



Документация библиотеки JMantic

Версия 0.6.0

23 января 2022 г.

Содержание

1	Введение	2
1.1	Описание библиотеки и её предназначения	2
1.2	Использованные технологии	2
1.3	Технические требования	2
1.4	Ссылка на репозиторий	2
2	Sc	3
2.1	Введение	3
2.2	Описание структуры языка SC	3
2.3	Язык SCg (Semantic Code Graphical) - графическая модификация языка SC . .	4
2.4	Основные графические примитивы языка SC	5
2.5	Язык SCs (Semantic Code string) – линейная модификация языка SC	6
2.6	Простые примеры для понимания сущности SC языка	9
3	Ostis	12
3.1	Обзор	12
3.2	Структура запрос	12
3.3	Структура ответ	12
3.4	Поле id	12
3.5	Поле type	13
3.6	Таблица базовых типов sc-элементов	21
3.7	Полная таблица поддерживаемых типов	21
4	Подробности реализации JMantic	23
4.1	Иерархия sc-типов	23
4.2	Интерфейс ScMemory	23
4.3	Реализация запросов и ответов Ostis	25
4.4	Создание поисковых шаблонов	25
5	Публичный api	27
5.1	DefaultScContext	27
5.2	UncheckedScContext	30
5.3	AsyncUncheckedScContext	33

1 Введение

1.1 Описание библиотеки и её предназначения

Библиотека JMantic создана для использования sc-памяти в java-программах. Глобально JMantic предоставляет возможность использовать любую sc-память, но на данный момент реализована только взаимодействие с sc-памятью в Ostis.

1.2 Используемые технологии

Для общения с Ostis применяется протокол WebSocket. Запросы и ответы представлены в виде сообщений в формате json с определённой структурой. Подробнее описано в разделе Ostis.

1.3 Технические требования

Исходный код библиотеки написан с использованием java версии 17. При тестировании использовался Ostis версии 0.6.0.

1.4 Ссылка на репозиторий

<https://github.com/artrayme/JMantic>

2 Sc

В данном разделе представлены основные понятия из Sc, которые требуются для понимания предметной области с которой работает JMantic.

2.1 Введение

В библиотеке JMantic есть 1 супертип - sc-элемент, и 3 конкретных типа sc-элемента.

2.1.1 ScElement

Любой объект в sc-памяти является ScElement. Характеризуется данный тип наличием sc-адреса, то есть, адреса сущности в sc-памяти. Sc-адреса в рамках одной базы уникальны.

2.1.2 ScNode

Данная сущность расширяет ScElement добавлением типа. ScNode может иметь тип константности, переменности и т.д. Однако, JMantic любую ScNode обрабатывает единым образом, вне зависимости от её типа.

2.1.3 ScEdge

ScEdge также, как и ScNode имеет тип. Однако, кроме типа, ScEdge соединяет две любые сущности ScElement.

2.1.4 ScLink

ScLink предназначена для хранения информации в sc-памяти. На данный момент доступно 4 формата сохранения информации: Целые числа, Дробные числа, Строки, Бинарная информация (не реализовано в JMantic). Для каждого типа информации (кроме бинарного) в библиотеке JMantic есть соответствующий интерфейс.

2.2 Описание структуры языка SC

Язык SC(Semantic Code) - это расширение языка SCB(Semantic Code Basic) путем включения в число текстовых элементов не только знаков множеств, но и переменных. Таким образом, элементы, входящие в состав sc-текстов (т.е. sc-элементы) делятся на следующие классы:

- sc-константы (константные sc-элементы; sc-элементы, являющиеся знаками множеств; sc-элементы, каждый из которых имеет одно значение, каковым является сам этот элемент; sc-элементы нулевого уровня; scb-элементы);
- простые sc-переменные (sc-элементы, значениями которых являются sc-константы; sc-элементы 1-го уровня; sc-переменные 1-го уровня);
- sc-метапеременные (sc-элементы, значениями которых являются sc-переменные; sc-элементы 2-го уровня).

В свою очередь, sc-переменные разбиваются на следующие подклассы:

- переменные, значениями которых являются знаки множеств неуточняемого типа;
- переменные, значениями которых являются знаки пар принадлежности;
- переменные, значениями которых являются знаки узловых множеств;
- метапеременные, значениями которых являются переменные, значениями которых являются знаки множеств неуточняемого типа;

- метапеременные, значениями которых являются переменные, значениями которых являются знаки пар принадлежности;
- метапеременные, значениями которых являются переменные, значениями которых являются знаки узловых множеств.

Язык SC, как и язык SCB, имеет две модификации - язык SCs (Semantic Code string) и язык SCg (Semantic Code graphical).

Вводятся также ребра, указывающие на совпадение либо возможные совпадения значения некоторой переменной со значениями переменной или константы.

2.3 Язык SCg (Semantic Code Graphical) - графическая модификация языка SC

Так как набор элементов языка SC по сравнению с языком SCB значительным образом расширен, то его графическая (нелинейная) модификация требует введения дополнительных графических примитивов для изображения различных типов sc-элементов. В связи с этим в языке SC приняты следующие соглашения:

- константные sc-элементы неуточняемого типа и константные sc-узлы изображаются кружочками;
- переменные sc-элементы неуточняемого типа и переменные sc-узлы изображаются квадратами, ориентированными по вертикали и горизонтали;
- изображение sc-метапеременных отличается тем, что изображающий их квадрат повернут на 45 градусов;
- изображение sc-элементов неуточняемого типа от изображения sc-узлов отличается уменьшенным размером;
- изображения дуг и ребер, являющихся константами, простыми переменными и метапеременными, отличаются друг от друга тем, что константные дуги и ребра представлены сплошными линиями (того или иного вида), простые переменные - пунктирными линиями, а метапеременные - штрих-пунктирными линиями.

2.4 Основные графические примитивы языка SC

Константы	Переменные	Мета-переменные	Пояснения
•	■	◆	изображение sc-элемента неуточняемого типа
○	□	◇	изображение sc-узла неуточняемого типа
●	■	◆	обозначение предметного множества

⊙	⊠	◇	обозначение узлового непредметного множества
⊕	⊞	⊞	обозначение множества знаков пар принадлежности
⊗	⊗	⊞	обозначение отношения
⊖	⊞	⊞	обозначение ориентированной связки
⊕	⊞	⊞	обозначение неориентированной связки
⊙	⊠	⊞	обозначение атомарной логической формулы
⇒	⇒	⇒	обозначение простой ориентированной пары с дополнительно уточняемой семантикой
→	→	→	обозначение пары принадлежности
→	→	→	обозначение пары не принадлежности
→	→	→	обозначение пары нечеткой принадлежности
≡	≡	≡	обозначение пары равенства значений

	обозначение неориентированной (неупорядоченной) пары с дополнительно уточняемой семантикой		
	замкнутая линия, являющаяся обозначением множества sc-элементов, изображенных внутри этой линии		
			шинная линия, являющаяся способом увеличения контактной зоны sc-узла
			линия, ограничивающая содержимое узла

2.5 Язык SCs (Semantic Code string) – линейная модификация языка SC

В число разделителей и ограничителей языка SCs входят все разделители и ограничители языка SCBs. Кроме этого, к указанному перечню разделителей и ограничителей добавляются новые:

