PythonでWebアーキテクチャを学ぶ

自己紹介

— 小沢周平

— @oza_shu

— MSP業界でWEBサービスのサーバ運用

吉祥寺.pm13のテーマ

— 「新しい挑戦、新しい視点」

— 新年度に向けた決意的

2018年の抱負
ちゃんと調べてちゃんと理解する

ちゃんと調べて、ちゃんと理解する

サーバの運用で障害対応する時、なんとなくで対応することも

— このログ出力で検索して、Qiitaみたりとか...

— そのQiitaとか読んでもふんわり理解だったりとか...

論理的解決を目指す

— ちゃんと調べてちゃんと理解する

— 課題を解決する

例えば

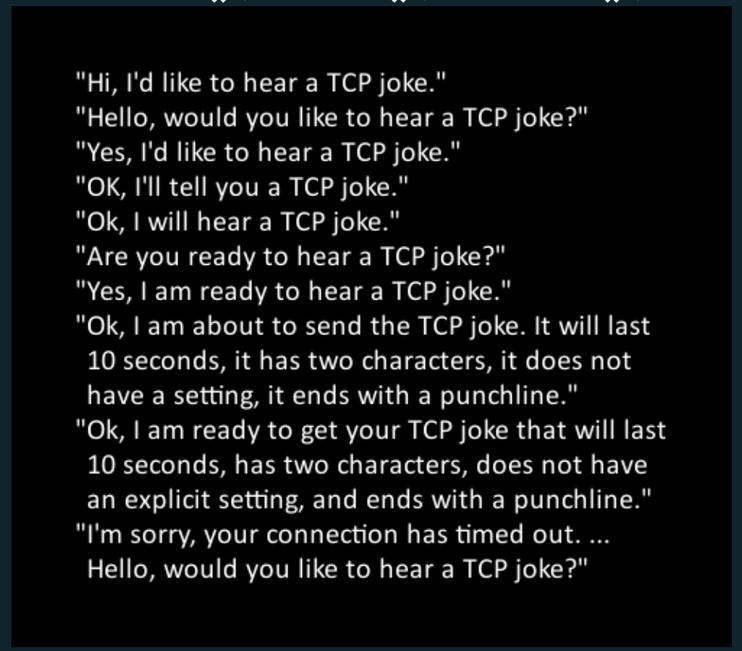
これらWebアーキテクチャをPythonでおさらい
- TCPコネクションはどんな通信をしているのか
- WEBサーバの動き、構造理解
- Apacheが詰まるってどういうこと
- C10K問題とは

TCP/IP通信

— IPアドレスとポートを指定してコネクション接続をする

— TCP jokeやSYN→SYN/ACK→ACK

— socket(),bind(),listen(),accept(),connect(),write(),read(),close()



シリアルモデル

— socket(),bind(),listen()でソケットの作成

```
def create_listen_socket(host, port):
    """ サーバーが接続要求を受け取るソケットを設定する """
    # アドレスファミリー、ソケットタイプ、プロトコル番号を指定して新しいソケットを作成
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind((host, port))
    sock.listen(100)
    return sock
```

— メッセージをsocketに接続して送信

```
if __name__ == '__main__':
    while True:
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.connect((HOST, PORT))
            print('\nConnected to {}:{}'.format(HOST, PORT))
            print("Type message, enter to send, 'q' to quit")
            msg = input()

            if msg == 'q': break
            chatmodule.send_msg(sock, msg) # 送信するまでBlock
            print('Sent message: {}'.format(msg))
            msg = chatmodule.recv_msg(sock) # messageを完全に受信するまでBlock
            print('Received echo: ' + msg)

        except ConnectionError:
            print('Socket error')
            break
        finally:
            sock.close()
            print('Closed connection to server\n')
```

