


```

var evt = e.type;
while (evt.length < 10) evt += ' ' +
    log(evt +
        ' keyCode=' + e.keyCode +
        ' which=' + e.which +
        ' charCode=' + e.charCode +
        ' char=' + String.fromCharCode(e.keyCode || e.charCode) +
        (e.shiftKey ? ' +shift' : '') +
        (e.ctrlKey ? ' +ctrl' : '') +
        (e.altKey ? ' +alt' : '') +
        (e.metaKey ? ' +meta' : ''));
if (document.getElementById(e.type + 'Stop').checked) {
    e.preventDefault ? e.preventDefault() : (e.returnValue = false);
}
}
function clearLog() {
    document.getElementById('keyLogger').value = '';
    document.getElementById('keyInput').focus();
}
function log(text) {
    var area = document.getElementById('keyLogger');
    area.value += text + '\n';
    area.scrollTop = area.scrollHeight;
}
</script>
</body>
</html>

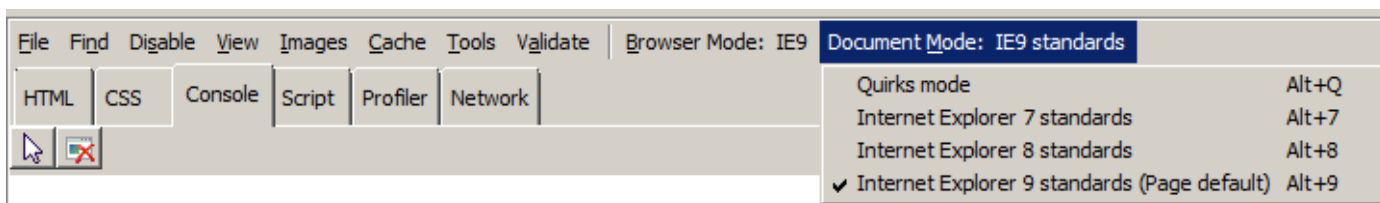
```

نکته: برای جلوگیری از اجرای مرورگرها در حالت Quirks حتما از تگ doctype در ابتدای فایل‌های html خود استفاده کنید. در غیر این صورت رفتارهای غیرمنتظره‌ای (مخصوصا در IE) مشاهده خواهید کرد. برای اجرای مرورگرها در حالت استاندارد html5 (بهترین حالت در حال حاضر) میتوانید از تگ زیر استفاده کنید:

```
<!doctype html>
```

دقت کنید که قبل از این خط هیچ چیز دیگری نوشته نشود وگرنه در IE از آن صرفنظر میشود!

یا اینکه در IE با استفاده از developer tools (دکمه F12) برای Document Mode گزینه‌ای غیر از Quirks mode (بهتر است از حالت IE9 یا بالاتر استفاده کنید) را انتخاب کنید.



برای کسب اطلاعات بیشتر راجع به doctype‌های مختلف و نیز حالت quirks میتوانید به [^](#) و [^](#) و [^](#) رجوع کنید. پیشنهاد میکنم که این منابع را حتما مطالعه کنید.

نکته: در کد بالا متد preventDefault در IE 8- تعریف نشده است (در واقع در IE تنها در نسخه 9 تعریف شده است). همچنین استفاده از پراپرتی returnValue در فایرفاکس و IE9 کار نمیکند! از این خط کد برای جلوگیری از رفتار پیشفرض رویداد استفاده شده است. همانطور که در ادامه میخوانید راه حل ساده‌تری نیز برای اینکار وجود دارد. متد String.fromCharCode برای نمایش کاراکتر کلید فشرده شده استفاده شده است. البته اگر کلید غیرکاراکتری فشرده شود ممکن است با نتایج غیرمنتظره‌ای روبرو شوید.

با استفاده از html تولیدی در مرورگرهای مختلف سعی کنید موارد زیر را آزمایش کنید:

کلیدهای کاراکتری چون { 6 | / a را بفشارید. در این حالت رویدادهای keydown و سپس keypress رخ خواهند داد. پس از رها کردن کلیدها نیز رویداد keyup رخ میدهد.

یکی از کلیدهای غیرکاراکتری مثل ctrl یا alt را بفشارید. در این حالت تنها رویدادهای keydown و keyup رخ خواهند داد و خبری از رویداد keypress نیست.

نکته : مرورگرهای FireFox و Opera در مورد بیشتر کلیدهای غیرکاراکتری نیز رویداد keypress را صدا خواهند زد! مرورگر IE این رفتار را تنها در مورد کلید Esc نشان میدهد. همچنین در IE و Opera کلید PrtScr هیچ رویدادی را فرا نمیخواند. ظاهراً تنها مرورگر Chrome بدرستی عمل میکند.

در حالت کلی فشردن کلیدهای غیرکاراکتری نباید رویداد keypress را فراخوانی کند.

بنابراین:

