


Webデザイン実習3B



2019/4/22

Kazuma Sekiguchi

class@cieds.jp

現在のフロントエンド

- JSなどでインタラク션을を付与する＝フロントエンド
 - サーバーサイドに対する言い方
- 最近のWebで一番変化の激しい分野
 - HTML + CSS + JS
 - コーダーにJSを付け加えた職種
- インタラクション、SPA (Single Page Application) なども担当になる

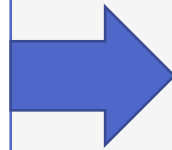
現在のフロントエンド

変化が早い

現在

これが旧来

jQuery+プラグイン
で動きを付与



jQuery否定
React, Angular, Vueなどの
利用にするSPA実装

WebGLなどによる3D対応
ゲームなどのコンテンツ
作成

jQuery否定
NativeJS、AltJSによる実装

変化が遅い

jQuery利用し続け
インタラクション実装

現在のフロントエンド

- JSの利用範囲が広がった
 - 活用できる部分が拡大してきている、ブラウザー側の対応が進んだ、ユーザビリティ向上のためにJSを利用する
 - 3D対応、VRなども範囲に入ってきている
- 適応するためにさまざまな解決策が登場している
 - 過渡期でもあるため、スタンダードが存在しない
 - いろいろな人がいろいろな提案を行っている
 - SPAへの対応が主（jQueryとSPAは相性が悪いいため、jQueryなどの排除に繋がってきている）

キーワード

- SPA (Single Page Application)
 - ページ遷移すること無く、アプリケーションのように操作が可能なWebアプリケーションのこと
 - React : Facebookが提供しているSPA構築のためのライブラリ
 - Angular : Googleが提供しているSPA構築のためのライブラリ
 - Vue.js : オープンソースで開発されているSPA構築のためのライブラリ
- WebGL
 - プラグインを使用せずにインタラクティブな 3D グラフィックスや 2D グラフィックスをレンダリングするための仕組み
 - Unityなどのソフトを用いてモデルを作成するのが一般的
 - 今後拡大が見込まれる
 - Three.js : WebGLベースでは無いが、Webで3Dを扱うためのライブラリ

キーワード

- AltJS

- JSを素で書くと機能が足りないため、別な言語で書いて最終的にJSに変換する言語
 - TypeScript : MSが作成したAltJS言語
 - Dart : Googleが作成したAltJS言語
- Babelという変換ソフトを利用するとまだ実装されていないJSの書き方を利用することも可能になっている

- Node.js

- PC上でJSのアプリを動作させるためのソフト（実際にはChromeのエンジン+ α ）
- AltJSなどを変換するときなどに利用する

キーワード

- HTMLテンプレートエンジン

- HTMLをそのまま書かずに、変換して利用するように記述する
- パーツごとに分けることが可能なため、共通パーツなどを作りやすい
 - Pug：以前はJadeと呼ばれた仕組み。デファクトスタンダード

- スタイルシート言語

- CSSをそのまま書かずに変換して利用するように記述する
- CSSファイルを分割したり、変数を利用できたりするため、現在はCSSはこちらで記述するのが普通
 - Sass/Scss：ほぼデファクトスタンダード
 - PostCSS：CSSにベンダープレフィックスなどを付け加えたりする

キーワード

- タスクランナー
 - AltJSやSCSSなどを変換するための仕組み
 - 保存と同時に自動的に変換したり、ブラウザをリロードしたりする
 - Node上で動作するのが一般的
 - Gulp：良く使われているタスクランナー
 - Grunt：旧来使われていたタスクランナー
 - NPM Script：最近よく使われ始めているタスクランナーというか仕組み。速度が早いとされる

