

# 京城一灯-VIP 编程题

## 第五页

### 1

versions 是一个项目的版本号列表，因多人维护，不规则，  
动手实现一个版本号处理函数

代码实现

versions 是一个项目的版本号列表，因多人维护，不规则，动手实现一个版本号处理函数

```
var versions = ["1.45.0", "1.5", "6", "3.3.3.3.3.3.3"];  
// 要求从小到大排序，注意'1.45'比'1.5'大  
function sortVersion(versions) {  
  // TODO  
}  
// => ['1.5', '1.45.0', '3.3.3.3.3.3', '6']
```

代码实现

```
function sortVersion(list){  
  return list.sort((a, b) => {  
    let aa = a.split('.')  
    let bb = b.split('.')  
    let length = aa.length>bb.length?aa.length:bb.length  
    for (var i=0; i < length; i++){  
      let x = aa[i] || 0;  
      let y = bb[i] || 0;  
      if(x-y !==0 )return x - y;  
    }  
  });  
}  
sortVersion(['1.0.0', '1.2.3.4.5', '2.12.1', '0.18.1', '3.3.2', '0.18.1'])
```

[illegible]

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

动手实现一个 repeat 方法

代码实现

一、实现方式一

二、实现方式二

动手实现一个 repeat 方法

```
function repeat(func, times, wait) {
  // 2000
}
const repeatFunc = repeat(alert, 4, 3000);
// 调用这个 repeatFunc ("helloworld"), 会alert4次 helloworld, 每次间隔3秒
```

代码实现



一、实现方式一

```
function repeat (func, times, wait) {
  if (typeof func !== 'function') return;
  if (times <= 0) return;
  return function (value) {
    let timesTmp = times
    let interval = setInterval(function(){
      func(value);
      timesTmp--;
      timesTmp === 0 && clearInterval(interval)
    }, wait)
  }
}
const repeatConsole = repeat(console.log, 4, 2000)
repeatConsole('helllww')
```

二、实现方式二

```
function repeat (func, times, wait) {
  return function (...args){
    let i = 0;
    let _args = [...args]
    let handle = setInterval(() => {
      i ++;
      if(i > times){
        clearInterval(handle);
        return;
      }
      func.apply(null, _args);
    }, wait);
  }
}
```

Copyright © 2018 jingchengyideng.com All Rights Reserved 京CP备1602342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 →](#)

使用 Css 实现一个水波纹效果

代码实现

```
<style>
  .wave-content {
    height: 666px;
    width: 666px;
    left: 255px;
    top: 139px;
    position: relative;
  }

  .wave {
    position: absolute;
    left: 0px;
    top: 0px;
    height: 100%;
    width: 100%;
    transform-origin: center center;
    background-color: transparent;
    border: 1px solid #979797;
    animation-duration: 7s;
    animation-name: vw;
    animation-timing-function: linear;
    animation-iteration-count: infinite;
    border-radius: 50%;
    opacity: 0;
  }

  .wave1 {
    animation-delay: 0s;
  }

  .wave2 {
    animation-delay: 1.5s;
  }

  .wave3 {
    animation-delay: 3s;
  }

  .wave4 {
    animation-delay: 4.5s;
  }



  @keyframes vw {
    0% {
      opacity: 0;
      transform: scale(0.5);
    }

    30% {
      opacity: 0.7;
      transform: scale(0.65);
    }

    70% {
      opacity: 0.1;
      transform: scale(0.85);
    }

    100% {
      opacity: -0.2;
      transform: scale(1);
    }
  }
</style>
<div class="wave-content">
  <div class="wave wave1"></div>
  <div class="wave wave2"></div>
  <div class="wave wave3"></div>
  <div class="wave wave4"></div>
</div>
```

Copyright © 2016 yidengguang.com All Rights Reserved 京ICP备16022243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 →

