


京城一灯-VIP 编程题第四页

 京程一灯


一灯大家庭

面试题库

面试经验

藏经阁

内推资源



上一题

下一题

实现一个 setter 方法

代码实现

实现一个 setter 方法

```
let setter = function (content, key, value) {
  // your code
};

let n = {
  a: {
    b: {
      c: { d: 1 },
      bx: { y: 1 },
    },
    ax: { y: 1 },
  },
};

// 修改值
setter(n, "a.b.c.d", 3);
console.log(n.a.b.c.d); //3
setter(n, "a.b.bx", 1);
console.log(n.b.bx); //1
```

代码实现

• 实现一

```
let setter = function (content, key, value) {
  let argArr = key.split('.');
  let i = argArr.shift();
  if (argArr.length==0){
    content[i]=value;
  }else{
    content[i]=setter(content[i],argArr.join('.'),value);
  }
  return content;
};

let n = {
};

let n = {
  a: {
    b: {
      c: { d: 1 },
      bx: { y: 1 },
    },
    ax: { y: 1 },
  },
};

// 修改值
setter(n, "a.b.c.d", 3);
console.log(n.a.b.c.d); //3
setter(n, "a.b.bx", 1);
console.log(n.a.b.bx); //1
```

• 实现二

```
let setter = function (content, key, value) {
  try {
    let keyArr = key.split(".");
    let (obj, k) = keyArr.reduce((content, k, i) => {
      if (i !== keyArr.length - 1) {
        return content[k];
      } else {
        return { obj: content, k };
      }
    }, content);
    obj[k] = value;
  } catch (e) {
    console.warn("输入key有误");
  }
};

let n = {
  a: {
    b: {
      c: { d: 1 },
      bx: { y: 1 },
    },
    ax: { y: 1 },
  },
};

// 修改值
setter(n, "a.b.c.d", 3);
console.log(n.a.b.c.d); //3
setter(n, "a.b.bx", 1);
console.log(n.a.b.bx); //1
```



• 实现三

```
let setter = function(content,key,value){
  var proto = key.split('.');
  var len = proto.length;
  proto.reduce(function(obj,val,index){
    if(index === len-1){
      return obj[val] = value;
    }
    if (typeof obj[val] !== 'object') obj[val] = {};
    return obj[val];
  },content)
}

let n = {
  a: {
    b: {
      c: { d: 1 },
      bx: { y: 1 },
    },
    ax: { y: 1 },
  },
};

// 修改值
setter(n, "a.b.c.d", 3);
console.log(n.a.b.c.d); //3
setter(n, "a.b.bx", 1);
console.log(n.a.b.bx); //1
```

Copyright © 2016 ystengqiangtang.com All Rights Reserved. 京ICP备1602242号-1

一灯大家庭面试题库面试经验藏经阁内推资源

[上一题](#)[下一题](#)

修改以下代码，使得最后一行代码能够输出数字 0-9
(最好能给出多种答案)

代码实现

方案一

方案二

方案三

修改以下代码，使得最后一行代码能够输出数字 0-9 (最好能给出多种答案)

```
var arrys = [];
for (var i = 0; i < 10; i++) {
  arrys.push(function () {
    return i;
  });
}
arrys.forEach(function (fn) {
  console.log(fn());
}); // 本行不能修改
```

代码实现

解决此题的关键就是知道在for循环的时候，如何记住我是谁。
解决方案清晰很多：块级作用域或者闭包

方案一

let 和const声明的变量遇到块都会形成一个块，此时是个变量，使用let

```
for (let i = 0; i < 10; i++) {
  arrys.push(function () {
    return i;
  });
}
```

方案二

自执行函数形成闭包


```
for (var i = 0; i < 10; i++) {
  (function(j){
    arrys.push(function () {
      return j;
    });
  })(i);
}
```

方案三

内部函数自执行形成闭包

```
for (var i = 0; i < 10; i++) {
  arrys.push((function (j) {
    return _ => j;
  })(i));
}
```