Константные разделители		Переменные разделители		Метапеременные разделители		Пояснение	
;		разделитель scs-предложений					
,		разделитель идентификаторов sc-элементов в сложных scs-предложениях					
,		разделитель слов в простых идентификаторах sc-элементов					
Begin		признак начала scs-текста, в котором гарантировано отсутствие неявной омонимии					
End;		признак конца scs-текста, в котором гарантировано отсутствие неявной омонимии					
/*	*/	левый и правый scs-ограничители комментария (в scs-тексте)					
/"	"/	левый и правый scs-ограничители содержимого sc-узла					
—	—	scbs-связки инцидентности					
→	←	→→	←←	→→→	←←←	scs-связки принадлежности	
↔	↔	↔↔	↔↔	↔↔↔	↔↔↔	scs-связки непринадлежности	
~→	~←	~→→	~←←	~→→→	~←←←	scs-связки нечеткой принадлежности	
{	}	{·	·}	{··	··}	scs-ограничители сложного идентификатора неориентированного множества	
<	>	<·	·>	<··	··>	scs-ограничители сложного идентификатора кортежа	
[]	[·	·]	[··	··]	scs-ограничители сложного идентификатора системы множеств	
:		::		:::		разделители атрибута и компонента кортежа	
/:	:/	/::	::/	/:::	:::/	scs-ограничители, неявно задающие дугу, выходящую из элемента, указанного внутри данных ограничителей, и входящего в sc-элемент, записанный непосредственно справа от правого ограничителя	
()	ограничители обозначения неидентифицируемой связи, ограничители аргументов функций					
⇒	⇐	⇒⇒	⇐⇐	⇒⇒⇒	⇐⇐⇐	связка, соответствующая ориентированной паре неуточняемого вида	
↔		↔↔		↔↔↔		связка, соответствующая неориентированной паре неуточняемого вида	
=		==		===		связка равенства значений	
≠		≠≠		≠≠≠		связка неравенства значений	

\approx		$\approx \approx$		$\approx \approx \approx$		связка нечеткого равенства значений
scs-связки бинарных отношений над множествами:						
\equiv		\equiv		\equiv		обозначение пары равенства множеств
\neq		\neq		\neq		обозначение пары неравенства множеств
\equiv		\equiv		\equiv		обозначение пары эквивалентности множеств по набору элементов
$\not\equiv$		$\not\equiv$		$\not\equiv$		обозначение пары неэквивалентности множеств по набору элементов
\square		\square		\square		обозначение пары пересекающихся множеств
\boxtimes		\boxtimes		\boxtimes		обозначение пары непересекающихся множеств
\subset	\supset	\subset	\supset	\subset	\supset	связка строгого включения множеств
$\not\subset$	$\not\supset$	$\not\subset$	$\not\supset$	$\not\subset$	$\not\supset$	связка строгого не включения множеств
\subseteq	\supseteq	\subseteq	\supseteq	\subseteq	\supseteq	связка включения множеств
$\not\subseteq$	$\not\supseteq$	$\not\subseteq$	$\not\supseteq$	$\not\subseteq$	$\not\supseteq$	связка не включения множеств
scs-связки бинарных отношений над числами:						
$<$	$>$	$<$	$>$	$<$	$>$	связка строгого сравнения чисел
\leq	\geq	\leq	\geq	\leq	\geq	связка нестрогого сравнения чисел
имена унарных функций над числами:						
$—$	обозначение операции арифметического отрицания					
<u><i>abs</i></u>	обозначение функции взятия абсолютного значения					
$!$	обозначение функции факториала					
<u><i>exp</i></u>	обозначение показательной функции					
<u><i>ln</i></u>	обозначение функции взятия натурального логарифма					
<u><i>sin</i></u> <u><i>cos</i></u> <u><i>tg</i></u> <u><i>ctg</i></u>	обозначения тригонометрических функций					

scs-связки бинарных функций над множествами:			
\cup	\cup	\cup	связка объединения множеств
\cap	\cap	\cap	связка соединения множеств
\cap	\cap	\cap	связка пересечения множеств
\setminus	\setminus	\setminus	связка разности множеств
Δ	Δ	Δ	связка симметрической разности множеств
\times	\times	\times	связка декартова произведения
scs-связки бинарных функций над числами:			
$+$	$+$	$+$	связка арифметического сложения
\cdot	\cdot	\cdot	связка арифметического умножения
$-$	$-$	$-$	связка арифметического вычитания
$/$	$/$	$/$	связка арифметического деления
\uparrow	\uparrow	\uparrow	связка возведения в степень
$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	связка взятия корня
имена бинарных функций над числами:			
<u><i>log</i></u>	обозначение функции взятия логарифма		

2.6 Простые примеры для понимания сущности SC языка

2.6.1 Пример на языке SCs:

В данном примере можно наглядно увидеть, как можно описать понятие "Устройство на языке SCs, конечно, не в полной мере, но в общих чертах однозначно.

Конкретно здесь мы видим, что устройство включается в такое понятие, как изобретение. А также есть ссылка на файл, в котором находится определение этого понятия на русском языке: рукотворный объект со сложной внутренней структурой, созданный для выполнения определённых функций.

```

1 concept_device
2
3   <= nrel_inclusion: concept_invention ;
4
5   => nrel_main_idtf:
6       [Device]
7       (*<-lang_ru;;*);

```

```

8      [Device]
9      (*<-lang_en;;*);
10
11  => nrel_idtf:
12      [device]
13      (*<- lang_ru;;*);
14
15  <- rrel_key_sc_element: ...
16      (*
17          <- definition;;
18
19          => nrel_main_idtf:
20              [Definition.(Device)]
21              (*<-lang_ru;;*);;
22
23          <= nrel_sc_text_translation: ...
24              (*
25                  -> rrel_example:
26                      "file://html/concept-device.html"
27                      (*<-lang_ru;;*);;
28                  *);;
29
30          <= nrel_using_constants: ...
31              (*
32                  -> subject;; //ims
33                  -> _structure;; //ims
34                  -> concept_function;;
35                  *);;
36      *);;

```

Содержимое файла, соответствующее пути в примере:

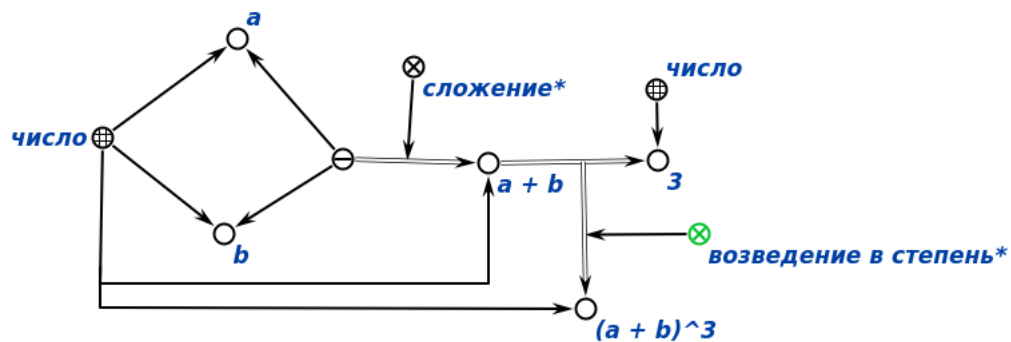
Устройство — это рукотворный объект со сложной внутренней структурой, созданный для выполнения определённых функций.</p>

2.6.2 Пример на языке SCg

Для примера на SCg было взято такое музыкальное направление как "Регги". На рисунке мы можем видеть, что Регги - это музыкальное направление, которое зародилось в такой стране, как Ямайка. Также здесь показано, что Боб Марли, певец и автор песен, является королём этого жанра.