请写一个函数，输出出多级嵌套结构的 Object 的所有 key 值

代码实现

请写一个函数，输出出多级嵌套结构的 Object 的所有 key 值

```
var obj = {
  a: '12',
  b: '23',
  first: {
    c: '34',
    d: '45',
    second: { j: '56', f: '67', three: { q: '78', h: '89', i: '90' } },
  },
};
// => [a,b,c,d,e,f,g,h,i]
```

代码实现

```
function getAllKey(obj) {
  if (typeof obj !== 'object') {
    return
  }
  let keys = []

  for (let index in obj) {
    if (obj[index] instanceof Object && !Array.isArray(obj[index])) {
      keys = keys.concat(getAllKey(obj[index]))
    } else {
      keys.push(index)
    }
  }
  return keys
}
getAllKey(obj)
```

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

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

按要求实现一个 sum 函数

参考代码实现

### 按要求实现一个 sum 函数

```
const a = sum(0); // => a === 0
const b = sum(1)(2); // => b === 3
const c = sum(4)(5); // a === 9
const k = sum(n1)...(nk) // k === n1 + n2 + ... + nk
```

### 参考代码实现

递归实现, 使用 toString 打印

思路: 当我们直接对函数使用 alert() 或 console.log() 时, 函数的 toString() 方法会被调用。注意, valueOf 方法会把数据类型转换成原始类型, toString 方法会把数据类型转换成 string 类型。如果是对对象返回, toString() 返回 "object type", 其中 type 是对象类型。正常情况下, 优先调用 toString(), 有运算符操作符的情况下, valueOf() 的优先级高于 toString(), 当调用 valueOf() 方法无法运算后还是会再调用 toString() 方法

```
function sum(a) {
  let temp = function (b) {
    return sum(a + b);
  };
  // temp.toString 这里就是 temp.valueOf 也可以
  temp.toString = function () {
    return a;
  };
  return temp;
}

let ans = sum(1)(2)(3);
console.log(ans);
```

函数柯里化

思路: 也是用到 toString 打印, 但是这里用到了函数式编程的思想, 这里的 sum(1)(2)(3)(4)...n 等价于 sum(1)(2,3)(4)...n, 也等价于 sum(1,2,3)(4)...n 等多种排列组合

```
let sum = 0;
function add(...args) {
  for (let i = 0; i < args.length; i++) {
    sum = sum + args[i];
  }
  return sum;
}

function currying(fn) {
  let val = null;
  let temp = function (...newArgs) {
    val = fn.apply(this, newArgs);
    return temp;
  };
  temp.toString = function () {
    sum = 0;
    return val;
  };
  return temp;
}

let addCurry = currying(add);
console.log(addCurry(1)(2)(3)(4, 5)); //15 这里是指数值为15, 本质是函数字符串串值
console.log(addCurry(1)(2)(3, 4, 5)); //15
console.log(addCurry(1, 2)(3, 4, 5)); //15
```

扩展: 求 sum(1)(2)(3)...n

```
// 递归
function sum(a) {
  return function (b) {
    if (b !== undefined) {
      return sum(a + b);
    } else {
      return a;
    }
  };
}

let ans = sum(1)(2)(3)();
console.log(ans);

// 柯里化
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);
// 注意调用方式的变化
console.log(addCurry(1)(2)(3)(4, 5)()); //15
console.log(addCurry(1)(2)(3, 4, 5)()); //15
console.log(addCurry(1)(2, 3, 4, 5)()); //15
```

Copyright © 2018 jikexing.com All Rights Reserved 京程一灯 102344号-1

给定起止日期，返回中间的所有月份

实现代码

给定起止日期，返回中间的所有月份

```
// 输入两个字符串 2018-08 2018-12  
// 输出他们中间的月份 [2018-9, 2018-10, 2018-11]
```


