火山平台语法学习手册

(普通用户图文版)

一. 相关术语:

1. 面向对象的程序设计方法

面向对象的程序设计方法是目前最先进的程序设计理念,也是被目前绝大多数程序设计语言都使用的一种程序设计方法,它可以最大限度地提高程序代码的可复用性和可维护性,如果您想学习程序设计,那么这种设计方法是必须需要掌握的.

为了降低用户的学习门槛,火山对这套理念进行了最大限度的精简,一些晦涩难懂的和一些不大常用/实用的内容均被剔除出去,只保留了其精华和必须需要掌握的部分,因此学习起来并不难.

注意:

A. 如果您对面向对象的程序设计方法了解不多,推荐您先阅读[面向对象的程序设计方法]章节,再来阅读本手册后续内容,将会帮助您对后续内容的理解;

B. 由于所基于系统类库的不同(语法层面是一致的),本手册中的例程示图如必要会提供火山视窗和火山安卓两个平台程序版本.

2. 名称:

火山平台中的名称分为以下这些:

A. 单名称:

单名称必须以英文字母/下划线字符/汉字字符开头,后面跟随英文字母/下划线字符/数字/汉字字符.如未特殊标注,本文档中所提出的所有"名称"均为单名称.

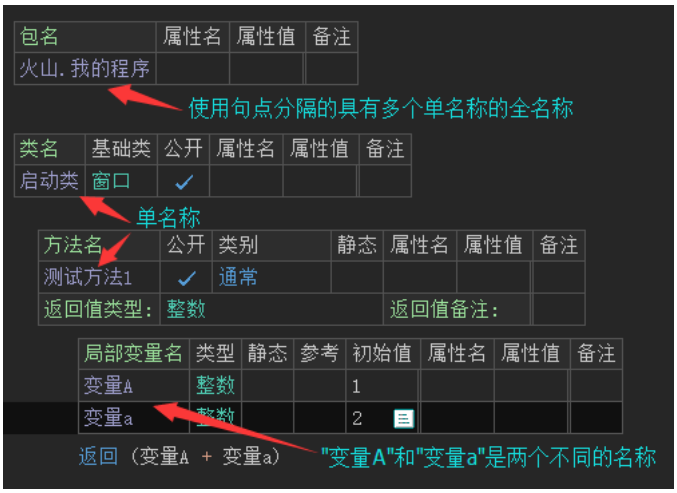
B. 全名称:

以句点字符组合在一起的单名称称为全名称,如"火山.程序". 单名称可以被认为是全名称的一种(只包括一个单名称的全名称).

实际上,目前只有包名才使用到由多个单名称组成的全名称.

注意: 火山中的名称对英文字母的**大小写敏感**,譬如"abc","ABC"所指定的不是同一名称.

下图为名称在程序中的使用和定义方法:



操作提示:

火山程序中的名称就其来源可以分为以下两种:

1. 系统中预先定义的名称. 主要是关键字和系统属性名称;

2. 用户程序中自行定义的名称. 主要是各种定义型程序成员(如类/方法/常量/变量/参数)的名称.

这些名称均可以通过系统内置的首拼或全拼输入法输入. 如:

上图中定义"变量A"和"变量a"时所使用的"**整数**"数据类型,可以使用首拼输入方式"zs"或者全拼输入方式"zhengshu"输入;

上图中所定义的"**测试方法1**"方法,可以使用首拼输入方式"csff1"或者全拼输入方式"ceshifangfa1"在程序语句中输入.

具体输入法匹配规则请见系统输入法插件被载入时在开发环境状态框中输出的提示信息:

```
载入插件成功:
    名称及版本: 首全拼输入法 1.0
    标识符: wutao.plugin.im.cn.pinyin.1
    级别: A
    可用于火山平台类别: 专用版
    插件说明: 本输入法为程序中的中文字词提供以首拼或全拼方式快速输入支持,可以用来输入程序中所涉及到的一切名称.
    本输入法支持南方音及多音字,输入时会自动判别所输入的拼音字母组合文本是首拼方式还是全拼方式,例如,使用"qz"或者"quzheng"均可以输入"取整"方法名称.
    在使用首拼输入方式时,需要注意纯韵母发音汉字的输入,如:"按钮"中的"按"字,它的发音是韵母"an",对于此类汉字,在首拼输入方式中必须写全,譬如"按钮"的首拼输入方式就应该为"ann"(即an, n).
    本输入法的使用注意事项:
    1. 为了匹配目标名称中的大小写英文字母和半角数字,请在相应位置处使用对应的大写英文字母和半角数字,如:"jsjA"匹配"计算机a"和"计算机A";
    2. 为了匹配目标名称中的汉字符号或不知道发音的汉字,请在相应位置处使用该汉字,如:"j算j1"匹配"计算机1";
    3. 拼音字母组合文本内不能包含除了"_"以外的所有其它半角符号.
```

3. 立即数

立即数用作表达一个直接的字面数据值,有以下几类:

1. 数值立即数:

A. 十进制整数或小数,小数支持使用科学计数法,如: 1.32e3

B. 十六进制整数: "0x"后跟数字0-9或字母A-F(大小写无关). 如: 0x12AC3F

C. 字符整数值: 使用单引号括住的字符,如'A','吴'.

如果欲强行指定数值的数据类型,可以使用"强制类型转换"操作符,譬如"(长整数)1",提供了一个数据类型为长整数的数值立即数.

如果某整数的数值超出了整数的最大有效范围,将自动被设定为长整数数据类型. 如: 0x123456789A 将被自动认为是长整数数据类型.

2. 逻辑型立即数: 为真/假.

3. 字符串立即数:

为用双引号括住的一段文本,文本内支持使用以下转义符:

转义符	解释
\b	退格符
\f	换页符
\r	回车符
\n	换行符
\t	水平制表符
\'	单引号
\"	双引号
\\	反斜杠
\u	后跟1-4个十六进制字符,为所对应字符的Unicode代码值.
\	后跟1-3个八进制字符,为所对应字符的代码字节值.
\x	后跟1-2个十六进制字符,为所对应字符的代码字节值.

如: "您好!\r\n祖国" 在"您好!"和"祖国"之间通过使用转义符插入了一个 回车和换行符.

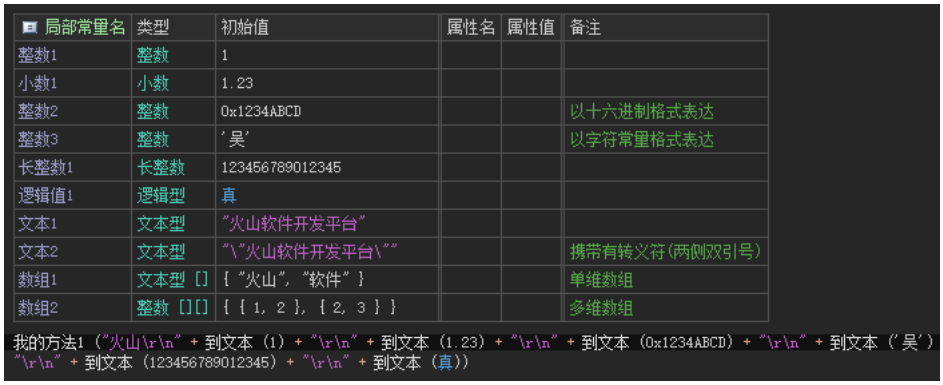
注意: 在火山视窗平台里面,如果欲达到换行效果,需要插入"\r\n"回车和换行两个字符.

4. 数组立即数(只能在提供常量/变量数组初始值时使用):

为使用花括号括住的立即数的组合,如: { 1, 2 }, 多维数组可以嵌套,如: { { 1, 2 }, { 3, 4 }, { 5, 6 } }

注意: 多维数组的同一维内成员数目必须相等,譬如如下格式的数组立即数是不允许的: { { 1, 2 }, { 3 } } ,因为其第2个维中的成员数目不相等.

下面为各类立即数在初始值和程序语句中的使用例图:



局部常里名	类型	初始值	属性名	属性值	备注
整数1	整数	1			
小数1	小数	1.23			
整数2	整数	0x1234ABCD			以十六进制格式表达
整数3	整数	'吴'			以字符常里格式表达
长整数1	长整数	123456789012345			
逻辑值1	逻辑型	真			
文本1	文本型	"火山软件开发平台"			
文本2	文本型	"\"火山软件开发平台\""			携带有转义符(两侧双引号)
数组1	文本型 []	{ "火山", "软件" }			单维数组
数组2	整数 [][]	{ { 1, 2 }, { 2, 3 } }			多维数组

```
我的方法1 ("火山\r\n" + 到文本 (1) + "\r\n" + 到文本 (1.23) + "\r\n" + 到文本 (0x1234ABCD) + "\r\n" + 到文本 ('吴')
"\r\n" + 到文本 (123456789012345) + "\r\n" + 到文本 (真))
```

4. 数据类型:

数据类型可以为以下几种:

A. 基本数据类型;

B. 用户程序中定义的类,称为**类数据类型**;

C. 数组数据类型

数组数据类型为基本或类数据类型后面跟随一个或多个数组维定义组成.

每个数组维定义由左右中括号("[]")组成,如果应用在变量定义上,可以在中括号内部置入具体成员数目,表示生成具有对应维数的数组变量实例. 如:

单维文本数组数据类型: 文本 []

多维整数数组数据类型: 整数 [] []

定义具有指定成员数目的整数数组实例变量: 整数 [3], 整数 [3][2]

数据类型可以在定义常量/变量/参数/方法返回值时使用,也可以在程序语句中使用,如:

视窗示例	

二. 火山程序结构:

一个火山程序的构成结构如下:

1. 包定义
    1. 文档嵌入行(普通用户无需了解)
    2. 文档注释行
    3. 类定义
        1. 类嵌入行(普通用户无需了解)
        2. 类注释行
        3. 类成员常量
        4. 类成员变量
        5. 类成员方法
            1. 方法参数
            2. 方法局部常量
            3. 方法局部变量
            4. 语句注释行
            5. 语句嵌入行(普通用户无需了解)
            6. 语句行

下面是一个火山程序的基本样图(天蓝色文字为说明,下同):

视窗示例	安卓示例

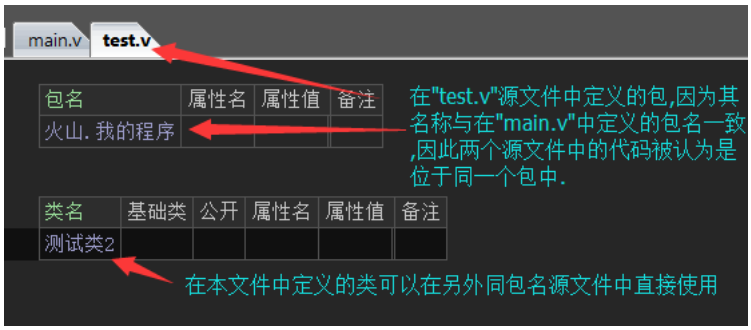
1. 包:

包名为全名称(即中间可以包括句点),通常使用的名称格式为以组织或事物的从大到小排列,譬如:"湖北.武汉.递归公司"/"递归公司.软件开发部"/"火山系统.安卓平台.测试程序"等等,这些都可以,自己维护这些代码时觉得清晰易懂就行了.

一个包内可以包含多个文档注释行/文档嵌入行/类定义,所有这些内容都被认为位于这个名称的包中.

每新建一个火山程序,都会在首部固定加入一个不可被删除的包定义成员,该成员始终位于源文件的首部而且只能存在一个:

多个程序文件可以具有相同的包名,此时其中的内容被认为处于同一个包中:



2. 注释

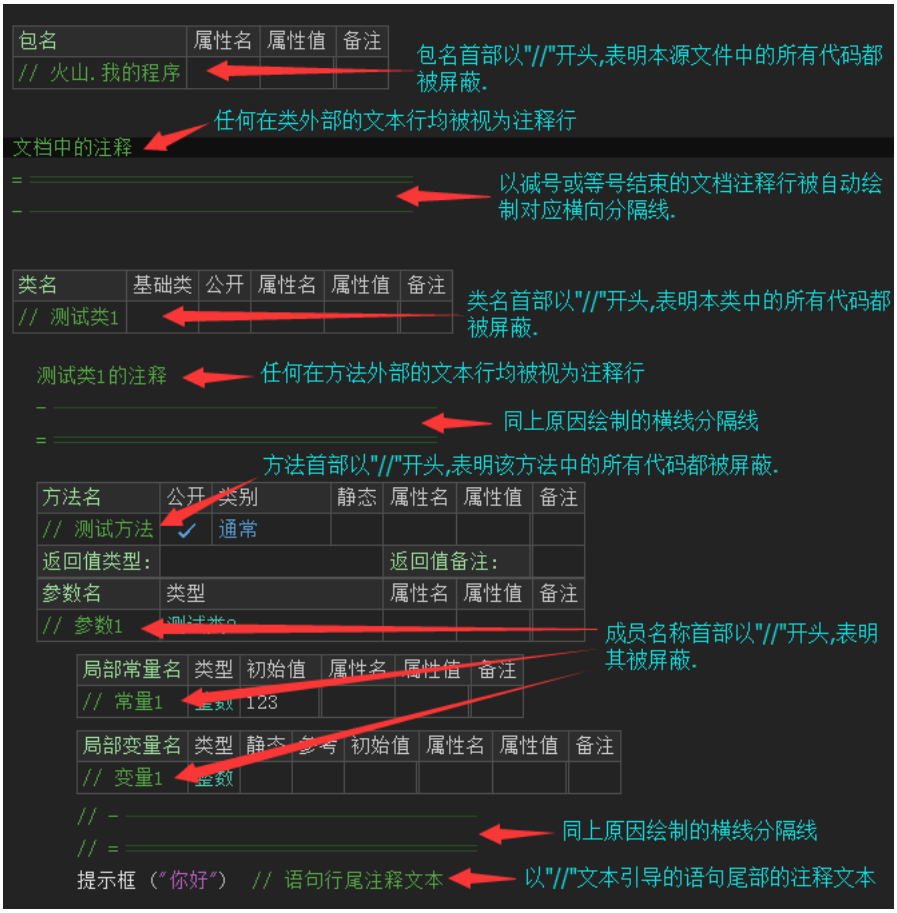
可以使用以下方式定义火山程序中的注释:

A. 任何定义型成员其名称如果以**连续两个'/'字符**引导,说明其处于被注释状态,其以及其中的所有内容(包括直接/间接子成员)在编译时都将被忽略;

B. 在类外部的程序行,被称为**文档注释行**; 在类内部但是在类方法外部的程序行,被称为**类注释行**;

C. 在类方法内部可以使用**连续两个'/'字符**引导一段一直到行尾的语句注释文本.

文档注释行/类注释行/从行首开始的语句注释文本,其尾部如果以一个或多个**减号/等号**字符结束,IDE将自动在其后绘制对应长度的单/双分隔线.可以在IDE设置选项中将此机制关闭.



从上图可以看到,**任何不在方法内部的文本行均被看作是注释行**,请注意这一点.

3. 类

类用作定义一个可以具有各种子成员的复合数据类型,这些子成员为:

1. 类注释行;

2. 类嵌入行(普通用户无需掌握);

3. 类成员变量;

4. 类成员常量;

5. 类成员方法(通常方法/属性读或写方法/事件定义或接收方法).

注意事项:

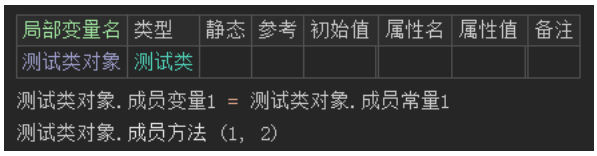
1. 同一个包中不允许出现相同的类名;

2. 类名为单名称(即其中不允许使用句点). 实际上包名是唯一允许中间使用句点的名称;

3. 名称为"**启动类**"的类被用作特殊指定用户程序的启动类,用户程序将从此处开始执行;

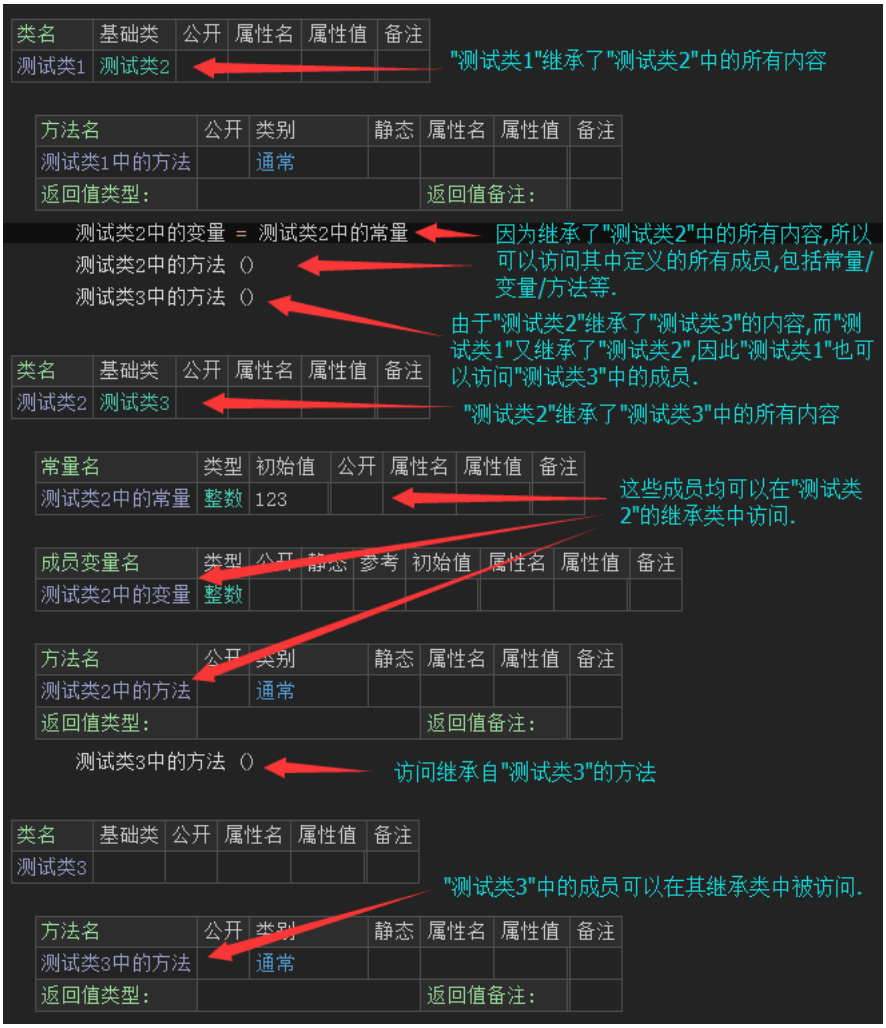
4. 引用类名时可以直接使用类名称,或者使用"**包名.类名**"进行引用,此格式被称为类的**全名称**,当访问其它包中的公开类时,必须提供类的全名称;

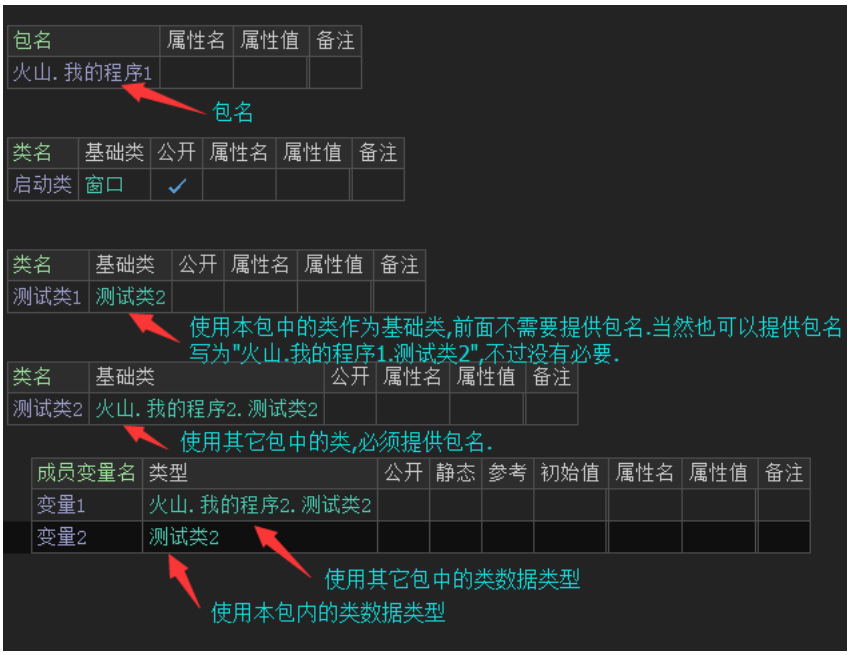
5. 在需要访问类实例对象的成员时,在类对象实例名称和欲访问的子成员名称之间使用**句点**分隔即可,前面指定具体访问哪一个对象实例,后面指定访问这个对象实例的具体哪个子成员. 如:

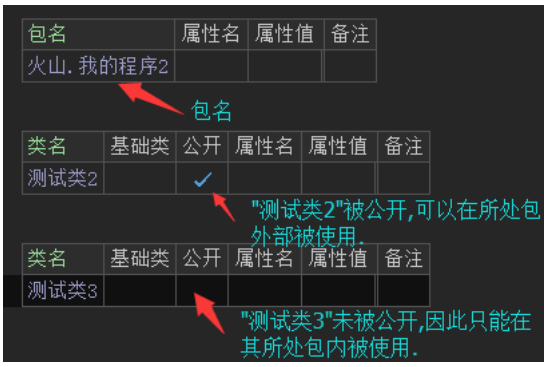


分别访问了"测试类"的实例对象"测试类对象"的"成员变量1"/"成员常量1"/"成员方法"三个子成员.

