JS 对象分类

从入门到工作：JS 全解

版权声明

本内容版权属杭州饥人谷教育科技有限公司（简称饥人谷）所有。

任何媒体、网站或个人未经本网协议授权不得转载、链接、转贴，
或以其他方式复制、发布和发表。

已获得饥人谷授权的媒体、网站或个人在使用时须注明「资料来源：饥人谷」。

对于违反者，饥人谷将依法追究责任。

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

对象需要分类吗

这是一个值得思考的问题

我们来做一个小程序

输出各种形状的面积和周长

正方形

- 代码

```
let square = {
  width: 5,
  getArea(){
    return this.width * this.width
  },
  getLength(){
    return this.width * 4
  }
}
```

- 分析

✓ 正方形拥有三个属性：边长、面积、周长

给我来一打正方形

- 这样写代码的，要么是新人，要么是傻子

```
let square = {
  width: 5,
  getArea(){
    return this.width * this.width
  },
  getLength(){
    return this.width * 4
  }
}
let square2 = {
  width: 6,
  getArea(){
    return this.width * this.width
  },
  getLength(){
    return this.width * 4
  }
}
let square3 = { ...
```

简单，用循环搞定

for 循环

不就是12个对象嘛

- 代码

```
let squareList = []
for(let i = 0; i<12; i++){
  squareList[i] = {
    width: 5,
    getArea(){
      return this.width * this.width
    },
    getLength(){
      return this.width * 4
    }
  }
}
```

- 看，搞定了
  - ✓ width 不全是 5 啊，怎么办

好办

- 代码

```
let squareList = []
let widthList = [5,6,5,6,5,6,5,6,5,6,5,6]
for(let i = 0; i<12; i++){
  squareList[i] = {
    width: widthList[i],
    getArea(){
      return this.width * this.width
    },
    getLength(){
      return this.width * 4
    }
  }
}
```

