

Модуль подсистемы “Сбор данных” <JavaLikeCalc>

Модуль:	JavaLikeCalc
Имя:	Вычислитель на Java-подобном языке.
Тип:	DAQ
Источник:	daq_JavaLikeCalc.so
Версия:	1.9.0
Автор:	Роман Савоченко
Описание:	Предоставляет основанные на java подобном языке вычислитель и движок библиотек. Пользователь может создавать и модифицировать функции и их библиотеки.
Лицензия:	GPL

Оглавление

Модуль подсистемы “Сбор данных” <JavaLikeCalc>.....	1
Введение.....	2
1. Java-подобный язык.....	4
1.1. Элементы языка.....	4
1.2. Операции языка.....	5
1.3. Встроенные функции языка.....	6
1.4. Операторы языка.....	6
1.5. Объект.....	8
1.6. Примеры программы на языке.....	11
2. Контроллер и его конфигурация.....	12
3. Параметр контроллера и его конфигурация.....	14
4. Библиотеки функций модуля.....	15
5. Пользовательские функции модуля.....	15

После любого изменения программы или конфигурации параметров выполняется перекомпиляция программы с упреждением связанных с функцией объектов значений TValCfg. Компилятор языка построен с использованием известного генератора грамматики «Bison», который совместим с не менее известной утилитой Yacc.

Язык использует неявное определение локальных переменных, которое заключается в определении новой переменной в случае присваивания ей значения. Причём тип локальной переменной устанавливается в соответствии с типом присваиваемого значения. Например, выражение $Qr = Q0 \cdot Pi + 0.01;$ определит переменную Qr с типом переменной $Q0$.

В работе с различными типами данных язык использует механизм автоматического приведения типов в местах, где подобное приведение является целесообразным.

Для комментирования участков кода в языке предусмотрены символы `«//»` и `«/* ... */»`. Всё, что идёт после `“//”` до конца строки и между `«/* ... */»` игнорируется компилятором.

В процессе генерации кода компилятор языка производит оптимизацию по константам и приведение типов констант к требуемому типу. Под оптимизацией констант подразумевается выполнение вычислений в процессе построения кода над двумя константами и вставка результата в код. Например, выражение $y = pi \cdot 10;$ свернётся в простое присваивание $y = 31.4159;$. Под приведением типов констант к требуемому типу подразумевается формирование в коде константы, которая исключает приведение типа в процессе исполнения. Например, выражение $y = x \cdot 10$, в случае вещественного типа переменной x , преобразуется в $y = x \cdot 10.0$.

Язык поддерживает вызовы внешних и внутренних функций. Имя любой функции вообще воспринимается как символ, проверка на принадлежность которого к той или иной категории производится в следующем порядке:

- ключевые слова;
- константы;
- встроенные функции;
- внешние функции, функции объекта и системных узлов OpenSCADA (DOM);
- уже зарегистрированные символы переменных, атрибуты объектов и иерархия объектов DOM;
- новые атрибуты системных параметров;
- новые параметры функции;
- новая автоматическая переменная.

Вызов внешней функции, как и атрибута системного параметра, записывается как адрес к объекту динамического дерева объектной модели системы OpenSCADA в виде: `<DAQ.JavaLikeCalc.lib_techApp.klapNotLin>`.

Для предоставления возможности написания пользовательских процедур управления различными компонентами OpenSCADA модулем предоставляется реализация API прекомпиляции пользовательских процедур отдельных компонентов OpenSCADA на реализации Java-подобного языка. Такими компонентами уже являются: Шаблоны параметров подсистемы «Сбор данных» и Среда визуализации и управления (СВУ).

1. Java-подобный язык

1.1. Элементы языка

Ключевые слова: if, else, while, for, break, continue, return, using, true, false.

Постоянные:

- десятичные: цифры 0–9 (12, 111, 678);
- восьмеричные: цифры 0–7 (012, 011, 076);
- шестнадцатеричные: цифры 0–9, буквы a-f или A-F (0x12, 0XAB);
- вещественные: 345.23, 2.1e5, 3.4E-5, 3e6;
- логические: true, false;
- строковые: «hello».

Типы переменных:

- целое: $-2^{31} \dots 2^{31}$;
- вещественное: $3.4 * 10^{308}$;
- логическое: false, true;
- строка: длина до 255 символов и без перехода на другую строку.

