Engineer

シングルページアプリケーション

一つのページを読み込み、JavaScriptで書き換えるウェブアプリ

どのようなものか

シングルページアプリケーションは、single-page appとも表記し、通常はSPAと略します。一つのHTML文書を読み込み、サーバーへ新しいページを要求する代わりに、JavaScriptでその文書を書き換えるウェブアプリケーションです。内部のリンクをクリックしても、ページは破棄されません。スクリプトがクリックを受け取り、次の画面に必要なデータを取得し、DOMの一部を書き換えてアドレスバーを更新します。そのため、URLには利用者がいる場所が引き続き示されます。

この用語を定義する標準はなく、ツールの開発者自身もそれを認めています。Next.jsは、この主題のガイドを「SPAの定義はさまざまです」という説明で始め、そのうえで、自分たちが使う定義をstrict SPAと呼んで定めています。特徴は二つあります。一つのHTMLファイルがアプリ全体を受け持ち、ブラウザー内のJavaScriptが、すべてのルート、ページ遷移、データのリクエストを処理します。そして、ページ全体は再読み込みしません。ルートが変わると、クライアント側のスクリプトが、すでに開いているページのDOMを書き換え、サーバーへ新しい文書を要求する代わりに、必要なときにデータを読み込みます。この記事は、この定義を使います。提示されている中で最も直接的であり、ほかのすべての前提となる、文書が置き換わらないという性質を明示しているためです。

それを支えるブラウザーの仕組みは標準化されています。HTML StandardのHistoryインターフェースは、何も読み込まずにセッション履歴へ新しい項目を追加してURLを設定する `history.pushState()` と、現在の項目を書き換える `history.replaceState()` を提供します。利用者がそれらの項目の間を戻ったり進んだりすると、ブラウザーはウィンドウに `popstate` イベントを発火し、再描画はページへ任せます。標準が従来のlocationとhistoryのAPIの後継として紹介する、より新しいNavigation APIは、反対側から処理します。ナビゲーションが始まると `navigate` イベントが発火し、その `intercept()` メソッドが、ページ自身の処理する同一文書内のナビゲーションへ切り替えます。

対照的なのは、通常の複数ページのウェブサイトです。リンクをたどるたびにサーバーへ新しい文書を要求し、ブラウザーは古い文書を破棄します。ページのHTMLを静的サイトジェネレーターで一度生成するか、サーバーサイドレンダリングでリクエストごとに生成するかは、別の問題です。それは、HTMLが届いた後の動作ではなく、HTMLの生成方法に関わります。Next.jsは意図的にこの境界を曖昧にしています。静的エクスポートはルートごとにHTMLファイルを書き出しますが、その間の移動は依然としてクライアント側で行います。

概念を判断に生かす

「シングルページアプリケーションとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 読み込みが速く、スマートフォンでも使え、お客様がどのブラウザーを使っていても同じように動く画面を提供します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 フルスタックのGIS地図開発を指揮 — DBからブラウザーまでのフルスタック地図開発

フルスタックのGIS地図開発を指揮 お問い合わせ →

なぜ重要なのか

文書を置き換えないため、その中にあるものはクリック後も残ります。入力途中のフォーム、再生中の動画、サイドパネルのスクロール位置、取得済みのデータなどです。ネットワークを通す必要があるのは、次の画面に必要なデータだけで、ヘッダー、スタイル、スクリプトを含む新しいページ全体ではありません。画面の切り替えが、ページ内の一つの領域を書き換えるだけで済むこともあります。

Next.jsは、定義とともにその負担も挙げています。厳密なSPAは、ページを操作できるようになるまでに大量のJavaScriptを読み込む必要があることが多く、一つのリクエストが返ってからでなければ次を開始できない、クライアント側のデータ取得の連鎖を管理するのも困難です。スクリプトを実行せずにページを読むものにも、しわ寄せが及びます。すべてのルートに一つのHTMLファイルを使うと、JavaScriptを実行しないクローラーは、どのURLを要求しても同じ、ほとんど空の文書を受け取ります。Next.jsは、自身のブラウザー専用コンポーネントにも同じ指摘をしています。スクリプトを実行せずにページを索引化するものが受け取るのは、コンポーネントではなくフォールバックです。

