DB project 2

12010641 牛景萱

12012427 黄柯睿

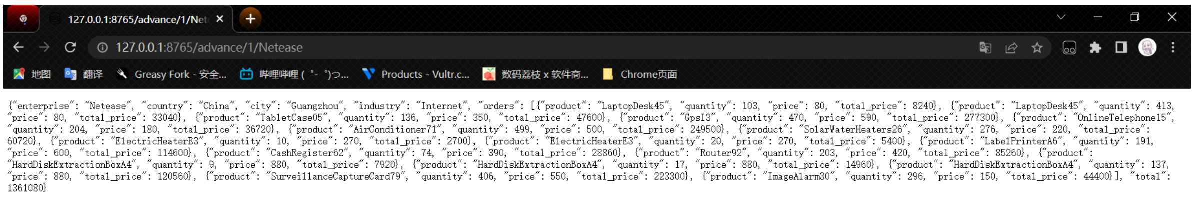
Bonus：

1. Design system functional requirements API.

1. Based on the actual requirements, we design the functions as follows

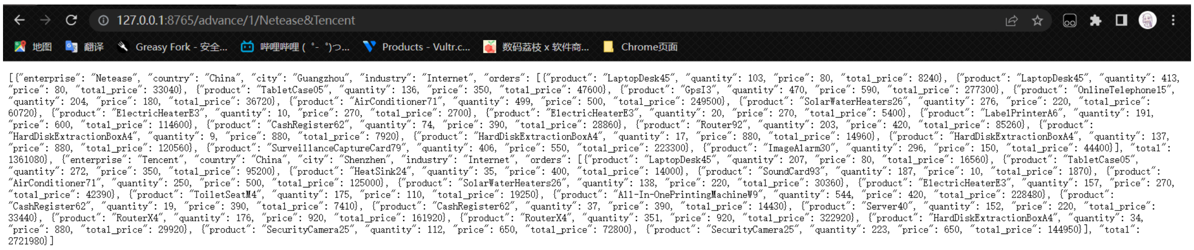
- get_enterprise_order : Retrieve and view all orders of a certain customer company, so that we can analyze customer needs and provide better services to customers.
- get_center_stock: Retrieve the model, stock, price and total sales of a supply center's products, so that the supply center can restock and adjust the price in time.

We show the result of running function execution through **RESTful** API, and change the URL to realize conditional query and multi-conditional query.



127.0.0.1:8765/advance/1/Netease

```
{"enterprise": "Netease", "country": "China", "city": "Guangzhou", "industry": "Internet", "orders": [{"product": "LaptopDesk45", "quantity": 103, "price": 80, "total_price": 8240}, {"product": "LaptopDesk45", "quantity": 413,
"price": 80, "total_price": 33040}, {"product": "TabletCase05", "quantity": 136, "price": 350, "total_price": 47600}, {"product": "GpsI3", "quantity": 470, "price": 590, "total_price": 277300}, {"product": "OnlineTelephone15",
"quantity": 204, "price": 180, "total_price": 36720}, {"product": "AirConditioner71", "quantity": 499, "price": 500, "total_price": 249500}, {"product": "SolarWaterHeaters26", "quantity": 276, "price": 220, "total_price":
60720}, {"product": "ElectricHeaterE3", "quantity": 10, "price": 270, "total_price": 2700}, {"product": "ElectricHeaterE3", "quantity": 20, "price": 270, "total_price": 5400}, {"product": "LabelPrinterA6", "quantity": 191,
"price": 600, "total_price": 114600}, {"product": "CashRegister62", "quantity": 74, "price": 390, "total_price": 28860}, {"product": "Router92", "quantity": 203, "price": 420, "total_price": 85260}, {"product":
"HardDiskExtractionBoxA4", "quantity": 9, "price": 880, "total_price": 7920}, {"product": "HardDiskExtractionBoxA4", "quantity": 17, "price": 880, "total_price": 14960}, {"product": "HardDiskExtractionBoxA4", "quantity": 137,
"price": 880, "total_price": 120560}, {"product": "SurveillanceCaptureCard79", "quantity": 406, "price": 550, "total_price": 223300}, {"product": "ImageAlarm30", "quantity": 296, "price": 150, "total_price": 44400}], "total":
1361080}
```



127.0.0.1:8765/advance/1/Netease&Tencent

```
[{"enterprise": "Netease", "country": "China", "city": "Guangzhou", "industry": "Internet", "orders": [{"product": "LaptopDesk45", "quantity": 103, "price": 80, "total_price": 8240}, {"product": "LaptopDesk45", "quantity": 413,
"price": 80, "total_price": 33040}, {"product": "TabletCase05", "quantity": 136, "price": 350, "total_price": 47600}, {"product": "GpsI3", "quantity": 470, "price": 590, "total_price": 277300}, {"product": "OnlineTelephone15",
"quantity": 204, "price": 180, "total_price": 36720}, {"product": "AirConditioner71", "quantity": 499, "price": 500, "total_price": 249500}, {"product": "SolarWaterHeaters26", "quantity": 276, "price": 220, "total_price":
60720}, {"product": "ElectricHeaterE3", "quantity": 10, "price": 270, "total_price": 2700}, {"product": "ElectricHeaterE3", "quantity": 20, "price": 270, "total_price": 5400}, {"product": "LabelPrinterA6", "quantity": 191,
"price": 600, "total_price": 114600}, {"product": "CashRegister62", "quantity": 74, "price": 390, "total_price": 28860}, {"product": "Router92", "quantity": 203, "price": 420, "total_price": 85260}, {"product":
"HardDiskExtractionBoxA4", "quantity": 9, "price": 880, "total_price": 7920}, {"product": "HardDiskExtractionBoxA4", "quantity": 17, "price": 880, "total_price": 14960}, {"product": "HardDiskExtractionBoxA4", "quantity": 137,
"price": 880, "total_price": 120560}, {"product": "SurveillanceCaptureCard79", "quantity": 406, "price": 550, "total_price": 223300}, {"product": "ImageAlarm30", "quantity": 296, "price": 150, "total_price": 44400}], "total":
1361080}, {"enterprise": "Tencent", "country": "China", "city": "Shenzhen", "industry": "Internet", "orders": [{"product": "LaptopDesk45", "quantity": 207, "price": 80, "total_price": 16560}, {"product": "TabletCase05",
"quantity": 272, "price": 350, "total_price": 95200}, {"product": "HeatSink24", "quantity": 35, "price": 400, "total_price": 14000}, {"product": "SoundCard93", "quantity": 187, "price": 10, "total_price": 1870}, {"product":
"AirConditioner71", "quantity": 250, "price": 500, "total_price": 125000}, {"product": "SolarWaterHeaters26", "quantity": 138, "price": 220, "total_price": 30360}, {"product": "ElectricHeaterE3", "quantity": 157, "price": 270,
"total_price": 42390}, {"product": "ToiletSeatM4", "quantity": 175, "price": 110, "total_price": 19250}, {"product": "All-In-OnePrintingMachineW9", "quantity": 544, "price": 420, "total_price": 228480}, {"product":
"CashRegister62", "quantity": 19, "price": 390, "total_price": 7410}, {"product": "CashRegister62", "quantity": 37, "price": 390, "total_price": 14430}, {"product": "Server40", "quantity": 152, "price": 220, "total_price":
33440}, {"product": "RouterX4", "quantity": 176, "price": 920, "total_price": 161920}, {"product": "RouterX4", "quantity": 351, "price": 920, "total_price": 322920}, {"product": "HardDiskExtractionBoxA4", "quantity": 34,
"price": 880, "total_price": 29920}, {"product": "SecurityCamera25", "quantity": 112, "price": 650, "total_price": 72800}, {"product": "SecurityCamera25", "quantity": 223, "price": 650, "total_price": 144950}], "total":
2721980}]
```

