



ant00N

1 июн 2015 в 16:53

Lua, ООП и ничего лишнего



4 мин



61K

ООП*, Lua*

Из песочницы

Однажды судьба свела меня с ней. С первого взгляда я был ослеплен и долгое время не мог отвести от нее взгляд. Шло время, но она не переставала меня удивлять, иногда казалось, что я изучил ее вдоль и поперек, но она снова переворачивала все мои представления. Ее гибкости не было предела, а потом я узнал, что она умеет еще и... ООП!

Как-то я всерьез занялся покорением ООП в lua. И все, что я находил в интернете по этой теме, было вырвиглазными нагромождениями кода с обилием нижних подчеркиваний, которые никак не вписывались в элегантность этого языка. Поэтому я решил искать простое решение.

После прочтения множества умных книжек и разбора нескольких ужасных реализаций ООП, я, крупица за крупицей, собирал все самое полезное и простое, пока не выработал свой стиль объектно ориентированного программирования на lua.

Создание класса и экземпляра

▼ `class Person`

```
--класс
Person= {}
--тело класса
function Person:new(fName, lName)
```

```

-- СВОЙСТВА
local obj= {}
    obj.firstName = fName
    obj.lastName = lName

-- метод
function obj:getName()
    return self.firstName
end

--чистая магия!
setmetatable(obj, self)
self.__index = self; return obj
end

--создаем экземпляр класса
vasya = Person:new("Вася", "Пупкин")

--обращаемся к свойству
print(vasya.firstName)    --> результат: Вася

--обращаемся к методу
print(vasya:getName())    --> результат: Вася

```

Как видите, все очень просто. Если кто-то путается где ставить точку, а где двоеточие, правило следующее: если обращаемся к свойству — ставим точку (object.name), если к методу — ставим двоеточие (object:getName()).

Дальше интереснее.

Как известно, ООП держится на трех китах: наследование, инкапсуляция и полиморфизм. Проведем «разбор полетов» в этом же порядке.

Наследование

Допустим, нам нужно создать класс унаследованный от предыдущего (Person).

▼ class Woman

```
Woman = {}  
--наследуемся  
setmetatable(Woman, {__index = Person})  
--проверяем  
masha = Woman:new("Марья", "Ивановна")  
print(masha:getName()) --->результат: Марья
```

Все работает, но лично мне не нравится такой вариант наследования, некрасиво. Поэтому я просто создаю глобальную функцию extended():

▼ extended()

```
function extended (child, parent)  
    setmetatable(child, {__index = parent})  
end
```

Теперь наследование классов выглядит куда красивее:

▼ class Woman

```
Woman = {};  
--наследуемся  
    extended(Woman, Person)  
--проверяем  
masha = Woman:new("Марья", "Ивановна")
```

```
print(masha:getName()) --->результат: Марья
```

Инкапсуляция

Все свойства и методы до этого момента в наших классах были публичные, но мы так же легко можем создавать и приватные:

▼ class Person

```
Person = {}
function Person:new(name)
    local private = {}
        --приватное свойство
        private.age = 18

    local public = {}
        --публичное свойство
        public.name = name or "Вася"    -- "Вася" - это значение
        --публичный метод
        function public:getAge()
            return private.age
        end

    setmetatable(public,self)
    self.__index = self; return public
end

vasya = Person:new()

print(vasya.name)          --> результат: Вася

print(vasya.age)           --> результат: nil

print(vasya:getAge())      --> результат: 18
```

Видите? Все почти так же как вы и привыкли.

Полиморфизм

Тут все еще проще.

▼ полиморфизм

```
Person = {}
function Person:new(name)
    local private = {}
    private.age = 18

    local public = {}
    public.name = name or "Вася"

    --это защищенный метод, его нельзя переопределить
    function public:getName()
        return "Person protected "..self.name
    end

    --это открытый метод, его можно переопределить
    function Person:getName2()
        return "Person "..self.name
    end

    setmetatable(public, self)
    self.__index = self; return public
end

--создадим класс, унаследованный от Person
Woman = {}
extended(Woman, Person) --не забываем про эту функцию

--переопределим защищенный метод
function Woman:getName()
```

```

function Woman:getName()
    return "Woman protected "..self.name
end

--переопределим метод getName2()
function Woman:getName2()
    return "Woman "..self.name
end

--проверим
masha = Woman:new()

print(masha:getName())    --> Person protected Вася

print(masha:getName2())   --> Woman Вася

```

Итак, что мы тут сделали?

- создали класс Person, с двумя методами: getName() и getName2(), первый из них защищен от переопределения;
- создали класс Woman и унаследовали его от класса Person;
- переопределили оба метода в классе Woman. Первый не переопределится;
- получили профит!

Кстати, открытые методы можно определять так же и вне тела класса:

▼ полиморфизм

```

Person = {}
function Person:new(name)
    local private = {}
    private.age = 18

    local public = {}
    public.name = name or "Вася"

    --это защищенный метод, его нельзя переопределить
end

```

```

        function public:getName()
            return "Person protected "..self.name
        end

        setmetatable(public,self)
        self.__index = self; return public
    end

    --это открытый метод, его можно
    function Person:getName2()
        return "Person "..self.name
    end
end

```

А что делать, если нужно вызвать метод базового класса, который у нас переопределен? Это тоже делается легко!
Синтаксис таков: РодительскийКласс.Метод(сам_объект, параметры (если есть)).

▼ class Woman

```

--создадим класс, унаследованный от Person
Woman = {}
extended(Woman, Person) --не забываем про эту функцию

--переопределим метод setName
function Woman:getName2()
    return "Woman "..self.name
end

print(masha:getName2()) --> Woman Вася

--вызываем метод родительского класса
print(Person.getName2(masha)) --> Person Вася

```

Постскриптум

На этом все, искренне надеюсь, что хоть кому-нибудь эта статья окажется полезной.

Напоследок приведу полный код, можете его скопипастить в IDE и убедиться в работоспособности.

▼ Полный код

```
function extended (child, parent)
    setmetatable(child, {__index = parent})
end

Person = {}
function Person:new(name)

    local private = {}
    private.age = 18

    local public = {}
    public.name = name or "Вася"

    --это защищенный метод, его нельзя переопределить
    function public:getName()
        return "Person protected "..self.name
    end

    --этот метод можно переопределить
    function Person:getName2()
        return "Person "..self.name
    end
    setmetatable(public, self)
    self.__index = self;
    return public
end
```



```
--создадим класс, унаследованный от Person
Woman = {}
extended(Woman, Person) --не забываем про эту функцию

--переопределим метод setName
function Woman:getName2()
    return "Woman "..self.name
end

masha = Woman:new()
print(masha:getName2()) --> Woman Вася

--вызываем метод родительского класса
print(Person.getName2(masha)) --> Person Вася
```

Теги: [луа](#), [ооп](#), [lua](#)

Хабы: [ООП](#), [Lua](#)

↑ +21 ↓

🔖 142



💬 21

Редакторский дайджест



Присылаем лучшие статьи раз в месяц



7

0

Карма Рейтинг

Антон @ant00N

Разработчик игр

Реклама