Why?React?

- なぜReactなのか？
  - JavaScriptの歴史を学ぶ

How?React?

- Reactの概要

JavaScriptの歴史

1. JavaScriptの誕生～第一次ブラウザ戦争

JavaScript: 1995年誕生

- **NetScape Communications**社によって開発
- Netscape Navigater2.0に実装
- 最初は「**LiveScript**」という名前だったが、当時人気のあった「**Java**」にあやかり「**JavaScript**」に変更した

Microsoft: 1996年 Internet Explorer3.0にJavascript実装

- Microsoftによる実装は「**JScript**」と呼ばれた
- NetScapeとは別の独自機能追加等も行っていた
- 開発者はそれぞれのブラウザに対応するサイトを作る必要があった

2. JavaScriptの標準化

なぜ、JavaScriptに標準化が必要なのか？

- JavaScriptはブラウザ向けのスクリプト言語として開発 → ブラウザがJavaScriptの言語を解釈しないといけない
- ブラウザは各企業が独自に開発している
- 独自仕様を実装したりしていたので互換性に乏しかった

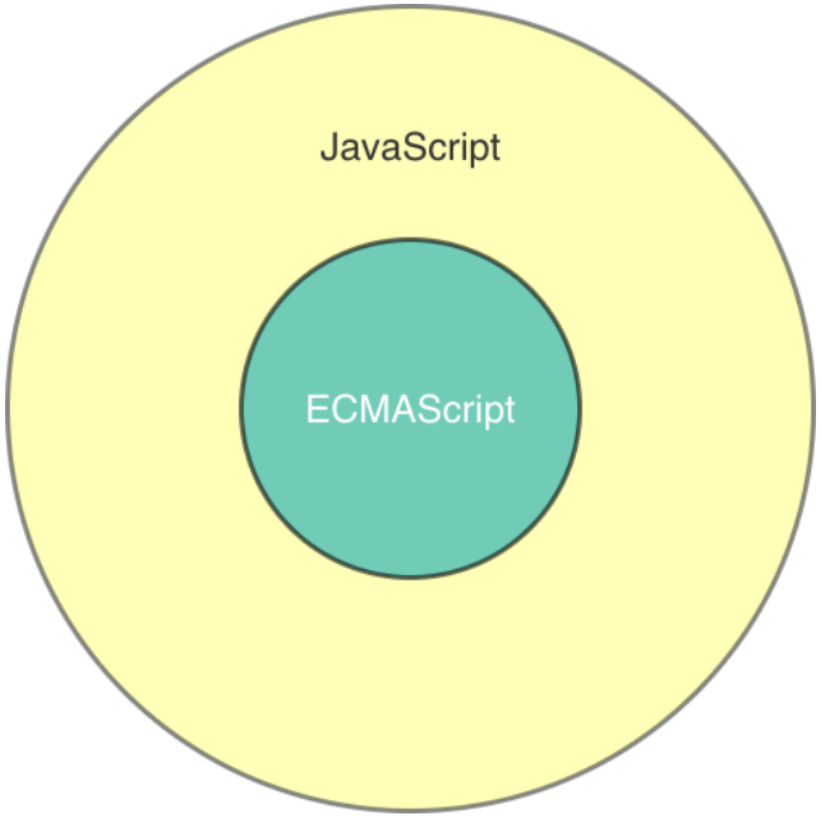
ECMAScriptの誕生

- JavaScriptのよりよい発展を目指して**ECMAScript**の誕生
- 標準化団体: ECMA Internationalによって標準化された
- **JavaScriptの中核となる言語仕様 / JavaScript = ECMAScriptに準拠した言語**

JavaScriptとECMAScript（参照：JavaScript Primer)

https://jsprimer.net/basic/introduction/

- JavaScript: 色々な実行環境がある
- ECMAScript = どの実行環境でも共通な動作のみが定義されている



第一次ブラウザ戦争の結果

- 1990年代後半, NetscapeとInternet Exploreの激しいシェア争いの結果、Internet Exploreが勝利！

JavaScript暗黒期

- JavaScriptの実装に絡んだブラウザのセキュリティホールが断続的に見つかる
- JavaScript起因のクラッシュ、悪用ウイルスの多発
- JavaScriptへの悪いイメージが定着
  https://currents.google.com/communities/101745672472327891411