2. We have also designed a trigger.

- contract_type_trigger: when inserting order records into the orders table orders, it automatically completes the order status according to *estimate_delivery_date, lodgement_date* and the current system time *current_date*: finished / unfinished. unfinished. 3.

3. Considering that the company needs to adjust the listed products according to the market and market dynamics, we designed the trigger for **change** or **delete** the product table.

```
create trigger product_update_trigger
    before update
    on project_2.public.product
    for each row
execute procedure product_check();

create trigger product_delete_trigger
    before delete
    on project_2.public.product
    for each row
    execute procedure product_delete_check();
```

- product_update_check(): Ensure that when the product information in the product table is changed, the updated information is synchronized with the order and inventory tables.
- product_delete_check(): Ensure that when a product is deleted, the product information in the inventory table is also deleted.

2.Index:

Ⅰ. We try to create indexes on the product_model, contract_manager and salesman_num columns of orders for the following reasons.

1. product and staff are the "base tables" in the given situation, which are less involved in change operations and more frequently used as query conditions. 2.

The orders table is a dependent table of the staff and product tables. orders table's product_model, contract_manager and salesman_num columns are foreign key columns.

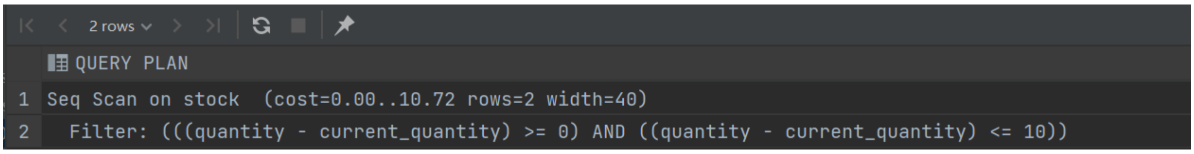
> If the slave table does not have an index containing a foreign key column, SQL Server needs to scan the entire slave table. The larger the slave table, the longer it takes to perform operations such as delete updates. It also tends to cause blocking in high concurrency situations. If the master table has uniquely aggregated or non-aggregated indexes, you can use the indexes of the master table to quickly locate them when inserting or modifying from the slave table.

3. Each order in orders, once formed and added to the order information, is not easily involved in update operations in reality.

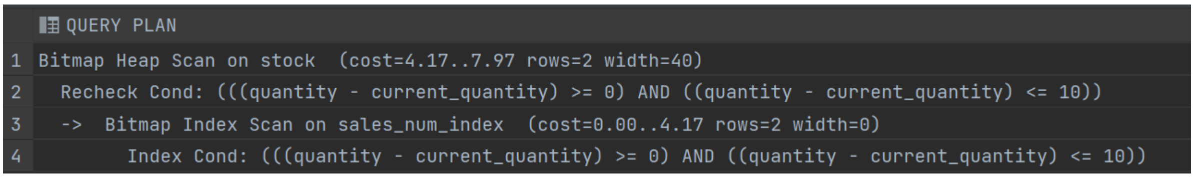
```
--btree 对文本模糊匹配表现更好
CREATE INDEX product_index ON project_2.public.orders USING
btree(product_model);
explain select * from  project_2.public.orders where product_model like
'Photo%';
--hash 适用于等值匹配
CREATE INDEX salesman_index ON project_2.public.orders using hash(salesman_num);
CREATE INDEX manager_index ON project_2.public.orders using
hash(contract_manager);
explain select * from project_2.public.orders where orders.salesman_num =
'11110405';
```

Ⅱ. For stock tables, create expression indexes to make it easier to search for products based on the number of sales.

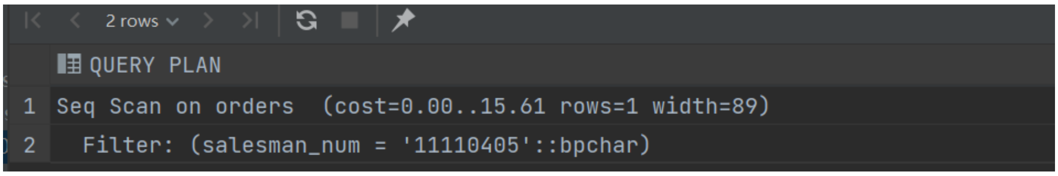
```
CREATE INDEX sales_num_index ON stock ((stock.quantity-stock.current_quantity));
explain select * from  stock where stock.quantity-stock.current_quantity between
0 and 10;
```