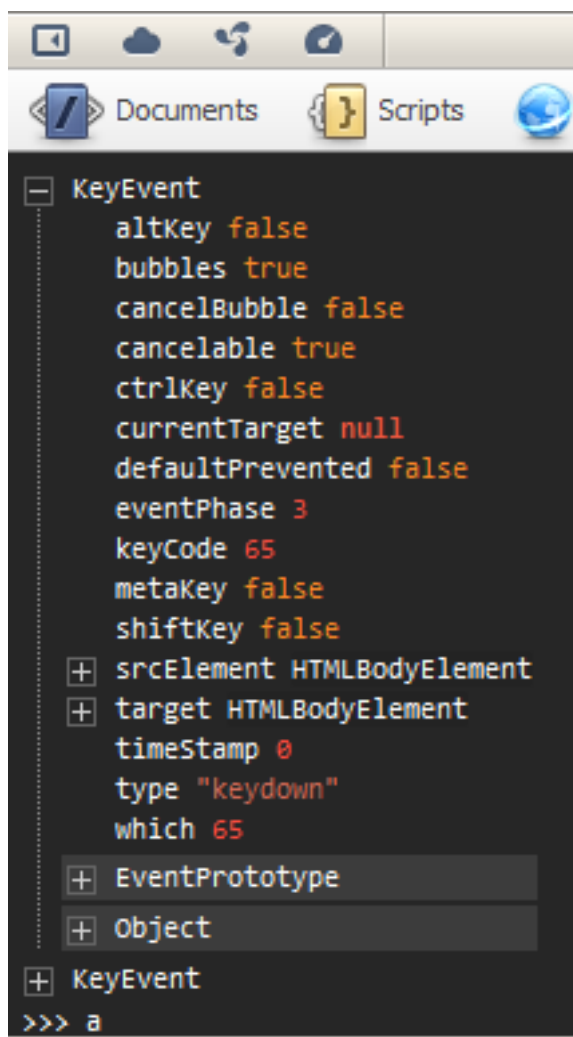
keydown و keyup برای همه کلیدها

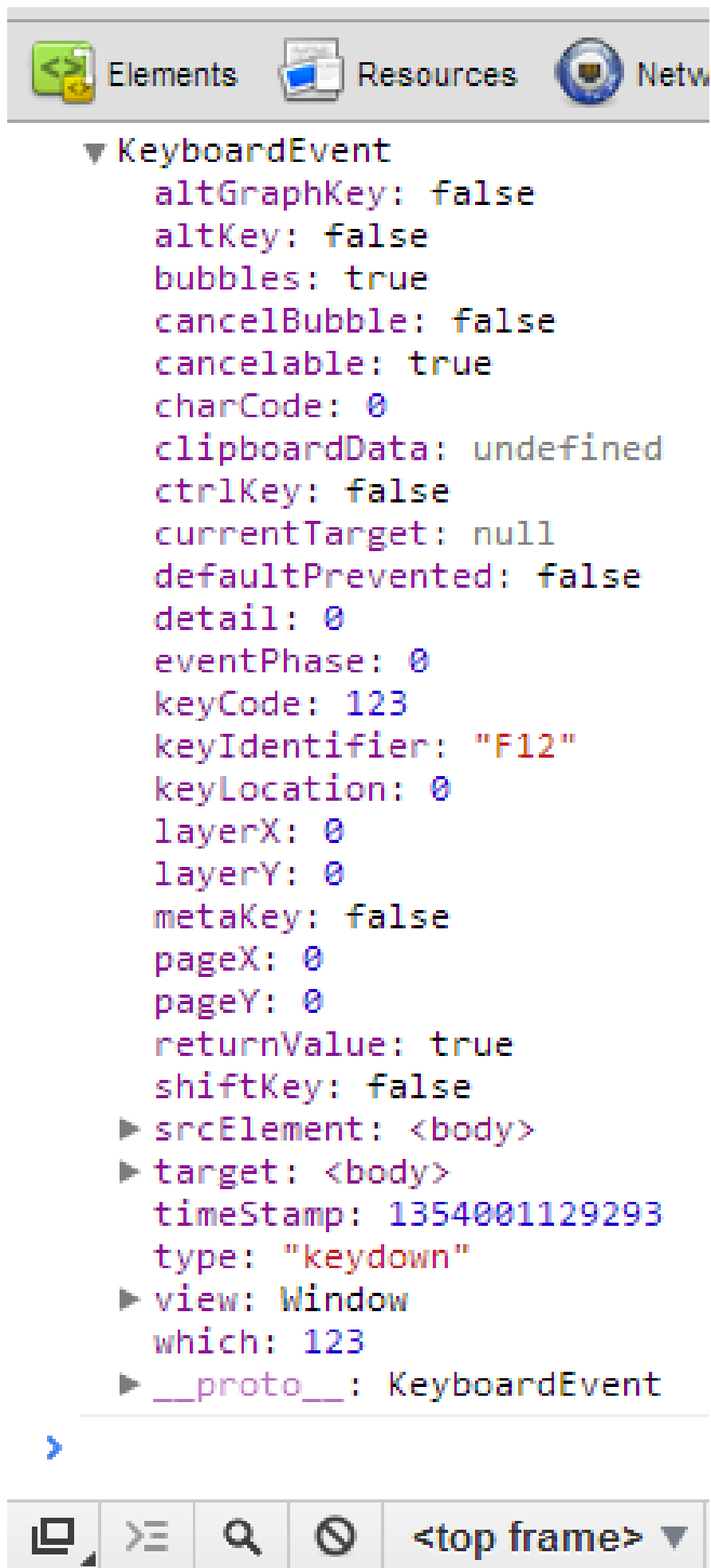
keypress برای کلیدهای کاراکتری

پراپرتی‌های رویدادهای کیبرد

برخلاف نسخه‌های قدیمی مرورگرها که هرکدام راه و روش خودشان را برای تعامل با این رویدادها برگزیده بودند، امروزه تمامی مرورگرها تقریباً از یک روش استاندارد و مشترک برای اینکار استفاده میکنند. در تصاویر زیر تمام اجزای آبجکت KeyboardEvent با استفاده از کد زیر در مرورگرهای فایرفاکس و IE نتایج جالبی مانند تصاویر زیر فراهم نمیکند!

```
document.onkeydown = function (e) {  
    e = e || event;  
    console.log(e);  
}
```





▼ KeyboardEvent

- altGraphKey: false
- altKey: false
- bubbles: true
- cancelBubble: false
- cancelable: true
- charCode: 0
- clipboardData: undefined
- ctrlKey: false
- currentTarget: null
- defaultPrevented: false
- detail: 0
- eventPhase: 0
- keyCode: 123
- keyIdentifier: "F12"
- keyLocation: 0
- layerX: 0
- layerY: 0
- metaKey: false
- pageX: 0
- pageY: 0
- returnValue: true
- shiftKey: false
- ▶ srcElement: <body>
- ▶ target: <body>
- timeStamp: 1354001129293
- type: "keydown"
- ▶ view: Window
- which: 123
- ▶ __proto__: KeyboardEvent

▶

🏠 > 🔍 ⌛ <top frame> ▼

همانطور که مشاهده میکنید تفاوتی بین مرورگرها در این آبجکت به چشم میخورد. برای کسب اطلاعات بیشتر در مورد این آبجکت و اجزای استاندارد آن در DOM Level 3 به [اینجا](#) مراجعه کنید. در ادامه به بررسی پراپرتی‌های مهم آرگومان این رویدادها (همان KeyboardEvent) میپردازیم.

