入力されたキーを受け取る（Internet Explorer 編）

ここから、キーボードで入力されたキーを JavaScript で受け取る方法について見ていきます。

キー入力の受付は、ブラウザによって若干違います。 最初は一番普及している、 Internet Explorer でキー入力を受け取る方法について見てみたいと思います。

今回は、「Ｂ」キーを押すと前のページに戻り、 「Ｆ」キーを押すと次のページに進むスクリプトを組んでみることにしましょう。 ページの移動に関して復習したい方は、以下のページを参照なさってください。

キー入力イベント

まず最初に、キー入力イベントについて見ておきましょう。 以下の３つがあります。

onkeydown…キーが押し下げられた時

onkeyup…キーを押した後、離された時

onkeypress…キーを押している最中

今回は、キー押下時「onkeydown」を用いたいと思います。 このイベントを JavaScript で受け取るには、 HTML のヘッダーに以下の一行を記入します。

document.onkeydown = 関数名;

キーが押された時の処理を記入した関数名を右辺に指定します。 これで入力されたキーを受け取れるようになります。

キー入力でページ移動するスクリプト

では具体的にキーイベントを利用したスクリプトを見ていきましょう。 今回、ページ移動処理を記入する関数の名称は「pageMove」にしました。 では HTML のヘッダーに以下のように記入してみてください。

```
<script>

//キー押下時に関数 pageMove()を呼び出す
document.onkeydown = pageMove;



function pageMove()
{
  if (event.keyCode == 66)  //「Ｂ」が押されたか確認
  {
    history.back();
  }
```

```
  if (event.keyCode == 70)　//「Ｆ」が押されたか確認
  {
    history.forward();
  }
}
```

</script>

ちなみにこのページのヘッダーに上記のスクリプトを記入しています。 キーボードの「Ｂ」や「Ｆ」を押して移動が可能か確かめてみてください （Internet Explorer と幾つかのブラウザでは動きますが、FireFox5 等では動きません）。

スクリプトの説明

では上記のスクリプトを見ていきましょう。

キー押下時の処理を指定する

document.onkeydown を使ってキーを押された時に関数 pageMove()を呼び出すようにしています。 右辺は関数名を指定なので、カッコは書きません。

関数 pageMove()

関数 pageMove()の中で、「Ｂ」または「Ｆ」が押されたか調べます。 特定のキーが押されたかどうか調べるには、以下のようにします。

if (event.keyCode == キーコード) { 処理 }

ここでキーコードというものが出てきましたが、 これは各キーに割り当てられている数字のことです。 「Ａ」ならば 65、「Ｚ」なら 90、スペースキーなら 32…という感じで、 キーによって特定の数字が付されているのです。

event.keyCode を使うと、押されたキーのキーコードを取得できます。 その数値を if 文を使って調べることにより、特定のキーが押されたか確認できるのです。

今回は「Ｂ」（キーコード 66）を押された時は history.back()で前のページに戻り、 「Ｆ」（キーコード 70）が押された時は history.forward()で次のページに進むようにしました。

キーコードを調べる

キーコードを調べるには、以下のページを用いることができます。

各ブラウザのキーコード表[JavaScript]

またリファレンス本等にも大抵載っています。1 冊持っておかれると良いかも知れません。私が愛用しているリファレンス本の最新版のリンクも掲載しておきます。

改訂第 5 版 JavaScript ポケットリファレンス

注意点として、この取得されるキーコードが、 onkeydown の時と onkeypress の時で違うということが挙げられます。 プログラムを組むときはしっかりキーコード表で確認することをお勧めします。

次のページでは、FireFox でキーボード入力を受け付ける方法について考えます。

入力されたキーを受け取る（FireFox 編）

前のページでは、Internet Explorer でキー入力を受け付ける方法について見ました。 他にも Opera や Google Chrome，Safari などの最新版でも動作します。

しかしブラウザシェア率第 2 位の FireFox では残念ながら動きません（Ver.24 の時点では）。

それでこのページでは、FireFox でも動作するスクリプトを組んでみたいと思います。

今回も前のページと同じく、「B」キーを押すと前のページに戻り、 「F」キーを押すと次のページに進むスクリプトを組んでみることにします。

FireFox でキー入力を受け付ける場合

FireFox でキー入力を受け付ける場合、event.keyCode が使えません。 それで、この部分を次のように変更します。

event → 関数の引数を指定

keyCode → which

具体的には、以下のようなコードになります。

```
document.onkeydown = pageMove;


function pageMove(evt)
{
  if (evt.which == 66)  //「B」が押されたか確認
  {
    history.back();
  }
  if (evt.which == 70)  //「F」が押されたか確認
  {
    history.forward();
  }
}
```

このページのヘッダーに、上記のスクリプトを記入しています。 動くかどうか確認してください。 Opera や Chrome、Safari 等では動作すると思います。 しかし今度は Internet Explorer で動作しません（IE10 では動作することを確認）。

では両方のブラウザで動かすにはどうしたら良いか、次に見てみることにしましょう。

両方のブラウザで動作させる

Internet Explorer と FireFox の両方でキー入力を受け付けるには、以下のようにします。

```
<script>

document.onkeydown = pageMove;

function pageMove(evt)
{
   var kcode;   //キーコードを格納する変数

   //document.all は Internet Explorer でのみ使用可能
   if (document.all)
   {
      kcode = event.keyCode;
   }
   else
   {
      kcode = evt.which;
   }

   if ( kcode == 66 ) { history.back(); }
   if ( kcode == 70 ) { history.forward(); }
}

</script>
```

変数 kcode を宣言して、取得したキーコードを格納するようにしています。
そして DOM 編の最初で少し触れましたが、 document.all を使って、Internet Explorer でのみ動作するスクリプトを記述します。 キーコードを取得するには、event.keyCode でした。 取得したキーコードは変数 kcode に代入します。
else 文の中（Internet Explorer 以外のブラウザ）では、「引数.which」を使ってキーコードを取得し、 変数 kcode に代入しています。
あとは変数 kcode に代入された数値を調べて、戻る・進むの処理を記述しています。

タイピングゲームを作る
前回までで、JavaScript でキーボード入力を受け付ける方法について学びました。 このページではその機能を利用して、簡単なタイピングゲームを作ってみたいと思います。
タイピングゲームといっても無数にあるので、本当に簡単なものにしたいと思います （本

分は JavaScript の勉強なので)。 ここではＡ～Ｚまでの文字をランダムに 200 文字表示して、 その入力速度を計ってみることにします。
このページのスクリプトを理解できれば、 もっと複雑で高性能なタイピングゲームも作れるようになるでしょう。 是非じっくりと読んで理解していってください。

タイピングゲーム作成の流れ
タイピングゲームを作るために、以下の流れに従ってスクリプトを組んでみることにしましょう。
問題を出題
A-Z の文字を配列 moji に格納します。
文字に対応するキーコードを、配列 kcode を作成して格納します。
0～25 までの乱数を 200 個作成して、配列 rnd に格納します。
何問目かをカウントする変数 cnt に、0 を代入します。
出題枠に innerHTML で、A-Z の文字をランダムに 200 文字表示します。
最初にキーボード入力された時
キー入力があると、キーコードを取得します。
取得したキーコードが問題文先頭文字のキーコードと同じか確認します。
同じ場合、現在時刻を取得して、変数 typStart に格納します。
変数 cnt を+1 します。
innnerHTML で、出題枠の最初の文字を削って 199 文字を表示し直します。
キーボード入力の続きと終了
キー入力がある度に、キーコードを取得します。
取得したキーコードが問題文先頭文字のキーコードと同じか確認します。
同じなら cnt を+1 します。
cnt が 200 になったら、現在の時間を取得して変数 typEnd に格納します。
typEnd-typStart で掛かった時間を計算します。
innerHTML で問題枠に「GAME 終了」と『時間』を表示します。
cnt が 200 未満なら、先頭文字を削除し、innerHTML で問題文を書き直します。