Встроенные константы: pi = 3.14159265, e = 2.71828182, EVAL_BOOL(2), EVAL_INT(-2147483647), EVAL_REAL(-3.3E308), EVAL_STR("<EVAL>")

Атрибуты параметров системы OpenSCADA (начиная с подсистемы DAQ, в виде <Тип модуля DAQ>.<Контроллер>.<Параметр>.<Атрибут>).

Функции объектной модели системы OpenSCADA.

1.2. Операции языка

Операции, поддерживаемые языком, представлены в таблице ниже. Приоритет операций уменьшается сверху вниз. Операции с одинаковым приоритетом входят в одну цветовую группу.

Символ	Описание
()	Вызов функции.
{ }	Программные блоки.
++	Инкремент (пост и пре).
--	Декремент (пост и пре).
-	Унарный минус.
!	Логическое отрицание.
~	Побитовое отрицание.
*	Умножение.
/	Деление.
%	Остаток от целочисленного деления.
+	Сложение
-	Вычитание
<<	Поразрядный сдвиг влево
>>	Поразрядный сдвиг вправо
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно
==	Равно
!=	Неравно
	Поразрядное «ИЛИ»
&	Поразрядное «И»
^	Поразрядное «Исключающее ИЛИ»
&&	Логический «И»
	Логический «ИЛИ»
?:	Условная операция (i=(i<0)?0:i;)
=	Присваивание.
+=	Присваивание со сложением.
-=	Присваивание с вычитанием.
*=	Присваивание с умножением.
/=	Присваивание с делением.

1.3. Встроенные функции языка

Виртуальной машиной языка предусматривается следующий набор встроенных функций общего назначения:

- `double max(double x, double x1)` — максимальное значение из x и $x1$;
- `double min(double x, double x1)` — минимальное значение из x и $x1$;
- `string typeof(ElTr vl)` — тип значения vl .

Для обеспечения высокой скорости работы в математических вычислениях модуль предоставляет встроенные математические функции, которые вызываются на уровне команд виртуальной машины:

- `double sin(double x)` — синус x ;
- `double cos(double x)` — косинус x ;
- `double tan(double x)` — тангенс x ;
- `double sinh(double x)` — синус гиперболический от x ;
- `double cosh(double x)` — косинус гиперболический от x ;
- `double tanh(double x)` — тангенс гиперболический от x ;
- `double asin(double x)` — арксинус от x ;
- `double acos(double x)` — арккосинус от x ;
- `double atan(double x)` — арктангенс от x ;
- `double rand(double x)` — случайное число от 0 до x ;
- `double lg(double x)` — десятичный логарифм от x ;
- `double ln(double x)` — натуральный логарифм от x ;
- `double exp(double x)` — экспонента от x ;
- `double pow(double x, double x1)` — возведение x в степень $x1$;
- `double sqrt(double x)` — корень квадратный от x ;
- `double abs(double x)` — абсолютное значение от x ;
- `double sign(double x)` — знак числа x ;
- `double ceil(double x)` — округление числа x до большего целого;
- `double floor(double x)` — округление числа x до меньшего целого.

1.4. Операторы языка

Общий перечень операторов языка:

- `var` — оператор инициализации переменной;
- `if` — оператор условия "Если";
- `else` — оператор условия "Иначе";
- `while` — описание цикла `while`;
- `for` — описание цикла `for`;
- `in` — разделитель цикла `for` для перебора свойств объекта;
- `break` — прерывание выполнения цикла;
- `continue` — продолжить выполнение цикла с начала;
- `using` — позволяет установить область видимости функций часто используемой библиотеки (`using Special.FLibSYS`;) для последующего обращения только по имени функции;
- `return` — прерывание функции и возврат результата, результат копируется в атрибут с флагом возврата (`return 123`);
- `new` — создание объекта, реализованы объект "Object", массив "Array" и регулярные выражения "RegExp".

1.4.1. Условные операторы

Языком модуля поддерживаются два типа условий. Первый это операции условия для использования внутри выражения, второй – глобальный, основанный на условных операторах.

Условие внутри выражения строится на операциях «?» и «:». В качестве примера можно записать следующее практическое выражение `<st_open=(pos>=100)?true:false;>`, что читается как «Если переменная `<pos>` больше или равна 100, то переменной `st_open` присваивается значение `true`, иначе - `false`.

Глобальное условие строится на основе условных операторов «if» и «else». В качестве примера можно привести тоже выражение, но записанное другим способом `<if(pos>100) st_open=true; else st_open=false;>`. Как видно, выражение записано по-другому, но читается также.

