E@sy コンフィグレータへの Vue.js の適用

ソリューション本部　開発二部
山下　剛史
杉山　晋梧
岡崎　充矩

1. はじめに

当社が自社開発した Web 見積作成支援システムとしてE@sy コンフィグレータがある。[1] これはWeb 上で商品の仕様選択を簡単に選択させるための仕様選択評価エンジンである。

今回、E@sy コンフィグレータに JavaScript の有名なフレームワークである Vue.js を適用した。本稿では Vue.js 適用により、E@sy コンフィグレータがどのように改善されたかについて述べる。

2. Vue.js 適用により期待できる改善効果

2. 1　保守性の向上

（1） Vue.js は近年注目の集まるフレームワーク

Vue.js は学習コストが低く、技術的障壁が低いとされる。[2] また、近年のトレンドであるため、情報収集が容易になってきている。

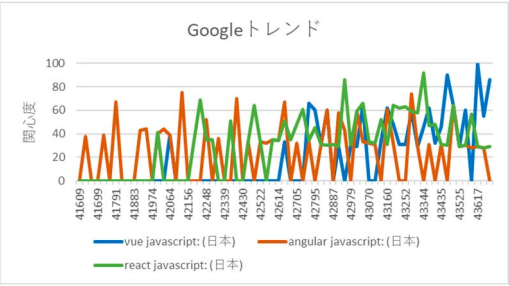


図 1　JavaScript フレームワークのトレンド

（2） MVVM

Vue.js はモデル(M)、ビュー(V)、ビューモデル(VM)の 3 層からなる MVVM と呼ばれるアーキテクチャを採用している。[3] これによりアプリケーション系の処理と画面制御系処理を明確に分離することができる。

① M 層(モデル)
JavaScript 内のデータやメソッドと連携し、 アプリケーション系の処理を担当する。
② V 層(ビュー)
ブラウザの DOM と連携し、画面制御系の処理を担当する。
③ V 層(ビューモデル)
M 層と V 層のオブジェクトを抱えることで、M 層と V 層の連携を行う。

Vue.js では、M 層と V 層のオブジェクトを抱えた VM オブジェクトを生成し、これを用いて制御を行う。

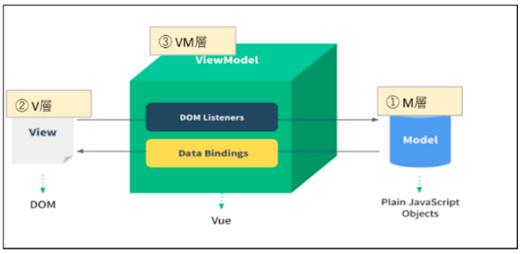


図 2 Vue.js における MVVM モデル
(https://jp.vuejs.org/index.html)

（3） 単一ファイルコンポーネントの利用

Vue.js には UI 単位でテンプレートを記述し、そのテンプレートにのみ有効なデータやメソッド、CSS を同一のファイル(以下、.vue とする)に定義できる「単一ファイルコンポーネント」という便利な仕組みがある。

.vue ファイルは以下の 3 つのタグで構成される。(図 3)

① template
UI のひな型を html ベースで記述する。
② script
データやメソッドを記述する。
③ style
CSS を記述する。

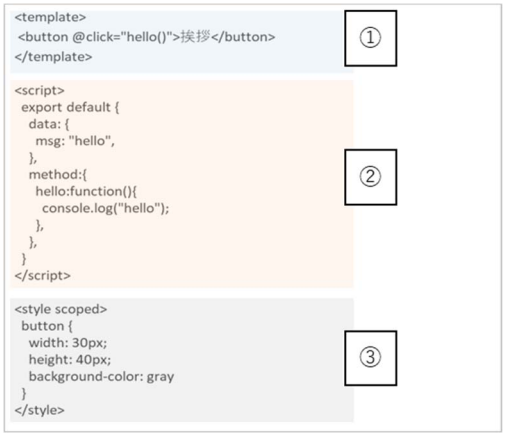


図 3 コンポーネント「HelloButton.vue」

VM オブジェクトは生成時、.vue ファイルを読み込み、その内容(M 層、V 層のオブジェクト)をデータとして抱える。①、③は MVVM モデルの概念的に V 層のオブジェクト、②は M 層のオブジェクトに該当する。

簡単な実装例として、クリックすると、「hello」とコンソールに出力するボタンのコンポーネント「HelloButton.vue」(図 3)を読み込むコンポーネントを示す。

