
# Документация библиотеки JMantic

Версия 0.3.1

13 декабря 2021 г.

# Содержание

# 1 Введение

## 1.1 Описание библиотеки и её предназначения

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

## 1.2 Использованные технологии

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

## 1.3 Технические требования

Исходных 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);
- просты 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 кторых являются перменные, значениями которых являются знаки пар принадлежности;
- метапеременные, значениями которых являются перменные, 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.

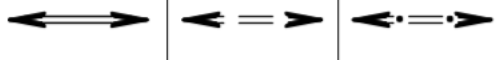
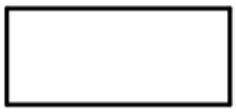
## 2.3 Язык SCg (Semantic Code Graphical) - графическая модификация языка SC

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 перменные - 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 непринадлежности |
|  |  |  | 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 связки, ограничители аргументов функций | | | | |
| =/=> | <=/= | =/=>> | <<=/= | =/=>>> | <<<=/= | связка, соответствующая ориентированной паре неуточняемого вида |
| <=/=> | | <<=/=>> | | <<<=/=>>> | | связка, соответствующая неориентированной паре неуточняемого вида |
| = | | == | | === | | связка равенства значений |
| ≠ | | ≠≠ | | ≠≠≠ | | связка неравенства значений |

| | | | | | | |
|---|---|---|---|---|---|---|
| ≈ | | ≈≈ | | ≈≈≈ | | связка нечеткого равенства значений |
| scs-связки бинарных отношений над множествами: | | | | | | |
| ⩭ | | ⩭ | | ⩭ | | обозначение пары равенства множеств |
| ⩮ | | ⩮ | | ⩮ | | обозначение пары неравенства множеств |
| ⫤ | | ⫤ | | ⫤ | | обозначение пары эквивалентности множеств по набору элементов |
| ⫥ | | ⫥ | | ⫥ | | обозначение пары неэквивалентности множеств по набору элементов |
| ⧈ | | ⧈ | | ⧈ | | обозначение пары пересекающихся множеств |
| ⧅ | | ⧅ | | ⧅ | | обозначение пары непересекающихся множеств |
| ⊂ | ⊃ | ⊂ | ⊃ | ⊂ | ⊃ | связка строгого включения множеств |
| ⊄ | ⊅ | ⊄ | ⊅ | ⊄ | ⊅ | связка строгого невключения множеств |
| ⊆ | ⊇ | ⊆ | ⊇ | ⊆ | ⊇ | связка включения множеств |
| ⊈ | ⊉ | ⊈ | ⊉ | ⊈ | ⊉ | связка невключения множеств |
| scs-связки бинарных отношений над числами: | | | | | | |
| < | > | < | > | < | > | связка строгого сравнения чисел |
| ≤ | ≥ | ≤ | ≥ | ≤ | ≥ | связка нестрогого сравнения чисел |

| | |
|---|---|
| имена унарных функций над числами: | |
| — | обозначение операции арифметического отрицания |
| ***abs*** | обозначение функции взятия абсолютного значения |
| **!** | обозначение функции факториала |
| ***exp*** | обозначение показательной функции |
| ***ln*** | обозначение функции взятия натурального логарифма |
| ***sin*** ***cos*** ***tg*** ***ctg*** | обозначения тригонометрических функций |

| | | | |
|---|---|---|---|
| scs-связки бинарных функций над множествами: | | | |
| ∪ | ∪ | ∪ | связка объединения множеств |
| ⊎ | ⊎ | ⊎ | связка соединения множеств |
| ∩ | ∩ | ∩ | связка пересечения множеств |
| \ | \ | \ | связка разности множеств |
| △ | △ | △ | связка симметрической разности множеств |
| × | × | × | связка декартова произведения |
| scs-связки бинарных функций над числами: | | | |
| + | + | + | связка арифметического сложения |
| • | • | • | связка арифметического умножения |
| – | – | – | связка арифметического вычитания |
| / | / | / | связка арифметического деления |
| ↑ | ↑ | ↑ | связка возведения в степень |
| √ | √ | √ | связка взятия корня |
| имена бинарных функций над числами: | | | |
| ***log*** | 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.

```
concept_device

    <= nrel_inclusion: concept_invention ;

    => nrel_main_idtf:
        [Device]
        (*<-lang_ru;;*);
```

```
        [Device]
        (*<-lang_en;;*);

    => nrel_idtf:
        [device]
        (*<- lang_ru;;*);

    <- rrel_key_sc_element: ...
        (*
            <- definition;;

            => nrel_main_idtf:
                [Definition.(Device)]
                (*<-lang_ru;;*);;

            <= nrel_sc_text_translation: ...
                (*
                    -> rrel_example:
                        "file://html/concept-device.html"
                        (*<-lang_ru;;*);;
                *);;

            <= nrel_using_constants: ...
                (*
                    -> subject;; //ims
                    -> _structure;; //ims
                    -> concept_function;;
                *);;
        *);;
```