1.4.2. Циклы

Поддерживаются три типа циклов: `while`, `for` и `for-in`. Синтаксис циклов соответствует языкам программирования: C++, Java и JavaScript.

Цикл **while** в общем записывается следующим образом:

`while(<условие>) <тело цикла>;`

Цикл **for** записывается следующим образом:

`for(<пре-инициализ>;<условие>;<пост-вычисление>) <тело цикла>;`

Цикл **for-in** записывается следующим образом:

`for(<переменная> in <объект>) <тело цикла>;`

Где:

`<условие>` — выражение, определяющее условие;

`<тело цикла>` — тело цикла множественного исполнения;

`<пре-инициализ>` — выражение предварительной инициализации переменных цикла;

`<пост-вычисление>` — выражение модификации параметров цикла после очередной итерации;

`<переменная>` — переменная, которая будет содержать имя свойства объекта при переборе;

`<объект>` — объект для которого осуществляется перебор свойств.

1.4.3. Специальные символы строковых переменных

Языком предусмотрена поддержка следующих специальных символов строковых переменных:

`"\n"` — перевод строки;

`"\t"` — символ табуляции;

`"\b"` — забой;

`"\f"` — перевод страницы;

`"\r"` — возврат каретки;

`"\""` — сам символ `"\`;

`"\041"` — символ `!` записанный восьмеричным числом;

`"\x21"` — символ `!` записанный шестнадцатеричным числом.

1.5. Объект

Языком предоставляется поддержка типа данных объект "Object". Объект представляет собой ассоциативный контейнер свойств и функций. Свойства могут содержать как данные четырёх базовых типов, так и другие объекты. Доступ к свойствам объекта может осуществляться посредством записи имён свойств через точку к объекту `<obj.prop>`, а также посредством заключения имени свойства в квадратные скобки `<obj["prop"]>`. Очевидно, что первый механизм статичен, а второй позволяет указывать имя свойства через переменную. Создание объекта осуществляется посредством ключевого слова `<new>`: `<varO = new Object()>`. Базовое определение объекта не содержит функций. Операции копирования объекта на самом деле делают ссылку на исходный объект. При удалении объекта осуществляется уменьшение счётчика ссылок, а при достижении счётчика ссылок нуля объект удаляется физически.

Разные компоненты могут доопределять базовый объект особыми свойствами и функциями. Стандартным расширением объекта является массив "Array", который создаётся командой `<varO = new Array(prm1,prm2,prm3,...,prmN)>`. Перечисленные через запятую параметры помещаются в массив в исходном порядке. Если параметр только один то массив иницируется указанным количеством пустых элементов. Особенностью массива является то, что он работает со свойствами как с индексами и полное их именование бессмысленно, а значит доступен механизм обращения только заключением индекса в квадратные скобки `<arr[1]>`. Массив хранит свойства в собственном контейнере одномерного массива. Цифровые свойства массива используются для доступа непосредственно к массиву, а символьные работают как свойства объекта. Детальнее про свойства и функции массива можно прочитать по [ссылке](#).

Объект регулярного выражения RegExp создаётся командой `<varO = new RegExp(pat,flg)>`, где `<pat>` - шаблон регулярного выражения, а `<flg>` - флаги поиска. Объект работы с регулярными выражениями, основан на библиотеке PCRE. При глобальном поиске устанавливается атрибут объекта "lastIndex", что позволяет продолжить поиск при следующем вызове функции. В случае неудачного поиска атрибут "lastIndex" сбрасывается в ноль. Детальнее про свойства и функции массива можно прочитать по [ссылке](#).

Для произвольного доступа к аргументам функции предусмотрен объект аргументов, обратиться к которому можно посредством символа "arguments". Этот объект содержит свойство "length" с количеством аргументов у функции и позволяет обратиться к значению аргумента посредством его номера или идентификатора. Рассмотрим перебор аргументов по циклу:

```
args = new Array();
for(var i=0; i < arguments.length; i++)
    arg[i] = arguments[i];
```

Частичными свойствами объекта обладают и базовые типы. Свойства и функции базовых типов приведены ниже:

- Логический тип, функции:
 - `bool isEval()`; — Проверка значения на "EVAL".
 - `string toString()`; — Представление значения в виде строки "true" или "false".
- Целое и вещественное число:

Свойства:

 - `MAX_VALUE` — максимальное значение;
 - `MIN_VALUE` — минимальное значение;
 - `NaN` — недостоверное значение.