基本属性表:

名称	解释
基础类	指定本类的基础类名称,可以是单名称/全名称,也可以是空文本(表示无基础类). 如果定义了非空基础类,当前类将自动继承所有来自该基础类的内容,如果该基础类还有基础类,将一并继承过来,此时本类称为这些类的"**继承类**". 注: 有一些编程语言也称"继承类"为"**子类**","基础类"为"**父类**",意思是一样的. 继承类访问其直接/间接基础类中的成员不需要其为公开状态. 例图说明: 
公开	指定本类是否对外公开. 公开类可以在所处包外部被访问,而非公开类只能在所处包内部被访问. 例图说明:





4. 类成员或局部常量

常量用作定义一个不允许在程序中进行修改的恒定值.

常量与变量有以下不同之处:

1. 常量只能在定义时被赋予初始值,不能在程序中被修改;

2. 在设置常量初始值时,只能提供立即数,不能提供其它常量;

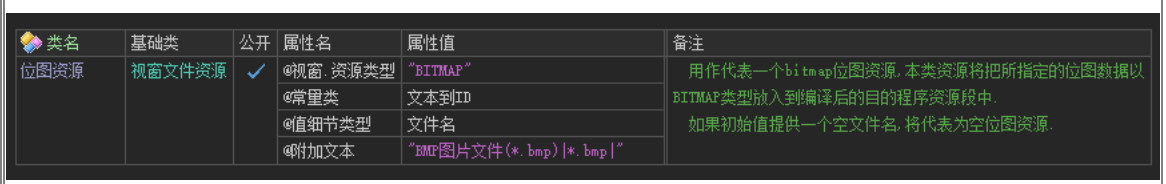
3. 常量的数据类型只能为基本数据类型(注: 还可以为常量类,但普通用户不需了解);

4. 常量的"**静态**"属性固定为真,即:常量并未存放在类的对象实例中,而只存放在类信息本身中,所以无论定义多少类对象实例,常量都只存在一份.因此常量的访问方式与静态成员变量的访问方式 是一样的,在其所处类/继承类外部需要以"**所处类名.常量名**"的方式访问.

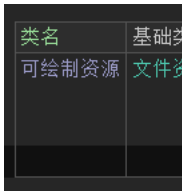
基本属性表:

名称	解释
类型	提供常量的数据类型,只能是基本数据类型或常量类.
公开	指定本常量是否公开.公开常量可以在所处类外部被访问,而非公开常量只能在本类或者其继承类中访问. 局部常量没有本属性.
初始值	提供常量的初始值立即数,必须对应常量的数据类型. 在设置常量初始值时,只能提供立即数,不能提供其它常量. 如果数据类型为常量类,则应该直接提供该常量类所对应基本数据类型的立即数,如:

视窗示例



上图为 视窗系统类库中的一个名为"位图资源"的常量类,其初始值数据类型为文本型,下图为一个设置了对应数据类型初始值的常量.此时 视窗平台编译器会自动对此类常量进行特殊处理,以收集并建立对应的视窗资源.



上图为安卓系统类库中的一型,下图为一个设置了对应常量进行特殊处理,以收集

注意: 普通用户只需要知道当常量/变量的数据类型为系统类库中所提供的常量类时,可以直接设置对应的基本数据类型立即数初始值即可,其它方面(譬如如何建立常量类),无需掌

常量一般用作定义一些恒定值,譬如圆周率的PI. 这样既能避免多次输入出错,还能提供程序的运行效率(便于优化).

下图是在火山安卓基本库的"数学运算类"中定义的2个数学常量及其使用例子,在需要使用这些值的地方可以使用常量来代替(如"数学运算类.E". "数学运算类.PI"),避免输入错误:

类名	基础类	公开	属性名	属性值	备注
数学运算类		✓	@全局类	真	提供常用数学运算功能

常量名	类型	初始值	公开	属性名	属性值	备注
E	小数	2.718281828459045235	✓			自然常数, 自然对数函数的底数, 有时被称为欧拉数.
PI	小数	3.14159265358979323846	✓			圆周率, 圆的周长与直径的比值.

局部变量名	类型	静态	参考	初始值	属性名	属性值	备注
变量1	小数						

```
变量1 = 数学运算类.PI * 2
```

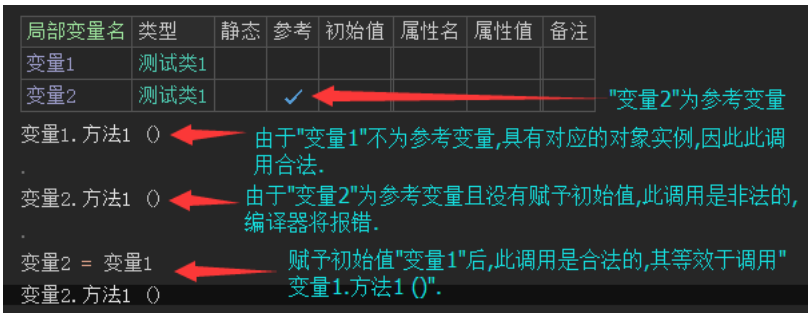
注: 火山视窗中的对应类名为"常数",用法是一致的.

5. 类成员或局部变量

变量的内容允许在程序中被动态修改.

基本属性表:

名称	解释
类型	提供变量的数据类型,可以是基本数据类型或者类数据类型.
静态	指定是否为静态变量. 静态成员变量有以下特点: **1.** 静态变量在所处类载入后即保持始终存在,并且未存放在类的对象实例中,而只存放在类信息本身中,所以无论定义多少类对象实例,静态成员变量都只存在一份; **2.** 由于静态成员变量存放在类本身中,所以在其所处类/继承类外部需要以"**所处类名.静态成员变量名**"的方式访问: 如上图,在"**测试类1**"中定义了一个名为"**静态成员变量1**"的成员变量,在测试类2中红色箭头所指向的代码是正确的访问方式,黄色箭头所指向的代码虽然也能被编译器所接受 无论在"**测试方法**"方法中定义了几个"**测试类1**"的对象实例("对象1"/"对象2"),"**测试类1**"的"**静态成员变量1**"始终只存在一份,因此红色和黄色箭头指向的代码所访问的"**静态** **3.** 由于静态变量初始化时不存在所处类实例,静态成员变量不支持自动挂接其事件到其所处类,在需要时必须通过"挂接事件"命令手动挂接.非静态的类成员变量,如果其数据类型
参考	指定是否为参考变量. **注意:** 在火山视窗平台中,不支持定义参考型数据,这是两个平台在语法层面上唯一不同的地方.当在火山视窗程序中定义参考变量时,将会如图所示: 因此下面的讲述均为针对火山安卓平台. 参考变量有以下特点: **1.** 参考变量本身并不会创建对象实例,而是用作保存到其它对象实例的指向性信息(即**参考**),访问参考变量等于访问该变量所参考到的另一个对象实例.反之非参考变量("参考" **2.** 由于其本身并未定义对象实例,因此参考变量在使用前必须首先赋值(即赋予其所参考到的对象实例). 想想我们手机通讯录中的联系人条目,每个条目均相当于一个"参考变量",它记录有指向某联系人(对象实例)的相关信息,但是不为该联系人(对象实例)本身. 请查看如下代码:



首部定义了一个"**测试类1**"的对象实例"**变量1**",然后定义了一个"**测试类1**"的参考变量"**变量2**".
两者的区别在于: 前者创建了真实存在的对象实例,而后者仅仅用作存放一个参考.
由于"**变量1**"存在对象实例,因此后面调用其"**方法1**"方法不会有问题,而紧跟其后的"**变量2.方法1** ()"调用就会被编译器报错,因为其尚未设置所欲参考到的对象实例.
在后面通过将"**变量1**"赋值过去,从而将"**变量2**"参考到了"**变量1**"所指向的对象实例,因此再调用"**变量2.方法1** ()" 就不会出错了,其等效于调用了"**变量1.方法1 ()**".

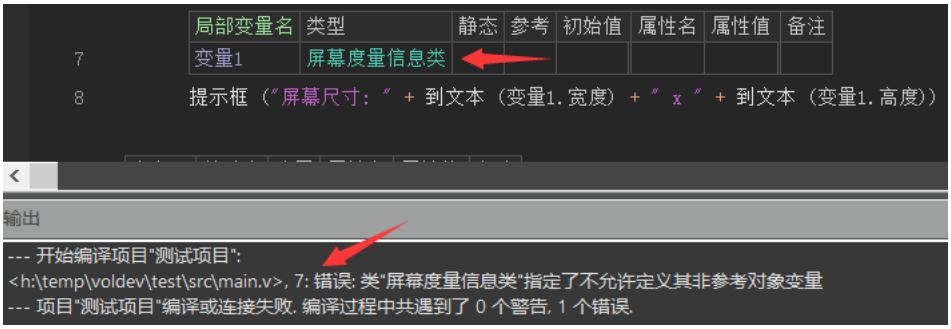
获得对象实例的各种方法:

1. 通过定义非参考变量的方式直接创建:

2. 通过所提供的类方法创建:

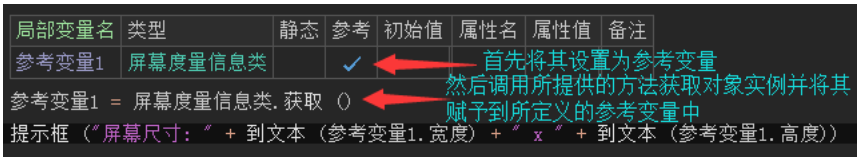
某些类不能通过直接定义该类的变量来创建其对象实例,因为欲创建其对象实例时可能还需要提供一些必要的参数信息或者满足其它前提. 如:
在定义"屏幕度量信息类"时,指定了不允许直接定义其非参考对象变量:

因此在编译下图程序时,将会出错:



我们查看一下该类所提供的所有方法,可以找到创建并返回其对象实例的方法为:

那么我们把前面的程序改为如下内容就可以了:



总结: 如果某个类指定了不能直接定义其对象实例变量,那么肯定可以通过调用某个方法获得其对象实例.绝大多数情况下该方法由此类自行提供,名称中一般包括"创建"/"新如"可绘制对象类"的如下方法:

注意,使用参考属性时最容易犯的错误:

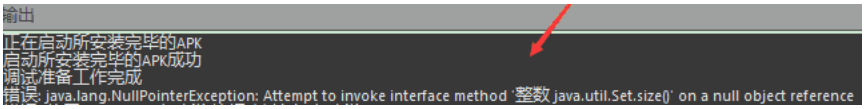
1. 遗漏设置必要的"参考"属性,如下图:

"方法1"和"方法2"中的"变量1"在定义时均没有设置"参考"属性,这样编译器会自动为其创建一个"测试类"对象实例,但是所创建的该对象实例在程序中根本没有被使用,从而导致
所以每当我们定义一个数据类型为类的变量时,一定要检查是否需要为其设置"**参考**"属性 ,判断方法就是编译器自动创建的对象实例有没有在程序中被使用到,如果没有被使用到

2. 使用空参考变量:

所谓"空参考变量",即没有参考到任何对象实例的变量,如:

像这样的代码,首先在编译时会报错.其次,即使编译通过,在运行时也会报告类似如下的空指针错误信息(NullPointerException, Java目的平台):



所以我们在使用参考变量前,一定要记得首先确保它已经参考到了一个非空对象的对象实例.

公开

指定本变量是否公开.公开变量可以在所处类外部被访问,而非公开变量只能在本类或者其继承类中访问.

局部变量没有本属性.

初始值

提供变量的初始值,必须对应变量的数据类型.

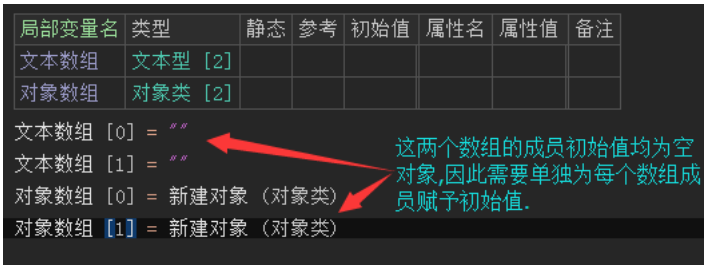
设置参考变量的初始值时,可以提供空对象.

如果未提供初始值,非数组变量的初始值如下:

数值型变量的初始值为0,逻辑型变量的初始值为假,文本型变量的初始值为""(即空文本).对于火山安卓平台,参考变量("参考"属性为真的变量)及非基本数据类型变量的初始值均
此所有非基本数据类型变量均为对应的对象 实例.

数组变量的初始值如下:

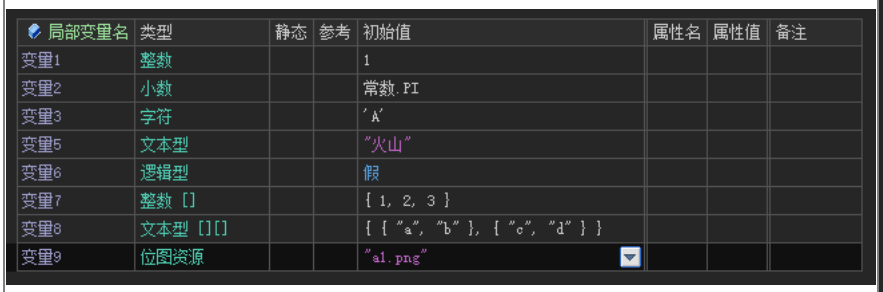
数值型数组变量的每个成员初始值为0,逻辑型数组变量的每个成员初始值为假,其它所有数据类型(包括文本型)数组变量的每个成员初始值 如果为火山安卓平台则为空对象,为
因此,在火山安卓平台中,对于非数值和逻辑型的数组,其每个成员在使用前必须首先被单独赋值,如:



与常量不同的是: 在设置变量初始值时,可以提供立即数,也可以提供常量.

变量初始值设置例图:

视窗示例



安卓

从上图中可以看出,"变量2"使用常量作为初始值,"变量9"和常量初始值一样,当数据类型为常量类时,可以为其设置所对应基本数据类型的立即数初始值.

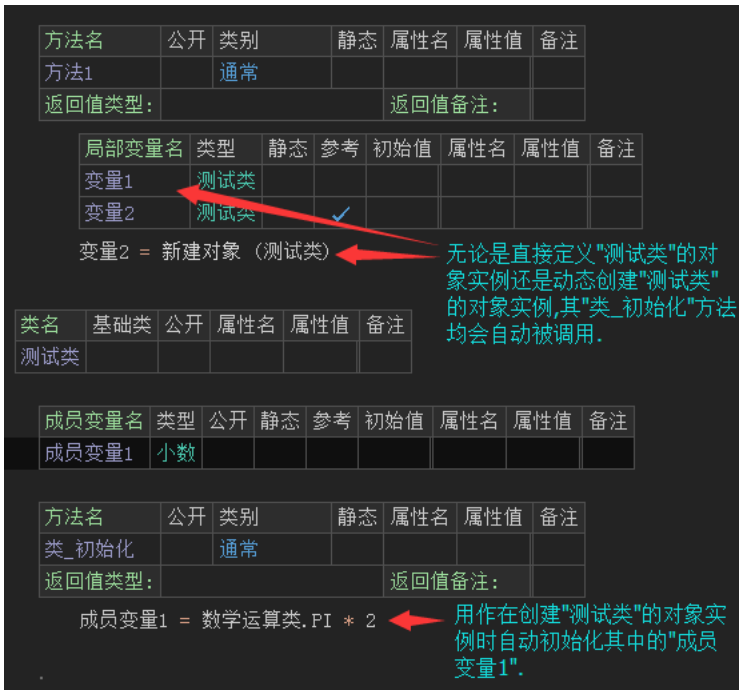
6. 类成员方法

定义类的方法,用作类对外提供其所支持的功能时使用.

