编程语言如何演化

—— 以 JS 的 private 为例

http://johnhax.net/2017/js-private/slide?qcon#0

Introduction of My Company and Myself

http://johnhax.net/2017/js-private/slide?qcon#1

百姓网

www.baixing.com

http://johnhax.net/2017/js-private/slide?qcon#2

github: @hax

zhihu: 贺师俊

weibo: @johnhax

编程语言

http://johnhax.net/2017/js-private/slide?qcon#4

两种

极端

http://johnhax.net/2017/js-private/slide?qcon#5

你用什么语言?

(鄙视脸)

http://johnhax.net/2017/js-private/slide?qcon#6

语言都一样！

（鄙视脸）

http://johnhax.net/2017/js-private/slide?qcon#7

人们总是

http://johnhax.net/2017/js-private/slide?qcon#8

高估短期

低估长期

http://johnhax.net/2017/js-private/slide?qcon#9

不太可能是
项目成败的
决定性因素

http://johnhax.net/2017/js-private/slide?qcon#10

影响团队的长期生产力

http://johnhax.net/2017/js-private/slide?qcon#11

主题：演化

http://johnhax.net/2017/js-private/slide?qcon#12

新特性

http://johnhax.net/2017/js-private/slide?qcon#13

What/How

http://johnhax.net/2017/js-private/slide?qcon#14

Why

http://johnhax.net/2017/js-private/slide?qcon#15

为什么要引入

新特性

http://johnhax.net/2017/js-private/slide?qcon#16

为什么新特性

长这样

http://johnhax.net/2017/js-private/slide?qcon#17

静态 ➡ 动态

http://johnhax.net/2017/js-private/slide?qcon#18

剧烈演化

http://johnhax.net/2017/js-private/slide?qcon#19

Rust/Swift

http://johnhax.net/2017/js-private/slide?qcon#20

不怎么变化

http://johnhax.net/2017/js-private/slide?qcon#21

Go

http://johnhax.net/2017/js-private/slide?qcon#22

JS?

http://johnhax.net/2017/js-private/slide?qcon#23

矛盾体

http://johnhax.net/2017/js-private/slide?qcon#24

private

http://johnhax.net/2017/js-private/slide?qcon#25

讨论历史可以追溯到ES4

贯穿了整个ES6的开发历史

最终决定postpone到ES7+

先后至少有4份proposal

对于JS程序员并没有立即可用的实践价值

http://johnhax.net/2017/js-private/slide?qcon#27

Stage 2

http://johnhax.net/2017/js-private/slide?qcon#28

ES ?

http://johnhax.net/2017/js-private/slide?qcon#29

对工程实践中系统演化有借鉴意义

http://johnhax.net/2017/js-private/slide?qcon#30

可见性

- public
- private
- protected

http://johnhax.net/2017/js-private/slide?qcon#31

可见性

- package-private
- internal
- internal protected
- friend（友元）

http://johnhax.net/2017/js-private/slide?qcon#32

最基本的是

public

private

http://johnhax.net/2017/js-private/slide?qcon#33

private

in JS

http://johnhax.net/2017/js-private/slide?qcon#34

ES5−

No Class

All public

http://johnhax.net/2017/js-private/slide?qcon#35

ES2015

MM Class

All public

http://johnhax.net/2017/js-private/slide?qcon#36

Maximally-Minimal

Class

http://johnhax.net/2017/js-private/slide?qcon#37

MM Class

- constructor
- static methods
- prototype methods

http://johnhax.net/2017/js-private/slide?qcon#38

```
class RGBColor {
  constructor(r, g, b) {
    this._c = [r, g, b]
  }
  toString() {
    const [r, g, b] = this._c
    return `rgb(${r}, ${g}, ${b})`
  }
  get red()   { return this._c[0] }
  get green() { return this._c[1] }
  get blue()  { return this._c[2] }
  static equals(c1, c2) {
    return c1.red   === c2.red
      &&   c1.green === c2.green
      &&   c1.blue  === c2.blue
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#39

```
function RGBColor(r, g, b) {
  this._c = [r, g, b]
}
RGBColor.prototype.toString = function () {
  const [r, g, b] = this._c
  return `rgb(${r}, ${g}, ${b})`
}
Object.defineProperty(RGBColor.prototype,
  'red', {
    get: function () { return this._c[0] }
  })
...
RGBColor.equals = function (c1, c2) {
  return c1.red   === c2.red
    &&   c1.green === c2.green
    &&   c1.blue  === c2.blue
}
```

http://johnhax.net/2017/js-private/slide?qcon#40

Only methods

No data properties

http://johnhax.net/2017/js-private/slide?qcon#41

为何有此限制?

http://johnhax.net/2017/js-private/slide?qcon#42

prototype 的坑

http://johnhax.net/2017/js-private/slide?qcon#43

