Token

Токен - это контейнер данных, полученных в ходе разбора фразы пользователя по правилам, описанным в паттерне.

Другими словами, токен можно воспринимать как семантическое дерево разбора. Это представление фразы пользователя в виде абстракции, не зависящей на конкретного языка.

Токены вложены друг в друга, поэтому мы говорим именно о дереве. При срабатывании паттерна в агента передается один токен - это корневой элемент, содержащий всю фразу и, как правило, вложенные токены.

Атрибуты токена

Каждый токен обязательно содержит подстроку текста из фразы - поле `source`.

Также имеются необязательные атрибуты:

`name` - имя токена, если оно задано в атрибуте `name` паттерна или в виде синонима

`value` - отображаемое значение, если оно определено в паттерне с помощью `:`

`tokens` - дочерние токены

Как работать с токенами

Класс `Token` предоставляет вспомогательные методы для поиска дочерних токенов по имени или по индексу.

`findTokenByName`

`findTokensByName`

`getTokensCount`

`getTokenByIndex`

`isEmpty`

`hasToken`

В токене может содержаться более одного токена с одинаковым именем (например, при использовании `repeat`). В таком случае `findTokenByName` вернёт только первый из них, а `findTokensByName` вернёт все.

Все дочерние токены нумеруются слева направо, начиная с нуля. Метод `getTokenByIndex` возвращает токен по индексу.

Конвертеры

По сути, токен - это "сырое" представление данных, полученных в результате разбора текста по паттерну. Иногда для работы с токеном достаточно этих данных (поля `name`, `source` и `value`). Но иногда требуется преобразовать токен в экземпляр более подходящего типа данных для реализации логики приложения.

Например, фраза *Двадцать пять* на русском языке неудобна для выполнения каких-либо вычислений. Её хорошо бы сперва преобразовать в целочисленное *25*.

Для этого используются конвертеры токенов. На вход они принимают токен, а на выходе возвращают экземпляр требуемого класса.

API содержит необходимые конвертеры для токенов всех глобальных паттернов. Для собственных токенов может понадобиться реализовать свои конвертеры.

Конвертеры глобальных паттернов

Все конвертеры реализованы в виде синглтонов, т.к. конвертер не должен хранить состояние. У каждого ковертера есть метод класса `getInstance`, который возвращает экземпляр конвертера. Это рекомендуемый способ реализации ваших собственных конвертеров.

NumberConverter - получает на вход токен глобального паттерна *Number*. Возвращает *Integer*.

DateConverter - получает на вход токен глобального паттерна *Date*. Возвращает обёртку *Date*, представляющую дату.

TimeConverter - получает на вход токен глобального паттерна *Time*. Возвращает обёртку *Time*, представляющую время.

DateTimeConverter - получает на вход токен глобального паттерна *DateTime*. Возвращает обёртку *DateTime*, представляющую дату-время.

LocationConverter - получает на вход токен глобального паттерна *Location*. Возвращает обёртку *Location*, представляющую географическое месторасположение (либо только координаты, либо конкретный город).

ContactConverter - получает на вход токен глобального паттерна *Contact*. Возвращает обёртку *Contact*, представляющую контакт из контактной книги пользователя.

Как реализовать свой конвертер

Ковертер - это просто реализация параметризованного интерфейса `TokenConverter`. Метод `convert` этого интерфейса принимает в качестве параметра токен и возвращает объект соответствующего класса.

Использование конвертеров

В общем случае для использования конвертера нужно сначала найти необходимый токен в родительском токене или в кукисах, а затем передать его на вход методу `convert` конвертера. Вот пример того, как это может быть сделано:

```
Token timeToken = request.getToken().findTokenByName(AssistantAgentContract.Tokens.TOKEN_TIME);
Time time = TimeConverter.getInstance().convert(timeToken);
```

В этом примере мы сперва ищем токен с именем *Time* (константа `AssistantAgentContract.Tokens.TOKEN_TIME`), а затем передаём его конвертеру `TimeConverter` в метод `convert`. Результатом является объект класса `Time`, представляющий информацию о времени. Далее вы можете использовать необходимые методы этого класса для использования данных о времени в логике вашего приложения (например, метод `getCalendar`).