УТВЕРЖДЕН
41327606.425000.462-01 32 01-ЛУ

ДИАЛОГОВЫЙ ПРОЦЕССОР

(подсистема проекта SOVA)

Руководство программиста

41327606.425000.462-01 32 01

Листов 20

Москва
2020
www.sova.ai

Подпись и дата	
Инв. №дубл.	
Взам. инв. №	
Подпись и дата	
Инв. №подл.	

АННОТАЦИЯ

Данный документ является руководством программиста (системного администратора) для информационной системы Диалоговый Процессор (подсистема проекта SOVA).

В документе приводится необходимая информация по установке и настройке программного обеспечения, а также сведения по администрированию системы.

Оформление настоящего документа произведено в соответствии с ГОСТ 19.503-79.

СОДЕРЖАНИЕ

1 ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

1.1 Назначение программы

Диалоговый Процессор SOVA предназначен для использования в системах автоматизации диалогов для предоставления необходимой информации в автоматизированном режиме и может быть интегрирован в сервисы сайтов (например в виджет чата), а так же в приложения при помощи встроенного диалогового интерфейса Web API.

1.2 Функции программы

Основной функцией программы Диалоговый Процессор SOVA является предоставление релевантных ответов на запросы пользователей, получаемых на естественном языке по каналам связи.

1.3 Минимальный состав технических средств

Минимальный состав используемых технических средств:

1) IBM PC совместимый с процессором x86 и выше
2) ОЗУ более 16 Мбайт
3) 128 МБ видеопамяти и выше
4) наличие свободного места на жестком диске 20 Мбайт.

1.4 Минимальный состав программных средств

Системные программные средства, используемые программой, должны быть представлены локализованной версией операционной системы Windows 10/XP/Vista/7.

Также для функционирования программы на ПК необходимо предустановленное программное обеспечение PostgreSQL Server.

1.5 Требования к персоналу (системному программисту)

Требования к персоналу (системному программисту)

Системный программист должен иметь минимум среднее техническое образование. В перечень задач, выполняемых системным программистом, должны входить:

1) задача поддержания работоспособности технических средств;

2) задача установки (инсталляции) и поддержания работоспособности системных программных средств - операционной системы и SQL Server;

3) задача установки (инсталляции) и поддержания работоспособности программы

2 СТРУКТУРА ПРОГРАММЫ

2.1 Сведения о структуре программы

Программа Диалоговый Процессор состоит из трех независимых частей - из одного запускаемого компонента и 2-х вызываемых.

2.2 Сведения о составных частях программы

В состав программы Диалоговый Процессор входят три компонента

1) Диалоговое ядро (engine)
2) Компилятор языка DL,
3) Система Бизнес Логики (BLS)

2.3 Сведения о связях между составными частями программы

2.4 Сведения о связях с другими программами

Программа Диалоговый Процессор в ходе своей работы подключается и использует базу данных PostgreSQL, поэтому для ее функционирования требуется предустановленная версия PostgreSQL.

3 НАСТРОЙКА ПРОГРАММЫ

3.1 Описание действий по настройке программы на условия конкретного применения

3.1.1 Диалоговое Ядро

Описание

Диалоговое Ядро - компонент Диалогового Процессора SOVA, предназначеный для использования в системах автоматизации диалогов (например в ботах).

Диалоговое ядро (engine), реализует функционал генерации текстовых ответов в реальном времени, используя для этого бинарную поисковую базу, подготовленную компилятором языка DL.

3.1.2 Быстрый старт

Для запуска Диалогового Ядра в стандартном режиме необходимо выполнить следующую команду:

```
docker-compose up -d
```

Диалоговое ядро запустится с Базой Знаний по умолчанию в режиме docker-compose и будет принимать запросы на localhost:2255.

Для обновления Базы Знаний, необходимо положить исходные данные в директорию `/dldata` (если ее нет, то она будет автоматичеки создана при запуске Диалогового Ядра) и выполнить команду:

```
./docker/update.sh
```

Запустится процесс компиляции Базы Знаний, причем бот Диалоговое Ядро будет работать и отвечать на запросы до самого момента замены скомпилированной Базы Знаний. В случае, если в данных Базы Знаний были допущены ошибки, компиляции выдаст полное их описание и прервет обновление.

