,6)
    if(start_sql.toLowerCase() == "select"){
        connection.query(sql, function (error, results, fields) {
            if (error) throw error;
            // console.log("The result is: ", results);
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify(results));
        });
    }
    else if(start_sql.toLowerCase() == "insert" || start_sql.toLowerCase() == "update" || start_sql.toLowerCase() == "delete"){
        connection.query(sql, function (error, results, fields) {
            if (error) throw error;
```

```
        });
        connection.query("SELECT * FROM "+table_name, function (error, results, fields) {
            if (error) throw error;
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify(results));
        });
    }
    else{
        connection.query(sql, function (error, results, fields) {
            if (error) throw error;
        });
    }

});

var server = app.listen(3030, function () {
    var host = server.address().address;
    var port = server.address().port;
    console.log("应用实例，访问地址为 http://%s:%s", host, port);
});
```

9.10 SQL 解析器

```
export default{
          name: "sql",
          data() {
              return {
              msg: "欢迎来到物流信息管理系统",
              bodyData: Array(),
              };
          },
          created: function() {

          },

          methods: {
              print() {
                  this.bodyData = []
                  let input = document.querySelector(".input");
                  let run = document.querySelector("button");
                  let output1 = document.querySelector(".output1");
```

```
let output2 = document.querySelector(".output2");
var box = document.getElementById('box');
let table_name;
let keys = [
    { name: "select", id: 1 },
    { name: "from", id: 2 },
    { name: "insert", id: 3 },
    { name: "into", id: 4 },
    { name: "values", id: 5 },
    { name: "update", id: 6 },
    { name: "set", id: 7 },
    { name: "where", id: 8 },
    { name: "create", id: 9 },
    { name: "table", id: 10 },
    { name: "delete", id: 22 },
    { name: "int", id: 18 },
    { name: "varchar", id: 19 }
];
let bodyData = [];
let hashs = [
    { name: "订单号", id: "id" },
    { name: "物品类型", id: "type" },
    { name: "Vip", id: "vip" },
    { name: "目的地", id: "destination" },
    { name: "到达时间", id: "arrival_time" },
    { name: "物流路径", id: "logistics_routes" },
    { name: "发货", id: "deliver" },
    { name: "起点", id: "source" },
    { name: "终点", id: "target" },
    { name: "时间消耗", id: "time" },
    { name: "价格消耗", id: "price" },
    { name: "综合消耗", id: "synthesis" },
    { name: "节点名", id: "name" },
]
let t = 0; let syn = 0; let p = 0; let q = 0;
let tokens = []; let res = ""; let Head = [];

function init() {
    res = "";
    syn = 0;
    tokens.length = 0;
    q = 0;
    p = 0;
```

```
                    Head.length = 0;
                    table_name="";
                    document.getElementById('box').innerHTML = "";
                }
                function hash_Head() {
                    for (let i = 0; i < Head.length; i++) {
                        for (let j = 0; j < hashs.length; j++) {
                            if (Head[i] == hashs[j].id) {
                                Head[i] = hashs[j].name;
                                break;
                            }
                        }
                    }
                }
                let insert_values = function () {
                    if (tokens[t].syns == 14) {
                        t++;
                        if (tokens[t].syns == 16) {
                            while (tokens[t].syns == 16) {
                                t++;
                                if (tokens[t].syns == 20 || tokens[t].syns == 21) {
                                    t++;
                                    if (tokens[t].syns == 16) {
                                        t++;
                                        // console.log(tokens[t].token);
                                        if (tokens[t].syns == 15) {
                                            res = "词法，语法正确!\n";
                                        } else if (tokens[t].syns == 12) {
                                            t++;
                                            if (tokens[t].syns != 16) {
                                                res = ",后语法错误!\n";
                                                break;
                                            }
                                        } else {
                                            res = "'后语法错误!\n";
                                            break;
                                        }
                                    } else {
                                        res = `${tokens[t - 1].token}` + "后语法错误!\n";
                                    }
                                } else {
```