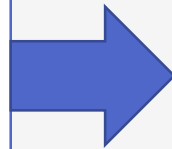
現在のフロントエンド

変化が早い

現在

これが旧来

jQuery+プラグイン
で動きを付与



jQuery否定
React, Angular, Vueなどの
利用にするSPA実装

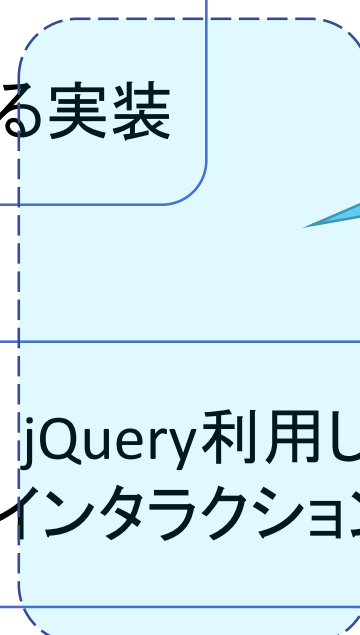
WebGLなどによる3D対応
ゲームなどのコンテンツ
作成

jQuery否定
NativeJS、AltJSによる実装

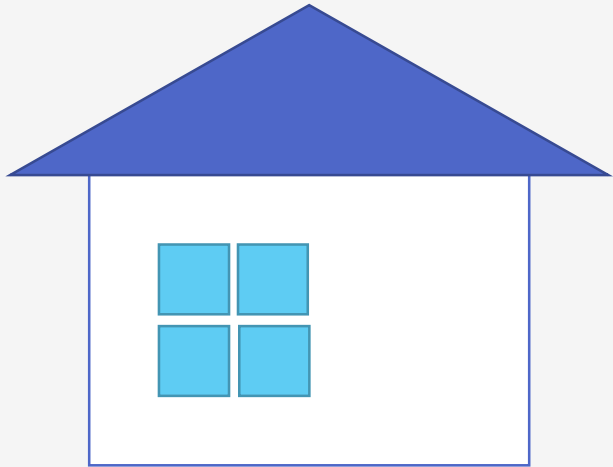
この辺り

変化が遅い

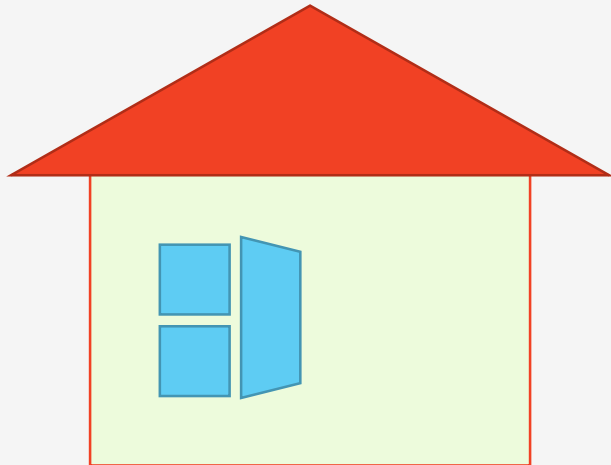
jQuery利用し続け
インタラクション実装



家からプログラミングを考える



- 家は特徴を持つ
 - 屋根：青色
 - 窓があるがしまっている
 - 壁：白これらがプロパティ



- 家を変える
 - 屋根：赤色
 - 窓が開いている
 - 壁：緑
- メソッドを使うことでプロパティを変える→色が変わったりする
 - メソッドに無いものは使えない

ブラウザーの仕事



- ブラウザーが表示するのはHTMLデータなど
 - HTMLはファイルじゃ無くてデータでも良い
- サーバーからデータを送ってきてそれを展開するのが本来のブラウザーの仕事

クライアント側での処理

- ブラウザーはHTMLとCSSを表示
 - ユーザーの操作に合わせて表示を変える、HTML、CSSだけでは表現できない、利用できないものがどうしても発生してきた
 - CSSを動的に書き換えることで、アニメーションを実現する
 - HTMLを動的に書き換えることで、マウスオーバー時に画像を変える
 - ユーザーの動きにあわせてCSSを書き換えて、インタラクションを行う
- これらはCSSだけでは不可能なことが多い

プログラミング言語

- ブラウザー側で利用できるプログラミング言語は JavaScript (JS)
 - 間違えてもJavaとは略さないこと（別な言語になる）
- JavaScriptを利用することで、ブラウザー上でさまざまなことが実現することが可能
 - HTMLの書き換え、CSSの書き換え、ブラウザー上でユーザーが実施した操作の取得などなど
 - 最近ではJSを利用して、デスクトップアプリやスマホアプリを作成することも可能
 - 極めて汎用性が高いが、習得が少し難しい言語

