进度事件

1. 进度事件的种类

进度事件用来描述资源加载的进度，主要由 AJAX 请求、<img>、<audio>、<video>、<style>、<link>等外部资源的加载触发，继承了 ProgressEvent 接口。它主要包含以下几种事件。

- abort：外部资源中止加载时（比如用户取消）触发。如果发生错误导致中止，不会触发该事件。
- error：由于错误导致外部资源无法加载时触发。
- load：外部资源加载成功时触发。
- loadstart：外部资源开始加载时触发。
- loadend：外部资源停止加载时触发，发生顺序排在 error、abort、load 等事件的后面。
- progress：外部资源加载过程中不断触发。
- timeout：加载超时时触发。

注意，除了资源下载，文件上传也存在这些事件。

下面是一个例子。

```
image.addEventListener('load', function (event) {
  image.classList.add('finished');
});

image.addEventListener('error', function (event) {
  image.style.display = 'none';
});
```

上面代码在图片元素加载完成后，为图片元素添加一个 finished 的 Class。如果加载失败，就把图片元素的样式设置为不显示。

有时候，图片加载会在脚本运行之前就完成，尤其是当脚本放置在网页底部的时候，因此有可能 load 和 error 事件的监听函数根本不会执行。所以，比较可靠的方式，是用 complete 属性先判断一下是否加载完成。

```
function loaded() {
  // ...
}

if (image.complete) {
  loaded();
} else {
  image.addEventListener('load', loaded);
}
```

由于 DOM 的元素节点没有提供是否加载错误的属性，所以 error 事件的监听函数最好放在 <img> 元素的 HTML 代码中，这样才能保证发生加载错误时百分之百会执行。

```
<img src="/wrong/url" onerror="this.style.display='none';" />
```

`loadend` 事件的监听函数，可以用来取代 `abort` 事件、`load` 事件、`error` 事件的监听函数，因为它总是在这些事件之后发生。

```
req.addEventListener('loadend', loadEnd, false);

function loadEnd(e) {
  console.log('传输结束，成功失败未知');
}
```

`loadend` 事件本身不提供关于进度结束的原因，但可以用它来做所有加载结束场景都需要做的一些操作。

另外，`error` 事件有一个特殊的性质，就是不会冒泡。所以，子元素的 `error` 事件，不会触发父元素的 `error` 事件监听函数。

2. ProgressEvent 接口

`ProgressEvent` 接口主要用来描述外部资源加载的进度，比如 AJAX 加载、`<img>`、`<video>`、`<style>`、`<link>` 等外部资源加载。进度相关的事件都继承了这个接口。

浏览器原生提供了 `ProgressEvent()` 构造函数，用来生成事件实例。

```
new ProgressEvent(type, options)
```

`ProgressEvent()` 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，这个参数是必须的。第二个参数是一个配置对象，表示事件的属性，该参数可选。配置对象除了可以使用 `Event` 接口的配置属性，还可以使用下面的属性，所有这些属性都是可选的。

- `lengthComputable`：布尔值，表示加载的总量是否可以计算，默认是false。
- `loaded`：整数，表示已经加载的量，默认是0。
- `total`：整数，表示需要加载的总量，默认是0。

`ProgressEvent` 具有对应的实例属性。`ProgressEvent.lengthComputable`、`ProgressEvent.loaded`、`ProgressEvent.total`

如果 `ProgressEvent.lengthComputable` 为 `false` ，`ProgressEvent.total` 实际上是没有意义的。

下面是一个例子。

```
var p = new ProgressEvent('load', {
  lengthComputable: true,
  loaded: 30,
  total: 100,
});
```

```
document.body.addEventListener('load', function (e) {
  console.log('已经加载: ' + (e.loaded / e.total) * 100 + '%');
});

document.body.dispatchEvent(p);
// 已经加载: 30%
```

上面代码先构造一个 `load` 事件，抛出后被监听函数捕捉到。

下面是一个实际的例子。

```
var xhr = new XMLHttpRequest();

xhr.addEventListener('progress', updateProgress, false);
xhr.addEventListener('load', transferComplete, false);
xhr.addEventListener('error', transferFailed, false);
xhr.addEventListener('abort', transferCanceled, false);

xhr.open();

function updateProgress(e) {
  if (e.lengthComputable) {
    var percentComplete = e.loaded / e.total;
  } else {
    console.log('不能计算进度');
  }
}

function transferComplete(e) {
  console.log('传输结束');
}

function transferFailed(evt) {
  console.log('传输过程中发生错误');
}

function transferCanceled(evt) {
  console.log('用户取消了传输');
}
```

上面是下载过程的进度事件，还存在上传过程的进度事件。这时所有监听函数都要放在 `XMLHttpRequest.upload` 对象上面。

```
var xhr = new XMLHttpRequest();

xhr.upload.addEventListener('progress', updateProgress, false);
xhr.upload.addEventListener('load', transferComplete, false);
xhr.upload.addEventListener('error', transferFailed, false);
```

```
xhr.upload.addEventListener('abort', transferCanceled, false);

xhr.open();
```