О библиотеке

Библиотека берёт на себя всю рутину по формированию сессий, созданию команд и отслеживанию и логированию данных при работе с телеграм бот API. Наследуется от wds-program-agent-lib

Стартер позволяет писать обработку команд в боте посредством создания компоненов-бинов, которым имеют динамические методы с не фиксированными параметрами. Ориентация была на модуль спринга - web. Контроллеры также строятся, помечая аннотацией сам класс и методы с аргументами методов

Зависимости

1. Spring boot 2.7.0
2. org.telegram.telegrambots
3. lombok
4. org.postgresql:postgresql
5. spring-boot-starter-web
6. spring-boot-starter-security
7. spring-boot-starter-data-jpa
8. spring-boot-starter-cache
9. retrofit
10. org.json.json
11. hibernate-types-55

Тестовый проект - https://github.com/Wladimir134679/test-telegram-bot-wds-lib

Конфигурация

В спринг конфигурации указывается:

program-agent:

```
  charts:
        bot-message: true
 telegram:
  bot:
    username: testBot
    token: "191526:AAF2pVOEU9w4PFZTg"
    session:
        time-life:
          chat: 60
          user: 10
```

1. program-agent.charts.bot-message - включает автоматическую генерацию истории обращений к боту в базу данных ()
2. username - имя бота
3. token - токен бота в системе телеграма, получаемы в BotFather
4. time-life - время жизни сессий
    a. chat - для чата
    b. user - для личного пользователя

О работе системе команд в боте

Система строится как WEB MVC в спринге. Аннотации вешаются на классы и на методы

1. CommandNames - вешается на класс, внутри себя несёт массив команд, на которые данные методы будут реагировать. Например */start.* По умолчанию работает с сообщениями, можно переключить на келлбеки(, type = TypeCommand.CALLBACK внутри), и тогда будут приходить данные келлбека. При этом келлбеки должны генерироваться специальным образом, об этом дальше.

2. CommandFirst - на метод, метод будет вызваться тогда, когда была вызвана сама команда, например */start,* отправится привествие
3. CommandOther - на метод, внутри сессии сгенерированной на пользователя внутри чата работает вызов на сообщение и на келлбэк. То есть после /start пользователь напишет любое сообщение, оно будет отправлено в этот методы в классе с /start.
4. ParamName - методы имеют динамические аргументы, перечисляются базовые типы, могут указываться как без так и с ParamName, обычно, ParamName используется для приметивов, например, чтобы получить chatId, userId, messageId и прочие данные в Long, например.
    a. На рисунке 1.1 показаны основные типы данных которые могут находиться в методе
    b. Для типа данных Long можно использовать ParamName со следующими именами:
        i. userId - для получения id пользователя
        ii. chatId - в каком чате происходит общение
        iii. messageId - id сообщение в чате

Также немного о сессиях - сессия генерируется на чат, внутри чата генерируется на пользователя. Поэтому если пользователь пишет в личных бота и в общем чате - это разные сессии для бота и команды.

```
Update update = commandContext.getUpdate();
Class<?> type = param.getType();
if (type == TelegramLongPollingEngine.class)
    return commandContext.getEngine();
if (type == CommandContext.class)
    return commandContext;
if (type == Update.class)
    return update;
if (type == Message.class)
    return update.getMessage();
if (type == CallbackQuery.class)
    return update.getCallbackQuery();
if (type == Chat.class)
    return update.getMessage().getChat();
if (type == Long.class)
    return getObjectLongParameter(param, commandContext);
if (type == ChatBotSession.class)
    return commandContext.getChatBotSession();
if (type == UserBotSession.class)
    return commandContext.getUserBotSession();
```

Рисунок 1.1

Келлбеки

Можно обрабатывать келлбеки от сообщений, но они должны быть сгенерированы определенным видом.

Для этого существуют вспомогательный класс *ru.wdeath.managerbot.lib.util.KeyboardUtil*

Он может сформировать клавиатуру по входному двухмерному массиву объектов, как просто текст, так и сложные сообщения для генерации данных. Для генерации такого объекта сущеуствует класс **CallbackDataSender,** который представляет из себя текст, видимы на клавиатуре и данные внутри, которые придут по нажатию, они будут распарсины и придут на обработку. В таких данных можно хранить номер следующией страницы в меню или же данные ID объекта обработки, например, показать имена товаров, а по нажатию в бота придёт ID товара, на который нажал пользователь.

Вот пример на рисунке 1.2, как используется данная система на практике

Рисунок 1.2

Формируется двухмерный массив кнопок, видимый текст, команда которая обрабатывается в @CommandNames и полезные данные, в данном случае преобразуется Long bot.getId() к строке, после чего будет обрабатываться в другом методе, показано на рисунке 1.3 - класс с командой и 1.4 метод по обработке меню настройки бота, если данные келлбека пустые - значит неверный запрос, пропустить.

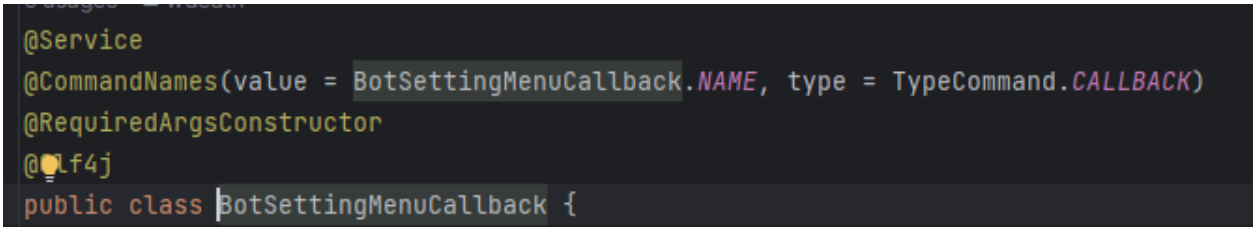

Рисунок 1.3

```
@CommandFirst
public void first(CommandContext context, @ParamName("chatId") Long chatId, @ParamName("messageId") Long messageId){
    String dataStringCallback = BotValidDataUtil.getDataStringCallback(context, chatId);
    if(dataStringCallback == null)
        return;
```

Рисунок 1.4

Немного о *CommandContext*

CommandContext несёт в себе все данные общения с пользователем. На рисунке 1.5 показаны все поля

```
@Data
public class CommandContext {

    private String name;
    private Object data;
    private TypeCommand typeCommand;
    private TelegramLongPollingEngine engine;
    private Update update;
    private UserBotSession userBotSession;
    private ChatBotSession chatBotSession;
}
```

Рисунок 1.5

1. name - имя команды
2. data - для сообщения это массив слов в сообщении, а для келлбека это строка внутри данных
3. typeCommand - келлбек или сообщение это
4. engine - сам движок бота, куда отправлять сообщения
5. update - данные который сейчас пришли в бота
6. userBotSession и chatBotSession - сессии соответственно пользователя и чата