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

Глава 1

Введение

1.1 Что такое программа

Программа - термин, в переводе означающий «предписание», то есть заданную последовательность действий. В динамическом языке программой принято называть последовательность комманд(Команда) записанную в формате функций с аргументами. Каждый аргумент указывается в квадратных скобочках. Аргументы между собой разделяется запятыми (например: скажи([привет],[как],[дела])). Основная задача программ в динамическом языке - управление состоянием и значениями элементов динамической формы(подробнее в разделе Динамическая форма). Должная сноровка и немного усилий после изучения данной книги позволит написать вам программу решающую любой бизнес-кейс возникающий на просторах вселенной. До сегодняшнего дня никто не знает пределов возможностей динамического языка.

1.1.1 Основные концепции

Предугадав тенденции развития функциональных языков программирования, динамический язык взял за основу концепцию именно данного типа языков, по факту в динамическом языке отсутствуют переменные, константы и остальные нагромождения ООП, все реализуется через функции или через чистые функции. Хотя результатом функций и являются объекты с которыми происходит взаимодействие, но по мнению автора(ов) при разработке программ на динамическом языке стоит придерживаться функциональных подходов программирования. Это позволит полноценно оценить все тонкости разработки. Константы и строки в динамическом языке являются чистыми функциями в результате возвращающими литеральное значение самих себя.

В частности чистая функция `this` или `__this__` возвращает объект над которым производится вычисление функции А функция `text` вернет строковый литерал соответствующий самому себе.

Вышесказанные концепции очень важны для дальнейшего понимания функционального ядра динамического языка.

1.2 Запуск динамических команд

Одним из первых вызовов, который бросает динамический язык начинающему разработчику, является необходимость настройки рабочего окружения HP Service Manager для редактирования шаблонов динамических форм. 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».

1.3 Первая программа

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 создани 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

```
>>>
```

1.4 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])
```

1.5 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, 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.

1.6 Упражнения

Откройте шаблон «Исполнятор» и 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 6 багов, и обнаруживается 4 новых?

Глава 2

Структура программы

2.1 Значения и типы

2.2 Идентификаторы

2.3 Возвращаемые значения

2.4 Порядок выполнения

В основу выполнения команд динамического языка лежит очень важная и 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.

Пример

Время шло и бизнес требования развития динамических форм внесли свои коррективы, а именно потребовалась реализация асинхронных методом, каждый из которых мог бы выполняться в отдельном потоке или эмулировать выполнение, что потребовало серьезных изменений в структуре разбора и выполнения функций.

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 «компиляции» 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 функции(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 писать сколько угодно сложные программные решения над динамическими формами.

2.5 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

2.6 Операторы ветвления

2.7 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»

2.8 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

Глава 4

Группы

4.1 Виды групп

4.2 Копирование групп

4.3 Удаление групп

Глава 5

Асинхронные команды

5.1 Автоисполнятор

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 предшествуюшей 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. 2.7