プログラミング言語とは

- 何らかのデータを与えると処理をして、結果を返す
 - この集合体（だけじゃないけど）
- 例：自販機
 - お金を投入→金額と偽物じゃ無いか確認→問題ない
 - 商品選択→売り切れじゃ無いか確認→問題ない
 - 商品落としまーす
- いくつかの処理を組み合わせて目的を達成する
 - 重要なのは、データという概念、処理という概念

プログラミング言語とは

- JavaScript

- プロトタイプ指向のオブジェクト指向言語（覚えなくて良い）
- ブラウザー上で動作して、HTMLやCSSを変化させて表示するための処理を行うことに特化したプログラミング言語
- 書き方が結構独特（他の言語を知らなければ気にならない）
- 複雑なことをしようとすると、書き方が途端に煩雑になる
- 参考、サンプルが非常に豊富。探せば直ぐに求めているものが見つかる可能性が高い
- 結果が直ぐ分かる

プログラミング言語とは

- 1文字でも間違えると動作しなくなる
 - HTMLやCSSは間違えても表示はされる
 - JSは1文字でも間違えると動作しない、最悪ブラウザが真っ白になる
 - どのプログラミング言語も同じではあるが、入力時注意
 - 打ち間違いはどうしても出るが、できるだけ減らす工夫をする
 - フォントを変更したり、文字を大きくしたり、誤字と思われるところにアンダーラインが引かれるようなエディタを試してみる
- 1度では大体動作しない
 - 慣れてても何度書いても間違いは無くならない
 - エラーがあったときに自分で回復する能力を

JavaScriptを書いてみる

- `<script></script>`の間に記述する
 - 別ファイルに記述してもOK
 - 別ファイルの場合は、拡張子をjsとして`<script src= “〇〇〇.js” ></script>`として読み込ませる
- 実行されるタイミングは基本的にブラウザでHTMLが表示された後

JSの基本

- CSSとあまり考え方は変わらない

```
.text{  
  color:#F00;  
}
```

- class= “text” という要素の色を赤色にしろ
- JSの場合

```
var elem = document.getElementsByClassName('text');  
elem[0].style.color="#F00";
```

JSの基本

- 変えたいものに対して、命令を出す
 - またはどのような状態になっているかを取得する
 - 設定と取得の繰り返しが主にやること
- 取得したものの状態に合わせて設定を変える

プログラミング言語

- JSでは変数、メソッド、イベントなどが用意されている
 - 変数＝一時的に保存しておくための場所。名前を付けることでいろいろな内容を保持可能
 - メソッド＝JavaScriptで利用することができる機能のこと。1つ1つでは大したことはできない
 - 組み合わせることで処理をしていくことが可能
 - イベント＝ユーザーのインタラクション
 - 後日説明

JSの基本的なもの

- 変数

- 何かの値やオブジェクトを格納しておくもの
- JSの場合、いろいろなものを格納可能
 - オブジェクトも格納可能
- 先頭にvarを付ける（変数宣言）
 - スペースを開けて変数名を指定（通常1文字目は英小文字）
- 数値と文字が違うので注意（" " で括れば文字）



```
var a = "JavaScript";
```

//aに"JavaScript"という文字列を格納

```
var b = 123;
```

//bに123という数値を格納

変数

- 定義関数を格納できるところはJS独特
- 通常は変数宣言を行って利用する（宣言しなくてもOKだが、最近では警告が出ることが多い）
 - 先頭にvarを付ける（変数宣言）

var a;//aという変数を利用する(宣言のみ)

var b = 1;//bという変数を利用すると宣言し、1という数値を格納

var c = "js">//cという変数を宣言し、jsという文字列を格納

a = 4;//宣言済みの変数aに4を格納

変数宣言

- プログラムの一番最初にまとめて記述することが多い
 - ほかに人が見たときにどういう変数を利用しているか把握しやすい
 - 変数に設定値（リンク先のURLや何らかのパラメータ）などを格納した場合、変更しやすい

JSの基本的なもの

- 何らかの動作をまとめたもの＝関数
- 引数は複数の値を指定可能。返り値は1つだけ

```
function 関数名(引数){  
  何らかの処理処理  
  return 返り値  
}
```

```
function keisan(a,b){  
    var c = a + b;  
    return c;  
}
```



```
var kotae = keisan(2,3);  
kotaeには5が入ってくる
```