```
function A() {}
A.prototype.myData = {name: 'hax'}
let a1 = new A()
a1.myData.name  // 'hax'
a1.myData.name = 'a1'  //    坑
```

http://johnhax.net/2017/js-private/slide?qcon#44

```
function A() {}
A.prototype.myData = {name: 'hax'}
let a1 = new A(), a2 = new A()
a1.myData.name = 'a1'  //    坑
a2.myData.name  // 'a1' --- WTF!
```

http://johnhax.net/2017/js-private/slide?qcon#45

```
function A() {}
A.prototype.myData = {name: 'hax'}
let a1 = new A, a2 = new A
a1.myData = {name: 'a1'}  // “正确”写法
a2.myData.name  // 'hax'
```

http://johnhax.net/2017/js-private/slide?qcon#46

什么样的属性可以放在 prototype 上?

http://johnhax.net/2017/js-private/slide?qcon#47

- 读 primitive
- 写 primitive
- 读 object
- 写 object
- 读 object 上的属性
- 写 object 上的属性

http://johnhax.net/2017/js-private/slide?qcon#48

MM Class

Only methods

http://johnhax.net/2017/js-private/slide?qcon#49

method

所有实例共享

http://johnhax.net/2017/js-private/slide?qcon#50

拖

Postpone

http://johnhax.net/2017/js-private/slide?qcon#51

Maximally-Minimal

Class

http://johnhax.net/2017/js-private/slide?qcon#52

求同存异

分而治之

http://johnhax.net/2017/js-private/slide?qcon#53

反而避免
久拖不决

http://johnhax.net/2017/js-private/slide?qcon#54

让实践说话

http://johnhax.net/2017/js-private/slide?qcon#55

拖延大法好

http://johnhax.net/2017/js-private/slide?qcon#56

为什么要有 Private？

http://johnhax.net/2017/js-private/slide?qcon#57

实践证明了...

http://johnhax.net/2017/js-private/slide?qcon#58

Prior Art

http://johnhax.net/2017/js-private/slide?qcon#59

- 命名约定（如下划线前缀）
- 名称变换（如python）
- 基于Symbol
- 基于闭包
- 基于WeakMap

http://johnhax.net/2017/js-private/slide?qcon#60

命名约定

http://johnhax.net/2017/js-private/slide?qcon#61

```
class RGBColor {
  constructor(r, g, b) {
    this._hex = r * 0x10000 + g * 0x100 + b
  }
  get red()   { return  this._hex >> 16         }
  get green() { return (this._hex >>  8) & 0xff }
  get blue()  { return  this._hex        & 0xff }
  static equals(c1, c2) {
    return c1._hex === c2._hex
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#62

命名约定

Pros and Cons

http://johnhax.net/2017/js-private/slide?qcon#63

- 简单易行
- 无性能损失
- 可能命名冲突
- 很容易绕过
- 更接近 protected

http://johnhax.net/2017/js-private/slide?qcon#64

名称变换

http://johnhax.net/2017/js-private/slide?qcon#65

```
class RGBColor {
  constructor(r, g, b) {
    this._hex = r * 0x10000 + g * 0x100 + b
  }
  get red()   { return  this._hex >> 16         }
  get green() { return (this._hex >>  8) & 0xff }
  get blue()  { return  this._hex        & 0xff }
  static equals(c1, c2) {
    return c1._hex === c2._hex
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#66

```
const hex = 'RGBColor:field:hex'
class RGBColor {
  constructor(r, g, b) {
    this[hex] = r * 0x10000 + g * 0x100 + b
  }
  get red()   { return  this[hex] >> 16          }
  get green() { return (this[hex] >>  8) & 0xff }
  get blue()  { return  this[hex]        & 0xff }
  static equals(c1, c2) {
    return c1[hex] === c2[hex]
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#67

名称变换

Pros and Cons

http://johnhax.net/2017/js-private/slide?qcon#68

- 简单易行
- 无性能损失
- 可能命名冲突
- 很容易绕过
- 更接近 protected

http://johnhax.net/2017/js-private/slide?qcon#69

- 简单易行，写法略微麻烦
- 无性能损失
- ~~可能命名冲突~~
- 可以被绕过
- ~~更接近 protected~~

http://johnhax.net/2017/js-private/slide?qcon#70

基于Symbol

http://johnhax.net/2017/js-private/slide?qcon#71

```
const hex = 'RGBColor:field:hex'
class RGBColor {
  constructor(r, g, b) {
    this[hex] = r * 0x10000 + g * 0x100 + b
  }
  get red()   { return  this[hex] >> 16          }
  get green() { return (this[hex] >>  8) & 0xff }
  get blue()  { return  this[hex]        & 0xff }
  static equals(c1, c2) {
    return c1[hex] === c2[hex]
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#72

