Динамические формы: Исчерпывающее руководство для новичков

Кирилл Кривошеев [1]

18 Апреля 2019

[1]hard-won with blood and sweat by the happy team ДРУГ

Оглавление

1 Введение **7**
1.1 Что такое программа 7
1.2 Сценарные языки программирования 8
1.3 Запуск динамических команд 8
1.4 Первая программа 8
1.5 Чувствительность к регистру 9
1.6 Арифметические функции 10
1.7 Упражнения 10

2 Структура программы **11**
2.1 Лексические конструкции языка 11
2.2 Значения и типы 12
2.3 Идентификаторы 12
2.4 Возвращаемые значения 13
2.5 Порядок выполнения 13
2.6 Функция call 15
2.7 Операторы ветвления 15
2.8 Циклы 15
2.9 Упражнения 17

3 Динамическая форма **19**
3.1 Что такое динамическая форма? 19
3.2 Текстовые поля 19
3.3 Списки 19
3.4 Чекбоксы 19
3.5 Виды групп 19
3.6 Копирование групп 20

3.7 Удаление групп 24

4 Среды выполнения программ **25**
4.1 FriendFace 25
4.2 friend-dom-core 25

5 Асинхронные команды **27**
5.1 Автоисполнятор 27
5.2 Сопрограммы 27

6 Паттерны программирования на динамическом языке **29**
6.1 Вложенные команды 29
6.2 Выделение функций 30
6.3 Упражнения 39

A Описание динамических команд **41**
A.1 setValue(value, id) 41
A.2 getUser(attr) 44
A.3 run(args) 45
A.4 if(condition, trueExpr, falseExpr) 45
A.5 isIn(args) 46
A.6 isNotIn(args) 46
A.7 isDigit(arg) 46
A.8 call(expr, id) 47
A.9 IAm(arg) 47
A.10 getTexts(map) 48
A.11 getValues(map) 48
A.12 find(attr, value, group) 49
A.13 plus(args)> 51
A.14 minus(args) -> * 51
A.15 mul(args) -> * 52
A.16 div(args) 52
A.17 indOf(string, substring) 53
A.18 substr(string, start, end) 53
A.19 len(string) 53
A.20 copyGroup(groupId, id, group) 54
A.21 getDistance(args) 54

A.22 execute3DTask(TASK, args, map) . . . 55
A.23 executeTask . . . 55
A.24 execute3DTask . . . 56
A.25 getUsersByOrganizationRequest . . . 56
A.26 setValues(arg0-arg4) . . . 56
A.27 getValue(id) . . . 58
A.28 setMandatory(bool, id) . . . 58
A.29 setVisible(bool, id) . . . 59
A.30 setEnabled(bool, id) . . . 59
A.31 setValid(bool, description, id) . . . 59
A.32 setSelectedValue(val, id) . . . 60
A.33 setWidth(val, id) . . . 61
A.34 getText(map, id) . . . 61
A.35 isMandatory(id) . . . 61
A.36 isVisible(id) . . . 62
A.37 isEnabled(id) . . . 62
A.38 isValid(id) . . . 62
A.39 isFileMandatory(id) . . . 62
A.40 isFileEnabled(id) . . . 63
A.41 clear(id) . . . 63
A.42 setList(_labels, _texts, id) . . . 63
A.43 setMask(value, id) . . . 64
A.44 setFocus(val, id) . . . 64
A.45 setMax(value, id) . . . 64
A.46 setMin(value, id) . . . 64
A.47 string(value) . . . 65
A.48 setBeforeSave(value, id) . . . 65
A.49 getApprovalStage(id) . . . 65
A.50 setCheckSign(bool, id) . . . 65
A.51 setFileMandatory(bool, id) . . . 66
A.52 openAlert(type, message) . . . 66
A.53 setSaveEnabled(bool, id) . . . 66
A.54 setFileEnabled(bool, id) . . . 67
A.55 setExternalSystem(extsystem, id) . . . 67
A.56 setInformationVisible(bool, id) . . . 67
A.57 setInitiator(value, id) . . . 68

A.58 setMultiFlow(bool, id) . . . 68
A.59 setRedirect(value, id, skipConvert) . . . 68
A.60 setRelationship(objectId, objectType, id) . . . 69
A.61 setAdditionalTemplate . . . 69
A.62 setScoringVisible(bool, id) . . . 69
A.63 getSysDate(value) . . . 70
A.64 getInitiator(value) . . . 70
A.65 setMinHours(value, id) . . . 70
A.66 setMaxHours(value, id) . . . 71
A.67 setContainCustomObjects(bool, id) . . . 71
A.68 getSize(param) . . . 71
A.69 setStep(value, id) . . . 72
A.70 setStyle(arg, value, id) . . . 72
A.71 compare(left, operand, rigth) . . . 72
A.72 setInitiatorVisible(bool, id) . . . 73
A.73 setDescription(text, id) . . . 73
A.74 getRelatedId() . . . 73
A.75 setCopyVisible(bool, id) . . . 73
A.76 setDeclineVisible(bool, id) . . . 74
A.77 setCloseVisible(bool, id) . . . 74
A.78 getClientType() . . . 74
A.79 isMobileClient() . . . 74
A.80 isWebClient() . . . 74
A.81 isIOSClient() . . . 75
A.82 isAndroidClient() . . . 75
A.83 isWebSigmaClient() . . . 75
A.84 isWebAlphaClient() . . . 75
A.85 isFace20() . . . 75
A.86 setAction(value, id) . . . 75
A.87 setMode(value, id) . . . 76
A.88 dateDiff(start, end, id) . . . 76
A.89 isCreateMode() . . . 76
A.90 isOpenMode() . . . 76
A.91 setSaveVisible(bool, id) . . . 76
A.92 _setValue(value, id, revertValues) . . . 77
A.93 _runSBTASK(el) . . . 77

A.94 _getValueByUser(user, attr, isUser) 77
A.95 createDocument 78
A.96 signAttachment(attachment,mode) 79

Глава 1

Введение

1.1 Что такое программа

Программа – комбинация компьютерных инструкций и данных, позволяющая аппаратному обеспечению вычислительной системы выполнять вычисления или функции управления.

Как правило, компьютерная программа написана программистом на языке программирования. Из программы, написанной на понятном человеку исходном коде, комьютер может сформировать машинный код – набор инструкций, который компьютер может исполнять напрямую. В качестве альтернативы, программа может быть исполнена при помощи интерпретатора.

Несмотря на различия между множеством существующих языков программирования, некоторые базовые инструкции пристуствуют почти в каждом из них:

ввод: получить данные с клавиатуры, из файла или другого устройства

вывод: отобразить данные на экране или отправить на другое устройство

арифметика: произвести базовые математические операции, такие как сложение или умножение

ветвление: проверить определенное состояние и выполнить соответствующий ему код

повторение: произвести определенное действие неоднократно, иногда с некоторыми вариациями

В динамическом языке программой принято называть последовательность команд, записанную в формате функций с аргументами. Каждый аргумент указывается в квадратных скобочках. Аргументы между собой разделяются

запятыми, например:

```
скажи([привет],[как],[дела])
```

1.2 Сценарные языки программирования

Сценарный язык (язык сценариев, жарг. скриптовый язык; англ. scripting language) – высокоуровневый язык программирования, автоматизирующий выполнение задач в специализированной среде выполнения. Как правило, сценарные языки являются интерпретируемыми, а не компилируемыми. Сценарный язык можно рассматривать как предметно-ориентированный язык (англ. domain-specific language, DSL — «язык, специфический для предметной области») для определенного окружения.

Язык программирования динамических форм (далее **Динамический язык**) – сценарный предметно-ориентированный язык, предназначнный для автоматизации управления состоянием интерактивных форм регистрации обращений в экосистеме ДРУГа.

1.3 Запуск динамических команд

Одним из первых вызовов, который бросают динамические формы начинающему разработчику, является необходимость настройки рабочего окружения HP Service Manager для редактирования шаблонов. Однако, благодаря мощным возможностям динамического языка, сделать первые шаги в разработке можно гораздо проще: достаточно открыть шаблон обращения «Исполнятор» в dev-среде портала «Лицо ДРУГа». Это простая динамическая форма, состящая всего из двух полей: «Команда» и «Результат».