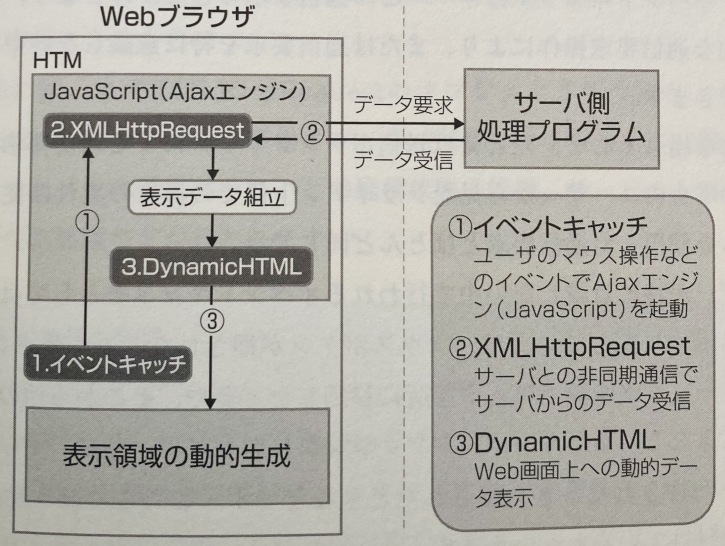
3. JavaScriptの復権：Ajaxの登場, jQueryの誕生

2005年：Ajax × Google Maps

- Ajaxの登場によって、再びJavaScriptが注目を集める。

手順① Webブラウザでのイベントキャッチ
手順② サーバとの非同期通信
手順③ Webブラウザ上での動的表示

■図1-8 Ajax方式での処理手順と機能概要



2006年：jQueryの誕生

- 「**write less, do mode**」 少ない記述で多くの実装ができる
  - クライアントサイドプログラミングの敷居を圧倒的に下げる
  - DOM操作, イベント処理, クロスブラウザ対応（jQueryで差異を吸収）
  - Ajaxとの相性も抜群！
  - JavaScript = **jQuery**

4.第二次ブラウザ戦争

- 五択の戦争: IE, Google Chrome, Firefox, Safari, Opera
  https://ja.wikipedia.org/wiki/ブラウザ戦争#/media/ファイル:Usage_share_of_web_browsers_(Source_StatCounter).svg

ここまでのおさらい

- JavaScriptはあくまで「**ブラウザ**」で動く前提だった
- **Ajax × jQuery** でインタラクティブなUI×UXの実現が可能になった

ここからのお話

- JavaScriptの進化
  - JavaScriptがサーバサイド（=ブラウザ外で）としても使われるようになる（**Node.js, npm**）
- JavaScript = ビルドが前提になってくる
- 脱JQuery → Reactへ・・・

5. JavaScriptの発展

Node.js の誕生

- 2009年 ライアン・ダールによって作られた
- ブラウザから独立したJavaScript実行環境
- Googleが開発した**Google V8 JavaScript Engine**という高速なJavaScriptエンジンが元になっている
- サーバサイド開発で用いられるようになる

＜参考＞
【ブラウザレンダリングにおけるJavaScript実行環境】
https://zenn.dev/ak/articles/c28fa3a9ba7edb#全体図

npm (モジュール/パッケージ管理システム)の台頭

- 当時の言語としての問題: **名前空間が1つしかない/依存関係が管理しずらい**
- 読み込み順が前後してしまうだけでバグがでてしまう...

```
<!-- jQuery を前提にjQuery UIが作られている-->
<script src="./jquery.js"></script>
<script src="./jquery-ui.js"></script>
```

- node.js独自のモジュール・パッケージ管理システムである「**npm**」が発展
- Webサーバとしてサーバーサイドで活用され始める（yahoo, LinkedIn, Paypal）

