之前做的是一些中车的数据清洗之类的

3.19superset 0.36 从源码搭建到添加Chart类型

源码搭建

源码下载与基础环境配置

- python3.7
- Anaconda3
- superset0.36
- Node.js

【源码下载链接】（GitHub - apache/superset: Apache Superset is a Data Visualization and Data Exploration Platform）

开启cmd使用Anaconda配置名为superset的环境，设置py=3.7并将下载好的源码整个解压到evns/superset/Lib/site-packages中。

```
conda create -n superset python=3.7 //创建新环境
conda activate superset  //激活环境
conda deactivate  //退出当前环境
```

后端环境搭建

激活superset虚拟环境，利用cmd进入site-packages/superset-0.36。安装相应的py依赖包。然后打开superset-0.36下的superset后端服务文件夹。打开config.py文件修改默认数据接口为Mysql，如下图所示，注意URL中不可夹带中文字符且末尾标注编码格式为utf-8，否则初始化数据库并加载官方示例时会报错。

```
# Your App secret key
SECRET_KEY = (
    "\2\1thisismyscretkey\1\2\e\y\y\h"  # pylint: disable=anomalous-backslash-in-string
)

# The SQLAlchemy connection string.
#SQLALCHEMY_DATABASE_URI = "sqlite:///" + os.path.join(DATA_DIR, "superset.db")
SQLALCHEMY_DATABASE_URI = 'mysql://root:lxyroot@127.0.0.1/superset-test?charset=utf8'
# SQLALCHEMY_DATABASE_URI = 'postgresql://root:password@localhost/myapp'
```

最后通过cmd进入superset/bin初始化数据库，新建管理用户并运行加载示例启动后端服务器。

```
cd /d .../site-packages/superset-0.36
pip install -r requirements.txt
//安装依赖包，通过txt批量安装的包可能部分存在问题，卸载对应的出现问题的包重新安装即可
//别忘了修改对应的config.py文件数据库URL
pip install apache-superset==0.36 //一定要和下载的包对应
cd /d .../superset-0.36/superset/bin
python superset db upgrade //初始化数据库要以python开头不然无法识别
set FLASK_APP=superset //初始化APP
flask fab create-admin //为APP创建管理用户
python superset load_examples //加载数据示例
python superset init //初始化管理用户权限
flask run //启动APP后端服务器
```

前端环境搭建

退出当前环境，利用cmd进入superset/superset-frontend前端配置文件夹，安装相应的依赖包，修改webpack.config.js文件夹中的include打包路径如下图所示。

```
{
  test: /\.jsx?$/,
  // include source code for plugins, but exclude node_modules within them
  exclude: [/superset-ui.*\/node_modules\//],
  include: [new RegExp(`${APP_DIR}/src`), /superset-ui.*\/src/,path.resolve(__dirname,'./src')],
  use: [babelLoader],
},
```

打包前端文件给后端，启动前端服务器，这样前端的调试环境就搭建好了

```
cd /d superset-0.36/superset-frontend //进入前端文件夹
npm install //安装前端依赖包，对应包信息存储在package.json当中
//修改打包路径
npm run dev //启动前端打包服务，实时传递前端文件修改信息
```

二次开发--添加新的Chart

本次二次开发的主要任务是添加新的chart type，任务可以分为两步

- 给前端添加相应的chart插件渲染逻辑
- 给后端添加相应的chart数据处理逻辑

因为superset插件的前端渲染逻辑几乎都已经集成到了名为superset-ui的包中，而此包的内部逻辑较为复杂，因此本次魔改参考的是位于superset-0.36\superset-frontend\src\visualizations下的两个还未添加到superset-ui中的chart。添加的chart类型是Mix_Line_bar（柱状折线图），引用依赖包Echarts以及d3。

下面是整体code修改的思维导图

添加一个新的chart需要的文件.nbmx

前端代码处理

1. 添加前端visualizations文件夹

要在前端文件夹superset-frontend\src\visualizations下建立以新chart命名的文件夹，文件夹结构如下图所示：

```
MixLineBar
    |--MixLineBar.js
    |--ReactMixLineBar.js
    |--TransformProp.js
    |--MixLineBarChartPlugin.js
    |--images
        |--thumbnail.png
        |--thumbnailLarge.png
```