keyCode

همان scan code کلید فشرده شده است. برای مثال اگر کلید a فشرده شود کاراکتر تولیدی ممکن است a یا A یا 'ش' (یا کاراکتری دیگر در زبانهای مختلف) باشد اما در تمامی حالات scan code مربوطه یا همان keyCode همیشه یکسان (65 برای کلید a) خواهد بود. این کد تنها به کلید فشرده شده بستگی دارد و نه به کاراکتر حاصله! البته در IE به هنگام رخ دادن رویداد keypress کد کاراکتر (همان char code) کلید فشرده شده در این پراپرتی قرار میگیرد! در این بین میان مرورگرهای مختلف تفاوتی وجود دارد که با یک جستجو در اینترنت میتوان به تمامی این کدها دسترسی پیدا کرد. خواندن مقاله کامل [JavaScript Madness: Keyboard Events](#) نیز خالی از لطف نیست.

charCode

همان کد ASCII (یا کد UTF-16 برای کاراکترهای یونیکد. اطلاعات بیشتر [^](#) و [^](#)) کاراکتر کلید فشرده شده است. ممکن است با keyCode برابر باشد. این پراپرتی در IE و Opera تعریف نشده است. در عمل ممکن است keyCode و charCode در پلتفرمهای مختلف و حتی بین سیستم عامل‌های مختلف در یک سخت افزار نتایج متفاوتی ارائه دهند. بنابراین آزمودن هر مورد مشکوک قبل از ریلیز نهایی محصول میتواند مفید باشد.

which

یک پراپرتی نسبتاً غیراستاندارد! است که ترکیبی از keyCode و charCode را برمیگرداند (اطلاعات بیشتر در [^](#) و [^](#)). این پراپرتی در IE تعریف نشده است.

shiftKey, ctrlKey, altKey, metaKey

پراپرتی‌هایی از نوع بولین که وضعیت کلیدهای Shift, Ctrl, Alt و Command (تنها در مک) را نشان میدهند. **نکته:** بدستن آوردن کد درست کاراکتر فشرده شده با توجه به وضعیت کلیدها و زبان انتخابی تنها با استفاده از رویداد keypress امکان پذیر است. با توجه به تفاوتی که بین مرورگرهای مختلف وجود دارد برای یافتن کاراکتر فشرده شده در رویداد keypress استفاده از متد زیر توصیه میشود:

```
function getCharacter(event) {
    if (event.which == null)
        return String.fromCharCode(event.keyCode); // IE
    else if (event.which != 0 && event.charCode != 0)
        return String.fromCharCode(event.which); // All others
    return null; // special key
}
```

توضیحات بیشتر و کاملتر این مبحث را میتوانید از [Document Object Model \(DOM\) Level 3 Events Specification](#) که آخرین نسخه آن در زمان تهیه این مطلب در تاریخ June 2012 14 انتشار یافته تهیه کنید. بطور ویژه برای مبحث رویدادهای کیبرد ([^](#)) اطلاعات بسیار بیشتری در دسترس است.

نکته: با توجه به اطلاعاتی که در سند فوق وجود دارد، به دلیل ماهیت همزمانی (Sync) رویدادهای کیبرد تا زمانی که تمام عملیات موجود در متد تعیین شده برای این رویدادها انجام نشود، رویداد بعدی (با توجه به ترتیبی که در ابتدای این مطلب آورده شده است) رخ نخواهد داد. برای تست این موضوع قطعه کد زیر را آماده کردم:

```
<html>
<body>
  <input id="inputText" type="text" />
  <script>
    document.onkeydown = keyDown;
    document.onkeypress = keyPress;
    document.onkeyup = keyUp;
    function keyDown(e) {
      var input = document.getElementById('inputText');
      input.value = 'keyDown started ...';
      input.disabled = true;
      var j = 0;
```

```

for (var i = 0; i < 999999999; i++) {
    j = i - j;
}
console.log(j);
//alert('keyDown');
input.value = 'keyDown finished.';
input.disabled = false;
}
function keyPress(e) {
    alert('keyPress');
    //console.log('keyPress');
}
function keyUp(e) {
    alert('keyUp');
    //console.log('keyUp');
}
}
</script>
</body>
</html>

```

اگر کد بالا در مرورگرهای مختلف امتحان کنید مشاهده میکنید که انجام عملیات سنگین در رویدادهای کیبرد موجب ایجاد وقفه در فراخوانی سایر رویدادها میشود.

با اجرای کد فوق در مرورگرهای مختلف نکات جالب زیر بدست آمد:

- در IE و کروم نمایش یک alert موجب از دست دادن فوکس document شده و بنابراین رویدادهای کیبرد بعد از نمایش alert کار نخواهند کرد! مثلاً اگر در keydown یک alert نمایش داده شود چون رویداد keyup بر روی پنجره alert رخ میدهد بنابراین keyup فراخوانی نمیشود ولی چون رویداد keypress با keydown همزمان است این اتفاق برای keypress نمی‌افتد. این مشکل در فایرفاکس پیش نمی‌آید. در اپرا در این حالت رویداد keypress هم رخ نمیدهد! البته رفتار IE در اجرای کد فوق کمی غیرمنتظره‌تر است و ظاهراً رویداد keypress هم رخ نمیدهد.

- در اجرای کد فوق در Firefox ظاهراً alert مربوط به keyup قبل از keypress نمایش داده میشود. البته اگر به جای alert از console.log (البته نیاز به نصب Firebug است) استفاده شود این به هم خوردگی ترتیب رویدادها وجود ندارد.

- در مرورگر Opera پس از فشردن کلید enter در نوار آدرس و پس از بارگذاری صفحه، فوکس بلافاصله به عنصر document سپرده میشود طوری‌که که رویداد keyup کلید enter در document بارگذاری شده فراخوانی میشود و در صورت سرعت بالای بارگذاری صفحه، کدهای مربوط به این رویداد اجرا میشوند. در سایر مرورگرها این مورد مشاهده نشد.

