Fluent 1.0.0

Preface

Ushbu Fluent librarysi sizga Spring framework bilan birgalikda telegram bot yozishingizni ancha osonlashtiradi. Telegram bot yasash juda oson va chunarli sababi siz endi ushbu librarydan foydalanish orqali siz faqat annoatatsiyalarni o'zi bilan bot yozishingiz mumkin.

Ushbu Fluent library ichki `telegrambots` librarysidan foydalangan.

1. Set Up

Ushbu bo'limda kod yozishgacha bo'lgan jarayoncha nimalarni qilishingiz kerakligi bilib olasiz.

1.1 System Requirements

Ushbu fluent librarysidan foydalanish uchun Java 17 versiyasi va Spring boot-ni 3.1 dan baland versiyasi bo'lishi kerak.

1.2 Dependency

Fluent dan foydalanish uchun POM faylingizga quyidagi depedency qo'shishingiz kerak.

Maven

```
<dependency>
    <groupId>org.khasanof</groupId>
    <artifactId>spring-boot-starter-fluent</artifactId>
    <version>1.0.0</version>
</dependency>
```

Gradle

```
dependencies {
    implementation 'org.khasanof:spring-boot-starter-fluent:1.0.0'
}
```

1.3 Configuration

Tepadagi dependency-larni qo'shib bo'lganimizdan so'ng application yaml yoki properties filega botimizni token va username-ni ko'ratishimiz kerak bo'ladi.

properties

```
fluent.bot.token=<your bot token>
```

```
fluent.bot.username=<your bot username>
fluent.bot.process-type=both
```

yml

```
fluent:
  bot:
    token: <your bot token>
    username: <your bot username>
    process-type: both
```

`token` va `username` tashqari `process-type` ham ko'rsatilgan u haqida pastgi bo'limlar to'liq tushintirilgan.

2. Using Simple Examples

Ushbu bo'limda telegram botga kelgan updatelarni Handler-lar yordamida qanday ushlashni bilib olamiz.

2.1 Handle Message

Telegram botdan kelgan oddiy xabarni qanday qilib handle qilishni ushbu bo'limda ko'rib chiqamiz.

Quyidagi misolga qarang:

```
@UpdateController
public class TestController {

    @HandleMessage("/start")
    public void test(Update update, AbsSender sender) throws TelegramApiException {
        String text = "start : " + update.getMessage().getText();
        SendMessage message = new SendMessage(update.getMessage().getChatId().
toString(), text);
        sender.execute(message);
    }

}
```

`@UpdateController` annotatsiyasi updatelarni handle qiladigan classlarni ustiga qo'yish majburiy. Ushbu annotatsiya classni telegramdan kelgan update handle qilishi ko'rsatish uchun kerak.

`@HandleMessage` annotatsiyasiga kelsak. Ushbu annotatsiya oddiy message sifatida kelgan updatelarni berilgan qiymatga mos kelgan taqdirdagina methodga kiritadi. Yani botga `/start` xabar yuborilgandagina ushbu method ishlaydi.

`Update` esa bu bot-dan kirib kelgan update undan o'zimizga kerakli barcha narsalarni olib

ishlatishimiz mumkin.

`AbsSender` esa bu ushbu bot-dan kelgan update-ga javob tariqasida xabar yoki istalgan boshqa narsa yubormoqchi bo'lganimizda ishlatishimiz uchun kerak bo'ladigan jo'natuvchi yani sender.

2.2 Handle Callback

2.2 bo'limda esa callbacklarni qanday qilib handle qilishni bilib olasiz.

Quyidagi misolga qarang:

```
@UpdateController
public class TestController {

    @HandleCallback(values = {"RU", "UZ"})
    private void callBack(Update update, AbsSender sender) throws TelegramApiException
{
        String text = "I'm Callback Handler!";
        SendMessage message = new SendMessage(update.getCallbackQuery().getMessage()
            .getChatId().toString(), text);
        sender.execute(message);
    }

}
```

`@HandleCallback` annotatsiya nomidan ko'rinib turib callback sifatida kelgan update-larni handle qilish uchun ishlatishimiz mumkin. Ushbu annotatsiya `String[]` qabul qiladi yani qaysi callbacklarni handle qilishini ushbu values methoddiga berish orqali ko'rsatishimiz mumkin.

