expression-analyzer 使用说明

作者单学成 Email：shanxuecheng@163.com QQ：502817600

本表达式解析器使用 java 语言实现。
支持算术运算、逻辑运算、比较运算；
支持 if 分支结构的解析；
支持在表达式中使用自定义函数；
支持多种数据类型：数字、日期、字符、字符串、布尔。

词法规则

数据类型

支持五种数据类型：数字、日期、布尔、字符、字符串

数字

数字类型在解析过程中将被转化成 BigDecimal 类型，支持以下格式：

整数 123；

小数 123.333；

指数 1.2E2 或 1.2e2（即 120）

日期

支持两种形式的日期数据，一种是精确到日的，写法为[yyyy-MM-dd]，如[2012-03-31]；另一种精确到秒，写法为[yyyy-MM-dd HH:mm:ss]，如[2012-03-31 09:06:10]。

在表达式解析过程中日期类型将转化为 java.util.Calendar。

Boolean

布尔常量的写法可以是 true、TRUE、false、FALSE。

字符

字符分普通字符和转义字符。

普通字符：’a’、’_’

转义字符：’\n’、’\r’

字符串

字符串用双引号表示，可包含转义字符，如”Hello World”、”Hello \r\n World”。

变量

变量名以字母或下划线开头，只能包含字母、数字、下划线，如：abc、_abc、_a1。

操作符

算术操作符：+、 -、 *、 /、 %（取余）；

逻辑操作符：&&、||、!

比较操作符：>、 >=、 <、 <=、 ==、 !=

赋值操作符：=

注释

在表达式中可以使用注释，注释以##开头，到行尾结束，不能跨行。
如：a = b * 100 + 1; ##注释

表达式使用

创建表达式

类 Expression 的实例表示一个表达式，可通过 Expression 的构造函数创建或者使用 ExpressionFactory 创建。

表达式由语句组成，每个语句必须由分号结尾。定义表达式时可包含一个语句或多个语句，解析结果是被执行的最后一个语句的结果。

Expression 有三个构造函数，可分别接受参数 Stirng、InputStream、Reader。下面以 String 为例创建表达式。

1. 使用构造函数创建

   Expression exp = new Expression(“a=1; a + a*100;”);

2. 使用 ExpressionFactory 创建

   ExpressionFactory factory = ExpressionFactory.getInstance();

   Expression exp = factory.getExpression("a=1; a + a*100");

执行表达式

通过调用 Expression 的 evaluate()方法执行表达式:

```
ExpressionFactory factory = ExpressionFactory.getInstance();
Expression exp = factory.getExpression("a=1;b=a*100;");
exp.lexicalAnalysis();//词法分析
Valuable result = exp.evaluate();//执行
```

注意：执行evaluate()的前提是已经执行词法分析，否则抛出异常。或者直接调用Expression的reParseAndEvaluate()方法，reParseAndEvaluate方法每次都重新执行词法分析。

表达式的执行结果以接口 Valuable 表示，Valuable 提供如下方法：

```
public DataType getDataType();
public int getIndex();
public BigDecimal getNumberValue();
public String getStringValue();
public Character getCharValue();
public Calendar getDateValue();
public Boolean getBooleanValue();
public Object getValue();
```

如果表达式的结果是数字，可以这样取值：

BigDecimal number = `result.getNumberValue();`

如果忽略执行结果的类型，可以通过 getValue 方法取得 Object()的执行结果。通过 getDataType 方法可以获得执行结果的具体类型。

以下部分将详细介绍各种表达式的具体使用。

算术表达式

算术表达式只支持数字类型的常量、变量或表达式，可使用括号运算符：

100 / (2 * 50);

100 / (50 + 50);

关于除法

在类 Expression 中有两个 public 静态域：

```
public static int DEFAULT_DIVISION_SCALE = 16;
public static RoundingMode DEFAULT_DIVISION_ROUNDING_MODE =
RoundingMode.HALF_UP;
```

这两个静态域只有在除法运算中才被使用，分别表示默认标度和默认的舍入模式，可以根据实际需要对这个两个静态域进行修改。

字符串连接

字符串和字符可以使用 + 连接。

例如，”Hello” + “World” + ‘!’;

逻辑表达式

逻辑表达式只支持布尔类型的常量、变量或表达式，可使用括号运算符：

(2>1 || false) && (false || 1<2);

比较运算

支持数字、字符、字符串、日期类型的比较，不支持布尔类型的比较。

“zxc” > “abc”;

‘b’ >= ‘a’;

1+1 < 1;

[2012-03-31] > [2012-03-01];

赋值语句

第一次给变量赋值时确定了变量的类型，后续语句中不能给变量赋以不同的类型，否则抛出 ArgumentsMismatchException 异常。

例执行语句 a = “Hello Word”; a=1; 将会抛出异常：

Type mismatch in assignment: cannot convert from String to Number.