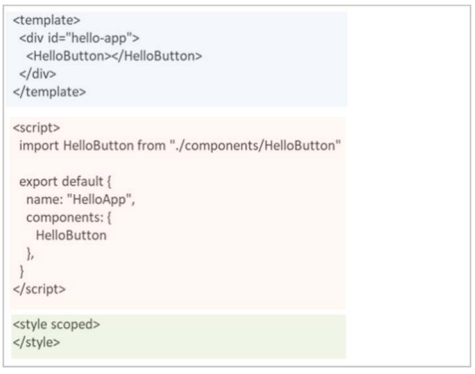


図 4 コンポーネント「App.vue」

コンポーネント「App.vue」が「HelloButton.vue」を読み込み、自身のテンプレートに HelloButton タグを記述する形で、使用する。コンポーネントを読み込む側を親コンポーネント、読み込まれる側を子コンポーネントという。

.vue に定義したデータやメソッド、CSS の影響範囲はこのファイルのテンプレートにのみ影響を及ぼす。このため、UI 単位で.vue ファイルを生成することには以下の利点があり、保守性の向上に寄与する。

① 1 つの.vue ファイルで UI の調整が可能。
② 子コンポーネントとして再利用が可能。

2．2 画面更新速度の向上

(1) 仮想 DOM

Vue.js には仮想 DOM [4] と呼ばれる仕組みがある。仮想 DOM は Vue.js 起動後、テンプレートの内容を元に JavaScript のオブジェクトとして生成される。その後、仮想 DOM の内容を DOM に連結する形で、ブラウザに表示される。(図 5)

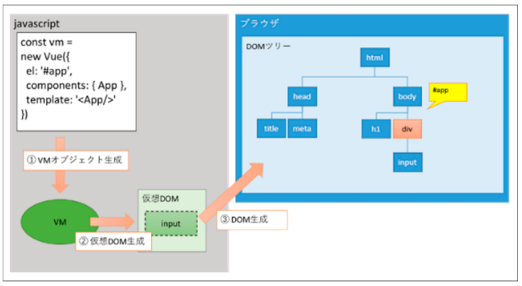


図 5 Vue.js 起動から DOM 生成まで

E@sy コンフィグレータの画面更新について、Vue.js 適用前はネイティブな JavaScript の記法を用いた DOM の直接操作(以下、直操作とする)によるものであった。対して、Vue.js 適用後は仮想 DOM を介して、DOM の更新を行うものであり、これは前者よりも更新速度の向上に期待できる。その理由を下記で説明する。

（2）Vue.js の DOM 更新

直操作の場合、要素を取得する際、ブラウザとのデータ通信が発生するのに対して、Vue.js の場合、仮想 DOM から、更新したい要素を割り出せるため、その必要がない。仮想 DOM による処理は無駄にブラウザとデータ通信を行わず、JavaScript 内の処理で済ませる仕組みであるため、更新速度の向上が期待できる。(図 6)

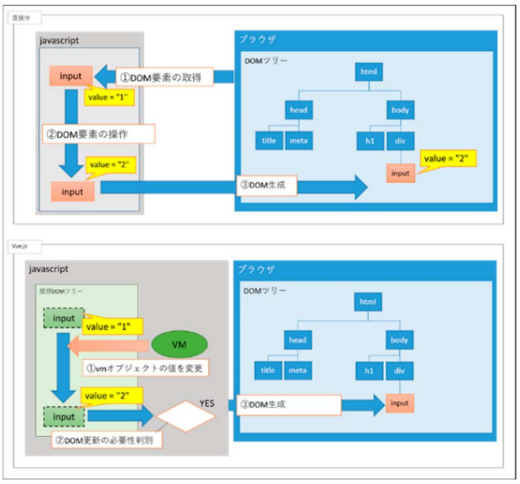


図 6 直操作と仮想 DOM の更新フロー

3. Vue.js 適用効果

3. 1 画面更新速度の向上

（1）E@sy コンフィグレータの課題

Vue.js 適用前の E@sy コンフィグレータは、仕様選択を行う度に JavaScript で次の仕様選択用の UI が表示されるよう、次の DOM 要素を動的に生成するプログラム実装となっている。

このことは、ブラウザの表示を調整する際に、以下の 3 つの課題が発生し、保守性を損なわせる要因となっていた。

① 課題 1
ネイティブな JavaScript の記述による。DOM の直操作のノウハウを知らなければメンテナンスが難しい。
② 課題 2
ノウハウがあったとしても、UI 修正の際、ネイティブな JavaScript の記述では一見してどこを修正すれば良いか分かりにくい。
③ 課題 3
新規の UI 生成プログラムを作成する際、その UI の DOM 構成(HTML 記述)となるように DOM を組み立てていく必要がある。

