## event 对象

### 1.event概念，作用

系统给我们封装的，任何事件都会有这个event对象，这个对象当中封装了和这个事件相关的一切信息；

### 2.event兼容性处理

如果是高级浏览器去调用函数的回调函数，它会把事件对象封装好传给回调函数的第一个形参;

如果是低版本浏览器去调用，它**会把事件对象封装好作为window的一个属性 window.event;**

**所**以我们在去拿事件对象的时候，要兼容性去拿

event = event || window.event;

### 3.三种鼠标位置

clientX & clientY

拿的是鼠标相对视口的 水平距离和垂直距离 相对的是视口的左上角（以视口左上角为原点）

pageX pageY

拿的是鼠标相对页面的 水平距离和垂直距离 相对的是页面的左上角（以页面左上角为原点）

offsetX offsetY

拿的是鼠标相对自身元素的 水平距离和垂直距离 相对的是自身元素左上角（以自身元素左上角为原点）

## 案例

鼠标跟随

学生管理系统

## 定时器：

### 1.单次定时器（延迟定时器）

一般用来做延迟效果 定时炸弹

案例 求出结果延迟5秒打印

### 2.多次定时器（循环定时器）

和循环类似都是为了重复去做一件事 闹钟

案例 每隔3秒打印我爱你

### 3.清除定时器：

清除单次定时器

清除多次定时器

注意：无论是延迟还是循环定时器，这两个都是window的一个方法，而且以后我们要存储这两个方法生成的定时器编号，都要使用全局变量进行存储；不要在函数内部定义局部变量进行存储定时器编号，因为我们可能在程序的任何位置去清除这个定时器；

### 4.强化案例：

年历

阅读协议

### 5.同步和异步：

所有的定时器事件回调都是异步操作（事件 ajax 生命周期回调都是异步代码）；

对于我们的代码，以后我们可以认为代码分为同步代码和异步代码；

异步代码是要等同步代码执行完成之后才会执行的；（js是单线程的）；

console.log(1)

    setTimeout(function(){

      console.log(3)

    },1000)

console.log(2)

    console.log(1)

    box.onclick=function(){

      console.log(3)

    }

console.log(2)

var a = 1

    console.log('我喜欢你')

    setTimeout(function(){

      console.log('我爱你')

    },1000)

    for(var i =0;i<40000;i++){

      for(var j =0;j<40000;j++){

        a++

      }

}

var a = 1

    console.log('我喜欢你')

    setTimeout(function(){

      console.log('你好')

    },1000)

    setTimeout(function(){

      console.log('世界')

    },4000)

    for(var i =0;i<40000;i++){

      for(var j =0;j<40000;j++){

        a++

      }

    }

数组方法中的回调函数是同步执行的

## 元素的大小和位置

### 1.js操作的样式都是行内样式

如果我们直接去拿元素设置好的大小和位置

<style>

position:absolute;

left:100px;

top:100px;

width:200px;

height:200px;

background-color:red;

</style>

console.log(box.style.width);

console.log(box.style.left);

这样我们是获取不到元素的样式的；因为开始行内我们并没有写元素的样式；上面的是用来给元素设置大小和位置。读取不到；

box.style.width = '300px';