- ظاهراً در تمام مرورگرها به غیر Opera کدهای جاوا اسکریپت در ثرد UI اجرا شده و موجب قفل شدن document میشود. بنابراین وسط اجرای کدهای سنگین نمیتوان مثلاً خواص عناصر UI را تغییر داد. درواقع مرورگر اپرا برخلاف سایر مرورگرها، رفتار ویژه‌ای در برخورد با جاوا اسکریپت و رویدادها و تغییرات عناصر DOM دارد. برای کسب اطلاعات بیشتر در این زمینه به [اینجا](#) مراجعه کنید. درضمن این مبحث کمی پیچیده‌تر از آن است که به نظر می‌آید ([^](#)). برای بررسی بیشتر میتوانید کد زیر را در مرورگرهای مختلف آزمایش کنید:

```

<html>
<head>
<script type="text/javascript">
function process() {
    var above = 0, below = 0;
    for (var i = 0; i < 200000; i++) {
        if (Math.random() * 2 > 1) {
            above++;
        }
        else {
            below++;
        }
    }
}
function test() {
    var result1 = document.getElementById('log');
    var start = new Date().getTime();
    console.log('start');
    for (var i = 0; i < 200; i++) {
        result1.value = 'time=' + (new Date().getTime() - start) + ' [i=' + i + ']';
        process();
    }
    result1.value = 'time=' + (new Date().getTime() - start) + ' [done]';
    console.log('end');
}
window.onload = test;
</script>

```

```
</head>
<body>
  <input id='log' />
</body>
</html>
```

اگر کد فوق را در مرورگرهایی غیر از اپرا اجرا کنید میبینید که تنها نتیجه نهایی نمایش داده میشود و فرایندهای میانی درون حلقه نمیتوانند تغییری در محتوای UI ایجاد کنند. طبق انتظار کروم از بقیه بسیار سریعتر بوده و سپس IE و پس از آن فایرفاکس قرار دارد. اما در مورد Opera وضع کاملاً فرق میکند و به دلیل به روز رسانی همزمان UI عملیات بسیار کندتر از بقیه مرورگرها به اتمام میرسد.

نکته : حالات استثنایی دیگری هم در اجرای کدهای مشابه در مرورگرهای مختلف پیش می‌آید که به دلیل پیچیده کردن بیش از حد بحث آورده نشده اند. فقط ذکر این نکته الزامی است که در حال حاضر میزان تفاوت رفتار مرورگرهای مختلف در برخورد با کدهای یکسان قابل ملاحظه است. بنابراین در هنگام توسعه سعی کنید حداقل در این چهار مرورگر معروف آزمایشات خود را به سرانجام برسانید. و در ادامه چند مثال ...

غیرفعال کردن ورودی کاربر

برای اینکار فقط کافی است در رویدادهای keydown یا keypress مقدار false برگشت داده شود. البته در مرورگر Opera برخی از کلیدها از این رفتار پیروی نمیکنند. مثل کاراکترهای ' و + و = که در صورت برگشت false در رویداد keydown باز هم رفتار پیش فرض را از خود نشان میدهد. اما برگرداندن مقدار false در رویداد keypress این مشکل را حل میکند (این موارد در نسخه 11.51 تست شدند). میتوانید با استفاده از کد زیر در تمام مرورگرها این موارد را آزمایش کنید:

```
<input onkeydown="return false" />
<input onkeypress="return false" />
```

تبدیل به حروف بزرگ

```
document.getElementById('myInputText').onkeypress = function (e) {
  var char = getCharacter(e || window.event);
  if (!char) return; // special key
  this.value += char.toUpperCase();
  return false; // برای اینکه کاراکتر اضافی نمایش داده نشود
}
```

در کد فوق متد getCharacter در بالا در قسمت پراپرتی‌های آرگومان رویداد نشان داده شده است. کد فوق چندان کامل نیست و همیشه کاراکتر فشرده شده را بدون توجه به موقعیت کرسر کیبرد در انتهای متن قرار میدهد. **نکته:** استفاده از عبارتی چون e || window.event به این دلیل است که در مرورگر IE آرگومان رویداد (همان e که به آن implicit event object نیز میگویند) به متد مربوطه ارسال نمیشود (تا نسخه 9 که تست کردم مسئله به همین صورت است) در عوض پراپرتی‌های این آرگومان از طریق window.event (یا همان event که به آن explicit event object نیز میگویند) در دسترس هستند. این فیلد در واقع آرگومان آخرین رویداد رخ داده در پنجره جاری را در خود ذخیره میکند. اما در سایر مرورگرها به صورت استاندارد مقدار این آرگومان به عنوان پارامتر رویداد به متد مربوطه ارسال میشود. جالب است که بدانید مرورگرهای Opera و Chrome از هر دو روش پشتیبانی میکنند. مرورگر فایرفاکس تنها از ارسال آرگومان رویداد به متد مربوطه پشتیبانی میکند. عبارت e || window.event در واقع شکل دیگر عبارت زیر است:

```
e ? e : window.event;
```

در جاوا اسکریپت اگر عملگر مقایسه ای در عبارت مقایسه آورده نشود مقدار عبارت با false مقایسه میشود. این مقایسه از نوع abstract است (در ادامه این مطلب توضیح داده شده است). در جاوا اسکریپت 0 و رشته خالی و null و undefined و امثال اینها در مقایسه abstract برابر false در نظر گرفته میشوند. بنابراین در عبارت مقایسه بالا اگر مقدار e مثلاً undefined (در IE)

باشد مقدار window.event بازگشت داده میشود و درغیراینصورت خود e برگشت داده میشود.

تنها عدد

```
document.getElementById('numberInputText').onkeypress = function (e) {
    e = e || window.event;
    var chr = getCharacter(e);
    if (!isNumeric(chr) && chr !== null) return false;
}
function isNumeric(n) {
    return !isNaN(parseFloat(n)) && isFinite(n);
}
function isNumber(val) {
    return val !== "NaN" && (+val) + '' === val + ''
}
```

در کد بالا متد isNumeric از [اینجا](#) گرفته شده است. متد isNumeric جی کوئری (که از نسخه 1.7 اضافه شده است) هم دقیقا از این روش استفاده میکند.

متد دوم یعنی isNumber (که در کد فوق از آن استفاده نشده است) روش دیگری را برای اطمینان از مقدار عددی بودن استفاده میکند. در این روش در صورتیکه val یک مقدار عددی نباشد +val برابر NaN میشود که در نهایت عبارت مقایسه ای مقدار false را برمیگرداند. البته به غیر از خود مقدار NaN که در شرط اول مورد بررسی قرار گرفته است. برای کسب اطلاعات بیشتر در مورد رفتار نسبتا عجیب جاوا اسکریپت با NaN و متد isNaN به [اینجا](#) سر بزنید.

