IKExpression 简易表达式解析器

使用说明

目录

1. IK 表达式介绍（IK Expression Introduction）

IK Expression 是一个开源的（OpenSource），可扩展的（Extensible），基于 java 语言开发的一个超轻量级（Super lightweight）的公式化语言解析执行工具包。

IK Expression 不依赖于任何第三方的 java 库。它做为一个简单的 jar，可以集成于任意的 Java 应用中。这包括了 JavaEE 应用（基于应用服务器的），Java 桌面应用以及 Java WebStart 方式的应用。

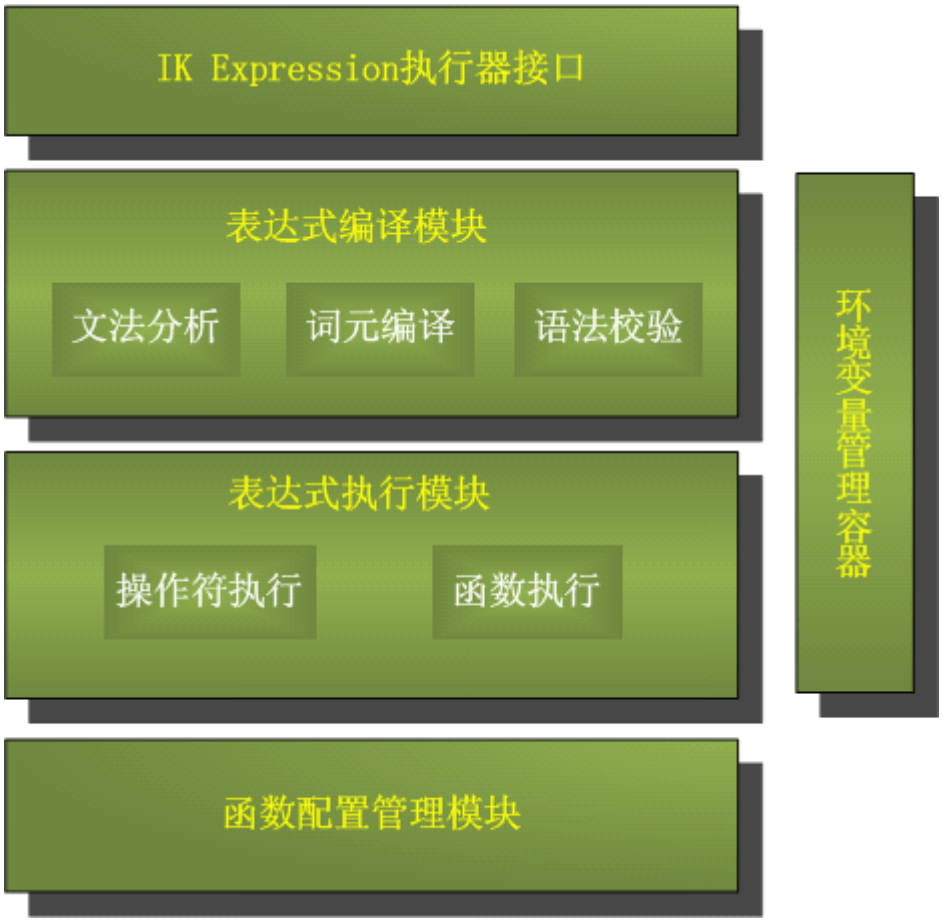
IK Expression 最初诞生的原因是为了能增强工作流引擎，如 jBPM 等对流程配置的灵活度。使其能在流程运行期获得同配置期一样灵活地对执行逻辑条件进行变更。经过扩展后的 IK Expression 还可以适用于各种常规业务系统的动态条件配置，如需要图形化配置应用的场合，或是模拟 Excel 电子表格的公式运算的场景。

同 EL 和 BeanScript 不同，IK Expression 的设计目标是面向最终用户的，因此它被设计成语法简单（像数学算式），通俗易懂（支持中文变量及函数名）但功能有限的解析引擎。如果你需要一个功能强大的表达式引擎，也许 IK Expression 并不是最好的选择。

1.1 概要（OverView）

IK Expression 是一个采用逆波兰式算法结合指针栈优化的公式解析引擎，它由表达式编译、、表达式执行、变量容器、以及函数配置管理四部分构成。它具有以下特点：

