IT前沿技术在线大学

www.zhenchuanx.com

JavaScript基础知识

先讲讲JavaScript中的this

this是什么？

this总是指向调用它所在方法的对象

this是什么？

this的指向与所在方法的调用位置有关，而与方法的声明位置无关

```
1   // 声明位置
2   var obj = {
3       name: 'Alice',
4       foo: function() {
5           console.log(this.name)
6       }
7   }
8
9   var otherObj = {
10      name: 'Cristiano',
11      foo: obj.foo
12  }
13
14  // 调用位置
15  obj.foo(); // Alice
16
17  otherObj.foo(); // Cristiano
```

this是什么？

在浏览器中，调用方法时没有明确对象的，this指向window。

```
// 声明位置
var obj = {
    name: 'Alice',
    foo: function() {
        console.log(this.name)
    }
}

var name = 'Bob'

var test = obj.foo;

test(); // Bob?
```

可以理解为执行了window.test()，但下道题不能这么想

```
1   // 声明位置
2   var obj = {
3       name: 'Alice',
4       foo: function() {
5           console.log(this.name)
6       }
7   }
8
9   var otherObj = {
10      name: 'Cristiano',
11      foo: function() {
12          var testFunc = obj.foo;
13          testFunc();
14      }
15  }
16
17  var name = 'Bob'
18
19  otherObj.foo(); // Bob
20
```

在浏览器中setTimeout、setInterval和匿名函数执行时的当前对象是全局对象window

```
var name = 'Bob'

// 声明位置
var obj = {
    name: 'Alice',
    showName: function() {
        console.log(this.name)
    },
    foo: function() {
        (function(cb) {
            cb();
        })(this.showName)
    }
}

obj.foo(); // Bob
```

this是什么？

在浏览器中，调用方法时没有明确对象的，this指向window。

Node中，这种情况，this是指向global吗？

this是什么？

In Node.js this is different. The top-level scope is not the global scope; `var something` inside a Node.js module will be local to that module.

但是在Node CLI下，与浏览器的行为保持一致

eval等同于在声明位置填入代码

```
1   var name = 'Bob'
2
3   // 声明位置
4   var obj = {
5       name: 'Alice',
6       showName: function() {
7           eval('console.log(this.name)')
8       }
9   }
10
11  obj.showName(); //Alice
```

apply和call能够强制改变函数执行时的当前对象，让this指向其他对象

```
// var name = 'Bob'
var name = 'Window'

// 声明位置
var obj = {
    name: 'Alice',
    showName: function() {
        console.log(this.name)
    }
}

var otherObj = {
    name: 'Cristiano'
}


obj.showName.apply(); //Window
obj.showName.apply(otherObj); //Cristiano
```

如何利用call或者apply实现bind?

你们知道其实eval也能call吗?

```
> var obj = {
    say: function () {
      eval.call(window, 'console.log(this)')
    }
  }
< undefined
> obj.say()
  ▶ Window {postMessage: f, blur: f, focus: f, close: f, frames: Window, …}
< undefined
> var obj2 = {
    say: function () {
      eval('console.log(this)')
    }
  }
< undefined
> obj2.say()
  ▶ {say: f}
< undefined
```

因为js的this太古怪，所以ES6开始，lamda表达式，或者有些人称作箭头函数，是在声明时候绑定this的

```
> var name = 'Bob';

  var obj = {
      name: "Alice",
      showName: () => {
          console.log(this.name);
      }
  }

  obj.showName();
  Bob
< undefined
>
```

有兴趣可以去看看Babel是怎么转的～

但是在use strict模式下，this的绑定规则有点不一样：https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Strict_mode

```
1   var name = 'Bob';
2   function foo() {
3       console.log(this.name)
4   }
5
6   (function() {
7       'use strict'
8       console.log(this)
9       foo.call(this);
10      foo.call(undefined);
11      foo.call(null);
12  })()
```

OK，我们看看练习

JavaScript中的声明提升

```
1    foo();
2
3    function foo() {
4        console.log( a ); // undefined
5        var a = 2;
6    }
```

为什么没有抛错？

Only declarations are hoisted

- 函数表达式不会提升（具名的也不行）

```
18    foo(); // TypeError
19    bar(); // ReferenceError
20    var foo = function bar() {
21        // ...
22        console.log('hello!')
23    };
```

Only declarations are hoisted

- 函数声明优先于变量声明提升

```
26    foo(); // 1
27    var foo;
28    function foo() {
29        console.log( 1 );
30    }
31    foo = function() {
32        console.log( 2 );
33    }
```

Only declarations are hoisted

- 后面出现的函数声明可以覆盖前面的（千万不要这么做）

```
51  //后面出现的函数声明可以覆盖前面的
52  foo() //3
53
54  function foo() {
55      console.log( 1 );
56  }
57  var foo = function() {
58      console.log( 2 );
59  };
60  function foo() {
61      console.log( 3 );
62  }
```

Only declarations are hoisted

- 声明提升不会被条件判断所控制

```
> if(someVar === undefined){
      alert("someVar未定义");
  }
▶Uncaught ReferenceError: someVar is not defined
      at <anonymous>:1:1
> if(someVar === undefined){
      someVar = 1;
      alert("someVar未定义");
  }
▶Uncaught ReferenceError: someVar is not defined
      at <anonymous>:1:1
> if(someVar === undefined){
      var someVar = 1;
      alert("someVar未定义");
  }
  alert: someVar未定义
<- true
```