```
                                res = "'后语法错误!\n";
                            }
                        }
                    } else {
                        res = "(后语法错误!\n";
                    }
                } else {
                    res = "values 后语法错误!\n";
                }
            }
            let create_word = function () {
                t++;
                if (tokens[t].syns == 19) {
                    t++;
                    if (tokens[t].syns == 14) {
                        t++;
                        if (tokens[t].syns == 21) {
                            t++;
                            if (tokens[t].syns == 15) {
                                t++;
                                if (tokens[t].syns == 12) {
                                    t++;
                                    if (tokens[t].syns == 15) {
                                        res = ",后面语法错误\n";
                                    }
                                } else if (tokens[t].syns == 15) {
                                    res = "词法，语法正确!\n";
                                } else {
                                    res = ")后面语法错误\n";
                                }
                            } else {
                                res = `${tokens[t].token}` + "后面语法错误\n";
                            }
                        } else {
                            res = "(后面语法错误\n";
                        }

                    } else {
                        res = "varchar 后面语法错误\n";
                    }
                } else if (tokens[t].syns == 18) {
                    t++;
                    if (tokens[t].syns == 12) {
```

```
                        t++;
                        if (tokens[t].syns == 15) {
                            res = ",后面语法错误\n";
                        }
                    } else if (tokens[t].syns == 15) {
                        res = "词法，语法正确!\n";
                    } else {
                        res = "int 后面语法错误!\n";
                    }
                } else {
                    res = `${tokens[t - 2].token}` + "后面语法错误\n";
                }
            }
            let select_where = function () {
                if (tokens[t].syns == 20) {
                    t++;
                    if (tokens[t].syns == 11) {
                        t++;
                        if (tokens[t].syns == 16) {
                            t++
                            if (tokens[t].syns == 21 || tokens[t].syns == 20) {
                                t++;
                                if (t < tokens.length && tokens[t].syns == 16) {
                                    res = "词法，语法正确!\n";
                                } else {
                                    res = `${tokens[t - 1].token}` + "后面语法出错!\n";
                                }
                            } else {
                                res = "语法出错!\n";
                            }
                        } else {
                            res = "=后面语法出错!\n";
                        }
                    } else {
                        res = `${tokens[t - 1].token}` + "后面语法出错!\n";
                    }
                } else {
                    res = "where 后面语法出错!\n";
                }
            }
```

```
                function judge_table() {
                    if (Head.length == 0) {
                        if (tokens[t].token == "nodes") {
                            Head = ["id", "name"];
                        } else if (tokens[t].token == "edges") {
                            Head = ["id", "source", "target", "time", "price", "synthesis"];
                        } else if (tokens[t].token == "products") {
                            Head = ["id", "type", "vip", "destination", "arrival_time", "logistics_routes", "deliver"];
                        } else {
                            res = "数据库中不存在该表\n";
                            return;
                        }
                    }
                }
                let select_from = function (t) {
                    //SELECT * FROM GOODSLIST
                    if (tokens[t].syns == 20) {
                        judge_table();
                        t++;
                        if (t >= tokens.length) {
                            res = "词法，语法正确!\n";
                        } else if (tokens[t++].syns == 8) {
                            if (tokens[t++].syns == 20) {
                                if (tokens[t++].syns == 11) {
                                    if (tokens[t++].syns == 16) {
                                        if (tokens[t].syns == 21 || tokens[t].syns == 20) {
                                            t++;
                                            if (t < tokens.length && tokens[t].syns == 16) {
                                                res = "词法，语法正确!\n"
                                            } else {
                                                res = `${tokens[t - 1].token}` + "后面语法出错!\n";
                                            }
                                        } else {
                                            res = "语法出错!\n";
                                        }
                                    } else {
                                        res = "=后面语法出错!\n";
                                    }
```

```
                            } else {
                                res = `${tokens[t - 2].token}` + "后面语法出错!\n";
                            }
                        } else {
                            res = "where 后面语法出错!\n";
                        }
                    } else {
                        res = `${tokens[t - 2].token}` + "后面语法出错!\n";
                    }
                } else {
                    res = "From 后面语法出错!\n";
                }
            }

            function createTable(headData, bodyData) {
                document.getElementById('box').innerHTML = "";
                let table = document.createElement("table");
                box.appendChild(table);
                table.border = "1px";
                table.width = "500px";
                table.cellSpacing = 0;
                //table.margin=auto;
                table.style.margin="auto";
                createHead(table, headData);
                createBody(table, bodyData);


