


ХОЧУ ПОМОЧЬ  
ПРОЕКТУ Б

# Библиотека Telethon в Python, Telegram клиент



80%

80%

80%

80%

80%


80%

80%


80%

80%

80%



Электроника



Одежда

Низкие цены ещё ниже.

и ещё и

и ещё и

avito.ru

РЕКЛАМА · 16+

Больше информации на сайте рекламодателя

Подробнее

[Сторонние пакеты и модули Python3.](#) / Библиотека Telethon в Python, Telegram клиент

## Асинхронная обертка API мессенджера Telegram

[Библиотека Telethon](#) предназначена для облегчения разработки программ на Python, которые могут взаимодействовать с платформой Telegram. Представляет собой асинхронную обертку API Telegram, которая делает всю тяжелую и нудную работу, тем самым позволяя сосредоточиться на разработке приложения.

## Установка модуля Telethon в виртуальное окружение:

```
# создаем виртуальное окружение
$ python3 -m venv .env --prompt VirtualEnv
# активируем виртуальное окружение
$ source .env/bin/activate
# обновляем установщик пакетов
python3 -m pip install --upgrade pip
# ставим пакет telethon
python3 -m pip install --upgrade telethon
```

## Содержание:

- [Авторизация в Telegram с помощью библиотеки Telethon;](#)
  - [Вход в качестве учетной записи бота;](#)
  - [Вход в систему через прокси-сервер;](#)
  - [Использование прокси-серверов MTProto;](#)
- [Дружественные методы работы с библиотекой telethon;](#)
- [Введение в получение обновлений;](#)
  - [Еще примеры получения обновлений;](#)
- [Понятие сущности entity пакета Telethon;](#)
- [Совместимость Telethon;](#)
- [Удобство Telethon;](#)
- [Скорость и накладные расходы Telethon.](#)

## Авторизация в Telegram с помощью библиотеки Telethon.

Прежде чем работать с API Telegram, необходимо получить собственный идентификатор и хеш API:

1. Необходимо войти в свою учетную запись Telegram, указав номер телефона учетной записи разработчика. Страница <https://my.telegram.org> "Delete Account or Manage Apps".
2. Далее нужно нажать "Инструменты разработки API": появится окно "Создать новое приложение". Заполните данные нового приложения. Нет необходимости вводить какой-либо URL-адрес, позже можно изменить только первые два поля (название приложения и краткое имя).
3. В конце нажмите "Создать заявку". Помните, что ваш хэш API является секретным, и Telegram не позволит вам его отозвать. Никуда его не публикуйте!

[Вверх](#)

*Примечание.* Идентификатор и хэш API используются только приложением, а не номером телефона. Другими словами можно использовать этот идентификатор и хэш с любым номером телефона или даже для учетных записей ботов.

После получения идентификатора и хэша API можно написать код для входа в учетную запись Telegram!

РЕКЛАМА

```
# файл test.py
from telethon import TelegramClient

# Подставляем собственные значения из `my.telegram.org`
api_id = 12345
api_hash = '0123456789abcdef0123456789abcdef'

# Первый аргумент - это имя файла `.session` (допускаются абсолютные пути).
with TelegramClient('anon', api_id, api_hash) as client:
    client.loop.run_until_complete(client.send_message('me', 'Hello, myself!'))
```

*Важно!* Не называйте скрипт telethon.py! Python попытается импортировать клиент из telethon.py, которого там нет. В итоге сценарий завершится неудачей с ошибкой `ImportError: cannot import name TelegramClient ....`

В первой строке импортируется имя класса, чтобы создать экземпляр клиента. Затем определяются переменные для удобного хранения идентификатора и хэша API.

Наконец, создается новый экземпляр TelegramClient и называем его client. Теперь можно использовать переменную client для чего угодно, например для отправки сообщения самому себе.

*Примечание.* Так как Telethon является асинхронной библиотекой, то необходимо [дождаться await](#) выполнения функций сопрограммы (или, в противном случае, запускать цикл до их завершения). В этом примере не создается [асинхронное определение async def](#).

Использование [контекстного менеджера with](#) является предпочтительным способом использования библиотеки. Он автоматически запустит клиент TelegramClient.start(), выполнив вход или регистрацию при необходимости.

Если файл .session уже существует и куда нибудь перемещен или переименован, то клиент не сможет снова войти (имейте это в виду)!

