拖拉事件

拖拉事件的种类

拖拉（drag）指的是，用户在某个对象上按下鼠标键不放，拖动它到另一个位置，然后释放鼠标键，将该对象放在那里。

拖拉的对象有好几种，包括元素节点、图片、链接、选中的文字等等。在网页中，除了元素节点默认不可以拖拉，其他（图片、链接、选中的文字）都是可以直接拖拉的。为了让元素节点可拖拉，可以将该节点的`draggable`属性设为`true`。

```html
<div draggable="true">
  此区域可拖拉
</div>
```

`draggable`属性可用于任何元素节点，但是图片（`<img>`）和链接（`<a>`）不加这个属性，就可以拖拉。对于它们，用到这个属性的时候，往往是将其设为`false`，防止拖拉这两种元素。

注意，一旦某个元素节点的`draggable`属性设为`true`，就无法再用鼠标选中该节点内部的文字或子节点了。

当元素节点或选中的文本被拖拉时，就会持续触发拖拉事件，包括以下一些事件。

- `drag`：拖拉过程中，在被拖拉的节点上持续触发（相隔几百毫秒）。
- `dragstart`：用户开始拖拉时，在被拖拉的节点上触发，该事件的`target`属性是被拖拉的节点。通常应该在这个事件的监听函数中，指定拖拉的数据。
- `dragend`：拖拉结束时（释放鼠标键或按下 ESC 键）在被拖拉的节点上触发，该事件的`target`属性是被拖拉的节点。它与`dragstart`事件，在同一个节点上触发。不管拖拉是否跨窗口，或者中途被取消，`dragend`事件总是会触发的。
- `dragenter`：拖拉进入当前节点时，在当前节点上触发一次，该事件的`target`属性是当前节点。通常应该在这个事件的监听函数中，指定是否允许在当前节点放下（drop）拖拉的数据。如果当前节点没有该事件的监听函数，或者监听函数不执行任何操作，就意味着不允许在当前节点放下数据。在视觉上显示拖拉进入当前节点，也是在这个事件的监听函数中设置。
- `dragover`：拖拉到当前节点上方时，在当前节点上持续触发（相隔几百毫秒），该事件的`target`属性是当前节点。该事件与`dragenter`事件的区别是，`dragenter`事件在进入该节点时触发，然后只要没有离开这个节点，`dragover`事件会持续触发。
- `dragleave`：拖拉操作离开当前节点范围时，在当前节点上触发，该事件的`target`属性是当前节点。如果要在视觉上显示拖拉离开操作当前节点，就在这个事件的监听函数中设置。
- `drop`：被拖拉的节点或选中的文本，释放到目标节点时，在目标节点上触发。注意，如果当前节点不允许`drop`，即使在该节点上方松开鼠标键，也不会触发该事件。如果用户按下 ESC 键，取消这个操作，也不会触发该事件。该事件的监听函数负责取出拖拉数据，并进行相关处理。

下面的例子展示，如何动态改变被拖动节点的背景色。

```javascript
div.addEventListener('dragstart', function (e) {
  this.style.backgroundColor = 'red';
}, false);

div.addEventListener('dragend', function (e) {
  this.style.backgroundColor = 'green';
}, false);
```