（2）Vue.js 適用の効果

上記の課題は.vue のテンプレートを使用することで全て解消できる。それぞれの課題が解消できる要因についていかに述べる。

① 課題 1 に対して
HTML ベースで記述するため、DOM の直操作の記述ノウハウは不要である。
② 課題 2 に対して
.vue ファイルは UI 単位で定義できるため、修正箇所の特定は容易である。
③ 課題 3 に対して
④ 生成したい UI の DOM 構成(HTML 記述) をほぼそのままテンプレートに記述するだけで良い。

3. 2 更新速度は体感的に不変

（1）DOM 更新速度の比較検証

Vue.js 適用前と Vue.js 適用後の更新速度の違いを検証した。(以下それぞれ A、B とする)。

A のプログラムを確認すると、 DOM 生成後、生成した DOM の全要素を取得し、JavaScript の変数としてキャッシュするという工夫がされていた。このため、逐一、ブラウザに対して操作対象となる要素を取得する必要がなくなり、その分、高速に動作していた。(図 7)

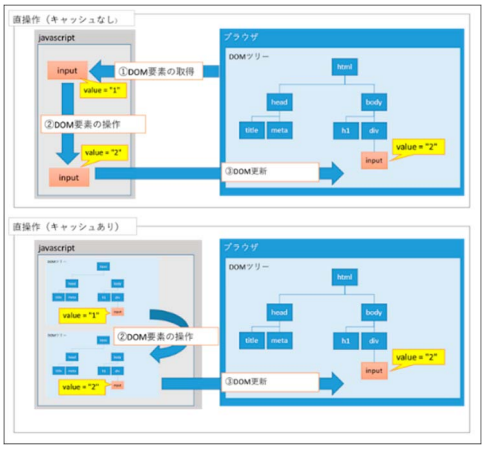


図 7 キャッシュの有無の違い

このように、既に十分なチューニングが為されていた結果、明瞭な改善が見受けられなかったといえるが、実際にどちらの方が高速であるかを比較検討するため、それぞれの実装方法に習い、DOM の生成から DOM の更新までを実行する 2 つの検証用プログラムを作成した。

① A Vue.js 適用前のプログラム

[DOM生成]
1. 指定した生成個数分、テキストをid付きで生成する。
2. 生成したテキストをgetElementByIdで全取得し、全ての更新対象となるDOMへの参照を配列で保持しておく。

[DOM更新]
1. 全ての更新対象となるDOMへの参照を保持した配列から、値を変更する。
2. 生成個数分、繰り返す。

```
// DOM生成
for (let i = 0; i < cnt; i++) {
 text = text + `<input type="text" id="{i}" value="1"/>`;
}
const tar = document.getElementById("tar") ;
tar.innerHTML = text
・・・・・
// 生成したDOMを配列に格納
for (let i = 0; i < cnt; i++) {
 text=  document.getElementById(String(i))
 cashe.push(text);
}
・・・・・
// 配列からDOMを更新
 cashe.forEach((d: HTMLInputElement, i: number) => {
  d.value = "2"
 }
```

図 8 プログラム A 抜粋

② B Vue.js 適用後のプログラム

[DOM生成]
1.親コンポーネント側で指定した生成個数分の要素を持つ更新用配列を用意する。その要素値は生成した1個の子コンポーネントの値と1対1でバインドさせる。
2.Vue.js起動時に配列の要素の数だけ、子コンポーネントを生成する。

[DOM更新]
1. 値更新用の配列から、値を変更する。
2. 生成個数分、繰り返す。

```
<template>
 <Speed
   v-for="(n,i) in objArr" :key="i" :idx="i+1" :val="n">
</Speed>
</template>

<script>
import Speed from './components/Speed'

export default {
 name: 'App',
 components: {
  Speed ,
 },
 data: function () {
  return {
    objArr:[],
  }
 },
}
</script>

<style>
</style>
```

図 9　親コンポーネント「App.vue」

```
<template>
   <input type="text" :value="val">
</template>

<script>
</script>

<style>
</style>
```