Функции:

 - `bool isEval()`; — Проверка значения на "EVAL".
 - `string toExponential(int numbs)`; — Возврат строки отформатированного числа в экспоненциальной нотации и количеством значащих цифр `<numbs>`. Если `<numbs>`

отсутствует то цифр будет столько сколько необходимо.

- *string toFixed(int numbs);* — Возврат строки отформатированного числа в нотации с фиксированной точкой и количеством цифр после десятичной точки *<numbs>*. Если *<numbs>* отсутствует то количество цифр после десятичной точки равно нулю.
- *string toPrecision(int prec);* — Возврат строки отформатированного числа с количеством значащих цифр *<prec>*.
- *string toString(int base);* — Возврат строки отформатированного числа целого типа с базой представления 8-восьмеричное, 10-десятичное и 16-шестнадцатеричное.

• Строка:

Свойства:

- *int length* — длина строки.

Функции:

- *bool isEval();* — Проверка значения на "EVAL".
- *string charAt(int symb);* — Извлекает из строки символ *<symb>*.
- *int charCodeAt(int symb);* — Извлекает из строки код символа *<symb>*.
- *string concat(string val1, string val2, ...);* — Возвращает новую строку сформированную путём присоединения значений *<val1>* и т.д. к исходной.
- *int indexOf(string substr, int start);* — Возвращает позицию искомой строки *<substr>* в исходной строке начиная с позиции *<start>*. Если исходная позиция не указана то поиск начинается с начала. Если искомой строки не найдено то возвращается -1.
- *int lastIndexOf(string substr, int start);* — Возвращает позицию искомой строки *<substr>* в исходной строке начиная с позиции *<start>* при поиске с конца. Если исходная позиция не указана то поиск начинается с конца. Если искомой строки не найдено то возвращается -1.
- *int search(string pat, string flg = "");* — Поиск в строке по шаблону *<pat>* и флагами шаблона *<flg>*. Возвращает положение найденной подстроки иначе -1.

```
var rez = "Java123Script".search("script","i");  
# rez = 7
```
- *int search(RegExp pat);* — Поиск в строке по шаблону RegExp *<pat>*. Возвращает положение найденной подстроки иначе -1.

```
var rez = "Java123Script".search(new RegExp("script","i"));  
# rez = 7
```
- *int match(string pat, string flg = "");* — Поиск в строке по шаблону *<pat>* и флагами шаблона *<flg>*/. Возвращает массив с найденной подстрокой (0) и подвыражениями (>1). Атрибут "index" массива устанавливается в позицию найденной подстроки. Атрибут "input" устанавливается в исходную строку.

```
var rez = "1 плюс 2 плюс 3".match("\\d+","g");  
# rez = [1], [2], [3]
```
- *int match(TRegExp pat);* — Поиск в строке по шаблону RegExp *<pat>*. Возвращает массив с найденной подстрокой (0) и подвыражениями (>1). Атрибут "index" массива устанавливается в позицию найденной подстроки. Атрибут "input" устанавливается в исходную строку.

```
var rez = "1 плюс 2 плюс 3".match(new RegExp("\\d+","g"));  
# rez = [1], [2], [3]
```
- *string slice(int beg, int end); string substring(int beg, int end);* — Возврат подстроки извлечённой из исходной начиная с позиции *<beg>* и заканчивая *<end>*. Если значение начала или конца отрицательно, то отсчёт ведётся с конца строки. Если конец не указан, то концом является конец строки.
- *Array split(string sep, int limit);* — Возврат массива элементов строки разделённых *<sep>* с ограничением количества элементов *<limit>*.
- *Array split(RegExp pat, int limit);* — Возврат массива элементов строки разделённых шаблоном RegExp *<pat>* с ограничением количества элементов *<limit>*.

```
rez = "1,2, 3 , 4 ,5".split(new RegExp("\\s*,\\s*"));
```

- ```
rez = [1], [2], [3], [4], [5]
```
- *string insert(int pos, string substr);* — Вставка в позицию *<pos>* текущей строки подстроку *<substr>*.
  - *string replace(int pos, int n, string str);* — Замена подстроки с позиции *<pos>* и длиной *<n>* в текущей строке на строку *<str>*.  

```
rez = "Javascript".replace(4, 3, "67");
rez = "Java67ipt"
```
  - *string replace(string substr, string str);* — Замена всех подстрок *<substr>* на строку *<str>*.  

```
rez = "123 321".replace("3", "55");
rez = "1255 5521"
```
  - *string replace(RegExp pat, string str);* — Замена подстрок по шаблону *<pat>* на строку *<str>*.  

```
rez = "value = \"123\"".replace(new
RegExp("\\([^\"]*)\\\"", "g"), "\"`$1'\"");
#rez = "value = `123'\""
```
  - *real toReal();* — преобразование текущей строки в вещественное число.
  - *int toInt(int base = 0);* — преобразование текущей строки в целое число, в соответствии с основанием *<base>* (от 2 до 36). Если основание равно 0 то будет учитываться префиксная запись для определения основания (123-десятичное; 0123-восьмеричное; 0x123-шестнадцатиричное).
  - *string parse(int pos, string sep = ".", int off = 0);* — выделение из исходной строки элемента *<pos>* для разделителя элементов *<sep>* от смещения *<off>*. Результирующее смещение помещается назад в *<off>*.
  - *string parsePath(int pos, int off = 0);* — выделение из исходного пути элемента *<pos>* от смещения *<off>*. Результирующее смещение помещается назад в *<off>*.
  - *string path2sep(string sep = ".");* — преобразование пути в текущей строке в строку с разделителем *<sep>*.

