Java Bot

Руководство по использованию

НИУ ИТМО Куприянов Артур, P3112 apploidyakutsk@gmail.com
https://github.com/AppLoidx/JavaBot

Оглавление

Цели проекта

Основной целью данного проекта является создание микросервиса[1], занимающейся рутинными задачами и предоставляющий легкодоступный интерфейс для пользователя.

Ядро

Java Bot Core

Центральную часть всей работы составляет модуль, работающий через взаимодействие с помощью команд.

Это позволяет создавать различные прикладные программы для работы с этим модулем, дальше именуемым как **Java Bot Core**.

Различные решения для различных задач с общими данными.

Преимущества такой архитектуры в том, что мы можем разместить Java Bot Core в удаленное место (например, в облаке) и создавать программы, отправляющие запрос.

Как итог, мы получаем слабосвязанные между собой модули. Иными словами, Java Bot Core не знает с кем он работает, ему это и не нужно. Все потому, что это ядро должно предоставлять общий функционал для решения конкретных задач.

Конкретный пример

Чтобы лучше понять принцип, рассмотрим пример Бота, созданного на основе такой архитуры.

Опустим реализацию бота и представим, что он уже работает и обрабатывает сообщения поступающие из социальной сети в Вконтакте. То есть, каждый раз, когда кто-то пишет боту в социальной сети, мы сможем его обработать в программе.

Теперь, необходимо обработать ввод пользователя.

На этом этапе все зависит от приложения. Например, один из них может иметь NAL[2], обрабатывающий запрос и интерпретирующий его в команду, понятную для Java Bot Core.

Далее, мы отправляем запрос на Java Bot Core, предоставляющий API для таких запросов и получаем ответ, который мы также можем обработать в нашем приложении, замет отобразив его пользователю.

---

[1] Микросервисы - это путь разбиения большого приложения на слабо связанные модули, которые коммуницируют друг с другом посредством просто API.
[2] NAL - Natural Language Processing

API-документация

Синтаксис команд

Общий синтаксис выглядит таким образом:

<ИМЯ_КОМАНДЫ> -[КЛЮЧИ] <ЗНАЧЕНИЕ> --[КЛЮЧ] <ЗНАЧЕНИЕ>

Каждая команда может отличаться друг от друга и реализовывать другой синтаксис и обработку опций, что будет показано при более детальном разборе Java Bot Core.

Пример команд:

weather --today – обращение к команде weather с ключом today

schedule -p 1 -a – обращение к команде schedule с ключом p со значением 1 и ключом a

note --teacher 2 – обращение к команде note с ключом teacher со значением 2

Базовый список команд

- **Queue** – создание, присоединение к очереди.
- **Note** – Записки в базу данных
- **Schedule** – расписание пар по группам и аудиториям
- **Reg** – регистрация в базу данных

Schedule v1.4

Команда schedule работает в двух режимах: по расписанию занятий группы и по аудиториям.

Чтобы команда работала в режиме аудитории, необходимо использовать ключ "-r <auditory_number>", указав номер аудитории. Режим по расписанию группы включен по умолчанию.

Работа schedule по расписанию

- -a -- вывести полное расписание (подавляет ключ -d)
- -g -- номер группы
- -d {int} - кол-во дней вперед

- -p {1/0} - четность недели (1 - четная, 0 -нечетная), если не указан, то выбирает текущую четность

(Если вызвать schedule без ключей, то по умолчанию стоит группа P3112, покажет сегодняшнее расписание)

Примеры:

schedule -g P3111 -d 2 -p 1 - выводит расписание группы P3111 через 2 дня по четной неделе.
schedule -g P3112 -a - выводит полное расписание группы P3112
schedule -g P3100 -a -p 0 - выводит полное расписание группы P3100 по нечетной неделе

Работа schedule по аудиториям

Чтобы команда работала в режиме поиска по аудиториям, необходимо указать ключ -r и указать номер аудитории.

По умолчанию она покажем промежутки свободного времени в данной аудитории.

Указав ключ -l (lessons), можно вывести занятия проходящие в этой аудитории.

Ключи:

- -r {номер_аудитории\номера_аудиторий_разделенные_знаком_"-"} -- ключ для указания номера аудитории
- -f {int} -- ключ для указания необходимого свободного промежутка времени (меньшие не будут выводиться)(в минутах)
- -d {int} -- кол-во дней вперед
- -p {1/0} -- четность недели (1 - четная, 0 -нечетная), если не указан, то выбирает текущую четность
- -l -- ключ для вывода занятий в этой аудитории

Примеры:

schedule -r 304 -l -p -1 - вывести занятия в аудитории 304 по четной неделе
schedule -r 304 -- вывести все свободные промежутки времени в аудитории до с 08:20 - 23:00
schedule -r 304 -f 20 -- вывести все свободные промежутки времени в аудитории большие или равные 20 минутам
schedule -r 304-305 -l -p -1 - вывести занятия в аудитории 304 и 305 по четной неделе

Note

Команда для просмотра и добавления записей объявлений. Есть реализация для программ.

