AHKv2类规范1.0

目录

[一、类的基础内容 2](#_Toc118143181)

[（一）类名，私有域与公有域 2](#_Toc118143182)

[1、类名 2](#_Toc118143183)

[2、私有域与公有域 2](#_Toc118143184)

[（二）元函数介绍与使用 3](#_Toc118143185)

[1、\_\_Init函数 3](#_Toc118143186)

[2、\_\_New函数 4](#_Toc118143187)

[3、\_\_Get、\_\_Set、\_\_Call函数 4](#_Toc118143188)

[4、\_\_Item函数 7](#_Toc118143189)

[5、\_\_Enum函数 8](#_Toc118143190)

[6、\_\_Delete函数 9](#_Toc118143191)

[7、ToString函数 10](#_Toc118143192)

[（三）继承 11](#_Toc118143193)

[二、关于库类的定义建议 11](#_Toc118143194)

[（一）自定义元函数以及自定义函数的配合使用 11](#_Toc118143195)

[（二）AHKv2-13-mono标准：自定义元函数规范及功能介绍 12](#_Toc118143196)

类作为面向对象编程中的重要一环，在各种编程语言中，都有很多规范和用法说明，下面我就AHK的类模块专门写出一则规范，目的是便于统一现行各类代码的使用逻辑，并对于新人书写AHK库提供指导。

# 一、类的基础内容

## （一）类名，私有域与公有域

### 1、类名

通常上讲，我们定义一个类，就是通过（Class+类名）来实现的，例如：

1. **class** a
2. {
3. }

就是定义了一个a这个类，其中a为类名，通常可以用如下方式获取：

1. b := a()
2. msgbox b.\_\_class
3. msgbox type(b)

### 2、私有域与公有域

我们学会创建类之后需要思考一个问题，什么样的数据是允许共享的，什么样的数据是只允许自己拥有的，我们来看下面这个例子：

1. **class** pig
2. {
3. **static** value := 100
4. name := "lily"
5. }

对于公有域和私有域均分为属性和函数两部分，先讲结论，上述二者均属于属性，其中value属于公有属性，name属于私有属性，从二者的调用中就可以看出差距：

1. a := pig()
2. msgbox pig.value
3. ; output => 100
4. msgbox a.name
5. ; output => "lily"

对于公有属性来说，调用时需要（类名.属性）来访问；而对于私有属性来说，调用时则是需要（类实例.属性）进行访问。

定义公有域的办法，就是在定义时，加上static这个静态前缀，用来标识这是公有域的属性或函数。

## （二）元函数介绍与使用

在AHK中，每个类都内置了几个元函数（在python中称为魔术函数），下面我逐一进行介绍：

### 1、\_\_Init函数

在类实例的创建过程中会涉及到两个函数，一是\_\_Init函数，另一个是\_\_New函数。我们先来介绍\_\_Init函数。

Init顾名思义就是进行初始化，我们来看一下例子：

1. **class** pig
2. {
3. \_\_Init()
4. {
5. **this**.name := 100
6. }
7. }
8. b := pig()
9. msgbox b.name

我们会发现，尽管在类中我们并没有定义name这个私有属性，但是在创建b这个实例的过程中，我们自动进行了初始化，使得b这个实例获得了name属性。

\*难点：this的理解。简单记忆就是，对于私有域，this指代的是实例本身；而对于公有域，this指代的是类本身。

### 2、\_\_New函数

那么初始化之后，如何创建类实例呢？通过上述几个例子，你可以观察到，我们通过形如（类实例:=类名()）来创建一个类实例。

那么如果我们需要创建一个独特的类实例应该怎么做？就是传参，那就需要用到\_\_New这个函数。我们再看以下例子：

1. **class** pig
2. {
3. \_\_New(name)
4. {
5. **this**.name := name
6. }
7. }
8. lily := pig("lily")
9. msgbox lily.name

在这里我们就创建了一个独特的pig实例lily，它具有name属性并且与我们输入的参数相关。

### 3、\_\_Get、\_\_Set、\_\_Call函数

以上几个函数都是与私有属性或私有函数相关，就放在一起讲了。

首先来看\_\_Get函数，我们来看以下例子：

1. **class** pig
2. {
3. \_\_Get(name, params)
4. {
5. msgbox name
6. msgbox params.length
8. **return** "unknown"
9. }
10. }
12. a := pig()
13. msgbox a.name

我们可以发现，当我们访问a的name属性时，会调用\_\_Get函数，他需要两个参数，第一个是属性的名称，第二个是传入的参数。Return的值就是我们访问后获取的值。

下面就第二个参数展开详述，继续看一个例子：

1. **class** pig
2. {
3. \_\_Get(name, params)
4. {
5. msgbox params.length
7. **return** "unknown"
8. }
9. }
11. a := pig()
12. msgbox a.name[2]

我们会发现这时候获得的params.length值为1，经过更为详细的分析就能知道此时的params为[2]，代表着我们传入了一个参数2。

\_\_Set函数与\_\_Get函数是一对元函数，理解了\_\_Get函数的运行机理就不难理解\_\_Set函数，下面看一个例子：

1. **class** pig
2. {
3. \_\_Set(name, params, value)
4. {
5. msgbox value
6. }
7. }
9. a := pig()
10. a.name := "lily"

我们会发现\_\_Set函数的传入参数相较\_\_Get函数多了一个value，即为所要赋的值。

\_\_Call函数与上述二者差距在于，\_\_Call函数面向的是私有函数，其余使用与\_\_Get基本保持一致。来看一个例子：

1. **class** pig
2. {
3. \_\_Call(name, params)
4. {
5. msgbox name
6. msgbox params.length
8. **return** "unknown"
9. }
10. }
12. a := pig()
13. msgbox a.name(2)

