tgbot

Введение.

tgbot - это библиотека на C++ для создания ботов Telegram. Она предоставляет разработчикам удобные инструменты для взаимодействия с Telegram Bot API, что упрощает процесс разработки ботов.

Задачи и цели

Основные задачи и цели использования tgbot включают:

- Автоматизация задач в Telegram.
- Создание интерактивных чат-ботов для различных целей (поддержка клиентов, игры, напоминания и т.д.).
- Интеграция Telegram ботов с внешними сервисами и базами данных.
- Обеспечение удобного интерфейса взаимодействия между пользователями и ботом.

Создание базового бота через BotFather;

Для создания бота в Telegram необходимо выполнить следующие шаги:

1. **Открытие чата с BotFather**: Найдите в Telegram пользователя @BotFather и начните с ним чат.
2. **Создание нового бота**:
    - Отправьте команду /newbot для начала процесса создания нового бота.
    - Следуйте инструкциям, предоставленным BotFather, такими как указание имени и имени пользователя (username) бота.
3. **Получение токена**: После успешного создания бота BotFather предоставит вам токен. Этот токен выглядит как длинная строка символов и используется для аутентификации запросов к Telegram Bot API.

API/ТОКЕН.

API (Application Programming Interface) - это набор правил и протоколов, позволяющих одной программе взаимодействовать с другой. **Telegram Bot API** - это интерфейс, предоставляемый Telegram, который позволяет разработчикам создавать программы, взаимодействующие с Telegram.

Токен - это уникальный ключ, который используется для аутентификации и авторизации запросов к Telegram Bot API. Он связывает ваш бот с вашим аккаунтом разработчика в Telegram и подтверждает, что запросы к API делаются от имени вашего бота.

пример токена:

```
123456789:ABCDEF1234ghIkl-zyx57W2v1u123ew11
```

Создание/Компиляция.

Создание проекта - через create в соответствующей IDE.

Компиляция проекта через GCC.

g++ main.cpp -o main --std=c++17 -I/usr/local/include -lTgBot -lboost_system -lssl -lcrypto -lpthread -lcurl

Запуск исполняемого файла.

./main

ресурс: https://disk.yandex.ru/d/BZVY2CDdRBh7bA

Основные объекты.

TgBot::Bot:

- Основной объект, представляющий бота. Используется для настройки и запуска бота.
- *Пример:* TgBot::Bot bot("...");

TgBot::Api:

- Объект, предоставляющий методы для взаимодействия с Telegram API.
- *Пример:* bot.getApi().sendMessage(chat_id, "Hello!");

TgBot::EventHandler:

- Объект, который обрабатывает события, такие как получение сообщений или команд.
- *Пример:* bot.getEvents().onCommand("start", callback_function);

TgBot::Message:

- Объект, представляющий сообщение в Telegram.
- *Пример:* message->chat->id

TgBot::TgLongPoll:

- Объект, который используется для опроса новых сообщений.
- *Пример:* TgBot::TgLongPoll longPoll(bot);

Пример обработка команды /help.

```
bot.getEvents().onCommand("help", [&bot](TgBot::Message::Ptr message) {

   bot.getApi().sendMessage(message->chat->id, "Доступные команды:\n/start - Запустить бота\n/help - Помощь\n/echo - Эхо-сообщение");

});
```

Пример обработка команды /echo.

для повторения сообщения

```
bot.getEvents().onCommand("echo", [&bot](TgBot::Message::Ptr message) {

   std::string text = message->text.substr(6); // Убираем "/echo "

   bot.getApi().sendMessage(message->chat->id, text);

});
```

Пример.

```
try {
    std::cout << "Имя бота: " << bot.getApi().getMe()->username.c_str() << std::endl;
    TgBot::TgLongPoll longPoll(bot);
    while (true) {
        std::cout << "Ожидание сообщений..." << std::endl;
        longPoll.start();
    }
} catch (TgBot::TgException& e) {
    std::cerr << "Ошибка: " << e.what() << std::endl;
}
```

***try-catch блок**: Обработка возможных ошибок.*
***bot.getApi().getMe()->username.c_str()**: Получение имени бота.*
***TgBot::TgLongPoll longPoll(bot)**: Создание объекта для опроса.*
***Цикл while (true)**: Бесконечный опрос сообщений.*
***longPoll.start()**: Ожидание и получение новых сообщений.*
***catch (TgBot::TgException& e)**: Обработка исключений и вывод ошибки.*

Базовый пример кода.

