# JS 对象

从入门到工作：JS 全解

# 版权声明

# 联系方式

如果你想要购买本课程
请微信联系 xiedaimala02 或 xiedaimala03

如果你发现有人盗用本课程
请微信联系 xiedaimala02 或 xiedaimala03

# 七种数据类型

请背诵并默写

# 五个 Falsy 值

请背诵并默写

# 答案

- 七种数据类型
  - ✓ number string bool symbol
  - ✓ null undefined
  - ✓ object
- 五个 falsy 值
  - ✓ null undefined
  - ✓ 0 NaN
  - ✓ '' (空字符串)

# 对象 object

第七种数据类型，唯一一种复杂类型

# 对象

- 定义
  - ✓ 无序的数据集合
  - ✓ 键值对的集合
- 写法

```
let obj = { 'name': 'frank', 'age': 18 }
let obj = new Object({'name': 'frank'})
console.log({ 'name': 'frank, 'age': 18 })
```

直接写不是对象。必须在前面加个东西才是对象（JS脑子有坑）。直接写是标签。

- 细节
  - ✓ 键名是字符串，不是标识符，可以包含任意字符
  - ✓ 引号可省略，省略之后就只能写标识符
  - ✓ 就算引号省略了，键名也还是字符串（重要）

# 属性名

每个 key 都是对象的属性名（property）

# 属性值

每个 value 都是对象的属性值

# 奇怪的属性名

- 所有属性名会自动变成字符串

```
let obj = {
  1: 'a',
  3.2: 'b',
  1e2: true,
  1e-2: true,
  .234: true,
  0xFF: true
};
Object.keys(obj)
=> ["1", "100", "255", "3.2", "0.01", "0.234"]
```

会变成100，然后100变成字符串。

如果加引号就不会变。

’0.01’:true

’0.234’:true

’255’:true

- 细节

Object.keys(obj) 可以得到 obj 的所有 key

# 变量作属性名

- 如何用变量做属性名
  - ✓ 之前都是用常量做属性名
  - ✓ let p1 = 'name'
  - ✓ let obj = { p1 : 'frank'} 这样写，属性名为 'p1'
  - ✓ let obj = { [p1] : 'frank' } 这样写，属性名为 'name'
- 对比
  - ✓ 不加 [ ] 的属性名会自动变成字符串
  - ✓ 加了 [ ] 则会当做变量求值
  - ✓ 值如果不是字符串，则会自动变成字符串

```
> var obj = {
    [1+2+3+4] : '十'
  }
> obj
< {10:'十'}
```

# 对象的隐藏属性

- 隐藏属性
  - JS 中每一个对象都有一个隐藏属性
  - 这个隐藏属性储存着其共有属性组成的对象的地址
  - 这个共有属性组成的对象叫做原型
  - 也就是说，隐藏属性储存着原型的地址
- 代码示例
  - var obj = {}
  - obj.toString() // 居然不报错
  - 因为 obj 的隐藏属性对应的对象上有 toString()

# 超纲知识

- 除了字符串，symbol 也能做属性名

```
let a = Symbol()
let obj = { [a]: 'Hello' }
```

- 这有什么用呢?
  - ✓ 目前，屁用都没用，很久很久以后会有用
  - ✓ 在学习「迭代」时会用到

# 增删改查

我们又遇到这个词了，这次增删改成对象的属性

# 删除属性

- delete obj.xxx 或 delete obj['xxx']
  - ✓ 即可删除 obj 的 xxx 属性
  - ✓ 请区分「属性值为 undefined」和「不含属性名」
- 不含属性名
  - ✓ 'xxx' in obj === false 包括自己的和共有的
- 含有属性名，但是值为 undefined
  - ✓ 'xxx' in obj && obj.xxx === undefined
- 注意 obj.xxx === undefined
  - ✓ 不能断定 'xxx' 是否为 obj 的属性
- 类比
  - ✓ 你有没有卫生纸?
  - ✓ A: 没有 // 不含属性名
  - ✓ B: 有，但是没带 // 含有属性名，但是值为 undefined

# 程序员就是这么严谨

没有就是没有，undefined 就是 undefined

绝不含糊

# 查看所有属性（读属性）

- 查看自身所有属性
  - Object.keys(obj)
- 查看自身+共有属性
  - console.dir(obj)
  - 或者自己依次用 Object.keys 打印出 obj.__proto__
- 判断一个属性是自身的还是共有的
  - obj.hasOwnProperty('toString')

# 原型

- 每个对象都有原型
  - 原型里存着对象的共有属性
  - 比如 obj 的原型就是一个对象
  - obj.__proto__ 存着这个对象的地址
  - 这个对象里有 toString / constructor / valueOf 等属性
- 对象的原型也是对象
  - 所以对象的原型也有原型
  - obj = {} 的原型即为所有对象的原型
  - 这个原型包含所有对象的共有属性，是对象的根
  - 这个原型也有原型，是 null