Для пользователей:

- -s – показать все объявления
- -a – добавить объявление
- -d – удалить объявление

Для программ

- -s – получить объявления в формате JSON
- остальные ключи функционируют также

Reg

Команда регистрации пользователя. Автоматически регистрирует имя, фамилия и vk id пользователя. При необходимости можно задать другие параметры через ключи:

- -g [номер_группы]
- -l [логин] -p [пароль]

Пароль и логин необходимы при регистрации в прикладных программах.

Для разработчиков

Подробная архитектура Java Bot Core

Рекомендуется просмотреть статью **Создание простой архитектуры бота для внедрения в систему на Java**[3], где подробно расписан пример создания подобной архитектуры.

Далее, будет рассмотрена архитектура исходной программы Java Bot Core. Все его элементы можно разделить на несколько частей:

- Модули
- Команды
- Вспомогательные классы
- Обработчик ввода

Разберем каждый из них.

Модули

Модули – это своего рода библиотеки, к которым обращаются команды. Иными словами, они содержат всю бизнес-логику самой реализации команды.

Команды

Команды – это обрабатывающие ввод пользователя объекты. Они в свою очередь, получив ввод, должны вернуть какое-либо значение или выполнить некую операцию.

Вспомогательные классы

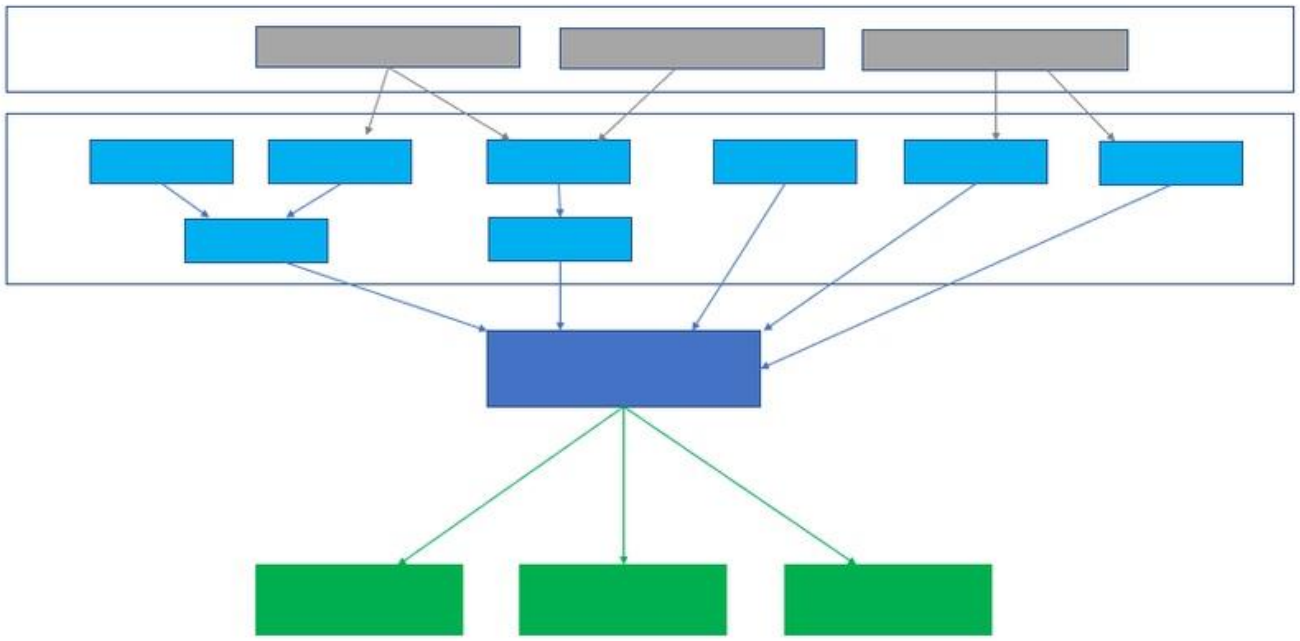
К вспомогательным классам относятся все классы, не входящие в другие, но не менее важные. Базовые классы, относящиеся к этой области – это определитель команд, совершающий выборку команды, и менеджер команд, содержащий все объекты команд. В общем случае эта область исполняет связующую роль между Обработчиком ввода и Командами.

