# 数据类型

用户语言 支持C语言的基本数据类型，Int 表示整型值； Double 和 Float 表示浮点型值； Bool 是布尔型值；String 是文本型数据。

## 一、 整数

下表列出了关于标准整数类型的存储大小和值范围的细节：

| 类型 | 存储大小 | 值范围 |
| --- | --- | --- |
| int8 | 1 字节 | -128 到 127 （ -2^7到2^7-1 ） |
| uint8 | 1 字节 | 0 到 255 （ 0到2^8-1 ） |
| int16 | 2 字节 | -32,768 到 32,767 （ -2^15到2^15-1 ） |
| uint16 | 2 字节 | 0 到 65,535 （ 0到2^16-1 ） |
| int32 | 4 字节 | -2,147,483,648 到 2,147,483,647 （ -2^31到2^31-1 ） |
| uint32 | 4 字节 | 0 到 4,294,967,295 （ 0到2^32-1 ） |
| int64 | 8 字节 | -9,223,372,036,854,775,808 到 +9,223,372,036,854,775,807 （ -2^63到2^63-1 ） |
| uint64 | 8 字节 | 0 到 18,446,744,073,709,551,615 （ 0到2^64-1 ） |

## 二、浮点数

下表列出了关于标准浮点类型的存储大小、值范围和精度的细节：

| 类型 | 存储大小 | 值范围 | 精度 |
| --- | --- | --- | --- |
| float | 4 字节 | 1.2E-38 到 3.4E+38 | 6 位小数 |
| double | 8 字节 | 2.3E-308 到 1.7E+308 | 15 位小数 |

## 三、 布尔

下表列出了关于布尔类型值范围的细节：

| 类型 | 存储大小 | 值范围 |
| --- | --- | --- |
| bool | 1字节 | 0:false; 1:true |

## 四、字符串

下表列出了关于字符串类型值范围的细节：

| 类型 | 存储大小 | 值范围 |
| --- | --- | --- |
| string | 4096字符 | - |

文档更新时间: 2020-07-03 16:23

# 基本运算符

运算符是检查、改变、合并值的特殊符号或短语。例如，逻辑与运算符 &&（如 a && b）。
用户语言 支持大部分标准 C 语言的运算符，且为了减少常见编码错误做了部分改进。如：赋值符（=）如果在 @ 之前作为条件逻辑判断，与"判等"运算符（==）等价，这样就消除了手误将判等运算符（==）写成赋值符导致代码错误的缺陷。

用户语言 还提供了特有的分隔运算符，例如 @ 或 ：，表示条件和执行的分隔，或真执行和假执行的分隔。

## 基本运算符列表

| 标识 | 优先级 | 名称 | 说 明 | 举例 |
|---|---|---|---|---|
| @ | 1 | 条件结果分隔符 | 条件、状态和结果分隔符 | (k1=1),(S=1)**@**D1=1 |
| : | 2 | 真假结果分隔符 | 结果表达式中的真假结果表达的分隔符 | (k1=1),(S=1)**@**D1=1**:**D1=0 |
| **;** | 5 | 结果分隔符 | 在结果中，表示两个结果之间的分隔符 | (k1=1),<br>(S=1)**@**D1=1**;**D2=0:D1=0**;**D2=1 |
| , | 5 | 逻辑"与" | 在条件、状态中，<br>表示两个关系表达式之间的与关系。 | (k1=1**,**k2=1),<br>(S1=1,S2=1)**@**D1=1:D1=0 |
| + | 4 | 逻辑"或" | 在条件、状态中，<br>表示两个关系表达式之间的或关系。 | (k1=1**+**k2=1),(<br>D1=1**+**D2=1)**@**V1=1:V1=2 |
| == | 6 | 逻辑"相等" | 在条件、状态中，<br>表示两个关系表达式之间的相等关系； | （k1**==**1），<br>(D1**==**0)**@**D1=1:D1=0 |
| != | 6 | 逻辑"不等于" | 在条件、状态中，<br>表示两个关系表达式之间的不等于关系。 | （k1**!=**1），<br>(D1**!=**0)**@**D1=1:D1=0 |
| >= | 7 | 逻辑"不小于" | 在条件、状态中，<br>表示两个关系表达式之间的不小于关系。 | （k1**>=**1），<br>(D1**>=**0)**@**D1=1:D1=0 |
| <= | 7 | 逻辑"不大于" | 在条件、状态中，<br>表示两个关系表达式之间的不大于关系。 | （k1**<=**1），<br>(D1**<=**0)**@**D1=1:D1=0 |
| < | 7 | 逻辑"小于" | 在条件、状态中，<br>表示两个关系表达式之间的小于关系。 | （k1**<**1），(D1**<**0)**@**D1=1:D1=0 |
| > | 7 | 逻辑"大于" | 在条件、状态中，<br>表示两个关系表达式之间的大于关系。 | （k1**>**1），(D1**>**0)**@**D1=1:D1=0 |
| = | 6 | 赋值 | 在结果表述式中时，用来表示赋值； | (k1=1),(S=1)**@**D1**=**1:D1**=**0 |
| ! | -1 | 结束符 | 在语句中表示一条逻辑语句的结束 | (k1=1),(S=1)**@**D1**=**1:D1**=**0 **！** |