### 2.Client系列 只读

**clientWidth**

**拿的是盒子 内容 + padding的宽；**

**clientHeight**

**拿的是盒子 内容 + padding的高；**

clientLeft

拿的是盒子左边框大小；

clientTop

拿的是盒子上边框大小；

### 3.Offset系列 只读

**offsetWidth**

**拿的是盒子 内容 + padding + border的宽**

**offsetHeight**

**拿的是盒子 内容 + padding + border的高**

**offsetLeft**

**拿的是元素的偏移量：可以认为就是拿的绝对定位left值**

**offsetTop**

**拿的是元素的偏移量：可以认为就是拿的绝对定位top值**

**这两个值，相对的参照物和绝对定位完全一样；**

### 4.Scroll系列

scrollWidth 只读

当内容比盒子小的时候，拿的是盒子的clientWidth

当内容比盒子大的时候，拿的是内容的offsetWidth+内容一侧的margin + 盒子的一侧内边距；

scrollHeight 只读

当内容比盒子小的时候，拿的是盒子的clientHeight

当内容比盒子大的时候，拿的是内容的offsetHeight+内容一侧的margin + 盒子的一侧内边距；

**scrollTop 可读可写**

**拿的是盒子内容向上滚动的距离**

**scrollLeft 可读可写**

**拿的是盒子内容向左滚动的距离**

### 5.总结

1）元素的大小：宽和高的获取：

以后我们拿元素的宽和高 先看元素有没有边框 如果没有边框 那么clientWidth和offsetWidth是一样的，如果有边框，看你需要不，需要的话就用offsetWidth 不需要就用clientWidth; scrollWidth几乎不用；

2）元素的位置（偏移量）的获取：

以后需要获取元素的位置直接通过offsetLeft和offsetTop去获取，但是注意相对的参照元素是谁；（和绝对定位参照类似）

3）视口宽高求法（固定的）

document.documentElement.clientWidth

document.documentElement.clientHeight

### 6.案例

案例：导航跟随

案例：盒子来回移动

## 系统滚动条的控制

### 1.系统滚动条的控制

html和body这两个元素overflow的scroll属性，控制着系统的滚动条

系统的滚动条有两个，一个是body身上的 一个是document身上的。我们平时看到的那个滚动条是document身上的。，如果我们想要控制系统滚动条哪个显示哪个关闭分以下情况：

1. 单独的给body或者html 设置overflow:scroll 滚动条打开的全部都是document的

2）如果两个元素同时设置overflow属性，body设置的是scroll,html设置是hidden,那么document的滚动条被关闭，body身上的滚动条会打开相反，body身上被关闭，document身上的被打开。

**3）如果两个元素同时设置overflow:hidden；那么系统的两个滚动条全部被关闭；**

4）如果两个都设置overflow:scroll,那么html会打开document身上的，而body会打开自己身上的滚动条；

### 2.禁止系统滚动条

html,body{

height:100%;

overflow:hidden;

}

为什么要加height:100%;

这个属性加上只是为了让设置更有说服力，只有内容超出才会被掩藏或者出现滚动条, 如果不设置，那么body和html高度将由内容自动撑开，也就是说body当中的内容永远不会溢出。overflow:hidden;

## 案例拖拽：

### 1.拖拽的原理（画图）

点击元素拿到元素的初始位置和鼠标的初始位置

移动的时候拿到鼠标最后的位置，可以求出鼠标的距离差

元素最终的位置 = 元素的初始位置 + 鼠标的距离差\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

注意： 两个方向都要去照顾；

### 2.pc拖拽事件的问题

事件在谁身上添加？

刚开始我们把所有事件全部添加在元素身上；

元素事件是怎么去加，全部平行的去加还是嵌套加，思考过程；

move事件在鼠标Up之后不会自动消失；

在up事件当中需要将鼠标的move事件和up事件一起取消掉；

当在元素内部我们有文本，文本可以拖拽，所以要取消浏览器的默认行为；

接着我们快速的去拖动元素，当鼠标移出元素，再次回去的时候，元素的move事件仍然存在；那么我们就该考虑事件添加是否有问题；

拖拽当中mousedown的事件确实给元素添加，但是mousemove和mouseup事件要添加在父元素身上，使用事件冒泡(委派)去解决元素的移动

并且鼠标移出元素的时候，我们的事件仍然有效；

### 3.在基础的拖拽事件上添加边界问题

当元素在四周的时候，不能超出范围，做出范围界定；

### 4.边界吸附效果

当元素距离四周边界50px时候立即吸附到边界（吸附效果）

### 5、元素碰撞问题（九宫格）

计算元素到视口上方和左边的距离 getBoundingClientRect() 只能读不能写

### 6、抽象出拖拽的基础模板