JavaScript面向对象（三）

this关键字

> 在之前讲构造函数的时候我们已经初步的接触过了this，他是一个指针，指向谁就是谁

`this`可以理解为他就是一个指针，具体表现形式要看**在调用时他指向了谁**

对象中的this

之前面向对象基础的时候，我们说过在面向对象中会经常使用`this`关键字，这个`this`在对象中到底只想谁?

```
var name = "张三";
var stu1 = {
    name: "李四",
    sayHello: function () {
        console.log(this);  // this-----> stu1
        console.log("大家好，我叫" + this.name);
    }
}
console.log(this);      // this-----> window
console.log(this.name); // 张三
stu1.sayHello()         // 李四
```

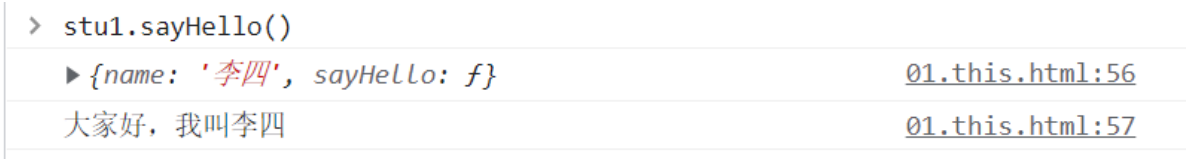
在上面的代码里面我们看到如果在全局环境下面`this`指向了`window`，在`shu1`对象的内部我们使用`this`他又指向了`stu1`，那么`this`到底只想谁?

1. 在全局环境下，使用`this`他就指向`window`

```
> console.log(this)
                                                    VM139:1
  ▶ Window {window: Window, self: Window, document: document, name: '张三', location: Location, …}
```

直接在控制台打印`this`，我们可以看到`this`指向了`window`全局对象

2. 在自定义对象里面使用 `this`，他就执行当前调用它的这个对象



在上面的代码里面我们的 `sayHello` 这一方法被 `stu1` 这个对象在调用，所以 `sayHello` 这一方法的内部使用 `this` 就指向了 `stu1`，方法里面的 `this.name` 就相当于 `shu1.name`

> 根据上面 代码的特点，全局环境的 `this` 指向 `window` 这个特点我们扩展一个点
>
> 之前在讲变量的时候，我们提到了**全局变量**与**局部变量**，全局变量就是在方法外使用 `var` 关键字来定义的变量，局部变量就是在方法内部定义的变量
>
> 以面向对象的角度去理解：**全局变量就是在全局环境下定于的变量**，全局环境就是 `window`，所以可以认为全局变量就是在 `window` 对象下面定义的变量
>
> 在JavaScript里面，一切皆对象，所以我们可以这个理解，`window` 就是一个**浏览器**的全局对象（最高对象），所以我们如果在 `window` 全局环境下使用 `var` 关键字定义的对象就是相当于是在 `window` 对象下面扩展了一个属性
>
> ```
> 1   var userName = "张三";
> ```
>
> 上面的代码在全局环境下定义了一个 `userName` 变量，等价于下面的代码

```
var userName = "张三";
/**
* 定义了一个全局变量 userName
* 相当于在window对象下面扩展了一个userName属性
*/
console.log(userName);          // 张三

console.log(window.userName);   // 张三

// 全局环境下面this 指向window
console.log(this.userName);     // 张三
```

这就是我们是我们能够通过`this.userName`打印“张三”的原因


```
// function abc() {
//   console.log("我是方法abc");
//   console.log(this);
// }
// 上面的方法也可以通过下面的函数表达式的方式来定义

// var abc = function () {
//   console.log("我是方法abc");
//   console.log(this);
// }
// abcs 一个全局的变量，全局变量相当于在window对象下面新增一个属性

// 所以上面的代码又可以用下面这种方式来定义
window.abc = function () {
  console.log("我是方法abc");
  console.log(this);
}


// 在这里调用的时候 我们法线当前的this指向的我window
// abc()
// 这种调用方式也可以理解为下面的调用方式
window.abc()
```

由上面的代码我们可以得出`this`总是指向调用它的对象也就是上面说到的两点：

1. 普通函数调用环境为`window`所以`this`指向`window`
2. 对象里面的方法调用，是通过对象`.`方法名来调用所以`this`指向调用它的这个对象

> **注意：** 在使用`window`全局对象来调用方法的时候我们可以把`window`省略

小练习

```
var userName = "标哥";

  var obj1 = {
    userName: "赛罗奥特曼",
    sayHello: function () {
      console.log(this.userName)
    }
  }

  obj1.sayHello()   // 打印什么？      "赛罗奥特曼"
  //把sayHello的方法赋值给了aaa
  var aaa = obj1.sayHello;

  aaa()             // 打印什么？     "标哥"
  /*
    普通函数调用    相当于window.aaa()
    这个时候的this就是window
    this.userName ----> window.userNmae
    -----> var userName
  */
```

