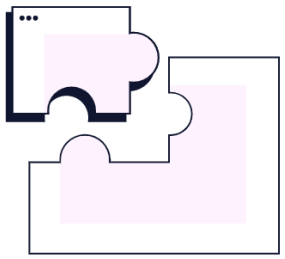
Jul. 2022

Unidad 5: Trabajando en Red



Objetivos de aprendizaje

Python proporciona dos niveles de acceso a los servicios de red.

1. **Nivel bajo**: puedes acceder al soporte básico de sockets en el sistema operativo subyacente,

   Permite implementar clientes y servidores

   para protocolos orientados a la conexión y sin conexión.

2. **Nivel alto**: Python también tiene bibliotecas que proporcionan

   a protocolos de red específicos a nivel de aplicación, como FTP, HTTP, etc.

En esta unidad vamos a ser capaces de entender el concepto más famoso para la realización de operaciones de red: la programación de sockets.

Introducción

- Usaremos la biblioteca de sockets para redes de Python a través de algunas recetas sencillas.
- El módulo de sockets de Python tiene utilidades basadas en clases y en instancias.

  La diferencia entre ellas es que el primero no necesita

  la instancia de un objeto y es un enfoque muy intuitivo.
- Si se necesita enviar datos a una aplicación de servidor
  - es más intuitivo crear un objeto socket

Introducción

Las recetas que se presentan se pueden clasificar en tres grupos como sigue:

1. Las utilidades **basadas en clases**

para extraer alguna información útil sobre el host, la red y cualquier servicio de destino.

2. Las utilidades **basadas en instancias**

Para algunas tareas comunes de los sockets

(tiempo de espera del socket, tamaño del buffer y el modo de bloqueo)

3. Las utilidades **basadas en clases como en instancias**

Para construir algunos clientes que realizan algunas tareas prácticas

(sincronizar la hora de la máquina con un servidor de Internet

o escribir un script genérico cliente/servidor).

¿Qué son los sockets *(zócalos)*?

- Son los puntos finales de un canal de comunicación bidireccional.
- Pueden comunicarse
  - dentro de un proceso,
  - entre procesos en la misma máquina
  - entre procesos en diferentes máquinas en diferentes continentes.
- Pueden ser implementados sobre un número de diferentes tipos de canales:
  - sockets de dominio Unix, TCP, UDP, etc.
- La biblioteca de sockets proporciona
  - clases específicas para manejar los transportes comunes
  - una interfaz genérica para manejar el resto.

1. Imprimir el nombre del host

- Obtener el nombre del host actual

  gethostname() -> string

```
>>> import socket
>>> socket.gethostname()
```

2. Recuperar la dirección IP de una máquina

gethostbyname(host) -> address

Para un host devuelve su dirección IP

- una cadena de la forma '255.255.255.255'

```
>>> import socket
>>> socket.gethostbyname(host)
```

3. Conversión de una dirección IPv4 a diferentes formatos

- >>> import socket
- >>> from binascii import hexlify

- **inet_aton**(string) -> bytes giving packed 32-bit IP representation
  Convierte una IP en una notación hexadecimal
- **inet_ntoa**(packed_ip) -> ip_address_string
  Convierte una IP a formato cadena de caracteres

4. Dado el puerto y el protocolo encontrar el nombre de servicio

- >>> import socket
- getservbyport(port[, protocolname]) -> string

Devuelve el nombre del servicio a partir de un número de puerto y un nombre de protocolo.

El nombre de protocolo opcional, si se da, debe ser 'tcp' o 'udp',

de lo contrario, cualquier protocolo será match.

5. Convirtiendo enteros a y desde el orden de bytes del host a la red

- >>> import socket
- Donde:
  - n red
  - h host
  - l large
  - s short
- Largo (32-bit integer)
  - ntohl(int) -> int
  - htonl(int) -> int
- Corto (16-bit integer)
  - ntohs(int) -> int
  - htons(int) -> int

6. Establecer y obtener el tiempo de espera del socket por defecto

- Usar los métodos de instancia getter/setter.
- 'timeout’ es un float en segundos
- >> s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- >> s.gettimeout()
- >> s.settimeout(timeout)

7. Manejo de errores de socket con elegancia

- Poner bloques try-except en las operaciones típicas de los sockets
  - crear un objeto socket
  - conectarse a un servidor
  - enviar datos
  - esperar una respuesta

- Utilizar el módulo **argparse** para obtener la entrada del usuario
  - módulo más potente que sys.argv.