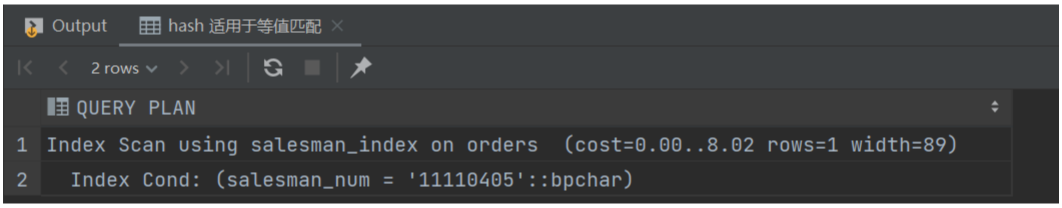
```
QUERY PLAN
Seq Scan on stock  (cost=0.00..10.72 rows=2 width=40)
  Filter: (((quantity - current_quantity) >= 0) AND ((quantity - current_quantity) <= 10))
```



```
QUERY PLAN
Bitmap Heap Scan on stock  (cost=4.17..7.97 rows=2 width=40)
  Recheck Cond: (((quantity - current_quantity) >= 0) AND ((quantity - current_quantity) <= 10))
  ->  Bitmap Index Scan on sales_num_index  (cost=0.00..4.17 rows=2 width=0)
        Index Cond: (((quantity - current_quantity) >= 0) AND ((quantity - current_quantity) <= 10))
```



```
QUERY PLAN
Seq Scan on orders  (cost=0.00..15.61 rows=1 width=89)
  Filter: (salesman_num = '11110405'::bpchar)
```

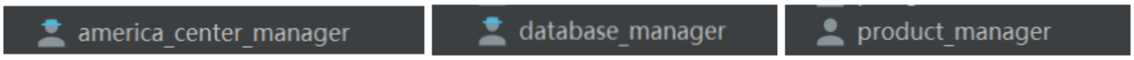


```
QUERY PLAN
Index Scan using salesman_index on orders  (cost=0.00..8.02 rows=1 width=89)
  Index Cond: (salesman_num = '11110405'::bpchar)
```

3.Role：

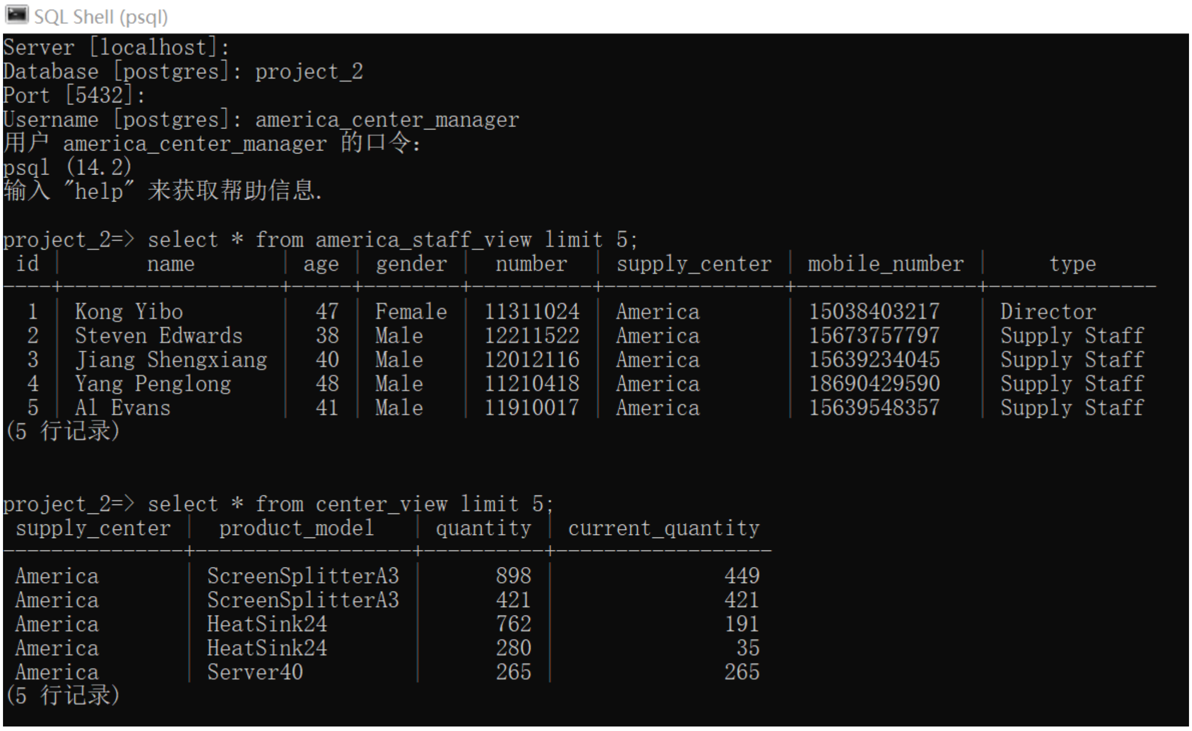
Based on the actual requirements, in addition to superuser postgres, we have created the following three new Roles

- product_manager, which manages product information, has all the permissions of the product table.
- database_manager that manages database instead of superuser: has the permission to create database and create users
- xxx_center_manager which manages the xxx supply center: has all permissions for the product inventory view of the supply center and the employee information view of the supply center