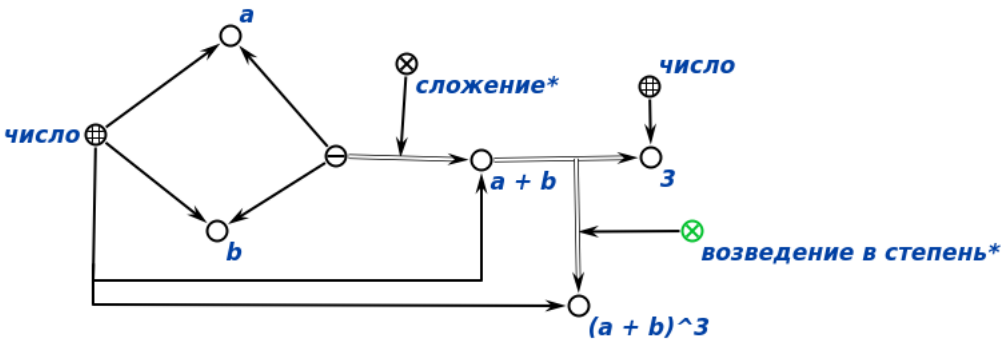
Содержимое файла, соответствующее пути в примере:

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

### 2.6.2 Пример на языке SCg

Для пример 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:

```
{
  "id": 1,
  "type": "request type",
  "payload": {
    ...
  }
}
```

Ответ:

```
{
{
  "id": 1,
  "status": true,
  "event": false,
  "payload": {
    ...
  }
}
}
```

## 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:

```
{
  "id": 2,
  "type": "create_elements",
  "payload": [
    {
      "el": "node",
      "type": 1
    },
    {
      "el": "link",
      "type": 2,
      "content": 45.4
    },
    {
      "el": "edge",
      "src": {
        "type": "ref",
        "value": 0
      },
      "trg": {
        "type": "ref",
        "value": 1
```

```
      },
      "type": 32
    }
  ]
}
```

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

Пример секции с указанием типа:

```
{
  "el": "link",
  "type": 2,
  "content": "data",
  "content_type": "string"
}
```

### 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:

```
{
  "id": 3,
  "type": "check_elements",
  "payload": [
    23123,
    432,
    ...
  ]
}
```

Пример ответа:

```
{
  "id": 3,
  "payload": [
    32,
    0,
    ...
  ]
}
```

### 3.5.3 delete_elements

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

```
{
  "id": 4,
  "type": "delete_elements",
  "payload": [
    23123,
    432,
    ...
  ]
}
```

### 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_requied, присвоив ему значение false.

Пример:

```
{
  "id": 5,
  "type": "search_template",
  "payload": [
    [
      {
        "type": "addr",
        "value": 23123
      },
      {
        "type": "type",
        "value": 32,
        "alias": "_edge1"
      },
      {
        "type": "type",
        "value": 2,
        "alias": "_trg"
      }
    ],
    [
      {
        "type": "addr",
        "value": 231342
      },
      {
        "type": "type",
        "value": 2000,
        "alias": "_edge2"
      },
      {
        "type": "alias",
```

```
        "value": "_edge1"
      },
      {
        "is_required": false
      }
    ],
    ...
  ]
}
```

Ответ для данного запроса будет выглядеть следующим образом:

```
{
  "id": 5,
  "payload": {
    "aliases": {
      "trg": 2,
      "edge1": 1,
      "edge2": 4
    },
    "addrs": [
      [23123, 412, 423, 231342, 282, 412],
      [23123, 6734, 85643, 231342, 4234, 6734],
      [23123, 7256, 252, 0, 0, 0],
      ...
    ]
  }
}
```

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

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

### 3.5.5 generate_template

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

Пример запроса:

```
{
  "id": 7,
  "type": "generate_template",
  "payload": {
    "templ":
    [
      {
        "params": [
          {
            "type": "addr",
```

```
            "value": 23123
          },
          {
            "type": "type",
            "value": 32,
            "alias": "_edge1"
          },
          {
            "type": "type",
            "value": 2,
            "alias": "_trg"
          }
        ]
      },
      {
        "params": [
          {
            "type": "addr",
            "value": 231342
          },
          {
            "type": "type",
            "value": 2000,
            "alias": "_edge2"
          },
          {
            "type": "alias",
            "value": "_edge1"
          }
        ]
      },
      ...
    ],
    "params": {
      "_trg": 564
    }
  }
}
```

Ответ на запрос:

```
{
  "id": 7,
  "payload": {
    "aliases": {
      "_trg": 2,
      "_edge1": 1,
      "_edge2": 4
    },
    "addrs": [23123, 4953, 564, 231342, 533, 4953]
  }
}
```

