看起来很简单，其实并非所有对象都会经历所有 readystate 阶段。文档中说有些对象会完全跳过某个阶段，但并未说明哪些阶段适用于哪些对象。这意味着 readystatechange 事件经常会触发不到 4 次，而 readyState 未必会依次呈现上述值。

在 document 上使用时，值为"interactive"的 readyState 首先会触发 readystatechange 事件，时机类似于 DOMContentLoaded。进入交互阶段，意味着 DOM 树已加载完成，因而可以安全地交互了。此时图片和其他外部资源不一定都加载完了。可以像下面这样使用 readystatechange 事件：

```
document.addEventListener("readystatechange", (event) => {
  if (document.readyState == "interactive") {
    console.log("Content loaded");
  }
});
```

这个事件的 event 对象中没有任何额外的信息，连事件目标都不会设置。

在与 load 事件共同使用时，这个事件的触发顺序不能保证。在包含特别多或较大外部资源的页面中，交互阶段会在 load 事件触发前先触发。而在包含较少且较小外部资源的页面中，这个 readystatechange 事件有可能在 load 事件触发后才触发。

让问题变得更加复杂的是，交互阶段与完成阶段的顺序也不是固定的。在外部资源较多的页面中，很可能交互阶段会早于完成阶段，而在外部资源较少的页面中，很可能完成阶段会早于交互阶段。因此，实践中为了抢到较早的时机，需要同时检测交互阶段和完成阶段。比如：

```
document.addEventListener("readystatechange", (event) => {
  if (document.readyState == "interactive" ||
      document.readyState == "complete") {
    document.removeEventListener("readystatechange", arguments.callee);
    console.log("Content loaded");
  }
});
```

当 readystatechange 事件触发时，这段代码会检测 document.readyState 属性，以确定当前是不是交互或完成状态。如果是，则移除事件处理程序，以保证其他阶段不再执行。注意，因为这里的事件处理程序是匿名函数，所以使用了 arguments.callee 作为函数指针。然后，又打印出一条表示内容已加载的消息。这样的逻辑可以保证尽可能接近使用 DOMContentLoaded 事件的效果。

> **注意**　使用 readystatechange 只能尽量模拟 DOMContentLoaded，但做不到分毫不差。load 事件和 readystatechange 事件发生的顺序在不同页面中是不一样的。

5. pageshow 与 pagehide 事件

Firefox 和 Opera 开发了一个名为**往返缓存**（bfcache，back-forward cache）的功能，此功能旨在使用浏览器“前进”和“后退”按钮时加快页面之间的切换。这个缓存不仅存储页面数据，也存储 DOM 和 JavaScript 状态，实际上是把整个页面都保存在内存里。如果页面在缓存中，那么导航到这个页面时就不会触发 load 事件。通常，这不会导致什么问题，因为整个页面状态都被保存起来了。不过，Firefx 决定提供一些事件，把往返缓存的行为暴露出来。

第一个事件是 pageshow，其会在页面显示时触发，无论是否来自往返缓存。在新加载的页面上，pageshow 会在 load 事件之后触发；在来自往返缓存的页面上，pageshow 会在页面状态完全恢复后触发。注意，虽然这个事件的目标是 document，但事件处理程序必须添加到 window 上。下面的例子

展示了追踪这些事件的代码：

```
(function() {
  let showCount = 0;

  window.addEventListener("load", () => {
    console.log("Load fired");
  });

  window.addEventListener("pageshow", () => {
    showCount++;
    console.log(`Show has been fired ${showCount} times.`);
  });
})();
```

这个例子使用了私有作用域来保证 showCount 变量不进入全局作用域。在页面首次加载时，showCount 的值为 0。之后每次触发 pageshow 事件，showCount 都会加 1 并输出消息。如果从包含以上代码的页面跳走，然后又点击“后退”按钮返回以恢复它，就能够每次都看到 showCount 递增的值。这是因为变量的状态连同整个页面状态都保存在了内存中，导航回来后可以恢复。如果是点击了浏览器的“刷新”按钮，则 showCount 的值会重置为 0，因为页面会重新加载。