2.3 Callback and Message are multiple groups

Ushbu bo'limda tepada ishlatgan annotatsiyalarimizni bitta ko'plik variantlarni qanday yozishni bilib olamiz.

2.3.1 Handle Callbacks

Quyidagi misolga qarang:

```
@UpdateController
public class TestController {

    @HandleCallbacks(values = {
        @HandleCallback(values = {"NEXT", "PREV"}),
        @HandleCallback(values = {"TOP", "BOTTOM"}),
        @HandleCallback(values = {"LAST", "FIRST"})
    })
    private void multiCallback(Update update, AbsSender sender) throws
```

```
TelegramApiException {
        String text = "I'm Callback Handler!";
        SendMessage message = new SendMessage(update.getCallbackQuery().getMessage()
            .getChatId().toString(), text);
        sender.execute(message);
    }

}
```

`@HandleCallbacks` annotatsiyasi bir nechta `@HandleCallback` annotatsiyalarni guruhlash uchun ishlatiladi.

2.3.2 Handle Messages

Quyidagi misolga qarang:

```
@UpdateController
public class TestController {

     @HandleMessages(values = {
            @HandleMessage(value = "start", scope = MatchScope.START_WITH),
            @HandleMessage(value = "end", scope = MatchScope.END_WITH),
            @HandleMessage(value = "boom", scope = MatchScope.EQUALS_IGNORE_CASE)
    })
    void multiMessageHandler(Update update, AbsSender sender) throws
TelegramApiException {
        String text = "Hello Everyone! MultiHandler";
        SendMessage message = new SendMessage(update.getMessage().getChatId().
toString(), text);
        message.setReplyMarkup(enterMenu());
        sender.execute(message);
    }

}
```

`@HandleMessages` annotatsiyasi ham huddi `@HandleCallbacks` bilan bir xil bir nechta faqat `@HandleMessage` annotatsiyasi guruhlash uchun ishlatiladi.

3. Handler Parameters

Ahamiyat bergan bo'lsangiz kerak deyarli barcha handler-larning kirib keluvchi parameterlarni bir xil tepadagi misol larda. Ushbu bo'limda ushbu parameterlar haqida bilib olasiz.

3.1 Update Parameter

Ushbu `Update` classi telegramdan kiruvchi update yani yangilanishini ifodalaydi. Har qanday update da ushbu classni ichidagi istalgan ixtiyoriy parameterlardan faqat bittasi bo'lishi mumkin.

Ushbu update-dan o'zizmizga kerakli parameterlarni olib ishlatishimiz mumkin misol uchun xabar yuborish uchun yoki kirib kelgan update-dan text olish uchun va hokazo-lar uchun.

```
@HandleMessage("/start")
public void test(Update update, AbsSender sender) throws TelegramApiException {
    String text = "start : " + update.getMessage().getText(); ①
    SendMessage message = new SendMessage(update.getMessage().getChatId().toString(),
text); ②
    sender.execute(message); ③
}
```

① statementga e'tibor bersangiz kelgan updatedan messageni qanday qilib olish ko'rsatilgan.

② statementga e'tibor bersangiz ushbu statementda kelgan updatedan chatId olish ko'rsatilgan ushbu chatId bizga hali juda ko'p kerak bo'ladi sababi chatId orqali biz ushbu kirib kelgan updatega javob qaytarish uchun foydalanamiz.

③ ikkinchi va uchinchi statementda ko'rgan bo'lsangiz `SendMessage` classni instance yaratilgan chatId va textdan foydalanib va ushbu classni `AbsSender` classni execute method foydalanib botga xabar yuborilgan.

3.2 AbsSender Parameter

Ushbu `AbsSender` classi faqat bot ga xabar yuborish uchun ishlatiladi. Boshqacha qilib aytganda biz AbsSenderdan foydalanib istalga formatdagi xabarimizni botga yuborishimiz mumkin misol uchun text, video, rasm, audio va hokazo.

