フロントエンド・フレームワーク戦国時代の終焉から振り返るJavaScriptとフロントエンドの歴史

吉田 裕美 EY-Office

目次

自己紹介

- 吉田 裕美 (Yoshida Yuumi)
- 有限会社 EY-Office https://www.ey-office.com/
- Twitter: @yuumi3

過去

- 大学の卒業研究でLisp処理系に関わる
- 組込系の会社に就職
- CADのベンチャー企業に転職
  - CADのコア部分の開発
  - シリコンバレーを訪問したり、米国人と仕事を経験

自己紹介

２０世紀が終わるので独立

- カリフォルニアの青い空をみた
- あまり会社に馴染めない人だった
- ネットの時代になった

自己紹介

2000年のEY-Officeを設立

- 有限会社 EY-Office https://www.ey-office.com/
- 開発
  - Web (React, Apollo GraphQL, Ruby on Rails ...)
  - iOS
- 教育
  - React
  - Ruby on Rails
  - iOS

活動

- 書籍：「作りながら学ぶReact」
  - もう古いので買ってはいけません😅
- 登壇
  - RubyKaigi 2008
  - Qcon Tokyo 2009
  - Ruby World Conference 2013
- ...

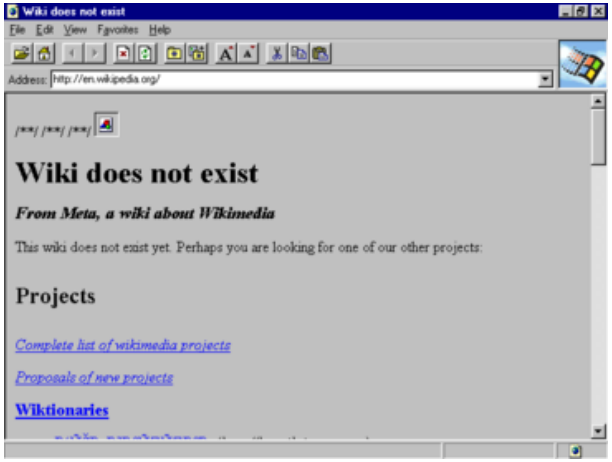
JavaScriptの誕生

かつてのマイクロソフトは怖い会社だった

第一次ブラウザー戦争

Netscape vs Internet Explorer(IE)

- Netscape 1.0 1994-10-15
- IE 1.0 1995-09-16
  （Windows95とは別売）
- バージョンアップ毎に新しいタグが追加される
- しかもブラウザー毎の互換性は？
- 最初はNetscapeが圧倒的に強かった
- 最終的にはIEが勝利



JavaScript

- Netscape 2.0 1995-09-18で登場
- 作者: Brendan Eich
- 当時**Java**が大注目されていたので**JavaScript**という名前になった
- 他ブラウザー(IE, Opera)もサポート

JavaScript言語

- ブラウザー内で動く
- C(Java)言語に似た文法
- 動的言語
- プロトタイプベースのオブジェクト指向
- 関数型言語から影響されている
- サンドボックス
- 便利なオブジェクト型データ（HashMapのような、インスタンスのような）

MSのJavaScript対応

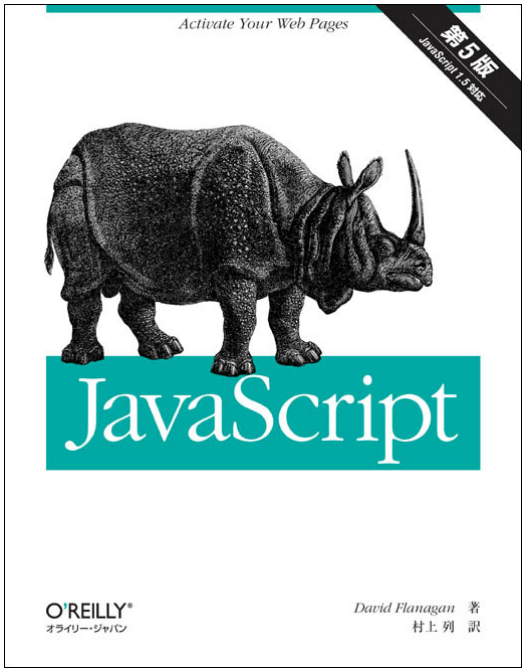
IE3.0 (1996-08-13)にJavaScriptが実装された。しかし！

- JScript
  - MS製JavaScript
  - Netscapeと互換ではなかった
- VBScript
  - Visual Basic風スクリプト言語も使えた
- Active X
  - マイクロソフト独自のコンポーネント技術
  - ブラウザー上でExcel等をインストールせずに実行できる
- 同じころVisual J++という独自Javaを作り問題となる

JavaScriptの標準化

MS製JavaScriptがデファクトスタンダードになってしまうのでは !?

- 標準仕様を作りEcma Internationalに管理してもらうことにした
  - 以前はECMA(European Computer Manufacturers Association)と呼ばれていた
- 標準仕様は**ECMAScript**と呼ばれた、略して**ES**
- 2015年以前は、**ES5**が良く使われた

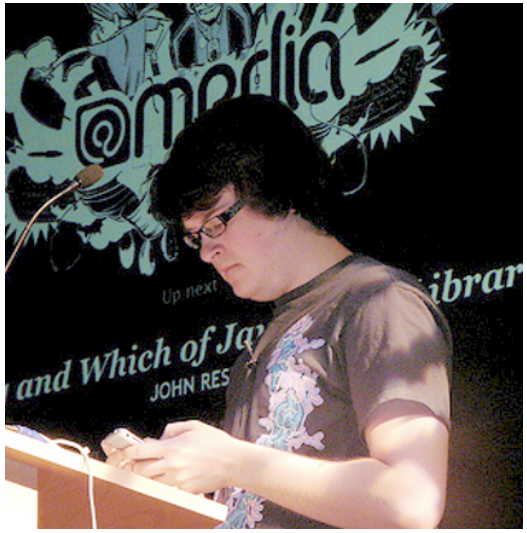


Ajax時代の覇者jQuery

jQuery

write less, do more

- 2006-08-26 リリース
- 作者： John Resig
- 16年経った現在でも使われている
- とても良い設計のライブラリー
    - 著書 JavaScript Ninjaの極意



jQueryの役割

- どのブラウザーでも動く共通API
- CSSセレクターのサポート
- 判りやすいDOM操作関数、イベント・ハンドラー
- Ajax
- 便利関数
- ...

