Материалы к занятию

Следующие методы позволяют изменить существующее сообщение в истории сообщений вместо отправки нового сообщения с результатом действия. Это наиболее полезно для сообщений со встроенной клавиатурой, использующих запросы обратного вызова, но также может помочь уменьшить беспорядок в разговорах с обычными чат-ботами.

Пожалуйста, обратите внимание, что в настоящее время редактировать сообщения можно только без reply_markup или с помощью встроенных клавиатур.

editMessageText - Используйте этот метод для редактирования текстовых сообщений, отправленных ботом или через бота (для встроенных ботов). При успешном выполнении возвращается отредактированное сообщение.

chat_id - Требуется, если не указан идентификатор inline_message_id. Уникальный идентификатор целевого чата или имя пользователя целевого канала (в формате @channelusername)
message_id - Требуется, если не указан идентификатор inline_message_id. Уникальный идентификатор отправленного сообщения
inline_message_id - Требуется, если не указаны chat_id и message_id. Идентификатор встроенного сообщения
text - Новый текст сообщения
parse_mode - Отправьте Markdown или HTML, если вы хотите, чтобы приложения Telegram отображали жирный, курсивный, текст фиксированной ширины или встроенные URL-адреса в сообщении вашего бота.
disable_web_page_preview - Отключает предварительный просмотр ссылок для ссылок в этом сообщении
reply_markup - Объект, сериализованный в формате JSON для встроенной клавиатуры.

editMessageCaption - Используйте этот метод для редактирования подписей сообщений, отправленных ботом или через бота (для встроенных ботов). При успешном выполнении возвращается отредактированное сообщение.

chat_id - Требуется, если не указан идентификатор inline_message_id. Уникальный идентификатор целевого чата или имя пользователя целевого канала (в формате @channelusername)
message_id - Требуется, если не указан идентификатор inline_message_id. Уникальный идентификатор отправленного сообщения
inline_message_id - Требуется, если не указаны chat_id и message_id. Идентификатор встроенного сообщения
caption - Новая подпись к сообщению

reply_markup - Объект, сериализованный в формате JSON для встроенной клавиатуры.

editMessageReplyMarkup - Используйте этот метод для редактирования только разметки ответов сообщений, отправленных ботом или через бота (для встроенных ботов). При успешном выполнении возвращается отредактированное сообщение.
chat_id - Требуется, если не указан идентификатор inline_message_id. Уникальный идентификатор целевого чата или имя пользователя целевого канала (в формате @channelusername)
message_id - Требуется, если не указан идентификатор inline_message_id. Уникальный идентификатор отправленного сообщения
inline_message_id - Требуется, если не указаны chat_id и message_id. Идентификатор встроенного сообщения
reply_markup - Объект, сериализованный в формате JSON для встроенной клавиатуры.

У ботов есть помимо обычного еще и инлайн режим.
Инлайн-режим (inline mode) — это специальный режим работы бота, с помощью которого пользователь может использовать бота во всех чатах.
Выглядит это так: пользователь вводит юзернейм бота в поле для ввода сообщения. После юзернейма можно ещё записать запрос (текст до 256 символов).
Появляется менюшка с результатами. Выбирая результат, пользователь отправляет сообщение.
Инлайн-режим можно включить в BotFather, там же можно выбрать плейсхолдер вместо стандартного "Search..."
В группе можно запретить использовать инлайн всем или некоторым участникам. В официальных приложениях Телеграм это ограничение объединено с ограничением на отправку стикеров и GIF.

Когда пользователь вызывает инлайн-режим, бот не может получить никакую информацию о контексте, кроме информации о пользователе. Таким образом, бот не может узнать ни чат, в котором вызвали инлайн, ни сообщение, на которое пользователь отвечает.
Но зато если включить в BotFather настройку "Inline Location Data", то бот сможет видеть геопозицию пользователей, когда они используют инлайн (на мобильных устройствах). Перед этим у пользователей показывается предупреждение.

С инлайн режимом мы работать не будем, но знать что он существует вы должны.