1.4 Первая программа

Talk is cheap. Show me the code

Linus Torvalds

По давней традиции, первая программа, которую начинающий разработчик пишет на новом языке, является «Hello, world!». На динамическом языке она выглядит так:

```
>>> setValue([Hello, world!],[result])
```

Для выполнения программы введите ее в поле «Команда» и переставьте курсор в поле «Результат». В итоге, в поле «Результат» вы увидите текст:

```
Hello, World!
```

Поздравляем! Вы сделали первый шаг на безумно интересном пути к освоению могучих возможностей создания интерактивных форм, который дает вам Динамический язык. Это пример команды **setValue**, которая устанавливает значение, указанное первым аргументом в квадратных скобках в поле, указанное во втором аргументе.

Здесь и далее, все команды, которые вводятся в поле «Команда» будут предваряться символами

```
>>>
```

1.5 Чувствительность к регистру

Динамческий язык нечувствителен к регистру. Это означает, что программы:

```
>>> setvalue([Hello, world!],[result])
```

и

```
>>> SETVALUE([Hello, world!],[result])
```

и даже

```
>>> sEtVaLuE([Hello, world!],[result])
```

С точки зрения интерпретатора динамического языка являются индентичными и произведут одинаковый результат. И несмотря на то, что последний пример выглядит, безусловно, наиболее круто, в примерах к книге мы будем придерживаться так называемого CamelCase, так как он наиболее удобочитаем:

```
>>> setValue([Hello, world!],[result])
```

1.6 Арифметические функции

Перейдем от «Hello, world!» к арифметике. Как и любой, уважающий себя язык программирования, динамический язык предоставляет возможность работы с основными арифметическими операциями, а именно **plus**, **minus** и **mul** для сложения, вычитания и умножения соотвественно, а также **div** для деления:

```
>>> setValue([plus([68],[1])],[result])
69
>>> setValue([minus([69],[1])],[result])
68
>>> setValue([mul([6],[7])],[result])
42
>>> setValue([div([84],[2])],[result])
42
```

На самом деле, указанные функции обладают гораздо большим набором возможностей, но для первого знакомства этого будет достаточно.

1.7 Упражнения

Откройте шаблон «Исполнятор» и, при помощи динамического языка, решите следующие задачи:

1. Сколько будет в градусах Цельсия температура в 69 Фаренгейт?

2. Сколько спринтов потребуется для вывода в промышленную эксплуатацию шаблона «Совместная поездка» если каждый спринт исправляется 4 багов, и обнаруживается 6 новых?

Глава 2

Структура программы

2.1 Лексические конструкции языка

Инструкции динамического языка представляют собой комбинации из конструкций всего двух типов: вызовов функций и литералов.

Основной конструкцией языка является *функция*. Она состоит из имени и параметров. Сразу после имени функции следует символ '`(`', за которым могут следовать один или несколько параметров, после этого вся конструкция завершается '`)`', Некоторые функции вовсе не принимают параметры, см. листинг 2.1(1). Также поддерживается несколько параметров которые можно разделять символом '`,`' пример 2.1(2,3), но не все среды выполнения (см. главу 4) поддерживают символ переноса строки, используя вместо него последовательность `\r\n`. Параметры функции можно переносить на следующую строку и добавлять пробелы для лучшей читаемости программы, 2.1(3)

Скобки после имени функции являются обязательными даже если у функции нет праметров. Каждый параметр функции необходимо обернуть в '`[`' '`]`'. Параметром функции может быть литеральное значение или вызов другой функции, вне зависмости от этого параметр необходмо обернуть в '`[`' '`]`'.

Функции строго предопределены в стандартной библиотеке функций (см. главу **??**).

Литералы представляют собой константы, включаемые непосредственно в текст программы. В отличие от прочих элементов языка (констант, переменных), литералы не могут быть изменены в тексте программы.

```
(1) func()

(2) func([arg0], [arg1], [argN])

(3) func(
        [arg0], [arg1]
        ,[arg2]
    )
```

2.2 Значения и типы

Динамический язык – динамически типизирован (неожиданно, не так ли?). Динамическая типизация – приём, широко используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной.

Правила, по которым происходит определение типа аргумента, следующие:

значение считается **числом**, если его строковое представление содержит только цифры и знак разделителя `,` ..

значение считается **логическим**, если строковое представление представляет собой последовательности `true` или `false`, вне зависимости от регистра символов.

В остальных случаях значением считается строкой символов.

2.3 Идентификаторы

В парадигме функцильнальных и логических языков программирования, понятие переменной определяется как имя, с которым может быть связано значение, или даже как место для хранения значения. В динамическом языке переменной можно считать строковый литерал, обозначающий идентификатор элемента динамической формы (см. главу 3).

2.4 Возвращаемые значения

Все функции стандартной библиотеки динамического языка возвращают значение. Например,

`getValue([foo])` – вернет значение элемента, определяемого по идентификатору «foo» формы.

Функции, созданные для выполнения побочных эффектов, возвращают значение, как правило – «id» элемента формы, над которым производится операция:

`setValue([foo], [bar])` – вернет «bar»

Даже функции, для которых в стандартной библиотеке возвращаемое значение не определено – вернут пустой литерал: `[]`

2.5 Порядок выполнения

В основу выполнения команд динамического языка лежит простая концепция – выполнение функций должно происходить по слоям из самого глубокого вложенной функции к корневому элементу, при этом функция не может быть выполнена, пока не выполнены все ее дочерние элементы.

Итак поэтапно разберем последовательность действий происходящих при парсинге команд:

1. Исходный строковый вид команды передается на вход анализатору, который содержит в себе регулярные выражения для парсинга

2. Происходит валидация кода(в первых версиях реализации валидации не было), в случае ошибки валидации – ошибки в количестве параметров функции, скобочках или невозможных символах - команда не будет выполнена и в лог действий будет выведена ошибка валидации

3. Если команда успешно провалидирована, начнется построение динамического дерева(Динамическое дерево). Важно отметить, что в ходе выполнения команды, дерево может менять свою структуру(команда if), а иногда даже создавать копии самого себя(CALL).

4. Для построения динамического дерева(не бинарное дерево Бинарное дерево) используется стандартный левосторонний обход, когда каждая

следующая функция последовательно добавляется и спускается вглубь к следующему дочернему узлу, обойдя все элементы у первого дочернего узла, переходит ко второму, третьему и т.д.

5. ПРИМЕЧАНИЕ. Особенными для разбора считаются функция IF, CALL и STRING в частности CALL и STRING останавливают дальнейший парсинг дочерних узлов функции и выставляют в результат строковое представление аргументов функции. Условная функция(оператор) IF – разбирает изначально только первый дочерний узел, оставшиеся два так же сохраняет в строковых представлениях

6. После разбора команды в динамическое дерево можно запускать ее на исполнение

7. В изначальной реализации динамического языка - динамическое дерево полностью преобразовывалось в очередь и выполняло все команды последовательно по кругу пока все функции не будут выполнены, в более современной реализации очередь содержит только активные элементы выполняющиеся непосредственно в текущей итерации и последовательно добавляет родительские узлы непосредственно в тот момент, когда имеется возможность их выполнить, напомним, что родительский узел может быть выполнен только в случае когда все его дочерние команды вернули результат

8. Выполнение начинается с самого глубокого уровня, если представить исходное динамическое дерево, то визуально его необходимо перевернуть и по слоям начать выполнять узлы дерева от листьев к корню, каждый следующий слой ожидает выполнения дочерних узлов предыдущего, в случае если функция является асинхронной, она не блокирует выполнение остальных команд на уровне

9. ОСОБЕННОСТИ. Функции CALL и IF являются функциями модифицирующими динамическое дерево. Так функция CALL создает локальную копию поддерева их своих аргументов при этому есть два варианта(CALL)

10. Функция IF на основе результата первого дочернего узла добавляет в исходное динамическое дерево либо второй, либо третий аргумент функ-

ции(третий может отсутствовать), происходит парсинг необходимой команды и динамическое дерево расширяется узлами дочерней функции внутри IF, параллельно выполняющейся в очереди начиная с уровня на котором размещен узел с IF(подробнее про IF)

11. В случае возникновения ошибок, возможны ситуации, которые прерывают дальнейшее выполнение команды.

12. Результат корневого узла дерева обычно не используется и ничего не возвращает, за исключением использование команды CALL.

Понимая концепции порядка выполнения команд, а также особенности запуска циклов обработки команд можно автоматизировать сколько угодно сложные наборы операций над динамическими формами.