Также на языке SCg можно описывать и математические выражения. В примере описано выражение $(a+b)$ в степени 3:



3 Ostis

В данном разделе описывается протокол и формат обмена сообщений. Раздел предназначен, в первую очередь, для разработчиков JMantic, и описывает внутренние механизмы работы библиотеки.

3.1 Обзор

Запросы и ответы к sc-machine работают по протоколу WebSocket, и представляют из себя текст в формате json с одним корневым блоком. Каждый запрос содержит идентификатор, тип запроса и полезную нагрузку. Ответ содержит идентификатор, который соответствует идентификатору запроса, статус и полезную нагрузку.

3.2 Структура запрос

- id — идентификатор запроса.
- type — тип запроса. Задаёт команду, которую нужно выполнить sc-machine.
- payload — последовательность данных запроса/ответа. Для разных типов запросов/ответов содержание данной секции будут отличаться. Может содержать как один набор данных, так и массив наборов данных.

3.3 Структура ответ

- id — идентификатор ответа.
- status — состояние ответа. Может принимать значения true и false. При успешном выполнении запроса данное поле будет содержать true. В ином случае - false. Если данный ответ сгенерирован событием sc-machine, данного поля может не быть.
- event — флаг ответа. Показывает, сгенерирован ли данный ответ событием. Если да — поле будет иметь значение true. В ином случае этого поля может не быть или значение равно false.
- payload — последовательность данных запроса/ответа. Для разных типов запросов/ответов содержание данной секции будут отличаться. Может содержать как один набор данных, так и массив наборов данных.

3.4 Поле id

Идентификатор представляет из себя целое число, которое служит для сопоставления ответа запросу при асинхронном режиме работы. То есть, на запросу с определённым id придёт ответ с таким же id.

Запрос:

```
1 {  
2   "id": 1,  
3   "type": "request type",  
4   "payload": {  
5     ...  
6   }  
7 }
```

Ответ:

```

1 {
2 {
3   "id": 1,
4   "status": true,
5   "event": false,
6   "payload": {
7     ...
8   }
9 }
10 }

```

3.5 Поле type

Тип запроса — это строка из набора допустимых запросов. Возможно использовать следующие типы запросов:

3.5.1 create_elements

create_elements — запрос на создание sc-элементов. Запрос может содержать набор элементов, которые нужно создать, но каждый элемент должен представлять один из трёх допустимых вариантов:

- node — sc-узел;
- edge — sc-дуга;
- link — sc-ссылка.

Каждый элемент должен иметь тип. Допустимые типы описаны в таблице(2). В секции payload можно указать как один элемент, так и несколько. В ответе на запрос в секции payload будут указаны адреса созданных объектов. При этом каждый адрес будет соответствовать элементу запроса в том же порядке (первый элемент - первый адрес). Пример:

```

1 {
2   "id": 2,
3   "type": "create_elements",
4   "payload": [
5     {
6       "el": "node",
7       "type": 1
8     },
9     {
10      "el": "link",
11      "type": 2,
12      "content": 45.4
13    },
14    {
15      "el": "edge",
16      "src": {
17        "type": "ref",
18        "value": 0
19      },
20      "trg": {
21        "type": "ref",
22        "value": 1

```

```

23     },
24     "type": 32
25   }
26 ]
27 }

```

При создании sc-ссылки можно также указать тип содержимого (см. описание запроса content).

Пример секции с указанием типа:

```

1 {
2   "el": "link",
3   "type": 2,
4   "content": "data",
5   "content_type": "string"
6 }

```

3.5.2 check_elements

check_elements — запрос на проверку существования sc-элементов. В payload указываются адреса sc-элементов, существование которых необходимо проверить. В секции payload ответа будут указаны числа, которые соответствуют типу sc-элемента в случае существования элемента. Если элемент не найден, в ответе будет указано число 0, иначе будет указан числовой код найденного элемента. Порядок элементов в ответе соответствует порядку элементов в запросе.

Пример запроса:

```

1 {
2   "id": 3,
3   "type": "check_elements",
4   "payload": [
5     23123,
6     432,
7     ...
8   ]
9 }

```

Пример ответа:

```

1 {
2   "id": 3,
3   "payload": [
4     32,
5     0,
6     ...
7   ]
8 }

```

3.5.3 delete_elements

delete_elements — запрос на удаление sc-элементов. Работает аналогично с запросом проверки существования. Пример:

```

1 {
2   "id": 4,
3   "type": "delete_elements",
4   "payload": [
5     23123,
6     432,
7     ...
8   ]
9 }

```

3.5.4 search_template

search_template — запрос на поиск sc-конструкции по шаблону. Шаблон представляет из себя список, каждый из элементов которого является тройкой элементов, часть из которых известны. В ответе, если шаблон найден, будут подставлены адреса неизвестных ранее элементов. Также можно указать некоторые тройки шаблона как необязательные. Тогда в ответе, если такие тройки не найдены, на их месте будут нули, но остальным тройкам шаблона будут подставлены адреса соответствующих элементов. Это можно сделать указав после блоков элементов тройки дополнительный блок is_required, присвоив ему значение false.

Пример:

```

1 {
2   "id": 5,
3   "type": "search_template",
4   "payload": [
5     [
6       {
7         "type": "addr",
8         "value": 23123
9       },
10      {
11        "type": "type",
12        "value": 32,
13        "alias": "_edge1"
14      },
15      {
16        "type": "type",
17        "value": 2,
18        "alias": "_trg"
19      }
20    ],
21    [
22      {
23        "type": "addr",
24        "value": 231342
25      },
26      {
27        "type": "type",
28        "value": 2000,
29        "alias": "_edge2"
30      },
31      {
32        "type": "alias",

```



```

33     "value": "_edge1"
34   },
35   {
36     "is_required": false
37   }
38 ],
39 ...
40 ]
41 }

```

Ответ для данного запроса будет выглядеть следующим образом:

```

1 {
2   "id": 5,
3   "payload": {
4     "aliases": {
5       "trg": 2,
6       "edge1": 1,
7       "edge2": 4
8     },
9     "addrs": [
10      [23123, 412, 423, 231342, 282, 412],
11      [23123, 6734, 85643, 231342, 4234, 6734],
12      [23123, 7256, 252, 0, 0, 0],
13      ...
14    ]
15  }
16 }

```

Также существует возможность указать шаблон поиска с помощью текста на языке SCs. Пример:

```

1 {
2   "id": 6,
3   "type": "search_template",
4   "payload": "person _-> .._p (* _=> nrel_email:: _[] *);;"
5 }

```

3.5.5 generate_template

generate_template — запрос на создание sc-конструкций по шаблону. Работает по схожему с поиском принципу.