Для доступа к системным объектам(узлам) OpenSCADA предусмотрен соответствующий объект, который создаётся путём простого указания точки входа "SYS" корневого объекта OpenSCADA, а затем, через точку указываются вложенные объекты в соответствии с иерархией. Например, вызов функции запроса через исходящий транспорт осуществляется следующим образом: *SYS.Transport.Sockets.out\_testModBus.messIO(strEnc2Bin("15 01 00 00 00 06 01 03 00 00 00 05"));*.

## 1.6. Примеры программы на языке

Приведём несколько примеров программ на Java-подобном языке:

```
//Модель хода исполнительного механизма шарового крана
if(!(st_close && !com) && !(st_open && com))
{
 tmp_up=(pos>0&&pos<100)?0:(tmp_up>0&&lst_com==com)?tmp_up-1./frq:t_up;
 pos+=(tmp_up>0)?0:(100.*(com?1.: -1.))/(t_full*frq);
 pos=(pos>100)?100:(pos<0)?0:pos;
 st_open=(pos>=100)?true:false;
 st_close=(pos<=0)?true:false;
 lst_com=com;
}
//Модель клапана
Qr=Q0+Q0*Kpr*(Pi-1)+0.01;
Sr=(S_kl1*l_kl1+S_kl2*l_kl2)/100.;
Ftmp=(Pi>2.*Po)?Pi*pow(Q0*0.75/Ti,0.5):(Po>2.*Pi)?
 Po*pow(Q0*0.75/To,0.5):pow(abs(Q0*(pow(Pi,2)-pow(Po,2))/Ti),0.5);
Fi=(Fi-7260.*Sr*sign(Pi-Po)*Ftmp)/(0.01*lo*frq);
Po+=0.27*(Fi-Fo)/(So*lo*Q0*frq);
Po=(Po<0)?0:(Po>100)?100:Po;
To+=(abs(Fi)*(Ti*pow(Po/Pi,0.02)-To)+(Fwind+1)*(Twind-To)/Riz)/
 (Ct*So*lo*Qr*frq);
```

## 2. Контроллер и его конфигурация

Контроллер этого модуля связывается с функциями из библиотек, построенных с его помощью, для обеспечения непосредственных вычислений. Для предоставления вычисленных данных в систему OpenSCADA в контроллере могут создаваться параметры. Пример вкладки конфигурации контроллера данного типа изображен на рис.2.

