

～Webアプリケーション開発講座～
JavaScript入門



～ サンプルを動かしてみよう ～

手順

- (1) eclipse上に新しい動的Webプロジェクトを作成する。
プロジェクト名：Sample_5_07_1
- (2) プロジェクトの srcパッケージ直下に workパッケージを作成する。
- (3) 配布のSample_5_07_1フォルダからソースコードを取得して
workパッケージ直下にインポートする。
- (4) 同じくweb.xmlをドライブから取得して置き換える。
WebContent > WEB-INF > web.xml
- (5) サーバーを起動してLogin.javaを起動する。

～ JavaScriptで動きのあるWebページを作る ～

localhost:8080/ServletSample08_1/Lo...

ログインSample（入力チェック①）

ユーザーID :

パスワード :

ログイン

ID／パスワード未入力で
ログインボタンが押されると…

localhost:8080/ServletSample08_1/Lo...

ログインSample（入力チェック①）

ユーザーID :

パスワード :

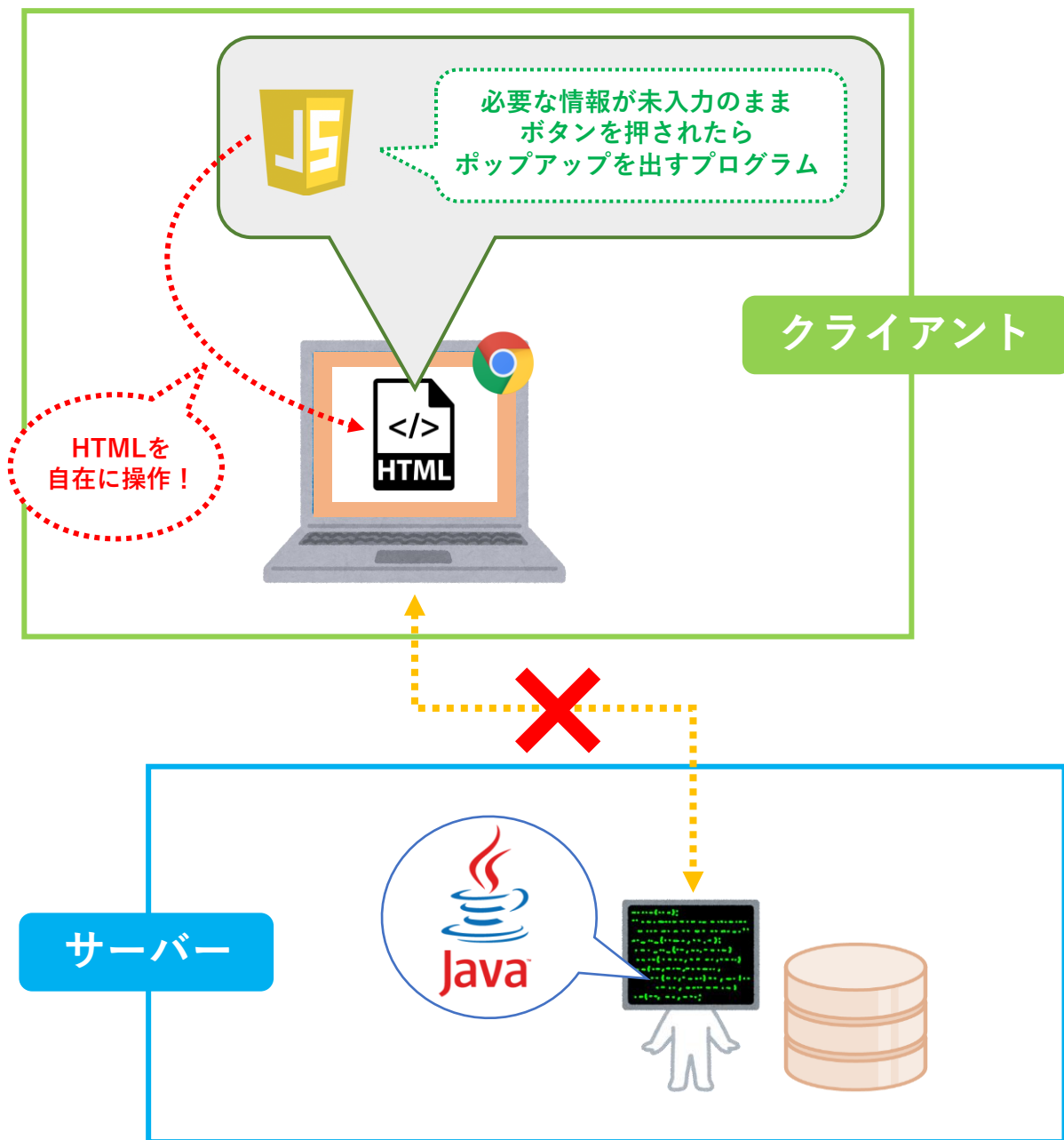
ログイン

localhost:8080 の内容
入力漏れの項目があります。

OK

ポップアップが
起動される！

～ JavaScriptで動きのあるWebページを作る ～



≪ JavaScript ≫

- **JavaScript**はプログラミング言語の一種です。（Javaとは関係ない）
最大の特徴は**Webブラウザ上で動作すること**で、クライアントの画面（HTMLやCSSなど）やブラウザの機能（ポップアップや戻るボタンなど）を自在に操ることが可能になります。
これにより**動きのあるリッチなWebページが作成可能**になります。
- JavaScriptはWebブラウザ上で動作し、**サーバーとの通信が発生しない**ためユーザーの操作に素早く対応できます。
基本的にページを跨ぐ動作はサーバレットで、ページ内で完結する動作はJavaScriptで組むという意識を持つとよいでしょう。
- JavaScriptは**scriptタグを使ってHTMLに埋め込む**形で記述します。
scriptタグはheadタグ直下かbodyタグ直下に記述します。
HTMLはブラウザに上から順番に読み込まれていきますが、scriptタグ内のJavaScript処理も同様に1行ずつ上から順に読み込み＆実行していきます。
こうした1行ずつ読み込んで実行していく実行方式を**インタプリタ**と言います。

～ JavaScriptで動きのあるWebページを作る ～

```
<html>
<head>
  <title>ログイン画面</title>
  <script type = "text/javascript">
    console.log( "▼-----scriptタグ (headタグ直下) 内での動作確認-----▼" );
    console.log( "HTMLは上から順にブラウザに読み込まれていきます。" );
    console.log( "scriptタグ内のJavaScript処理も上から順に実行されます。" );
    console.log( "▽まだID「ID_USER_ID」の要素は読み込まれていないため値を取得できない（エラー）" );
    console.log( document.getElementById("ID_USER_ID").name );
  </script>
</head>
<body>
  <h1>ログインSample (入力チェック①) </h1>
  <form action="ExecuteLogin" method="post">
    <p>ユーザーID:
      <br>
      <input type="text" name="USER_ID" maxlength="20" id="ID_USER_ID">
    </p>
    <p>パスワード: <br>
      <input type="password" name="PASSWORD" maxlength="20" id="ID_PASSWORD">
    </p>
    <input type="submit" value="ログイン" id="ID_SUBMIT" onclick="return preCheck()" >
  </form>
  <script type = "text/javascript">
    console.log( "▼-----scriptタグ (bodyタグ直下) 内での動作確認-----▼" );

    console.log( " (ローカルスコープである関数preCheck内の処理は初期の読み込みでは実行されない) " );
    function preCheck(){
      var elmUserId   = document.getElementById("ID_USER_ID");
      var elmPassword = document.getElementById("ID_PASSWORD");
      var canSubmit = true;
      if(elmUserId.value == "" || elmPassword.value == ""){
        alert("入力漏れの項目があります。");
        canSubmit = false;
      }
      return canSubmit;
    }
  </script>
</body>
</html>
```

scriptタグはheadタグ/bodyタグどちらの中で書いてもOK。

HTMLは上から順に読み込んでいき、scriptタグ内のJavaScriptの処理も同様に処理されていきます。
この処理は1行ずつコンパイル&実行されます。
(インタプリタ方式)

～ JavaScriptの文法 ～

≪ JavaScript文法の特筆事項（データ） ≫

□ 変数（プリミティブ）

JavaScriptは動的型付け言語という**変数に型を持たない**言語です。
定義したプリミティブな変数には数値型でも文字型でも論理型でも、
どんな型のデータも代入することが可能です。
変数宣言の方法は以下の3つが用意されています。

var 変数名 … 通常の変数宣言
let 変数名 … スコープ内でのみ有効な変数（ローカル変数）の宣言
const 変数名 … 読み取り専用の定数の宣言

□ 変数（オブジェクト）

Java同様に変数とメソッドの集合体であるオブジェクトを扱えます。
オブジェクト内で管理される変数はプロパティと呼ばれます。

□ データ型

扱えるデータは以下の6種類です。typeof演算子を用いると変数内の
データ型を確認することができます。

Number型 … 整数、小数点数（浮動小数点数）、Infinity（無限） など
String型 … 文字列
Boolean型 … true/false
Null型 … null
Undefined型 … undefined（変数宣言時、とりあえず設定される初期値）
Symbol型 … 識別子 ※難しいので見なかったことにしましょう。