AbsSender class 2ta asosiy methodlari bor ular `execute` va `executeAsync`.

ikkala methodlarni bir birdan farqi shundaki `execute` method current threaddan foydalanib botga xabar yuboradi. `executeAsync` method esa boshqa thread ushbu ishni amalga oshiradi.

```
sender.executeAsync(message, new SentCallback<Message>() {
    @Override
    public void onResult(BotApiMethod<Message> method, Message response) {
        // ...
    }

    @Override
    public void onError(BotApiMethod<Message> method, TelegramApiRequestException
apiException) {
        // ...
    }

    @Override
    public void onException(BotApiMethod<Message> method, Exception exception) {
        // ...
    }
});
```

4. Send Message, Photo and etc

Ushbu 4 chi bo’lim botga har xil narsalarni yuborishni bilib olamiz.

4.1 Send Message

Bot ga oddiy xabar yuborishni tepadagi misollarda ko’p kordik. Shunga uni qayta ko’rishimiz shart emas. Endi kelgan update reply qilib javob yuborishni ko’ramiz, uning uchun biz ushbu `SendMessage` classni `setReplyToMessageId` method foydalanib reply xabar yuborishimiz mumkin.

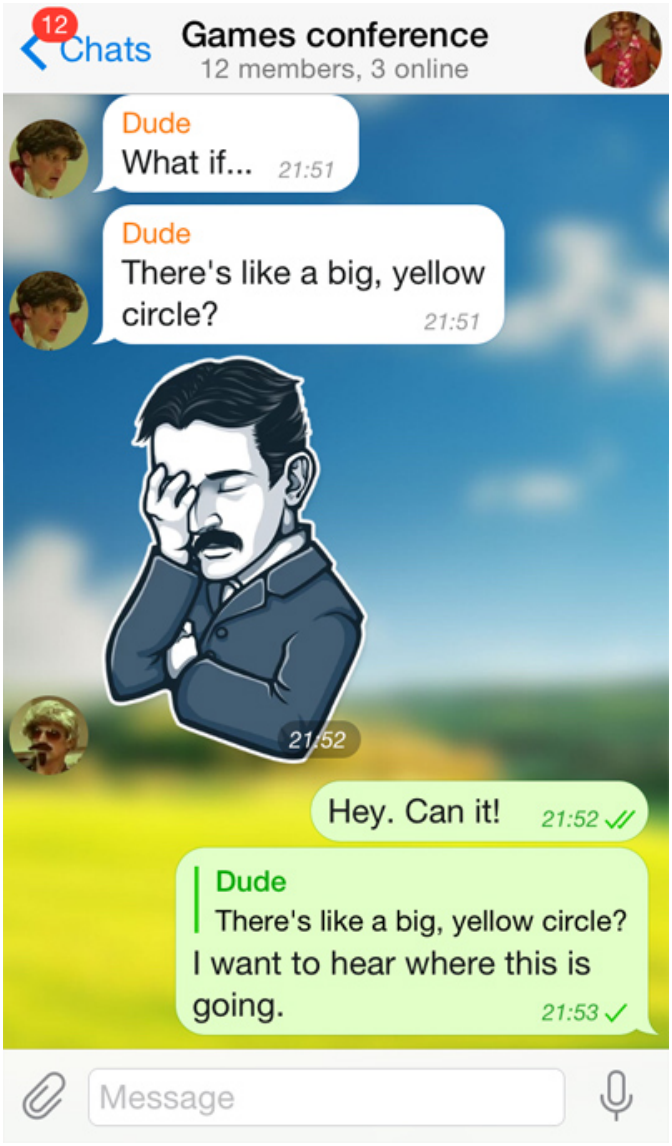

Figure 1. Reply Message Example

Following Example

```
SendMessage message = new SendMessage(update.getMessage().getChatId().toString(),
text);
message.setReplyToMessageId(update.getMessage().getMessageId()); ①
sender.execute(message);
```

① `setReplyToMessageId` method qaysi message reply belgilashimiz uchun ushbu messageId si kerak

bo’ladi. Ushu kodda kelgan updatedagi message reply qilib botga xabar yuborilgan.