Пример запроса:

```

1 {
2   "id": 7,
3   "type": "generate_template",
4   "payload": {
5     "templ":
6     [
7       {
8         "params": [
9           {
10            "type": "addr",

```

```

11         "value": 23123
12     },
13     {
14         "type": "type",
15         "value": 32,
16         "alias": "_edge1"
17     },
18     {
19         "type": "type",
20         "value": 2,
21         "alias": "_trg"
22     }
23 ]
24 },
25 {
26     "params": [
27         {
28             "type": "addr",
29             "value": 231342
30         },
31         {
32             "type": "type",
33             "value": 2000,
34             "alias": "_edge2"
35         },
36         {
37             "type": "alias",
38             "value": "_edge1"
39         }
40     ]
41 },
42 ...
43 ],
44 "params": {
45     "_trg": 564
46 }
47 }
48 }

```

Ответ на запрос:

```

1 {
2     "id": 7,
3     "payload": {
4         "aliases": {
5             "_trg": 2,
6             "_edge1": 1,
7             "_edge2": 4
8         },
9         "addrs": [23123, 4953, 564, 231342, 533, 4953]
10     }
11 }

```

Как и в случае поиска, шаблон может быть представлен текстом на языке SCs:

```

1 {
2   "id": 8,
3   "type": "generate_template",
4   "payload": {
5     "templ": "person _-> .._p (* _=> nrel_email:: _[test@email.com]
6       *);;",
7     "params": {
8       ".._p": 5314
9     }
10  }

```

3.5.6 events

events — запрос на подпись на события или удалить существующие подписки. Тип события представляет из себя строку и указывается в разделе payload. Существует 6 видов событий, на которые можно подписаться:

- add_outgoing_edge — генерирует событие, при создании выходящей sc-дуги из наблюдаемого sc-элемента
- add_ingoiing_edge — генерирует событие, при создании входящей sc-дуги в наблюдаемый sc-элемент;
- remove_outgoing_edge — генерирует событие, при удалении выходящей sc-дуги из наблюдаемого sc-элемента
- remove_ingoiing_edge — генерирует событие, при удалении входящей sc-дуги в наблюдаемый sc-элемент;
- content_change — генерирует событие при изменении содержания у наблюдаемой sc-ссылки;
- delete_element — генеирует событие при удалении наблюдаемого sc-элемента.

Чтобы подписаться на событие, необходимо в разделе payload указать раздел create, в котором перечислить все события, на которые необходимо подписаться. Сам блок описания вида события представлен типом события из списка а также адресом sc-элемента, который будет генерировать ответы при событиях.

Для удаления подписки на событие необходимо в блоке payload создать блок delete, в котором перечислить идентификаторы запросов, которыми была осуществлена подписка.

Пример:

```

1 {
2   "id": 9,
3   "type": "events",
4   "payload": {
5     "create": [
6       {
7         "type": "add_output_edge",
8         "addr": 324
9       }
10    ],
11    "delete": [
12      2, 4, 5

```

```
13 ]
14 }
15 }
```

3.5.7 keynodes

keynodes — запрос на получение и редактирование ключевых узлов (keynodes). С помощью данной команды можно найти адрес узла в памяти sc-machine по его идентификатору или создать новый идентификатор(идентификаторы в рамках sc-machine уникальны).

- find — поиск sc-элемента по идентификатору
- resolve — поиск sc-элемента по идентификатору. Если идентификатора не существует, то будет создан новый, адрес которого будет содержаться в ответе. При этом, кроме идентификатора, в запросе нужно указать тип элемента (должен быть узлом).

Пример запроса:

```
1 {
2   "id": 11,
3   "type": "keynodes",
4   "payload": [
5     {
6       "command": "find",
7       "idtf": "any system identifier that NOT exist"
8     },
9
10    {
11      "command": "find",
12      "idtf": "any system identifier that exist"
13    },
14
15    {
16      "command": "resolve",
17      "idtf": "NOT exist",
18      "elType": 1
19    },
20
21    {
22      "command": "resolve",
23      "idtf": "exist",
24      "elType": 1
25    }
26  ]
27 }
```

Пример ответа:

```
1 {
2   "id": 11,
3   "status": true,
4   "payload": [
5     0,
6     321,
7     435,
```

```

8      324
9    ]
10  }

```

3.5.8 content

content — запрос на получение и редактирование содержимого sc-ссылки. Sc-ссылки могут хранить один из 4 типов:

- int — целое число;
- float — вещественное число;
- string — строка;
- binary — бинарные данные.

Для работы с содержимым sc-ссылке используются 2 блока: get и set. Блок set служит для изменения содержимого существующей sc-ссылки, и принимает тип хранимого значения, само значение, адрес sc-ссылки в sc-памяти. Блок get служит для получения содержимого sc-ссылки, принимая адрес этой ссылки в sc-памяти. В ответе на get-блок будет содержаться значение хранимого типа (либо null, если содержимого нет) и сам тип. Ответ на set-запрос представляет из себя одно слово — true, если значение записалось и false если запись не удалась.

Пример запроса:

```

1  {
2    "id": 10,
3    "type": "content",
4    "payload": [
5      {
6        "command": "set",
7        "type": "int",
8        "data": 67,
9        "addr": 3123
10     },
11     {
12       "command": "get",
13       "addr": 232
14     },
15   ]
16 }

```

Ответ на предыдущий запрос будет следующим:

```

1  {
2    "id": 10,
3    "payload": [
4      true,
5      {
6        "value": 56.7,
7        "type": "float"
8      },
9    ]
10 }

```

3.6 Таблица базовых типов sc-элементов

Тип	Шестнадцатеричный	Десятичный
sc_type_node	0x1	1
sc_type_link	0x2	2
sc_type_uedge_common	0x4	4
sc_type_dedge_common	0x8	8
sc_type_edge_access	0x10	16
sc_type_const	0x20	32
sc_type_var	0x40	64
sc_type_edge_pos	0x80	128
sc_type_edge_neg	0x100	256
sc_type_edge_fuz	0x200	512
sc_type_edge_temp	0x400	1024
sc_type_edge_perm	0x800	2048
sc_type_node_tuple	0x80	128
sc_type_node_struct	0x100	256
sc_type_node_role	0x200	512
sc_type_node_norole	0x400	1024
sc_type_node_class	0x800	2048
sc_type_node_abstract	0x1000	4096
sc_type_node_material	0x2000	8192
sc_type_arc_pos_const_perm	0x8B0	2224
sc_type_arc_pos_var_perm	0x8D0	2256

Рис. 1 – Базовые типы sc-элементов в json-формате

Остальные виды sc-элементов получаются комбинированием характеристик из этой таблицы с помощью операции логического ИЛИ. Например, переменная унарная дуга может быть получена комбинацией кода унарной дуги `sc_type_uedge_common` и кода переменности `sc_type_var`:

$$sc_type_uedge_common_var = sc_type_uedge_common | sc_type_var$$

Что будет равно сумме десятичных кодов элементов: $4 + 64 = 68$.

3.7 Полная таблица поддерживаемых типов

На данный момент jmantis поддерживает 48 типов sc-элементов.