            }
            // 创建表头
            function createHead(table, columnNames) {
                // thead -> table
                let thead = document.createElement("thead");
                table.appendChild(thead);
                // tr -> thead
                let tr = document.createElement("tr");
                thead.appendChild(tr);
                tr.style.height = "35px";
                tr.style.background = "transparent";t
                table.style.margin="auto";
                // th -> tr
                columnNames.forEach((column) => {
                    let th = document.createElement("th");
```

```
                        tr.appendChild(th);
                        th.textContent = column;
                    });
                }
                // 创建内容
                function createBody(table, bodyData) {
                    // tbody -> table
                    let tbody = document.createElement("tbody");
                    table.appendChild(tbody);

                    // tr -> tbody
                    bodyData.forEach((record) => {
                        let tr = document.createElement("tr");
                        tbody.appendChild(tr);

                        // td -> tr
                        Object.values(record).forEach((text) => {
                            let td = document.createElement("td");
                            tr.appendChild(td);
                            td.textContent = text;
                        });
                    });
                }
                let s = input.value;
                s = s.toLowerCase();
                init();
                while (p < s.length) {
                    let token = "";
                    while (s[p] == " " || s[p] == '\n' || s[p] == '\t') p++;
                    if (s[p] >= "a" && s[p] <= "z") {
                        token += s[p++];
                        while ((s[p] >= "0" && s[p] <= "9") || (s[p] >= "a" && s[p] <= "z") || s[p] == '_' || s[p] == '-') token += s[p++];
                        syn = 20;
                        for (let i = 0; i < keys.length; i++) {
                            if (token === keys[i].name) {
                                syn = keys[i].id;
                                break;
                            }
                        }
                    } else if (s[p] >= '0' && s[p] <= '9') {
                        token += s[p++];
                        while (s[p] >= '0' && s[p] <= '9') token += s[p++];
```

```
                        syn = 21;
                    } else switch (s[p]) {
                        case '=': token += s[p++], syn = 11;
                            break;
                        case ',': token += s[p++], syn = 12;
                            break;
                        case ';': token += s[p++], syn = 13;
                            break;
                        case '(': token += s[p++], syn = 14;
                            break;
                        case ')': token += s[p++], syn = 15;
                            break;
                        case '\'': token += s[p++], syn = 16;
                            break;
                        case '*': token += s[p++], syn = 17;
                            break;
                        default: syn = -1;
                    }
                    if (syn == -1) {
                        res = "词法错误!\n";
                        // console.log("Error!");
                        break;
                    } else {
                        res += `[${syn}, ${token}]`;
                        res += '\n';
                        // console.log(`[${syn}, ${token}]`);
                        tokens[q++] = { "syns": syn, "token": token };
                    }
                }
                output1.innerHTML = res;
                res = "";
                if (syn == -1) {
                    res = "词法错误!\n";
                } else {
                    t = 0;
                    if (tokens[t].syns == 1) {
                        //SELECT * FROM GOODSLIST where id = '1001'
                        t++;
                        if (tokens[t].syns == 17) {
                            t++;
                            if (tokens[t++].syns == 2) {
                                select_from(t);
                            } else {
```

```
                                res = "*后面语法出错!\n";
                            }
                        } else if (tokens[t].syns == 20) {
                            while (tokens[t].syns === 20) {
                                Head.push(tokens[t].token);
                                t++;
                                if (tokens[t].syns === 2) {
                                    t++;
                                    select_from(t);
                                    break;
                                } else if (tokens[t].syns === 12) {
                                    t++;
                                    if (tokens[t].syns !== 20) {
                                        res = `${tokens[t - 1].token}` + "后面出错!\n";
                                        break;
                                    }
                                } else {
                                    res = `${tokens[t - 1].token}` + "后面出错!\n";
                                    break;
                                }
                            }
                        } else {
                            res = "select 后面语法出错!\n";
                        }
                    } else if (tokens[t].syns == 3) {
                        //INSERT INTO Persons (LastName, Address) VALUES ('Wilson', 'Champs-Elysees')
                        t++;
                        if (tokens[t].syns == 4) {
                            t++;
                            table_name = tokens[t].token;
                            if (tokens[t].syns == 20) {
                                judge_table();
                                t++;
                                if (tokens[t].syns == 14) {
                                    t++;
                                    if (tokens[t].syns == 20) {
                                        while (tokens[t].syns == 20) {
                                            t++;
                                            if (tokens[t].syns == 12) {
                                                t++;
```