4.1 Send Photo

Endi esa botga rasm yuborishni ko’ramiz.

Ushbu `SendPhoto` nomidan nima vazifa bajarishi ma’lum rasm yuborish uchun ishlatiladi. `SendPhoto` classi ham `SendMessage` juda o’xshaydi. `SendPhoto` ni farqi shundaki u text o’ringa InputFile classni qabul qiladi. InputFIle classi esa File qabul qiladi. Biz ushbu classga istalgan filimizni Urlni yoki `InpuStream` formatda file berib yuborishimiz mumkin.

Following Example

```
SendPhoto sendPhoto = new SendPhoto(update.getMessage().getChatId().toString(),
        new InputFile(new File("...")); ①
sender.execute(sendPhoto);
```

Send bilan boshlgan bir qancha classlardan foydalanib botga istalgan narsani yuborishimiz mumkin. Pastdagi rasmda `Send` bilan boshlangan class ro’yxati ko’rsatilgan biz ushbu classlar tepada ko’rsatilgan 2tasidan foydalanganimizga o’xshab foydalanishimiz mumkin. Qolgan `Send` classlarimizni pastdagi bo’limlarimizda birma bir ko’rsatib ketamiz.

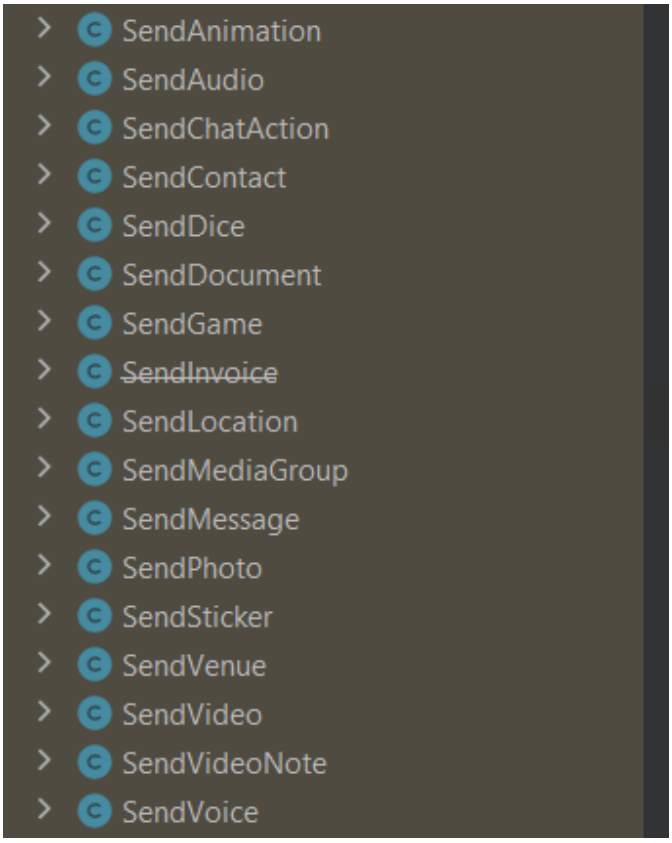

Figure 2. Send Classes

5. Annotations

Fluent telegramdan kelgan updatelarni handle qilish uchun quyidagi annotatsiyalarni qo'llab-quvvatlaydi.

5.1 Controller Annotations

Controller annotatsiyalari classlarni ustiga qoyiladigan annotatsiyalar, ushbu annotatsiyalarni qo'yishdan maqsad classni nima maqsad ishlatilishi belgilashdir. Misol uchun `@UpdateController` ushbu annotatsiyani bundan oldingi bo'limlarda ko'rdik ushbu annotatsiyani maqsadi classni telegramdan kelgan updatelarni handle qilishi ko'rsatish.

Annotation	Description
@UpdateController	Ushbu annotatsiya telegramdan kelgan updatelarni handle qilishini ko'rsatish uchun classlarga qo'yiladi. Huddi `@Controller` annotatsiyasiga o'xshaydi lekin ikklasi ham boshqa maqsadlarda ishlatiladi.
@ExceptionController	Ushbu annotatsiya esa Springdagi `@ControllerAdvice` maqsadi bir xil methodda sodir bo'lgan exceptionlarni ushlaydigan yani handle qiladigan class sifatida Spring ko'rsatish. `@ExceptionController` esa shuni Fluentga ko'rsatish uchun ishlatiladi.