نکته : تفاوت == با === (یا != و !==) - در جاوا اسکریپت دو روش برای مقایسه مقادیر متغیرها وجود دارد. اولی (== یا !=) مقایسه را با تبدیل نوع داده‌ها انجام میدهد (که اصطلاحا به آن type-converting equality comparison یا abstract comparison میگویند) و دومی (=== یا !==) مقایسه را با مقادیر واقعی و بدون تبدیل نوع داده انجام میدهد (که به آن equality without coercion یا strict equality comparison گفته میشود). در واقع در مقایسه strict تنها وقتی که دو متغیر از یک نوع باشند ممکن است مقدار true برگشت داده شود. برای روشنتر شدن مطلب به مثالهای زیر توجه کنید (\wedge) : `false == 0` // `true` `0 === false` // `false`, because they are of a different type `1 == "1"` // `true`, auto type coercion `1 === "1"` // `false`, because they are of a different type توضیحات بهتری در [اینجا](#) آورده شده است.

برای کسب اطلاعات کاملتر میتوانید به [ECMAScript Language Specification](#) و قسمت مقایسه [strict](#) و [abstract](#) مراجعه کنید. (رابطه‌ها و تفاوت‌های میان ECMAScript و JavaScript و JScript و ActionScript در [اینجا](#) آورده شده است).

کار با Char Code و Scan Code

همانطور که قبلا هم اشاره شد میان کد کاراکتر و کلید فشرد شده بر روی کیبرد تفاوت وجود دارد. برای بهره برداری از کلیدهای میانبر در صفحات وب به کد کلید فشرد شده (همان scan code) نیاز است و نه به کد کاراکتر آن. همچنین کلیدهای ویژه و غیرکاراکتری دارای کد کاراکتر نیستند. بیشتر مرورگرها رویداد Keypress را برای کلیدهای غیرکاراکتری فرا نمیخوانند. بنابراین برای این موارد رویدادهای keydown و keyup مفید هستند. امروز تمام مرورگرها از جدول کدهای یکسانی برای scan code استفاده میکنند که نمونه‌های آن را میتوانید در \wedge و \wedge و \wedge مشاهده کنید. نکته ای که باید درباره پراپرتی keyCode یادآوری شود این است که جدای از وضعیت کیبرد (مثل زبان یا موقعیت کلید capsLock) در تمام حالات در ازای فشردن شده یک کلید خاص (یا ترکیبی از کلیدهای کنترلی با سایر کلیدها) همواره باید یک کد یکسان برگشت داده شود.

پس یک charCode کد یونیکد کاراکتر کلید فشرد شده است که تنها در رویداد keypress در دسترس است. یک keyCode خود کلید فشرد شده یا همان scan code است که در رویدادهای keydown و keyup در دسترس است.

نکته: برای تمام کلیدهای الفبایی-عددی scan code با char code یکی است. مثلا برای حروف الفبایی scan code یک کلید با کد ASCII حرف انگلیسی بزرگ آن کلید برابر است.

بنابراین برای بررسی کلید ترکیبی ctrl + a میتوان از کد زیر در رویداد keydown استفاده کرد:

```
e.ctrlKey && e.keyCode == 'A'.charCodeAt(0)
```

و فرقی نمیکند که کاراکتر حاصله 'a' یا 'A' یا 'ش' باشد.

نکته: برای تمام کلیدهای کیبرد به غیر از '؛' و '=' و '-' تمام مرورگرها از کدهای یکسانی استفاده میکنند. میتوانید این کلیدها را با

استفاده از قطعه کد اول این مطلب در مرورگرهای مختلف آزمایش کنید.

نکته: با برگشت مقدار `false` در رویداد `keydown` یا `keypress` رفتار پیشفرض کلید یا ترکیب کلیدهای مربوطه غیرفعال میشود. البته به غیر از عملیاتیهای سطح سیستم عامل (مثل `alt+F4`). چند مثال دیگر در ادامه ...

جابجایی تصویر

کدهای زیر را در یک فایل `html` ذخیره کرده و توسط یک مرورگر فایل حاصله را باز کنید.

```
<html>
<body>
  <div id="dotnettips" style="width: 35px; height: 35px; background-image:
url(http://dotnettips.info/favicon.ico);
  position: absolute; left: 10px; top: 10px;" tabindex="0">
</div>
<script>
  document.getElementById('dotnettips').onkeydown = function (e) {
    e = e || event;
    switch (e.keyCode) {
      case 37: // left
        this.style.left = parseInt(this.style.left) - this.offsetWidth + 'px';
        return false;
      case 38: // up
        this.style.top = parseInt(this.style.top) - this.offsetHeight + 'px';
        return false;
      case 39: // right
        this.style.left = parseInt(this.style.left) + this.offsetWidth + 'px';
        return false;
      case 40: // down
        this.style.top = parseInt(this.style.top) + this.offsetHeight + 'px';
        return false;
    }
  }
</script>
</body>
</html>
```

بر روی تصویر کلید کرده (یا با استفاده از `Tab` فوکس را روی تصویر قرار دهید) و با استفاده از کلیدهای مکان نما (Arrow keys) سعی کنید تصویر را در صفحه جابجا کنید.

در کد فوق برای جلوگیری از رفتار پیش فرض کلیدهای مکان نما (جابجایی محتوای صفحه در صورت وجود اسکروول) مقدار `false` برگشت داده شده است.

نکته : استفاده از خاصیت `tabindex` برای امکان فوکس بر روی `div` اجباری است.

نکته : استفاده از `event` به جای `window.event` تا زمانی که یک متغیر در اسکوپ جاری با نام `event` وجود نداشته باشد مشکلی ایجاد نمیکند.

Caps Lock

متأسفانه راه مستقیمی برای دریافت وضعیت کلید `caps lock` در جاوا اسکریپت وجود ندارد. ظاهراً تنها راه حل موجود برای این مسئله بررسی کد کاراکتر کلید فشرده شده در رویداد `keypress` و بررسی آن با توجه به وضعیت پراپرتی `shift` این رویداد است. بدین ترتیب که اگر کد کاراکتر مربوط به حروف بزرگ بوده درحالیکه کلید شیفت نگه داشته نشده است بنابراین کلید `Caps Lock` روشن است و بالعکس. البته با ترکیب این روش و نیز رصد `scan code` کلید `Caps Lock` (کد آن برابر 20 است) در رویداد `keydown` میتوان وضعیت بهتری پدید آورد. قطعه کد زیر برای اینکار است. در این کد از یک متغیر گلوبال برای نگهداری وضعیت دکمه `caps lock` استفاده میشود.

```
<html>
<body>
  <input id="inputText" type="text" />
  <div id="divCapsLock" style="color: Red; display: none;">Caps Lock is ON!</div>
  <script>
    var capsLock = null;
    document.onkeypress = keyPress;
    document.onkeydown = keyDown;
    function keyDown(e) {
      e = e || event;
```

```

    capsLock = (e.keyCode == 20 && capsLock !== null) ? !capsLock : capsLock;
    document.getElementById('divCapsLock').style.display = capsLock ? 'block' : 'none';
}
function keyPress(e) {
    if (capsLock !== null) return;
    e = e || window.event;
    var charCode = e.charCode || e.keyCode;
    capsLock = (charCode >= 97 && charCode <= 122 && e.shiftKey) || (charCode >= 65 && charCode <= 90
&& !e.shiftKey);
    document.getElementById('divCapsLock').style.display = capsLock ? 'block' : 'none';
}
</script>
</body>
</html>