# 一、赋值运算符

赋值运算符（a = b），表示用 b 的值来初始化或更新 a 的值：

```
b = 10
a = 5
a = b
// a 现在等于 10
```

# 二、比较运算符

高级编程语言中常用的比较运算符都可以在 用户语言 中使用：

等于（a == b），判断条件中两个对象或表达式的相等关系
不等于（a != b），判断条件中两个对象或表达式的不等关系
大于（a > b），判断条件中两个对象或表达式的大于关系
小于（a < b），判断条件中两个对象或表达式的小于关系
大于等于（a >= b），判断条件中两个对象或表达式的大于等于关系
小于等于（a <= b），判断条件中两个对象或表达式的小于等于关系

每个比较运算都返回了一个标识表达式是否成立的布尔值：

```
1 == 1   // true, 因为 1 等于 1
2 != 1   // true, 因为 2 不等于 1
2 > 1    // true, 因为 2 大于 1
1 < 2    // true, 因为 1 小于2
1 >= 1   // true, 因为 1 大于等于 1
2 <= 1   // false, 因为 2 并不小于等于 1
```

# 三、逻辑运算符

逻辑与 （a ，b）或者写为（a && b）,在条件中，表示两个关系表达式之间的与关系。
逻辑或 （a + b）或者写为（a || b）,在条件中，表示两个关系表达式之间的或关系。

逻辑与运算符（a , b）表达了只有 a 和 b 的值都为 true 时，整个表达式的值才会是 true。参考如下伪代码示例：

```
a = true
b = false
if a && b {
    print("Welcome!")
} else {
    print("ACCESS DENIED")
}
// 输出“ACCESS DENIED”
```

逻辑或运算符（a || b）是一个由两个连续的 | 组成的中置运算符。它表示了两个逻辑表达式的其中一个为 true，整个表达式就为 true。参考如下伪代码示例：

```
a = true
b = false
if a || b {
    print("Welcome!")
} else {
    print("ACCESS DENIED")
}
// 输出“Welcome!”
```

# 四、分隔符

条件动作分隔符（ a==1 @ b=1 ），当 a==1成立时，执行b=1
真假动作分隔符（ a==d @ b=1 : C=1 ），当 a==d 成立时，执行b=1；不成立时，执行c=1
动作分隔符（ a==1 @ b=1 ; d=1 ），当a==1成立时，执行b=1，再执行d=1

# 五、结束符

结束符 (a==d @ b=1 ! )，用来表示一条逻辑语句的结束，该运算符为可选运算符。

# 六、括号明确优先级

左括号“(”
右括号“)”

为了一个复杂表达式更容易读懂，在合适的地方使用括号来明确优先级是很有效的，虽然它并非必要的。左右括号搭配使用。

```
if (a && b) || true  {
    print("Welcome!")
} else {
    print("ACCESS DENIED")
}
// 输出“Welcome!”
```

# 标识符

用户语言语句中的变量名、常量名、类名、对象名、函数名统称为标识符。

**命名规则：**

（1）标识符不能包含平台语法的操作符“@”、“，”、“；”、“+”、“（”、“）”、“<”、“>”、“<=”、“>=”、“!=”、“:”、“!”）。

（2）标识符支持中英文，区分字母大小写。

（3）
“int8”、“uint8”、“int16”、“uint16”、“int32”、“uint32”、“int64”、“uint64”、“float”、“double”、“string”、“bool”、“true”、”false”、“null”关键字不能用作标识符。

（4）标识符不能是纯数字。

---

文档更新时间: 2020-07-03 16:24

# 常量与变量

常量和变量把一个名字（比如 a 或者 b ）和一个指定类型的值（比如数字 10 或者字符串“Hello”）关联起来。常量的值一旦设定就不能改变，而变量的值可以随意更改。

