IT前沿技术在线大学

www.zhenchuanx.com

前端基础知识（一）

this是什么？

一般而言，在Javascript中，this指向函数执行时的当前对象。

一般而言，在Javascript中，this指向函数执行时的当前对象。

In JavaScript, as in most object-oriented programming languages, this is a special keyword that is used within methods to refer to the object on which a method is being invoked.

值得注意，该关键字在Javascript中和执行环境，而非声明环境有关

```
var someone = {
    name: 'Bob',
    showName: function () {
        alert(this.name)
    }
};

var other = {
    name: 'Tom',
    showName: someone.showName
}

other.showName();    // Tom
```

当没有明确的执行时的当前对象时，this指向全局对象window，Node是global。

```
1   var name = 'Tom'
2
3   var Bob = {
4       name: 'Bob',
5       show: function () {
6           alert(this.name)
7       }
8   }
9
10  var show = Bob.show
11  show()    // Tom
12
```

这道题我们也能理解成调用了window.show()，但下面的题目并不能这么理解

```
var name = 'window'

var Bob = {
    name: 'Bob',
    showName: function () {
        alert(this.name)
    }
}

var Tom = {
    name: 'Tom',
    showName: function () {
        var fun = Bob.showName
        fun()
    }
};

Tom.showName()    // window
```

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'
var nameObj = {
    name: 'Tom',
    showName: function () {
        alert(this.name)
    },
    waitShowName: function(){
        !function (__callback) {
            __callback()
        }(this.showName)
    }
}

nameObj.waitShowName()    // Bob
```

eval等同于在该地方填入代码

```
1   var name = 'window'
2
3   var Bob = {
4       name: 'Bob',
5       showName: function () {
6           eval('alert(this.name)')
7       }
8   }
9
10  Bob.showName()     // Bob
```

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 = 'window'

var someone = {
    name: 'Bob',
    showName: function () {
        alert(this.name)
    }
}

var other = {
    name: 'Tom'
}

someone.showName.apply()    // window
someone.showName.apply(other)    // Tom
```

想当年满地都是如何利用call或者apply实现bind的面试题，大家知道怎么实现吗？

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 obj = {
    say: () => {
      console.log(this)
    }
  }
< undefined
> obj.say()
  ▶ Window {postMessage: f, blur: f, focus: f, close: f, frames: Window, …}
< undefined
```

有兴趣可以去看看Babel是怎么转的～

哦哦哦，注意在use strict模式下，有点不一样：https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Strict_mode

```
function foo() {
    console.log(this.a)
}

(function(){
    "use strict";
    console.log(this)
    foo.call(this)
    foo.call(undefined)
    foo.call(null)
})()
```

OK，我们看看练习

Javascript声明提升

```
1    hello()
2
3    function hello () {
4        console.log('Hello, world!')
5    }
```

为什么能运行呢?

只有声明能提升

```
> 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
```

let 的小剧场一时半会讲不清楚，
大家还是看文章吧

Javascript 继承

OOP就要让我讲了，虽然原则上js并不一定需要OOP

前人使用了很多黑魔法来实现继承，
我们先看看ES6怎么继承

OK，回来看看黑魔法

给大家点时间看看练习

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

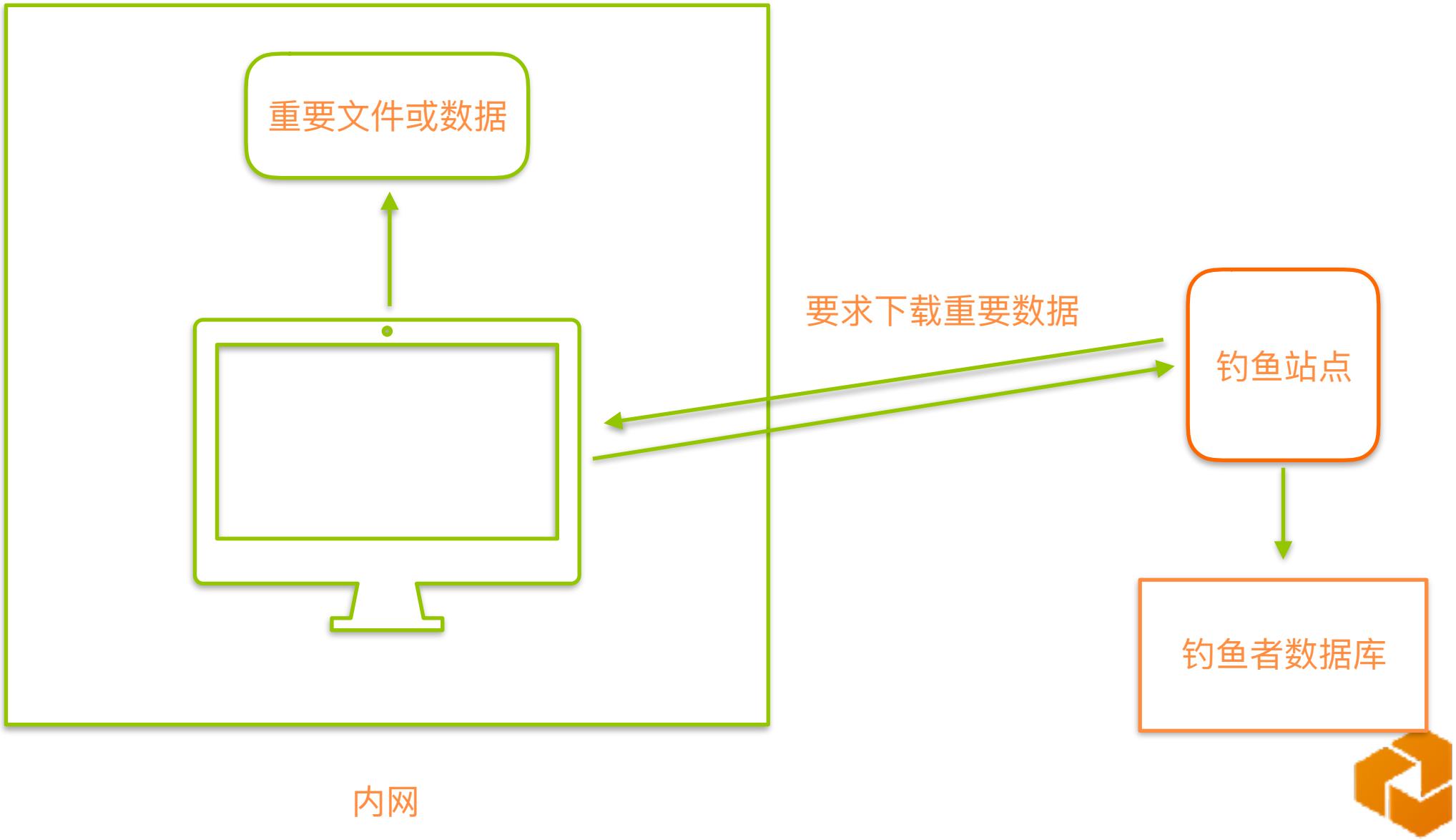
同源策略

- 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

模块化

我们先看看别人的PPT回顾一下历史

看看我们showcase

IT前沿技术在线大学