```
                                            if (tokens[t].syns != 20) {
                                                res = `${tokens[t - 1].token}` + "后面语法错误!\n";
                                                break;
                                            }
                                        } else if (tokens[t].syns == 15) {
                                            t++;
                                            if (tokens[t].syns == 5) {
                                                t++;
                                                //console.log(t);
                                                insert_values();
                                            } else {
                                                res = ")后面语法错误!\n";
                                            }
                                            break;
                                        } else {
                                            res = `${tokens[t - 1].token}` + "后面语法错误!\n";
                                            break;
                                        }
                                    }
                                } else {
                                    res = "(后面语法错误!\n";
                                }
                            } else if (tokens[t].syns == 5) {
                                t++;
                                insert_values();
                            } else {
                                res = `${tokens[t - 1].token}` + "后面语法错误!\n";
                            }
                        } else {
                            res = "into 后面语法错误!\n";
                        }
                    } else {
                        res = "insert 后面语法错误!\n";
                    }

                } else if (tokens[t].syns == 22) {
                    //DELETE FROM Person WHERE LastName = 'Wilson'
                    t++;
                    if (tokens[t].syns == 2) {
                        t++;
```

```
                            table_name = tokens[t].token;
                            select_from(t);
                        } else {
                            res = "delete 后面语法错误!\n";
                        }

                    } else if (tokens[t].syns == 6) {
                        // UPDATE Person SET FirstName = 'Fred' WHERE LastName = 'Wilson'
                        t++;
                        if (tokens[t].syns == 20) {
                            table_name = tokens[t].token;
                            // console.log(tokens[t].token);
                            judge_table();
                            t++;
                            if (tokens[t].syns == 7) {
                                t++;
                                if (tokens[t].syns == 20) {
                                    while (tokens[t].syns == 20) {
                                        select_where();
                                        if (res != "词法，语法正确!\n") {
                                            break;
                                        }
                                        t++;
                                        if (t >= tokens.length) {
                                            res = "词法，语法正确!\n";
                                            break;
                                        } else if (tokens[t].syns == 8) {
                                            t++;
                                            select_where();
                                            break;
                                        } else if (tokens[t].syns == 12) {
                                            t++;
                                            if (tokens[t].syns != 20) {
                                                res = ",后面语法错误!\n";
                                                break;
                                            }
                                        } else {
                                            res = `${tokens[t - 1].token}` + "后面语法错误!\n";
                                        }
                                    }
                                } else {
```

```
                                if (res === "") {
                                    // console.log("hhhhh");
                                    res = "set 后面语法错误!\n";
                                }
                            }
                        } else {
                            res = `${tokens[t - 1].token}` + "后面语法错误!\n";
                        }
                    } else {
                        res = "update 后面语法错误!\n";
                    }
                } else if (tokens[t].syns == 9) {
                    t++;
                    if (tokens[t].syns == 10) {
                        t++;
                        if (tokens[t].syns == 20) {
                            table_name = tokens[t].token;
                            t++;
                            if (tokens[t].syns == 14) {
                                t++;
                                if (tokens[t].syns == 20) {
                                    while (tokens[t].syns == 20) {
                                        Head.push(tokens[t].token);
                                        // console.log(tokens[t].token);
                                        create_word();
                                        //console.log("res");
                                        // console.log(res);
                                        // console.log(tokens[t].token);
                                    }
                                }
                                if (res == "") {
                                    if (tokens[t].syns == 15) {
                                        res = "词法，语法正确!\n";
                                    } else {
                                        res = "(后面语法错误\n";
                                    }
                                }
                            } else {
                                res = `${tokens[t - 1].token}` + "后面语法错误\n";
                            }
                        } else {
```

```
                                res = "table 后面语法错误\n";
                            }
                        } else {
                            res = "create 后面语法错误\n";
                        }
                    } else {
                        res = "操作符错误!\n";
                    }
                }