其中images中存储的是一大一小两张图表缩略图；MixLineBar.js需要包含两部分内容：表单数据处理和图表渲染逻辑；ReactMixLineBar.js是向superset-ui包注册自己的MixLineBar.js文件；TransformProp.js获取后端数据并分类；MixLineBarChartPlugin.js则是从superset-ui继承的的插件类，包含chart说明，TransformProp.js分类的数据以及MixLineBar路径。

2. 在Presets中注册新图表的ChartPlugin

在文件superset-frontend\src\visualizations\presets\MainPreset.js的开头import之前添加的MixLineBarChartPlugin并且在末尾插入生成新的Chart需要的key值。

3. 在VizTypeControl中注册新图表的key值

文件路径为superset-frontend\src\explore\components\controls\VizTypeControl.jsx。VizTypeControl继承于React.PureComponent，主要作用是在重新渲染之前做浅层比较来达到只重新渲染发生变化的子组件，这一步只需要在用于比较图表类型的数组中添加新图表的key值即可。

4. 新增new chart的表单布局

在superset-frontend\src\explore\controlPanels中存储着所有chart的表单布局信息，用来收集用户提交的针对不同表单的数据查询和图表设置数据。分别对应了label: t('Query')和label: t('Chart Options') 。{t}引用于superset-ui的translations模块。

5. 新增表单组件

原有的表单组件可能无法收集到我们绘制图表的必要信息，可以自行在superset-frontend/src/explore/controls.jsx中可以导出重构的controls中修改或者添加组件。

6. 在setup中注册表单布局

修改superset-frontend\src\setup\setupPlugins.ts文件，在开头导入我们在controlPanels中导入的图表表单布局，并在末尾加入生成表单布局需要的key值。

7. 将依赖包写入packages.json

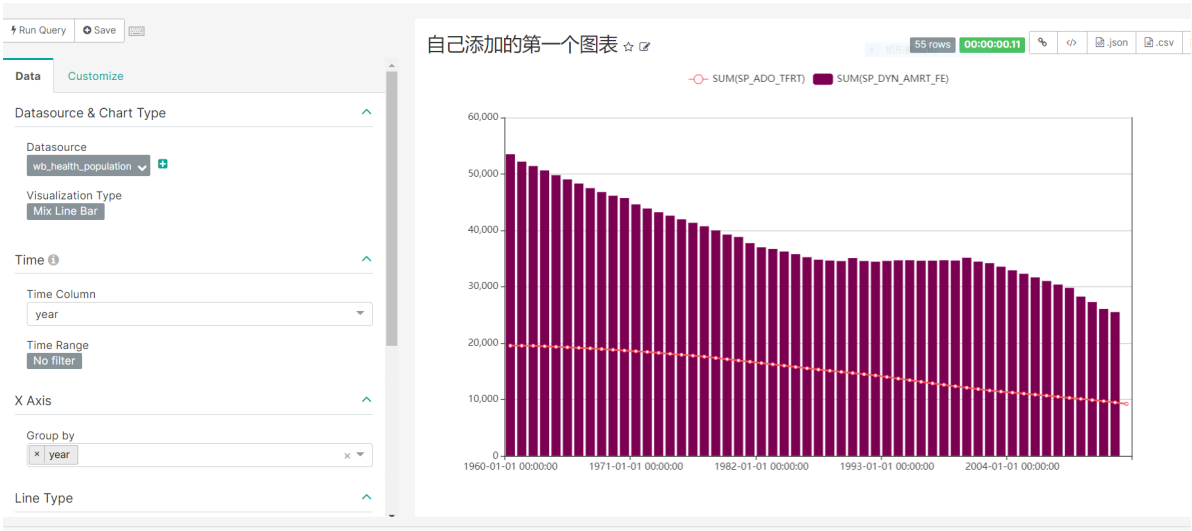
因为在图表渲染的时候我们使用了Echarts包，这个包不存在于原有的依赖包的列表当中（在node_modules中可以查看已经存在的依赖包），因此要在packages.json中加入对应的依赖名称和版本信息 "echarts": "^4.7.0" 。以便用npm install将其引入到node_modules中。

后端代码处理

后端viz.py文件包含了所有chart图表数据处理模式，我们只需要在该文件中新建我们的图表类继承BaseViz并且重写表单数据处理和查询 数据库返回数据处理逻辑即可，而这两部分的功能由BaseViz中的query_obj和get_data两个函数负责。所以只需要重写这两个函数即可。需要注意的是还要重写 viz_type与verbose_name 两个属性，前者是在数据回传前端中的key值，必须和我们前端定义的key值一致，后者是在页面显示的时候图表label。