В случае, если компиляция прошла успешно, скомпилированная База Значний автоматически устанавливается в Диалоговое Ядро, а ее копия копируется в директорию `/dldata` под именем `dldata.ie2` и может быть использована Диалоговым Ядром для быстрого старта с этой Базой Знаний.

3.1.3 Сборка Диалогового Ядра на Debian 10

Описан процесс сборки диалогового ядра в системе команд OS Linux Debian 10. В случае использования другой сборки Linux или в случае запуска программы на платформе Windows или MacOS, используйте соответствующий дистрибуттив и инструкцию.

Для сборки диалогового ядра выполните:

```
//|Установка зависимостей:
    apt-get update && apt-get upgrade -y cmake \
    build-essential flex libgmp-dev \
    libmemcached-dev libreadline-dev libicu-dev
// Сборка Диалогового Ядра:
    mkdir cmake_build && cd cmake_build && cmake ../build && make -j 5
```

3.1.3.1 Компоненты Диалогового Ядра:

a) Компилятор языка DL

Параметры командной строки

```
// Вызов Компилятора Языка DL:
    InfCompiler
// Получение подсказки по использованию:
    InfCompiler --help
// Вывод текущей версии компилятора и информации по сборке:
    InfCompiler --version
// Запуск компиляции:
    InfCompiler [ConfigFile] [OPTIONS]
```

Конфигурационный файл

```
 // Конфигурация компилятора языка DL.

 [Main]
 // Корневая директория для Диалогового Ядра. Значение по умолчанию - родитель
 ская диретория конфигурационного файла.
 // RootDir = /usr/local/InfEngine
```

```
[Functions]
// Директория с библиотеками внешних функций.
RootDir = release/lib/functions

// Список внешних функций.
ConfigFile = conf/functions.lst


[Log]
// Уровень логирования. Возможные значения: [ "NONE", "ERROR", "WARN", "INFO"
, "DEBUG" ]. Значение по умолчанию: "NONE".
Level = INFO

// Идентификатор логирования. По умолчанию: "InfEngine Compiler".
Identificator = InfEngine Compiler


[Swap]
// Swap файл для компиляции.
FilePath = tmp/swap

// Ограничение памяти. Значение по умолчанию: "0" ( Swap отключен ).
MemoryLimit = 10000000


[InfCompiler]
// Корневой каталог для DL данных.
DLDataDir = data/dlstable

// Путь к списку файлов с шаблонами на языке DL.
DLPatternsList = dl.lst

// Путь к списку файлов со словарями на языке DL.
DLDictsList = dict.lst

// Путь к списку переменных, используемых в Диалоговом Ядре.
DLVarsList = defvars.lst

// Путь к целевой базе.
TargetPath = dldata.ie2

// Флаг жесткости компиляции. Возможные значения: [ "TRUE", "FALSE" ]. Значен
ия по умолчанию: "TRUE"
StrictMode = true

// Путь к файлу с алиасами на языке DL.
DLAliases = aliases.dl

// Путь к списку файлов с синонимами.
SynonymsConfFile = syn.lst

// Корневой путь для файлов с синонимами.
SynonymsRootDir = synonyms
```

б) Сервис Диалогового Ядра

Параметры командной строки

```
// Вызов Сервиса Диалогового Ядра

    InfServer

// Получение подсказки по использованию:

    InfServer –help

// Вывод текущей версии компилятора и информации по сборке:

    InfServer --version
```

Конфигурационный файл

```
// Конфигурация Диалогового Ядра.

[Functions]
// Директория с библиотеками внешних функций.
RootDir = release/lib/functions

// Список внешних функций.
ConfigFile = conf/functions.lst


[Log]
// Уровень логирования. Возможные значения: [ "NONE", "ERROR", "WARN", "INFO",
"DEBUG" ]. Значение по умолчанию: "NONE".
Level = WARN

// Идентификатор логирования. По умолчанию: "InfEngine Server".
Identificator = InfEngine Server

// Flag for logging all data flow between server and clients. Default value: "
false".
// Флаг для логгирования всех данных проходящих через Диалоговое Ядро.
DataFlow = false


[Cache]
// Сервер кэширования.
Servers = localhost:11211

// TTL для кэш записей. Значение по умолчанию: "604800" ( одна неделя ).
TTL = 604800


[InfServer]
// Путь к бинарной Базе Знаний.
BasePath = release/dldata/dldata.ie2
```

```
// Максимальное число запросов, обрабатываемых одним процессом. Значение по ум
олчанию: "10000".
MaxRequestsNumber = 1000

// Таймаут для запроса. Значение по умолчанию: "30".
TimeOut = 30

// Путь к файлу с алиасами на языке DL.
DLAliases = release/dldata/aliases.dl

// Флаг отладочного режима.
DebugMode = YES
```

