说说动画

直播间礼物动画，互动等业务

@keyframes是CSS3的一种规则，可以用来定义CSS动画的一个周期的行为，可以创建简单的动画。
CSS3动画（animation @keyframes）

```
.right_circle{
  box-sizing: border-box;
  border-top: 5px solid #53ECFF;
  border-right: 5px solid #53ECFF;
  right: 0;
  -webkit-animation: circle_right 30s linear;
  opacity: 0;
}
@-webkit-keyframes circle_right{
  0%{
    -webkit-transform: rotate(-135deg);
  }
  1% {
    opacity: 1;
  }
  50%,100%{
    -webkit-transform: rotate(45deg);
    opacity: 1;
  }
}
@-webkit-keyframes circle_left{
  0%,50%{
    -webkit-transform: rotate(-135deg);
  }
  51% {
    opacity: 1;
  }
  100%{
    -webkit-transform: rotate(45deg);
    opacity: 1;
  }
```

礼物动画：

```
componentDidMount() {
  console.log('componentDidMount--->', this.props.msgForGift);
  setTimeout(() => {
    let query;
    if (EnvUtil.isWX()) {
      query = Taro.createSelectorQuery().in(this.$scope);
    } else {
      query = Taro.createSelectorQuery().in(this);
    }
    query
      .select('#giftInfo')
      .boundingClientRect(rect => {
        if (!rect) return;
```

```
        this.gitfInfoWidth = rect.width;
        this.startAnimation = Taro.createAnimation({
          timingFunction: 'ease-in'
        })
          .translateX(this.gitfInfoWidth)
          .step({ duration: 500 })
          .export();
        this.endAnimation = Taro.createAnimation({
          // delay: 1500,
          timingFunction: 'ease-in'
        })
          .translateX(-this.gitfInfoWidth)
          .step({ duration: 500 })
          .export();

        this.numberAnimation = Taro.createAnimation({
          timingFunction: 'ease-out',
          delay: 200
        })
          .scale(1.3)
          .step({ duration: 50 })
          .export();
        this.numberAnimationEnd = Taro.createAnimation({
          timingFunction: 'ease-out',
          delay: 100
        })
          .scale(1)
          .step({ duration: 50 })
          .export();
      })
      .exec();
  }, 1000);
}
```

首屏加载时间，前端性能、对资源的合并处理等

懒加载

clientHeight：元素的可视高度
scrollTop：滚动条的滚动距离，就是滚动条距离容器顶部的距离
scrollHeight：容器实际内容的高度，包括超出视窗的部分

```
我们判断，当scrollTop+clientHeight = scrollHeight成立时，我们就获取数据，然后push到原数据中
<div class="sendDialog" style="max-height:200px;overflow-y:auto;"
@scroll="getNewData">
      <el-radio-group v-model="workerRadio">
        <el-radio v-for="(item, index) in workergroup" :key="index"
:label="item.id">
          <span class="workerInfo">{{item.name}}</span>
          <span>{{item.mobile}}</span>
        </el-radio>
      </el-radio-group>
</div>
getNewData (el) {
```

```
  let height = el.target.scrollHeight - el.target.scrollTop -
el.target.clientHeight   //滚动条距离底部的距离  scrollHeight是整个可滚动的高度，
scrollTop是滚动条距离顶部的高度，clientHeight是div的可视高度
  // console.log('height', height)
  if(height < 10){
    this.page++
    if(this.totalPage > this.page){  //在加载完最后一页的数据时停止加载
      this.sendWorker()     //调数据的方法，在此省略
    }
  }
}
```

