Demo

js面向对象编程

对象概念

1.万物皆对象（{}(..基本定义的对象,Array,Function等都为对象）

vs其他面向对象语言

大部分面向对象编程语言都有类概念

js无类的概念，但有原型(prototype)，js通过原型实现面向对象编程

原型

```
// 例子一
var $eauty = {
    name: '美女',
    leg_length: 1.2,
    shop: function(){
        alert(this.name + ' is shopping');
    }
};
var nemo = {
    name: '尼莫'
};
// 将nemo的原型指向对象$eauty便可拥有其所有属性
nemo.__proto__ = $eauty;
alert(nemo.name);       // 尼莫
alert(nemo.leg_length); // 1.2
nemo.shop();            // 尼莫 is shopping!
```

原型链

概念：当用obj.xxx访问对象属性时，js引擎会先在obj这个对象上查找属性，如果没找到，就沿其原型对象上找，如果还没找到，就一直上溯到Object.prototype对象，最后如果还没找到，就只能返回undefined。这样的属性链条叫做原型链。

```
// 例子二
// 给Object.prototype加eat属性，由此nemo和$eauty对象都具有eat属性
Object.prototype.eat = function(){
    alert(this.name + ' is eating');
};
nemo.eat();
$eauty.eat();
// 以上的原型链为：
// nemo -> $eauty -> Object.prototype -> null
// nemo -> Object.prototype -> null
```

内置对象原型链

```
// 创建一个Array对象
var arr = [1,2,3];
// 其原型链为： arr -> Array.prototype -> Object.prototype -> null
alert(arr.join('|'));  // '1|2|3'
alert(arr.concat(4));  // [1,2,3,4]
// arr可以直接使用join、concat等方式，是因为Array对象的prototype定义了这些方法
// 如果不确定某个内置对象有那些属性，也可以直接打印其原型查看
console.log(Array.prototype.concat);


function foo(){
    return 'foo'
}
// 其原型链为： arr -> Function.prototype -> Object.prototype -> null
// 由于Function.prototype定义了apply()、call()方法，因此，所有函数都可以调用这两个方法。


// 为Function.prototype添加属性，所有的函数都具有了alert方法
Function.prototype.alert = function(str){
    alert(this) + ' is the instance of Function');
};
foo.alert(); // foo is the instance of Function
```

构造函数也是函数

```
function Func(){
    alert('构造器也是函数，属于Function类')
}
var func = new Func(); // 构造器也是函数，属于Function类

// func为func的构造器
alert(func.constructor === Func); // true
// func为Func的实例
alert(func instanceof Func);    // true

// 构造器Func的构造器为Function,Function为所有函数的构造器
alert(Func.constructor === Function); // true
// 构造器Func为类Function的实例
alert(Func instanceof Function);   // true
```

用法

```
// 例子一：
// 创建Coder构造器
function Coder(name){
    this.name = name;
    this.code = function(){
        alert(this.name + ' is busy with coding');
    }
}

// 实例化Coder
var xiaoyun = new Coder('小云');
var xiaoshan = new Coder('小珊');
// 原型链：xiaoyun 与 xiaoshan -> Coder.prototype -> Object.prototype -> null
// xiaoyun.constructor === xiaoshan.constructor ===
Coder.prototype.constructor === Coder
alert(xiaoyun.name);  // 小云
xiaoyun.code();       // 小云 is busy with coding!
alert(xiaoshan.name); // 小珊
xiaoshan.code();      // 小珊 is busy with coding!
```

```
// 例子一中，xiaoyun和xiaoshan的code方法看上去完全相同，但
console.log(xiaoyun.code === xiaoshan.code); // false
// 说明xiaoyun和xiaoshan指向两个不同的code地址，如果此处有1000个实例，就要占用1000个地址，这样无疑消耗了内存
// 用原型化解：
Coder.prototype.code = function(){
    alert(this.name + ' is busy with coding');
};
// 这样xiaoyun和xiaoshan指向同一个地址的code,节约了内存
console.log(xiaoyun.code === xiaoshan.code); // true
```

构造函数

方式一：子类原型指向父类实例

```
// 创建Youth构造器
function Youth(name){
    this.interest = ['singing','coding'];
}
// 给Youth添加sing方法
Youth.prototype.sing = function(){
    alert(this.name + ' is singing a beautiful song!');
};

// 创建Coder构造器
function Coder(name){
    this.name = name;  // 此处不可省略
}
// Coder原型指向Youth实例
Coder.prototype = new Youth();
Coder.prototype.code = function(){
    alert(this.name + ' is busy with coding');
};
// 实例化Coder
var xiaoyun = new Coder('小云');
xiaoyun.sing(); // 小云 is singing a beautiful song!
xiaoyun.code(); // 小云 is busy with coding!

// 给xiaoyun动态添加一个兴趣
xiaoyun.interest.push('basketball');
console.log(xiaoyun.interest);  // ["singing", "coding", "basketball"]
// 实例化小珊
var xiaoshan = new Coder('小珊');
console.log(xiaoshan.interest); // ["singing", "coding", "basketball"]
// 咦，小珊怎么也有basketball的兴趣？
```