方法的名称:

1. 方法的名称必须为单名称,而且必须在类的所有成员名称中唯一;

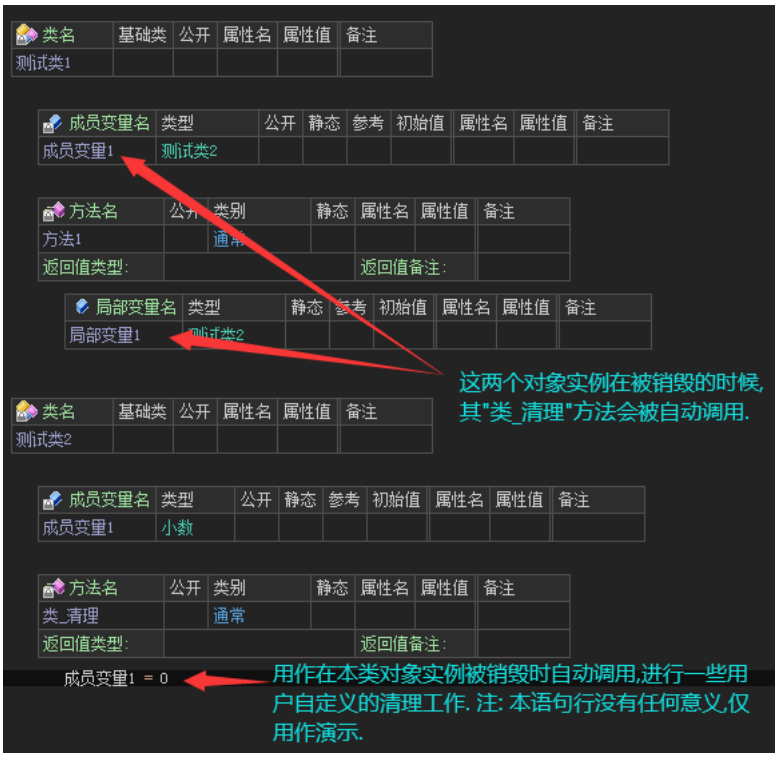
2. 名称为"**类_初始化**"的方法为类的初始化方法(在类对象被创建时自动调用),该方法不携带任何参数并且不返回值,对是否公开没有要求. 如下图:



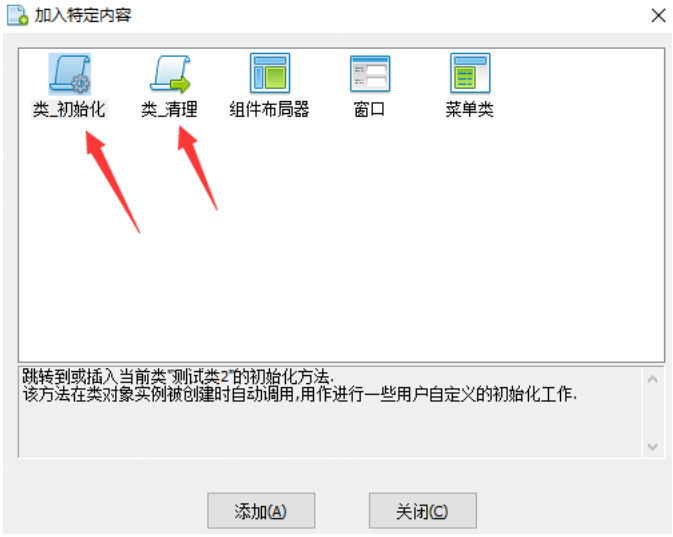
3. 名称为"**类_清理**"的方法为类的清理方法(在类对象被销毁时自动调用),该方法不携带任何参数并且不返回值,对是否公开没有要求.

注意:在火山安卓平台上,由于内存垃圾自动回收机制的限制,类清理方法不被支持.

以下为在火山视窗平台上的演示程序:



提示: "类_初始化"和"类_清理"方法在开发环境中可以通过按下Ctrl+I组合键调用对应对话框自动插入对应空方法:



方法的基本属性表:

名称	解释
返回值类型	提供方法执行完毕后所返回数据的数据类型 如果指定了非空数据类型,则方法中就必须调用"**返回**"关键字语句来返回对应数据类型的数据: 
静态	指定是否为静态方法. 注意: **1.** 与类的静态成员变量一样,类的静态方法需要通过"类名.方法名"的方式来引用; **2.** 在静态方法内部,只能访问其所处类或者其所处类的基础类中的成员常量或者静态成员变量. 如:

其中定义了一个名为"测试方法"的静态方法,其中第一行代码同时访问了类中的"静态变量1"和"常量1",这是被允许的,然后后面的一行代码访问了非静态变量"变量1"(红色箭头指向处),这是不被允许的,编译器将报错.

3. 如果静态方法所定义第一个参数的数据类型为方法所处类本身而且没有指定参数匹配和需求类型(普通用户无需了解),那么可以基于该类对象以动态格式来调用该静态方法,编译器将自动进行转换:

譬如上面的程序在"测试类"中定义了一个名为"方法1"的静态方法,它第一个参数的数据类型为"测试类"自身,那么在程序中其它位置调用这个方法时,可以采用以下两种方式:

1. 测试类.方法1 (变量1, 1)

2. 变量1.方法1 (1)

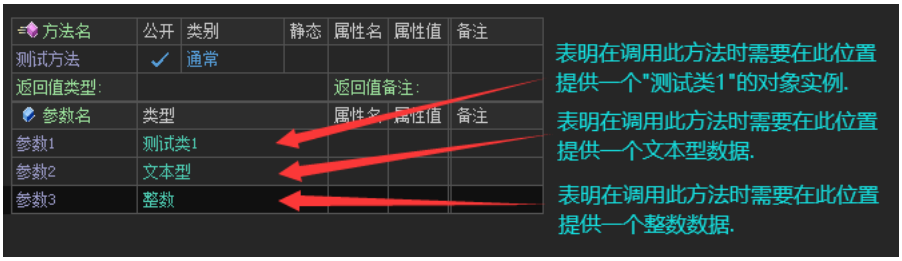
第一种方式是标准的静态方法访问方式,第二种就是优化后的方式,其第一个参数被移动到方法访问对象上,这种方式在编译时将被自动转换为第一种方式.

采用这种优化方式的具体要求为:

静态方法第一个参数的数据类型必须为其所处类本身.

只要满足这个要求,该静态方法被调用时第一个参数就可以被移动到其方法访问对象上,其后续参数正常填写.

类别	指定方法的具体类别,可以为以下几种之一(关于属性和事件类别方法后面有专门章节描述,此处了解一下就可以了): **1. 通常:** 表明本方法为通常方法 **2. 属性读:** 表明本方法为属性读取方法. 该属性可以在程序语句中被读取. 如果属性读方法为静态方法,所要求的定义格式: **A**. 必须定义且只能定义一个参数,该参数的数据类型必须为属性读方法所处类本身; **B**. 必须定义有返回值,该返回值的数据类型不能为数组,该数据类型即为本属性被读取时的数据类型. 如果属性读方法不为静态方法,所要求的定义格式: **A**. 不能定义参数; **B**. 必须定义有返回值,该返回值的数据类型不能为数组,该数据类型即为本属性被读取时的数据类型. 注意: **A**. 在程序中,必须以与变量相同的引用方式来访问属性读方法. 如: "类对象1.属性1"; **B**. 如果存在同名属性写方法,则两者的数据类型必须一致; **C**. 在全局类中不能定义属性读方法(普通用户无需掌握). **3. 属性写:** 表明本方法为属性写入方法. 该属性除了可以在程序语句中被赋值,还可以在该类对象变量的扩展属性表的"**属性**"列中被赋予初始值.

	如果属性写方法为静态方法,所要求的定义格式: **A.** 必须未定义返回值; **B.** 必须定义且只能定义两个参数,第一个参数的数据类型必须为属性写方法所处类本身,第二个参数的数据类型不能为数组,该数据类型即为本属性被写入时 所需要的数据类型. 如果属性写方法不为静态方法,所要求的定义格式: **A.** 必须未定义返回值; **B.** 必须定义且只能定义一个参数,该参数的数据类型不能为数组,该数据类型即为本属性被写入时所需要的数据类型. 注意: **A.** 在程序中,必须以与变量相同的引用方式来访问属性写方法. 如: "类对象1.属性1 = 1"; **B.** 如果存在同名属性读方法,则两者的数据类型必须一致; **C.** 在全局类中不能定义属性写方法(普通用户无需掌握). **4. 定义事件:** 定义本类对象将会发送事件的名称及格式,此种方法必须满足以下格式要求: **A.** 方法体必须为空; **B.** 返回值数据类型必须为整数(安卓和服务器子平台无此限制); **C.** 不能为静态方法. **5. 接收事件:** 定义本类对象将会接收本类中成员变量对象所发送的事件,此种方法且必须满足以下要求: **A.** 返回值数据类型必须为整数(安卓和服务器子平台无此限制); **B.** 不能为静态方法; **C.** 方法名称格式必须为: 事件对象类名 + "_" + 欲接收事件名; **D.** 方法的第1个参数的数据类型必须为欲接收其事件的事件对象类名; **E.** 方法的第2个参数的数据类型必须为整数,用作接收"挂接事件"关键字调用所提供的"标记值"参数值(非该方式挂接事件则此参数值固定为0); **F.** 方法其余参数的数目及数据类型必须与欲接收事件的定义方法一致. 注: 开发环境中有对应的快捷功能自动生成指定事件的接收方法.
公开	指定本方法是否公开.公开方法可以在所处类外部被访问,而非公开方法只能在本类或者其继承类中访问.
参数表	为方法定义一个参数表,用作指定调用此方法时所需要提供的参数:  需要注意的是: **1.** 数据类型为**类**或**文本型**的参数始终以参考方式传递对象(这一点上视窗和安卓平台是一致的). 譬如上图的参数1和参数2,如果在外部调用"测试方法 (测试类1对象, 文本变量1, 123)",那么"参数1"将是指向"测试类1对象"的参考,操作"参数1"等效于操作"测试类1对象", "参数2"将是指向"文本变量1"的参考,操作"参数2"等效于操作"文本变量1"; **2.** 调用方法时,必须加上用小括号括住的参数表,即使该方法的参数表为空,也必须加上用小括号括住的空参数表. 如: 假设前面的"测试方法"没有定义任何参数,调用它时也必须使用"测试方法()"格式.

7. 类成员属性

类成员属性用作表达或修改类的特征时使用,可以使用类方法或者类成员变量来定义.

根据其可访问方式分为3类: "**可读成员属性**" / "**可写成员属性**" / "**可读写成员属性**".

1. 定义"可读成员属性":

"**可读成员属性**"用作支持对属性值的读取.

插入一个方法,将其"**类别**"列设置为"**属性读**",然后保证其满足以下格式要求:

A. 访问权限必须为"**公开**";

B. 不能为静态方法(实际上静态方法也可以用作定义属性,不过这个不需要普通用户掌握);

C. 没有参数;

D. 必须定义有一个返回值,该返回值的数据类型即为该属性的**数据类型**.

譬如以下代码在"**测试类**"中定义了一个数据类型为"**整数**"的可读属性"**我的属性**":

注意: 虽然"**我的属性**"是以成员方法的形式定义的,但是在程序中访问时不能以方法的访问方式"**对象.我的属性()**"来访问,而应该以访问"**成员变量**"的方式来访问,即:"**对象.属性名**".

如,访问上面可读属性的语句为:

其中黄色箭头指向处定义了"**测试类**"的一个对象实例,红色箭头处读取了其"**我的属性**"的属性值并将其赋值到"**整数变量1**".

实际上,读取"**测试类对象.我的属性**"时,等效于调用了"**我的属性**"属性读取方法.

2. **定义"可写成员属性":**

顾名思义,"**可写成员属性**"就是用作支持对属性值的写入.它的定义方式与定义"**可读成员属性**"类似,只是所定义方法的格式要求不同.

插入一个方法,将其"**类别**"列设置为"**属性写**",然后保证其满足以下格式要求:

A. 访问权限必须为"**公开**";

B. 不能为静态方法(实际上静态方法也可以用作定义属性,不过这个不需要普通用户掌握);

C. 只有一个参数,该参数的数据类型即为该可写属性的**数据类型**;

D. 没有返回值.

譬如以下代码在"**测试类**"中定义了一个数据类型为"**整数**"的可写属性"**我的属性**":

同样,虽然此处"**我的属性**"是以成员方法的形式定义的,但是在程序中访问时不能以方法的访问方式"**对象.我的属性(欲写入的属性值)**"来访问,而应该以访问"**成员变量**"的方式来访问,即:"**对象.属性名 = 欲写入的属性值**".

如,访问上面可写属性的语句为:

其中黄色箭头指向处定义了"**测试类**"的一个对象实例,红色箭头处将其"**我的属性**"的属性值赋值为123.

实际上,写入"**测试类对象.我的属性**"时,等效于调用了"**我的属性**"属性写入方法.

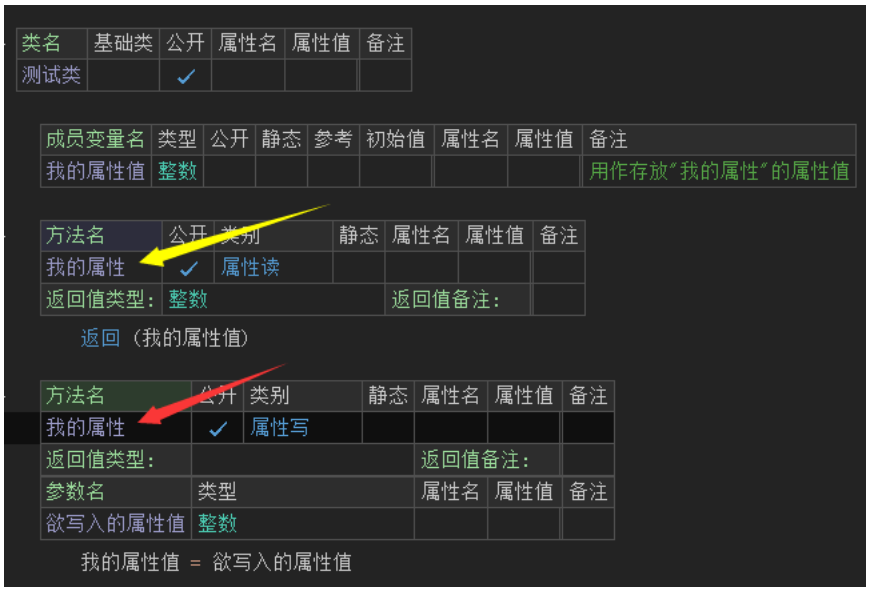
"**可写成员属性**"还有另外一种访问方式,就是直接在对象的属性表中设置,如:

等效于前面通过语句"测试类对象.我的属性 = 123"对该属性的写入.

操作小提示:要想知道当前对象有哪些"**可写属性**",在"**属性名**"列上**按下空格**即可:

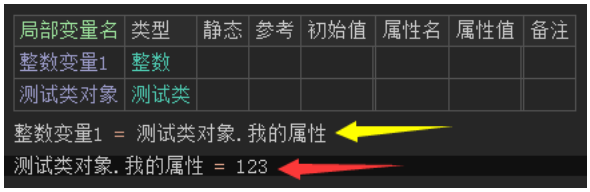
3. **定义"可读写成员属性":**

很容易理解,"**可读写成员属性**"就是既可以**被读取**又可以**被写入**的属性,定义它的方式也很简单,就是同时定义名称相同的**属性读**和**属性写**方法,如:



其中黄色箭头指向定义了"**我的属性**"的**属性读**方法,红色箭头指向定义了"**我的属性**"的**属性写**方法.

前面已经讲过,"**可读属性**"用作支持对类属性的**读取**操作,"**可写属性**"用作支持对类属性的**写入**操作,编译器会根据当前操作是读取还是写入自动调用对应的**属性读/写方法**.当以以下方式对"**我的属性**"进行访问时:



黄色箭头指向处将调用"**我的属性**"的属性读方法,而红色箭头指向处将调用"**我的属性**"的属性写方法.

定义"**可读写成员属性**"时需要注意的是:

"**属性读方法**"和"**属性写方法**"的数据类型必须一致,也就是说: "**属性读方法**"的返回值与同名"**属性写方法**"第一个参数的数据类型必须一致:

如图中黄色和红色箭头所指向的数据类型必须一致.

4. 定义"可读写成员变量属性":

有时候我们想直接把一个"**成员变量**"同时定义为"**成员属性**",譬如前面的例子所定义的"**我的属性**"也可以这样定义:

具体方法就是为该成员变量设置属性值为**真**的"**@属性变量**"系统属性(系统属性是指以'**@**'字符开头的系统预定义属性)即可.

操作小提示:要想知道当前对象有哪些"系统属性",在"**属性名**"列上输入'**@**'字符,要想知道当前对象有哪些"用户程序属性",在"**属性名**"列上输入**空格**字符:

注意:这些系统属性,除了在本文档中提到的,一般用户无需了解.

此处定义的"**我的属性**",与前面通过"**属性读/写方法**"定义的"**我的属性**"没有任何区别,同样可以在对象的属性表中使用:

那么问题来了,既然可以这么简单地定义成员属性,那么为什么还要那么麻烦去通过"**属性读/写方法**"来定义呢?

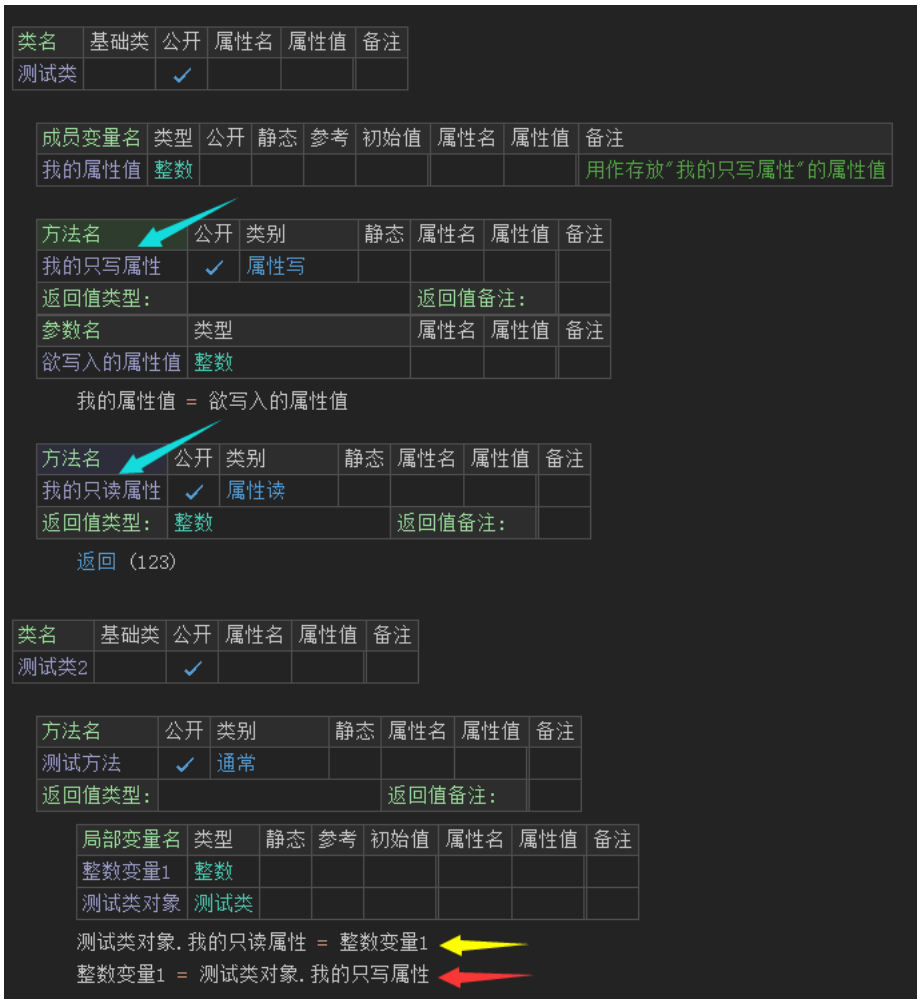
道理很简单:通过"**属性读/写方法**"来定义可以使用程序代码对属性的**读/写**操作进行具体控制,还可以进行一些额外的特定操作,而定义"**成员变量属性**"就没办法达到这个目的了,本处两者效果一致只是一个特例.