扩展：判断是否到达底部

```
if (clientHeight + scrollTop - scrollHeight >= 0) {
  this.showMore = false;
  this.unRead = 0;
  // console.log("手动滚动---->1.到达底部");
  /*
  if (this.atMessage.length > 0) {
    this.cleanAtMessage();
  }
  */
} else {
  // console.log("手动滚动---->2.没到达底部");
}
```

pc公屏通行

说说与pc端的通信,QtWebChannel
在main.js 初始化QtWebChannel，
qt发送json字符串，前端通过 JSON.parse()解析字符串，store.dispatch(),再vuex 处理业务逻辑,页面通过 computed: mapState(["message", "atMessage"])获取对应数据

```
export function initMutual() {
  if (window['qt'] !== undefined) {
    new QWebChannel(window['qt'].webChannelTransport, (channel) => {
      let { objects } = channel;
      context = objects.content;
      /** 接收 pc 端传递的信息 */
      objects.content.CppToJs.connect(function (message) {
        try {
          const _message = JSON.parse(message)
          const { type } = _message
          if (type === 0) {
            // console.log("是否清空数据", type === 0)
            store.dispatch('cleanMessage')
          } else {
            store.dispatch('setMessage', _message)
          }
        }
        catch (err) {
          const errMes = {
            type: 1,
            text: "消息错误"
          }
```

```
            store.dispatch('setMessage', errMes)
            // console.log("消息错误:添加错误消息", message)
          }
        });
      });
    } else {
      // console.log("qt对象获取失败！");
    }
}
```

公屏边缘虚化效果

视频流的播放

```
<video
  ref="refvideo"
  :id="'refvideo' + index"
  autoplay="autoplay"
  muted
  controls
  crossorigin="anonymous"
  style="width:100%;height:275px;background-color: #000;"
/>
import Hls from "hls.js";
const video = document.getElementById("refvideo" + index);
if (Hls.isSupported()) {
  let hls = new Hls();
  hlsArr.push(hls);
  hls.loadSource(this.videoList[index].streamUrl);
  hls.attachMedia(video);
  hls.on(Hls.Events.MANIFEST_PARSED, function() {
    video.play();
  });
}

销毁:
stopLive() {
  for (let index = 0; index < this.hlsArr.length; index++) {
    //const video = document.getElementById("refvideo" + index);
    //console.log("video:", this.hlsArr[index]);
    //video.pause();
    this.hlsArr[index].destroy();
  }
  this.hlsArr = [];
  console.log("停止", this.hlsArr);
},
```

音频播放

```
initAutio(url, isAutoPlay = false) {
  // console.log("mounted:url---->", url, "startTime:", this.startTime);
  let audio = new Audio(url);
  audio.volume = 0.5;
  this.audio = audio;
  audio.load();
```

```
  let _this = this;
  //时间计算 start
  const timeStamp = new Date(this.startTime).getTime();
  this.startTimeStamp_temp = timeStamp;
  this.startTimeStamp = timeStamp;
  this.audio.playbackRate = this.speed;
  //时间计算 end
  audio.oncanplay = function() {
    let time = audio.duration;
    let _seconds = time;
    let _second = Math.round(_seconds);
    if (_seconds < 10) {
      _seconds = "0" + _seconds;
    }
    _this.durations_all = _seconds;
  };
  if (isAutoPlay) {
    //console.log("自动播放：", audio);
    _this.playAudio();
  }
},
//监听进度
playAudio(){
        //console.log("播放:", url);
      if (this.audio === null) {
        this.$message.warning(`没有可播放的音频`);
        return;
      }
      let that = this;
      if (this.audioStatusType === 0) {
        this.audio.play();
        this.audioStatusType = 1;
        //console.log("播放中：---------------->");
      }
      if (this.audioStatusType === 3) {
        this.audio.play();
        this.audioStatusType = 1;
        //console.log("播放完毕重新播放：---------------->");
      }
      this.audio.autoplay = true;
      this.audio.play();
      let audioI = 0;
      this.audio.addEventListener("loadstart", function() {
        that.audioStatus = "加载中...";
      });
      this.audio.addEventListener("loadeddata", function() {
        that.audioStatus = "加载完成...";
      });
      this.audio.addEventListener("error", function() {
        that.audioStatus = "加载失败...";
      });
      this.audio.addEventListener("timeupdate", function() {
        // console.log("this.speed__0", that.speed);
        that.audio.playbackRate = that.speed;
        audioI++;
        if (audioI <= 1) {
          //that.audioStatus = "播放中...";
          that.audioStatusType = 1;
```

```
        }
        //计算播放百分比start
        that.barWidth =
          (that.audio.currentTime / that.durations_all) * 100 + "%";
        //计算播放百分比end
        that.durations_seconds = that.audio.currentTime;

        //计算实时时间:开始时间+已播放时间
        that.startTimeStamp =
          that.startTimeStamp_temp + that.audio.currentTime * 1000;
      });
      this.audio.addEventListener("ended", function() {
        console.log("ended---->");
        that.audioStatusType = 3;
      });
}
```