实现代码

```
const getMonths = (startDateStr, endDateStr) => {  
  let startTime = getDate(startDateStr).getTime()  
  const endTime = getDate(endDateStr).getTime()  
  const result = []  
  while (startTime < endTime) {  
    let curDate = new Date(startTime)  
    result.push(formatDate(curDate))  
    curDate.setMonth(curDate.getMonth() + 1)  
    startTime = curDate.getTime()  
  }  
  return result.slice(1)  
}  
  
const getDate = (dateStr) => {  
  const [year, month] = dateStr.split('-')  
  return new Date(year, month - 1)  
}  
  
const formatDate = (date) => {  
  return `${date.getFullYear()}-${(date.getMonth()+1).toString().padStart(2, '0')}`  
}  
  
console.log(getMonths('2018-08', '2018-12'))
```








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 →

请实现如下的函数

请实现如下的函数

```

/*
 * 可以批量请求数据，所有的 url 地址在 urls 参数中，
 * 同时可以通过 max 参数控制请求的并发数，当所有请
 * 求结束之后，需要执行 callback 回调函数，发请求的
 * 函数可以直接使用 fetch 即可
 */

```

**1) 思路**

1 fetch 请求介绍 2 开发，开发，某个完成后可以继续发请求 3 所有请求结束后 callback 4 注意细节 ①参数错误，参数不正确，接口地址不正确，最大数不正确，回调函数不正确；②接口错误 5 边界值 6 不能在 while 中使用 fetch，因为 while 是同步，它永远不会等待异步的 fetch 结果回来 7 使用 for 循环 + 递归的方法

**2) 代码实现**

```

/**
 * @param { Array } urls 请求地址数组
 * @param { Number } max 最大并发请求数
 * @param { Function } callback 回调函数
 */
function parallelFetch(urls, max, callback) {
  // 如果当前环境不支持 fetch，则使用 XMLHttpRequest 进行
  if (!window.fetch || "function" !== typeof window.fetch) {
    throw Error("当前环境不支持 fetch 请求，程序终止");
  }

  // 如果参数有误，则提示输入正确的参数
  if (!urls || urls.length <= 0) {
    throw Error("urls is empty: 请传入正确的请求地址");
  }

  const _urlLength = urls.length; // 请求地址数组的长度
  const _max = max || 1; // 保证最大并发数的有效性
  let _currentIndex = 0; // 当前请求地址的索引
  let _maxFetch = max <= _urlLength ? max : _urlLength; // 当前可以正常请求的数量，保证最大并发数的安全性
  let _finishedFetch = 0; // 当前完成请求的数量，用于判断何时调用回调

  console.log(`开始并发请求，接口总数为 ${_urlLength}，最大并发数为 ${_maxFetch}`);

  // 根据最大并发数进行循环请求，之后通过 Promise 进行请求
  for (let i = 0; i < _maxFetch; i++) {
    fetchFunc();
  }

  // 请求方法
  function fetchFunc() {
    // 如果所有请求都已完成，则执行回调方法
    if (_finishedFetch === _urlLength) {
      console.log(`当前一共 ${_urlLength} 个请求，已完成 ${_finishedFetch} 个`);
      if ("function" === typeof callback) return callback();
      return false;
    }

    // 如果当前请求的索引大于等于请求地址数组的长度，则不继续请求
    if (_currentIndex >= _urlLength) {
      _maxFetch = 0;
    }

    // 如果可请求的数量大于0，表示可以继续发起请求
    if (_maxFetch > 0) {
      console.log(`当前正发起第 ${_currentIndex + 1} 次请求，当前一共 ${_urlLength} 个请求，已完成 ${_finishedFetch} 个，请求地址为: ${urls[_currentIndex]}`);
      // 发起 fetch 请求
      fetch(urls[_currentIndex])
        .then((res) => {
          // todo 业务逻辑，正常的逻辑，异常的逻辑
          // 当前请求结束，正常请求的数量 +1
          _maxFetch += 1;
          _finishedFetch += 1;
          fetchFunc();
        })
        .catch((err) => {
          // todo 异常处理，处理异常逻辑
          // 当前请求结束，正常请求的数量 +1
          _maxFetch += 1;
          _finishedFetch += 1;
          fetchFunc();
        });
      // 每次请求，当前请求地址的索引 +1
      _currentIndex += 1;
      // 每次请求，可以正常请求的数量 -1
      _maxFetch -= 1;
    }
  }


  let urls = [1];
  for (let i = 0; i < 100; i++) {
    urls.push(`https://jsonplaceholder.typicode.com/todos/${i+1}`);
  }

  const max = 10;
  const callback = () => {
    console.log("我请求完了");
  };

  parallelFetch(urls, max, callback);