□ 文字列リテラル

文字列の囲いは「 ” ” 」 「 ' ' 」 どちらでも構いません。

≪ JavaScript文法の特筆事項（その他） ≫

□ コメント

JavaScriptのコメントはJavaと同じで「 // 」 「 /* */ 」 を用います。

□ スコープ

JavaScriptのスコープは以下の2つです。

グローバルスコープ … 関数の外（scriptタグ直下）
ローカルスコープ … 関数の中

Javaと違って単純に「 { } （ブロック） 」内がスコープという話ではない点に注意しましょう。（ forなどの制御構文内はスコープとみなされない ）

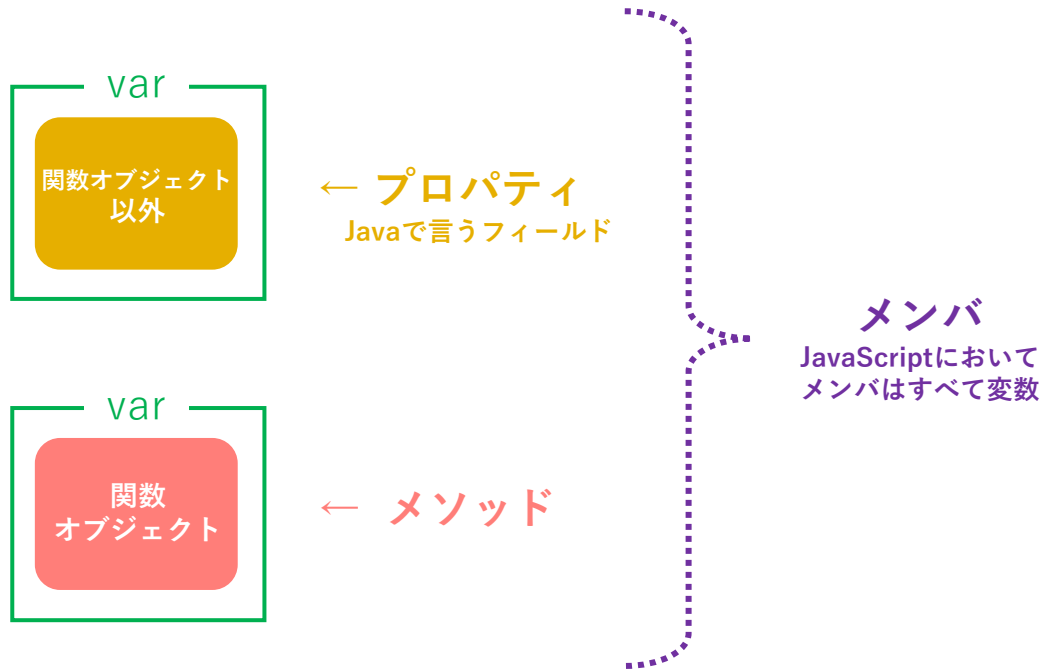
□ デバッグ

JavaScriptはブラウザで動作する言語であり、その挙動のチェックは
ブラウザ上でなければできません。これに伴いデバッグもブラウザで
提供されているデバッグツールを用いることが一般的です。

例） Google Chromeのデベロッパーツール

～ JavaScriptの文法 ～

オブジェクト



《JavaScript文法の特筆事項（関数）》

□関数（function）

JavaにおけるメソッドはJavaScriptでは関数、またはfunctionと呼ばれます。以下のようにscriptタグ内で記述しておけば呼び出して使用できるようになります。

```
function 関数名( 引数 ){ まとめた処理 }
```

Javaと違い戻り値に関する定義がありませんが、処理中にreturn文を書けば戻り値は返ります。

こうして定義された関数は文書全体の解析時に事前に読み込まれているため、定義されている行より上の処理からでも利用できます。

□関数オブジェクトと無名関数

JavaScriptでは**関数をオブジェクトとして扱うことが可能**です。
(関数オブジェクト)

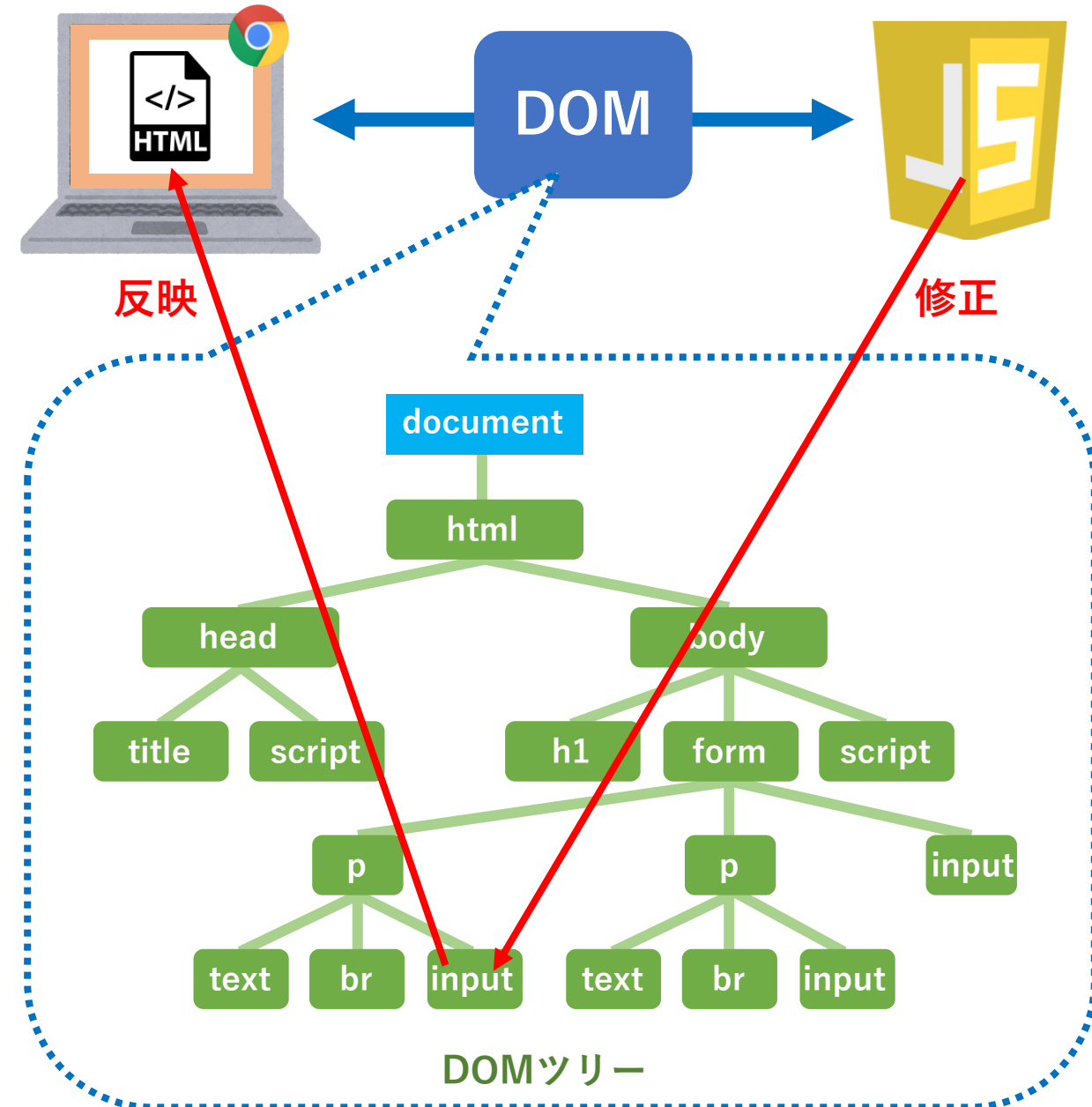
つまり**変数に入れることが可能**で、以下のように定義します。

```
var 関数名 = function( 引数 ){ まとめた処理 };
```

※ 関数の実行時は 関数名(引数) というように引数情報が必要です。

この右辺で作成されたオブジェクトを**無名関数**と言います。
無名関数は定義された行で生成されるため、この行より上の処理からは利用できません。

～ DOM ～



≪DOM／DOMツリー≫

□HTMLやXMLといったWebページを構成する文書情報をアプリケーションから操作するために用意されたAPIを**DOM (Document Object Model)**と言います。DOMを使用することで動きのあるWebページを作ることが可能になります。

□ブラウザは文書情報を読み込むとその内容を元にアプリケーションから操作可能なオブジェクトを生成します。このオブジェクトのことを**DOMツリー**と言います。

DOMツリーの情報とブラウザ上のHTMLなどの文書情報は連動しており、アプリケーションからDOMツリーに修正を加えるとその修正内容はブラウザ上へと即時反映されます。

□DOMツリーは複数の要素が階層構造で連っており、一つ一つの要素を**Node (ノード)**と呼びます。Nodeの中でも文書内のタグ情報と連動しているものは**Element (エレメント)**と呼び、JavaScriptから操作することになるのは基本的にこちらになります。

Elementの階層構造は文書のタグ構成と一致しており、htmlを頂点としてその下にheadとbody、headの下にはtitleなど、bodyの下にh1やformなどといった具合で構成されます。

□ブラウザすべての機能と情報をアプリケーションから操作可能とするオブジェクトを**ブラウザオブジェクト**と言い、DOMツリーはその中でも画面の表示情報を管理する**document**オブジェクトの直下に構築されていきます。

～ ブラウザオブジェクトとDOMツリー ～

プログラムで操作できるようブラウザのすべての情報をオブジェクト化したものを
ブラウザオブジェクトと言います。

ブラウザ
すべての情報

ブラウザオブジェクト

window

history

location

document

location

document

DOMツリー

html

head

body

title

script

h1

form

script

input

p

p

text

br

input

text

br

input



～ ブラウザオブジェクトとDOMツリー ～

ブラウザオブジェクトは階層構造で構成され、その最上位に位置するオブジェクトが**Windowオブジェクト**です。ブラウザのウィンドウ内すべての情報を管理します。