                //数据库操作
                hash_Head();
                // console.log("Hello World!");
                // console.log(table_name);
                output2.innerHTML = res;
                if (res == "词法，语法正确!\n") {
                    console.log(s);
                    this.$axios({
                        method:"post",
                        url:"http://localhost:3030/sql", // sql 接口
                        data: {"sql":s, "table_name":table_name},
                    }).then((response)=>{
                        let len=response.data.length;
                        let Productsinfo
                        for(let i=0;i<len;i++){
                        // console.log("data:",response.data[i])
                        Productsinfo=response.data[i];
                        this.bodyData.push(Productsinfo)
                        }
                    }).catch(function (error) {
                        // 处理错误情况
                        console.log(error);
                        //  this.msg1="error";
                    })
                    setTimeout(()=>{
                        //这里就写你要执行的语句即可，先让数据库的数据加载进去数组中你在从数组中取值就好了
                        createTable(Head, this.bodyData)
                    },800)
                }

            }
        },
```

```
    }
    </script>
```

9.11 网页整体样式

```
<template>
        <div id="app">
          <router-view />
        </div>

      </template>

      <script>
      export default {
        name: "App",
      };
      </script>

      <style>
      #app {
        font-family: "Avenir", Helvetica, Arial, sans-serif;
        -webkit-font-smoothing: antialiased;
        -moz-osx-font-smoothing: grayscale;
        text-align: center;
        color: #2c3e50;
        margin-top: 60px;
      }

      /* 标题样式 */
      .title{
        font-size: 40px ;
        color:rgb(255, 255, 255);
        font-family:kaiti;
        margin-top: 0px ;
        margin-bottom:0px;
        /* background-color:rgb(188, 223, 145); */
        height:100%;
        background: -webkit-linear-gradient(left, #25c481, #25b7c4);
        background: linear-gradient(to right, #25c481, #25b7c4);
      }

      /* -----------------------导航栏样式--------------------- */
      ul {
```

```
        /* top:-50; */
        list-style-type: none;
        margin-top:-106px;
        margin-left: -10px;
        padding: 0;
        width: 7%;
        background-color: rgb(255, 255, 255);
        position: fixed;
        height: 130%;
        overflow: auto;
    }

    li a {
        font-family:微软雅黑;
        font-size:20px;
        display: block;
        color: #000;
        padding: 8px 16px;
        text-decoration: none;
    }

    li a.active {
        background-color: #4CAF50;
        color: white;
    }

    li a:hover:not(.active) {
        background-color: rgb(188, 223, 145);
        color: white;
    }

    .btn2{
        background-color: #4CAF50;
        color: white;
        width: 200px;
        padding: 12px;
        font-size:18px;
        border: none;
        cursor: pointer;
        display: flex;
        margin-top:0;
        margin-left:20%;
        float:left;
```

```
        text-align:center;
    }

    /* -----------------------导航栏样式--------------------- */

    /* 搜索框样式 */
    .search{
        padding: 4px;
        text-align:center;
        margin-top:30px;
        float:left;
    }

    /*-------------------------- 下拉菜单样式--------------------------- */
    .dropbtn {
        background-color: #f1f1f1;
        color: black;
        width: 120px;
        padding: 12px;
        font-size:18px;
        border: white;
        cursor: pointer;
        display: flex;
        margin-top:-10%;
        margin-left:70;
        justify-content:center;
    }

    .dropdown {
        position: relative;
        display: inline-block;
        margin-left:100px;
        text-align:center;
        float:left;
    }
    .dropdown2 {
        position: relative;
        display: inline-block;
        margin-left:0px;
        text-align:center;
        float:left;
    }

```

```
.dropdown-content {
    display: none;
    position: absolute;
    background-color: #f9f9f9;
    min-width: 160px;
    box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
}

.dropdown-content a {
    color: black;
    padding: 12px 16px;
    text-decoration: none;
    display: block;
}

