Value-Expression 值表达式

作者	时间	版本	描述
黄杰添	2022年6月20日	V1.0	初始版本
黄杰添	2022年10月8日	V1.1	增加Enums表达式

值表达式的设计的初衷是为了支持物联网协议的报文解析/生成，是报文(Source) 和 值(Target)之间的解析层。最初基于水文协议的数据定义(eg. N(3.1)), VE表达式操作字符串数据源， 并最终解析成定义的泛型的值和值字符视图。

(string)003821 **--->** N(6,2) **--->** (double)38.21

1. VE表达式 格式说明

基本格式 `Type(Format)` 由一个Type字符起始, 紧跟着括号内的Format表示Type的补充说明，根据Type字符的不同而不同

- Type 表示为该要素的数据类型 eg N表示BCD编码数字，S表示字符， B表示字节，T表示时间，A表示数组
- Format 表示该数据类型的补充定义，如 当Type=N Format可以是一个大于0的数字，如3.1 完整的 `N(3,1)` 表示总长3个字节的数字，小数点1位。更多示例参考 附录 B

2. VE表达式 Type列表

Type 标识符	类型说明	编解码	占位符支持	基本格式	完整示例	示例说明
N	数值 Number		不支持	N(字符数[,小数位])	N(6,1)	6字符的数字 例如 016751 将翻译为 1675.1 小数位如不声明或小于0则为0
S	字符 String		支持非数字占位符	S(字符数)	S(16) S(4-2-2 2:2:2)	16个字符的字符 14个字符的字符, 例如 20220617104300 会被解释为 2022-06-17 10:43:00
HN	HEX数值 Number	HEX	不支持	HN(字符数[,小数位])	HN(6,2)	6字符的HEX数字 例如 016751 将翻译为 919.85 小数位如不声明或小于0则为0
HS	Hex字符 String	HEX	支持非数字占位符	HS(字符数)	HS(48) HS(4-2-2 2:2:2)	24个字节的Hex字符 14个字节的HEX字符, 例如 32303232303631373130313630 30 会被解释为 2022-06-17 10:43:00
HF	HEX-FLOAT 数值 Float	IEEE 754	不支持	HF(字符数)	HF(8)	8个字符的HEX-FLOAT字符，例如 3F99999A 经IEEE754解码为 1.2
E	枚举值 String		不支持	E(字符数,键-值...)	E(2,01-成功,02-失败)	2个字符的枚举值,例如01翻译为成功, 02翻译为失败. 注意以 - 号区分, 如后续还有 - 号仅仅取第一个 - 号
T	时间字符 Timestamp			T(字符数,解析格式[,展示格式])	T(12,yyMMddHHmmss,yyyy-MM-dd HH:mm:ss)	12个字符的字符，例如220711121212 按照 yyMMddHHmmss解析后展示为yyyy-MM-dd HH:mm:ss格式,最终结果为2022-07-11 12:12:12 展示格式如不声明则为解析格式
B	字节 byte[]	不固定	不支持	B(字符数[,编码格式])	B(1024,HEX)	1024个字符转换为HEX编码 = 512个字节，编码格式如不声明默认为UTF-8
A	数组 Object[]	不固定	视数组元素类型而定	A(Type(Format))	A(HN(2),12)	共24个字节的Hex数字, 拆分为12组，默认在数据解析完成后以逗号 ‘，’间隔
C	组合 List	不固定	视数组元素类型而定	C(Type1(Format),Type2(Format)...)	C(T(12,yyMMddHHmmss),N(6,2),S(10))	共12+6+10=28个字符的组合字符,解析为: 1. 时间数据 2. 数值数据 3. 字符数据 例如 220711165900004875012345678 9 拆解为 1. 220711165900 2. 48.75 3. 0123456789 结果展示为:220711165900,48.75,0123456789
E	枚举 String		不支持	E(字符数,值-表示...)	E(2,01-成功,02-失败)	共2字符，01时解析为成功，02时解释为失败

