
# Informe Teórico
# “Middleware”

**Integrante.**
Matías Jesús Egaña Alfaro.

**Asignatura**
Sistemas Distribuidos.

**Profesor**
Juan Prudencio Torres Ossandon.

# Introducción

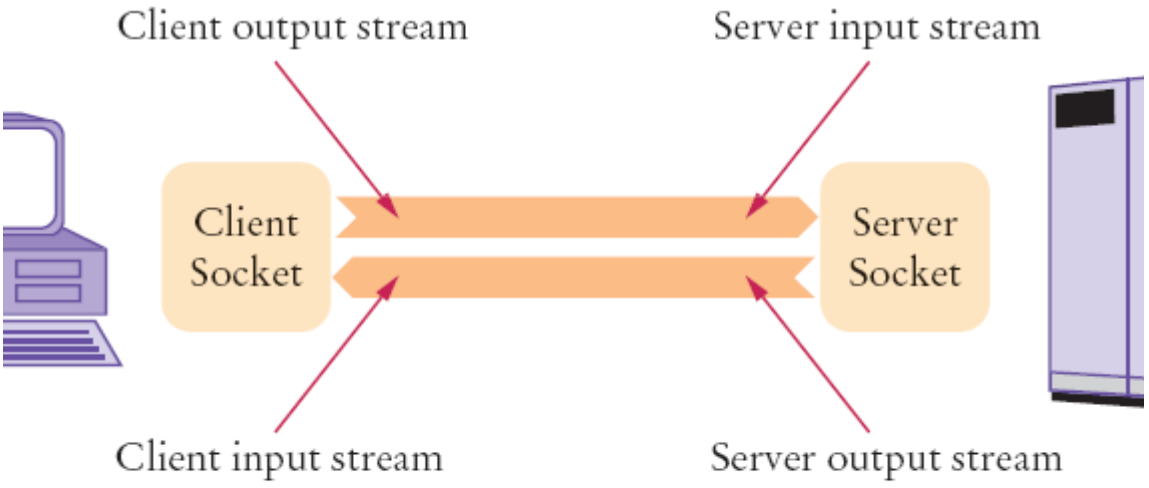
En este presente informe tiene como objetivo dar una explicación sobre el funcionamiento y entendimiento de un “Middleware” desde un código proporcionado por el profesor de la asignatura, desde el github: [sumedhe/python-middleware](sumedhe/python-middleware).

# Teoría

Antes de todo explicaremos brevemente algunos conceptos, que no serán de utilidad más adelante y es importante conocer su concepto.

## Socket

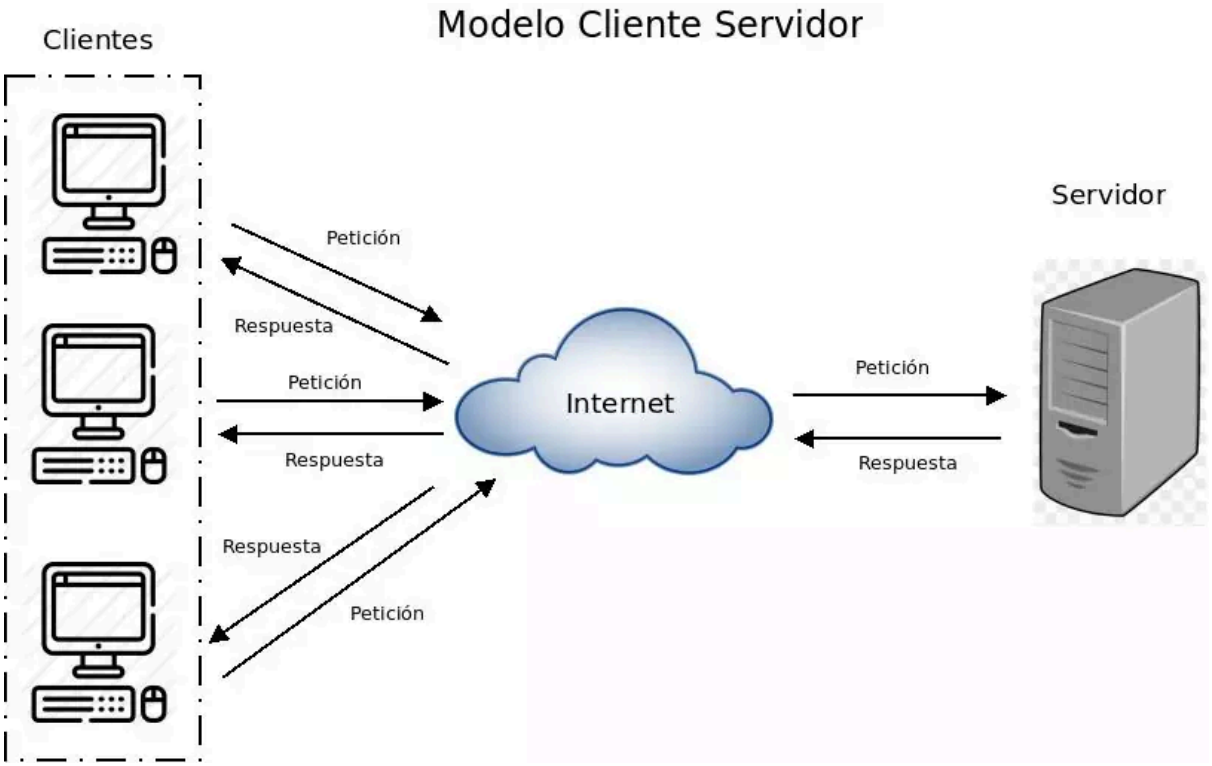
Es un punto final de un  enlace de comunicación bidireccional entre dos programas que se ejecutan en la red. Esta permite que los programas se comuniquen entre sí, ya sea en la misma computadora o en computadoras diferentes a través de una red.