Copyright © 2016 yidengputaotang.com All Rights Reserved. ©ICP备16022242号-1

 京程一灯

一灯大家庭 面试题库 面试经验 藏经阁 内推资源

上一步 下一步

实现一个函数柯里化

函数柯里化

1.1 概念

1.2 比如实现 `addCurry(2)(3) => 6`

实现一个函数柯里化

函数柯里化

1.1 概念

柯里化 (Currying) 是把接受多个参数的函数转变为接受一个单一参数的函数，并且返回接受余下的参数且返回结果的新函数的技术。

1.2 比如实现 `add(1)(2)(3) => 6`

参数长度固定

```
const curry = (fn) =>
(judge = (...args) =>
  args.length === fn.length
  ? fn(...args)
  : (...arg) => judge(...args, ...arg));
const add = (a, b, c) => a + b + c;
const curryAdd = curry(add);
console.log(curryAdd(1)(2)(3)); // 6
```

参数长度不固定

```
function add (...args) {
  //求和
  return args.reduce((a, b) => a + b)
}

function currying (fn) {
  let args = []
  return function temp (...newArgs) {
    if (newArgs.length) {
      args = [
        ...args,
        ...newArgs
      ]
      return temp
    } else {
      let val = fn.apply(this, args)
      args = [] //保证再次调用时清空
      return val
    }
  }
}

let addCurry = currying(add)
console.log(addCurry(1)(2)(3)) //16
```

Copyright © 2016 yidengputang.com All Rights Reserved. 京CP备 16022542号-1

Promise 链式调用如何实现

Promise 链式调用

1 链式调用原理

• `promise1 = new Promise(resolve => { resolve, reject }) => { ... }` 中的 `resolve` 是在内部执行的，但是后续得 `resolve` 可能是在异步操作中
 • `promise1.then` 会向 `promise1` 添加回调，然后返回一个新的 `promise2`，这个新的 `promise2` 的决议逻辑之和由前一个的 `resolve/promise` 方法
 • `promise1` 决议后会自动执行回调，避免执行 `then` 中传入的 `onFulfilled(promise1.value)`，然后返回 `x`，再执行 `resolve/promise(promise2, x, promise1.resolve, promise1.reject)`
 • 如果 `x` 是个未决议的 `Promise` 或者未决议的 `thenable` 对象，那么就不会 `promise1.resolve(x)` 而是 `promise2`
 • 如果 `x` 是个 `pending` 状态的 `promise` 或者 `thenable` 对象，那么执行 `x.then`，将 `resolve/promise` 放入 `x` 的回调函数队列，等待 `x` 决议后 `x.value` 成功赋值，然后执行 `resolve/promise(promise2, x.value, promise1.resolve, promise1.reject)`
 • 在内部实现中我们了 `promise1.then` 函数 `resolve => { promise1.resolve, ... } promise1.resolve`，然后传入的 `onFulfilled(promise1.value)` 则是对 `promise2` 的 `resolve/promise` 传入 `promise1` 的决议回调和 `onFulfilled` 回调
 • `promise2` 的决议
 • `promise1.then` 上，阻止了链式调用