调试结果

最终实现了在superset网站中新加chart的选项中可以选择之前添加的MixLineBar图生成相应的配置页面，并且在选择配置参数和查询参数后根据查询到的数据在网页右侧渲染出柱状折线图：



参考

> superset集成Echart

> supertset源码安装

> Superset (apache.org)

> MixLineBar - Examples - Apache ECharts

3.23 前端button更改

button调用路径（以chart为例）

ExploreChartHearder/ExploreActionButtons.jsx中引用了EmbedCodeButton.jsx-(这两个同在src/explore/components目录中)-->Amplification.jsx(自己加的这个放大按钮在src/components中)

结果示例：

探寻数据流

以添加新的查询语句界面左侧的组件渲染为例子看前端数据流是怎么获得的：

上图中整个模块在src/SqlLab/components/SqlEditorLefrBar.jsx中渲染，其中上半部分三条文本框组成的模块又在src/components/TableSelector.jsx中渲染由SELB调用,而第一个文本框数据库连接的选取又调用了src/components/AsyncSelect.jsx渲染，其中AS从服务器抓取数据的函数，它需要用的函数在TS中定义了两个dbMutator和onDatabaseChange前者接受查询数据并把数据转化成map列表的格式，后者返回链接ID和链接内的数据库名字列表。

接下来重点探究AS中调用的SupersetClient类（在superset-ui中的connection文件夹中）的get函数是如何链接数据库并返回数据的：

下面是get函数源码：

```
async get(requestConfig) {
  return this.request(_extends({}, requestConfig, {
    method: 'GET'
  }));
}
```

```
async request(_ref) {
  let {
    body,
    credentials,
    endpoint,
    headers,
    host,
    method,
    mode,
    parseMethod,
    postPayload,
    signal,
    stringify,
    timeout,
    url
  } = _ref;
  return this.ensureAuth().then(() => (0, _callApi.default)({
    body,
    credentials: credentials != null ? credentials : this.credentials,
    headers: _extends({}, this.headers, {}, headers),
    method,
    mode: mode != null ? mode : this.mode,
    parseMethod,
    postPayload,
    signal,
    stringify,
    timeout: timeout != null ? timeout : this.timeout,
    url: this.getUrl({
      endpoint,
      host,
      url
    })
  }));
}
```

```
function _extends() { _extends = Object.assign || function (target) { for (var i = 1; i < arguments.length; i++) { var source = argume
```

其中_extends是一个把所有可枚举属性的值从一个或多个源对象分配到目标对象的函数

综上，他在这个地方执行了异步请求拿到了一个用json表示的表单内容

4.9数据库数据流探索

前端数据源选择的请求与回传

neo4j-admin import --database neo4j --nodes=import\nodes_events.csv --nodes=import\nodes_channel.csv --nodes=import\nodes_things.csv --relationships=import\links.csv --ignore-extra-columns=true --ignore-empty-strings=true这个是把csv文件导入图数据库用的语句

重要的连接类ConnectorRegistry

定义在

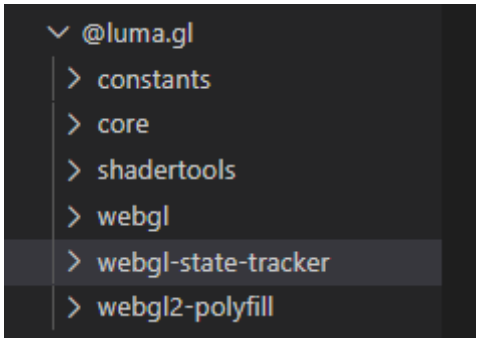
点击更换数据源的请求

请求 URL: http://127.0.0.1:5000/superset/datasources/

luma包多版本报错的解决方法

下面是superset原生的luma包文件夹，多余的是错误引入的高版本，直接给删咯

然后另起一个工程只下载deck.gl的高版本把包依赖中的luma给直接复制到本工程需要的高版本的luma文件夹中



4.13添加neo4j数据库

修改添加数据库处的test链接按钮

1.改写views/core.py 文件中关于url：“superset/testconn”的响应函数

2.在后端db_engine_specs文件夹中新建neo4j的引擎，engine = ‘http’对应make_url中的backend。

3.在后端superset\db_engine_specs\base.py中import py2neo

4.改写models/cores.py文件中的set_sqlalchemy_uri和get_sqla_engine两个函数

根据前端输入的sqla-url生成engine

4.14

testconn调试通过

全局搜索共23处调用了get_sqla_engine继续修改

解决不能保存的问题，初步观察是保存需要刷新schema的问题

5.修改了后端superset\models\bcore.py中的database类的inspector函数并且添加了database_kind判断函数（返回值是一个bool值）

