.

第二课 this

课程目录

- **this 概念**
- **this 的应用**

一．this 的概念

面向对象语言中 this 表示当前对象的一个引用，但在 JavaScript 中 this 不是固定不变的，它会随着执行环境的改变而改变，它的作用就是在函数体内部，指代当前的运行环境，this 是专门用来获取当前运行的上下文环境（Execution Context）的。

全局环境：JavaScript 代码运行首先进入的全局环境

函数内部环境：当函数被调用时，会运行当前函数中环境代码

eval：计算某个字符串，并执行其中的的 JavaScript 代码，不做过多讲解

函数在定义的时候无法确定 this 引用取值，因为函数没有被调用，也就没有运行的上下文环境，因此在函数中 this 的引用取值，是在函数被调用的时候确定的。函数在不同的情况下被调用，就会产生多种不同的运行上下文环境，所以 this 的引用取值也就是随着执行环境的改变而改变了。

二．this 的应用

全局环境

this 是 window。

```
console.log(this);
```

函数作为对象的方法

在对象方法中，作为对象的一个方法被调用时，this 指向调用它所在方法的对象。

```
var obj = {
  abc: "111",
  cba: "222",
  add : function() {
    return this.abc + this.cba;
  }
};
obj.add();
```

在构造函数中的 this

通过 new 操作符来构建一个构造函数的实例对象，这个构造函数中的 this 就指向这个新的实例对象。同时构造函数 prototype 属性下面方法中的 this 也指向这个新的实例对象。

```
function Abc(){
    this.name = '星星课堂';
    this.num = 100;
}
Abc.prototype.getNum = function(){
    return this.num;
}
var a1 = new Abc();
```

```
console.log(a1.name);
console.log(a1.getNum());
```

普通函数

普通函数在全局下调用时，其中的 this 指向的是 window。

```
function fn1(){
    console.log(this);
}
fn1();
```

特例，之前说过对象下面的方法中的 this 指向的是这个对象，但是如果在对象方法内部声明一个函数，这个函数的 this 在对象方法执行的时候指向就不是这个对象了，而是指向 window 了。

```
var obj = {
    abc:function(){
        function cba(){
            console.log(this);
        }
        cba();
    }
}
obj.abc();
```

事件回调函数

this 指向的是触发事件的 DOM 元素。

```
var btn1 = document.getElementById('btn1');
btn1.addEventListener('click',function(){
    console.log(this);
});
```

call，apply，bind 显式修改 this 指向

call ,apply ,bind 是函数对象的方法。他们都是为了改变某个函数运行时的 this 指向。

call 方法

call 方法是每个函数建立的时候就有的方法，另外 call 方法也可以传入其他的参数。

```
var obj1 = {
    name:'111',
    num:'222',
    add:function(text) {
        console.log(this.name + this.num + text);
    }
}
var obj2 = {
    name:"333",
    num: "666",
}
```

```
obj1.add.call(obj2,'星星课堂')
```

apply 方法

apply()方法与 call 方法用法基本相同 ,不同点主要是 call()方法的第二个参数和之后的参数可以是任意数据类型，而 apply 的第二个参数是数组类型或者 arguments 参数集合。

```
var obj1 = {
    name:'111',
    num:'222',
    add:function(text) {
        console.log(this.name + this.num + text);
    }
}
var obj2 = {
    name:"333",
    num: "666",
}
obj1.add.call(obj2,['星星课堂']);
```

bind()方法（IE6,7,8 不支持）

bind()方法也能修改 this 指向，不过调用 bind()方法不会执行 test()函数，也不会改变 test()函数本身，只会返回一个已经修改了 this 指向的新函数，这个新函数可以赋值给一个变量，调用这个变量新函数才能执行 test1()。

```
function test(m,n){
    console.log(this.abc);
}
var test1 = test.bind({abc : 321});//这里修改 test 中的 this 指向为{abc:123}，并且返回一个新函数并赋值给 test1
test1();//这里输出 321
```

call()方法和 bind()方法的区别在于 **1.bind 的返回值是函数，并且不会自动调用执行，2.两者后面的参数的使用也不同。**call 是 把第二个及以后的参数作为原函数的实参传进去，而 bind 实参在其传入参数的基础上往后获取参数执行。

```
function fn(a,b,c){
    console.log(a,b,c);
}
var fn1 = fn.bind({abc : 123},600);
fn(100,200,300)   //这里会输出--> 100,200,300
fn1(100,200,300)   //这里会输出--> 600,100,200
fn1(200,300)   //这里会输出--> 600,200,300
fn.call({abc : 123},600) //这里会输出--> 600,undefined,undefined
```

谢谢观看！

我是星星课堂老师：周小周