2 链式调用需求


• `promise1 => promise2 => promise3`，因为 `promise1` 的决议 `promise2` 的决议回调函数执行

3 链式调用实现

因为 `promise1.then` 传入的 `onFulfilled` 不是一个函数，此时 `onFulfilled` 会强制写为 `val => val`

4 promise 代码实现

```
// Promise 三种状态
const PENDING = "Pending";
const FULFILLED = "Fulfilled";
const REJECTED = "Rejected";
// promise 内部状态
function promiseResolutionProcedure(promise2, x, resolve, reject) {
  // 判断是否已经决议
  if (promise2 === x) {
    throw new Error("循环引用 promise");
  }
  // 处理 promise 对象
  if (x instanceof MyPromise) {
    if (x.state === PENDING) {
      x.then(y => {
        promiseResolutionProcedure(promise2, y, resolve, reject);
      }, reject);
    } else {
      x.state === FULFILLED ? resolve(x.value) :
      x.state === REJECTED ? reject(x.value);
    }
  }
  // 处理 thenable 对象
  if (typeof x === "object" || typeof x === "function" || x == null) {
    if (typeof x.then === "function") {
      x.then(y => {
        promiseResolutionProcedure(promise2, y, resolve, reject);
      }, reject);
    } else {
      resolve(x);
    }
  } else {
    reject(x);
  }
}
class MyPromise {
  static all(promiseArray) {
    return new MyPromise((resolve, reject) => {
      let resultArray = [];
      let successTimes = 0;
      function processResult(index, data) {
        resultArray[index] = data;
        successTimes++;
        if (successTimes === promiseArray.length) {
          // 全部成功
          resolve(resultArray);
        }
      }
      for (let i = 0; i < promiseArray.length; i++) {
        promiseArray[i].then(
          (data) => {
            processResult(i, data);
          },
          (err) => {
            // 任意失败
            reject(err);
          }
        );
      }
    });
  }
  constructor(fn) {
    this.state = PENDING;
    this.value = undefined;
    this.resolveCallbacks = [];
    this.rejectCallbacks = [];
    const resolve = (val) => {
      if (
        (typeof val === "object" || typeof val === "function") &&
        val.then
      ) {
        promiseResolutionProcedure(this, val, resolve, reject);
        return;
      }
      setImmediate(() => {
        if (this.state === PENDING) {
          this.state = FULFILLED;
          this.value = val;
          // 所有回调函数 全部
          this.resolveCallbacks.map(fn => fn());
        }
      });
    };
    const reject = (val) => {
      if (
        (typeof val === "object" || typeof val === "function") &&
        val.then
      ) {
        promiseResolutionProcedure(this, val, resolve, reject);
        return;
      }
      setImmediate(() => {
        if (this.state === PENDING) {
          this.state = REJECTED;
          this.value = val;
          // 所有回调函数 全部
          this.rejectCallbacks.map(fn => fn());
        }
      });
    };
    fn(resolve, reject);
  }
  then(
    onFulfilled = (val) => val,
    onRejected = (err) => {
      throw new Error(err);
    }
  ) {
    let promise2 = null;
    // 当前已经决议的 promise
    if (this.state === FULFILLED) {
      promise2 = new MyPromise((resolve, reject) => {
        const x = onFulfilled(this.value);
        promiseResolutionProcedure(promise2, x, resolve, reject);
      });
    }
    // 当前未决议的 promise
    if (this.state === REJECTED) {
      promise2 = new MyPromise((resolve, reject) => {
        const x = onRejected(this.value);
        promiseResolutionProcedure(promise2, x, resolve, reject);
      });
    }
    // 当前未决议的 promise
    if (this.state === PENDING) {
      promise2 = new MyPromise((resolve, reject) => {
        this.resolveCallbacks.push(() => {
          const x = onFulfilled(this.value);
          promiseResolutionProcedure(promise2, x, resolve, reject);
        });
        this.rejectCallbacks.push(() => {
          const x = onRejected(this.value);
          promiseResolutionProcedure(promise2, x, resolve, reject);
        });
      });
    }
    return promise2;
  }
}
```


 京程一灯

一灯大家庭 面试题库 面试经验 藏经阁 内推资源

上一题 下一题

将 153812.7 转化为 153,812.7

代码实现

1.1 正则版

1.2 Api版: toLocaleString

1.3 Api版: Intl对象

将 153812.7 转化为 153,812.7

代码实现

1.1 正则版

```
function formatNumber(num) {  
  /*  
   ①/\b(?=(\d{3})+(?!\d))/g: 正则匹配非单词边界\b, 即除了\之前的位置, 其他字符之间的边界, 后面必须跟着3个数字直到字符串末尾  
   ②(\d{3})+: 必须是1个或多个的3个连续数字;  
   ③(?!\d): 第2步中的3个数字不允许后面跟着数字;  
  */  
  return (num+'').replace(/\b(?=(\d{3})+(?!\d))/g, ',');  
}
```