```

در متد رویداد keypress تکست باکس ابتدا بررسی میشود که آیا قبلا متغیر گلوبال capsLock مقداری غیرنال دارد. اگر مقدار غیرنال داشته باشد دیگر نیازی نیست تا کد کاراکتر کلید بررسی شود زیرا وضعیت جاری کلید معلوم است. بنابراین در ادامه کار صرفه جویی میشود. دلیل استفاده از رویدادهای سطح document به جای خود تکست باکس این است تا از کوچکترین فرصتها! برای تعیین وضعیت جاری کلید caps lock استفاده شود. یعنی بتوان پس از فشردن اولین کلید کاراکتری بدون توجه به موقعیت فوکس در صفحه، این وضعیت را از حالت نامشخص خارج کرد.

نکته : در سلسله مراتب رویدادهای اجزای یک document همیشه ابتدا رویدادهای مربوط به عناصر فرزند فراخوانده میشود و رویدادهای مربوط به عناصر document و window در پایان صدا زده میشوند. برای آزمودن این مورد قطعه کد زیر را در مرورگرهای مختلف امتحان کنید:

```

<html>
<body>
  <div id='divInput'>
    <input id="inputText" type="text" />
  </div>
  <script>
    document.getElementById('inputText').onkeydown = inputKeyDown;
    document.getElementById('divInput').onkeydown = divKeyDown;
    document.onkeydown = documentKeyDown;
    window.onkeydown = windowKeyDown;
    function divKeyDown(e) {
      console.log('divKeyDown');
    }
    function inputKeyDown(e) {
      console.log('inputKeyDown');
    }
    function documentKeyDown(e) {
      console.log('documentKeyDown');
    }
    function windowKeyDown(e) {
      console.log('windowKeyDown');
    }
  </script>
</body>
</html>

```

نکته : اگر از تگ doctype استفاده نشود (همانطور که در ابتدای این مطلب اشاره شد)، در IE رویدادهای عنصر window فراخوانی نمیشوند.

در هر صورت برای مشخص کردن وضعیت کلید caps lock نیاز است تا کاربر ابتدا کلیدی را بفشارد و در حال حاضر روش و راه حل دیگری وجود ندارد! درضمن اگر صفحه کلیدی غیر از انگلیسی استفاده شود روش فوق جواب نخواهد داد و باید بررسی‌های بیشتری انجام شود. البته در مورد زیانهای چون فارسی که روشن یا خاموش بودن caps lock تاثیری در کاراکتر حاصله ندارد کاری نمیتوان کرد و نمیتوان از وضعیت caps lock باخبر شد! هرچند در این موارد معمولا وضعیت این دکمه مهم نیست زیرا بیشترین کاربرد این گونه هشدارها در ورودی‌های رمز عبور است در صورتیکه زبان صفحه کلید انگلیسی (یا مشابه آن) باشد. برای اینکار کد زیر به عنوان راه حل بهتر پیشنهاد میشود:

```

<html>
<body>
  <input id="inputText" type="text" />
  <div id="divCapsLock" style="color: Red; display: none;">Caps Lock is ON!</div>
  <script>
    var capsLock = null;

```

```

var hasFocus = false;
document.onkeyup = keyUp;
document.onkeypress = keyPress;
document.getElementById('inputText').onfocus = focus;
document.getElementById('inputText').onblur = focus;
function warnCapsLock() {
    document.getElementById('divCapsLock').style.display = (capsLock != null && capsLock && hasFocus)
    ? 'block' : 'none';
}
function focus() {
    hasFocus = !hasFocus;
    warnCapsLock();
}
function keyUp(e) {
    e = e || event;
    capsLock = (e.keyCode == 20 && capsLock !== null) ? !capsLock : capsLock;
    warnCapsLock();
}
function keyPress(e) {
    if (capsLock != null) return;
    e = e || window.event;
    var charCode = e.charCode || e.keyCode;
    capsLock = (charCode >= 97 && charCode <= 122 && e.shiftKey) || (charCode >= 65 && charCode <= 90
    && !e.shiftKey);
    warnCapsLock();
}
</script>
</body>
</html>

```

اگر دقت کنید میبینید که رویداد keydown در این کد نهایی با keyup جایگزین شده است. در صورت استفاده از رویداد keypress یک مشکل کوچک بوجود می‌آید و آن این است که اگر کاربر کلید caps lock را فشرده و سپس آن را در حالت فشرده نگه دارد اتفاق بدی خواهد افتاد! در این حال چون رویداد keydown مرتباً فراخوانده میشود وضعیت متغیر capsLock در این کد کاملاً نامعتبر خواهد بود. بنابراین بهتر است تا از رویداد keyup که تنها یکبار فراخوانده میشود استفاده شود.

نکته : اگر کاربر پس از مشخص شدن وضعیت کلید Caps Lock در این کد، فوکس را به پنجره دیگری تغییر داده و سپس وضعیت این دکمه در آنجا تغییر دهد این کد دیگر درست کار نخواهد کرد. برای حل این مشکل هم راه حل زیر وجود دارد:

```

window.onblur = function () { capsLock = null; }

```

با این کار وضعیت متغیر capsLock ریست میشود.

نکته : در آزمایش این کد دقت کنید که زبان کیبرد حتماً انگلیسی باشد.