ダイアログを表示させる**alertメソッド**やウィンドウを閉じる**closeメソッド**など、ウィンドウに関連するメソッドも多数定義されています。

ブラウザ
すべての情報

ブラウザオブジェクト

window

history

location

document

location

document

DOMツリー

html

head

body

title

script

h1

form

script

input

p

p

text

br

input

text

br

input



～ ブラウザオブジェクトとDOMツリー ～

例えば`alertメソッド`を使ってアプリケーションからダイアログの表示をさせたい場合は`window.alert()` と書きます。
「window.」は省略可能であるため、単に `alert()` と記述しても構いません。

ブラウザ
すべての情報

ブラウザオブジェクト

window

history

location

document

location

document

DOMツリー

html

head

body

title

script

h1

form

script

input

p

p

text

br

input

text

br

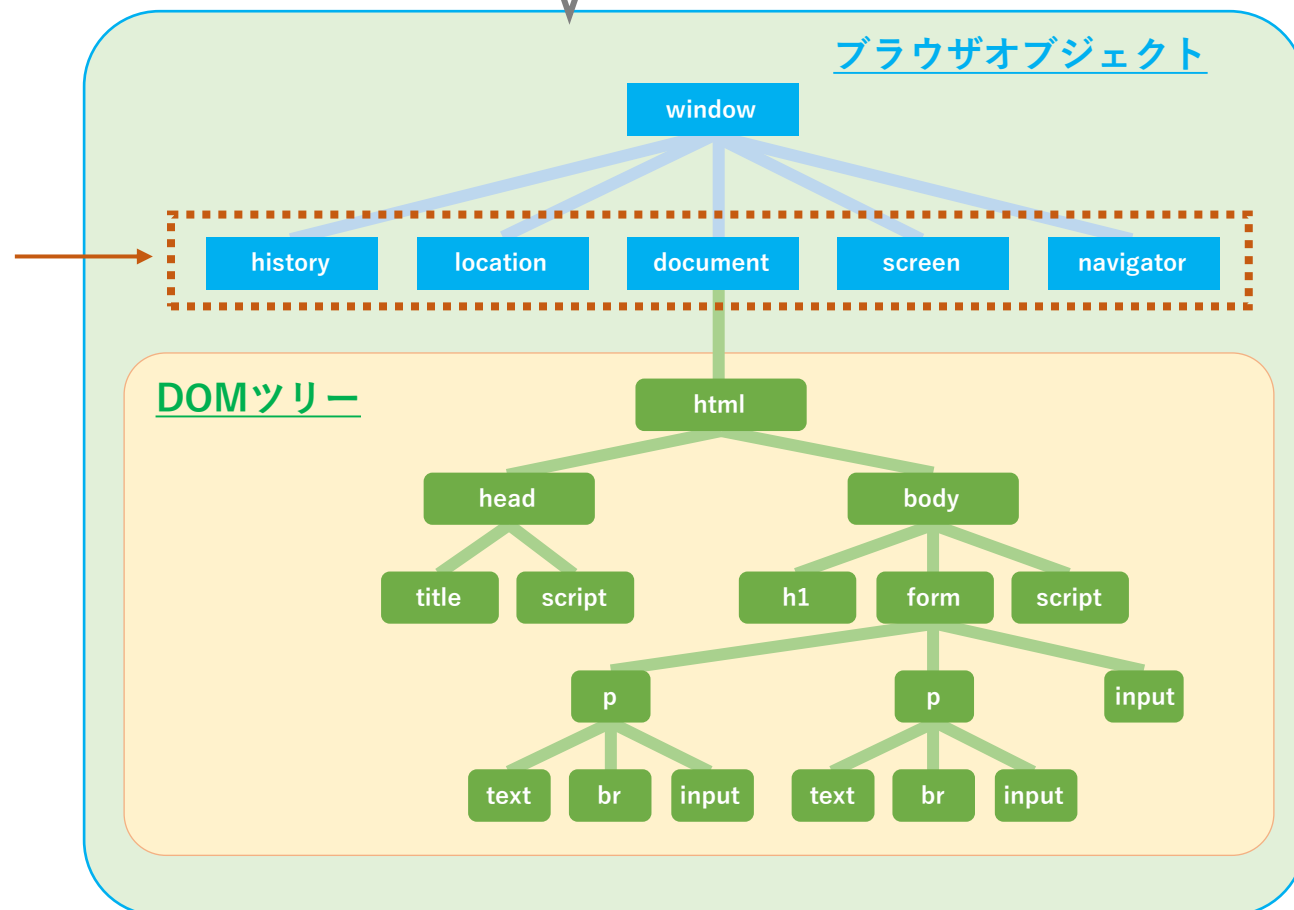
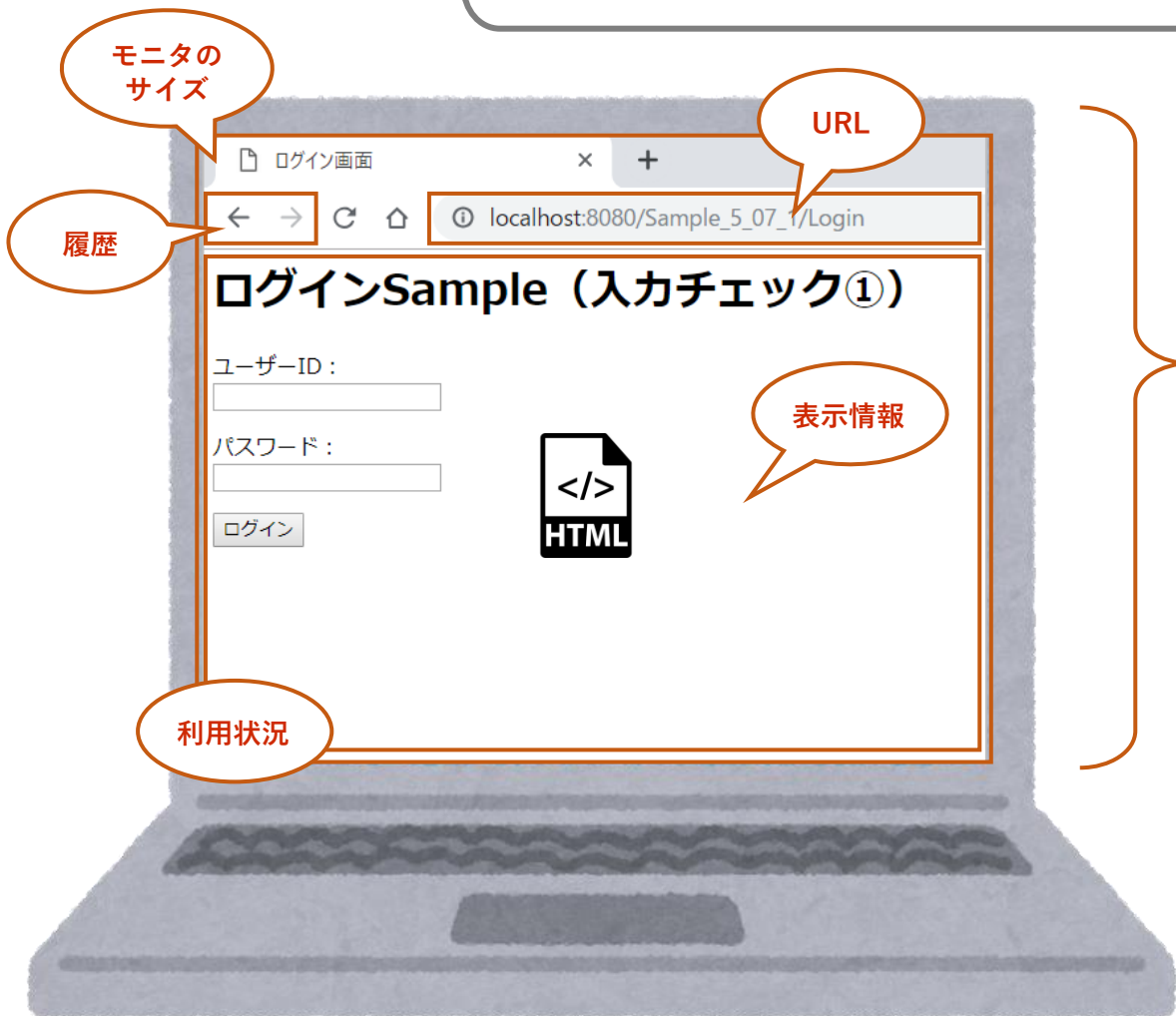
input



