­08 – 事见高级： 2020-08-24 LUO JIN

主要学习: 注册事件（绑定事件），删除事件（解绑事件），DOM事件流，事件对象，阻止事件冒泡，事件委托（代理，委派），常用的鼠标事件，常用的键盘事件

**1. 注册事件（绑定事件）**

**1.1 注册事件概述**

给元素添加事件，称为**注册事件**或者绑定事件。 注册事件有两种方式：**传统方式**和**方法监听注册**方式

|  |  |
| --- | --- |
| **传统注册方式**   * 利用 on 开头的事件 **onclick** * btn.onclick = function() {} * 特点： 注册事件的**唯一性** * **同一个元素同一个事件只能设置一个处理函数，最后注册的处理函数将会覆盖前面注册的处理函数** | **方法监听注册方式**   * w3c 标准 推荐方式 * **addEventListener**() 它是一个方法 * IE9 **之前**的 IE 不支持此方法，可使用 **attachEvent**() 代替 * **特点：同一个元素同一个事件可以注册多个监听器** * **按注册顺序依次执行** |

**【代码演示】：**

|  |
| --- |
| <button>传统注册事件</button>      <button>方法监听注册事件</button>      <button>ie9 attachEvent</button>      <script>          var btns = document.querySelectorAll('button');          // 1. 传统方式注册事件          btns[0].onclick = function() {              alert('hi');          }          btns[0].onclick = function() {                  alert('hao a u');              }              // 2. 事件侦听注册事件 addEventListener              // (1) 里面的事件类型是字符串 必定加引号 而且不带on              // (2) 同一个元素 同一个事件可以添加多个侦听器（事件处理程序）          btns[1].addEventListener('click', function() {              alert(22);          })          btns[1].addEventListener('click', function() {                  alert(33);              })              // 3. attachEvent ie9以前的版本支持（了解就好了！：D）          btns[2].attachEvent('onclick', function() {              alert(11);          })      </script> |

**1.2 addEventListener 事件监听方式**

eventTarget.**addEventListener(type, listener[, useCapture])**

**eventTarget.addEventListener()**方法将指定的监听器注册到 eventTarget（目标对象）上，当该对象触发指定的事件时，就会执行事件处理函数。

该方法接收三个参数：

* **type**：事件类型字符串，**比如 click 、mouseover ，注意这里不要带 on**
* **listener**：事件处理函数，事件发生时，会调用该监听函数
* **useCapture**：可选参数，是一个布尔值，默认是 false。

**1.3 attachEvent 事件监听方式（了解即可）**

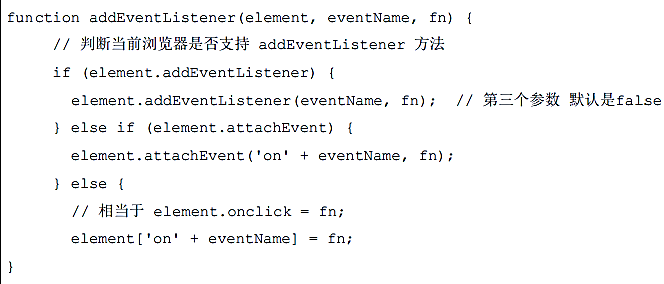
**eventTarget.attachEvent(eventNameWithOn, callback) 了解一下就好了！**

**eventTarget.attachEvent()**方法将指定的监听器注册到 eventTarget（目标对象） 上，当该对象触 发指定的事件时，指定的回调函数就会被执行。

该方法接收两个参数：

* **eventNameWithOn**：事件类型字符串，比如 onclick 、onmouseover **，这里要带 on**
* **callback**： 事件处理函数，当目标触发事件时回调函数被调用
* **注意**：IE8 及早期版本支持

**1.4 注册事件兼容性解决方案（了解！）**



**兼容性处理的原则： 首先照顾大多数浏览器，再处理特殊浏览器**

**2. 删除事件（解绑事件）**

**2.1 删除事件的方式**

**1. 传统注册方式**： eventTarget.onclick = null;

**2. 方法监听注册方式：**

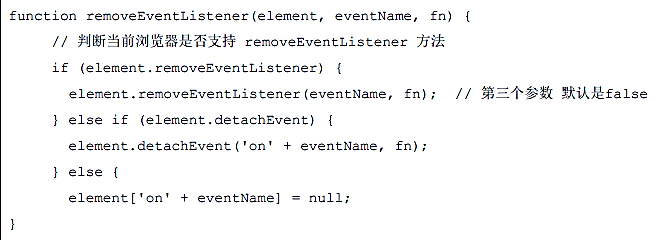
① eventTarget.removeEventListener(type, listener[, useCapture]);

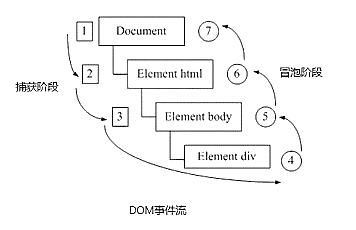
② eventTarget.detachEvent(eventNameWithOn, callback);

**【代码演示】**

|  |
| --- |
| <div>1</div>      <div>2</div>      <div>3</div>      <script>          var divs = document.querySelectorAll('div');          divs[0].onclick = function() {                  alert(11);                  // 1. 传统方式删除事件 点击一次后在点击就没有了                  divs[0].onclick = null;              }              // 2. removeEventListener 删除事件          divs[1].addEventListener('click', fn) // 里面的fn 不需要调用加小括号          function fn() {              alert(22);              divs[1].removeEventListener('click', fn);          }          // 3. detachEvent          divs[2].attachEvent('onclick', fn1);          function fn1() {              alert(33);              divs[2].detachEvent('onclick', fn1);          }      </script> |

**2.2 删除事件兼容性解决方案（了解）**



**3. DOM 事件流**

**事件流**描述的是从页面中接收事件的顺序。

**事件**发生时会在元素节点之间按照特定的顺序传播，这个**传播过程**即 **DOM 事件流**。

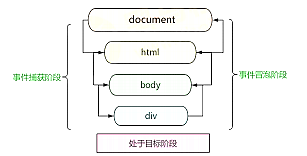
比如我们给一个div 注册了点击事件：

* DOM 事件流分为3个阶段： 1. 捕获阶段 2. 当前目标阶段 3. 冒泡阶段

**事件冒泡**： IE 最早提出，事件开始时由最具体的元素接收，然后**逐级向上**传播到到 DOM 最顶层节点的过程。

**事件捕获**： 网景最早提出，由 DOM 最顶层节点开始，然后**逐级向下**传播到到最具体的元素接收的过程

【理解】我们向水里面扔一块石头，首先它会有一个下降的过程，这个过程就可以理解为从最顶层向事件发生的最具体元素（目标点）的捕获过程；之后会产生泡泡，会在最低点（ 最具体元素）之后漂浮到水面上，这个过 程相当于事件冒泡。



**注意**

1. **JS 代码**中只能执行捕获或者冒泡其中的一个阶段。

2. onclick 和 attachEvent 只能得到冒泡阶段。

3. **addEventListener**(‘type‘, listener[, useCapture])第三个参数如果是 true，表示在事件捕获阶段调用事件处理程序；如果是 false（不写默认就是false），表示在事件冒泡阶段调用事件处理程序。

4. **实际开发中我们很少使用事件捕获，我们更关注事件冒泡**。

5**. 有些事件是没有冒泡的，比如 onblur、onfocus、onmouseenter、onmouseleave**

6. **事件冒泡有时候会带来麻烦**，**有时候又会帮助很巧妙的做某些事件**，后面讲解。

【**代码演示**】

|  |
| --- |
| <div class="father">          <div class="son">son盒子</div>      </div>      <script>          // dom 事件流 三个阶段          // 1. JS 代码中只能执行捕获或者冒泡其中的一个阶段。          // 2. onclick 和 attachEvent（ie） 只能得到冒泡阶段。          // 3. 捕获阶段 如果addEventListener 第三个参数是 true 那么则处于捕获阶段  document -> html -> body -> father -> son          // var son = document.querySelector('.son');          // son.addEventListener('click', function() {          //     alert('son');          // }, true);          // var father = document.querySelector('.father');          // father.addEventListener('click', function() {          //     alert('father');          // }, true);          // 4. 冒泡阶段 如果addEventListener 第三个参数是 false 或者 省略 那么则处于冒泡阶段  son -> father ->body -> html -> document          var son = document.querySelector('.son');          son.addEventListener('click', function() {              alert('son');          }, false);          var father = document.querySelector('.father');          father.addEventListener('click', function() {              alert('father');          }, false);          document.addEventListener('click', function() {              alert('document');          })      </script> |

**4. 事件对象**

**4.1 什么是事件对象**

**eventTarget.onclick = function(event) {}**

**eventTarget.addEventListener('click', function(event) {}）**

//这个 event 就是事件对象，我们还喜欢的写成 e 或者 evt

官方解释：event 对象代表**事件的状态**，**比如键盘按键的状态、鼠标的位置、鼠标按钮的状态**。

简单理解：事件发生后，跟事件相关的一系列信息数据的集合都放到这个对象里面，这个对象就是事件对象 event，它有很多属性和方法。

比如：

1. 谁绑定了这个事件。

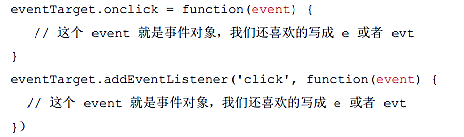
2. **鼠标**触发事件的话，会得到鼠标的相关信息，如鼠标位置。

3. **键盘**触发事件的话，会得到键盘的相关信息，如按了哪个键。

【代码演示】

|  |
| --- |
| <div>123</div>      <script>          // 事件对象          var div = document.querySelector('div');          div.onclick = function(e) {                  // console.log(e);                  // console.log(window.event);                  // e = e || window.event;                  console.log(e);              }              // div.addEventListener('click', function(e) {              //         console.log(e);          //     })          // 1. event 就是一个事件对象 写到我们侦听函数的 小括号里面 当形参来看          // 2. 事件对象只有**有了事件才会存在**，它是系统给我们自动创建的，不需要我们传递参数          // 3. 事件对象 是 我们事件的一系列相关数据的集合 跟事件相关的 比如鼠标点击里面就包含了鼠标的相关信息，鼠标坐标啊，如果是键盘事件里面就包含的键盘事件的信息 比如 判断用户按下了那个键          // 4. 这个事件对象我们可以自己命名 比如 event 、 evt、 e          // 5. 事件对象也有兼容性问题 ie678 通过 window.event 兼容性的写法  e = e || window.event;      </script> |

**4.2 事件对象的使用语法**



这个 event 是个形参，系统帮我们设定为**事件对象**，不需要传递实参过去。

当我们注册事件时， event 对象就会被系统自动创建，并依次传递给事件监听器（事件处理函数）。

**4.3 事件对象的兼容性方案**

事件对象本身的获取存在兼容问题：

1. 标准浏览器中是浏览器给方法传递的参数，只需要定义形参 e 就可以获取到。

2. 在 IE6~8 中，浏览器不会给方法传递参数，如果需要的话，需要到 window.event 中获取查找。 【解决】: e = e || window.event;

**4.4 事件对象的常见属性和方法**

e.target 和 this 的区别：

* this 是事件**绑定**的元素， 这个函数的调用者（绑定这个事件的元素）
* e.target 是事件**触发**的元素。



|  |
| --- |
| div>123</div>      <ul>          <li>abc</li>          <li>abc</li>          <li>abc</li>      </ul>      <script>          // 常见事件对象的属性和方法          // 1. e.target 返回的是触发事件的对象（元素）  this 返回的是绑定事件的对象（元素）          // 区别： **e.target 点击了那个元素，就返回那个元素**  **this 那个元素绑定了这个点击事件，那么就返回谁**          var div = document.querySelector('div');          div.addEventListener('click', function(e) {              console.log(e.target);              console.log(this);          })          var ul = document.querySelector('ul');          ul.addEventListener('click', function(e) {                  // 我们给ul 绑定了事件  **那么this 就指向ul**                  console.log(this);                  console.log(e.currentTarget);                  // **e.target 指向我们点击的那个对象 谁触发了这个事件 我们点击的是li e.target 指向的就是li**                  console.log(e.target);              })              // 了解兼容性              // div.onclick = function(e) {              //     e = e || window.event;              //     var target = e.target || e.srcElement;              //     console.log(target);          // }          // 2. 了解： 跟 this 有个非常相似的属性 currentTarget  ie678不认识      </script> |

【代码演示】

|  |
| --- |
| <div>123</div>      <a href="http://www.baidu.com">百度</a>      <form action="http://www.baidu.com">          <input type="submit" value="提交" name="sub">      </form>      <script>          // 常见事件对象的属性和方法 +  // 1. 返回事件类型          var div = document.querySelector('div');          div.addEventListener('click', fn);          div.addEventListener('mouseover', fn);          div.addEventListener('mouseout', fn);          function fn(e) {              console.log(e.type);          }          // 2. 阻止默认行为（事件） 让链接不跳转 或者让提交按钮不提交          var a = document.querySelector('a');          a.addEventListener('click', function(e) {                  e.preventDefault(); //  dom 标准写法              })              // 3. 传统的注册方式          a.onclick = function(e) {              // 普通浏览器 e.preventDefault();  方法              // e.preventDefault();              // 低版本浏览器 ie678  returnValue  属性              // e.returnValue;              // 我们可以利用return false 也能阻止默认行为 没有兼容性问题 特点： **return 后面的代码不执行了， 而且只限于传统的注册方式**  **return false;**              alert(11);          }  </script> |