构造函数里面的`this`

在之前讲构造函数的时候，我们就说过构造函数在调用的时候，对象当中的`this`就指向了当前的这个对象

```
function Penson(userName, age) {
```

```
    this.userName = userName;
    this.age = age;
    this.sayHello = function () {
      console.log("大家好，我叫" + this.userName);
    }
}

var p1 = new Penson('张三', 20)

console.log(p1.userName);
console.log(p1.age);
p1.sayHello();
// p1这个对象是构造函数创建的对象 在构造函数内部的this就会指向当前对象
// 这里的this.userName 等价于 p1.userName
```

函数调用方式的不同决定this指向

之前我们讲过了函数的调用形式

> 1. 方法名+()调用
> 2. new 函数名 构造函数的调用
> 3. call()/apply()调用

1. 在全局环境下调用方法 `方法名+()`
    - 这种方式的 `this` 指向全局对象 `window`
2. 对象内部的方法 `对象名.方法名()`
    - 这种方式的 `this` 指向当前调用它的对象
3. 构造函数调用 `new 方法名()`
    - 这种方式的 `this` 指向当前创建的对象
4. `call()/apply()`
    - 这种方式的 `this` 指向参数传递的那个对象

补充

在之前我们已经讲过了方法的调用方式，在JS里在，函数的调用是有很多种方式

1. `函数名()` 这种方式来调用
2. `new 函数名()` 来调用

除了以上两种方式外，还有两种调用方式

通过`call()`的方式来调用

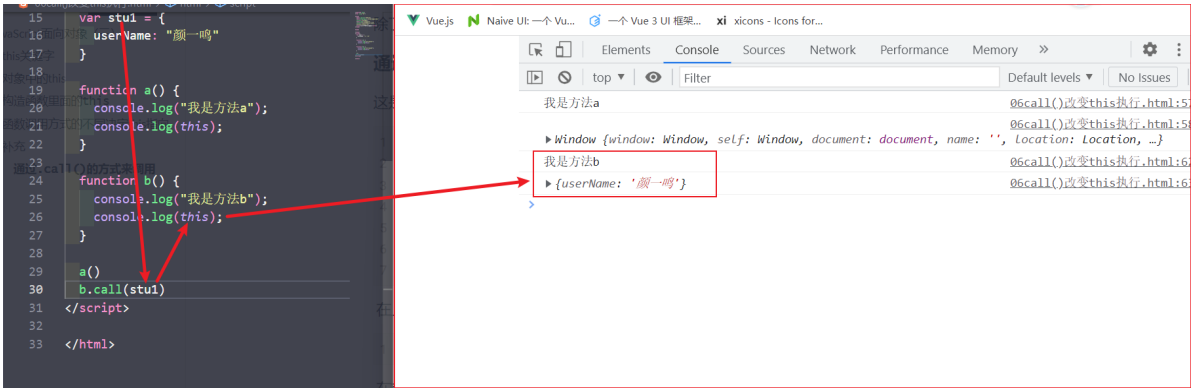
这是一种最新的调用方式，具体我们看代码

```
function a() {
    console.log("我是a方法");
}

// a() // 以前的调用方式

a.call()
```

在上面的代码中我们看到函数还可以通过`call()`的方式来调用，它的语法格式如下：

```
函数名.call(this指向?,...原函数参数?);
```

在前面我们讲到了`this`的指向问题，普通函数在调用的时候`this`执行全局`window`对象，在构造函数通过`new`去调用的时候内部的`this`指向当前创建的这个对象，在对象的调用自己的方法时，对象内部的`this`指向当前调用自己的这个对象



代码分析：

1. 在上面的代码里面，我们通过`a()`的方式是普通函数的调用，所以它的`this`就指向了全局对象`window`
2. 在`b.call(stu1)`是通过`call()`方法来调用，这样里面的`this`就指向了`stu1`这个对象

通过上面的代码我们可以充分说明，`call`是可以改变`this`指向的

同时函数在通过`call`去调用的时候也是可以传递参数的

```
var obj = {
    name: "张三"
}
function sum(x, y) {
return x + y
}

var x = sum(1, 2)      // 普通函数调用
console.log(x);

var y = sum.call(obj, 1, 2)     // call调用 第一个参数是this指向，后面跟原函数参数
console.log(y);
```

通过`apply()`的方式来调用

函数通过`apply()`调用其实与`call()`调用作用是一样的，语法如下：

```
函数名.apply(this指向?,[...原函数参数]?)
```

```
var stu1 = {
    userName: "张三"
}

function a() {
    console.log("我是方法a");
    console.log(this);
}

a()
a.call(stu1)     // call调用改变this指向 stu1
a.apply(stu1)    // apply调用改变this指向 stu1
```

> 上面的代码可以看到 `call` 与 `apply` 调用效果是一模一样的，都可以改变 `this` 的指向，它们唯一的区别就是后面的参数传递不一样

```
var obj = {
    name: "aa"
}

function sum(x, y) {
    return x + y
}

var a = sum(1, 2)
console.log(a);

var b = sum.call(obj, 1, 2)       // call 是将原函数的参数一个一个传进去
console.log(b);

var c = sum.apply(obj, [1, 2])  // apply 是讲原函数的参数放到一个数组里面在传进去
console.log(c);
```

> 通过上面的代码我们发现 `call` 是向原函数内部一个一个的传递参数，而 `apply` 是吧所有的参数都放到一个数组里面然后再去传递