～ ブラウザオブジェクトとDOMツリー ～

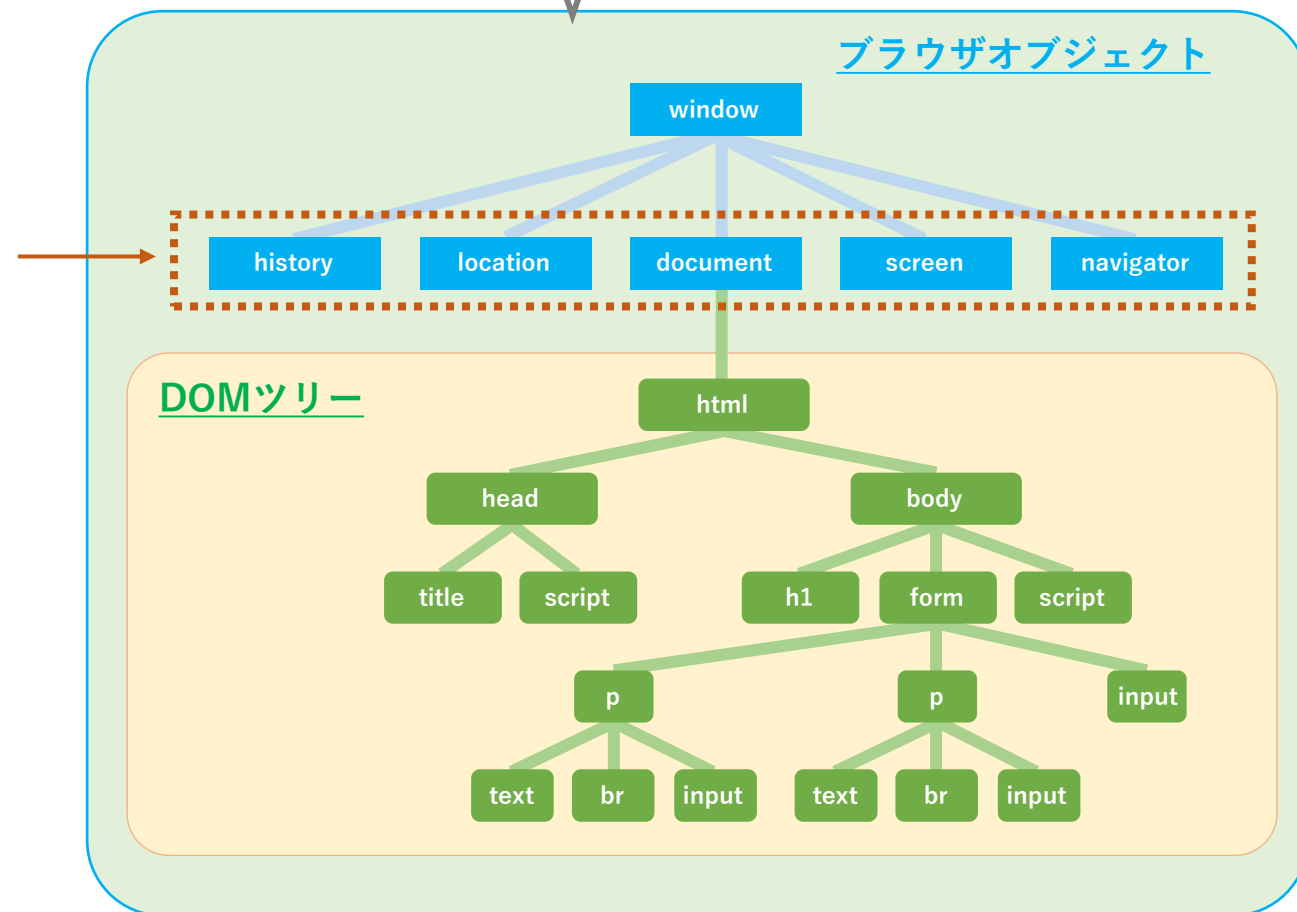
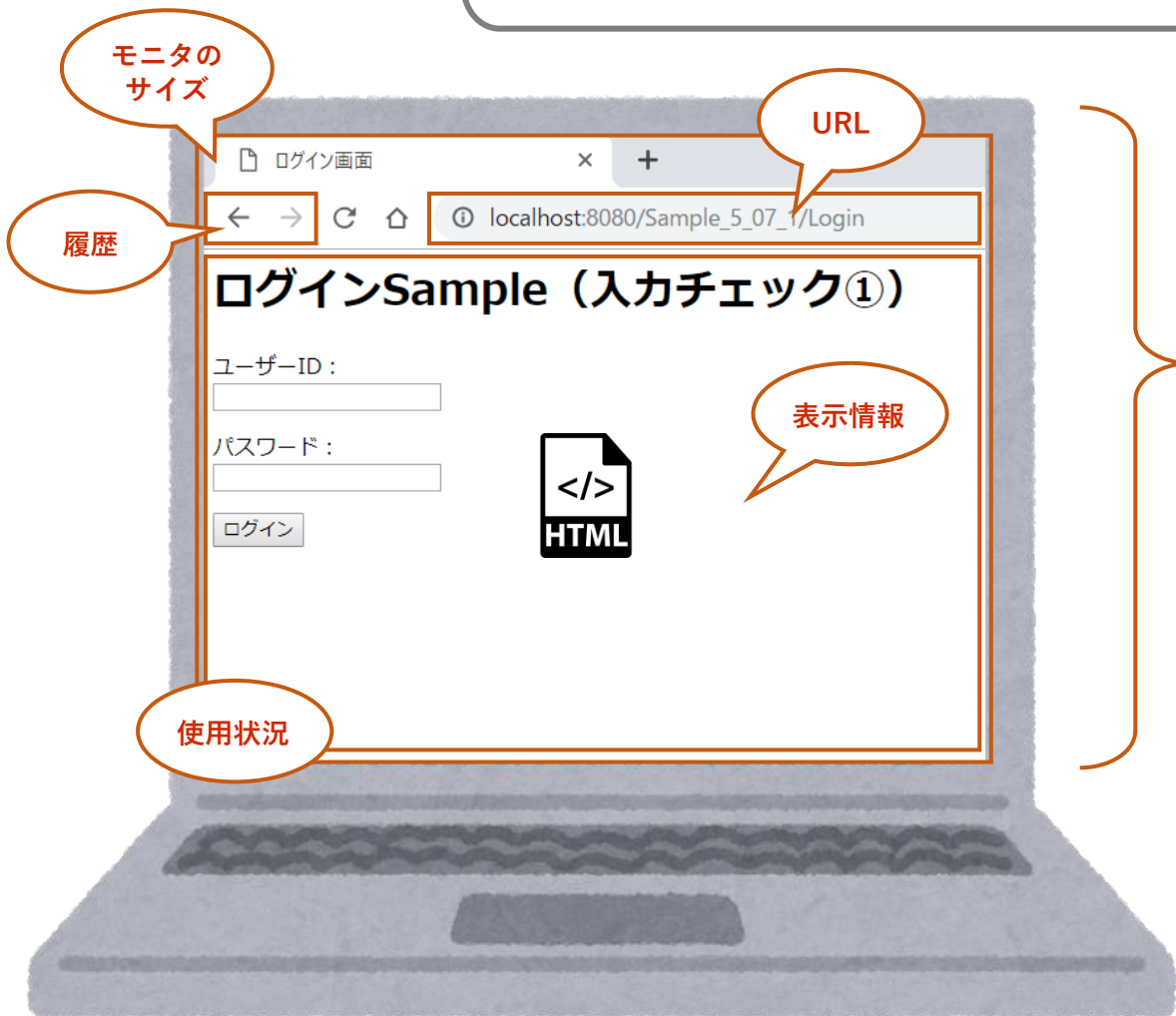
Windowオブジェクトの直下にはブラウザの機能ごとに複数のオブジェクトが存在します。
主要なものとして以下の5つが挙げられます。

history (履歴) / location (URL) / document (表示情報)
screen (モニタのサイズ) / navigator (利用状況)