## Вход в качестве учетной записи бота.

Также можно использовать пакет Telethon для своих ботов (обычных учетных записей ботов, а не пользователей). Для этого все равно понадобится идентификатор и хэш API, но процесс очень похож:

```
from telethon.sync import TelegramClient

api_id = 12345
api_hash = '0123456789abcdef0123456789abcdef'
bot_token = '12345:0123456789abcdef0123456789abcdef'

# если нам нужен явный токен бота,
# то нужно вручную вызвать `TelegramClient.start()`.
bot = TelegramClient('bot', api_id, api_hash).start(bot_token=bot_token)

# Тогда можем использовать экземпляр клиента как обычно.
with bot as bot_client:
    ...
```

Для получения аккаунта бота, необходимо поговорить с [@BotFather](#).

## Вход в систему через прокси-сервер.

Для доступа к Telegram через прокси сервер, необходим Python >= 3.6 и установка модуля python-socks[asyncio]. Затем в коде необходимо изменить строку с созданием клиента:

```
# было
TelegramClient('anon', api_id, api_hash)

# стало (замените протокол, IP и порт на собственные).
proxy=("socks5", '127.0.0.1', 4444)
TelegramClient('anon', api_id, api_hash, proxy=proxy)
```

Аргумент proxy должен быть словарем (или кортежем для обратной совместимости), состоящим из параметров, как в описании [модуля PySocks](#).

До 

Вверх

 значения аргумента proxy\_type:

- socks.SOCKS5 или 'socks5'
- socks.SOCKS4 или 'socks4'
- socks.HTTP или 'http'
- python\_socks.ProxyType.SOCKS5
- РЕКЛАМА python\_socks.ProxyType.SOCKS4
- python\_socks.ProxyType.HTTP

Пример:

```
proxy = {
    # обязательный протокол
    'proxy_type': 'socks5',
    # обязательный IP-адрес прокси
    'addr': '1.1.1.1',
    # обязательный номер порта прокси-сервера
    'port': 5555,
    # имя пользователя, если прокси требует авторизации (необязательно)
    'username': 'foo',
    # пароль, если прокси-сервер требует авторизации (необязательно)
    'password': 'bar',
    # использовать удаленное или локальное разрешение, по умолчанию удаленное (необязательно)
    'rdns': True
}
client = TelegramClient('anon', api_id, api_hash, proxy=proxy)
```

Для обратной совместимости с PySocks возможен следующий формат (но не рекомендуется):

```
proxy = (socks.SOCKS5, '1.1.1.1', 5555, True, 'foo', 'bar')
client = TelegramClient('anon', api_id, api_hash, proxy=proxy)
```

## Использование прокси-серверов MTPROTO.

Прокси MTPROTO - это альтернатива Telegram обычным прокси, работающая немного по-другому. Доступны следующие протоколы:

- ConnectionTcpMTPROXYAbridged;
- ConnectionTcpMTPROXYIntermediate;
- ConnectionTcpMTPROXYRandomizedIntermediate (предпочтительный).

Если используется прокси-сервер MTPROTO, то на данный момент необходимо вручную указать эти специальные режимы подключения. В итоге код будет выглядеть следующим образом:

```
from telethon import TelegramClient, connection

# необходимо изменить соединение
client = TelegramClient(
    'anon',
    api_id,
    api_hash,

    # Используем один из доступных режимов подключения.
    # Обычно этот вариант работает с большинством прокси.
    connection=connection.ConnectionTcpMTPROXYRandomizedIntermediate,

    # Затем передайте данные прокси в виде кортежа:
    # (имя хоста, порт, секрет прокси)
    # Если у прокси нет секрета, то вместо него пишем:
    # '00000000000000000000000000000000'
    proxy=('mtproxy.example.com', 2002, 'secret')
)
```

В будущих обновлениях команда разработчиков telethon может упростить использование прокси-серверов MTPROTO (например, убрать аргумент connection).

Короче говоря, тот же код, который выше, но без комментариев, чтобы было понятнее:

```
from telethon import TelegramClient, connection

client = TelegramClient(
    'anon', api_id, api_hash,
    connection=connection.ConnectionTcpMTPROXYRandomizedIntermediate,
```

Вверх

```
    проxy=('mtproxy.example.com', 2002, 'secret')
)
```

# Дружественные методы работы с библиотекой telethon

РЕКЛАМА

Рассмотрим более длинный пример, с целью изучения некоторых методов, которые может предложить библиотека telethon. Они известны как "дружественные методы", и их необходимо использовать всегда, если это возможно.

