**http://www.cnblogs.com/lrzw32/p/4696655.html**

[**一步一步实现JS拖拽插件**](http://www.cnblogs.com/lrzw32/p/4696655.html)

js拖拽是常见的网页效果，本文将从零开始实现一个简单的js插件。

**一、js拖拽插件的原理**

常见的拖拽操作是什么样的呢？整过过程大概有下面几个步骤：

　　1、用鼠标点击被拖拽的元素

　　2、按住鼠标不放，移动鼠标

　　3、拖拽元素到一定位置，放开鼠标

这里的过程涉及到三个dom事件：onmousedown,onmousemove,onmouseup。所以拖拽的基本思路就是：

　　1、用鼠标点击被拖拽的元素触发onmousedown

　　　　（1）设置当前元素的可拖拽为true，表示可以拖拽

　　　　（2）记录当前鼠标的坐标x,y

　　　　（3）记录当前元素的坐标x,y

　　2、移动鼠标触发onmousemove

　　　　（1）判断元素是否可拖拽，如果是则进入步骤2，否则直接返回

　　　　（2）如果元素可拖拽，则设置元素的坐标

　　　　　　元素的x坐标 = 鼠标移动的横向距离+元素本来的x坐标 = 鼠标现在的x坐标 - 鼠标之前的x坐标 + 元素本来的x坐标

　　　　　　元素的y坐标 = 鼠标移动的横向距离+元素本来的y坐标 = 鼠标现在的y坐标 - 鼠标之前的y坐标 + 元素本来的y坐标

　　3、放开鼠标触发onmouseup

　　　　（1）将鼠标的可拖拽状态设置成false

**二、根据原理实现的最基本效果**

在实现基本的效果之前，有几点需要说明的：

　　1、元素想要被拖动，它的postion属性一定要是relative或absolute

　　2、通过event.clientX和event.clientY获取鼠标的坐标

　　3、onmousemove是绑定在document元素上而不是拖拽元素本身，这样能解决快速拖动造成的延迟或停止移动的问题

代码如下：

[复制代码](javascript:void(0);)

1 var dragObj = document.getElementById("test");

2 dragObj.style.left = "0px";

3 dragObj.style.top = "0px";

4

5 var mouseX, mouseY, objX, objY;

6 var dragging = false;

7

8 dragObj.onmousedown = function (event) {

9 event = event || window.event;

10

11 dragging = true;

12 dragObj.style.position = "relative";

13

14

15 mouseX = event.clientX;

16 mouseY = event.clientY;

17 objX = parseInt(dragObj.style.left);

18 objY = parseInt(dragObj.style.top);

19 }

20

21 document.onmousemove = function (event) {

22 event = event || window.event;

23 if (dragging) {

24

25 dragObj.style.left = parseInt(event.clientX - mouseX + objX) + "px";

26 dragObj.style.top = parseInt(event.clientY - mouseY + objY) + "px";

27 }

28

29 }

30

31 document.onmouseup = function () {

32 dragging = false;

33 }

[复制代码](javascript:void(0);)

**三、代码抽象与优化**

上面的代码要做成插件，要将其抽象出来，基本结构如下：

[复制代码](javascript:void(0);)

1 ; (function (window, undefined) {

2

3 function Drag(ele) {}

4

5 window.Drag = Drag;

6 })(window, undefined);

[复制代码](javascript:void(0);)

用自执行匿名函数将代码包起来，内部定义Drag方法并暴露到全局中，直接调用Drag，传入被拖拽的元素。

首先对一些常用的方法进行简单的封装：

[复制代码](javascript:void(0);)

1 ; (function (window, undefined) {

2 var dom = {

3 //绑定事件

4 on: function (node, eventName, handler) {

5 if (node.addEventListener) {

6 node.addEventListener(eventName, handler);

7 }

8 else {

9 node.attachEvent("on" + eventName, handler);

10 }

11 },

12 //获取元素的样式

13 getStyle: function (node, styleName) {

14 var realStyle = null;

15 if (window.getComputedStyle) {

16 realStyle = window.getComputedStyle(node, null)[styleName];

17 }

18 else if (node.currentStyle) {

19 realStyle = node.currentStyle[styleName];