モジュール：名前空間の問題を解決

パッケージ：npm（パッケージ管理システム）で解決

モジュールについて

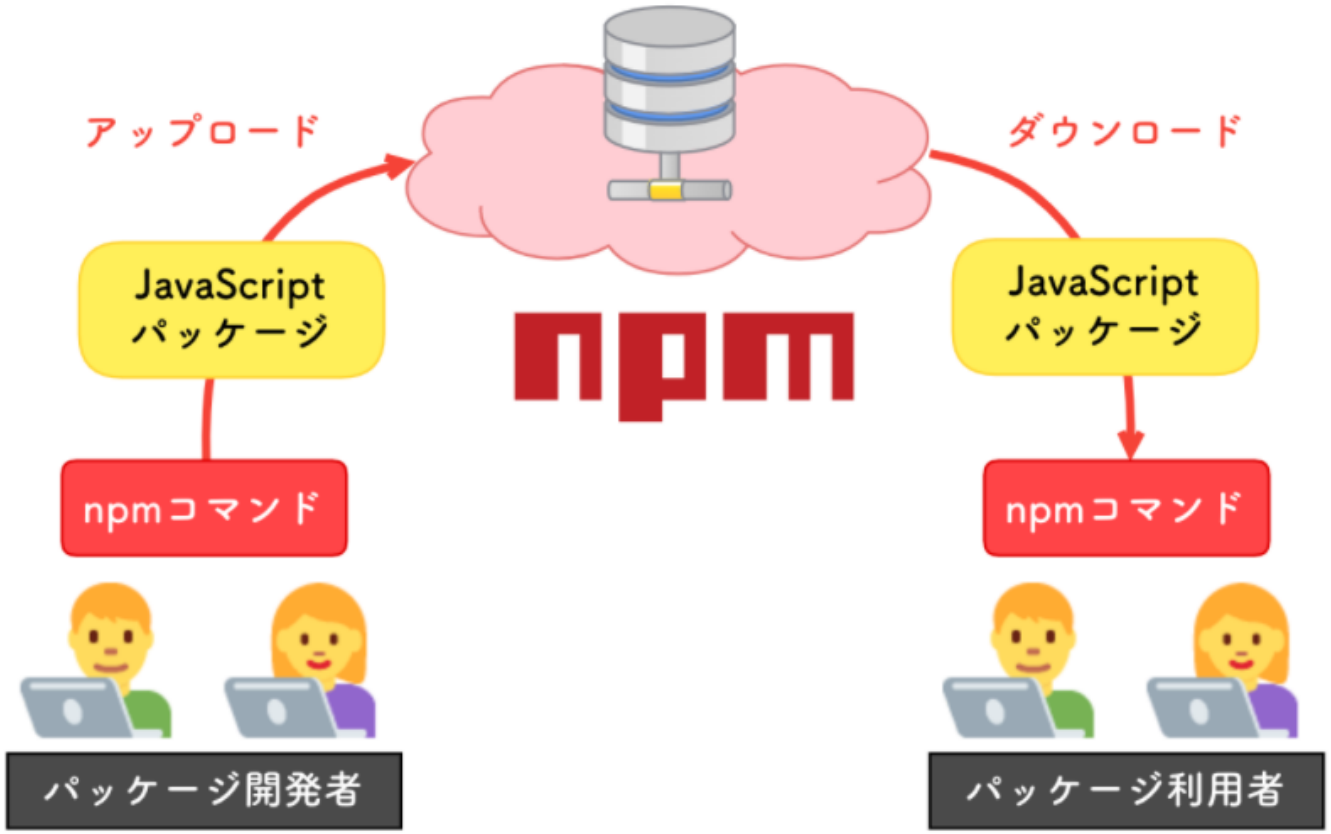
- 1995年の誕生から長い間ずっと、JavaScriptにはコードから他のファイルを読み込む仕組み：**モジュール**という仕組みがなかった
    - あるモジュールの関数を別のモジュールから呼び出す等・・・
- Node.jsでモジュールの仕組みが導入され、現在は**ECMAScript**でモジュールが定義されている
  https://jsprimer.net/basic/module/
- 1ファイル1モジュールが基本
- **moduleサンプルコード**
  https://codesandbox.io/s/moduletesuto-2mmr7?file=/index.html

