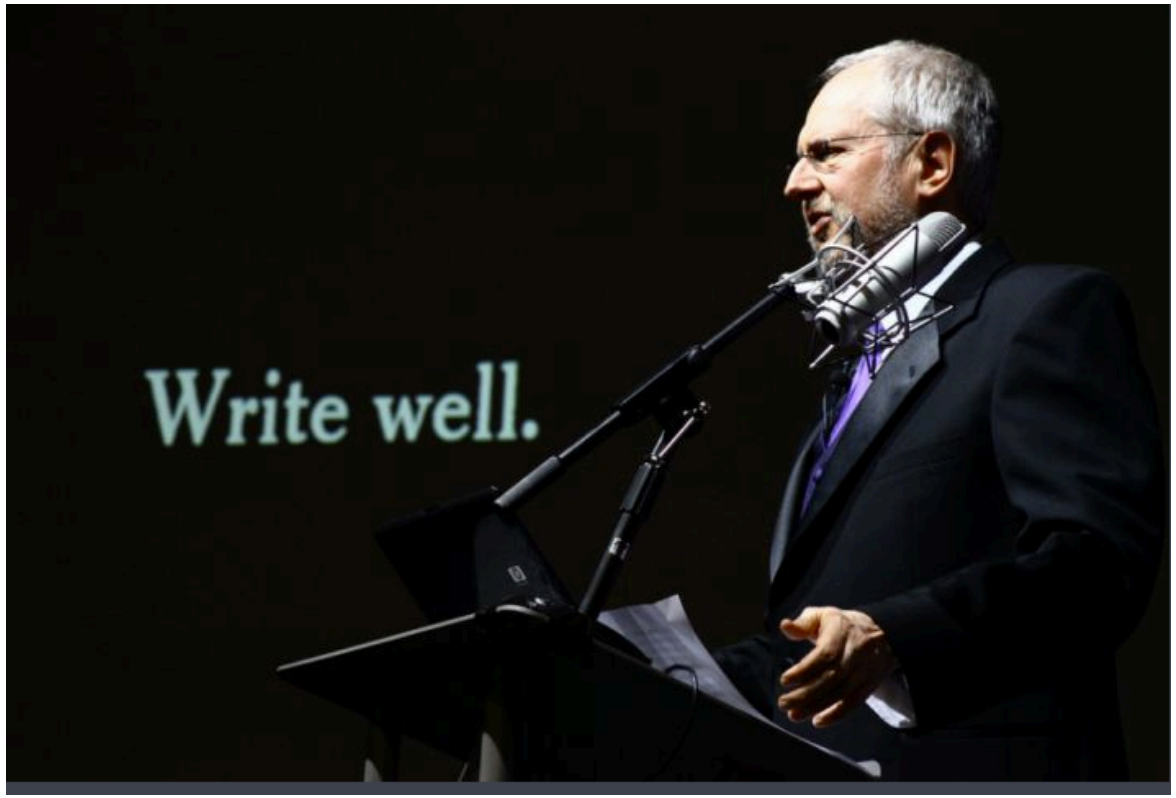
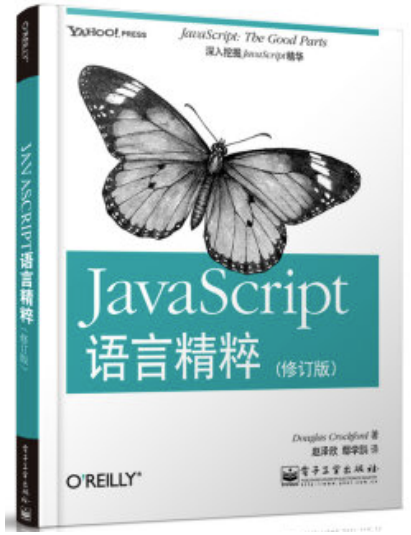
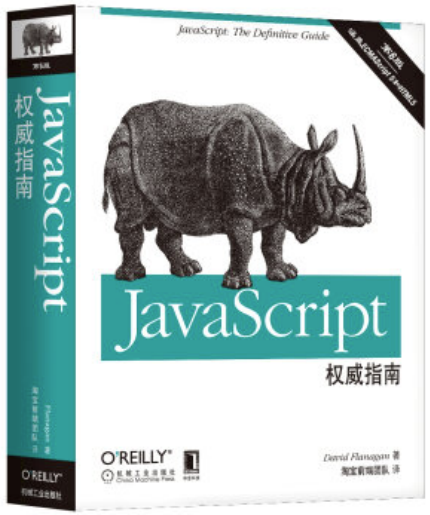
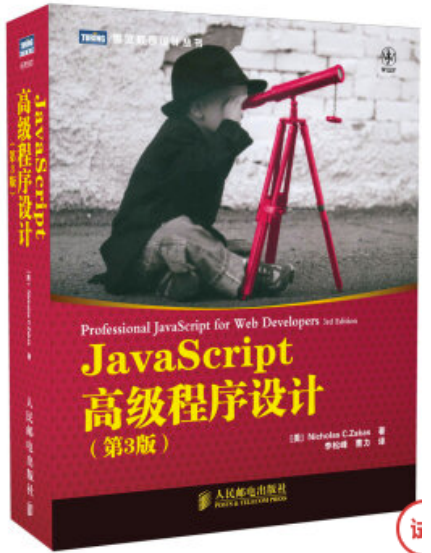
JS全栈开发