**5. 阻止事件冒泡（\*\*\*\*\*面试经常会问~~~！！！）**

**5.1 阻止事件冒泡的两种方式**

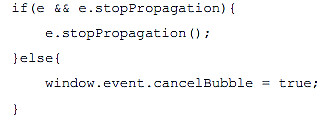
事件冒泡：开始时由 最具体的元素接收，然后逐级向上传播到到 DOM 最顶层节点。

事件冒泡本身的特性，会带来的坏处，也会带来的好处，需要我们灵活掌握。

阻止事件冒泡

* 标准写法：利用事件对象里面的 stopPropagation()方法：**e.stopPropagation() 用的多**
* 非标准写法：IE 6-8 利用事件对象cancelBubble属性：e.cancelBubble = true;

**5.2 阻止事件冒泡的兼容性解决方案（了解）**

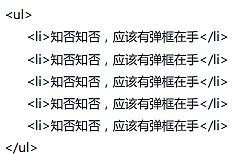


|  |
| --- |
| <div class="father">          <div class="son">son儿子</div>      </div>      <script>          // 常见事件对象的属性和方法          // 阻止冒泡  dom 推荐的标准 stopPropagation()          var son = document.querySelector('.son');          son.addEventListener('click', function(e) {              alert('son');              e.stopPropagation(); // stop 停止  Propagation 传播              e.cancelBubble = true; // 非标准 cancel 取消 bubble 泡泡          }, false);          var father = document.querySelector('.father');          father.addEventListener('click', function() {              alert('father');          }, false);          document.addEventListener('click', function() {              alert('document');          })      </script> |

**6. 事件委托（代理、委派）**

事件冒泡本身的特性，会带来的坏处，也会带来的好处，需要我们灵活掌握。生活中有如下场景：

咱们班有100个学生， 快递员有100个快递， 如果一个个的送花费时间较长。同时每个学生领取的时候，也 需要排队领取，也花费时间较长，何如？

【解决方案】： 快递员把100个快递，委托给班主任，班主任把这些快递放到办公室，同学们下课自行领取即可。 优势： 快递员省事，**委托**给班主任就可以走了。 同学们领取也方便，因为相信班主任。

以前，把ul 中的li, 获取过来，给每个循环添加点击事件，太麻烦。点击每个 li 都会弹出对话框，以前需要给每个 li 注册事件，是非常辛苦的，而且访问 DOM 的次数越多，这就 会延长整个页面的交互就绪时间。

**事件委托：**事件委托也称为事件代理， 在 jQuery 里面称为事件委派。

**事件委托的原理：**不是每个子节点单独设置事件监听器，而是事件监听器设置在其父节点上，然后利用冒泡原理影响设置每个子节点。 以上案例：给 ul 注册点击事件，然后利用事件对象的 target 来找到当前点击的 li，因为点击 li，事件会冒泡到 ul 上， ul 有注册事件，就会触发事件监听器。（**面试常问道！！！）** 把点击事件委托给他父亲，给父亲加事件监听器

**事件委托的作用：**我们只操作了一次 DOM ，提高了程序的性能。

【代码演示】

|  |
| --- |
| <ul>          <li>知否知否，点我应有弹框在手！</li>          <li>知否知否，点我应有弹框在手！</li>          <li>知否知否，点我应有弹框在手！</li>          <li>知否知否，点我应有弹框在手！</li>          <li>知否知否，点我应有弹框在手！</li>      </ul>      <script>          // 事件委托的核心原理：给父节点添加侦听器， 利用事件冒泡影响每一个子节点          var ul = document.querySelector('ul');          ul.addEventListener('click', function(e) {              // alert('知否知否，点我应有弹框在手！');              // e.target 这个可以得到我们点击的对象              e.target.style.backgroundColor = 'pink';          })      </script> |

