JavaScript 押题手写篇

手写节流 throttle、防抖 debounce

记忆题，写博客，甩链接。

节流：

```
// 节流就是「技能冷却中」
const throttle = (fn, time) => {
  let 冷却中 = false
  return (...args) => {
    if(冷却中) return
    fn.call(undefined, ...args)
    冷却中 = true
    setTimeout(()=>{
      冷却中 = false
    }, time)
  }
}
// 还有一个版本是在冷却结束时调用 fn
// 简洁版，删掉冷却中变量，直接使用 timer 代替
const throttle = (f, time) => {
  let timer = null
  return (...args) => {
    if(timer) {return}
    f.call(undefined, ...args)
    timer = setTimeout(()=>{
      timer = null
    }, time)
  }
}
```

使用方法：

```
const f = throttle(()=>{console.log('hi')}, 3000)

f() // 打印 hi
f() // 技能冷却中
```

防抖：

```
// 防抖就是「回城被打断」
const debounce = (fn, time) => {
  let 回城计时器 = null
  return (...args)=>{
    if(回城计时器 !== null) {
      clearTimeout(回城计时器) // 打断回城
    }
    // 重新回城
    回城计时器 = setTimeout(()=>{
      fn.call(undefined, ...args) // 回城后调用 fn
      回城计时器 = null
    }, time)
  }
}
```

手写发布订阅

记忆题，写博客，甩链接

```
const eventHub = {
  map: {
    // click: [f1 , f2]
  },
  on: (name, fn)=>{
    eventHub.map[name] = eventHub.map[name] || []
    eventHub.map[name].push(fn)
  },
  emit: (name, data)=>{
    const q = eventHub.map[name]
    if(!q) return
    q.map(f => f.call(null, data))
    return undefined
  },
  off: (name, fn)=>{
    const q = eventHub.map[name]
    if(!q){ return }
    const index = q.indexOf(fn)
    if(index < 0) { return }
    q.splice(index, 1)
  }
}

eventHub.on('click', console.log)
eventHub.on('click', console.error)

setTimeout(()=>{
  eventHub.emit('click', 'frank')
},3000)
```

也可以用 class 实现。

```
class EventHub {
  map = {}
  on(name, fn) {
    this.map[name] = this.map[name] || []
    this.map[name].push(fn)
  }
  emit(name, data) {
    const fnList = this.map[name] || []
    fnList.forEach(fn => fn.call(undefined, data))
  }
  off(name, fn) {
    const fnList = this.map[name] || []
    const index = fnList.indexOf(fn)
    if(index < 0) return
    fnList.splice(index, 1)
  }
}
// 使用
const e = new EventHub()
e.on('click', (name)=>{
  console.log('hi '+ name)
})
e.on('click', (name)=>{
  console.log('hello '+ name)
})
setTimeout(()=>{
  e.emit('click', 'frank')
},3000)
```

手写 AJAX

记忆题，写博客吧

```
const ajax = (method, url, data, success, fail) => {
  var request = new XMLHttpRequest()
  request.open(method, url);
  request.onreadystatechange = function () {
    if(request.readyState === 4) {
      if(request.status >= 200 && request.status < 300 || request.status ===
        success(request)
      }else{
        fail(request)
      }
    }
  };
  request.send();
}
```

手写简化版 Promise

记忆题，写博客吧

```
class Promise2 {
  #status = 'pending'
  constructor(fn){
    this.q = []
    const resolve = (data)=>{
      this.#status = 'fulfilled'
      const f1f2 = this.q.shift()
      if(!f1f2 || !f1f2[0]) return
      const x = f1f2[0].call(undefined, data)
      if(x instanceof Promise2) {
        x.then((data)=>{
          resolve(data)
        }, (reason)=>{
          reject(reason)
        })
      }else {
        resolve(x)
      }
    }
    const reject = (reason)=>{
      this.#status = 'rejected'
      const f1f2 = this.q.shift()
      if(!f1f2 || !f1f2[1]) return
      const x = f1f2[1].call(undefined, reason)
      if(x instanceof Promise2){
        x.then((data)=>{
          resolve(data)
        }, (reason)=>{
          reject(reason)
        })
      }else{
        resolve(x)
      }
    }
    fn.call(undefined, resolve, reject)
  }
  then(f1, f2){
    this.q.push([f1, f2])
  }
}

const p = new Promise2(function(resolve, reject){
  setTimeout(function(){
    reject('出错')
  },3000)
```

```
})

p.then( (data)=>{console.log(data)}, (r)=>{console.error(r)} )
```

手写 Promise.all

记忆题，写博客吧。

要点：

1. 知道要在 Promise 上写而不是在原型上写
2. 知道 all 的参数（Promise 数组）和返回值（新 Promise 对象）
3. 知道用数组来记录结果
4. 知道只要有一个 reject 就整体 reject

```
Promise.prototype.myAll
Promise.myAll = function(list){
  const results = []
  let count = 0
  return new Promise((resolve,reject) =>{
    list.map((item, index)=> {
      item.then(result=>{
          results[index] = result
          count += 1
          if (count >= list.length) { resolve(results)}
      }, reason => reject(reason) )
    })
  })
}
```

进一步提问：是否知道 Promise.allSettled()

手写深拷贝

方法一，用 JSON：

```
const b = JSON.parse(JSON.stringify(a))
```

答题要点是指出这个方法有如下缺点：

1. 不支持 Date、正则、undefined、函数等数据
2. 不支持引用（即环状结构）
3. 必须说自己还会方法二

方法二，用递归：

要点：

1. 递归
2. 判断类型
3. 检查环
4. 不拷贝原型上的属性

```
const deepClone = (a, cache) => {
  if(!cache){
    cache = new Map() // 缓存不能全局，最好临时创建并递归传递
  }
  if(a instanceof Object) { // 不考虑跨 iframe
    if(cache.get(a)) { return cache.get(a) }
    let result
    if(a instanceof Function) {
      if(a.prototype) { // 有 prototype 就是普通函数
        result = function(){ return a.apply(this, arguments) }
      } else {
        result = (...args) => { return a.call(undefined, ...args) }
      }
    } else if(a instanceof Array) {
      result = []
    } else if(a instanceof Date) {
      result = new Date(a - 0)
    } else if(a instanceof RegExp) {
      result = new RegExp(a.source, a.flags)
    } else {
      result = {}
    }
    cache.set(a, result)
    for(let key in a) {
      if(a.hasOwnProperty(key)){
        result[key] = deepClone(a[key], cache)
      }
    }
    return result
  } else {
    return a
  }
}

const a = {
  number:1, bool:false, str: 'hi', empty1: undefined, empty2: null,
  array: [
    {name: 'frank', age: 18},
    {name: 'jacky', age: 19}
  ],
  date: new Date(2000,0,1,20,30,0),
  regex: /\.(j|t)sx/i,
  obj: { name:'frank', age: 18},
  f1: (a, b) => a + b,
  f2: function(a, b) { return a + b }
```

```
}
a.self = a

const b = deepClone(a)

b.self === b // true
b.self = 'hi'
a.self !== 'hi' //true
```

手写数组去重

1. 使用计数排序的思路，缺点是只支持字符串
2. 使用 Set（面试已经禁止这种了，因为太简单）
3. 使用 Map，缺点是兼容性差了一点

```
var uniq = function(a){
  var map = new Map()
  for(let i=0;i<a.length;i++){
    let number = a[i] // 1 ~ 3
    if(number === undefined){continue}
    if(map.has(number)){
      continue
    }
    map.set(number, true)

  }
  return [...map.keys()]
}
```