# 查看属性

```
> let obj={name:'frank',age:18}
< undefined
> obj['name']
< "frank"
> obj[name]
< undefined
> window.name='age'
< "age"
> obj[name]
< 18
> obj['na'+'me']
< "frank"      //'na'+'me'不是字符串，所以先求值，得到'name'字符串值
```

- 两种方法查看属性
  - ✓ 中括号语法：obj['key']
  - ✓ 点语法：obj.key 当key是数字时不行
  - ✓ 坑新人语法：obj[key] // 变量 key 值一般不为 'key'

中括号里如果是变量，必须先求变量的值

- 请优先使用中括号语法
  - ✓ 点语法会误导你，让你以为 key 不是字符串
  - ✓ 等你确定不会弄混两种语法，再改用点语法

```
> obj[console.log('name')]
  name
< undefined
//console.log返回undefined。由于没有字符串，所以把undefined变成字符串，即obj['undefined']。由于obj没有这个key，所以得到udnefined。
```

obj.name 等价于 obj['name']

obj.name 不等价于 obj[name]

简单来说，这里的 name 是字符串，而不是变量

let name = 'frank'

obj[name] 等价于

obj['frank']

而不是

obj['name'] 和 obj.name

谁记错了，谁带绿帽子

# 考题

看你分清变量 name 和常量字符串 'name' 没有

# 必须搞清楚这一题

- 代码

```
let list = ['name', 'age', 'gender']
let person = {
        name:'frank', age:18, gender:'man'}
for(let i = 0; i < list.length; i++){
  let name = list[i]
  console.log(person__???__)
}
```

使得 person 的所有属性被打印出来

- 选项

1. console.log(person.name)
2. console.log(person[name])

# 什么时候想通这一题
# 什么时候睡觉

就是这么刚

# 区分 name 和 'name' 为什么这么重要

因为如果你现在不搞清楚

那么你在学 Vue 的时候，会更加迷惑

# 再讲一遍，以示重要

时光倒流

# 必须搞清楚这一题

- 代码

```
let list = ['name', 'age', 'gender']
let person = {
        name:'frank', age:18, gender:'man'}
for(let i = 0; i < list.length; i++){
  let name = list[i]
  console.log(person__???__)
}
```

使得 person 的所有属性被打印出来

- 选项

1. console.log(person.name)
2. console.log(person[name])

# 修改或增加属性（写属性）

- 直接赋值

```
let obj = {name: 'frank'} // name 是字符串
obj.name = 'frank' // name 是字符串
obj['name'] = 'frank'
obj[name] = 'frank' // 错，因 name 值不确定
obj['na'+'me'] = 'frank'
let key = 'name'; obj[key] = 'frank'
let key = 'name'; obj.key = 'frank' // 错
因为 obj.key 等价于 obj['key']
```

- 批量赋值

```
Object.assign(obj, {age: 18, gender: 'man'})
```

# 修改或增加共有属性

- 无法通过自身修改或增加共有属性
  - let obj = {}, obj2 = {} // 共有 toString
  - obj.toString = 'xxx' 只会在改 obj 自身属性
  - obj2.toString 还是在原型上
- 我偏要修改或增加原型上的属性
  - obj.__proto__.toString = 'xxx' // 不推荐用 __proto__
  - Object.prototype.toString = 'xxx'
  - 一般来说，不要修改原型，会引起很多问题

# 修改隐藏属性

- 不推荐使用 __proto__

```
let obj = {name:'frank'}
let obj2 = {name: 'jack'}
let common = {kind: 'human'}
obj.__proto__ = common
obj2.__proto__ = common
```

- 推荐使用 Object.create

```
let obj = Object.create(common)
obj.name = 'frank'
let obj2 = Object.create(common)
obj2.name = 'jack'
```

规范大概的意思是，要改就一开始就改，别后来再改

# 总结

- 删
  - delete obj['name']
  - 'name' in obj // false
  - obj.hasOwnProperty('name')  // false
- 查
  - Object.keys(obj)
  - console.dir(obj)
  - obj['name']
  - obj.name // 记住这里的 name 是字符串
  - obj[name]  // 记住这里的 name 是变量

# 总结2

- 改
  - 改自身 obj['name'] = 'jack'
  - 批量改自身 Object.assign(obj, {age:18, ...})
  - 改共有属性 obj.__proto__['toString'] = 'xxx'
  - 改共有属性 Object.prototype['toString'] = 'xxx'
  - 改原型 obj.__proto__ = common
  - 改原型 let obj = Object.create(common)
  - 注：所有 __proto__ 代码都是强烈不推荐写的
- 增
  - 基本同上：已有属性则改；没有属性则增。

# 作业内容

- 一篇博客
  - ✓ 题目《JS 对象基本用法》
  - ✓ 内容1：声明对象的两种语法
  - ✓ 内容2：如何删除对象的属性
  - ✓ 内容3：如何查看对象的属性
  - ✓ 内容4：如何修改或增加对象的属性
  - ✓ 内容5：'name' in obj和obj.hasOwnProperty('name')的区别