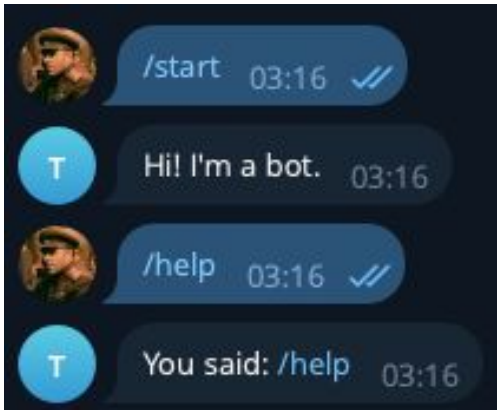
```cpp
#include <tgbot/tgbot.h>

int main() {
    TgBot::Bot bot("...");

    // Обработка /start
    bot.getEvents().onCommand("start", [&bot](TgBot::Message::Ptr message) {
        bot.getApi().sendMessage(message->chat->id, "Hi! I'm a bot.");
    });

    // Обработка всех сообщений
    bot.getEvents().onAnyMessage([&bot](TgBot::Message::Ptr message) {
        if (message->text == "/start") return;
        bot.getApi().sendMessage(message->chat->id, "You said: " + message->text);
    });

    try {
        TgBot::TgLongPoll longPoll(bot);
        while (true) longPoll.start();
    } catch (TgBot::TgException& e) {
        cerr << "error: " << e.what() << endl;
    }
    return 0;
}
```



Список основных функций #1.

getMe

- **Описание**: Получение информации о боте.
- **Пример использования**: bot.getApi().getMe()

sendMessage

- **Описание**: Отправка текстового сообщения пользователю или в чат.
- **Пример использования**: bot.getApi().sendMessage(chat_id, "Hello!")

forwardMessage

- **Описание**: Пересылка сообщения из одного чата в другой.
- **Пример использования**: bot.getApi().forwardMessage(chat_id, from_chat_id, message_id)

sendPhoto

- **Описание**: Отправка фотографии.
- **Пример использования**: bot.getApi().sendPhoto(chat_id, "photo.jpg")

sendAudio

- **Описание**: Отправка аудиофайла.
- **Пример использования**: bot.getApi().sendAudio(chat_id, "audio.mp3")

Список основных функций #2.

sendDocument

- **Описание**: Отправка документа.
- **Пример использования**: bot.getApi().sendDocument(chat_id, "document.pdf")

sendVideo

- **Описание**: Отправка видеоролика.
- **Пример использования**: bot.getApi().sendVideo(chat_id, "video.mp4")

sendAnimation

- **Описание**: Отправка анимации (GIF).
- **Пример использования**: bot.getApi().sendAnimation(chat_id, "animation.gif")

sendVoice

- **Описание**: Отправка голосового сообщения.
- **Пример использования**: bot.getApi().sendVoice(chat_id, "voice.ogg")

sendVideoNote

- **Описание**: Отправка видео заметки.
- **Пример использования**: bot.getApi().sendVideoNote(chat_id, "video_note.mp4")

Список основных функций #3.

sendLocation

- **Описание**: Отправка местоположения.
- **Пример использования**: bot.getApi().sendLocation(chat_id, latitude, longitude)

sendChatAction

- **Описание**: Отправка статуса действия (печатает, загружает фото и т.д.).
- **Пример использования**: bot.getApi().sendChatAction(chat_id, "typing")

Функции управления обновлением;

1. **getUpdates**
   - **Описание**: Получение обновлений для бота.
   - **Пример использования**: bot.getApi().getUpdates()
2. **setWebhook**
   - **Описание**: Установка вебхука для получения обновлений.
   - **Пример использования**: bot.getApi().setWebhook("https://example.com/webhook")
3. **deleteWebhook**
   - **Описание**: Удаление установленного вебхука.
   - **Пример использования**: bot.getApi().deleteWebhook()

Функции управления чатами #1;

getChat

- **Описание**: Получение информации о чате.
- **Пример использования**: bot.getApi().getChat(chat_id)

getChatAdministrators

- **Описание**: Получение списка администраторов чата.
- **Пример использования**: bot.getApi().getChatAdministrators(chat_id)

getChatMember

- **Описание**: Получение информации об участнике чата.
- **Пример использования**: bot.getApi().getChatMember(chat_id, user_id)

kickChatMember

- **Описание**: Исключение пользователя из чата.
- **Пример использования**: bot.getApi().kickChatMember(chat_id, user_id)

unbanChatMember

- **Описание**: Разблокировка пользователя в чате.
- **Пример использования**: bot.getApi().unbanChatMember(chat_id, user_id)