- 支持基础运算符+ - × / % 逻辑运算符！ && || 三元运算符？：以及特有的#集合运算。支持括号优先级，对&&，||，？：有短路优化处理。
- 支持函数执行，函数扩展，支持变量定义。
- 完整 Jar 包大小 90K，API 简单易学，超轻量级，无第三方类库依赖。



IK Expression 组件结构图

1.2 升级变更（Changes Log）

Version2.1 的变更：

- 新增 org.wltea.expression.ExpressionEvaluator. preparedCompile（String expression， Collection<Variable> variables）方法和类 org.wltea.expression.PreparedExpression 用来支持表达式预编译。

用户可以像处理 SQL 语句那样，对需要循环执行的相同表达式进行预编译处理，而后设置不同的执行参数，得到不同的计算结果。详见“样例代码”和”API 说明”.

- 修订了对声明为 Object 类型的参数传入 Integer、Double、String 等类型的值时，会抛出类型不匹配异常的问题。原来版本中，参数必须显式申明为 DATATYPE_OBJECT 类型，新版本则不需要，所有 java 类型默认继承于 Object。
- 修改了 ExpressionEvaluator 的两个 compile 方法，不再抛出 IllegalExpressionException 异常，而改为运行期异常，以简化用户编码。

Version2.0.5 的变更：

- 支持表达式的换行编辑，方便表达式编写和阅读，如：

```
$CALCDATE([2008-03-01],
          0,0,
          -(10 + (23 - 3) * (4 / 5)) % 6,
          0,0,0)
```

- 提供了更多的开放 API，如：表达式验证 API，这使得表达式在执行前可以检查其语法正确性。
- 修订了“？：”三元操作符的“:“号两端对不同类型的数值不兼容的 BUG。如：原有版本中，表达”100 > 10？100：10.5“将会抛出“参数类型不兼容“异常，在新版本中，则返回 double 型的 100.0

Version2.0.2 的变更：

- 修订了自定义函数返回类型为 void 时，表达式校验会抛出异常的 BUG。

对于 void 返回类型，IK-Expression 将当作 null 类型进行计算。

Version2.0.1 的变更：

- 添加了对表达式的折行书写的支持（即对表达式中出现 CRLF 的支持），方便于用户编写逻辑复杂的长表达式，增强表达式可阅读性。

Version2.0 的变更：

- 增加了“？：“三元操作符。
- 增强了“+“操作符，支持对 null 型，Date 型，Boolean 型的连接操作。
- 增加了自定义函数扩展，支持配置方式扩展函数和 API 编码方式扩展函数。
- 增加了函数别名映射功能，可配置中文函数别名。
- 增加了函数参数和返回值对 java Object 类型的支持。
- 优化“||”、“&&”、“？：“操作符，实现相应逻辑的短路处理。
- 修订了 Date 类型的==比较逻辑。原有逻辑判定时间差小于 1 秒为相等，现在更正为精确到秒。
- 修订了==比较逻辑对数值类型运算的 bug，使其支持 Integer，Float，Long，Double 类型的混合比较，以及当这四种类型混合比较时，其中一个参数为 null 的情况。
- 修改了 V1.0 中类 Variable 的方法 createVariable 的参数。
- 废除了 V1.0 中的“：“操作符（使用“？：“三元操作符代替）
- 废除了 V1.0 中类 ExpressionEvaluator 的 compileExpression 方法。

2. 快速入门（Quick Start）

2.1 下载（Downloadables Overview）

GoogleCode 开源项目 ：http://code.google.com/p/ik-expression/

GoogleCode SVN 下载：http://ik-expression.googlecode.com/svn/trunk/

2.2 安装部署

IK Expression 的安装部署十分简单，安装包包含：

1. 使用说明文档（即本文档）
2. IKExpression2.*X.Y*.jar（*X.Y* 为子版本号）
3. IKExpression.cfg.xml

其中，IKExpression2.*X.Y*.jar 部署于项目的 lib 目录中；IKExpression.cfg.xml（函数定义配置）文件放置在代码根目录（对于 web 项目，通常是 WEB-INF/classes 目录，同 hibernate、log4j 等配置文件相同）下即可。

2.3 API 简易教程（API Tutorial）

代码样例

HelloWorld

```
/**
 * Hello World Example
 * @param args
 */
public static void main(String[] args){
    if(args.length == 0){
        args = new String[1];
        args[0] = "IK Expression";
```

```
    }
    //定义表达式
    String expression = "\"Hello World \" + 用户名";
    //给表达式中的变量“用户名”付上下文的值
    List<Variable> variables = new ArrayList<Variable>();
    variables.add(Variable.createVariable("用户名", args[0]));
    //执行表达式
    Object result = ExpressionEvaluator.evaluate(expression,
variables);
    System.out.println("Result = " + result);
}
```

