

JS对象模型

JavaScript 是一种基于原型（Prototype）的面向对象语言，而不是基于类的面向对象语言。它基于原型、原型链来实现面向对象。

C++、Java有类Class和实例Instance的概念，类是一类事物的抽象，而实例则是类的实体。

定义类

字面式声明方式

```
1  var obj = { property_1:  value_1,    // property_# 可以是一个标识符...
2                      2:      value_2,    // 或一个数字...
3                      ["property" +3]: value_3, // 或一个可计算的key名...
4                      // ...,
5                      "property n": value_n }; // 或一个字符串
6
7  let a = 1;
8  let b = 'xyz';
9  let c = [1,2,3];
10 let d = x => x + 1;
11
12 var obj = {
13   'a':a,    // 引号不省略明确使用该字符串为属性名
14   b:b,      // 引号可以省，但依然转换为字符串作为属性名
15   [b]:100,  // 计算b的值然后在转换为字符串作为属性名
16   1:200,    // 将1转换为字符串
17   c,        // c作为属性名，值为数组[1,2,3]
18   d         // d作为属性名，值为函数
19 }
20
21 console.log(obj);
22 console.log(obj.d(1000))
23
24 for (let k in obj) {
25   console.log(typeof k, k, obj[k])
26 }
```

这种方法也称作字面值创建对象。Js 1.2开始支持。

对象的键key只能是字符串类型，最后都会被转换成字符串。

ES6之前——构造器

- 1、定义一个函数（构造器）对象，函数名首字母大写
- 2、使用this定义属性
- 3、使用new和构造器创建一个新对象

```
1  // 定义类，构造器
2  function Point(x, y) {
3    this.x = x;
4    this.y = y;
5    this.show = () => {console.log(this,this.x,this.y)};
```

```

6     console.log('Point~~~~~');
7 }
8
9 console.log(Point);
10 var p1 = new Point(4, 5); // 千万记得new
11 console.log(p1);
12 console.log('-----');
13
14 // 继承
15 function Point3D(x,y,z) {
16     Point.call(this,x,y); // "继承"
17     this.z = z;
18     console.log('Point3D~~~~~');
19 }
20
21 console.log(Point3D);
22 var p2 = new Point3D(14,15,16);
23 console.log(p2);
24 p2.show();

```

new 构建一个新的通用对象，new操作符会将新对象的this值传递给Point3D构造器函数，函数为这个对象创建z属性。

从上句话知道，new后得到一个对象，使用这个对象的this来调用构造器，那么如何执行“基类”的构造器方法呢？使用Point3D对象的this来执行Point的构造器，所以使用call方法，传入子类的this。最终，构造完成后，将对象赋给p2。

注意：如果不使用new关键字，就是一次普通的函数调用，this不代表实例。

ES6中的class

从ES6开始，新提供了class关键字，使得创建对象更加简单、清晰。

1. 类定义使用class关键字。创建的本质还是函数，是一个特殊的函数
2. 一个类只能拥有一个名为constructor的构造器方法。如果没有显式的定义一个构造方法，则会添加一个默认的constructor方法。
3. 继承使用extends关键字
4. 一个构造器可以使用super关键字来调用一个父类的构造函数
5. 类没有私有属性

```

1 // 基类定义
2 class Point {
3     constructor(x,y) /*构造器*/ {
4         this.x = x;
5         this.y = y;
6     }
7     show() /*方法*/ {
8         console.log(this,this.x,this.y);
9     }
10 }
11
12 let p1 = new Point(10,11)
13 p1.show()
14
15 // 继承
16 class Point3D extends Point {
17     constructor (x,y,z) {

```

```

18         super(x,y);
19         this.z = z;
20     }
21 }
22
23 let p2 = new Point3D(20,21,22);
24 p2.show()
25

```

重写方法

子类Point3D的show方法，需要重写

```

1  // 基类定义
2  class Point {
3      constructor(x,y) /*构造器*/ {
4          this.x = x;
5          this.y = y;
6      }
7      show() /*方法*/ {
8          console.log(this,this.x,this.y);
9      }
10 }
11
12 let p1 = new Point(10,11)
13 p1.show()
14
15 // 继承
16 class Point3D extends Point {
17     constructor (x,y,z) {
18         super(x,y);
19         this.z = z;
20     }
21
22     show(){ // 重写
23         console.log(this,this.x,this.y, this.z);
24     }
25 }
26
27 let p2 = new Point3D(20,21,22);
28 p2.show();