jQueryはオワコン？

最近、終わった技術、オワコン、脱jQueryなど言われています
（最終リリース 2021-10-07）

- ~~どのブラウザーでも動く~~ → IEサポート終了で規格統一
- ~~CSSセレクターのサポート~~ → `Document.querySelector()`
- 判りやすいDOM操作関数、イベント・ハンドラー
- ~~Ajax~~ → `fetch()` API
- ~~便利関数~~ → ES6(?)
- ...

jQueryは全て上手く行ったわけではない

- ◎ jQuery
- △ jQuery UI ： UIコンポーネント
- X jQuery Mobile ： Mobile向けUIライブラリー
- ○ Sizzle ： CSSセレクターライブラリー
- ○ QUnit ： テスティング・フレームワーク

C10K問題、Node.jsの誕生

Node.jsはどちらかというとバックエンドですが、モジュールの仕様や開発ツールとしてフロントエンドにも大きな影響をあたえています。

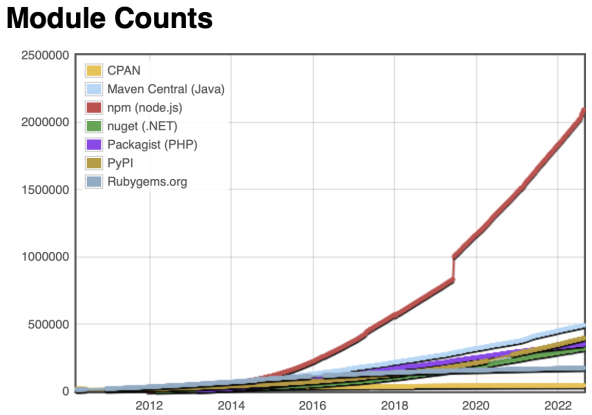
Node.js

- 2009-05-27リリース
- 作者： Ryan Dahl
- PC、サーバー等で動くJavaScript処理系
  - バックエンド・サーバー
  - フロントエンド開発ツール

Node.js

特徴

- ファイルIO、プロセス管理、ネットワーク等をサポート
- マルチプラットフォーム
  - Linux, Mac, Windows
- モジュールをサポート
  - npm パッケージ管理システム
  - 大量のライブラリー Module Counts
- Chrome V8 (JavaScript engine) を利用し高速



C10K問題

Webサーバソフトウェアとクライアントの通信において、クライアントが約1万台に達すると、Webサーバーのハードウェア性能に余裕があるにも関わらず、レスポンス性能が大きく下がる問題。
https://ja.wikipedia.org/wiki/C10K問題

- Apache1 : プロセス
  - プロセスは大量のリソースを消費する
- Apache2 : スレッド
  - プロセスに比べ必要なリソースは少ないが0ではない
- Nginx : 非同期処理
  - IO、通信等で処理がブロックされない

C10K問題

同期処理

```
const fs = require('fs');
let text = fs.readFileSync("sample.txt");
console.log(text);
```

非同期処理

```
const fs = require('fs');
fs.readFile("sample.txt",
  function(err, text) {  // ファイルの読み込終了時に呼び出される
    console.log(text);   // エラーは無視😅
  }
);
console.log("直ぐに次の仕事できます😄");
```

非同期処理の闇

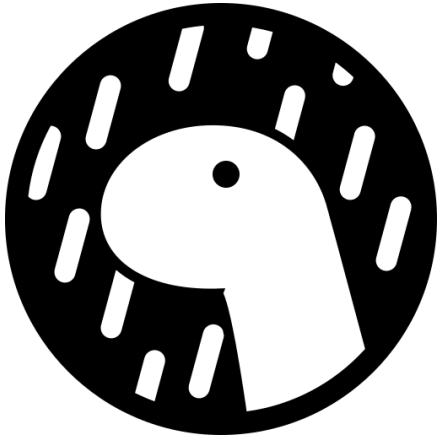
Callback Hell 😱

```
const fs = require('fs');
fs.readFile("sample1.txt",
  function(err, text1) {
    if (err) {
      // エラー処理
    } else {
      fs.readFile("sample2.txt",
        function(err, text2) {
          if (err) {
            // エラー処理
        } else {
          console.log(text1 + text2);
        }
    });
  });
);
```

Deno

Node.js作者Ryan Dahlが新たに開発した
JavaScript処理系

- Node.jsに関する10の反省点(YouTube)
- パッケージ管理システムを内蔵、脱npm
- APIの再設計
- TypeScriptのサポート
- 高速化
- ...



JavaScriptの停滞とAltJS

黒歴史（？）

JavaScriptの停滞

バージョン	仕様公開日
1	1997-06
2	1998-06
3	1999-12
4	放棄
5	2009-12
2015	2015-06
…	…

OSSの仕様決定プロセス

1. 委員会方式
    - Java, C, JavaScript, ...
2. 優しい終身の独裁者
    - Linux, Ruby, Python(?) ...

AltJS

- JavaScriptに取って代わるべく開発された言語
- 実行時にはJavaScriptに変換されて動く
- ES6リリース以前は盛り上がっていた

代表的なAltJS

- Dart： Googleが開発Flutterでも採用
- CoffeeScript ： Ruby on Railsで正式採用された
- Haxe： ActionScript(Flash)の代替として開発された
- ~~TypeScript 後で述べます~~

満を持して登場したES6

ようこそモダンな世界へ

満を持して登場したES6

- 問題点の改良
- モダンな言語機能の導入
- **ES5と上位互換**
- モジュールの導入
- 言語仕様名のルールを変更
  - ES2015のようにリリース年を使う
  - その時点で確定した仕様を毎年リリース
  - ES6は長年使われてきたので今でも使われる
  - ES6はES2015以降の全ESを指すこともある
- ES2016以降は、大きな文法変更ない（今のところ）

ES6での主な変更点 ： クラス定義

ES5

```
var Jyanken = (function() {
  function Jyanken(hand) {
    this.hand = hand;
  }
  jyanken.prototype.poi = function() {
    return this.hand = Math.floor(Math.random() * 3);
  };
  jyanken.prototype.judge = function(your) {
    var judgment = (computer_hand - human_hand + 3) % 3;
    if (judgment == 0) {
      return "引き分け";
    } else if (judgment == 1) {
      return "勝ち";
    } else {
      return "負け";
    }
  };
  return Jyanken;
})();
```

ES6での主な変更点 ： クラス定義

ES6

```
class Jyanken {
  constructor(hand) {
    this.hand = hand
  }
  poi() {
    this.hand = Math.floor(Math.random() * 3)
  }
  judge(your) {
    const judgment = (computer_hand - human_hand + 3) % 3
    if (judgment == 0) {
      return "引き分け"
    } else if (judgment == 1) {
      return "勝ち"
    } else {
      return "負け"
    }
  }
}
```

