后端4

网络连接与健壮性测试

因为 tcp/ip 并没有在协议内部实现心跳检测, 极为依赖应用程序自己来实现心跳检测. 所以在 web 程序这种一般要 pre connect 的模式下, 一般是初始化后一直使用一条连接, 不同数据库驱动在连接池以及自动重连上的实现有所不同, 但是为了程序的健壮性, 一般都要考虑 auto reconnect 的问题.

一般使用 Socket 五元组确定一个"网络连接":

(protocol, source ip, source port, target ip, target port)

网络异常

- Timeout: 网络请求超时
- Connection refuse : 此端口没有服务绑定(bind)
- Connection reset : 连接被重置, 网络连接被 rst 包重置(非正常关闭)

查看网络连接

可以使用 ss 和tcpkill 来模拟"杀死"(RST)一个网络连接.

示例

Java redis 客户端驱动库 Jedis 的pub/sub连接不会自动重连:

https://www.bilibili.com/video/BV18dNqeZEQw?t=825.8

测试机器:

代码块

```
ssh wh@150.158.144.155
密码: fccdjny
```

SS

这是 iproute2 包中用于查看网络连接状态的命令.

ubuntu安装

代码块

```
1   sudo apt install iproute2
```

centos安装

```
代码块
1   sudo yum install iproute
```

查询 socket 连接信息

在一个webserver上查询连接到 443 端口的tcp网络连接

`ss -nt sport = :443`

查询本机的多个端口(80和443)

`ss -nt '( sport = :443 or sport = :80 )'`

查询本机网络连接的远程机器多个端口(80和443)

`ss -nt '( dport = :443 or dport = :80 )'`

另一种查询本地‘src’ (source)本地端口的写法, 注意和sport 的区别:

`ss -nt '( src :443 or src :80 )'`

查询网络连接另一端的连接

`ss -nt '( dst :443 or dst :80 )'`

查看指定网段连接

要查看与特定目标的活动连接，请定义包含 IP 地址或地址的表达式。例如，要查看 192.168.x.x 网络上的连接：

`ss dst 192.168/16`

杀死连接(RST)

`sudo ss -K -nt dst = 10.84.71.178`

可以用这个命令来替换 tcpkill 命令来模拟杀掉连接(RST包重置连接)

示例

查看本机 8001 端口的连接, -n 将端口以数字打印, -p 打印进程名字.

ss -np sport = :6390

```
wh@VM-16-16-ubuntu:~$ ss -np sport = :6390
Netid                  State                   Recv-Q                  Send-Q
                                     Local Address:Port
                Peer Address:Port
tcp                    ESTAB                   0                       0
                                     10.0.16.16:6390
             221.232.104.30:36340
```

杀掉连接:

sudo ss -np sport = :6390 -K

作业

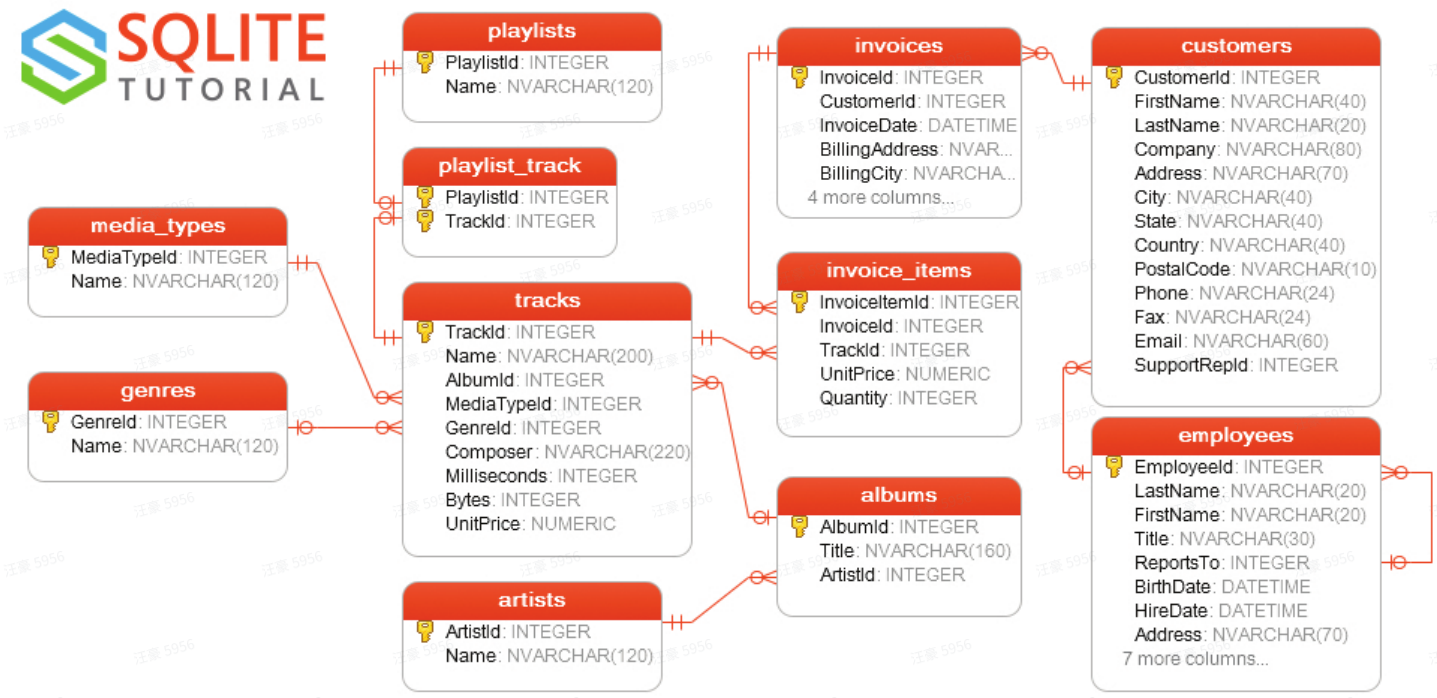
1. 数据库接口增删改查

参考后端3 中的 chinook 中的er图, 在 fastapi 中实现 `employees` `customers` 这两个表的增删改查接口.