Функции управления чатами #2;

restrictChatMember

- **Описание**: Ограничение прав пользователя в чате.
- **Пример использования**: bot.getApi().restrictChatMember(chat_id, user_id, permissions)

promoteChatMember

- **Описание**: Назначение пользователя администратором чата.
- **Пример использования**: bot.getApi().promoteChatMember(chat_id, user_id, canChangeInfo, canPostMessages, canEditMessages, canDeleteMessages, canInviteUsers, canRestrictMembers, canPinMessages, canPromoteMembers)

setChatAdministratorCustomTitle

- **Описание**: Установка кастомного заголовка для администратора.
- **Пример использования**: bot.getApi().setChatAdministratorCustomTitle(chat_id, user_id, custom_title)

setChatPermissions

- **Описание**: Установка прав для участников чата.
- **Пример использования**: bot.getApi().setChatPermissions(chat_id, permissions)

curl.h

curl/curl.h — это заголовочный файл из библиотеки libcurl, который предоставляет функции для работы с сетевыми протоколами по передаче данных, такими как HTTP и FTP. Он является частью библиотеки libcurl, которая позволяет выполнять HTTP-запросы, загружать и отправлять файлы, работать с HTTPS и многими другими сетевыми протоколами.

Основные возможности curl/curl.h:

1. **HTTP-запросы**: Выполнение GET, POST, PUT, DELETE и других HTTP-запросов к серверам.
2. **Загрузка и отправка файлов**: Возможность загрузки и отправки файлов через сетевые протоколы.
3. **HTTPS**: Поддержка защищенной передачи данных через HTTPS с использованием SSL/TLS.
4. **Множество протоколов**: Поддержка множества сетевых протоколов, таких как FTP, SCP, SMTP и многих других.

Пример - обработка текстового документа;

```
// Функция для загрузки файла с Telegram серверов
size_t WriteCallback(void* contents, size_t size, size_t nmemb,  void* userp) {
   ((std::string*)userp)->append(( char*)contents, size * nmemb);
   return size * nmemb;
}

std::string downloadFile(const std::string& fileUrl) {
   CURL* curl;
   CURLcode res;
   std::string readBuffer;
   curl = curl_easy_init();
   if (curl) {
       curl_easy_setopt(curl, CURLOPT_URL, fileUrl.c_str());
       curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,   WriteCallback);
       curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
       res = curl_easy_perform(curl);
       curl_easy_cleanup(curl);
   }
   return readBuffer;
}
```

#include <curl/curl.h>

main();

```
TgBot::Bot bot("тут токен");

bot.getEvents().onCommand("start", [&bot](TgBot::Message::Ptr message) {
   bot.getApi().sendMessage(message->chat->id,"Привет! Отправьте мне .txt файл.");
});

bot.getEvents().onAnyMessage([&bot](TgBot::Message::Ptr message) {
   if (message->document) {
       std::string fileId = message->document->fileId;
       TgBot::File::Ptr file = bot.getApi().getFile(fileId);
       std::string filePath = file->filePath;
       std::string fileUrl ="https://api.telegram.org/file/bot"+ std::string("тут токен") + "/" + filePath;
       std::string fileContent = downloadFile(fileUrl);
       // Сохранение файла локально
       std::ofstream outFile("downloaded_file.txt");
       outFile << fileContent;
       outFile.close();
       // Чтение/вывод содержимого файла
       std::istringstream fileStream(fileContent);
       std::string line;
       std::string fileText;
       while (std::getline(fileStream, line)) fileText += line +"\n";
       bot.getApi().sendMessage(message->chat->id,"Содержимое файла:\n" + fileText);
   } else {
       bot.getApi().sendMessage(message->chat->id,"Отправьте документ в формате .txt");
   }
});
...
try -catch {}
```

Пример - обработка изображения;

```
// Функция для загрузки файла с Telegram серверов
size_t WriteCallback(void* contents, size_t size, size_t nmemb,  void* userp) {
   ((std::string*)userp)->append(( char*)contents, size * nmemb);
   return size * nmemb;
}

std::string downloadFile(const std::string& fileUrl) {
   CURL* curl;
   CURLcode res;
   std::string readBuffer;
   curl = curl_easy_init();
   if (curl) {
       curl_easy_setopt(curl, CURLOPT_URL, fileUrl.c_str());
       curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,   WriteCallback);
       curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
       res = curl_easy_perform(curl);
       curl_easy_cleanup(curl);
   }
   return readBuffer;
}

void saveFile(const std::string& filename,  const std::string& content) {
   std::ofstream outFile(filename, std::ios::binary);
   outFile.write(content.c_str(), content.size());
   outFile.close();
}
```