**7. 常用的鼠标事件**

**7.1 常用的鼠标事件**



**1．禁止鼠标右键菜单**

**contextmenu**主要控制应该何时显示上下文菜单，主要用于程序员取消默认的上下文菜单

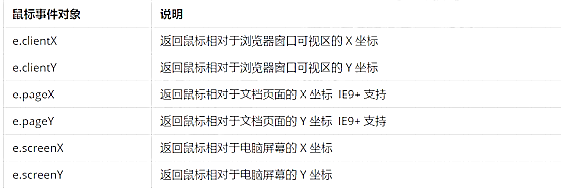


**2.禁止鼠标选中**（**selectstart** 开始选中）



**7.2 鼠标事件对象**

event对象代表事件的状态，跟事件相关的一系列信息的集合。现阶段我们主要是用鼠标事件对象 MouseEvent 和键盘事件对象 KeyboardEvent。



【代码演示】

|  |
| --- |
| <script>          // 鼠标事件对象 MouseEvent          document.addEventListener('click', function(e) {              // 1. client 鼠标在 –》**可视区** 的x和y坐标              console.log(e.clientX);              console.log(e.clientY);              console.log('---------------------');              // 2. page 鼠标在页面文档的x和y坐标（**开发用的多！！！！）**              console.log(e.pageX);              console.log(e.pageY);              console.log('---------------------');              // 3. screen 鼠标在电脑屏幕的x和y坐标（开发用的少）              console.log(e.screenX);              console.log(e.screenY);          })      </script> |

【案例】跟随鼠标的天使：这个天使图片一直跟随鼠标移动

【案例分析】

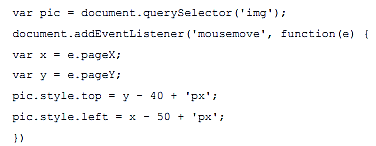
① 鼠标不断的移动，使用鼠标移动事件： mousemove

② 在页面中移动，给document注册事件

③ 图片要移动距离，而且不占位置，我们使用绝对定位即可

④ 核心原理： 每次鼠标移动，我们都会获得最新的鼠标坐标， 把这个x和y坐标做为图片的 top和left 值就可以移动图片

【实现代码】



|  |
| --- |
| <style>          img {position: absolute; top: 2px;} //如果加了定位，top值应该给一个值+px      </style>  </head>  <body>      <img src="images/angel.gif" alt="">      <script>          var pic = document.querySelector('img'); // 获取图片          document.addEventListener('mousemove', function(e) {              // 1. mousemove只要我们鼠标移动1px 就会触发这个事件              // console.log(1);              // 2.核心原理： 每次鼠标移动，我们都会获得最新的鼠标坐标， 把这个x和y坐标做为图片的top和left 值就可以移动图片              var x = e.pageX;              var y = e.pageY;              console.log('x坐标是' + x, 'y坐标是' + y);              //3.千万不要忘记给left 和top 添加px 单位 -50，-40让鼠标在中间位置              pic.style.left = x - 50 + 'px';              pic.style.top = y - 40 + 'px';          });      </script> |

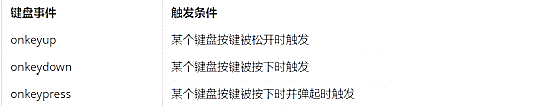
**应用：京东买东西的时候点击放大图片，有一个框框会随着鼠标滑动而滑动**

**8. 常用的键盘事件**

**8.1 常用键盘事件**

事件除了使用鼠标触发，还可以使用键盘触发， 注意 **给文档 document 添加键盘事件**

注意： onkeypress 和前面2个的区别是，它不识别功能键，比如左右箭头，shift 等



|  |
| --- |
| <script>          // 常用的键盘事件          //1. keyup 按键弹起的时候触发          // document.onkeyup = function() {          //         console.log('我弹起了');          //     } // **addEventListener不用加 on**          document.addEventListener('keyup', function() {              console.log('我弹起了');          })          //3. keypress 按键按下的时候触发  不能识别功能键 比如 ctrl shift 左右箭头啊          document.addEventListener('keypress', function() {                  console.log('我按下了press');              })              //2. keydown 按键按下的时候触发  能识别功能键 比如 ctrl shift Alt 左右箭头啊          document.addEventListener('keydown', function() {                  console.log('我按下了down');              })              // 4. 三个事件的执行顺序  keydown -- keypress -- keyup |