除了常用的属性，pageshow 的 event 对象中还包含一个名为 persisted 的属性。这个属性是一个布尔值，如果页面存储在了往返缓存中就是 true，否则就是 false。可以像下面这样在事件处理程序中检测这个属性：

```
(function() {
  let showCount = 0;

  window.addEventListener("load", () => {
    console.log("Load fired");
  });

  window.addEventListener("pageshow", () => {
    showCount++;
    console.log(`Show has been fired ${showCount} times.`,
                `Persisted? ${event.persisted}`);
  });
})();
```

通过检测 persisted 属性可以根据页面是否取自往返缓存而决定是否采取不同的操作。

与 pageshow 对应的事件是 pagehide，这个事件会在页面从浏览器中卸载后，在 unload 事件之前触发。与 pageshow 事件一样，pagehide 事件同样是在 document 上触发，但事件处理程序必须被添加到 window。event 对象中同样包含 persisted 属性，但用法稍有不同。比如，以下代码检测了 event.persisted 属性：

```
window.addEventListener("pagehide", (event) => {
  console.log("Hiding. Persisted? " + event.persisted);
});
```

这样，当 pagehide 事件触发时，也许可以根据 persisted 属性的值来采取一些不同的操作。对 pageshow 事件来说，persisted 为 true 表示页面是从往返缓存中加载的；而对 pagehide 事件来说，persisted 为 true 表示页面在卸载之后会被保存在往返缓存中。因此，第一次触发 pageshow 事件时 persisted 始终是 false，而第一次触发 pagehide 事件时 persisted 始终是 true（除非页面不符合使用往返缓存的条件）。

> **注意**　注册了 onunload 事件处理程序（即使是空函数）的页面会自动排除在往返缓存之外。这是因为 onunload 事件典型的使用场景是撤销 onload 事件发生时所做的事情，如果使用往返缓存，则下一次页面显示时就不会触发 onload 事件，而这可能导致页面无法使用。

6. hashchange 事件

HTML5 增加了 hashchange 事件，用于在 URL 散列值（URL 最后#后面的部分）发生变化时通知开发者。这是因为开发者经常在 Ajax 应用程序中使用 URL 散列值存储状态信息或路由导航信息。

onhashchange 事件处理程序必须添加给 window，每次 URL 散列值发生变化时会调用它。event 对象有两个新属性：oldURL 和 newURL。这两个属性分别保存变化前后的 URL，而且是包含散列值的完整 URL。下面的例子展示了如何获取变化前后的 URL：

```
window.addEventListener("hashchange", (event) => {
  console.log(`Old URL: ${event.oldURL}, New URL: ${event.newURL}`);
});
```

如果想确定当前的散列值，最好使用 location 对象：

```
window.addEventListener("hashchange", (event) => {
  console.log(`Current hash: ${location.hash}`);
});
```

17.4.8　设备事件

随着智能手机和平板计算机的出现，用户与浏览器交互的新方式应运而生。为此，一批新事件被发明了出来。**设备事件**可以用于确定用户使用设备的方式。W3C 在 2011 年就开始起草一份新规范，用于定义新设备及设备相关的事件。

1. orientationchange 事件

苹果公司在移动 Safari 浏览器上创造了 orientationchange 事件，以方便开发者判断用户的设备是处于垂直模式还是水平模式。移动 Safari 在 window 上暴露了 window.orientation 属性，它有以下 3 种值之一：0 表示垂直模式，90 表示左转水平模式（主屏幕键在右侧），–90 表示右转水平模式（主屏幕键在左）。虽然相关文档也提及设备倒置后的值为 180，但设备本身至今还不支持。图 17-9 展示了 window.orientation 属性的各种值。

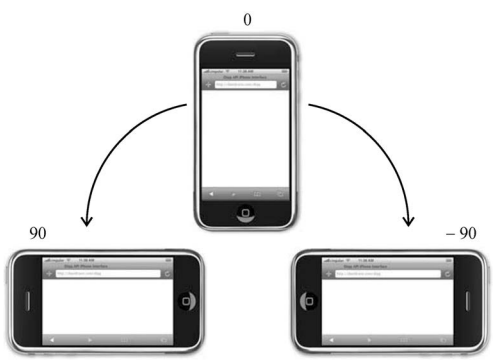

图　17-9

每当用户旋转设备改变了模式，就会触发 orientationchange 事件。但 event 对象上没有暴露任何有用的信息，这是因为相关信息都可以从 window.orientation 属性中获取。以下是这个事件典型的用法：

```
window.addEventListener("load", (event) => {
  let div = document.getElementById("myDiv");
  div.innerHTML = "Current orientation is " + window.orientation;

  window.addEventListener("orientationchange", (event) => {
    div.innerHTML = "Current orientation is " + window.orientation;
  });
});
```

