# vJass 系列教程 7

面向对象编程（二） 结构高级语法

Aeris ▸ NJU ▸ 2008-7-9

# vJass 系列教程 7

面向对象编程（二） 结构高级语法

## 内容提要

我们接着上一章教程来介绍结构的高级语法。

## 结构（二）

### 初始化和清除——构造方法和析构方法

从上一节教程中，我们可以看到，结构可以动态创建和删除。那么，在创建和删除结构时，我们怎样对结构初始化和清除呢？这就是构造方法和析构方法所做的工作。

把上一章教程的例子稍作修改，加上新内容。

```
struct Point
    integer x
    integer y

    method move takes integer nx, integer ny returns nothing
        set this.x = nx
        set this.y = ny
    endmethod

    // 构造方法，完成对象实例的初始化
    static method create takes integer nx, integer ny returns Point
        local Point p

        set p = Point.allocate()

        set p.x = nx
        set p.y = ny

        call BJDebugMsg("Point::Point() Initialization")

        return p
    endmethod

    // 析构方法，负责清除和扫尾
    method onDestroy takes nothing returns nothing
        set x = 0
        set y = 0

        call BJDebugMsg("Point::onDestroy() called!")
    endmethod
```

```
endstruct

function PointTest takes nothing returns nothing
    local Point p

    set p = Point.create(1, 2)
    set p.x = 5
    set p.x = 8

    call BJDebugMsg(I2S(p.x) + " : " + I2S(p.y))

    call p.destroy()
endfunction
```

例子为 Point 结构定义了构造方法和析构方法，完成对象的初始化和清除。

从例子中，我们可以看到，构造方法的定义语法和一般编程模式是：

```
static method create takes 参数列表 returns 结构名
    local 结构名 instance

    set instance = 结构名.allocate()

    // 各种初始化，例如初始化成员
    set instance.成员 = 值
    ...

    return instance
endmethod
```

构造方法的名称约定为 create，而且必须是静态方法（加上 static 关键字，下文会介绍），在创建结构实例的时候被调用。可以带参数，必须返回该结构类型。如果你的构造方法是带参数的，那么创建实例的时候就要使用参数来调用了，就像例子中的一样。在构造函数内部，通常的做法是使用一个局部变量来存放动态分配的结构实例，然后把结构的成员初始化好，最后返回该实例。动态分配实例使用结构的 allocate 函数，这个函数是编译器自动生成的，任何结构都包含。

我们也可以看到，析构方法的定义语法和编程模式是：

```
method onDestroy takes nothing returns nothing
    // 清除和扫尾工作，例如
    call RemoveLocation(this.loc)
    set this.loc = null
    ...
endmethod
```

析构方法的定义语法是固定的，不能更改，必须是“**method** onDestroy **takes nothing returns nothing**”。和构造函数不同，它是一个成员方法而不是静态方法。

## 静态成员

从前面的很多例子，我们都可以看到，结构的构造方法一律都加了修饰符 static。当然，不一定是方法，成员变量也可以加 static 修饰符。当 static 修饰符修饰一个成员时，表示该成员是**静态**的。所谓“静态”，

意思是说，它是和该**结构本身**相联系的，而**不是**和结构的一个**实例**相联系。所以，静态方法中**不能**使用 this 引用，因为它不和对象相联系。举个单位的例子，我们知道单位有很多属性，例如生命值，不同的单位生命值是不尽相同的，也就是说，生命值这个属性是和单位这个结构的实例相联系的，它是单位这个结构的成员变量。而单位个数这个属性，不和任何具体单位相联系，它是单位这个结构本身的属性，因此，单位个数是静态成员。从上面这个例子也可以看到，某种程度上静态成员和全局变量有一定的相似之处。没错，的确如此，但是，静态成员是和结构相联系的，所以多个结构可以有同名的静态成员，而全局变量就不可以同名。另外，更重要的是，静态成员表明了它和这个结构逻辑上的相关性，这是全局变量做不到的。

静态成员的语法非常简单，就是在成员定义前面加上修饰符 static 而已，访问静态成员使用

**结构名.静态成员名（通用方法，无论在哪里都可以用）**

或者

**this.静态成员名（仅在普通成员函数里，这时可以使用 this 引用）**

或者

**对象名.静态成员名（当有一个该结构的对象存在时）**

下面用一个简单的例子来说明。

```
// 可以统计自身实例数目的结构，可以用于泄漏检查
struct CountInstance
    // 结构的实例数
    static integer numberOfInstance = 0

    static method create takes nothing returns CountInstance
        local CountInstance self = CountInstance.allocate()

        // 每创建一个对象，就增加计数器
        set CountInstance.numberOfInstance = CountInstance.numberOfInstance
+ 1

        // 如果有其他初始化，放在这里
        // ...

        return self;
    endmethod

    method onDestroy takes nothing returns nothing
        // 每销毁一个对象，计数器减1
        set CountInstance.numberOfInstance = CountInstance.numberOfInstance
- 1
    endmethod

    // 显示成员数
    static method howMany takes nothing returns nothing
```

```
        call BJDebugMsg("本结构有" + I2S(CountInstance.numberOfInstance) + "
个实例")
    endmethod
endstruct
```

## 静态成员的初始化

我们都知道，普通成员是在生成对象实例时，也就是构造方法 create 里面初始化的，而静态成员（变量）不和具体成员挂钩，怎么初始化呢？初始化静态成员，通常有 3 种方法。

**方法 1**：直接指定初始化值

就是定义的时候直接写初始化值，这种做法非常简单有效，但是只能用于**简单类型变量**，如整数、实数等类型的初始化。

例如上面这个例子做的：`static integer numberOfInstance = 0`

**方法 2**：在 library 或者 scope 的**初始化函数**里，或者**地图初始化触发**里初始化（**通用办法**）

这个也很简单实用，特别是在方法 1 不行的时候。唯一要注意的是，地图初始化触发的执行是晚于 library 或者 scope 的初始化的，因此如果放在地图初始化触发里的话，要保证这个变量初始化之前不被使用。这里就不举例了。

**方法 3**：使用 onInit 静态方法（**高级方法，使用要谨慎**）

这个是比较高级的特殊方法，一般情况下用得不多。结构可以定义特殊的静态方法 onInit，这个方法会在地图初始化之前被调用（这时所有触发、所有变量、包括 library 和 scope 都没有初始化，所以这时是不能访问变量，调用触发的）。例子如下：

```
struct EarlyInitialize
    static integer i

    // 特殊方法，必须这样定义，会在地图初始化前调用
    static method onInit takes nothing returns nothing
        set EarlyInitialize.i = 0

        // 非法！会引起地图出错！因为触发尚未初始化
        call TriggerExecute(某触发)
    endmethod
endstruct
```