5. 定义"只读成员属性"和"只写成员属性":

定义"**只读成员属性**"和"**只写成员属性**"很简单:

只提供了"**属性读方法**"的属性就是**只读属性**,只提供了"**属性写方法**"的属性就是**只写属性**.

如果想对"**只读成员属性**"进行写操作,或者想对"**只写成员属性**"进行读操作,编译器都会报错:



如上图,在"**测试类**"中定义了名为"**我的只读属性**"的只读属性(未提供该名称的属性写方法)和名为"**我的只写属性**"的只写属性(未提供该名称的属性读方法),那么在黄色和红色箭头所指向处的代码在编译时都会报错.

可以通过此方式对属性的读写权限进行分别控制.

6. 填写成员属性表:

火山中所有定义型成员(包/类/方法/常量/变量/参数)都具有一个属性表(由紧挨着的"**属性名**"和"**属性值**"两列组成),用作指定该成员相关属性的值.

A. 以下属性可以在所有成员的属性表中被使用:

1. 火山系统定义的全局扩展属性或项目插件定义的项目扩展属性(属性名以'@'开头,普通用户无需掌握).

B. 以下属性可以在类的属性表中被使用,用作在类对象被创建时自动初始化相关属性值:

1. 类自身或其基础类中定义的可读写成员变量属性;
2. 类自身或其基础类中定义的所有属性写方法;
3. 以上属性的**子属性**,如"可读属性1.可读属性2.可写属性3",前面的父属性必须均为可读取属性,最后一个属性必须为可写入属性.

C. 以下属性可以在类成员变量的属性表中被使用:

1. 类成员变量的数据类型类或其基础类中定义的可读写成员变量属性;
2. 类成员变量的数据类型类或其基础类中定义的所有属性写方法;
3. 以上属性的**子属性**,如"可读属性1.可读属性2.可写属性3",前面的父属性必须均为可读取属性,最后一个属性必须为可写入属性.

D. 属性值:

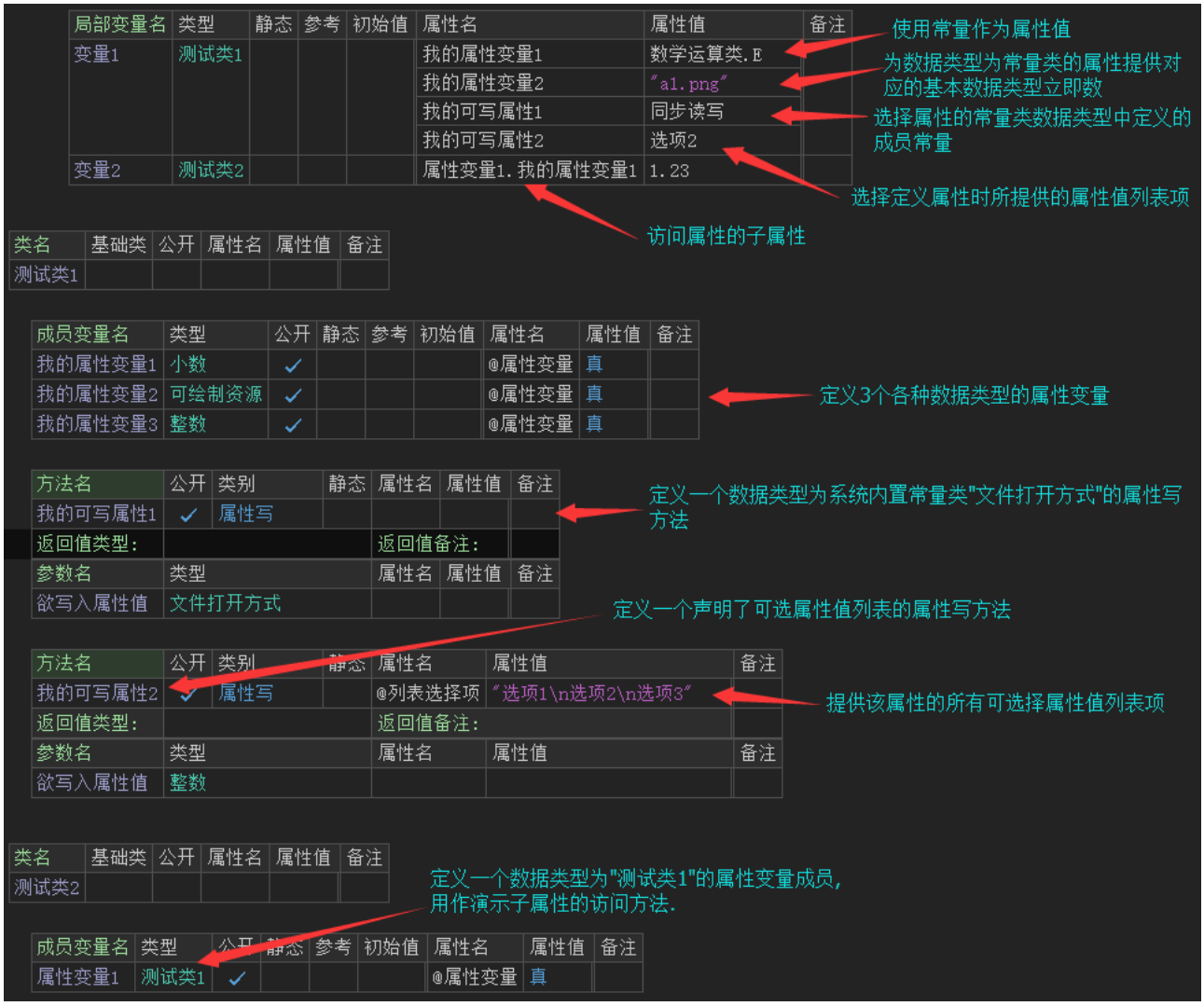
1. 属性值可以引用程序中定义的常量或者提供对应数据类型的立即数;
2. 如果属性指定只能从其提供的选择列表中选择属性值(使用"@列表选择项"系统全局属性指定,普通用户知道即可,无需掌握),则属性值只能从这些列表项中选择;
3. 如果属性数据类型为常量类:

   A. 如果该常量类中定义有常量成员,则属性值只能从这些常量成员中选择;

   B. 否则可以直接提供该常量类所对应数据类型的立即数. 譬如,假设属性A的数据类型为"可绘制资源"(安卓平台)或"位图资源"(视窗平台),那么属性值除了可以为其提供一个同样数据类型的常量以外,还可以直接为其提供一个 对应图片文件名文本.

   注意: 普通用户只需要知道当属性的数据类型为系统类库中所提供的常量类时,可以如此处理即可,其它方面(譬如如何建立常量类),无需掌握,可以将其当前普通的类同样看待.

属性表填写实例如下图:



下图为所使用到的安卓平台系统类"文件打开方式"的定义,了解一下即可:

类名	基础类	公开	属性名	属性值	备注
文件打开方式		✓	@文档	"category = \"辅助类\""	提供打开文件的可用方式
			@常量类	文本型	

常量名	类型	初始值	公开	属性名	属性值	备注
只读		"r"	✓			以只读的方式打开文件，任何对文件进行的写操作都将失败.
读写		"rw"	✓			以读写的方式打开文件，如果所欲打开文件不存在，将自动创建.
同步读写		"rws"	✓			以同步读写的方式打开文件，一旦对文件内容或文件信息(譬如文件的创建时间/文件的最后修改时间/文件拥有者/文件访问权限等等)进行了修改操作，将立即同步更新到相应设备，以确保所进行的修改操作不会因为缓存而丢失.
同步读写2		"rwd"	✓			以同步读写的方式打开文件，与"rws"模式的区别是，该模式仅立即同步对文件内容的修改，而不同步对文件信息的修改.

8. 类成员事件

成员事件用来类对外发送通知时使用.

一个很简单的例子: 用作"**按钮**"的类必须在用户单击按钮时向外部发送"**被单击**"事件,用作"**时钟**"的类必须向外部定时发送"**时钟周期**"事件,等等.

类的其它三类"**成员变量**" / "**成员属性**" / "**成员方法**"都是被动接受来自外部的访问,而"**成员事件**"是主动向外部发送通知,这是两者之间的最主要不同.

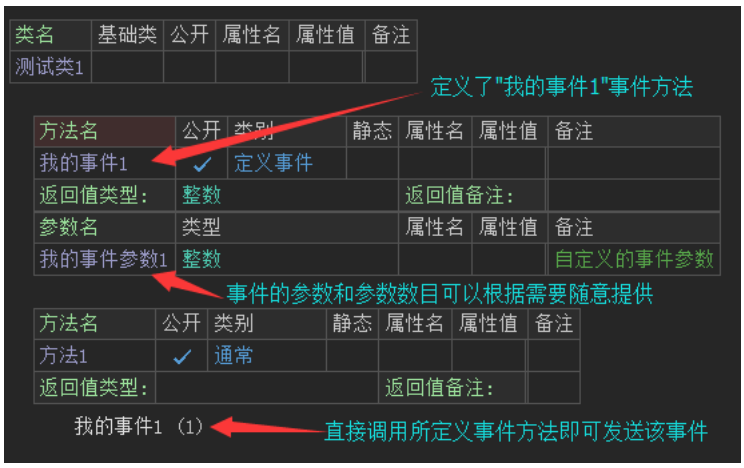
A. 定义"**成员事件**":

插入一个方法,将其"**类别**"列设置为"**定义事件**",然后保证其满足以下格式要求:

1. 访问权限必须为"**公开**";
2. 不能为静态方法;
3. 返回值必须为整数;
4. 方法体必须为空.

事件定义方法对参数表没有要求,用户可以根据自己的需要随意定义.

譬如下图我们为"测试类1"定义了一个"我的事件1"事件:



B. 在类中发送事件:

在类中的代码内,当需要发送事件时,直接调用该事件的"**事件定义方法**"即可. 如上图.

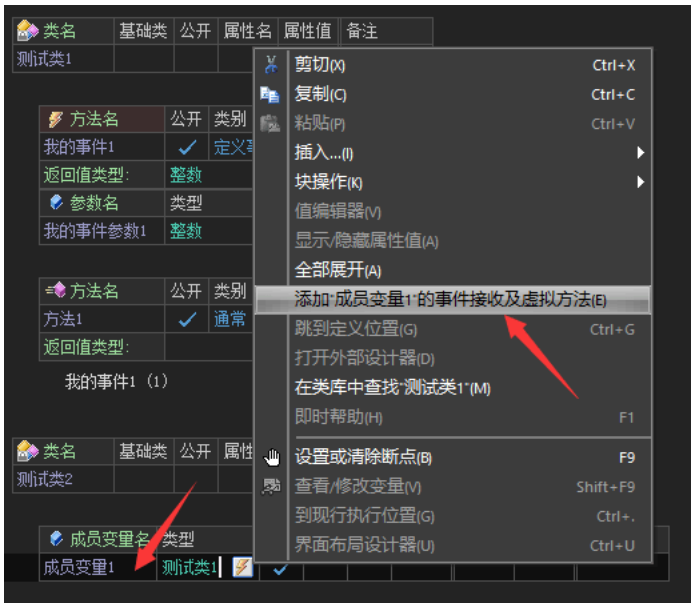
当调用"**事件定义方法**"时,如果该事件定义方法上挂接了对应的"**事件接收方法**"(见下),会自动去调用该"**事件接收方法**"并返回其所返回的整数值,否则会直接返回整数值0.

C. 接收其它类所发送过来的事件:

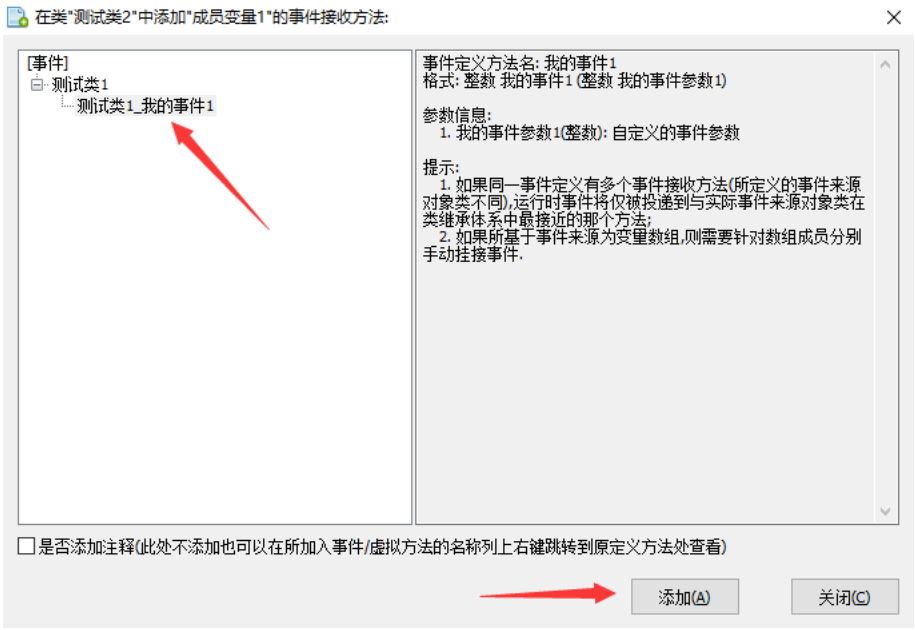
要想接收其它类所发送的事件,必须首先定义相应的**事件接收方法**,譬如以下代码在"**测试类2**"中定义了一个前面的"**测试类1**"的对象变量:

想要接收其"**我的事件1**"事件,需要如下操作:

鼠标右键单击"**成员变量1**"的定义行:



选择其中的"**添加成员变量1的事件接收及虚拟方法方法**"菜单项:



再选择其中的"**测试类1_我的事件1**",然后单击"**添加**"按钮,会自动在程序中插入对应的事件接收方法:

当然,你也可以自己手工创建并填写符合此格式的事件接收方法,效果是一样的.

查看上面所生成的事件接收方法,可以发现它的格式要求:

1. 方法名称必须为: "事件定义方法所处类名"+下划线+"事件定义方法名称";
2. 不能为静态方法;
3. 方法的第一个参数必须为固定的"**来源对象**"参数,其数据类型为发送事件的类,用作提供具体是哪个对象发送过来的事件;
4. 方法的第二个参数必须为固定的"**标记值**"参数,其数据类型为整数,用作动态挂接事件时使用(见后);
5. 方法的后续参数表必须与对应的"**事件定义方法**"一致,用作提供在事件定义方法所处类中发送事件(调用该事件定义方法)时所传递过来的具体参数值;
6. 方法的返回值必须为整数,此返回值将被传递回在事件定义方法所处类中调用该事件定义方法的调用方.

一旦为**类成员变量**对象的"**事件定义方法**"定义了对应的"**事件接收方法**",那么该事件就被自动挂接到了此接收方法上,在事件定义方法所处类中一旦调用该"**事件定义方法**",此"**事件接收方法**"就会被自动调用.

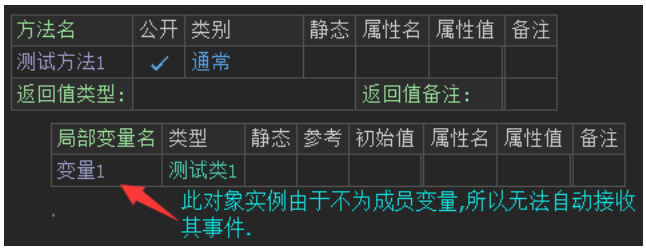
注意,由于同一类的同一事件均被发送到同一个事件接收方法,因此必须充分使用"**来源对象**"和"**标记值**"两个参数进行区分. 如:

如上图,"成员变量1"和"成员变量2"由于其类型都为"测试类1",因此收到其"我的事件1"后,都会调用"测试类1_我的事件1"事件接收方法,在该方法中可以通过判断具体来源对象来进行区分处理.

附: 这种特性是不是比其它编程语言更强大? 不再是只有在被设计窗体上的窗口组件才能发送事件了,也不再需要为了让其能发送事件去开发 专用的窗口哑组件了.在火山中,任何代码位置处的**对象**均可以发送事件.

D. 动态挂接其它类所发送过来的事件:

如前所述,一旦为**类成员变量**对象的"**事件定义方法**"定义了对应的"**事件接收方法**",那么该事件就被自动挂接到了当前类中的对应"**事件接收方法**"上,但是其它代码位置处对象的事件是不会自动挂接到当前类中的,譬如下面代码:



在"**测试方法1**"中定义了一个"**测试类1**"的局部变量对象,此时该对象上的"**我的事件1**"事件是不会自动挂接到当前类的"**测试类1_我的事件1**"事件接收方法上的.也就是说,当前类此时将无法接收到来自此"**测试类1**"对象实例的"**我的事件1**"事件.

如果需要接收该局部变量对象的事件,必须调用"挂接事件"关键字明确挂接其事件到当前类:

"**挂接事件(变量1)**"语句被执行后,此"**测试类1**"对象的"**我的事件1**"事件就被挂接到了当前类的"**测试类1_我的事件1**"事件接收方法上,以后当前类就可以接收到来自该对象的"**我的事件1**"事件了:

在调用"挂接事件"关键字时可以额外再提供一个标记值参数,如: "**挂接事件 (变量1,123)**",此时该"**测试类1**"对象一旦发送事件,事件接收方法的"标记值"参数将接收到此处所提供的标记值"**123**",便于程序中对此事件进行特定处理.

9. 虚拟方法

虚拟方法为可以在继承类中将其覆盖的方法.

对于普通用户来说,不需要掌握如何定义虚拟方法,只需要了解如何覆盖系统类中定义好的虚拟方法即可.

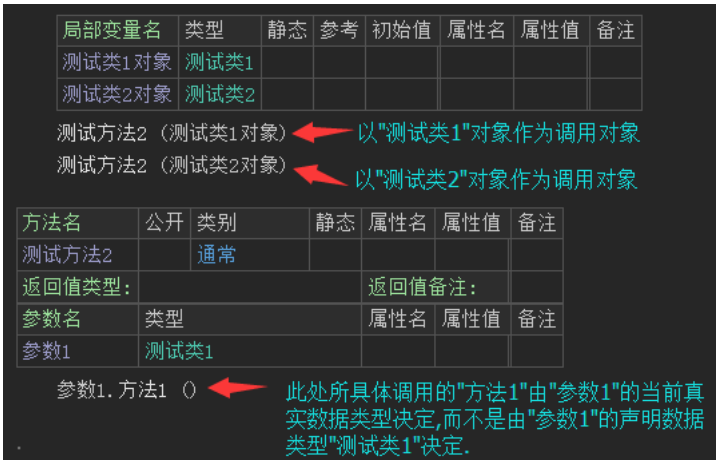
虚拟方法相比普通方法的不同之处: 虚拟方法被调用时,所调用到的实际方法由调用对象的**运行时真实数据类型**而不是其**声明时数据类型**决定.

听起来挺拗口,而且不好理解,不要紧,下面我们来逐步讲解.