常量和变量在使用前声明。

## 一、常量

根据数据类型划分，有12种基本数据类型的常量，参考数据类型一节。

### 1.1 常量的声明

```
{
    "objectId": -3,
    "variableName": "按下",
    "variableKey": "按下",
    "dataType": "uint8",
    "defaultValue": 1
}
```

### 1.2 常量结构的字段说明

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| objectId | int | 对象ID。<br>-3：表示常量； |
| variableName | string | 常量名称 |
| variableKey | string | 常量key。常量的唯一标识。 |
| dataType | string | 数据类型 |
| defaultValue | string | 常量值 |

## 二、变量

根据数据类型划分，有12种基本数据类型的常量，参考2.1节。当然，为了更好的应用平台用户语言，在应用系统中最好选择合适的变量。

根据所属关系划分，分为全局变量和成员变量。对象的状态由这些成员变量的值决定。

### 2.1 变量的声明

**全局变量的声明：**

```
{
    "objectId": -1,
    "variableName ": "TimeValue",
    "variableKey": "-1.TimeValue",
    "eventType": 1,
    "dataType": "int32",
    "valueDescription": "",
    "defaultValue": 0,
}
```

**成员变量的声明：**

```
{
    "objectId": 1,
    "variableName": "TimeValue",
    "variableKey": "objName.TimeValue",
    "eventType": 1,
    "dataType": "int32",
    "valueDescription": "",
    "defaultValue": 0,
}
```

当objectId大于等于0时，是成员变量对应关联对象的id。variableName为变量名同时是关联对象的属性名。

## 2.2 变量结构的字段说明

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| objectId | int | 对象ID。<br>-3：表示常量；<br>-2：表示虚拟机全局变量；<br>-1：表示应用全局变量；大于等于0时，<br>表示成员变量对应关联对象的id。 |
| variableName | string | 变量名称 |
| variableKey | string | 变量key。变量的唯一标识。 |
| eventType | int | 事件类型（0：不建事件，1，2，3，....事件优先级递减） |
| dataType | string | 数据类型 |
| valueDescription | string | 值描述。用于规定变量的范围。<br>如”[-100，100]”表示该变量有效值的范围是-100至100的闭区间。<br>”[松开=0，按下=1，....]”表示改变量只支持值为”松开”、 ”0”<br>、”按下”、”1”等。 |
| defaultValue | string | 默认值 |

# 函数

## 一、 函数的概念

“函数”是用户语言中非常重要的概念，和Java语言中的函数并没有区别，定义方式也一样，由开发人员根据一些特定的功能来完成，它是完成特定功能的一段程序，这里的函数是由相应的编程语言来实现的，并不是用户语言来实现。函数在用户语言语句中直接调用，函数可以有返回值，也可以没有返回值。

函数可以放在用户语言语句的条件和结果中，也可以单独调用，也可以把函数的结果赋给特定的变量，但此时编制用户语言程序时要注意函数的结果和变量是匹配的。

- 虚拟机全局函数：

用户语言本身提供了一些固定、通用的函数，这些叫做虚拟机全局函数。

如： TimeRange(-1, -1, -1,7,0,0, -1, -1, -1,9,0,0)，即判断当前时间是否在7:00到9:00之间，如果在7:00-9:00之间返回“1”，否在返回“0”；

- 应用函数：

用户语言还提供了一种标准的函数调用接口，开发者可以根据这些接口规则，利用其它通用语言开发出完成不同功能的函数，这些函数叫做应用函数。应用函数包括应用全局函数和成员函数。

如：fAllLedPowerOff ()，调用该函数后则把全部led全部关闭。一般用户语言层的外部函数是放到应用语言层去编制，即作为应用语言层的一部分，而为用户语言的特定变量进行调用。

## 二、 函数的声明

### 2.1 虚拟机全局函数的声明：

```
{
    "objectId":-2,
    "fullName":"Add",
    "functionName ":"Add",
    "functionKey":"-2.1",
    "className":"-2",
    "returnDataType ":"float",
    "isSynchronize":1,
    "paramList":[
        {
            "dataType":"float",
            "valueDescription":""
        },
        {
             "dataType":"float",
            "valueDescription":""
        }
    ]
}
```

## 2.2 应用全局函数的声明：

```
{
    "objectId":-1,
    "fullName":"Message",
    "functionName":"Message",
    "functionKey":"-1.1",
    "className":"-1",
    "returnDataType":"void",
    "isSynchronize":1,
    "paramList":[
        {
            "dataType":"string",
            "valueDescription":""
        }
    ]
}
```

