G's ACADEMY
TOKYO

HTML/CSS/JavaScript

授業ルール 円滑に授業をすすめるために

- 授業開始後にはBracketsは既に起動してる状態！
- 隣の人と話す場合は、周りに迷惑がない大きさで。
- 演習中＆コードを打ってるとき、周りの人で困ってたら教えてあげてください。できる人の近くに行った方がいいです！
- 動かない！{}や;や"などの打ち間違いが多い。
  右クリック→要素の検証→consoleでエラー確認
- 常に保存！「 Ctrl ＋ S」キーで保存！
  ※「ファイル」→「保存」

本資料の概要

第０章	HTML/CSS基本	HTML/CSSの基本を学ぶ、よく使うものだけ学びましょう
第１章	JavaScript文法	Javascriptの基本を学ぶ
第２章	Webブラウザにおける JavaScript API	HTMLやCSSに変更を加え、 イベント/セレクタ /Dom 基本操作を学ぶ
第３章	グラフィックス	Canvasの基礎から応用方法を学ぶ
第４章	マルチ・メディア	Audio/Video要素の基礎知識と簡単な使い方を学ぶ
第５章	Session History and Navigation	HistoryAPIの基礎と簡単なサンプルで仕様を理解する
第６章	表示制御	HTML5の画面表示制御方法とアニメーションの基本知識を学ぶ
第７章	ストレージ	WebStorage、IndexDB、FileAPIの基本知識と利用方法をサンプルで学ぶ
第８章	通信	WebSocketや非同期データ通信APIの基礎知識と使用方法を学ぶ
第９章	GeoLocation	Geolocationの基礎知識と緯度経度の取得方法を学ぶ
第１０章	WebWorkers パフォーマンス	WebWorkersやパフォーマンスに関する基礎知識の習得

マニュアル

- https://developer.mozilla.org/ja/docs/Web/JavaScript

JavaScript開発環境

- Brackets
- Atom Editor
- Microsoft Visual Studio Code

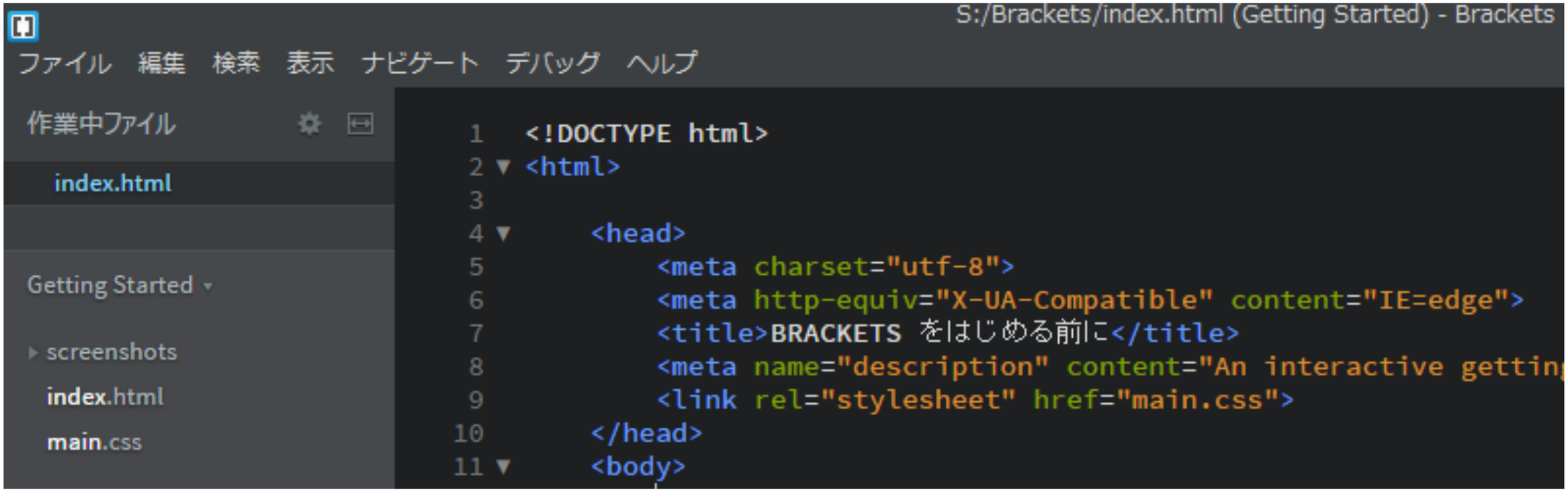
- DreamWeaver
- Eclipse JSDT
- Aptana Studio
- Firefox with Firebug + エディタ（Emacs等）

Brackets

- MITライセンスのオープンソースエディター
- PhotoshopなどAdobe製品との連携が良好
- HTML/CSS/JavaScirptなどの入力支援機能
- １クリックでLiveプレビュー確認ができる
- フォルダ選択が簡単（選択したフォルダがプロジェクト）
- CSSの色もマウスオーバーでリアルタイムに確認可能
- メニュー等が日本語表示
- 拡張機能が豊富

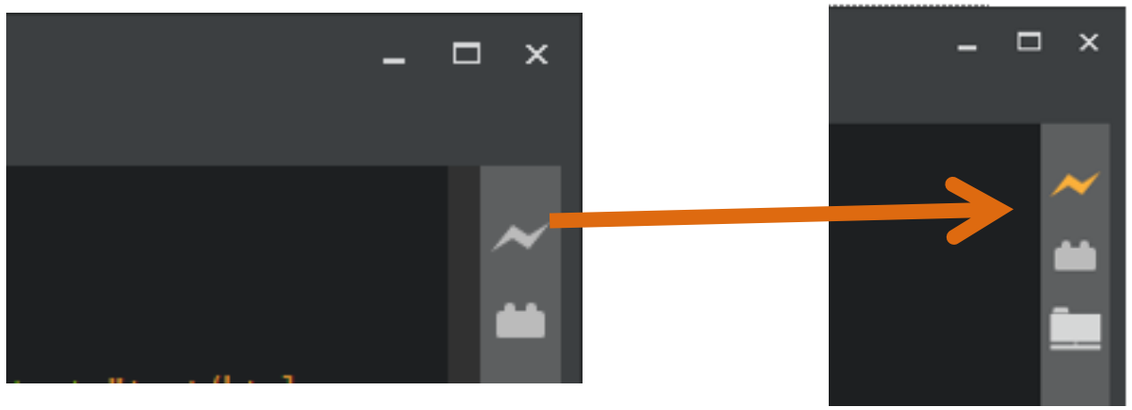
Bracketsの起動

- スタート＞すべてのプログラム＞AdobeBrackets＞Brackets

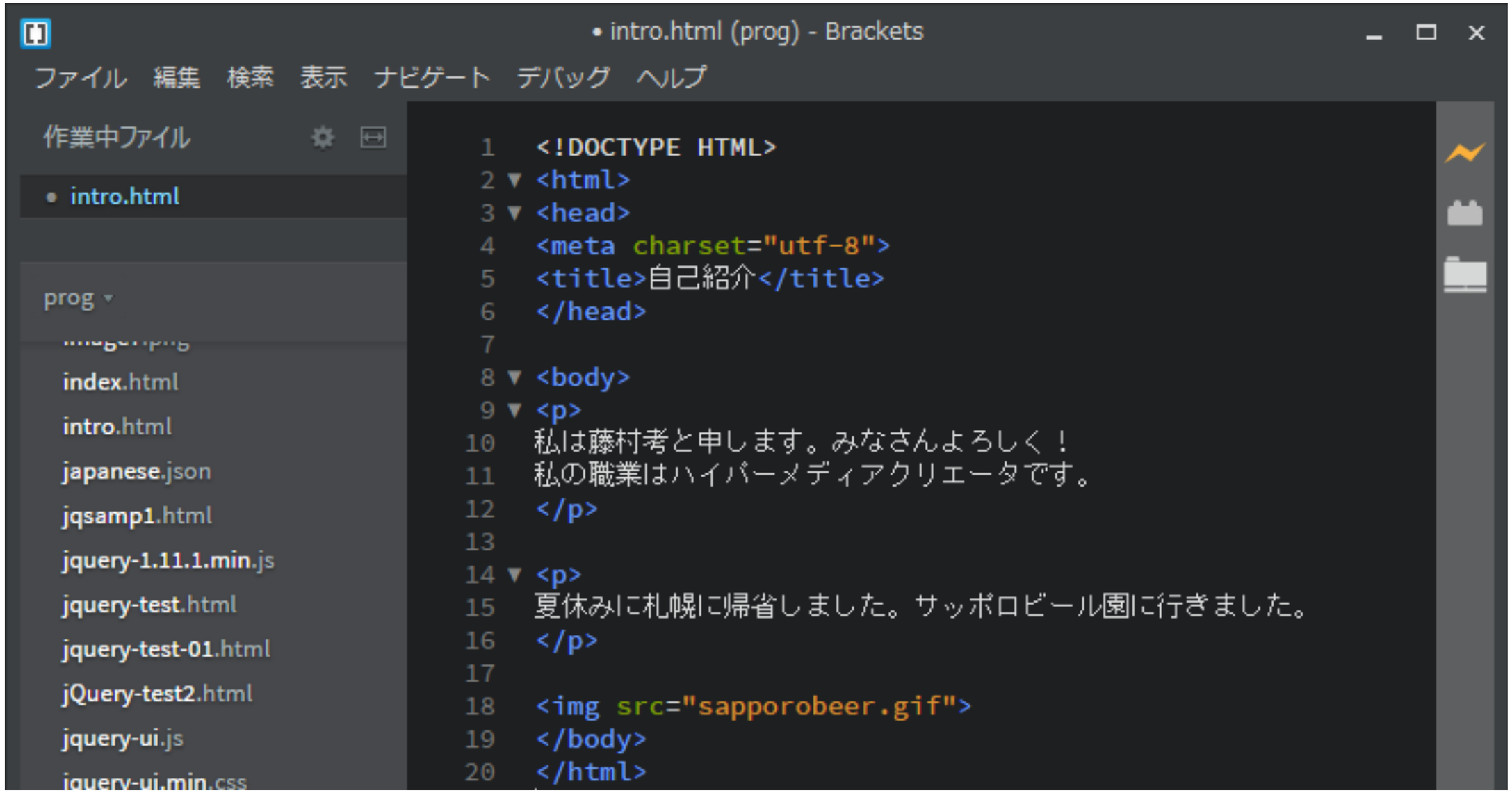


Adobe Brackets 使い方

- ファイルメニュー「フォルダーを開く…」で作業用フォルダ（sample）を指定する
- 右上のライブビューボタンをクリックするとChromeブラウザが起動する



使ってみよう


• intro.html (prog) - Brackets
ファイル 編集 検索 表示 ナビゲート デバッグ ヘルプ
作業中ファイル
• intro.html
prog
index.html
intro.html
japanese.json
jqsamp1.html
jquery-1.11.1.min.js
jquery-test.html
jquery-test-01.html
jQuery-test2.html
jquery-ui.js
jquery-ui.min.css
1 <!DOCTYPE HTML>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>自己紹介</title>
6 </head>
7
8 <body>
9
10 私は藤村考と申します。みなさんよろしく！
11 私の職業はハイパーメディアクリエータです。
12
13
14
15 夏休みに札幌に帰省しました。サッポロビール園に行きました。
16
17
18 <img src="sapporobeer.gif">
19 </body>
20 </html>

第０章

HTML/CSS基本

HTML/CSSの関係

【 各言語の役割 】

HTML	構造・文章・意味
CSS	デザイン（装飾）、レイアウト

HTML基礎

HTML基本

◇HTML基本

htmlファイルとは →

○○○.html と拡張子がhtmlのファイル

htmlファイルの中に記述するのは →

htmlマークアップ言語

htmlファイルはダブルクリックで、ブラウザにて確認も可能

※別のファイルを読み込んでる場合にはその階層構造を維持する

※本来はWebサーバにhtmlファイル群を置き、URLからアクセスする。

HTML基本

◇HTML基本

htmlの開始タグ　→　**<html>**

htmlの終了タグ　→　**</html>**

◇HTMLファイル内に記述

```
<!DOCTYPE html>
<html>
  <head>
  <meta charset="UTF-8">
  </head>
  <body>
      <h1>見出し１</h1>
   <div></div>
  </body>
</html>
```

doctype：文章宣言

html要素：　記述領域

head要素：初期設定

body要素：表示領域

HTML基本

◇HTML要素

例） HTMLを学ぶ
　　開始タグ　　終了タグ

HTMLの要素は「開始タグ」と「終了タグ」のセットで基本的に記述します。
上記例ではdivタグを使用してますが、divタグには一つの区画・範囲・まとまりと言った意味で一般的に解釈されています。本来divタグには特に意味を持たないのですが、実際はそのように解釈されています。
（imgタグ、inputタグなど、終了タグを使用しないタグもあります。）

HTML基本

◇HTML属性

例） HTMLを学ぶ
div要素のid属性にはatt1値がセットされています
div要素のclass属性にはcss1値がセットされています。

HTML要素に対して、「情報・設定値」を追加できます。それが属性です。
属性はHTML要素のタグに対して記述できます。
※上記例では、divタグに対して属性を追加記述してます。

◇HTML要素

Webアプリでよく使われるタグを抜粋

header・・・・・・・・・・・・・・・・ サイト名、Webサイトロゴ、メニュー
h1~h6 ・・・・・・・・・・・・・・・ 見出し
nav ・・・・・・・・・・・・・・・・・・ ナビゲーション
ul ・・・・・・・・・・・・・・・・・・・ 箇条書き、リスト表示
li ・・・・・・・・・・・・・・・・・・・ リストの1アイテムとして使用
p ・・・・・・・・・・・・・・・・・・・ 段落
footer ・・・・・・・・・・・・・・・ 著作権、規約、リンク、連絡先情報、住所
div ・・・・・・・・・・・・・・・・・・ 意味はない（レイアウト・装飾に使用）
a ・・・・・・・・・・・・・・・・・・・ リンクを作成
img ・・・・・・・・・・・・・・・・ 画像
form ・・・・・・・・・・・・・・・ 入力フォーム(input,select,textArea)

