进阶–补环境自吐 2

题目:携程补环境

纯文本

```
const hook = true, compress = true

delete process
delete global
delete require
delete module
delete Buffer
delete __dirname
delete __file__

const hook_funcs = ['toString', 'hasOwnProperty']
const constructor_excepts = [Date, RegExp]

const $toString = Function.toString;
const myFunction_toString_symbol = Symbol('('.concat('', ')_', (Math.random() + '').toStr
const myToString = function () {
    return typeof this == 'function' && this[myFunction_toString_symbol] || (result = $to
};

function set_native(func, key, value) {
    return Object.defineProperty(func, key, {
        "enumerable": false,
        "configurable": true,
        "writable": true,
        "value": value
    })
};
delete Function.prototype['toString'];
```

```
set_native(Function.prototype, "toString", myToString);
set_native(Function.prototype.toString, myFunction_toString_symbol, "function toString()

function func_set_natvie(func) {
    return set_native(func, myFunction_toString_symbol, `function ${myFunction_toString_s
}

let _origin_string_idx_of = String.prototype.indexOf
String.prototype.indexOf = function () {
    let result = _origin_string_idx_of.apply(this, arguments)
    console.log(`[string ${this}].indexOf`, arguments, result)
    return result
}
func_set_natvie(String.prototype.indexOf)

let _origin_regexp_test = RegExp.prototype.test
RegExp.prototype.test = function () {
    let result = _origin_regexp_test.apply(this, arguments)
    console.log(`[regexp ${this}].test`, arguments, result)
    return result
}
func_set_natvie(RegExp.prototype.test)

let _stringify_prototypes = []

function _stringify(e) {
    let ret = _stringify_prototypes.find(k => e instanceof k)
    return ret ? `[object ${ret.name}]` : e
}

function common_proxy(obj, opts = {}) {
    let {
        identifier,
        prototype,
        stringify,
        native
    } = opts
    if (native) {
        func_set_natvie(obj)
    }
    if (!hook) return obj
    let obj_type = typeof obj
    if (obj_type != "object" && obj_type != "function") return

    if (!identifier) {
        if (prototype) {
            identifier = prototype.name + '.prototype.' + obj.name
        } else {
            identifier = obj instanceof Function ? `[function ${obj.name}]` : obj.toStrin
        }
    }
    if (!identifier.startsWith('[')) {
```

```
        identifier = `[` + identifier + ']'
    }
    if (stringify) {
        _stringify_prototypes.push(obj)
    }
    return new Proxy(obj, {
        construct() {
            let result = Reflect.construct.apply(this, arguments);
            console.log(`${identifier}.new => `, arguments, result);
            return constructor_excepts.find(e => result instanceof e) ? result : common_p
        },
        apply() {
            try {
                let result = Reflect.apply.apply(this, arguments);
                console.log(`${identifier}.apply => `, _stringify(arguments[1]), argument
                return result
            } catch (e) {
                console.log(`${identifier}.apply => `, _stringify(arguments[1]), argument
                throw e
            }
        },
        // 代理这个对象的属性设置  a.b , a["b"]
        get: function () {
            let result = Reflect.get.apply(this, arguments)
            if (typeof arguments[1] == "string" && !arguments[1].startsWith("_")) {
                if (hook_funcs.includes(arguments[1])) {
                    result = common_proxy(result, {
                        identifier: identifier + `.` + arguments[1]
                    })
                }
                console.log(
                    `${identifier}.get => `, arguments[1], _stringify(result)
                )
            }
            return result
        },
        // 代理这个对象的属性设置  a.b = 1, a["b"]  1
        set: function () {
            let result = Reflect.set.apply(this, arguments)
            if (typeof arguments[1] == "string" && !arguments[1].startsWith("_")) {
                console.log(`${identifier}.set => `, arguments[1], _stringify(arguments[2
            }
            return result
        },
        // in 操作符的捕捉器  "xx" in a
        has: function () {
            let result = Reflect.has.apply(this, arguments)
            console.log(`${identifier}.has => `, arguments[1], result);
            return result
        },
        // Object.getOwnPropertyNames(a) 方法和 Object.getOwnPropertySymbols(a) 方法的捕捉器
        ownKeys: function () {
```

```
            let result = Reflect.ownKeys.apply(this, arguments)
            console.log(`${identifier}.ownKeys => `, result.length);
            return result
        },
        // delete a.xxx
        deleteProperty: function () {
            let result = Reflect.deleteProperty.apply(this, arguments)
            console.log(`${identifier}.delete => `, arguments, result);
            return result
        },
    })
```

0、首先补到能够返回值（参见上节课内容【进阶-补环境自吐1】）

1\require nodejs 是有的，浏览器是没有的，不要画蛇添足

2\window 的 Location 和它的实例 localtion 都要补

3\document 有三个子元素（document.childNodes[1].childNodes）分别是 head、body、还有个啥...

4\我们如何补 HTMLDocument？先看属性描述符

getOwnPropertyDescriptor(document.__proto__.__proto__,'body')

发现 body 在原型链上的 Document 上。于是按下图补充

JavaScript

```
class HTMLDocument{
  ...
  get body(){
    //document.body即调用这个函数
  }
  set body(){
  }
}
等价：
Object.defineProperty(......)
```

然后在浏览器上访问 body，发现 body.__proto__是一个 HTMLBodyElement,那我们构造它

JavaScript

```
class HTMLBodyElement{

}

class HTMLDocument{
  ...
  get body(){
    //document.body即调用这个函数
    return common_proxy(new HTMLBodyElement)
  }
  set body(){
```

```
}
```

5\发现 platform 的引用，这个值代表的是系统

```JavaScript
class Navigator{
   ...
   get platform(){
      return 'MacIntel'
   }

}
```

6\userAgent 来回进行了一次赋值，为了测试 userAgent 是否可以被修改。因为 userAgent 的属性描述符是没有 set 方法的，所以是不能被修改的

```JavaScript
//[object Navigator].get =>  userAgent undefined
//[object Navigator].set =>  userAgent ctrip.com
//[object Navigator].get =>  userAgent ctrip.com
//[object Navigator].set =>  userAgent undefined
//Object.getOwnPropertyDescriptor(Navigator.prototype,'userAgent')
//{set: undefined, enumerable: true, configurable: true, get: ƒ}
class Navigator{
   ...
   get platform(){
      return 'MacIntel'
   }
   get userAgent(){
     //这里写死了get，所以怎么set都无所谓
      return 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,
   }

}
```

7\createElement 返回的内置方法 ƒ createElement() { [native code] } 是无法看的，但我们先不管

```JavaScript
function createElement(){}
func_set_natvie(createElement) // 用func_set_natvie保护toString方法，防止被识别
document = common_proxy(createElement)
```

8\webdriver、callPhantom、callSelenium 都应该为 false 或者空

9\external、documentElement 等对象同理，先利用模版创建 class 对象，然后对应赋值即可

10\现在看，我们的结果正确很多了，但是还是有错误，网上看我们构建的空对象 HtmlElement 的 children 默认返回值

11\在对象里定义变量的时候，一定要用 let 声明，不要使用 var 或者不声明，否则 nodejs 有可能会报错