8. Modificar el tamaño del buffer de envío/recepción de un socket

- Recuperar y modificar las propiedades de un objeto socket llamando a los métodos
  - getsockopt() [constante simbólica del nivel, opción, valor]
  - setsockopt()

- sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- sock.getsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF)
- sock.setsockopt(socket.SOL_TCP, socket.TCP_NODELAY, 1)
- sock.setsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF, **integer**)
- sock.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, **integer**)

9. Cambiar un socket al modo de bloqueo/no bloqueo

- Los sockets TCP se colocan en modo de bloqueo (por defecto)
- Llamar a:
  - setblocking(1) para establecer el bloqueo
  - setblocking(0) para desbloqueo
- Vinculamos el socket a un puerto específico
- Escuchar las conexiones entrantes
- s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- s.setblocking(1)
- s.bind((direccionIP, 0))

10. Reutilización de direcciones de socket

- Ejecutar un servidor de sockets siempre en un puerto específico
  - incluso después de cierres intencionales o inesperadados
- [Errno 98] Dirección ya está en uso
- Solución: Activar la opción de reutilización de sockets (SO_REUSEADDR)
- s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
- s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
- s.getsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR)
- puerto = 8282
- s.bind((‘’, puerto))
- s.listen(1)

11. Impresión de la hora actual desde el servidor de la hora de Internet

- Protocolo de Tiempo de Red (NTP).
- Sincronizar la hora de tu máquina con uno de los servidores NTP
- Se utilizará la librería **ntplib**
- import ntplib
- from time import ctime
- ntp_client = ntplib.NTPClient()
- respuesta = ntp_client.request('pool.ntp.org')
- print(ctime(respuesta.tx_time))

12. Escribir un cliente SNTP

- Crear un cliente SNTP sin utilizar ninguna librería de terceros
- import socket, struct, sys, time
- NTP_SERVER = "es.pool.ntp.org"
- TIME1970 = 2208988800

- client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
- data = '\x1b' + 47 * '\0'
- client.sendto(data.encode('utf-8'), (NTP_SERVER, 123))
- data, address = client.recvfrom(1024)
- t = struct.unpack('!12I', data)[10]
- t -= TIME1970

13. Escribiendo una aplicación TCP -servidor

- Crear un objeto socket TCP
- Establecer la dirección de reutilización
- Vincular el socket al puerto dado en nuestra máquina local.
- Escuchar a múltiples clientes en una cola
- Esperar a que el cliente se conecte y envíe datos
- Cuando los datos son recibidos, el servidor devuelve los datos

13. Escribiendo una aplicación TCP -cliente

- Crear un socket de cliente utilizando el argumento del puerto
- Conectar al servidor
- Enviar el mensaje, “Mensaje de prueba”
- Recibir el mensaje de vuelta
- Atrapar cualquier excepción durante la sesión

14. Escribiendo una aplicación UDP -servidor

- El método recvfrom() lee los mensajes del socket
- y devuelve los datos
- y la dirección del cliente.

14. Escribiendo una aplicación UDP -cliente

- Crear un socket de cliente usando el argumento del puerto
- Conectar al servidor
- Enviar el mensaje “mensaje de prueba”
- Recibir el mensaje de vuelta

15. Realizar una búsqueda de DNS

- Obtener la dirección IP de un host específico
- socket.getaddrinfo(URL, port)

16. Realizar una petición HTTP usando sockets sin procesar (*raw*)

- Crearemos un programa que recibe una URL como entrada
  y después de realizar la petición HTTP
  devuelve la respuesta del servidor web solicitado

- Para leer y escribir bytes de datos desde utilizamos la librería de sockets TCP

- Es necesario crear una función que pueda analizar una URL
  y devolver los componentes de nombre de host y ruta de la URL.

