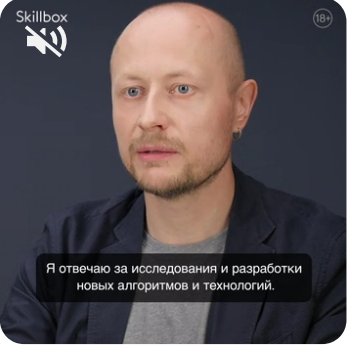



Низкоуровневый сетевой интерфейс в Python



skillbox.ru

Data Scientist с нуля до Junior - 9 проектов в портфолио
Станьте дата-инженером, аналитиком данных или ML-инженером. з/п от 120 000 руб.
[Узнать больше](#)

РЕКЛАМА · 16+

[Стандартная библиотека Python3.](#) / Низкоуровневый сетевой интерфейс в Python

Модуль [socket](#) обеспечивает доступ к интерфейсу сокета BSD. Он доступен во всех современных системах Unix, Windows, MacOS и, возможно, на дополнительных платформах.

Он включает в себя функции создания объекта сокета [Socket](#), который и обрабатывает канал данных, а также функции, связанных с сетевыми задачами, такими как преобразование имени сервера в IP адрес и форматирование данных для отправки по сети.

Примечание. Поведение модуля может зависеть от платформы, поскольку выполняются вызовы API сокетов операционной системы.

Интерфейс Python представляет собой прямую трансляцию системного вызова Unix и интерфейса библиотеки для сокетов в объектно-ориентированный стиль Python. Функция [socket.socket\(\)](#) возвращает объект [Socket](#), методы которого реализуют различные системные вызовы сокетов.

Типы параметров функций модуля несколько более высокоуровневые, чем в интерфейсе языка C: как и в случае операций чтения/записи с файлами, распределение буфера при операциях приема данных происходит автоматически, а длина буфера неявно определяется операциями отправки.

Сокеты можно настроить для работы в качестве сервера и прослушивания входящих сообщений или для подключения к другим приложениям в качестве клиента. После подключения обоих концов сокета TCP/IP обмен данными становится двунаправленным.

Пример создания и использования сокетов на примере TCP/IP сервер и клиента.

Этот пример, основанный на стандартной документации библиотеки, принимает входящие сообщения и передает их обратно отправителю. Он начинается с создания сокета TCP/IP, а затем метод `sock.bind()` используется для связывания сокета с адресом сервера.

Примечание. Для успешного тестирования примера, код клиента и сервера необходимо запускать в разных окнах терминала. Код сервера запускается первым.

TCP/IP сервер.

```
# test-server.py
import socket
import sys

# создаемTCP/IP сокет
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Привязываем сокет к порту
server_address = ('localhost', 10000)
print('Старт сервера на {} порт {}'.format(*server_address))
sock.bind(server_address)

# Слушаем входящие подключения
sock.listen(1)

while True:
    # Вверх
    # м соединения
```

```
print('Ожидание соединения...')
connection, client_address = sock.accept()
try:
    print('Подключено к:', client_address)
    # Принимаем данные порциями и ретранслируем их
    while True:
        data = connection.recv(16)
        print(f'Получено: {data.decode()}')
        if data:
            print('Обработка данных...')
            data = data.upper()
            print('Отправка обратно клиенту.')
            connection.sendall(data)
        else:
            print('Нет данных от:', client_address)
            break

finally:
    # Очищаем соединение
    connection.close()
```

Вызов метода `sock.listen(1)` переводит сокет в режим сервера, а метод `sock.accept()` ожидает входящего соединения. Целочисленный аргумент у метода `.listen` - это количество соединений, которые система должна поставить в очередь в фоновом режиме, прежде чем отклонять новых клиентов. В этом примере предполагается, что одновременно будет работать только одно соединение.

Метод `sock.accept()` возвращает открытое соединение между сервером и клиентом вместе с адресом клиента. На самом деле соединение представляет собой другой сокет на другом порту (назначенный ядром). Данные считываются из соединения с помощью метода `sock.recv()` и передаются с помощью `sock.sendall()`.

Когда общение с клиентом завершено, соединение необходимо очистить с помощью `sock.close()`. В этом примере используется [блок try/finally](#), чтобы гарантировать, что метод `sock.close()` всегда вызывается, даже в случае ошибки.

