1. **js 数据 reduce是什么?**

   reduce() 方法接收一个函数作为累加器，数组中的每个值（从左到右）开始缩减，最终计算为一个值。

   reduce() 可以作为一个高阶函数，用于函数的 compose。

```
var arr = [1, 2, 3, 4, 5]
sum = arr.reduce(function(prev, cur, index, arr) {
console.log(prev, cur, index)
return prev + cur
})
console.log(arr, sum)
//输出结果
1 2 1
3 3 2
6 4 3
10 5 4
[1, 2, 3, 4, 5] 15
```

   reduce回调函数中有4个参数:

   prev: 第一项的值或者上一次叠加的结果值 cur: 当前会参与叠加的项 index： 当前值的索引 arr: 数组本身

2. **js 字符串反转怎么做？join()方法后面跟参数吗？不跟参数默认是什么**

   split()方法将一个字符串对象的每个字符拆出来，并且将每个字符串当成数组的每个元素

   split() 方法用于把一个字符串分割成字符串数组。

   提示： 如果把空字符串 ("") 用作 separator，那么 stringObject 中的每个字符之间都会被分割。

   注意： split() 方法不改变原始字符串。

   string.split(*separator*,*limit*)

   separator 可选。字符串或正则表达式，从该参数指定的地方分割 string Object。

   limit 可选。该参数可指定返回的数组的最大长度。如果设置了该参数，返回的子串不会多于这个参数指定的数组。如果没有设置该参数，整个字符串都会被分割，不考虑它的长度。

   reverse()方法用来改变数组，将数组中的元素倒个序排列，第一个数组元素成为最后一个，最后一个变成第一个。

   reverse() 方法用于颠倒数组中元素的顺序。

   join()方法将数组中的所有元素边接成一个字符串

   join() 方法用于把数组中的所有元素转换一个字符串。

   array.join(*separator*)

   separator 可选。指定要使用的分隔符。如果省略该参数，则使用逗号作为分隔符。

   方法一:

```
function reverseString(str) { return str.split("").reverse().join(""); }
reverseString("hello"); // => olleh
```

方法二：

```
var str ='abcde'
var strarr = str.split('')
console.log(strarr) //["a", "b", "c", "d", "e"]
var arr = new Array
//while循环  只要条件为 true，循环能够一直执行代码块。
while(strarr.length) {
arr.push(strarr.pop())   //pop()  返回最后一个元素  这样循环添加到新的数组
}
console.log(arr.join(''))  //edcba  在这边单引号和双引号没区别
```

3. **es6中 let const 和 var区别?**

   使用var声明的变量，其作用域为该语句所在的函数内，且存在变量提升现象； 使用let声明的变量，其作用域为该语句所在的代码块内，不存在变量提升； 使用const声明的是常量，在后面出现的代码中不能再修改该常量的值。

4. **解释一下什么是事件冒泡和事件捕获？ 先后顺序是什么？ 可以在捕获得时候选择冒泡吗？ 事件捕获发生在哪一个阶段?**

   事件冒泡： 微软提出了名为`事件冒泡(event bubbling)`的事件流。事件冒泡可以形象地比喻为把一颗石头投入水中，泡泡会一直从水底冒出水面。也就是说，事件会从最内层的元素开始发生，一直向上传播，直到document对象。