5.2 Handler Annotations

Handler annotatsiyalari methodlarni ustiga qoyiladigan annotatsiyalar, ushbu annotatsiyalarni qo'yishdan maqsad telegramdan kelgan updatelarni methodga map qilishni yani kiritish ko'rstish uchun. Handler annotatsiyalarni Spring ni `@RequestMapping` va shunga o'xshash annotatsiyalariga juda o'xshaydi lekin ikklasi ham boshqa maqsadlarda ishlatiladi. Bittasi kelgan HTTP requestlarni handle qilsa, ikkinchi telegramdan kelgan updatelarni handle qilish uchun.

Annotation	Description
@HandleAny	Ushbu annotatsiya telegramdan kelgan istalgan update handle qilish uchun ishlatiladi. `@HandleAny` annotatsiya qo'yilgan method kelgan har qanday update handle qiladi.
@HandleMessage	Ushbu annotatsiya telegramdan text formatda kelgan updatelarni handle qilish uchun ishlatiladi. `@HandleMessage` annotatsiyasini bundan oldingi bo'limlarda ko'rdik va ushbu annotatsiyadan foydalanib o'zimiz istagan text formatdagi updatelarni qabul qila olishimiz mumkin.

Annotation	Description
@HandleCallback	Ushbu annotatsiya telegramdan callback formatda kelgan updatelarni handle qilish uchun ishlatiladi. `@HandleCallback` annotatsiyasidan foydalanib o'zingiz istagan callback formatdagi updatelarni qabul qilishingiz mumkin. Misol uchun aynan bitta formatga mos bo'lgan yoki bir nechta formatlarga mos bo'lgan.
@HandlePhoto	Ushbu annotatsiya telegramdan photo yani rasm formatda kelagn updatelarni handle qilish uchun ishlatiladi. `@HandlePhoto` annotatsiyasidan foydalanib biz rasm formatida kelagn updateni caption yoki sizega qarab filter qilishimiz mumkin. Shunda faqat o'zimizga kerak updatelarnigina qabul imkoniga ega bo'lamiz.
@HandleDocument	Ushbu annotatsiya telegramdan document formatda kelgan updatelarni handle qilish uchun ishlatiladi.
@HandleVideo	Ushbu annotatsiya telegramdan video formatda kelgan updatelarni handle qilish uchun ishlatiladi.
@HandleVideoNote	Ushbu annotatsiya telegramdan video note o'zbekchasiga etganda dumaloqcha video formatda kelgan updatelarni handle qilish uchun ishlatiladi.
@HandleException	Ushbu annotatsiya handler method birida exception sodir bo'lganda exception handle qilish uchun ishlatiladi. Springni `@ExceptionHandler` annotatsiyasiga o'xshash.
@BotVariable	Ushbu annotatsiya `@HandleMessage` annotatsiyasi bilan birgalikda ishlatiladi. Springni `@PathVariable` annotatsiya o'xshash. Ushbu annotatsiya haqida pastki bo'limlar to'liq ma'lumot misollari bilan birgalikda berilgan. Birinchi versiya uchun shu annotatsiyalar.

5.3 @HandleAny

`@HandleAny` annotatsiyasi istalgan update handle qilish uchun ishlatiladi. `@HandleAny` boshqa handler annotatsiyalaridan birinchi ishlaydi yani kelgan update birinchi `@HandleAny` annotatsiya bor method kiradi undan keyin boshqa handler methodlarga birin ketin kirishni boshlaydi.

`@HandleAny` annotatsiyasi 2ta parameter qabul qiladi.