## 2.3 成员函数的声明：

```
{
    "objectId":0,
    "fullName":"host.SendMsg",
    "functionName":"SendMsg",
    "functionKey":"-1.2",
    "className":"Host",
    "returnDataType":"void",
    "isSynchronize":1,
    "paramList":[
        {
            "dataType":"string",
            "valueDescription":""
        }
    ]
}
```

# 三、函数结构

## 3.1 函数结构各个字段说明

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| objectId | Int | 对象ID。-1：表示应用全局函数；-2：表示虚拟机全局函数；<br>大于等于0表示成员函数。 |
| fullName | string | 函数全名称 |
| functionName | string | 函数名称 |
| functionKey | string | 函数key |
| className | string | 类名称。”-1”: 表示应用全局函数；”-2”：表示虚拟机全局函数。 |
| returnDataType | string | 返回值类型 |
| paramList | List<br><Param> | 参数类型列表 |
| isSynchronize | int | 是否线程同步 |

## 3.2 Param字段数据

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| dataType | string | 数据类型 |
| valueDescription | string | 值描述 |

end

# 类

类作为一种通用而又灵活的结构，是开发者构建代码的基础。程序员可以使用定义常量、变量和函数的语法，为类定义属性、添加方法。它有以下特点：

- 定义属性用于存储值
- 定义方法用于提供功能
- 继承允许一个类继承另一个类的特征
- 类型转换允许在运行时检查和解释一个类实例的类型

## 一、定义类

```
{
    "className": "主机",
    "fieldList": [
        {
            "name": "确定按键",
            "className": "int8"
        }
    ],
    "classId": 0,
    "parentClassName": ""
}
```

## 二、 类结构的字段说明

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| classId | int | 类ID |
| className | string | 类名称 |
| parentClassName | string | 父类名称 |
| fieldList | List<fieldAttribute> | 字段列表 |

**fieldAttribute字段数据**

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| name | string | 字段名称 |
| className | string | 类型名称 |

## 三、类的实例

类的定义仅描述了什么是 “主机”类。它们并没有描述一个特定“主机”实体。为此，需要创建类的一个实例。

创建类实例的语法如下：

end

文档更新时间: 2020-07-03 16:23

# 表达式

## 用户语言的表达式

“表达式”是由用户语言规定的变量、运算符和函数等组成的关系表达式，其中最常用的由赋值表达式和逻辑（判断）表达式，它是用户语言中表达事物变化的最基本单元，它描述的是自然界事物的某一种状态或者特定行为，如下表所示：

| 序号 | 表达式 | 自然界客观事物的变化或状态 | 表达式类型 |
| --- | --- | --- | --- |
| 1 | K==1 | 当开关K按下 | 关系表达式 |
| 2 | D==0 | 当灯D在熄灭的状态 | 关系表达式 |
| 3 | CL=1 | 把窗帘打开 | 赋值表达式 |
| 4 | TH>10 | 当温度高于10℃的时候 | 关系表达式 |
| 5 | BJ=1 | 报警 | 赋值表达式 |
| 6 | H>=10 | 当高度大于等于10米的时候 | 关系表达式 |
| 7 | k1==1 + k2==1 | 变量K1等于1 或 变量K2等于2 | 逻辑表达式 |
| 8 | k1==1 ，k2==1 | 变量K1等于1 且 变量K2等于2 | 逻辑表达式 |

### 1.1 赋值表达式

- 语法

变量=值

- 功能

把左边的“值”赋给右边的“变量”。

- 含义

在应用系统中用来改变设备的状态、设置虚拟变量的状态等，如开关的“通”、“断”状态，温度传感器的温度值等等。

例：

K1=1

D1=1

D1=0

### 1.2 关系表达式

程序设计中需要经常对运算对象之间的大小进行比较，如：大小、相等等关系，这样的运算符称为关系运算符，用关系运算符将数值或表达式连接起来的式子就是关系表达式，满足关系表达式运算符关系的结果称为“真”，否则为假。