上面代码中，`div`节点被拖动时，背景色会变为红色，拖动结束，又变回绿色。

下面是一个例子，展示如何实现将一个节点从当前父节点，拖拉到另一个父节点中。

```javascript
/* HTML 代码如下
 <div class="dropzone">
   <div id="draggable" draggable="true">
     该节点可拖拉
   </div>
 </div>
 <div class="dropzone"></div>
 <div class="dropzone"></div>
 <div class="dropzone"></div>
*/

// 被拖拉节点
var dragged;

document.addEventListener('dragstart', function (event) {
  // 保存被拖拉节点
  dragged = event.target;
  // 被拖拉节点的背景色变透明
  event.target.style.opacity = 0.5;
}, false);

document.addEventListener('dragend', function (event) {
  // 被拖拉节点的背景色恢复正常
  event.target.style.opacity = '';
}, false);

document.addEventListener('dragover', function (event) {
  // 防止拖拉效果被重置，允许被拖拉的节点放入目标节点
  event.preventDefault();
}, false);

document.addEventListener('dragenter', function (event) {
```

```
  // 目标节点的背景色变紫色
  // 由于该事件会冒泡，所以要过滤节点
  if (event.target.className === 'dropzone') {
    event.target.style.background = 'purple';
  }
}, false);

document.addEventListener('dragleave', function( event ) {
  // 目标节点的背景色恢复原样
  if (event.target.className === 'dropzone') {
    event.target.style.background = '';
  }
}, false);

document.addEventListener('drop', function( event ) {
  // 防止事件默认行为（比如某些元素节点上可以打开链接），
  event.preventDefault();
  if (event.target.className === 'dropzone') {
    // 恢复目标节点背景色
    event.target.style.background = '';
    // 将被拖拉节点插入目标节点
    dragged.parentNode.removeChild(dragged);
    event.target.appendChild( dragged );
  }
}, false);
```

关于拖拉事件，有以下几个注意点。

- 拖拉过程只触发以上这些拖拉事件，尽管鼠标在移动，但是鼠标事件不会触发。
- 将文件从操作系统拖拉进浏览器，不会触发`dragstart`和`dragend`事件。
- `dragenter`和`dragover`事件的监听函数，用来取出拖拉的数据（即允许放下被拖拉的元素）。由于网页的大部分区域不适合作为放下拖拉元素的目标节点，所以这两个事件的默认设置为当前节点不允许接受被拖拉的元素。如果想要在目标节点上放下的数据，首先必须阻止这两个事件的默认行为。

```html
<div ondragover="return false">
<div ondragover="event.preventDefault()">
```

上面代码中，如果不取消拖拉事件或者阻止默认行为，就不能在`div`节点上放下被拖拉的节点。

DragEvent 接口

拖拉事件都继承了`DragEvent`接口，这个接口又继承了`MouseEvent`接口和`Event`接口。

浏览器原生提供一个`DragEvent()`构造函数，用来生成拖拉事件的实例对象。

```javascript
new DragEvent(type, options)
```

`DragEvent()`构造函数接受两个参数，第一个参数是字符串，表示事件的类型，该参数必须；第二个参数是事件的配置对象，用来设置事件的属性，该参数可选。配置对象除了接受`MouseEvent`接口和`Event`接口的配置属性，还可以设置`dataTransfer`属性要么是`null`，要么是一个`DataTransfer`接口的实例。

`DataTransfer`的实例对象用来读写拖拉事件中传输的数据，详见下文《DataTransfer 接口》的部分。

DataTransfer 接口概述

所有拖拉事件的实例都有一个`DragEvent.dataTransfer`属性，用来读写需要传递的数据。这个属性的值是一个`DataTransfer`接口的实例。

浏览器原生提供一个`DataTransfer()`构造函数，用来生成`DataTransfer`实例对象。

```javascript
var dataTrans = new DataTransfer();
```

`DataTransfer()`构造函数不接受参数。

拖拉的数据分成两方面：数据的种类（又称格式）和数据的值。数据的种类是一个 MIME 字符串（比如`text/plain`、`image/jpeg`），数据的值是一个字符串。一般来说，如果拖拉一段文本，则数据默认就是那段文本；如果拖拉一个链接，则数据默认就是链接的 URL。

拖拉事件开始时，开发者可以提供数据类型和数据值。拖拉过程中，开发者通过`dragenter`和`dragover`事件的监听函数，检查数据类型，以确定是否允许放下（drop）被拖拉的对象。比如，在只允许放下链接的区域，检查拖拉的数据类型是否为`text/uri-list`。