ユーザ定義関数

- 何らかの処理をまとめたもの
- 同じような処理を繰り返す際に利用する
 - イチイチ同じ処理を何度も書くのは面倒。間違えも増える
- ()内に引数を指定可能
 - 引数はカンマで区切ることで複数指定可能
 - 引数に値を与えて関数を呼び出せば、その引数を利用して処理してくれる
- ユーザ定義関数から更にほかのユーザ定義関数を呼び出すことも可能
- 記述場所は自由
 - 通常まとめて書いておく
 - 関数だけのJSファイルを作ることも多い

ユーザ定義関数

```
function 関数名(引数1,引数2・・・){  
  処理の内容  
  return 処理の結果  
}
```

基本的な記述方法

```
function Calc(a,b){  
  var c = a + b;  
  return c;  
}
```

作成(Calcという関数を作成)

```
var g = Calc(4,6); //gには10が格納される  
function Calc(a,b){  
  var c = a + b;  
  return c;  
}
```

Calcを利用

```
function caution(){  
  alert("注意！！");  
}
```

returnの無いものも可能

変数宣言の場所による違い

- ユーザ定義関数内で宣言した変数
 - その関数内でのみ利用可能＝ローカル変数
- 関数外で宣言した変数
 - ユーザ定義関数内でも利用可能＝グローバル変数
- グローバル変数の場合、意図しないところで上書きすることがあるため、できるだけ避ける
 - 無理なことも多い

```
var a = 1;  
function showAlert(){  
    alert(a);//1が表示される  
}
```

```
function SetA(){  
    var a = 1;  
}  
function showAlert(){  
    alert(a);//何も表示されない  
}
```

変数宣言の場所による違い

```
var a = 1;
function showAlert(){
    alert(a); // 1が表示される
}
```

グローバル変数としてaを利用

```
function SetA(){
    var a = 1;
}
function showAlert(){
    alert(a); // 何も表示されない
}
```

aはローカル変数のため、ShowAlertでは利用できない

```
var a; // グローバル変数としてaを宣言
SetA(); // 関数呼び出し
showAlert(); // 関数呼び出し
function SetA(){
    a = 4;
}
function showAlert(){
    alert(a); // 4が表示される
}
```

aはグローバル変数。ほかの関数で変更されるとそのまま変数の中身も変化する

条件

- 条件に応じて、処理を分けるケースは多い
 - クリックされてもこれ以上右に動かさない→左に動かす
 - いわゆる判断をさせる場合に利用
 - 条件式を記述して判断させる
 - ifのみ必須
 - 他は任意
 - else ifは複数回利用可能

```
if(条件式1){  
    条件式1に当てはまるならここが動作  
}  
else if(条件式2){  
    条件式2に当てはまるならここが動作  
}  
else{  
    条件式に当てはまらない場合に動作  
}
```

条件式

- 0以外の数、true , 何らかの文字列は全て条件式で当てはまる
 - 0,falseだと条件式に当てはまらない
- 比較して判断すること
 - `if(a > 3)` → aが3よりも大きければtrueになる
 - `if(a == 3)` → aが3のときだけtrueになる
 - `if((a >= 3) && (a < 100))` → aが3以上で、100未満のときだけtrueになる
 - `if((a >= 100) || (a < 10))` → aが100以上または10未満のときだけtrueになる

条件式の成立

- 成立する場合

- `if(true)`
- `if(1以上の数値)`
- `if('何らかの文字')`
- 条件式が成立しない場合

- 不成立の場合

- `if(false)`
- `if(null)`
- `if(undefined)`
- `if(0)`
- 条件式が成立しない場合

ループ

- 同じ処理を繰り返す際に利用
 - いくつか方法がある
 - プログラムは楽するために利用する＝同じ処理はできるだけループを使って処理する
 - 条件が成立している限りループする
 - 初期値に0を入れて1回回るごとに1を足していくと100回回った段階で条件を満たさなくなる

```
for(var i =0; i< 100; i++){  
    var a = i;  
    document.write(a);//aを表示。0～99まで  
    表示される  
}
```

```
for(初期値; 条件; カウントアップ){  
    繰り返す内容  
}
```