Douglas Crockford









全栈概况

- 前端：MVVM模式的数据驱动开发方式
- 后端：以nodejs为核心的生态系统
- 通信：resetful方式
- 设备：从传统浏览器到手机混合式开发
- 优化：从客户端，通信，服务端的全面性能提升
- 自动化：前后端的全面构建方式开发
- 开发流程：需求为中心的快速迭代方式

工具

- chrome
- node
- webstorm
- postman
- git
- mongodb

函数及闭包

函数

函数的定位

- 可以被调用的代码段（有名或匿名）
- 指引一批对象的“原型指针”，从而扮演“类”的角色
- 是对象，可以被传递（实参或返回）
- 第一公民

Different types of functions

- named function
- inner function
- recursive function
- anonymous function
- Immediately Invoked Function Expressions (IIFE)
- https://developer.mozilla.org/en-US/docs/Glossary/Function

As a Code Snippet

注意

- 无重载
- 形参无类型
- 形参与实参的个数问题
  - 函数内部的arguments,rest,this实参
  - Function.prototype的length,name
- apply,call,bind方式调用介绍
  - 与this有关（暂不介绍）

ES6 的Arrow Notation

- `(a,b,…)=>[{return …}|expression]`
- 注意`this`是在声明时被锁定
- 鼓励使用，对代码进行简化

Inner Function

- 由定义期产生的变量作用域问题
  - 内部函数可以直接引用外部函数的局部变量
- 闭包的产生

As a Class

注意：

- prototype引用
- 产生的对象，所具有的constructor引用
- 函数内的this引用

- 总结：new的函数调用，所产生的威力是什么？
  - 1. 产生了新的对象，并返回
  - 2. 新的对象在函数中被称为this
  - 3. 新的对象的construtor就是函数自身
  - 4. 新的对象的__proto__指向了函数的prototype

ES6的Class

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  toString() {
    return '(' + this.x + ', ' + this.y + ')';
  }
}
```

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

Point.prototype.toString = function () {
  return '(' + this.x + ', ' + this.y + ')';
};

var p = new Point(1, 2);
```

As a Object Instacne

回调方式的出现

- 函数做为对象本身，可以传递，代表着：算法可以被传递
  - sort函数分析
- 也代表着我们可以改变“同步”做事情的习惯
  - 写一个异步处理函数
  - 看一下node的函数库

闭包的产生

- 内部函数的变量作用域，导致其定义期对局部变量可见
- 函数作为对象：可以被外部函数返回，而在外部对内部函数的引用，就会形成闭包现象
- 脱离了外部函数的调用，闭包自身就会形成对局部变量的隔离封装，从而打造出“私有的，有状态的对象”

闭包的应用

意义何在

- 用函数编程的语法，打造出有状态的对象
- 实例：
  - 用构造函数和闭包两种不同的方式，打造一款有初值的计数器

案例：偏函数+高阶函数的组合

- 高阶概念：能够传入函数的函数
- 偏函数概念：根据传入参数，返回闭包的函数

```
function createComparisonFunction(propertyName){
    //对于"全局"，本函数是一个"闭包"
    return function(object1,object2){
        //对于"全局"，本匿名函数是一个"闭包"
        //对于"createComparisonFunction"，本匿名函数也是"闭包"
        var value1=object1[propertyName];
        var value2=object2[propertyName];
        if(value1<value2) return -1;
        else if(value1>value2) return 1;
        else return 0;
    }
}
```

块级作用域

```
function work(){
    (function(){
    var m=100;
    })();
    //即时函数执行后脱离作用域链，m不可见
    console.log(m);//m is not defined
}
work();
```

私有变量

- javascript没有私有成员概念

```
/*模拟私有成员的概念*/
function MyObject(){
    var privateVariable=10;
    var privateFunction=function(){return "john"};
    //特权方法（privileged method）
    this.publicMethod=function(){
        privateVariable++;//访问局部变量
        return privateFunction;//返回内部函数（闭包）
    }
}
var m=new MyObject();
m.publicMethod();//闭包的调用

/*模仿JavaBean对象*/
function Person(name,age){
    this.getAge()=function(){return age;};
    this.setAge(value)=function(){age=value};
    this.getName=function(){return name};
}
```

坑：

```
var result;
result=(function(){
    var fs=[5];
    for(var i=0;i<5;i++){
        fs[i]=function(){return i;};
    }
    return fs;
})();

for(var i=0;i<5;i++){
    console.log(result[i]());//5,5,5,5,5
}
```

```
var result;
result=(function(){
    var fs=[5];
    for(var i=0;i<5;i++){
        fs[i]=  (
                    function(num){return function(){return num;}}
                )(i);
    }
    return fs;
})();

for(var i=0;i<5;i++){
    console.log(result[i]());//0,1,2,3,4
}
```

