

Slide URL

<https://vu5.sfc.keio.ac.jp/slide/>

Web情報システム構成法

第9回 JavaScript入門 (2)

萩野 達也 (hagino@sfc.keio.ac.jp)

JavaScript入門（前回）

- ▶ オブジェクト指向について
 - ▶ JavaScriptの誕生
 - ▶ プロトタイプベースのオブジェクト指向
- ▶ 言語
 - ▶ 構文および制御構造
 - ▶ 代入
 - ▶ 条件文
 - ▶ 繰り返し
 - ▶ 関数
 - ▶ データ型
 - ▶ 基本
 - ▶ オブジェクト
- ▶ HTMLへの埋め込み
 - ▶ `<script> …… </script>`
 - ▶ Document Object Model

JavaScriptの実行

▶ 読み込み時

- ▶ `<script> …… </script>` は読み込み時に実行される.
- ▶ 関数などは定義されるだけなので, 実際の実行ではない.
- ▶ 変数の初期化なども行われる.

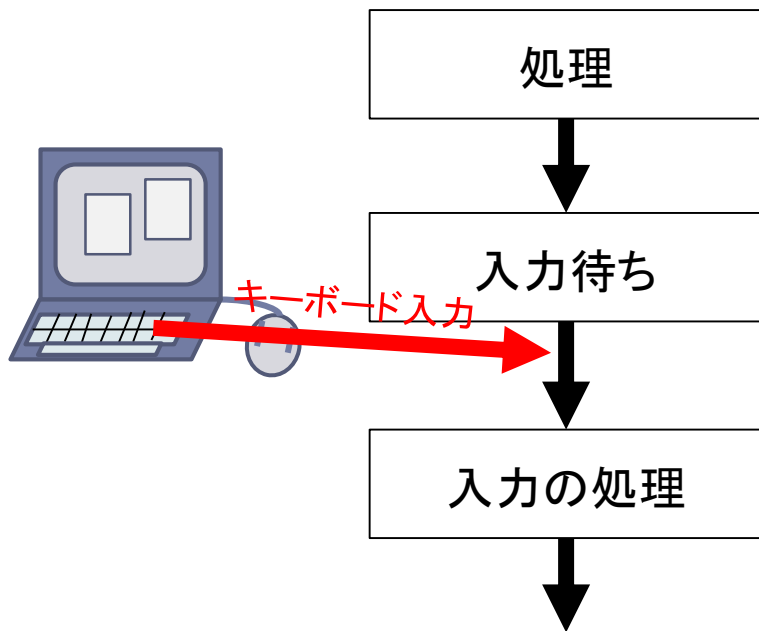
▶ イベント発生時

- ▶ ボタンを押すなどの**イベント**が発生した時に, 指定されたプログラムが実行される.
 - ▶ **イベントハンドラ**
 - イベント処理を行うプログラム
 - 前もってイベントごとに登録しておく
- ▶ イベントは非同期に発生する.