～ ブラウザオブジェクトとDOMツリー ～

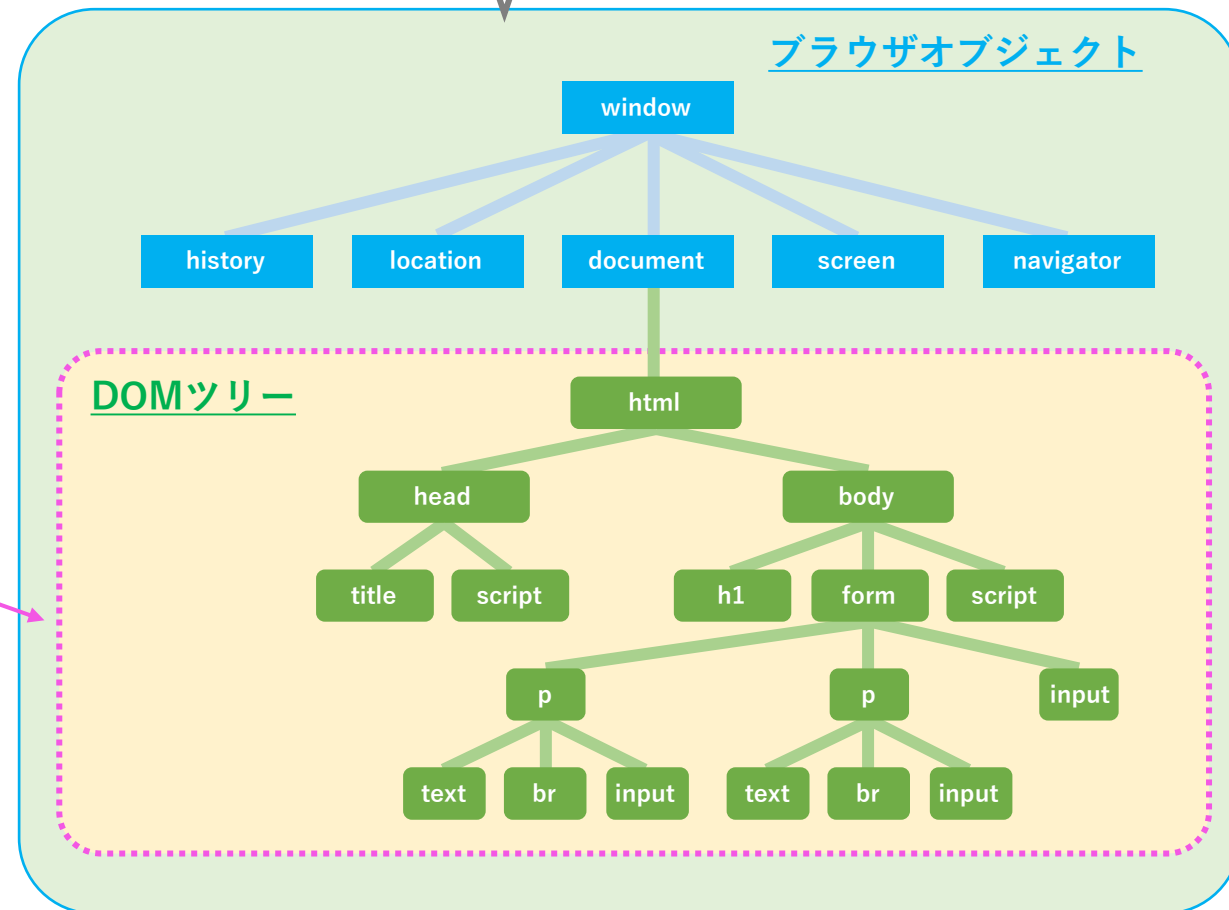
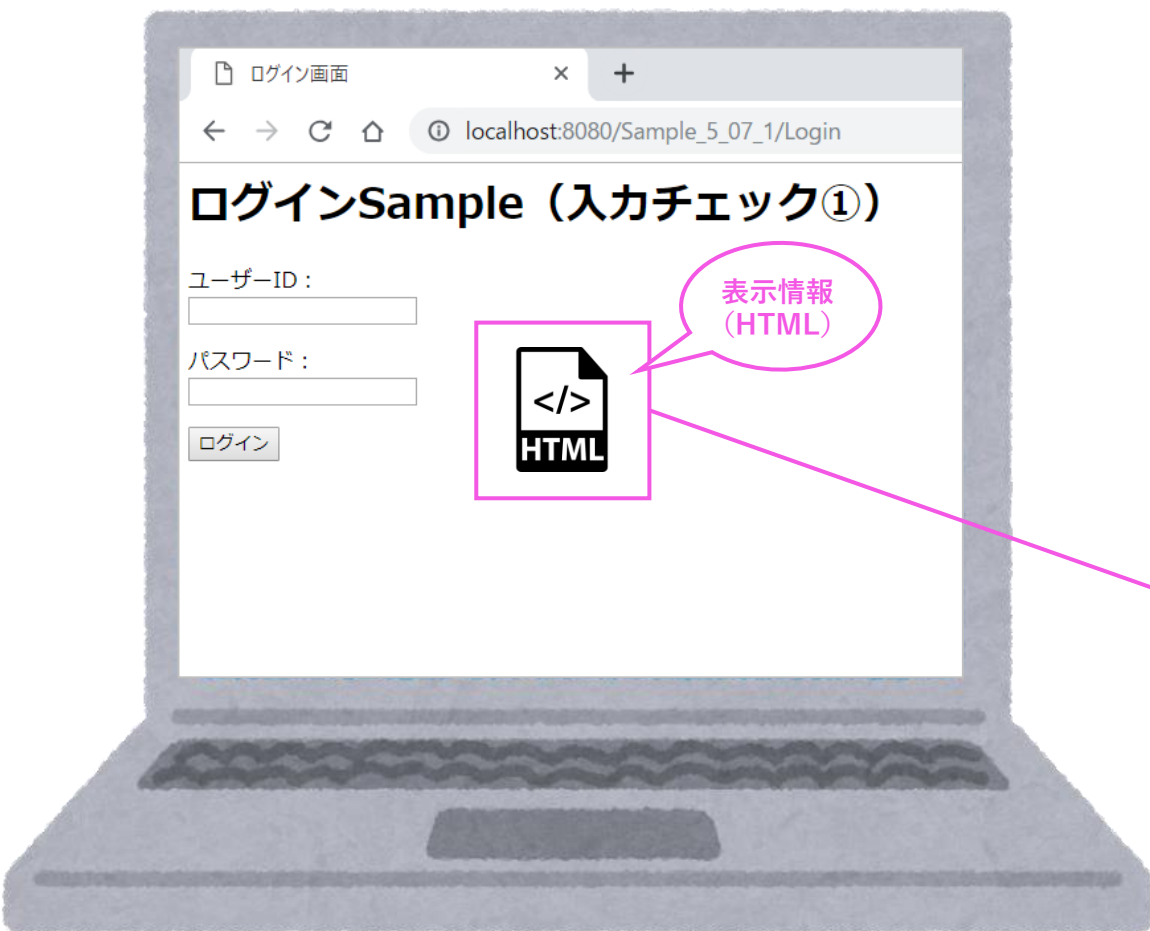
例えば**history**オブジェクトで管理されている**back**メソッドは「前ページに戻る」、つまりブラウザに設置されている戻るボタンを押すのと同じ機能を持ちます。これをアプリケーションから実行する際は **window.history.back()** と書きます。「window.」は省略可能であるため **history.back()** と記述しても構いません。



～ ブラウザオブジェクトとDOMツリー ～

画面の表示内容进行操作すべく、読み込んだHTMLなどの文書情報をオブジェクトの階層構造として再構築したものが**DOMツリー**です。

文書情報の読み込みと同時にブラウザオブジェクトである**documentオブジェクト**の直下に構築されていきます。



～ ブラウザオブジェクトとDOMツリー ～

画面の表示内容进行操作すべく、読み込んだHTMLなどの文書情報をオブジェクトの階層構造として再構築したものが**DOMツリー**です。

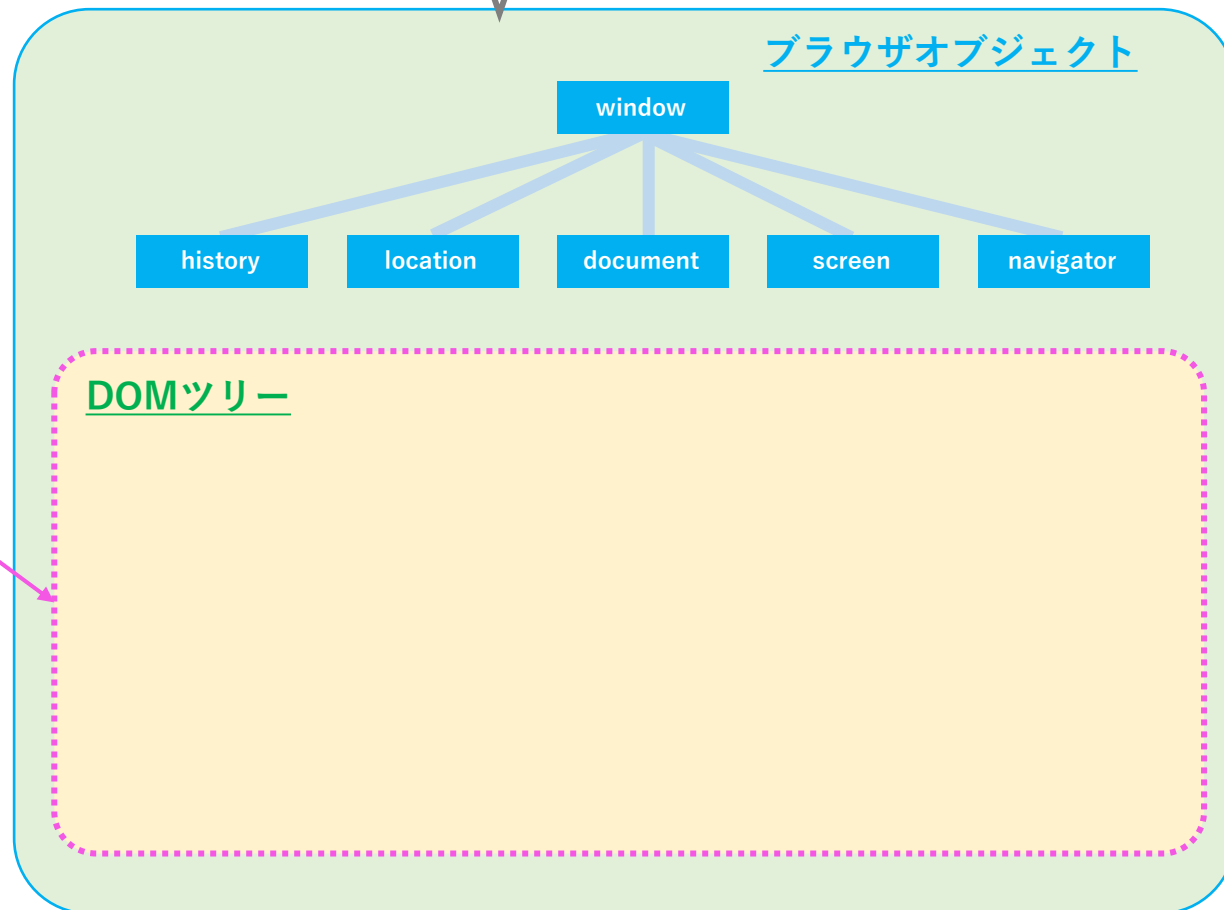
文書情報の読み込みと同時にブラウザオブジェクトである**documentオブジェクト**の直下に構築されていきます。