6.在新建的neo4j.py的引擎中重写了父类的get_schema_names函数返回的是默认链接的图数据库的名字

以上修改的基本思路是GraphService代替engine 用GraphService.defauil_graph代替inspector 并用defauil_graph.name代替shcema_name

neo4j url链接”http://neo4j:asd134679852@127.0.0.1:7474“

添加执行cypher语句功能

4.15

添加sql lab中运行输入框中的cypher语句并回传json数据的功能

请求url：http://127.0.0.1:5000/superset/sql_json/ 响应写在views/core.py中

调用了superset文件夹中sql_lab.py文件中的get_sql_results函数，其又调用了

该文件中的execute_sql_statements函数（执行sql语句）

1.修改sql_lab.py文件中的execute_sql_statements函数，加入根据判断数据库决定调用了该文件中的execute_sql_statement函数还是我新加的execute_cypher_statement函数

2.在sql_lab.py文件中新加execute_cypher_statement函数处理cypher函数

3.前者返回的是Table类型的SupersetResultSet类（位于同级的result_set.py中），后者返回的是我自己在result_set.py中中添加的SupersetResultSet_neoj4类是json类型的数据

4.重写_serialize_and_expand_data函数，原先是把dataframe格式的数据转化成record格式的。现在在里面加入判断并引用自己写的list_to_dict()函数处理从去数据库取到的函数。

5.在superset/dataframe.py中创建list_to_dict()函数

6.重写db_engine_specs文件夹中的neo4j引擎加入处理游标缓存区的数据的功能。

在sql lab中添加显示neo4j数据库回传的json数据功能

4.19

解决sql_lab中的查询到的数据的显示问题，之前只能够显示table式的数据，但是现在返回的是一种dict格式的数据。

7.查询前端显示sql_lab react实现函数在前端文件夹/SqlLab/components/ResultSet.jsx中

8.修改后端的execute_sql_statements函数在相应中增加了一个“if_neo4j”字段用来让前端知道返回的数据应该是json还是list。

9.引用‘react-json-view’修改，前端文件夹里面的SqlLab/components/ResultSet.jsx中的ResultSet类中的render()函数

4.20

因为chart的Table形式的数据源和database是相互独立的因此下一步解决chart的Table形式的数据源的添加。而且SqlLab中的数据export到表格中似乎也行不通。

要重写query_obj 函数 和 sql中的 sqlTable类中的query函数

实现chart从图数据库读取数据

4.23

在前端的superset-frontend/src/explore/controls.jsx文件中添加表单组件，label_choose等等，可以输入label筛选节点。