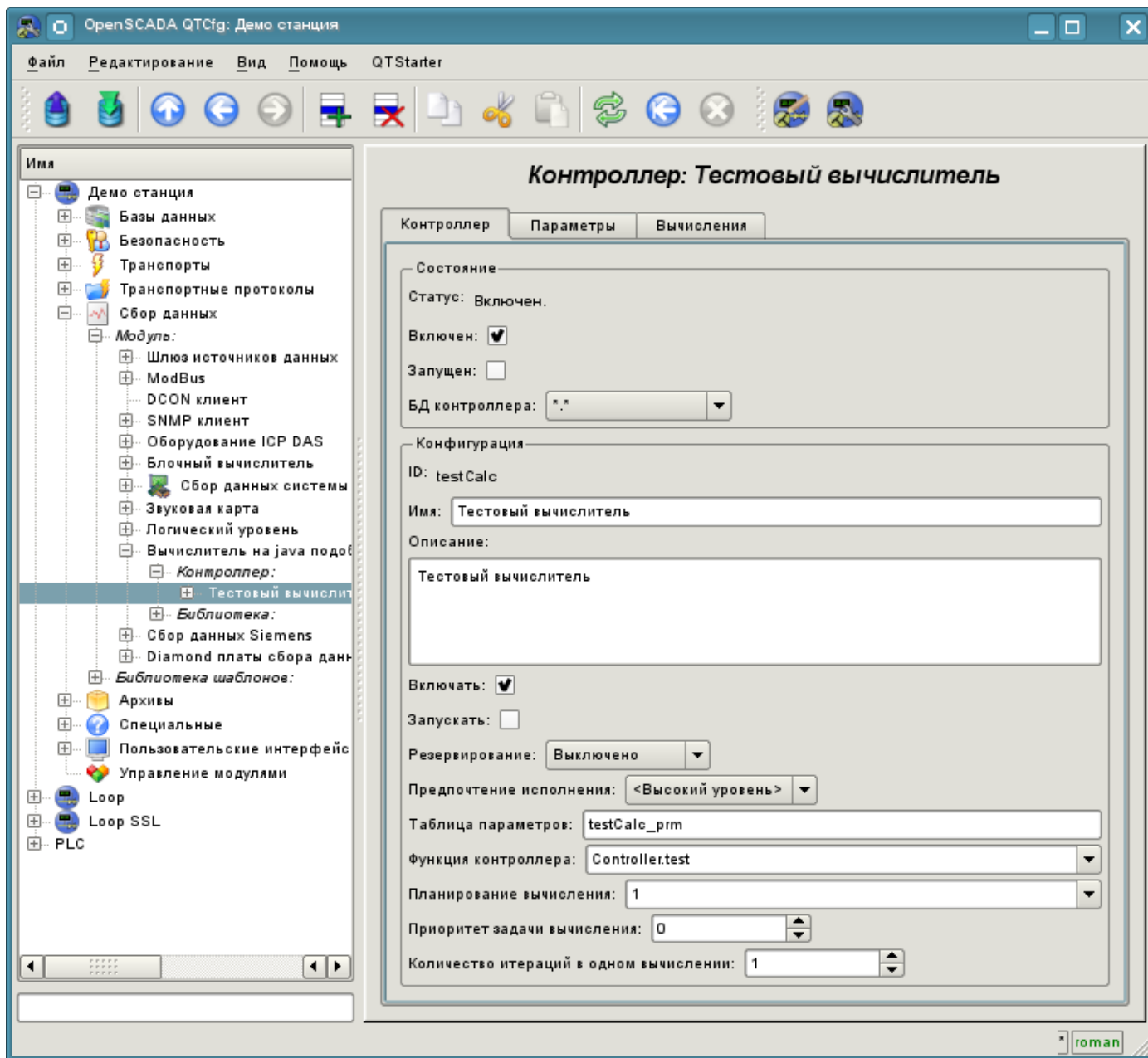


Рис.2. Вкладка конфигурации контроллера.

С помощью этой вкладки можно установить:

- Состояние контроллера, а именно: Статус, «Включен», «Запущен» и имя БД, содержащей конфигурацию.
- Идентификатор, имя и описание контроллера.
- Состояние, в которое переводить контроллер при загрузке: «Включен» и «Запущен».
- Режим горизонтального резервирования и предпочтение исполнения данного контроллера.
- Имя таблицы для хранения параметров.
- Адрес вычислительной функции.

- Период, приоритет и число итераций в одном цикле задачи вычисления.
- Период автоматической синхронизации блоков с БД.
- Сохранить/загрузить контроллер в БД.

Вкладка "Вычисления" контроллера(Рис. 3) содержит параметры и текст программы, непосредственно выполняемой контроллером. Модулем предусмотрен ряд специальных параметров, доступных в программе контроллера:

- *f\_freq* - Частота вычисления программы контроллера, только чтение.
- *f\_start* - Флаг первого выполнения программы контроллера, запуск, только чтение.
- *f\_stop* - Флаг последнего выполнения программы контроллера, останов, только чтение.
- *this* - Объект данного контроллера.