表示情報
(HTML)

```
<html>
<head>
  <title>ログイン画面</title>
  <script type = "text/javascript">
    console.log( "▼----scriptタグ (headタグ直下) 内での動作確認----▼" );
    console.log( "HTMLは上から順にブラウザに読み込まれていきます。" );
    console.log( "scriptタグ内のJavaScript処理も上から順に実行されます。" );
    console.log( "▼まだID「ID_USER_ID」の要素は読み込まれていないため値を取得できない(エラー)" );
    console.log( document.getElementById("ID_USER_ID").name );
  </script>
</head>
<body>
  <h1>ログインSample (入力チェック①)</h1>
  <form action="ExecuteLogin" method="post">
    <p>ユーザーID :
      <br>
      <input type="text" name="USER_ID" maxlength="20" id="ID_USER_ID">
    </p>
    <p>パスワード : <br>
      <input type="password" name="PASSWORD" maxlength="20" id="ID_PASSWORD">
    </p>
    <input type="submit" value="ログイン" id="ID_SUBMIT" onclick="return preCheck()" >
  </form>
  <script type = "text/javascript">

    console.log( "▼----scriptタグ (bodyタグ直下) 内での動作確認----▼" );

    console.log( " (ローカルスコープである関数preCheck内の処理は初期の読み込みでは実行されない) " );
    function preCheck(){
      var elmUserId   = document.getElementById("ID_USER_ID");
      var elmPassword = document.getElementById("ID_PASSWORD");
      var canSubmit = true;
      if(elmUserId.value == "" || elmPassword.value == ""){
        alert("入力漏れの項目があります。");
        canSubmit = false;
      }
      return canSubmit;
    }
  </script>
</body>
</html>
```



～ ブラウザオブジェクトとDOMツリー ～

画面の表示内容进行操作すべく、読み込んだHTMLなどの文書情報をオブジェクトの階層構造として再構築したものが**DOMツリー**です。

文書情報の読み込みと同時にブラウザオブジェクトである**documentオブジェクト**の直下に構築されていきます。

読み込み
&
DOMツリー構築

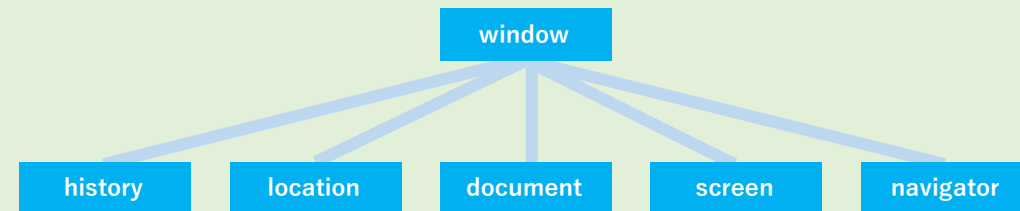
表示情報
(HTML)

```
<html>
<head>
  <title>ログイン画面</title>
  <script type = "text/javascript">
    console.log( "▼----scriptタグ (headタグ直下) 内での動作確認----▼" );
    console.log( "HTMLは上から順にブラウザに読み込まれていきます。" );
    console.log( "scriptタグ内のJavaScript処理も上から順に実行されます。" );
    console.log( "▼まだID「ID_USER_ID」の要素は読み込まれていないため値を取得できない(エラー)" );
    console.log( document.getElementById("ID_USER_ID").name );
  </script>
</head>
<body>
  <h1>ログインSample (入力チェック①)</h1>
  <form action="ExecuteLogin" method="post">
    <p>ユーザーID :
      <br>
      <input type="text" name="USER_ID" maxlength="20" id="ID_USER_ID">
    </p>
    <p>パスワード : <br>
      <input type="password" name="PASSWORD" maxlength="20" id="ID_PASSWORD">
    </p>
    <input type="submit" value="ログイン" id="ID_SUBMIT" onclick="return preCheck()" >
  </form>
  <script type = "text/javascript">

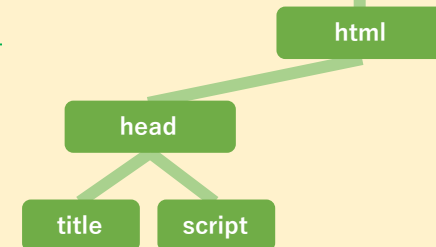
    console.log( "▼----scriptタグ (bodyタグ直下) 内での動作確認----▼" );

    console.log( " (ローカルスコープである関数preCheck内の処理は初期の読み込みでは実行されない) " );
    function preCheck(){
      var elmUserId = document.getElementById("ID_USER_ID");
      var elmPassword = document.getElementById("ID_PASSWORD");
      var canSubmit = true;
      if(elmUserId.value == "" || elmPassword.value == ""){
        alert("入力漏れの項目があります。");
        canSubmit = false;
      }
      return canSubmit;
    }
  </script>
</body>
</html>
```

ブラウザオブジェクト



DOMツリー



～ ブラウザオブジェクトとDOMツリー ～

画面の表示内容进行操作すべく、読み込んだHTMLなどの文書情報をオブジェクトの階層構造として再構築したものが**DOMツリー**です。

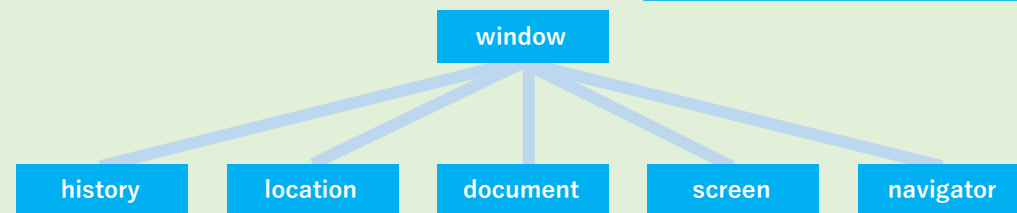
文書情報の読み込みと同時にブラウザオブジェクトである**documentオブジェクト**の直下に構築されていきます。

読み込み
&
DOMツリー構築

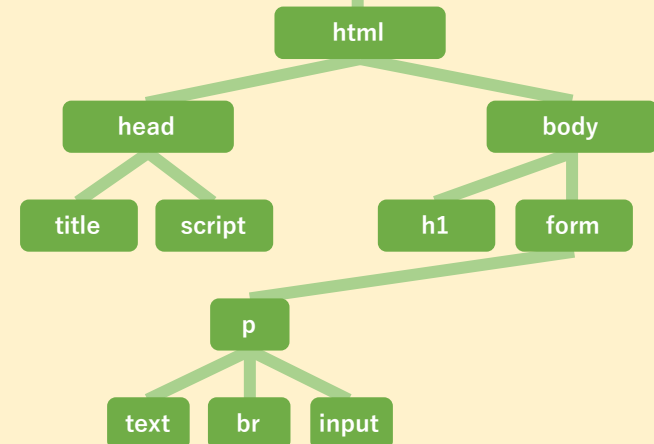
表示情報
(HTML)

```
<html>
<head>
  <title>ログイン画面</title>
  <script type = "text/javascript">
    console.log( "▼----scriptタグ (headタグ直下) 内での動作確認----▼" );
    console.log( "HTMLは上から順にブラウザに読み込まれていきます。" );
    console.log( "scriptタグ内のJavaScript処理も上から順に実行されます。" );
    console.log( "▼まだID「ID_USER_ID」の要素は読み込まれていないため値を取得できない(エラー)" );
    console.log( document.getElementById("ID_USER_ID").name );
  </script>
</head>
<body>
  <h1>ログインSample (入力チェック①)</h1>
  <form action="ExecuteLogin" method="post">
    <p>ユーザーID :
      <br>
      <input type="text" name="USER_ID" maxlength="20" id="ID_USER_ID">
    </p>
    <p>パスワード : <br>
      <input type="password" name="PASSWORD" maxlength="20" id="ID_PASSWORD">
    </p>
    <input type="submit" value="ログイン" id="ID_SUBMIT" onclick="return preCheck()" >
  </form>
  <script type = "text/javascript">
    console.log( "▼----scriptタグ (bodyタグ直下) 内での動作確認----▼" );
    console.log( " (ローカルスコープである関数preCheck内の処理は初期の読み込みでは実行されない) " );
    function preCheck(){
      var elmUserId = document.getElementById("ID_USER_ID");
      var elmPassword = document.getElementById("ID_PASSWORD");
      var canSubmit = true;
      if(elmUserId.value == "" || elmPassword.value == ""){
        alert("入力漏れの項目があります。");
        canSubmit = false;
      }
      return canSubmit;
    }
  </script>
</body>
</html>
```

ブラウザオブジェクト



DOMツリー



～ ブラウザオブジェクトとDOMツリー ～

画面の表示内容进行操作すべく、読み込んだHTMLなどの文書情報をオブジェクトの階層構造として再構築したものが**DOMツリー**です。

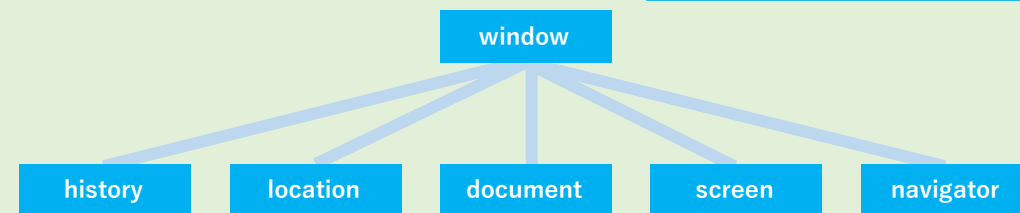
文書情報の読み込みと同時にブラウザオブジェクトである**documentオブジェクト**の直下に構築されていきます。