ただし、クラスベースのオブジェクト指向言語になったわけではなく、いぜんプロトタイプベースという点はかわりません。

ES6での主な変更点 ： アロー関数

```
function square(x) {
  return x * x;
}
```

上が従来の関数定義でしたが、ES6では以下のようにコンパクトなコードになります。

```
const square = (x) => { return x * x }
```

関数本体が1つの式で書ける場合は以下のように、さらにコンパクトに書けます。

```
const square = (x) =>  x * x
```

ES6での主な変更点 ： 変数定義

従来は変数の宣言には `var` がありましたが、ES6では **let** と **const** が追加されました。

```
const pi = 3.14
pi = 3        ← ここでエラーが発生します。
```

let は var と似ていますが、同じ変数名を再定義できない、スコープがvarより狭ので、letを使うことをお勧めします。

```
let x = 3
x = 4
let x = 4  ← ここでエラーが発生します。

if (true) { var y = 3 }
console.log(y)    ← 3が表示される

if (true) { let z = 4 }
console.log(z)    ← ここでエラーが発生します。
```

ES6での主な変更点 ： 分割代入（Destructuring）

配列やオブジェクトの要素の値を変数に代入できます。とくにオブジェクトの要素を変数に代入するのはReactのコードで良く使われます。

```
let [a, b] = [1, 2]
console.log(a)    ← 1が表示される
console.log(b)    ← 2が表示される

let o1 = {x:1.1, y:2.2, z:3.3}
let {x, y} = o1
console.log(x)    ← 1.1が表示される
console.log(y)    ← 2.2が表示される
```

ES6での主な変更点 ： テンプレート文字列

```
const x = "comment"
console.log( "<---" + x + "--->")
```

文字列に値を埋め込むには文字列の連結を使っていましたが、テンプレート文字列を使うことでより自然に書けます。

```
const x = "comment"
console.log( `<---${x}--->`)
```

また、改行を含む文字列を表現できます。

```
console.log( `2階
1階`)
```

ES6での主な変更点 ： async/await

ES5

```
fs.readFile("sample.txt",
  function(err, text) {
    console.log(text);
  }
);
console.log("直ぐに次の仕事できます😄");
```

ES6(ES2017)

```
const f = async () => {
  const text = await fs.readFile("sample.txt");
  console.log("ファイル読み込み完了😄");
  console.log(text);
}
```

縁の下の力持たち

モダンJavaScriptをささえるツールたち

全てNode.js上で動きます。

Babel

全てのブラウザーがES6をサポートするまでは、ES6は使えないのでは？

そこでBabel

- ES6をES5に変換するツール
- トランスコンパイラ、トランスパイラ、コンパイラなどとも呼ばれる
- プラグインで他の変換も出来る
- ES仕様検討時から使えたりする

ESLint

JavaScriptの静的コード分析ツール、実行前にコードの問題点を指摘してくれるツール

- JavaScriptは柔軟な動的言語なので実行するまでエラーが検出しずらい
- チェックルールをカスタマイズ/設定できる

webpack

JavaScriptのモジュールを結合して1つのJavaScriptファイルにまとまるツール

- Bable,ESLintやその他のツールを起動できるので開発環境のまとめ役になる
- 開発用サーバー機能もある

ころでJavaScriptのモジュール機能ってどうなってるの？

1. Node.jsのCommonJS
2. ES6で導入されたES Modules

現在、CommonJSからES Modulesに移行しつつありますが、CommonJSで書かれた膨大なnpmライブラリーがあるので完全な移行には時間がかかると思います。

CommonJS

Node.jsで導入されたモジュール・システム

```
// add.js
function add(x, y) {
  return x + y;
}

module.exports = add;
```

module.exportsで関数や変数、クラス等を公開。

```
// index.js
var add = require('./add');

console.log(add(2,3));
```

requireでモジュールを読み込むことで、他モジュールの関数や変数、クラス等を使用する。

ブラウザーでのCommonJS

ブラウザーにはrequire機能はありません。 そこで、webpack等の開発ツールでJSファイルを読み込み、requireがあればそのファイルを読み込み1つのJSファイルに結合したJSファイルを作成します。以前はBrowserifyツールも使われていました。

```
// webpackで変換された、前ページのコード
//    説明用に一部変更しています

var r = function(e, n) {
  return e + n;
};

console.log(r(2, 3));
```

ES Modules

ES6で`import`, `export`等のモジュール用の機能が導入され、importでモジュールの読み込みが行われます。Viteなどの先進的ツールを使うとES Modulesを使い爆速の開発環境を構築できます。

```
// add.mjs
function add(x, y) {
  return x + y;
}
export default add;
```

```
// index.mjs
import add from './math.mjs';
console.log(add(2,3));
```

ただし、現状はCommonJSで作られたライブラリーが大多数であるため、Babel等でimport/exportはCommonJSに置き換えられwebpackでバンドルされます。

天才言語処理系制作者が作った TypeScript

TypeScript

- JavaScriptに型宣言を導入した言語
- Turbo Pascal, Delphi, C#などを開発したAnders Hejlsbergも開発に関わっている
- 最終的にはJavaScriptに変換され実行される
- VSCodeとも連携しエディター上でエラーチェックできる
- JavaScriptからの移行が容易

TypeScript

- 最近の開発ではTypeScriptを使う事が当たり前になった（当社調べ）
- JavaScriptで発生するバグで一番多いのはtypo（当社調べ）
- ESLintでは発見できないバグも多い
- 型宣言はプログラムの理解を助ける
  - 関数の引数
  - オブジェクトの定義
- 型推論があるので全てに型宣言を書く必要はない
- 不明なデータはとりあえず `any` 型で始められる
- 大抵のツール、ライブラリーがTypeScriptをサポートしている

TypeScript

JavaScript

```
const person1 = {name: '山田', age: '28'};
console.log(person1.mame);
```