方式二：借用构造函数

```
// 创建Youth构造器
function Youth(name){
    this.name = name;
    this.interest = ['singing','coding'];
}
// 给Youth添加sing方法
Youth.prototype.sing = function(){
    alert(this.name + ' is singing a beautiful song!');
};

// 创建Coder构造器
function Coder(name){
    Youth.call(this,name); // 此处省略this.name = name
}
// 创建xiaoyun实例
xiaoyun = new Coder('小云');
// 创建xiaohei实例
xiaoshan = new Coder('小珊');
// 给xiaoyun加上篮球的兴趣
xiaoyun.interest.push('basketball');
// 给xiaoshan加上跳舞的兴趣
xiaoshan.interest.push('dancing');
console.log(xiaoyun.interest);  // ["singing", "coding", "basketball"]
console.log(xiaoshan.interest); // ["singing", "coding", "dancing"]

// 从以上打印的结果看到xiaoyun和xiaoshan的interest属性互不干扰，But:
console.log(xiaoyun.sing);  // undefined
console.log(xiaoyun.sing);  // undefined
// 咦，为啥xiaoyun和xiaoshan都没有sing方法了？
```

继承

方式三：组合继承

```
// 创建Youth构造器
function Youth(name){
    this.name = name;
    this.interest = ['singing','coding'];
}
// 给Youth添加sing方法
Youth.prototype.sing = function(){
    alert(this.name + ' is singing a beautiful song!');
};

// 创建Coder构造器
function Coder(name){
    // 继承父类的基本属性和引用属性并具有传参功能
    Youth.call(this,name);
}
// Coder原型指向Youth实例
Coder.prototype = new Youth();
Coder.prototype.code = function(){
    alert(this.name + ' is busy with coding');
};
// 实例化小云
var xiaoyun = new Coder('小云');
xiaoyun.sing(); // 小云 is singing a beautiful song!
xiaoyun.code(); // 小云 is busy with coding!
// 实例化小珊
var xiaoshan = new Coder('小珊');
xiaoshan.sing(); // 小珊 is singing a beautiful song!
xiaoshan.code(); // 小珊 is busy with coding!

// 给xiaoyun动态添加一个兴趣
xiaoyun.interest.push('basketball');
console.log(xiaoyun.interest);  // ["singing", "coding", "basketball"]
// 给xiaoshan动态添加一个兴趣
xiaoshan.interest.push('dancing');
console.log(xiaoshan.interest); // ["singing", "coding", "dancing"]

// 从以上打印的结果看到xiaoyun和xiaoshan公用同一个sing方法和code方法，达到节约内存和方法复用目的
console.log(xiaoyun.sing === xiaoshan.sing); // true
console.log(xiaoyun.code === xiaoshan.code); // true
```

方式四：class继承（ES6）

class

```
// 创建Coder类
class Coder {
    // 构造函数
    constructor(name){
        this.name = name;
    }
    // code是定义在原型上的函数
    code() {
        alert(this.name + ' is busy with coding');
    }
}
// 实例化小云
var xiaoyun = new Coder('小云');
// 实例化小珊
var xiaoshan = new Coder('小珊');
xiaoyun.code();   // 小云 is busy with coding!
xiaoshan.code();  // 小珊 is busy with coding!
// xiaoyun和xiaoshan指向同一个方法
console.log(xiaoyun.code === xiaoshan.code); // true
```

class继承

```
// 创建Youth类
class Youth {
    constructor(name,interest){
        this.name = name;
        this.interest = ['singing','coding'];
    }
    // 创建定义在原型上的sing方法
    sing() {
        alert(this.name + ' is singing a beautiful song!');
    }
}

// 创建Coder类并继承自Youth
class Coder extends Youth {
    // 构造函数
    constructor(name,interest){
        // super用于调用父类的构造方法
        super(name,interest);
    }
    // 创建定义在原型上的code方法
    code() {
        alert(this.name + ' is busy with coding');
    }
}

// 实例化Coder
var xiaoyun = new Coder('小云');
xiaoyun.sing();  // 小云 is singing a beautiful song!
xiaoyun.code();  // 小云 is busy with coding!
// 实例化小珊
var xiaoshan = new Coder('小珊');
xiaoshan.sing();  // 小珊 is singing a beautiful song!
xiaoshan.code();  // 小珊 is busy with coding!

// 给xiaoyun动态添加一个兴趣
xiaoyun.interest.push('basketball');
console.log(xiaoyun.interest);  // ["singing", "coding", "basketball"]
// 给xiaoshan动态添加一个兴趣
xiaoshan.interest.push('dancing');
console.log(xiaoshan.interest); // ["singing", "coding", "dancing"]

// 从以上打印的结果看到xiaoyun和xiaoshan公用同一个sing方法和code方法，达到节约内存和方法复用目的
console.log(xiaoyun.sing === xiaoshan.sing); // true
console.log(xiaoyun.code === xiaoshan.code); // true
```