в) Процесс Сервер

Процесс Сервер – это сервис, реализующий запуск Сервисов Диалогового Ядра в мультипроцессном режиме

Параметры командной строки

```
ap-process-server path-to-inf-server
```

Кофигурационный файл

```
# Путь к InfServer.
FilterPath ./bin/InfServer

# Параметры запуска InfServer.
FilterParam ./conf/InfServer.conf

# Сокет, который процесс-сервер будет слушать.
Listen tcp::2255

# Логгирование.
LogLevel 0
SyslogFacility local3

# Путь к файлу с pid'ом процесс сервера.
PidFile ./tmp/process-server.pid

# Максимальное количество одновременных процессов InfServer.
MaxFilters 1
# Количество процессов InfServer, которые будут запущены при старте процесс-с
ервера.
StartFilters 0
# Минимальное разрешенное количество запущенных процессов InfServer.
MinSpareFilters 0
```

База Знаний

Пример Базы Знаний и формата всех необходимых для компиляции файлов представлен в директории /docker/dldata.

3.1.4 Диалоговый Процессор

3.1.4.1 Описание

Диалоговый Процессор является компонентом проекта SOVA и предназначен для формирования ответов на текстовые запросы пользователей, поступающие в режиме связного диалога на естественном языке. Диалоговый Процессор формирует ответ на запрос на основании внутренних весовых алгоритмов в контексте текущего диалога в соответствии с предварительно подготовленным набором сценариев и правил формирования ответов, записанных на специализированном языке Dialog Language (DL).

3.1.4.2 Быстрый старт

Для запуска Диалогового Процессора в стандартном режиме выполнитt команды:

```
    git clone git@github.com:sovaai/dp.git sova cd sova

    ./install.sh

    docker-compose up -d

    curl -XPOST localhost/api/create_inf -d '{"buid":"6944d0b0-ca59-4007-97bf-867d
6c4385a9", "profile":"common"}'
```

Диалоговый Процессор запустится с Базой Знаний по умолчанию в режиме docker-compose и будет принимать запросы на localhost. Далее можно открыть браузер по адресу http://localhost. На этой странице будет загружен виджет, с помощью которого можно вести диалог с ботом.

3.1.4.3 Конфигурационный файл

Параметры конфигурации указываются в файле переменных окружения .env . Он копируется из файла-примера .env.example скриптом установки install.sh:

```
POSTGRES_USER=postgres POSTGRES_PASSWORD=pgpassword POSTGRES_DB=sova NLAB_SOVA_SEN
    TRY_DSN=https://sentry.com/ NLAB_SOVA_VERSION=0.0.2 NLAB_SOVA_DEBUG=True

NLAB_SOVA_DB_DSN=postgres://postgres:pgpassword@db:5432/sova NLAB_DLPATH=./var/dld
    ata/sample
```

3.2 Настройка программы на состав технических средств

Программа Диалоговый Процессор не требует каких-либо настроек на состав технических средств.

В случае использования другой программной платформы, например, при использовании другой сборки Linux или в случае запуска программы на платформе Windows или MacOS - используйте соответствующий дистрибутив и инструкцию.

3.3 Настройка программы на состав программных средств

Программа Диалоговый Процессор не требует каких-либо настроек на состав программных средств.

Программа может быть установлена в любой каталог. Для установки данной программы достаточно запустить инсталлятор и следовать шагам установки. Каких-либо настроек после установки программы не требуется.

3.4 Выбор функций программы

Для настройки диалогов программы под ваши портебности - пополните базу знаний системы необходимыми сценариями диалогов, записанными на языке DL.