简单来说，你可以把\_\_Call理解为把\_\_Get调用时的[]变成()。

\*注意：这里建议不要轻易自己创建\_\_Get、\_\_Set、\_\_Call函数，如果需要建立后面有相关帮助可以查看，可能会遇到死循环等问题。

### 4、\_\_Item函数

这个函数对于数据类型类有重要意义，例如Array和Map类。我们来看一个例子：

1. **class** pig
2. {
3. name := array()
5. \_\_New(name)
6. {
7. **this**.name.push(name\*)
8. }
10. \_\_Item[index]
11. {
12. Get => **this**.name[index]
13. Set => **this**.name[index] := value
14. }
15. }
17. a := pig(["lily", "lisa"])
18. msgbox a[2]
19. a[2] := "nana"
20. msgbox a[2]

我们可以发现，\_\_Item的作用就是让类实例拥有和Array类似的使用方式。其中Get和Set意义类似\_\_Get函数和\_\_Set函数。

### 5、\_\_Enum函数

这个算是比较麻烦的一个元函数，它的机理比较复杂。先来个示例体验一下：

1. **class** pig
2. {
3. name := array()
5. \_\_New(name)
6. {
7. **this**.name.push(name\*)
8. }
10. \_\_Enum(\_)
11. {
12. **this**.index := 1
13. **return** fn
15. fn(&value)
16. {
17. **if** **this**.index > **this**.name.length
18. {
19. **this**.index := 1
20. **return** **false**
21. }
23. value := **this**.name[**this**.index++]
24. **return** **true**
25. }
26. }
27. }
29. a := pig(["lily", "lisa"])
30. **for** i in a
31. msgbox i

我们一点点分析\_\_Enum的各项参数，首先是他输入的参数，就是例子中的“\_“，他指代的是迭代返回数目，例如for i in a迭代数目就是1，而for i, j in a迭代数目为2。再看return fn，这里fn是一个函数，需要接收引用用以返回迭代值，返回为true或false，当fn返回false时停止迭代。

我们仔细剖析“接收引用用以返回迭代值“这句话的含义，我们可以这么理解，在for i in a中，我们临时命令一个未知变量为i，它作为一个容器，从\_\_Enum函数中取得值并在for循环中可以使用（超出for循环范围后失去作用）。引用的数目根据迭代的数目自行设计，在AHK的设计理念中，迭代最多数目不得超过19。

### 6、\_\_Delete函数

如何判断一个类实例是否删除呢？这个就要介绍现在最流行的引用计数法。请看下面这两个例子：

1. **class** pig
2. {
3. name := array()
5. \_\_New(name)
6. {
7. **this**.name.push(name\*)
8. }
10. \_\_Delete()
11. {
12. msgbox 1
13. }
14. }
16. a := pig(["lily", "lisa"])
17. a := 1
18. **class** pig
19. {
20. name := array()
22. \_\_New(name)
23. {
24. **this**.name.push(name\*)
25. }
27. \_\_Delete()
28. {
29. msgbox 1
30. }
31. }
33. a := pig(["lily", "lisa"])
34. b := a
35. a := 1

上述两个例子哪一个实例被删除了？当然是第一个，在第二个中，b还保留着实例的信息，相当于实例仍被引用，就不会触发垃圾回收机制，也就不会将实例删除。

\_\_Delete函数的触发条件就是当一个实例的所有引用结束时，触发这个函数，这个函数没有输入参数。

### 7、ToString函数

这个函数是AHKv2beta后续版本的更新内容，主要用于String强制转换类型的使用以及msgbox的使用，以及其他需要String的情况。

下面简单给一个实例，这个函数本身并不困难：

1. **class** pig
2. {
3. name := array()
5. \_\_New(name)
6. {
7. **this**.name.push(name\*)
8. }
10. ToString()
11. {
12. **return** "type: pig"
13. }
14. }
16. a := pig(["lily", "lisa"])
17. msgbox(a)

我们可以得到返回值为"type: pig"。

\*注意：在AHKv2beta后续版本中，如果一个类没有ToString函数，使用msgbox的时候将不会返回空值，而会直接报错。

## （三）继承

简单讲讲类的继承，其实原则非常简单，就是将被继承类的一切公有域和私有域复制到继承类上去，关键字为extends。例如：

1. **class** list extends array
2. {
4. }

新的类list继承了array的一切属性，比如push、pop等等。

# 二、关于库类的定义建议

## （一）自定义元函数以及自定义函数的配合使用

我认为，对于AHK的库而言，他所写的类一定要包含一些共同之处，比如我想写一个新的数据类型list，那我想要知道list里有多少元素，可以自定义一个\_\_Len函数，再在全局范围内，写一个len函数用于调用类的\_\_Len以便捷获取长度，如下例：

1. **class** list extends array
2. {
3. \_\_Len()
4. {
5. **return** **this**.length
6. }
7. }
9. len(obj)
10. {
11. **return** obj.\_\_len()
12. }
14. a := list(1, 2, 3)
15. msgbox len(a)

## （二）AHKv2-13-mono标准：自定义元函数规范及功能介绍

|  |  |  |
| --- | --- | --- |
| **AHKv2-13-mono标准** | | |
| **函数名** | **功能** | **备注** |
| \_\_cmp | 规定该类的比较度量 |  |
| \_\_def | 规定该类函数的标准模板 |  |
| \_\_future | 在不考虑兼容性的情况下添加的函数 |  |
| \_\_hash | 规定该类的哈希值计算 |  |
| \_\_len | 规定该类的容量度量 |  |
| \_\_rely | 规定该类的前置库 |  |
| static \_\_version | 规定该类的版本号 |  |