Тип элемента	Десятичный код элемента
EdgeUCommon	4
EdgeDCommon	8
EdgeUCommonConst	36
EdgeDCommonConst	40
EdgeUCommonVar	68
EdgeDCommonVar	72
EdgeAccess	16
EdgeAccessConstPosPerm	2224
EdgeAccessConstNegPerm	2352
EdgeAccessConstFuzPerm	2608
EdgeAccessConstPosTemp	1200
EdgeAccessConstNegTemp	1328
EdgeAccessConstFuzTemp	1584
EdgeAccessVarPosPerm	2256
EdgeAccessVarNegPerm	2384
EdgeAccessVarFuzPerm	2640
EdgeAccessVarPosTemp	1232
EdgeAccessVarNegTemp	1360
EdgeAccessVarFuzTemp	1616
Const	32
Var	64
Node	1
Link	2
Unknown	undefined
NodeConst	33
NodeVar	65
LinkConst	34
LinkVar	66
NodeStruct	257
NodeTuple	129
NodeRole	513
NodeNoRole	1025
NodeClass	2049
NodeAbstract	4097
NodeMaterial	8193
NodeConstStruct	289
NodeConstTuple	161
NodeConstRole	545
NodeConstNoRole	1057
NodeConstClass	2081
NodeConstAbstract	4129
NodeConstMaterial	8225
NodeVarStruct	321
NodeVarTuple	193
NodeVarRole	577
NodeVarNoRole	1089
NodeVarClass	2113
NodeVarAbstract	4161
NodeVarMaterial	8257

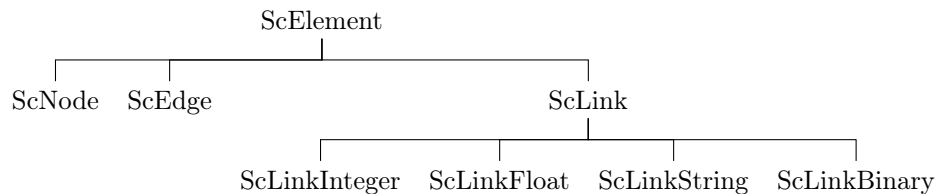
Рис. 2 – Список всех типов sc-элементов

4 Подробности реализации JMantic

4.1 Иерархия sc-типов

В JMantic представлен один абстрактный родительский тип `ScElement` и 3 типа конкретных sc-элементов: `ScNode`, `ScEdge`, `ScLink`. Также `ScLink` имеет 4 подтипа, для каждого варианта возможного содержимого: `ScLinkInteger`, `ScLinkFloat`, `ScLinkString`, `ScLinkBinary`.

Дерево типов выглядит следующим образом:



4.2 Интерфейс ScMemory

Интерфейс `ScMemory` является ядром библиотеки. Именно данный интерфейс описывает контракт между java-кодом и sc-памятью. В интерфейсе определено множество методов.

4.2.1 createNodes

Данный метод, как понятно из названия, создаёт узлы (один или несколько) в sc-памяти. Методу передаётся список типов. Для каждого элемента списка будет создан узел с соответствующим типом. Метод вернёт список объектов типа `ScNode`.

В контексте `Ostis`, данный метод создаёт и отправляет один запрос на создание группы элементов.

4.2.2 createEdges

Как и метод `createNodes`, данный создаёт sc-элементы в остисе, а точнее – дуги. Для создания дуг в этот метод необходимо передать узлы, между которыми создастся дуга, а также тип создаваемой дуги. Метод возвращает список объектов, которые реализуют `ScEdge`.

Как и в случае с `createNodes`, в `Ostis` будет послан один запрос с набором всех дуг, которые необходимо создать.

4.2.3 createLinks

Данного метода нет в интерфейсе, но вместо него объявлены методы для каждого типа `ScLink`. Все контракты применимы к любому из группы методов.

Каждый метод создаёт в sc-памяти `ScLink` с содержимым определённого типа (`integer`, `float`, `string`). Для использования метода, ему необходимо передать список sc-типов самих sc-элементов (`var`, `const` и т.д.), а также список содержимого соответствующего типа (`integer`, `float`, `string`). Для каждого элемента списка типов будет создан `ScLink` с соответствующим содержимым из списка содержимого. В результате вернётся список объектов, которые наследуются от одного из конкретных вариантов `ScLink` (`ScLinkInteger`, `ScLinkFloat`, `ScLinkString`)

В каждом методе в `Ostis` будет отправлен один запрос. То есть, один запрос, но все `ScLink` в этом запросе будут иметь единый тип содержимого.

4.2.4 deleteElements

Метод для удаления любого sc-элемента из sc-памяти. Ему нужно передать список sc-элементов, которые хочется удалить. Вернёт метод статус — true, если удаление прошло успешно, false в ином случа.

Как и все предыдущие методы, при работе с Ostis будет отправлен один запрос с списком адресов всех элементов, которые нужно удалить.

4.2.5 findByTemplateNodeEdgeNode

Данный метод является одним из методов поиска по шаблону. Если точнее, то этот метод ищет конструкции вида узел-дуга-узел при известном одном из узлов. Для работы методу требуется передать ScNode, который будет играть роль фиксированного узла, а также тип дуги и тип второго узла. В результате работы, метод вернёт список ScEdge, где каждая дуга одним концом присоединена к фиксированному узлу.

Если библиотека настроена на работу с Ostis, то в данном методе будет составлен один запрос, который будет послан серверу.

4.2.6 find

Универсальный метод для поиска любых допустимых конструкций в sc-памяти. Данный метод принимает объект типа ScPattern, и возвращает список найденных конструкций по данному шаблону.

4.2.7 findByPattern3

Данный метод является частным случаем поискового запроса find. Если точнее, данный метод предназначен для поиска по шаблону из 3-х элементов. Отличие от универсального метода поиска в наличии проверки типов на этапе компиляции а также облегченный способ создания объектов шаблонов ScPattern3 с помощью фабрик DefaultScPattern3Factory и ScPattern3FactoryWithNames.

4.2.8 findByPattern5

Данный метод является частным случаем поискового запроса find. Если точнее, данный метод предназначен для поиска по шаблону из 5-и элементов. Отличие от универсального метода поиска в наличии проверки типов на этапе компиляции а также облегченный способ создания объектов шаблонов ScPattern5 с помощью фабрик DefaultScPattern5Factory и ScPattern5FactoryWithNames.

4.2.9 setLinkContent

Данного метода не объявлено в интерфейсе, но вместо него есть группа методов для каждого типа содержимого ScLink с похожими названиями. Методы служат для изменения значения уже существующих ScLink. Для работы необходимо передать список ScLink, содержимое которых будет меняться, а также список содержимого. Содержимое каждого элемента списка ScLink будет заменено на соответствующий элемента списка содержимого. В результате метод вернёт ScLink с измененным содержимым.

При работе с Ostis будет послан один запрос для каждого метода по отдельности. То есть, при вызове одного метода, будет послан один запрос. Но при вызове нескольких методов подряд, будет послано несколько запросов.

4.2.10 getLinkContent

Метод также заменён на группу методов со схожими названиями. Работают они аналогично методам setLinkContent.