- 看，又搞定了
  - ✓ 垃圾代码，浪费了太多内存，自己画内存图就知道了

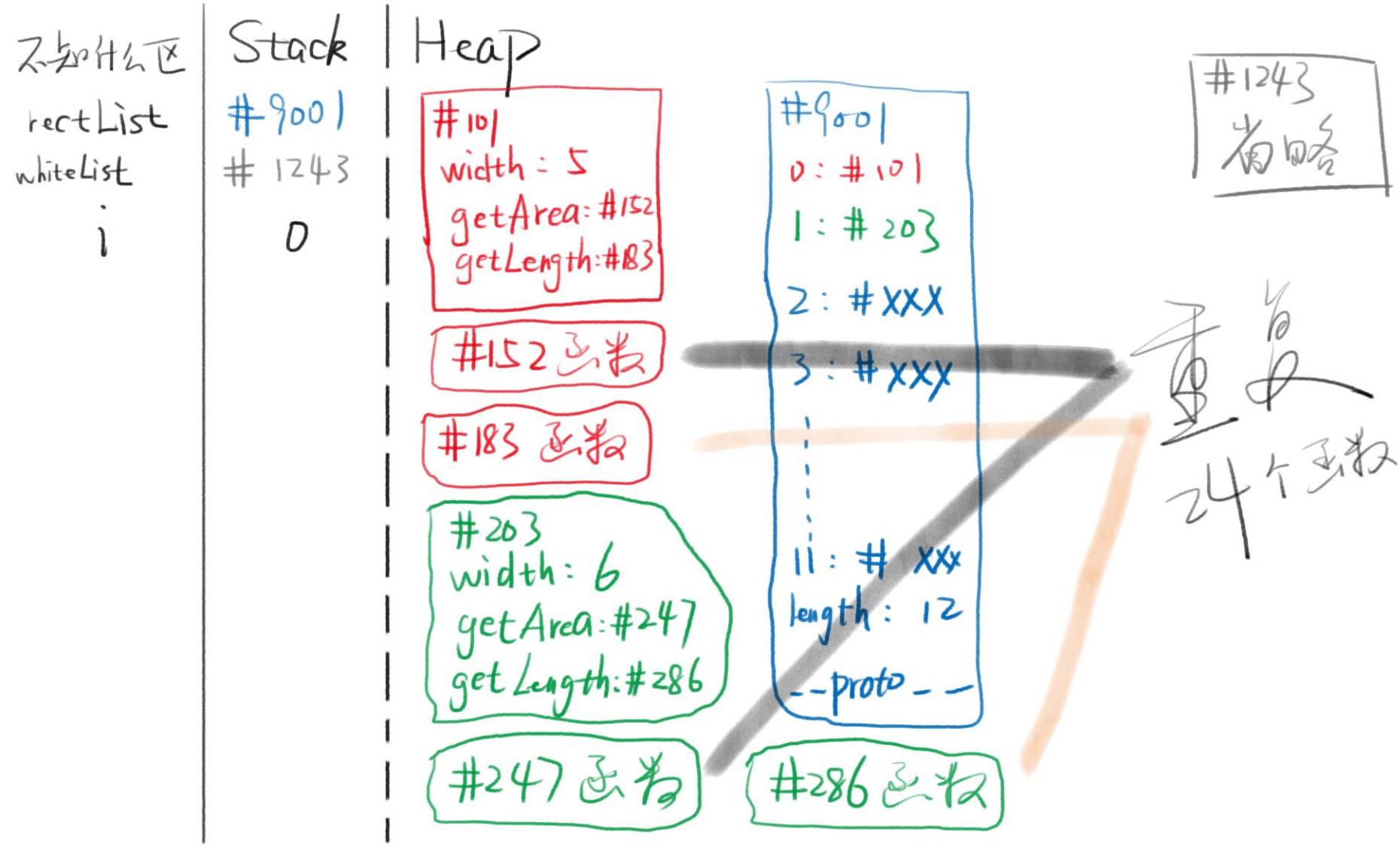
在看录屏的你

赶紧动手给我画内存图！

你画了之后，再看我怎么画

会画内存图的人，比其他人理解得更好

内存图：重复的函数


不知什么区
rectList
whiteList
i
Stack
#9001
#1243
0
Heap
#101
width: 5
getArea: #152
getLength: #183
#152 函数
#183 函数
#203
width: 6
getArea: #247
getLength: #286
#247 函数
#9001
0: #101
1: #203
2: #XXX
3: #XXX
11: #XXX
length: 12
__proto__
#286 函数
#1243
省略
重复
24个函数

所以你看，你的代码很垃圾

继续改

借助原型

将12个对象的共用属性放到原型里

好办

- 代码

```
let squareList = []
let widthList = [5,6,5,6,5,6,5,6,5,6,5,6]
let squarePrototype = {
  getArea(){
    return this.width * this.width
  },
  getLength(){
    return this.width * 4
  }
}
for(let i = 0; i<12; i++){
  squareList[i] = Object.create(squarePrototype)
  squareList[i].width = widthList[i]
}
```

- 看，双搞定了
  - ✓ 还是垃圾代码！你创建 square 的代码太分散了！

太分散了吗

那就把代码抽离到一个函数里，然后调用函数

抽离到函数

构造函数：可以构造出对象的函数

```
let squareList = []
let widthList = [5,6,5,6,5,6,5,6,5,6,5,6]
function createSquare(width){ //此函数叫做构造函数
  let obj = Object.create(squarePrototype)
  // 以 squarePrototype 为原型创建空对象
  obj.width = width
  return obj
}
let squarePrototype = {
  getArea(){
    return this.width * this.width
  },
  getLength(){
    return this.width * 4
  }
}
for(let i = 0; i<12; i++){
  squareList[i] = createSquare(widthList[i])
  // 这下创建 square 很简单了吧!
}
```

squarePrototype 原型
和
createSquare 函数
还是分散的

能不能组合在一起

函数和原型结合

```
let squareList = []
let widthList = [5,6,5,6,5,6,5,6,5,6,5,6]

function createSquare(width){
  let obj = Object.create(createSquare.squarePrototype) // 先使用后定义？NO
  obj.width = width
  return obj
}
createSquare.squarePrototype = { //把原型放到函数上，结合够紧密了吗?
  getArea(){
    return this.width * this.width
  },
  getLength(){
    return this.width * 4
  },
  constructor: createSquare //方便通过原型找到构造函数
}
for(let i = 0; i<12; i++){
  squareList[i] = createSquare(widthList[i])
  console.log(squareList[i].constructor)
  // constructor 可以知道谁构造了这个对象：你妈是谁?
}
```

只是定义了，没使用。在最后使用的。
还是先定义再使用。

这段代码几乎完美

为什么不固定下来，让每个 JS 开发者直接用呢

new 操作符

让我们感受 JS 之父的爱

函数和原型结合（重写）

```
let squareList = []
let widthList = [5,6,5,6,5,6,5,6,5,6,5,6]
function Square(width){
  this.width = width
}
Square.prototype.getArea = function(){
  return this.width * this.width
}
Square.prototype.getLength = function(){
  return this.width * 4
}
for(let i = 0; i<12; i++){
  squareList[i] = new Square(widthList[i])
  console.log(squareList[i].constructor)
}
// 多美，几乎没有一句多余的废话

// 每个函数都有 prototype 属性，这是 JS 之父故意的
// 每个 prototype 都有 constructor 属性，也是故意的
```