4.25

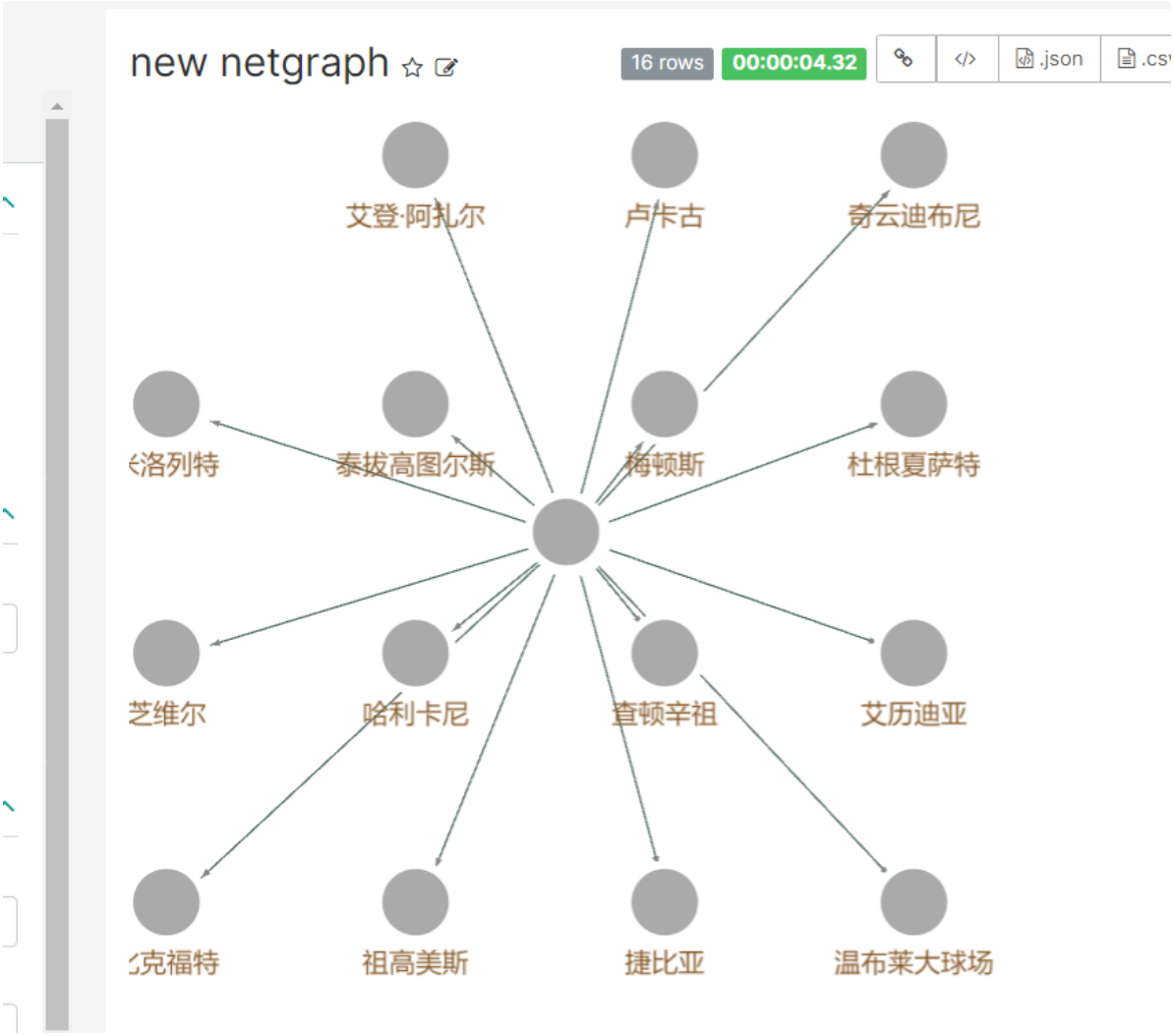
修改viz.py中的构成query函数（被重写的函数都写在了NetGraphViz类中）。

4.28

修改get_df函数中其调用的类database中的query（）函数以及众多嵌套调用的函数。

5.7

最终的实现形式如下图所示



完善chart界面的一些扩展功能

5.10

改写chart展示界面中的扩展内容：view query、view result 、view slampes（查看整个数据库数据，数据一多，响应时间贼长基本没用）等等

!(C:\Users\sdu20\Desktop\superset数据对比\截图的有用的数据图像\QQ图片20210520213918.png)

扩展内容前端渲染逻辑在前端文件夹中的src/explore/components/DisplayQueryButton.jsx文件中，修改这个即可修改前端的扩展显示内容。加入了根据viz_type判断应该渲染jsondata可视化还是Tabledata可视化

后端还要相应的添加根据数据库判断回传什么形式的数据，在最终处理函数在viz.py中，不然之前没有回传json数据的功能。

修改后的示例：

5.11

虚拟机配置druid数据库并用superset链接

5.14添加chart根据数据库的变化实时刷新的功能按钮

对于supert整个前端界面渲染的一些理解

前端整体采用react-redux框架，主要的store数据都写在了...Reducer中，包括其根据在...Actions中定义的各种类型的action该做出何种dispatch响应。