npm（パッケージ管理システム）について

- パッケージ: パッケージとは**package.json**で記述されたファイルやディレクトリ
- JavaScriptの便利ライブラリのインストールをコマンドで行えるもの
- 依存関係の解決

npm よく使うコマンドまとめ
https://qiita.com/standard-software/items/2ac49a409688733c90e7

npmってなんですか？
https://sho03.hatenablog.com/entry/2021/02/01/194409


アップロード
ダウンロード
JavaScript
パッケージ
JavaScript
パッケージ
npm
npmコマンド
npmコマンド
パッケージ開発者
パッケージ利用者

Node.js クライアント開発（React開発）でどう関係してくるの？

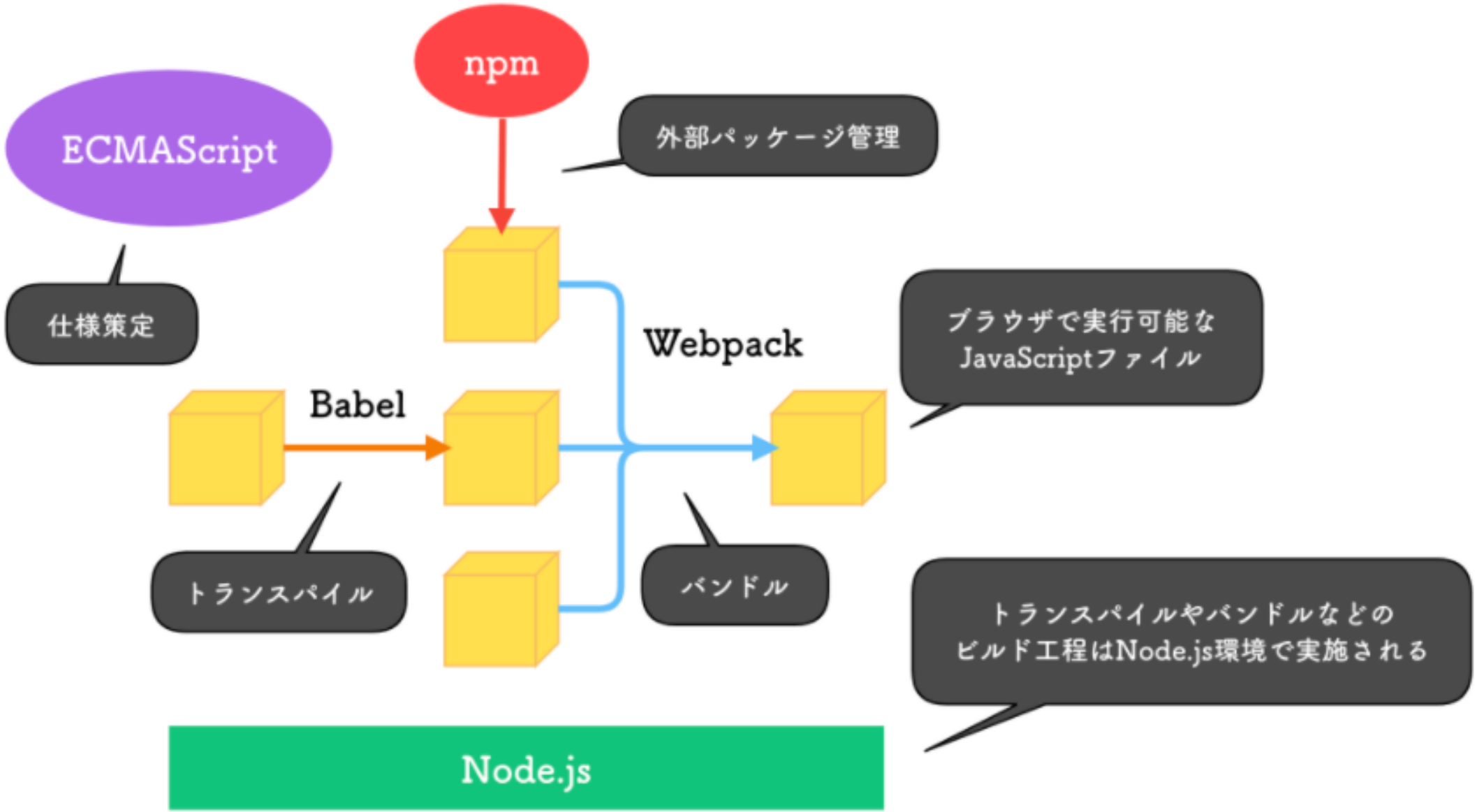
- バックエンド開発の為にnpmは使われるパッケージ管理システムであったが、**フロントエンド用のパッケージ**を提供するのに使われるようになった