假设存在以下两个类: "测试类1"和"测试类2",其中"测试类1"是"测试类2"的基础类,同时这两个类中都定义了一个名叫"**方法1**"的方法:

这两个方法的定义格式完全相同(具有相同的名称/返回值数据类型/参数表),而且均定义了"**@虚拟方法**"属性.这就是"虚拟方法"名词的来由,凡是定义了"@虚拟方法"属性的方法,均称为**虚拟方法**.

使用下面这段代码对这两个方法进行调用:



在"测试方法2"中对"方法1"进行了调用,按照常理来说,由于"参数1"的数据类型为"测试类1",因此应该始终是调用"测试类1"中定义的"方法1"才对.如果"方法1"不是虚拟方法,这种假设是对的,但是由于其是虚拟方法,所以此处所具体调用的是哪个类里面的"方法1"由"参数1"的当前运行时实际数据类型决定,即: "参数1"如果指向的是"测试类1"的对象,则调用的就是"测试类1"的"方法1",如果指向的是"测试类2"的对象,则调用的就是"测试类2"的"方法1".

很明显,"测试方法2 (测试类1对象)"语句调用的是"测试类1"中的"方法1",而"测试方法2 (测试类2对象)"语句调用的是"测试类2"中的"方法1".

使用虚拟方法有什么好处? 最重要的好处就是可以在继承类中通过虚拟方法覆盖改写基础类中的同名方法,此特性可以用作以下几个方面:

1. **修改基础类中的原有功能**

   注意: 本条普通用户只需稍作了解即可.

   考虑一下这个应用场景,在程序中一直使用来自第三方库的名为"测试类1"的类,我们发现其"方法1"不能满足需要而我们又无法去直接修改它,如果该方法为虚拟方法,则我们可以建立一个"测试类1"的继承类"测试类2",并将其"方法1"覆盖,然后将所有创建"测试类1"对象的代码均改为创建"测试类2"的对象即可.

2. **建立多态类**

   注意: 本条普通用户只需稍作了解即可.

   虚拟方法可以用作实现基于同一基础类的继承类表现出不同的行为.

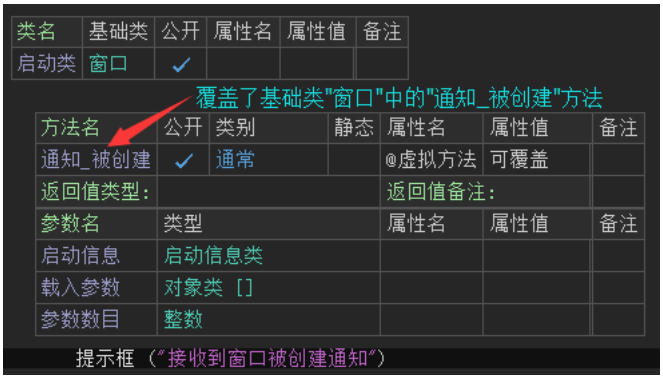
   假设有一个画板类,其中可以绘制各种图形(如三角形/圆/矩形等),按照面向对象的设计方法,我们可以定义一个名为"图形类"的基础类,该类定义了一个名为"绘制"的虚拟方法,然后再定义"图形类"的各种继承类,如"三角形类","圆形类","矩形类"等等,在这些继承类中,将基础类中的"绘制"方法覆盖以实现其特定的绘制工作.在画板类中进行绘制时,接收一个"图形类"的对象,调用其"绘制"方法即可绘制各种不同的图形,而不用管具体接收的是哪种图形类对象.

3. **在基础类中调用位于继承类中的方法**

   火山类库广泛使用本特性向用户继承类发送通知,因此本条普通用户必须了解.

   由于火山安卓平台的类库中虚拟方法的应用场合较多,下面以该平台程序作为例子进行说明.

   这是一个非常简单的安卓程序,用作在所处窗口被创建时显示一个提示框:

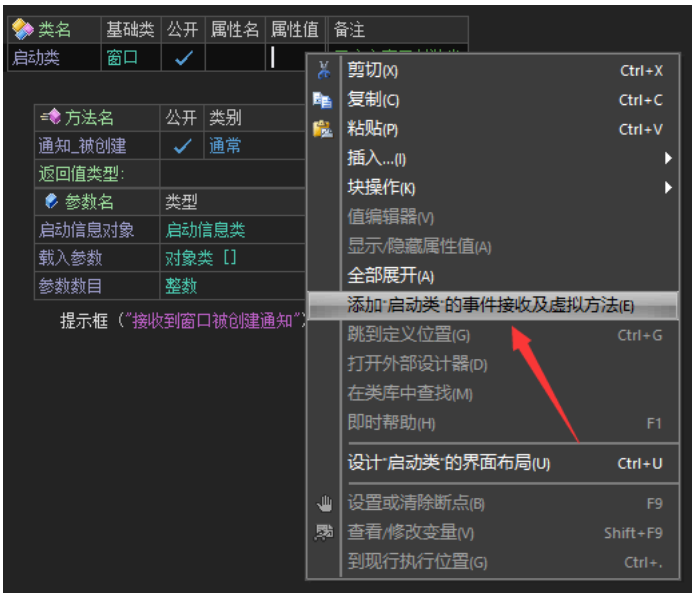


"启动类"为系统类"窗口"的继承类,其中覆盖了该类中定义的虚拟方法"通知_被创建":

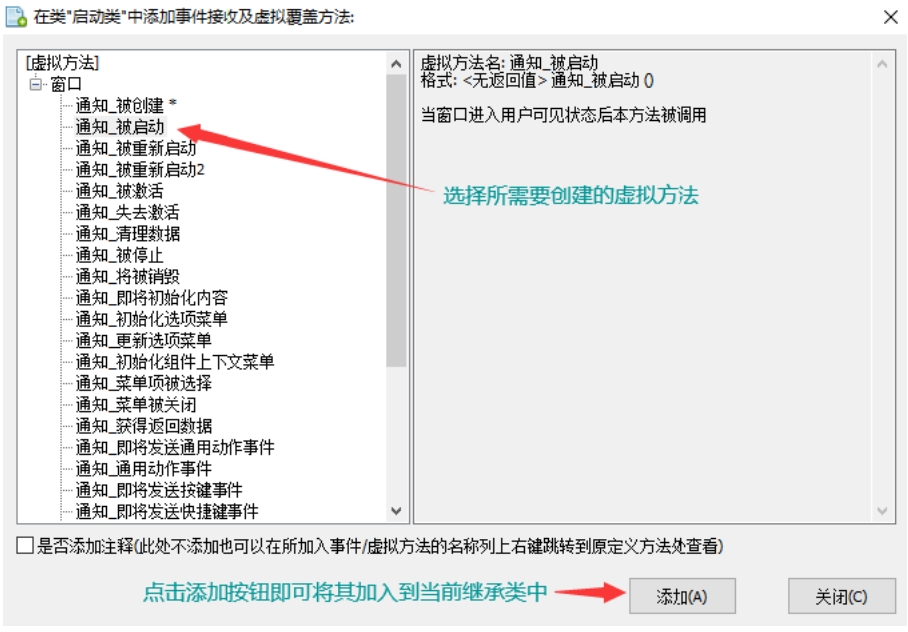
方法名	公开	类别	静态	属性名	属性值	备注
通知_被创建	✓	通常		@虚拟方法	可覆盖	当窗口被创建后本方法被调用 本方法被调用后将紧跟着调用"通知_被启动"方法
返回值类型:				返回值备注:		
参数名	类型			属性名	属性值	备注
启动信息	启动信息类					提供载入本窗口时所传递过来的启动信息, 必定不为空对象.
载入参数	对象类 []					提供载入本窗口时所传递过来的所有参数("载入窗口2"方法传递) 注意: 载入本窗口时所传递过来的所有基本数据类型参数均已经被自动转换为对应的封装类对象, 所以取用时需要使用对象类的对应"对象到xxx"方法将其转换回去.
参数数目	整数					提供"载入参数"数组的成员数目, 如果为0, 则"载入参数"可能为空对象.

当安卓程序启动后,将自动创建用户的"启动类"窗口实例对象,当该窗口被创建时,系统类中将自动调用其"通知_被创建"方法,由于窗口实例对象的真实数据类型为"启动类",因此此时所实际调用的是用户"启动类"中的"通知_被创建"方法,从而达到了在窗口被创建时通知用户程序的目的.

在火山开发环境中,可以执行以下菜单功能来自动创建用作基础类中的虚拟覆盖方法,在继承类中的任意位置点击右键,选择"添加xxx的 事件接收及虚拟方法"菜单项:



选中所需要创建的虚拟覆盖方法,然后点击"添加"按钮:



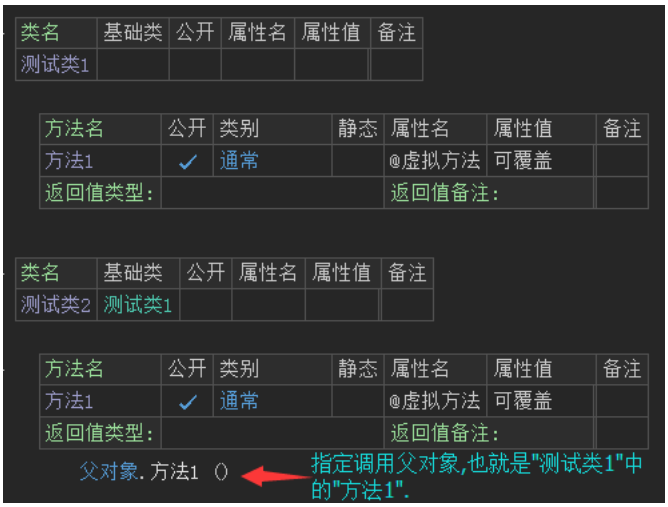
即可将对应的虚拟覆盖方法加入到当前类中,下图为添加后的结果:

方法名	公开	类别	静态	属性名	属性值	备注
通知_被启动	✓	通常		@虚拟方法	可覆盖	
返回值类型:				返回值备注:		

```
父对象.通知_被启动 ()
```

最后在所添加的方法体中加入自己的处理代码即可.

注意,在继承类的虚拟方法中,有时候想调用基础类中被覆盖的虚拟方法,可以采用类似以下语句调用:



10. 方法语句

程序语句在方法中使用,用作提供具体的方法实现代码.

下面为程序语句的简单概念,稍作了解即可:

程序语句可以使用表达式进行描述,表达式由使用操作符组合在一起的语句单元组成,分为以下几类:

1. 一元表达式,如: -变量1, (长整数)1
2. 二元或多元表达式,如: 变量1 + 变量2, 变量1 * 123 * 变量2
3. 命令或方法调用表达式,如: 方法1 (123), 如果 (变量1)
4. 表达式可以嵌套,如: -(变量1 + 变量2), 如果 (变量1 == 真)

表达式中的语句单元可以是: 数据类型/变量/常量/参数/方法/名称类关键字/立即数 等.

A. 语法格式描述文本的规则:

语法格式描述文本为由以下运算符组合的一个或多个语法项构成,这些运算符的优先级按顺序从小到大递增:

| : 备选项(在多个项目中选择其中任意一个).
() : 分组项(提供一组备选项).
[] : 可选存在项(可以存在也可以不存在).
{} : 重复存在项(可以重复存在0到n次).

语法项分为字面文本项(用双引号括住)和语法替换项两种.

B. 语句行的语法格式描述文本:

语句: 赋值表达式 | 调用表达式
表达式: 一元表达式 | 表达式 二元操作符 表达式
一元表达式: 基本表达式 | 一元操作符 一元表达式
基本表达式: 操作数 | 类型强转表达式 | 对象成员访问表达式 | 数组成员访问表达式 | 调用表达式
操作数: 立即数 | 名称 | "(" 表达式 ")"
赋值表达式: (局部变量名称 | 所处方法参数名称 | 对象成员访问表达式 | 数组成员访问表达式) "=" 表达式
类型强转表达式: "(" 类型名称 ")" 一元表达式
对象成员访问表达式: 类成员名称 | 类访问名称 "." 类成员名称 | 基本表达式 "." 类成员名称 | "父对象" "." 父类成员名称 | 本对象
数组成员访问表达式: 基本表达式 "[" 表达式 "]" { "[" 表达式 "]" }
调用表达式: 对象成员访问表达式 调用参数表 | 命令关键字名称 调用参数表
一元操作符: "-"
二元操作符: "&&" | "||" | "==" | "!=" | "<" | "<=" | ">" | ">=" | "属于" | "*" | "/" | "%" | "+" | "-"
调用参数表: "(" [表达式 { "," 表达式 }] ")"
类型名称: 基本类型名称 | 类访问名称
基本类型名称: 字节 | 短整数 | 字符 | 整数 | 变整数 | 长整数 | 单精度小数 | 小数 | 逻辑型 | 文本型
类访问名称: 类的定义名称 | 类所处包名 "." 类的定义名称

注意: 在火山程序中,没有**流程线**,只有**子语句体**. 即:

1. 任何程序语句都可以拥有一条或多条下属语句,这些下属语句称为该语句的**子语句体**,该语句称为这些下属语句的**父语句**;
2. 子语句体可以嵌套. 也就是说,子语句可以继续拥有子语句体.

如图:

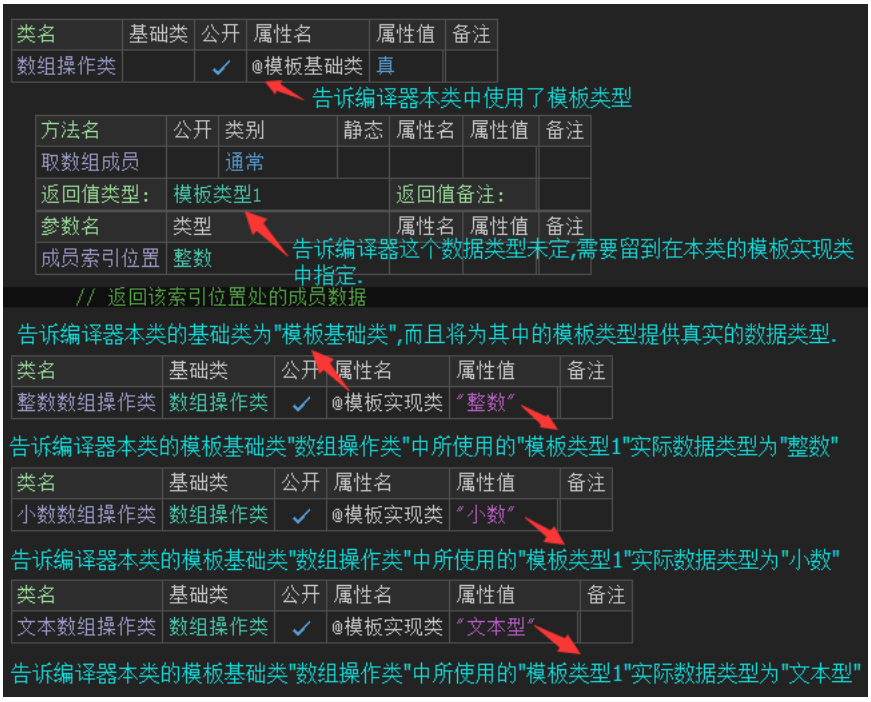
其中,所有子语句体与其所处的父语句之间均使用虚线括住标注出来了. 即:

语句行12-16是语句行11的子语句体;
语句行15-16是语句行14的子语句体;
语句行19-26是语句行18的子语句体;
语句行22-23是语句行21的子语句体;
语句行25-26是语句行24的子语句体.

在使用虚线标注子语句体的时候,系统对"**如果**"和"**否则**"语句进行了特殊处理,将两者的子语句体线连接起来了,这样看起来 结构更清晰一些.

三. 火山平台关键字:

1. 基本数据类型关键字:

名称	输入字1	解释
字节	sbyte	字节(有符号)基本数据类型,有效值范围从-128到127,占用1个字节空间. **特例:** 在服务器子平台中,本数据类型无符号,即有效值范围从0到255.
短整数	short	短整数基本数据类型,有效值范围从-32768到32767,占用2个字节空间.
字符	wchar	宽字符基本数据类型,有效值范围从0到65535,占用2个字节空间. **特例:** 在服务器子平台中,本数据类型等同于32位整数,占用4个字节空间,有效值范围从-2147483648到2147483647.
整数	int	整数基本数据类型,有效值范围从-2147483648到2147483647,占用4个字节空间. **特例:** 在编译64位服务器子平台程序时,有效值范围等同于长整数,占用8个字节空间. 在服务器子平台中如欲限定使用32位整数请使用"字符"数据类型等同替代.
变整数	vint	变整数基本数据类型,在编译64位视窗子平台程序时等同于长整数类型,编译32位视窗子平台程序时等同于整数类型,在其它情况下均等同于整数类型.
长整数	long	长整数基本数据类型,有效值范围从-9223372036854775808到9223372036854775807,占用8个字节空间.
单精度小数	float	单精度小数基本数据类型,有效值范围从-3.40E+38到3.40E+38,占用4个字节空间.
小数	double	双精度小数基本数据类型,有效值范围从-1.7E+308到1.7E+308,占用8个字节空间.
逻辑型	bool	逻辑型基本数据类型,有效值为真/假.
文本型	string	文本型基本数据类型,用作记录一段字符串文本.
模板类型1 -> 模板类型8		注意: 本关键字普通用户只需要了解一下即可. 模板数据类型,只能在模板基础类中使用. 模板数据类型本身并不是真实存在的数据类型,所对应的真实数据类型需要由模板实现类来提供. 我们举个例子,假设我们需要开发一个数组操作类,用作支持对各种数组类型进行操作,如果不使用模板类型,那么我们需要写很多类似的类: 整数数组操作类, 小数数组操作类, 文本数组操作类 ..... 而这些类的处理代码是完全一样的,唯一不同之处就是其所操作的数据类型不同,使用模板类型,我们可以把所有代码都放到模板基础类里面,然后定义对应的模板实现类即可. 如:  这样能大大减少工作量,也能让程序具有更好的可维护性(只需维护一处即可).

注释:

1. "**输入字**"的意思是在程序中还可以通过输入此文本来输入该关键字.

使用图例:

下图为一个定义了各种基本数据类型变量的程序:

成员变量名	类型	公开	静态	参考	初始值	属性名	属性值	备注
字节变量	字节							
短整数变量	短整数							
字符变量	字符							
整数变量	整数							
长整数变量	长整数							
小数变量	小数							
逻辑型变量	逻辑型							
文本型变量	文本型							

说明:

A. "字节/短整数/字符/整数/长整数/小数"这几种数据类型被统称为数值数据类型,它们按照所能容纳数值范围的大小(也称为**容量**)从小到大顺序排列为:

字节 < 短整数 < 字符 < 整数 < 长整数 < 小数

B. 如何确定数值计算表达式的结果数据类型:

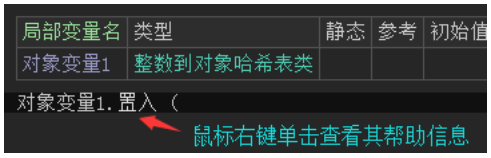
数值计算表达式的结果数据类型确定方法为:

为数值计算表达式中具有最大容量的数值数据类型

假设有一个数值计算表达式: "变量1 + 变量2", 其中"变量1"的数据类型为整数,"变量2"的数据类型为小数,那么这个表达式计算后结果的数据类型就是其中最大容量的数据类型: 小数.