InlineQuery - Этот объект представляет входящий встроенный запрос. Когда пользователь отправляет пустой запрос, ваш бот может вернуть некоторые результаты по умолчанию или трендовые результаты.

id - Уникальный идентификатор для этого запроса

from - Отправитель
location - Уникальный идентификатор для этого местоположения отправителя запроса, только для ботов, которые запрашивают местоположение пользователя
query - Текст запроса
offset - Смещение результатов, которые будут возвращены, может контролироваться ботом

answerInlineQuery - Используйте этот метод для отправки ответов на встроенный запрос. В случае успеха возвращается значение True.
Допускается не более 50 результатов на запрос.

inline_query_id - Уникальный идентификатор для ответа на запрос
results - Сериализованный в формате JSON массив результатов для встроенного запроса
cache_time - Максимальное время в секундах, в течение которого результат встроенного запроса может кэшироваться на сервере. Значение по умолчанию равно 300.
is_personal - Передайте значение True, если результаты могут быть кэшированы на стороне сервера только для пользователя, отправившего запрос. По умолчанию результаты могут быть возвращены любому пользователю, отправившему тот же запрос
next_offset - Передайте смещение, которое клиент должен отправить в следующем запросе с тем же текстом, чтобы получить больше результатов. Передайте пустую строку, если результатов больше нет или если вы не поддерживаете разбивку на страницы. Длина смещения не может превышать 64 байта
switch_pm_text - Если он передан, клиенты отобразят кнопку с указанным текстом, которая переключает пользователя в приватный чат с ботом и отправляет боту стартовое сообщение с параметром switch_pm_parameter
switch_pm_parameter - Параметр для начального сообщения, отправляемого боту при нажатии пользователем кнопки переключения

InlineQueryResult - Этот объект представляет один результат встроенного запроса. Клиенты Telegram в настоящее время поддерживают результаты следующих 19 типов:

- InlineQueryResultCachedAudio
- InlineQueryResultCachedDocument
- InlineQueryResultCachedGif
- InlineQueryResultCachedMpeg4Gif
- InlineQueryResultCachedPhoto
- InlineQueryResultCachedSticker
- InlineQueryResultCachedVideo
- InlineQueryResultCachedVoice
- InlineQueryResultArticle
- InlineQueryResultAudio
- InlineQueryResultContact
- InlineQueryResultDocument

- InlineQueryResultGif
- InlineQueryResultLocation
- InlineQueryResultMpeg4Gif
- InlineQueryResultPhoto
- InlineQueryResultVenue
- InlineQueryResultVideo
- InlineQueryResultVoice

Более подробно о работе с inline режимом вы можете ознакомиться в официальной документации

На прошлых занятиях мы немного рассмотрели как работать через Bot Api без использования библиотек, но на сегодняшний день огромное количество различных библиотек, которые максимально удобно реализуют взаимодействие с Bot Api. Нам же остается изучить методы и классы какой либо библиотеки, для того чтобы начать работать.

Мы с вами начнем знакомство с библиотекой aiogram. В отличие от других библиотек aiogram - асинхронная библиотека.
Асинхронное программирование — это особенность современных языков программирования, которая позволяет выполнять операции, не дожидаясь их завершения.

Для начала давайте создадим каталог для бота, организуем там virtual environment (далее venv) и установим библиотеку aiogram

Теперь установим модуль:
pip install aiogram
Также нам понадобится библиотека python-dotenv для файлов конфигурации
pip install python-dotenv

Давайте создадим файл bot.py с базовым шаблоном бота на aiogram

```
import asyncio
import logging
from aiogram import Bot, Dispatcher, types

# Включаем логирование, чтобы не пропустить важные сообщения
logging.basicConfig(level=logging.INFO)
# Объект бота
bot = Bot(token="Ваш токен")
# Диспетчер
dp = Dispatcher(bot)
```

```
# Хэндлер на команду /start
@dp.message_handler(commands=["start"])
async def cmd_start(message: types.Message):
   await message.answer("Hello!")


# Запуск процесса поллинга новых апдейтов
async def main():
   await dp.start_polling(bot)


if __name__ == "__main__":
   asyncio.run(main())
```

Первое, на что нужно обратить внимание: aiogram — асинхронная библиотека, поэтому ваши хэндлеры тоже должны быть асинхронными, а перед вызовами методов API нужно ставить ключевое слово await, т.к. эти вызовы возвращают корутины.