发生`drop`事件时，监听函数取出拖拉的数据，对其进行处理。

DataTransfer 的实例属性

DataTransfer.dropEffect

`DataTransfer.dropEffect`属性用来设置放下（drop）被拖拉节点时的效果，会影响到拖拉经过相关区域时鼠标的形状。它可能取下面的值。

- copy：复制被拖拉的节点
- move：移动被拖拉的节点

- link：创建指向被拖拉的节点的链接
- none：无法放下被拖拉的节点

除了上面这些值，设置其他的值都是无效的。

```javascript
target.addEventListener('dragover', function (e) {
  e.preventDefault();
  e.stopPropagation();
  e.dataTransfer.dropEffect = 'copy';
});
```

上面代码中，被拖拉元素一旦`drop`，接受的区域会复制该节点。

`dropEffect`属性一般在`dragenter`和`dragover`事件的监听函数中设置，对于`dragstart`、`drag`、`dragleave`这三个事件，该属性不起作用。因为该属性只对接受被拖拉的节点的区域有效，对被拖拉的节点本身是无效的。进入目标区域后，拖拉行为会初始化成设定的效果。

DataTransfer.effectAllowed

`DataTransfer.effectAllowed`属性设置本次拖拉中允许的效果。它可能取下面的值。

- copy：复制被拖拉的节点
- move：移动被拖拉的节点
- link：创建指向被拖拉节点的链接
- copyLink：允许`copy`或`link`
- copyMove：允许`copy`或`move`
- linkMove：允许`link`或`move`
- all：允许所有效果
- none：无法放下被拖拉的节点
- uninitialized：默认值，等同于`all`

如果某种效果是不允许的，用户就无法在目标节点中达成这种效果。

这个属性与`dropEffect`属性是同一件事的两个方面。前者设置被拖拉的节点允许的效果，后者设置接受拖拉的区域的效果，它们往往配合使用。

`dragstart`事件的监听函数，可以用来设置这个属性。其他事件的监听函数里面设置这个属性是无效的。

```javascript
source.addEventListener('dragstart', function (e) {
  e.dataTransfer.effectAllowed = 'move';
});
```

```
target.addEventListener('dragover', function (e) {
  ev.dataTransfer.dropEffect = 'move';
});
```

只要`dropEffect`属性和`effectAllowed`属性之中，有一个为`none`，就无法在目标节点上完成`drop`操作。

DataTransfer.files

`DataTransfer.files`属性是一个 FileList 对象，包含一组本地文件，可以用来在拖拉操作中传送。如果本次拖拉不涉及文件，则该属性为空的 FileList 对象。

下面就是一个接收拖拉文件的例子。

```javascript
// HTML 代码如下
// <div id="output" style="min-height: 200px;border: 1px solid black;">
//   文件拖拉到这里
// </div>

var div = document.getElementById('output');

div.addEventListener("dragenter", function( event ) {
  div.textContent = '';
  event.stopPropagation();
  event.preventDefault();
}, false);

div.addEventListener("dragover", function( event ) {
  event.stopPropagation();
  event.preventDefault();
}, false);

div.addEventListener("drop", function( event ) {
  event.stopPropagation();
  event.preventDefault();
  var files = event.dataTransfer.files;
  for (var i = 0; i < files.length; i++) {
    div.textContent += files[i].name + ' ' + files[i].size + '字节\n';
  }
}, false);
```

上面代码中，通过`dataTransfer.files`属性读取被拖拉的文件的信息。如果想要读取文件内容，就要使用`FileReader`对象。

```javascript
div.addEventListener('drop', function(e) {
  e.preventDefault();
```

```
  e.stopPropagation();

  var fileList = e.dataTransfer.files;
  if (fileList.length > 0) {
    var file = fileList[0];
    var reader = new FileReader();
    reader.onloadend = function(e) {
      if (e.target.readyState === FileReader.DONE) {
        var content = reader.result;
        div.innerHTML = 'File: ' + file.name + '\n\n' + content;
      }
    }
    reader.readAsBinaryString(file);
  }
});
```

