

# Web システムにおける生成 AI を利用した JavaScript の ES6 化と Vanilla JS 化の取り組み

中野 裕司<sup>1,2)</sup>, 喜多 敏博<sup>1,2)</sup>, 久保田 真一郎<sup>1,2)</sup>, 杉谷 賢一<sup>1,2)</sup>

1) 熊本大学 半導体・デジタル研究教育機構, 2) 熊本大学 大学院社会文化科学教育部 教授システム学専攻

nakano@cc.kumamoto-u.ac.jp

## ES6 and Vanilla JS for JavaScript using Generative AI in a Web System

Hiroshi Nakano<sup>1,2)</sup>, Toshihiro Kita<sup>1,2)</sup>, Shinichiro Kubota<sup>1,2)</sup>, Kenichi Sugitani<sup>1,2)</sup>

1) Research and Education Institute for Semiconductors and Informatics, Kumamoto University,

2) Graduate School of Instructional Systems, Kumamoto University

### 概要

生成 AI を利用して, jQuery に大きく依存した Single Page Application 型の Web アプリケーションの Vanilla.JS 化, すなわち脱 jQuery 化を試みた. JavaScript のソースコードを関数群とそれ以外の共通部分に分割し, 共通部分と html, css をまず Vanilla.JS 化し, それを利用して個々の関数の Vanilla.JS 化を試みた結果, ほとんど文法エラーを伴わない, 指定のライブラリを用いた, Vanilla.JS 化されたソースコードを短時間で得ることができた. ただし, 全体として, 元の Web アプリケーションの機能は実現されず, かなりの手直しが必要であることもわかった. しかしながら, 手作業で変換することに比較すれば, 十分効率化できると考える.

### 1 はじめに

大学等高等教育機関に置いても, 様々なサービスを Web アプリケーションで提供している. 提供するサービスの維持管理, 機能追加, 改善等に関して, 少なくともソースコードの改変が可能なものに関して, 大幅な変更が必要になる場合がある.

今回紹介する事例は, コメント込で 5000 行程度の規模の WebAPI を用いた SPA (Single Page Application) 形式の Web アプリケーションで, Web ブラウザ上のクライアントプログラムは JavaScript とそのライブラリを中心に構成され, ES6 より前の記法で主に記述され, jQuery[1] を基盤に jQuery に依存するライブラリを主に使用している.

問題は, モバイル対応に jQueryMobile[2] を利用していたが, そのサポートが終了し, 開発が停止したことによる. これだけでも困ることではあるが, 開発が停止したことにより, jQuery のバージョンも挙げられなくなってしまった. これが最大の問題ではあるが, 他にも, JavaScript が ES6 以前の形式で書かれているためスコープの問題を内包している可能性がある, jQuery の記法が多用されている, jQuery に依存する

ライブラリが主に使われている等細かな問題もある.

そこで, 使用ライブラリも含めて対象のクライアント側のソースコードの脱 jQuery 化を, 生成 AI で試みたので, その結果を報告する.

### 2 現状分析

現状の JavaScript プログラムは, 表 1 に示す JavaScript ライブラリを用いている. 先に述べたとおり, 最大の問題は jQueryMobile の開発が停止し, このライブラリを使う限り jQuery のバージョンアップもできないことであるが, 他のライブラリの多くも jQuery に依存しており, jQuery 自体の今後を危ぶむ声もあるため, jQuery から脱却できないかと考えた. jQuery を使用しないことを Vanilla.JS と呼ぶことがある [3] ということで, ここでは Vanilla.JS 化と呼ぶことにする.

表 1 に示すとおり, jQuery, jQueryMobile, dataTables, jquery.i18n.properties の利用を取りやめ, 代替ライブラリに変更したい. 代替ライブラリは, 他のライブラリに依存しないこと, 現行のサービスが継続可能な機能を持っていることを基準に選考した.