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

```
{
  "id": 8,
  "type": "generate_template",
  "payload": {
    "templ": "person _-> ..._p (* _=> nrel_email:: _[test@email.com]
        *);;",
    "params": {
      "..._p": 5314
    }
  }
}
```

### 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_ingoing_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_ingoing_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:

```
{
  "id": 9,
  "type": "events",
  "payload": {
    "create": [
      {
        "type": "add_output_edge",
        "addr": 324
      }
    ],
    "delete": [
      2, 4, 5
```

```
    ]
  }
}
```

### 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:

```
{
  "id": 11,
  "type": "keynodes",
  "payload": [
    {
      "command": "find",
      "idtf": "any system identifier that NOT exist"
    },

    {
      "command": "find",
      "idtf": "any system identifier that exist"
    },

    {
      "command": "resolve",
      "idtf": "NOT exist",
      "elType": 1
    },

    {
      "command": "resolve",
      "idtf": "exist",
      "elType": 1
    }
  ]
}
```

Пример ответа:

```
{
  "id": 11,
  "status": true,
  "payload": [
    0,
    321,
    435,
```

```
    324
  ]
}
```

### 3.5.8 content

content — запрос на получение и редактирование содержимого sc-ссылок. 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:

```
{
  "id": 10,
  "type": "content",
  "payload": [
    {
      "command": "set",
      "type": "int",
      "data": 67,
      "addr": 3123
    },
    {
      "command": "get",
      "addr": 232
    },
  ]
}
```

Ответ на предыдущий запрос будет следующим:

```
{
  "id": 10,
  "payload": [
    true,
    {
      "value": 56.7,
      "type": "float"
    },
  ]
}
```

## 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 jmantic 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 |

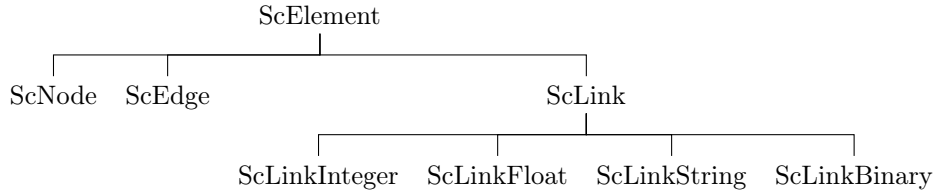
**Рис. 2** – Список всех типов sc-элементов

# 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 и 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, 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. Метода 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 фиксированому узлу.

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

#### 4.2.6 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.

#### 4.2.7 getLinkContent

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

### 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**.
- 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 **SearchByTemplateRequest**.
- content — запрос на получение и редактирование содержимого sc-links. Представлен интерфейсами **SetLinkContentRequest** и **GetLinkContentRequest** .

# 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-а, 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-а, 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-а, 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.

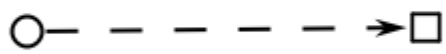


**Рис. 3** – Пример искомой конструкции

### 5.1.11 findAllConstructionsNodeEdgeLink

– 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.

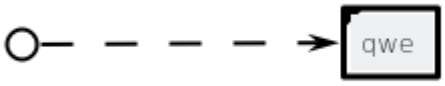



**Рис. 4** – Пример искомой конструкции

### 5.1.12 findAllConstructionsNodeEdgeLinkWithRelation

– 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.

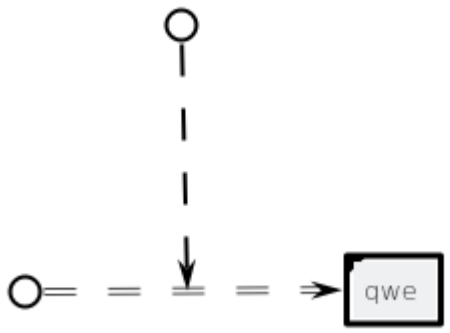



**Рис. 5** – Пример искомой конструкции

### 5.1.13 setIntegerLinkContent

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

### 5.1.14 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.

### 5.1.15 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.

### 5.1.16 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.

### 5.1.17 getFloatLinkContent

– получение значения sclink. Данный метод служит для получения актуального дробного значения данной ScFloatrLink, 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.

#### 5.1.18 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 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-а, 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-а, 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 остис, 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.

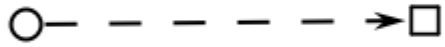


**Рис. 6** – Пример искомой конструкции

### 5.2.11 setIntegerLinkContent

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

### 5.2.12 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.

### 5.2.13 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.

#### 5.2.14 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.

#### 5.2.15 getFloatLinkContent

– получение значения sclink. Данный метод служит для получения актуального дробного значения данной ScFloatrLink, 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.

#### 5.2.16 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.

### 5.3 AsyncUncheckedScContext

Данный тип Sc-контекста является, в больше степени, эксперементальным. Он, как и 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.