2.6 Функция call

Если результат выполнения команды внутри CALL является динамической функцией создается дерево(Порядок выполнения) если результат не является динамической функцией - исходная команда внутри CALL. Выполняется над другим элементом динамической формы указанным во втором параметре функции CALL

2.7 Операторы ветвления

2.8 Циклы

Вооружившись ранее изученным набором команд, вам не составит труда реализовать цикл на динамическом языке.

Напишем программу, которая выведет в поле «Результат» 100 точек:

Добавьте на динамическую форму скрытое поле с 'id' : 'counter'. Это поле будет служить счетчиком цикла.

Ниже приведен полный листинг программы:

```
if(
    [isnotin([getValue([counter])],[100])],
```

```
    [run(
        [setValue(
            [plus(
                [getValue([result])],
                [.]
            )],
            [result]
        )],
        [setValue(
            [plus(
                [getValue([counter])],
                [1]
            )],
            [counter]
        )],
        [call(
            [getValue([cmd])],
            [this]
        )]
    )]
)]
```

Запустив программу, можно убедиться, что в поле «Результат» вывелось ровно 100 точек.

Теперь рассмотрим принцип работы программы подробнее:

```
if(
    [isnotin([getValue([counter])],[100])],
```

В данном фрагменте кода мы проверяем, не выполняется ли условие неравенства нашего счетчика значению 100.

В этом случае выполняем тело цикла, состоящее из 3-х команд внутри блока «run»:

```
setValue([plus([getValue([result])],[.])],[result])
```

Здесь происходит добавление новой точки в поле с результатом.

```
setValue([plus([getValue([counter])],[1])],[counter])
```

Увеличиваем значение счетчика на 1.

```
call([getValue([cmd])],[this])
```

И, главная часть, обеспечивающая зацикливание нашей программы - вызываем повторное выполнение скрипта при помощи функции «Call»

2.9 Упражнения

Откройте шаблон «Исполнятор» и решите следующие задачи:

1. Напишите программу, которая выводит на экран числа от 1 до 100. При этом вместо чисел, кратных трем, программа должна выводить слово «Fizz», а вместо чисел, кратных пяти — слово «Buzz». Если число кратно и 3, и 5, то программа должна выводить слово «FizzBuzz»

Глава 3

Динамическая форма

3.1 Что такое динамическая форма?

3.2 Текстовые поля

3.3 Списки

3.4 Чекбоксы

Если заглянуть в истоки зарождения динамических форм, изначально не было реализовано поддержки элементов типа группа и поэтому было принято решение о том чтобы постараться с эмулировать понятие группы так чтобы элементы группировались по параметру **groupid**, так появились плоские группы и legacy группы(в рамках реализации domcore названные - нормализованные группы), позже механизм групп был добавлен в интерфейсы ECO, но для полноценного функционирования механизма групп в старых шаблонах необходимо уметь работать со всеми видами групп. Остановимся на типах групп подробнее в следующем разделе.

3.5 Виды групп

На момент написания книги можно выделить три основные типа групп:

1. Плоские группы. Плоские группы представляют собой элементы груп-

пируемые по параметру **groupid**, но при этом такие группы не имею заголовка как такового, при отображении не визуализируются как группа, основной задачей, которой решают данные группы - сортировка и как ни странно группировка элементов в одном блоке. Важно заметить при сохранении группы такого формата должны быть преобразованы обратно в список без группы.

2. Legacy группы. Основное отличие от плоских групп в наличие заголовка. Есть отличительная особенность поддержки Legacy групп Лице Друга 1.0 и domcore библиотеке: В первом случае поддерживается наличие более одного заголовка в группе, во втором же принято решение что заголовок группы может быть один. Все остальные заголовки не обрезаются при обработке динамической формы. При визуализации Legacy Group могут сворачиваться и управляться командой динамического языка **setFocus**. При сохранении они могут не разворачиваться и отправляться как нормальные группы, за исключением шаблонов помеченных атрибутом needunwrap, требующего развернуть

3. Группы. Стандартный механизм групп приходящий с заголовком внутренними элементами. Работа происходит в точности как с Legacy группами, за исключением, что их никогда не нужно разворачивать

Каждая группа имеет несколько механизмов - копирование, удаление и сворачивание. Если сворачивание группы это в большей степени чисто визуальный функционал, за исключением момента с использование функции **setFocus** над заголовком группы. То копирование и удаление групп это отдельный функционал о котором мы поговорим в следующих разделах.

3.6 Копирование групп

Копирование групп - стандартный функционал поддерживаемый динамическим языком и позволяющий гибко настраивать механизм копирования. Копирование группы вызывается командой динамического языка **copyGroup**(Подробнее о ней можно прочитать в разделе Команды динамического языка). Стандартно принято прописывать копирование группы либо в sbcommand(раздел Динамические формы), либо при нажатии на кнопку addButton вложенную в группу. Функционал копирования групп привязан к исходному состоянию

объекта в динамической форме, а не к его состоянию в момент копирования. Чтобы понять давайте посмотрим на примере: У нас из формы пришла группа:


```
{
    "objectType": "group",
    "logicalType": "group",
    "id": "groupaddress",
    "label": "Пункт отправления/прибытия",
    "mandatory": false,
    "style": "group",
    "visible": true,
    "width": "",
    "sbobject": null,
    "sbdbfield": "groupAddressTo",
    "sbonload": "[run(\n\t[if([iam([approver])],\n\t\t[if([isin([getApproval$
    "sbcopyinfo": false,
    "childs": [
        {
        "objectType": "select",
        "logicalType": "combo",
        "id": "addressto",
        "label": "Маршрут следования",
        "mandatory": true,
        "sbcommand": "",
        "sbmask": "",
        "sbmodify": true,
        "sbtask": "if(\r\n\t[run([setValue([getDistance([ getValues([ find([
        "sbtitle": "",
        "sbtype": "address",
        "sbstyle": "width:65%",
        "style": "combo",
        "visible": true,
        "width": "",
        "sbobject": null,
        "sbdbfield": "addressTo",
        "groupid": ""
```

```
},
{
"objectType": "text",
"logicalType": "date",
"id": "wait",
"label": "Время ожидания",
"mandatory": false,
"sbcommand": "run([setMinHours([0],[setMaxHours([2],[this])])], [if
"sbmask": "HH:mm",
"sbmodify": true,
"sbtask": "if(\n    [run([setValue([getDistance([getValues([find([
"sbtitle": "",
"sbtype": "",
"sbstyle": "width:35%",
"style": "text",
"visible": false,
"width": "",
"sbcopyinfo": false,
"childs": null,
"groupid": "distanation"
},
{
"objectType": "checkbox",
"logicalType": "buttonAdd",
"id": "addgroupaddress",
"label": "Добавить еще один адрес",
"mandatory": false,
"sbcommand": "",
"sbmodify": true,
"sbtask": "copyGroup([groupaddress],[this])",
"sbtitle": "",
"sbtype": "buttonadd",
"sbstyle": "",
"style": "checkbox",
"visible": true,
"width": "",
```

```
        "sbdbfield": "addGroupAddress",
        "text": "if([run([setValue([getDistance([getValues([find([id],[addres
        "groupid": "distanation"
        }
    ]
}
```

Мы изменяем заполняем значение Маршрут следования и время ожидания, и при нажатии на кнопку "Добавить еще один адрес» выполняется команда sbtask в которой вызывается копировани группы и происходит магия - копируется группа, но все заполненные нами поля не учитываются и поля Маршрут следования и время ожидания в новой созданной группе будут пустыми. Точно так же происходит и с новыми элементами, если в группе динамическим языком добавятся новые элементы, то при копировании они не будут учтены и новая группа создастся без них. Итого - при копировании группы не учитываются никакие изменения произведенные с элементами группы и самой группой. В том числе не учитываются изменения видимости, обязательности, возможности изменений и т.д.

