DAY16 运动函数

2023年5月12日 17:41

运动的概述

1.通过 js 的方式，让物体属性发生变化，比如说物体匀速运动，淡入淡出，反弹，抛物线等动作。

2.定时器(setInterval/clearInterval)

3.操作css属性以及值

offsetLeft/offsetTop/offsetWidth/offsetHeight

getComputedStyle获取任意的css属性值

给css属性进行赋值(逐个赋值，cssText，选择器)

具体的运动

1.匀速运动

2.缓冲运动(透明度，链式，多物体，多属性) - 封装

3.边界运动

4.抛物线运动 - 公式

匀速运动

一.匀速运动：速度是固定，让盒子动起来。

```
const box = document.querySelector('.box');
box.onclick = function () {
  let speed = 10;
  // 初始值
  // 速度
  // 运动和停止(定时器改变)
  setInterval(() => {
    box.style.left = box.offsetLeft + speed + 'px';
  }, 20);
};
```

二.定时器和事件问题

定时器叠加，点击box开启一个定时器，如果不断的去点击，定时器就会叠加，速度就会越来越快

保证只开一个定时器，这样不会因为事件的多次触发影响运动的速度

```
const box = document.querySelector('.box');
let timer = null;//定时器的返回值
box.onclick = function () {
  let speed = 1;
  clearInterval(timer);//开启下一个定时器之前，关闭前面的定时器，保证定时器只有一
```

个。

```
    timer = setInterval(() => {
      box.style.left = box.offsetLeft + speed + 'px';
    }, 20);
  };
```

三.定时器的时间

```
  const box = document.querySelector('.box');
  let timer = null;//定时器的返回值
  box.onclick = function () {
    let speed = 10;
    clearInterval(timer);//开启下一个定时器之前，关闭前面的定时器，保证定时器只有一
个。
    timer = setInterval(() => {
      box.style.left = box.offsetLeft + speed + 'px';
    }, 1000 / 60);
  };
```

缓冲运动

```
    .box {
      width: 100px;
      height: 100px;
      background-color: red;
      position: absolute;
      left: 0px;
      top: 0;
    }

  <div class="box"></div>
```

一.缓冲运动：由快到慢，离目标点越远，速度越快，越近越慢最终停下来。

```
  const box = document.querySelector('.box');
  let timer = null;//定时器的返回值
  let speed = 0;//速度
  box.onclick = function () {
    clearInterval(timer);
    timer = setInterval(() => {
      // 求缓冲的速度(大->小)，利用目标点
      speed = Math.ceil((700 - box.offsetLeft) / 10); //70 63 ....
      // 判断运动停止
      if (box.offsetLeft === 700) {
        clearInterval(timer);
      } else {
        box.style.left = box.offsetLeft + speed + 'px';
```

```
    }
    console.log(speed);
  }, 1000 / 60)
};
```

封装函数

函数的好处

封装：打包多条代码，将常用的功能进行打包，下面使用方便一些。

重用：封装好的函数可以反复使用。

可扩展：函数的功能可以无限叠加。

函数的封装过程

1.直接先实现功能

2.打包成函数

3.将函数代码中可变的内容整理成参数

4.迭代其他的功能

```
function bufferMove(element, attr, target) {//element:运动的元素  attr:运动的属性   target:目标点
  let timer = null;//定时器的返回值
  let speed = 0;//速度
  clearInterval(timer);
  timer = setInterval(() => {
    // 0.获取当前属性值
    let current = parseInt(window.getComputedStyle(element)[attr]);
    // 1.合并速度
    speed = (target - current) / 10;
    speed = speed > 0 ? Math.ceil(speed) : Math.floor(speed);
    // 2.判断运动停止
    if (current === target) {
      clearInterval(timer);
    } else {
      element.style[attr] = current + speed + 'px';
    }
    console.log(speed);
  }, 1000 / 60)
}
const box = document.querySelector('.box');
box.onclick = function () {
  bufferMove(this, 'left', 600);
};
```

1.多物体运动 - 定时器的问题

给每一个物体添加一个定时器(将定时器的返回值绑定到元素对象身上，当自定义属性)

```
    .box {
      width: 200px;
      height: 100px;
      background-color: red;
      margin-bottom: 10px;
    }


  <div class="box"></div>
  <div class="box"></div>
  <div class="box"></div>

  function bufferMove(element, attr, target) {//element:运动的元素  attr:运动的属性   target:目标点
    let speed = 0;//速度
    clearInterval(element.timer);
    element.timer = setInterval(() => {
      // 0.获取当前属性值
      let current = parseInt(window.getComputedStyle(element)[attr]);
      // 1.合并速度
      speed = (target - current) / 10;
      speed = speed > 0 ? Math.ceil(speed) : Math.floor(speed);
      // 2.判断运动停止
      if (current === target) {
        clearInterval(element.timer);
      } else {
        element.style[attr] = current + speed + 'px';
      }
    }, 1000 / 60)
  }
  const boxs = document.querySelectorAll('.box');
  boxs[0].onmouseover = function () {
    bufferMove(this, 'width', 1000);
  };
  boxs[0].onmouseout = function () {
    bufferMove(this, 'width', 200);
  };
  boxs[1].onmouseover = function () {
    bufferMove(this, 'width', 1000);
  };
  boxs[1].onmouseout = function () {
    bufferMove(this, 'width', 200);
  };
  boxs[2].onmouseover = function () {
    bufferMove(this, 'width', 1000);
  };
  boxs[2].onmouseout = function () {
    bufferMove(this, 'width', 200);
  };
```

2.透明度运动 - 取值赋值的问题

2.1.获取当前值的时候，因为采用了parseInt，透明度的值本身就是小数，产生问题

解决方式：将获取的值扩大100倍，赋值的时候缩小100倍，前提必须判断是在做透明度运动，通过属性进行判断

解决方式：将目标值也扩大100倍

2.2.赋值的时候不能有单位

解决方式：继续判断区分赋值

```
function bufferMove(element, attr, target) {//element:运动的元素  attr:运动的属性   target:目标点
  let speed = 0;//速度
  clearInterval(element.timer);
  element.timer = setInterval(() => {
    // 0.获取当前属性值
    let current = null;
    if (attr === 'opacity') {//透明度运动
      current = parseFloat(window.getComputedStyle(element)[attr] * 100);
    } else {//其他运动
      current = parseInt(window.getComputedStyle(element)[attr]);
    }
    // 1.合并速度
    speed = (target - current) / 10;
    speed = speed > 0 ? Math.ceil(speed) : Math.floor(speed);
    // 2.判断运动停止
    if (current === target) {
      clearInterval(element.timer);
    } else {
      if (attr === 'opacity') {//透明度赋值
        element.style[attr] = (current + speed) / 100;
      } else {
        element.style[attr] = current + speed + 'px';
      }
    }
  }, 1000 / 60)
}
const box = document.querySelector('.box');
box.onmouseover = function () {
  bufferMove(this, 'opacity', 100);
};
box.onmouseout = function () {
  bufferMove(this, 'opacity', 10);
};
```

3.链式运动 - 上一次运动结束继续开启下一次的运动，一个属性一个属性去完成。

上一个属性运动完成，开始下一个属性的运动

链式运动配置为可选的，可以没有

```
function bufferMove(element, attr, target, fn) {//element:运动的元素  attr:运动的属性  target:目标点  fn:链式运动的回调函数
    let speed = 0;//速度
    clearInterval(element.timer);
    element.timer = setInterval(() => {
      // 0.获取当前属性值
      let current = null;
      if (attr === 'opacity') {//透明度运动
        current = parseFloat(window.getComputedStyle(element)[attr] * 100);
      } else {//其他运动
        current = parseInt(window.getComputedStyle(element)[attr]);
      }
      // 1.合并速度
      speed = (target - current) / 10;
      speed = speed > 0 ? Math.ceil(speed) : Math.floor(speed);
      // 2.判断运动停止
      if (current === target) {
        clearInterval(element.timer);
        // 上一个属性运动结束，开启下一个属性
        // fn存在，同时fn为函数，调用函数
        fn && typeof fn === 'function' && fn();
      } else {
        if (attr === 'opacity') {//透明度赋值
          element.style[attr] = (current + speed) / 100;
        } else {
          element.style[attr] = current + speed + 'px';
        }
      }
    }, 1000 / 60)
  }
  const box = document.querySelector('.box');
  box.onmouseover = function () {
    bufferMove(this, 'width', 1000, () => {
      bufferMove(this, 'height', 600)
    });
  };
  box.onmouseout = function () {
    bufferMove(this, 'height', 100, () => {
      bufferMove(this, 'width', 100)
    });
  };
```

4.多属性同时运动 - 运动目前是基于单属性运动

遍历的方式

通过将属性和目标设置为对象格式，利用遍历对象的方式一次次执行我们的属性运动

```
// functon bufferMove(element,attr,target,fn) //element:运动的元素  attr:运动的属性  target:目标点  fn:链式运动的回调函数
function bufferMove(element, obj, fn) {//element:运动的元素  obj:运动的属性和目标值  fn:链式运动的回调函数
    let speed = 0;//速度
    clearInterval(element.timer);
    element.timer = setInterval(() => {
      for (let attr in obj) {//遍历属性   attr:运动的属性   obj[attr]:目标点
        // 0.获取当前属性值
        let current = null;
        if (attr === 'opacity') {//透明度运动
          current = parseFloat(window.getComputedStyle(element)[attr] * 100);
        } else {//其他运动
          current = parseInt(window.getComputedStyle(element)[attr]);
        }
        // 1.合并速度
        speed = (obj[attr] - current) / 10;
        speed = speed > 0 ? Math.ceil(speed) : Math.floor(speed);
        // 2.判断运动停止
        if (current === obj[attr]) {
          clearInterval(element.timer);
          // 上一个属性运动结束，开启下一个属性
          // fn存在，同时fn为函数，调用函数
          fn && typeof fn === 'function' && fn();
        } else {
          if (attr === 'opacity') {//透明度赋值
            element.style[attr] = (current + speed) / 100;
          } else {
            element.style[attr] = current + speed + 'px';
          }
        }
      }
    }, 1000 / 60)
  }
  const box = document.querySelector('.box');
  box.onmouseover = function () {
    bufferMove(this, { width: 600, height: 600 });
  };
  box.onmouseout = function () {
    bufferMove(this, { width: 100, height: 100 });
  };
```

5.多属性运动的bug

有一个属性到了目标点，整个定时器关闭了，运动结束了。

解决方式

确保每一个属性都到了目标点，才结束运动。

```
// functon bufferMove(element,attr,target,fn) //element:运动的元素  attr:运动的属性  target:目标点  fn:链式运动的回调函数
function bufferMove(element, obj, fn) {//element:运动的元素  obj:运动的属性和目标值  fn:链式运动的回调函数
    let speed = 0;//速度
    clearInterval(element.timer);
    element.timer = setInterval(() => {
      let flag = true;
      for (let attr in obj) {//遍历属性   attr:运动的属性   obj[attr]:目标点
        // 0.获取当前属性值
        let current = null;
        if (attr === 'opacity') {//透明度运动
          current = parseFloat(window.getComputedStyle(element)[attr] * 100);
        } else {//其他运动
          current = parseInt(window.getComputedStyle(element)[attr]);
        }
        // 1.合并速度
        speed = (obj[attr] - current) / 10;
        speed = speed > 0 ? Math.ceil(speed) : Math.floor(speed);
        // 2.判断运动停止
        if (current !== obj[attr]) {//属性没有到目标点，继续运动
          if (attr === 'opacity') {//透明度赋值
            element.style[attr] = (current + speed) / 100;
          } else {
            element.style[attr] = current + speed + 'px';
          }
          flag = false;//通过标记查看运动是继续还是继续
        }
      }
      // 如果flag是false，说明运动没有结束，继续运动
      // 如果flag是true，说明运动结束了
      if (flag) {
        clearInterval(element.timer);
        // 上一个属性运动结束，开启下一个属性
        // fn存在，同时fn为函数，调用函数
        fn && typeof fn === 'function' && fn();
      }
    }, 1000 / 60)
  }
  const box = document.querySelector('.box');
  box.onmouseover = function () {
    bufferMove(this, { width: 1000, height: 101 });
  };
```

```
box.onmouseout = function () {
  bufferMove(this, { width: 100, height: 100 });
};
```