同期入力と非同期入力

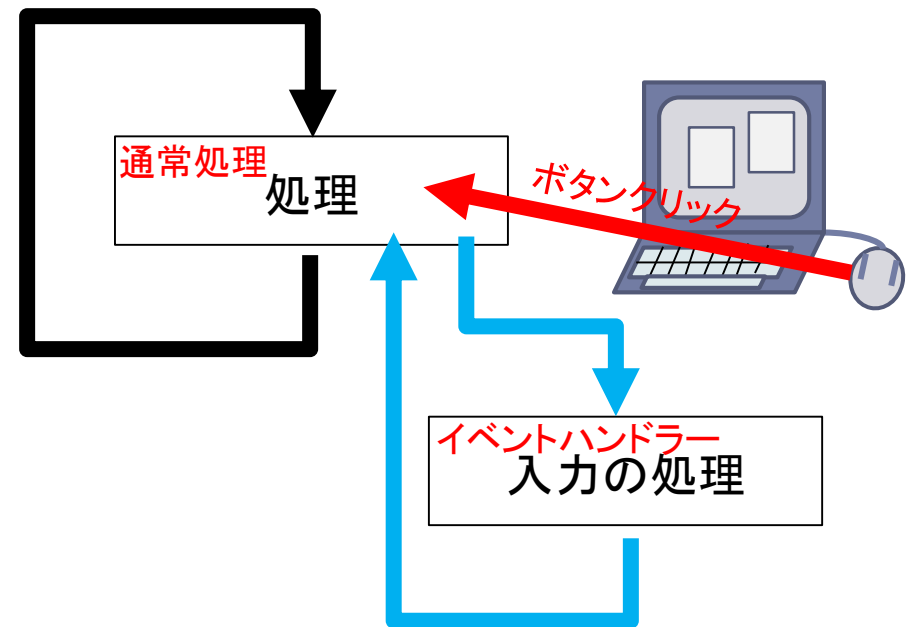
▶ 同期入力

- ▶ プログラムが指定した時に
入力を行う。



▶ 非同期入力

- ▶ プログラムが別の処理中
にも入力が発生する。



JavaScript Event

- ▶ キーボード関係

- ▶ keydown, keyup, keypress

- ▶ マウス関係

- ▶ mouseover, mousedown, mouseup

- ▶ 要素関係

- ▶ click, focus, input

- ▶ ウィンドウ関係

- ▶ resize, scroll

イベント処理

▶ イベントハンドラ

- ▶ イベントが発生した時に行う処理を記述
 - ▶ HTML要素の属性として指定
 - ▶ 要素やオブジェクトに対してイベントハンドラを設定する

▶ HTMLでの指定

```
<button onclick="イベントハンドラ">ボタン</button>
```

```
<p>どこでも<span onclick='window.alert("Hello!")'>クリック</span>できる</p>
```

▶ JavaScript内での指定

- ▶ HTML要素に関係ないものは、この方法でしか指定できない

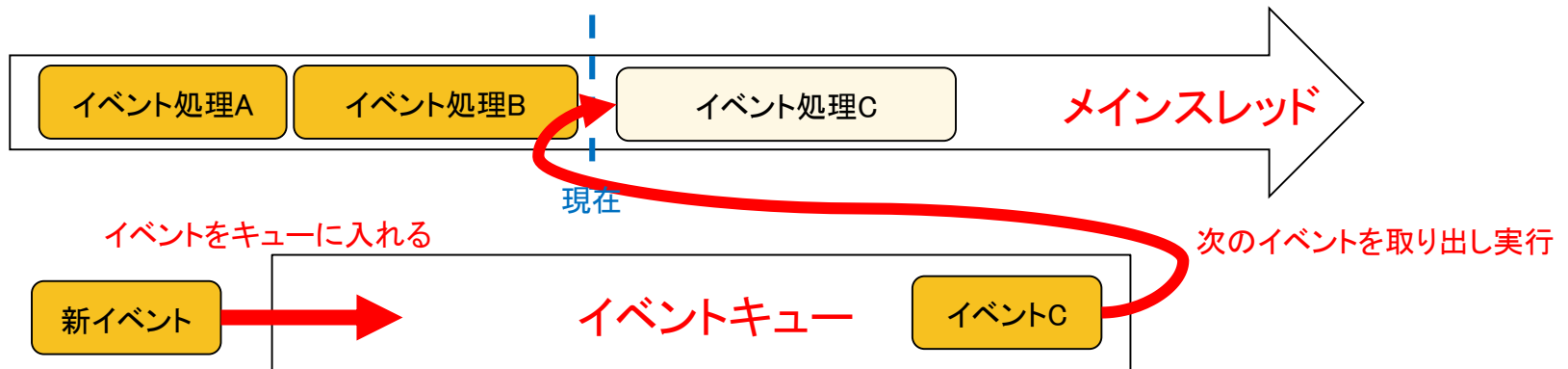
```
element.addEventListener(イベント, 関数);
```

```
document.getElementById('myBtn').addEventListener('click', function(e) {  
    document.getElementById('demo').textContent = "Hello World!";  
});
```

- ▶ 設定したイベントハンドラには発生したイベントが渡される。

JavaScriptの並列処理

- ▶ JavaScriptは基本的にシングルスレッドです。
 - ▶ 同時に複数の処理を行わない。
 - ▶ 変数をロックして保護するなどの必要はない。
 - ▶ イベントはキューに貯められ、一つずつ処理される。