例）
```
<ul>
    <li> 食パン</li>
    <li> アンパン</li>
    <li>クリームパン</li>
</ul>
```

◇HTML要素

```
<body>
 < header >
  <h1>ページタイトル</h1>
  <nav>
    <ul>
      <li><a href="home.html">HOME</a></li>
      <li><a href="./about.html ">ABOUT</a></li>
      <li><a href="../menu.html">MENU</a></li>
      <li><a href="../../access.html">ACCESS</a></li>
    </ul>
  </nav>
 </ header >

 <div> <!-- main要素でもOK -- >
  <h1> Webクラス</h1>
  <p>文章・・・</p>
      <h2>HTML</h2>
      <p>文章・・・</p>
      <img src="./images/sample.png" alt="サンプル用のイメージです">
 </div>

 < footer >
  <p> <small>Copyright © デジタルハリウッド</small></p>
 </ footer >
```

./	＝	現在のフォルダ
../	＝	１つ上のフォルダ
../../	＝	２つ上のフォルダ

<img src＝"画像ファイルを指定">

CSS基礎

CSSでHTMLを装飾しよう！

◇CSS基本

CSSの開始タグ　→　<style>

CSSの終了タグ　→　</style>

◇HTML内へのCSS記述例

```
<html>
  <head>
  <meta charset="UTF-8”>
  <style>
      ～この中に記述～
  </style>
  </head>
  <body>
  <h1>見出し１ </h1>
  <div>ここに名前とか</div>
  …
```

◇CSS基本

【font-size】文字の大きさ

```
font-size:12px;    /* 数字が大きい程、文字を大きく表示する */
```

【color】文字の色

```
color:#0000ff;    /* 青 */
color:#000000;  /* 黒 */
color:#ff0000;   /* 赤 */
```

色の参考：「原色大辞典」 http://www.colordic.org/

◇CSSの記述例

```
<style>
  div {
        font-size:12px;
    color:#0000ff;
  }
</style>
```

◇CSS基本

【background-color】背景

background-color:#000000;　/* 黒 */
background-color:#cccccc;　/* グレー */

【border】枠に線をひく

border: 2px double #0000ff;　/* 線の太さ　二重線　青 */
border: 3px solid #ff0000;　/* 線の太さ　一線　赤 */
border: 5px dotted #777777;　/* 線の太さ 点線　グレー */

色の参考：「原色大辞典」 http://www.colordic.org/

◇CSSの記述例

```
div {
   border: 3px solid #ff0000;
   background-color:#000000;
   color:#0000ff;
}
```

◇CSS基本

【margin】「タグの外側」に余白をつける

margin ・・・・・・・・・・・・・・・・・・・・・・・ 上下左右にマージン
margin-top・・・・・・・・・・・・・・・・・・・・ 上マージンを指定する
margin-bottom・・・・・・・・・・・・・・・・・ 下マージンを指定する
margin-left・・・・・・・・・・・・・・・・・・・・ 左マージンを指定する
margin-right・・・・・・・・・・・・・・・・・・・ 右マージンを指定する

◇CSSの記述例

```
div {
   border: 3px solid #ff0000;
   background-color:#000000;
   color:#0000ff;
   margin: 10px;
}
```

◇CSS基本

【padding】「タグの内側」に余白をつける

padding･････････････････････ 上下左右にパディング
padding-top･････････････････上パディングを指定する
padding-bottom･･････････････下パディングを指定する
padding-left･･････････････････左パディングを指定する
padding-right･････････････････右パディングを指定する

◇CSSの記述例

```
div {
  border: 3px solid #ff0000;
  background-color:#000000;
  color:#0000ff;
  padding:10px;
}
```

◇CSS基本

【width】背景

width:100px;　/* 固定横幅　（数値は任意）　*/

width:100%;　/* 可変横幅　（数値は任意）　*/

※%は親のHTMLタグの幅に合わせて定義される

【height】枠に線をひく

height:100px;　/* 固定横幅　（数値は任意）　*/

height:100%;　/* 可変横幅　（数値は任意）　*/

※%は親のHTMLタグの幅に合わせて定義される

◇CSSの記述例

```
div {
   border: 3px solid #ff0000;
   width:400px;
   height:300px;
}
```

CSSセレクタ

CSSセレクタはHTMLの属性を指定することができます

◇CSSの記述例

**Element → テスト **

#id → テスト

.class → テスト

[name=a] → テスト

◇HTML内への記述例

テスト

```
[name=a]{
    border:1px solid #ff0000;
}
```

CSS:TIPS

◇TIPS

ブラウザ表示時のHTMLとブラウザにできる隙間を消す方法。
（bodyタグに対してmargin:0; をセット）

◇HTML内への記述例

```
body{
    margin:0;
}
```

◇li(リストの点)を非表示にする方法

```
li{
    list-style-type:none;
}
```

CSS:TIPS

◇TIPS

スマートフォンなどのメニューを作成するTips

```
ul{
    padding: 0;
    border-top: 1px solid #000000;
    display:table;  /* 定義 */
    table-layout: fixed;
    width:100%;
}
li{
    border: 1px solid #666;
    background-color: #FFF;
    display:table-cell;
}
```

第１章

Javascript文法

Javascript概要

Javascript概要

【 各言語の役割 】

HTML	構造・文章・意味
CSS	デザイン（装飾）、レイアウト
Javascript	ユーザー操作・イベント発生からの インタラクティブな動き

注意）JavaScript と Java は似ている部分もありますが、基本的には異なっています

Javascript基礎

HTMLのscriptを記載する場所

※サンプル：sample01.html を参照

```
<!doctype html>
<html lang="ja">
<head>

<meta charset="utf-8">
<title> sample01 </title>
<script>
   //一行コメント
    /*
      JavaScriptコード
      複数行コメント
    */
</script>

</head>
<body>
 <h1>sample01:コメント確認</
</body>
</html>W
```

文字化けを起こさないように、
head要素（タグ）の一番上に記述するのがベストです。
※最近ではUTF-8が基本です。
※他にはShift_JIS,EUC-JPなどあります。
W

一般的に、
head要素（タグ）内に記述、
最初はtitle要素の下に記述と覚えておくといいでしょう。

【 サンプル動作確認 】

◇alertを使用

```
<script>
    alert("Hello, world"); //シングルクォートで文字列を括っている
</script>
```

注意）行末に「 ； 」セミコロンを記述する。

◇console.logを使用 ［表示方法：ブラウザ → 画面右クリック → “要素を検証”を選択 ］

```
<script>
    console.log("Hello, world"); //ダブルクォートで文字列を括っている
</script>
```

他にも以下console.* があります。

dir(), assert(),debug(),dirxm(),error(),info(),log()
profile(),profileEnd(),trace(),warn()

Javascriptの基礎

※サンプル：sample01.html

【 javascriptの書き方 】

◇HTML5での記述

```
<head>
<script> /* JavaScriptコードをscripタグ内に記述！ */ </script>
</head>
```

【 コメント 】

◇一行コメント

```
//一行コメント
```

◇複数行コメント

```
/*
JavaScriptコード
複数行コメント
*/
```

※サンプル：sample04.html

【 変数 （*variable*）】

数値や文字を入れる箱のことです。
javascriptは他言語と違い、変数の型を使用しません。
（ 自動で認識し、型変換がおこなわれます ）

```
<script>
    var name = 'hello';     //文字列(string) ※文字列は「'」「"」どちらでもOK
    var name = "world" //文字列(string) ※文字列は「'」「"」どちらでもOK
    var age = 100;  //数値(number) //数値は「'」「"」を使用しなくてもOK
    var age = -10;   //数値(number) //数値は「'」「"」を使用しなくてもOK
    var num = null;  //何もない
</script>
```

入力エラー
- 初めに「数値」が書かれている場合
- 「ー」などが書かれている場合
- 宣言しないで使用すると、
  グローバル変数という扱いになります。

エスケープ処理
エスケープ文字は「¥」（バックスラッシュ）
特殊文字の改行やタブは「¥n」「¥t」

javascriptの基礎

【 型 （*data type*）】

```
string      'str';      // "文字列"
number      1;          // 数値
boolean     true;       // 「真 = true」 or 「偽 = false」
object                  // 沢山のデータを含めることができる
undefined               // 何のデータも代入されていない状態"を指す
```

プリミティブ型（*primitive data type*）
データを格納するための最も基本的な型。
整数値・実数値といった具体的な「値」を
格納するための変数を作るための型、それが「プリミティブ型」。

number型 の注意

JavaScriptの数字型の特徴：

- 正負の数値や小数も統一して扱える。
- 小数点以下15桁以上は四捨五入されます。

※サンプル：sample05.html

【 文法チェック 】

◇Strictモード

```
"use strict";
```

strict 比較項目	非 strict	strict
宣言されていない変数への代入	グローバル変数が作成される	error
書込み不可の変数への代入 書き込み不可プロパティへの代入	何も起きない	error
削除不可のプロパティの削除	何も起きない	error
プロパティ名の重複	最後に宣言したものがプロパティの値となる	error
関数の引数名の重複	重複した引数がそれより前にある同名の引数を隠す。	error
8 進数標記	8 進数として処理される	error

strict モードでは、いくつかの識別子を予約語にします。その対象は implements、interface、let、package、private、protected、public、static、yieldです。strict モードでは、これらを変数や引数の名前として使用できません。

スクリプト全体への適用
スクリプト全体で strict モードを呼び出すには、他のいかなる文よりも前に “use strict”; という文をそのまま追加します。また関数だけの指定も可能で関数内部最初に記述します。

【 演算子（*operator*）】

「＋」や「−」などの計算を行うためのものです

＋	足し算、または文字の結合
−	引き算
*	かけ算
/	割り算
%	割った余り

```
<script>
    var sum1 = 1 + 9;
    var sum2 = 1 - 5;
    var sum3 = 2 * 4;
    var sum4 = 10 / 2;
    var sum5 = 10 % 3;
</script>
```

【 変数として使用できない文字 】

他の構文や将来的に使われる言語を変数として使用することができない

予約語とキーワード

“ MDN javascript 予約語 “で検索！！

https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Reserved_Words

※サンプル：sample07.html

【 配列 （*array*）】

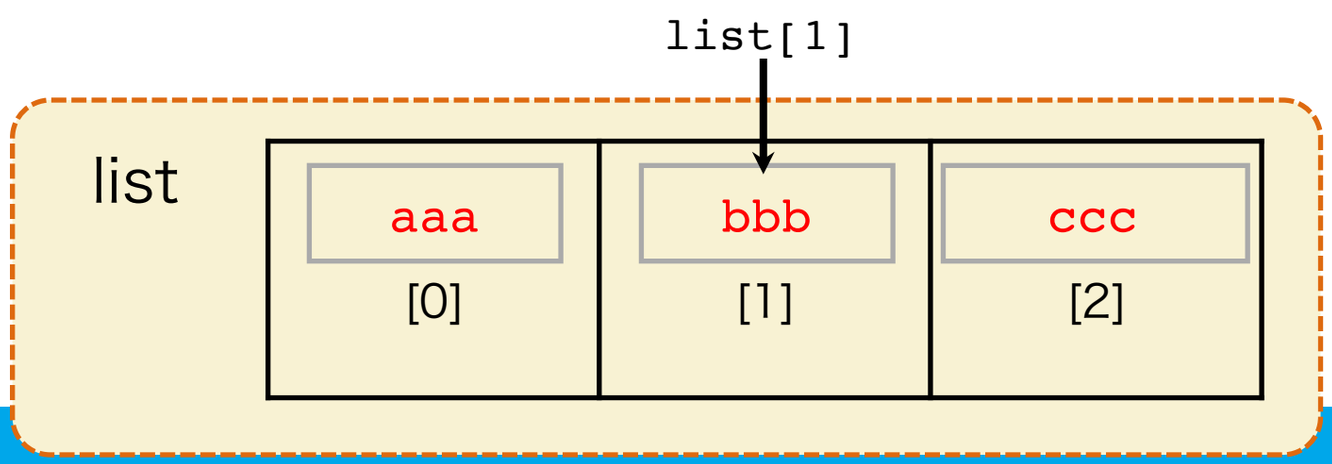
複数の値に順番をつけてまとめて扱う方法です。
配列の順番を識別する番号を「インデックス」と呼びます。
インデックスは「 0 」から始まります。

```
<script>
    var list1 = ['aaa','bbb','ccc'];
    var list2 = new Array('aaa',' bbb',' ccc');
</script>
```

※サンプル：sample_003.html

【 配列のアクセスイメージ （*array*）】

インデックス「0」から値が格納されます。
list[1]では「css」が取得可能



条件分岐

【 **条件分岐**（Conditional branching） 】

プログラム中で、ある条件が満たされているかどうかよって、
次に実行するコードを切り替える命令

- **if文**

  結果が真か偽かによって、それぞれ対応するコードを切り替える

- **switch文**

  条件の値に対応する文を実行する

  http://ja.wikipedia.org/wiki/Switch%E6%96%87

※サンプル：if.html

【 IF分岐 処理 】

if文を使用することで、条件によって違う処理をおこなうことができる

```
if(条件式){  //条件に合えばtrueでここの{…}内を処理
    条件式の結果がtrueの場合実行されるスクリプト
}
```

```
if(条件式){ //条件に合えばtrueでここの{…}内を処理
    条件式の結果がtrueの場合実行されるスクリプト
}else{
    条件式の結果がfalseの場合実行されるスクリプト
}
```

```
if(条件式1){            //条件に合えばtrueでここの{…}内を処理
    条件式の結果がtrueの場合実行されるスクリプト
}else if(条件式2){   //条件に合えばtrueでここの{…}内を処理
    条件式の結果がtrueの場合実行されるスクリプト
}
```

javascriptの基礎

```
//複数条件の分岐処理も可能！
 if(条件式1){
        条件式の結果がtrueの場合実行されるスクリプト
    }else if(条件式2){
        条件式の結果がtrueの場合実行されるスクリプト
   }else if(条件式3){
        条件式の結果がtrueの場合実行されるスクリプト
   }
//複数条件の分岐処理と「条件以外」の指定も可能！
if(条件式1){
        条件式の結果がtrueの場合実行されるスクリプト
    }else if(条件式2){
        条件式の結果がtrueの場合実行されるスクリプト
   }else if(条件式3){
        条件式の結果がtrueの場合実行されるスクリプト
   }else{
        条件式の結果がfalseの場合実行されるスクリプト
   }
```

※サンプル：if_exam.html

【 IF分岐 ：比較演算子】if文の条件式には「比較演算子」を使用します

```
var value = 10;
if(value == 10){  //Q.条件は合います？
    条件式の結果がtrueの場合実行されるスクリプト
}

if(value > 0) {  //Q.条件は合います？
    条件式の結果がtrueの場合実行されるスクリプト
}else{
        条件式の結果がfalseの場合実行されるスクリプト
}
```