バグわかりますか？

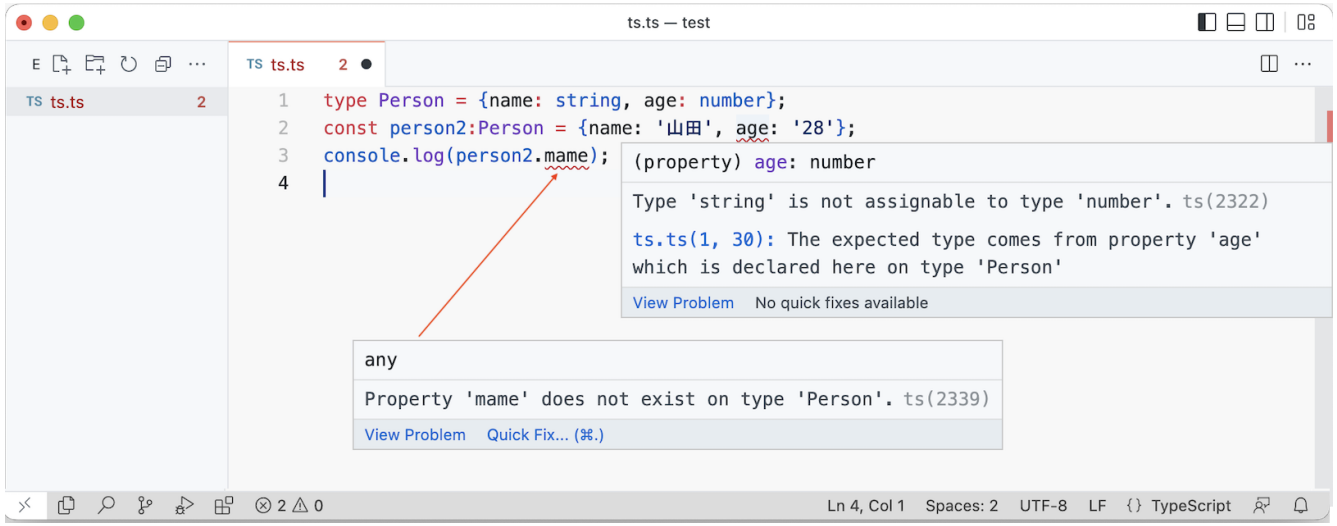
TypeScript

TypeScriptに変換

```
type Person = {name: string, age: number};

const person1:Person = {name: '山田', age: '28'};
console.log(person1.mame);
```

VSCodeのエラー表示

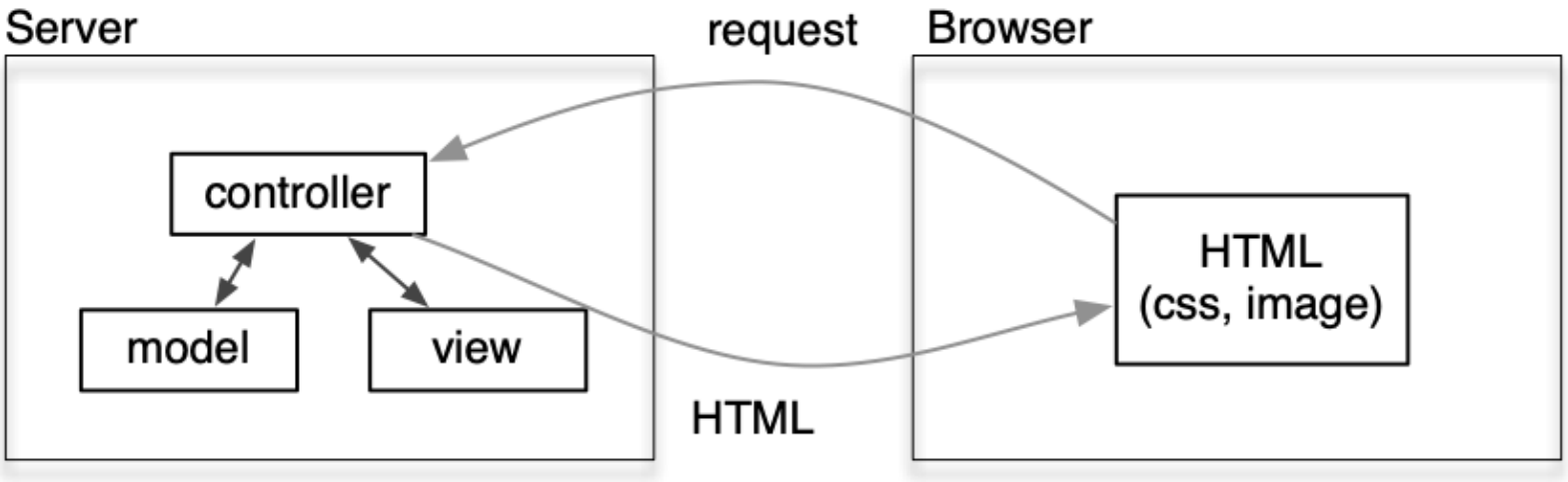


SPAとフロントエンド(SPA)フレームワーク戦国時代

死屍累累

SPAとは？ Webアーキテクチャの変化

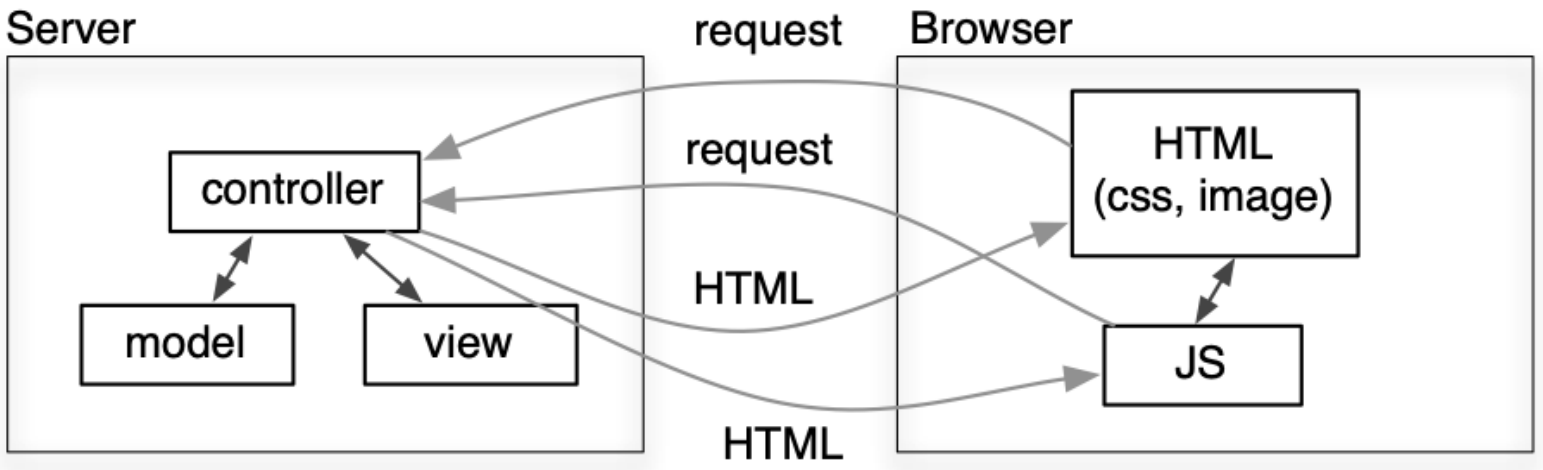
古き良きWebアプリ



- 昔は良かった😊

SPAとは？ Webアーキテクチャの変化

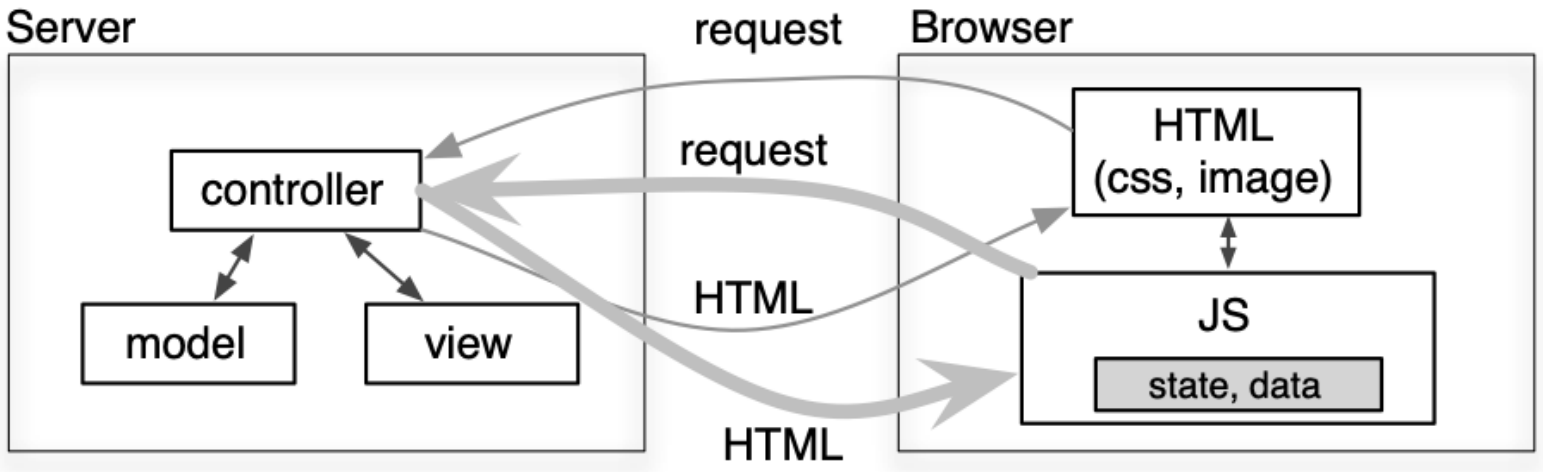
簡単なAjax



- シンプルな便利機能は嬉しい😊

SPAとは？ Webアーキテクチャの変化

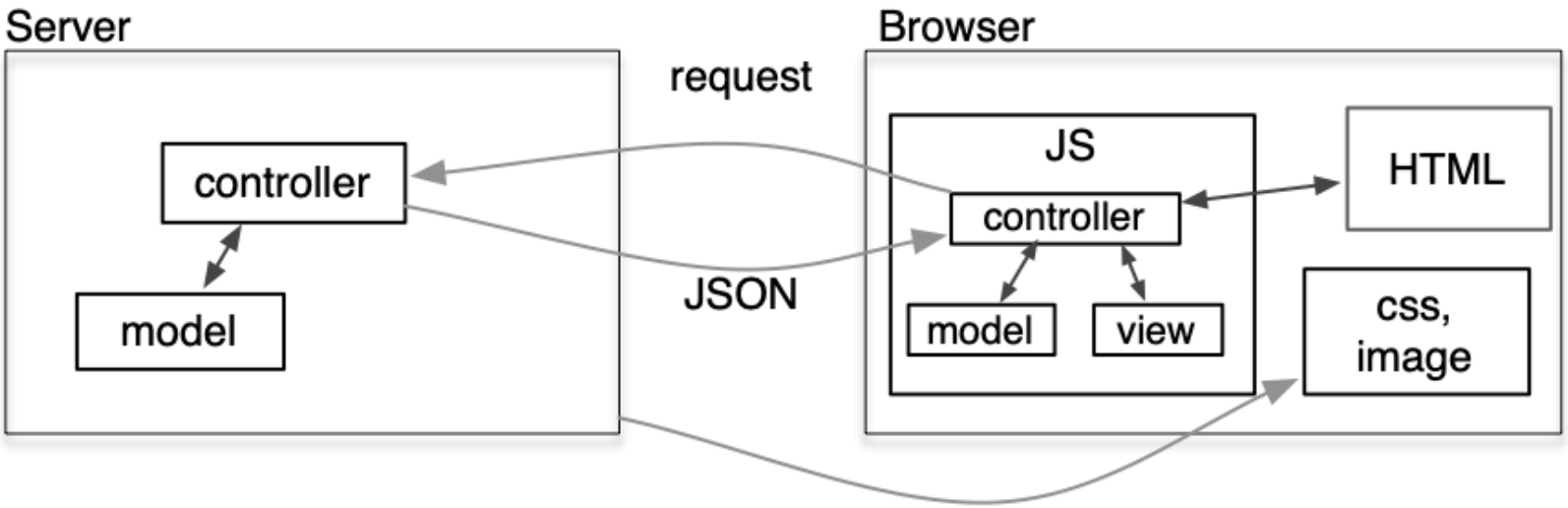
複雑なAjax



- リッチなUIが求められる 😅
- コード・通信が複雑怪奇になり、バグが取れない 😅

SPAとは？ Webアーキテクチャの変化

SPA(Single Page Application)



- リッチなUIと速い応答を実現 😊
- サーバーとブラウザーの役割分担が明確 😊
- サーバー側はスマフォと共通できる

SPAフレームワーク戦国時代

- AngularJS(Angular)
- Ember.js
- ExtJS
- Knockout.js
- Meteor.js
- React
- Vue.js
- ...

SPAフレームワーク戦国時代

AngularJS(Angular)

- 早い時期（2009年）に現れたSPAフレームワーク
- モデルとビューが自動的に同期する双方向データ結合
- フルスタック
- ライブラリーサイズも大きく、性能も良くなかった
- バージョン2で互換性のない変更を行いAngularになった（2016年）
- Angularは性能が上がり、コンポーネント指向になった

SPAフレームワーク戦国時代

React

- 2013年にリリース
- 世界最大級のサービスFacebookで実際に使われている
- 宣言的UI
- **Simple**と言われる
- 上位互換を保ちつつ、ときどき大規模な機能追加を行っている
- スマフォ開発に使えるReact Native

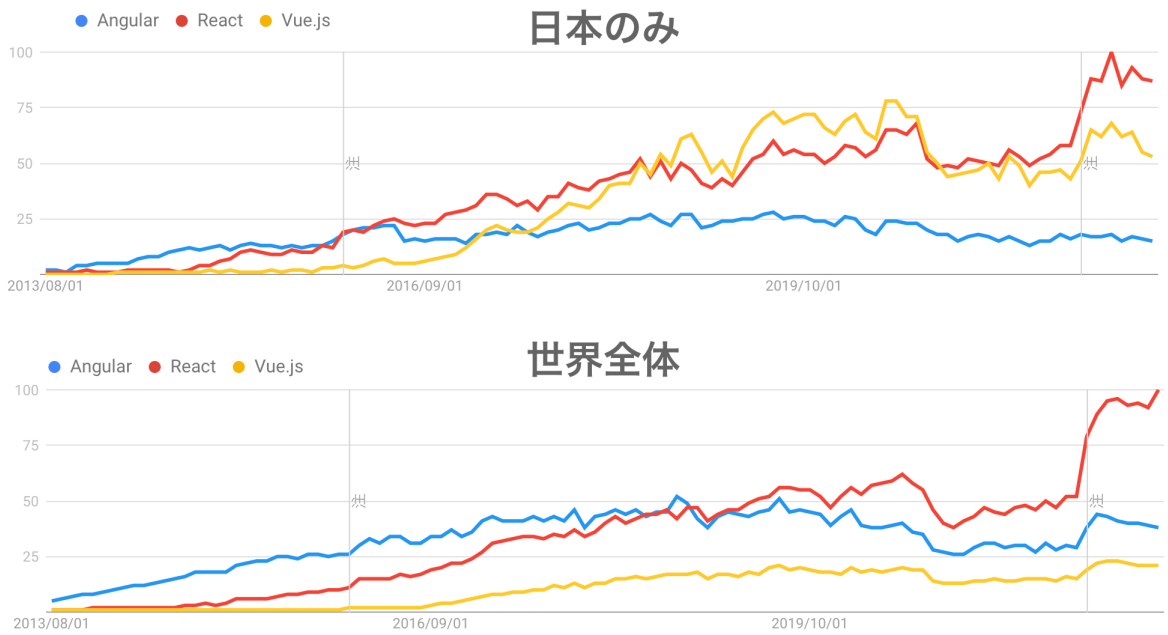
SPAフレームワーク戦国時代

Vue.js

- 2014年にリリース
- 宣言的UI
- 従来のWeb開発のスタイルを尊重
- **Easy**と言われる
- オフィシャルなサポートライブラリーをもつ
- 最初から日本語ドキュメントがあった
- TypeScript対応に手間取った

SPAフレームワーク戦国時代

勢力図（Googleトレンド）



State of JS

React超入門

簡単なReactの入門説明を行います

Reactの始め方

1. node.jsのインストール

- https://nodejs.org/ja/ の**推奨版**をダウンロード・インストール
- パッケージ管理システムを使ってインストールしてもよい
- バージョン14以降のnode.jsが必要

2. Reactプロジェクトの作成・実行

```
$ npx -y create-react-app APP_NAME --template typescript
$ cd APP_NAME
$ code .
$ npm start
```

React

Reactの重要な概念

- コンポーネント
- JSX
- Props
- イベントハンドラー
- State

コンポーネント

- 画面の要素をコンポーネントに分解
- コンポーネントは関数で定義
- コンポーネントは表示用HTML(JSX)を戻す

```
const Hello = () => {
  return <h1>Hello word!</h1>;
}
```

- コンポーネントは定義した関数名のタグで参照

```
<div>
  <Hello>
</div>
```

- ネット上に大量のOSSコンポーネントがある 一覧の例

JSX

- React内のJSはHTMLがリテラルとして書ける拡張が入っています（Bable等が変換します）
- HTMLを変数に代入したり、関数に渡したり、関数の戻り値などに出来ます
- JSの言語仕様との関係で標準のHTMLと違う部分もあります
- `{}` 内にはJSの式が書けます

```
const Sample = () => {
  return (
    <div className="sum">
      <p>合計：{1 + 2 + 3}</p>
    </div>
  );
}
```

JSX

条件によりHTMLを変えたい場合は条件演算子（三項演算子）を使います

```
const Sample = () => {
  const sum = -55;
  return (
    {sum < 0 ? <p style={{color: "red"}}>{-sum}<p> : <p>{sum}</p>}
  );
}
```

JSX

繰り返しは、map関数を使います

```
const Sample = () => {
  const items = [11, 22, 33];
  return (
    <ul>
      {items.map((item, ix) => {
        return <li key={ix}>{item}</li>;
      })}
    </ul>
  );
}
```

- 繰り返し要素にはReactの都合で、ユニークな値を持つ `key` 属性を付けます

Props

コンポーネントには引数（Props）を渡す事ができます

```
type HelloProps = {   // HelloコンポーネントのPropsの型宣言
  word: string;
}
const Hello: React.FC<HelloProps> = (props) => {
  return <h1>Hello {props.word}!</h1>;
}

// 別の書き方
const Hello: React.FC<HelloProps> = ({word}) => {
  return <h1>Hello {word}!</h1>;
}
```

呼び出し

```
  <Hello word="World" />
  <Hello word={"ab" + "c"} />
```

イベントハンドラー

ボタンを押した、キーを入力した等のイベントに反応するには以下のように書きます

```
const buttonClicked = () => { alter("Hello!"); }
<button onClick={buttonClicked}>押す</button>

// 別の書き方
<button onClick={() => { alter("Hello!"); }}>押す</button>
```

```
<input type="text" onChange={(e) => console.log(e.target.value);} />
```

State

コンポーネントが状態やデータを持つには**State**を使います

- stateの作成には `useState` ホック(特殊な関数)を使います
- stateの値が変更されるとコンポーネントは再表示されます

```
const Sample = () => {
  const [count, setCount] = useState(0);

  return (
    <p>{count}</p>
    <button onClick={() => { setCount(count + 1); }}>+1</button>
  )
}
```

