# 二 面向对象补充

**# 面向对象铺垫-对象创建方式总结**

**## 第一种：创建对象的方式，json方式**

```

//推荐使用的场合： 作为函数的参数，临时只用一次的场景。比如设置函数原型对象。

var obj = {};

//对象有自己的 属性 和 行为

// 属性比如： 年龄、姓名、性别

// 行为： 吃饭、睡觉、走路、讲课等... 动作

var obj2 = {

name: 'laoma',

age: 18,

sayHi: function(){

console.log( name + 'say hi' );

}

};

//添加其他属性：

obj2.newProp = 123;// js的动态特性，如果没有要访问的属性，直接添加属性。

// 缺点： 不能作为对象创建的模板，也就是不能用new进行构造新对象。

//=================================================

```

**## 第二种： 创建面向对象的方式： new Object()的方式。 不推荐使用。**

```

var obj3 = new Object();

//添加属性

obj3.name = 'kitty';

obj3.sayHi = function() {

console.log( name + ' ' + 'say hi');

};

//跟上面的方式一样，只能临时用一下这个对象，不想作为new的构造模板是可以的。

```

**## 第三种： 构造函数构造对象方法**

```

//把 一个函数对象 当做构造函数来使用，一般要把 函数对象的首字母大写

function Persion() {

this.name = '123'; // 通过this可以直接给 构造出来的对象添加属性。

this.sayHi = function() {

console.log( this.name );

};

}

var p = new Persion();

//\*\* new 运算符的作用：

// 第一步：

// 执行构造函数（new后面的那个函数），在构

//造函数内部创建一个空对象，

// 第二步： 把上面的空对象跟构造函数的原型对象进行关联。

// 第三步：然后把this 指向当前空对象

//在构造函数执行结束后，把空对象返回 给 p

console.log( p.name ); //p.name 从构造函数里面创建的。

p.sayHi(); //在此方法内部的 this执行 p对象。

//绘制原型 和 构造函数 和对象之间的关系。

```

**## 第三种方式的升级改造版本：**

```

//第三种方式有个缺点： 对象的内部的函数会在每个对象中都存一份

//如果创建的对象非常多的话，那么非常浪费内存。函数的行为是所有对象

//可以共有，不需要每个对象都保存一份。所以，可以把函数放到原型中

//进行声明，那么所有对象都有了公共的函数，而且内存中只保留一份。

//所有的属性写到对象的内部

//第三种的升级版：

function Sprite() {

this.name = '123';

this.age = 19;

}

Sprite.prototype = {

sayHi: function() {

},

init: function() {

}

};

//继续升级： 把属性的设置做成参数化：

function Sprite( sname, sage ) {

this.name = sname || '';

this.age = sage || 18;

}

Sprite.prototype = {

sayHi: function() {

},

init: function() {

}

};

//问题： 1、调用者如果传递参数的顺序发生变化，那么废了

//问题： 2、 参数增减都会导致函数声明变化，调用的地方也可能发生变化。

//如何解决：继续升级

function Sprite( option ) { //我用一个对象把所有参数覆盖。灵活性就很强了。

//顺序无所谓，添加参数也无所谓了。

this.name = option.sname || '';

this.age = option.sage || 18;

}

Sprite.prototype = {

sayHi: function() {

},

init: function() {

}

};

//继续优化，把初始化的代码 放到init函数中

function Sprite( option ) { //我用一个对象把所有参数覆盖。灵活性就很强了。

//顺序无所谓，添加参数也无所谓了。

this.init( option );

}

Sprite.prototype = {

sayHi: function() {

},

init: function( option ) {

this.name = option.sname || '';

this.age = option.sage || 18;

}

};

//以后的canvas课程的开发暂时先用这种的方式

```

-------------------

**## 补充js方法调用的四种模式**

- 方法调用模式

```

function Persion() {

var name1 = "itcast",

age1 = 19,

show1 = function() {

console.log(this.name);

};

return {

age : age1,

name : name1,

show : show1

};

}

var p = new Persion();

p.show(); //在show方法中的this指向了p对象。

```

- 函数调用模式

```

function add( a, b) {

this.result = a + b;

}

add( 3, 9 ); //此方法执行的时候，this指向了window

console.log(result);

```

- 构造器调用模式

```

function Persion（）{

this.name = "123";

this.age = 19;

this.show = function(){

console.log(this.name);

};

}

var p = new Persion();

p.show();// 在show方法中方法this，指向了p对象实例。

```

- call 和 apply调用模式

```

function add(a,b){

this.result = a + b;s

}

var p = {}; //定义一个空对象。

add.call(p,3,4); //在这个方法调用的时候，this指向了p

console.log(p.result);

//apply和call是一样的用法，只不过apply第二个参数用数组进行传递。

```

### 1.1

### 1.2

### 1.3

## 2

### 2.1

### 2.2

### 2.3

### 2.4

### 2.5

## 3

### 3.1

### 3.2