这个例子会在 load 事件触发时显示设备初始的朝向。然后，又指定了 orientationchange 事件处理程序。此后，只要这个事件触发，页面就会更新以显示新的朝向信息。

所有 iOS 设备都支持 orientationchange 事件和 window.orientation 属性。

> **注意** 因为 orientationchange 事件被认为是 window 事件，所以也可以通过给<body>元素添加 onorientationchange 属性来指定事件处理程序。

2. deviceorientation 事件

deviceorientation 是 DeviceOrientationEvent 规范定义的事件。如果可以获取设备的加速计信息，而且数据发生了变化，这个事件就会在 window 上触发。要注意的是，deviceorientation 事件只反映设备在空间中的朝向，而不涉及移动相关的信息。

设备本身处于 3D 空间即拥有 *x* 轴、*y* 轴和 *z* 轴的坐标系中。如果把设备静止放在水平的表面上，那么三轴的值均为 0，其中，*x* 轴方向为从设备左侧到右侧，*y* 轴方向为从设备底部到上部，*z* 轴方向为从设备背面到正面，如图 17-10 所示。

图 17-10

当 deviceorientation 触发时，event 对象中会包含各个轴相对于设备静置时坐标值的变化，主要是以下 5 个属性。

- alpha：0~360 范围内的浮点值，表示围绕 *z* 轴旋转时 *y* 轴的度数（左右转）。
- beta：–180~180 范围内的浮点值，表示围绕 *x* 轴旋转时 *z* 轴的度数（前后转）。
- gamma：–90~90 范围内的浮点值，表示围绕 *y* 轴旋转时 *z* 轴的度数（扭转）。
- absolute：布尔值，表示设备是否返回绝对值。
- compassCalibrated：布尔值，表示设备的指南针是否正确校准。