Еще одной важной особенностью является визуальное отображения кнопок копирования. Функционально не запрещено, но визуально скрываются все кнопки для копирования групп кроме последней в наборе. То есть нажать на кнопку и добавить можно только в конец группы, но это только визуальное ограничение сделанное для более эстетичного вида. Исключительной особенностью обладают группы элементы которых помещаются в одну строчку. Визуальное отображение элементов разделено от функционального. Если элементы группы для всех скопированных групп из одного множества удовлетворяют условию что все элементы по ширине могу уместиться в одной строке(кнопка для копирования группы не учитывается при вычислениях), то визуально все группы склеиваются в одну и отображаются как общая группа с одним заголовком, при этом функциональная часть остается неизменной, там все также это представляется как разные группы. Если копирование работает над Плоской группой - проверять условия описанные выше не имеет смысла, так как плоские группы не имеют заголовка, поэтому проверка производится только для групп типа Lygacy и стандартных групп

3.7 Удаление групп

Удаление групп - операция позволяющая удалить любую в том числе и первую группу в подмножетсве скопированных групп. Эта функция одна из немногих, которые не реализованы в динамическом языке, а разрешаются с использованием вызовов обработчиков через UI. Удаление групп возможно только в том случае если групп в подмножестве больше 1-ой(2, 3 и т.д.). При добавлении группы через копирование у всех групп появляется возможность закрыть(удалить группу) визуально отображается в виде крестика над группой или элементом если группы склеились в одну. При нажатии на крестик происходит поиск группы и удаление ее, а дополнительно перед удалением у той группы, которую удаляют находят элемент **buttonAdd** и вызывают команду выполнение команды динамического языка установленную в параметр **text** этого поля. Что позволяет например вести пересчеты расстояний и стоимости поездки при удалении адреса из маршрута.

Глава 4

Среды выполнения программ

4.1 FriendFace

4.2 friend-dom-core

Глава 5

Асинхронные команды

5.1 Автоисполнятор

Автоисполнятор или АИ – представляет собой Task Manager или систему асинхронного выполнения задач. Задачи представляют собой скрипты написанные на языке groovy. Groovy – объектно-ориентированный язык программирования разработанный для платформы Java как альтернатива языку Java с возможностями Python, Ruby и Smalltalk. Каждый скрипт идентифицируется по уникальному имени (например: `FRIEND_GET_LIMIT_BY_USER_TPL_DOUBLE`. Кроме имени скрипт на вход принимает дополнительные параметры. Существует два типа скриптов - стандартные и скрипты с обработкой динамической формы Стандартные скрипты на вход принимают только данные заданные непосредственно в функции динамического языка. Функции с обработкой формы дополнительно преобразовывают текущее состояние формы в JSON объект и передает ее в тело функции. Для вызова АИ в динамическом языке используют функции: **functionName** и **_functionName**, где **functionName** является любым именем функции, которая не входим в список динамических функций(см. прил. А) Функции без нижнего подчеркивания делают запрос к стандартным скриптам, с нижнем подчеркиванием – к скриптам с обработкой формы. Количество аргументов таких функций не ограничено. Первым аргументом передается название скрипта, а последним – элемент над которым производится выполнение.

5.2 Сопрограммы

Глава 6

Паттерны программирования на динамическом языке

6.1 Вложенные команды

Расмотрим классическую задачу, встречающуюся в большинстве шаблонов динамических форм. Зачастую требуется установить ряд свойств динамического элемента, например, "выключить"и "скрыть":

```
run(
    [setVisible([false],[this])],
    [setMandatory([false],[this])],
    [setEnabled([false],[this])]
)
```

Указанный фрагмент кода прекрасно справляется с поставленной задачей. Однако, скрипт можно заметно облегчить, применив технику «Вложенных команд»:

Обратим внимание, что каждая динамическая команда в данном примере возвращает значение – id динамического элемента. Таким образом, мы можем передавать команды в качестве аргумента с ожидаемым значением id в другие команды. Применив данный подход дважды, сократим код примера до следующей записи:

```
setEnabled([false],[setMandatory([false],[setVisible([false],[this])])])
```

Мы также избавились от необходимости использования динамической команды «run»!

В общем случае, алгоритм рефакторинга по технике «Вложенных команд» выглядит так:

1. В списке аргументов функции «run» убедитесь, что возвращаемые значения всех команд семантически совпадают с входными параметрами других команд. В большинстве случаев - это id элементов на форме.

2. Возьмите последнюю команду из списка и замените подходящий литерал аргумента предшествующей команды ее вызовом.

3. Удалите последнюю команду из списка.

4. Если в списке аргументов функции «run» осталась одна команда – замените блок «run» этой команды, иначе повторяйте шаги 2 – 3.

Внимание! Избегайте злоупотреблений рассмотренным приемом – избыточная вложенность усложняет читаемость и поддерживаемость кода: удалить или добавить команду в середину цепочки вызова становится затруднительно. Рекомендуемый уровень вложенности – не более 3 – 4.

6.2 Выделение функций

Рассмотрим задачу формирования списка месяцев в зависимости от текущей даты: в элемент типа select требуется вывести название текущего и двух последующих месяцев. Вот как эта задача решается в одном из шаблонов обращений:

```
if(
    [isin(
        [substr([getsysdate([+0])],[3],[5])],
        [01]
    )],
    [setSelectedValue(
```

```
        [null],
        [setValue(
            [Март],
            [setValue(
                [Февраль],
                [setValue(
                    [Январь],
                    [this]
                )]
            )]
        )]
    )],
    [if(
    [isin(
        [substr([getsysdate([+0])],[3],[5])],
        [02]
    )],
    [setSelectedValue(
        [null],
        [setValue(
            [Апрель],
            [setValue(
                [Март],
                [setValue(
                    [Февраль],
                    [this]
                )]
            )]
        )]
    )],
    [if(
        [isin(
            [substr([getsysdate([+0])],[3],[5])],
            [03]
        )],
        [setSelectedValue(
```

```
            [null],
            [setValue(
                [Май],
                [setValue(
                    [Апрель],
                    [setValue(
                        [Март],
                        [this]
                    )]
                )]
            )]
        )],
    [if(
        [isin(
            [substr([getsysdate([+0])],[3],[5])],
            [04]
        )],
        [setSelectedValue(
            [null],
            [setValue(
                [Июнь],
                [setValue(
                    [Май],
                    [setValue(
                        [Апрель],
                        [this]
                    )]
                )]
            )]
        )],
    [if(
        [isin(
            [substr([getsysdate([+0])],[3],[5])],
            [05]
        )],
        [setSelectedValue(
```

```
            [null],
            [setValue(
                [Июль],
                [setValue(
                    [Июнь],
                    [setValue(
                        [Май],
                        [this]
                    )]
                )]
            )]
        )],
    [if(
        [isin(
            [substr([getsysdate([+0])],[3],[5])],
            [06]
        )],
        [setSelectedValue(
            [null],
            [setValue(
                [Август],
                [setValue(
                    [Июль],
                    [setValue(
                        [Июнь],
                        [this]
                    )]
                )]
            )]
        )],
    [if(
        [isin(
            [substr([getsysdate([+0])],[3],[5])],
            [07]
        )],
        [setSelectedValue(
```

```
            [null],
            [setValue(
                [Сентябрь],
                [setValue(
                    [Август],
                    [setValue(
                        [Июль],
                        [this]
                    )]
                )]
            )]
        )],
    [if(
        [isin(
            [substr([getsysdate([+0])],[3],[5])],
            [08]
        )],
        [setSelectedValue(
            [null],
            [setValue(
                [Октябрь],
                [setValue(
                    [Сентябрь],
                    [setValue(
                        [Август],
                        [this]
                    )]
                )]
            )]
        )],
    [if(
        [isin(
            [substr([getsysdate([+0])],[3],[5])],
            [09]
        )],
        [setSelectedValue(
```

```
            [null],
            [setValue(
                [Ноябрь],
                [setValue(
                    [Октябрь],
                    [setValue(
                        [Сентябрь],
                        [this]
                    )]
                )]
            )]
        )],
    [if(
        [isin(
            [substr([getsysdate([+0])],[3],[5])],
            [10]
        )],
        [setSelectedValue(
            [null],
            [setValue(
                [Декабрь],
                [setValue(
                    [Ноябрь],
                    [setValue(
                        [Октябрь],
                        [this]
                    )]
                )]
            )]
        )],
    [if(
        [isin(
            [substr([getsysdate([+0])],[3],[5])],
            [11]
        )],
        [setSelectedValue(
```

```
            [null],
            [setValue(
                [Январь],
                [setValue(
                    [Декабрь],
                    [setValue(
                        [Ноябрь],
                        [this]
                    )]
                )]
            )]
        )],
    [if(
        [isin(
            [substr([getsysdate([+0])],[3],[5])],
            [12]
        )],
        [setSelectedValue(
            [null],
            [setValue(
                [Февраль],
                [setValue(
                    [Январь],
                    [setValue(
                        [Декабрь],
                        [this]
                    )]
                )]
            )]
        )],
    []
)])])])])])])])])])])])])])])])])
```

Как видим, данный пример содержит много повторяющихся элементов участков кода - по одному фрагменту на каждый из 12 месяцев. Хорошо, что в нашем календаре их не 42! Представьте объем доработок при изменении требований к задаче: например, заказчику потребуется выводить по 6 названий

месяцев!

Приступим же к рефакторингу!

Первым делом, обратим внимание на повторяющийся фрагмент кода:

```
[substr([getsysdate([+0])],[3],[5])]
```

Это скрипт получения значения номера текущего месяца. Вычислим его один раз и запомним в отдельном скрытом поле формы:

```
<text id="month_num" visible="false"
        sbcommand="setValue([substr([getsysdate([+0])],[3],[5])],[this])]"
 />
```

Далее, нам потребуется структура для выставляения соответствия номерам месяцев их названий. С этой задачей отлично справляется элемент «select» :

```
<select id="months" visible="false">

<option label="Январь">1</option>
<option label="Февраль">2</option>
<option label="Март">3</option>
<option label="Апрель">4</option>
<option label="Май">5</option>
<option label="Июнь">6</option>
<option label="Июль">7</option>
<option label="Август">8</option>
<option label="Сентябрь">9</option>
<option label="Октябрь">10</option>
<option label="Ноябрь">11</option>
<option label="Декабрь">12</option>
<option label="Январь">13</option>
<option label="Февраль">14</option>

</select>
```

Обратите внимание, что мы указали 2 «лишних» месяца – это потребуется для корректной работы нашего алгоритма в ноябре и декабре.

Добавим на динамическую форму select для вывода результата:

```
<select id="result" visible="true">
```

Реализуем функцию добавления в наш новый контрол очередного месяца:

```
<text id="add_month">
setValue(
    [getValue(
        [setSelectedValue(
            [plus(
                [getValue([month_num])],
                [getValue([i])]
            )],
            [months]
        )],
        [result]
    )],
    [result]
)
</text>
```

Где «i» - простое скрытое поле для хранения счетчика:

```
<text id="month_num" visible="false">0</text>
```

Для добавления нового месяца в итоговый select достаточно выполнить вызвать нашу функцию «add_month» через оператор call:

```
call([getValue([add_month])],[this])
```

Теперь, для решения нашей задачи потребуется выполнить скрипт «add_month» трижды, меняя значение «i»:

```
run(
    [call([getValue([add_month])],[this])],
    [setValue([1],[i])],
    [call([getValue([add_month])],[this])],
    [setValue([2],[i])],
    [call([getValue([add_month])],[this])],
    [setSelectedValue([null],[result])]
)
```

6.3 Упражнения

1. Используя код разобранного выше примера, решите задачу: в элемент типа select требуется вывести название текущего и 5 последующих нечетных месяцев. Для решения задачи рекомендуется использовать подход к реализации циклов, рассмотренный в разд. 2.8

Приложение А

Описание динамических команд

A.1 setValue(value, id)

Устанавливает значение первого параметра в элемент, указанный во втором параметре.

1. Если элемент не select и значение первого параметра строка, то в text запишется строкоговое значение
2. Если элемент не select и значение первого параметра Map, то в text запишется первое значение из Map
3. Если элемент не select и значение первого параметра пустая Map-а, то в text запишется пустая строка
4. Если элемент select и первый параметр Map, то options будут записаны значения из Map
5. Если элемент select и первый параметр пустая Map, то будут удалены все options и в text будет устанвлена пустая строка.
6. Если элемент select и значение первого параметра строка, то будет добавлен один option с label и text равный значение второго параметра.

Возвращает: string - Значение второго параметра (идентификатор элемента)

Параметр	Тип	Описание
value	string | DLMap	Новое значение элемента
id	string	Идентификатор элемента

Пример

Элемент:

```
<form>
<text id="1" label="Наименование товара" style="text" type="2">
Ручка шариковая для сотрудника
</text>
</form>
```

1. setValue([Значение 1],[this]) -
установит в текущий элемент значение test

Результат:

```
<form>
  <text id="1"
 label="Наименование товара"
 style="text" type="2">
 Значение 1
   </text>
</form>
```

2. setValue([DLMap{"1": "Значение 1", "2": "Значение 2"}], [this])

Результат:

```
<form>
  <text id="1" _parentId="0"
label="Наименование товара" style="text" type="2">
Значение 1
  </text>
```

```
</form>
```

3. setValue([DLMap{}], [this])

Результат:

```
<form>
<text id="1"  label="Наименование товара" style="text" type="2"/>
</form>
```

Элемент:

```
<form>
<select id="f1"> Выбранное значение
  <option id="f1_0"
label="Отсутствует оплата за замещение должностей">
Выбранное значение</option>
  </select>
</form>
```

4. setValue([DLMap{"1": "Значение 1", "2": "Значение 2"}], [this])

Результат:

```
<form>
<select id="f1"> Выбранное значение
<option id="1" label="Значение 1">1</option>
<option id="2" label="Значение 2">2</option>
</select>
</form>
```

5. setValue([DLMap{}], [this])

Результат:

```
<form>
```

```
<select id="f1">
</select>
</form>

6.setValue([Значение 1],[this])

Результат:

<form>
<select id="f1"> Выбранное значение
<option id="Значение 1" label="Значение 1">Значение 1</option>
</select>
</form>
```

A.2 getUser(attr)

Функция возвращает информацию о пользователе

Возвращает: string | boolean - Информация о пользователе

Параметр	Тип	Описание
attr	string	Значение агрумента

```
Допустимые значения агрумента:

getuser(fio)
getuser(firstName)
getuser(secondName)
getuser(middleName)
getuser(tabNum)
getuser(departName)
getuser(position) - Должность
getuser(vsp)
getuser(address)
getuser(phone)
getuser(phoneInner)
getuser(phoneMobile)
```

```
getuser(identificator)
getuser(id)
getuser(kadrCode)
getuser(category)
getuser(terbank)
getuser(findByTabno)
getuser(territoryCode)
```

A.3 run(args)

Выполняет вложенные выражения. Порядок выполнения выражений не определен. Функция доступна только в выражениях динамического языка.

Возвращает: boolean - Возвращает всегда “true”

Параметр	Описание
args	Список выражений динамического языка

Пример

```
run(
  [setValue([1],[id1])],
  [setValue([2],[id2])],
  [setValue([3],[id3])
])
```

A.4 if(condition, trueExpr, falseExpr)

Вычисляет выражение в первом аргументе, если результат вычислений равно true то выполняется выражение во втором аргументе, иначе третий

Возвращает: boolean - Возвращает значение первого аргумента true/false

Параметр	Описание
condition	Условие
trueExpr	Выражение 1
falseExpr	Выражение 2

Пример

```
if([idDigit([3])],
  [setValue([Число],[this])],
  [setValue([Строка],
[this])])
```

A.5 isIn(args)

Провереряет первый аргумент равен одному из следующих аргументов

Параметр	Тип	Описание
args	DLMap | *	Список параметров

Пример

```
isIn([1],[2],[56],[1])
```

A.6 isNotIn(args)

Провереряет первый аргумент не равен одному из следующих аргументов

Параметр	Описание
args	Список параметров

Пример

```
isNotIn([1],[2],[56],[1])
```

A.7 isDigit(arg)

Проверяет, является ли заданный аргумент числом

Параметр	Описание
arg	Аргумент

Пример

```
isDigit([34])
```

A.8 call(expr, id)

Выполняет команду динамического языка, переданную в первом параметре над элементом второго параметра, поддерживает выражения с асинхронными вызовами.

Возвращает: string идентификатор второго аргумента

Параметр	Тип	Описание
expr		Выражение на динамическом языке
id	string	Идентификатор элемента

Пример

```
call([setValue([test],[this])],[id])
```

A.9 IAm(arg)

Функция проверяет является ли пользователь приложения заявителем, инициатором или согласующим лицом в текущем обращении

Параметр	Описание
arg	Параметр может принимать значения creator, initiator или approver

Пример

```
iAm([creator])
```

A.10 getTexts(map)

Возвращает Map которая в value содержит видимые значения “label” option-элементов.

Возвращает: DLMap Map которая в value содержит видимые значения “label” option-элементов.

