面向对象程序设计

from 百度百科

面向对象程序设计简单地说就是一种**层次模型**，这种层次模型能够被**重用**。

层次结构的**上层具有通用性**，但是**下层结构则具有特殊性**。（动物-哺乳动物-灵长类-人）在继承的过程中类则可以从最顶层的部分继承一些方法和变量。类除了可以**继承**以外同时还能够进行修改或者添加。通过这样的方式能够有效提高工作效率。

类是具有相同特性（数据元素，体现为变量）和行为（功能，体现为函数）的对象的抽象。因此，**对象的抽象**是**类**，**类的具体化**就是**对象**，也可以说**类的实例**是**对象**，类实际上就是一种**数据类型**。我们可以自己定义类，也就是说，在脚本中可以自己定义数据类型。

类具有**属性和方法**，可以理解为类内的**变量和函数**。属性就是变量，函数就是方法。

注册具名类

对于gdscript来说，所有脚本文件都是未命名的类，而我们可以给类（脚本文件）命名。还可以配合使用 `@icon` 注解，向其括号中输入图片的路径，来将该图片作为该类的图标使用。这样，你的类就会和新的图标一起显示在编辑器中：

```
@icon('res://dog.webp')
class_name Animal # 给类取名
extends Node # 继承的关键字extends

var is_breathing = true

func move():
    print('I have moved 1 meter!')
```

继承

我们用一个例子来说明什么是类和继承，狗**属于**动物，死的哈士奇**属于**狗

Animal.gd

```
class_name Animal # 给类取名

extends Node # 继承的关键字extends

var is_breathing = true

func move():
    print('I have moved 1 meter!')
```

Dog.gd

```
class_name Dog extends Animal

func bark():
    print('I am barking!')
```

Dead_Husky.gd

```
class_name Dead_Husky

extends Dog

func _ready():
    is_breathing = false

func bark():
    print('I cannot bark')
```

1. 子类继承父类所有的属性和方法，包括其父类从父类的父类等中继承的属性和方法。
2. 子类可以修改父类的同名属性或方法，但只对子类生效
3. 子类可以添加独属于自己的属性或方法

为什么要用面向对象程序设计：以改动弓箭手 move 方法为例
当然在godot中弓箭手这种大多数是场景，场景无法继承另一个场景，但弓箭手的各种行为由脚本控制，不同种弓箭手之间的行为脚本也可以进行继承

类的实例化

类是一个模板，对它进行实例化可以得到很多具有相似特征的“同一类”对象
将脚本绑定到一个节点上，在游戏开始时这个脚本所代表的类就实例化为对象。
调用类的属性和方法用 . 操作符

Dog.gd

```
class_name Dog extends Animal

func _ready():
    bark()
    move()
    print(is_breathing)

func bark():
    print('I am barking!')
```

Dead_Husky.gd

```
class_name Dead_Husky

extends Dog

func _ready():
    is_breathing = false
    bark()
    move()
    print(is_breathing)

func bark():
    print('I cannot bark')
```

调用父类

```
func _ready():
    super.some_func(x) # 调用父类方法
    var x = super.age # 访问父类属性

func some_func(x):
    super(x) # 调用父类中同名的方法
```

单例（自动加载）

Godot 的场景系统虽然强大而灵活，但有一个缺点：无法保存多个场景都需要的信息（例如玩家的分数或者背包），也即游戏的全局变量。
自动加载的单例是解决需要在场景之间存储持久性信息的常见用例的实用工具。所谓单例也就是一个实例化的类，但是只实例化一次，且其变量和方法都可以被其它脚本调用。自动加载脚本时，godot引擎会创建一个 Node 并把脚本附加上去。

player.gd

```
var health = 114514

func attack():
    print("attack")
```

test.gd

```
func _ready():
    print(player.health)
    player.attack
```

. 操作符

1. 在类内调用自己类的属性和方法时可以不用，参考上面 `Dog` 类和 `Dead_Husky` 类，但实际上这种写法是一种省略。完整的写法是 `self.is_breathing` , `self.bark()` 等，即调用自己类的方法
2. 在一个类中调用其他类(对象)的属性和方法时一定要用 `.` 操作符，
   如 `arr.push_back(5)` , `player.health`

实例化场景

在godot中场景也是一种“模板”，也可以被实例化。
有两种方法可以实例化场景

1. 在GUI中实例化场景
2. 用代码实例化场景