main();

```
TgBot::Bot bot("тут токен");

bot.getEvents().onCommand("start", [&bot](TgBot::Message::Ptr message) {
   bot.getApi().sendMessage(message->chat->id,   "Привет! Отправьте мне изображение." );
});

bot.getEvents().onAnyMessage([&bot](TgBot::Message::Ptr message) {
   if (!message->photo.empty()) {
       TgBot::PhotoSize::Ptr photo = message->photo.back();    // Получаем наибольшее изображение
       std::string fileId = photo->fileId;
       TgBot::File::Ptr file = bot.getApi().getFile(fileId);
       std::string filePath = file->filePath;
       std::string fileUrl =  "https://api.telegram.org/file/bot"  + std::string("тут токен") + "/" + filePath;

       std::string fileContent = downloadFile(fileUrl);

       // Сохранение изображения локально
       saveFile("downloaded_image.jpg" , fileContent);

       bot.getApi().sendMessage(message->chat->id,   "Изображение сохранено как downloaded_image.jpg" );
   } else {
       bot.getApi().sendMessage(message->chat->id,   "Пожалуйста, отправьте изображение." );
   }
});
```

Пример - получение списка админов чата;

```
TgBot::Bot bot("тут токен");
bot.getEvents().onCommand( "admins", [&bot](TgBot::Message::Ptr message) {
   // Получаем ID чата, из которого пришло сообщение
   long long chat_id = message->chat->id;
   try {
       // Получаем список администраторов чата
       std::vector<TgBot::ChatMember::Ptr> administrators = bot.getApi().getChatAdministrators(chat_id);
       // Формируем сообщение с информацией об администраторах
       std::string response = "Администраторы чата: \n";
       for (const auto& admin : administrators) {
           response += admin->user->username + " - " + admin->status + "\n";
       }
       // Отправляем сообщение с информацией об администраторах обратно в чат
       bot.getApi().sendMessage(chat_id, response);
   } catch (TgBot::TgException& e) {
       std::cerr << "Ошибка при получении администраторов чата: " << e.what() << std::endl;
   }
});

try {...} catch (TgBot::TgException& e) {...}
return 0;
```

Пример - установка прав доступа участника;

```
TgBot::Bot bot("тут токен");
bot.getEvents().onCommand("set_permissions", [&bot](TgBot::Message::Ptr message) {
    // Получаем ID чата, из которого пришло сообщение
    long long chat_id = message->chat->id;
    try {
        // Создаем объект ChatPermissions и устанавливаем необходимые права
        TgBot::ChatPermissions permissions;
        permissions.canSendMessages = false;
        permissions.canSendMediaMessages = false;
        permissions.canSendPolls = false;
        permissions.canSendOtherMessages = false;
        permissions.canAddWebPagePreviews = false;
        permissions.canInviteUsers = false;
        // Устанавливаем новые права доступа для чата
        bot.getApi().setChatPermissions(chat_id, permissions);
        bot.getApi().sendMessage(chat_id, "Права доступа для чата успешно обновлены!");
    } catch (TgBot::TgException& e) {
        std::cerr << "Ошибка при установке прав доступа: " << e.what() << std::endl;
    }
});
try-catch ...;
```

Пример - исключение пользователя из чата;

```
TgBot::Bot bot("тут токен");
bot.getEvents().onCommand("kick", [&bot](TgBot::Message::Ptr message) {
    // Получаем ID чата
    long long chat_id = message->chat->id;
    // Проверяем, есть ли упоминание пользователя для исключения
    if (message->text.empty() || message->text ==  "/kick") {
        bot.getApi().sendMessage(chat_id,  "Используйте команду в формате /kick @username для исключения пользователя."  );
        return;
    }
    // Получаем имя пользователя, которого нужно исключить
    std::string username_to_kick = message->text.substr( 6); // Убираем "/kick "
    try {
        // Выполняем запрос на исключение пользователя из чата
        bot.getApi().kickChatMember(chat_id, username_to_kick);
        bot.getApi().sendMessage(chat_id,  "Пользователь @"  + username_to_kick +  " был исключён из чата." );
    } catch (TgBot::TgException& e) {
        std::cerr <<  "Ошибка при исключении пользователя: "  << e.what() << std::endl;
        bot.getApi().sendMessage(chat_id,  "Ошибка при исключении пользователя: "  + std::string(e.what()));
    }
});
```