4.2.11 findKeynodes

Метод для поиска ключевых узлов по известному идентификатору. Если идентификатор существует, будет возвращён непустой Optional, в котором лежит узел. В противном случае, Optional будет пуст.

4.2.12 resolveKeynodes

Метод для поиска и создания узлов по идентификатору. Если узла с переданным идентификатором не существует, будет создан новый узел с заданным типом, а иначе будет просто возвращён узел с указанным идентификатором (как в методе findKeynodes).

4.3 Реализация запросов и ответов Ostis

4.3.1 Запрос

Абстрактный запрос к sc-machine представлен интерфейсом **ScRequest**. Сервер sc-machine принимает следующие типы запросов:

- create_elements — запрос на создание sc-элементов. В библиотеке ему соответствует интерфейс **CreateElementsRequest**.
- check_elements — запрос на проверку существования sc-элементов. В библиотеке он представлен интерфейсом **CheckScElTypeRequest**.
- delete_elements — запрос на удаление sc-элементов. В библиотеке данный запрос описывает интерфейс **DeleteElementsRequest**.
- search_template — запрос на поиск sc-конструкции по шаблону. Запросу соответствует интерфейс **FindByPatternRequest**.
- keynodes — запрос на получение и редактирование ключевых узлов (keynodes). В библиотеке представлен интерфейсом **KeynodeRequest**.
- content — запрос на получение и редактирование содержимого sc-links. Представлен интерфейсами **SetLinkContentRequest** и **GetLinkContentRequest**.

4.4 Создание поисковых шаблонов

Для поиска по шаблону в интерфейсе ScMachine описаны 3 метода: findByPattern3, findByPattern5, find. Функционально методы аналогичны, различие заключается лишь в наличии проверки java-типов на этапе компиляции, а также удобства создания шаблонов.

- findByPattern3. Для использования данного метода необходимо передать объект типа ScPattern3. Получить такой объект можно двумя способами. Первый способ наиболее предпочтительный и удобный — использование одной из фабрик (DefaultScPattern3Factory, ScPattern3FactoryWithNames). В этом случае фабрике необходимо только передать исходные типы, а вся работа с generic-ами происходит внутри. Второй способ создания заключается в создании непосредственно объекта класса ScPattern3Impl, но это несколько сложнее, так как необходимо изучить семантику generic-параметров класса.
- findByPattern5. Для использования данного метода необходимо передать объект типа ScPattern5. Получить такой объект можно двумя способами. Первый способ наиболее предпочтительный и удобный — использование одной из фабрик (DefaultScPattern5Factory,

ScPattern5FactoryWithNames). В этом случае фабрике необходимо только передать искомые типы, а вся работа с generic-ами происходит внутри. Второй способ создания заключается в создании непосредственно объекта класса ScPattern5Impl, но это несколько сложнее, так как необходимо изучить семантику generic-параметров класса.

- **find.** Данный метод является обобщением поисковых шаблонов и может искать любые допустимые конструкции (включая конструкции на 3 и на 5 элементов). Для поиска необходимо передать объект типа ScPattern, который, в отличие от ScPattern3 и ScPattern5 не имеет такой строгой типобезопасности. Создать такой шаблон можно только одним способом – созданием одной из реализаций интерфейса ScPattern (на данный момент доступен класс DefaultWebsocketScPattern). Однако, после создания необходимо заполнить объект шаблона правилами поиска. Каждое правило поиска представляет из себя тройку поисковых элементов. Поисковый элемент не является sc-элементом, но служит для описания sc-элементов, которые необходимо найти. На данный момент доступно 3 поисковых элемента:
 1. **ScFixedElement.** Служит для указания в шаблоне конкретных (известных) sc-элементов. Может представлять любой элемент типа ScElement. При поиске указанные элементы являются "опорными" и не меняются (то есть, в каждой найденной конструкции будут присутствовать все фиксированные элементы).
 2. **ScTypedElement.** Данный поисковый элемент описывает тип sc-элемента, который необходимо найти. Может принимать как значения типов узлов, дуг, линков, так и значение "неизвестного элемента который в библиотеке представлен enum-ом UnknownScElement. Также поисковый элемент данного типа может иметь алиас. Алиас, как понятно из названия, задаёт второе имя. Это может пригодиться в случае, если неизвестный элемент, который будет найден при поиске, участвует в другой конструкции. Как самый простой пример - поиск на 5 элементов, который можно составить из двух троек: {фиксированный-узел, **дуга**, узел}, {фиксированный-узел-отношения, дуга-отношения, **дуга**}. В примере мы не знаем адрес дуги, но при этом есть необходимость для второй тройки как-то эту дугу идентифицировать.
 3. **ScAliasedElement.** Данный поисковый элемент описывает алиас для ScTypedElement. Подробнее в предыдущем пункте.

5 Публичный api

В данном разделе описаны классы и интерфейсы, которые предоставляют внешний пользовательский механизм по работе с библиотекой JMantic

5.1 DefaultScContext

Данный тип Sc-контекста является рекомендуемым к использованию, так как данная реализация поддерживает наиболее актуальные методы и полностью реализует типобезопасность java. Кроме этого, вынуждает пользователя обрабатывать исключения, выбрасываемые ScMemory.

DefaultScContext предоставляет программисту следующие методы:

5.1.1 createNode

Создание узла заданного типа. Данный метод создаёт узел в sc-памяти и возвращает объект, который является реализацией интерфейса ScNode. Полученный объект имеет адрес и тип.

5.1.2 createNodes

Создание группы узлов. Данный метод позволяет оптимизировать групповые запросы создания узлов. При создании узла по одному, происходит генерация json-a, отправка на сервер остис, разбор ответа для каждого узла лично. В групповом же варианте создаётся только один json-запрос.

5.1.3 createEdge

Создание дуги заданного типа между заданным sc-элементами. Данный метод создаёт в sc-памяти sc-дугу и возвращает объект, который является реализацией интерфейса ScEdge. Полученный объект имеет адрес, тип, а также адреса двух sc-элементов, между которыми находится данная дуга.

5.1.4 createEdges

Создание группы дуг. Данный метод позволяет оптимизировать групповые запросы создания дуг. При создании узла по одному, происходит генерация json-a, отправка на сервер остис, разбор ответа для каждой дуги лично. В групповом же варианте создаётся только один json-запрос.

5.1.5 createIntegerLink

Создание sc-link для хранения целочисленных значений. Данный метод создаёт в sc-памяти объект типа ScLink с целочисленным содержимым и устанавливает некоторое значение. Возвращает же метод одну из реализаций интерфейса ScIntegerLink, которая имеет адрес, тип хранимого значения и само хранимое значение.

5.1.6 createFloatLink

Создание sc-link для хранения дробных значений. Данный метод создаёт в sc-памяти объект типа ScLink с дробным содержимым и устанавливает некоторое значение. Возвращает же метод одну из реализаций интерфейса ScFloatLink, которая имеет адрес, тип хранимого значения и само хранимое значение.

5.1.7 createStringLink

Создание sc-link для хранения целочисленных значений. Данный метод создаёт в sc-памяти объект типа ScLink с целочисленным содержимым и устанавливает некоторое значение. Возвращает же метод одну из реализаций интерфейса ScIntegerLink, которая имеет адрес, тип хранимого значения и само хранимое значение.