【 比較演算子（relation）**】**２つの値を比べる際に使用します。

==	等しければ　true
!=	等しくなければ　true
>	左側の方が大きければ　true
<	左側の方が小さければ　true
>=	左側の方が大きい、または同じなら　true
<=	左側の方が小さい、または同じなら　true

※サンプル：if_logical.html

【 **論理演算子**（Logical operator）】

２つの真偽値をさらに比較する際に使用します

&&	AND処理：両方とも同じなら true
||	OR処理：どちらかが trueなら　true
!	NOT演算：trueならfalseへ　falseならtrueへ

```
var value = 10;
if(value >= 5 && value <= 15){  //valueは５～１５の数値であればtrue
    条件式の結果がtrueの場合実行されるスクリプト
}

if(value == 5 || value == 10 ) { //valueは５か10の数値であればtrue
    条件式の結果がtrueの場合実行されるスクリプト
}else{
        条件式の結果がfalseの場合実行されるスクリプト
}
```

※サンプル：switch.html

【 switch 分岐処理 】

switch文を使用することで、条件によって違う処理をおこなうことができる

```
switch(条件式){
        case 値1:
       条件式の結果がtrueの場合実行されるスクリプト
       break;
        case 値2:
       条件式の結果がtrueの場合実行されるスクリプト
       break;
        case 値3:
       条件式の結果がtrueの場合実行されるスクリプト
       break;
    default:
        条件式の結果がtrueの場合実行されるスクリプト
       break;
}
```

POINT:

break; は {...}の処理から抜けることができる

反復処理

【 反復処理（Iterate）】

プログラム中で、ある条件が満たされているかどうかよって、
次に実行するコードを切り替える命令

- **for文**

  条件が真の間だけ処理を続けていくループ文

- **while文**

  条件が真の間だけ処理を続けていくループ文

【 インクリメント・デクリメント（*increment*）】

演算子の短縮したようのなものです。

i++	i = i+1	インクリメント
i--	i = i - 1	デクリメント

※サンプル：sample10.html

※サンプル：sample11.html

【 for 処理 】

for文を使用することで、反復処理をおこなうことができる

```
for( 条件式 ){
    条件式の結果がtrueの場合実行されるスクリプト
}
```

例）for文

```
for( var i=0; i<10; i++ ){
    console.log( i );  // “i”に代入されてる値を表示
}
```

練習：for文

```
for( var i=0; i<10; i++ ){
    console.log( i );  // “i”に代入されてる値を表示
    if(i==5){
        break;
    }
}
```

※サンプル：sample12.html

【 for in 処理 】

for in文を使用することで反復処理をおこなうことができる

```
var obj = {オブジェクトプロパティ};
for( 変数名 in オブジェクト){
    条件式の結果がtrueの場合実行されるスクリプト
 }
```

例）for in文

```
var obj = {a: 1, b: 2, c: 3, d: 4};
for( val in obj){
    console.log( val );        //プロパティ名
    console.log( obj[val] ); //プロパティ値
}
```

※サンプル：sample13.html

【 while 処理 】

while文を使用することで反復処理をおこなうことができる

```
while( 条件式 ){
    条件式の結果がtrueの場合実行されるスクリプト
}
```

例）While文

```
var i = 0;
while ( i < 3 ) {
    i++;
    console.log(i);
}
```

練習：While文

```
var i = 0;
while ( i < 3 ) {
    i++;
    console.log(i);
    break;
}
```

Function 関数

関数 (function)

※サンプル：function01.html

【 関数定義】

関数とは、一つの処理（再利用可能な処理など）をまとめたもので、
関数名を呼ぶだけで、関数内に記述した処理を実行することができます。

```
<script>
    //ブラウザに読み込まれたら処理開始
     var str = “関数をしらない。";
     alert( str );
</script>
```

```
<script>
  //関数定義(読み込まれても実行しない)
  function test(){
      var str = "関数を知った。";
      alert( str );
  }
   //関数実行（関数名を呼んで実行開始！）
 test();
</script>
```

※関数内外で同名の変数があっても違う変数です。

関数 (function)

※サンプル：function02.html

【 引数（Argument）と戻り値（Return value） 】

関数を呼ぶときに、「引数」を使って値を渡す事できます。
※関数内の変数は外部から参照できません。

```
<script>
//関数定義
function addition(a, b) {
    var total = a + b;
    return total; //return は関数外に値を渡す
}

//引数と戻り値（t=戻り値）
var t = addition(10, 20);
console.log( t );

</script>
```

引数のPOINT：
・引数は少ないほうがいいです。
・関数内の変数は外部から参照できません。

※サンプル：function03.html

【 変数のスコープ 】

スコープ範囲にも２つの種類があります
・グローバル変数： 全ての領域から参照が可能
・ローカル変数： 関数内だけや特定の場所からしか参照ができない

```
<script>
var global = 1;
function test(){
    var local = 2;
    console.log("確認_3："+local);
    console.log("確認_4："+global);
}
test();  //関数実行

console.log“確認_1："+global);
console.log("確認_2："+local); //エラーになる
</script>
```

※関数内でvar を使って変数宣言しないと外部の変数を参照します。

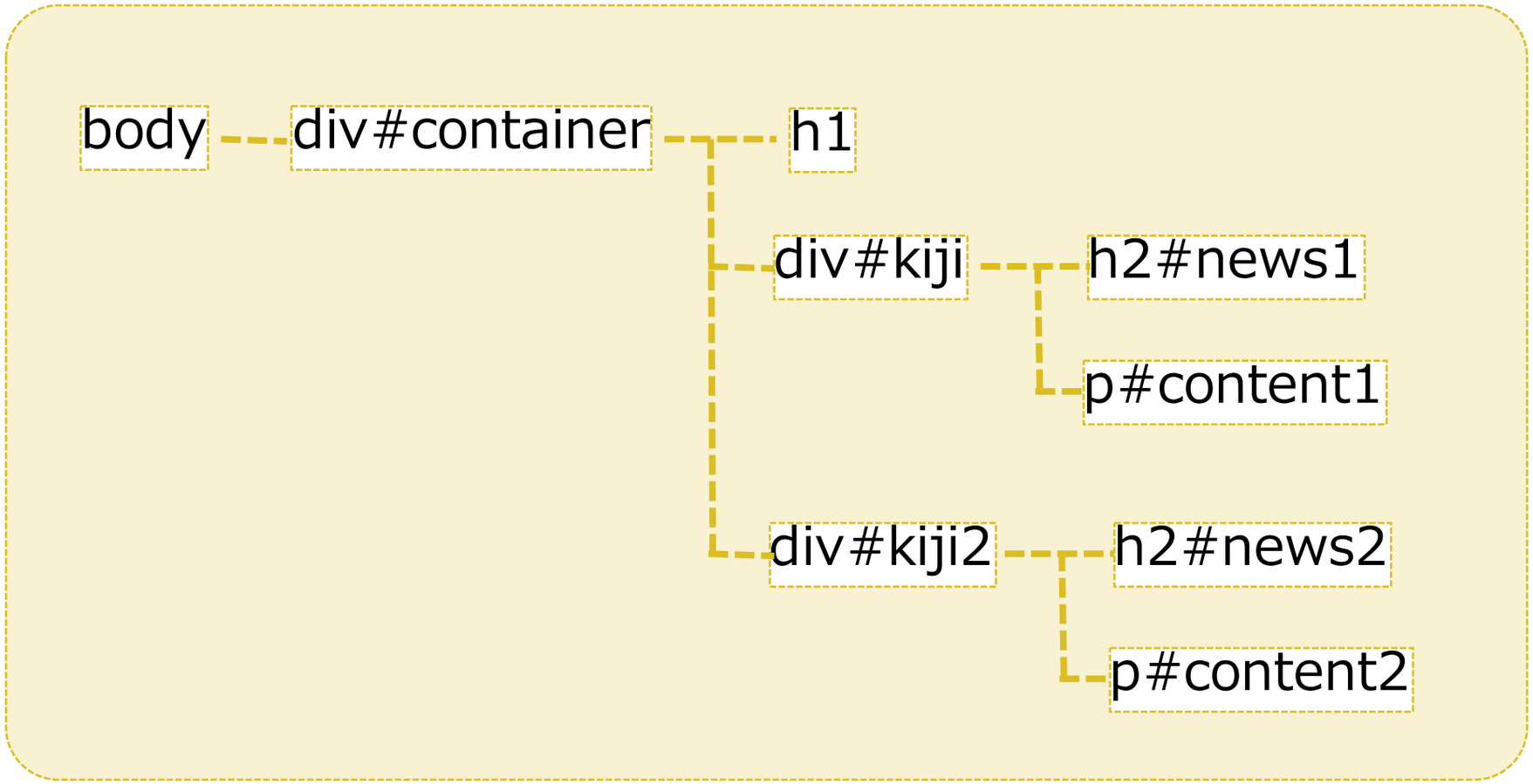
DOM

DOM

【DOM (Document Object Model) 】

DOM は、「Document Object Model」の略です。
プログラムから「HTML 文書」を利用するための、標準化された仕様です。

HTML 文書がブラウザに読み込まれると、ブラウザ内に部品１つ１つが「DOM オブジェクト」として自動的に生成されます。



※次のページでChrome開発者ツールでDOMを検証

ja.wikipedia.org/wiki/Document_Object_Model

ウィキペディア
フリー百科事典

メインページ
コミュニティ・ポータル
最近の出来事

ページ　ノート　閱覧　編集

Document Object Model

Document Object Model (DOM) は、W3Cから勧告されてい
書やXML文書をアプリケーションから利用するためのAPIである
ら Level 3 まで勧告されている。

XMLを読み込む別のAPIであるSAXと異なり、XMLデータをツ!

Elements　Network　Sources　Timeline　Profiles　Resources　Audits　Console　NetBeans

```
<!DOCTYPE html>
<html lang="ja" dir="ltr" class="client-js ve-not-available">
▶<head>…</head>
▼<body class="mediawiki ltr sitedir-ltr ns-0 ns-subject page-Document_Object_Model
skin-vector action-view">
    <div id="mw-page-base" class="noprint"></div>
    <div id="mw-head-base" class="noprint"></div>
  ▼<div id="content" class="mw-body" role="main">
      <a id="top"></a>
    ▶<div id="siteNotice">…</div>
      <div class="mw-indicators">
      </div>
      <h1 id="firstHeading" class="firstHeading" lang="ja">Document Object Model</h1>
    ▼<div id="bodyContent" class="mw-body-content">
        <div id="siteSub">出典: フリー百科事典『ウィキペディア（Wikipedia）』</div>
        <div id="contentSub"></div>
      ▶<div id="jump-to-nav" class="mw-jump">…</div>
      ▼<div id="mw-content-text" lang="ja" dir="ltr" class="mw-content-ltr">
        ▶<table class="vertical-navbox nowraplinks" style="float:right;clear:right;
```

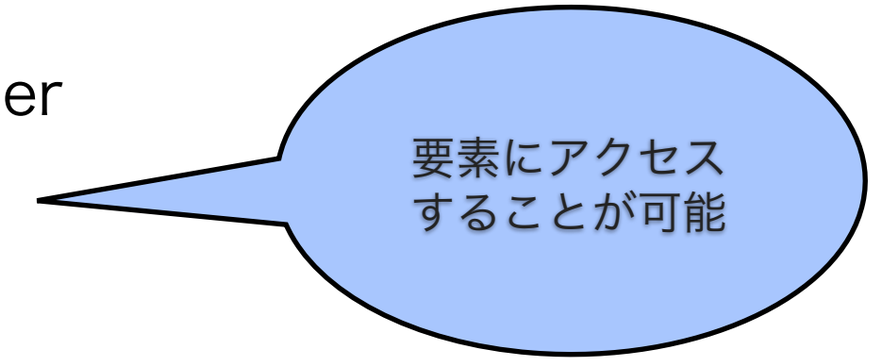
G's ACADEMY TOKYO

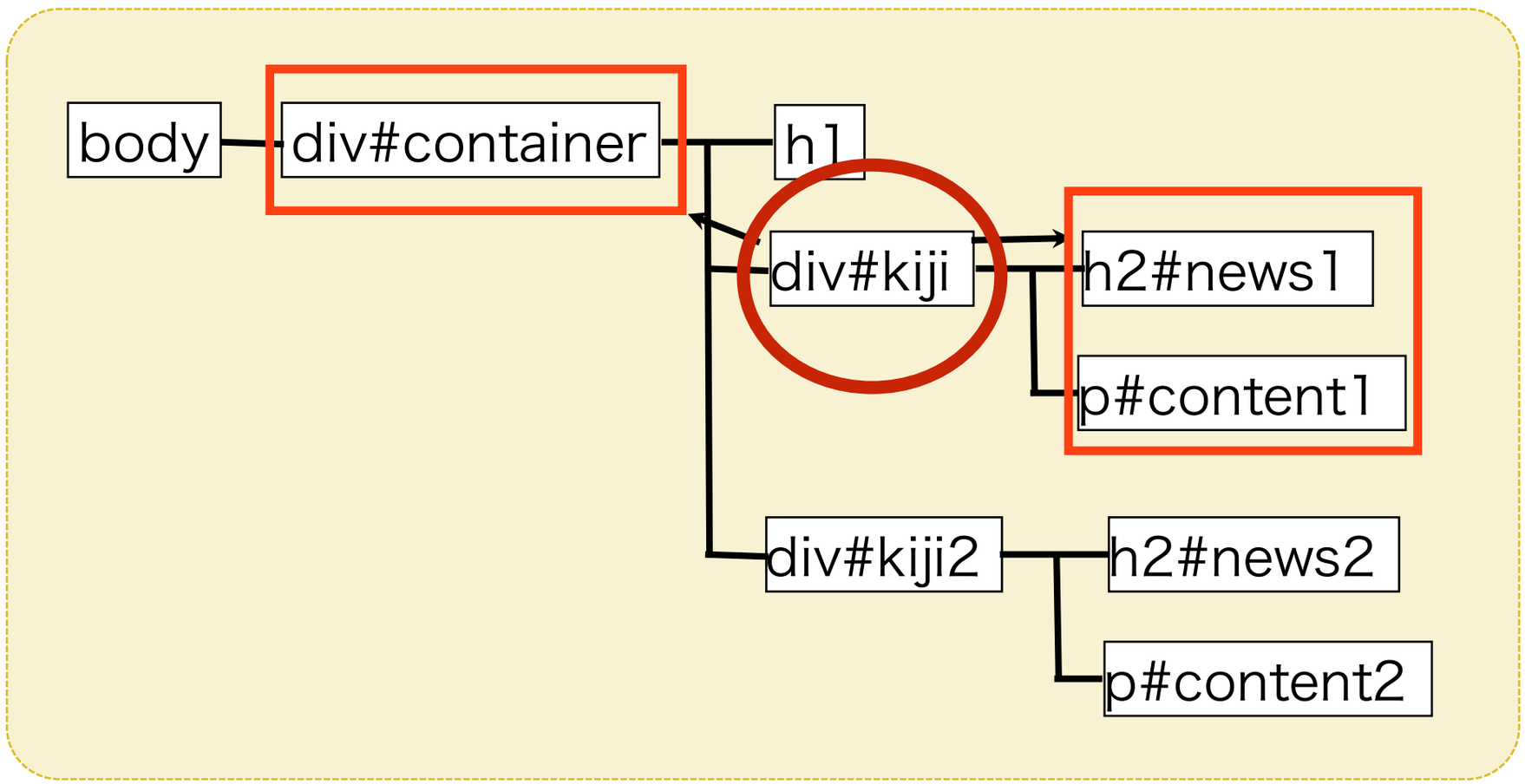
DOM

【親と子の関係(セレクタ)】

javascriptでは親要素と子要素などの関連したデータを操作することが可能です

kijiの親：div#container
kijiの子：h2#news1
　　p#content1





※サンプル：dom01.html

【 Web API インターフェース： document.*】

id属性の値を参照し、要素を抽出します。（id指定のため、戻りは1要素のみ）

```
取得された要素 = document.getElementById(id名);
```