Next.jsの答えは、クライアント側のナビゲーションを残し、それ以外を移すことです。JavaScriptのバンドルを自動的に分割し、異なるルートへのHTMLの入口を複数生成するため、利用者はそのルートに不要なコードを読み込まずに済みます。`next/link` コンポーネントはルートを先読みし、アプリのルーティング状態

を、リンクや共有に使えるURLに保持しながら、完全にクライアント側で描画するSPAと同じくらい素早く切り替わる操作感を実現します。静的エクスポートはさらに進めて、ルートごとに描画済みのページを書き出します。利用者はクライアント用バンドルを待たずに内容を受け取れ、ページ間のクライアント側遷移は、厳密なSPAと同じように速いままです。

また、一度選んだら変えられないわけでもありません。Next.jsは、静的サイト、またはクライアント側ですべてを描画する厳密なSPAとして始め、成長に合わせてサーバー機能を追加するプロジェクトを説明しています。既存のSPAも、最初から書き直すことなく移行できるとしています。

実際の活用例

以下のルーターは、この記事のために記述した、この考え方の最小限の動作例です。ページに `<main>` 要素があり、サーバーがアプリのすべてのパスに同じHTML文書を返すことを前提としています。二つの画面は、アプリが実際に表示する内容の代わりです。

```
// router.js
const views = {
  "/": { title: "Orders", text: "Your recent orders." },
  "/settings": { title: "Settings", text: "Delivery preferences." },
};
const main = document.querySelector("main");

function show(path, moveFocus) {
  const view = views[path] ?? views["/"];
  const heading = document.createElement("h1");
  const body = document.createElement("p");
  heading.textContent = view.title;
  heading.tabIndex = -1;
  body.textContent = view.text;
  main.replaceChildren(heading, body);
  document.title = view.title;
  if (moveFocus) heading.focus();
}

document.addEventListener("click", (event) => {
  const link = event.target.closest("a[href]");
  const modified = event.metaKey || event.ctrlKey || event.shiftKey;
  if (!link || link.origin !== location.origin) return;
  if (modified || event.button !== 0) return;
  event.preventDefault();
  history.pushState(null, "", link.href);
  show(location.pathname, true);
});
```

```
window.addEventListener("popstate", () => show(location.pathname, true));
show(location.pathname, false);
```

クリックハンドラーは、同じオリジンへのリンクに対する、修飾キーを伴わない通常の左クリックを引き受け、修飾キー付きのクリックは通過させます。そのため、新しいタブでリンクを開く操作も引き続き使えます。`preventDefault()` はブラウザー自身のナビゲーションを止め、`pushState()` はリンクのURLを使ってセッション履歴に新しい項目を記録します。中央の引数は歴史的な理由だけで残っているため、空文字列になっています。`show()` は画面を置き換え、文書のタイトルを設定し、新しい見出しへフォーカスを移します。`tabIndex` を `-1` にすると、Tabキーの移動順には含めず、スクリプトからフォーカスできるようになります。`popstate` リスナーは、利用者が戻ったり進んだりしたときに対応する画面を再描画し、最後の行は、ページを開いたURLの画面を描画します。これは、サーバーが `/settings` にも `/` と同じ文書を返す場合だけ動作します。実際のアプリでは、開始ページを黙って表示する代わりに、ページが見つからないことを示す独自の画面を表示し、ファイルやサイト内のほかの部分へのリンクには介入しないようにする必要があります。

創業者が現在NextMarinerの名義で取り組んでいる業務では、Next.js 16のフロントエンドが、実際の製品で生じるこうしたコストを示しています。そのポートフォリオ記事では、反応が速いと感じられ、実際に対話的に操作できる状態を保つ必要があった認証付きダッシュボードを取り上げ、すべてのデータをクライアントで取得すると何が起きるかを説明しています。初回の読み込みが遅くなり、N+1のパターンが現れます。各コンポーネントが起動して、それぞれ独自のリクエストを送り、やがて1ページが何度もの往復通信の連鎖になるのです。これはNext.jsのガイドが警告するクライアント側のデータ取得のウォーターフォールが、実際に起きた例です。対策では、必要な箇所のクライアント側の対話機能を維持し、最初のデータ取得をサーバーへ移しました。ダッシュボードと大きなディレクトリー一覧では、`createPrefetchedServerPage` というファクトリーが、必要なデータをすでに備えた各ページをクライアントへ渡します。記事は残りの状況も率直に説明しています。ほかの認証付きページは今もクライアントで描画され、対応の順番を待っています。つまり、移行が機能していて、次に対応するページも明確になっている段階を記録しており、移行が完了したという話ではありません。