表 1 廃止ライブラリと代替ライブラリ

現行ライブラリ名	jQuery 依存	代替ライブラリ (jQuery 非依存)
jQuery	そのもの	不使用 (JavaScript ES6)
jQueryMobile	依存	Bootstrap5
dataTables	依存	tabulator
jquery.i18n.properties	依存	不使用 (JavaScript 内包)
Chart	非依存	Chart (継続利用)

ソースコード 1 元の JavaScript (抜粋)

```

1 // グローバル変数
2 var locale = defLocale; // 現在のロケール
3 var selectedYear = null;
4 ...
5 // 初期化
6 $(function() {
7     loadMessages(); // 言語にあった文字列表示
8     ...
9     $(".langSelector").click(function() {
10         ...
11     });
12
13 function loadMessages() { // 関数 ...
14     ...
15 }
16
17 function selectList() { // 関数 ...
18     $.ajax({
19         ...
20     });
21
22 var globalVariable = ...;
23
24 $.fn.dataTableExt.oSort['stdNo-asc'] =
25     function(x, y) {
26         ...
27     };

```

現行の JavaScript のソースコードはコメント行を含めると 5000 行程度あるが、およその構造を、ソースコード 1 に示す。全体としては、グローバル変数、jQuery の初期化、関数群、jQuery のオブジェクトへのメソッド追加等から成り立っている。

ソースコードの多くは関数群から成り立っており、13-15 行、17-20 行のような関数が、38 3 個確認される。また、6,9,18 行等、jQuery 依存のコードは非常に多く見られる。このような状況で非 jQuery 化を試みる場合、初期化やグローバル変数の情報も必要になる。

2,3,22 行あたりから、var が多く使われており、JavaScript バージョン ES6 以前のコードが主体となっている。この点も修正可能であれば修正したい。

今回のプログラムは WebAPI を用いた SPA である

ことは、修正に有利であると考え、サーバとのデータのやりとりは、WebAPI のみで行っているため、該当する jQuery のコードは一對一に、JavaScript に置き換えることができると思われる。また、サーバ側のプログラムに一切手を加えずにテストが可能である。

### 3 変換方針

以上のようなソースコードの状況であることから、生成 AI による変換を試みた。

まず、JavaScript 及び html, css のソースコードをそのまま生成 AI に変換させようとしたところ、トークン数超過でそのままでは無理であることがわかった。そこで、html, css, JavaScript の共通に必要な部分（初期化、グローバル変数等）をベースとして、それに関数を 1 つだけ加えた状態で変換する。といった操作に関数の数だけ繰り返すことにした。その流れを図 1 に示す。

この一連の処理は Python プログラムで、コメント除去、ソースコード分割、ChatGPT の API による変換等、自動で処理し、全体で 8 分程度で完了する。

まず最初に、プログラム中のすべてのコメントを機能的に除去することとした。これは、古いコードがコメントアウトされて含まれていたり、コメントが必ずしも処理を正確に表していない場合もあり、かえって変換に悪影響を与えるのではと考えたからである。また、新たにコメントを追加するように生成 AI には司令している。

図 1 に示すとおり、コメントを除去した後、関数は 1 つ 1 つファイルに分け（今回は 38 個）、残りの JavaScript をまとめて base 部分の元ソースコードとした。このベース部分と html, css をまず生成 AI にかけて、脱 jQuery (Vanilla.JS 化) コードの生成を試みた。この新しい 3 点セットに、先に切り分けた 3 8 個の JavaScript の関数を 1 つづつ加えて、ChatGPT で Vanilla.JS 化を行った。