Под handler обычно подразумевается обработчик чего-то (каких-то событий, входящих соединений, сообщений и т.д.)

Корутина (coroutine) - подпрограмма (функция), которая может начинаться, приостанавливаться и завершаться в произвольный момент времени. Корутины описываются синтаксисом async/await

Диспетчер регистрирует функции-обработчики, дополнительно ограничивая перечень вызывающих их событий через фильтры. После получения очередного апдейта (события от Telegram), диспетчер выберет нужную функцию обработки, подходящую по всем фильтрам, например, «обработка сообщений, являющихся изображениями, в чате с ID икс и с длиной подписи игрек». Если две функции имеют одинаковые по логике фильтры, то будет вызвана та, что зарегистрирована раньше.

Чтобы зарегистрировать функцию как обработчик сообщений, нужно сделать одно из двух действий:
1. Навесить на неё декоратор, как в примере выше. С различными типами декораторов мы познакомимся позднее.
2. Напрямую вызвать метод регистрации у диспетчера или роутера.
Рассмотрим следующий код:

```
import asyncio
import logging
from aiogram import Bot, Dispatcher, types

# Включаем логирование, чтобы не пропустить важные сообщения
logging.basicConfig(level=logging.INFO)
```

```
# Объект бота
bot = Bot(token="Ваш токен")
# Диспетчер
dp = Dispatcher(bot)


# Хэндлер на команду /start
@dp.message_handler(commands=["start"])
async def cmd_start(message: types.Message):
   await message.answer("Hello!")


# Хэндлер на команду /test1
@dp.message_handler(commands=["test1"])
async def cmd_test1(message: types.Message):
   await message.reply("Test 1")


# Хэндлер на команду /test2
async def cmd_test2(message: types.Message):
   await message.reply("Test 2")


# Запуск процесса поллинга новых апдейтов
async def main():
   await dp.start_polling(bot)


if __name__ == "__main__":
   asyncio.run(main())
```

Хэндлер cmd_test2 не сработает, т.к. диспетчер о нём не знает. Исправим эту ошибку и отдельно зарегистрируем функцию:

```
import asyncio
import logging
from aiogram import Bot, Dispatcher, types

# Включаем логирование, чтобы не пропустить важные сообщения
logging.basicConfig(level=logging.INFO)
# Объект бота
bot = Bot(token="Ваш токен")
# Диспетчер
dp = Dispatcher(bot)


# Хэндлер на команду /start
@dp.message_handler(commands=["start"])
async def cmd_start(message: types.Message):
   await message.answer("Hello!")


# Хэндлер на команду /test1
@dp.message_handler(commands=["test1"])
async def cmd_test1(message: types.Message):
```

```
   await message.reply("Test 1")


# Хэндлер на команду /test2
async def cmd_test2(message: types.Message):
   await message.reply("Test 2")

dp.register_message_handler(cmd_test2, commands="test2")
# Запуск процесса поллинга новых апдейтов
async def main():
   await dp.start_polling(bot)


if __name__ == "__main__":
   asyncio.run(main())
```

Рассмотрим еще пример простого эхо бота

```
from aiogram import Bot, types
from aiogram.dispatcher import Dispatcher
from aiogram.utils import executor



bot = Bot(token='Ваш токен')
dp = Dispatcher(bot)


@dp.message_handler(commands=['start'])
async def process_start_command(message: types.Message):
   await message.reply("Привет!\nНапиши мне что-нибудь!")


@dp.message_handler(commands=['help'])
async def process_help_command(message: types.Message):
   await message.reply("Напиши мне что-нибудь, и я отпрпавлю этот текст
тебе в ответ!")


@dp.message_handler()
async def echo_message(msg: types.Message):
   await bot.send_message(msg.from_user.id, msg.text)


if __name__ == '__main__':
   executor.start_polling(dp)
```