タイピングゲームのスクリプト
以下のスクリプトを HTML のヘッダーに記入してみてください。 今回は解説は殆どしていませんので、スクリプト中のコメントを参考になさってください。

```
<script>

document.onkeydown = typeGame;　//キー押下時に関数 typeGame()を呼び出す
```

```
//文字を格納する配列
var moji = new Array("Ａ","Ｂ","Ｃ","Ｄ","Ｅ","Ｆ","Ｇ","Ｈ","Ｉ",
                     "Ｊ","Ｋ","Ｌ","Ｍ","Ｎ","Ｏ","Ｐ","Ｑ","Ｒ",
                     "Ｓ","Ｔ","Ｕ","Ｖ","Ｗ","Ｘ","Ｙ","Ｚ");

//キーコードを格納する配列
var kcode = new Array(65,66,67,68,69,70,71,72,73,
                      74,75,76,77,78,79,80,81,82,
                      83,84,85,86,87,88,89,90);

//0～25 までの乱数を格納する配列
var rnd = new Array();




//グローバル変数群
var mondai = "";          //問題の文字列を格納
var cnt=0;                //何問目か格納
var typStart,typEnd;      //開始時と終了時の時刻を格納




//0～25 までの乱数を 200 個作成して配列 rnd に格納する関数
function ransu()
{
  for ( var i = 0 ; i < 200 ; i++ )
  {
    rnd[i] = Math.floor( Math.random() * 26 );
  }
}
```

```
//タイピングゲームの問題をセットする関数
function gameSet()
{
   //問題文とカウント数をクリアする
   mondai="";
   cnt=0;

   //乱数作成関数の呼び出し
   ransu();

   //問題文の作成（配列 moji の要素をランダムに 200 文字繋げる）
   //mondai= "" + moji[rnd[0]] + moji[rnd[1]] + … + moji[rnd[199]]となる
   for ( var i = 0 ; i < 200 ; i++)
   {
      mondai =    mondai + moji[ rnd[i] ];
   }

   //問題枠に表示する
   document.getElementById("waku").innerHTML = mondai;
}



//キー入力を受け取る関数
function typeGame(evt)
{
   var kc;   //入力されたキーコードを格納する変数

   //入力されたキーのキーコードを取得
   if (document.all)
   {
      kc = event.keyCode;
   }
   else
   {
```

```
    kc = evt.which;
}
//入力されたキーコードと、問題文のキーコードを比較
if (kc == kcode[ rnd[cnt] ])
{
    //以下、キーコードが一致した時の処理

    //最初の 1 文字が入力された時間を記録する
    if (cnt==0)
    {
        typStart = new Date();
    }

    cnt++; //カウント数を＋1 にする

    //全文字入力したか確認
    if ( cnt < 200)
    {
        //問題文の頭の一文字を切り取る
        mondai = mondai.substring(1,mondai.Length);

        //問題枠に表示する
        document.getElementById("waku").innerHTML = mondai;
    }
    else
    {
        //全文字入力していたら、終了時間を記録する
        typEnd = new Date();

        //終了時間－開始時間で掛かったミリ秒を取得する
        var keika = typEnd - typStart;

        //1000 で割って「切捨て」、秒数を取得
        var sec = Math.floor( keika/1000 );

        //1000 で割った「余り(%で取得できる)」でミリ秒を取得
```

```
        var msec = keika % 1000;

        //問題終了を告げる文字列を作成
        var fin="GAME 終了　時間："+sec+"秒"+msec;

        //問題枠にゲーム終了を表示
        document.getElementById("waku").innerHTML = fin;
    }
  }
}
</script>
```

そして body タグの onload イベントで、関数 gameSet()を呼び出して問題を出題させます。BODY 内には id 名「waku」を付けた div タグ（p タグや td タグ等も可）を一つ作っておきます。

```
<body onload="gameSet()">

<div id="waku"></div>
```

上記スクリプトのサンプルです。出題枠は CSS で少し装飾しています。詳しくはソースを覗いてみて下さい。

スクリプトの補足

上記スクリプトで少しだけ補足しておきましょう。配列 rnd は 0～25 のランダムな数字が入っています。 仮に rnd[0]=3 だとすると、moji[rnd[0]]="Ｄ",kcode[rnd[0]]=68 となります。 キーボードのＤを押された時のキーコードは 68 なので、問題文と対応しているわけです。 rnd[n]の n の値が何になろうと、moji[rnd[n]]のキーコードは kcode[rnd[n]]なのです。

仮に今 10 文字入力したとしましょう。そうすると問題文は先頭から 10 文字削除されています。 ということで先頭は moji[rnd[10]]が表示されています。 それに対応するキーコードは kcode[rnd[10]]になります。

では 10 の替わりに変数 cnt に置き換えるとどうでしょうか？ 現在表示されている先頭文字に対応するキーコードは kcode[rnd[cnt]]と表されます。 入力されたキーと問題文のキーコードを比較する if 文の中で kcode[rnd[cnt]]が出てきた理由がお分かり頂けたでしょうか。

他に文字の切り取りや乱数の作成、時間の計算等について復習なさりたい方は、以下のリンクを参照なさってください。

グローバル変数， Math.floor()， Math.random()， substring()， innerHTML， new Date()

少し改良したタイピングゲーム
上記のタイピングゲームでは、文字がザ〜と流れて目がチカチカします。 それで少しだけ改良したものを作ってみました。プレイして比べてみてください。

このスクリプトでは、文字を切り取らずに色を薄く（灰色）していきます。 ただ上のままのスクリプトで色変えを行うと、 折り返し部分で文字が次の行に行ったり戻ったりして見にくくなりました。 それでテーブルを使って文字を表示するようにしています。
テーブルの各セルに ID 名を付けて、そのセルの文字色を灰色にするようにしています。 詳しく解説はしませんが、関心がおありの方は是非ソースを覗いて研究なさってください。