最後に、変換された 38 個の関数と変換に使用した

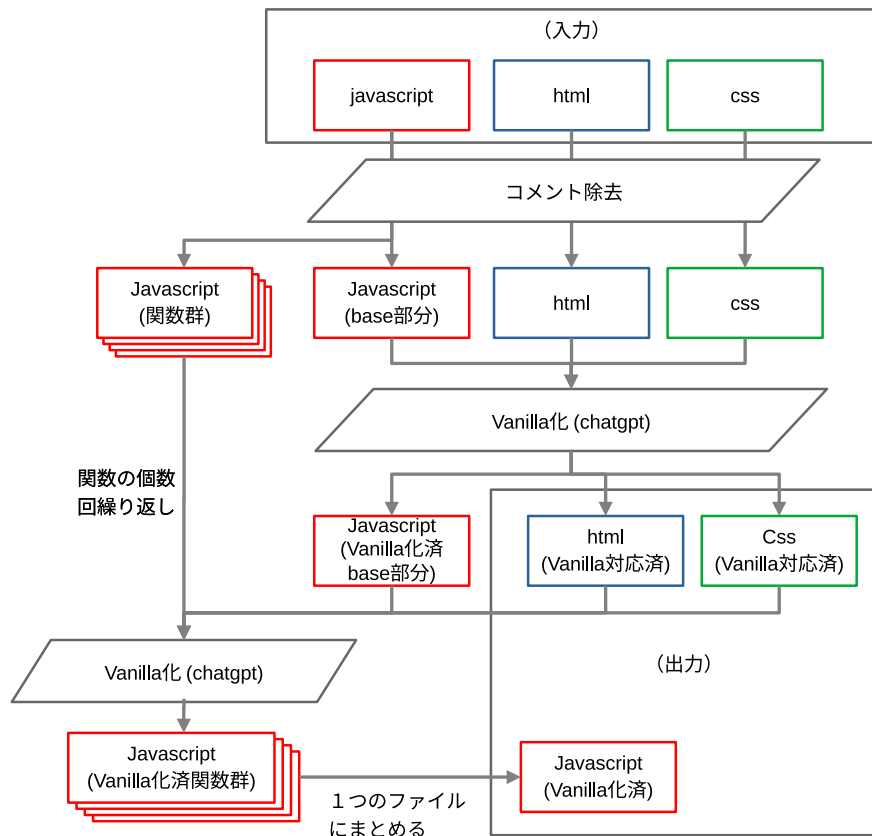


図1 ソースコード変換の流れ

JavaScript の base 部分をつないで1つのファイルにし、それを JavaScript の出力とした。

本変換プログラムの最初の Vanilla.JS 化の部分をソースコード 2 に示す。ほとんど変換方針に従ったプロンプトを記載したものであることがわかる。

二番目の Vanilla.JS 化の処理は、38 回繰り返すことになるが、基本的にはほとんど同じなので省略する。ただし、書き換える対象は与えられた関数のみとしている。

#### ソースコード 2 ChatGPT へのプロンプト付近

```

1 gptModel = "gpt-4o-mini"
2 ...
3 completion=client.chat.completions.create(
4   model=gptModel,
5   temperature=0.3,
6   messages=[
7     {"role": "system",
8      "content": "あなたは優秀かつ説明が上手な開発者です。"},
9   ],
10  {"role": "user",
11   "content": f'''
12 以下に示す
13   html、JavaScript、cssのソースコードを
14  html5、ES6、CSS3の規格に従って
   書き換えてください。

```

ただし、以下ことを全て守ってください。

- ・ jQueryライブラリの使用を止めてください。
  - ・ jquery.i18n.propertiesライブラリの使用を止めてください。
  - ・ jQueryMobileライブラリの使用を止めて、代わりに Bootstrap5ライブラリを使ってください。
  - ・ dataTablesライブラリと関連するライブラリの使用を止めて代わりに tabulatorライブラリを使ってください。
  - ・ Chartライブラリは使用してください。
  - ・ 以上以外のライブラリは使用しないでください。
  - ・ 使用するライブラリは全て cdn等の外部リンクで利用してください。
  - ・ Global変数は、Globalのままにしてください。
  - ・ 未実装（空）の関数は作らないでください。
  - ・ varは使わないでください。
  - ・ 元のコードは全て省略せずに変換してください。
  - ・ 必要に応じてソースコードにコメントを日本語で追加してください。
- ```

''html
{ogHtml}
''
''javascript
{ogBase}
''

```

```

44 ...
45 )
46 cvBaseAll
47 = completion.choices[0].message.content

```

#### 4 変換結果

ソースコード 3 に、変換した JavaScript コードの一部を示す．変換するたびに変換結果が異なるため、一例として見てほしい．一般に文法エラーは極めて少なかった．また、ライブラリも指定のものを使い、JavaScript も ES6 以上を用いている．WebAPI によるデータ取得部分はおおよそ動作している．コメントは少ないが、内容は概して的確であった．ただし、期待通りの動作をするかという点、手直しの必要な場合が多く、この規模のサービス全体として機能するものは 10 回程度の試行ではなかった．

では役に立たないかという点、十分役に立つと判断できる．多くの手直しが必要ではあるが、文法エラーがない、以前と同じ名前の関数群で構成されたソースコードで、WebAPI のデータ取得はほぼできているため、修正の開始地点としては、十分役に立つと考えられる．

ソースコード 3 変換後の JavaScript (抜粋)