图 17-11 展示了 alpha、beta 和 gamma 值的计算方式。

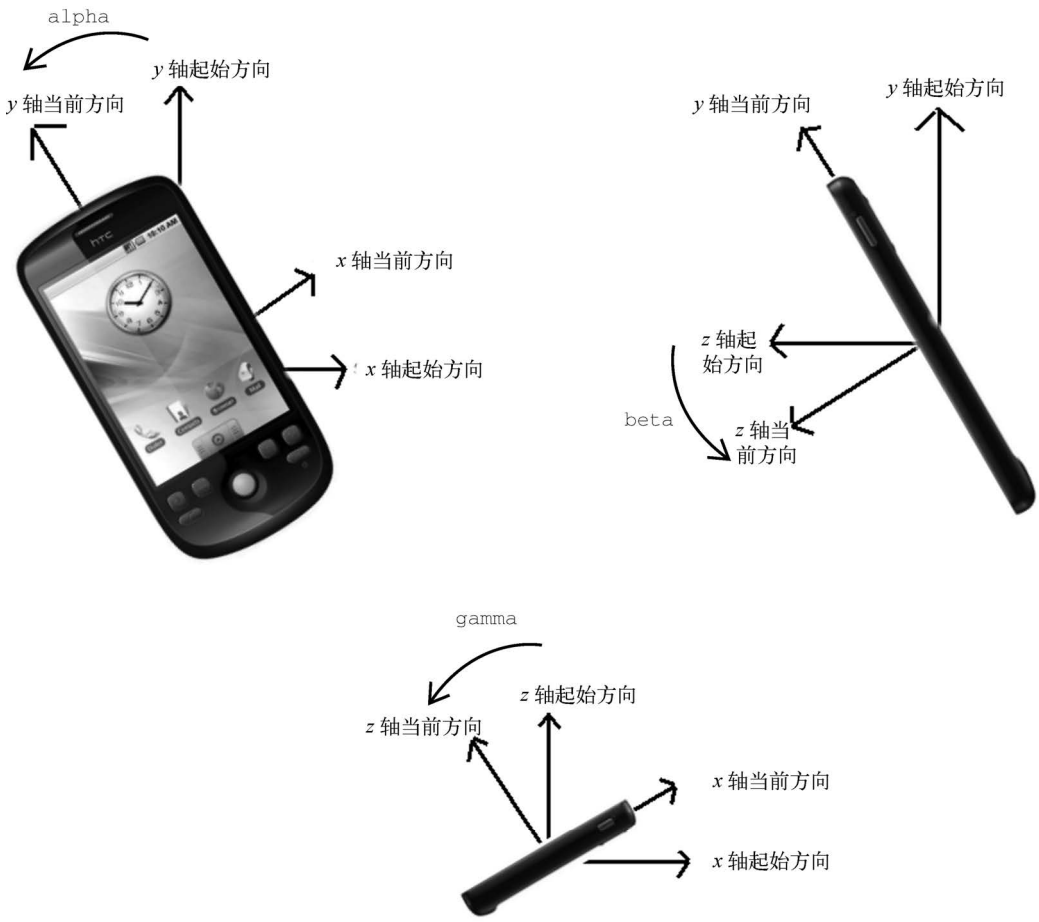

图　17-11

下面是一个输出 alpha、beta 和 gamma 值的简单例子：

```
window.addEventListener("deviceorientation", (event) => {
  let output = document.getElementById("output");
  output.innerHTML =
    `Alpha=${event.alpha}, Beta=${event.beta}, Gamma=${event.gamma}<br>`;
});
```

基于这些信息，可以随着设备朝向的变化重新组织或修改屏幕上显示的元素。例如，以下代码会随着朝向变化旋转一个元素：

```
window.addEventListener("deviceorientation", (event) => {
  let arrow = document.getElementById("arrow");
  arrow.style.webkitTransform = `rotate(${Math.round(event.alpha)}deg)`;
});
```

这个例子只适用于移动 WebKit 浏览器，因为使用的是专有的 webkitTransform 属性（CSS 标准的 transform 属性的临时版本）。“箭头”（arrow）元素会随着 event.alpha 值的变化而变化，呈现出指南针的样子。这里给 CSS3 旋转变形函数传入了四舍五入后的值，以确保平顺。

3. devicemotion 事件

DeviceOrientationEvent 规范也定义了 devicemotion 事件。这个事件用于提示设备实际上在移动，而不仅仅是改变了朝向。例如，devicemotion 事件可以用来确定设备正在掉落或者正拿在一个行走的人手里。

当 devicemotion 事件触发时，event 对象中包含如下额外的属性。

- acceleration：对象，包含 x、y 和 z 属性，反映不考虑重力情况下各个维度的加速信息。
- accelerationIncludingGravity：对象，包含 x、y 和 z 属性，反映各个维度的加速信息，包含 *z* 轴自然重力加速度。
- interval：毫秒，距离下次触发 devicemotion 事件的时间。此值在事件之间应为常量。
- rotationRate：对象，包含 alpha、beta 和 gamma 属性，表示设备朝向。

如果无法提供 acceleration、accelerationIncludingGravity 和 rotationRate 信息，则属性值为 null。为此，在使用这些属性前必须先检测它们的值是否为 null。比如：

```
window.addEventListener("devicemotion", (event) => {
  let output = document.getElementById("output");
  if (event.rotationRate !== null) {
    output.innerHTML += `Alpha=${event.rotationRate.alpha}` +
                        `Beta=${event.rotationRate.beta}` +
                        `Gamma=${event.rotationRate.gamma}`;
  }
});
```

17.4.9 触摸及手势事件

Safari 为 iOS 定制了一些专有事件，以方便开发者。因为 iOS 设备没有鼠标和键盘，所以常规的鼠标和键盘事件不足以创建具有完整交互能力的网页。同时，WebKit 也为 Android 定制了很多专有事件，成为了事实标准，并被纳入 W3C 的 Touch Events 规范。本节介绍的事件只适用于触屏设备。