読み込み
&
DOMツリー構築

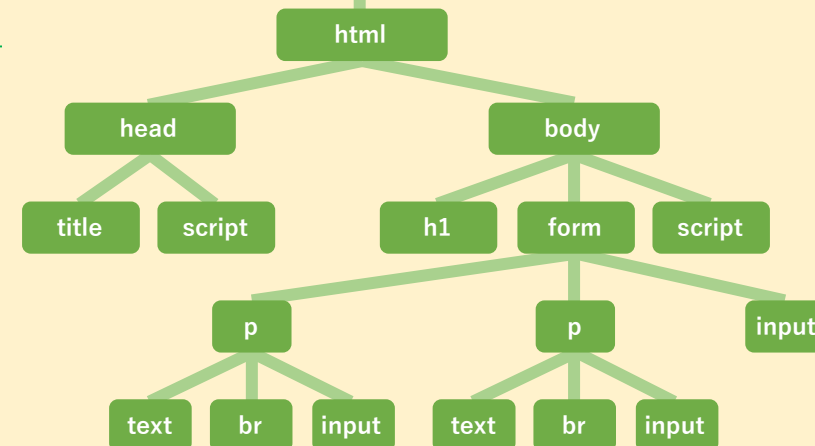
表示情報
(HTML)

```
<html>
<head>
  <title>ログイン画面</title>
  <script type = "text/javascript">
    console.log( "▼----scriptタグ (headタグ直下) 内での動作確認----▼" );
    console.log( "HTMLは上から順にブラウザに読み込まれていきます。" );
    console.log( "scriptタグ内のJavaScript処理も上から順に実行されます。" );
    console.log( "▼まだID「ID_USER_ID」の要素は読み込まれていないため値を取得できない(エラー)" );
    console.log( document.getElementById("ID_USER_ID").name );
  </script>
</head>
<body>
  <h1>ログインSample (入力チェック①)</h1>
  <form action="ExecuteLogin" method="post">
    <p>ユーザーID :
      <br>
      <input type="text" name="USER_ID" maxlength="20" id="ID_USER_ID">
    </p>
    <p>パスワード : <br>
      <input type="password" name="PASSWORD" maxlength="20" id="ID_PASSWORD">
    </p>
    <input type="submit" value="ログイン" id="ID_SUBMIT" onclick="return preCheck()" >
  </form>
  <script type = "text/javascript">
    console.log( "▼----scriptタグ (bodyタグ直下) 内での動作確認----▼" );
    console.log( " (ローカルスコープである関数preCheck内の処理は初期の読み込みでは実行されない) " );
    function preCheck(){
      var elmUserId = document.getElementById("ID_USER_ID");
      var elmPassword = document.getElementById("ID_PASSWORD");
      var canSubmit = true;
      if(elmUserId.value == "" || elmPassword.value == ""){
        alert("入力漏れの項目があります。");
        canSubmit = false;
      }
      return canSubmit;
    }
  </script>
</body>
</html>
```

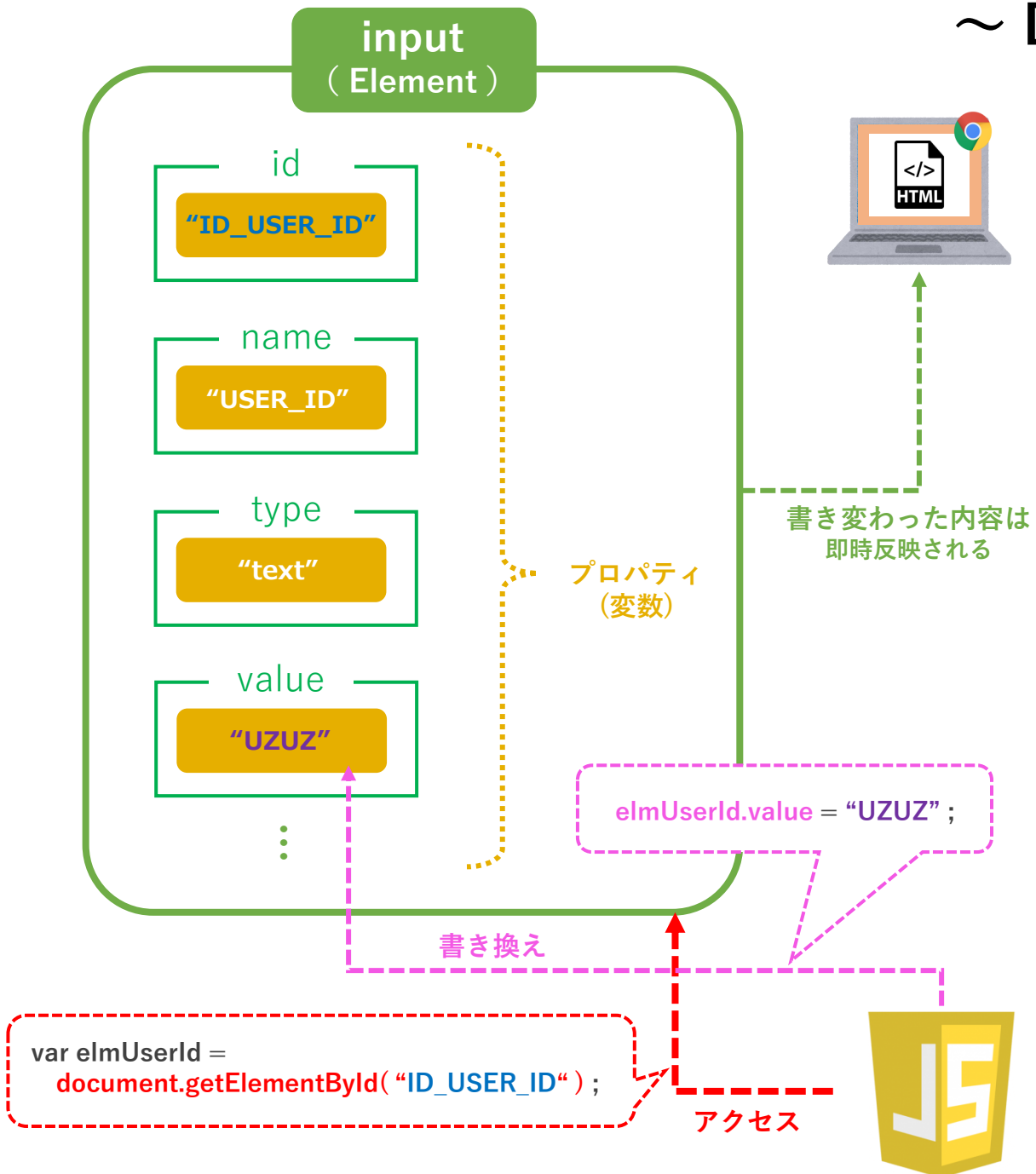
ブラウザオブジェクト



DOMツリー



～ DOM ～



≪ DOMの操作 ≫

□ Elementは文書内のタグの情報をプロパティとして持つオブジェクトです。このプロパティの値を書き換えるとブラウザ上の情報に即時反映されます。

例えばinputタグのプロパティ「value」は入力値を管理する変数ですが、このプロパティに文字列を代入する処理を書いておくと実行時にテキストボックスへと入力されます。

□ Elementのプロパティを書き換えるためにはまずElementにアクセスする必要があります。アクセスする方法はいくつかありますが、まず覚えていただきたいのが**id属性を用いたアクセス法**です。id属性は**Elementを一意に特定するための目印**のようなもので、DOMツリー全体で重複して設定することが許されていません。

以下のように**getElementByIdメソッド**で取得したいElementのid属性を指定すると該当のElementオブジェクトを取得できます。

```
document.getElementById( "取得したいElementのid属性" )
```

このオブジェクトは参照型変数なので、取得したオブジェクト内のプロパティを書き換える = 大元のDOMツリーの内容を書き換える ということになります。

他では例えばname属性を用いたアクセス法などもありますが、**name属性はフォームによるサーバーとの通信で用いるもの**、HTML操作はそれ以外の要素で行う方がベターと思っておくと混乱せずに済むかもしれません。

～ イベントハンドラ ～

ログインSample (入力チェック①)

ユーザーID :

パスワード :

ログイン

ID/パスワード未入力で
ログインボタンが押されると...

検知

イベントハンドラ

実行

ダイアログが
起動される！

localhost:8080 の内容

入力漏れの項目があります。

OK

ログインSample (入力チェック①)

ユーザーID :

パスワード :

ログイン

《 イベントハンドラ 》

□ ブラウザ上では以下のような様々な出来事が起きます。

- ・ ページの読み込みが完了した
- ・ フォームを送信しようとした
- ・ ユーザーがボタンの上にマウスを乗せた
- ・ ユーザーがウィンドウを閉じた
- ・ エラーが発生した

などなど...