Параметр	Тип	Описание
map	DLMap	Map-а ключи которой содержат id элементов на форме

Наример, если в getTexts передать Map `{"f1": "-"}`с id элемента "f1"и выполнить над формой:

```
<form>
<select id="f1"> Значение 1
  <option id="f1_0"
 label="Отсутствует оплата за замещение должностей">
 Значение 1
   </option>
</select>
</form>
```

Функция вернет Map структуру вида

```
"f1\_Отсутствует оплата за замещение должностей" :
"Отсутствует оплата за замещение должностей"}
```

A.11 getValues(map)

Возвращает Map которая в value содержит значения “text” option-элементов.

Возвращает: DLMap - Возвращает Map которая в value содержит значения “text” option элементов.

Параметр	Тип	Описание
map	DLMap	Map ключи которой содержат id элементов на форме

Пример

Если в getValues передать Map `{"f1": "-"}` с id элемента "f1"и выполнить над формой

```
<form>
  <select id="f1"> Значение 1
    <option id="f1_0"
label="Отсутствует оплата за замещение должностей">Значение 1
    </option>
  </select>
</form>
```

Функция вернет Map структуру

```
{
"f1_Отсутствует оплата за замещение должностей" :
"Значение 1"
}
```

A.12 find(attr, value, group)

Универсальная функция, выполняет поиск элементов в зависимисти от первого аргумента. Если первый аргумент равен “id” то выполняется поиск элементов, id которых начинается со второго аргумента. Если первый аргумент равен “type”, то выполняется поиск элементов, sbtype - тип которых равен второму аргументу. В третьем параметре можно передать id группы в которой нужно выполнить поиск.

Возвращает: DLMap - Map в которой ключи это id найденных элементов, а value их значения

Param	Type	Description
attr	string	Атрбут по которому будет выполнен поиск, принимает значения “id” или “type”
value	string	Значение для поиска

Param	Type	Description
group	string	Идентификатор группы в которой будет выполнен поиск

Пример

```
Пример формы:

<form>
  <text id="f1" _parentId="0"
label="Наименование товара" style="text" type="2">
Ручка шариковая для сотрудника</text>

  <select id="f2"
 label="Вопрос" sbtype="suggest"  style="combo">
    <option id="id0empty" label=""/>
    <option id="id00"
label="Отсутствует оплата за работу в ночное время">a1</option>
    <option id="id01"
label="Отсутствует оплата за замещение должностей">a2</option>
  </select>
</form>

1. find([id], [f])

Результат:

DLMap {"f1": "Ручка шариковая для сотрудника","f2": ""}

2. find([type],[suggest])

Результат:

DLMap {"f2": ""}
```

A.13 plus(args)>

Универсальная функция сложения, которая умеет складывать числа, строки и время в формате HH:mm Первый аргумент может использоваться для формата чисел (для форматирования используется реализация Java DecimalFormat)

Возвращает: Результат сложения

Параметр	Описание
args	Список аргументов

Пример

```
plus([1],[2],[3])
6
plus([],[],[1],[],[2],[3],[],[4],[],[])
10
plus([1],[3],[a])
13a
plus([1],[01:02],[3])
66
plus([a],[01:02],[3])
a01:023
plus([#0.00],[123],[0,0000001])
123.00
plus([1],[DLMap{a : 1, b: 2, c: 3}],[3])
10
plus([1],[DLMap{a : 01:00, b: 2, c: 3}],[3])
69
plus([a],[DLMap{a : 01:00, b: 2, c: 3}],[3])
a01:00233
```

A.14 minus(args) -> *

Функция вычитания Первый аргумент может использоваться для формата чисел (для форматирования используется реализация Java DecimalFormat)

Возвращает: * - Результат вычитания из первого аргумента всех последующих аргументов

Параметр	Описание
args	Числа, например [1],[2],[3]

Пример

```
minus([4],[2],[1])
1
minus([#0.00],[123,0000002],[0,0000001])
123.00
```

A.15 mul(args) -> *

Функция умножения Первый аргумент может использоваться для формата чисел (для форматирования используется реализация Java DecimalFormat)

Возвращает: * - Результат умножения

Параметр	Описание
args	Числа, например [1],[2],[3]

A.16 div(args)

Функция деления Первый аргумент может использоваться для формата чисел (для форматирования используется реализация Java DecimalFormat)

Возвращает: * - Результат деления

Параметр	Описание
args	Числа, например [1],[2],[3]

A.17 indOf(string, substring)

Возвращает позицию с которой начинается подстрока substring в строке source
Возвращает: number | * - Позиция в строке

Параметр	Тип	Описание
string	DLMap | string	Строка
substring	DLMap | string	Искомая подстрока

Пример

```
indOf([console.log],[ole])
4
```

A.18 substr(string, start, end)

Возвращает подстроку с позиции start до позиции end
Возвращает: string - Подстрока

Параметр	Тип	Описание
string	string | DLMap	Строка в которой произоводится поиск
start	number | *	Начальная позиция
end	number | *	Конечная позиция (не включая)

Пример

```
substr([console.log], [2], [4])
```

A.19 len(string)

Возвращает длину строки
Возвращает: number | * - Длина строки

Параметр	Тип	Описание
string	DLMap | string	Строка

Пример

```
len([console.log])
11
```

A.20 copyGroup(groupId, id, group)

Копирует группу элементов. Поддерживает два вида групп: группировка по атрибуту setGroupId (ПОДРУГА) или sbgroup (SMIT); группировка по типу элемента - (object type) group. Возвращает второй аргумент.

Возвращает: string - Значение второго аргумента

Param	Type	Description
groupId	string	Идентификатор группы
id	string	Идентификатор
group	string	Идентификтор группы в которой нужно выполнить копирование группы

A.21 getDistance(args)

Возвращает из Яндекса расстояние между координатами

Возвращает: DLMap - Словарь со значениями distance - расстояние в метрах, distance_text - расстояних в км., time - время поездки в секундах, time_text - время в часах

Параметр	Тип	Описание
args	DLMap	Словарь состоящий из координат “широта:долгота”

Пример

```
address =   { // "longitude:latitude:GUID"
                  "a":"37.589283:55.745850:GUID"},
              "b":"36.715736:55.745850:GUID",
              "c":"37.515736:55.673816:GUID",
              "d":"37.715736:55.773816:GUID",
            }

getDistance([address])

Резульат:

DLMap {
 distance: '353637.28597939014',
 distance_text: '350 км',
 time: '25415.397077530622',
 time_text: '7 ч 4 мин'
 }
```

A.22 execute3DTask(TASK, args, map)

Получение данных из автосполнятора в виде 3DMap
Возвращает: DLMap - Ответ от автосполнятора

Параметр	Тип	Описание
TASK	DLMap | string	Название задачи в автосполняторе
ARG0	DLMap | string	Произвольный аргумет 1
ARG1	DLMap | string	Произвольный аргумет 2
ARGN	DLMap | string	Произвольный аргумет N
map	DLMap	Произвольный аргумет

A.23 executeTask

Получение данных из автосполнятора

Возвращает: Object | * - Ответ от автосполнятора

Параметр	Тип	Описание
TASK	DLMap | string	Название задачи в автосполняторе
ARG0	DLMap | string	Произвольный аргумет
ARG1	DLMap | string	Произвольный аргумет
ARGN	DLMap | string	Произвольный аргумет

A.24 execute3DTask

Получение данных из автосполнятора в виде 3DMap объекта с actions

Возвращает: DLMap | * - Ответ от автосполнятора

Параметр	Тип	Описание
TASK	DLMap | string	Название задачи в автосполняторе
ARG0	DLMap | string	Произвольный аргумет
ARG1	DLMap | string	Произвольный аргумет
ARGN	DLMap | string	Произвольный аргумет
DLMap	DLMap | string	текущее состояние динамической формы

A.25 getUsersByOrganizationRequest

Получение пользователя по подразделению

Параметр	Тип	Описание
realtyObj	DLMap | string	Идентификатор обьекта недвижимости
text	DLMap | string	Поисковый запрос

A.26 setValues(arg0-arg4)

Выполняет комманды 3DMaps над динамичской формой. Поддерживаются команды: attr, value, new, valueselected, command, delete, beforesave

Возвращает: null id - элемент над которым выполняет 3DMap(может отсутствовать, если запрос к АИ) tid - идентификатор запроса который выполняется в рамках обработки callback - функция обработки isDefaultSettings - включает или отключает silentMode