4 ПРОВЕРКА ПРОГРАММЫ

Описание способов проверки, позволяющих дать общее заключение о работоспособности программы

Работоспособность программы Диалоговый Процессор проверяется описанными ниже способом:

1) Загрузите сценарии диалогов, записанных на языке DL в директорию базу знаний Системы
2) Запустите все компоненты системы
3) Запустите чат бот, в состав которого интегрирован компонент диалоговый процессор.
4) Проверьте соответствие ответов чат бота на запросы пользователя сценариям базы данных.

- В случае успешной установки чат бот будет формировать релевантные ответы на запросы
- В случае неудачи в ответ на запросы чат бот будет выдавать пустую строку или бессмысленные сообщения.

4.1 Методы прогона программы

Проверка работоспособности программы

1) Проверьте наличие правильно установленного SQL Server на локальной машине.

2) При запущенной программе попробуйте соединиться с БД, методом, описанным в п.4.4.1 данного руководства.

3) Далее попробуйте отобразить таблицу с помощью блока "Выберите таблицу"

5 ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ

5.1 Описание дополнительных функциональных возможностей программы

5.1.1 InfoServer Chat API

Общие сведения

В данном разделе содержится описание программного интерфейса InfoServer Chat API (версиия 2.2.1), предназначенного для подключения клиентского приложения к сервису (далее — API).

API Диалоговый Процессор обеспечивает автоматическое ведение диалога с пользователем в режиме текстового чата.

5.1.1.1 Уникальный идентификатор бота (UUID)

API поддерживает работу с множеством виртуальных консультантов — ботов.

Для идентификации конкретного бота, с которым работает клиентское приложение, используется уникальный идентификатор бота (UUID), который присваивается при заведении бота в API.

5.1.1.2 Уникальный идентификатор чата (CUID)

API поддерживает одновременное ведение множества параллельных диалогов (чатов) с различными пользователями.

Для идентификации чата, к которому относится очередной запрос, используется уникальный идентификатор чата (CUID).

Изначально CUID присваивается API при инициализации первого чата с данным пользователем.

Клиентское приложение должно использовать полученный CUID при каждом обращении к API и для этого хранить CUID локально для каждого пользователя.

5.1.1.3 Описание протокола

Взаимодействие с API происходит по протоколу, работающему поверх HTTP/HTTPS посредством запросов методом POST. Передача данных осуществляется в формате JSON в кодировке UTF-8.

Адрес обращения:

```
http://<domain>/api/Chat.<метод>
```

Ответ API

При успешном завершении всех обращений ответ API возвращается в контейнере:

```
{
"result": {/* ответ API */},
"id": "0"
}
В случае ошибки API возвращает следующую структуру:
{
"error": {
"code": <error code>,
"message": "<error message>"
},
"id":"0"
}
```

5.1.1.4 Структура "context"

Структура "context" используется для передачи и получения контекста — набора переменных, связанных с запросом или ответом — и имеет формат:

```
"context": {
<имя переменной 1>: <значение>,
<имя переменной 2>: <значение>, ...
}
Например,
"context": {
```

```
"status": "ok",
"count": 5
}
```

5.1.2 Описание методов CHAT.*

5.1.2.1 CHAT.INIT

Инициализация чата.

```
Chat.init
```

URL

```
POST http://<domain>/api/Chat.init
```

Запрос:

```
{
"uuid": <string>,   // Идентификатор бота.
"cuid": <string>,   // Идентификатор чата. Необязательный параметр.
"context": <object>  // Контекст чата. Необязательный параметр.
}
```

Идентификатор чата (CUID):

Если текущий пользователь уже осуществлял чат с ботом - передайте CUID, который был ранее возвращен API.

Если время жизни чата, идентифицированного CUID, еще не истекло, API восстановит его, а противном случае - откроет новый чат и вернет новый CUID.

Если текущий пользователь ранее не взаимодействовал с API, то передача CUID не требуется. В этом случае API открывает новый чат и возвращает новый CUID.

а) Контекст чата ("context"):

Необязательный параметр.

В контексте метода Chat.init передаются переменные, описывающие окружение, в котором ведется данный чат. Например, имя пользователя или язык интерфейса.

Требования к контексту, который должен передаваться при инициализации чата, зависят от используемого бота.

Контекст чата может быть изменен при помощи события CONTEXT (см. Chat.event).