3. 书写说明

3.1 格式书写要求

除非表达式支持占位符(eg S(4-2-2 2:2:2))或者格式(eg T), 否则不允许额外的空格, 表达式间隔符均使用 , 号

3.2 关于不定长数据的声明

字符(S),HEX字符(HS),字节数组(Bytes) 三个元素支持不定长数据的声明，声明方式为将 字符数 置为 0或者负数 , 例如 `B(0,HEX)` 表示所有字符都为字节数组的值

4. Java开发者指南

添加依赖

```
    <dependency>
        <groupId>org.fourfaith</groupId>
        <artifactId>fourfaith-boot-utils</artifactId>
        <version>3.0.0-SNAPSHOT</version>
    </dependency>
```

4.1 Java组件

组件	组件说明
VExpFactory	表达式工厂, 一个表达式工厂包含多个表达式生产者(VExpProducer), 生产者需要调用 `vExpFactory.attach(vExpProducer)` 手动添加到工厂。
VExpProducer	表达式生产者, 一个生产者用于生产对应 Type类型的表达式实例(VExp)，同时将生产过的实例进行缓存。
VExp	表达式，一个表达式用于表达数据并生成表达值。
ExpValue	表达值，通过表达值可以获取指定类型的结果，或者调用 `toStringValue()` 获取字符值，同时表达值还存储了源数据的游标, 这个下标是当前表达式解析数据后源的游标，可以获取这个游标以帮助下一次的数据表达。

4.2 已支持的Type实现类

Type	VExpProducer
N	org.fourfaith.boot.utils.vexpression.producer.NumberExpProducer
S	org.fourfaith.boot.utils.vexpression.producer.StringPartsExpProducer
HN	org.fourfaith.boot.utils.vexpression.producer.HexNumberExpProducer
HS	org.fourfaith.boot.utils.vexpression.producer.HexStringPartsExpProducer
HF	org.fourfaith.boot.utils.vexpression.producer.HexFloatExpProducer
E	org.fourfaith.boot.utils.vexpression.producer.EnumsExpProducer
T	org.fourfaith.boot.utils.vexpression.producer.TimestampExpProducer
B	org.fourfaith.boot.utils.vexpression.producer.BytesExpProducer
A	org.fourfaith.boot.utils.vexpression.producer.ArraysExpProducer
C	org.fourfaith.boot.utils.vexpression.producer.CombinedExpProducer
E	org.fourfaith.boot.utils.vexpression.producer.EnumsExpProducer

4.3 示例代码

本例演示了日期 数值 字符 组合表达式

```
    public static void main(String[] args) {
        VExpFactory vExpFactory = VExpFactory.newWorkshop();
        vExpFactory.attach(new TimestampExpProducer());
        vExpFactory.attach(new NumberExpProducer());
        vExpFactory.attach(new StringPartsExpProducer());
        vExpFactory.attach(new CombinedExpProducer(vExpFactory));

        String data = "22071111340101235 46";

        VExp<?> vExp = vExpFactory.getExpression("T(12,yyMMddHHmmss,yyyy-MM-dd
HH:mm:ss)");
        ExpValue<?> expValue = vExp.express(data);
        System.out.println(expValue);

        VExp<?> vExp1 = vExpFactory.getExpression("N(6,3)");
        ExpValue<?> expValue1 = vExp1.express(data, expValue.getCursor());
        System.out.println(expValue1);

        VExp<?> vExp2 = vExpFactory.getExpression("S(1)");
        ExpValue<?> expValue2 = vExp2.express(data, expValue1.getCursor());
        System.out.println(expValue2);

        VExp<?> expression =
vExpFactory.getExpression("C(T(12,yyMMddHHmmss,yyyy-MM-dd
HH:mm:ss),N(6,3),S(1))");
        System.out.println(expression.express(data));

        /*
        执行结果
        2022-07-11 11:34:01
        12.354
        6
        2022-07-11 11:34:01,12.354,6
        */
    }
```