```
const hex = Symbol()
class RGBColor {
  constructor(r, g, b) {
    this[hex] = r * 0x10000 + g * 0x100 + b
  }
  get red()   { return  this[hex] >> 16          }
  get green() { return (this[hex] >>  8) & 0xff }
  get blue()  { return  this[hex]        & 0xff }
  static equals(c1, c2) {
    return c1[hex] === c2[hex]
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#73

基于Symbol

Pros and Cons

http://johnhax.net/2017/js-private/slide?qcon#74

- 简单易行
- 写法略微麻烦
- 无性能损失
- 无命名冲突
- 可以被绕过

http://johnhax.net/2017/js-private/slide?qcon#75

- 简单易行
- 写法略微麻烦
- 无性能损失
- 无命名冲突
- ~~可以被绕过~~

http://johnhax.net/2017/js-private/slide?qcon#76

Unguessability

Unforgeability

http://johnhax.net/2017/js-private/slide?qcon#77

```
Object.getOwnPropertySymbols(obj)
```

http://johnhax.net/2017/js-private/slide?qcon#78

- 简单易行
- 写法略微麻烦
- 无性能损失
- 无命名冲突
- 可通过reflection访问

http://johnhax.net/2017/js-private/slide?qcon#79

基于闭包

http://johnhax.net/2017/js-private/slide?qcon#80

```
class RGBColor {
  constructor(r, g, b) {
    this._hex = r * 0x10000 + g * 0x100 + b
  }
  get red()   { return  this._hex >> 16         }
  get green() { return (this._hex >>  8) & 0xff }
  get blue()  { return  this._hex        & 0xff }
  static equals(c1, c2) {
    return c1._hex === c2._hex
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#81

```
class RGBColor {
  constructor(r, g, b) {
    const hex = r * 0x10000 + g * 0x100 + b
    Object.defineProperties(this, {
      red:   { get: function () { return  hex >> 16         } },
      green: { get: function () { return (hex >>  8) & 0xff } },
      blue:  { get: function () { return  hex        & 0xff } },
    })
  }
  static equals(c1, c2) {  // How to implement it?
    // return c1.hex === c2.hex
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#82

```
class RGBColor {
  constructor(r, g, b) {
    this._hex = r * 0x10000 + g * 0x100 + b
  }

  get red()   { return  this._hex  >> 16         }
  get green() { return (this._hex  >>  8) & 0xff }
  get blue()  { return  this._hex         & 0xff }
  static equals(c1, c2) {
    return c1._hex === c2._hex
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#83

```
class RGBColor {
  constructor(r, g, b) {
    const hex = r * 0x10000 + g * 0x100 + b
    this.hex = function () { return hex }
  }
  get red()   { return  this.hex() >> 16          }
  get green() { return (this.hex() >>  8) & 0xff }
  get blue()  { return  this.hex()        & 0xff }
  static equals(c1, c2) {
    return c1.hex() === c2.hex()
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#84

基于闭包

Pros and Cons

http://johnhax.net/2017/js-private/slide?qcon#85

- 无命名冲突
- 外部（包括static方法）完全无法访问
- 有一定性能代价
- 与 ES6+ 类的方法语义不协调

http://johnhax.net/2017/js-private/slide?qcon#86

prototype方法 VS privilege方法（delegate）
对privilege方法 call/apply/bind 行为不确定
不在prototype上的方法无法进行 super 调用

基于WeakMap

http://johnhax.net/2017/js-private/slide?qcon#88

```
class RGBColor {
  constructor(r, g, b) {
    this._hex = r * 0x10000 + g * 0x100 + b
  }


  get red()   { return  this._hex  >> 16         }
  get green() { return (this._hex  >>  8) & 0xff }
  get blue()  { return  this._hex         & 0xff }
  static equals(c1, c2) {
    return c1._hex === c2._hex
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#89

```
const privates = new WeakMap
function hex(instance) {
  return privates.get(instance).hex
}
class RGBColor {
  constructor(r, g, b) {
    privates.set(this, {
      hex: r * 0x10000 + g * 0x100 + b
    })
  }
  get red()   { return  hex(this)  >> 16         }
  get green() { return (hex(this)  >>  8) & 0xff }
  get blue()  { return  hex(this)         & 0xff }
  static equals(c1, c2) {
    return hex(c1) === hex(c2)
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#90