Javascript 继承

OOP就不要让我讲了，虽然原则上js并不一定需要OOP

我们知道OOP的三的特性是：封装、继承、多态

- 凡是不希望别人知道内部实现的则进行封装，内外隔离
- 凡是系统需要归一化，为了处理方便对所处理的对象有统一要求，则使用继承和多态

注意规则不是银弹，更多的可能需要在自己的编程过程中体验了～

前人使用了很多黑魔法来实现继承，
我们看看ES6怎么继承

OK，回来看看黑魔法

我们来看看练习

跨域解决方案

同源策略

Netscape 最开始为了Cookie而创建的规则

- 协议相同
- 域名相同
- 端口相同

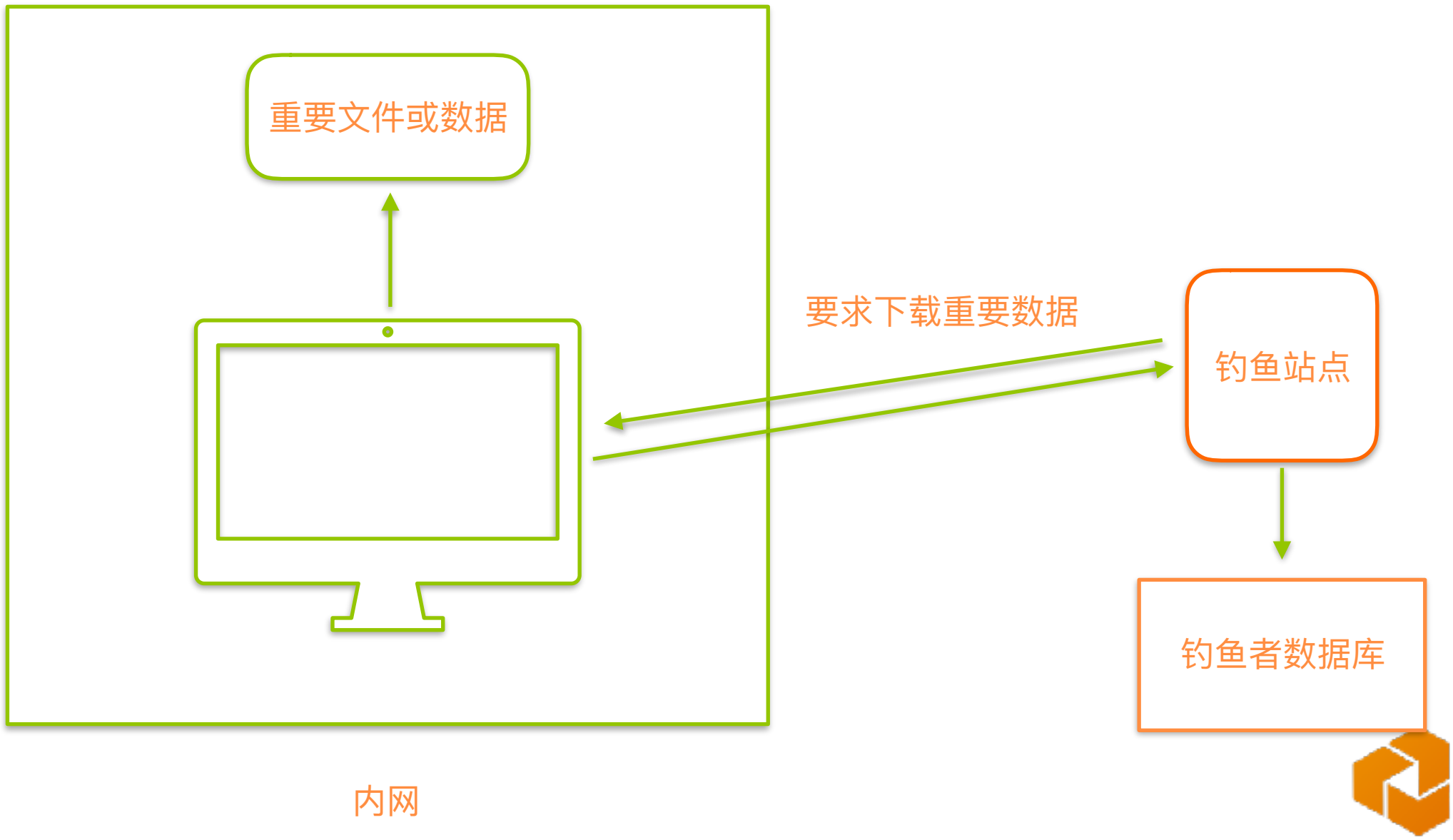
同源策略

- Cookie、LocalStorage、Indexed DB无法读取
- DOM无法获得
- AJAX请求无法发出

别的估计大家都能理解，那为什么AJAX也被同源策略限制呢？

这主要出于数据安全考虑，不同源的网站，
不应当能获取其数据

举个例子



OK，我们看一下showcase

IT前沿技术在线大学