```

Copyright © 2016 jingchengyideng.com All Rights Reserved. 京CP备16022242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

请实现一个 `cacheRequest` 方法，保证发出多次同一个 `ajax` 请求时都能拿到数据，而实际上只发出一次请求

代码实现

- `cache` 构建 `Map`，用作缓存数据，把 `URL` 作为 `key`，用于判断是否同一个请求。
- 请求的更多参数可传 `Option`，如：`GET`、`data` 等。
- 每次请求检查缓存，有则返回缓存数据，无则发起请求。
- 请求成功时，保存数据到 `cache` 并返回，失败则弹出警告。
- 网络繁忙，阻塞请求在 `pending` 状态，防止阻塞请求造成死锁。
- 代码中 `ajax` 请求用 `setTimeout` 的函数代替，可自行封装 `request` - 函数或用 `axios` 代替。

请实现一个 `cacheRequest` 方法，保证发出多次同一个 `ajax` 请求时都能拿到数据，而实际上只发出一次请求

代码实现

```
const request = (url, option) => new Promise((res) => {
  setTimeout(() => {
    res({data: option})
  }, 2000)
})

const cache = new Map();
const cacheRequest = (url, option) => {
  let key = `${url}${option.method}`;
  if (cache.has(key)) {
    if (cache.get(key).status === 'pending') {
      return cache.get(key).myWait;
    }
    return Promise.resolve(cache.get(key).data);
  } else {
    // 无缓存，发起真实请求
    let requestApi = request(url, option);
    cache.set(key, {status: 'pending', myWait: requestApi});
    return requestApi.then(res => {
      // console.log(cache)
      cache.set(key, {status: 'success', data: res});
      // console.log(cache)
      return Promise.resolve(res);
    }).catch(err => {
      cache.set(key, {status: 'fail', data: err});
      Promise.reject(err);
    });
  }
}
```

调用

```
cacheRequest('url1')
  .then(res => console.log(res))
cacheRequest('url1')
  .then(res => console.log(res))

setTimeout(() => {
  cacheRequest('url1')
    .then(res => console.log(res))
  }, 4000)
```

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

# 11

实现 bind 方法，不能使用 call、apply、bind

bind

1.1 概念

1.2 特点

1.3 模拟实现

实现 bind 方法，不能使用 call、apply、bind

bind

1.1 概念

bind() 方法创建一个新的函数，在 bind() 被调用时，这个新函数的 this 被指定为 bind() 的第一个参数，而其余参数将作为新函数的参数，供调用时使用。

通俗一点，bind与apply/call一样都能改变函数this指向，但bind并不会立即执行函数，而是返回一个绑定了this的新函数，你需要再次调用此函数才能达到最终执行。


1.2 特点

- 可以修改函数this指向。
- bind返回一个绑定了this的新函数。
- 支持函数链式化。
- 新函数的this无法再被修改，使用call、apply也不行。

1.3 模拟实现

- 不准使用call、apply、bind 可以自己模拟一个apply

```
Function.prototype.myapply = function (context, ...args) {  
  if (typeof this !== 'function') {  
    throw new TypeError('not function')  
  }  
  const fn = this  
  let result = null  
  
  context = context || window  
  args = args && args[0] || []  
  context.fn = fn  
  result = context.fn(...args)  
  delete context.fn  
  
  return result  
}  
Function.prototype.mybind = function (context) {  
  var me = this;  
  var args = Array.prototype.slice.call(arguments, 1);  
  var F = function () {};  
  F.prototype = this.prototype;  
  var bound = function () {  
    var innerArgs = Array.prototype.slice.call(arguments);  
    var finalArgs = args.concat(innerArgs);  
    return me.myapply(this instanceof F ? this : context || this, finalArgs);  
  }  
  bound.prototype = new F();  
  return bound;  
}
```