对比

```
function createSquare(width){
  let obj = Object.create(createSquare.squarePrototype)
  obj.width = width
  return obj
}
createSquare.squarePrototype = {
  getArea(){
    return this.width * this.width
  },
  getLength(){
    return this.width * 4
  },
  constructor: createSquare
}
let square = createSquare(5)
```

上面的代码被简化为下面的代码
唯一的区别是要用 new 来调用

```
function Square(width){
  this.width = width
}
Square.prototype.getArea = function(){
  return this.width * this.width
}
Square.prototype.getLength = function(){
  return this.width * 4
}
let square = new Square(5)
```

总结

- new X() 自动做了四件事情
  - ✓ 自动创建空对象
  - ✓ 自动为空对象关联原型，原型地址指定为 X.prototype
  - ✓ 自动将空对象作为 this 关键字运行构造函数
  - ✓ 自动 return this
  - ✓ ——这就是 JS 之父的爱
- 构造函数 X
  - ✓ X 函数本身负责给对象本身添加属性
  - ✓ X.prototype 对象负责保存对象的共用属性

题外话：代码规范

- 大小写
  - 所有构造函数（专门用于创建对象的函数）首字母大写
  - 所有被构造出来的对象，首字母小写
- 词性
  - new 后面的函数，使用名词形式
  - 如 new Person()、new Object()
  - 其他函数，一般使用动词开头
  - 如 createSquare(5)、createElement('div')
  - 其他规则以后再说

接下来总结一个重要的公式

也是 JS 里唯一的一个公式

如何确定一个对象的原型

- 为什么
  - ✓ let obj = new Object() 的原型是 Object.prototype
  - ✓ let arr = new Array() 的原型是 Array.prototype
  - ✓ let square = new Square() 的原型是 Square.prototype
  - ✓ let fn = new Function() 的原型是 Function.prototype
- 因为 new 操作故意这么做的
  - new X() 自动做了四件事情
    - ✓ 自动创建空对象
    - ✓ 自动为空对象关联原型，原型地址指定为 X.prototype
    - ✓ 自动将空对象作为 this 关键字运行构造函数
    - ✓ 自动 return this
    - ✓ ——这就是 JS 之父的爱

结论

你是谁构造的
你的原型就是谁的 prototype 属性对应的对象

原型公式

对象.__proto__ === 其构造函数.prototype

参考资料

- 《__proto__ 和 prototype 存在的意义是什么》方方

来做几个题

看看你理解这个公式了吗

难度1

let x = {}

请问：

1. x 的原型是什么？
2. x.__proto__ 的值是什么？
3. 上面两个问题是等价的吗？
4. 请用内存图画出 x 的所有属性

难度2

- let square = new Square(5)

请问：

1. square 的原型是什么？
2. square.__proto__ 的值是什么？
3. 请用内存图画出 x 的所有属性

难度3

请问：

1. Object.prototype 是哪个函数构造出来的？ 不知道。
2. Object.prototype 的原型是什么？ null。没有原型
3. Object.prototype.__proto__？
4. 请用内存图画出上述内容

构造函数、prototype、new

我们已经通过 Square 理解了

Square 最终版（存疑）

- 代码

```
function Square(width){
  this.width = width
}
Square.prototype.getArea = function(){
  return this.width * this.width
}
Square.prototype.getLength = function(){
  return this.width * 4
}
let square = new Square(5)
square.width
square.getArea()
square.getLength()
```

正方形搞定了
再给我写个圆呗

甲方爸爸总是会提出新(wu)的(li)需求

Circle

- 代码

```
function Circle(radius){
  this.radius = radius
}
Circle.prototype.getArea = function(){
  return Math.pow(this.radius,2) * Math.PI
}
Circle.prototype.getLength = function(){
  return this.radius * 2 * Math.PI
}
let circle = new Circle(5)
circle.radius
circle.getArea()
circle.getLength()
```

圆搞定了
再给我写个长方形呗

甲方爸爸总是会提出新(wu)的(li)需求

Rectangle

- 代码

```
function Rect(width, height){
  this.width = width
  this.height = height
}
Rect.prototype.getArea = function(){
  return this.width * this.height
}
Rect.prototype.getLength = function(){
  return (this.width + this.height) * 2
}
let react = new Rect(4,5)
rect.width
rect.height
rect.getArea()
rect.getLength()
```