边界运动：

物体运动到边界，速度取反。

```
const box = document.querySelector('.box');
let timer = null;
let speedx = 3;
let speedy = 6;
box.onclick = function () {
  clearInterval(timer);
  timer = setInterval(() => {
    let l = box.offsetLeft + speedx;
    let t = box.offsetTop + speedy;
    if (t >= document.documentElement.clientHeight - box.offsetHeight) {
      t = document.documentElement.clientHeight - box.offsetHeight;
      speedy *= -1;
    } else if (t <= 0) {
      t = 0;
      speedy *= -1;
    }
    if (l >= document.documentElement.clientWidth - box.offsetWidth) {
      l = document.documentElement.clientWidth - box.offsetWidth;
      speedx *= -1;
    } else if (l <= 0) {
      l = 0;
      speedx *= -1;
    }
    box.style.left = l + 'px';
    box.style.top = t + 'px';
  }, 1000 / 60);
}
```

碰撞运动

```
const box = document.querySelector('.box');
let timer = null;
let speedx = 3;
let speedy = 6;
box.onclick = function () {
  clearInterval(timer);
  timer = setInterval(() => {
    speedy++;//模拟加速度
    let l = box.offsetLeft + speedx;
    let t = box.offsetTop + speedy;
    if (t >= document.documentElement.clientHeight - box.offsetHeight) {
      t = document.documentElement.clientHeight - box.offsetHeight;
      speedy *= -0.9;//模拟力的分解以及损害
```

```
        speedx *= 0.9;
      } else if (t <= 0) {
        t = 0;
        speedy *= -0.9;
        speedx *= 0.9;
      }
      if (l >= document.documentElement.clientWidth - box.offsetWidth) {
        l = document.documentElement.clientWidth - box.offsetWidth;
        speedx *= -1;
      } else if (l <= 0) {
        l = 0;
        speedx *= -1;
      }
      // 判断运动停止
      if (Math.abs(speedx) < 1) {
        speedx = 0;
      }
      if (Math.abs(speedy) < 1) {
        speedy = 0;
      }
      if (speedx === 0 && speedy === 0 && t ===
document.documentElement.clientHeight - box.offsetHeight) {
        clearInterval(timer);
      }
      box.style.left = l + 'px';
      box.style.top = t + 'px';
      console.log(speedx, speedy);
    }, 1000 / 60);
  }
```

名词解释 - 非常重要

1.API: 通俗的讲就是别人写好的代码，或者编译好的程序，提供给你使用，就叫作api(API可以称之为说明文档)

2.插件：把项目中的某一部分功能的JS代码进行封装, 具有具体的业务逻辑，有针对性，如果项目中有类似的需求，直接引入插件代码即可.

3.组件：类似于插件，但是插件一般都是把JS部分进行封装，组件不仅封装了JS部分，还有HTML和CSS部分，以后再使用，直接按照说明文档引入使用。

4.类库：提供了一些真实项目开发中常用的方法，这些方法做了一些完善处理，比如兼容处理，方便我们开发和维护。

5.框架：比上面的三个都要庞大，它不仅提供了常用的方法，而且也支持一些插件的扩展，提供了非常优秀的代码管理设计思想。是一套完整的项目解决方案，对项目的侵入性比较大，如果需要换框架，则需要重新架构整个项目。VUE / REACT

CDN(内容分发网络)

https://www.bootcdn.cn/

有一些知名的公司，将开发中常用的工具放到自己服务器上面，提供给所有的开发者使用。

引入的js或者css文件后缀名包含min，表示压缩版本(文件更小),弊端是可读性很差。

swiper

一.swiper

- Swiper是纯javascript打造的滑动特效插件，面向手机、平板电脑等移动终端。
- Swiper能实现触屏焦点图、触屏Tab切换、触屏轮播图切换等常用效果。
- Swiper开源、免费、稳定、使用简单、功能强大，是架构移动终端网站的重要选择!

https://www.swiper.com.cn/index.html

二.如何使用

1.找到对应官网

2.学习api

3.按照提供的顺序进行使用

三.开始使用

1.首先加载插件

2.添加HTML内容

3.你可能想要给Swiper定义一个大小

4.初始化Swiper。

5.完成。恭喜你，现在你的Swiper应该已经能正常切换了。