**8.2 ASCII 表**

**8.3 键盘事件对象**



注意： **onkeydown 和 onkeyup 不区分字母大小写，onkeypress 区分字母大小写**。

【代码演示】key : 有兼容性问题，：D Ummmmm~~` = keyCode来代替

|  |
| --- |
| <script>          // 键盘事件对象中的keyCode属性可以得到相应键的ASCII码值          // 1. 我们的keyup 和keydown事件不区分字母大小写  a 和 A 得到的都是65          // 2. 我们的keypress 事件 区分字母大小写  a  97 和 A 得到的是65          document.addEventListener('keyup', function(e) {              // console.log(e);              console.log('up:' + e.keyCode);              // 我们可以利用keycode返回的ASCII码值来判断用户按下了那个键【应用】              if (e.keyCode === 65) {                  alert('您按下的a键');              } else {                  alert('您没有按下a键')              }          })          document.addEventListener('keypress', function(e) {              // console.log(e);              console.log('press:' + e.keyCode);          })      </script> |

【案例】： 模拟京东按键输入内容，当我们按下 s 键， 光标就定位到搜索框

【案例分析】：

① 核心思路：检测用户是不是按下了S键，如果是，光标定位到搜索框

② 使用键盘事件对象里边的keyCode判断用户按下了是否是S键

③ 搜索框获得焦点，使用JS里边的focus（）方法

【实现代码】：

|  |
| --- |
| <input type="text">      <script>          // 核心思路： 检测用户是否按下了s 键，如果按下s 键，就把光标定位到搜索框里面          // 使用键盘事件对象里面的keyCode 判断用户按下的是否是s键          // 搜索框获得焦点： 使用 js 里面的 focus() 方法          var search = document.querySelector('input');  // “keydown”的话按下S的话，抬起会输入到搜索框里，不太合适，所以用“keyup”          document.addEventListener('keyup', function(e) {              // console.log(e.keyCode); 得到 s = 83              if (e.keyCode === 83) {                  search.focus();              }          })      </script> |

【案例】： 模拟京东快递单号查询

【案例要求】：要求：**当我们在文本框中输入内容时，文本框上面自动显示大字号的内容**。



【案例分析】:

1. 快递单号输入内容时，上面的大字号盒子con会显示
2. 表单检验用户输入：给表单添加键盘事件
3. 同时快递单号里边的值value获取赋值给con盒子innerHTML 作为内容
4. 如果快递单号内容为空，大盒子隐藏
5. 注意：**keydown 和keypress** 在文本框里边的特点：他们两个事件触发的时候，**文字还没有落入文本框；有先后顺序， keydown 如果开发一款游戏，一按下就发射子弹，一直发射发射**
6. Keyup 事件触发的时候，文字已经落入文本框里边了
7. 当失去焦点的时候，就隐藏con这个盒子
8. 当获得焦点的时候，并且文本和内容不为空，显示con这个盒子

【实现代码】

|  |
| --- |
| <style>          \* {margin: 0; padding: 0; }          .search {position: relative; width: 178px; margin: 100px;}          .con {              display: none;              position: absolute;              top: -40px;              width: 171px;              border: 1px solid rgba(0, 0, 0, .2);              box-shadow: 0 2px 4px rgba(0, 0, 0, .2);              padding: 5px 0;              font-size: 18px;              line-height: 20px;              color: #333;          }          .con::before {              content: '';              width: 0;              height: 0;              position: absolute;              top: 28px;              left: 18px;              border: 8px solid #000;              border-style: solid dashed dashed;              border-color: #fff transparent transparent;          }      </style>  </head>  <body>      <div class="search">          <div class="con">123</div> // 用来显示隐藏          <input type="text" placeholder="请输入您的快递单号" class="jd">      </div>      <script>          // 快递单号输入内容时， 上面的大号字体盒子（con）显示(这里面的字号更大）          // 表单检测用户输入： 给表单添加键盘事件          // 同时把快递单号里面的值（value）获取过来赋值给 con盒子（innerText）做为内容          // 如果快递单号里面内容为空，则隐藏大号字体盒子(con)盒子          var con = document.querySelector('.con');          var jd\_input = document.querySelector('.jd');          jd\_input.addEventListener('keyup', function() {                  // console.log('输入内容啦');                  if (this.value == '') {                      con.style.display = 'none';                  } else {                      con.style.display = 'block';                      con.innerText = this.value;                  }              })              // 当我们失去焦点，就隐藏这个con盒子          jd\_input.addEventListener('blur', function() {                  con.style.display = 'none';              })              // 当我们获得焦点，就显示这个con盒子          jd\_input.addEventListener('focus', function() {              if (this.value !== '') { // 里边有内容才让他显示出来                  con.style.display = 'block';              }          })      </script> |