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](#)

用原生 js 实现自定义事件

JS 自定义事件


JavaScript自定义事件类似设计的观察者模式。通过状态的变更来监听行为。主要功能解耦，易于扩展。多用于组件、模块间的交互。

原型模式下的自定义事件

```
var EventTarget = function() {
    this._listener = {};
};

EventTarget.prototype = {
    constructor: this,
    addEvent: function(type, fn) {
        if (typeof type === "string" && typeof fn === "function") {
            if (typeof this._listener[type] === "undefined") {
                this._listener[type] = [fn];
            } else {
                this._listener[type].push(fn);
            }
        }
        return this;
    },
    addEvents: function(obj) {
        obj = typeof obj === "object"? obj : {};
        var type;
        for (type in obj) {
            if (type && typeof obj[type] === "function") {
                this.addEvent(type, obj[type]);
            }
        }
        return this;
    },
    fireEvent: function(type) {
        if (type && this._listener[type]) {
            //event 参数设置
            var events = {
                type: type,
                target: this
            };
            for (var length = this._listener[type].length, start=0; start<length; start+=1) {
                //改变this指向
                this._listener[type][start].call(this, events);
            }
        }
        return this;
    },
    fireEvents: function(array) {
        if (array instanceof Array) {
            for (var i=0, length = array.length; i<length; i+=1) {
                this.fireEvent(array[i]);
            }
        }
        return this;
    },
    removeEvent: function(type, key) {
        var listeners = this._listener[type];
        if (listeners instanceof Array) {
            if (typeof key === "function") {
                for (var i=0, length=listeners.length; i<length; i+=1){
                    if (listeners[i] === key){
                        listeners.splice(i, 1);
                        break;
                    }
                }
            } else if (key instanceof Array) {
                for (var lis=0, lenkey = key.length; lis<lenkey; lis+=1) {
                    this.removeEvent(type, key[lenkey]);
                }
            } else {
                delete this._listener[type];
            }
        }
        return this;
    },
    removeEvents: function(params) {
        if (params instanceof Array) {
            for (var i=0, length = params.length; i<length; i+=1) {
                this.removeEvent(params[i]);
            }
        } else if (typeof params === "object") {
            for (var type in params) {
                this.removeEvent(type, params[type]);
            }
        }
        return this;
    }
};
```

Copyright © 2016 yidengpublog.com All Rights Reserved. ICP备 16022428-1

 京程一灯

一灯大家庭 面试题库 面试经验 藏经阁 内推资源

上一题 下一题

要求用不同方式对 A 进行改造实现 A.name 发生变化时立即执行 A.getName

代码实现

方法一

```
let _name = 'sfd'
A = {
  get name() {
    return _name
  },
  set name(name) {
    _name = name
    this.getName()
  },
  getName: function() {
    console.log(this.name)
  }
}
```

方法二

```
const A = {
  getName: function() {
    console.log(this.name)
  }
}
let _name = 'sfd'
Object.defineProperty(A, 'name', {
  enumerable: true,
  configurable: true,
  get() {
    return _name
  },
  set(name) {
    _name = name
    this.getName()
  }
})
```

方法三

```
const _A = {
  name: 'sfd',
  getName: function() {
    console.log(this.name)
  }
}
const A = new Proxy(_A, {
  get(target, key, receiver) {
    return Reflect.get(target, key, receiver)
  },
  set(target, key, value, receiver) {
    const res = Reflect.set(target, key, value, receiver)
    target.getName()
    return res
  }
})
```

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