

JS is JS

编写JavaScript独有风格的高质量代码

About

- * My influences were awk, C, HyperTalk, and Self, combined with management orders to "make it look like Java."

—Brendan Eich

Agenda

- * 抽象
 - * 面向对象
 - * 函数式
 - * 过程式
- 

The background of the slide features a traditional Chinese landscape painting, likely a 'Shan Shui' (mountain-water) genre, rendered in a style reminiscent of a fan painting. The scene depicts a misty, mountainous landscape with a winding river or path leading through the valleys. The painting is divided into several vertical sections, suggesting it might be a collection of smaller scenes or a single scene viewed from different angles. The overall color palette is muted, with soft greens, greys, and earthy tones. A thin, horizontal gold line runs across the middle of the slide, passing behind the title text.

抽象

抽象的方法

1. 简化

- * 将复杂物体的一个或几个特性抽出去，而只注意其他特性的行动或过程

2. 归纳

- * 将几个有区别的物体的共同性质或特性，形象地抽取出来或孤立地进行考虑的行动或过程。

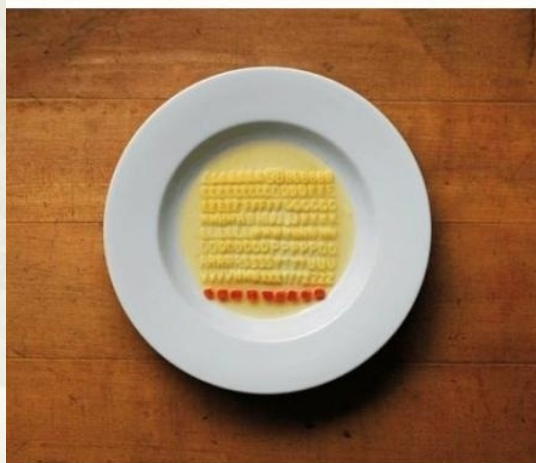
抽象的方法——简化

毛主席教导我们：“千万不要忘记阶级斗争。”林副主席说：“不懂得什么是阶级，不懂得什么是剥削，就不懂得革命。不弄清过去的苦，就不知道今天的甜，还会把今天的甜也误认为是苦。”

解放前，贫农张大伯被迫向地主“钱剥皮”借钱3元，被勒索“月三分”（就是每一个月的利息是上月所欠钱的30%）的利息，张大伯在10个月后才还清债务，这笔债务是多少钱？张大伯被地主剥削了多少元利息？

```
function calculate() {  
    return 3*Math.pow(1.3, 10)-3;  
}
```

抽象的方法——归纳



同一事物的抽象可能不同



对问题抽象

- * 教师分苹果问题：

- * 有一个老师分苹果给3个小朋友，每个小朋友分3个苹果，请问小朋友们一共有多少个苹果？

```
function totleApples(kidsCount,appleCount) {  
    return kidsCount*appleCount;  
}
```

对问题错误的抽象

*

```
function totalApples(count) {  
  return count*count;  
}
```

过于具体的抽象

```
function Apple() {  
}  
function Kid() {  
  this.apples = [];  
  this.receiveApple = function (apple) {  
    apples.push(apple);  
  }  
}  
function Teacher() {  
  this.dispatchApple = function(kid) {  
    kid.receiveApple(new Apple());  
    kid.receiveApple(new Apple());  
    kid.receiveApple(new Apple());  
  }  
}  
  
function totalApples() {  
  var kids = [new Kid(), new Kid(), new Kid()];  
  var teacher = new Teacher();  
  kids.forEach(function(kid) {
```



面向过程抽象

过程式

- * 过程是一种最常见的抽象
- * 过程式不是一种落后的编程范式
- * 程序 = 数据 + 过程

过程式

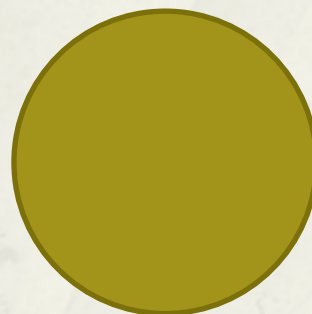
```
var data1,data2,data3;  
function process_main() {  
    data1 = .....  
    process1();  
    process2();  
    function process1(){  
        data2 = .....  
    }  
    function process2(){  
        data3 = .....  
    }  
}
```

The background features a large, semi-circular fan shape. Inside the fan's segments is a traditional Chinese ink wash landscape painting, depicting mountains, trees, and a winding path. A thin horizontal line crosses the center of the fan.