4.4 自定义开发

如何声明一个属于自己的表达式

1. 定义Type标识符，约定使用 **1-2 个长度的字母**声明一个Type标识符, 注意查看 `VExpType` 避免重复
2. 实现接口 VExp 实现表达式
3. 继承抽象类 VExpProducer 实现生产者

原则上, 建议遵守 Type(Format) 基本格式统一风格并在一定程度上保证可读性

VExp 接口

```
package org.fourfaith.boot.utils.vexpression;
```

```java
/**
 * <p>
 *     值表达式
 * </p>
 *
 * @author Kern
 */
public interface VExp<T> {

    /**
     * 获取解析后的值长度
     * @return 值长度
     */
    int getExpectedStringLength();

    /**
     * 获取原始数据长度
     * @return int
     */
    int getExpectedValueLength();

    /**
     * 解析值
     * @param src 原始值
     * @return value
     */
    ExpValue<T> doExpress(String src);

    /**
     * 获取值类型
     * @return class
     */
    Class<T> getVType();


    /**
     * 解析值
     * @param src 原始值
     * @return value
     */
    default ExpValue<T> express(String src) {
        return express(src, 0);
    }

    /**
     * 解析值
     * @param src 原始值
     * @param startIndex 起始坐标
     * @return value
     */
    default ExpValue<T> express(String src, int startIndex) {
        if (src == null) {
            return null;
        }
        int endIndex = getEndIndex(src, startIndex);
        if (src.length() < endIndex) {
            return null;
        }
```

```
        String valSrc = src.substring(startIndex, endIndex);
        ExpValue<T> expValue = doExpress(valSrc);
        expValue.setCursor(endIndex);
        return expValue;
    }

    default int getEndIndex(String src, int startIndex) {
        int valueLength = getExpectedValueLength();
        return valueLength <= 0 ? src.length() : startIndex + valueLength;
    }



}
```

VExpProducer 抽象类

```
package org.fourfaith.boot.utils.vexpression;


import org.fourfaith.boot.utils.vexpression.producer.*;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

/**
 * <p>
 *      值表达式生产者
 * </p>
 *
 * @author Kern
 */
public abstract class VExpProducer<VE extends VExp<?>> {

    private final Map<String, VE> expressionMap = new ConcurrentHashMap<String,
VE>();

    public static VExpProducer<?>[] allStandardProducers() {
        return new VExpProducer[]{
                new ArraysExpProducer(),
                new BytesExpProducer(),
                new CombinedExpProducer(),
                new EnumsExpProducer(),
                new HexFloatExpProducer(),
                new HexNumberExpProducer(),
                new HexStringPartsExpProducer(),
                new NumberExpProducer(),
                new StringPartsExpProducer(),
                new TimestampExpProducer(),
        };
    }

    /**
     * 表达式的类型标识
```

```
     * @return Type
     */
    public abstract VExpType getType();

    /**
     * 生产表达式
     * @param expression 表达式
     * @return VExp
     */
    public abstract VE produce(String expression);

    /**
     * 校验表达式
     * @param expression 表达式
     * @return true | false
     */
    public abstract boolean valid(String expression);

    /**
     * 解析值
     * @param expression 表达式
     * @return value
     */
    public VE getExpression(String expression) {
        VE ve = expressionMap.get(expression);
        if (ve == null) {
            try {
                ve = produce(expression);
            } catch (Exception e) {
                e.printStackTrace();
            }
            expressionMap.put(expression, ve);
        }
        return ve;
    }

    protected String skinning(String expression) {
        return
expression.substring(expression.indexOf(VExpConstants.LEFT_PARENTHESES) + 1,
                expression.lastIndexOf(VExpConstants.RIGHT_PARENTHESES));
    }
}
```