- React等、JSフレームワークでの大規模開発となると、様々なパッケージが必要になり、そのパッケージ達が特定のバージョンで依存し合っている
- トランスコンパイル（変換処理）・バンドル（まとめる）→ **ブラウザが解釈可能な状態に変換してくれる**

【参考】

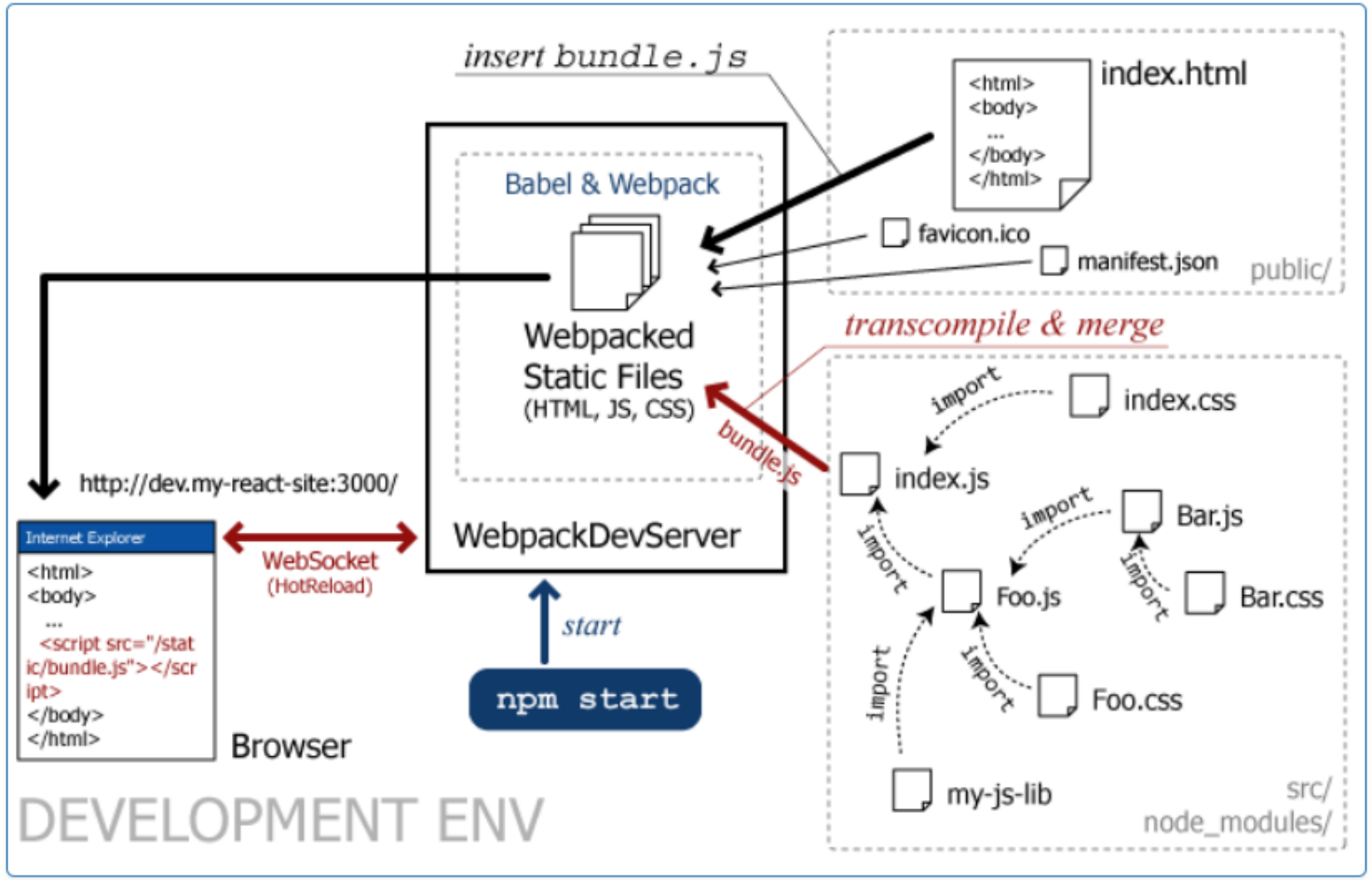
なぜReactアプリ開発でNode.jsが必要なのか

https://qiita.com/rpf-nob/items/6823fb8728754386ef30#なぜreactアプリ開発でnodejsが必要なのか