路由守卫

```
router.beforeEach((to, from, next) => {

  //next({ path: `/login?redirect=${redirectUrl}`, replace: true });
});
```

动画

1.关键帧的定义

```
  不同于过渡动画只能定义首尾两个状态，关键帧动画可以定义多个状态，或者用关键帧的话来说，过渡动画只能定义第一帧和最后一帧这两个关键帧，而关键帧动画则可以定义任意多的关键帧，因而能实现更复杂的动画效果。
@keyframes mymove{
    from{初始状态属性}
    to{结束状态属性}
}
或
@keyframes mymove{
    0%{初始状态属性}
    50%（中间再可以添加关键帧）
    100%{结束状态属性}
}
```

2.animation API

简写:

animation:动画名称 动画持续时间 动画的过渡类型 延迟的时间 定义循环次数 定义动画方式

```
animation-name
    *检索或设置对象所应用的动画名称
    *必须与规则@keyframes配合使用，  eg:@keyframes mymove{} animation-name:mymove;

animation-duration
    *检索或设置对象动画的持续时间
    *说明：animation-duration:3s; 动画完成使用的时间为3s
```

```
animation-timing-function
   *检索或设置对象动画的过渡类型
   *属性值
      linear：线性过渡。等同于贝塞尔曲线(0.0, 0.0, 1.0, 1.0)
      ease：平滑过渡。等同于贝塞尔曲线(0.25, 0.1, 0.25, 1.0)
      ease-in：由慢到快。等同于贝塞尔曲线(0.42, 0, 1.0, 1.0)
      ease-out：由快到慢。等同于贝塞尔曲线(0, 0, 0.58, 1.0)
      ease-in-out：由慢到快再到慢。等同于贝塞尔曲线(0.42, 0, 0.58, 1.0)
      step-start:马上跳到动画每一结束桢的状态

animation-delay
    检索或设置对象动画延迟的时间
    说明：animation-delay:0.5s; 动画开始前延迟的时间为0.5s)

animation-iteration-count
    检索或设置对象动画的循环次数
    属性值
        animation-iteration-count: infinite | number;
  infinite：无限循环
  number: 循环的次数
animation-direction
    检索或设置对象动画在循环中是否反向运动
    属性值
  normal：正常方向
  reverse：反方向运行
  alternate：动画先正常运行再反方向运行，并持续交替运行
  alternate-reverse：动画先反运行再正方向运行，并持续交替运行
animation-play-state
   检索或设置对象动画的状态
   属性值
  animation-play-state:running | paused;
  running:运动
  paused: 暂停
  animation-play-state:paused; 当鼠标经过时动画停止，鼠标移开动画继续执行
```

animation vs transition

```
*相同点：都是随着时间改变元素的属性值。

*不同点：transition需要触发一个事件(hover事件或click事件等)才会随时间改变其css属性； 而
anima