5.1.8 deleteElement

Удаление любого ScElement. Данный метод удаляет sc-элемент из sc-памяти по адресу элемента. Возвращает метод логическое значение: true, если удаление прошло успешно; false, если закончилось с ошибкой.

5.1.9 deleteElements

Удаление группы элементов. Данный метод позволяет оптимизировать групповые запросы удаления элементов. При удалении элементов по одному, происходит генерация json-a, отправка на сервер остис, разбор ответа для каждого элемента лично. В групповом же варианте создаётся только один json-запрос.

5.1.10 findAllConstructionsNodeEdgeNode

Поиск всех конструкций вида узел-дуга-узел. Данный метод принимает один константный узел, относительно которого будет производиться поиск, а также тип искомой дуги и тип искомого узла. Метод вернёт список найденных конструкций, а если точнее, метод вернёт список из объектов-реализаций интерфейса ScEdge. Из данных объектов можно получить соседние узлы.

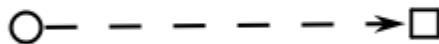


Рис. 3 – Пример искомой конструкции

5.1.11 findAllConstructionsNodeEdgeLink

(Удалён. Заменён на универсальные поисковые шаблоны конструкций на 3)

Поиск всех конструкций вида узел-дуга-sclink. Данный метод принимает один константный узел, относительно которого будет производиться поиск, а также тип искомой дуги, тип искомой sclink и тип содержимого искомой sclink. Метод вернёт список найденных конструкций, а если точнее, метод вернёт список из объектов-реализаций интерфейса ScEdge. Из данных объектов можно получить соседние элементы.

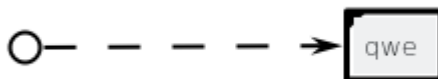


Рис. 4 – Пример искомой конструкции

5.1.12 findAllConstructionsNodeEdgeLinkWithRelation

(Удалён. Заменён на универсальные поисковые шаблоны конструкций на 5)

Поиск всех конструкций вида узел-дуга-sclink. Данный метод принимает один константный узел, относительно которого будет производиться поиск, а также тип искомой дуги, тип искомой sclink, тип содержимого искомой sclink, адрес узла-отношения, тип дуги отношения. Метод вернёт список найденных конструкций, а если точнее, метод вернёт список из объектов-реализаций интерфейса ScEdge. Из данных объектов можно получить соседние элементы.

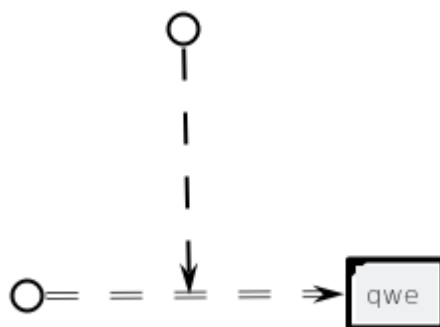


Рис. 5 – Пример искомой конструкции

5.1.13 find(ScPattern3)

Поиск всех конструкций по шаблону на 3 элемента. Данный метод принимает один шаблон и возвращает список найденных конструкций (также на 3 элемента). Для создания шаблона можно воспользоваться как обычным использованием класса ScPattern3Impl (то есть, ручным заполнением правил поиска), так и, что более предпочтительно, с помощью фабрик DefaultScPattern3Factory, ScPattern3FactoryWithNames.

5.1.14 find(ScPattern5)

Поиск всех конструкций по шаблону на 5 элементов. Данный метод принимает один шаблон и возвращает список найденных конструкций (также на 5 элементов). Для создания шаблона можно воспользоваться как обычным использованием класса ScPattern5Impl (то есть, ручным заполнением правил поиска), так и, что более предпочтительно, с помощью фабрик DefaultScPattern5Factory, ScPattern5FactoryWithNames.

5.1.15 find(ScPattern)

Поиск всех конструкций по универсальному шаблону. Данный метод принимает один шаблон и возвращает список найденных конструкций (количество элементов в конструкции будет равно количеству найденных по шаблону элементов). Создать шаблон можно только вручную с помощью одной из имплементаций (на данный момент - DefaultWebsocketScPattern).

5.1.16 setIntegerLinkContent

Задание значения sclink. Данный метод задаёт целочисленные значения в ScIntegerLink в sc-памяти. Возвращает метод логическое значение: true, если операция прошла успешно; false, если закончилось с ошибкой.

5.1.17 setFloatLinkContent

3

Задание значения sclink. Данный метод задаёт дробное значения в ScFloatLink в sc-памяти. Возвращает метод логическое значение: true, если операция прошла успешно; false, если закончилось с ошибкой.

5.1.18 setStringLinkContent

Задание значения sclink. Данный метод задаёт строковое значения в ScStringLink в sc-памяти. Возвращает метод логическое значение: true, если операция прошла успешно; false, если закончилось с ошибкой.

5.1.19 getIntegerLinkContent

Получение значения sclink. Данный метод служит для получения актуального целочисленного значения данной ScIntegerLink, делая запрос в базу. Однако, значение, хранимое в sclink можно также получить вызовом метода getContent. В этом случае не гарантируется то, что значение в sclink совпадает со значением в sc-памяти.

5.1.20 getFloatLinkContent

Получение значения sclink. Данный метод служит для получения актуального дробного значения данной ScFloatLink, делая запрос в базу. Однако, значение, хранимое в sclink можно также получить вызовом метода getContent. В этом случае не гарантируется то, что значение в sclink совпадает со значением в sc-памяти.

5.1.21 getStringLinkContent

Получение значения sclink. Данный метод служит для получения актуального целочисленного значения данной ScStringLink, делая запрос в базу. Однако, значение, хранимое в sclink можно также получить вызовом метода getContent. В этом случае не гарантируется то, что значение в sclink совпадает со значением в sc-памяти.

5.1.22 findKeynode

Поиск ключевых узлов. Данный метод служит для нахождения ключевого узла по заданному идентификатору. Принимает в параметрах строку – идентификатор, а возвращает Optional, который либо содержит в себе найденный узел, либо пуст, если переданный идентификатор не существует в базе.

5.1.23 resolveKeynode

Поиск и создание ключевых узлов. Данный метод служит для нахождения ключевого узла по заданному идентификатору. Принимает в параметрах строку и тип узла – идентификатор и тип создаваемого узла соответственно. Если узел с данным идентификатором существует, метод вернёт этот узел. В случае, если не существует, будет создан новый узел с заданными идентификатором и типом узла.

5.2 UncheckedScContext

Данный тип Sc-контекста не рекомендуется к реальному использованию, так как скрывает исключения, выбрасываемые в ScMemory. Этот Sc-контекст удобно использовать для демонстрации работы и тестирования, ведь позволяет не оборачивать все методы в блоки try-catch.

В данном контексте приведены следующие методы:

5.2.1 createNode

Создание узла заданного типа. Данный метод создаёт узел в sc-памяти и возвращает объект, который является реализацией интерфейса ScNode. Полученный объект имеет адрес и тип.