画面の再表示と高速化

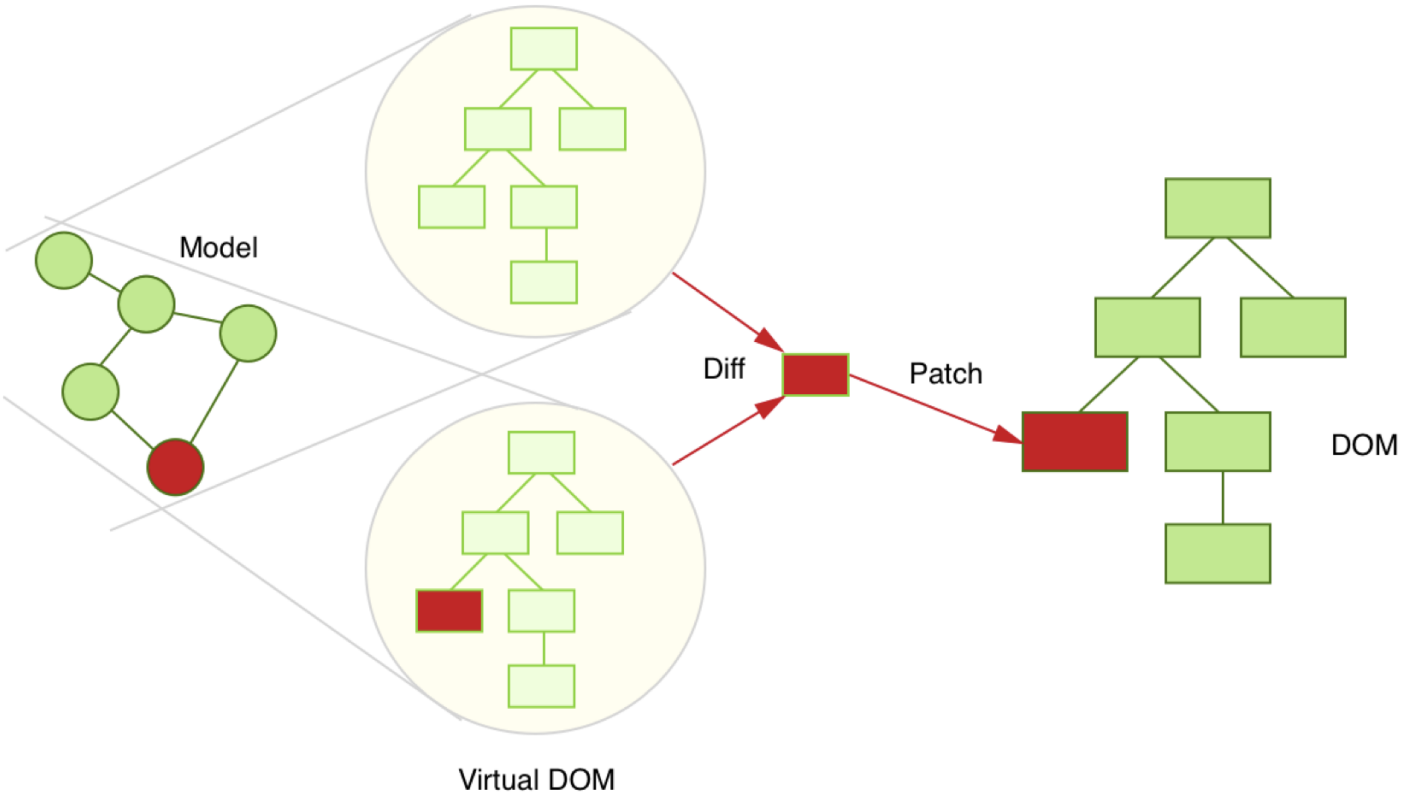
Reactコンポーネントは以下の場合、再描画されます

- 最初に画面が表示されたとき
- Props(引数)がかわったとき
- Stateの値が更新されたとき

ただし毎回、全画面を再表示すると遅くなってしまうのでVirtual DOMを使い必要最低限の部分のみを再描画します。

- メモリー上に画面に対応するDOM構造（Virtual DOM）を持っています
- Virtual DOMは現在の表示と更新した表示の2つあります
- 2つのVirtual DOMの差分を取り、必要な部分もにブラウザーの画面を変更します

Virtual DOM



http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html より

簡単なアプリ

コンピュータと対戦できるジャンケン・アプリのコードを見ていきましょう。

ソースコード： https://github.com/yuumi3/mane_jyanken

ReactとVue.js

ReactとVue.jsは似ています

- 宣言的なUI
- 高速化にVirtual DOMを利用
- コンポーネント指向
- コンポーネントの状態をStateやdataで管理している
- 親コンポーネントから子コンポーネントへにはプロパティー(props)を通して値が渡される

ReactとVue.js

ReactとVue.jsの違い

- Reactは統一性のあるシンプルで強力な（コンピューターサイエンス的な）技術を使い作られています
  - 例： JSX, 関数コンポーネント、Hooks
  - **Simple**
- Vue.jsは利便性を重視し、従来のHTML、CSS、JSの文脈から大きくずれないように作られています
  - **Easy**

EY-Officeブログ：TypeScriptを使ったReactとVue3の比較

GraphQLはバックエンドの革命児か？

フロントエンドの発展はバックエンドにも影響を与えている

Backend as a Service

フロントエンドが出来ればバックエンドは
FirebaseのようなBackend as a Service
を使えば、バックエンドの構築。管理を行
わなくて良い。

- Authentication
- Database
- Storage
- Function

Headless CMS

従来のWordPressのような一体型のCMSではなく、CMSのバックエンドのみを提供するサービス

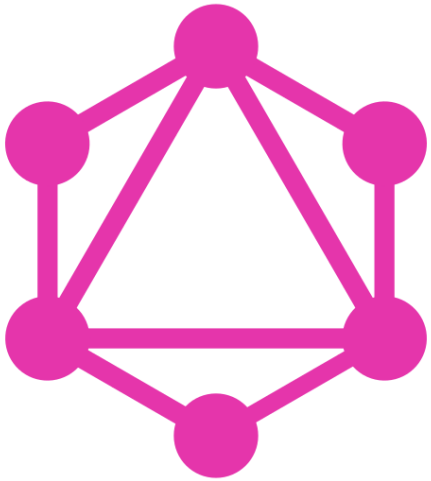
- 機能
  - コンテンツ作成・管理
  - コンテンツ提供API
  - 認証・認可

利用者はReact/Vue等でフロントエンドを作りコンテンツはHeadless CMSから取得し表示

GraphQL

RESTAPIにかわるAPIの方式

- Facbookが提唱したOSS
- クエリ言語
  - 必要な項目をクライアント側から指定出来る
  - 関連情報も一度に取得可能
- 関数（API）、データ型の定義に基づく
- Apollo GraphQLのような優れたライブラリーがある
- マイクロサービスをまとめるGraphQL Federation