```text
  <head>
    <title>动画</title>
  </head>
  <style>
    /*
一.transition(过渡):是指从一个状态到另一个状态的变化。
参考：
https://www.w3school.com.cn/cssref/pr_transition.asp

transition 属性是一个简写属性，用于设置四个过渡属性：
1.transition-property:规定设置过渡效果的 CSS 属性的名称。
2.transition-duration:规定完成过渡效果需要多少秒或毫秒。
```
```

```
  transition-duration: time;
3.transition-timing-function:规定速度效果的速度曲线。
  linear  规定以相同速度开始至结束的过渡效果（等于 cubic-bezier(0,0,1,1)）。
  ease  默认。动画以低速开始，然后加快，在结束前变慢。
  ease-in 动画以低速开始。
  ease-out  动画以低速结束。
4.transition-delay:定义过渡效果何时开始。

简写：transition: property duration timing-function delay;
 */
    .demo {
      transition: background 1s linear;
      background: blue;
      width: 100px;
      height: 100px;
      transition: width 2s;
    }
    /*
二.animation 属性是一个简写属性，用于设置六个动画属性：
参考：
https://www.w3school.com.cn/cssref/pr_animation.asp

1.animation-name: 规定需要绑定到选择器的 keyframe 名称。。
2.animation-duration: 规定完成动画所花费的时间，以秒或毫秒计。
3.animation-timing-function:规定动画的速度曲线。
  linear  动画从头到尾的速度是相同的。
  ease  默认。动画以低速开始，然后加快，在结束前变慢。
  ease-in 动画以低速开始。
  ease-out  动画以低速结束。
4.animation-delay:规定在动画开始之前的延迟。
5.animation-iteration-count:规定动画应该播放的次数。
6.animation-direction:规定是否应该轮流反向播放动画。

简写：语法
animation: name duration timing-function delay iteration-count direction;
 */
    .num {
      background: yellow;
      animation: myNum 1s 1;
      /*
            -webkit-animation-name: scaleDraw; 关键帧名称
            -webkit-animation-timing-function: ease-in-out; 动画的速度曲线
            -webkit-animation-iteration-count: infinite;  动画播放的次数
            -webkit-animation-duration: 5s; 动画所花费的时间
      合起来写：
      -webkit-animation: scaleDraw 5s ease-in-out infinite;
       */
    }
    @keyframes myNum {
      0% {
        transform: scale(1); /*开始为原始大小*/
      }
      /*
      25% {
        transform: scale(1.1); 放大1.1倍
      }
      50% {
        transform: scale(1);
```

```
      }
      */
      75% {
        transform: scale(1.1);
      }
    }
    .myMove {
      width: 100px;
      height: 100px;
      background: red;
      position: relative;
      animation: mymove 5s infinite;
    }
    @keyframes mymove {
      from {
        left: 0px;
      }
      to {
        left: 100px;
      }
      transform: scale(1.1);
    }

    /*参考*/
    /*参考*/
    /*参考*/
    .bounce-enter-active {
      /*进入时间*/
      animation: bounce-in 3s;
    }
    .bounce-leave-active {
      /*离开时间
   */
      animation: bounce-in 1s reverse;
    }
    @keyframes bounce-in {
      /*开始放大倍数*/
      0% {
        transform: scale(0);
      }
      50% {
        /*50%时放大倍数：x轴和y轴*/
        transform: scale(2.2, 1.6);
      }
      100% {
        /*最终放大倍数*/
        transform: scale(1);
      }
    }
  </style>
  <script>
    function buttonClick() {
      const box = document.getElementById("demoId");
      const width = box.clientWidth;
      const numDom = document.getElementById("demoId");
      const numwidth = numDom.clientWidth;
      if (width >= 300) {
        box.style.width = "100px";
```

```
      } else {
        box.style.width = "300px";
      }
      console.log("box", box);
      console.log("box", width);
    }
  </script>
  <body>
    <!--<transition name="bounce">-->
    <div id="demoId" class="demo">
      <div id="nubId" class="num">12</div>
    </div>
    <!--</transition>-->
    <div>
      <button type="button" onclick="buttonClick();">按钮</button>
    </div>
    <div class="myMove"></div>
  </body>
</html>
```