america_center_manager database_manager product_manager

America_center_manager example：



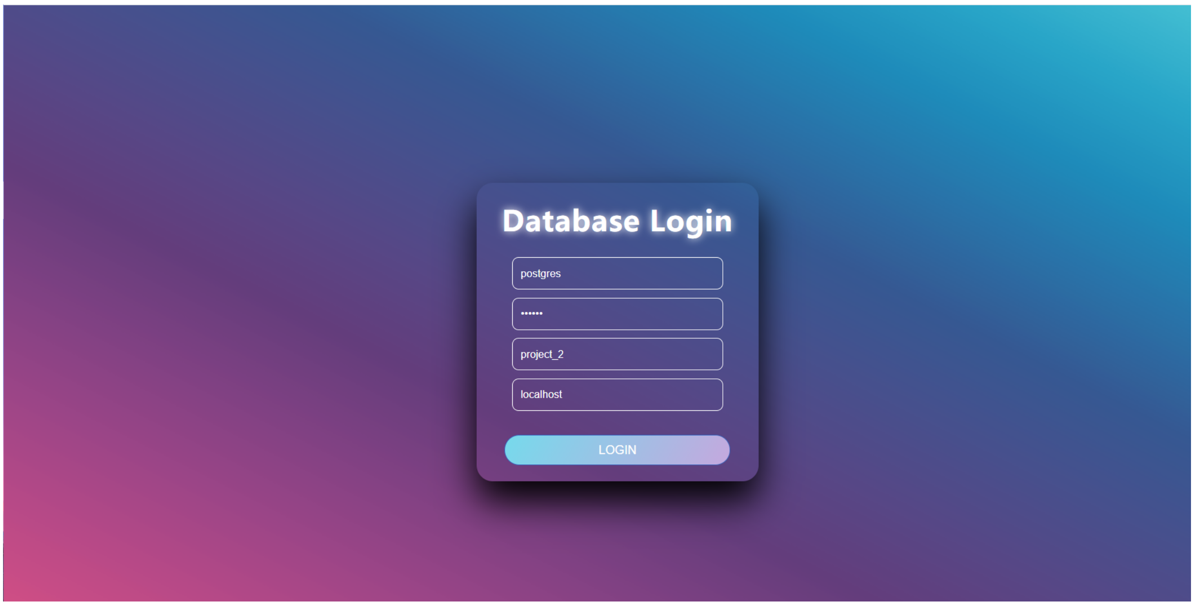
```
Server [localhost]:
Database [postgres]: project_2
Port [5432]:
Username [postgres]: america_center_manager
用户 america_center_manager 的口令:
psql (14.2)
输入 "help" 来获取帮助信息.

project_2=> select * from america_staff_view limit 5;
 id |        name         | age | gender |  number  | supply_center | mobile_number |     type
----+---------------------+-----+--------+----------+---------------+---------------+--------------
  1 | Kong Yibo           |  47 | Female | 11311024 | America       | 15038403217   | Director
  2 | Steven Edwards      |  38 | Male   | 12211522 | America       | 15673757797   | Supply Staff
  3 | Jiang Shengxiang    |  40 | Male   | 12012116 | America       | 15639234045   | Supply Staff
  4 | Yang Penglong       |  48 | Male   | 11210418 | America       | 18690429590   | Supply Staff
  5 | Al Evans            |  41 | Male   | 11910017 | America       | 15639548357   | Supply Staff
(5 行记录)


project_2=> select * from center_view limit 5;
 supply_center |  product_model   | quantity | current_quantity
---------------+------------------+----------+------------------
 America       | ScreenSplitterA3 |      898 |              449
 America       | ScreenSplitterA3 |      421 |              421
 America       | HeatSink24       |      762 |              191
 America       | HeatSink24       |      280 |               35
 America       | Server40         |      265 |              265
(5 行记录)
```

4. web

This front-end we mainly implement two pages, one is the login page, used for user authentication, and one is the database management page

The following are screenshots of the two pages.

login interface:



Database Login

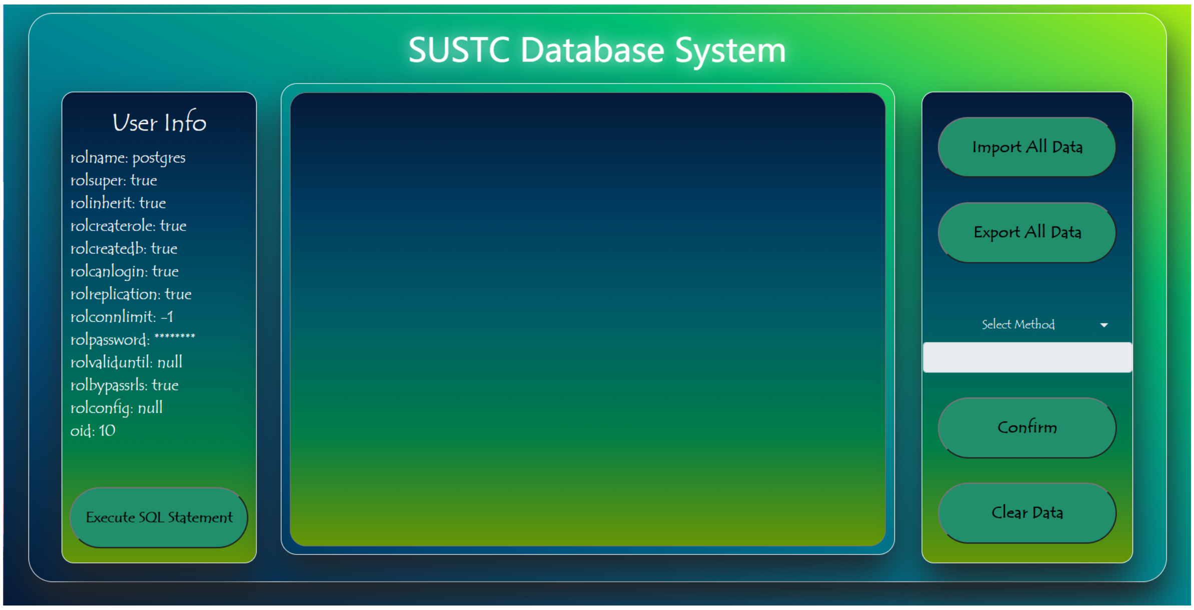
postgres

••••••

project_2

localhost

database interface：



SUSTC Database System

User Info

rolname: postgres
rolsuper: true
rolinherit: true
rolcreaterole: true
rolcreatedb: true
rolcanlogin: true
rolreplication: true
rolconnlimit: -1
rolpassword: ********
rolvaliduntil: null
rolbypassrls: true
rolconfig: null
oid: 10

During login verification, the user needs to enter four pieces of information: user name, password, database and host address. After the user enters the correct information, the backend will connect to the database and initialize the database connection pool for later use.

The operations supported by the front-end are.

1. one-click import operation
2. one-click export operation (save query results as txt files)
3. Q6-Q13 query operation
4. Q12-Q13 auto-completion (input part of the characters can automatically complete the rest of the characters)

5. panel clearing operation (only clear the front-end display, will not clear the back-end query results)
6. manually enter SQL statements and print the execution results to the workbench (if it is a select statement)

5.Database connection pooling

The connection pool we use for this database is PooledDB under the DBUtils package of python

```
pool = PooledDB(
    creator=pg,
    mincached=1,
    maxcached=20,
    blocking=True,
    port=5432,
    database='project_2',
    user='postgres',
    password='123456',
    host='localhost',
    ping=0
)
```

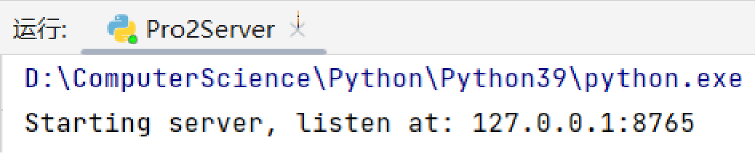
The initial connection is set to 1, the maximum connection is set to 20, and blocking is set to True

For a single server, we also implement a caching feature: that is, if a client repeatedly requests the same content from the server, it reads it from the cache instead of requesting a connection from the data. Caching greatly enhances the database's ability to handle highly concurrent single queries.