1.2 Api版: toLocaleString

- 优点: 简单粗暴, 直接调用 API
- 缺点: Intl兼容性不太好, 不过 toLocaleString 的话, IE6 都支持

方法返回这个数字在特定语言环境下的表示字符串, 具体可看MDN描述

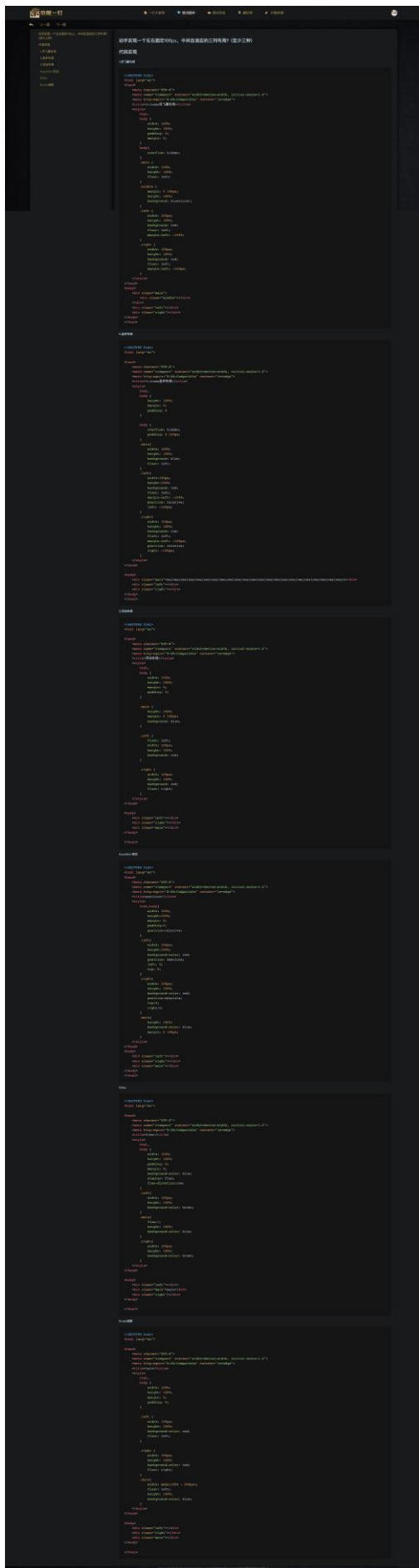
```
function formatNumber(num){  
  return num.toLocaleString('en-US');  
}
```


1.3 Api版: Intl对象

Intl 对象是 ECMAScript 国际化 API 的一个命名空间, 它提供了精确的字符串对比, 数字格式化, 日期和时间格式化。Collator, NumberFormat 和 DateTimeFormat 对象的构造函数是 Intl 对象的属性。

```
function formatNumber(num){  
  return new Intl.NumberFormat().format(num);  
}
```

Copyright © 2016 yidengqutang.com All Rights Reserved 京ICP备16022342号-1





一灯大家园

面试题库

面试经验

藏经阁

内推资源

上一题

下一题

请给出识别 Email 的正则表达式

代码实现

1.常规方案

2.安全方案

请给出识别 Email 的正则表达式

代码实现

1.常规方案

2.安全方案

1) 规则定义

- 以大写字母 [A-Z]、小写字母 [a-z]、数字 [0-9]、下划线 [_]、减号 [-] 及点号 [.] 开头，并需要重复一次至多次 [+]。
- 中间必须包含 @ 符号。
- @ 之后需要连接大写字母 [A-Z]、小写字母 [a-z]、数字 [0-9]、下划线 [_]、减号 [-] 及点号 [.]，并需要重复一次至多次 [+]。
- 结尾必须是点号 [.] 连接2至4位的大小写字母 [A-Za-z]{2,4}。