こうした出来事のことを**イベント**と言い、イベントを検知した際に実行される処理のことを**イベントハンドラ**と言います。

□ イベントハンドラは以下のようなイベントに対応したプロパティが用意されており、これを検知した後に実行させたい関数オブジェクトを仕込むことで設定完了となります。

- | | |
|----------------------|---------------|
| ・ ページの読み込みが完了した | → onLoad |
| ・ フォームが送信された | → onSubmit |
| ・ ユーザーがボタンの上にマウスを乗せた | → onMouseOver |
| ・ ユーザーがウィンドウを閉じた | → onUnload |
| ・ エラーが発生した | → onError |

～ サンプルを動かしてみよう ～

手順

- (1) eclipse上に新しい動的Webプロジェクトを作成する。
プロジェクト名: **Sample_5_07_2**
- (2) プロジェクトの srcパッケージ直下に workパッケージを作成する。
- (3) 配布のSample_5_07_2フォルダからソースコードを取得して
workパッケージ直下にインポートする。
- (4) 同じく **web.xml**をドライブから取得して置き換える。
WebContent > WEB-INF > web.xml
- (5) サーバーを起動してLogin.javaを起動する。

～ イベントハンドラ ～

イベントハンドラの設定方法

<Sample_5_07_1>

```
<form action="ExecuteLogin" method="post" name="FORM_LOGIN_INFO">
  <p>ユーザーID :
    <br>
    <input type="text" name="USER_ID" maxlength="20" id="ID_USER_ID">
  </p>
  <p>パスワード : <br>
    <input type="password" name="PASSWORD" maxlength="20" id="ID_PASSWORD">
  </p>
  <input type="submit" value="ログイン" id="ID_SUBMIT" onclick="return preCheck()" >
</form>
<script type="text/javascript">
  console.log( "▼----bodyタグ内での動作確認----▼" );
  console.log( "▽getElementById以外の要素の取得方法" );
  console.log( "getElementById      : " + document.getElementById("ID_USER_ID").name );
  console.log( "elements[INDEX]       : " + document.forms[0].elements[0].name );
  console.log( "elements[name属性]    : " + document.forms["FORM_LOGIN_INFO"].elements["" );
  console.log( "name (略記)          : " + document.FORM_LOGIN_INFO.USER_ID.name );

  function preCheck(){
    var elmUserId = document.getElementById("ID_USER_ID");
    var elmPassword = document.getElementById("ID_PASSWORD");
    var canSubmit = true;
    if(elmUserId.value == "" || elmPassword.value == ""){
      alert("入力漏れの項目があります。");
      canSubmit = false;
    }
    return canSubmit;
  }
</script>
```

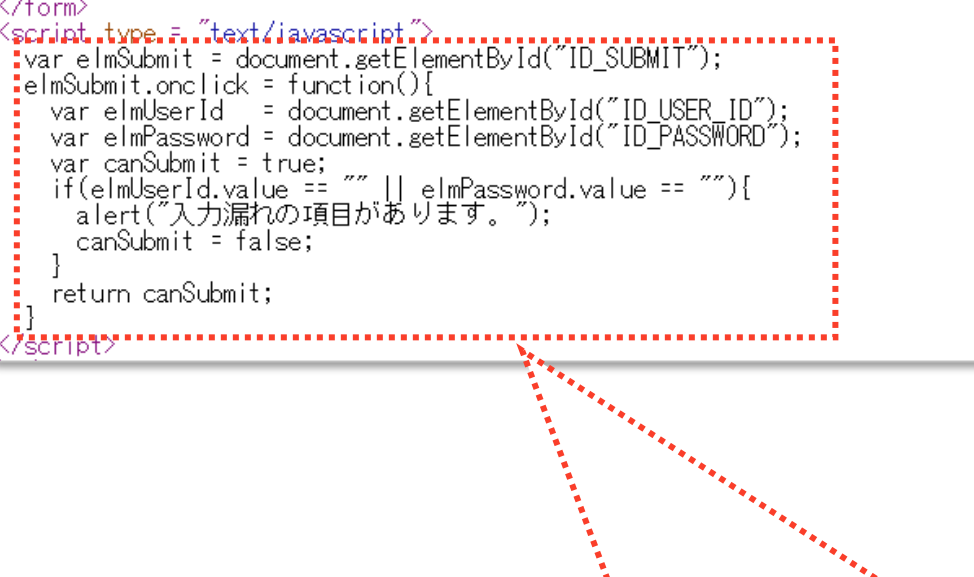


HTMLに直接仕込む（非推奨）

タグ内にonclick要素を追加して関数を仕込む方法。
「return false」を設定することでクリック処理を無効にすることができる。

<Sample_5_07_2>

```
<form action="ExecuteLogin" method="post" name="FORM_LOGIN_INFO">
  <p>ユーザーID :
    <br>
    <input type="text" name="USER_ID" maxlength="20" id="ID_USER_ID">
  </p>
  <p>パスワード : <br>
    <input type="password" name="PASSWORD" maxlength="20" id="ID_PASSWORD">
  </p>
  <input type="submit" value="ログイン" id="ID_SUBMIT" >
</form>
<script type="text/javascript">
  var elmSubmit = document.getElementById("ID_SUBMIT");
  elmSubmit.onclick = function(){
    var elmUserId = document.getElementById("ID_USER_ID");
    var elmPassword = document.getElementById("ID_PASSWORD");
    var canSubmit = true;
    if(elmUserId.value == "" || elmPassword.value == ""){
      alert("入力漏れの項目があります。");
      canSubmit = false;
    }
    return canSubmit;
  }
</script>
```



プロパティに関数オブジェクトを仕込む（推奨）

scriptタグ内でプロパティに無名関数を仕込む方法。
無名関数の戻り値が false ならクリック処理を無効にすることができる。

～ JavaScriptの外部ファイル化 ～

<Sample_5_07_2 (ログイン画面)>

```
<html>
<head>
  <title>ログイン画面</title>
</head>
<body>
  <h1>ログインSample (入力チェック②)</h1>
  <form action="ExecuteLogin" method="post">
    <p>ユーザーID :
      <br>
      <input type="text" name="USER_ID" maxlength="20" id="ID_USER_ID">
    </p>
    <p>パスワード : <br>
      <input type="password" name="PASSWORD" maxlength="20" id="ID_PASSWORD">
    </p>
    <input type="submit" value="ログイン" id="ID_SUBMIT" >
  </form>
  <script type="text/javascript">
    var elmSubmit = document.getElementById("ID_SUBMIT");

    elmSubmit.onclick = function(){
      var elmUserId = document.getElementById("ID_USER_ID");
      var elmPassword = document.getElementById("ID_PASSWORD");
      var canSubmit = true;
      if(elmUserId.value == "" || elmPassword.value == ""){
        alert("入力漏れの項目があります。");
        canSubmit = false;
      }
      return canSubmit;
    }
  </script>
</body>
</html>
```

<Sample_5_07_3 (ログイン画面)>

```
<html>
<head>
  <title>ログイン画面</title>
</head>
<body>
  <h1>ログインSample (入力チェック③)</h1>
  <form action="ExecuteLogin" method="post">
    <p>ユーザーID :
      <br>
      <input type="text" name="USER_ID" maxlength="20" id="ID_USER_ID">
    </p>
    <p>パスワード : <br>
      <input type="password" name="PASSWORD" maxlength="20" id="ID_PASSWORD">
    </p>
    <input type="submit" value="ログイン" id="ID_SUBMIT" >
  </form>
  <script type="text/javascript" src="js/brank-check.js"></script>
</body>
</html>
```

<外部ファイル (brank-check.js)>

```
var elmSubmit = document.getElementById("ID_SUBMIT");
elmSubmit.onclick = function(){
  var elmUserId = document.getElementById("ID_USER_ID");
  var elmPassword = document.getElementById("ID_PASSWORD");
  var canSubmit = true;
  if(elmUserId.value == "" || elmPassword.value == ""){
    alert("入力漏れの項目があります。");
    canSubmit = false;
  }
  return canSubmit;
}
```

～ サンプルを動かしてみよう ～

手順

- (1) eclipse上に新しい動的Webプロジェクトを作成する。
プロジェクト名：Sample_5_07_3
- (2) プロジェクトの srcパッケージ直下に workパッケージを作成する。
- (3) 配布のSample_5_07_3フォルダからソースコードを取得して
workパッケージ直下にインポートする。
- (4) 同じく web.xmlをドライブから取得して置き換える。
WebContent > WEB-INF > web.xml
- (5) プロジェクトのWebContentフォルダ直下に js という名前のフォルダ
を作成してbrank-check.jsを配置する。
- (6) サーバーを起動してLogin.javaを起動する。

【演習】

プロジェクト「Ex_5_06」に
入力チェック機能を追加して
ページに動きのあるサイトへと作り替えましょう！



動きが出てきて
楽しくなってきた♪

【演習】

手順

- (1) eclipse上に新しい動的Webプロジェクトを作成する。
プロジェクト名：Ex_5_07
- (2) プロジェクトの srcパッケージ直下に workパッケージを作成する。
- (3) プロジェクト「 Ex_5_07 」を参考に演習に取り組む。
 - ・ Javaリソース > src > work 直下にjavaソースコードを配備
 - ・ WebContent > WEB-INF 直下にweb.xmlを配備

【演習】

ユーザー名およびパスワードのどちらかでも未入力状態でフォームボタンが押された場合「入力漏れの項目があります。」というメッセージをポップアップで出してフォーム処理を中断する。
※JavaScriptの処理はjsファイルに外部化すること（ファイル名は任意）

演習（ログイン）

ユーザー名：

uzuz001

パスワード：

.....

ログイン

ログアウトしました。

[ログイン画面に戻る](#)

わんちゃん暮らし改善アンケートフォーム

名前：MOCO

年齢：

性別： ☒ オス ☐ メス

満足度： とても満足 ▼

飼い主へのご意見・ご感想をご記入ください：

回答する(SaveSurveyを起動)

[回答一覧画面へ](#)

[ログアウトする](#)

未入力または数値以外の情報が入力された状態でフォームボタンが押された場合「不正な入力項目があります。」というメッセージをポップアップで出してフォーム処理を中断する。
※JavaScriptの処理はjsファイルに外部化すること（ファイル名は任意）

未入力でフォームボタンが押された場合「不正な入力項目があります。」というメッセージをポップアップで出してフォーム処理を中断する。
※JavaScriptの処理はjsファイルに外部化すること（ファイル名は任意）

アンケートにご回答頂きありがとうございました。

or

エラーが発生しました。

[回答画面に戻る](#)

ご意見

わんちゃんをもっとよこせ。

もっと遊んでほしい。

グレース 6 オス 普通 カリカリ以外のたべものを食べてみたい。

[回答画面に戻る](#)