.dropdown-content a:hover {background-color: #f1f1f1}

.dropdown:hover .dropdown-content {
    display: block;
}

.dropdown:hover .dropbtn {
    background-color: rgb(188, 223, 145);
}

.dropdown2:hover .dropdown-content {
    display: block;
}

.dropdown2:hover .dropbtn {
    background-color: rgb(188, 223, 145);
}

/*--------------------------- 总体表格统一样式------------------------ */
table{
  display:blocks;
  text-align: center;
  width: 90%;
  margin-left:160px;
  border-collapse: collapse;
  border: 0;
}
.tbl-header{
```

```
      background-color: rgba(255,255,255,0.3);
     }
    .tbl-content{
      height:300px;
      overflow-x:auto;
      margin-top: 0px;
      border: 1px solid rgba(255,255,255,0.3);
    }
    th{
      padding: 20px 15px;
      text-align: center;
      font-weight: 500;
      font-size: 18px;
      color: #fff;
      text-transform: uppercase;
    }
    td{
      padding: 15px;
      text-align: center;
      vertical-align:middle;
      font-weight: 300;
      font-size: 18px;
      color: #fff;
      border-bottom: solid 1px rgba(255,255,255,0.1);
    }
    @import url(https://fonts.googleapis.com/css?family=Roboto:400,500,300,700);
    body{
      background: -webkit-linear-gradient(left, #25c481, #25b7c4);
      background: linear-gradient(to right, #25c481, #25b7c4);
      font-family: 'Roboto', sans-serif;
    }
    section{
      margin: 50px;
    }
    /* follow me template */
    .made-with-love {
      margin-top: 40px;
      padding: 10px;
      clear: left;
      text-align: center;
      font-size: 10px;
      font-family: arial;
```

```
      color: #fff;
    }
    .made-with-love i {
      font-style: normal;
      color: #F50057;
      font-size: 14px;
      position: relative;
      top: 2px;
    }
    .made-with-love a {
      color: #fff;
      text-decoration: none;
    }
    .made-with-love a:hover {
      text-decoration: underline;
    }

    /* for custom scrollbar for webkit browser*/
    ::-webkit-scrollbar {
        width: 6px;
    }
    ::-webkit-scrollbar-track {
        -webkit-box-shadow: inset 0 0 6px rgba(0,0,0,0.3);
    }
    ::-webkit-scrollbar-thumb {
        -webkit-box-shadow: inset 0 0 6px rgba(0,0,0,0.3);
    }
    /*----------------------------- 新样式------------------------- */

    /* 按钮样式 1 */
    .button1 {
      margin: 10px;
      text-align: center;
      background-color:transparent;
      /* border:none; */
      border-color:rgb(197, 255, 172);
      color:white;
      font-size:16px;
      width:90px;
    }

    /* 按钮样式 2 */
    .button2 {
```

```
      margin: 10px;
      text-align: center;
      background-color:transparent;
      /* border:none; */
      border-color:rgb(197, 255, 172);
      color:white;
      font-size:16px;
      width:70px;
    }

    h3,
    h4 {
      font-weight: normal;
    }

    .frame {
      width: 90%;
      margin: 40px auto;
      text-align: center;
    }

    button {
      margin: 20px;
      text-align: center;
    }

    </style>
```

十、 具体贡献

二叉树存储数据	李正阳
dijkstra 算法	张力群
数据表设计	周琨瑜 齐秋逸
VUE 框架	刘振强 张力群 周琨瑜
文件系统实现数据库	李正阳 齐秋逸
产生多进程访问和互斥操作	刘振强 李正阳

数据库中读取数据	齐秋逸 刘振强
数据库中写入数据	周琨瑜 张力群
词法分析器	李正阳
语法分析器	齐秋逸 张力群
语义分析器	李正阳 周琨瑜 刘振强
前后端交互接口（增删改查）	齐秋逸，周琨瑜，张力群，刘振强
SQL 集成 对应接口	张力群，李正阳
网页整体样式	刘振强，李正阳，齐秋逸
CSS 样式渲染	齐秋逸，刘振强
网站框架和逻辑设计	张力群，周琨瑜
代码整合和纠错	张力群，周琨瑜，齐秋逸，刘振强
视频录制	李正阳，周琨瑜
报告书写	张力群，齐秋逸，刘振强，李正阳，周琨瑜

十一、 源代码地址：

https://github.com/wenxilzy/Intelligent-logistics-query-system