2) 代码

```
var pattern = /^[A-Za-z0-9_\-\.]+\@([A-Za-z0-9_\-\.]+\.[A-Za-z]{2,4})$/;

pattern.test('cn42du@163.com') //true;
pattern.test('ifat3@sina.com.cn') // true;
pattern.test('ifat3.it@163.com') // true;
pattern.test('ifat3_--@42du.cn') // true;
pattern.test('ifat3@42du.online') // false;
pattern.test('邮箱@42du.cn') // false;
```

3) 说明

这是最常用的邮件正则表达式验证方案，也适合大多数的应用场景。从以上测试可以看出，该表达式不支持 .online 及 .store 结尾的域名。如需兼容这类域名（大于4位），调整正则结尾 {2,4} 的限制部分即可（例：{2,8}）。另一个问题是邮件用户名不能包括中文。

4) 添加限制

- 用户名可以包括中文[A-Za-z0-9_\-\.u4e00-u9fa5]
- 域名结尾最长可为8位以右

```
var pattern = /^[A-Za-z0-9_\-\.u4e00-u9fa5]+\@([A-Za-z0-9_\-\.]+\.[A-Za-z]{2,8})$/;

pattern.test('cn42du@163.com') //true;
pattern.test('ifat3@sina.com.cn') // true;
pattern.test('ifat3.it@163.com') // true;
pattern.test('ifat3_--@42du.cn') // true;
pattern.test('ifat3@42du.online') // true;
pattern.test('邮箱@42du.cn') // true;
```

2.安全方案

在手机验证码出现之前，差不多邮箱验证是保证用户唯一性的唯一条件。而临时邮箱（也称10分钟邮箱或一次性邮箱）的出现，则使得邮箱验证及账户激活这种机制失去了意义。而临时邮箱的地址是不可枚举的，我们只能才采用白名单的方式，只允许有限的邮箱域名通过验证。

1) 继续添加限制

- 邮箱域名只能是163.com、qq.com或者42du.cn。

```
var pattern = /^[A-Za-z0-9_\-\.]+\@([163.com|qq.com|42du.cn])$/;
```



2) 说明

这种方式虽然能保证安全性，但是如果白名单太长会造成模式字符串太长。这时可以将邮箱域名白名单写成数组，利用正则表达式做初步验证，用白名单做域名的二次验证。

3) 代码改善

```
var isEmail = function (val) {
    var pattern = /^[A-Za-z0-9_\-\.]+\@([A-Za-z0-9_\-\.]+\.[A-Za-z]{2,4})$/;
    var domains = ['qq.com','163.com','vip.163.com','263.net','yeah.net','sohu.com','sina.cn','sina.com','eyou.com','gmail.com','hotmail.com'];
    if(pattern.test(val)) {
        var domain = val.substring(val.indexOf("@")+1);
        for(var i = 0; i < domains.length; i++) {
            if(domain == domains[i]) {
                return true;
            }
        }
    }
    return false;
}
// 输出 true
isEmail('cn42du@163.com');
```

Copyright © 2016 yskingyideng.com All Rights Reserved. 京ICP备16022242号-1

一灯大家庭面试题库面试经验收藏夹内推资源

[← 上一题](#)[下一题 →](#)

手写实现 sleep 函数

代码实现

- Promise 实现

```
const sleep = time => {  
  return new Promise(resolve => setTimeout(resolve, time))  
}  
sleep(1000).then(() => {  
  console.log(1)  
})
```

- Generator 实现

```
function* sleepGenerator(time) {  
  yield new Promise(function(resolve, reject) {  
    setTimeout(resolve, time);  
  })  
}  
sleepGenerator(1000).next().value.then(() => {console.log(1)})
```

- async 实现

```
function sleep(time) {  
  return new Promise(resolve => setTimeout(resolve, time))  
}  
async function output() {  
  let out = await sleep(1000);  
  console.log(1);  
  return out;  
}  
output();
```