基于WeakMap
Pros and Cons

http://johnhax.net/2017/js-private/slide?qcon#91

- 无命名冲突
- 外部无法访问
- 写法比较麻烦
- 有一定性能代价

http://johnhax.net/2017/js-private/slide?qcon#92

Prior Art

http://johnhax.net/2017/js-private/slide?qcon#93

- 命名约定（如下划线前缀）
- 名称变换（如python）
- 基于Symbol
- 基于闭包
- 基于WeakMap

http://johnhax.net/2017/js-private/slide?qcon#94

- 命名约定（如下划线前缀）
- ~~名称变换（如python）~~
- 基于Symbol
- ~~基于闭包~~
- 基于WeakMap

http://johnhax.net/2017/js-private/slide?qcon#95

简单加入 private 关键字行不行?

http://johnhax.net/2017/js-private/slide?qcon#96

```
class RGBColor {
  private c
  constructor(r, g, b) {
    this.c = [r, g, b]
  }
  toString() {
    const [r, g, b] = this.c
    return `rgb(${r}, ${g}, ${b})`
  }
  get red()   { return this.c[0] }
  get green() { return this.c[1] }
  get blue()  { return this.c[2] }
  static equals(c1, c2) {
    return c1.red   === c2.red
      &&   c1.green === c2.green
      &&   c1.blue  === c2.blue
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#97

大多数语言就是这样！

包括 TypeScript（JS 超集）

http://johnhax.net/2017/js-private/slide?qcon#98

```
class RGBColor {
  private hex
  constructor(r, g, b) {
    this.hex = r * 0x10000 + g * 0x100 + b
  }
  toString() {
    return `rgb(${this.red}, ${this.green}, ${this.blue})`
  }
  get red()   { return this.hex >> 16 }
  get green() { return (this.hex >> 8) & 0xff }
  get blue()  { return this.hex & 0xff }
  static equals(c1, c2) {
    return c1.hex === c2.hex  //    Semantic?
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#99

没有类型信息！

http://johnhax.net/2017/js-private/slide?qcon#100

所以，现在的 proposal：

http://johnhax.net/2017/js-private/slide?qcon#101

```
class RGBColor {
  #hex
  constructor(r, g, b) {
    #hex = r * 0x10000 + g * 0x100 + b
  }
  toString() {
    return `rgb(${this.red}, ${this.green}, ${this.blue})`
  }
  get red()   { return #hex >> 16 }
  get green() { return (#hex >> 8) & 0xff }
  get blue()  { return #hex & 0xff }
  static equals(c1, c2) {
    return c1.#hex === c2.#hex
  }
}
```

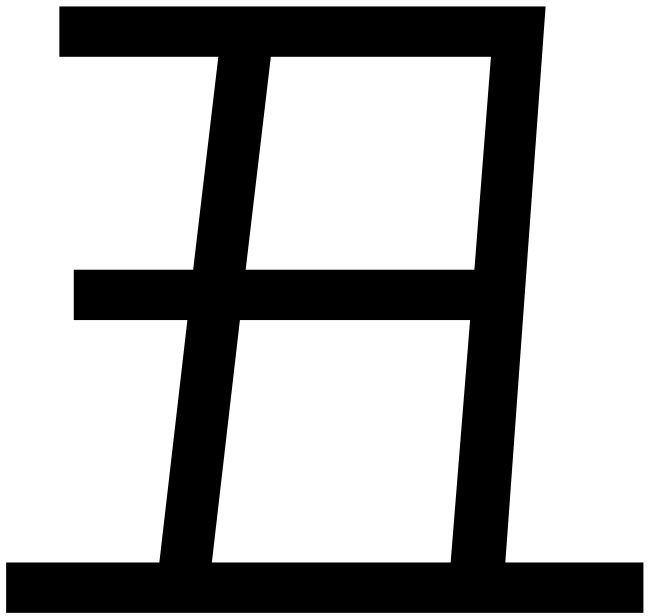
http://johnhax.net/2017/js-private/slide?qcon#102

第一反应?

http://johnhax.net/2017/js-private/slide?qcon#103

WTF!