Код ниже показывает, как войти в Telegram, получить информацию о себе, отправить сообщения, файлы, войти в чат, распечатать сообщения и загрузить файлы.

```
from telethon import TelegramClient

# подставляем собственные значения
api_id = 12345
api_hash = '0123456789abcdef0123456789abcdef'
client = TelegramClient('anon', api_id, api_hash)

async def main():
    # Получение информации о себе
    me = await client.get_me()

    # `me` - это пользовательский объект. Можно красиво напечатать
    # любой объект Telegram с помощью метода `.stringify`:
    print(me.stringify())

    # Можно получить доступ ко всем атрибутам объектов Telegram с помощью
    # оператора точки. Например, чтобы получить имя пользователя:
    username = me.username
    print(username)
    print(me.phone)

    # Можно распечатать все диалоги/разговоры, в которых вы участвуете:
    async for dialog in client.iter_dialogs():
        print(dialog.name, 'has ID', dialog.id)

    # Можно отправлять сообщения самому себе...
    await client.send_message('me', 'Hello, myself!')
    # ... к какому-нибудь идентификатору чата
    await client.send_message(-100123456, 'Hello, group!')
    # ... к каким-то контактам
    await client.send_message('+34600123123', 'Hello, friend!')
    # ... или даже к любому имени пользователя
    await client.send_message('username', 'Testing Telethon!')

    # конечно, можно использовать `markdown` в своих сообщениях:
    message = await client.send_message(
        'me',
        'This message has bold, `code`, italics and '
        'a [nice website](https://example.com)!',
        link_preview=False
    )

    # Отправка сообщения возвращает объект отправленного
    # сообщения, который можно использовать повторно
    print(message.raw_text)

    # можно отвечать на сообщения напрямую, если есть объект сообщения.
    await message.reply('Cool!')

    # или отправляйте файлы, музыку, документы, альбомы...
    await client.send_file('me', '/home/me/Pictures/holidays.jpg')

    # можно распечатать историю сообщений любого чата:
    async for message in client.iter_messages('me'):
        print(message.id, message.text)

    # можно также загружать мультимедиа из сообщений!
    # Метод вернет путь, по которому был сохранен файл.
    if message.photo:
        path = await message.download_media()
        # печатает после завершения загрузки
        print('File saved to', path)
```

Вверх



```
with client:
    client.loop.run_until_complete(main())
```

Прежде чем продолжить, необходимо понимать, что делает представленный код, обратить внимание на то, как вызываются и используются методы и т. д. :

**ВАЖНО!** Обратите внимание, что Telethon - это асинхронная библиотека, и поэтому к ней нужно привыкнуть и изучить основы модуля [asyncio](#). Это очень поможет. Для начала, можно написать весь свой код внутри некоторого асинхронного определения, например:

```
client = ...

async def do_something(me):
    ...

async def main():
    # Большая часть кода должна находиться здесь.
    # можно создать и использовать свою собственную асинхронную функцию (do_something).
    # функции должны быть асинхронными, если им нужно ждать получения данных.
    me = await client.get_me()
    await do_something(me)

with client:
    client.loop.run_until_complete(main())
```

После осознания всего этого, можно использовать (если хотите) хук [telethon.sync](#), но необходимо учитывать, что можно столкнуться с другими проблемами (у [iPython](#), [Anaconda](#) и т. д. есть некоторые проблемы с ним).

## Введение в получение обновлений.

Получение обновлений - важная тема для такой платформы обмена сообщениями, как Telegram. Всегда встает потребность получать уведомления, когда приходит новое сообщение, когда участник присоединяется, когда кто-то начинает печатать и т. д. Для этого можно использовать события events.

**Важно!** Настоятельно рекомендуется включать логирование при работе с событиями, так как исключения в обработчиках событий по умолчанию скрыты. Для этого необходимо добавить следующий фрагмент кода в самый верх сценария:

```
import logging
logging.basicConfig(format='[%(levelname)s] 5s/%(asctime)s] %(name)s: %(message)s',
                    level=logging.WARNING)
```

Начнем с примера автоматизации ответов:

```
from telethon import TelegramClient, events

client = TelegramClient('anon', api_id, api_hash)

@client.on(events.NewMessage)
async def my_event_handler(event):
    if 'hello' in event.raw_text:
        await event.reply('hi!')

client.start()
client.run_until_disconnected()
```

В этом небольшом фрагменте кода некоторые вещи могут быть неясны. Разберем его:

```
@client.on(events.NewMessage)
```

Этот декоратор Python прикреплен к определению функции my\_event\_handler() и по сути означает, что при событии NewMessage будет вызвана упомянутая функция обратного вызова:

```
async def my_event_handler(event):
    if 'hello' in event.raw_text:
        await event.reply('hi!')
```

Если происходит событие NewMessage и в тексте сообщения содержится слово *hello*, то отвечаем на это событие сообщением *hi!* event.reply('hi!').