- ES5 实现

```
function sleep(callback, time) {  
  if(typeof callback === 'function'){  
    setTimeout(callback, time)  
  }  
}  
  
function output(){  
  console.log(1);  
}  
sleep(output, 1000);
```

Copyright © 2016 yidengguang.com All Rights Reserved.京ICP备16022242号-1

10

Css 画一个三角形
代码实现

Css 画一个三角形
代码实现
· 利用border

```
/** 正三角 */
.triangle {
  width: 0;
  height: 0;
  border-style: solid;
  border-width: 0 25px 40px 25px;
  border-color: transparent transparent rgb(245, 129, 127) transparent;
}

/** 倒三角 */
.triangle {
  width: 0;
  height: 0;
  border-style: solid;
  border-width: 40px 25px 0 25px;
  border-color: rgb(245, 129, 127) transparent transparent transparent;
}
```

11

用 Promise 封装一个 ajax
代码实现



用 Promise 封装一个 ajax
代码实现

```
const promiseAjax = function(data){
  function formatParams(param) {
    var arr = [];
    for (var name in param) {
      arr.push(encodeURIComponent(name) + "=" + encodeURIComponent(param[name]));
    }
    arr.push(("v=" + Math.random()).replace(".", ""));
    return arr.join("&");
  }
  if (!data) data = {}
  data.params= data.params || {}
  return new Promise((resolve, reject) => {
    const xhr = new XMLHttpRequest();

    if (data.type === 'get') {
      data.params = formatParams(data.params); //options.data请求的数据
      xhr.open("GET", data.url + "?" + data.params, true);
      xhr.send(null);
    } else if (options.type === "post") {
      xhr.open("POST", data.url, true);
      xhr.setRequestHeader("Content-type", "application/json");
      xhr.send(data.params);
    }

    xhr.onreadystatechange = function () {
      if (xhr.readyState === 4) {
        if (xhr.status === 200) {
          resolve(xhr.response)
        } else {
          reject(xhr.responseText);
        }
      }
    }
  })
}
```

Copyright © 2018 gduanquang.com All Rights Reserved. 京ICP备16022047号-1

一灯大家庭面试题库面试经验面试指南内推资源

[← 上一题](#)[下一题 →](#)

实现以下代码

代码实现

```
function add() {  
  // your code  
}  
function one() {  
  // your code  
}  
function two() {  
  // your code  
}  
console.log(add(one(two()))); //3  
console.log(add(two(one()))); //3
```

实现以下代码

代码实现

```
function add() {  
  // your code  
  return arguments[0].reduce((a,b)=>a+b)  
}  
function one() {  
  // your code  
  if(arguments.length==0){  
    return 1  
  }else{  
    return [arguments[0],1]  
  }  
}  
function two() {  
  if(arguments.length==0){  
    return 2  
  }else{  
    return [arguments[0],2]  
  }  
}  
  
console.log(add(one(two()))); //3  
console.log(add(two(one()))); //3
```

Copyright © 2018 jobinterviewing.com All Rights Reserved. 面试题(1025) 4/5

京程

一灯大家庭 面试题库 面试经验 面试指南 内推资源

上一题 下一题

实现函数接受任意二叉树，求二叉树所有根到叶子路径组成的数字之和

代码实现

实现函数接受任意二叉树，求二叉树所有根到叶子路径组成的数字之和

```
class TreeNode {
  value: number
  left?: TreeNode
  right?: TreeNode
}

function getPathSum(root) {
  // your code
}

// 例子，一棵二叉树如下定义，路径包括: 1 -> 2, 1 -> 3
const node = new TreeNode();
node.value = 1;
node.left = new TreeNode();
node.left.value = 2;
node.right = new TreeNode();
node.right.value = 3;
getPathSum(node); // return 7 = (1+2) + (1+3)
```