数据库连接信息:

代码块

```
# 使用pip安装 postgresql 客户端
pip install pgcli
# 连接pg
pgcli.exe  -U u1 -h 41e1af074877.c.methodot.com -p 30290 -d chinook
# 密码是1234qwer
```



Chinook示例数据库包含11个表，如下：

1. `employees` (员工表)存储员工数据，例如 ID、姓氏、名字等。它还有一个名为 ReportsTo 的字段来指定谁向谁报告。
2. `customers` (客户表)存储客户数据。
3. `invoices` (发票抬头表)存储发票抬头数据。与发票明细表是一对多关系。
4. `invoice_items` (发票明细表)存储发票明细数据。
5. `artists` (艺术家表)存储艺术家数据。这是一个包含 ID 和姓名的简单表。
6. `albums` (专辑表)存储了曲目列表的数据。每张专辑属于一位艺术家，但一位艺术家可能有多张专辑。
7. `media_types` (媒体类型表)存储媒体类型，例如 MPEG 音频和 AAC 音频文件。
8. `genres` (流派表)存储摇滚、爵士、金属等音乐类型。
9. `tracks` (曲目表)存储歌曲的数据。每首曲目属于一张专辑。
10. `playlists` (播放列表)存储播放列表的数据。每个播放列表包含一个曲目列表。每个曲目可能属于多个播放列表。
11. `playlist_track` (播放列表与曲目中间表)播放列表表和曲目表之间的关系是多对多的。此表用于反映这种关系。

2. Redis 五种数据结构包装为 fastapi 的接口

公网redis连接信息(课程结束后弃用)

```
代码块
host:150.158.144.155
port: 6390
password: 1234qwer666
```

Strings/Numbers

```
代码块
import redis

# Instantiate a Redis client, connecting to localhost on port 6379
r = redis.Redis(
    host='150.158.144.155',
    port=6390,
    password='1234qwer666',
    db=0  # The default Redis database index
)
```

```
10
11  # 1. SET command: store a string under 'mykey'
12  r.set("mykey", "hello from Windows")
13
14  # 2. GET command: retrieve the value stored at 'mykey'
15  value = r.get("mykey")
16  print(value)   # Output is b'hello from Windows', since redis-py returns bytes.
17
18  # 3. Convert bytes to string
19  print(value.decode())  # prints "hello from Windows"
20
21  # 4. DEL command: remove 'mykey' from Redis
22  r.delete("mykey")
23
24  r.incr('mynum', 1)
25  r.get('mynum')
26  r.incr('mynum', 7)
27  r.get('mynum')
```

List

代码块

```
1   # LPUSH: Push an element to the head (left) of the list
2   r.lpush("task_queue", "task1")
3
4   # RPUSH: Push an element to the tail (right) of the list
5   r.rpush("task_queue", "task2")
6   r.rpush("task_queue", "task3")
7
8   # LPOP: Pop (remove and return) the element at the head
9   task = r.lpop("task_queue")
10  print(task)  # b'task1'
11
12  # Optional: RPOP removes and returns the element at the tail
13  task = r.rpop("task_queue")
14  print(task)  # b'task3'
```

Hash

代码块

```
1   # HSET: Store 'name' and 'email' fields for a user hash key
```

```
r.hset("user:1001", "name", "Alice")
r.hset("user:1001", "email", "alice@example.com")

# HGET: Retrieve a single field from the hash
email = r.hget("user:1001", "email")
print(email.decode('utf-8'))  # alice@example.com

# HDEL: Remove a field from the hash
r.hdel("user:1001", "email")
```

Set

代码块

```
# SADD: Add multiple members to a set
r.sadd("tags:python", "redis", "windows", "backend")

# SMEMBERS: Retrieve all unique members in the set
tags = r.smembers("tags:python")
print(tags)  # {b'redis', b'windows', b'backend'}
```

Sorted sets

代码块

```
# ZADD: Add members with scores
r.zadd("leaderboard", {"player1": 10, "player2": 20})

# ZRANGE: Retrieve members in ascending order of score
leaders = r.zrange("leaderboard", 0, -1, withscores=True)
print(leaders)  # [(b'player1', 10.0), (b'player2', 20.0)]
```

fastapi与redis示例:

https://github.com/republicroad/pydemo/blob/main/redisdemo.py

参考资料:

fastapi与postgresql最佳实践:

https://freedium.cfd/https://medium.com/@rameshkannanyt0078/effective-database-connection-handling-in-fastapi-best-practices-for-developers-cf448aa6f292

ss命令参考:

https://linux-audit.com/cheat-sheets/ss/
https://www.cyberciti.biz/tips/linux-investigate-sockets-network-connections.html

redis安装:
https://redis.io/learn/howtos/quick-start?_gl=1*1p8dgcm*_gcl_au*MTg2MzIzOTc5NS4xNzQ5NTM1Mzgz#setup-redis

redis 数据结构练习:

https://www.datacamp.com/tutorial/python-redis-beginner-guide

https://redis.io/learn/howtos/quick-start/cheat-sheet#stringsnumbers