Параметр	Тип	Описание
ARG0	3DMap	map 3DMap структруа с коммандами
ARG1	string	id or tid
ARG2	string | function	tid or callback
ARG3	function | undefined	callback
ARG4	boolean | undefined	isDefaultSettings

Пример

```
Пример 3DMap.
[
 {
   "id": "cost",
   "actions": [
     {
       "action": "value",
       "values": [
         {
           "id": "22.25",
           "value": "22.25"
         }
       ]
     },
     {
       "action": "command",
       "values": [
         {
           "id": "100",
           "value": "setSaveEnabled([true],[cost])"
         }
       ]
```

```
    },
    {
      "action": "attr",
      "values": [
        {
          "id": "error",
          "value": ""
        },
        {
          "id": "rightValue",
          "value": "true"
        }
      ]
    }
  ]
 }
 ]
// map, id, tid, callback, isDefaultSettings
```

A.27 getValue(id)

Возвращает текущее значение элемента

Возвращает: DLMap со значениями элемента или пустую

Параметр	Описание
id	id элемента

A.28 setMandatory(bool, id)

Функция устанавливает обязательность заполнения контрола

Возвращает: id контрола

Параметр	Тип	Описание
bool		

Параметр	Тип	Описание
id	String | DLMap	id контола

A.29 setVisible(bool, id)

Функция устанавливает признак видимости элемента на форме
Возвращает: id контола

Параметр	Тип	Описание
bool		
id	String | DLMap	id контола

A.30 setEnabled(bool, id)

Функция устанавливает признак возможности редактирования элемента
Возвращает: id контола

Параметр	Тип	Описание
bool		
id	String | DLMap	id контола

A.31 setValid(bool, description, id)

Функция устанавливает контролу признак ошибки

Параметр	Описание
bool	
description	Текст ошибки
id	

A.32 setSelectedValue(val, id)

Функция устанавливает выбранное значение для контролов типа Select
Возвращает: id элемента

Параметр	Тип	Описание
val	String | DLMap	Значение option-элемнта - элемента списка
id		id элемента

Пример

```
setselectedvalue([a1],[f2])

Пример формы:

<form>
<select id="f2" setGroupId="group1" label="Вопрос" >
  <option id="id0empty" label=""/>
  <option id="id00"
label="Отсутствует оплата за работу в ночное время">a1
  </option>
  <option id="id01"
label="Отсутствует оплата за замещение должностей">a2
  </option>
</select>
</form>

Результат:

<form>
<select id="f2" setGroupId="group1" label="Вопрос" >a1
  <option id="id0empty" label=""/>
  <option id="id00"
label="Отсутствует оплата за работу в ночное время">a1
  </option>
  <option id="id01"
```

```
label="Отсутствует оплата за замещение должностей">a2
  </option>
</select>
</form>
```

A.33 setWidth(val, id)

Функция устанавливает ширину контрола
Возвращает: id элемента

Параметр	Описание
val	Значение ширины контрлоа
id	id элемента

A.34 getText(map, id)

1. Если элемент не select, то возвращает map где ключ и значени value элемента
2. Если элемент select и передан один аргумент - id элемента, то функция вернет map с text и label выбранного option- элемента
3. Если элемент select который не имеент выбранных option-элементов, то вернется map с одим пустым значением
4. Если первый аргумент функции map со значениями text - option элемнтов, то функция вернет map с text и label option- элементов

Param
map
id

A.35 isMandatory(id)

Возвращает обязательно ли поле для заполнения

Параметр	Описание
id	id элемента

A.36 isVisible(id)

Возвращает видимо ли поле на форме

Параметр	Описание
id	id элемента

A.37 isEnabled(id)

Возвращает доступен ли элемент для редактирования

Параметр	Описание
id	id элемента

A.38 isValid(id)

Возвращает правильно ли заполенен элемент формы

Параметр	Описание
id	id элемента

A.39 isFileMandatory(id)

Возвращает обязательность вложений при создании обращений

Tod: why func has id of element in params

Параметр	Описание
id	id элемента

A.40 isFileEnabled(id)

Возвращает признак доступности вложений

Параметр	Описание
id	id элемента

A.41 clear(id)

Очищает значение элемента
Возвращает: string - id элемента

Параметр	Тип	Описание
id	String | DLMap	id элемента

A.42 setList(_labels, _texts, id)

Устанавливает option-элементы
Возвращает: string - id элемента

Параметр	Тип	Описание
_labels	DLMap | string	строка label разделенная pipe-символом
_texts	DLMap | string	строка text разделенная pipe-символом
id		id элемента

Пример

```
setList([label1|label2|label3],[id1|id2|id3],[this])
```

A.43 setMask(value, id)

Устанавливает маску ввода

Параметр	Описание
value	Маска ввода
id	id элемента

A.44 setFocus(val, id)

Устанавливает фокус на элементе

Параметр	Тип	Описание
val	boolean	Значение
id		id элемента

A.45 setMax(value, id)

Устанавливает максимальное значение для элемента

Параметр	Описание
value	Максимальное значение
id	id элемента

A.46 setMin(value, id)

Устанавливает минимальное значение для элемента

Параметр	Описание
value	Минимальное значение
id	id элемента

A.47 string(value)

Возвращает строковое значение первого параметра
Возвращает: string - Строковое значение первого параметра

Param
value

A.48 setBeforeSave(value, id)

Устанавливает комманду, которая будет выполнена перед сохранением
Возвращает: string - id элемента

Параметр	Описание
value	Комманда
id	id элемента

A.49 getApprovalStage(id)

Возвращает название этапа согласования по обращению
Возвращает: string - Название этапа согласования
Throws:

- Error("approvalStage у внешнего обьекта не определен")

Параметр	Описание
id	id элемента

A.50 setCheckSign(bool, id)

Устанавливает признак подписания формы при сохранения обращения

Throws:

- FunctionNotImplementedError

Параметр	Описание
bool	
id	id элемента

A.51 setFileMandatory(bool, id)

Устаналивает признак обязательности вложения файла
Возвращает: string - id элемента

Параметр	Описание
bool	
id	id элемента

A.52 openAlert(type, message)

Откpывает окно с сообщением
Throws:

- FunctionNotImplementedError

Параметр	Описание
type	Тип сообщения info, warning, error
message	Текст сообщения

A.53 setSaveEnabled(bool, id)

Устанавливает признак доступности кнопкаи “Сохранить”

Возвращает: string - id элемента

Параметр	Описание
bool	
id	id элемента

A.54 setFileEnabled(bool, id)

Устанавливает признак доступности кнопки “Вложить”
Возвращает: string - id элемента

Параметр	Описание
bool	
id	id элемента

A.55 setExternalSystem(extsystem, id)

Устанавливает код внешней системы
Возвращает: string - id элемента

Param	Type	Description
extsystem	SM | FRIEND	Код внешней системы SM или FRIEND
id		id элемента

A.56 setInformationVisible(bool, id)

Устанавливает признак видимости поля “Информация”
Возвращает: string - id элемента

Параметр	Описание
bool	

Параметр	Описание
id	элемента

A.57 setInitiator(value, id)

Устанавливает инициатора в обращении
Возвращает: string - id элемента

Параметр	Описание
value	Идентификатор инициатора
id	id элемента

A.58 setMultiFlow(bool, id)

Устанавливает признак мультипоточного ввода по шаблону
Возвращает: * - id элемента

Параметр	Описание
bool	
id	id элемента

A.59 setRedirect(value, id, skipConvert)

Устанавливает ссылку на которую надо перейти
Возвращает: * - id элемента

Параметр	Тип	Описание
value	string	Ссылка
id	string	id элемента
skipConvert	Boolean	не преобразовывать ссылку LD1 > LD2

A.60 setRelationship(objectId, objectType, id)

Используется только с ПОДРУГа! Функция устанавливает связь созданного обращения с обьектом типа objectType с номером objectId

Возвращает: id элемента

Параметр	Описание
objectId	Номер обьекта
objectType	Тип обьекта ZNO/ZNU
id	id элемента

A.61 setAdditionalTemplate

Устанавливает шаблона по которому нужно дополнительно создать обращение

Throws:

- FunctionNotImplementedError

Param	Type	Description
extsystem	SM | FRIEND	Код внешней системы SM или FRIEND
templateId		Идентификатор шаблона обращения
id		id элемента

A.62 setScoringVisible(bool, id)

Устанавливает признак видимости поля «Скоринг» в обращении

Параметр	Описание
bool	

Параметр	Описание
id	id элемента

A.63 getSysDate(value)

Возвращает смещение относительно текущей даты
Возвращает: Дата

Параметр	Описание
value	Смещение в днях

Пример

```
getSysDate([+5]) - вернет текущую дату + 5 дней
```

A.64 getInitiator(value)

Получение информации об инициаторе обращения
Возвращает: string - Информация по параметру

Param	Description
value	Параметр fio, domaincode, ou, tabnum, id, departcode, position, guid, email, vsp, tbcode

A.65 setMinHours(value, id)

Устаналивает минимальное значение в часах
Возвращает: * - id элемента

Параметр	Описание
value	Значение часов в 24 - часовом формате

Параметр	Описание
id	id элемента

A.66 setMaxHours(value, id)

Устаналивает максимальное значение в часах
Возвращает: * - id элемента

Параметр	Описание
value	Значение часов в 24 - часовом формате
id	id элемента

A.67 setContainCustomObjects(bool, id)

Устанавливает признак того что созданное обращение будет содержать “хранимые” обьекты, например обьет закупки, поездки и т.д.