Ответ API (структура "result"):

```
{
      "cuid": <string>,       // Идентификатор чата.
"inf": {
"name": <string>  // Имя бота.
},
"events": {
// Настройки и состояние событий.
// Используются в специальных проектах.
},
}
```

Идентификатор чата (CUID), полученный в ответе API, может отличаться от ранее сохраненного. В этом случае старый CUID становится недействительным, и во всех дальнейших запросах к API необходимо использовать новый идентификатор.

5.1.2.2 CHAT.REQUEST

Получение ответа и реакции бота на запрос пользователя.

```
Chat.request
```

URL

```
POST http://<domain>/api/Chat.request
```

Запрос:

```
{
"cuid": <string>,   // Идентификатор чата.
"text": <string>,   // Текст запроса, UTF-8.
"context": <object>  // Контекст запроса. Необязательный параметр.
}
```

Ответ API (структура "result"):

```
{
"text": {
"value": <string> // Ответ бота.
```

```
},
"cuid": <string>,   // Идентификатор чата.
"context": <object> // Контекст ответа. Необязательный.
"id": <string>          // Идентификатор ответа.
}
```

Идентификатор чата (CUID), полученный в ответе API, может отличаться от ранее сохраненного. В этом случае старый CUID становится недействительным, и во всех дальнейших запросах к API необходимо использовать новый идентификатор.

5.1.2.3 CHAT.EVENT

Получение ответа и реакции бота на событие.

```
Chat.event
```

URL

```
POST http://<domain>/api/Chat.event
```

Запрос:

```
{
"cuid": <string>,       // Идентификатор чата.
"euid": <string>,       // Идентификатор события.
"context": <object>     // Контекст события. Необязательный параметр.
}
```

Идентификатор события (EUID):
Идентифицирует тип события (описание типов событий см. ниже).

Контекст события:
Необязательный параметр.

Ответ API:
Аналогичен ответу при вызове Chat.request.

Типы событий:

Событие READY

```
EUID: 00b2fcbe-f27f-437b-a0d5-91072d840ed3
```

Используется в начале нового чата для получения приветственного сообщения от бота.

Событие INACTIVE

```
EUID: 29e75851-6cae-44f4-8a9c-f6489c4dca88
```

Используется для получения реакции на бездействие пользователя.

Например, может отправляться для получения анонсов, которые должны показываться пользователю в режиме ожидания.

Для использования в специальных проектах:

В контексте события INACTIVE может отправляться переменная "count", значением которой должно быть число, показывающее, какой раз подряд возникает событие.

```
"context": {
"count": 1
},
```

Событие CONTEXT

```
EUID: 7330d8fc-4c64-11e3-af49-080027ab4d7b
```

Используется для изменения контекста чата (описания окружения, в котором ведется данный чат).

Изначально контекст чата передается API при вызове Chat.init и может быть изменен с помощью события CONTEXT.

5.2 Описание действий, которые необходимо предпринять по этим сообщениям

ПЕРЕЧЕНЬ ТЕРМИНОВ И СОКРАЩЕНИЙ

- API -(программный интерфейс приложения, интерфейс прикладного программирования) (англ. application programming interface, API [эй-пи-ай][1]) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой
- Диалоговый Процессор - система, работающая в режиме диалога, при котором она отвечает на каждую команду пользователя и по мере надобности обращается к нему за информацией
- ОЗУ - операти́вная па́мять или операти́вное запомина́ющее устро́йство
- Диалоговое ядро (engine) - цетральная часть программы, выполняющая основные функции этой программы.

- Компилятор языка DL - специальная программа, которая переводит текст программы, написанный на языке программирования, в набор машинных кодов.
- База знаний - база данных, о знаниях в предметной области, разработанная для оперирования знаниями (метаданными), содержит структурированную информацию для использования устройством (или человеком) с конкретной целью. Базы знаний работают совместно с системами поиска информации, имеют классификационную структуру и формат представления знаний.
- Бот - виртуальный собеседник, искусственный интеллект.
- CUID - уникальный идентификатор чата, используется для идентификации чата, к которому относится очередной запрос.
- POST - вид запроса
- JSON - формат передачи данных

Лист регистрации изменений

Номера листов (страниц)					Всего листов (страниц) в докум.	№ документа	Входящий № сопроводит. докум. и дата	Подп.	Дата
Изм.	измененных	замененных	новых	аннулированных					