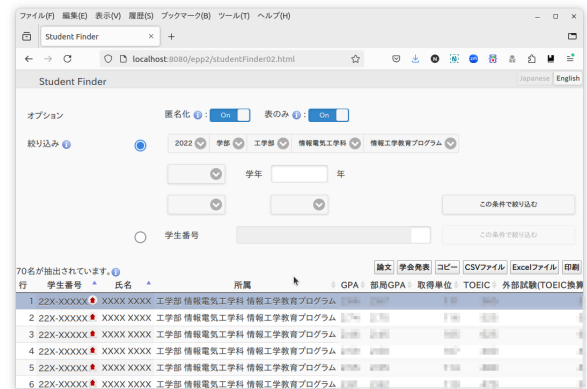
```

1 ...
2 let locale = defLocale;
3 let selectedYear = null;
4 ...
5 document
6   .addEventListener(
7     'DOMContentLoaded', () => {
8     ...
9   }
10 $(function() {
11   loadMessages();
12   ...
13   document
14     .querySelectorAll('.langSelector')
15     .forEach(button => {
16     ...
17   });
18
19   function loadMessages() {
20     fetch('bundle/ep2_${locale}.json')
21       .then(response => response.json())
22       .then(data => {
23         // langSelectorの無効化と有効化
24         document
25           .querySelectorAll('.langSelector')
26           .forEach(button => {
27           ...
28         });
29         ...

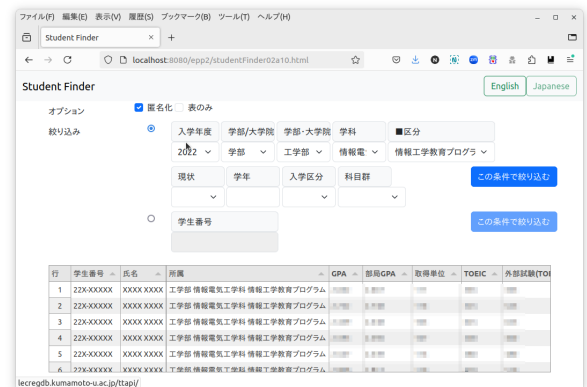
```

表 2 JavaScript ソースコード変換に関するデータ

| 項目                    | 変換前  | 変換後  |
|-----------------------|------|------|
| ソースコード行数<br>(コメント含まず) | 3672 | 2804 |
| 平均行数 (関数あたり)          | 85.2 | 72.8 |
| 平均コメント行数 (関数あたり)      | 0    | 4.1  |
| 関数数                   | 38   | 38   |
| 変換時間 (分)              |      | 8    |
| 関数あたり変換時間 (秒)         |      | 12   |



(a)



(b)

図 2 (a) 変換前の Web アプリケーション, (b) 変換後開発中の Web アプリケーション

表 2 に、変換に関する諸データを示す．本変換に 8 分程度要しているが、1 関数あたりでは 12 秒程度しかかかっていない．また、変換に関するコストもかかっているが、1 ドル未満である．

図 2 に示すように、現在この方法で変換したソースコードを元に Vanilla.JS 版の開発を進めているが、人手で変換することを考えれば、かなり省力化できたと思われる．また、ここで Vanilla.JS 化することで、特定のライブラリへの依存がなくなるだけでなく、利用

可能なライブラリの幅が広がることを考えると、そのメリットも大きいと考える。

## 5 まとめと今後の課題

生成 AI を利用して、jQuery に大きく依存した WebAPI を用いた SPA 型の Web アプリケーションの Vanilla.JS 化 (脱 jQuery 化) を試みた。JavaScript のソースコードを関数群とそれ以外の共通部分に分割し、共通部分と html, css をまず Vanilla.JS 化し、それを利用してここの関数の Vanilla.JS 化を試みた結果、ほとんど文法エラーを伴わない、指定のライブラリを用い、Vanilla.JS 化されたソースコードを短時間で得ることができた。ただし、全体として、元の Web アプリケーションの機能は実現されず、かなりの手直しが必要であることもわかった。しかしながら、手作業で変換することを考えれば、十分に立つレベルであった。

今後、プロンプトの工夫等で変換精度を上げることが試みたい。また、今回、css のソースコードを用いたが、ライブラリに依存する部分も大きいため、取捨選択が必要であろう。

## 参考文献

- [1] j Query (web site) : <https://jquery.com/>  
(2024-10-21 確認)
- [2] JQuery (web site) :  
<https://jquerymobile.com/> (2024-10-21 確認)
- [3] What is Vanilla JavaScript ? :  
<https://www.geeksforgeeks.org/what-is-vanilla-javascript/> (2024-10-21 確認)