問題が起きるところ

戻るボタンを押すとアプリを離れてしまう。 スクリプトが `pushState()` を呼ばずに画面を切り替えると、その画面の項目がセッション履歴に残りません。戻る操作は、利用者が見たすべての画面を飛ばしてアプリの外へ出てしまい、再読み込

みすると、最初にアプリを開いたときのURLへ戻ります。利用者がページだと考える画面ごとに履歴へ項目を追加し、入力中のフィルターなど、独立した履歴項目として残すべきでない変更には `replaceState()` を使い、 `popstate` も処理してください。Next.jsがルーティング状態をURLに保持するよう勧める理由も同じです。それによって、画面にリンクし、共有できるからです。

特定の画面への直接リンクが、一つのファイルしか知らないサーバーへ届く。 厳密なSPAでは、一つのHTMLファイルがすべてのルートを受け持ちますが、アドレスバーには `/settings` が表示されています。再読み込みしたときや、共有されたリンクを開いたときにブラウザーが要求するのは、そのパスです。 `index.html` しかないサーバーは404を返します。アプリのすべてのパスに同じ文書を返すようサーバーを設定するか、ルートごとにファイルを生成してください。Next.jsの静的エクスポートは後者です。 `out` フォルダーに `index.html` 、 `404.html` 、 `blog/post-1.html` などを保持し、ガイドは、拡張子のないURLを対応するファイルへ結び付けるnginxルール `try_files $uri $uri.html $uri/ =404;` を示しています。

フォーカスが古い画面の場所に残る。 `pushState()` が変更するのは、セッション履歴とURLだけです。スクリプトが画面を切り替えても、キーボードフォーカスはクリックしたリンクに残るか、それを保持していた要素を削除すると失われます。新しい画面が表示されたことも、ページが通知しなければ支援技術には伝わりません。例のように、タイトルを設定し、新しい画面の見出しへフォーカスを移してください。Navigation APIは、この処理をより多く引き受けます。 `intercept()` で処理するナビゲーションは、既定で、 `autofocus` を持つ最初の要素か、なければbodyへフォーカスを移します。ハンドラーが終わると、ブラウザーは利用者へ完了を通知します。標準は、そこに支援技術も含まれるとしています。アプリ自身でフォーカスを置く場合は、 `focusReset: "manual"` でこの動作を無効にします。

すべての負担が最初に集中する。 厳密なSPAは、何かを操作できるようになるまで大量のJavaScriptを読み込む必要があることが多く、最初のデータリクエストも、そのコードが動いてからでなければ始まりません。ルートごとにバンドルを分割して、現在の画面に必要なものだけをダウンロードさせ、最初のリクエストはサーバー側で開始してください。Next.jsの対応パターンでは、Server Componentでデータ取得を開始し、完了を待たずにPromiseをClient Componentへ渡し、そこで `use()` によって値を取り出します。アプリの残りを描画する前にリクエストが始まるので、ガイドによれば、レスポンスは直ちにストリーミングを開始でき、クライアントは、それぞれが前の完了を待つリクエストの連鎖を避けられます。

参考資料

- Next.jsでシングルページアプリケーションを構築する方法 — 厳密なSPAの定義、その負担、クライアント側のナビゲーションを維持しながら処理をサーバーへ移すNext.jsのパターンを説明しています。
- HTML Standard：Historyインターフェース — `pushState()`、`replaceState()`、セッション履歴の項目を追加、置換する仕組みを定めています。
- HTML Standard：Navigation API — `navigate` イベント、`intercept()`、それに伴うフォーカスとスクロールの処理を定めています。
- Next.jsアプリケーションの静的エクスポートを作成する方法 — ルートごとのHTMLファイル、静的エクスポートではできないこと、nginxによる拡張子のないURLの配信を説明しています。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「シングルページアプリケーションとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/single-page-application/