1. type - orqali biz qaysi typedagi updatelarni handle qilishni ko'rsatish uchun ishlatishimiz mumkin. default holatda biz `@HandleAny` type ko'rsatmasak HandleType.MESSAGE ni oladi.

misol uchun quyidagi codega qarang.

```
@HandleAny(type = HandleType.MESSAGE, proceed = Proceed.PROCEED)
private void handleAnyMessagesV2(Update update, AbsSender sender) {
    String chatId = update.getMessage().getChatId().toString();
    SendMessage message = new SendMessage(chatId, "Handler Any Message");
    sender.execute(message);
}

@HandleAny(type = HandleType.STICKER, proceed = Proceed.NOT_PROCEED)
private void handleAnyStickers(Update update, AbsSender sender) {
    String chatId = update.getMessage().getChatId().toString();
    SendMessage message = new SendMessage(chatId, "Handler Any Sticker");
    sender.execute(message);
}

@HandleAny(type = HandleType.PHOTO, proceed = Proceed.PROCEED)
private void handleAnyPhoto(Update update, AbsSender sender) {
    String chatId = update.getMessage().getChatId().toString();
    SendMessage message = new SendMessage(chatId, "Handler Any Photo");
    sender.execute(message);
}

@HandleAny(type = HandleType.DOCUMENT, proceed = Proceed.PROCEED)
private void handleAnyDocument(Update update, AbsSender sender) {
    String chatId = update.getMessage().getChatId().toString();
    SendMessage message = new SendMessage(chatId, "Handler Any Document");
    sender.execute(message);
}
```

Tepadagi code da hal formatda kelgan updatelarni `@HandleAny` annotatsiyasi orqali qanday handle qilish ko'rsatilgan

2.. proceed - orqali biz `@HandleAny` annotatsiyasi qoyilgan method bajarilgandan so'ng undan keyingi handler methodlar bajarilishi yoki bajarilmasligini belgilashimiz mumkin. Agar Proceed.PROCEED turgan bo'lsa o'zidan keyingi method bajarilishiga ruhsat beradi. Agar aksi bo'lsa unda o'zidan keyingi handler methodlarni bajarilishiga ruhsat bermaydi. Qiymat belgilanmagan holda `@HandleAny` type parameteri *HandleType.MESSAGE* ni, proceed parameteri esa *Proceed.PROCEED* oladi.

Quyidagi misolga qarang.

```
@UpdateController
public class SimpleController {
```

```
    @HandleAny
    void handleAnyMessage(Update update, AbsSender sender) {
        String chatId = update.getMessage().getChatId().toString();
        SendMessage message = new SendMessage(chatId, "Handler Any Message");
        sender.execute(message);
    }

    @HandleMessage(value = "abs", scope = MatchScope.START_WITH)
    void startWithAbsHandler(Update update, AbsSender sender) {
        String chatId = update.getMessage().getChatId().toString();
        String text = "Start With 'abs' : " + update.getMessage().getText();
        SendMessage message = new SendMessage(chatId, text);
        sender.execute(message);
    }

}
```

Figure 3. Handle Any Proceed Test

Endi PROCEED ni NOT_PROCEED ga almashtirib ishlatib ko'ramiz.

```
@UpdateController
public class SimpleController {

    @HandleAny(proceed = Proceed.NOT_PROCEED)
    void handleAnyMessage(Update update, AbsSender sender) {
        String chatId = update.getMessage().getChatId().toString();
        SendMessage message = new SendMessage(chatId, "Handler Any Message");
        sender.execute(message);
    }
```

```
    @HandleMessage(value = "abs", scope = MatchScope.START_WITH)
    void startWithAbsHandler(Update update, AbsSender sender) {
        String chatId = update.getMessage().getChatId().toString();
        String text = "Start With 'abs' : " + update.getMessage().getText();
        SendMessage message = new SendMessage(chatId, text);
        sender.execute(message);
    }

}
```

Figure 4. Handle Any Not Proceed Test

rasmdagi natijani ko'rgan bo'lsangiz faqat `@HandleAny` method ishladi va undan keyin handlar methodlar bajarilmadi.

`@HandleAny` annotatsiyasi qoyilgan method kirib keladigan parameterlarsiz ham yozishimiz mumkin. Lekin Update yoki AbsSender o'zini kiritishimiz hozircha mumkin emas!.

```
@HandleAny
void handleAnyMessage() {
    log.info("Handle Any Message!");
}
```