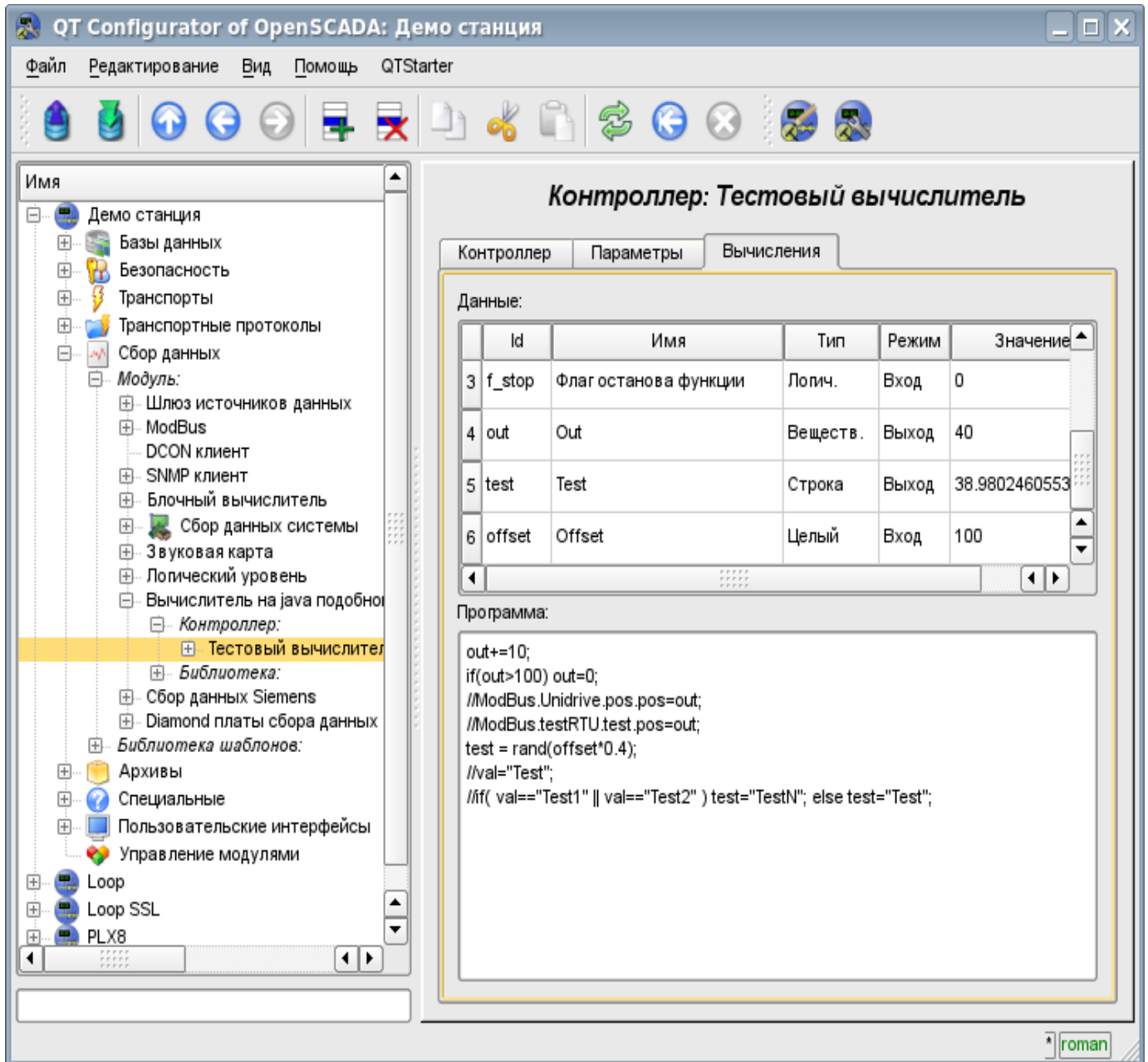


Рис.3. Вкладка «Вычисления» контроллера.

### **3. Параметр контроллера и его конфигурация**

Параметр контроллера данного модуля выполняет функцию предоставления доступа к результатам вычисления контроллера в систему OpenSCADA, посредством атрибутов параметров. Из специфических полей вкладка конфигурации параметра контроллера содержит только поле перечисления параметров вычисляемой функции, которые необходимо отразить.