函数

系统函数

表达式解析器目前只提供了三个系统函数：abs（取绝对值）、max（取最大值）和 judge（判断）。

函数的调用支持嵌套：

Expression expression = factory.getExpression("1 + max(1,abs(-2)) + abs(-1);");

judge 函数共接受三个参数，第一个参数为布尔类型，第二和第三个参数为任意类型，例如 judge(2>1, “2”, “1”)将返回字符串”2”。

函数参数可以是固定个数的参数，如 judge，也可以是任意多个参数，如 abs 和 max

自定义函数

自定义函数需要继承抽象类 Function，有四个方法需要实现：getName()、getArgumentNum()、getArgumentsDataType()、executeFunction()。

getName 返回函数名，函数名不能为空。

getArgumentNum 返回参数个数，当返回值小于 0 时，表示参数个数不限。

getArgumentsDataType 返回函数参数类型数组。当参数个数不限时，所有参数类型必须相同，本方法须提供一个参数类型。

executeFunction 接收参数数组，实现函数执行逻辑，并返回结果值。下面介绍如何自定义无参函数、固定参数函数和任意数量参数的函数。

1） 无参函数

定义无参函数时，getArgumentNum()返回 0，且构造函数中无需传入参数类型数组。比

如定义一个函数取得当前日期，函数名为 getDate：

```
public class CurrentDate extends Function {
    @Override
    public String getName() {
        return "getDate";
    }
    @Override
    public int getArgumentNum() {
        return 0;
    }
    @Override
    public DataType[] getArgumentsDataType() {
        return null;
    }
    @Override
    protected Object executeFunction(Valuable[] arguments) {
        Calendar date = Calendar.getInstance();
        date.setTime(new Date());
        return date;
    }
}
```

2） 固定参数函数

以上面提到的 judge 函数为例：

```
public class Judge extends Function {
    @Override
    public int getArgumentNum() {
        return 3;
    }
    @Override
    protected Object executeFunction(Valuable[] arguments) {
        boolean condition = arguments[0].getBooleanValue();
        if(condition)
            return arguments[1].getValue();
        else
            return arguments[2].getValue();
    }
    @Override
    public String getName() {
        return "judge";
    }
    @Override
    public DataType[] getArgumentsDataType() {
```

```
            return new DataType[]{DataType.BOOLEAN, DataType.ANY,
            DataType.ANY};
    }
}
```

getArgumentNum()返回 3，构造函数中 DataType.ANY 表示参数为任意类型

3） 任意数量参数的函数

以上面提到的 max 函数为例：

```
public class Max extends Function {
    @Override
    public int getArgumentNum() {
        return -1;
    }
    @Override
    public Object executeFunction(Valuable[] arguments) {
        BigDecimal result;
        if(arguments.length == 0) {
            result = new BigDecimal("0");
        } else {
            result = arguments[0].getNumberValue();
            for(int i=1; i<arguments.length; i++)
                if(result.compareTo(arguments[i].getNumberValue()) < 0)
                    result = arguments[i].getNumberValue();
        }
        return result;
    }
    @Override
    public String getName() {
        return "max";
    }
    @Override
    public DataType[] getArgumentsDataType() {
        return new DataType[]{DataType.NUMBER};
    }
}
```

getArgumentNum()返回-1，表示参数为任意个数。对于任意数量参数的函数，参数必须为同一类型，所以在构造函数中传入的参数类型数组为：new DataType[]{DataType.NUMBER}。

自定义函数注册

系统函数不需要注册，可以直接使用。使用自定义函数，必须先调用 Expression 的 addFunction 方法，将函数注册到表达式，才能使用。以上面的取当前日期函数为例，

```
Expression expression = factory.getExpression("getDate();");
expression.addFunction(new CurrentDate());
expression. reParseAndEvaluate ();
```

关于函数名

本表达式解析器不支持函数名与变量名相同，如果函数名与变量名相同，将优先被解析为函数名。

If 语句

If 语句的使用形式为：

```
if(Boolean)        endif    或
if(Boolean)        else     endif
```

对于在分支内部定义的变量，其作用域仅为所在分支，在后续语句中不能直接使用，否则抛出变量为定义异常。

If 语句的执行结果是最后一个被有效执行的语句，如：

```
if(2>1)
    a=abs(-3);
    if(false)
        a=a+1;
    else
        a=a+2;
    endif
    a=a+1;
else
    if(2>1)
        a=9;
    endif
endif
```

执行结果为 6。

获得表达式中涉及的所有变量名

Expression 提供函数 getVariableNames()，以 Set<String>返回表达式中的所有变量名。

但调用此方法的前提是已经执行词法分析，否则抛出异常。

获取变量值

表达式执行完成后，可以通过 getVariableValueAfterEvaluate(String)取得变量的值，传入参数为变量名；还可以调用 getAllVariableValueAfterEvaluate()取得所有变量的值，返回类型为 Map<String, Valuable>。

设置变量值

表达式中某变量的初始值也可从外部设置，调用 initVariable(String, Object);
如：

```
Expression expression = factory.getExpression("a + 1;");
expression. initVariable ("a", 1);
expression. reParseAndEvaluate ();       //结果为 2
```