   因此在事件冒泡的概念下在p元素上发生click事件的顺序应该是**p -> div -> body -> html -> document**

   事件捕获：网景提出另一种事件流名为`事件捕获(event capturing)`。与事件冒泡相反，事件会从最外层开始发生，直到最具体的元素。

   因此在事件捕获的概念下在p元素上发生cli。

   发生顺序是：**先捕获再冒泡**

5. **url？ 后面 怎么解析成对象**

```
    function GetRequest() {
        let url =
          'https://www.baidu.com/?
product=shirt&color=blue&newuser&size=m#Hello'
        const urlSearch = '?product=shirt&color=blue&newuser&size=m#Hello'
        let theRequest = new Object()
       //indexOf()方法返回调用它的 String中第一次出现的指定值的索引。如果未找到该值，
则返回 -1
        //判断urlSerach有没有?
        if (urlSearch.indexOf('?') != -1) {
          //substring() 方法用于提取字符串中介于两个指定下标之间的字符。
          let str = urlSearch.substring(1) //截取?后面的字符串
          // split() 方法用于把一个字符串分割成字符串数组。 如果是("")则每个字符都分割
          strs = str.split('&')
          for (let i = 0; i < strs.length; i++) {
              //decodeURI 解码 可能查询参数会被编码
            theRequest[strs[i].split('=')[0]] = decodeURI(strs[i].split('=')
[1])
          }
        }
        return theRequest
```

```
}
console.log(GetRequest().product)

//在最新浏览器中使用  获取search后的值
getSearch()
function getSearch() {
  const urlSearch = '?product=shirt&color=blue&newuser&size=m#Hello'
  // URLSearchParams返回一个 URLSearchParams 对象
  const urlParams = new URLSearchParams(urlSearch)
  // URLSearchParams.has() 返回 Boolean 判断是否存在此搜索参数
  console.log(urlParams.has('product')) // true
  // URLSearchParams.get()获取指定搜索参数的第一个值。
  console.log(urlParams.get('product')) //shirt
  console.log(urlParams.getAll('product')) //["shirt"]
}

getAllUrlParams()
function getAllUrlParams() {
  const url = '?product=shirt&color=blue&newuser&size=m#Hello'
  var queryString = url
    ? url.split('?')[1]
    : window.location.search.slice(1)
  // 用来存储我们所有的参数
  var obj = {}
  // 如果没有传参，返回一个空对象
  if (!queryString) {
    return obj
  }
  // stuff after # is not part of query string, so get rid of it
  queryString = queryString.split('#')[0]
  // 将参数分成数组
  var arr = queryString.split('&')
  for (var i = 0; i < arr.length; i++) {
    // 分离成key:value的形式
    var a = arr[i].split('=')
    // 将undefined标记为true
    var paramName = a[0]
    var paramValue = typeof a[1] === 'undefined' ? true : a[1]
    // 如果调用对象时要求大小写区分，可删除这两行代码
    paramName = paramName.toLowerCase()
    if (typeof paramValue === 'string')
      paramValue = paramValue.toLowerCase()
    // 如果paramName以方括号结束, e.g. colors[] or colors[2]
    if (paramName.match(/\[(\d+)?\]$/)) {
      // 如果paramName不存在，则创建key
      var key = paramName.replace(/\[(\d+)?\]/, '')
      if (!obj[key]) obj[key] = []
      // 如果是索引数组 e.g. colors[2]
      if (paramName.match(/\[\d+\]$/)) {
        // 获取索引值并在对应的位置添加值
        var index = /\[(\d+)\]/.exec(paramName)[1]
        obj[key][index] = paramValue
      } else {
        // 如果是其它的类型，也放到数组中
        obj[key].push(paramValue)
      }
    } else {
      // 处理字符串类型
```

```
            if (!obj[paramName]) {
              // 如果如果paramName不存在，则创建对象的属性
              obj[paramName] = paramValue
            } else if (obj[paramName] && typeof obj[paramName] === 'string')
{
              // 如果属性存在，并且是个字符串，那么就转换为数组
              obj[paramName] = [obj[paramName]]
              obj[paramName].push(paramValue)
            } else {
              // 如果是其它的类型，还是往数组里丢
              obj[paramName].push(paramValue)
            }
          }
        }
        return obj
      }
```

6. **pm2 怎么实现得多线程？实现原理是什么？**

   https://quincychen.cn/pm2-implementation/

   在如今机器的 CPU 都是多核的背景下，Node 的单线程设计已经没法更充分的" 压榨" 机器性能了。所以从 v0.8 开始，Node 新增了一个内置模块——“cluster”，故名思议，它可以通过一个父进程管理一坨子进程的方式来实现集群的功能。

   pm2基于 `cluster模块` 进行了封装，它能自动监控进程状态、重启进程、停止不稳定进程、日志存储等。利用 `pm2` 时，可以在不修改代码的情况下实现负载均衡集群。

7. **为什么采用websocket？ 要考虑优点得同时 也要考虑缺点？ 轮询和websocket 要多种考虑**

   节省每次请求的header http的header一般有几十字节 Server Push 服务器可以主动传送数据给客户端。

   WebSocket的优点是实现了双向通信，缺点是服务器端的逻辑非常复杂

8. **es6 中箭头函数和普通函数得区别？ 箭头函数中this得指向？ js 改变this指向得方法有哪些？ 有什么区别?**

   1. 语法更加简洁和清晰

      从上面的基本语法示例中可以看出，箭头函数的定义要比普通函数定义简洁、清晰得多，很快捷。

   2. 箭头函数不会创建自己的this！(重点理解)
   3. 箭头函数继承而来的this指向永远不会改变。
   4. .call/.apply()/.bind()方法可以用来动态修改函数执行时this的指向
   5. 箭头函数不能作为构造函数使用
   6. 箭头函数没有自己的arguments
   7. 箭头函数没有原型prototype
   8. 箭头函数不能用作Generator函数，不能使用yeild关键字

9. **js数组得队列？ 怎么出队 怎么入队?**

   队列，是一种特殊的列表，它与栈不同的是， 队列只能在队尾插入元素，在队首删除元素，就像我们平时排队买票一样。队列用于存储按顺序排列的数据，遵循 先进先出(FIFO，First-In-First-Out) 的原则，也是计算机常用的一种数据结构，别用于很多地方，比如提交给操作系统的一系列进程，打印池任务等。

同栈有点类似，队列的操作主要也是有两种：向队列中插入新元素和删除队列中的元素，即入队和出队操作，我们采用 enqueue 和 dequeue 两个方法。

```
enqueue：向队列添加元素
//向队列末尾添加一个元素，直接调用 push 方法即可

function enqueue ( element ) {
    this.dataStore.push( element );
}

dequeue：删除队首的元素
//删除队列首的元素，可以利用 JS 数组中的 shift 方法

function dequeue () {
    if( this.empty() ) return 'This queue is empty';
    else this.dataStore.shift();
}

empty：判断队列是否为空
//我们通过判断 dataStore 的长度就可知道队列是否为空

function empty(){
    if( this.dataStore.length == 0 ) return true;
    else return false;
}
```

10. **跨域是什么？ 怎么解决？ zhidao.baidu.com 和 baike.baidu.com 会发生跨域吗？ 跨域发生在服务器端还是浏览器端?**

    https://juejin.im/post/5c23993de51d457b8c1f4ee1

11. **跨域中 cors 怎么解决跨域？具体原理是什么？ 响应头和返回头 有什么区别?**