DataTransfer.types

`DataTransfer.types`属性是一个只读的数组，每个成员是一个字符串，里面是拖拉的数据格式（通常是 MIME 值）。比如，如果拖拉的是文字，对应的成员就是`text/plain`。

下面是一个例子，通过检查`dataTransfer`属性的类型，决定是否允许在当前节点执行`drop`操作。

```javascript
function contains(list, value){
  for (var i = 0; i < list.length; ++i) {
    if(list[i] === value) return true;
  }
  return false;
}

function doDragOver(event) {
  var isLink = contains(event.dataTransfer.types, 'text/uri-list');
  if (isLink) event.preventDefault();
}
```

上面代码中，只有当被拖拉的节点是一个链接时，才允许在当前节点放下。

DataTransfer.items

`DataTransfer.items`属性返回一个类似数组的只读对象（DataTransferItemList 实例），每个成员就是本次拖拉的一个对象（DataTransferItem 实例）。如果本次拖拉不包含对象，则返回一个空对象。

DataTransferItemList 实例具有以下的属性和方法。

- `length`：返回成员的数量
- `add(data, type)`：增加一个指定内容和类型（比如`text/html`和`text/plain`）的字符串作为成员

- `add(file)`：`add`方法的另一种用法，增加一个文件作为成员
- `remove(index)`：移除指定位置的成员
- `clear()`：移除所有的成员

DataTransferItem 实例具有以下的属性和方法。

- `kind`：返回成员的种类（`string`还是`file`）。
- `type`：返回成员的类型（通常是 MIME 值）。
- `getAsFile()`：如果被拖拉是文件，返回该文件，否则返回`null`。
- `getAsString(callback)`：如果被拖拉的是字符串，将该字符传入指定的回调函数处理。该方法是异步的，所以需要传入回调函数。

下面是一个例子。

```javascript
div.addEventListener('drop', function (e) {
  e.preventDefault();
  if (e.dataTransfer.items != null) {
    for (var i = 0; i < e.dataTransfer.items.length; i++) {
      console.log(e.dataTransfer.items[i].kind + ': ' + e.dataTransfer.items[i].type);
    }
  }
});
```

DataTransfer 的实例方法

DataTransfer.setData()

`DataTransfer.setData()`方法用来设置拖拉事件所带有的数据。该方法没有返回值。

```javascript
event.dataTransfer.setData('text/plain', 'Text to drag');
```

上面代码为当前的拖拉事件加入纯文本数据。

该方法接受两个参数，都是字符串。第一个参数表示数据类型（比如`text/plain`），第二个参数是具体数据。如果指定类型的数据在`dataTransfer`属性不存在，那么这些数据将被加入，否则原有的数据将被新数据替换。

如果是拖拉文本框或者拖拉选中的文本，会默认将对应的文本数据，添加到`dataTransfer`属性，不用手动指定。

```html
<div draggable="true">
  aaa
```

```

上面代码中，拖拉这个`<div>`元素会自动带上文本数据`aaa`。

使用`setData`方法，可以替换到原有数据。

```html
<div
  draggable="true"
  ondragstart="event.dataTransfer.setData('text/plain', 'bbb')"
>
  aaa
</div>
```

上面代码中，拖拉数据实际上是`bbb`，而不是`aaa`。

下面是添加其他类型的数据。由于`text/plain`是最普遍支持的格式，为了保证兼容性，建议最后总是保存一份纯文本格式的数据。

```javascript
var dt = event.dataTransfer;

// 添加链接
dt.setData('text/uri-list', 'http://www.example.com');
dt.setData('text/plain', 'http://www.example.com');

// 添加 HTML 代码
dt.setData('text/html', 'Hello there, <strong>stranger</strong>');
dt.setData('text/plain', 'Hello there, <strong>stranger</strong>');

// 添加图像的 URL
dt.setData('text/uri-list', imageurl);
dt.setData('text/plain', imageurl);
```

可以一次提供多种格式的数据。