执行结果： Result = Hello World IK Expression

预编译执行样例

```
/**
 * Hello World Example
 * @param args
 */
public static void main(String[] args){
    if(args.length == 0){
        args = new String[1];
        args[0] = "IK Expression V2.0.5";
    }
    //定义表达式
    String expression = "\"Hello \" + 版本";
    //给表达式中的变量［版本］付上下文的值
    List<Variable> variables = new ArrayList<Variable>();
    variables.add(Variable.createVariable("版本", args[0]));

    //预编译表达式
    PreparedExpression pe  =
ExpressionEvaluator.preparedCompile(expression, variables);
    //执行表达式
    Object result = pe.execute();
    System.out.println("Result = " + result);

    //更改参数，再次执行预编译式
    pe.setArgument("版本", "IK Expression V2.1.0");
    result = pe.execute();
    System.out.println("Result = " + result);

}
```

执行结果：

Result = Hello IK Expression V2.0.5

Result = Hello IK Expression V2.1.0

API 说明

- 类 org.wltea.expression.ExpressionEvaluator

  该类是 IK-Expression 主调用类，大部分执行方法都在这里。

方法 1：

public static Object evaluate(String expression, Collection<Variable> variables)

说明：传入表达式和表达式上下文的变量，执行表达式返回结果

参数 1 ：String expression, 要传入执行的表达式

参数 2 ：Collection<Variable> variables 表达式上下文的变量集合（详细请看类 org.wltea.expression.datameta.Variable 的说明）。

返回值：表达式执行结果，可能是以下类型的 java 对象中的一种：

Int、Long、Float、Double、Boolean、String、Date、List、Object。

方法 2：

public static Object evaluate(String expression)

说明：对方法 1 的重载，执行简单的没有变量的表达式。请参考方法 1 说明.

方法 3：

public static String compile(String expression, Collection<Variable> variables)

说明 :传入表达式和表达式上下文的变量 ,对表达式进行语法编译测试 ,返回逆波兰式表示 。

参数 1 ：String expression, 要传入执行的表达式

参数 2 ：Collection<Variable> variables 表达式上下文的变量集合（详细请看类 org.wltea.expression.datameta.Variable 的说明)。

返回值：表达式的逆波兰式表示

方法 4：

public static String compile(String expression)

说明：对方法 3 的重载，编译简单的没有变量的表达式。请参考方法 3 说明.

方法 5：

public static PreparedExpression preparedCompile(String expression , Collection<Variable> variables)

说明：传入表达式和表达式上下文的变量，对表达式进行预编译处理，返回预编译完的表达式对象。

参数 1 ：String expression, 要传入执行的表达式

参数 2 ：Collection<Variable> variables 表达式上下文的变量集合（详细请看类 org.wltea.expression.datameta.Variable 的说明)。

返回值：PreparedExpression 预编译完的表达式对象。

- 类 org.wltea.expression.PreparedExpression

  该类表示 IK-Expression 的预编译表达式。

方法 1：

public synchronized void setArgument(String name , Object value)

说明 :设定预编译表达式的执行时的参数值。如果设定的参数不在预编译处理的参数集合中，则抛出 IllegalArgumentException 异常。

参数 1 ：String name, 参数名，可以是中文字符

参数 2 ：Object value，参数的值，可以是下类型的 java 对象中的一种：

Int、Long、Float、Double、Boolean、String、Date、List、Object

方法 2：

public Object execute()

说明：执行预编译表达式

返回值：表达式执行结果，可能是以下类型的 java 对象中的一种：

Int、Long、Float、Double、Boolean、String、Date、List、Object。

- 类 org.wltea.expression.datameta.Variable

该类是用来表示表达式的上下文变量的，上面的例子中用到了别名为“用户名”的上下文变量，这是也是表达式最有用的地方。例如，在 jBPM 的流程定义中，我们需要定义一个报销审批流程中，用来决定流程分支走向的表达式： （申请金额 > 10000）？“总经理审批”：“部门经理审批”

这里需要定义一个别名为 “申请金额 “变量。变量通过 evaluate(String expression, Collection<Variable> variables)方法中的 variables 参数传入表达式中。而 Variable 类型变量的构造十分的简单，它是标准的 POJO。

方法 1：

public static Variable createVariable(String varName , Object varValue)

说明：根据参数别名和参数值，构造 Variable 实例

参数 1 ：String varName, 参数的别名，可以是中文别名

参数 2 ：Object varValue，参数的值， 可以是下类型的 java 对象中的一种：

Int、Long、Float、Double、Boolean、String、Date、List、Object。

返回值：org.wltea.expression.datameta.Variable 类的实例

方法 2：(直接使用构造函数)

public Variable(String varName , DataType varDataType , Object varValue)

说明：根据指定的参数类型、参数别名和参数值，构造 Variable 实例

参数 1 ：String varName, 参数的别名，可以是中文别名

参数 2 ：DataType varDataType, 变量类型，它是

org.wltea.expression.datameta. BaseDataMeta.DataType 枚举类型，包括的枚举值有:

```
//NULL类型      DATATYPE_NULL ,
//字符窜        DATATYPE_STRING ,
//布尔类        DATATYPE_BOOLEAN ,
//整型数        DATATYPE_INT ,
//长整型数      DATATYPE_LONG ,
//浮点数        DATATYPE_FLOAT ,
//双精度浮点        DATATYPE_DOUBLE ,
//日期时间      DATATYPE_DATE ,
//集合对象      DATATYPE_LIST,
//通用对象类型  DATATYPE_OBJECT,
```

参数 3 ：Object varValue，参数的值， 可以是下类型的 java 对象中的一种：

Int、Long、Float、Double、Boolean、String、Date、List、Object。

返回值：org.wltea.expression.datameta.Variable 类的实例

3. 表达式公式规范（Expression Formula Specification）

3.1 数据类型（Types, Values, and Variables）

a) 数字型 ：

i. 整形 integer ： -2321 ， 34234

ii. 长整型 long ：3245235235L

iii. 单精度浮点 float ： 342.555F

iv. 双精度浮点 double： 234234.3423

b) 字符型： “a-zA-Z012456789”

c) 布尔型：true、false

d) 日期时间型：[2008-08-08] 或 [2009-01-01 12:33:14]

e) 扩展类型：List 对象集合 *（该类型不支持表达式字面定义，由操作符或函数运算结果生成）*

f) 通用对象：Object 类型

3.2 运算符（Operators）

运算符描述	符号	操作数个数	补充说明
逻辑取反	!	一元	对布尔型取反运算
取负（负号）	-	一元	对所有数值型参数取负值
算术乘	*	二元	支持数值型计算
算术除	/	二元	支持数值型计算

取余（模）	%	二元	支持数值型计算
加	+	二元	● 支持数值型计算 ● 支持字符窜连接 ● 所有类型的参数同字符串相加都转成字符串 ● 遇到 null 型同字符串相加，忽略 null
算术减	-	二元	支持数值型计算
小于	<	二元	支持数值型、日期型、字符型比较
小等于	<=	二元	支持数值型、日期型、字符型比较
大于	>	二元	支持数值型、日期型、字符型比较
大等于	>=	二元	支持数值型、日期型、字符型比较
逻辑等	==	二元	● 支持数值型、日期型、字符型、Object 型比较 （注，对日期时间类型比较，误差精确到秒。对 Object 类型则依靠它的 equals 实现） ● 支持 Null 同数值型、日期型、字符型、Object 型比较

逻辑不等	！=	二元	• 支持数值型、日期型、字符型比较 • 支持 Null 同数值型、日期型、字符型比较
逻辑与	&&	二元	布尔型运算
逻辑或	||	二元	布尔型运算
~~取值~~	~~:~~	~~三元~~	Version 2.0 中不再支持：运算符， 使用？：三元运算符代替
选择	？：	三元	用法说明：等同于 if ..then.. else..如： (申请金额>10000)?“总经理审批”：“部门经理审批” 当 （申请金额 > 10000）为 true 时，返回“总经理审批”字符串；否则，返回部门经理审批”。 这里返回值可以是任意类型。
集合添加	#	二元	用法说明：有点类似 List 的 add 方法。如： “直接上级”# “部门经理”#“总经理” 这将返回含有 3 个字符串的 List 集合