مدیریت کلیدها

برای مدیریت کلیدهای کیبرد راههای متنوعی وجود دارد. مثلاً میتوان از قطعه کد زیر استفاده کرد:

```

var Client = {};
Client.Keyboard = {};
Client.Keyboard.EnableKeyDown = true;
Client.Keyboard.EnableKeyUp = false;
Client.Keyboard.CurrentKeyEvent = null;

window.onkeydown = function (event) {
    if (!Client.Keyboard.EnableKeyDown) return true;
    return KeyboardEvents(event);
};

window.onkeyup = function (event) {
    if (!Client.Keyboard.EnableKeyUp) return true;
    return KeyboardEvents(event);
};

function Rise(event) {
    var e = Client.Keyboard.CurrentKeyEvent;
    if (event) {
        var data = { shift: e.shiftKey, ctrl: e.ctrlKey, alt: e.altKey };
        if (!event(data)) return false;
        return event(data);
    }
}

```

```
    return true;
}

function KeyboardEvents(e) {
    e = e || window.event;
    Client.Keyboard.CurrentKeyEvent = e;
    switch (e.keyCode) {
        case 8:
            return Rise(Client.Keyboard.Backspace);
        case 9:
            return Rise(Client.Keyboard.Tab);
        case 13:
            return Rise(Client.Keyboard.Enter);
        case 16:
            return Rise(Client.Keyboard.Shift);
        case 17:
            return Rise(Client.Keyboard.Ctrl);
        case 18:
            return Rise(Client.Keyboard.Alt);
        case 19:
            return Rise(Client.Keyboard.Pause);
        case 20:
            return Rise(Client.Keyboard.CapsLock);
        case 27:
            return Rise(Client.Keyboard.Esc);
        case 33:
            return Rise(Client.Keyboard.PageUp);
        case 34:
            return Rise(Client.Keyboard.PageDown);
        case 35:
            return Rise(Client.Keyboard.End);
        case 36:
            return Rise(Client.Keyboard.Home);
        case 37:
            return Rise(Client.Keyboard.Left);
        case 38:
            return Rise(Client.Keyboard.Up);
        case 39:
            return Rise(Client.Keyboard.Right);
        case 40:
            return Rise(Client.Keyboard.Down);
        case 44:
            return Rise(Client.Keyboard.PrtScr);
        case 45:
            return Rise(Client.Keyboard.Insert);
        case 46:
            return Rise(Client.Keyboard.Delete);

        //////////////////////////////////////
        case 48:
            return Rise(Client.Keyboard.Num0);
        case 49:
            return Rise(Client.Keyboard.Num1);
        case 50:
            return Rise(Client.Keyboard.Num2);
        case 51:
            return Rise(Client.Keyboard.Num3);
        case 52:
            return Rise(Client.Keyboard.Num4);
        case 53:
            return Rise(Client.Keyboard.Num5);
        case 54:
            return Rise(Client.Keyboard.Num6);
        case 55:
            return Rise(Client.Keyboard.Num7);
        case 56:
            return Rise(Client.Keyboard.Num8);
        case 57:
            return Rise(Client.Keyboard.Num9);

        //////////////////////////////////////
        case 65:
            return Rise(Client.Keyboard.A);
        case 66:
            return Rise(Client.Keyboard.B);
        case 67:
            return Rise(Client.Keyboard.C);
        case 68:
            return Rise(Client.Keyboard.D);
        case 69:
            return Rise(Client.Keyboard.E);
```

```
case 70:
    return Rise(Client.Keyboard.F);
case 71:
    return Rise(Client.Keyboard.G);
case 72:
    return Rise(Client.Keyboard.H);
case 73:
    return Rise(Client.Keyboard.I);
case 74:
    return Rise(Client.Keyboard.J);
case 75:
    return Rise(Client.Keyboard.K);
case 76:
    return Rise(Client.Keyboard.L);
case 77:
    return Rise(Client.Keyboard.M);
case 78:
    return Rise(Client.Keyboard.N);
case 79:
    return Rise(Client.Keyboard.O);
case 80:
    return Rise(Client.Keyboard.P);
case 81:
    return Rise(Client.Keyboard.Q);
case 82:
    return Rise(Client.Keyboard.R);
case 83:
    return Rise(Client.Keyboard.S);
case 84:
    return Rise(Client.Keyboard.T);
case 85:
    return Rise(Client.Keyboard.U);
case 86:
    return Rise(Client.Keyboard.V);
case 87:
    return Rise(Client.Keyboard.W);
case 88:
    return Rise(Client.Keyboard.X);
case 89:
    return Rise(Client.Keyboard.Y);
case 90:
    return Rise(Client.Keyboard.Z);
    //////////////////////////////////////
case 91:
    //case 219: // opera
    return Rise(Client.Keyboard.LeftWindow);
case 92:
    return Rise(Client.Keyboard.RightWindow);
case 93:
    return Rise(Client.Keyboard.ContextMenu);
    //////////////////////////////////////
case 96:
    return Rise(Client.Keyboard.NumPad0);
case 97:
    return Rise(Client.Keyboard.NumPad1);
case 98:
    return Rise(Client.Keyboard.NumPad2);
case 99:
    return Rise(Client.Keyboard.NumPad3);
case 100:
    return Rise(Client.Keyboard.NumPad4);
case 101:
    return Rise(Client.Keyboard.NumPad5);
case 102:
    return Rise(Client.Keyboard.NumPad6);
case 103:
    return Rise(Client.Keyboard.NumPad7);
case 104:
    return Rise(Client.Keyboard.NumPad8);
case 105:
    return Rise(Client.Keyboard.NumPad9);
    //////////////////////////////////////
case 106:
    return Rise(Client.Keyboard.Multiply);
case 107:
    return Rise(Client.Keyboard.Add);
case 109:
    return Rise(Client.Keyboard.Subtract);
case 110:
    return Rise(Client.Keyboard.DecimalPoint);
case 111:
```

```

        return Rise(Client.Keyboard.Divide);
        ///////////////////////////////////////////////////////////////////
case 112:
    return Rise(Client.Keyboard.F1);
case 113:
    return Rise(Client.Keyboard.F2);
case 114:
    return Rise(Client.Keyboard.F3);
case 115:
    return Rise(Client.Keyboard.F4);
case 116:
    return Rise(Client.Keyboard.F5);
case 117:
    return Rise(Client.Keyboard.F6);
case 118:
    return Rise(Client.Keyboard.F7);
case 119:
    return Rise(Client.Keyboard.F8);
case 120:
    return Rise(Client.Keyboard.F9);
case 121:
    return Rise(Client.Keyboard.F10);
case 122:
    return Rise(Client.Keyboard.F11);
case 123:
    return Rise(Client.Keyboard.F12);
    ///////////////////////////////////////////////////////////////////
case 144:
    return Rise(Client.Keyboard.NumLock);
case 145:
    return Rise(Client.Keyboard.ScrollLock);
case 186:
case 59: // opera & firefox
    return Rise(Client.Keyboard.SemiColon);
case 187:
case 61: // opera
    //case 107: //firefox
    return Rise(Client.Keyboard.Equal);
case 188:
    return Rise(Client.Keyboard.Camma);
case 189:
    return Rise(Client.Keyboard.Dash);
case 190:
    return Rise(Client.Keyboard.Period);
case 191:
    return Rise(Client.Keyboard.Slash);
case 192:
    return Rise(Client.Keyboard.GraveAccent);
case 219:
    return Rise(Client.Keyboard.OpenBracket);
case 220:
    return Rise(Client.Keyboard.BackSlash);
case 221:
    return Rise(Client.Keyboard.CloseBracket);
case 222:
    return Rise(Client.Keyboard.SingleQuote);
    }
};