C. 如何在帮助中查看模板类型的真实数据类型:

假设有下面这段代码:



当你右键单击未语句行上的"**置入**"方法,然后选择查看其帮助菜单项后,看到帮助页面中参数部分为以下内容:

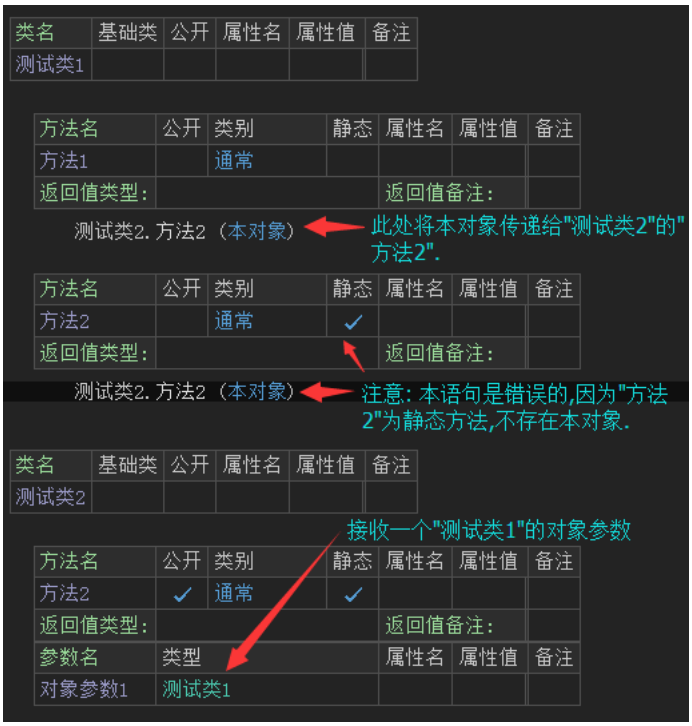
你一定会感到困惑,"模板类型1"和"模板类型2"所对应的真实数据类型到底是什么呢?

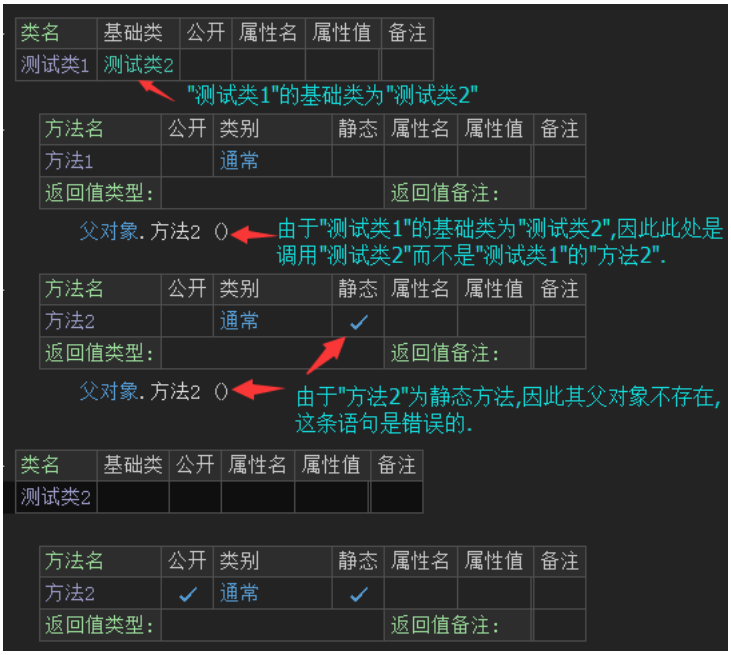
很简单,请继续单击该"模板类型1"/"模板类型2"链接,譬如点击前者将看到如下页面:

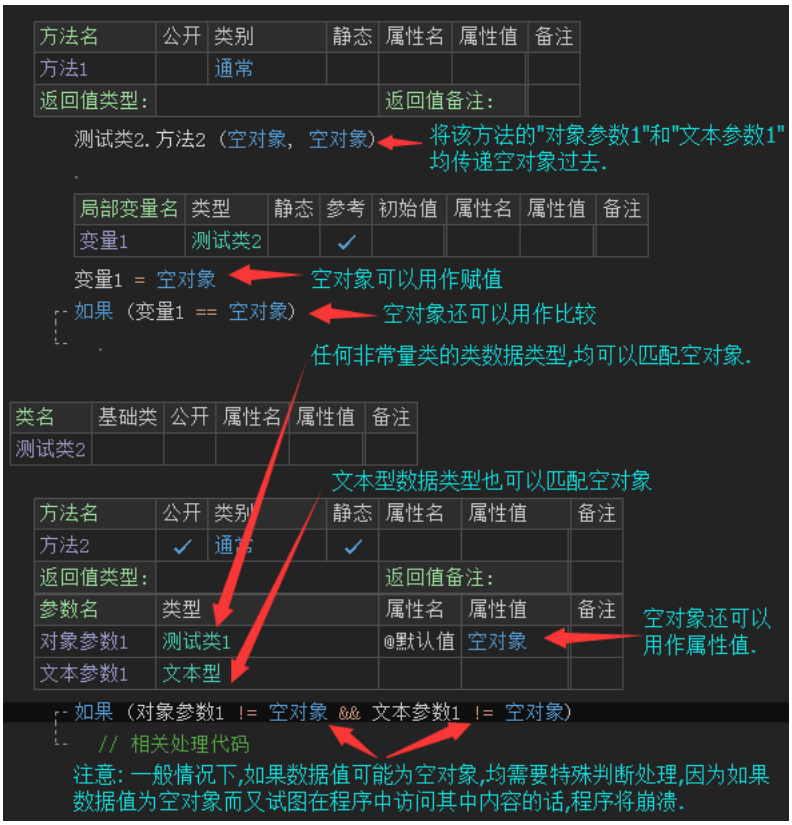
根据"对象变量1"的当前数据类型"整数到对象哈希表类",就可以得知,"模板类型1"当前所对应的真实数据类型为"整数类",这就是"整数到对象哈希表类"为其模板基础类中的"置入"方法第一个参数 提供的真实数据类型.

2. 名称关键字:

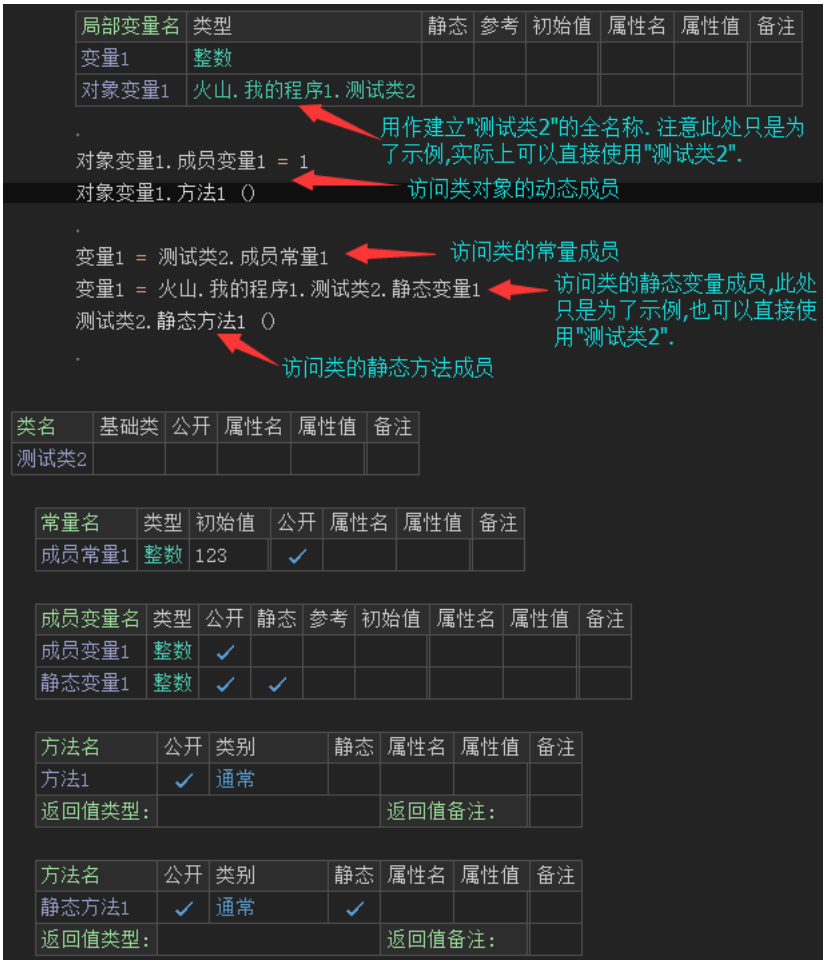
名称	输入字1	解释
对象名称关键字:		
本对象	this	用作在程序语句中代表所处类的对象本身. **注意:** 如果语句所处方法为静态方法,由于此时根本不存在当前类的实例对象,所以不能使用本关键字.

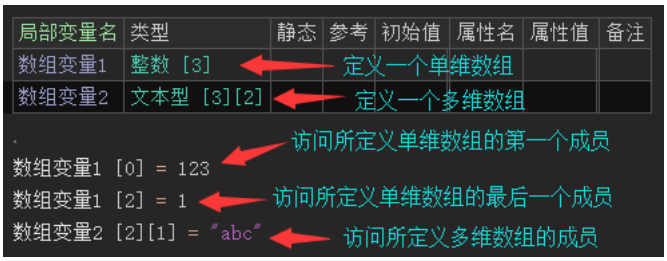
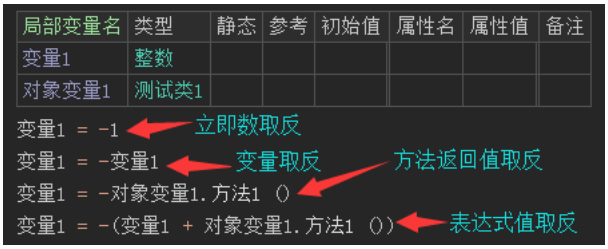
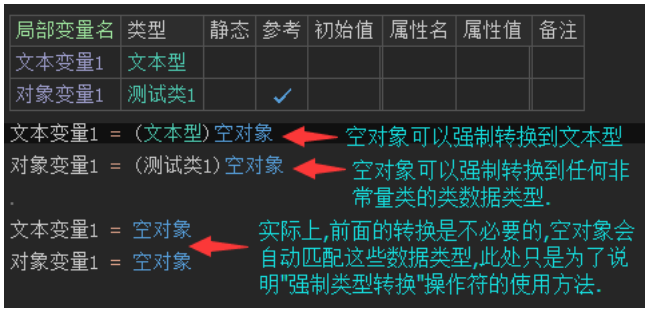
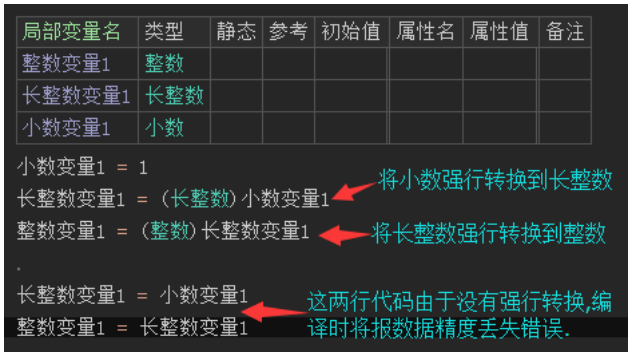


父对象	super	用作在类成员方法中代表所处类的父类对象. **注意:** 如果语句所处方法为静态方法,由于此时根本不存在当前类的实例对象,所以不能使用本关键字. 
立即数名称关键字:		
真	true	用作代表逻辑值立即数真
假	false	用作代表逻辑值立即数假
空对象	null	用作代表空对象,可以匹配所有非常量类的类数据类型以及文本型.



3. 操作符关键字:

名称	特性	优先级	输入字1	首/左侧参数	右侧参数	解释
.	[可扩展]	1		欲访问类/类对象名称	欲访问类成员名称	句点分隔操作符,用作分隔类/类对象与其成员名称,或者用作建立包/类的全名称. 
[]	[只能为参数] [可扩展]	1		欲访问数组数据	[整数] 欲访问数组成员索引值	数组成员访问操作符,用作访问所指定索引位置处的数组成员. 也可以在定义数组数据类型时使用. 数组成员访问索引值从0开始,有效范围为从0到数组成员数-1,分别对应数组的第一个和最后一个成员.

						
-	[只能为参数] [右结合]	2		[数值] 欲取反的数值		算术取反操作符,用作返回将指定数值进行符号翻转后的结果值 
强制类型转换	[只能为参数] [右结合]	2		欲转换到数据类型名称	欲转换数据类型的数据	类型强转操作符,用作将数据强行转换到所指定的数据类型. 调用格式为: **(欲强行转换到的数据类型)欲转换类型的数据** 允许以下数据类型之间进行强制转换: **1.** "空对象"可以强制转换到任何非常量类的类数据类型或文本型:  **2.** 数值数据类型之间可以强行转换. 注意: 当从容量大的数值数据类型强制转换到容量小的数值数据类型时,其中数值精度将会存在被丢失的风险. 譬如,将小数数值强行转换到整数数值将丢失所有小数点后面的数值部分.  **3.** 常量类可以强制转换到其所对应的基本数据类型,非立即数基本数据类型数据可以强制转换到对应的常量类; **4.** 类数据类型只能强制转换到其直接/间接基础类或继承类. **注意:** 用户必须确保对象实例的运行时数据类型匹配该被强行转换到的数据类型,否则将导致程序运行时出错.

						
*	[只能为参数] [可扩展]	3		[数值] 被乘数	[数值] 乘数	算术相乘操作符,用作计算两个数值的相乘结果. 例如: 数值变量1 * 123
/	[只能为参数] [可扩展]	3		[数值] 被除数	[数值] 除数	算术相除操作符,用作计算两个数值的相除结果. 例如: 数值变量1 / 123
%	[只能为参数]	3				算术模除操作符,用作计算两个整数的相除后的余数. 例如: 5 % 4 等于 1
+	[只能为参数] [可扩展]	4		[数值/文本] 被加数	[数值/文本] 加数	算术相加操作符,用作计算两个数值/文本的相加结果. 例如: 数值相加: 数值变量1 + 123 文本相加: "abc" + 到文本 (1)
-	[只能为参数] [可扩展]	4		[数值] 被减数	[数值] 减数	算术相减操作符,用作计算两个数值的相减结果. 例如: 数值变量1 - 123
<=	[只能为参数]	5		[数值] 被比较数值	[数值] 比较数值	小于等于逻辑比较操作符,当左侧参数小于等于右侧参数时返回真. 例如: 如果 (数值变量1 <= 123)
>=	[只能为参数]	5				大于等于逻辑比较操作符,当左侧参数大于等于右侧参数时返回真. 例如: 如果 (数值变量1 >= 123)
<	[只能为参数]	5				小于逻辑比较操作符,当左侧参数小于右侧参数时返回真. 例如: 如果 (数值变量1 < 123)
>	[只能为参数]	5				大于逻辑比较操作符,当左侧参数大于等于右侧参数时返回真. 例如: 如果 (数值变量1 > 123)
属于	[只能为参数]	5	instanceof	[对象] 被检查对象	用作检查的类名	返回左侧对象是否为右侧类或者其直接/间接继承类的实例对象,即左侧对象能否被安全转换到右侧类数据类型. 注意: 左侧对象的数据类型必须为类,而且必须与右侧类之间存在继承/被继承关系或者等于右侧类本身.

						
==	[只能为参数]	6		[数值/逻辑型/文本/数组/对象] 被比较值	[数值/逻辑型/文本/数组/对象] 比较值	等于/不等于逻辑比较操作符,当左侧参数等于/不等于右侧参数时返回真. **注意:** **1.** 对于数组对象,唯一能够与其进行比较的是"空对象"; **2.** 对于文本数据,将比较两者实际文本内容是否相同(区分字母大小写),也可以将文本数据与"空对象"进行比较; **3.** 类对象比较仅比较两者是否指向同一个对象实例,而不会去对比两者所指向对象实例中存放的数据内容是否相同(火山安卓平台比较两个对象是否参考到同一对象实例,火山视窗平台比较两个对象的所处地址是否相同). 注意,在火山视窗平台中: **A.** 如欲比较两个对象的数据内容是否相同,可以使用系统类库中所提供的"**对象内容是否相同**"全局方法; **B.** "**字节集类**"是一个特例,该类的对象之间进行比较将比较两者的实际数据内容是否相同. 例图: 
!=	[只能为参数]	6				
&&	[只能为参数] [可扩展]	7	且	[逻辑型] 逻辑值一	[逻辑型] 逻辑值二	并且逻辑操作符,当左侧参数和右侧参数均为逻辑值真时返回真. 例如: 如果 (数值变量1 >= 100 && 数值变量1 <= 200) 仅当"数值变量1"大于等于100且小于等于200时判断表达式才为真
||	[只能为参数]	8	或			或者逻辑操作符,当左侧参数和右侧参数其中任意一个为逻辑值真时返回真. 例如: 如果 (数值变量1 >= 100 || 数值变量2 <= 200) 仅当"数值变量1"大于等于100或者"数值变量2"小于等于200时判断表达式才为真
=	[右结合]	9		赋值到的变量/可写属性	用作提供赋值用数据	赋值操作符,将右侧参数的值赋予给左侧参数所指定的变量/可写属性. 例如: 变量1 = 1 对象变量1.属性1 = 变量2

注释:

1. 表格中的优先级值越小表明该操作符优先级越高. 以下为操作符优先级在表达式中的处理算法:

在表达式中,优先级高的操作符将优先组合其两侧运算元. 如:

```
1 + 3 * 2
```

在以上表达式中,算术相乘操作符的优先级为3,而算术相加操作符的优先级为4,因此相乘操作符的优先级要比相加操作符的高,所以编译器将先运算"3 * 2",然后再运算"1 + 6",从而得到最终结果7.

如果两个操作符的优先级一致,如:

```
1 + 3 - 2
```

则将按照操作符的排列顺序进行依次处理,因此编译器将先运算"1 + 3",然后运算"4 - 2",从而得到最终结果2.

如果欲强制调节操作符的处理顺序,可以使用小括号. 如:

```
(1 + 3) * 2
```

此时相加操作符将优先于相乘操作符处理,从而得到最终结果8.

2. "特性"列中的"**[只能为参数]**"表示该操作符只能位于语句参数中; "**[右结合]**"表示操作符 将优先结合其右侧的操作元,"**[可扩展]**"表明右侧参数可以被扩展多个.