写一个方法遍历指定对象的所有属性

```
let obj = {aa:1,b:2}
function fn(obj){
  console.log(Object.keys(obj))
  let arr = Object.keys(obj)
  for(let i=0;i<arr.length;i++){
    console.log(arr[i])
  }
}
fn(obj)
```

举例说明什么是匿名函数？它有什么优缺点呢

匿名函数，没有名字的函数
调用方式，
1自执行
2赋值给一个变量，通过变量调用（需要在函数定义之后调用）

```
匿名函数最大的用途是创建闭包（这是JavaScript语言的特性之一），并且还可以构建命名空间，以减少全局变量的使用
const fun = function () {
  var num = 0;
  return function () {
    num++;
    console.log(num);
  };
};

const fun1 = fun()

fun1() // 1
fun1() // 2
fun1() // 3
fun1() // 4
```

写一个方法检测页面中的所有标签是否正确闭合

可以使用template对该HTML进行编译，然后对比两者是否一致，如果不一致表示有未闭合的标签/不符合规范的语法，被编译器自动修正了。

```
const areAllTagsClosed = html => {
  const template = document.createElement('template');
  template.innerHTML = html;
  return template.innerHTML === html;
}

areAllTagsClosed(`<div><span></div>`);  // false
areAllTagsClosed(`<div><br></div>`);  // true
```

template 内的html编译后在未挂载dom前不会触发事件以及javascript，可以在其挂载dom之前对其进行消毒处理，该手段也是防XSS的主要方法之一。

举例说明常用的BOM属性和方法有哪些

BOM即浏览器对象模型（brower object model）

DOM即文档对象模型（document object model）

BOM包含windows（窗口）、navigator（浏览器）、screen（浏览器屏幕）、history（访问历史）、location（地址）等
定时器：setInterval()：周期性定时器　setTimeout()：一次性定时器

打开新链接方式四种方式：

① 在当前窗口打开新链接，可后退
html: target="_self"

js:[window.]open("url","_self")

② 在当前窗口打开新链接，禁止后退

js:location.replace("url");//使用新链接，替换旧链接，同时打开新连接

③ 在新窗口打开新链接，可同时开多个

html: target="_blank"

js:[window.]open("url","_blank");

④ 在新窗口打开新链接，只能打开一个

html: target="自定义窗口名"

写一个方法，传入数字x，从一个一维数组里找到两个数字符合“n1 + n2 = x”

```
function fn(arr,sum){
  let l = arr.length
  for (var i = 0; i < l; i++) {
    const b =sum-arr[i]
    for (var j = 0; j < l; j++) {
```

```
      if(b===arr[j]){
        console.log([arr[i],arr[j]])
      }
      //
    }
  }
}
function fn(arr,sum){
  arr.filter((currentValue,index,arr)=>{
    let l = arr.length
    const b =sum-currentValue
    for (var i = 0; i < l; i++) {
      if(b===arr[i]){
        console.log([arr[index],arr[i]])
      }
    }
  })
}
fn([2, 4, 8, 6, 10, 9, 7],16)
```

写一个方法，输出最大值

```
function fn(arr){
  var max = arr[0];
  for (var i = 0; i < arr.length; i++) {
    if(max < arr[i]) {
        max = arr[i];
      }
  }
  console.log("max",max)
}
fn([2, 4, 8, 6, 10, 9, 7])
```

用原生js获取DOM元素的方法有哪些

```
getElementById
getElementByName --var x=document.getElementsByName("myInput");  <input
name="myInput" type="text" size="20" />
getElementByClassName
getElementByTagName
querySelector  --> querySelector() 方法仅仅返回匹配指定选择器的第一个元素。如果你需要返回
所有的元素，请使用 querySelectorAll() 方法替代。
querySelectorAll
document.documentElement ---获取html
document.body ---获取body
```