*“Diagrama funcionamiento de comunicación por sockets”.*

## Modelo Cliente-Servidor

Es una forma de organizar la interacción entre aplicaciones distribuidas, donde los clientes solicitan servicios o recursos a servidores que los proporcionan. Esta arquitectura es ampliamente utilizada en una variedad de aplicaciones, incluidas las aplicaciones web, las aplicaciones móviles y los sistemas empresariales.




*"Diagrama arquitectura Cliente-Servidor".*

## Middleware

Es un software que actúa como una capa intermedia entre diferentes componentes de un sistema distribuido. Facilitando así la comunicación, la interoperabilidad y la gestión de servicios entre aplicaciones distribuidas.

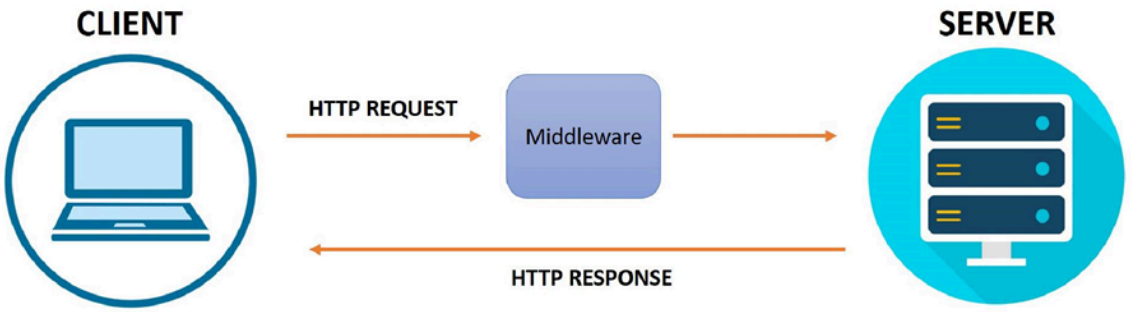



*Diagrama explicativo de cómo interactúa el middleware en la arquitectura Cliente-Servidor".*

# Corrección de código

En un comienzo el código se encuentra con problemas de sintaxis, entre estos en python. Por lo que se debieron arreglar, en los archivos con nombres: **gcd_service.py** y **queue_handler.py.**

Estos contaron con problemas sintácticos, esto puede ser debido a la sintaxis de python 2. Además hay que agregar un encode() a los mensajes que recibe como respuesta después de conectarse con el middleware.

Teniendo las siguientes modificaciones.

## gcd_service.py

Server\gcd_service.py

```
@@ -13,11 +13,11 @@ server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13 13   server.bind((HOSTNAME, PORT))
14 14   server.listen(5)  # max backlog of connections
15 15
16 -   print 'Listening on {}:{}'.format(HOSTNAME, PORT)
   16 +   print ('Listening on {}:{}'.format(HOSTNAME, PORT))
17 17
18 18   def handle_client_connection(client_socket):
19 19       request = client_socket.recv(1024)
20 -       print request
   20 +       print('request')
```

*"Modificación de los print en el código "gcd_service.py". Captura desde Github Desktop donde se aprecian con colores rojo y verde (la versión antigua y la nueva respectivamente)".*

Lo siguiente fue agregarle un encode() al conectarse con el middleware, debe mandar un mensaje y este debe ser decodificado, para ser mostrado al usuario.

```
43 43       # Connect to the middleware
44 44       sock.connect((MIDDLEWARE_HOSTNAME, MIDDLEWARE_PORT))
45 -       sock.send("addservice {} {} {}".format(SERVICE_NAME, HOSTNAME, PORT))
   45 +       sock.send("addservice {} {} {}".format(SERVICE_NAME, HOSTNAME, PORT).encode())
46 46
```

*"Modificación de los print en el código "gcd service.py", donde se le agrega encode(). Captura desde Github Desktop donde se aprecian con colores rojo y verde (la versión antigua y la nueva respectivamente)".*

Teniendo listo modificado el archivo “gcd_service.py”.