3.3 分割符（Separators）

a) 括号 "(" ")" —— 标识优先级

b) 逗号 "," —— 分隔函数的参数

c) 方括号 "[" "]" —— 标识日期型常量

d) 双引号 """ —— 标识字符型常量

e) 美元号 "$" —— 函数标识前缀

f) 转义符 "\" —— 字符串转义，支持\\ , \” , \r , \n , \t

3.4 内部函数（Inner Functions）

内置函数是目前解析器已经实现的一些非常简单、实用的函数

函数名	参数	返回值类型	说明
$CONTAINS	String ,String	Boolean	字符串包含比较，同 java 的 String.contians()
$STARTSWITH	String, String	Boolean	字符串前缀比较，同 java 的 String.startsWith()
$ENDSWITH	String, String	Boolean	字符串后缀比较同 java 的 String.endsWith()
$CALCDATE	Date , int , int ,int ,int ,int, int	Date	日期计算函数 Date 是原始日期值； 依次排序的 int 参数，分别代表：年、月、日、时、分、秒。传入正数表示向后计算，负数表示向前计算。 如: $CALCDATE([2008-08-08] , 2 ,

			0 , 0 , 0 , 0 , 0)等于[2010-08-08] $CALCDATE([2008-08-08 12:30:00] , 0 , 0 , 0 , -8 , -15 , 0)等于[2008-08-08 04:15:00];
$SYSDATE	无	Date	当前系统日期（时间）,同 java 的 new Date()
$DAYEQUALS	Date , Date	Boolean	日期相等比较 [2008-08-08 00:00:01]与[2008-08-08 23:59:59]是同一天，将返回 true

3.5 语法约束（Lexical Structure）

- 变量命名遵循 java 变量命名规范（如，不能以数字打头，不能用系统操作符打头等）。
- 函数声明以“$”符号打头，自定义函数命名遵循 java 方法命名规范。
- 日期型常量使用“[]”符号界定，格式为 [yyyy-MM-dd 24h-mm-ss]，不支持毫秒。
- 用户自定义函数别名不能重复。（详细请参阅本文 4.1 章节）
- 用户自定义函数的参数和返回值类型限定于 3.1 章节描述的数据类型。（详细请参阅本文 4.1 章节）

3.6 公式样例（Formula Example）

1. +、-、*、/、%（取模） 常规的算术运算，支持括号优先级 ：

   如：常见的 OA 中用于年休假工资计算公式

   3000 / 21.5 *（12 - 转正月份）/ 2

   其中，“转正月份”可以是上下文变量

2. 不同数据类型的字符串连接 ：

   “ABC” +（123+10） 运算结果 “ABC133”

   “ABC” + 123 + 10 运算结果“ABC12310”

   “[2009-08-08] + false + 123 + \"a String\" + null”

   运算结果 “2009-08-08 00:00:00false123a String” （PS:忽略 null 型变量）

3. > >= < <= == !=逻辑比较运算，返回布尔值 ：

   3-1.数值大小比较：1234>223 运算结果 true

   3-2.字符大小比较： “1234” > “223” 运算结果 fasle

   3-3.日期大小比较：[2008-12-23] >= [2008-08-08] —— true

   3-4 同 null 的 == 与 != 比较 ： 申请人!=null (其中，“申请人”为执行上下文的变量)

4. 逻辑与、逻辑或、逻辑非运算：

   true && $DAYEQUALS([2008-01-01] , [2008-11-01]) ——false

   true || $DAYEQUALS([2008-01-01] , [2008-11-01]) —— true

   true && ! $DAYEQUALS([2008-01-01] , [2008-11-01]) —— true

5. 结果连接运算“#”：

1000/10 # [2008-12-23]>$SYSDATE() # “ABC” +123

运算结果 包含 100 ， false ， “ABC123”三种不同类型对象的 List

6. 函数与操作符混合、嵌套调用，如：

```
$DAYEQUALS(
    $CALCDATE(
        $SYSDATE() , 0 , 0 ,
        (8+11-5*(6/3)) * (2- 59 % 7) ,
        0 ,0,0 ),
    [2009-10-01]
)
```

运算结果 false

4. 高级特性（Advance）

4.1 函数定制（Functions Customize）

IK-Expression 最吸引人的特性莫过于它允许你以非常简单的方式扩展你的自定义函数。IK-Expression 带有一个 xml 配置文件 *IKExpression.cfg.xml*。在使用 IK-Expression 时，该配置文件应放置于 class 的根目录中(如同 spring 和 hibernate 等的配置文件一样)。配置文件内部格式如下：