对象需要分类吗

再次回到开头的问题

需要分类

- 理由一
  - 有很多对象拥有一样的属性和行为
  - 需要把它们分为同一类
  - 如 square1 和 square2
  - 这样创建类似对象的时候就很方便
- 理由二
  - 但是还有很多对象拥有其他的属性和行为
  - 所以就需要不同的分类
  - 比如 Square / Circle / Rect 就是不同的分类
  - Array / Function 也是不同的分类
  - 而 Object 创建出来的对象，是最没有特点的对象

类型 V.S. 类

- 类型
  - ✓ 类型是 JS 数据的分类，有 7 种
  - ✓ 四基两空一对象
- 类
  - ✓ 类是针对于对象的分类，有无数种
  - ✓ 常见的有 Array、Function、Date、RegExp 等

数组对象

- 定义一个数组
  - let arr = [1,2,3]
  - let arr = new Array(1,2,3) // 元素为 1,2,3
  - let arr = new Array(3) // 长度为 3
- 数组对象的自身属性
  - '0' / '1' / '2' / 'length'
  - 注意，属性名没有数字，只有字符串
- 数组对象的共用属性
  - 'push' / 'pop' / 'shift' / 'unshift' / 'join'
  - 其实就是英语小课堂啦，用法都在 MDN
  - 后面会有单独课程教这些 API

shift:从数组中删除第一个元素，并返回元素的值。
unshift:将一个或多个元素添加到数组的开头，并返回数组的新长度。
join：arr.join('方')用某个东西连起数组元素。返回字符串。

添加元素　刚刚添加的元素

函数对象

- 定义一个函数
  - function fn(x,y){return x+y}
  - let fn2 = function fn(x,y){return x+y}
  - let fn = (x,y) => x+y
  - let fn = new Function('x','y', 'return x+y')
- 函数对象自身属性
  - 'name' / 'length'
- 函数对象共用属性
  - 'call' / 'apply' / 'bind'
  - 后面会有单独课程介绍函数

JS 终极一问

- window 是谁构造的
  - ✓ Window
  - ✓ 可以通过 constructor 属性看出构造者
- window.Object 是谁构造的
  - ✓ window.Function
  - ✓ 因为所有函数都是 window.Function 构造的
- window.Function 是谁构造的
  - ✓ window.Function
  - ✓ 因为所有函数都是 window.Function 构造的
  - ✓ 自己构造的自己？并不是这样，这是「上帝」的安排
  - ✓ 浏览器构造了 Function，然后指定它的构造者是自己

```
function Square(width){
  this.width = width
}
Square.prototype.getArea = function(){
  return this.width * this.width
}
Square.prototype.getLength = function(){
  return this.width * 4
}
```

非常遗憾

这个代码被某些前端认为是过时的

```
class Square{
  constructor(width){
    this.width = width
  }
  getArea(){
    return this.width * this.width
  }
  getLength: function(){
    return this.width * 4
  } // 函数的两种写法都对
}
```

ES 6 引入了新语法

class 统治天下

class 语法引入了更多概念

```
class Square{
  static x = 1
  width = 0
  constructor(width){
    this.width = width
  }
  getArea(){
    return this.width * this.width
  }
  getLength: function(){
    return this.width * 4
  }
  get area2(){ // 只读属性
    return this.width * this.width
  }
}
```

新手建议

- 这么多新语法怎么学
  - 两种学法
  - 花一大块时间把 MDN class 文档全部看完，并记下来
  - 看到方方用什么，就学一点，课程学完，就都学会了
  - 推荐第二种学法，因为新语法实在太多了
  - 在实践中学，记得更牢
- 到底有多少新语法
  - 我已经整理了 ES6 的所有新语法，点击查看
  - 关于类和对象的新语法有页面1，页面2和页面3
  - 所以前端学得越早，越轻松，当年我只用学 ES3

用 class 重写 Circle

- 代码

```
class Circle{
  constructor(radius){
    this.radius = radius
  }
  getArea(){
    return Math.pow(this.radius,2) * Math.PI
  }
  getLength(){
    return this.radius * 2 * Math.PI
  }
}
let circle = new Circle(5)
circle.radius
circle.getArea()
circle.getLength()
```

用 class 重写 Rect

- 代码

```
class Rect{
  constructor(width, height){
    this.width = width
    this.height = height
  }
  getArea(){
    return this.width * this.height
  }
  getLength(){
    return (this.width + this.height) * 2
  }
}
let react = new Rect(4,5)
rect.width
rect.height
rect.getArea()
rect.getLength()
```

原型好，还是类好?

都是用来给对象分类的

再见

下节课学习数组、函数等