常用的关系运算符有：

| 标识 | 优先级 | 名称 | 说 明 | 举例 |
| --- | --- | --- | --- | --- |
| == | 6 | 逻辑"相等" | 在条件、状态中，表示两个关系表达式之间的相等关系； | （k1**==**1）,(D1**==**0)**@**D1=1:D1=0 |
| != | 6 | 逻辑"不等于" | 在条件、状态中，表示两个关系表达式之间的不等于关系。 | （k1**!=**1）,(D1**!=**0)**@**D1=1:D1=0 |
| >= | 7 | 逻辑"不小于" | 在条件、状态中，表示两个关系表达式之间的不小于关系。 | （k1**>=**1）,(D1**>=**0)**@**D1=1:D1=0 |
| <= | 7 | 逻辑"不大于" | 在条件、状态中，表示两个关系表达式之间的不大于关系。 | （k1**<=**1）,(D1**<=**0)**@**D1=1:D1=0 |
| < | 7 | 逻辑"小于" | 在条件、状态中，表示两个关系表达式之间的小于关系。 | （k1**<**1）,(D1**<**0)**@**D1=1:D1=0 |
| > | 7 | 逻辑"大于" | 在条件、状态中，表示两个关系表达式之间的大于关系。 | （k1**>**1）,(D1**>**0)**@**D1=1:D1=0 |

- 含义

在应用系统中进行状态比较，以便判断条件是否满足。

例：

D=1

TH>=20

SF=1

## 1.3 逻辑（判断）表达式

- 语法

"表达式1""逻辑运算符""表达式2"

- 功能

把"表达式1"和"表达式2"进行逻辑运算，其值是"真"（true）或"假"（false）

表一：逻辑运算符

| 标识 | 优先级 | 名称 | 说 明 | 举例 |
| --- | --- | --- | --- | --- |
| , | 5 | 逻辑"与" | 在条件、状态中，表示两个关系表达式之间的与关系。 | (k1=1**,**k2=1),(S1=1,S2=1)**@**D1=1:D1=0 |
| + | 4 | 逻辑"或" | 在条件、状态中，表示两个关系表达式之间的或关系。 | (k1=1**+**k2=1),( D1=1**+**D2=1)**@**V1=1:V1=2 |

end

# 语言语句

## 一、 用户语言的“语句”

“语句”是由一个“逻辑表达式”或用户语言规定运算符和多个“逻辑表达式”等组成的表达式；

用户语言“语句”的实质是描述自然界客观事物变化的一种表现形式，自然界中任何事物的变化都可以用用户语言的“语句”来表述。

它描述的是“客观事物从一个状态到另一个状态变化的完整过程”，这是用户语言（不只是用户语言）最重要的概念，也是用户语言的核心思想。

## 二、 语句的格式及含义

### 2.1 格式：

条件@动作

或

(条件),(状态)@真动作:假动作

### 2.2 含义：

当“条件”“成立”时，执行“动作”

或

当“条件”“成立”，且“状态”“成立”时，执行“真动作”；

当“条件”“成立”且“状态”“不成立”时，执行“假动作”。

### 2.3 说明：

- 其中的“成立”就是bool变量的“true”，“不成立”是bool变量的“false”，这里主要是为了方便理解，而描述成“成立”和“不成立”。
- 一条用户语言完整的语句由“条件”,“状态”,“@”,“真动作”,“：”,“假动作”。
- “真动作”和“假动作”在描述中也统称为“动作”。
- 其中“条件”、“状态”和“动作”可以是一条逻辑表达式，也可以是合法平台语法运算符构成的复合逻辑表达式，可形成任何的逻辑和嵌套关系。
- 在实际应用中，一条语句可以由其中的几个部分组成，不一定是全部，而是根据实际需要而定。

### 2.4 实例：

(k1=1),(D1=0) @ V1=1:V1=0

语句解释：当K1=1时；如果D1=0，执行V1=1，否则执行V1=0。

# 特殊元素

“特殊元素”是用户语言为了支持语句中使用日期、时间和延时等操作而做的适配，便于开发者和调试员理解。它本质上是对硬件HAL库函数的封装，支持中文、英文、和国际常用表示方式，并根据应用场景不同不断拓展，具体内容如下：

## 一、 时间日期

- 用户语言支持的时间日期格式如下：

| 类别 | 格式例子 |
| --- | --- |
| 时间点 | 15时00分00秒 |
| | 14:00:00 |
| 日期 | 2019年05月23日 |
| | 2019/05/23 |
| | 2014-09-01 |
| 阴历 | 阴历4月19日 |
| 阳历 | 阳历10月23日 |
| 时间段 | 08:00:00-23:00:00 |
| | 2019/05/21 07:00:00-2019/06/30 23:00:00 |
| | 2014-09-01 07:00:00-2014-09-01 08:00:00 |
| | 2014年09月01日 07时00分00秒-2022年09月01日 08时00分00 |

