



# Документация библиотеки JMantic

Версия 0.2.0

21 ноября 2021 г.

## Содержание

1	Введение	2
2	Sc	3
3	Ostis	4
4	Подробности реализации JMantic	15

# **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 16. 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.

## 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 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 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

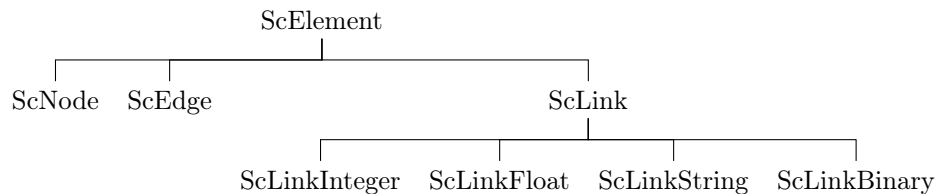
**Рис. 2** – Список всех типов sc-элементов

## 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`, 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 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 элемент списка содержимого. В результате метод вернёт ScLink с изменённым 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**.