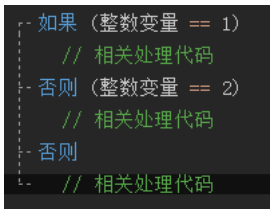
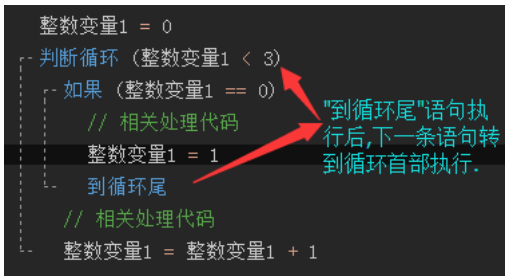
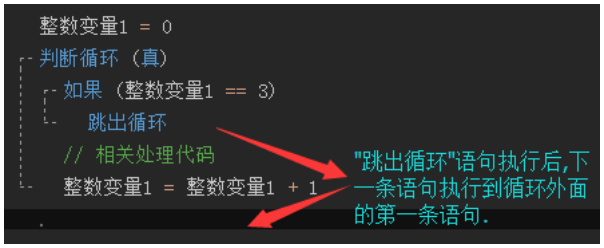
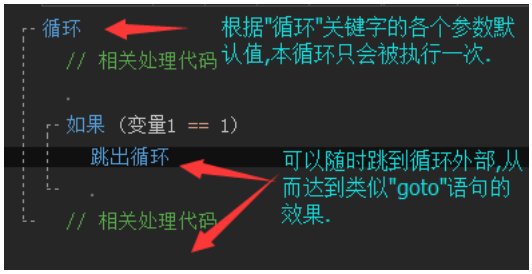
4. 命令关键字:

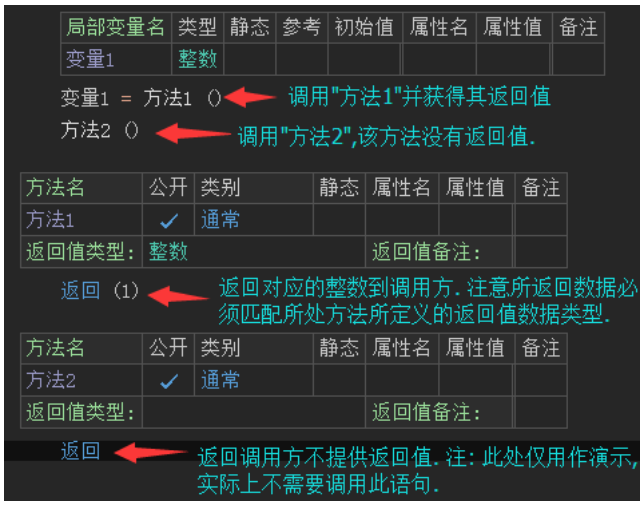
名称	特性	输入字1	参数类型	参数名称	参数解释	返回值	解释
循环类关键字:							

判断循环	[需求语句体]	while	逻辑型	判断值	当此参数值为真时进入循环体,为假时将跳过循环体.		本命令根据提供的逻辑参数的值,来决定是否进入循环体(本语句的子语句体).如果提供的逻辑参数值为真,程序顺序执行下一条语句进入循环体,否则跳转到本命令循环体的下一条语句处.
							整数变量1 = 0
							判断循环 (整数变量1 < 3) ← 循环执行3次
							// 相关处理代码
							整数变量1 = 整数变量1 + 1

循环	[需求语句体]
	[隐藏空参数表]
		for	整数	[可省略] 变量起始值	定义循环变量的起始数值.
					如果被省略,默认值为0.
							本命令将利用变量对循环体(本语句的子语句体)内的命令进行循环执行.第一次执行此命令时将使用"变量起始值"参数初始化"循环变量"参数所指定的变量.每次(包括第一次)执行到此命令处都将判断循环变量内的值是否已经到达"变量目标值"参数所指定的值,如已等于或超过,则跳转到循环体的下一条语句处,否则进入循环体.
							例图,正向遍历处理所指定的数组:
							索引变量 整数
							数组成员数目 整数
							数组成员数目 = 取数组成员数 (欲遍历数组) ← "欲遍历数组"为所需要处理的数组.
							循环 (, 数组成员数目, 索引变量)
							// 处理当前数组成员: 欲遍历数组 [索引变量]
							正向循环3次:
							局部变量名 类型 静态 参考 初始值 属性名 属性值 备注
							索引变量 整数
							循环 (, 3, 索引变量)
							// 相关处理代码
							逆向遍历处理所指定的数组:
							索引变量 整数
							数组成员数目 整数
							数组成员数目 = 取数组成员数 (欲遍历数组) ← "欲遍历数组"为欲处理的数组.
							循环 (数组成员数目, 0, 索引变量, -1)
							// 处理当前数组成员: 欲遍历数组 [索引变量 - 1]
							逆向循环3次:
							局部变量名 类型 静态 参考 初始值 属性名 属性值 备注
							索引变量 整数
							循环 (3, 0, 索引变量, -1)
							// 相关处理代码

			整数	[可省略][需求可写变量] 循环变量	本整数型变量将用作控制循环执行次数,在循环尾部将自动递增或递减该变量内的数值(由变量递增值参数决定).循环体中的用户程序可以直接取用此变量中的值. 如果被省略,将自动使用内部临时变量.		
			整数	[可省略][需求立即数] 变量递增值	每次执行到循环尾部时都将把此值加入到循环变量中去,此值可为正数或负数,如为正数则递增,为负数则递减. 注意: 必须为本参数提供非0整数立即值,以便编译器建立循环结束条件.		

					如果被省略,则默认值为1.		
流程控制类关键字:							
如果	[需求语句体]	if	逻辑型	判断条件	本条件值的结果决定下一步程序执行位置		本命令根据所提供逻辑参数的值,来决定是否改变程序的执行位置.如果提供的逻辑参数值为真,程序继续顺序向下执行进入本命令的子语句体,然后跳过本命令后续所有的"否则"命令,为假则将跳过本命令的子语句体. 例如: 如果 (整数变量 == 1)
否则	[需求语句体] [隐藏空参数表]	elseif	逻辑型	判断条件	本条件值的结果决定下一步程序执行位置. 注意: 本参数可忽略以不提供,但是此时不能再后续跟随其它的"否则"语句.		本命令只能放在"如果"或其它"否则"命令的后面,根据所提供逻辑参数的值,来决定是否改变程序的执行位置.如果提供的逻辑参数值为真,程序继续顺序向下执行进入本命令的子语句体,然后跳过本命令后续所有的"否则"命令,为假则将跳过本命令的子语句体. 例图: 
到循环尾	[隐藏空参数表]	continue					本命令转移当前程序执行位置到当前所处循环体的尾部 注意: 由于火山的所有循环类别关键字均没有循环尾语句,所以此关键字也可以看作为"到循环首". 例图: 
跳出循环	[隐藏空参数表]	break					本命令转移当前程序执行位置到当前所处循环体尾部的下一条语句处. 例图:  结合前面的"循环"语句,可以达到类似basic语言的goto语句的效果: 
返回	[隐藏空参数表]	return		[可省略] 返回值	当所处方法定义有非空返回值数据类型时,必须提供返回到调用方的具体值,否则必须省略掉本参数,即两者必须对应.		本命令转移当前程序执行位置到调用本语句所处方法的下一条语句处,并可根据需要返回一个值到调用语句处. 例图:



运算类关键字:

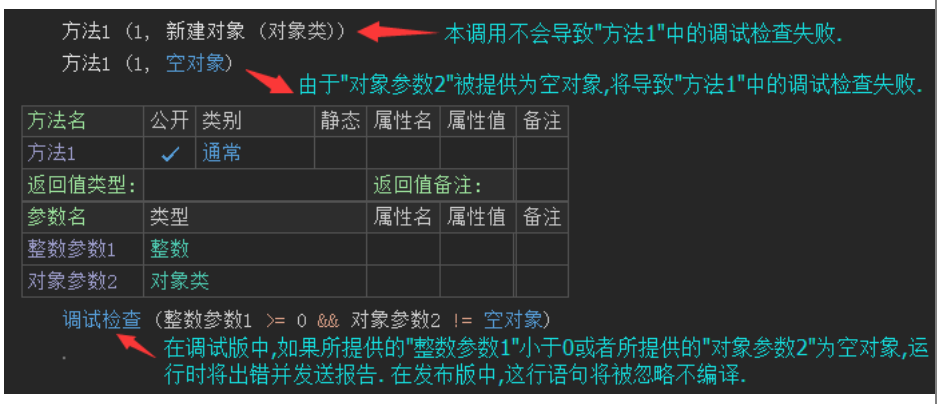
取反			逻辑型	待取反逻辑值	提供将其反转的逻辑值	逻辑型	将所指定逻辑值进行反转(假反转为真,真反转为假),返回反转后的结果. 例如: "取反 (真)" 将返回结果值"假".
位取反			所有整数型	待取反整数值	提供将其所有位反转的整数值	对应整数型	将所指定整数值的每一个位值进行反转(位值1将被反转为位值0,位值0将被反转为位值1),返回反转后的结果. 例如: "位取反 (0xFFFFFFF0)" 将返回结果值"0x0F".
位与			所有整数型	整数参数一	提供用作位操作的整数参数值一	对应整数型	将参数1的每一个位值和参数2的对应位置处位值进行与操作(仅当两个位值均为1时才置为1,否则置为0),返回运算后的结果. 例如: "位与 (0xFFFF1234, 0x0000FFFF)" 将返回结果值"0x00001234".
			所有整数型	整数参数二	提供用作位操作的整数参数值二		
位或			同"位与"				将参数1的每一个位值和参数2的对应位置处位值进行或操作(当两个位值中任意一个为1时即置为1,否则置为0),返回运算后的结果. 例如: "位或 (0x80000000, 1)" 将返回结果值"0x80000001".
位异或							将参数1的每一个位值和参数2的对应位置处位值进行异或操作(当两个位值不相等时置为1,否则置为0),返回运算后的结果. 例如: "位异或 (0x80000000, 0x80000001)" 将返回结果值"1".
位左移			所有整数型	待位移整数值	提供被位移的整数值	对应整数型	将参数1的每一位值向左移动(不考虑符号位)参数2所指定的数目,移动所留下的右侧空位值使用0补齐,返回运算后的结果. 例如: "位左移 (1, 2)"将返回结果值"4".
			所有整数型	位移数目	提供进行位移的位数		
位右移			同"位左移"				将参数1的每一位向右移动(不考虑符号位)参数2所指定的数目,移动所留下的左侧空位值使用0补齐,返回运算后的结果. 例如: "位右移 (0x80000000, 2)"将返回结果值"0x20000000".

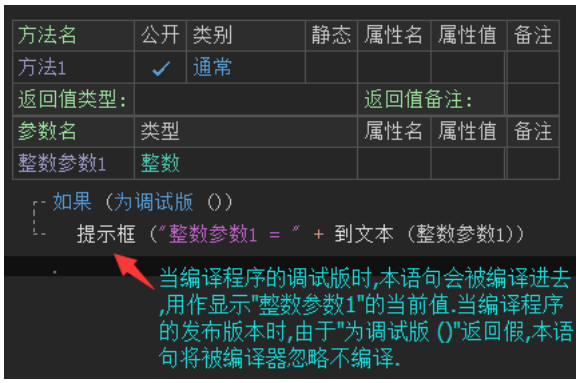
编译时处理关键字:

编译出错		cerror					仅在编译程序时起作用,编译器一旦编译到具有本关键字的语句,就会报告发现了编译错误并停止编译. 本关键字普通用户无需了解.

调试类关键字:

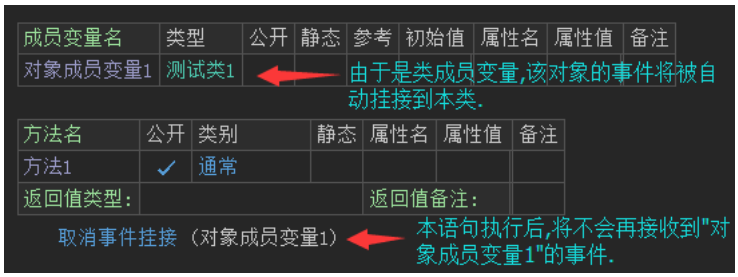
调试检查		assert	逻辑型	检查值	在调试版中,当此参数值为假时,程序将中断执行并报错.		本命令的调用语句仅在程序所编译的调试版本中存在,在程序所编译的发布版中将被忽略不编译. 所谓"**调试版**",即当调试运行当前项目时所编译的程序可执行版本. 在调试版中,当所提供的参数值为假时,程序将在此处中断执行并报错. 本关键字在增强程序可靠性方面非常有用,可以在程序中大量使用本关键字语句检查相关处理数据的合法性,又不用担心编译程序的发布版本时这些检查语句降低了程序执行效率. 如:
			文本型	[可省略] 检查失败信息	指定当检查失败时(检查值参数为假)所输出的信息.如果被省略,则默认为空文本.		
为调试版						逻辑型	返回当前所编译程序是否为调试版本. 所谓"**调试版**",即当调试运行当前项目时所编译的程序可执行版本. 使用本关键字可以加入一些仅在编译调试版时才会编译进去的语句,如:





其它:

挂接事件			对象	欲挂接其事件的对象	提供欲将其所定义事件挂接到当前类对应事件接收方法的对象		将指定对象所支持的事件挂接到当前类对象中的对应事件接收方法上. 本命令仅用作动态挂接对象事件,类中定义的成员对象变量除非明确指定不自动挂接(普通用户无需了解),均会自动挂接事件. 具体请参见前面的"动态挂接其它类所发送过来的事件"章节.
			整数	标记值	用作提供欲挂接其事件的对象所对应的标记值,由用户自行定义.该值将被原值发送给事件接收方法,用作区分具体的事件来源.如果被省略,则默认为0.		
取消事件挂接			对象	欲取消其事件挂接的对象	提供欲取消将其所定义事件挂接到当前类对应事件接收方法的对象		不再将指定对象所支持的事件挂接到当前类对象中的对应事件接收方法上. 无论是自动挂接事件的类成员对象变量,还是手动调用挂接事件关键字挂接的对象,均可以调用本关键字取消其事件挂接. 事件挂接一旦取消,该对象的事件将不会再被接收到.



注释:

1. "特性"列中的"**[需求语句体]**"表示该命令需要携带一个子语句体,"**[隐藏空参数表]**"表示当该命令的参数表为空时将被省略掉不显示;
2. "参数表"列中的"**[可省略]**"表示该参数可以被省略不提供;"**[需求可写变量]**"表示必须为该参数提供一个可写入变量或参数;"**[需求立即数]**"表示必须为该参数提供一个立即数.

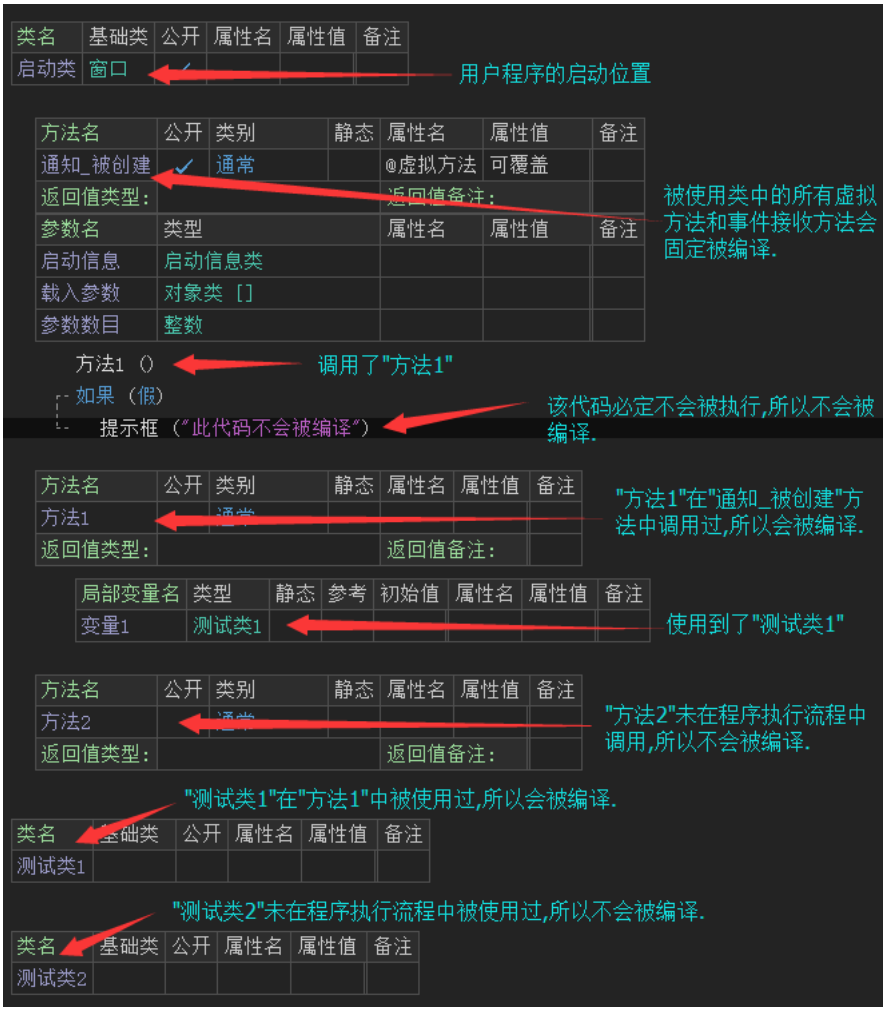
四. 其它:

1. 按需编译

火山编译器实行的是按需编译模式,也就是说,凡是不可能被执行到的代码,一概不进行编译. 具体为:

从程序的启动位置开始,所有未在程序执行流程中的代码都将不会被编译.

以火山安卓开发平台举例,譬如下图:



2. 扩展流程控制

除了命令关键字外,火山平台的核心类库中封装了一些自定义流程控制全局方法.

火山视窗平台的例程解决方案("samples\vprj_win\samples.vsln")中提供了一个名为"**自定义流程控制**"的样例项目用作列举它们的使用方法,可以打开查看.相关图示如下:

```
整数数组1.枚举循环 ()
    如果 (整数数组1.取枚举值 () == 1003)
        跳出循环
    编辑框1.加入文本行 ("整数数组枚举循环: " + 到文本 (整数数组1.取枚举值 ()))

文本数组1.枚举循环 ()
    编辑框1.加入文本行 ("文本数组枚举循环: " + 文本数组1.取枚举值 ())

对象数组1.枚举循环 ()
    编辑框1.加入文本行 ("对象数组枚举循环: " + 到文本 (((测试类1)对象数组1.取枚举值 ()).成员1))
```

枚举循环

3. 部件程序

在某些情况下,可能希望程序的功能实现源代码不公开但是又不影响最终编译,此时可以使用部件程序.

部件程序与普通程序的区别在于,部件程序可以将普通程序分为两个部分,一个是其公开接口部分(如被公开的类/方法等),另一个是其功能实现部分,公开接口部分在系统程序编辑器中可以被阅读(不能修改),仅在编辑项目时使用,功能实现部分则在编辑器中被隐藏不可见,仅在编译项目时使用,两者互不影响.

当根据普通程序生成其部件程序时,其中如存在名为"**启动类**"的类,将被自动忽略跳过.

除了以上区别之外,部件程序与普通程序没有任何差异,所有可以使用普通程序的场合均可使用部件程序.

4. 部件DLL程序

部件DLL程序仅在火山视窗软件开发平台中有效,在视窗项目属性中将"**编译结果类型**"属性设置为"**部件DLL动态链接库**"即可.