Вверх **ние**. Обработчики событий должны быть асинхронными по определению. Telethon - это асинхронная библиотека, основанная на [asyncio](#), которая является более безопасным и зачастую более быстрым подходом чем [потoki threads](#).

Другими словами, необходимо дождаться `await` всех вызовов, использующих сетевые запросы, а это большинство из них.

## Еще примеры получения обновлений.

```
@client.on(events.NewMessage(outgoing=True, pattern=r'\.save'))
async def handler(event):
    if event.is_reply:
        replied = await event.get_reply_message()
        sender = replied.sender
        await client.download_profile_photo(sender)
        await event.respond(f'Saved your photo {sender.username}')
```

Событие в коде фильтрует сообщения (только те, которые отправляем `outgoing=True`, активируют метод), затем фильтруем по регулярному выражению `r'.save'`, которое будет соответствовать сообщениям, начинающимся с `'.save'`.

Внутри функции проверяем, отвечает ли событие на другое сообщение или нет. Если да, то получаем ответное сообщение и отправителя этого сообщения, а также загружаем его фотографию профиля.

Теперь удалим сообщения, содержащие слово `'heck'`. Например, если в чате запрещено ругаться.

```
@client.on(events.NewMessage(pattern=r'(?i).*heck'))
async def handler(event):
    await event.delete()
```

С помощью регулярного выражения `r'(?i).*heck'` сопоставляем слово `heck` без учета регистра в любом месте сообщения.

Здесь представлен только объект события `|events.NewMessage|`, но есть еще много других полезных событий.

Дополнительно смотрите материал ["Все о получении обновлений модулем Telethon"](#).

## Понятие сущности entity пакета Telethon.

В библиотеке широко используется [понятие entity "сущности"](#). Сущность будет ссылаться на любой объект `User`, `Chat` или `Channel`, который API может вернуть в ответ на определенные методы, такие как `GetUsersRequest`.

Когда нужен пользователь или чат, где произошло событие, необходимо использовать следующие методы:

```
async def handler(event):
    # Правильно
    chat = await event.get_chat()
    sender = await event.get_sender()
    chat_id = event.chat_id
    sender_id = event.sender_id

    # ПЛОХО. Не делайте этого
    chat = event.chat
    sender = event.sender
    chat_id = event.chat.id
    sender_id = event.sender.id
```

События похожи на сообщения, но не содержат всей информации, которую содержит сообщение! Когда получаем сообщение вручную, оно будет содержать всю необходимую информацию. Когда получаем обновление о сообщении, оно не содержит всей информации, поэтому придется использовать методы, а не свойства.

Запомните как правило: новые события сообщений ведут себя так же, как объекты сообщений, поэтому с ними можно делать все, что можно делать с объектом сообщения.

## Совместимость Telethon.

При разработке любой библиотеки, некоторые решения неизбежно окажутся неправильными в будущем. Одним из таких решений было использование потоков. Начиная с Python 3.6 стало возможным использование модуля `asyncio` с пользой для такой библиотеки, как Telethon.

Если приложение Telegram имеет старый код, просто используйте старые версии библиотеки! В этом нет ничего плохого, кроме отсутствия новых обновлений или исправлений, при этом возможно использовать исправленную версию с помощью `pip install telethon==0.19.1.6`.

Нет смысла поддерживать синхронную версию Telethon, т.к. опыт показывает, что у людей нет времени обновляться, а было несколько изменений и чисток. Использование более старой версии - правильный путь.

Вверх

Иногда принимаются и другие небольшие решения. Все это будет отражено в журнале изменений (истории версий), который следует прочитать при обновлении.