1. 触摸事件

iPhone 3G 发布时，iOS 2.0 内置了新版本的 Safari。这个新的移动 Safari 支持一些与触摸交互有关的新事件。后来的 Android 浏览器也实现了同样的事件。当手指放在屏幕上、在屏幕上滑动或从屏幕移开时，**触摸事件**即会触发。触摸事件有如下几种。

- touchstart：手指放到屏幕上时触发（即使有一个手指已经放在了屏幕上）。
- touchmove：手指在屏幕上滑动时连续触发。在这个事件中调用 preventDefault()可以阻止滚动。

- touchend：手指从屏幕上移开时触发。
- touchcancel：系统停止跟踪触摸时触发。文档中并未明确什么情况下停止跟踪。

这些事件都会冒泡，也都可以被取消。尽管触摸事件不属于 DOM 规范，但浏览器仍然以兼容 DOM 的方式实现了它们。因此，每个触摸事件的 event 对象都提供了鼠标事件的公共属性：bubbles、cancelable、view、clientX、clientY、screenX、screenY、detail、altKey、shiftKey、ctrlKey 和 metaKey。

除了这些公共的 DOM 属性，触摸事件还提供了以下 3 个属性用于跟踪触点。

- touches：Touch 对象的数组，表示当前屏幕上的每个触点。
- targetTouches：Touch 对象的数组，表示特定于事件目标的触点。
- changedTouches：Touch 对象的数组，表示自上次用户动作之后变化的触点。

每个 Touch 对象都包含下列属性。

- clientX：触点在视口中的 *x* 坐标。
- clientY：触点在视口中的 *y* 坐标。
- identifier：触点 ID。
- pageX：触点在页面上的 *x* 坐标。
- pageY：触点在页面上的 *y* 坐标。
- screenX：触点在屏幕上的 *x* 坐标。
- screenY：触点在屏幕上的 *y* 坐标。
- target：触摸事件的事件目标。

这些属性可用于追踪屏幕上的触摸轨迹。例如：

```
function handleTouchEvent(event) {
  // 只针对一个触点
  if (event.touches.length == 1) {
    let output = document.getElementById("output");
    switch(event.type) {
      case "touchstart":
        output.innerHTML += `<br>Touch started:` +
                            `(${event.touches[0].clientX}` +
                            ` ${event.touches[0].clientY})`;
        break;
      case "touchend":
        output.innerHTML += `<br>Touch ended:` +
                            `(${event.changedTouches[0].clientX}` +
                            ` ${event.changedTouches[0].clientY})`;
        break;
      case "touchmove":
        event.preventDefault(); // 阻止滚动
        output.innerHTML += `<br>Touch moved:` +
                            `(${event.changedTouches[0].clientX}` +
                            ` ${event.changedTouches[0].clientY})`;
        break;
    }
  }
}

document.addEventListener("touchstart", handleTouchEvent);
document.addEventListener("touchend", handleTouchEvent);
document.addEventListener("touchmove", handleTouchEvent);
```

以上代码会追踪屏幕上的一个触点。为简单起见，代码只会在屏幕有一个触点时输出信息。在 touchstart 事件触发时，触点的位置信息会输出到 output 元素中。在 touchmove 事件触发时，会取消默认行为以阻止滚动（移动触点通常会滚动页面），并输出变化的触点信息。在 touchend 事件触发时，会输出触点最后的信息。注意，touchend 事件触发时 touches 集合中什么也没有，这是因为没有滚动的触点了。此时必须使用 changedTouches 集合。

这些事件会在文档的所有元素上触发，因此可以分别控制页面的不同部分。当手指点触屏幕上的元素时，依次会发生如下事件（包括鼠标事件）：

(1) touchstart
(2) mouseover
(3) mousemove（1 次）
(4) mousedown
(5) mouseup
(6) click
(7) touchend

2. 手势事件

iOS 2.0 中的 Safari 还增加了一种手势事件。**手势事件**会在两个手指触碰屏幕且相对距离或旋转角度变化时触发。手势事件有以下 3 种。

- gesturestart：一个手指已经放在屏幕上，再把另一个手指放到屏幕上时触发。
- gesturechange：任何一个手指在屏幕上的位置发生变化时触发。
- gestureend：其中一个手指离开屏幕时触发。