面向对象抽象

面向对象

- * 对象是一个朴素的概念，大约在人2-3岁时产生



描述每一个对象

- * 长方形

- * 四条边、四个角都是直角
- * 有长、宽



- * 平行四边形

- * 四条边、对边平行
- * 有两个边长、倾斜角



- * 圆形

- * 没有边，所有点到原点距离相等
- * 有半径



- * 三角形

- * 三条边
- * 有三条边长

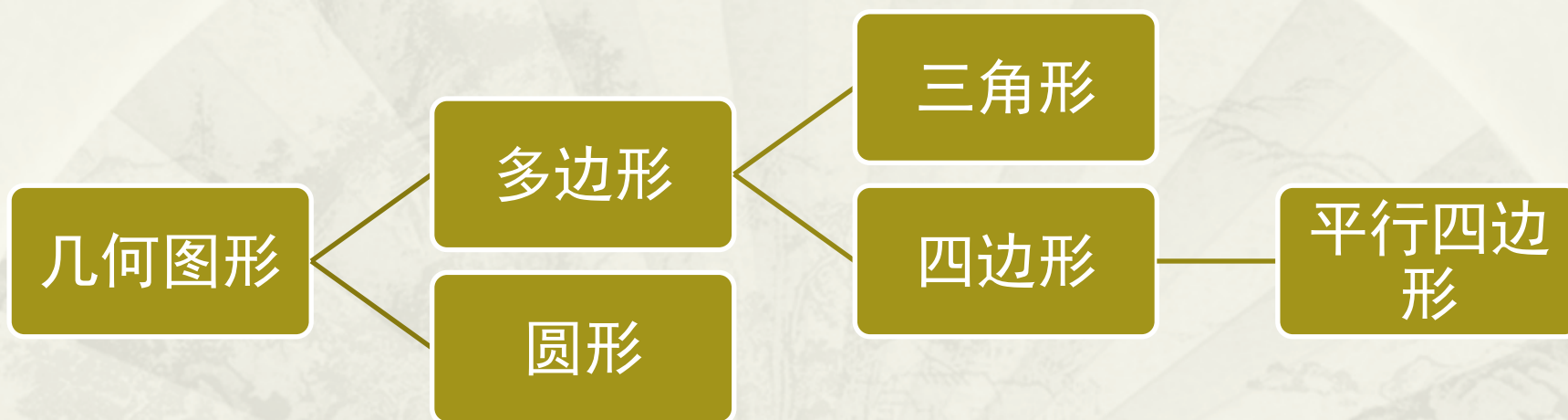


JS中描述独立对象

```
var object = {  
  width:100,  
  height:200,  
};
```



分类描述对象



JS中分类描述对象

```
function Parent()  
{  
}
```

```
function Child(x)  
{  
    Parent.call(this);  
    this.x = x;  
}
```

面向对象——原型



原型方式描述对象



JS中原型方式描述对象

```
function Cat()  
{  
}
```

```
function Tiger()  
{  
  this.draw(“王”)  
}
```

```
Tiger.prototype = new Cat;
```



函数式抽象

Lambda演算

* λ 演算

- * 其实就是替换
- * $\lambda x.x + 2$ 表示把 x 换成 $x+2$
- * $\lambda x.(\lambda x.x + 2)$ 表示把 x 换成 $\lambda x.x + 2$
- * 函数式编程以 λ 演算为理论基础

函数是“第一型”

- * 能做参数
- * 能作为返回值
- * 能赋值给变量
- * 能被存储到数据结构中
- * 有直接量
- * 能运行时产生

函数式

```
function add(a, b) {  
    return a + b;  
}
```

```
function mul(a,b,add) {  
    var r = a;  
    for(var i = 1; i<b; i++) {  
        add(r,a);  
    }  
}
```

闭包(Closure)

* 闭包(Closure) —Lexical Closure

```
var a, b;  
function dosth() {  
    return a+b;  
}
```

```
function createClosure() {  
    var a, b;  
    function dosth() {  
        return a+b;  
    }  
}
```

函数式

* Currying(柯里化)