```

子类中直接重写父类的方法即可。

如果需要使用父类的方法，使用super.method()的方式调用。

使用箭头函数重写上面的方法

```

1  // 基类定义
2  // 基类定义
3  class Point {
4      constructor(x,y) /*构造器*/ {
5          this.x = x;
6          this.y = y;
7          //this.show = function () {console.log(this,this.x,this.y)};
8          this.show = () => console.log('Point');
9      }
10 }

```

```

11
12 // 继承
13 class Point3D extends Point {
14     constructor (x,y,z) {
15         super(x,y);
16         this.z = z;
17         this.show = () => console.log('Point3D');
18     }
19 }
20
21 let p2 = new Point3D(20,21,22);
22 p2.show(); // Point3D

```

从运行结果来看，箭头函数也支持子类的覆盖

```

1 // 基类定义
2 class Point {
3     constructor(x,y) /*构造器*/ {
4         this.x = x;
5         this.y = y;
6         this.show = () => console.log('Point');
7     }
8     // show() /*方法*/ {
9         // console.log(this,this.x,this.y);
10    // }
11 }
12
13 // 继承
14 class Point3D extends Point {
15     constructor (x,y,z) {
16         super(x,y);
17         this.z = z;
18         //this.show = () => console.log('Point3D');
19     }
20
21     show(){ // 重写
22         console.log('Point3D');
23     }
24 }
25
26 let p2 = new Point3D(20,21,22);
27 p2.show(); // Point

```

上例优先使用了父类的属性show

```

1 // 基类定义
2 class Point {
3     constructor(x,y) /*构造器*/ {
4         this.x = x;
5         this.y = y;
6         //this.show = () => console.log('Point');
7     }
8     show() /*方法*/ {
9         console.log(this,this.x,this.y);
10    }
11 }

```

```

12
13 // 继承
14 class Point3D extends Point {
15     constructor (x,y,z) {
16         super(x,y);
17         this.z = z;
18         this.show = () => console.log('Point3D');
19     }
20 }
21
22 let p2 = new Point3D(20,21,22);
23 p2.show(); // Point3D

```

优先使用了子类的属性。

总结

父类、子类使用同一种方式类定义属性或者方法，子类覆盖父类。

访问同名属性或方法时，优先使用属性。

静态属性

静态属性目前还没有得到很好的支持。

静态方法

在方法名前加上static，就是静态方法了。

```

1  class Add {
2      constructor(x, y) {
3          this.x = x;
4          this.y = y;
5      }
6      static print(){
7          console.log(this.x); // ? this是什么
8      }
9  }
10
11 var add = new Add(40, 50);
12 console.log(Add);
13 Add.print();
14 //add.print();
15 add.constructor.print(); // 实例可以通过constructor访问静态方法

```

静态方法中的this绑定的是Add类，而不是Add的实例

注意：

静态的概念和Python的静态不同，相当于Python中的类变量。

this的坑

虽然Js和 C++、Java一样有this，但是Js的表现是不同的。

原因在于， C++、Java是静态编译型语言，this是编译期绑定，而Js是动态语言，运行期绑定。

```

1  var school = {

```

```

2     name : 'magedu',
3     getNameFunc : function () {
4         console.log(this.name);
5         console.log(this);
6         return function () {
7             console.log(this === global); // this是否是global对象 globalThis
8             return this.name;
9         };
10    };
11 };
12
13 console.log(school.getNameFunc());
14
15 /* 运行结果
16 magedu
17 { name: 'magedu', getNameFunc: [Function: getNameFunc] }
18 true
19 undefined
20 */
21
22 function a(x,y,z) {
23     console.log(this);
24 }
25
26 a(1,2,3); // 看看this是什么

```

为了分析上面的程序，先学习一些知识：

函数执行时，会开启新的执行上下文EnvironmentContext。

创建this属性，但是this是什么就要看函数是怎么调用的了。

- 1、myFunction(1,2,3)，普通函数调用方式，this指向**全局对象**。全局对象是nodejs的global，或者浏览器中的window。非严格模式，函数中this是全局对象；严格模式，函数中this是undefined。最新版本中建议使用**globalThis**来统一全局变量。
- 2、myObject.myFunction(1,2,3)，对象方法的调用方式，this指向包含该方法的对象。
- 3、call、apply、bind方法调用时，要看第一个参数是谁。

分析上例

magedu 和 { name: 'magedu', getNameFunc: [Function: getNameFunc] } 很好理解。

第三行打印的true，是 console.log(this == global) 执行的结果，说明当前是global，因为调用这个返回的函数是直接调用的，这就是个普通函数调用，所以this是全局对象。