只有在两个手指同时接触事件接收者时，这些事件才会触发。在一个元素上设置事件处理程序，意味着两个手指必须都在元素边界以内才能触发手势事件（这个元素就是事件目标）。因为这些事件会冒泡，所以也可以把事件处理程序放到文档级别，从而可以处理所有手势事件。使用这种方式时，事件的目标就是两个手指均位于其边界内的元素。

触摸事件和手势事件存在一定的关系。当一个手指放在屏幕上时，会触发 touchstart 事件。当另一个手指放到屏幕上时，gesturestart 事件会首先触发，然后紧接着触发这个手指的 touchstart 事件。如果两个手指或其中一个手指移动，则会触发 gesturechange 事件。只要其中一个手指离开屏幕，就会触发 gestureend 事件，紧接着触发该手指的 touchend 事件。

与触摸事件类似，每个手势事件的 event 对象都包含所有标准的鼠标事件属性：bubbles、cancelable、view、clientX、clientY、screenX、screenY、detail、altKey、shiftKey、ctrlKey 和 metaKey。新增的两个 event 对象属性是 rotation 和 scale。rotation 属性表示手指变化旋转的度数，负值表示逆时针旋转，正值表示顺时针旋转（从 0 开始）。scale 属性表示两指之间距离变化（对捏）的程度。开始时为 1，然后随着距离增大或缩小相应地增大或缩小。

可以像下面这样使用手势事件的属性：

```
function handleGestureEvent(event) {
  let output = document.getElementById("output");
  switch(event.type) {
    case "gesturestart":
      output.innerHTML += `Gesture started: ` +
                          `rotation=${event.rotation},` +
                          `scale=${event.scale}`;
```

```
        break;
      case "gestureend":
        output.innerHTML += `Gesture ended: ` +
                            `rotation=${event.rotation},` +
                            `scale=${event.scale}`;
        break;
      case "gesturechange":
        output.innerHTML += `Gesture changed: ` +
                            `rotation=${event.rotation},` +
                            `scale=${event.scale}`;
        break;
    }
}

document.addEventListener("gesturestart", handleGestureEvent, false);
document.addEventListener("gestureend", handleGestureEvent, false);
document.addEventListener("gesturechange", handleGestureEvent, false);
```

与触摸事件的例子一样，以上代码简单地将每个事件对应到一个处理函数，然后输出每个事件的信息。

> **注意**　触摸事件也会返回 rotation 和 scale 属性，但只在两个手指触碰屏幕时才会变化。一般来说，使用两个手指的手势事件比考虑所有交互的触摸事件使用起来更容易一些。

17.4.10　事件参考

本节给出了 DOM 规范、HTML5 规范，以及概述事件行为的其他当前已发布规范中定义的所有浏览器事件。这些事件按照 API 和/或规范分类。

> **注意**　只包含带厂商前缀事件的规范不在本参考中。

Ambient Light events
devicelight

App Cache events
cached
checking
downloading
noupdate
obsolete
updateready

Audio Channels API events
headphoneschange
mozinterruptbegin
mozinterruptend

Battery API events
chargingchange
chargingtimechange
dischargingtimechange
levelchange

Broadcast Channel API events
message

Channel Messaging API events
message

Clipboard API events
beforecopy
beforecut
beforepaste

copy
cut
paste

Contacts API events
contactchange
error
success

CSS Font Loading API events
loading
loadingdone
loadingerror

CSSOM events
animationend
animationiteration
animationstart
transitionend

CSSOM View events
resize
scroll

Device Orientation events
compassneedscalibration
devicemotion
deviceorientation

Device Storage API events
change

DOM events
abort
beforeinput
blur
click
compositionend
compositionstart
compositionupdate
dblclick
error
focus
focusin
focusout
input
keydown
keypress
keyup
load
mousedown
mouseenter
mouseleave
mousemove
mouseout
mouseover
mouseup
resize
scroll
select
unload
wheel

Download API events
statechange

Encrypted Media Extensions events
encrypted
keystatuschange
message
waitingforkey

Engineering Mode API events
message

File API events
abort
error
load
loadend
loadstart
progress

File System API events
error
writeend

FMRadio API events
antennaavailablechange
disabled