真传X

IT前沿技术在线大学

www.zhenchuanx.com

基础知识（一）

大纲

- this
  - sloppy mode
  - strict mode
  - call/apply/bind
  - arrow functions
  - nodejs
- hoisting & scope
  - var
  - function
  - let/const
- oop & inheritance
  - classical inheritance
  - prototypal inheritance
- cross-origin
  - cors

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

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

var otherObj = {
  name: 'bbb',
  getName: obj.getName
};

// 调用位置
obj.getName(); // aaa

otherObj.getName(); // bbb

```

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

```
8   // 声明位置
9   var obj = {
10      name: 'aaa',
11      getName: function() {
12          console.log(this.name);
13      }
14  };
15
16  var name = 'ccc';
17
18  var getName = obj.getName;
19
20  getName(); // ccc ? undefined ?
```

```
8   //·声明位置
9   var·obj·=·{
10  ··name:·'aaa',
11  ··getName:·function()·{
12  ····console.log(this.name);
13  ··}
14  };
15
16  var·otherObj·=·{
17  ··name:·'bbb',
18  ··getName:·function()·{
19  ····var·getName·=·obj.getName;
20  ····getName();
21  ··}
22  };
23
24  var·name·=·'ddd';
25
26  otherObj.getName();·//·ddd·?·undefined·?
```

this 是什么

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 that = this;

(function() {
  console.log(this === that); // true? false?
})();

setTimeout(function() {
  console.log(this === that); // true? false?
}, 0);
```

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 REPL下，与浏览器的行为保持一致(timers除外)

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

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

var otherObj = {
    name: 'bbb'
};

var name = 'fff';

// 调用位置
obj.getName.call();
obj.getName.call(otherObj);
obj.getName.apply();
obj.getName.apply(otherObj);
obj.getName.bind(this)();
obj.getName.bind(otherObj)();
```

*eval*等同于在声明位置填入代码

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

```
8   (function() {
9     'use strict';
10
11    function getName() {
12      console.log(this);
13    }
14
15    console.log(this); // undefined
16    getName.call(this); // undefined
17    getName.call(null); // null
18    getName.call(undefined); // undefined
19  })();
```

由于js的this太古怪，所以ES6开始，箭头函数(lamda 表达式)，在声明时候绑定this

有兴趣可以看看 babel 是怎么将箭头函数转换成 ES5的

https://github.com/FE-star/exercise4

HOISTING & SCOPE

```
8   foo();
9
10  function foo() {
11      console.log(bar); // undefined
12      var bar = 'aaa';
13  }
```

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)

```
8   foo(); // TypeError: foo is not a function
9   bar(); // ReferenceError: bar is not defined
10
11  var foo = function bar() {
12    // ...
13    console.log('bar');
14  };
```

函数声明优先于变量声明提升

```
8   console.log(typeof foo); // function
9
10  function foo() {
11    console.log(1);
12  }
13
14  var foo = 'bar';
15
16  console.log({ foo }); // { foo: 'bar' }
```

后面出现的函数声明可以覆盖前面的(同名覆盖原则)

```
8     foo(); // 2
9
10    function foo() {
11      console.log(1);
12    }
13
14    function foo() {
15      console.log(2);
16    }
```

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
```

let / const & scope / closures

http://es6.ruanyifeng.com/#docs/let

https://zhuanlan.zhihu.com/p/28140450

https://css-tricks.com/javascript-scope-closures/

oop & inheritance

面向对象 vs 面向过程

封装/继承/多态

classical inheritance

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Classes

prototypal inheritance

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Inheritance_and_the_prototype_chain