`f(a,b,c);`

`f(a); //抛出异常?`

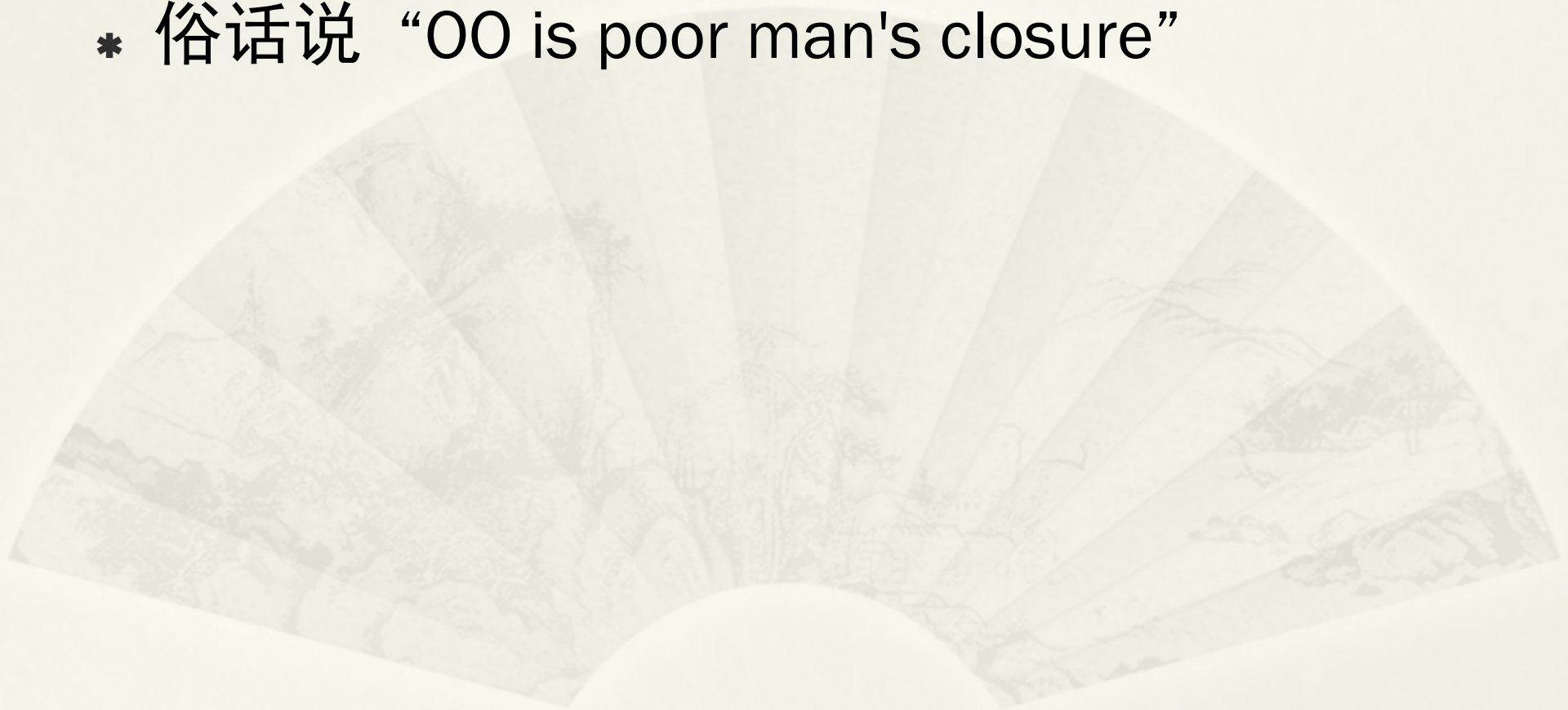
`f(a,b,c);`

`f2 = f(a); //f2接受剩余参数`

`f2(b,c);`

设计函数式API

- * 俗话说 “OO is poor man's closure”



```
function Node() {  
    this. addEventListener = function(type,listener){...};  
    this. removeEventListener = function(type,listener)  
    {...};  
}  
var node = new Node();  
node. addEventListener (type,listener);  
node. addEventListener (type,listener);
```

```
function addEventListener = function(node,type,listener){...};  
function removeEventListener = function(node,type,listener){...};  
var node = new Node();  
var addEventListenerToNode=addEventListener(node );  
addEventListenerToNode(type,listener)  
addEventListenerToNode(type,listener)
```

理解函数式范式

- * 函数式是一种只关注输入输出的抽象风格 
- * 函数式就是用函数编程 

JS中应用函数式

- * 在设计中引入函数式特性
- * 纯粹函数式编程



创建独有风格

* JQuery

- * ~~面向对象~~

- * ~~函数式~~

- * ~~过程式~~

- * 链式表达 ✓

- * 声明式编程 ✓

脱离语言束缚

这是对象

这不是对象，这是命名空间

```
var connection = DB.connect(  
{  
  host: ".....",  
  port: ".....",  
  name: "....."  
})
```

这也不是对象，这是数据

灵活运用语言特性

- * Closure
- * Object
- * Function
- * Eval
- * Prototype
- * Arguments
- * Property

Q&A

- * 网名： winter
- * 真名： 程劭非
- * csf178@163.com
- * weibo.com/wintercn @寒冬winter



北京站 · 2012年4月18~20日
www.qconbeijing.com (11月启动)

QCon杭州站官网和资料
www.qconhangzhou.com

全球企业开发大会

INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE