Javascript基础

知书堂 Python第二期 先导课程

2020年8月22日

目录

变量声明

- 函数内使用 var 声明的变量只能在函数内容访问，如果不使用，var 则是全局变量。
- 使用 let 关键字来实现块级作用域
- const 用于声明一个或多个常量，**不能通过再赋值修改**，也不能再次声明

类型

- Undefined 如果声明了一个变量，但没有任何赋值，那么这个变量就是Undefined；未定义的变量也都是undefined类型
- NaN 的值表示不是一个数字，Not a number
- Number-JavaScript中,就只有一种数字类型-双倍精度的浮点数类型
- 对象--一切都是对象，函数也是对象，数组也是对象

对象表达式

- obj.foo = true
- obj[‘a’+’bc’] = true
- var obj = {‘a’: 1}
- var abc = ‘1’;
  var obj = {abc}

- 标识符作为属性名
- 表达式作为属性名
- 字面量定义
- 对象表达式

函数声明与箭头函数

```
// 普通函数
function functionName(prameter1, parameter2,…) {}

// 匿名函数
let a = function(){}

// 箭头函数
let a = (parameter1, parameter2,…) => {}
```

数组

```
// 数组的声明
let arr = [];
var arr = new Array();
//数组元素的添加与删除
arr[0] = '1'
arr[1] = 2
arr[10] = 10
arr.push('11')
```

```
delete arr[10]
arr.splice(1, 1)
// 数组的合并
[1,2].concat([1,2])
```

数组解构、对象解构

```
let arr = [1, 2, 3];
let [a, b, c ] = arr;
console.log(a, b, c)

let arr = { a: 1, b: 2 };
let {a: a1, b: b1} = arr;
console.log(a1, b1)
let {a, b} = arr;
```

- 方便从数组中取值
- 方便从对象中取值

展开操作符

```
let arr = [1, 2, 3]
console.log(...arr)


function test(a,b,c) { }
var args = [0,1,2];
test(...args);
var arr1=['a','b','c'];
var arr2=[...arr1,'d','e'];
```

- 展开运算符（spread operator）允许一个表达式在某处展开。

同步与异步

同步：进商场的时候，出示健康码，排队，前面一个人完成后后面一个人才能进行

异步：前面一个人手机信号不好，那你等等，后面的人先来

```
setTimeout(() => console.log('i am time out'), 1)
console.log('i am a console log')
```

Ajax

//异步的JavaScript与XML技术

传统的Web应用允许用户端填写表单（form），当提交表单时就向网页服务器发送一个请求。服务器接收并处理传来的表单，然后送回一个新的网页。

AJAX应用可以仅向服务器发送并取回必须的数据，并在客户端采用JavaScript处理来自服务器的回应。

- Form放弃本页面，然后再请求数据
- Ajax在提交、请求、接收时，都是异步进行，网页不需要刷新

Ajax

```
function reqListener
(){ console.log(this.responseText);}
var oReq = new XMLHttpRequest();
oReq.addEventListener("load", reqListener);
oReq.open("GET",
"http://localhost:9090/index.html");
oReq.send();
console.log('after oReq.send')
```

- 这是JavaScript标准提供的进行异步数据请求的代码片段

回调地狱

```
setTimeout(function(){
    console.log("I am first")
    setTimeout(function(){
        console.log("I am second")
        setTimeout(function(){
            console.log("I am third")
        }, 1000)
    },1000)
},1000)
```

- 编写复杂的异步代码逻辑时，就会出现类似于左边这样的代码，俗称回调地狱

链式调用- Promise

```
let wait = (ms) => new Promise((resolve) => setTimeout(resolve, ms))


wait(1000)
    .then(() => {console.log("i am first"); return wait(1000)})
    .then(() => {console.log("i am second"); return wait(1000)})
    .then(() => console.log("i am third"))
```

- 这是一段用链式调用改造的代码，是不是比上面的代码清晰多了

Promise

```
// 简单的创建一个Promise
let promise = new Promise(functionName)
let promise = new Promise(function(resolve, reject) {
    if ("xxx") {
        resolve()
    }else{
        reject()
    }
})
```

Promise-状态

一个promise对象有三个状态

1. Pending
2. Fulfilled
3. Rejected

- 默认是pending状态
- 调用resolve后，编程fulfilled状态，调用then函数
- 调用reject函数后，变成Rejected状态

Promise-错误传递

```
let wait = (ms) => new Promise((resolve) =>
setTimeout(resolve, ms))


wait(1000)
    .then(() => new Promise((resolve, reject) =>{reject('i am
an error');} ))
    .then(() => {console.log("i am first"); return wait(1000)})
    .then(() => {console.log("i am second"); return wait(1000)})
    .then(() => console.log("i am third")).catch(err =>
{console.error(err)})
```

- 遇到异常抛出或者reject，promise顺着调用链寻找下一个onRejected失败回调，或者由catch指定的回调函数

Promise-思考题

```
Promise.resolve('foo')
.then(Promise.resolve('bar'))
.then(function (result) {
    console.log(result);
  });
```

- 这样的代码能运行么？
- 代码最终会输出什么

Promise all

```
let p1 = new Promise((resolve, reject) => {   resolve('成功了') })
let p2 = new Promise((resolve, reject) => {   resolve('success') })
let p3 = Promse.reject('失败')
Promise.all([p1, p2]).then((result) => {
  console.log(result)               //['成功了', 'success']
}).catch((error) => {  console.log(error)})

Promise.all([p1,p3,p2]).then((result) => {   console.log(result)
}).catch((error) => {   console.log(error)      // 失败了，打出 '失败'
})
```

- Promise.all可以将多个Promise实例包装成一个新的Promise实例。同时，成功和失败的返回值是不同的，成功的时候返回的是一个结果数组，而失败的时候则返回最先被reject失败状态的值。

Promise race

```
let p1 = new Promise((resolve, reject) => {
  setTimeout(() => {   resolve('success')   },1000)})

let p2 = new Promise((resolve, reject) => {
  setTimeout(() => {     reject('failed')   }, 500) })

Promise.race([p1, p2]).then((result) => {
  console.log(result)
}).catch((error) => {
  console.log(error)  // 打开的是 'failed'
})
```

- 顾名思义，Promse.race就是赛跑的意思，意思就是说，Promise.race([p1, p2, p3])里面哪个结果获得的快，就返回那个结果，不管结果本身是成功状态还是失败状态。

模块

```
npm install webpack -g --registry=https://registry.npm.taobao.org
npm install webpack-cli -g --registry=https://registry.npm.taobao.org
```

打包命令

```
webpack main.js -o dist/main.js
```

- 使用模块的前提-webpack
- 本质上，webpack 是一个现代 JavaScript 应用程序的静态模块打包器(module bundler)。当 webpack 处理应用程序时，它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 bundle。

使用命名导出

```
function cube(x) {
  return x * x * x;
}

export { cube};
```

```
// 导入
import { cube } from 'my-module.js';
// 导入全部
Import * as myModule from ‘my-module.js’
```

重命名导出

```
function cube(x) {
  return x * x * x;
}
// 重命名导出
export { cube as cubeFunc};
```

```
// 重命名导入
import {cubeFunc as cube }
from 'my-module.js';
```

默认导出

```
let sum = function(a, b)
{return a+b;}
export default sum;
```

```
// 无需知道导出的变量名
import sss from
'./array_example'
console.log(sss(1, 2))
```

本节课所用到工具

- Nodejs & npm
- python的http server
  - python -m http.server --bind 127.0.0.1 9090
- Webpack and webpack-cli
- Vscode
- Chrome的调试器

感谢大家的参与

交流QQ群：480110023