6. server

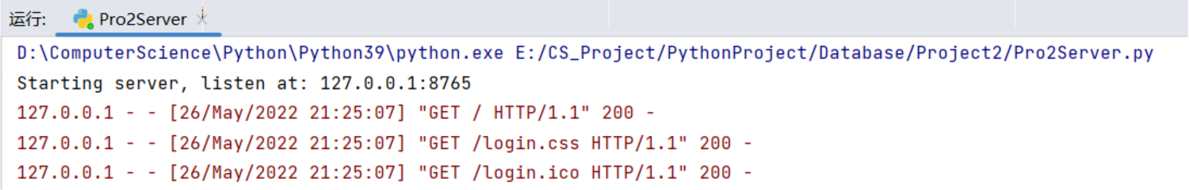
The backend supports http connection and RESTful service, http connection is used for front and backend interaction, RESTful service is used for some complex queries, and then the backend uses psycopg2 to connect to the database

The successful run screen looks like this.



```
D:\ComputerScience\Python\Python39\python.exe
Starting server, listen at: 127.0.0.1:8765
```

The result of processing the request when it is encountered is as follows.



```
D:\ComputerScience\Python\Python39\python.exe E:/CS_Project/PythonProject/Database/Project2/Pro2Server.py
Starting server, listen at: 127.0.0.1:8765
127.0.0.1 - - [26/May/2022 21:25:07] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [26/May/2022 21:25:07] "GET /login.css HTTP/1.1" 200 -
127.0.0.1 - - [26/May/2022 21:25:07] "GET /login.ico HTTP/1.1" 200 -
```

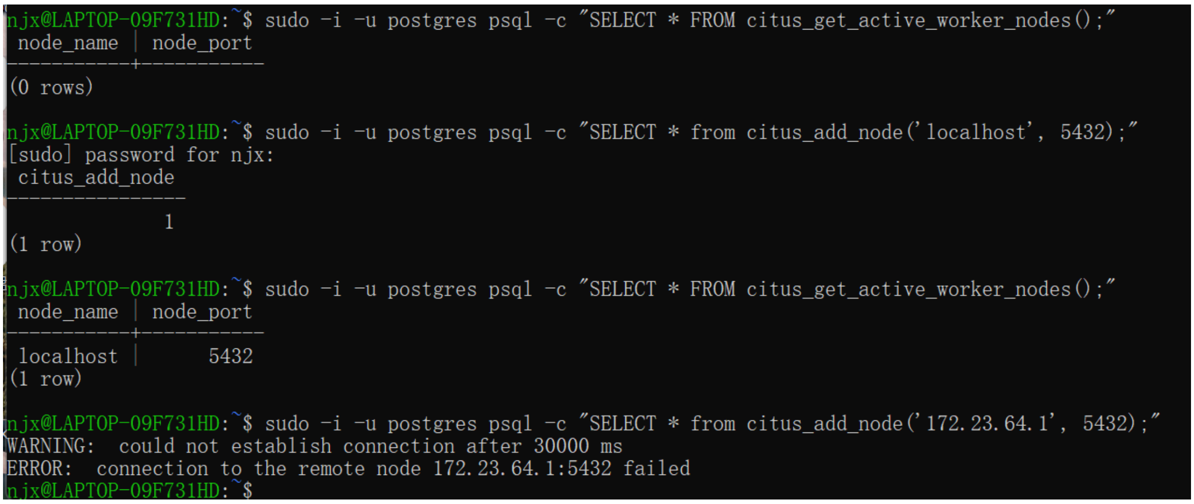
7. Server Distribution：

1. Using Citus, a distributed middleware for PostgreSQL databases

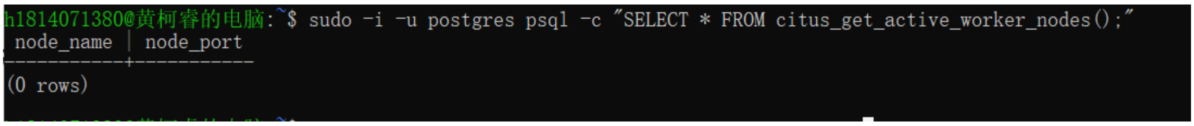
We first consider using Citus to solve the PostgreSQL horizontal scaling problem to support larger data volumes and greater write and query performance.

We deploy citus (with citus extensions installed) on two separate Linux systems (wsl) on two hosts with the add node bit local IP.

On each node, open port 5432.



```
njx@LAPTOP-09F731HD:~$ sudo -i -u postgres psql -c "SELECT * FROM citus_get_active_worker_nodes();"
 node_name | node_port
-----------+-----------
(0 rows)

njx@LAPTOP-09F731HD:~$ sudo -i -u postgres psql -c "SELECT * from citus_add_node('localhost', 5432);"
[sudo] password for njx:
 citus_add_node
----------------
              1
(1 row)

njx@LAPTOP-09F731HD:~$ sudo -i -u postgres psql -c "SELECT * FROM citus_get_active_worker_nodes();"
 node_name | node_port
-----------+-----------
 localhost |      5432
(1 row)

njx@LAPTOP-09F731HD:~$ sudo -i -u postgres psql -c "SELECT * from citus_add_node('172.23.64.1', 5432);"
WARNING:  could not establish connection after 30000 ms
ERROR:  connection to the remote node 172.23.64.1:5432 failed
njx@LAPTOP-09F731HD:~$
```



```
h18140713800黄柯睿的电脑:~$ sudo -i -u postgres psql -c "SELECT * FROM citus_get_active_worker_nodes();"
 node_name | node_port
-----------+-----------
(0 rows)
```

Since the IP of wsl is the local IP, and the local IP belongs to the campus network intranet IP, which is not a static IP, NAT network address translation and port mapping should be considered for deployment. It enables two hosts (servers) to communicate with each other, thus realizing the distribution. The specific implementation can be configured in docker, here we understand the concept without specific implementation.

2. Manual implementation of simple server distribution

Since the one above requires a public IP, or a cloud server, and since the network segment where we use wsl is our computer's subnet, not under the campus network's subnet, it is not possible to bind the campus network's IP, and the IP assigned to us by our campus network server is dynamically changed over time, it will be inconvenient to switch. So we finally gave up the above distributed design and we are going to implement our own distributed manually.

Our idea is to open a separate proxy.py as a proxy server, which maintains a list of servers, and when there is a query result, select a server from the server list to query and return the query result, and when there is a need to make changes (such as update, delete, etc.), all the servers need to be changed. The advantage of doing this is to reduce the pressure on the server caused by high concurrency and to avoid query failure caused by a single server going down. The IP bound to the proxy server is the campus network IP, so as long as the device and the proxy server are under the same subnet (i.e. campus network), they can be used as servers.

8. Stress test:

For this part of stress testing we use Jmeter software under Apache for stress testing. The software is written on java.

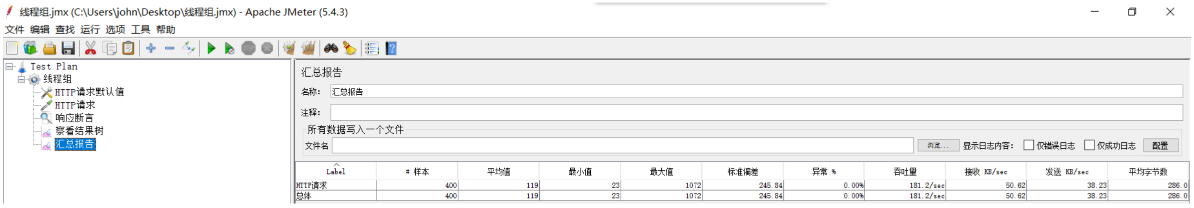
The final number of threads we chose to test was.

1server (no distribution): 400 threads, 600 threads, 800 threads, 1000 threads

2server: 1000 threads, 1200 threads, 1400 threads, 1600 threads, 1800 threads

The results at 1server are as follows:

400 threads：


线程组.jmx (C:\Users\john\Desktop\线程组.jmx) - Apache JMeter (5.4.3)
文件 编辑 查找 运行 选项 工具 帮助
Test Plan
线程组
HTTP请求默认值
HTTP请求
响应断言
察看结果树
汇总报告

汇总报告

名称：汇总报告

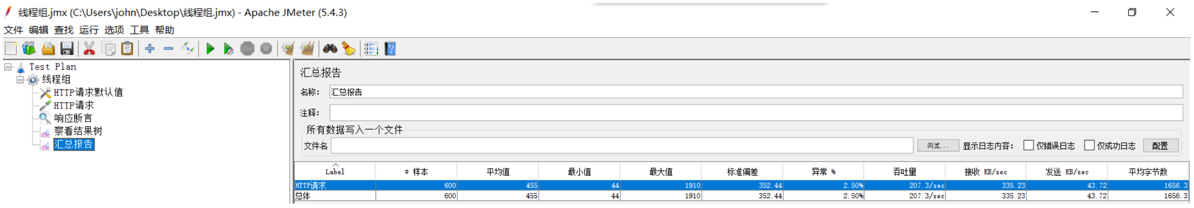
注释：

所有数据写入一个文件

文件名 浏览... 显示日志内容：□ 仅错误日志 □ 仅成功日志 配置

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	400	119	23	1072	245.84	0.00%	181.2/sec	50.62	38.23	286.0
总体	400	119	23	1072	245.84	0.00%	181.2/sec	50.62	38.23	286.0

600 threads：


线程组.jmx (C:\Users\john\Desktop\线程组.jmx) - Apache JMeter (5.4.3)
文件 编辑 查找 运行 选项 工具 帮助
Test Plan
线程组
HTTP请求默认值
HTTP请求
响应断言
察看结果树
汇总报告

汇总报告

名称：汇总报告

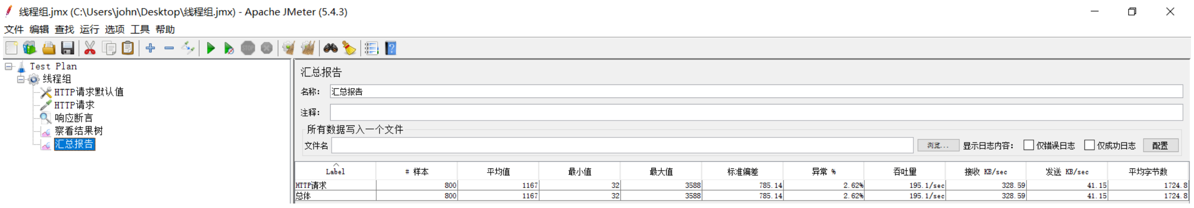
注释：

所有数据写入一个文件

文件名 浏览... 显示日志内容：□ 仅错误日志 □ 仅成功日志 配置

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	600	455	44	1910	352.44	2.50%	207.3/sec	335.23	43.72	1656.3
总体	600	455	44	1910	352.44	2.50%	207.3/sec	335.23	43.72	1656.3

800 threads：


线程组.jmx (C:\Users\john\Desktop\线程组.jmx) - Apache JMeter (5.4.3)
文件 编辑 查找 运行 选项 工具 帮助
Test Plan
线程组
HTTP请求默认值
HTTP请求
响应断言
察看结果树
汇总报告

汇总报告

名称：汇总报告

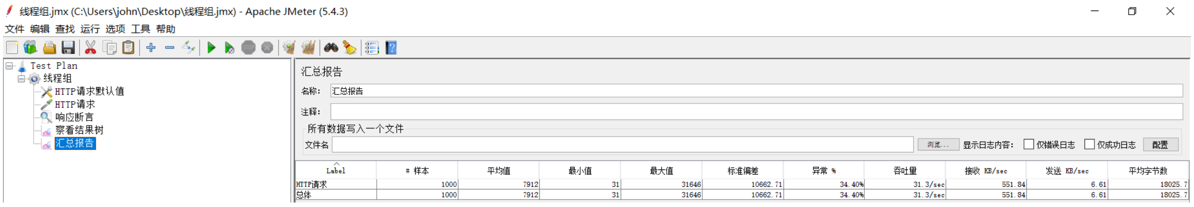
注释：

所有数据写入一个文件

文件名 浏览... 显示日志内容：□ 仅错误日志 □ 仅成功日志 配置

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	800	1167	32	3588	785.14	2.62%	195.1/sec	328.59	41.15	1724.8
总体	800	1167	32	3588	785.14	2.62%	195.1/sec	328.59	41.15	1724.8