实现轮播图

```
关键方法 setInterval 以及计数器的把控 其次为达成优美的切换效果需要善用css过渡动画
```

map()等方法的实现

forEach()方法的实现

```
Array.prototype.myForEach = function(cb){ // 在Array对象的原型上添加 myForEach()方法， 接受一个回调函数
  for(let i=0; i<this.length; i++){ // this指的是当前实例化的数组对象
    let item = this[i]; // 定义回调函数接受的三个参数
    let index = i;
    let array = this;
    cb(item,index,array) // 调用
  }
}
```

map()：有返回值，无论返回什么都添加到新数组中。

```
Array.prototype.myMap = function(cb) {
  let newArr = []; // 定义一个新的数组，用来接受返回值
  for (let i = 0; i < this.length; i++) {
    let item = this[i];
    let index = i;
    let array = this;
    let result = cb(item, index, array);
    newArr.push(result) // 将回到函数的结果push到新的数组中
  }
  return newArr // 返回这个新的数组
}
```

filter()方法的实现

```
Array.prototype.myFilter = function(cb) {  // 实现方法和map()方法差不多
  let newArr = [];
  for (var i = 0; i < this.length; i++) {
    let item = this[i];
    let index = i;
    let array = this;
    let result = cb(item, index, array);
    if (result) { // 判断回调函数的结果否为true，为true则将 该项的item push到新的数组中
      newArr.push(item);
    }
  }
  return newArr; // 返回新的数组
}
```

找出出现个数

```
var search = function(nums, target) {
    let times = 0
    for(let i=0;i<nums.length;i++){
        if(nums[i]===target){
            times = times+1
        }
    }
    console.log(times)
};
search([5,7,7,8,8,10],8)
```

一个长度为n-1的递增排序数组中的所有数字都是唯一的，并且每个数字都在范围0～n-1之内。在范围0～n-1内的n个数字中有且只有一个数字不在该数组中，请找出这个数字。

```
var missingNumber = function(nums) {
  let a = nums[0]
  for (var i = 0; i < nums.length; i++) {
    if(a!==nums[i]){
      // console.log("---",a)
      return a
    }
    a = a+1
  }
};
missingNumber([0,1,2,3,4,5,6,7,9])

条件：0 开始，
var missingNumber = function(nums) {
  let arrLength = nums.length
  let _lenth = arrLength+1
};
missingNumber([0,1,2,3,4,5,6,7,9])
```

二分法

Math.floor() 返回小于或等于一个给定数字的最大整数。

二分法查找 js 算法

```
二分法查找算法：
采用二分法查找时，数据需是排好序的。
主要思想是：（设查找的数组区间为array[s，e]）
（1）确定该区间的中间位置m
（2）将查找的值T与array[m]比较,若相等，查找成功返回此位置；否则确定新的查找区域，继续二分查找。
区域确定如下：
这里设array从小到大排列，
array[m]>T由数组的有序性可知array[m,……,e]>T;
故新的区间为array[s,……，m-1]，
类似上面查找区间array[s,……，m-1]。
每一次查找与中间值比较，判断是否查找成功，不成功当前查找区间缩小一半，循环查找，即可。
时间复杂度:O(log2n)。
```

```
// let arr = [0, 1, 2, 4, 5, 6, 7, 8];
let arr = [0, 1, 2, 4, 5, 6, 7, 8,9];
let arr2 = [88, 77, 66, 55, 44, 33, 22, 11];

BinarySearch(arr2, 77);
// BinarySearch(arr, 2);

function BinarySearch(arr, target) {
    let s = 0;
    let e = arr.length - 1;
    let m = Math.floor((s + e) / 2);
    let sortTag = arr[s] <= arr[e];//确定排序顺序
    console.log("sortTag:",sortTag,"e:",e,"m",m) //arr2:fase e:7 m:3

    while (s < e && arr[m] !== target) {
        if (arr[m] > target) {
            sortTag && (e = m - 1);
            !sortTag && (s = m + 1);
        } else {
            !sortTag && (e = m - 1);
            sortTag && (s = m + 1);
        }
        m = Math.floor((s + e) / 2);
    }

    if (arr[m] == target) {
        console.log('找到了,位置%s', m);
        return m;
    } else {
        console.log('没找到');
        return -1;
    }
}
```