HTMLのid指定は１つと決められています。同一の ID が複数与えられている場合の挙動(HTML 文書としてはエラーになりますが)は定義されてない

要素名を参照し、要素を抽出します。（複数の場合あり）

```
取得された要素 = document.getElementsByTagName(“要素名");
取得された要素 = document.getElementsByClassName (“要素名");
```

例)testのidを取得することができ、参照や変更が可能

```
<div id=”test”>id属性を持つdiv要素</div>
```

```
var test = document.getElementById(‘test’);
```

変数からスタイルシートを指定するこも可能

```
stest.style.color="#f00";
stest.style.fontSize = "40px";
```

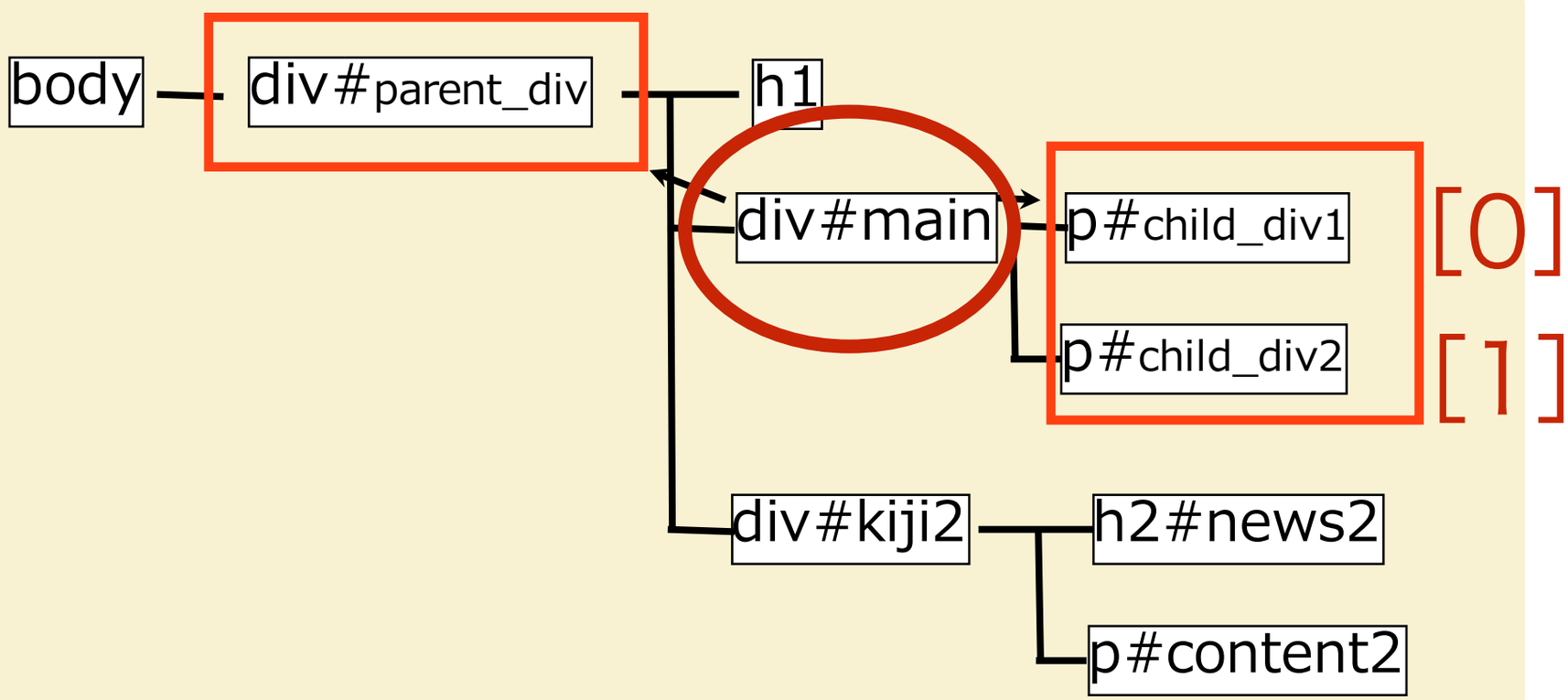
DOM

※サンプル：dom02.html

【 親と子を取得 】

```
Var target= document.getElementById('main');
var parent= target.parentElement;
var child = target.children;  //配列で取得
```

parent:div要素を取得
child:配列で各要素を取得



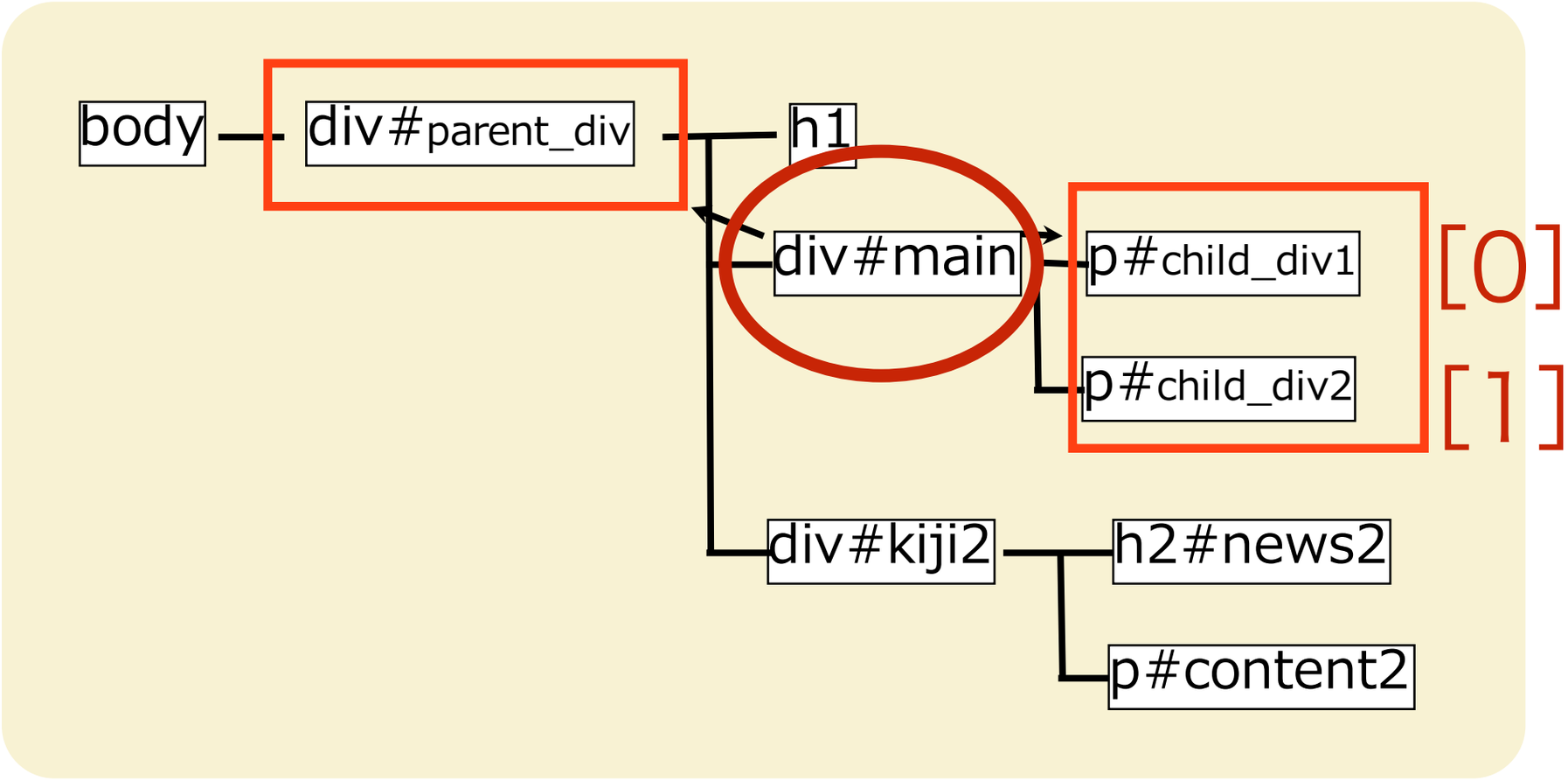
DOM

※サンプル：dom03.html

【親と子の関係を理解する 】

要素の追加や削除が簡単にできるようになる

targetに要素を追加： insertBefore(最初),appendChild（末尾）
targetの要素を削除： removeChild



Selectors API

Selectors API

※サンプル：selectors.html

【 Selectors API 】

class が left-columnの要素を配列として全て取得
document.querySelectorAll(“.left-column”**)**;

class が . left-columnもしくは . right-columnの要素を配列として全て取得
document.querySelectorAll(“.left-column, .right-column”**)**;

id が main_contentの要素を取得
document.querySelector(“#main_content”**)**;

div 直下の p を取得
document.querySelector(“div>p”**)**;

Input要素type属性buttonの要素を取得
document.querySelector("input[type=’button]"**)**;

class が left-columnの最初の要素を取得
document.querySelector(“.left-column”**)**;

Event

Evnet

※サンプル：event01.html

【 イベント 】

イベントは、ページの読み込み、マウスボタンをクリックや、マウスオーバーなどの「 何かしらの動作（**イベント**）が起こった時」に発生します。

入力...　ボタン

【 イベントハンドラ 】

イベントハンドラは、イベントを検出し、イベント発生のタイミングで必要な処理（関数など含む）を実行します。以下がよく使われるイベントハンドラの一例。

onLoad	ページや画像の読み込みが完了した時に発生
onClick	要素やリンクをクリックした時に発生
onChange	フォーム要素の選択、入力内容が変更された時に発生

記述例）

```
var mybtn= document.querySelector(‘#mybutton’);
mybtn.onclick = function(event){
    alert('Hello world');
};
```

◇HTML要素の属性を使った Javascript Evnet（非推奨）

```
<button onclick="alert('Hello world!')”>
```

【 イベントハンドラ：一覧 】

onBlur	ページやフォーム要素からフォーカスが外れた時に発生
onFocus	ページやフォーム要素にフォーカスが当たった時に発生
onChange	フォーム要素の選択、入力内容が変更された時に発生
onSelect	テキストが選択された時に発生
onSubmit	フォームを送信しようとした時に発生
onReset	フォームがリセットされた時に発生
onAbort	画像の読み込みを中断した時に発生
onError	画像の読み込み中にエラーが発生した時に発生
onLoad	ページや画像の読み込みが完了した時に発生
onUnload	ウィンドウを閉じた時、他のページに切り替えた時に発生
onClick	要素やリンクをクリックした時に発生
onDblClick	要素をダブルクリックした時に発生
onKeyUp	押していたキーをあげた時に発生
onKeyDown	キーを押した時に発生
onKeyPress	キーを押してる時に発生
onMouseOut	マウスが離れたした時に発生
onMouseOver	マウス乗った時に発生
onMouseUp	クリックしたマウスを上げた時に発生
onMouseDown	マウスでクリックした時に発生
onMouseMove	マウスを動かしている時に発生

Event（DOMレベル0）

◇イベントハンドラ：その１

```
//特定の要素を取得
var mybtn= document.querySelector('#mybutton');
```

//イベントハンドラと関数（処理）を一括記述する方法　※サンプル：event01.html

```
mybtn.onclick = function(event){
    alert('Hello world');
};
```

//イベントハンドラと関数を分ける方法　※サンプル：event02.html

```
function alertMsg (evt) {
    alert(evt);
}
mybtn.onclick = alertMsg;
```

//イベントハンドラ削除　※サンプル：event03.html

```
mybtn.onclick = null;
```

※オブジェクトの1つのイベントに対して、1つのハンドラしか登録できません。
　同一のイベントに後から登録すると、先の登録が上書きされます。

Event（DOMレベル２）

※サンプル：event04.html

◇イベントハンドラ：その２

```
var mybtn= document.querySelector('#mybutton');

//イベントハンドラと関数を分ける「イベントリスナー」という方法
mybtn.addEventListener("click", alertMsg, false);
mybtn.addEventListener("dblclick", alertMsg, false);

//イベントハンドラと関数を分ける方法
function alertMsg (evt) {
    alert(evt);
}
```

```
//イベントハンドラ削除
mybtn. removeEventListener("dblclick", 関数名, false);
```

※第三引数：useCaptureがtrueの場合はそのイベントを最優先で処理の意味
※addEventListenerを使用すると複数のイベントが登録可能というメリットがある
※イベントリスナの一削除はゲームコンテンツなど処理が多い場合には欠かせません。

Event（thisとは？）

※サンプル：this.html

javascriptを利用してるとthisというキーワードをよく見かけます。
このthisはどのようなときに使うのでしょうか？

以下１例ですが、

```
window.onload = function() {
    //関数定義
    var mybtn = document.querySelector("#mybutton");
    mybtn.onclick = function () {
        console.log("id: " + this.id);
        console.log("name: " + this.name);
        console.log("class: " +this.className);
    }
};
//以下HTMLコード
<button id="mybutton" name="test_btn" class="f1">クリックしてコンソールで確認</button>
```

イベント＆関数で使用した場合、イベント発生箇所がthisとして取り扱われます。
※thisは利用方法によって変わります。コード内での処理で関数が実行された場合はグローバルオブジェクトとして解釈されたりするので、場合によることを知る必要があります。

Dateオブジェクト

Dateオブジェクト

※サンプル：date.html

【 Dateオブジェクト (日付)】

DateオブジェクトはJavaScriptにあらかじめ用意されてるオブジェクトです。

```
var now  = new Date();
```

【 主要メソッド・プロパティ (よく使用されるものを抜粋)】

```
var year    = now.getFullYear();
var month = now.getMonth()+1;  //+１を入れる。
var date    = now.getDate();
var day     = now.getDay();
var h       = now.getHours();
var m       = now.getMinutes();
var s       = now.getSeconds();

console.log( year+"-"+ month+"-"+date+"  "+h+":"+m+":"+s );
```

console.logなどで表示する場合に+を使って変数と文字列を組み合わせて表示します。
※文字列を組み合わせず“+”でつなげると全て数字なので、計算処理をおこなってしまいます。そこは気をつけるポイントです。

Windowオブジェクト

Windowオブジェクト

※サンプル：window.html

【 とは？】

windowオブジェクトはJavaScriptにあらかじめ用意されてるオブジェクトです。
JavaScriptであらかじめ用意されている関数/オブジェクトは、全てwindowオブジェクトのプロパティです。※windowオブジェクトからグローバル変数にもアクセス可能

```
window.location.href(‘URL’);
window.document.getElementById(’id名');
Window.alert(window.グローバル変数名);
```

【 windowオブジェクトの省略 】

windowオブジェクトはJavaScriptでは特殊な扱いになっており、windowオブジェクトのプロパティにアクセスするときにwindowを省略出来ます。

```
window.alert(”ハロー！”);  →  alert(”ハロー！”)
```

【 主要メソッド・プロパティ（多数があるので抜粋） 】

```
window.open("http://hakuhin.jp/");
window.close();    //Windowを閉じる
window.closed;    //Windowが閉じたか？
window.screenX; //ウインドウの横位置取得
window.screenY; //ウインドウの縦位置取得
window.moveBy(横座標,縦座標);  //ウインドウ移動
window.scrollTo(水平方向、垂直方向);  //スクロール位置
```

例外処理

例外処理

※サンプル：try.html

【 try… catch… finally 基本】

```
//一般的
try {
	alert(b);
} catch(e) {
	alert("エラー！ : " + e.message);
}

//finallyあり
try {
	 alert(b);
} catch(e) {
	alert("エラー！ : " + e.message);
} finally {
	alert("おしまい"); //ここは処理が実行される
}
```

例外処理はエラー処理の一つとして使用します。

例外処理２

※サンプル：try2.html

【例外処理 応用】

```
//一般的（メイン処理）
try {
	var str=“”;
	stringLength(str); //ローカル関数処理
} catch(e) {
	//ローカル関数で発生したエラーを取得！※1
	alert(“エラーが発生：" + e.message);
} finally {
	alert("必ず実行！"); //ここは処理が実行される
}
//関数：共通パーツとして処理
function stringLength(str) {
	var len = str.length;
	if(len==0 || len==NULL) {
				throw new Error(“エラー！”); //※1：自らエラーを投げる場合
	}
	return len;
}
```

throw new Error (“文字列”); を関数内で使用すると効果的なエラー処理が実装できる

Random

random(乱数)

※サンプル：random.html

【 random(乱数)】

random関数はJavaScriptにあらかじめ用意されてるMathオブジェクトの１関数です。
MathオブジェクトのMath.floor（切り捨て関数）と組み合わせて使います。
結果として●●の箇所に数字を入れると「１～●●」の間の数が取得できます

```
var num = Math.floor( Math.random() * ●● + 1);
```

【 例 】

```
var num = Math.floor( Math.random() * 5 + 1);

console.log( num );
```

ポイントは、
サンプルの使い方をそのまま使っていくことです。「ランダムな数字を作りたいなー」と思ったら、上記のサンプルをそのまま使いましょう！！

そして、調べる時間を短縮し、作成することに時間をかけましょう！

【演習】おみくじアプリ作成

◇仕様：

- サンプル" omikuji_tpl.html "を使用して作成
- “おみくじボタン”クリック後に「大吉・中吉・小吉・吉・凶」の５種類をランダムでHTMLに表示する
  表示記述例）
  document.querySelector("#id名").innerHTML="大吉";
- 「大吉・中吉・小吉・吉・凶」はIF文の中に記述します

第2章

HTML5 API

ブラウザ対応状況

ブラウザのサポート状況を確認

http://caniuse.com/

About | News | February 21, 2015 - Latest support data upd... | Compare browsers | Index

AdChoices ▶ HTML5 CSS3 ▶ HTML5 Canvas SVG ▶ Browser Testing ▶ JavaScript Canvas

Can I use ? Settings

Detected your country as "Japan". Would you like to import usage data for that country?

Import No thanks

CSS

- ::selection CSS pseudo-element
- :placeholder-shown CSS pseudo-class
- @font-face Web fonts
- Blending of HTML/SVG elements
- calc() as CSS unit value
- 2.1 selectors
- background-attachment
- background-blend-mode
- background-position edge offsets
- box-decoration-break
- clip-path property
- Counters
- currentColor value

HTML5

- Audio element
- Autofocus attribute
- Canvas (basic support)
- Canvas blend modes
- classList (DOMTokenList)
- Color input type
- contenteditable attribute (basic support)
- Custom Elements
- Custom protocol handling
- Datalist element
- dataset & data-* attributes
- Date and time input types
- Details & Summary elements
- Dialog element
- Download attribute

SVG

- Inline SVG in HTML5
- SVG (basic support)
- SVG effects for HTML
- SVG favicons
- SVG filters
- SVG fonts
- SVG fragment identifiers
- SVG in CSS backgrounds
- SVG in HTML img element
- SVG SMIL animation
- **All SVG features**

JS API

- Ambient Light API

Canvas(2d)

canvasの基礎

【canvas要素を知ろう】

<canvas>要素の属性は主に３つ！

- ID
- width
- height

記述例：

```
<canvas  id="＊＊"  width="1024"  height="764"></canvas>
```

==========

Id=”＊＊” → ユニークID名を記述

width = ”横幅を記述”

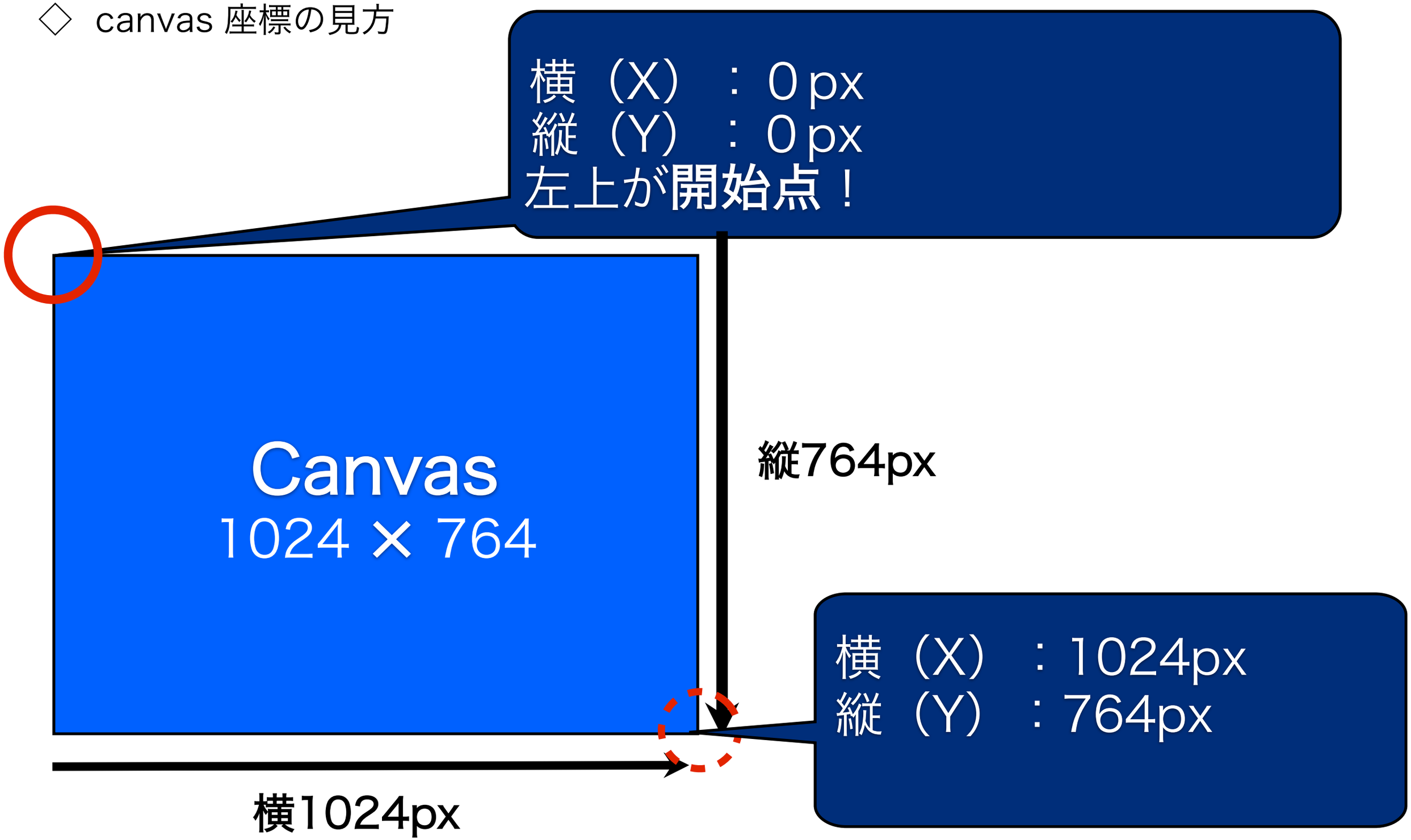
height = " 縦幅（高さ）を記述 ”

==========

◇注意：サイズ属性のサイズ指定はstyle属性では使用できません！

canvasの基礎

◇ canvas 座標の見方



canvasの基礎

【 canvasで描画準備 】

canvas 要素は描画機能にアクセスするための ”getContext” と呼ばれるDOMメソッドを持っています。
” getContext ” は描画コンテキストという1つの引数しか取ることができません。現在利用可能なコンテキストは ” 2d ”コンテキストの1つです。
※今後は２D以外に３Dも利用できる日が来るかもしれません。

ーーー

canvasの基礎

【 canvasで文字を描画 】

サンプル
canvas_font.html

取得したコンテキストで文字を描画をおこないます

```
//カラー指定
ctx.fillStyle     = "#ff0000";
//fontサイズ、書式
ctx.font          = "20px _sans";
//文字の設置位置
ctx.textBaseline = "top";
//表示文字と座標
ctx.fillText("Hello World!", 100, 80 );
```

ctx.textBaseline:文字の設置するベースライン
http://www.html5.jp/canvas/ref/property/textBaseline.html

canvasの基礎

【 canvasで矩形を描画 】

サンプル
canvas_sikaku.html

サンプル
canvas_sankaku.html

取得したコンテキストで矩形を描画します

```
ctx.rect(0,0,60,60);
ctx.stroke();
```

指示形態は以下の通りです
Rect(座標、幅と高さ)

◆塗りつぶす場合

```
ctx.fillStyle = "#000";
ctx.rect(10,10,40,40);
ctx.fill();
```

塗りつぶす場合は**fill**を使用します。

必ず色をつける場合は「**fill**」と指定してください

canvasの基礎

【 canvasで円を描画 】

サンプル
canvas_arc.html

◇取得したコンテキストで円を描画します

```
ctx.arc(100, 100, 50, 0, Math.PI*2, false);
```

◇指示形態は以下の通りです
arc(座標、半径、円のスタート度、エンド度(描画)、回転)

◇塗りつぶす場合

```
ctx.fillStyle = "#000";
ctx.arc(100, 100, 50, 0, Math.PI*2, false);
ctx.fill();
```

塗りつぶす場合は**fill**を使用します。

必ず色をつける場合は「**fill**」と指定してください

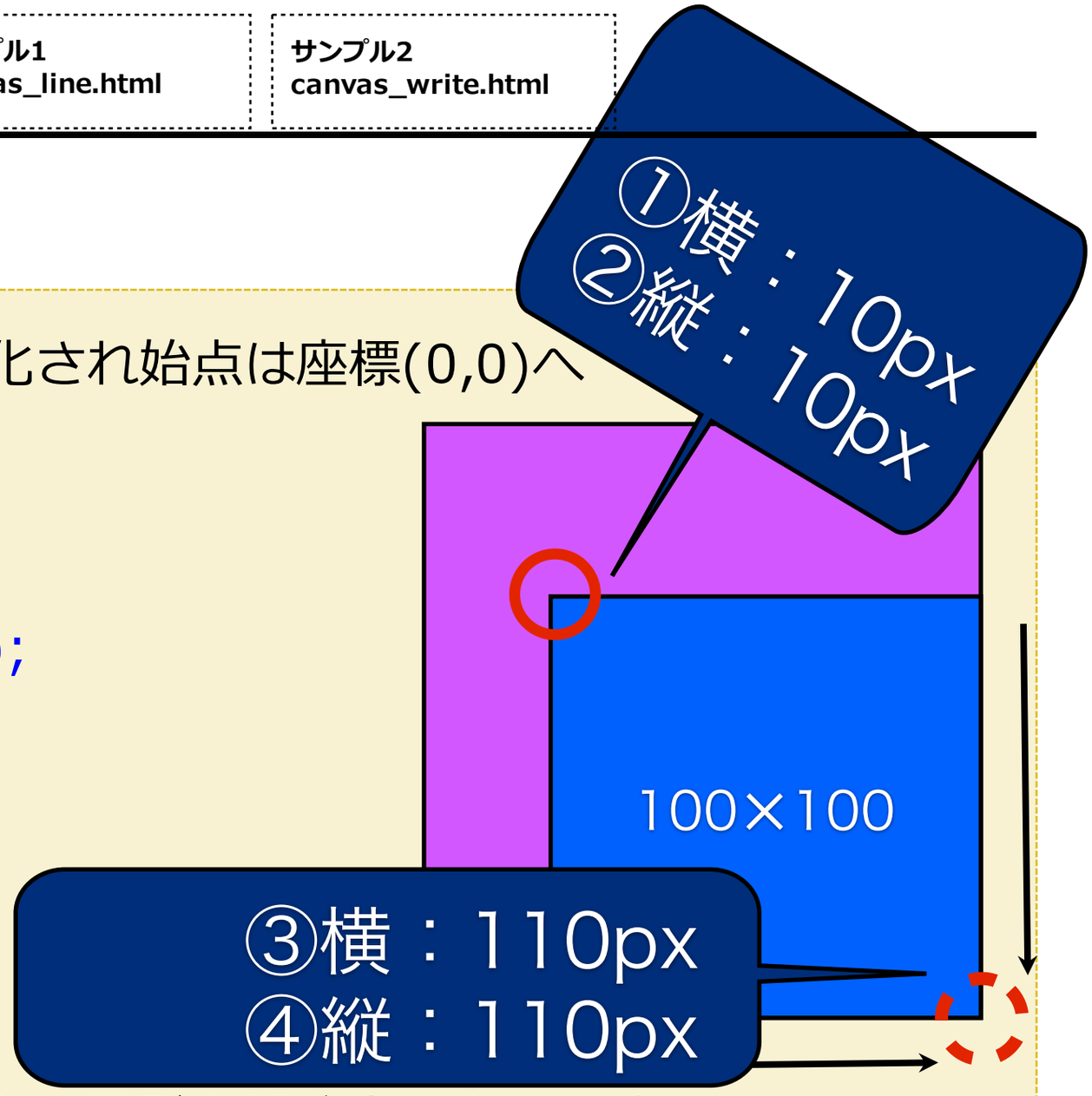
canvasの基礎

サンプル1
canvas_line.html

サンプル2
canvas_write.html

【 canvasで線を描画 】

```
//context. beginPath()が呼ばれると初期化され始点は座標(0,0)へ
context.beginPath();
// x と y は新しい始点の座標となります。
context.moveTo(①開始横軸, ②開始縦軸);
// x と y は”線の終点の座標”となります。
context.lineTo(③終了横軸, ④終了縦軸);
//線を描く
context.stroke();
//現在の点から始点に向けて直線を描くことで図形を閉じようとします。
※もし図形がすでに閉じられていれば関数は何もしません。
context.closePath();
```



SVG

SVG

SVGは「Scalable Vector Graphics」の略です。Scalableは「形を保ったまま拡大・縮小ができる」といった意味で、PNGのような画像フォーマットの1つです。HTMLがタグによって文書の構造を記述するのと同様に、SVGはタグで図形を記述します。※文字コードはUTF-8、拡張子は.svgとしてください。

◇現在SVGに対応しているブラウザーのバージョンです。

ブラウザー覧	対応バージョン
Internet Explorer	9以降
Firefox	3以降
Google Chrome	4以降
Safari	3.2以降
Opera	9以降

◇SVGの描画要素 一覧

プロパティ	説明
四角形	rect
円	circle
楕円	elipse
直線	line
連続直線	polyline
多角形	polygon
テキスト	text

SVG

【 SVGの記述方法 】

◇HTMLに直接埋め込む　サンプル： svg.html

```
<svg height="1000px" width="1000px">
    <rect id="myRect" height="100px" width="100px" fill="blue"/>
</svg>
```

◇imgタグやCSSを用いる　サンプル： svg_css.html

```
.box{
  background-image: url(sample_svg.svg);
}
```

◇objectタグを用いる　サンプル： svg_object.html

```
<object type="image/svg+xml" data="sample_svg.svg"></object>
```

◇embedタグを用いる　サンプル： svg_embed.html

```
<embed src="sample_svg.svg" type="image/svg+xml" />
```

video

Video

サンプル
video/video.html

【 videoタグ基礎：コーデック】

◆コーデックの種類
Webブラウザで使用できるコーデックにはいくつか種類があり、以下がその一覧です。

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

- *H.264（mp4）：　Safari, (Chrome, IE9)*
- *ogv：　Chrome, Firefox, Opera*
- *webm：　Chrome, Opera, (Firefox)*

ーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーーー

ブラウザごとに違うため、３つのムービーを準備する必要があります。
※唯一ここが面倒です。
（ogvとmp4、２つでもだいたいはカバーできます。）

※最新のブラウザの対応についてこちらを参照ください
http://caniuse.com/#search=video

Video

サンプル
video/video.html

【 videoタグで映像を再生 】

videoタグには再生できる拡張子が違うので、それぞれ用意しておく必要があります

記述例）

```
<video id="video" width="640" height="360" controls poster="video.png">
  <source src="small.mp4" type="video/mp4">
  <source src="small.ogv" type="video/ogg">
  <source src="small.webm" type="video/webm">
  <p>※ご利用のブラウザでは再生することができません。</p>
</video>
```

POINT:
controls は必須です。これがないとコントローラーが表示されません。
poster=“*****.png” は最初に表示される静止画です。

Video

◇Javascriptで制御

HTML要素

```
<div>
    <button id="video_play">Play</button>
    <button id="video_pouse">Play</button>
</div>
```

Javascript処理

```
var cont = document.getElementById("video")
//再生開始
function videoPlay(){
    cont.play();
}
//再生を停止
function videoPause(){
    cont.pause();
}
 //イベントハンドラ
 document.getElementById("video_play").onclick = videoPlay;
document.getElementById("video_pouse").onclick = videoPause;
```

audio

audio

サンプル
audio/audio.html

【 audioタグ基礎：コーデック】

◇コーデックの種類
Webブラウザで使用できるコーデックにはいくつか種類があり、以下がその一覧です。

ーーー

audio

サンプル
audio/audio.html

◇audio要素

<audio>タグは、音声を再生する際に使用します。 再生する音声ファイルは、src属性で指定します。

<audio>〜</audio>の中には、 <audio>タグがサポートされていない環境で表示させるメッセージを記述することができます。

◇HTMLソース

```
<audio id="audio" controls autoplay loop>
    <source src="audio/sample.mp3" type="audio/mp3">
    <source src="audio/sample.wav" type="audio/wav">
    <p>※ご利用のブラウザでは再生することができません。</p>
</audio>
```

◇属性

属性	説明
src	音声ファイルを指定する
preload	音声をあらかじめ読み込む（auto、metadata、none）
autoplay	音声を自動再生する
loop	ループ再生を指定する
controls	インターフェスを表示する

audio

◇Javascriptで制御

HTML要素

```
<div>
    <button id=”audio_play">再生</button>
    <button id=”audio_pouse">止める</button>
</div>
```

Javascript処理

```
var cont = document.getElementById("audio");
//再生開始
function audioPlay(){
    cont.play();
}
//再生を停止
function audioPause(){
    cont.pause();
}
 //イベントハンドラ
document.getElementById("audio_play").onclick   = audioPlay();
document.getElementById("audio_pouse").onclick = audioPause();
```

AppCache

AppCache(1)

「アプリケーションキャッシュ：CacheManifestの基本」
アプリケーションキャッシュとは、ご利用のPC（OR スマートデバイス）にWEB上で見たファイルをキャッシュする機能の事です。
通常のブラウザのキャッシュよりも強力で、iPhoneやiPadではよく利用される技術です。

処理の流れ：

１回目HTMLをリクエスト：

初めてアクセスした際にブラウザにキャッシュを保存

２回目HTMLをリクエスト：

”キャッシュファイル”にアクセス、画像や重いファイル等を利用しても快適に閲覧できます。
※回線が切れていてもキャッシュにデータがあれば閲覧できます。

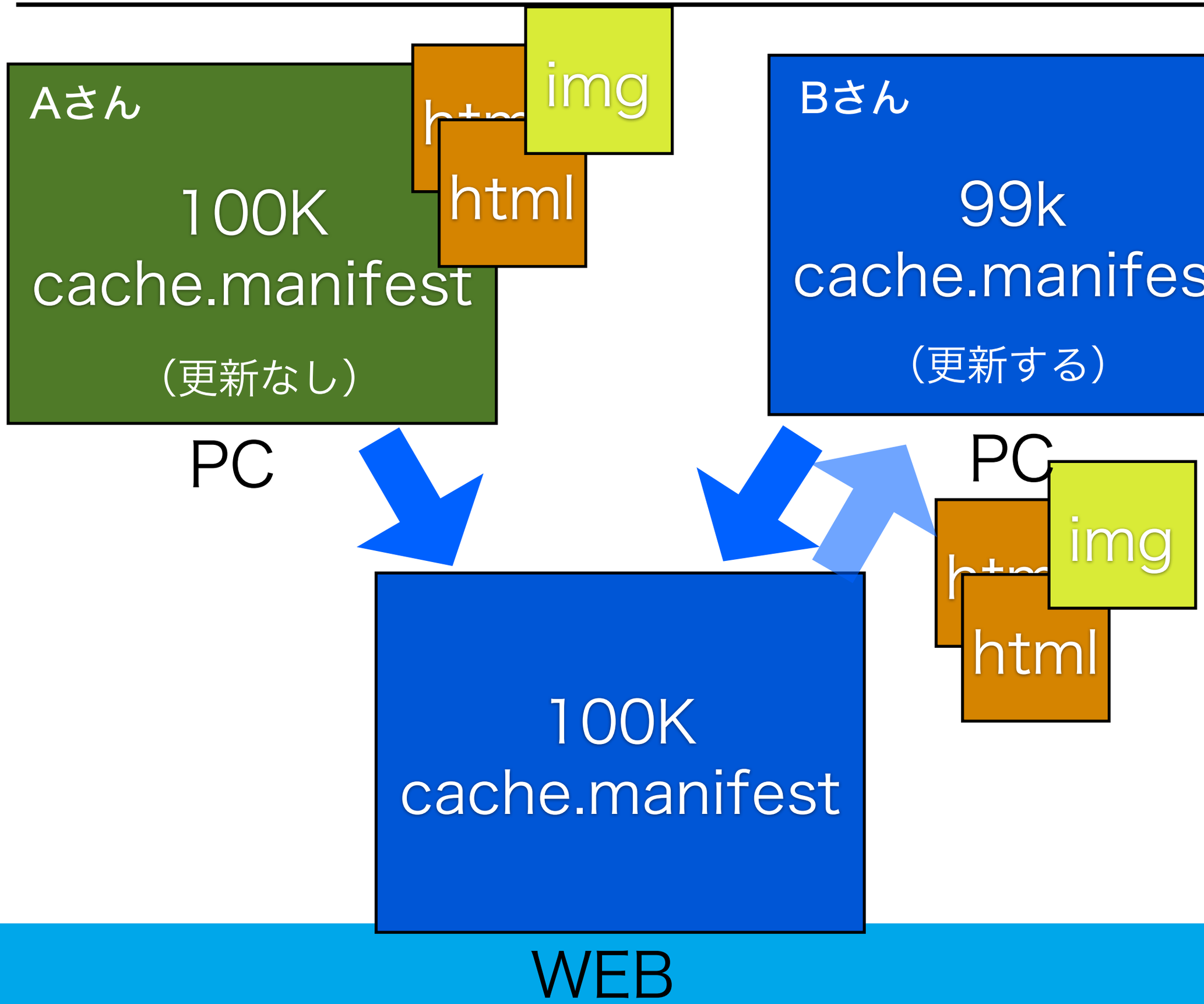
AppCache(2)

少し細かい裏処理内容を説明〜

2回目以降のアクセス：

WEBに接続されていれば、キャッシュ上のCacheManifestファイル
chacheManifestファイルの比較処理が走ります。比較処理では、Web上の
”CacheManifestファイル”とキャッシュ上のCacheManifestファイルのデータ(
ファイルByte数)量が同一であれば、ダウンロードせずキャッシュのHTMLを読
み込みます。”※次ページ概要図を参考”
比較処理でデータ量に変化（同一でない場合）があれば、ファイルが更新されて
ると判断し、WEBから新HTMLをダウンロードし
更新する仕組みで作られています。
HTML5のすばらしい機能の1つといえます。

AppCache(3)　概要図



AppChache（ サンプル１ ）

index.html

```
<!DOCTYPE HTML>
<html manifest="cache.manifest">        ← ここが大事！
```

.htaccess

```
AddType text/cache-manifest .manifest        ← この一行だけ！
```

AppChache （サンプル２）

◇cache.manifest （ファイル記述方法: appCache/cache.manifest）

```
CACHE MANIFEST

# Version:0

#画像などもキャッシュ可能（大きいサイズの画像はキャッシュした方がいい）
CACHE:
css/style.css
l_get.html
l_remove.html
l_set.html
index.html
js/jquery-1.6.4.min.js

#サーバサイドスクリプトなど（キャッシュしないファイル）
NETWORK:
```

は1行コメント
Version:0は必要ない。ファイル更新後にversionに文字を追加することでファイル量変更するために記述している。

History API

History API

◇Hystory APIとは？

History APIはHTML5の機能のひとつ。Historyその言葉どおり「履歴」のことでhistoryという名前のオブジェクトになります。

◇主な機能

ブラウザの「戻る・進む」ボタンのイベントを取得してページ遷移させたり、遷移せずにページの内容を動的に変えることができるものです。

◇history関数

history関数	説明
history.back();	一つ前に戻る
history.forward();	一つ前に進む
history.go(2);	２つ進む
history.go(-2);	２つ戻る
history.length	現在の履歴の数を表示

※Windowオブジェクトは省略してます。

History API 新機能

◇機能概要

ブラウザの履歴の一つ一つの記録を**「スタック」**といい、一つ履歴が増える度にスタックが増えることになります。ページ推移を行わなくても追加できるようになりました。

◇使用関数

//ページ履歴登録
pushState(第1引数:**state**, 第2引数:**title**, 第3引数:**URL**[省略可]);
//履歴情報取得
popstate(第１引数: eventオブジェクト);

```
１．history.pushState(null,‘history-test’,’/test1’); //ページ遷移せずに履歴追加
※上記実行、ブラウザの戻るボタンをクリック

２．history.pushState(null,'historytest',’/test2’); //ページ遷移せずに履歴追加
※上記実行、ブラウザの戻るボタンをクリック

3．URLの変化を監視して、イベントを実行する
window.addEventListener('popstate',function(e){
        console.log(e.state);
},false);
```