[3] Ссылка - https://vk.com/@apploidxxx-sozdanie-prostoi-arhitektury-bota-dlya-vnedreniya-v-sistemu

Обработчик ввода

Это первое место, куда попадает запрос к Java Bot Core. Здесь же находятся Сервер приёма данных и Интерфейс ядра, также именуемый как Commander. Основная его задача – это прием и отправка данных, а также их обработка с помощью вспомогательных классов.

Архитектура



Карта цветов:

1. **Серый**. **Модули**. Уровень сторонних и пользовательских библиотек и модулей, предназначенных для решения различных задач. К ним будут ссылаться команды.
2. **Голубой. Команды**. Они идут как отдельные объекты, которые обращаются при необходимости к модулям, тем не менее, они должны иметь как можно меньший объем памяти, так как команд может быть очень много.
3. **Синий. Обработчик ввода и вспомогательные классы**. Общий интерфейс, позволяющий упростить обращение ко множеству команд. Паттерн Facade. Цель внедрения такого интерфейса — это сведение к минимуму зависимости подсистем друг от друга и обмена информацией между ними.
4. **Зеленый. Различные реализации**. Это уже различные системы, обращающиеся к боту, а именно к синему интерфейсу.

Реализация в Java Bot Core

Рассмотрим реализацию архитектуры в Java Bot Core

Обработчик ввода и вспомогательные классы

Здесь находятся несколько классов:

- **Commander**
- **CommandDeterminant**
- **CommandManager**

CommandManager (Менеджер команд) – отвечает за список доступных команд и содержит их объекты.
CommandDeterminant (Определитель команд) – отвечает за выборку команды
Commander – выполняет первичную обработку ввода и возвращает ответ

Команды

Команды хранятся как отдельные классы и при запуске их объекты добавляются в CommandManager. Все виды команд наследуются от суперкласса Command и реализуют его метод init().

Модули

Модули хранятся в отдельных пакетах. Особых требований по ним – нет.

Путь запроса

1. Сначала запрос поступает в Commander.
2. Вызывается метод определителя команд
3. Производиться выборка команды с помощью CommandDeterminant
4. Исполнение команды через CommandManager
5. Возвращение результата выполнения команды

Суперкласс Command

Актуальный исходный код суперкласса доступен на GitHub[4]

Это абстрактный класс-родитель для всех исполняемых в Commander команд.

Основные элементы - это поле name и метод init()

[4] https://github.com/AppLoidx/JavaBot/blob/master/src/main/java/core/commands/Command.java

Поле name

Поле commandName – это идентификатор и имя команды. При выполнении команды, определяется первое слово и сравнивается с этим полем каждой команды. В случае совпадения – исполняется эта команда. Следует определить его в методе setName, напрямую обращаясь к этому полю.

Метод init()

Метод, выполняющий инициализацию команды. После выборки команды вызывается этот метод и в аргументах передается запрос в виде массива, разделенный пробелом.

Минимальная реализация пользовательской команды

```
public class Unknown extends Command {

    @Override
    protected void setName() {
        commandName = "unknown";
    }
    @Override
    public String init(String... args) {
        return "Не распознанная команда";
    }
}
```

Использование модуля KeysReader

Класс KeysReader предназначен для обработки ключей команды, вводимых пользователем. На момент версии 1.0.1 содержит два статических метода:

- readKeys(String[] words) - возвращает Map с паттерном: <String ключ, String значение>, где значение может быть пустым. Присутствует JavaDoc. В большинстве случаев достаточно передать ему входной аргумент метода init() и получить карту ключей.
- readOrderedKeys(String[] words) - действует также, как и readKeys, но поддерживает сортировку ключей. Иными словами, он нужен, если важна последовательность введённых ключей. Возвращает TreeMap с паттерном: <Integer index, <String ключ, String значение>>.

```
public String init(String... args) {
    Map<String, String> keysMap = KeysReader.readKeys(args);
    String name;


    if(keysMap.containsKey("-n") || keysMap.containsKey("--name")){
        name = keysMap.get("-n");
    } else {
        return "Не указан обязательный ключ -n [имя_очереди]";
    }
```

Метаданные и UserInfoReader в VK

Метаданные носят в себе информацию о пользователе, например, имя. У них есть специальный синтаксис, и они добавляются в ввод пользователя самим обработчиком команд сервисов VK. Поэтому команды, которым нужен номер или имя пользователя, могут получить эти данные.

Для этих целей есть класс UserInfoReader. Он предоставляет несколько статических методов для считывания метаданных. Исходный код представлен на GitHub[5]

Список метаданных из VK:

- User ID: id в vk пользователя
- First Name
- Last Name
- Program – для проверки типа запроса (программный\пользовательский)

Создание прикладных программ

Реализация в прикладных программах представлена в отдельной документации.

[5] https://github.com/AppLoidx/JavaBot/blob/master/src/main/java/core/common/UserInfoReader.java