IKExpression.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<function-configuration>
    <!-- 系统函数默认配置 -->
    <bean class="org.wltea.expression.function.SystemFunctions">
        <function name="CONTAINS" method="contains">
            <parameter-type>java.lang.String</parameter-type>
            <parameter-type>java.lang.String</parameter-type>
        </function>
        <function name="STARTSWITH" method="startsWith">
            <parameter-type>java.lang.String</parameter-type>
            <parameter-type>java.lang.String</parameter-type>
        </function>
        <function name="ENDSWITH" method="endsWith">
            <parameter-type>java.lang.String</parameter-type>
            <parameter-type>java.lang.String</parameter-type>
        </function>
        <function name="CALCDATE" method="calcDate">
            <parameter-type>java.util.Date</parameter-type>
            <parameter-type>int</parameter-type>
            <parameter-type>int</parameter-type>
            <parameter-type>int</parameter-type>
            <parameter-type>int</parameter-type>
            <parameter-type>int</parameter-type>
            <parameter-type>int</parameter-type>
        </function>
        <function name="SYSDATE" method="sysDate" />
        <function name="DAYEQUALS" method="dayEquals">
```

```
            <parameter-type>java.util.Date</parameter-type>
            <parameter-type>java.util.Date</parameter-type>
        </function>
    </bean>

    <!-- 用户函数配置  ，请在这里定制您自己的函数-->

</function-configuration>
```

配置文件中默认配置了系统内部函数定义（在没有绝对必要的原因下，不建议修改系统默认函数配置）。在默认配置的下方，用户可以定义自己的函数，格式如下：

用户自定义函数配置

```
<bean class="org.wltea.expression.test.TestFunctions">
  <constructor-args>
    <constructor-arg type="java.lang.Integer">123</constructor-arg>
    <constructor-arg type="java.lang.String">aa</constructor-arg>
  </constructor-args>
  <function name="问好" method="sayHello">
    <parameter-type>java.lang.String</parameter-type>
  </function>
</bean>
```

这里自定义了一个名称为“问好”的函数，它有一个 String 类型的参数。该函数映射对应于 org.wltea.expression.test.TestFunctions 类的 sayHello 方法，而类 org.wltea.expression.test.TestFunctions 具有一个构造函数，构造函数带有 Integer 型和 String 型的参数。配置中给出了构造函数的初始化参数“123”和“aa”。通过上述定义，用户就可以在表达式中使用该函数，如:

$问好（当前用户），其中“当前用户”为表达式的上下文变量。

上述例子直观的展示了用户函数自定义的过程。下面 ,我们将系统的了解一下 IK-Expression 的函数扩展定义规则和约束：

1. 在 IK-Expression 中 ,函数直接对应于 java 的一个类的一个明确的方法。如：

$CONTAINS对应org.wltea.expression.function.SystemFunctions类的contains方法；

2. 所有的函数定义前，必须先定义对应的 java 类。如果该类使用带参数的构造函数，则必须提供明确的构造参数，如：

```
<bean class="org.wltea.expression.test.TestFunctions">
<constructor-args>
<constructor-arg type="java.lang.Integer">123</constructor-arg>
<constructor-arg type="java.lang.String">aa</constructor-arg>
</constructor-args>
   <function name="问好" method="sayHello">
      <parameter-type>java.lang.String</parameter-type>
   </function>
</bean>
```

3. java 类的加载和实例化在初始化阶段一次性完成。目前 IK-Expression 仅支持单例形式的加载，即对一个 java 类仅实例化一次。

4. 定义函数时，必须明确定义函数对应 java 方法，以及 java 方法的参数类型和顺序，如：

```
<function name="CONTAINS" method="contains">
      <parameter-type>java.lang.String</parameter-type>
      <parameter-type>java.lang.String</parameter-type>
</function>
```

你可以使用不同的函数名（英文的和中文的），对应相同的 java 方法，但对 java 中的方法重载必须使用不同的函数名对应(即，IK-Expression 不支持函数名重载)

5. 函数的参数和返回值只能是 IK-Expression 支持的数据类型(请参考 3.1 数据类型 章节).

6. 为了增加灵活性，IK-Expression 给出了一个通过编码方式添加自定义函数的 static 方法。

函数扩展 API 说明

- 类 org.wltea.expression.function.FunctionLoader

方法 1：

public static void addFunction(String functionName, Object instance, Method method)

参数 1 ：String functionName, 要定义的函数名称（中英文皆可）。

参数 2 ：Object instance 函数要映射的 java 类的实例。

参数 3 ：Method method 函数要映射的 java 类的方法对象。

（全文终）