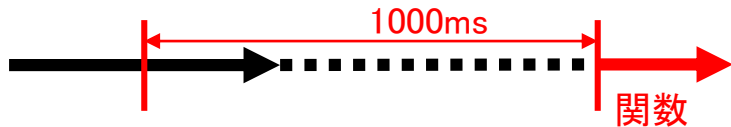


- ▶ あるイベントの処理が重いと、処理が滞る。
 - ▶ WebWorkersを使ってバックグラウンドで処理を行うことも可能。
 - ▶ マルチスレッドになる。
 - ▶ WebWorkersに処理のためのメッセージを送り、結果をイベントとして受け取る。
 - ▶ WebWorkersは直接はDOMを操作することはできない。

タイマーの利用

- ▶ 一定時間後に何かの処理を行いたい
 - ▶ setTimeout でタイムアウトのイベント処理を指定

```
timer = setTimeout(関数, ms時間);
```

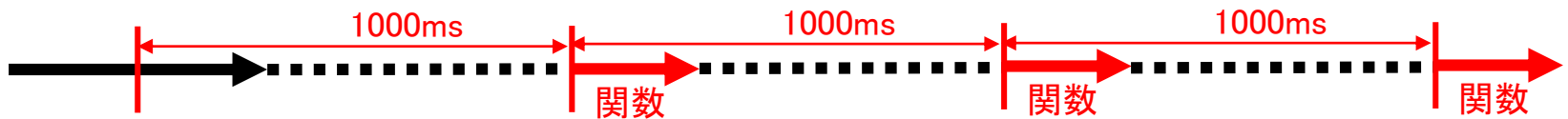


```
setTimeout(関数, 1000);
```

タイマーを止めるには `clearTimeout(timer)`

- ▶ 一定時間間隔で何かの処理を行いたい
 - ▶ setInterval で一定間隔で実行する処理を指定

```
timer = setInterval(関数, ms時間);
```



```
setInterval(関数, 1000);
```

タイマーを止めるには `clearInterval(timer)`

setIntervalの例

▶ デジタル時計を作ってみよう

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Javascript Timer</title>
  </head>
  <body>
    <h1>Javascript Clock</h1>
    <p id="clock"></p>
    <script>
      function showTime() {
        var element = document.getElementById('clock');
        var now = new Date();
        element.textContent = now.getHours() + ':' + now.getMinutes()
                              + ':' + now.getSeconds();
      }
      showTime();
      setInterval(showTime, 1000);
    </script>
  </body>
</html>
```

Ajax

- ▶ Ajax = Asynchronous JavaScript + XML
 - ▶ Web 2.0で登場した.
 - ▶ JavaScriptとXMLを使って非同期にサーバとの通信を行う.
- ▶ Webページを取得するHTTPは基本的に同期的
 - ▶ ページを取得するリクエストをサーバに送り, 文書が返ってくるまで待つ
- ▶ 非同期的な処理を行いたい
 - ▶ 最初に軽いページとして全体を受け取り, ユーザがブラウ視している間に徐々に中身を増やしていく
 - ▶ ユーザの要求に従って内容をサーバから取得する
 - ▶ フォームの送信を行わずに, サーバにデータを送る

XMLHttpRequest オブジェクト

- ▶ JavaScript内からhttpを使ってサーバにアクセスしデータ
を取得する

HTTPを行うオブジェクトの生成

```
var xhr = new XMLHttpRequest();
```

```
xhr.onreadystatechange = function(){
```

```
  if (this.readyState == 4 && this.status == 200) {  
    xhr.responseText にサーバから送られてきたデータが入っている  
  }
```

```
};
```

```
xhr.open("GET", "URL", true);
```

```
xhr.send();
```

送られてきた
データの処理

HTTPリクエストのメソッドとURI

非同期処理を指定

リクエストを送る

GETまたはPOSTメソッドを指定

GETとPOSTでのデータの受け渡し

▶ GET

- ▶ URLに問い合わせの形で追加する

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function(){ ... };
xhr.open("GET", "http://.../chat.php?method=get&id=123");
xhr.send();
```

▶ POST

- ▶ send でデータを渡す
- ▶ データの形式を指定する必要がある
 - ▶ GETと同じにするには application/x-www-form-urlencoded

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function(){ ... };
xhr.open("POST", "http://.../chat.php");
xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhr.send("method=post&user=abc&message=Hello%20World");
```

URIに書くことのできる文字には制限があるため encodeURIComponent を使ってエンコードすると良い

```
... + '&message=' + encodeURIComponent(m);
```

JSON

- ▶ サーバとクライアントでJavaScriptのオブジェクトをやり取りする場合には、JSON形式を用いることが多い。
- ▶ JSON = JavaScript Object Notation
 - ▶ JavaScript以外でも利用できるようにJavaScriptのデータを表現したもの
 - ▶ JavaScriptデータのシリアライズ

JavaScriptデータ

```
{ name:"Hagino", age:20, class:["Web", "Haskell"] }
```

JSON



シリアライズ (stringify)



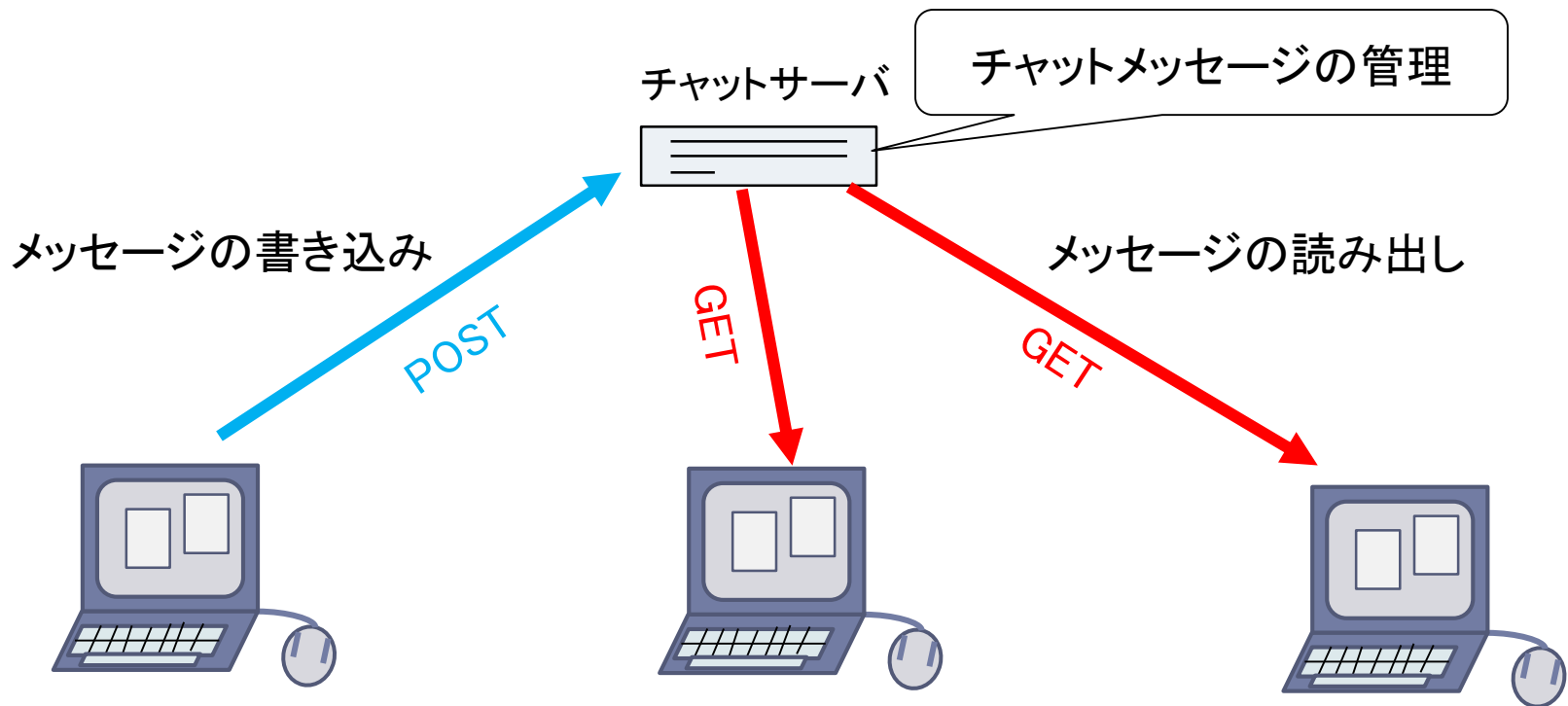
デコード (parse)