```javascript
var dt = event.dataTransfer;
dt.setData('application/x-bookmark', bookmarkString);
dt.setData('text/uri-list', 'http://www.example.com');
dt.setData('text/plain', 'http://www.example.com');
```

上面代码中，通过在同一个事件上面，存放三种类型的数据，使得拖拉事件可以在不同的对象上面，`drop`不同的值。注意，第一种格式是一个自定义格式，浏览器默认无法读取，这意味着，只有某个部署了特定代码的节点，才可能`drop`（读取到）这个数据。
```

DataTransfer.getData()

`DataTransfer.getData()`方法接受一个字符串（表示数据类型）作为参数，返回事件所带的指定类型的数据（通常是用`setData`方法添加的数据）。如果指定类型的数据不存在，则返回空字符串。通常只有`drop`事件触发后，才能取出数据。

下面是一个`drop`事件的监听函数，用来取出指定类型的数据。

```javascript
function onDrop(event) {
  var data = event.dataTransfer.getData('text/plain');
  event.target.textContent = data;
  event.preventDefault();
}
```

上面代码取出拖拉事件的文本数据，将其替换成当前节点的文本内容。注意，这时还必须取消浏览器的默认行为，因为假如用户拖拉的是一个链接，浏览器默认会在当前窗口打开这个链接。

`getData`方法返回的是一个字符串，如果其中包含多项数据，就必须手动解析。

```javascript
function doDrop(event) {
  var lines = event.dataTransfer.getData('text/uri-list').split('\n');
  for (let line of lines) {
    let link = document.createElement('a');
    link.href = line;
    link.textContent = line;
    event.target.appendChild(link);
  }
  event.preventDefault();
}
```

上面代码中，`getData`方法返回的是一组链接，就必须自行解析。

类型值指定为`URL`，可以取出第一个有效链接。

```javascript
var link = event.dataTransfer.getData('URL');
```

下面的例子是从多种类型的数据里面取出数据。

```javascript
function doDrop(event) {
  var types = event.dataTransfer.types;
  var supportedTypes = ['text/uri-list', 'text/plain'];
  types = supportedTypes.filter(function (value) { types.includes(value) });
```

```
  if (types.length) {
    var data = event.dataTransfer.getData(types[0]);
  }
  event.preventDefault();
}
```

DataTransfer.clearData()

`DataTransfer.clearData()`方法接受一个字符串（表示数据类型）作为参数，删除事件所带的指定类型的数据。如果没有指定类型，则删除所有数据。如果指定类型不存在，则调用该方法不会产生任何效果。

```javascript
event.dataTransfer.clearData('text/uri-list');
```

上面代码清除事件所带的`text/uri-list`类型的数据。

该方法不会移除拖拉的文件，因此调用该方法后，`DataTransfer.types`属性可能依然会返回`Files`类型（前提是存在文件拖拉）。

注意，该方法只能在`dragstart`事件的监听函数之中使用，因为这是拖拉操作的数据唯一可写的时机。

DataTransfer.setDragImage()

拖动过程中（`dragstart`事件触发后），浏览器会显示一张图片跟随鼠标一起移动，表示被拖动的节点。这张图片是自动创造的，通常显示为被拖动节点的外观，不需要自己动手设置。

`DataTransfer.setDragImage()`方法可以自定义这张图片。它接受三个参数。第一个是`<img>`节点或者`<canvas>`节点，如果省略或为`null`，则使用被拖动的节点的外观；第二个和第三个参数为鼠标相对于该图片左上角的横坐标和右坐标。

下面是一个例子。

```javascript
/* HTML 代码如下
 <div id="drag-with-image" class="dragdemo" draggable="true">
   drag me
 </div>
*/

var div = document.getElementById('drag-with-image');
div.addEventListener('dragstart', function (e) {
  var img = document.createElement('img');
  img.src = 'http://path/to/img';
  e.dataTransfer.setDragImage(img, 0, 0);
```

```
}, false);
```