代码实现

```
function getPathSum(root) {
  let num = 0;

  function addNum(node) {
    if (node.left) {
      num += (node.value + node.left.value)
      if (node.left.left) {
        addNum(node.left)
      } else if (node.left.right) {
        addNum(node.left)
      }
    }
    if (node.right) {
      num += (node.value + node.right.value)
      if (node.right.left) {
        addNum(node.right)
      } else if (node.right.right) {
        addNum(node.right)
      }
    }
  }

  addNum(node)
  console.log(num)
}

class TreeNode {
}

const node = new TreeNode();
node.value = 1;
node.left = new TreeNode();
node.left.value = 2;
node.right = new TreeNode();
node.right.value = 3;
// 下面数据便于测试使用 结果: 33
node.left.left = new TreeNode();
node.left.left.value = 4
node.left.right = new TreeNode();
node.left.right.value = 5
node.right.left = new TreeNode();
node.right.left.value = 6
node.right.right = new TreeNode();
node.right.right.value = 7
```

Copyright © 2016 jidongquansheng.com All Rights Reserved. 京程 16022014 版 - 4

14

请实现\$on,\$emit
代码实现

请实现 \$on,\$emit
代码实现

```
interface CacheProps {
  [key: string]: Array<({data?: unknown} => void)>;
}

class Observer {

  private caches: CacheProps = {}; // 事件中心

  on (eventName: string, fn: (data?: unknown) => void){ // eventName事件名-独一无二, fn订阅后执行的自定义行为
    this.caches[eventName] = this.caches[eventName] || [];
    this.caches[eventName].push(fn);
  }

  emit (eventName: string, data?: unknown) { // 发布 => 将订阅的事件进行统一执行
    if (this.caches[eventName]) {
      this.caches[eventName].forEach((fn: (data?: unknown) => void) => fn(data));
    }
  }

  off (eventName: string, fn?: (data?: unknown) => void) { // 取消订阅 => 若fn不传, 直接取消该事件所有订阅信息
    if (this.caches[eventName]) {
      const newCaches = fn ? this.caches[eventName].filter(e => e !== fn) : [];
      this.caches[eventName] = newCaches;
    }
  }
}
```

京程一灯

一灯大家庭 面试题库 面试经验 面试指南 内推资源

上一题 下一题

实现一个功能，发送请求 5s 时间后，如果没有数据返回，中断请求，提示错误。
中断请求，提示错误

代码实现

实现一个功能，发送请求 5s 时间后，如果没有数据返回，中断请求，提示错误

代码实现

调用XMLHttpRequest.abort(), 可以在请求发出后，立刻中止请求。
再设置一个定时器，定时计时完成后，如果没有数据返回则中断请求。
简单实现

```
/**
 * @param (Object) params
 */
const request = (params) => {
  const option = {
    timeout: 5000,
    ...params
  }
  return new Promise((resolve, reject) => {
    const xhr = new XMLHttpRequest()
    let isTimeout = false
    const timer = setTimeout(function () {
      isTimeout = true;
      xhr.abort();
      reject('request is timeout !!!')
    }, option.timeout)
    xhr.open('GET', option.url);
    xhr.onreadystatechange = function () {
      if (xhr.readyState === 4) {
        if (isTimeout) return; // 定时器中止请求
        clearTimeout(timer); // 取消等待的超时
        if ((xhr.status >= 200 && xhr.status < 300) || xhr.status === 304) {
          resolve(xhr.responseText)
        } else {
          reject('Request was unsuccessful !!! ${xhr.status}')
        }
      }
    }
    // 可以根据不同的请求方法发送数据
    xhr.send(null);
  })
}
```

调用:

```
const getData = async () => {
  const opt = {
    url: 'http://localhost:3000/timeout'
  }
  try {
    const res = await request(opt)
  } catch (error) {
    console.log(error) // request is timeout !!!
  }
}

getData()
```

Copyright © 2018 jingchengyideng.com All Rights Reserved. 京程一灯 1602704386-1