※サンプル：pushstate.html

History API 新機能

http://universions.com/tour.html

CREATION

制作以外のことに時間を取られず、
クリエイティブな業務に集中する

Page Visibility API

Page Visibility API

◇Page Visibility API

ページの表示状態を取得するAPIで、表示してる状態では処理実行、非表示状態では処理が走らないように作成することが可能です。他のアプリ、タブを閲覧してる際に、VIDEOや時間が進まないようにすることができるのです。

- ウィンドウが最小化された場合
- 同じウィンドウの他のタブを見ている場合

など、非表示状態を取得することができます。

◇メリット

Web ページが見えていないときに不要なタスクを実行しないようにすることができます。CPUやメモリ等のリソースの節約が可能になります。

◇現在対応しているブラウザーのバージョン

ほとんどのブラウザで使用可能です、IEは10以降より使用可能
スマートフォンはOperaMiniではまだ対応が遅れてるようです。

ブラウザ対応（確認）
http://caniuse.com/#feat=pagevisibility

Page Visibility API

◇Page Visibility API

addEventListener “visibilitychange” のイベントリスナーで登録します。
イベントリスナーで常に「表示・非表示」の状態を監視します。

```
document.addEventListener(“visibilitychange”, handler); //イベントリスター登録

function handler() {

    if (document.visibilityState == "visible") {
        // 可視状態
        console.log(“表示”);
    } else if (document.visibilityState == "hidden") {
        // 不可視状態
        console.log(“非表示”);
    } else if (document.visibilityState == "prerender") {
        // プリレンダリング状態
        console.log(“プリレンダリング状態”);
    }

}
```

※サンプル：pagevisibility.html

Timing control for script-based animations

Timing control for script-based animations

◇スクリプト・ベースのアニメーションのタイミング制御

window オブジェクトの requestAnimationFrame() というメソッドを定義しています。
setTimeout() やsetInterval() とは異なります。
（setTimeout,setInterval は時間という枠で制御するもの）
requestAnimationFrame() はアニメーションの実装専用です。
そのため requestAnimationFrame() は setTimeout() や setInterval() に付随する欠点が何もありません。使い方はとてもシンプルです。

```
function animate(time) {                    // Animation loop
  draw(time);                               // A function that draws the current animation frame
  requestAnimationFrame(animate); // Keep the animation going
};

requestAnimationFrame(animate);    // Start the animation
```

※サンプル： animate.html

WebStorage

WebStorage

シンプルに使える sessionStorage

ウィンドウ/タブ単位でデータを一時的に保存できる。

ウィンドウやタブが開いている間、Webアプリケーション利用に関するデータをそれぞれの単位で保存し、ウィンドウ／タブが閉じられると、データは消滅します。

ウィンドウ／タブ間では異なるsessionStorageとなり共有はできない。

サーバサイドで使われるSESSIONの仕様ににている。

WebStorage

サンプル
sessionstorage/*.html

シンプルに使える
～ sessionStorage ～

属性	説明
DATA取得	sessionStorage.getItem(KEYネーム);
DATA登録or更新	sessionStorage.setItem(KEYネーム, 値);
DATAを全削除	sessionStorage.clear();
１レコード削除	sessionStorage.removeItem(KEYネーム);

WebStorage

シンプルに使える localStorage

ブラウザ内に永続的にデータを保存するストレージ。

保存は「ドメイン名:ポート番号」の組み合わせ「オリジン」単位で保存されます。（例：http://www.localhost:80）
「オリジン」が同等であればブラウザを閉じた後も再度データにアクセス可能。

保存量は、「オリジン単位：５M」、Cookieが4KB。
保存期間は特になし。

自身の意志で削除しない限りデータは残ります。

WebStorage

サンプル
localstorage/*.html

シンプルに使える
～ localStorage ～

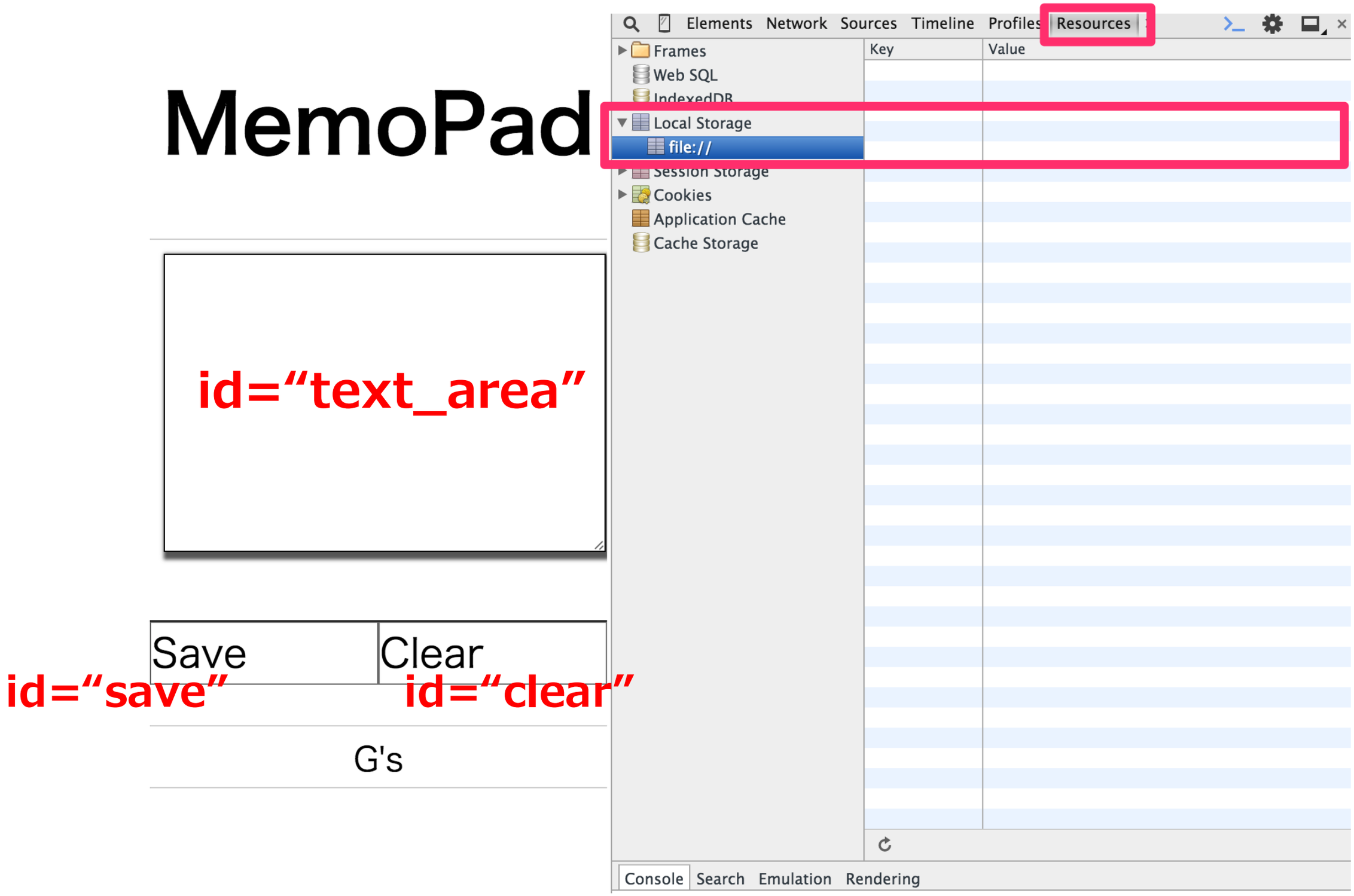
属性	説明
DATA取得	localStorage.getItem(KEYネーム);
DATA登録or更新	localStorage.setItem(KEYネーム, 値);
DATAを全削除	localStorage.clear();
１レコード削除	localStorage.removeItem(KEYネーム);

WebStorage

MemoPadを作成

- MemoPad

サンプル
memo/index.html



MemoPad

サンプル
memo/index.html

HTML

```
<header>
  <h1>MemoPad</h1>
</header>

<main>
  <textarea id="text_area"></textarea>…入力表示
  <ul>
    <li id="save">Save</li>   ・・・データ保存
    <li id="clear">Clear</li>   ・・・データ削除
  </ul>
</main>

<footer><small>G's</small></footer>
```

MemoPad

サンプル
memo/index.html

JS

```
//最初に必ず記述！

Window.onload = function(){

    //この中にSCRIPTを記述していく

};
```

MemoPad

サンプル
kadai/index.html

JS

```
//1.Save クリックイベント
document.querySelector("#save").onclick=function(){
  var value = document.querySelector("#text_area").value;
  localStorage.setItem("memo", value);
   alert("Saveしました");
};
```

MemoPad

サンプル
memo/index.html

JS

```
//2.clear クリックイベント
document.querySelector("#clear").onclick=function(){
  localStorage.removeItem("memo");
  alert("Clearしました");
  document.querySelector("#text_area").value="";
};
```

MemoPad

サンプル
memo/index.html

JS

```
//3.ページ読み込み：保存データ取得表示
if(localStorage.getItem("memo")){
	var value = localStorage.getItem("memo");


document.querySelector("#text_area").value=value;
}
```

IndexedDB API

IndexedDB API

◇ IndexedDB API

IndexedDBはSQLデータベースと違い、インデックスを使用することにより高パフォーマンス検索が可能な構造化された多くのデータを保存するKey-value型のクライアントサイドストレージAPIです。LocalStorage が比較的少量のデータを保存するのに有用であるのに対し、 IndexedDBは構造化された非常に多くのデータの保存するのに有用な手段を提供します。

※最新のブラウザの対応についてこちらを参照ください
http://caniuse.com/#search=indexedd

◇indexed db : データベース接続

データベース接続	var idb = window.indexedDB;

◇indexed db : open()メソッドイベントハンドラ

イベントハンドラ名	説明
onsuccess	成功した場合
onupgradeneeded	DBの新規作成時またはバージョン変更時に1回だけ実行される関数
onerror	失敗した場合

IndexedDB API

サンプル
index_db/indexed.html

◇ IndexedDB ：オブジェクトストアの作成・削除

イベントハンドラ名	説明
idb.createObjectStore	オブジェクトストアの作成
idb.deleteObjectStore	オブジェクトストアの削除

◇in-line key と out-of-line key

イベントハンドラ名	説明
in-line key	var store_name=“demo_store”; var store = **idb**.createObjectStore(store_name, { autoIncrement: true, keyPath: "id” }); var store_name=“demo_store”; var store = **idb**.createObjectStore(store_name, { autoIncrement: false, keyPath: "id” });
out-of-line key	

あとの細かいことはサンプルファイルで解説

※サンプル応用：indexed.html

File API

File API

サンプル
file/file_readinfo.html

◇FileAPI

HTML5で定義されたファイル操作に関するAPIです。File APIのFileオブジェクトを利用することで、ローカル（PC内）にある『ファイル情報』や『データ本体』の読み込みができます（テキストファイル、画像ファイルなど）。
JavaScriptで直接ローカルにあるファイルのデータを処理して結果をユーザーに提示できるので、ブラウザー上のJavaScriptのみで完結するWebアプリの可能性が広がります。

◇現在File APIに対応しているブラウザーのバージョンです。

ブラウザー覧	対応バージョン
Internet Explorer	10以降
Firefox	3.5以降
Google Chrome	6以降
Safari	５以降
Opera	10以降

◇File情報を取得するプロパティ（ file_readinfo.html ）

プロパティ	説明
name	名前
type	コンテンツタイプ
size	サイズ（バイト単位）
lastModifiedDate	最終更新日時

File API

サンプル
file/file_readview.html

サンプル
file/file_readimage.html

◇ファイルの読み取り

FileReaderオブジェクト	FileReaderオブジェクト内容
FileReader.readAsText (Blob|File, opt_encoding)	result プロパティにはファイル/ブロブ データが**「テキスト文字列」**として格納されます。デフォルトでは、この文字列は「UTF-8」としてデコードされます。オプションのエンコード パラメータを使用すると、他の形式を指定できます。
FileReader.readAsDataURL (Blob|File)	result プロパティにはデータ URL としてエンコードされたファイル/ブロブ データが格納されます。**（画像/音声）**
FileReader.readAsArrayBuffer (Blob|File)	result プロパティには、ファイル/ブロブ データが ArrayBuffer オブジェクトとして格納されます。**（テキスト文字列でない）**

◇その他のメソッドとプロパティ

プロパティ名	内容
state	在の読み込み処理の状態を取得する（読み取り専用）
result	読み込み成功後に、中身のデータを取得する。（読み取り専用） ※resultは readAsTextなど上記FileReaderオブジェクト を実行した後、onloadイベントのタイミングで取得できます。
abort	読み込みを破棄する

File API

サンプル
file/file_graph.html

◇FileReaderのイベント

プロパティ名	内容
onloadstart	読み込み開始
onprogress	読み込み中
onabort	読み込みを破棄する
onerror	エラーが発生した
onload	読み込み成功して完了
onloadend	読み込み完了（エラーを含む）

◇ブラウザ対応チェック処理

```
if (window.File && window.FileReader && window.FileList && window.Blob) {
      // サポートしてるので、処理をここに記述
} else {
     alert('ブラウザに対応してません');
}
```

◇サンプルファイル一覧

サンプルファイル名	サンプル操作で使用するファイル
file_readinfo.html	data/responsive.jpg
file_readview.html	data/file_readview.txt
file_readimage.html	data/responsive.jpg
file_graph.html	data/chart_data.txt

Websocket

WebsocketAPI

◇WebsocketAPIとは？

HTTPはリクエスト・レスポンスで通信するため、クライアントがサーバにリクエストを送った時にしか、サーバからデータを取得することができませんでした。
WebSocketは常時コネクションを維持することで、サーバ側からもクライアント側からもリアルタイムに双方向データ通信することができます。

インターネットの標準化団体(W3CとIETF)が関与している、サーバー/ブラウザ間通信のための双方向通信用技術規格です。元々はHTML5の仕様の一部として策定されていましたが、現在はHTML5から切り離された規格として策定が進んでいます

◇WebsocketAPIのメリット・デメリット

リアルタイム通信が可能になる（メッセンジャーのようなアプリが作れる）
Node.JSを使用すればサーバ・クライアントともにJavascriptで構築可能
サーバエラーが発生すると処理が落ち、接続が切れる
サーバサイド側の構築・設定が別途必要（一般的なレンタルサーバでは不可）