Server\gcd_service.py

```
@@ -13,11 +13,11 @@ server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13 13   server.bind((HOSTNAME, PORT))
14 14   server.listen(5)  # max backlog of connections
15 15
16    - print 'Listening on {}:{}'.format(HOSTNAME, PORT)
   16 + print ('Listening on {}:{}'.format(HOSTNAME, PORT))
17 17
18 18   def handle_client_connection(client_socket):
19 19       request = client_socket.recv(1024)
20    -     print request
   20 +     print('request')
21 21       try:
22 22           params   = request.split(' ')
23 23           response = gcd(int(params[0]), int(params[1]))
@@ -42,12 +42,12 @@ def register_service():
42 42
43 43       # Connect to the middleware
44 44       sock.connect((MIDDLEWARE_HOSTNAME, MIDDLEWARE_PORT))
45    -     sock.send("addservice {} {} {}".format(SERVICE_NAME, HOSTNAME, PORT))
   45 +     sock.send("addservice {} {} {}".format(SERVICE_NAME, HOSTNAME, PORT).encode())
46 46
47 47       # Get response
48 48       response = sock.recv(4096)
49 49       sock.close()
50    -     print response
   50 +     print(response)
51 51
52 52
53 53   # Register the service
@@ -56,5 +56,5 @@ register_service()
56 56   # Accept requests
57 57   while True:
58 58       client_sock, address = server.accept()
59    -     print 'Accepted connection from {}:{}'.format(address[0], address[1])
   59 +     print('Accepted connection from {}:{}'.format(address[0], address[1]))
60 60       handle_client_connection(client_sock)
```

*“Cambios del codigo “”gcd_service.py”*

## queue_handler.py

Ahora mostraremos las modificaciones del archivo “queue_handler.py”.

Middleware\queue_handler.py

```
@@ -14,7 +14,7 @@ SLEEP_TIME = 0.2

14 14
15 15     # Consume messages
16 16     def handle_request_queue():
17    -           print 'Request queue handler started'
   17 +           print('Request queue handler started')
18 18
```

*“Modificación de los print en el código “queue_handler.py”. Captura desde Github Desktop donde se aprecian con colores rojo y verde (la versión antigua y la nueva respectivamente)”.*

```
    # Send responses to client
    def handle_response_queue():
-           print 'Request queue handler started'
+           print('Request queue handler started')

            while True:
                    if (RESPONSE_QUEUE.isEmpty()):
                            time.sleep(SLEEP_TIME)
                    else:
                            # Send response to the client
-                           (client_socket, message) = RESPONSE_QUEUE.dequeue()
-                           client_socket.send(message)
-                           client_socket.close()
+                           (client_socket, message) = RESPONSE_QUEUE.dequeue()
+                           client_socket.send(message.encode())
+                           client_socket.close()
```

*“Modificación de los print en el código “queue_handler.py”, donde se le agrega encode(). Captura desde Github Desktop donde se aprecian con colores rojo y verde (la versión antigua y la nueva respectivamente)”.*

Teniendo listo modificado el archivo “queue_handler.py”.

Middleware\queue_handler.py

```
@@ -14,7 +14,7 @@ SLEEP_TIME = 0.2

# Consume messages
def handle_request_queue():
-         print 'Request queue handler started'
+         print('Request queue handler started')

         while True:
                 if (REQUEST_QUEUE.isEmpty()):
@@ -22,28 +22,27 @@ def handle_request_queue():
                 else:
                         # Consume
                         (client_socket, message) = REQUEST_QUEUE.dequeue()
-
                         handle_request(client_socket, message)

# Send responses to client
def handle_response_queue():
-         print 'Request queue handler started'
+         print('Request queue handler started')

         while True:
                 if (RESPONSE_QUEUE.isEmpty()):
                         time.sleep(SLEEP_TIME)
                 else:
                         # Send response to the client
-                        (client_socket, message) = RESPONSE_QUEUE.dequeue()
-                        client_socket.send(message)
-                        client_socket.close()
+                        (client_socket, message) = RESPONSE_QUEUE.dequeue()
+                        client_socket.send(message.encode())
+                        client_socket.close()

# Handle the request
def handle_request(client_socket, message):
         # Parse message
         try:
                 (action, params) = get_params(message)
-                print "Action: " + action
+                print ("Action: " + action.decode())

                 # Handle request
                 if (action.lower() == "addservice"): # Add new service
@@ -57,7 +56,7 @@ def handle_request(client_socket, message):
                 else:
                         response = "Service not found"
         except Exception as e:
```

```
59  58          except Exception as e:
60      -               print e
    59  +               print (e)
61  60              response = "Invalid request format"
62  61
63  62          RESPONSE_QUEUE.enqueue((client_socket, str(response)))
```

*"Cambios del codigo "queue_handler.py".*

# Ejecución

Para ejecutar este código es necesario contar con el intérprete de python 2. En mi caso se usó la versión 2.7.18. Así mismo, se requiere tener java instalado, para esto se tenía instalada la versión "21.0.1".