注意：时间日期只可以出现在语句的条件和状态当中。

## 二、 星期

- 用户语言支持的星期格式如下：

| | 中文 | 英文 | 备注 |
| --- | --- | --- | --- |
| 星期 | 星期一 | Monday或Mon | |
| | 星期二 | Tuesday或Tues | |
| | 星期三 | Wednesday或Wed | |
| | 星期四 | Thursday或Thur | |
| | 星期五 | Friday或Fri | |
| | 星期六 | Saturday或Sat | |

| | | | |
|---|---|---|---|
| | 星期日 | Sunday或Sun | |
| 特殊名称 | 工作日 | weekdays | 星期一至星期五 |
| | 双休日 | weekends | 星期六、星期日 |
| | 礼拜天 | Sunday或Sun | 星期日 |

注意：“星期”只可以出现在语句的条件和状态当中。

# 三、 延时

- 用户语言支持的星期格式如下：

| | 使用格式 | 备注 |
|---|---|---|
| 中文 | 延时0时1分0秒后（1, "K=1,T=0@V (https://github.com/V)=1:V=0"） | 参数1：延时id；<br>参数2：<br>无状态的语句 |
| 英文 | Delay(1, 60, "K=1,T=0@V (https://github.com/V)=1:V=0"） | 同上 |

注意：延时函数只可以出现在语句的结果当中。

- 取消延时使用说明：

| | 使用格式 | 备注 |
|---|---|---|
| 中文 | 取消延时（1） | 参数1：延时id； |
| 英文 | CancelDelay(1) | 同上 |

注意：取消延时函数只可以出现在语句的结果当中。

end

# 编译源文件

用户语言的源文件，作为编译器的输入文件，需要具有"语句表""变量表""函数表""类表""对象表"和"编译器配置信息"等最基本数据表。

## 一、源文件组成

### 1.1 语句(logic)表

#### Json格式声明

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| logicList | List [logic] | Key为状态迁移表表名 |

#### Logic字段数据

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| no | int | 编号 |
| logic | string | 语句内容 |

### 1.2 变量(variable)表

#### Json格式声明

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| variableList | List [variableData] | |

#### variableData字段数据

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| objectId | Int | 对象ID |
| variableName | string | 变量名称 |
| variableKey | string | 变量key |
| eventType | Int | 事件类型 |
| dataType | string | 数据类型 |
| valueDescription | string | 值描述 |
| defaultValue | string | 默认值 |

## 1.3 函数(function)表

**Json格式声明**

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| functionList | List[functionData] | |

**functionData字段数据**

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| objectId | Int | 对象ID |
| fullName | string | 函数全名称 |
| functionName | string | 函数名称 |
| functionKey | string | 函数key |
| className | string | 类名称 |
| returnDataType | string | 返回值类型 |
| paramList | List | 参数类型列表 |
| isSynchronize | int | 是否线程同步 |

**Param字段数据**

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| dataType | string | 数据类型 |
| valueDescription | string | 值描述 |

## 1.4 类（class）表

**Json格式声明**

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| classList | List[classData] | |

**classData字段数据**

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| classId | Int | 类ID |
| className | string | 类名称 |
| parentClassName | string | 父类名称 |

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| fieldList | List< fieldAttribute > | 字段列表 |

**fieldAttribute字段数据**

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| id | Int | 字段ID |
| name | string | 字段名称 |
| className | string | 类型名称 |

### 1.5 对象(object)表

**Json格式声明**

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| objectList | List[objectData] | |

**objectData字段数据**

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| objectId | Int | 对象ID |
| objectName | string | 对象名称 |
| className | string | 类名称 |

### 1.6 参数数据

**Json格式声明**

| 字段名称 | 字段类型 | 说明 |
| --- | --- | --- |
| name | string | 配置项名称 |
| description | string | 配置项描述 |

## 二、文件模板

下面给出简单的源文件模板：

```
{
      "logicList": [
                {
            "no": 0,
            "logic": "主机1.确定按键 = 按下 @  主机1.语音提示(\"请将锅放在炉台上，开始烹饪！\")"
        }
    ],
    "variableList": [
            {
                "objectId": 0,
                  "variableName": "主机1.确定按键",
                  "variableKey": "0.0",
                  "eventType": 1,
                  "dataType": "int8",
                  "valueDescription": "[按下=1,松开=0]",
                  "defaultValue": "0"
            }
        ],
    "functionList": [
        {
            "className": "主机",
```


end

---

文档更新时间: 2020-07-03 16:23