GraphQL

Schema定義

```
type Query {          // データ取得APIの定義
  user(id: Int!): User
  users: [User]
}
type Mutation {       // データ更新APIの定義
  userLogin(email: String!, password: String!): User
  userLogout: Void
  userRegister(user: UserInput!): User
}

type User {           // Userデータ型定義
  id: Int!
  name: String!
  email: String!
  group_id: Int!
  group: Group!
  created_at: DateTime!
  updated_at: DateTime!
}
input UserInput {     // User入力データ型定義 + バリデーション定義
  name: String! @constraint(minLength: 1)
  email: String! @constraint(format: "email")
  password: String! @constraint(minLength: 8)
  group_id: Int!
}
```

GraphQL

データ取得(query)コード例

```
const {[getUsers, { loading, error, data }] = useLazyQuery<{users: User[]}>(gql`
  query Query {
    users {
      id name email
      group {
        id name
      }
    }
  }
`)
```

データ更新（mutation）コード例

```
const [doLogin, { data, loading, error }] = useMutation<{userLogin: User}>(gql`
  mutation UserLogin($email: String!, $password: String!) {
    userLogin(email: $email, password: $password) {
      id name email
    }
  }
`)
```

フロントエンド・フレームワークの近未来

残念ながら私には未来を知る能力はありませんので、フロントエンドで最近話題になっている技術について紹介します。

SPA(Single Page Application)の欠点

- 初期表示まで時間がかかる
    i. サイズの大きなJSファイルをダウンロード
    ii. React/Vueの初期化
    iii. サーバーからのデータ取得等
    iv. HTMLの構築
- SEOで不利
    - ~~Googleのクローラーは~~JS~~を解釈しない~~ 現在はOK
    - ただしクロールの優先順位が下がり、ニュースサイトなどでは致命的
    - 社内システム等では問題ない

SSR (Server Side Rendering)

動作原理

1. 初期画面はサーバー側で作りブラウザーにHTMLを戻しつつ、SPAを配信・起動
2. 以後はSPAとして動作

サーバー側でもReact/Vueを動かす必要があるのでNode.jsが必要

- SSR用フレームワークが使われている
  - Next.js React用
  - Nuxt.js Vue用

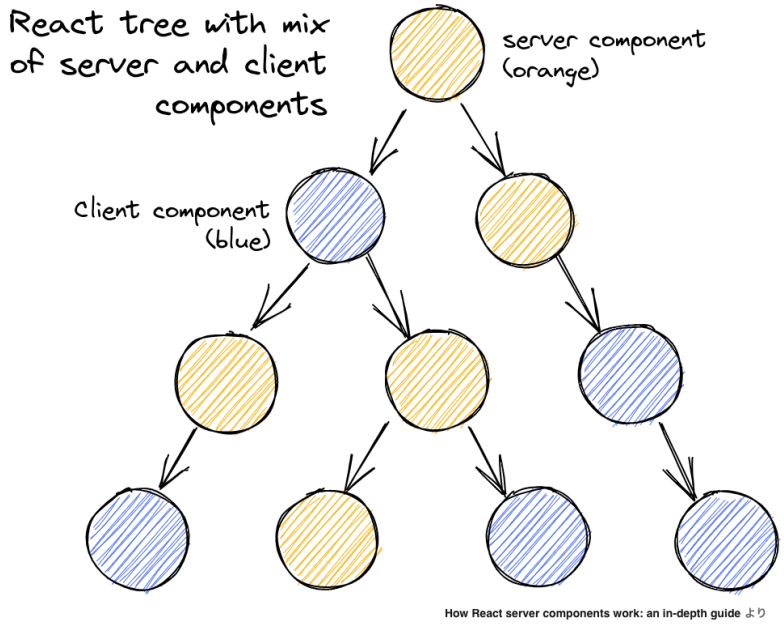
SSG (Static Site Generation)

ブログや企業紹介サイトのように、ほとんどのページはスタティックなHTMLでよいサイトもある。しかしReact/Vueの魅力的なコンポーネントや動的コンテンツを使いたい。このような要望に対応したのがSSG

- Gatsby 次世代のWordPress(?)
  - CMSだけではなく、データベース（バックエンド）から取得したデータを使ったページを自動生成できる
  - Reactベース
- Next.js SSG機能もサポート
  - Reactベース
- Astro
  - React, Vue, Svelteなどを使える

RSC (React Server Components)

Reactアプリ内のコンポーネントをコンポーネント単位でサーバーでもクライアント（ブラウザー）でも実行できるようにする技術。最新のReact 18に実験的機能として実装された。



How React server components work: an in-depth guide より

夢が拡がる 😊

Vue.js以降に出来たフロントエンド・ライブラリー

- Svelte : Virtual DOMは使っていない
- Preact : 軽量React
- Solids : 高速 JSXを使うがReactとは違う設計思想
- Inferno : 高速 Reactライク

どうなるのでしょうか？

モダンCDN

CDN(content delivery network)は「世界展開しているような大企業が使う高価なサービス」というイメージを持っていませんか？

しかし、最近あらわれたCDNサービスは使いやすく安価です

- Cloudflare
- Fastly
- Netlify
- Vercel
- ...

これらを上手く使うと高速で高負荷に強いサイトが、大きなコストをかけずに構築できます。

Edge computing

いくつかのモダンCDNでは、世界中に配置されたCDNの上でnode.jsが動作します。Edge computingという文脈でも語られています。

- SSRの実行
- 独自ロジックの分散実行
- 一部ではKey-ValueストアやRDB(SQlite3)をサポートしている
    - Edge間でのレプリケーション等もサポートするらしい
- 高速、高スケーラビリティーのWebアプリに役立つ（？）
- 将来が楽しみな技術 😍

WebAssembly

- WebAssembly 公式に近い情報(MDN)
- ブラウザー上でネイティブコードに近い速度で動く仮想マシン
- JS同様のサンドボックスの中で動作する
- JSに変わるものではなくJSを補強するもの
- C/C++, Rust, Go ...言語のコンパイラーがある
- 応用分野
  - 画像処理、ビデオ処理
  - 暗号
  - ...
- 将来が楽しみな技術 😍

おまけ Wasmer

The Universal WebAssembly Runtime

ブラウザーではなく、Windiows/Linux/macOSでWebAssemblyを動かせる環境

- 21世紀のJVM？
- 将来が楽しみな技術 😊

おつかれさまでした 🤦

QA