Клиентская программа настраивает свой сокет иначе, чем сервер. Вместо привязки к порту и прослушивания он использует метод `sock.connect()` для подключения сокета непосредственно к удаленному адресу.

TCP/IP клиент.

```
# test-client.py
import socket
import sys

# Создаем TCP/IP сокет
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Подключаем сокет к порту, через который прослушивается сервер
server_address = ('localhost', 10000)
print('Подключено к {} порт {}'.format(*server_address))
sock.connect(server_address)

try:
    # Отправка данных
    mess = 'Hello World!'
    print(f'Отправка: {mess}')
    message = mess.encode()
    sock.sendall(message)

    # Смотрим ответ
    amount_received = 0
    amount_expected = len(message)
    while amount_received < amount_expected:
        data = sock.recv(16)
        amount_received += len(data)
        mess = data.decode()
        print(f'Получено: {data.decode()}')

finally:
```

[Вверх](#)

```
print('Закрываем сокет')
sock.close()
```

После установления соединения данные могут быть отправлены через сокет с помощью метода `sock.sendall()` и получены с помощью `sock.recv()`, как и на сервере. Когда все сообщения отправлены, а копия получена, то сокет закрывается, чтобы освободить порт.

Работа клиента и сервера вместе.

Клиент и сервер должны запускаться в отдельных окнах терминала, чтобы они могли взаимодействовать друг с другом. Выходные данные сервера показывают входящее соединение и данные, а также ответ, отправленный обратно клиенту.

```
Старт сервера на localhost порт 10000
Ожидание соединения...
Подключено к: ('127.0.0.1', 34800)
Получено: Hello World!
Обработка данных...
Отправка обратно клиенту.
Получено:
Нет данных от: ('127.0.0.1', 34800)
Ожидание соединения...
...
```

Выходные данные клиента показывают исходящее сообщение и ответ сервера.

```
Подключено к localhost порт 10000
Отправка: Hello World!
Получено: HELLO WORLD!
Закрываем сокет
```

Содержание раздела:

- [КРАТКИЙ ОБЗОР МАТЕРИАЛА.](#)
- [Советы по программированию сокетов](#)
- [Константы, определяемые модулем socket](#)
- [Семейства сокетов, поддерживаемых модулем socket](#)
- [Функция socket\(\) модуля socket, создает новый сокет](#)
- [Функция create_connection\(\) модуля socket](#)
- [Функция create_server\(\) модуля socket](#)
- [Функция socketpair\(\) модуля socket](#)
- [Функция fromfd\(\) модуля socket](#)
- [Функция fromshare\(\) модуля socket](#)
- [Объект Socket модуля socket](#)
- [Функция close\(\) модуля socket](#)
- [Функция getaddrinfo\(\) модуля socket](#)
- [Функция getfqdn\(\) модуля socket](#)
- [Функция gethostbyname\(\) модуля socket](#)
- [Функция gethostbyname_ex\(\) модуля socket](#)
- [Функция gethostname\(\) модуля socket](#)
- [Функция gethostbyaddr\(\) модуля socket](#)
- [Функция getnameinfo\(\) модуля socket](#)
- [Функция getprotobyname\(\) модуля socket](#)
- [Функция getservbyname\(\) модуля socket](#)
- [Функция getservbyport\(\) модуля socket](#)
- [Функция has_dualstack_ipv6\(\) модуля socket](#)
- [Функции getdefaulttimeout\(\) и setdefaulttimeout\(\) модуля socket](#)
- [Функции MSG_LEN и MSG_SPACE модуля socket](#)
- [Функция sethostname\(\) модуля socket](#)
- [Функция if_nameindex\(\) модуля socket](#)
- [Функции if_nametoindex\(\) и if_indextoname модуля socket](#)

Вверх

- [Функции различных преобразований модуля socket](#)
- [Ошибки и исключения, определяемые модулем socket](#)

ХОЧУ ПОМОЧЬ
ПРОЕКТУ



DOCS-Python.ru™, 2023 г.

(Внимание! При копировании материала ссылка на источник обязательна)

@docs_python_ru

Вверх