Param
bool
id

A.68 getSize(param)

Возвращает количество option-элементов в элементе если передан идентификатор элемента или количество полей в DLMap если передана DLMap
Возвращает: * - id элемента

Параметр	Тип	Описание
param	string | DLMap	id элемента или DLMap

A.69 setStep(value, id)

Устанавливает шаг для выбора значения

Возвращает: * - id элемента

Параметр	Описание
value	Размер шага
id	id элемента

A.70 setStyle(arg, value, id)

Устанавливает стиль для атрибута

Поддерживаемые стили:

color - цвет элемента badge - текст badge - элемента badge-color - цвет badge - элемента width - ширина элемнта (только web-версии Лица ДРУГА)

Возвращает: string - id элемента

Параметр	Описание
arg	Стиль
value	Значение
id	id элемента

A.71 compare(left, operand, rigth)

Функция сравнения

Доступны операторы сравнения >, >=, ==, <, <=

Параметр	Описание
left	Перый аргумент для сравнения
operand	Оператор сравнения
rigth	Второй аргумент для сравнения

A.72 setInitiatorVisible(bool, id)

Устаналивает признак видимости поля смены ВК
Возвращает: string - id элемента

Параметр	Описание
bool	
id	id элемента

A.73 setDescription(text, id)

Устанавливает лейбл у элемента
Возвращает: String - Идентификатор элемента

Параметр	Тип	Описание
text	String	Тексе лейбла
id	String	Идентификатор элемента

A.74 getRelatedId()

Возвращает идентификатор связанного обращения
Возвращает: String - Идентификатор связанного обращения

A.75 setCopyVisible(bool, id)

Устанавливает видимость кнопки копирования шаблона
Возвращает: String - Возвращает второй аргумент

Параметр	Тип
bool	Boolean
id	String

A.76 setDeclineVisible(bool, id)

Устанавливает видимость кнопки “Отменить обращение”

Возвращает: String - Возвращает второй аргумент

Параметр	Тип
bool	Boolean
id	String

A.77 setCloseVisible(bool, id)

Устанавливает видимость кнопки “Закрыть группу”

Возвращает: String - Возвращает второй аргумент

Параметр	Тип
bool	Boolean
id	String

A.78 getClientType()

Возвращает тип клиента. Возможные значение: ios, android, web-sigma, web-alpha

Возвращает: string - Тип клиента

A.79 isMobileClient()

Возвращает является ли клиент мобильным приложением

A.80 isWebClient()

Возвращает является ли клиент web-приложением

A.81 isIOSClient()

Возвращает является ли клиент iOs-приложением

A.82 isAndroidClient()

Возвращает является ли клиент Android-приложением

A.83 isWebSigmaClient()

Возвращает является ли клиент web sigma приложением

A.84 isWebAlphaClient()

Возвращает является ли клиент web alpha приложением

A.85 isFace20()

Функция возвращает true если это web прложение face20 Костыль нужен для авто редиректа на /friendface если шаблон не работает на face20. Оставляем пока face20 не стабилизируется.

A.86 setAction(value, id)

Устанавливает значение в атрибут sbaction элемента
Возвращает: id

Param
value
id

A.87 setMode(value, id)

Устанавливает значение в атрибут sbmode элемента
Возвращает: id

Param
value
id

A.88 dateDiff(start, end, id)

Возвращает разницу в днях

Param
start
end
id

A.89 isCreateMode()

Форма в режиме создания обращения

A.90 isOpenMode()

Форма в режиме созданного обращения

A.91 setSaveVisible(bool, id)

Устанавливает видимость кнопки сохранения шаблона
Возвращает: String - Возвращает второй аргумент

Параметр	Тип
bool	Boolean

Параметр	Тип
id	String

A.92 _setValue(value, id, revertValues)

Устанавливает значение первого параметра в элемент, указанный во втором параметре.

Возвращает: String - Возвращает второй аргумент

Param	Type	Description
value	DLMap | string	
id	String	
revertValues	bool | undefined	управляет пердачей флага по перевёрнутым значениям в DLMap

A.93 _runSBTASK(el)

Запускает выполнение sbtask в случае если выполнются внутренние условия.

Параметр	Тип
el	Element

A.94 _getValueByUser(user, attr, isUser)

Получаем общие кадровые данные по пользователю или инициатору.

Param	Type	Description
user	User | Initiator	
attr	string	

Param	Type	Description
isUser	boolean	проверка данные у пользователя или инициатора(заглушка временная)

A.95 createDocument

Заполняет шаблон документа (templateName) данными из обращения и возвращает его в качестве вложения с названием fileName.extension | Param | Type | Description | | — | — | — | | fileName | String | Имя возвращаемого вложенияя | | templateName | String | Имя шаблона документа, куда будут подставлены данные из обращения | | extension | String | Расширение возвращаемого вложения (доступные значения PDF). Не обязательный параметр |

Пример:

```
setValue(
  [createDocument(
    [Заявление], [ЗаявлениеШаблон], [PDF])],
  [attachment_el_id]
)
```

Делегирует обращение к серверу и формирование документа-вложения функции doCreateDocument(tid, params, callback) Пример:

```
DOMCore.doCreateDocument = (tid, params, callback) => {
  //params.fileName - имя возвращаемого вложения
  //params.templateName - имя шаблона документа
  //params.extension - расширение возвращаемого вложения
  //params.interaction - объект Interaction
  const response = anyDelegateFunction(params.fileName,
  params.templateName, params.extension, params.interaction);
  callback(tid, response, null);
}
```

Функция доступна из JS шаблона через глобальный вызов doCreateDocument
Пример:

```
var fileName = 'Заявление';
var templateName = 'ЗаявлениеШаблон';
var extension = 'PDF';
doCreateDocument(fileName, templateName,
  extension, onSuccess, onError);

var onSuccess = function(attachmentData) {
  console.log(JSON.stringify(attachmentData));
}

var onError = function(err) {
  console.error(err && err.userMessage);
}
```

A.96 signAttachment(attachment,mode)

Запускает процесс подписи вложения (attachment) и возвращает подписанное вложение<

Param	Type	Description
attachment	String	Объект вложения(й) (AttachmentDTO) сериализованный в JSON
mode	String	Режим подписи (доступные значения CLOUD - облачная подпись, TABLET - подпись через ТМ)

Пример:

```
setValue(
```

```
  [signAttachment(
    [getValue([nonsignedattachment])],
    [CLOUD]
  )],
  [attachmentelement]
)
```

Делегирует подписание функции
doSignAttachment(tid, signAttachmentParams, callback)
Пример:

```
DOMCore.doSignAttachment = (tid, params, callback) => {
  // params.attachment - подписываемое вложения
  // params.signMethod - метод подписания (CLOUD, TABLET)
  const response = anyDelegateFunction(params);
  callback(tid, response, null);
}
```

Функция доступна из JS шаблона через глобальный вызов

```
doSignAttachment(attachment, method, onSuccess, onError)
```

Пример:

```
var attachment = {asyncId: 12345};
var method = 'CLOUD';
doSignAttachment(attachment, method, onSuccess, onError);

var onSuccess = function(attachmentData) {
  console.log(JSON.stringify(attachmentData));
}

var onError = function(err) {
  console.error(err && err.userMessage);
}
```