...Container意味着一个容器，里面包含各种小组件，这些组件都能够拿到store数据，并根据store数据的改变做出实时的热更新

修改Query/Stop button组并向其中添加新的button——flash

5.15

添加按钮允许用户选择实时刷新图表，

chart界面的Run Query/STOP 和save按钮在前端文件夹的src\explore\components\QueryAndSaveBtns.jsx中，渲染要求输入Onclick函数在superset-frontend\src\explore\components\ExploreViewContainer.jsx中 定义为onQuery（），实际执行的渲染逻辑则在superset-frontend\src\components\Button.jsx中

执行往store中插入查询结果（一些数据库名字之类的）的函数定义在superset-frontend\src\chart\chartAction.js中的runAnnotationQuery函数，这个被AnnotationLayerControl.jsx 调用并集成到ControlMap中，然后ControlMap导出为control被ControlPanelsContainer.jsx调用，这个文件又成为ExploreViewContainer.jsx的子组件，而ExploreViewContainer.jsx作为根组件被包在Provider的APP层中并被定义为热更新的。

执行往stroe中插入查询结果（渲染图表需要的数据）的函数定义在superset-frontend\src\chart\chartAction.js的 exploreJSON（）函数，这个函数被这个文件夹里的postChartFromData()函数调用，这个函数被同级./chart.js中的runQuery函数调用，这个函数在组件刷新的时候判断triggerQuery这个bool变量来决定是否被执行。前端点击runQuery按钮出发Onquery函数时改变的是triggerQuery这个bool变量。

store中的loading参数是用来表示是否在加载是否显示正在加载的图标

因此现在有两种修改方式对图表进行实时刷新:

1、新加button并设置onclick函数每隔一段时间就设定一次triggerQuery这个bool变量为真。

2、在superset-ui中添加ajax长链接请求，新加button并设置onclick函数触发action来触发这个长链接。

5.17

做出以下的代码修改：

在superset-frontend\src\explore\components\QueryAndSaveBtns.jsx中添加一个flash Button

在superset-frontend\src\explore\components\ExploreViewContainer.jsx中添加onFlash()函数，绑定action，anction函数onflash（）用来改变store中的flashQuery状态。

在src/chart/chart.jsx文件中添加执行轮询的函数runFlashQuery（这个函数调用了同级的轮询逻辑函数flashPostChartFromData）。并根据store中的flashQuery状态启用这个函数。

5.20 虚拟机链接本地代理，虚拟系统的网络设置代理，ip为本机的virtualbox ip（用ipconfig查看）端口为7890，cash的端口，要在cash中打开允许局域网链接

5.20 链接materialize

5.21-5.26实现了用kafka通过debezium链接mysql数据库

1.实现了用kafka通过debezium链接mysql数据库，kafka对外暴露的可消费topic采用avro序列化（原生的json格式不被materialize支持，从json格式的topic中创建的source只有一个data字段，存储的类似于mysql的日志，最多转换成jsonb的格式，这种格式不支持SQL查询，整合难度大且数据冗长）。

上述组件统一部署在虚拟机的docker当中，编写的启动配置存放在/home/lxy/桌面/mysql-kafka-avro/docker-compose-mysql-avro-connector.yaml。建立kafka与mysql的链接配置存放在mysql-kafka-avro/register-mysql-avro.json中。另外两个read.me文件存放的是启动docker容器的终端语句和启动materialize的终端语句，mzdata文件夹存放了materialize数据库的文件（该数据库中的source：customers我已经链接好MYSQL的customers表，可以在其基础上创建视图）

5.26-5.28实现superset与materialize相连。

2.将windows中的superset迁移到虚拟机当中的linux—ubuntu中，并实现superset与materialize相连。

相连的时候存在的问题：materialize对外暴露的是可查询的view，而superset在添加chart的table类数据源时检测不到视图的存在。

解决方案：在database类中添加函数if_materialize函数判断要读取table的数据库是不是materialize数据库，是就直接用show sources+show views两个SQL语句查询table。（materialize对sqla的支持很浅只能这么做）。在Database对象下面创建VIEWS_SOURCES类用作返回对象，这个对象将用于在添加table源的时候自动向table中填充columns。

5.29 编写数据库自动变换脚本

自动创建并更新MYsql数据库中的refresh_data表中数据的python程序要在docker-kafka-debezuim-mysql程序启动之后，materialize启动之前运行。

注意脚本在没关闭docker.的前提下二次启动会导致表被创建两次，materialize读取数据紊乱

6.1实现WebSocket全双工通信扩展

6.1