https://github.com/FE-star/lianxi5

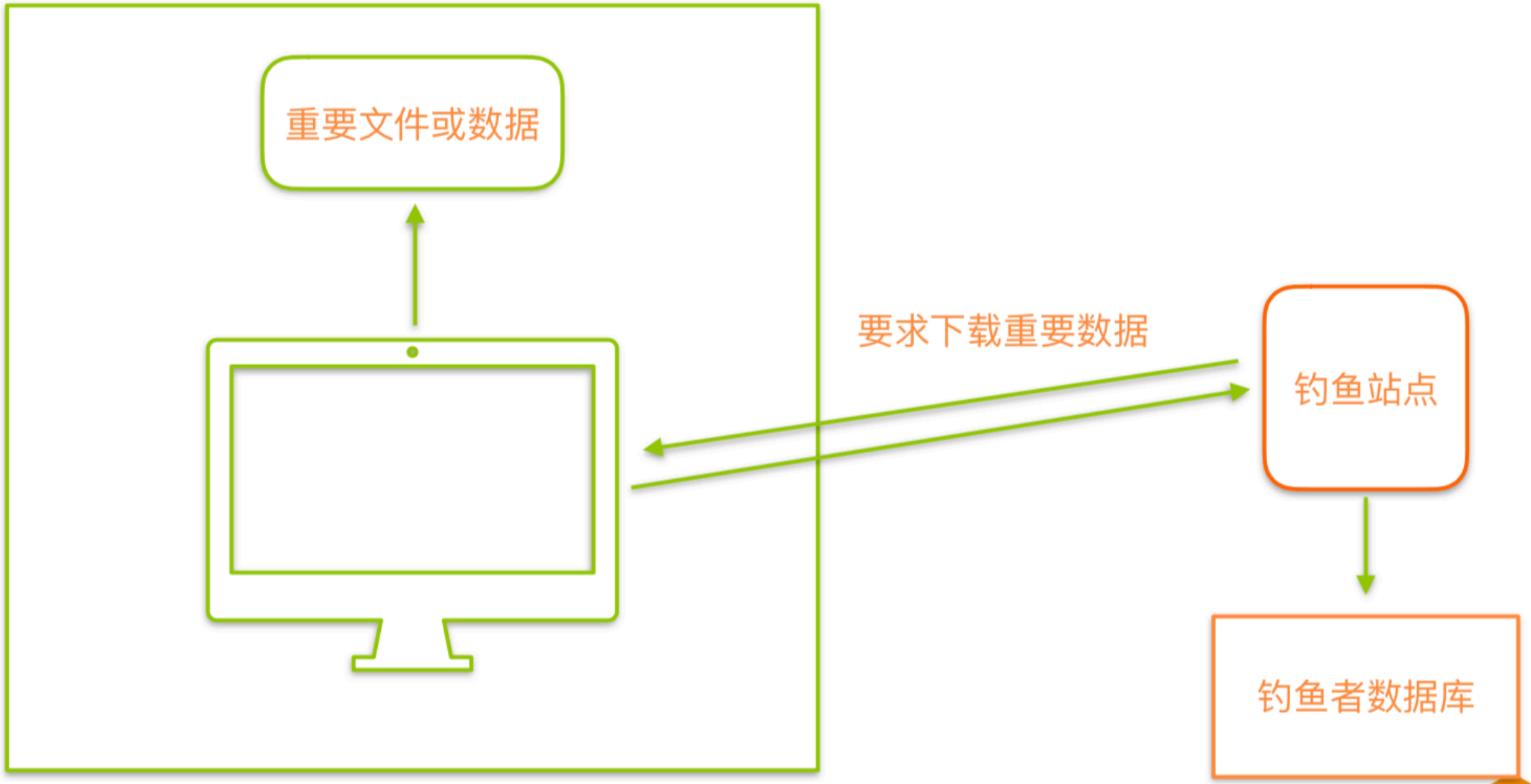
cross-origin

protocol / hostname / port

协议 / 域名 / 端口

https://github.com:443

举个例子



1、 JSONP (script src GET, 不支持 POST)

2、 document.domain + iframe （主域名一致)

3、 location.hash + iframe （不方便，有长度限制)

4、 window.name + iframe （兼容性)

5、 postMessage （新API，主域名一致)

6、 CORS （服务器配置，占用主域带宽)

8、 WebSocket （实现成本)

https://github.com/FE-star/showcase1

IT前沿技术在线大学