16. Realizar una petición HTTP usando sockets sin procesar (*raw*)

- Utilizar la biblioteca de sockets para realizar una solicitud HTTP

```
sock = socket.socket()
    sock.connect(ip, 80)
    peticion = HTTP_REQUEST.format(host=host, path=path)
    sock.send(bytes(peticion, 'utf8'))
    respuesta = b''

    while True:
        trozo = sock.recv(BUFFER_SIZE)
        if not trozo:
            break
        respuesta += trozo

    sock.close()
    cuerpo = respuesta.split(b'\r\n\r\n', 1)[1]
```

16. Realizar una petición HTTP usando sockets sin procesar (*raw*)

1. Obtener la información del nombre de host, la dirección IP de una URL dada.
   Realizando una división de cadena simple utilizando el carácter /.
   Eliminando la parte inicial de la URL
2. Crear una conexión de socket con el servidor web.
3. Crear la petición HTTP rellenando la plantilla llamada HTTP_REQUEST.
4. Leer continuamente los resultados (como una serie de trozos)
   hasta el final de la respuesta.
5. Concatenar los trozos de datos en una variable llamada respuesta.
6. Respuesta contiene tanto cabeceras como cuerpo HTTP.
   Utilizamos el método split para extraer solamente el cuerpo.
7. Convertir el objeto bytes a una cadena y devolverla.

17. Obtención de datos JSON desde un servicio web RESTful

JSON es el formato de serialización más popular para servicios web

Python tiene soporte incorporado para

analizar este formato.

creación de salidas JSON (para enviar datos a servicios web)

subir continuamente datos en formato JSON

a un servidor remoto usando servicios web.

17. Obtención de datos JSON desde un servicio web RESTful

1. Definir una constante llamada URL que tiene la URL de la API del servicio web
2. Llamar a la API realizando una petición HTTP GET
   el servidor devuelve su salida en formato JSON.
3. Analizar la respuesta en estructuras de datos de Python.
   acceder a la clave iss_position
   y a la información de latitud y longitud
4. Realizar un bucle de 10 iteraciones,
   con un tiempo de espera de 1 segundo entre cada bucle
   antes de llamar a la API e imprimir los resultados analizados.

17. Obtención de datos JSON desde un servicio web RESTful

```
import requests
import time

URL = 'http://api.open-notify.org/iss-now.json'

for i in range(10):
      datos = requests.get(URL).json()
      posicion = datos['iss_position’]
      print(f"{i} lat: {posicion['latitude’]} …")
      time.sleep(1)
```

18. Creación de un servidor HTTP

Crear un servidor web en Python

servirá páginas web con contenido dinámico.

Puede ser una poderosa herramienta para los proyectos

ofrece la posibilidad de interactuar con cualquier dato o información

en tiempo real

desde cualquier dispositivo de la red

Ejemplo:

Un servidor web que muestre el tiempo que lleva en marcha en segundos.

```
tiempo_inicio = time.monotonic()
tiempo_en_marcha = time.monotonic() - tiempo_inicio
```

18. Creación de un servidor HTTP

```
<!DOCTYPE HTML>
<html lang="en">
<head>
    <title>Servidor HTTP del Curso</title>
</head>
<body>
    <h1>Prueba</h1>
    <b>Tiempo: </b> {tiempo_en_marcha} s.
</body>
</html>
```

18. Creación de un servidor HTTP

```
<!DOCTYPE HTML>
<html lang="en">
<head>
    <title>Servidor HTTP del Curso</title>
</head>
<body>
    <h1>Prueba</h1>
    <b>Tiempo: </b> {tiempo_en_marcha} s.
</body>
</html>
```

```
import socket
import time
sock = socket.socket()
sock.bind(('0.0.0.0', HTTP_PORT))
sock.listen(TCP_BACKLOG)
conn = sock.accept()

html = TEMPLATE.format(tiempo)
html = bytes(html.encode())
conn.send(html)
conn.close()
```

18. Creación de un servidor HTTP

La función **socket_listen** se llama para enlazar y escuchar el puerto HTTP por defecto.

La función **serve_requests** servirá interminablemente todas las peticiones HTTP entrantes.

Se registra La hora de inicio del servidor web

Llamamos al **método accept**, que bloquea el código hasta que llegue una nueva petición.

Cuando recibimos una nueva petición,

consumimos todas las cabeceras de la petición HTTP

llamando repetidamente al método **readline**

hasta que hayamos detectado el final de las cabeceras.

Generar nuestra respuesta HTML

Utilizando el **método send** enviamos la respuesta al cliente HTTP.

Cerramos la conexión con el cliente.