Если хотите окунуться в `asyncio`, вот некоторые вещи, которые понадобятся, чтобы начать миграцию действительно старого кода:

РЕКЛАМА

```
# 1. Импортируйте клиент из telethon.sync
from telethon.sync import TelegramClient

# 2. Измените этого монстра...
try:
    assert client.connect()
    if not client.is_user_authorized():
        client.send_code_request(phone_number)
        me = client.sign_in(phone_number, input('Enter code: '))

    ... # ОСТАЛЬНАЯ ЧАСТЬ КОДА
finally:
    client.disconnect()

# ...на следующую строку:
with client:
    ... # ОСТАЛЬНАЯ ЧАСТЬ КОДА

# 3. client.idle() больше не существует.
# Измени это...
client.idle()
# на следующую строку:
client.run_until_disconnected()

# 4. `client.add_update_handler` больше не существует.
# Измените это...
client.add_update_handler(handler)
# ...на следующую строку:
client.add_event_handler(handler)
```

Кроме того, все обработчики обновлений должны быть асинхронными, т.е. нужно ожидать вызовов методов, которые зависят от сетевых запросов, таких как получение чата или отправителя. Если обновления не используются, то все готово!

## Удобство Telethon.

Весь код ниже предполагает, что выполнено одно из следующих действий:

```
from telethon import TelegramClient, sync
# или
from telethon.sync import TelegramClient
```

Это делает примеры короче и облегчает размышление.

Для быстрых скриптов, не требующих обновлений (например публикация сообщений в канал Telegram), гораздо удобнее забыть об `asyncio` и просто работать с последовательным кодом. Такое поведение может оказаться мощным гибридом для работы под управлением Python REPL.

```
# Обратите внимание, что `telethon.sync`
# будет управлять циклом `asyncio`
from telethon.sync import TelegramClient

with TelegramClient(...) as client:
    # обратите внимание на отсутствие `await` или loop.run_until_complete().
    # т.к. цикл не выполняется, это делается за кулисами.
    print(client.get_me().username)

    message = client.send_message('me', 'Hi!')
    import time
    time.sleep(5)
    message.delete()

    # Также можно использовать гибрид синхронной
    # части и асинхронных обработчиков событий.
    from telethon import events
    @client.on(events.NewMessage(pattern='(?i)hi|hello'))
    def handler(event):
        wait event.reply('hey')
```

Вверх

```
client.run_until_disconnected()
```

Контекстный менеджер `with` и некоторые методы, такие как, `.start`, `.disconnect` и `.run_until_disconnected`, по умолчанию работают как в синхронном, так и в асинхронном контекстах для удобства и во избежание небольших накладных расходов при использовании таких методов, как отправка сообщений, получение сообщений и т.д. Это сохраняет лучшее из оба мира как разумный вариант по умолчанию.

*Примечание. Как правило, находясь внутри `async def` и при этом нужен клиент, то необходимо дождаться `await` вызовов API. Если вызываете другие функции, которым также требуются вызовы API, сделайте их `async def` и тоже ожидайте их. В противном случае в асинхронном режиме делать нечего.*

## Скорость и накладные расходы Telethon.

Когда будете готовы микро-оптимизировать свое приложение или просто не требуется вызывать какие-либо неосновные методы из синхронного контекста, то просто избавьтесь от импорта `telethon.sync` и работайте внутри `async def`:

Волшебный модуль `telethon.sync` по сути оборачивает каждый метод `asyncio.run(main())` с некоторыми другими хитростями. Это накладные расходы, которые платите, если импортируете его, и то, что экономите, если не делаете этого.

```
import asyncio
from telethon import TelegramClient, events

async def main():
    async with TelegramClient(...) as client:
        # обратите внимание на скобки, здесь ожидается
        # вызов `client.get_me()`, а не свойства `.username`.
        print((await client.get_me()).username)

        message = await client.send_message('me', 'Hi!')
        await asyncio.sleep(5)
        await message.delete()

        @client.on(events.NewMessage(pattern='(?i)hi|hello'))
        async def handler(event):
            await event.reply('hey')

        await client.run_until_disconnected()

asyncio.run(main())
```

Библиотека везде использует `asyncio`. Хотите научиться делать все правильно? Несмотря на то, что модулю [asyncio посвящен отдельный раздел](#) на <https://docs-python.ru>, в документации к библиотеке Telethon существует материал под названием [Освоение asyncio для использования с Telethon](#), который познакомит с миром `asyncio`.

### Содержание раздела:

- [КРАТКИЙ ОБЗОР МАТЕРИАЛА.](#)
- [Понимание сущностей entity библиотеки Telethon](#)
- [Все о получении обновлений модулем Telethon](#)
- [Файл сессии сеанса модуля Telethon](#)
- [Понятие типов Chats и Channels в модуле Telethon](#)
- [Освоение asyncio для использования с Telethon](#)
- [Переход с Telegram Bot API на библиотеку Telethon](#)
- [Ошибки RPCError модуля Telethon](#)
- [События events модуля Telethon](#)