20 }

21 return realStyle;

22 },

23 //获取设置元素的样式

24 setCss: function (node, css) {

25 for (var key in css) {

26 node.style[key] = css[key];

27 }

28 }

29 };

30

31 window.Drag = Drag;

32 })(window, undefined);

[复制代码](javascript:void(0);)

在一个拖拽操作中，存在着两个对象：被拖拽的对象和鼠标对象，我们定义了下面的两个对象以及它们对应的操作：

首先的拖拽对象，它包含一个元素节点和拖拽之前的坐标x和y：

[复制代码](javascript:void(0);)

1 function DragElement(node) {

2 this.node = node;//被拖拽的元素节点

3 this.x = 0;//拖拽之前的x坐标

4 this.y = 0;//拖拽之前的y坐标

5 }

6 DragElement.prototype = {

7 constructor: DragElement,

8 init: function () {

9 this.setEleCss({

10 "left": dom.getStyle(node, "left"),

11 "top": dom.getStyle(node, "top")

12 })

13 .setXY(node.style.left, node.style.top);

14 },

15 //设置当前的坐标

16 setXY: function (x, y) {

17 this.x = parseInt(x) || 0;

18 this.y = parseInt(y) || 0;

19 return this;

20 },

21 //设置元素节点的样式

22 setEleCss: function (css) {

23 dom.setCss(this.node, css);

24 return this;

25 }

26 }

[复制代码](javascript:void(0);)

还有一个对象是鼠标，它主要包含x坐标和y坐标：

[复制代码](javascript:void(0);)

1 function Mouse() {

2 this.x = 0;

3 this.y = 0;

4 }

5 Mouse.prototype.setXY = function (x, y) {

6 this.x = parseInt(x);

7 this.y = parseInt(y);

8 }

[复制代码](javascript:void(0);)

这是在拖拽操作中定义的两个对象。

如果一个页面可以有多个拖拽元素，那应该注意什么：

1、每个元素对应一个拖拽对象实例

2、每个页面只能有一个正在拖拽中的元素

为此，我们定义了唯一一个对象用来保存相关的配置：

1 var draggableConfig = {

2 zIndex: 1,

3 draggingObj: null,

4 mouse: new Mouse()

5 };

这个对象中有三个属性：

（1）zIndex：用来赋值给拖拽对象的zIndex属性，有多个拖拽对象时，当两个拖拽对象重叠时，会造成当前拖拽对象有可能被挡住，通过设置zIndex使其显示在最顶层

（2）draggingObj：用来保存正在拖拽的对象，在这里去掉了前面的用来判断是否可拖拽的变量，通过draggingObj来判断当前是否可以拖拽以及获取相应的拖拽对象

（3）mouse：唯一的鼠标对象，用来保存当前鼠标的坐标等信息

最后是绑定onmousedown，onmouseover，onmouseout事件，整合上面的代码如下：

[复制代码](javascript:void(0);)

1 ; (function (window, undefined) {

2 var dom = {

3 //绑定事件

4 on: function (node, eventName, handler) {

5 if (node.addEventListener) {

6 node.addEventListener(eventName, handler);

7 }

8 else {

9 node.attachEvent("on" + eventName, handler);

10 }

11 },

12 //获取元素的样式

13 getStyle: function (node, styleName) {

14 var realStyle = null;

15 if (window.getComputedStyle) {

16 realStyle = window.getComputedStyle(node, null)[styleName];

17 }

18 else if (node.currentStyle) {

19 realStyle = node.currentStyle[styleName];

20 }

21 return realStyle;

22 },

23 //获取设置元素的样式

24 setCss: function (node, css) {

25 for (var key in css) {

26 node.style[key] = css[key];

27 }

28 }

29 };

30

31 //#region 拖拽元素类

32 function DragElement(node) {

33 this.node = node;

34 this.x = 0;

35 this.y = 0;

36 }

37 DragElement.prototype = {

38 constructor: DragElement,

39 init: function () {

40 this.setEleCss({

41 "left": dom.getStyle(node, "left"),

42 "top": dom.getStyle(node, "top")

43 })