"部件DLL动态链接库"和普通的"DLL动态链接库"没有本质的区别,都是动态链接库,只是前者的功能可以被输出到项目外部使用,并且由于相关实现代码位于编译后的DLL中,因此安全性是绝对有保证的,可广泛应用于多人协作或闭源项目中.

系统在编译部件DLL程序时,会自动生成其接口程序,使用方将生成的所有接口程序加入到其项目中即可使用部件DLL中的功能.

亦可指定系统在编译部件DLL程序时自动打包生成对应的火山模块,使用方将该模块安装到其火山系统中即可使用其中的功能.

部件DLL中的以下内容会被自动输出到所生成的接口程序中:

1. 所有公开类及其中的公开成员常量和公开成员方法(不会包含具体功能实现代码).

以下类不会被输出到所生成的接口程序中(即使满足前面的条件):

1. 系统类(其所处源文件位于当前所运行火山系统安装目录下的);
2. 模板基础类,其所有模板实现类将被自动展开后输出;
4. 启动类;
5. 窗口组件类;
6. 所有设置了值为假的"@输出到部件"属性的类(初级用户无需了解).

以下方法不会被输出到所生成的接口程序中(即使满足前面的条件):

1. 嵌入式方法;
2. 虚拟方法;
3. 类的初始化和清理方法;
4. 事件接收方法;
5. 所有设置了值为假的"@输出到部件"属性的方法(初级用户无需了解).

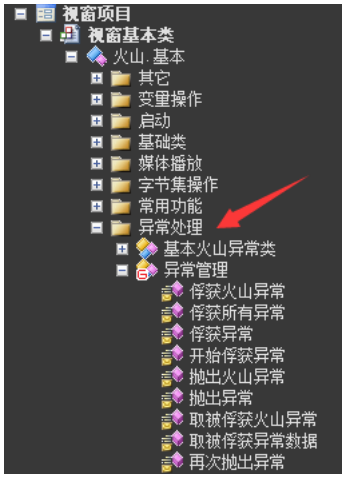
以下常量不会被输出到所生成的接口程序中(即使满足前面的条件):

1. 所有设置了值为假的"@输出到部件"属性的常量(初级用户无需了解).

系统在编译部件DLL程序时会自动处理该DLL程序中指定的所有附属文件(通过"视窗.附属文件"属性),并会将其复制到所生成的接口数据目录中及在所生成的接口程序中设置相关属性.

5. 异常管理

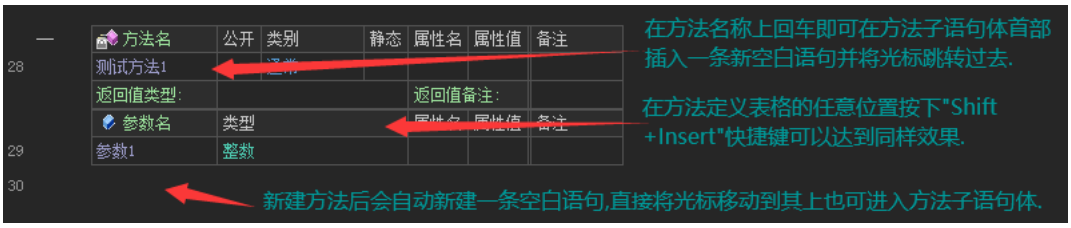
同样,异常管理火山平台也没有提供对应的关键字,而改用系统类封装解决,以下为火山视窗类库中的相关封装类:



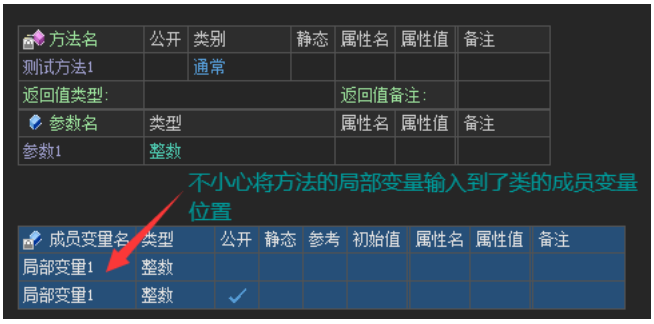
5. 常用开发环境操作方法

1. 编辑新建方法子语句体:

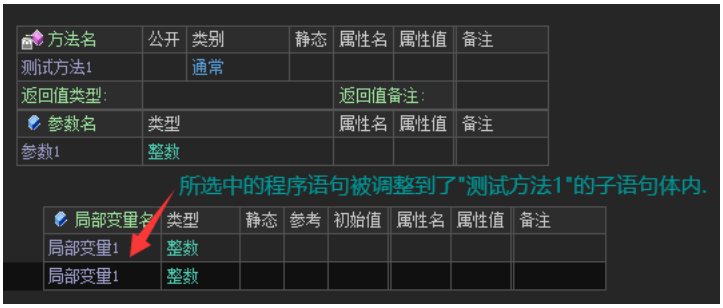
按照下图中的说明即可进入编辑所指定新建方法的子语句体:



2. 使用 "**Ctrl+[**" 和 "**Ctrl+]**" 组合快捷键可以将当前光标所处语句或者所选中语句块左移或右移,以改变其当前缩进层次,从而改变其所处语句块:
如图,假设不小心将局部变量设置在了方法的外面,可以使用 "**Ctrl+]**" 组合快捷键将其调整进去:



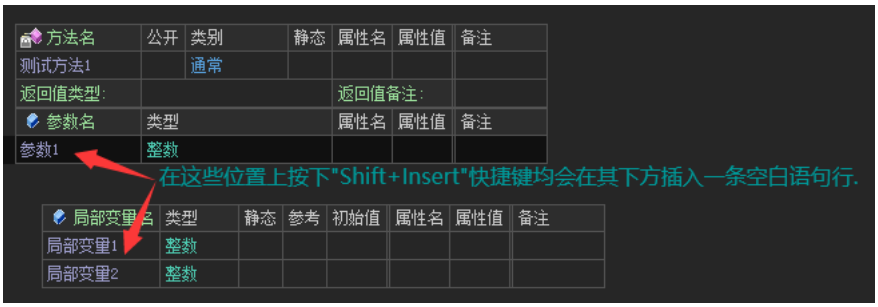
选中所欲调整层次的语句行,再按下 "**Ctrl+]**" 组合快捷键将其调整进去:



同样,类似以下的程序,我想把被选中的部分移动到外层语句块中:

选中所欲调整层次的语句行,再按下 "**Ctrl+[**" 组合快捷键将其调整过去:

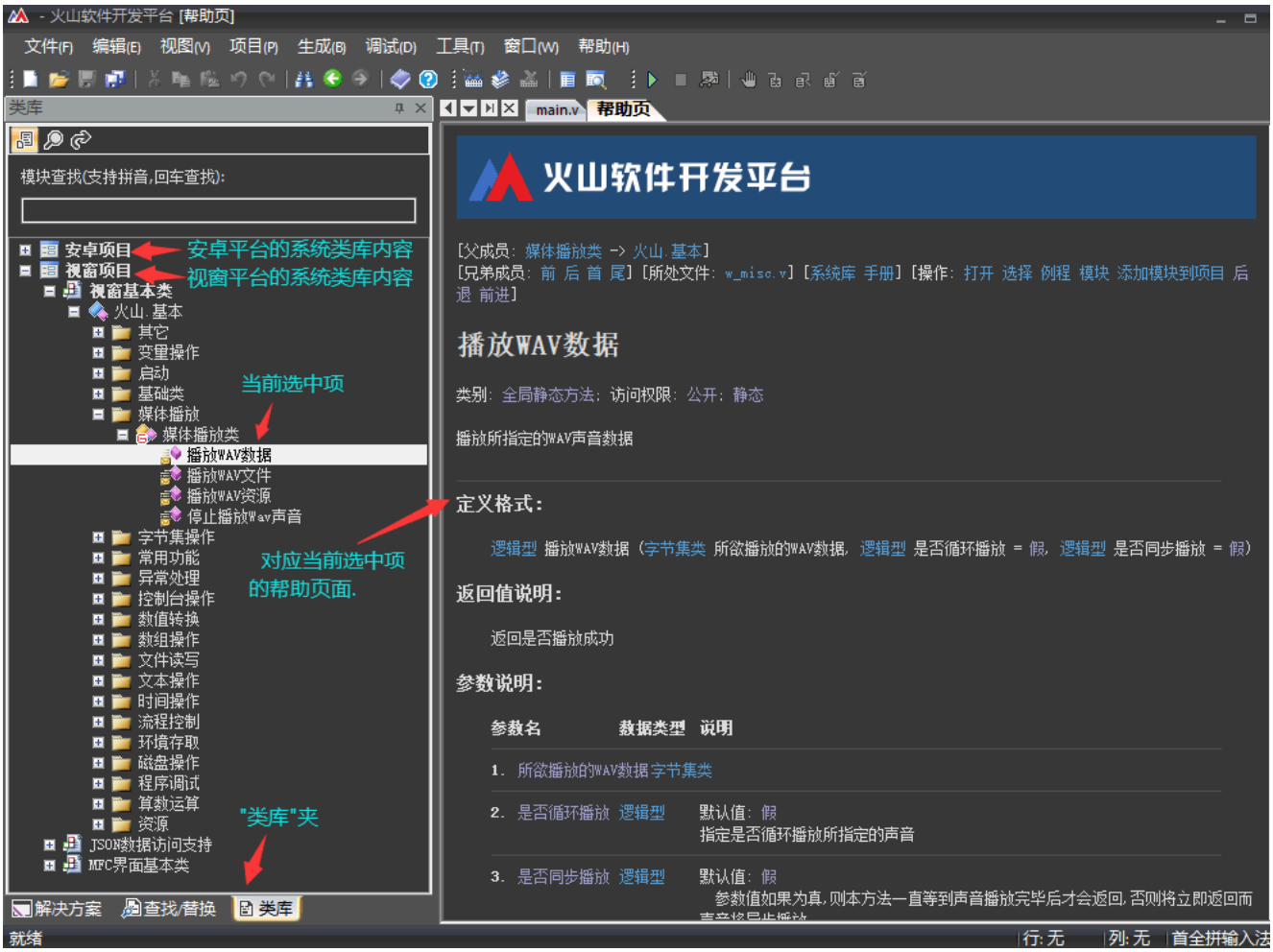
3. 另外几个使用得比较多的快捷键为"**Insert**"和"**Shift+Insert**",用作向前/向后插入行.其中,在方法的参数/常量/变量定义表格上按下"**Shift+Insert**"快捷键,将固定在其下方插入一条空白语句行,如图:



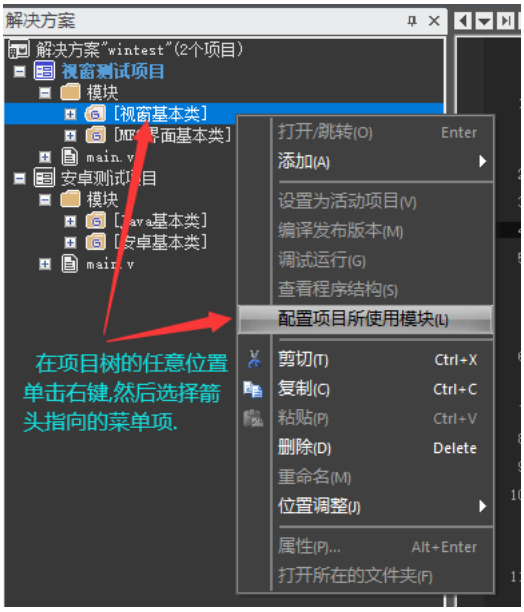
在"局部变量1"的定义行上按下"**Shift+Insert**"快捷键后的结果:

4. 开发过程中充分利用"**即时帮助(F1)**和"**程序结构查看器(Shift+F1)**"可以获得所有系统类和用户程序类的详细使用帮助信息:

5. 在开发环境左侧的"**类库**"夹中展开当前项目类型,可以获得该类型项目当前可用的所有系统类库模块信息:



注意对应模块如果欲使用,需要首先在程序中将其配置进去:

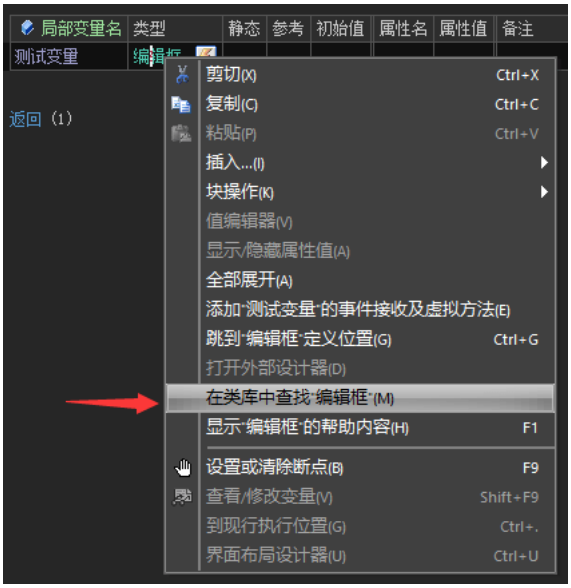


然后选中对应的模块确认即可:

6. 查找某名称所处的模块

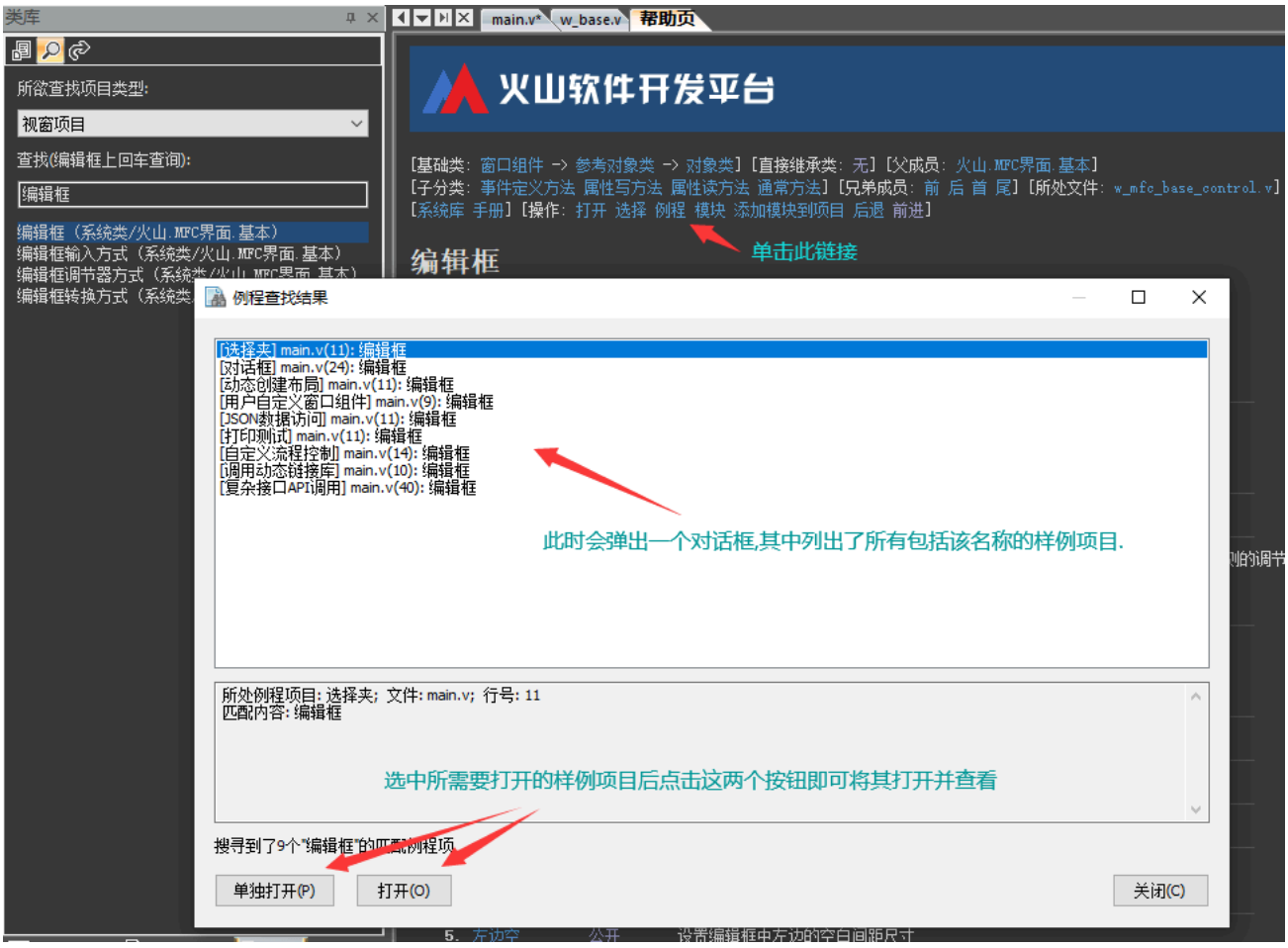
用户如果在编译项目时遇到"名称未找到"错误，有可能是因为其所处模块没有被加入到项目中，可以通过如下操作来查找(以火山视窗平台举例):

A. 在编辑器内该名称文本上单击右键，选择"**在类库中查找**"菜单项:



B. 找到后在其帮助页顶部操作栏上点击"**添加模块到项目**"链接 即可将该名称所处模块添加到当前项目:

7. 在帮助页面中点击“**例程**”链接（如上图黄色箭头）可以搜寻并打开包括所指定名称的例程，这对于了解该名称的使用方法很有好处:


类库
所欲查找项目类型:
视窗项目
查找(编辑框上回车查询):
编辑框
编辑框（系统类/火山.MFC界面.基本）
编辑框输入方式（系统类/火山.MFC界面.基本）
编辑框调节器方式（系统类/火山.MFC界面.基本）
编辑框转换方式（系统类
main.v*
w_base.v
帮助页
火山软件开发平台
[基础类: 窗口组件 -> 参考对象类 -> 对象类] [直接继承类: 无] [父成员: 火山.MFC界面.基本]
[子分类: 事件定义方法 属性写方法 属性读方法 通常方法] [兄弟成员: 前 后 首 尾] [所处文件: w_mfc_base_control.v]
[系统库 手册] [操作: 打开 选择 例程 模块 添加模块到项目 后退 前进]
单击此链接
编辑框
例程查找结果
[选择夹] main.v(11): 编辑框
[对话框] main.v(24): 编辑框
[动态创建布局] main.v(11): 编辑框
[用户自定义窗口组件] main.v(9): 编辑框
[JSON数据访问] main.v(11): 编辑框
[打印测试] main.v(11): 编辑框
[自定义流程控制] main.v(14): 编辑框
[调用动态链接库] main.v(10): 编辑框
[复杂接口API调用] main.v(40): 编辑框
此时会弹出一个对话框,其中列出了所有包括该名称的样例项目.
所处例程项目: 选择夹; 文件: main.v; 行号: 11
匹配内容: 编辑框
选中所需要打开的样例项目后点击这两个按钮即可将其打开并查看
搜寻到了9个"编辑框"的匹配例程项
单独打开(P)
打开(O)
关闭(C)
5. 左边空
公开
设置编辑框中左边的空白间距尺寸

--- 完 ---