— メッセージをsocketから受信

```
def handle_client(sock, addr):
    """ sockを通じてclientからデータを受けとり,echoを返す """
    try:
        msg = chatmodule.recv_msg(sock) # messageを完全に受信するまでblock
        print('{}: {}'.format(addr, msg))
        chatmodule.send_msg(sock, msg) # 送信するまでblock
    except (ConnectionError, BrokenPipeError):
         print('Socket error')
    finally:
        print('Closed connection to {}'.format(addr))
        sock.close()

if __name__ == '__main__':

    listen_sock = chatmodule.create_listen_socket(HOST, PORT)
    # ソケット自身のアドレスを返す
    # この関数は、IPv4/v6ソケットのポート番号を調べる場合などに使用。
    addr = listen_sock.getsockname()
    print('Listening on {}'.format(addr))

    while True:
        client_sock, addr = listen_sock.accept()
        print('Connection from {}'.format(addr))
        handle_client(client_sock, addr)
```

クライアントとサーバのソケット通信の動きをまとめると

1. サーバはsocket、bind、listenでクライアントの接続を待ち受ける
2. 接続きたらacceptにより実際のデータの読み出しまで待つ
3. データがきたら、リクエストを処理して、クライアントにレスポンスを返す
4. クライアントとの接続をcloseで閉じて、またaccept待ち状態になりライアントからの接続をLISTEN

クライアントからの接続をaccept()したあと、
ループを抜けるまでは新規のクライアント接続をブロックしている
これでは1:1の通信しかできない

マルチプロセスモデル

- forkさせて各プロセスが各クライアントと通信できる
- workerモデル
  - accept()したあとの処理をプロセスに処理させる
- preforkモデル
  - accept()からclose()までの処理をプロセスに処理させる

コード例 workerモデル

```
while True:
    client_sock,addr = listen_sock.accept()
    proc = Process(target=handle_client,
                   args=[client_sock, addr])
    proc.start()
    print('Connection from {}'.format(addr))
    proc.join(1)
```

デメリット:Apache詰まる

MaxClientsに達すると、
accept()がされないので、
接続は未処理となり詰まる

マルチスレッドモデル

基本的にはマルチプロセス/スレッドは同じ
- 1コネクション1スレッドのモデル
  - リクエストごとにスレッドを生成
- スレッドプール
  - 事前にスレッドをPoolしておくモデル

スレッドはメモリ共有しているので、スレッドセーフな実装が必要になる
(キューやロックの処理は割愛)

コード例 1コネクション1スレッドのモデル

```
if __name__ == '__main__':
    listen_sock = chatmodule.create_listen_socket(HOST, PORT)
    addr = listen_sock.getsockname()
    print('Listening on {}'.format(addr))
    while True:
        client_sock,addr = listen_sock.accept()
        # Thread は自動的にhandle_client()関数を実行し、同時にこのwhile loopを実行
        thread = threading.Thread(target=handle_client,
                                  args=[client_sock, addr],
                                  daemon=True)
        thread.start()
        print('Connection from {}'.format(addr))
```

C10K問題

インターネットが発展してWebサーバーが同時に1万のクライアントを処理する時代では
マルチプロセス/スレッドでは処理ができないので、
解決策として、イベント駆動アーキテクチャであるNginxが誕生
Nginxが目指すものはイベントループで1つのスレッドで数万の同時接続を処理すること。

もちろんマルチスレッド・アプローチで解決できたり、
他にはScalingの問題でもある。

— アーバン・エアーシップ社
    — C500k問題に直面した模様
— C10Mの問題
    — 今は、1000万の同時接続への挑戦

イベント駆動モデル

新しいイベントがキューに入れられ、
スレッドはイベントループを実行
イベントループでのスレッドを複数の接続にマッピング
接続、リクエストから発生したイベントを処理させる

イベント駆動型プログラミングを書かないと理解できない。。。

イベント駆動モデル

今後は以下も調べておきたい

- イベント駆動型プログラミング
  - それは何であり、どのように機能するのですか？
- asyncioモジュール
- asyncioベースのプログラミング
- Twisted
- Gevent

学んだこと

- TCP接続のながれ
- Apacheのつまりはなんなのか
- ss,netstat,lsofコマンドへの理解

WEBの次はDB

どうすればDBの気持ちになれるのか
良いメソッドあれば教えてください

まとめ

今年はちゃんと調べて、ちゃんと理解する!!