```

نحوه استفاده از این کد به صورت زیر است:

```

// ctrl + s
Client.Keyboard.S = function (e) {
    if (e.ctrl) {
        // انجام عملیات موردنظر
    }
    return true;
}

```

اگر در متد الصاق شده به پراپرتیهای Client.Keyboard هیچ مقداری برگشت داده نشود، باتوجه به کد موجود در متد Rise، عملیات پیشفرض کلید مربوطه در پنجره مرورگر غیرفعال خواهد شد. بنابراین در کد بالا بعد از شرط، مقدار true برگشت داده شده است.

هرچند سعی کردم که تفاوت میان مرورگرهای مختلف را درنظر بگیرم ولی احتمال اینکه برخی از کدها به دلیل تفاوت میان

مرورگرهای مختلف در کد بالا آورده نشده باشد وجود دارد!

نکته: کد فوق بیشتر برای روشن تر شدن موضوع ارائه شده است چون راه حل های بهتری هم برای مدیریت کلیدهای کیبرد وجود دارد. مثل استفاده از کتابخانه های [Mousetrap](#) یا [HotKey](#) یا [jQuery Hotkeys](#) یا [KeyboardJS](#). البته در این میان بهترین کتابخانه موجود به نظر من همان Mousetrap است که تنها کتابخانه موجود است که علاوه بر پشتیبانی از ترکیب کلیدها با کلیدهای کنترلی، از توالی کلیدها (keys sequence) نیز پشتیبانی میکند که کار جالبی است. در پایان این نکته را یادآور میشوم که امروزه توسعه اپلیکیشن های تحت وب بدون استفاده مناسب از امکانات سمت کلاینت طرفداری ندارد. پس بهتر است هرچه بیشتر در مورد این زبان مرموز و پر از رمز و راز (JavaScript) بدانیم.

نظرات خوانندگان

نویسنده: فرهاد یزدان پناه
تاریخ: ۲۱:۲۱۳۹۱/۱۱/۱۴

دستتون درد نکنه
مقاله خیلی کامل و جامعی بود.
بسیار سود بردم.

نویسنده: moh
تاریخ: ۰:۷۱۳۹۲/۰۳/۲۷

با سلام
امکانش هست در مورد این تابع که دقیقا چه کاری انجام میدهد توضیح دهید <.....preventDefault
تفاوتش با return false چی هست

نویسنده: یوسف نژاد
تاریخ: ۱۱:۵۰۱۳۹۲/۰۳/۲۷

سوال خوبی است. قبل از پاسخ به سوال شما باید کمی درباره رویدادها در عناصر DOM توضیح داده بشه.
در صفحات html به صورت پیش فرض، هر رویدادی که در عناصر زیردست یا فرزند رخ میدهد به ترتیب در تمام عناصر والدش
انتشار می‌یابد. به این فرایند event propagation یا event bubble up میگویند.
در حالت عادی برای تغییر رفتار پیش فرض مدیریت رویدادهای عناصر DOM در مرورگرها، با استفاده از زبان جاوا اسکریپت چند
روش وجود دارد.
مثلا آرگومان رویداد (همان متغیر معروف e) در مرورگرهای مدرن دو متد برای اینکار دارد:

```
e.preventDefault()  
e.stopPropagation()
```

متد preventDefault همانطور که از نامش هم پیداست از اجرای رفتار پیش فرض رویداد جاری جلوگیری میکند. مثلا با قرار دادن
این متد در رویداد مربوط به کلیک یک لینک، رفتار پیش فرض کلیک روی لینک (همان انتقال به آدرس تنظیم شده در پراپرتی src
لینک) اجرا نمیشود.

متد stopPropagation هم همانطور که از نامش پیداست از انتشار رویداد جاری به عناصر والدش جلوگیری میکند.
راه حل دیگری که برای تغییر این رفتارهای پیش فرض وجود دارد استفاده از return false در انتهای رویداد است. این کار
تأثیری همانند استفاده از متد preventDefault دارد. تفاوت این دو روش آن است که برگشت مقدار false تنها میتواند در انتهای
یک متد استفاده شود بنابراین اگر خطایی در متد مربوطه رخ دهد احتمال اجرا نشدن این خط و در نتیجه اجرای ادامه فرایند با
اجرای رفتار پیش فرض رویداد مربوطه وجود دارد. اما متد preventDefault را میتوان در ابتدای متد رویداد استفاده کرد و خیال
خود را راحت کرد!

نکته: دو متد اشاره شده در نسخه‌های قدیمی مرورگرها (مثلا IE 8 و قبلتر) وجود ندارد!
برای آزمایش تمامی این موارد میتوان از کد زیر که برای اینکار آماده کردم استفاده کرد:

```
<html>  
<body>  
<div onclick="document.body.appendChild(outer)" style='border:solid 1px'>  
  Outer Div  
  <a id="inner" href="index2.htm">index2</a>  
</div>  
<script>  
var outer = document.createElement('div');  
outer.innerText = 'outer';  
var inner = document.createElement('div');  
inner.innerText = 'inner';  
  
document.getElementById('inner').onclick = function(e) {
```

```
e = e || window.event;
document.body.appendChild(inner);
//e.preventDefault();
//e.stopPropagation();
//e.returnValue = false;
return false;
};
</script>
</body>
</html>
```

البته در صورت استفاده از jQuery به دلیل اینکه این کتابخانه از نمونه ای سفارشی برای آرگومان رویدادها استفاده میکند فرایندهای موجود کمی تفاوت دارد.

اطلاعات بیشتر: [^](#) و [^](#)