npm
外部パッケージ管理
ECMAScript
仕様策定
Webpack
ブラウザで実行可能な
JavaScriptファイル
Babel
トランスパイル
バンドル
トランスパイルやバンドルなどの
ビルド工程はNode.js環境で実施される
Node.js

開発時の構成

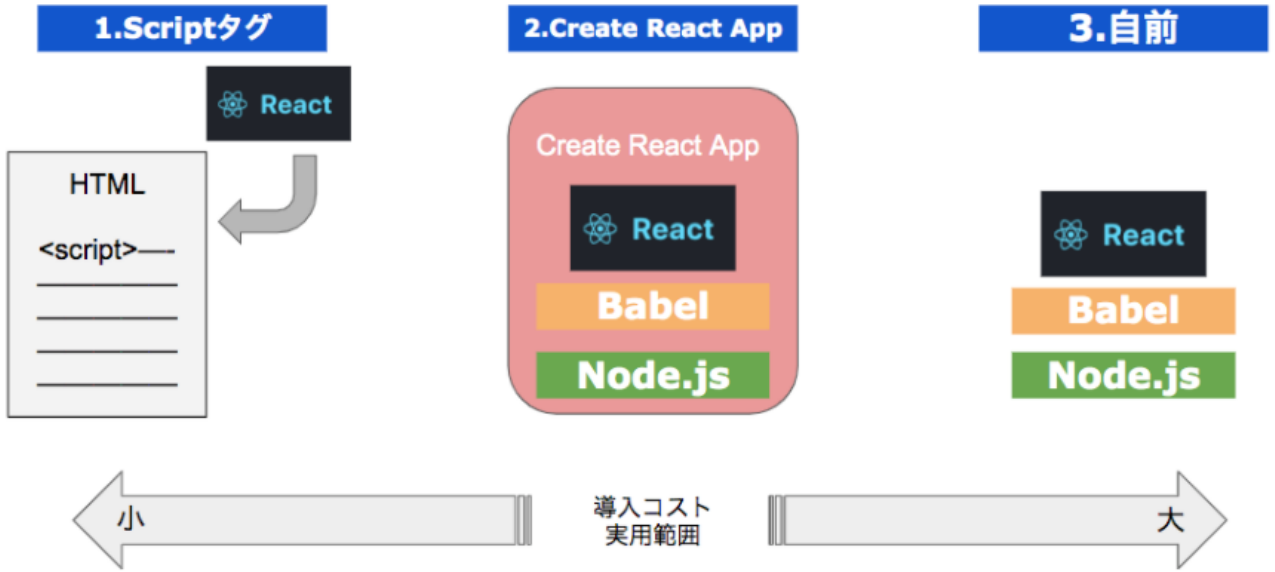
開発時には `npm start` で起動する Webpack 開発サーバを利用することができる。