js 二分法

二分法查找算法：
采用二分法查找时，数据需是排好序的。
主要思想是：（设查找的数组区间为array[s, e]）
（1）确定该区间的中间位置m
（2）将查找的值T与array[m]比较,若相等，查找成功返回此位置；否则确定新的查找区域，继续二分查找。
区域确定如下：
这里设array从小到大排列,
array[m]>T由数组的有序性可知array[m,……,e]>T;
故新的区间为array[s,……，m-1],
类似上面查找区间array[s,……，m-1]。
每一次查找与中间值比较，判断是否查找成功，不成功当前查找区间缩小一半，循环查找，即可。
时间复杂度:O(log2n)。

```
const arr1 = [1,4,5,8,12,16,18,19,20,21,22,23,34,44,56]

function binarySearch(arr, num) {
    let len = arr.length
    let leftIndex = 0
    let rightIndex = len - 1
```

```
    console.log("len",len)
    while(leftIndex <= rightIndex) {
        let mid = Math.floor((leftIndex + rightIndex)/2);
      console.log("leftIndex",leftIndex,"rightIndex",rightIndex,"mid",mid);
        if(num === arr[mid]) { // 找到返回mid
            return mid
        } else if (num > arr[mid]) { // 比中间值大，说明在 mid 到 rightIndex 之间；否则就在 mid 到 leftIndex 之间
            leftIndex = mid + 1
        } else {
            rightIndex = mid - 1
        }
    }
    return -1 // 没找到返回-1
}
console.log(binarySearch(arr1, 12))
```

两数之和

本题可以通过三种方式去解答

暴力法1:双层for循环进行遍历，属于最基础的检索与判断方式O(n*n)*
*暴力法2:通过python的**str in list**方式逐个遍历，虽然代码看似简单，但每一次in操作的复杂度一样是O(n),所以总体复杂度O(n*n)*
之所以把这道题归档在HashMap类中，是因为通过HashMap的方式，能在O(n)的时间复杂度下完成

输入：nums = [2,5,11,15,7], target = 9
输出：[0,1]
解释：因为 nums[0] + nums[1] == 9 ， 返回 [0, 1]

暴力法：

```
var twoSum = function(nums, target) {
  for (var i = 0; i < nums.length; i++) {
    let left = nums[i]
    for (var j = i+1; j < nums.length; j++) {
      if(left + nums[j]===target){
        return[i,j]
      }
    }
  }

};
// twoSum([2,5,11,15,7],9)
twoSum([2,99,9,8,3,99,4],7)
```

对象键值

```
var twoSum = function (nums, target) {
  //将出现过的数字的索引进行保存到对象中
  let prevNums = {};
  for (let i = 0; i < nums.length; i++) {
    // 当前数字
    const currentNum = nums[i];
    // 差值方式
    const targetNum = target - currentNum;
```

```
    // 获取差值的下标  ->没有 undefined  存进去 ->有  返回下标
    const targetNumIndex = prevNums[targetNum];
    if (targetNumIndex === undefined) {
      //判断当前没有目标值对应数的索引，那么将数据存放到preNums
      prevNums[currentNum] = i;
    } else {
    //返回结果
    console.log("prevNums",prevNums)
      return [targetNumIndex, i];
    }
  }
};
twoSum([2,99,9,8,3,99,4],7)
```