44 .setXY(node.style.left, node.style.top);

45 },

46 setXY: function (x, y) {

47 this.x = parseInt(x) || 0;

48 this.y = parseInt(y) || 0;

49 return this;

50 },

51 setEleCss: function (css) {

52 dom.setCss(this.node, css);

53 return this;

54 }

55 }

56 //#endregion

57

58 //#region 鼠标元素

59 function Mouse() {

60 this.x = 0;

61 this.y = 0;

62 }

63 Mouse.prototype.setXY = function (x, y) {

64 this.x = parseInt(x);

65 this.y = parseInt(y);

66 }

67 //#endregion

68

69 //拖拽配置

70 var draggableConfig = {

71 zIndex: 1,

72 draggingObj: null,

73 mouse: new Mouse()

74 };

75

76 function Drag(ele) {

77 this.ele = ele;

78

79 function mouseDown(event) {

80 var ele = event.target || event.srcElement;

81

82 draggableConfig.mouse.setXY(event.clientX, event.clientY);

83

84 draggableConfig.draggingObj = new DragElement(ele);

85 draggableConfig.draggingObj

86 .setXY(ele.style.left, ele.style.top)

87 .setEleCss({

88 "zIndex": draggableConfig.zIndex++,

89 "position": "relative"

90 });

91 }

92

93 ele.onselectstart = function () {

94 //防止拖拽对象内的文字被选中

95 return false;

96 }

97 dom.on(ele, "mousedown", mouseDown);

98 }

99

100 dom.on(document, "mousemove", function (event) {

101 if (draggableConfig.draggingObj) {

102 var mouse = draggableConfig.mouse,

103 draggingObj = draggableConfig.draggingObj;

104 draggingObj.setEleCss({

105 "left": parseInt(event.clientX - mouse.x + draggingObj.x) + "px",

106 "top": parseInt(event.clientY - mouse.y + draggingObj.y) + "px"

107 });

108 }

109 })

110

111 dom.on(document, "mouseup", function (event) {

112 draggableConfig.draggingObj = null;

113 })

114

115

116 window.Drag = Drag;

117 })(window, undefined);

[复制代码](javascript:void(0);)

调用方法：Drag(document.getElementById("obj"));

注意的一点，为了防止选中拖拽元素中的文字，通过onselectstart事件处理程序return false来处理这个问题。

**四、扩展：有效的拖拽元素**

我们常见的一些拖拽效果很有可能是这样的：



弹框的顶部是可以进行拖拽操作的，内容区域是不可拖拽的，怎么实现这样的效果呢：

首先优化拖拽元素对象如下，增加一个目标元素target，表示被拖拽对象，在上图的登录框中，就是整个登录窗口。

被记录和设置坐标的拖拽元素就是这个目标元素，但是它并不是整个部分都是拖拽的有效部分。我们在html结构中为拖拽的有效区域添加类draggable表示有效拖拽区域：

[复制代码](javascript:void(0);)

1 <div id="obj1" class="dialog" style="position:relative;left:50px">

2 <div class="header draggable">

3 拖拽的有效元素

4 </div>

5 <div class="content">

6 拖拽对象1

7 </div>

8 </div>

[复制代码](javascript:void(0);)

然后修改Drag方法如下：

[复制代码](javascript:void(0);)

function drag(ele) {

var dragNode = (ele.querySelector(".draggable") || ele);

dom.on(dragNode, "mousedown", function (event) {

var dragElement = draggableConfig.dragElement = new DragElement(ele);

draggableConfig.mouse.setXY(event.clientX, event.clientY);

draggableConfig.dragElement

.setXY(dragElement.target.style.left, dragElement.target.style.top)

.setTargetCss({

"zIndex": draggableConfig.zIndex++,

"position": "relative"

});

}).on(dragNode, "mouseover", function () {

dom.setCss(this, draggableStyle.dragging);

}).on(dragNode, "mouseout", function () {

dom.setCss(this, draggableStyle.defaults);

});

}

[复制代码](javascript:void(0);)