このサーバは自動でソースの修正を検知してビルドを行い、WebSocket 経由でブラウザの表示をリロードするため開発効率が非常に良い。ただしパフォーマンスやセキュリティは考慮されていないためプロダクション用途のサーバではない。

導入方法

https://tech-brook.com/525/



さて、オススメは2.の手段になります。1.はReactを使えるのですが、本当に初歩のことしかできません。JSXも使いませんし、何より処理負担が大きく、サービスを作るにあたって実用的ではありません。Reactというものはどういったものかを知るだけと思ってください。

3.についてはReactを使っていく上で、必要になるNode.jsやBabelを自力で導入していく必要があるので、初めてには障壁画高いです。

ECMAScriptをより詳しく

- ECMAScript 2015年: ES2015（第6版）
  - 2015年からは毎年仕様策定がリリースされている
  - ECMAScriptのバージョンの歴史 : https://jsprimer.net/basic/ecmascript/
- 言語としての進化
  - jQueryを使わずに、標準仕様に準拠したJavaScriptで開発しよう
  - ECMAScriptに準拠した各JavaScriptライブラリ・フレームワークの台頭

6. Why? React? : なぜReact（フレームワーク）で開発するのか？

- JavaScriptは言語として進化している
  - **Reactは特にJavaScript標準の記法・考え方を重視している**
    → JavaScript構文をしっかり押さえていれば習得しやすい かつ 応用が効きやすい（JavaScriptがベースにあるので）
  - 標準のJavaScriptが使いやすくなった（非同期通信等）
    https://jsprimer.net/use-case/ajaxapp/promise/
- SPAを開発しやすい（**Single Page Application**）
  - 初回ロードで必要なリソースをまとめて読み込む
  - ユーザーの操作に応じて、動的にUIを書き換える。ページ遷移における再読み込み（ロード）がない。 例: Facebook, GoogleMap etc...
    https://www.oro.com/ja/technology/001/
  - API開発（バックエンド開発）と分業が可能