神奇的this引用

还是看下的我的简书

- https://www.jianshu.com/p/35ec17c66863

```
//初始化
function work(){
    var init=function(){
        console.log("一些初始化动作.....");
    }
    init();
    var work=function(){
        console.log("实际工作......");
    }
    return work;
};
//用自身重写自身
work=work();//一些初始化动作.....
work();//实际工作......
```

```
//一次性函数
(function(){
    //内部所有声明都不会污染“global”
    var x;
    var m=function(){};
})();
```

函数借用

- “Function对象”有两个方法（非继承得到），apply和call
- 可以实现Function对象调用时“指定this”的作用。
- 上例可以下如下修改：

```
//"this"的变化
this.color="red";
function sayColor(){
    console.log(this.color);
};
//sayColor();//red
sayColor.apply(this);//red
var o={color:"green"};
//o.sayColor=sayColor;
//o.sayColor();//green

sayColor.apply(o /*形参数组*/);//指定this=o;
sayColor.call(o /*形参表*/);//指定this=o;
```

函数的内部属性（调用时产生）

- arguments，“类数组形式”的引用类型，表明传入的参数
- this,表示函数执行的“环境对象”
- arguments.callee,表示函数的“调用者”（注意：不同于this）

```
//参数个数可变的“加法器”
function sum(){
    var total=0;
    for(var i=0;i<arguments.length;i++){
        total+=arguments[i];
    }
    return total;
};
var rs=sum();
var rs1=sum(1,2,6);
console.log(rs+"......."+rs1);//0.....9
```

```
//"this"的变化
this.color="red";
function sayColor(){
    console.log(this.color);
};
sayColor();//red

var o={color:"green"};
o.sayColor=sayColor;
o.sayColor();//green
```

递归

- arguments.callee,指向当前函数的指针，在递归场合下，可以实现“函数名称与实现的解藕”
- 注意：“use strict”不能使用

```
//名称与实现"耦合"
var factorial=function(n){
    if(n<=1) return 1;
    else return n*factorial(n-1);
}
var f=factorial;
factorial=null;

var rs=f(3);//Exception
console.log(rs);
```

```
//解藕合
var factorial=function(n){
    if(n<=1) return 1;
    else return n*arguments.callee(n-1);
}
var f=factorial;
factorial=null;
var rs=f(3);
console.log(rs);
```

函数的“静态”属性（编译期确定）

- length,形参表长度
- prototype,原型“指针”
- apply&call,“反射”调用方法，可以调用时改变“this”
- bind,生成“反射”调用方法，使用时传入”this”

```
//"this"的变化
this.color="red";
function sayColor(){
    console.log(this.color);
};

var o={color:"green"};
//生成新的函数
var sayColor1=sayColor.bind(o);
sayColor1();
```

异步函数及Promise

```
function open(filename,callback) {
    console.log("javascript:\t正在系统调用, 传入: "+filename);
    setTimeout(function () {
        let fd=11;
        console.log("system:\t获取到fd:" + fd+ "...正在通知回调! ");
        callback(fd);
    },2000);
}
function write(fd,content,callback) {
    console.log("javascript:回调中..获取到 fd:"+fd+",正在进行系统调用,将: "+content+" 写入 ");
    setTimeout(function () {
        let len=100;
        console.log("system:写入完成了:" + len + "...正在通知回调! ");
        callback(len);
    },3000);
}


open("a.txt",function (fd) {
    write(fd,"abc",function (len) {
        console.log("javascript:获取了: "+len+"字节");
    })
})

```

Promise

```
29  function openWithPromise(filename) {
30      console.log("javascript:\t正在系统调用，传入："+filename);
31      let p=new Promise(function (resolve,reject) {
32          setTimeout(function () {
33              let random=Math.random();
34              if(random>0.3) {
35                  let fd = 11;
36                  console.log("system:\t获取到fd:" + fd+ "...正在通知回调！");
37                  resolve(fd);
38              }
39              else{
40                  reject("读入文件出现异常！");
41              }
42          },2000);
43      });
44      return p;
45  }
```

Promise

```
47  function writeWithPromise(fd,content) {
48      console.log("javascript:回调中..获取到 fd:"+fd+",正在进行系统调用,将: "+content+" 写入 ");
49      let p=new Promise((resolve ,reject)=>{
50          setTimeout(function () {
51              let random=Math.random();
52              if(random>0.4) {
53                  let len = 100;
54                  console.log("system:写入完成了:" + len + "...正在通知回调! ");
55                  resolve(len);
56              }
57              else{
58                  reject("写入文件时出现异常");
59              }
60          },3000);
61      });
62      return p;
63  }
```

Promise调用

```
openWithPromise("abc.txt")
    .then(function (fd) {
        return writeWithPromise(fd,"abc");
    }).catch(function (errInfo) {
        console.log(errInfo);
    }).then(function (len) {
        console.log("回调中得到:"+len+"字节");
    }).catch(function (errInfo) {
        console.log(errInfo);
    })

```