主要修改的是绑定mousedown的节点变成了包含draggable类的有效元素，如果不含有draggable，则整个元素都是有效元素。

**五、性能优化和总结**

由于onmousemove在一直调用，会造成一些性能问题，我们可以通过setTimout来延迟绑定onmousemove事件，改进move函数如下

[复制代码](javascript:void(0);)

1 function move(event) {

2 if (draggableConfig.dragElement) {

3 var mouse = draggableConfig.mouse,

4 dragElement = draggableConfig.dragElement;

5 dragElement.setTargetCss({

6 "left": parseInt(event.clientX - mouse.x + dragElement.x) + "px",

7 "top": parseInt(event.clientY - mouse.y + dragElement.y) + "px"

8 });

9

10 dom.off(document, "mousemove", move);

11 setTimeout(function () {

12 dom.on(document, "mousemove", move);

13 }, 25);

14 }

15 }

[复制代码](javascript:void(0);)

总结：

整个拖拽插件的实现其实很简单，主要是要注意几点

　　1、实现思路：元素拖拽位置的改变就等于鼠标改变的距离，关键在于获取鼠标的变动和元素原本的坐标

      2、通过setTimeout来延迟加载onmousemove事件来提供性能

**六、jquery插件化**

简单地将其封装成jquery插件，主要是相关的dom方法替换成jquery方法来操作

[复制代码](javascript:void(0);)

1 ; (function ($, window, undefined) {

2 //#region 拖拽元素类

3 function DragElement(node) {

4

5 this.target = node;

6

7 node.onselectstart = function () {

8 //防止拖拽对象内的文字被选中

9 return false;

10 }

11 }

12 DragElement.prototype = {

13 constructor: DragElement,

14 setXY: function (x, y) {

15 this.x = parseInt(x) || 0;

16 this.y = parseInt(y) || 0;

17 return this;

18 },

19 setTargetCss: function (css) {

20 $(this.target).css(css);

21 return this;

22 }

23 }

24 //#endregion

25

26 //#region 鼠标元素

27 function Mouse() {

28 this.x = 0;

29 this.y = 0;

30 }

31 Mouse.prototype.setXY = function (x, y) {

32 this.x = parseInt(x);

33 this.y = parseInt(y);

34 }

35 //#endregion

36

37 //拖拽配置

38 var draggableConfig = {

39 zIndex: 1,

40 dragElement: null,

41 mouse: new Mouse()

42 };

43

44 var draggableStyle = {

45 dragging: {

46 cursor: "move"

47 },

48 defaults: {

49 cursor: "default"

50 }

51 }

52

53 var $document = $(document);

54

55 function drag($ele) {

56 var $dragNode = $ele.find(".draggable");

57 $dragNode = $dragNode.length > 0 ? $dragNode : $ele;

58

59

60 $dragNode.on({

61 "mousedown": function (event) {

62 var dragElement = draggableConfig.dragElement = new DragElement($ele.get(0));

63

64 draggableConfig.mouse.setXY(event.clientX, event.clientY);

65 draggableConfig.dragElement

66 .setXY(dragElement.target.style.left, dragElement.target.style.top)

67 .setTargetCss({

68 "zIndex": draggableConfig.zIndex++,

69 "position": "relative"

70 });

71 },

72 "mouseover": function () {

73 $(this).css(draggableStyle.dragging);

74 },

75 "mouseout": function () {

76 $(this).css(draggableStyle.defaults);

77 }

78 })

79 }

80

81 function move(event) {

82 if (draggableConfig.dragElement) {

83 var mouse = draggableConfig.mouse,

84 dragElement = draggableConfig.dragElement;

85 dragElement.setTargetCss({

86 "left": parseInt(event.clientX - mouse.x + dragElement.x) + "px",

87 "top": parseInt(event.clientY - mouse.y + dragElement.y) + "px"

88 });

89

90 $document.off("mousemove", move);

91 setTimeout(function () {

92 $document.on("mousemove", move);

93 }, 25);

94 }

95 }

96

97 $document.on({

98 "mousemove": move,

99 "mouseup": function () {

100 draggableConfig.dragElement = null;

101 }

102 });

103

104 $.fn.drag = function (options) {

105 drag(this);

106 }

107

108 })(jQuery, window, undefined)

[复制代码](javascript:void(0);)