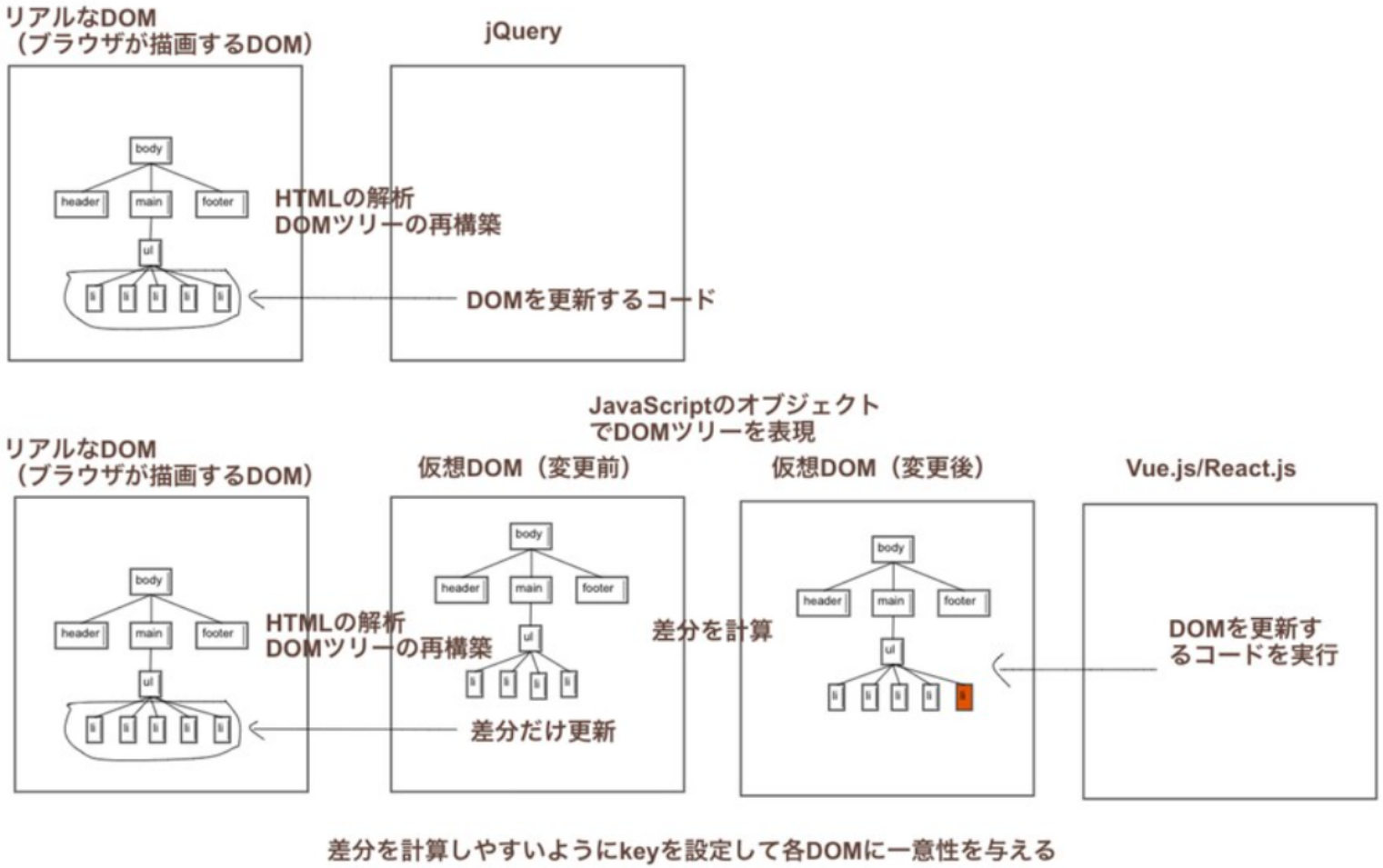
続き

- 仮想DOM
  - 以前: `HTML（DOM）= 操作する対象`
  - JSX: `HTML（DOM）= 拡張されたJavascript（JSX）の値（ファーストオブジェクト）`
  - よりDOMを扱いやすくなる
  - 直接DOMを更新するのではなく, JavaScriptのオブジェクトで仮想DOMツリーを実現 → **仮想DOM変更前と仮想DOM変更後の差分を計算して本DOMを更新**
    ※ 詳細は次のスライドで紹介
- その他、Reactの特徴「状態管理」「コンポーネント指向」「宣言的View」etc...
- **トランスコンパイル&バンドル** = JavaScriptを気軽にビルドできるようになった
  - TypeScript等、AltJSも組み合わせて開発＆保守しやすくする

Reactの概要

仮想DOM

- key を付与して差分が検知できるようにする
- https://ja.reactjs.org/docs/lists-and-keys.html#basic-list-component



宣言的View

手続き型（jQuery）

- DOM（Document Object Model）を直接いじって動的ページを生成する。
- 見た目（HTML）と操作（JavaScript）が分離していて、何をやったら何がどう変更されるか把握しにくい。
- あちこちから画面が参照・更新され、どこでどのような影響が出るかが分かりにくい傾向がある。
  https://codesandbox.io/s/lt-sample-1-forked-q6kx8?file=/src/App.js

宣言的View

宣言的（React）

https://codesandbox.io/s/lt-sample-1-forked-pk90o?file=/src/App.js

- HTMLの中に JavaScript が書かれていて、見た目と動作が同じ場所に書かれている
- アプリの内部状態がこの状態だったら、この見た目になるべし
- 見た目と動作の分離を防ぐソースコードが書ける

コンポーネントベース

- Webページやアプリケーションの構成単位を細かい部品単位で作成し、作成した部品を組み合わせていくのが構築手法。
- コンポーネント化することで再利用が可能