第四行undefined，就是因为this是global，全局中没有name属性。

这就是函数调用的时候，调用方式不同，this对应的对象不同，它已经不是C++、Java的指向实例本身了。

this的问题，这是历史遗留问题，新版只能保留且兼容了。

而我们在使用时，有时候需要明确的让this必须是我们期望的对象，如何解决这个问题呢？

1 显式传入

```

1 var school = {
2     name : 'magedu',
3     getNameFunc : function () {
4         console.log(this.name);
5         console.log(this);
6         return function (that) {
7             console.log(this == global); // this是否是global对象
8             return that.name;

```

```

9      };
10     }
11  }
12
13  console.log(school.getNameFunc()(school));
14
15  /* 运行结果
16  magedu
17  { name: 'magedu', getNameFunc: [Function: getNameFunc] }
18  false
19  magedu
20  */

```

通过主动传入对象，这样就避开了this的问题

2 ES3 (ES-262第三版) 引入了apply、call方法

```

1  var school = {
2      name : 'magedu',
3      getNameFunc : function () {
4          console.log(this.name);
5          console.log(this);
6          return function () {
7              console.log(this == global); // this是否是global对象
8              return this.name;
9          };
10     }
11 }
12
13 console.log(school.getNameFunc().call(school)); // call方法显式传入this对应的对象
14
15 /* 运行结果
16 magedu
17 { name: 'magedu', getNameFunc: [Function: getNameFunc] }
18 false
19 magedu
20 */

```

apply、call方法都是函数对象的方法，第一参数都是传入对象引入的。

apply传其他参数需要使用数组

call传其他参数需要使用可变参数收集

```

1  function Point(x,y) {
2      this.x = x;
3      this.y = y;
4      console.log(this === global);
5      console.log('Point ~~~~');
6  }
7
8  var p1 = Point(4, 5);
9  console.log(p1); // 打印什么?
10
11 console.log('~~~~~')

```

```

12
13 p2 = new Object();
14 console.log(p2);
15
16 p3 = Point.call(p2, 10, 11);
17 //p3 = Point.apply(p2, [11, 13]);
18 console.log(p3); // 打印什么?
19 console.log(p2); // 打印什么?

```

3 ES5 引入了bind方法

bind方法来设置函数的this值

```

1  var school = {
2      name : 'magedu',
3      getNameFunc : function () {
4          console.log(this.name);
5          console.log(this);
6          return function () {
7              console.log(this == global); // this是否是global对象
8              return this.name;
9          };
10     }
11 }
12
13 console.log(school.getNameFunc().bind(school)); // bind方法绑定
14
15 /* 运行结果
16 magedu
17 { name: 'magedu', getNameFunc: [Function: getNameFunc] }
18 [Function: bound ]
19 */

```

只打印了三行，说明哪里有问题，问题出在bind方法用错了。

```

1  var school = {
2      name : 'magedu',
3      getNameFunc : function () {
4          console.log(this.name);
5          console.log(this);
6          return function () {
7              console.log(this == global); // this是否是global对象
8              return this.name;
9          };
10     }
11 }
12
13 var func = school.getNameFunc();
14 console.log(func);
15
16 var boundfunc = func.bind(school); // bind绑定后返回新的函数
17 console.log(boundfunc);
18 console.log(boundfunc());
19
20 /* 运行结果

```



```

21 magedu
22 { name: 'magedu', getNameFunc: [Function: getNameFunc] }
23 [Function]
24 [Function: bound ]
25 false
26 magedu
27 */

```

apply、call方法，参数不同，调用时传入this。
bind方法是为函数先绑定this，调用时直接用。

4 ES6引入支持this的箭头函数

ES6 新技术，就不需要兼容this问题。

```

1  var school = {
2      name : 'magedu',
3      getNameFunc : function () {
4          console.log(this.name);
5          console.log(this);
6          return () => {
7              console.log(this == global); // this是否是global对象
8              return this.name;
9          };
10     }
11 }
12
13 console.log(school.getNameFunc()());
14
15 /* 运行结果
16 magedu
17 { name: 'magedu', getNameFunc: [Function: getNameFunc] }
18 false
19 magedu
20 */

```

ES6 新的定义方式如下

```

1  class School{
2      constructor(){
3          this.name = 'magedu';
4      }
5
6      getNameFunc() {
7          console.log(this.name);
8          console.log(this, typeof(this));
9          return () => {
10             console.log(this == global); // this是否是global对象
11             return this.name;
12         };
13     }
14 }
15
16 console.log(new School().getNameFunc()());
17 /* 运行结果

```

```
18 magedu
19 school { name: 'magedu' } 'object'
20 false
21 magedu
22 */
```

以上解决this问题的方法，bind方法最常用。