Para comenzar será necesario tener abiertas tres terminales de los archivos: **main.py, gcd_service.py y Client.java.**

El primero se encarga de crear un servidor de socket que puede aceptar múltiples conexiones de clientes simultáneamente, manejar las solicitudes y respuestas de manera asíncrona utilizando hilos y colas,  procesando así los datos recibidos.

```
PROBLEMAS 4   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL   PUERTOS                py - Middleware

-a----        19-04-2024      8:38           3713 .gitignore
-a----        19-04-2024      8:38           1097 LICENSE
-a----        19-04-2024      8:38            841 README.md


PS C:\Users\mati_\OneDrive\Documentos\GitHub\Proyectos uls\JT\python-middleware> cd Middleware
PS C:\Users\mati_\OneDrive\Documentos\GitHub\Proyectos uls\JT\python-middleware\Middleware> py.exe .\main.py
Request queue handler started
Request queue handler started
Listening on localhost:9999
```

*"Ejecución de la terminal de main.py, donde se encuentra esperando peticiones".*

El segundo implementa un servidor de socket que calcula el MCD[1] de dos números enteros y registra este servicio en un middleware antes de aceptar conexiones de clientes.

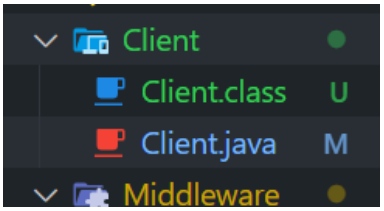
```
PS C:\Users\mati_\OneDrive\Documentos\GitHub\Proyectos uls\JT\python-middleware> cd Server
>> python gcd_service.py
Listening on localhost:9993
Service 'gcd' is added successfully
```

*"Ejecución de la terminal de gcd_service,py, donde se añade la operación y lo agrega en el middleware".*

[1] **M**áximo **C**omún **D**ivisor.

Por último, el tercero es un cliente java que se conecta al servidor de middleware, envía un mensaje con los argumentos proporcionados desde la línea de comandos, y luego imprime la respuesta recibida del servidor.

Este genera un archivo en formato de bytecode de Java. Este bytecode es ejecutable por la Máquina Virtual de Java (JVM). Cuando se ejecuta el programa Java utilizando se usan los comando java Client, la JVM carga y ejecuta el bytecode contenido en el archivo Client.class.




```
PS C:\Users\mati_\OneDrive\Documentos\GitHub\Proyectos uls\JT\python-middleware> cd Client
>> javac Client.java
```

*"Ejecución de la terminal de Client,java, donde genera un archivo compilado ".*

Finalmente se manda el siguiente comando y nos entrega un resultado:

```
PS C:\Users\mati_\OneDrive\Documentos\GitHub\Proyectos uls\JT\python-middleware\Client> java Client gcd 12 18
6
```

*"En la terminal se da los numero 12 y 18 dando como respuesta el 6, que es el MCD de los números"*

Ahora si nos fijamos en las demás terminales, vemos que en "gcd_service.py", se conecta y recibe una petición y en la terminal "main.py se conecto a servidor y mandó la acción, retornando al cliente la respuesta que es: 6.

```
PROBLEMAS 4  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

PS C:\Users\mati_\OneDrive\Documentos\GitHub\Proyectos uls\JT\python-middleware> cd Server
>> python gcd_service.py
Listening on localhost:9993
Service 'gcd' is added successfully
Accepted connection from 127.0.0.1:58773
request
```

```
PS C:\Users\mati_\OneDrive\Documentos\GitHub\Proyectos uls\JT\python-middleware> cd Middleware
>> python main.py
Request queue handler startedRequest queue handler started

Listening on localhost:9999
Accepted connection from 127.0.0.1:58708
Action: addservice
Accepted connection from 127.0.0.1:58772
Action: gcd
```

*"Terminales gcd_service.py y main.py, respectivamente".*

# Conclusiones

Este ejercicio fue muy beneficioso, puesto que se pudo entender el funcionamiento de un middleware, junto con su importancia en una arquitectura cliente-servidor, donde se pidió una acción, esta es recibida y se desencadena la acción solicitada.

Por lo que un middleware es fundamental, esto porque proporciona una capa de abstracción, gestión de servicios, escalabilidad y gestión de concurrencia que simplifica el desarrollo y la operación de sistemas distribuidos.