1000 threads：


线程组.jmx (C:\Users\john\Desktop\线程组.jmx) - Apache JMeter (5.4.3)
文件 编辑 查找 运行 选项 工具 帮助
Test Plan
线程组
HTTP请求默认值
HTTP请求
响应断言
察看结果树
汇总报告

汇总报告

名称：汇总报告

注释：

所有数据写入一个文件

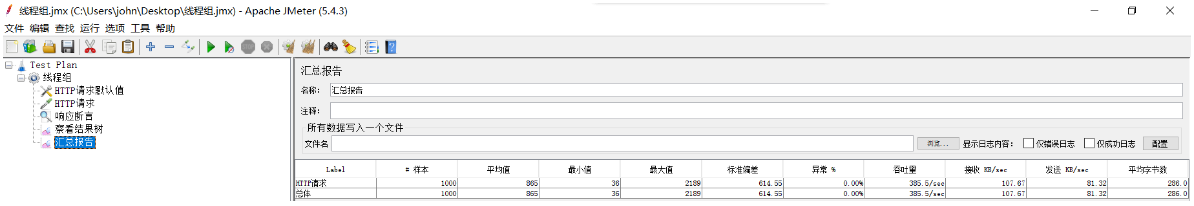
文件名 浏览... 显示日志内容：□ 仅错误日志 □ 仅成功日志 配置

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	1000	7912	31	31646	10662.71	34.40%	31.3/sec	551.84	6.61	18025.7
总体	1000	7912	31	31646	10662.71	34.40%	31.3/sec	551.84	6.61	18025.7

As you can see, from 600 threads onwards, the server's processing power starts to drop and packet loss starts to occur, and by 1000 threads, the server is already very laggy and eventually has a 34% exception

2server时结果如下：

1000 threads：


线程组.jmx (C:\Users\john\Desktop\线程组.jmx) - Apache JMeter (5.4.3)
文件 编辑 查找 运行 选项 工具 帮助
Test Plan
线程组
HTTP请求默认值
HTTP请求
响应断言
察看结果树
汇总报告

汇总报告

名称：汇总报告

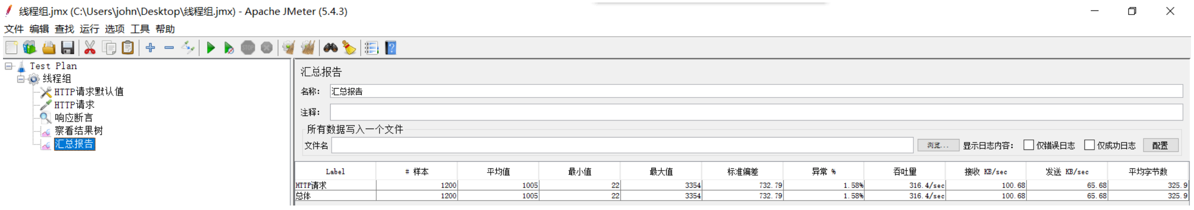
注释：

所有数据写入一个文件

文件名 浏览... 显示日志内容：□ 仅错误日志 □ 仅成功日志 配置

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	1000	865	36	2189	614.55	0.00%	385.5/sec	107.67	81.32	286.0
总体	1000	865	36	2189	614.55	0.00%	385.5/sec	107.67	81.32	286.0

1200 threads：


线程组.jmx (C:\Users\john\Desktop\线程组.jmx) - Apache JMeter (5.4.3)
文件 编辑 查找 运行 选项 工具 帮助
Test Plan
线程组
HTTP请求默认值
HTTP请求
响应断言
察看结果树
汇总报告
浏览...
配置

汇总报告

名称： 汇总报告

注释：

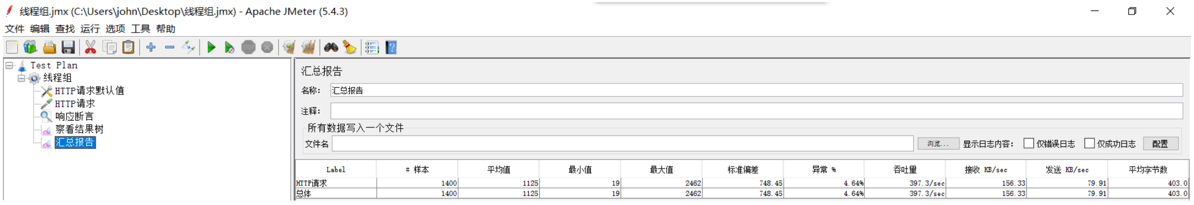
所有数据写入一个文件

文件名

显示日志内容： □ 仅错误日志 □ 仅成功日志

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	1200	1005	22	3354	732.79	1.58%	316.4/sec	100.68	65.68	325.9
总体	1200	1005	22	3354	732.79	1.58%	316.4/sec	100.68	65.68	325.9

1400 threads：


线程组.jmx (C:\Users\john\Desktop\线程组.jmx) - Apache JMeter (5.4.3)
文件 编辑 查找 运行 选项 工具 帮助
Test Plan
线程组
HTTP请求默认值
HTTP请求
响应断言
察看结果树
汇总报告
浏览...
配置

汇总报告

名称： 汇总报告

注释：

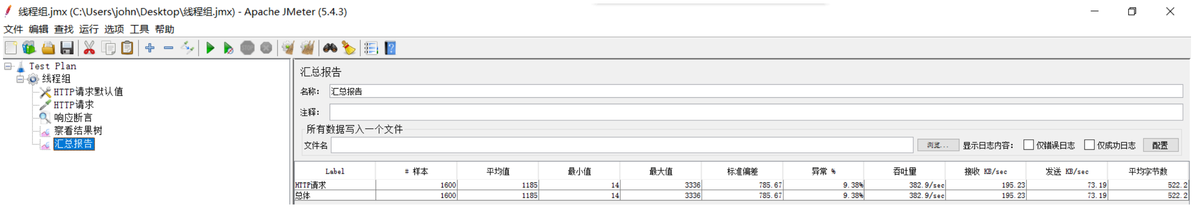
所有数据写入一个文件

文件名

显示日志内容： □ 仅错误日志 □ 仅成功日志

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	1400	1125	19	2462	748.45	4.64%	397.3/sec	156.33	79.91	403.0
总体	1400	1125	19	2462	748.45	4.64%	397.3/sec	156.33	79.91	403.0