WebsocketAPI

イベント解説	イベントハンドラ
WebSocketサーバーに接続時	**var ws** = new WebSocket("ws://localhost:8000");
コネクション確立時	**ws.onopen** = function() {…}
メッセージ受信時	**ws.onmessage** = function(e) {console.log(e.data);}
エラー発生時	**ws.onerror** = function() {…}
コネクション切断時	**ws.onclose** = function(event) {}
メッセージ送信（メソッド）	**ws.send**(“メッセージ文字列");
コネクションを閉じる（メソッド）	**ws.close**();

```
var ws = new WebSocket("ws://localhost:8000"); // socketのopen
ws.onopen = function() {          // コネクション接続
    console.log("connection is opened.");
}
ws.onclose = function() {         // コネクション切断時
    console.log("connection is closed.");
};
ws.onmessage = function (e) {  // メッセージ受信
    console.log( e.data );
}
function sendMessage() {          // メッセージ送信
    ws.send(“メッセージ送信");
}
```

XML Http Request

XML Http Request

◇とは？

XMLHttpRequest (XHR) は、非同期なデータの通信を実現するための API です。
Ajaxを聞いたことがあるかもしれませんが、その根幹をなす技術です。

◇使用時に気をつけるPOINT
スクリプトのオリジン(http://www.○○○.com:80)とリクエスト先のオリジンが
一緒でなくてはいけない。（同じURLのドメインと言えば、イメージしやすい）
端的に言えば、相対パスでリンクが貼れる範囲ということです。
※*Cross-Origin Resource Sharing*

◇利用事例：
よく見かける利用方法として、ページ遷移せずにサーバからデータを読み込み、
ページを更新する方法に使用されます。

◇仕様メリット
ブラウザはシングルスレッドで処理します。そのため、同期通信を行なった場合に、
通信が終わるまで「UIブロッキング」が発生して画面が固まったような状態になり
ます。その「UIブロッキング」状態を避けるために非同期通信を利用するのです。

XMLHttpRequest

サンプル
xhr/xhr01.html

サンプル
xhr/xhr02.html

イベント解説	イベントハンドラ
XMLHttpRequestオブジェクト生成	**var xhr** = new XMLHttpRequest();
HTTPリクエスト送信の準備	**xhr.open**(‘GET’, ‘file.txt’, true);
ファイルタイプを指定	**xhr.responseType** = 'text'; *(text/json/blob/document/arraybuffer)*
HTTPレスポンス受信完了時の処理を定義	**xhr.onload** = function(e) {};
HTTPリクエストを送信	**xhr.send**();
HTTPレスポンス受信データを取得	**xhr.response**
HTTPステータスコード（通信結果）	**xhr.status**

```
//XMLHttpRequestのAjax通信の基本的な流れ
var xhr = new XMLHttpRequest(); //XMLHttpRequestオブジェクト生成
xhr.open(‘GET’,‘file.txt’);              //引数1:POST/GET, 引数2:URL
xhr.responseType = ‘text’;               //blob=画像
xhr.onload = function(e) {
    if (this.status == 200) {            //通信ステータス［200=成功］
        console.log(this.response);   // データ取得
    }
};
xhr.send();                                 // HTTPリクエストを送信
```

geolocation

geolocationとは

◇ geolocation

位置情報を取得することができるAPIです。

geolocationがサポートされているブラウザであれば、デバイス関係なしで使用することが可能です。

座標位置からMAPと連携や、ピンポイント天気予報の作成、
ローカルニュースなど位置情報からサービスを広げることも可能です

サポート状況の確認
http://caniuse.com/#feat=geolocation

geolocationとは

◇　位置を取得

位置を取得する方法が２つあります。

・現在の位置を取得する
navigator.geolocation.getCurrentPosition(成功,失敗,option)

・位置を定期的に取得する
navigator.geolocation.watchPosition(成功,失敗,option)

成功時には「 position 」オブジェクトを取得します
取得内容は次ページの通りです。

◇　取得停止

定期的に取得する「watchPosition」を停止する方法
「watchPosition」は制作時にidを返します。
idを使用して「clearWatch」で停止させます

```
navigator.geolocation.clearWatch(watchID)
```

geolocationとは

◇positionプロパティ

位置取得が成功すると以下プロパティを取得できます。

プロパティ	説明
position.coords	緯度経度などの情報
position.timestamp	位置を取得した時刻

◇ 位置を取得

position.coords で取得した経度緯度の情報について

プロパティ	説明
coords.latitude	緯度
coords.longitude	経度
coords.altitude	高度
coords.accuracy	正確性
coords.altitudeAccuracy	高度の正確性
coords.heading	方位
coords.speed	速度

geolocationとは

◇ optionの指定

３つめの引数では関数を実行する時のオプションを指定します
指定しなくても可能

プロパティ	説明
option.enableHighAccuracy	より高い正確性を求める(取得時間が遅くなる)スマフォではGPSを利用する（バッテリー消費が早い）
option.timeout	タイムアウトまでの時間
option.maximumAge	位置情報の有効期限

◇ 失敗時の取得

失敗時には「errorオブジェクト」を取得します
取得内容は以下の通りです。

プロパティ	説明
error.code	エラーコード 1.位置情報の取得が許可されていない 2.位置情報の取得が利用できない 3.タイムアウト
error.message	エラーメッセージ

WebWorker

Web Workers

◇HTML5以前～：

JavaScriptでは「 マルチスレッドプログラミング 」をおこなうことができませんでした。
JavaScriptは Java のようなスレッド処理ができないので、
時間のかかる処理を実行するとブラウザーが停止状態になってしまい、
その間ユーザーは「UIブロッキング」で操作できない問題がありました。

◇HTML5以降～：

JavaScriptで「 マルチスレッドプログラミング 」が可能です。
HTML5 関連APIである 「 Web Workers 」を利用します！
つまりは「Web Workers」で並行処理が可能となります！

※Workers内ではDOMやUI操作に関する処理はできません。

Web Worker

サンプル
webworkers/not_worker.html

サンプル
webworkers/worker.html

◇ワーカー処理の流れ

1.最初の作業としてメイン ページで新しい Worker オブジェクトを作成する必要があります。

```
var worker = new Worker('task.js');
```

2.ワーカーを作成したら、postMessage() メソッドを呼び出してワーカーを開始します。

```
worker.postMessage(“文字列 or 配列データ”); // Start the worker.
```

3.ワーカーファイル側の「onmessage」が処理を開始

```
onmessage = function (event) {
    //負荷の大きい処理を実行…
    postMessage(“処理完了”);
};
```

4.ワーカーファイル側の「postMessage」から結果値を受信処理

```
worker.onmessage = function (event) {
    console.log( event.data );
 };
```

※サンプルはFireFox最新版でおこないます。

Navigation Timing

Navigation Timing API

◇Navigation Timing API

ブラウザーがWebの画面を表示する時にどこにどれだけ時間がかかっているか知るためのAPI

開発者ツールのコンソール「コマンドライン」に以下コマンドを実行。

```
window.performance
```

Elements Network Sources Timeline Profiles Resources Audits Console NetBeans

<top frame> ▼ Preserve log

```
> window.performance
<· ▼ Performance {onwebkitresourcetimingbufferfull: null, memory: MemoryInfo, timing: PerformanceTiming, navigation:
   PerformanceNavigation, getEntries: function…} i
   ▶ memory: MemoryInfo
   ▼ navigation: PerformanceNavigation
       redirectCount: 0
       type: 0
     ▶ __proto__: PerformanceNavigation
     onwebkitresourcetimingbufferfull: null
   ▼ timing: PerformanceTiming
       connectEnd: 1428997945367
       connectStart: 1428997945367
       domComplete: 1428997946370
       domContentLoadedEventEnd: 1428997945830
       domContentLoadedEventStart: 1428997945764
       domInteractive: 1428997945764
       domLoading: 1428997945430
       domainLookupEnd: 1428997945367
       domainLookupStart: 1428997945367
       fetchStart: 1428997945367
```

http://caniuse.com/#feat=nav-timing (ブラウザ対応状況)

Navigation Timing API

◇window.performance.navigation.type

0: TYPE_NAVIGATE (クリックされたかURLを入力してページに移動してきた)
1: TYPE_RELOAD (リロードしてページを表示した)
2: TYPE_BACK_FORWARD (ブラウザの「進む」「戻る」ボタンでページを移動した)

◇window.performance.timing

window.performance.timingオブジェクトは ユーザーがページを開こうとしたとき(ナビゲーション開始)からロードが終わるまでの各イベント毎の時間情報です。
データ分析する際には、各項目の「start～end」の差を求める必要があります。

```
var timing = window.performance.timing;
console.log("onLoadイベント: " + (timing.loadEventEnd - timing.loadEventStart));
console.log("リクエスト時間: " + (timing.responseStart - timing.requestStart));
console.log("レスポンス時間: " + (timing.responseEnd - timing.responseEnd));
console.log("DOMの構築時間: " + (timing.domComplete - timing.domLoading));
console.log("appキャッシュ: " + (timing.domainLookupStart - timing.fetchStart));
console.log("DNS取得時間: " + (timing.domainLookupEnd - timing.domainLookupStart));
console.log("TCP接続時間: " + (timing.connectEnd - timing.connectStart));
console.log("Redirect: " + (timing.redirectEnd - timing.redirectStart));
```

サンプル： navigation/timing.txt
開発者ツールのコンソール「コマンドライン」に貼り付けて実行

High Resolution Time

High Resolution Time API

◇High Resolution Time API とは？

```
var mili      = Date.now();  //ミリ秒
var micro = performance.now();  //マイクロ秒
```

ナビゲーション開始時を起点とし、マイクロ秒で値取得できるnow()メソッドのみを提供します。performance.now()にでマイクロ秒での比較が可能です。より細かい粒度での測定が可能な高分解能タイマーになります。

開発者ツールのコンソール「コマンドライン」に貼り付けて実行

ブラウザ確認
http://caniuse.com/#search=High%20R

jQuery概要

jQueryのメリット

- 主要ブラウザに対応できる
- CSSセレクタ互換の記述が可能
- 拡張ライブラリが豊富
- 使用が簡単

  コード記述量を抑えられる

  Ajax,アニメーション,etc…

デザイナに優しい

jQueryとJavaScriptの記述例

セレクタ → **メソッド**

何に対して → 関数を実行する

```
$("#email").html("HTMLや文字列");

document.querySelector("#email").innerHTML="文字列";
```

同じ意味！jQuery短い！

jQuery準備

jQuery使用方法

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4   <meta charset="utf-8">
5   <meta name="viewport" content="width=device-width">
6   <script src="js/jquery-2.1.3.min.js"></script>
7   <link rel="stylesheet" href="css/sample.css">
8   <title>sample</title>
9   <script>
10  $(document).ready(function(){
11  //ここに記述していきます！
12
13
14
15  });
16  </script>
17  </head>
18  <body>
19
20  <header>
21      <h1>jQuery練習</h1>
22  </header>
```

①読み込んで使う

②この中に記述

jQuery最初のルール

- **基本**

```
<script>
 $(document).ready(function(){
        //DOM構築後のタイミングで処理される
 }
</script>
```

- "$"と"jQuery"というものだけ使えるようになる。
  （短い $ を使うことが多い）

※Wordpressでは、jQueryを使用する。

jQuery基本

jQuery文法

セレクタ → メソッド → イベント

何に対して → どのような事を → どのタイミングで

例)

```
$("#email").on("click",function(){
    $("#email").html("HTMLや文字列");
});
```

セレクタ は覚える！

セレクタ → メソッド → イベント

何に対して → どのような事を → どのタイミングで

jQuery［セレクタ］

- CSSセレクタ互換の記述が可能

#id → $("#id名")

.class → $(".class名")

Element → $("element名")

[name=a] → $("[name=a]")

サンプルファイルで練習：$(セレクタ).html("練習");

jQuery［セレクタ］

セレクタ（CSSと同様）

複数セレクタを指定	$("div, p, a")
要素の親子関係	$("div > p")
先祖子孫	$("div p")
最初の要素	$("li:first")
偶数の要素	$("td:even")
奇数の要素	$("td:odd")

などなど…

メソッド は覚える！

セレクタ → メソッド → イベント

何に対して → どのような事を → どのタイミングで

jQuery[メソッド]

例）

```
$("セレクタ").メソッド();

var elem = '<a href="#">次ページ</a>';
$("#id").html(elem);
$("#id").text(elem);
$("#id").css("color","#ff0");
$("#id").show(4000);        //hide
$("#id").prepend(elem);     //要素の先頭にHTML要素を追加
$("#id").append(elem);      //要素の最後にHTML要素を追加
$("#id").empty();           //子要素を全て削除
$("#id").remove();          //要素を全て削除
```

サンプルファイルを使って練習

イベントは覚える！

セレクタ → メソッド → イベント

何に対して → どのような事を → どのタイミングで

jQuery[イベント]

例）

```
$("#id").on("click", function(){
    $("セレクタ").css("color","#ff0");
    $("セレクタ ").append("<p>123</p>");
});
```

"click"," dblclick","mouseout" ,"mousedown" ,"mouseup","mousemove" "mouseenter","mouseleave" ,"change","select","focus","submit","resize", "scroll","ready","keydown","keypress","keyup"…

onメソッドはページをロードした後に追加された要素に対してもイベント取得が可能！

サンプルファイルを使って練習

最初はこれ覚えておけば！

セレクタ → メソッド → イベント

何に対して → どのような事を → どのタイミングで

書き方は1つの文法のみ！！！

演習

【演習】おみくじアプリ作成

◇仕様：

- サンプル" omikuji_tpl.html "を使用して作成
- “おみくじボタン”クリック後に「大吉・中吉・小吉・吉・凶」の５種類をランダムでHTMLに表示する
  表示記述例）$("#id名").html("大吉");
- 「大吉・中吉・小吉・吉・凶」はIF文の中に記述します

課題

【課題】じゃんけんアプリ作成（event）

◇仕様：

- サンプル“janken_tpl.html”を使用して作成
- ユーザー操作：“グー”、“パー”、“チョキ”のボタンをクリック
- IF分岐処理を沢山つかう場合もある。
- 流れ：クリック → 乱数を発生 → IF文 →「◯◯」HTMLに表示
  ※ １＝パー、２＝グー、３＝チョキとか先に決めておくといい