http://johnhax.net/2017/js-private/slide?qcon#104



http://johnhax.net/2017/js-private/slide?qcon#105

不要急，

继续看：

http://johnhax.net/2017/js-private/slide?qcon#106

```
class Foo {
  //instance members
  own x=0, y=0  // two data properties
  own #secret   // a private field
                // initial value undefined
  own *[Symbol.iterator](){yield this.#secret}
                // a generator method
  own #callback(){}  //a private instance method
  //class constructor members
  static #p=new Set(), q=Foo.#p
               // a private field and a property
               // of the class constructor
  static get p(){return Foo.#p} //accessor method
  //prototype methods
  setCallback(f){this.#callback=f}
  constructor(s){
     this.#secret = s
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#107

丑爆了

http://johnhax.net/2017/js-private/slide?qcon#108

Why not use the "private" keyword, like Java or C#?

针对“#”产生大量的

消极反应

http://johnhax.net/2017/js-private/slide?qcon#110

当然也有不少提议

http://johnhax.net/2017/js-private/slide?qcon#111

不过绝大多数都是

行不通的

http://johnhax.net/2017/js-private/slide?qcon#112

语言演化

http://johnhax.net/2017/js-private/slide?qcon#113

http://johnhax.net/2017/js-private/slide?qcon#114

- 新 API
- 增加语法糖
- 新特性（语法+语义）
- 改变语义
- 改变语法
- 删除特性

http://johnhax.net/2017/js-private/slide?qcon#115

例子……

http://johnhax.net/2017/js-private/slide?qcon#116

回到 private 问题

http://johnhax.net/2017/js-private/slide?qcon#117

分成两部分

http://johnhax.net/2017/js-private/slide?qcon#118

语法 / 语义

http://johnhax.net/2017/js-private/slide?qcon#119

语法问题

sigil

http://johnhax.net/2017/js-private/slide?qcon#120

难道真的不行吗?

http://johnhax.net/2017/js-private/slide?qcon#121

PHP

http://johnhax.net/2017/js-private/slide?qcon#122

```php
class RGBColor {
  private $hex;
  function __construct($r, $g, $b) {
    $this->hex = $r * 0x10000 + $g * 0x100 + $b;
  }
  function __toString() {
    return "rgb({$this->red()}, {$this->green()}, {$this->blue()})";
  }
  function red()   { return $this->hex >> 16; }
  function green() { return ($this->hex >> 8) & 0xff; }
  function blue()  { return $this->hex & 0xff; }
  static function equals($c1, $c2) {
    return $c1->hex === $c2->hex;  //    Semantic?
  }
}
```

http://johnhax.net/2017/js-private/slide?qcon#123

PHP 可以的?

JS 为啥不行?

http://johnhax.net/2017/js-private/slide?qcon#124

因为 PHP 是最好的语言……

http://johnhax.net/2017/js-private/slide?qcon#125

因为：

http://johnhax.net/2017/js-private/slide?qcon#126

语义复杂

http://johnhax.net/2017/js-private/slide?qcon#127

prototype 可变

http://johnhax.net/2017/js-private/slide?qcon#128

压缩混淆

http://johnhax.net/2017/js-private/slide?qcon#129

属性访问
性能问题

http://johnhax.net/2017/js-private/slide?qcon#130

引擎实现

复杂性

http://johnhax.net/2017/js-private/slide?qcon#131

语法问题

@ vs

http://johnhax.net/2017/js-private/slide?qcon#132

语义问题

soft vs hard

http://johnhax.net/2017/js-private/slide?qcon#133

大部分语言的private机制
都是某种程度上soft的

http://johnhax.net/2017/js-private/slide?qcon#134

但可以使用 SecurityManager
之类的机制限制 refection

http://johnhax.net/2017/js-private/slide?qcon#135

Symbol 机制

http://johnhax.net/2017/js-private/slide?qcon#136

语法成本

http://johnhax.net/2017/js-private/slide?qcon#137

Private symbol?

http://johnhax.net/2017/js-private/slide?qcon#138

Proxy

动态代理

http://johnhax.net/2017/js-private/slide?qcon#139

膜

Membrane

http://johnhax.net/2017/js-private/slide?qcon#140

实现 host 接口

- 不可 hack
- 跨 realm

http://johnhax.net/2017/js-private/slide?qcon#141

协调问题

- public properties
- own/prototype/static
- decorators

http://johnhax.net/2017/js-private/slide?qcon#142

平衡

http://johnhax.net/2017/js-private/slide?qcon#143

- common use cases
- mental model
- performance
- backward compatibility
- forward compatibility

http://johnhax.net/2017/js-private/slide?qcon#144

FAQ

github: @hax

zhihu: 贺师俊

weibo: @johnhax

http://johnhax.net/2017/js-private/slide?qcon#145