1600 threads：


线程组.jmx (C:\Users\john\Desktop\线程组.jmx) - Apache JMeter (5.4.3)
文件 编辑 查找 运行 选项 工具 帮助
Test Plan
线程组
HTTP请求默认值
HTTP请求
响应断言
察看结果树
汇总报告
浏览...
配置

汇总报告

名称： 汇总报告

注释：

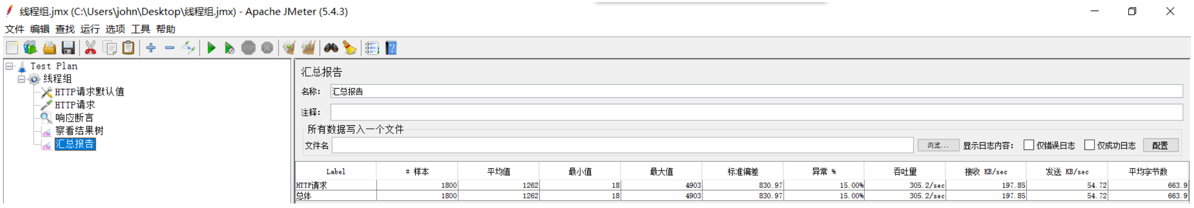
所有数据写入一个文件

文件名

显示日志内容： □ 仅错误日志 □ 仅成功日志

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	1600	1185	14	3336	785.67	9.38%	382.9/sec	195.23	73.19	522.2
总体	1600	1185	14	3336	785.67	9.38%	382.9/sec	195.23	73.19	522.2

1800 threads：


线程组.jmx (C:\Users\john\Desktop\线程组.jmx) - Apache JMeter (5.4.3)
文件 编辑 查找 运行 选项 工具 帮助
Test Plan
线程组
HTTP请求默认值
HTTP请求
响应断言
察看结果树
汇总报告
浏览...
配置

汇总报告

名称： 汇总报告

注释：

所有数据写入一个文件

文件名

显示日志内容： □ 仅错误日志 □ 仅成功日志

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	1800	1262	18	4903	830.97	15.00%	305.2/sec	197.85	54.72	663.9
总体	1800	1262	18	4903	830.97	15.00%	305.2/sec	197.85	54.72	663.9

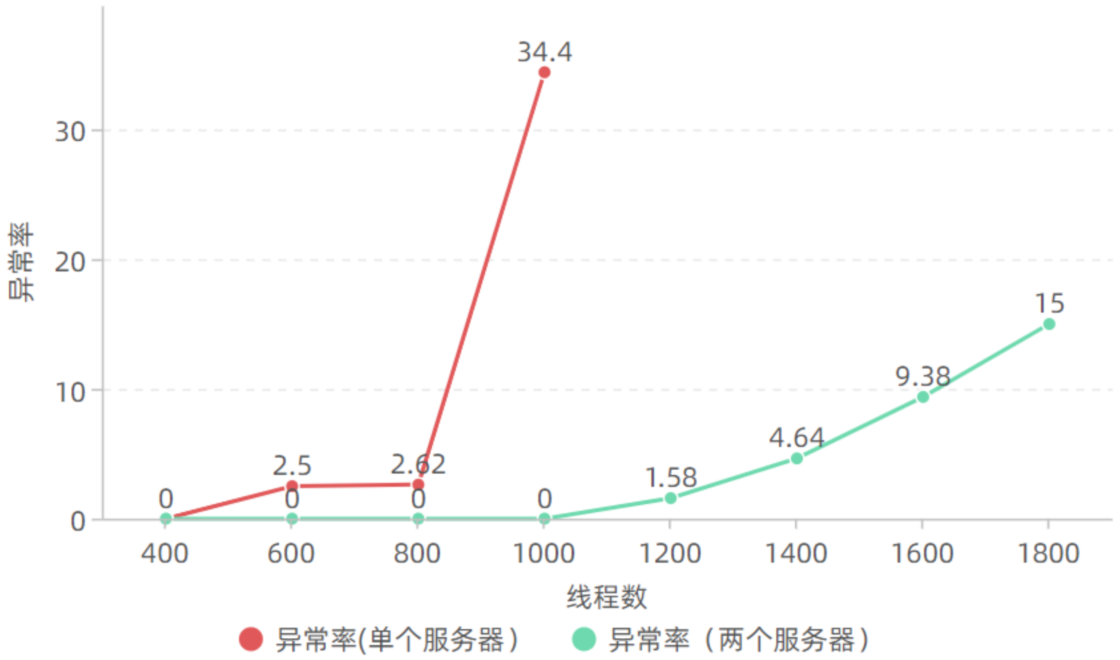
As you can see, with just two servers after using distributed, it is already easy to handle 1000 threads without errors.

The packet loss only starts at 1200 threads, and at 1800 threads there are only about 15% exceptions and no significant lag. You can see that the efficiency of distributed optimization for queries is exponentially increased

Here is a graph based on the exception rate.

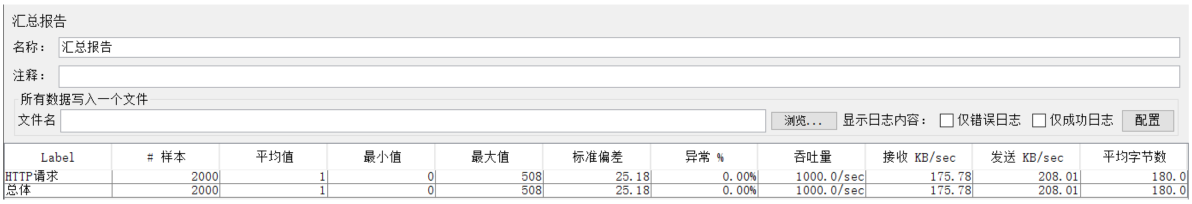
压力测试

单位：%



数据来源：自测

Extends: Add cache stress test (2000 threads) with no error.



汇总报告

名称： 汇总报告

注释：

所有数据写入一个文件

文件名

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 %	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
HTTP请求	2000	1	0	508	25.18	0.00%	1000.0/sec	175.78	208.01	180.0
总体	2000	1	0	508	25.18	0.00%	1000.0/sec	175.78	208.01	180.0