5.2.2 createNodes

Создание группы узлов. Данный метод позволяет оптимизировать групповые запросы создания узлов. При создании узла по одному, происходит генерация json-a, отправка на сервер остис, разбор ответа для каждого узла лично. В групповом же варианте создаётся только один json-запрос.

5.2.3 createEdge

Создание дуги заданного типа между заданным sc-элементами. Данный метод создаёт в sc-памяти sc-дугу и возвращает объект, который является реализацией интерфейса ScEdge. Полученный объект имеет адрес, тип, а также адреса двух sc-элементов, между которыми находится данная дуга.

5.2.4 createEdges

Создание группы дуг. Данный метод позволяет оптимизировать групповые запросы создания дуг. При создании узла по одному, происходит генерация json-a, отправка на сервер остис, разбор ответа для каждой дуги лично. В групповом же варианте создаётся только один json-запрос.

5.2.5 createIntegerLink

Создание sc-link для хранения целочисленных значений. Данный метод создаёт в sc-памяти объект типа ScLink с целочисленным содержимым и устанавливает некоторое значение. Возвращает же метод одну из реализаций интерфейса ScIntegerLink, которая имеет адрес, тип хранимого значения и само хранимое значение.

5.2.6 createFloatLink

Создание sc-link для хранения дробных значений. Данный метод создаёт в sc-памяти объект типа ScLink с дробным содержимым и устанавливает некоторое значение. Возвращает же метод одну из реализаций интерфейса ScFloatLink, которая имеет адрес, тип хранимого значения и само хранимое значение.

5.2.7 createStringLink

Создание sc-link для хранения целочисленных значений. Данный метод создаёт в sc-памяти объект типа ScLink с целочисленным содержимым и устанавливает некоторое значение. Возвращает же метод одну из реализаций интерфейса ScIntegerLink, которая имеет адрес, тип хранимого значения и само хранимое значение.

5.2.8 deleteElement

Удаление любого ScElement. Данный метод удаляет sc-элемент из sc-памяти по адресу элемента. Возвращает метод логическое значение: true, если удаление прошло успешно; false, если закончилось с ошибкой.

5.2.9 deleteElements

Удаление группы элементов. Данный метод позволяет оптимизировать групповые запросы удаления элементов. При удалении элементов по одному, происходит генерация json-а, отправка на сервер отис, разбор ответа для каждого элемента лично. В групповом же варианте создаётся только один json-запрос.

5.2.10 findAllConstructionsNodeEdgeNode

Поиск всех конструкций вида узел-дуга-узел. Данный метод принимает один константный узел, относительно которого будет производиться поиск, а также тип искомой дуги и тип искомого узла. Метод вернёт список найденных конструкций, а если точнее, метод вернёт список из объектов-реализаций интерфейса ScEdge. Из данных объектов можно получить соседние узлы.

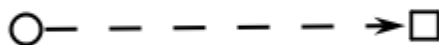


Рис. 6 – Пример искомой конструкции

5.2.11 find(ScPattern3)

Поиск всех конструкций по шаблону на 3 элемента. Данный метод принимает один шаблон и возвращает список найденных конструкций (также на 3 элемента). Для создания шаблона можно воспользоваться как обычным использованием класса ScPattern3Impl (то есть, ручным заполнением правил поиска), так и, что более предпочтительно, с помощью фабрик DefaultScPattern3Factory, ScPattern3FactoryWithNames.

5.2.12 find(ScPattern5)

Поиск всех конструкций по шаблону на 5 элементов. Данный метод принимает один шаблон и возвращает список найденных конструкций (также на 5 элементов). Для создания шаблона можно воспользоваться как обычным использованием класса ScPattern5Impl (то есть, ручным заполнением правил поиска), так и, что более предпочтительно, с помощью фабрик DefaultScPattern5Factory, ScPattern5FactoryWithNames.

5.2.13 find(ScPattern)

Поиск всех конструкций по универсальному шаблону. Данный метод принимает один шаблон и возвращает список найденных конструкций (количество элементов в конструкции будет равно количеству найденных по шаблону элементов). Создать шаблон можно только вручную с помощью одной из имплементаций (на данный момент - DefaultWebsocketScPattern).

5.2.14 setIntegerLinkContent

Задание значения sclink. Данный метод задаёт целочисленные значения в ScIntegerLink в памяти. Возвращает метод логическое значение: true, если операция прошла успешно; false, если закончилось с ошибкой.

5.2.15 setFloatLinkContent

Задание значения `sclink`. Данный метод задаёт дробное значения в `ScFloatLink` в `sc`-памяти. Возвращает метод логическое значение: `true`, если операция прошла успешно; `false`, если закончилось с ошибкой.

5.2.16 setStringLinkContent

Задание значения `sclink`. Данный метод задаёт строковое значения в `ScStringLink` в `sc`-памяти. Возвращает метод логическое значение: `true`, если операция прошла успешно; `false`, если закончилось с ошибкой.

5.2.17 getIntegerLinkContent

Получение значения `sclink`. Данный метод служит для получения актуального целочисленного значения данной `ScIntegerLink`, делая запрос в базу. Однако, значение, хранимое в `sclink` можно также получить вызовом метода `getContent`. В этом случае не гарантируется то, что значение в `sclink` совпадает со значением в `sc`-памяти.

5.2.18 getFloatLinkContent

Получение значения `sclink`. Данный метод служит для получения актуального дробного значения данной `ScFloatLink`, делая запрос в базу. Однако, значение, хранимое в `sclink` можно также получить вызовом метода `getContent`. В этом случае не гарантируется то, что значение в `sclink` совпадает со значением в `sc`-памяти.

5.2.19 getStringLinkContent

Получение значения `sclink`. Данный метод служит для получения актуального целочисленного значения данной `ScStringLink`, делая запрос в базу. Однако, значение, хранимое в `sclink` можно также получить вызовом метода `getContent`. В этом случае не гарантируется то, что значение в `sclink` совпадает со значением в `sc`-памяти.

5.2.20 findKeynode

Поиск ключевых узлов. Данный метод служит для нахождения ключевого узла по заданному идентификатору. Принимает в параметрах строку – идентификатор, а возвращает `Optional`, который либо содержит в себе найденный узел, либо пуст, если переданный идентификатор не существует в базе.

5.2.21 resolveKeynode

Поиск и создание ключевых узлов. Данный метод служит для нахождения ключевого узла по заданному идентификатору. Принимает в параметрах строку и тип узла – идентификатор и тип создаваемого узла соответственно. Если узел с данным идентификатором существует, метод вернёт этот узел. В случае, если не существует, будет создан новый узел с заданными идентификатором и типом узла.

5.3 AsyncUncheckedScContext

Данный тип `Sc`-контекста является, в большей степени, экспериментальным. Он, как и контекст `UncheckedScContext`, скрывает исключения `ScMemory`, позволяя писать код без блоков `try-catch`. Также при вызове методов, будет возвращён не результат выполнения метода, а объект `Future`. То есть, сами вызовы методов не являются блокирующими, что может быть очень полезно в некоторых ситуациях, ведь код `java`-программы продолжает исполнение не дожидаясь ответа, который занимает сотни миллисекунд.

Все доступные методы и их описания совпадают с методы в `UncheckedScContext`.