図 10　子テンプレート「Speed.vue」

（2）検証方法と検証環境

前述の 2 種のプログラム(A、B)それぞれで、生成個数を 1000 個と 10000 個の場合に分けて、下記の操作を 10 回ずつ行い、更新速度を実測した。

①検証方法

●操作1
　全てのテキストを異なる値に更新
●操作2
　テキストの半分は同じ値、残りを異なる値に更新
●操作3
　全てのテキストを同じ値で更新

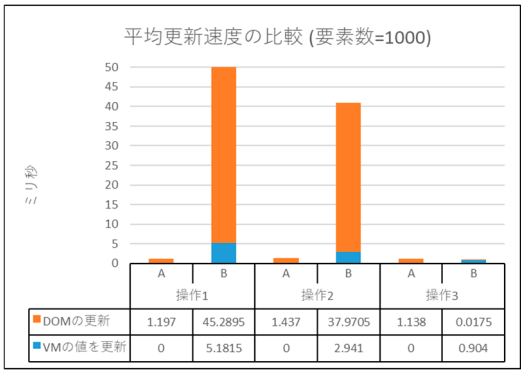
②検証環境

●OS: Windows10 Pro
●CPU: Intel® Core™ i3-6100U CPU @2.30GHz
●メモリ: RAM(16GB)
●ブラウザ: Google Chrome (version 76.0.3809.132)
●Vue.js: version 2.9.6

（3）検証結果と考察

① 検証結果

生成個数が 1000 個と 10000 個の場合でそれぞれの結果を示す。(図 11、図 12)



	操作1 A	操作1 B	操作2 A	操作2 B	操作3 A	操作3 B
DOMの更新	1.197	45.2895	1.437	37.9705	1.138	0.0175
VMの値を更新	0	5.1815	0	2.941	0	0.904

図 11　生成個数が 1000 個の場合

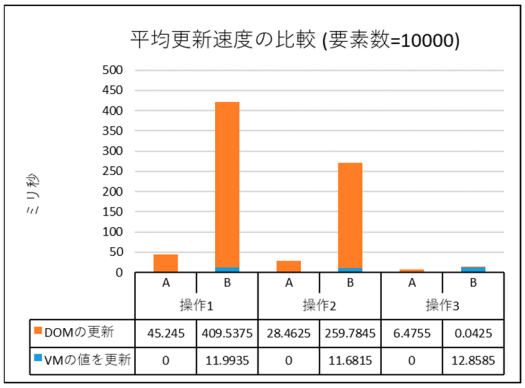


図 12　生成個数が 10000 個の場合

② 考察

全般的に A の方が高速であるが、DOM 更新速度の差がその要因となっている。全てのテキストの値を変更する操作 1 のとき、その差は最も大きい。生成個数が 1000 個のとき、A と B の更新速度の差は、最大で約 49 ミリ秒であった。対して、生成個数を 10000 個のとき、その差は最大で約 376 ミリ秒となり、更に拡大している。

3. 3　総括

更新速度に関しては Vue.js 適用前の方が早いことは否めない。ただし、実績として、E@sy コンフィグレータを適用したシステムにおいて選択項目数は多い事例でも 600 個である。その内 WEB に表示が必要、即ちブラウザにレンダリングを要求し、生成する DOM の個数は多くて 300 個程度である。生成個数が 1000 個のとき、更新速度の差は最大約 49 ミリ秒であり、それ以下の生成個数であることを踏まえると、この更新速度の悪化はほとんど問題とならない。ただし、今後より多くの選択項目の表示が必要なシステムに Easy コンフィグレータを適用するケースが生じた場合は、更新速度に支障をきたさないかを考慮すべきである。

対して、保守性に関しては、.vue ファイルのテンプレートに HTML ベースで記述できるようになったことで大きく向上した。E@sy コンフィグレータを他システムに適用する場合、顧客の要望に応じて、カスタマイズを行う必要性が生じるが、その作業負担が軽減され、保守コストの削減に繋がることが今後、期待できる。

4. おわりに

今回の改修について、速度面では劣化してしまったが、現実的画面要素数では無視できる範囲であり、保守性については、概ね、改善できた。

今後も、多様な技術を試して、自社エンジンの改良に貢献したい。

＜参考文献＞

1) 営業見積作成支援コンフィグレータ:E@SYコンフィグレータ
https://www.apptec.co.jp/service/easy.html

2) プログラミング初心者にオススメ！シンプルで学習障壁が低い JavaScript のフレームワーク「Vue.js」の特徴と学習方法
https://ferret-plus.com/5781

3) Vue.js ではじめる MVVM 入門
https://design.dena.com/engineering/vue-js%E3%81%A7%E3%81%AF%E3%81%98%E3%82%81%E3%82%8Bmvvm%E5%85%A5%E9%96%80/

4) Vue.js の仮想 DOM と差分レンダリングの仕組み
https://blog.engineer.adways.net/entry/2018/01/19/200000