## 4. Библиотеки функций модуля

Модуль предоставляет механизм для создания библиотек пользовательских функций на Java-подобном языке. Пример вкладки конфигурации библиотеки изображен на Рис.4. Вкладка содержит базовые поля: состояния, идентификатор, имя и описание, а также адрес таблицы, хранящей библиотеку. Во вкладке «Функции» библиотеки кроме перечня функций содержится форма копирования функций.

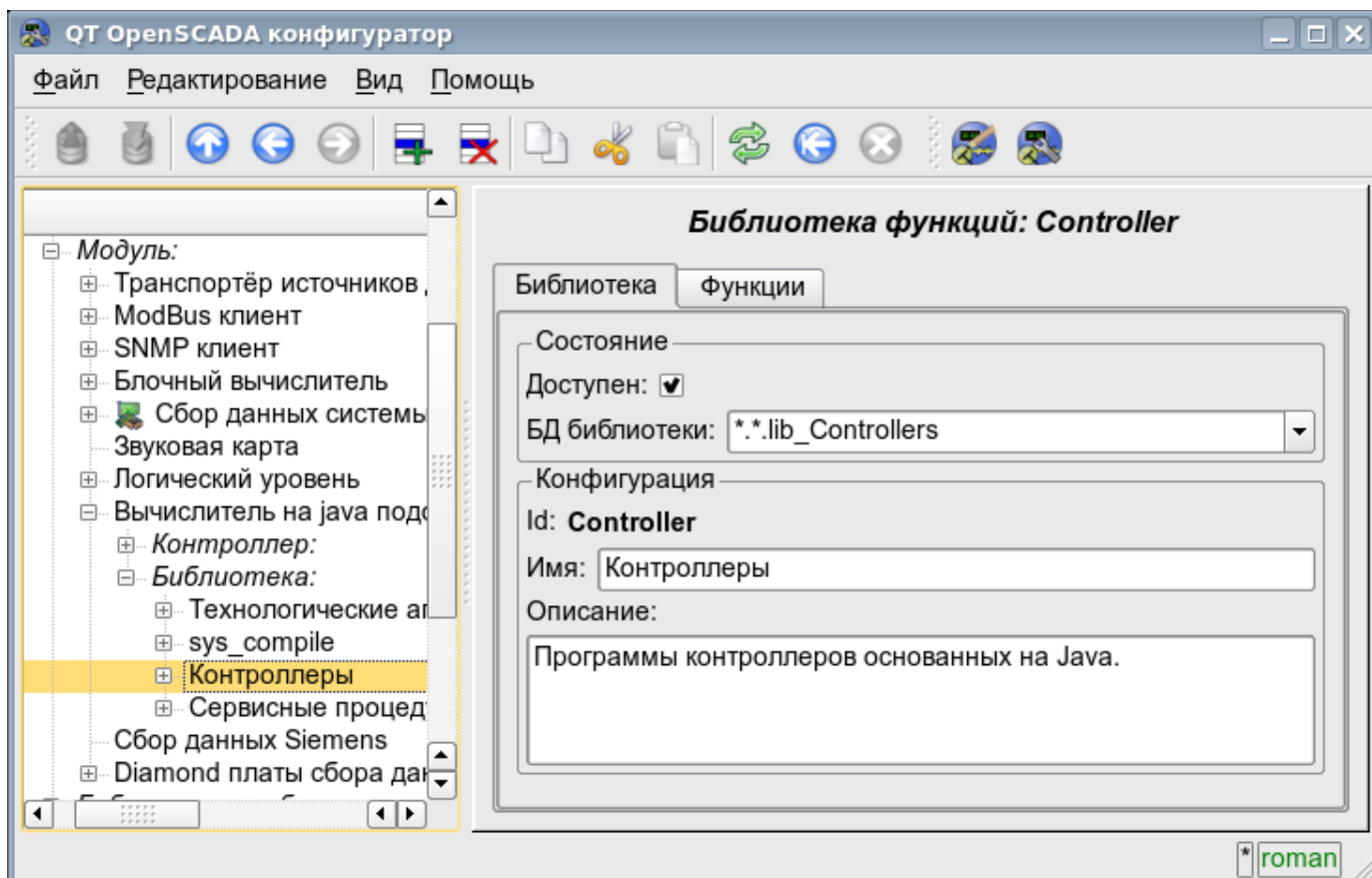


Рис.4. Вкладка конфигурации библиотеки.

## 5. Пользовательские функции модуля

Функция, также как и библиотека, содержит базовую вкладку конфигурации, вкладку формирования программы и параметров функции (Рис.1), а также вкладку исполнения созданной функции.