ななちゃんのIT教室

クリじいイベント調査隊の巻

by nara.yasuhiro@gmail.com

ななちゃんが
イベントの秘密解明に挑戦するという お話

第 0.2 版　2017 年 5 月 28 日

フリー素材
http://freeillustration.net

いらすとやフリー素材
http://www.irasutoya.com/

もくじ

第1回　イベント？

なな：こんどは、何を調べるの？

クリ：JavaScript でとても重要な、イベントじゃ。

なな：どういうこと？

クリ：従来の、C 言語のようなプログラミング言語では、普通は、長いメインプログラムの中で、マウスや、キーボードの状態をチェックして、それに対応する処理を記述していた。 たとえば、キーボードのキーが押されるのを待つ部分でプログラムが待機するような形になっていた。その時、マウスが操作されていても、キーボードのキーが押されるまで対応できなかった。

なな：ハンバーガーショップで、店員さんが「ここで召し上がりますか、テイクアウトなさいますか？」と聞いているときに、お客が「ミネストローネスープはありますか？」と聞いても答えてくれないということね。

クリ：そうじゃ。それに対して、JavaScript では、「マウスがクリックされたら何をするか」、「キーボードのキーが押されたら何をするか」を、それぞれ小分けにして記述するんじゃ。「マウスがクリックされた」とか「キーボードのキーが押された」というのは、システムというか、ブラウザというか、JavaScript の裏方さんが常に監視しているんじゃ。プログラマは「何をするか」の部分だけを記述するんじゃ。

なな：なるほど。「ミネストローネスープがあるか」と聞かれたら「ありますと答える」、「テイクアウトします」と言われたら、「レジにそのように入力する」、「以上です」と言われたら、「未入力項目があったらお客に聞く、すべての項目がそろっていたら合計金額をお客に伝えて、お金を受け取って、注文された商品を揃える」ということね。 お客様がどんな順序で話しかけても対応できるのね。

クリ：そうじゃ。JavaScript では、マウスクリックなどの出来事を「イベント」と呼んでいる。それぞれのイベントに、「マウスクリック」、「マウス移動」、「キーボード押下」などと名前を決めてあって、プログラマが、自分が利用するものに、処理関数（メソッド）を書いておくんじゃ。このようなメソッドを「イベントハンドラ」とか、「イベントリスナー」と呼んでいる。たとえば、「クリック → myClickfunction を実行してね」というふうに。 基本的なイベントの名前は、GlobalEventHandlers クラスのプロパティとして定義してあって、対象を明示しないと window が対象になる。 各 DOM 要素に、そのコピーのようなものがくっついているが、対象範囲がその要素だけに限定されるなどの修正がなされている。たとえば、「onclick= myClickfunction」とすれば、ブラウザのどこでクリックしても呼び出されるが、「myButton.onclick = myClickfunction」とすれば、myButton 要素の上でクリックしたときだけ呼び出されるということじゃ。

なな：クリじい、長い説明、お疲れ様。

クリ：なんの、なんの。

なな：マウスやキーボード以外にも、イベントはあるの？

クリ：きわめてたくさんある。セレクトイベント、スクロールイベント、ファイル操作イベント、ドラッグ＆ドロップイベント、タイマーイベント、チェンジイベントなど。人間、つまりユーザの操作がからんでいて、人間が操作を行っている間、プログラムがそれを監視するのはもったいないので、操作が完了したらプログラムを呼び出してもらう、というものが多いな。ディスク装置からファイルのデータを読み出す時に、読み出し完了まで待つとか。アニメーション表示のために、0.1 秒ごとに描画処理を呼び出すというようなものもある。

なな：「監視するのはもったいない」というけど、その間に別の仕事をしていて、イベントが発生したら、やっていた仕事を中断するの？

クリ：JavaScript の場合は、シングルスレッドと言って、中断はないんじゃ。

なな：じゃあ、コンピュータが監視している間、JavaScript は何をするの？

クリ：作業依頼をして、イベントハンドラの設定を終えたら何もしない。眠っておる。
「果報は寝て待て」じゃな。

なな：眠っているのに、「もったいない」？

クリ：「眠っている」というのは、 すぐに対応できるように待機しているという意味じゃ。 プログラムで監視すると、その処理のために、イベントが発生した時に、すぐに応答できなくなるんじゃ。

なな：りょうかい。

クリ：それでは、次回から、それぞれのイベントと、そのハンドラの使い方を調べてゆこう。

なな：わくわく！

第2回　イベントの設定

先生：DOM の要素ノードなどに、イベントリスナを登録することで、マウスクリックなどのイベントに応答する JavaScript プログラムを指定することができます。登録方法が 3 種類あります。

なな：ひとつ目は？

先生：myButton.addEventListener('click', function(event) { alert('Hello world'); }, false); の形です。myButton は、ボタンなどの DOM 要素ノードです。イベントの種類は、ここでは click を指定しています。on click ではないので要注意。false の部分は、通常は false と指定してください。Internet Explorer 8 以前は、この方法の代わりに、.attachEven() を使う必要がありましたが、今は、IE 11 とか、Edge が主流なので気にしないように。

なな：ふたつ目は？

先生：HTML のほうで、<button onclick='alert('Hello world!')'> のように指定する形。 alert('Hello world!') の代わりに、go() のように、自分で作った JavaScript の関数（メソッド）を呼び出すこともできます。<button onclick='go1();go2()'> のように、二つ以上の関数呼び出しをならべることもできます。並べた順に呼び出されます。

なな：みっつ目は？

先生：myButton.onclick = function(event) { alert('Hello world'); }; の形です。このみっつ目の方法の問題は、ひとつの要素のひとつのイベントにつき、ひとつのハンドラしか定義できないことです。myButton.onclick = の文がふたつ以上あると、最後に実行するものだけが有効になります。myButton.addEventListener なら、同じイベントに対する設定がふたつ以上あっても、すべてが実行されることになります。ひとつ目とみっつ目の方法の関数では、event 引数 をとる関数を定義できます。この引数からいろいろな情報を得ることができます。たとえば、

event.target	イベントを最初に送出した要素への参照。
event.currentTarget	イベントを検出した要素への参照。
event.timeStamp	イベントが生成された時刻。
event.type	イベントの名前。(大文字小文字を区別しない)

なな：マウスの場合は？

先生：下記のような、イベント発生時のマウスカーソルの位置座標を得ることができます。

event.pageX/event.pageY	表示されていない部分を含め、ページの左上を基点とする座標
event.screenX/event.screenY	ディスプレイの左上を基点とする座標
event.clientX/event.clientY	ブラウザの左上を基点とする座標
event.offsetX/event.offsetY	イベントが発生した要素の左上を基点とする座標

なな：マウスのイベントは、どんな要素に設定できるの？

先生：個別の要素なら、その要素の部分でのイベントだけ拾えます。document.body や、document に設定した場合は、ブラウザ内のどこで発生したイベントでも拾えます。どこで発生したイベントなのかは、イベントハンドラで event.target によって確認できます。event は、イベントハンドラの 引数です。

なな：キーボードのイベントは？

先生：onkeydown などのイベントを個別の要素、document.body、document に設定できます。event.keyCode などから、どのキーが押されたかの情報が得られます。

なな：次回から、いよいよ、個別のイベントの使い方ね。

第3回　マウスイベント、タッチイベント、ポインタイベント

クリ：まず、マウス関連のものから見てゆこう。下記のようなものがあるんじゃ。

- onmouseenter　マウスポインタが要素内に入った時
- onmouseleave　マウスポインタが要素外に出た時
- onmouseout　マウスポインタが要素内から離れた時
- onmouseover　マウスポインタが要素内に入った時
- onmousemove　マウスポインタが移動した時
- onmousedown　マウスのボタンが押された時
- onmouseup　マウスのボタンが離された時
- onmousewheel　マウスのホイールが回された時

- onclick　マウスの左ボタンがクリックされた時
- oncontextmenu　マウスの右ボタンがクリックされた時。
- ondblclick　マウスの左ボタンがダブルクリックされた時

マウス/キー/タッチ共通の上位イベント

なな：スマホやタブレットのタッチ操作は？

クリ：タッチ操作に対するイベントがあるんじゃ。

タップ＝左クリック
ダブルタップ＝ダブルクリック
長タップ＝右クリック

- ontouchstart　指がタッチスクリーンに接触した時
- ontouchmove　指がタッチスクリーン上でドラッグした時
- ontouchend　指がタッチスクリーンを離れた時
- ontouchcancel　タッチスクリーンの接触が妨害された時。3本目の指がタッチしたなど。

しかし、両方を意識して、両方書き並べるのは面倒だし、今後、そういうことが増えてくるので、マウス/タッチパネル/ペンなどのデバイスをまとめて「ポインタ」という抽象的デバイスとして扱うようになってきている。

- onpointerdown　マウスのボタンが押されたのに相当するような操作の時
- onpointermove　マウスポインタが移動したのに相当するような操作の時
- onpointerup　マウスのボタンが離されたのに相当するような操作の時
- onpointercancel　タッチスクリーンで3本目の指がタッチしたのに相当するような操作の時
- onpointerover　マウスポインタが要素内に入ったのに相当するような操作の時
- onpointerout　マウスポインタが要素内から離れたのに相当するような操作の時
- onpointerenter　マウスポインタが要素内に入ったのに相当するような操作の時
- onpointerleave　マウスポインタが要素外に出たのに相当するような操作の時

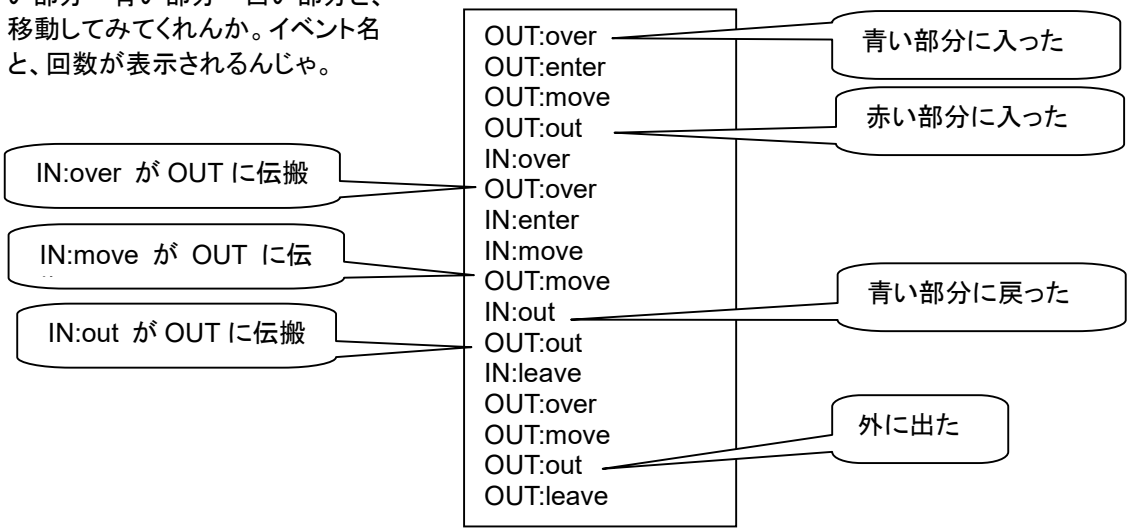
これらを使えば、いろいろなデバイスにまとめて対応できる。これからは、これがお勧めじゃな。

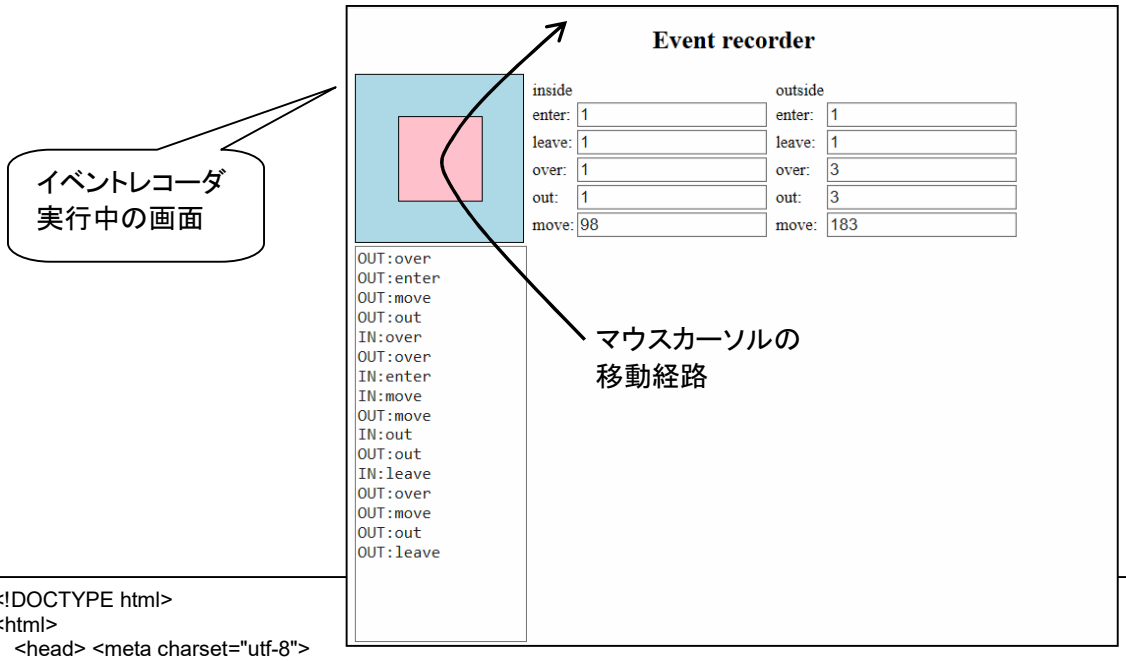
なな：pointerleave（mouseleave） と pointerout（mouseout） の区別、つまり、leave と out の違いは？

クリ：たとえば、大きな div 要素の内部に、小さな div 要素があった場合に差が出てくる。外部から、大きい div に入った時が enter、小さい div に移動した時、大きい div からは out するが、leave はしない。小さい div から戻ってきた時に over になって、外部に出たら leave ということじゃ。

なな：分かりにくいわね。クリじい、秘密の道具はないの？

クリ：わしはドラえもんか？　まあ、いいじゃろう。「イベントレコーダー」！　マウスカーソルを、白地→青い部分→赤い部分→青い部分→白い部分と、移動してみてくれんか。イベント名と、回数が表示されるんじゃ。






```
<!DOCTYPE html>
<html>
  <head> <meta charset="utf-8">
    <title>Event recorder</title>
    <style> *       { font-size: 20px; }
            h1      { font-size: 30px; text-align:center; }
            #outd { width:200px; height:200px; border:solid 1px; background-color:lightblue; }
            #ind   { width:100px; height:100px; border:solid 1px; background-color:pink; margin :50px 50px; }
    </style>
  </head><body>
    <h1>Event recorder</h1>
    <table><tr>
        <td><div id=outd><div id=ind></div></div></td>
        <td><table> <tr><td>inside</td></tr>
            <tr><td>enter:</td><td><input type=text value=0 id=enter1></td></tr>
            <tr><td>leave:</td><td><input type=text value=0 id=leave1></td></tr>
            <tr><td>over: </td><td><input type=text value=0 id=over1></td></tr>
            <tr><td>out:   </td><td><input type=text value=0 id=out1></td></tr>
            <tr><td>move: </td><td><input type=text value=0 id=move1></td></tr>
        </table></td> <td><table>
            <tr><td>outside</td></tr>
            <tr><td>enter:</td><td><input type=text value=0 id=enter2></td></tr>
            <tr><td>leave:</td><td><input type=text value=0 id=leave2></td></tr>
            <tr><td>over: </td><td><input type=text value=0 id=over2></td></tr>
            <tr><td>out:   </td><td><input type=text value=0 id=out2></td></tr>
            <tr><td>move: </td><td><input type=text value=0 id=move2></td></tr>
        </table></td></tr>
        <tr><td><textarea cols=17 rows=20 id=ta></textarea></td></tr>
    </table> <script>
ta.value = "", isd1 = isd2 = 0;;
var enter1c = leave1c = over1c = out1c = move1c =
    enter2c = leave2c = over2c = out2c = move2c = 0;
ind.addEventListener ("pointerenter",function() { enter1.value = ++enter1c;
                                        ta.value += "IN:enter\n"; ta.scrollTop = ta.scrollHeight; isd1 = 1; });
ind.addEventListener ("pointerleave",function() { leave1.value = ++leave1c;
                                        ta.value += "IN:leave\n";ta.scrollTop = ta.scrollHeight; isd1 = 0; });
ind.addEventListener ("pointerover", function() { over1.value    = ++over1c;
                                        ta.value += "IN:over\n"; ta.scrollTop = ta.scrollHeight; isd1 = 1; });
ind.addEventListener ("pointerout",    function() { out1.value     = ++out1c;
                                        ta.value += "IN:out\n"; ta.scrollTop = ta.scrollHeight; isd1 = 0; });
ind.addEventListener ("pointermove", function() { move1.value    = ++move1c;
                                      if (isd1 == 1) { ta.value += "IN:move\n"; ta.scrollTop = ta.scrollHeight; isd1 = 2;}});
outd.addEventListener("pointerenter", function() { enter2.value = ++enter2c;
                                        ta.value += "OUT:enter\n"; ta.scrollTop = ta.scrollHeight; isd2 = 1; });
outd.addEventListener("pointerleave", function() { leave2.value = ++leave2c;
                                        ta.value += "OUT:leave\n"; ta.scrollTop = ta.scrollHeight; isd2 = 0; });
outd.addEventListener("pointerover", function() { over2.value    = ++over2c;
                                        ta.value += "OUT:over\n"; ta.scrollTop = ta.scrollHeight; isd2 = 1; });
outd.addEventListener("pointerout",    function() { out2.value     = ++out2c;
                                        ta.value += "OUT:out\n"; ta.scrollTop = ta.scrollHeight; isd2 = 0; });
outd.addEventListener("pointermove", function() { move2.value    = ++move2c;
                                  if (isd2 == 1) { ta.value += "OUT:move\n"; ta.scrollTop = ta.scrollHeight; isd2 = 2;}});
    </script>
  </body>
</html>
```

第4回　子供の喧嘩に親が：イベントの伝搬

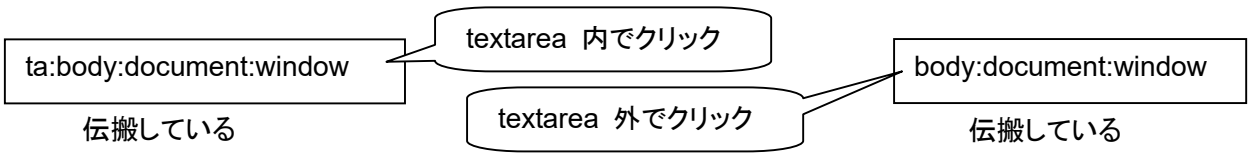
なな：右クリックは、なぜ「contextmenu」なの？

クリ：「印刷」とか、「コピー」などの右クリックメニュー（コンテクストメニュー）を出す操作に使われるからじゃ。プログラマが指定するイベントハンドラ（メソッド）の返り値を false にすると、コンテクストメニューが出なくなる。

なな：ウェブのページで、右クリックすると「コンテクストメニューは禁止です」なんて出る場合があるけど、この仕組みを使っているのね。

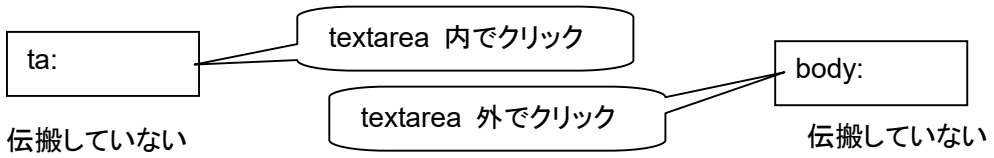
クリ：そうじゃな。ちなみに、window の中に document、その中に body、その中に div、その中に input があるという場面で、input 上でクリックすると、click イベントは、input → div → body → documet → window と、順番に伝わってゆくんじゃ。でも、イベントハンドラの中で stopPropagation() を呼び出せば、上位への伝搬を中止できるし、preventDefault() を呼び出せば、上位へ伝搬するが、デフォルト処理（ブラウザのコンテクストメニューとか、文字列選択など）は中止ことができる。両方呼びさせば、上位への伝搬も、デフォルト処理も中止できる。イベントハンドラで false を return で返せば、preventDefault() と同様の効果になる。

```
<textarea id=ta rows=20 cols=50></textarea>
<script>
ta.value = "";
window.onclick              = function(e) { ta.value += "window\n";    ta.scrollTop = ta.scrollHeight; };
document.onclick            = function(e) { ta.value += "document:"; ta.scrollTop = ta.scrollHeight; };
document.body.onclick = function(e) { ta.value += "body:";          ta.scrollTop = ta.scrollHeight; };
ta.onclick =                  function(e) { ta.value += "ta:";              ta.scrollTop = ta.scrollHeight; }; </script>
```

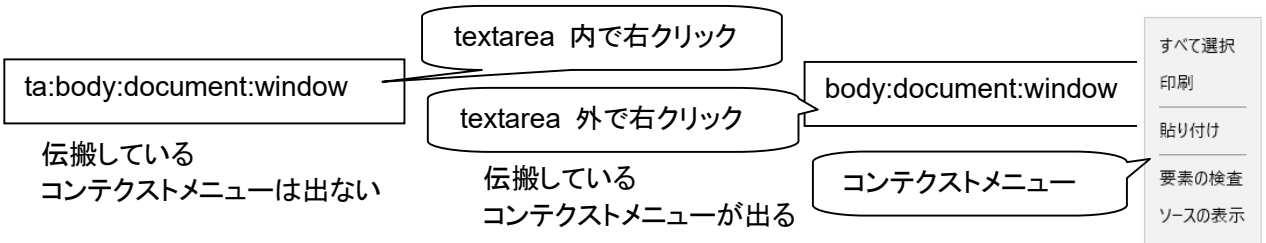


クリックした要素は e.target、イベントを検出した要素は e.currentTarget で得ることができる。

```
<textarea id=ta rows=20 cols=50></textarea>
<script>
ta.value = "";
window.onclick              = function(e) { ta.value += "window\n"; };
document.onclick            = function(e) { ta.value += "document:"; e.stopPropagation();
                                            ta.scrollTop = ta.scrollHeight; };
document.body.onclick = function(e) { ta.value += "body:";          e.stopPropagation();
                                            ta.scrollTop = ta.scrollHeight; };
ta.onclick                  = function(e) { ta.value += "ta:";              e.stopPropagation();
                                            ta.scrollTop = ta.scrollHeight; }; </script>
```

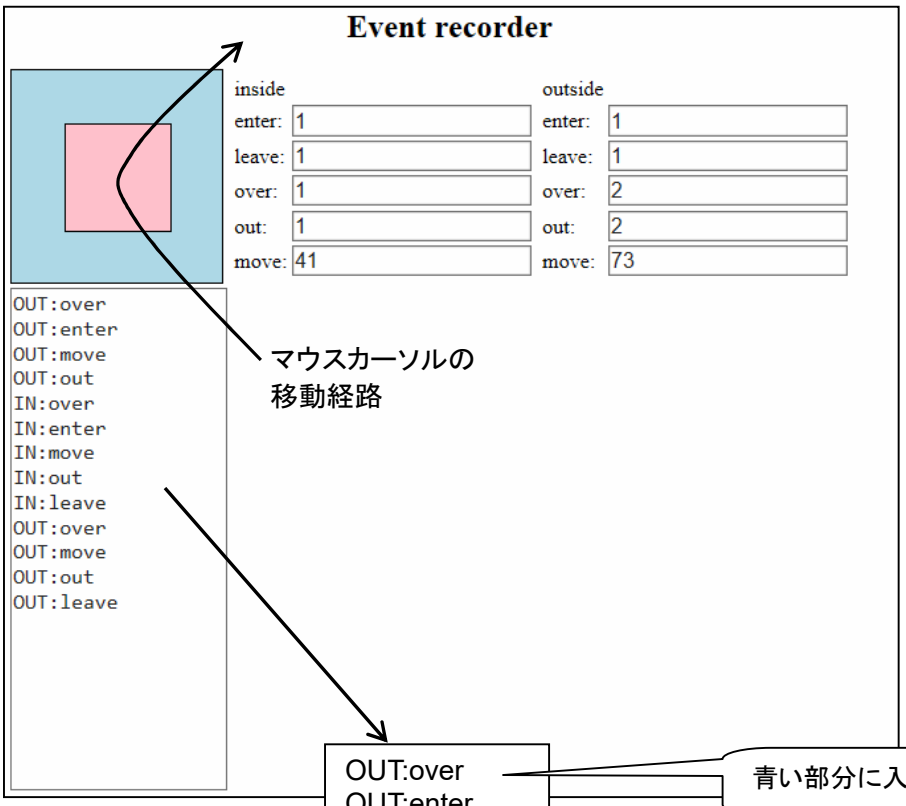



```
<textarea id=ta rows=20 cols=50></textarea>
<script>
ta.value = "";
window.oncontextmenu              = function(e) { ta.value += "window\n";    ta.scrollTop = ta.scrollHeight; };
document.oncontextmenu            = function(e) { ta.value += "document:"; ta.scrollTop = ta.scrollHeight; };
document.body.oncontextmenu = function(e) { ta.value += "body:";          ta.scrollTop = ta.scrollHeight; };
ta.oncontextmenu                  = function(e) { ta.value += "ta:";              ta.scrollTop = ta.scrollHeight;
                                                  e.preventDefault(); }; </script>
```



クリ：さきほどの「イベントレコーダ」に stopPropagation() を追加すると、動作が人間の直感に近づくんじゃ。

```
ind.addEventListener ("pointerenter",
                        function() { enter1.value = ++enter1c;
                                     ta.value += "IN:enter\n";
                                     ta.scrollTop = ta.scrollHeight; isd1 = 1; });
ind.addEventListener ("pointerleave",
                        function() { leave1.value = ++leave1c;
                                     ta.value += "IN:leave\n";
                                     ta.scrollTop = ta.scrollHeight; isd1 = 0; });
ind.addEventListener ("pointerover",
                        function(e) { over1.value    = ++over1c;
                                     ta.value += "IN:over\n"; e.stopPropagation();
                                     ta.scrollTop = ta.scrollHeight; isd1 = 1; });
ind.addEventListener ("pointerout",
                        function(e) { out1.value     = ++out1c;
                                     ta.value += "IN:out\n"; e.stopPropagation();
                                     ta.scrollTop = ta.scrollHeight; isd1 = 0; });
ind.addEventListener ("pointermove",
                        function(e) { move1.value    = ++move1c; e.stopPropagation();
                                    if (isd1 == 1) { ta.value += "IN:move\n";
                                     ta.scrollTop = ta.scrollHeight; isd1 = 2;}});
```



OUT:over
OUT:enter
OUT:move
OUT:out
IN:over
IN:enter
IN:move
IN:out
IN:leave
OUT:over
OUT:move
OUT:out
OUT:leave

青い部分に入った

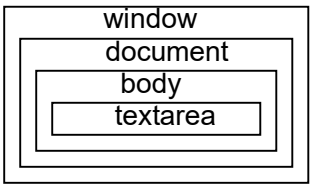
赤い部分に入った

青い部分に戻った

外に出た

クリ: addEventListener の第 3 引数を false から true に変えた場合、そして、さらに、e.stopPropagation(); を追加した場合の調査結果を見てもらおう。

```
<textarea id=ta rows=20 cols=50></textarea>
<script>
ta.value = "";
window.addEventListener("click",
                        function(e) {
                            ta.value += "window:\n";
                            ta.scrollTop = ta.scrollHeight; }, false);
document.addEventListener("click",
                        function(e) {
                            ta.value += "document:";
                            ta.scrollTop = ta.scrollHeight; }, false);
document.body.addEventListener("click",
                        function(e) {
                            ta.value += "body:";
                            ta.scrollTop = ta.scrollHeight; }, false);
ta.addEventListener("click",
                        function(e) {
                            ta.value += "ta:";
                            ta.scrollTop = ta.scrollHeight; }, false);
</script>
```



e.stopPropagation(); 入れると

texttarea に出力される内容

```
ta:body:document:window
body:document:window
```

ta 内クリック
ta 外クリック

```
ta:body:
body:
```

伝搬が body で ストップ

```
<textarea id=ta rows=20 cols=50></textarea>
<script>
ta.value = "";
window.addEventListener("click",
                        function(e) {
                            ta.value += "\nwindow:";
                            ta.scrollTop = ta.scrollHeight; }, true);
document.addEventListener("click",
                        function(e) {
                            ta.value += "document:";
                            ta.scrollTop = ta.scrollHeight; }, true);
document.body.addEventListener("click",
                        function(e) {
                            ta.value += "body:";
                            ta.scrollTop = ta.scrollHeight; }, true);
ta.addEventListener("click",
                        function(e) {
                            ta.value += "ta:";
                            ta.scrollTop = ta.scrollHeight; }, true);
</script>
```

e.stopPropagation();

```
window:document:body:ta:
window:document:body:
```

ta 内クリック
ta 外クリック

```
window:document:body:
window:document:body:
```

ta 内でクリックしたのに ta が無い！

クリ: 実際は、細かいことは考えず、addEventListener の第 3 引数は false にすれば良いんじゃ！ 第 3 引数を書かなければ false を指定したのと同じになる。

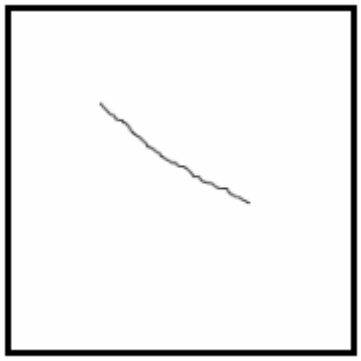
なな：いろいろ難しい話が多かったけど、マウスイベントは、何に使えるの？

クリ：たとえば、お絵かきソフトに使えるんじゃ。マウスの通った道筋をつなげばできる。

なな：なるほど。おもしろそうね！

クリ：実用的なペイントソフトを作るのは、それなりに大変じゃけど、下記のような、ごく簡単なプログラムで、マウスでドラッグした部分に線が引けるというのは面白いじゃろう。これでも、mousemove の座標に点を打つだけだと、マウスの動きが速いときに点線になってしまうので、直線でつなぐというのがミソなんじゃ。

なな：なるほど、なるほど！



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Paint</title>
  </head>
  <body>
    <canvas width=200px height=200px id=cv style="border:solid"></canvas><br>
    <script>
var ctx = cv.getContext('2d');

var i = 0, x, y;
cv.addEventListener("pointerdown",    function(e) {
                                          i = 1; x = e.offsetX; y = e.offsetY; });
cv.addEventListener("pointermove",    function(e) {
                                          if (i==1) { ctx.beginPath();
                                                      ctx.moveTo(x, y);
                                                      x = e.offsetX; y = e.offsetY;
                                                      ctx.lineTo(x, y);
                                                      ctx.stroke(); } });
cv.addEventListener("pointerup",       function(e) { i = 0; });
cv.addEventListener("pointerleave", function(e) { i = 0; });
    </script>
  </body>
</html>
```

第5回　キーボードイベント

クリ：キーボードに関するイベントには、下記のようなものがある。

・onkedown　　キーが押された時（大文字/小文字 区別しない）(shift も 1 キー)
・onkeypress　キーが押された時（大文字/小文字 区別する）(shift + a → A)
・onkeyup　　キーが離された時

使い方は

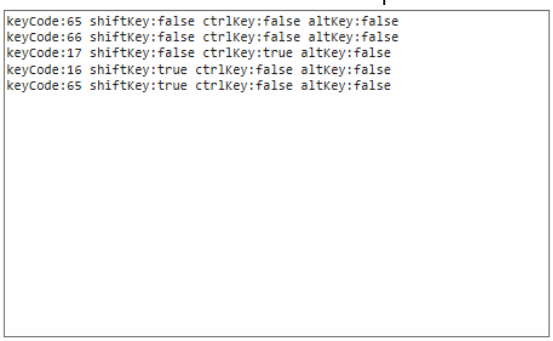
```
document.onkeydown = function (e){
    if (e.keyCode == c) { ... }
}
```

引数（イベント）のプロパティは

プロパティ	型	解説
keyCode	Number	キーコード（onkeydown）
charCode	Number	文字コード（onkeypress）
shiftKey	Boolean	Shift キーの押下状態
ctrlKey	Boolean	Ctrl キーの押下状態
altKey	Boolean	Alt キーの押下状態

キー	keyCode
0	48
A	65
F1	112
BackSpace	8
Tab	9
Clear	12
Enter	13
Command	15
Shift	16
Ctrl	17
Alt	18
CapsLock	20
Esc	27
Space	32
←	37
↑	38
→	39
↓	40
Insert	45
Delete	46
NumLock	144

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>Key event</title>
 </head>
 <body>
   <textarea id=ta rows=20 cols=70></textarea>
   <script>
ta.value = "";
document.onkeydown = function (e) {
    ta.value += "keyCode:"    + e.keyCode;
    ta.value += " shiftKey:"  + e.shiftKey;
    ta.value += " ctrlKey:"   + e.ctrlKey;
    ta.value += " altKey:"    + e.altKey + "\n";
    ta.scrollTop = ta.scrollHeight;
}
   </script>
 </body>
</html>
```

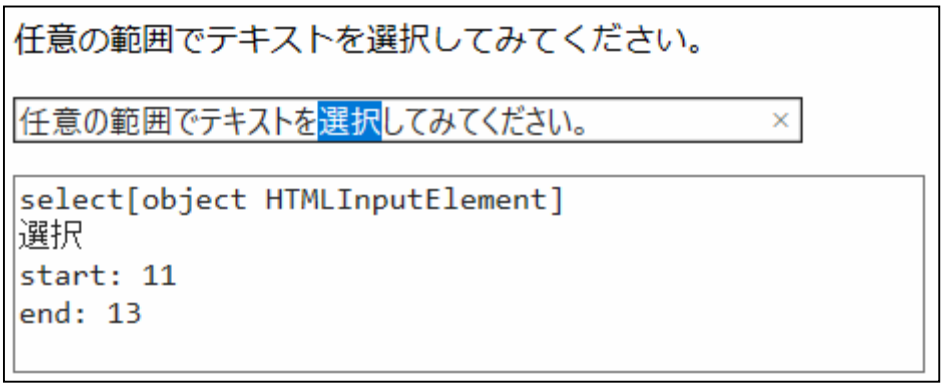



```
keyCode:65 shiftKey:false ctrlKey:false altKey:false
keyCode:66 shiftKey:false ctrlKey:false altKey:false
keyCode:17 shiftKey:false ctrlKey:true altKey:false
keyCode:16 shiftKey:true ctrlKey:false altKey:false
keyCode:65 shiftKey:true ctrlKey:false altKey:false
```

第6回　セレクトイベント

クリ：セレクトイベントって？

クリ：画面上の文字列の一部をマウスドラッグで選択した時に発生するイベントじゃ。



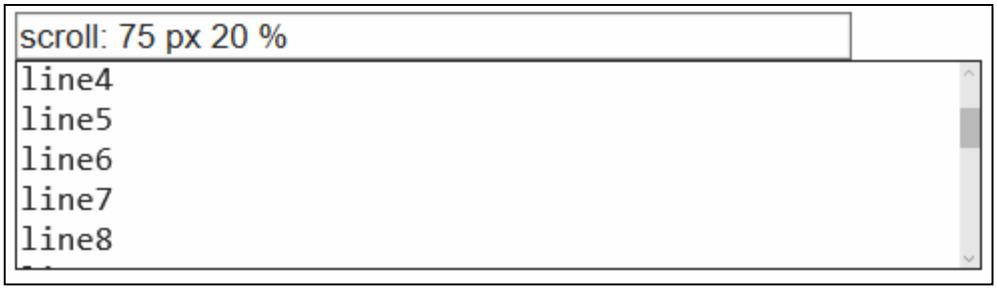

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>Scroll event</title>
    <style>body,input, textarea{font-size:20px;}</style>
 </head>
 <body>
    <p>任意の範囲でテキストを選択してみてください。</p>
    <p><input type=text size=50 value="任意の範囲でテキストを選択してみてください。"></p>
    <textarea rows=5 cols=50 id=output></textarea>
    <script>
window.onselect = function(e) {
    var input = e.target;
    var text, start, tail;
    if (input == "[object HTMLBodyElement]") {
          var selObj = window.getSelection();
        var range   = selObj.getRangeAt(0);
        text = "" + selObj;
        start = range.startOffset;
        tail = range.endOffset;
    } else {
        text = input.value.slice(Number(input.selectionStart),Number(input.selectionEnd));
        start = input.selectionStart;
        tail = input.selectionEnd;
    }
    output.value = e.type + e.target + "\n";
    output.value += text + "\n";
    output.value += "start: " + start + "\n";
    output.value += "end: " + tail + "\n";
}
    </script>
 </body>
</html>
```

第7回　スクロールイベント

なな：スクロールインベントって？

クリ：スクロール可能な DOM 要素上で、スクロール操作を行った時に発生するイベントじゃ。

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>Scroll event</title>
    <style>input,textarea{font-size:20px;}</style>
 </head>
 <body>
    <input type=text id=output size=50>
    <br>
    <textarea rows=5 cols=50 id=input></textarea>
    <script>
for (var i=1; i<21; i++) input.value += "line" + i + "\n";

input.onscroll = function(e) {
    var percent = Math.floor(input.scrollTop /
                                    (input.scrollHeight - input.offsetHeight) * 100);
    output.value = e.type + ": "    + input.scrollTop + " px " + percent + " %";
}
    </script>
 </body>
</html>
```



第8回　ファイル操作イベント

なな：ファイル操作イベントって？

クリ：ファイル読み込みをスタートしたあと、読み込みが完了した時に発生するイベントじゃ。ついでに、ファイル書き込みも紹介するが、ブラウザの場合、セキュリティ上の問題で、書き込みができるのは、ダウンロードフォルダに保存するファイルだけ。

ファイル入力

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>File menu</title>
 </head>
 <body>
  <input id="fileSel" type="file" multiple="true" onchange="readFile()" size=50>
  <br><textarea rows=30 cols=50 id=out></textarea>
  <script>
function printFile(evt) {
    var cntnt = evt.target.result;
    out.value = cntnt;
}

function readFile() {
    var reader = new FileReader();
    reader.onload = printFile;
    var files = document.getElementById("fileSel").files;
    reader.readAsText(files[0]);
}
  </script>
 </body>
</html>
```

C:\Users\User\Desktop\console.html　参照...

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="utf-8">
   <title>コンソール</title>
 </head>
 <body>
   <h3>コンソール</h3>
   <textarea rows="19" cols="80" id=pg
autofocus>1 + 2;</textarea>
   <br><input type=button onClick=go() value="実
行">
   <br>システムからのメッセージ
   <br><textarea rows="20" cols="80"
id=log></textarea>
   <script>
var geval = eval;

var logp = document.getElementById("log");
var pgp  = document.getElementById("pg");
var logd;

function clog(s) { logp.value += s; }

function log(s) { logd += s; }

function typeIs(obj) {
    return(Object.prototype.toString.call
(obj).slice(8, -1)); }
```

おまけ：ファイルへの書き込み。ファイルは、ダウンロードフォルダに置かれる。

ファイル出力

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>blob test</title>
 </head>
 <body>
     <script>
function fsave(data,fname,ftype) {
    var blob = new Blob([data], {type: ftype});
    if(window.navigator.msSaveBlob) {
        window.navigator.msSaveBlob(blob, fname); }
    else {
        var a = document.createElement("a");
        a.href = URL.createObjectURL(blob);
        a.target = '_blank';
        a.download = fname;
        var pp=document.body.appendChild(a);
        pp.click();
        pp.parentElement.removeChild(pp); }
}

fsave("abc","test.txt","text/plain");
alert("done");
     </script>
  </body>
</html>
```

a 要素を作り、あたかもユーザがクリックしたかのように動作させる

ファイル書き込みを依頼たら、用済みの a 要素を削除

内容は「abc」、ファイル名は「test.txt」、タイプは「text/plain」

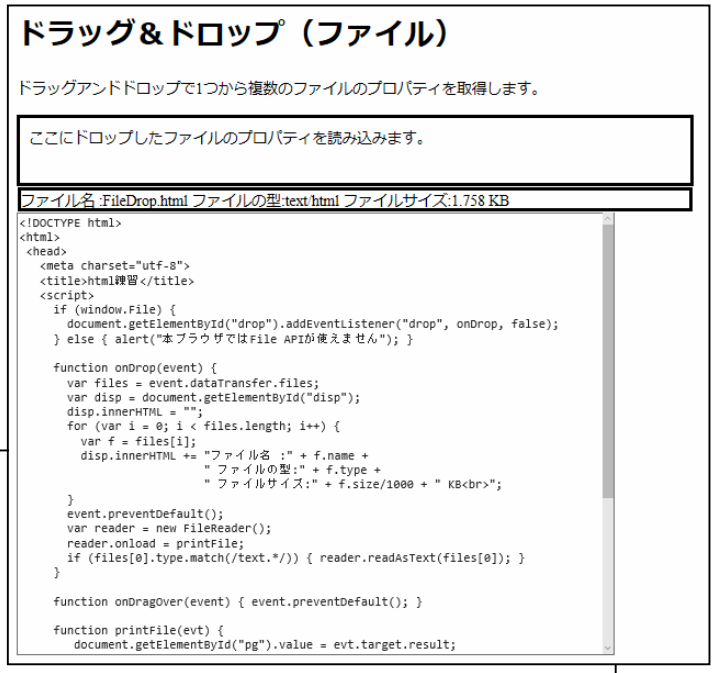
第9回　ドラッグ＆ドロップイベント

なな：ドラッグ＆ドロップイベントって？

クリ：ユーザが、ファイルアイコンなどを
ドラッグ＆ドロップ操作した時に
発生する院ベントじゃ。

なな：このサンプルプログラムは？

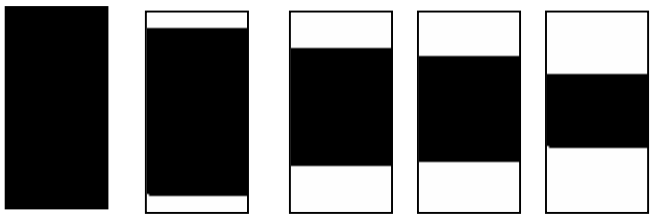
クリ：ブラウザ外からドラッグ＆ドロップで
ファイルを指定するサンプルじゃ。
ファイル内容がテキストの場合は、
内容を画面表示する。



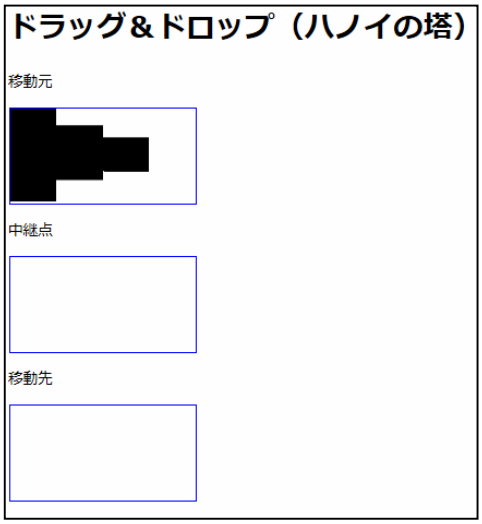

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="utf-8">
   <title>html 練習</title>
   <script>
     if (window.File) {
       document.getElementById("drop").addEventListener("drop", onDrop, false);
     } else { alert("本ブラウザでは File API が使えません"); }

     function onDrop(event) {
       var files = event.dataTransfer.files;
       var disp = document.getElementById("disp");
       disp.innerHTML = "";
       for (var i = 0; i < files.length; i++) {
         var f = files[i];
         disp.innerHTML += "ファイル名  :" + f.name +
                           "  ファイルの型:" + f.type +
                           "  ファイルサイズ:" + f.size/1000 + " KB<br>";
       }
       event.preventDefault();
       var reader = new FileReader();
       reader.onload = printFile;
       if (files[0].type.match(/text.*/)) { reader.readAsText(files[0]); }
     }

     function onDragOver(event) { event.preventDefault(); }

     function printFile(evt) {
         document.getElementById("pg").value = evt.target.result;
     }
   </script>
 </head>
 <body>
    <h1>ドラッグ＆ドロップ（ファイル）</h1>
    <p>ドラッグアンドドロップで 1 つから複数のファイルのプロパティを取得します。</p>
    <div id="drop"
         style="width:700px; height:50px; padding:10px; border:3px solid"
         ondragover="onDragOver(event)" ondrop="onDrop(event)">
       ここにドロップしたファイルのプロパティを読み込みます。</div>
<div id="disp" style="width:718px; height:20px; margin:1px;
                      border:3px solid #000000;"></div>
<textarea id=pg cols=85 rows=30 border=1></textarea>
 </body>
</html>
```

クリ：画面上の画像をドラッグ＆ドロップで移動するというサンプルプログラムじゃ。

```
<!DOCTYPE html>
<html lang="ja">
    <head>
        <meta charset=utf-8>
        <title>ドラッグ＆ドロップ（ハノイの塔）</title>
        <style>
            div         { width:201px; height:102px; margin:1px;
                          background-color:#ffffff; border:1px solid #0000ff; }
        </style>
        <script>
            function dragstart(event){
                event.dataTransfer.setData("text", event.target.id);
            }

            function dragover(event){
                event.preventDefault();
            }

            function drop(event){
                var id_name = event.dataTransfer.getData("text");
                var drag_elm =document.getElementById(id_name);
                event.currentTarget.appendChild(drag_elm);
                event.preventDefault();
            }
        </script>
    </head>
    <body>
        <h1>ドラッグ＆ドロップ（ハノイの塔）</h1>
        <p>移動元</p>
        <div ondragover="dragover(event)" ondrop="drop(event)"><img
              src="1.png" id="1" draggable="true" ondragstart="dragstart(event)"><img
              src="2.png" id="2" draggable="true" ondragstart="dragstart(event)"><img
              src="3.png" id="3" draggable="true" ondragstart="dragstart(event)"></div>
        <p>中継点</p>
        <div ondragover="dragover(event)" ondrop="drop(event)"></div>
        <p>移動先</p>
        <div ondragover="dragover(event)" ondrop="drop(event)"></div>
    </body>
</html>
```



1.png　1.5.png　2.png　2.5.png　3.png



第10回　タイマーイベント

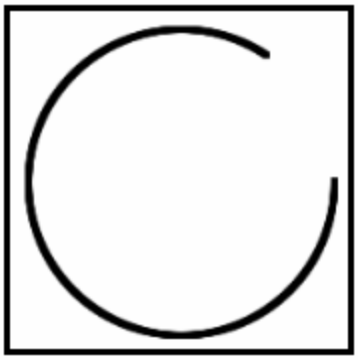
なな：タイマーイベントって？

クリ：on…. のようには表現しないが、指定した時間後とか、指定した時間間隔で発生するイベントじゃ。

setInterval(実行する関数，時間指定 [，引数1，引数2，…])　一定間隔に繰り返し。時間指定は ミリ秒 単位。

setTimeout(実行する関数，時間指定 [，引数1，引数2，…])　指定時間後に一回。時間指定は ミリ秒単位。

ゆっくり、円を描画するアニメーションの例じゃ。

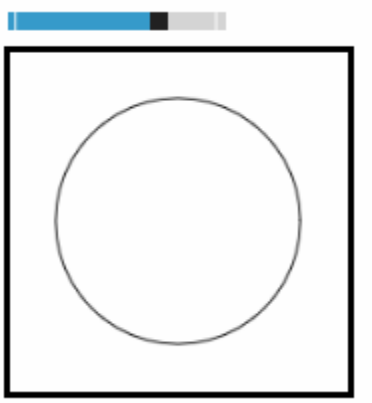


```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Paint</title>
  </head>
  <body>
    <canvas width=200px height=200px id=cv style="border:solid"></canvas><br>
    <script>
var ctx = cv.getContext('2d');

var a = 0;

setInterval(step, 5);

function step() {
    var x = 90 * Math.cos(a);
    var y = 90 * Math.sin(a);
    a += 0.01;
    if (a > 2*Math.PI) {
        a -= 2*Math.PI;
        ctx.clearRect(0,0,200,200);
    }
    ctx.fillRect(x+100, y+100, 4, 4);
}
    </script>
  </body>
</html>
```

第11回　チェンジイベント

なな：チェンジイベントって？。

クリ：input 要素の内容が書き換わった時に発生するイベントじゃ。type=range の input を使っている。



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Paint</title>
  </head>
  <body>
    <input type=range min=1 max=100 step=1 onchange=go(this.value)><br>
    <canvas width=200px height=200px id=cv style="border:solid"></canvas><br>
    <script>
var ctx = cv.getContext('2d');

function go(r) {
    ctx.clearRect(0,0,200,200);
    ctx.beginPath();
    ctx.arc(100, 100, r, 0, 2*Math.PI);
    ctx.stroke();
}
    </script>
  </body>
</html>
```

クリ：チェンジイベントが発生するタイミングはブラウザごとに異なるので要注意。内容を修正している時に、時々刻々発生するブラウザがあるかと思うと、修正後にフォーカスを外した時に発生するブラウザもある。

クリ：input 要素の一覧じゃ。html5 で新たに追加されたものも結構あるぞ。

type="text　"一行テキストボックスを作成する（初期値）
type="button" 汎用ボタンを作成する
type="checkbox"　チェックボックスを作成する
type="radio"　ラジオボタンを作成する
type="password"　パスワード入力欄を作成する
type="file"　サーバーへファイルを送信する
type="submit" 送信ボタンを作成する
type="image" 画像ボタンを作成する
type="reset"　リセットボタンを作成する
type="hidden" 画面上は表示されない隠しデータを指定する

HTML5から追加になったもの。

type="range" レンジの入力欄を作成する
type="color"　色の入力欄を作成する
type="email"　メールアドレスの入力欄を作成する
type="search "検索テキストの入力欄を作成する
type="tel　"電話番号の入力欄を作成する
type="time"　時間の入力欄を作成する
type="datetime-local" UTC（協定世界時）によらないローカル日時の入力欄を作成する
type="url"　URLの入力欄を作成する
type="datetime"　UTC（協定世界時）による日時の入力欄を作成する
type="date"　日付の入力欄を作成する
type="month" 月の入力欄を作成する
type="week"　週の入力欄を作成する
type="number"　数値の入力欄を作成する

第12回　window のイベント

なな：window のイベントって？。

クリ：ここでは、DOM 部品から document を経由して上がってくるイベントではなくて、windows だけのイベントについて考えよう。

・onload　　ウェブページのロード完了時
・onresize　　ウィンドウのリサイズ時
・onscroll　　ウィンドウ、textarea などのスクロール時

これ以外にも、onclose、onunload、onbeforeunload といった、他のページに移る時に発生するイベントもあるが、ブラウザごとに仕様や対応状況が異なるので、ここでは触れないことにしよう。

window は省略することが可能。windows.onload と、onload は同じ意味になる。

window.onload はよく使うものだが、ふたつ以上あると、さいごに設定したものだけが有効になるので要注意じゃ。ライブラリに window.onload があるのに気づかずに、ユーザプログラムで設定すると、ライブラリの定義が実行されなくなってしまう。設置が複数あっても、すべてが有効になる window.addEventListener (“load”, func, false) を使うのがお勧めじゃ。window.onload との混在でも大丈夫。windows.onload と、document.body.onload は同一に扱われ、重複すると、最後に設定されたものが有効になる。

window.resize は、ブラウザの画面サイズが変更された時、window.scroll は、ブラウザの画面でスクロール操作が行われた時に発生するイベント。

ブラウザの画面サイズを JavaScript で調べるには、下記のプロパティを使う

- screen.width、screen.height　　モニターの幅/高さ
- window.innerWidth、window.innerHeight　　ブラウザ表示面の幅/高さ（スクロールバー含む）
- window.outerWidth、window.outerHeight　　ブラウザ外観の幅/高さ
- document.body.clientWidth、document.body.clientHeight
  ドキュメント全体の幅/高さ（スクロールバー含まず、画面に出ていない部分を含む）
- document.documentElement.clientWidth、document.documentElement.clientHeight
  ドキュメントの表示領域（スクロールバー含まず、画面に出ている部分だけ）

ブラウザのスクロール量を JavaScript で調べるには、下記のプロパティを使う

- window.scrollX、window.scrollY　　Firefox、Safari、Chrome だけ
- document.documentElement.scrollLeft、document.documentElement.scrollTop　　IE、Firefoxだけ

実用的には、下記の記述を使えば、どちらでも大丈夫。

- window.scrollX || document.documentElement.scrollLeft　　横スクロール量
- window.scrollY || document.documentElement.scrollTop　　縦スクロール量