```
{"name":"Hagino", "age":"20", "class":["Web", "Haskell"]}
```

- ▶ **x = JSON.stringify(obj);**
 - ▶ JavaScriptのデータobjをJSONとしてシリアライズした文字列を返す
- ▶ **obj = JSON.parse(x);**
 - ▶ JSON文字列をデコードしてJavaScriptのデータを返す

チャットを作ってみよう

- ▶ 複数人がメッセージを書き込み、それを共有できるチャットのアプリケーションを作ってみましょう。



チャットサーバAPI

▶ チャットサーバURL

- ▶ `http://web.sfc.keio.ac.jp/~hagino/wis/chat.php`

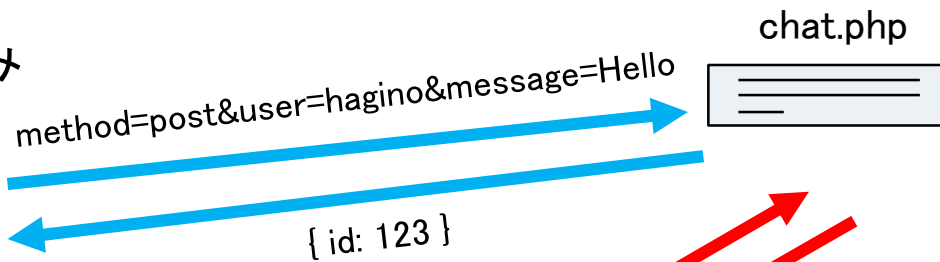
▶ メッセージの書き込み

▶ 引数

- ▶ `method=post`
- ▶ `user=ユーザ名`
- ▶ `message=メッセージ`

▶ 戻り値 (JSON)

- ▶ `{ id: メッセージ番号 }`



▶ メッセージの読み出し

▶ 引数

- ▶ `method=get`
- ▶ `id=メッセージ番号`

▶ 戻り値 (JSON)

- ▶ `{ id: メッセージ番号, user: ユーザ名, message: メッセージ }`



課題:チャットのクライアントを作成しなさい

- ▶ チャットサーバのAPIを使って、チャットを行うクライアントを作成しなさい。
 - ▶ chat.php が動作しない場合は、chat.phpとchatserver.phpをコピーし、サーバ名とポート番号を変更し、自分の環境で動作させてください。
 - ▶ CSSを使ってスタイルをカスタマイズしなさい。

▶ 提出

- ▶ <https://vu5.sfc.keio.ac.jp/kadai/>
- ▶ HTML (JavaScript) を提出
- ▶ JavaScriptはHTMLに埋め込むこと
- ▶ 締め切り: 6月17日正午

chat.html例

```
<!DOCTYPE html>
<html>
  ...
  <body>
    <header><h1>チャット</h1></header>
    <article>
      <form>
        <div>
          <label>氏名: <input type="text" id="u"></label>
          <label>メッセージ: <input type="text" id="m"></label>
          <input type="submit" value="書き込む"
            onclick="sendMessage();return false;">
        </div>
      </form>
      <div id="c"></div>
      <script>
        function sendMessage() { ... }
        function getMessage(id) { ... }
        getMessage(0);
      </script>
    </article>
  </body>
</html>
```


サンプルchat.html

▶ メッセージの書き込み

- ▶ 氏名とメッセージを入力できるformを用意
- ▶ 「書き込み」ボタンでsendMessageを呼び出す
- ▶ formの本来のsubmitを抑制するためにreturn falseとする
- ▶ sendMessage関数
 - ▶ formの入力テキストを取り出し, method=postとしてchat.phpに送る

▶ メッセージの読み出し

- ▶ getMessageによりサーバからメッセージを受け取る
- ▶ getMessage関数
 - ▶ method=getとしてchat.phpに送る
 - ▶ idは自分が受け取っている最新のメッセージの番号を与える
 - ▶ idの初期値は0で始める
 - ▶ 受け取ったメッセージをHTMLの適当なところに挿入する
 - ▶ 次のメッセージを受け取るために, 再帰的にgetMessageを呼び出す

まとめ

- ▶ JavaScriptの続き
 - ▶ 同期処理と非同期処理
 - ▶ イベント
 - ▶ タイマー
 - ▶ XMLHttpRequest