報道各位

2019 年 11 月 6 日
株式会社インプレスR&D
https://nextpublishing.jp/

レガシーコードを理解して、モダンなアーキテクチャーに改善しよう！！

『迷わない！困らない！レガシーフロントエンド安全改善ガイド』発行

技術の泉シリーズ、11 月の新刊

インプレスグループで電子出版事業を手がける株式会社インプレス R&D は、『迷わない！困らない！レガシーフロントエンド安全改善ガイド』(著者:麦島 一)を発行いたします。

最新の知見を発信する『技術の泉シリーズ』は、「技術書典」や「技術書同人誌博覧会」をはじめとした各種即売会や、勉強会・LT 会などで頒布された技術同人誌を底本とした商業書籍を刊行し、技術同人誌の普及と発展に貢献することを目指します。

『迷わない！困らない！レガシーフロントエンド安全改善ガイド』
https://nextpublishing.jp/isbn/9784844378075

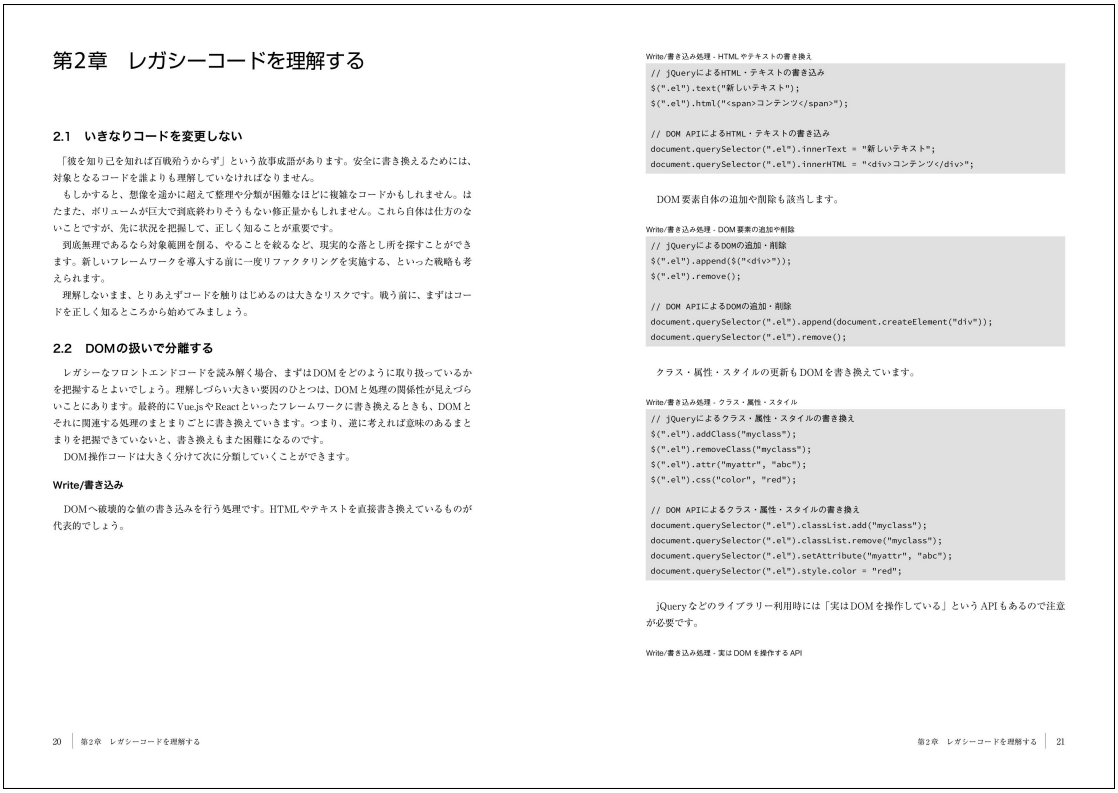
著者:麦島 一
小売希望価格:電子書籍版 1600 円(税別)／印刷書籍版 2200 円(税別)
電子書籍版フォーマット:EPUB3／Kindle Format8
印刷書籍版仕様:B5 判／カラー／本文 202 ページ
ISBN:978-4-8443-7807-5
発行:インプレス R&D

＜＜発行主旨・内容紹介＞＞

本書はレガシーなフロントエンドコードを安全かつ確実にモダンに改善していくためのノウハウをまとめた一冊です。筆者が経験したフロントエンドの改善経験をベースに、実践的で現場で使える内容になっています。また、jQuery で書かれたレガシーコードに Vue.js/TypeScript/Jest などを段階的に導入する流れを各章毎に「実践編」として掲載しており、実際に手を動かしながら学べます。

改善のための考え方や手法を知りたい方はもちろん、モダンなアーキテクチャーそのものを学びたい方にも最適な一冊です。
（本書は、次世代出版メソッド「NextPublishing」を使用し、出版されています。）

改善の第一歩として、レガシーコードを理解するためのポイントを解説



第2章　レガシーコードを理解する

2.1　いきなりコードを変更しない

「彼を知り己を知れば百戦危うからず」という故事成語があります。安全に書き換えるためには、対象となるコードを誰よりも理解していなければなりません。
もしかすると、想像を遥かに超えて整理や分類が困難なほどに複雑なコードかもしれません。はたまた、ボリュームが巨大で到底終わりそうもない修正量かもしれません。これら自体は仕方のないことですが、先に状況を把握して、正しく知ることが重要です。
到底無理であるなら対象範囲を削る、やることを絞るなど、現実的な落とし所を探すことができます。新しいフレームワークを導入する前に一度リファクタリングを実施する、といった戦略も考えられます。
理解しないまま、とりあえずコードを触りはじめるのは大きなリスクです。戦う前に、まずはコードを正しく知るところから始めてみましょう。

2.2　DOMの扱いで分離する

レガシーなフロントエンドコードを読み解く場合、まずはDOMをどのように取り扱っているかを把握するとよいでしょう。理解しづらい大きい要因のひとつは、DOMと処理の関係性が見えづらいことにあります。最終的にVue.jsやReactといったフレームワークに書き換えるときも、DOMとそれに関連する処理のまとまりごとに書き換えていきます。つまり、逆に考えれば意味のあるまとまりを把握できていないと、書き換えもまた困難になるのです。
DOM操作コードは大きく分けて次に分類していくことができます。

Write/書き込み

DOMへ破壊的な値の書き込みを行う処理です。HTMLやテキストを直接書き換えているものが代表的でしょう。

20　第2章　レガシーコードを理解する

Write/書き込み処理 - HTMLやテキストの書き換え

```
// jQueryによるHTML・テキストの書き込み
$(".el").text("新しいテキスト");
$(".el").html("<span>コンテンツ</span>");

// DOM APIによるHTML・テキストの書き込み
document.querySelector(".el").innerText = "新しいテキスト";
document.querySelector(".el").innerHTML = "<div>コンテンツ</div>";
```

DOM要素自体の追加や削除も該当します。

Write/書き込み処理 - DOM要素の追加や削除

```
// jQueryによるDOMの追加・削除
$(".el").append($("<div>"));
$(".el").remove();

// DOM APIによるDOMの追加・削除
document.querySelector(".el").append(document.createElement("div"));
document.querySelector(".el").remove();
```

クラス・属性・スタイルの更新もDOMを書き換えています。

Write/書き込み処理 - クラス・属性・スタイル

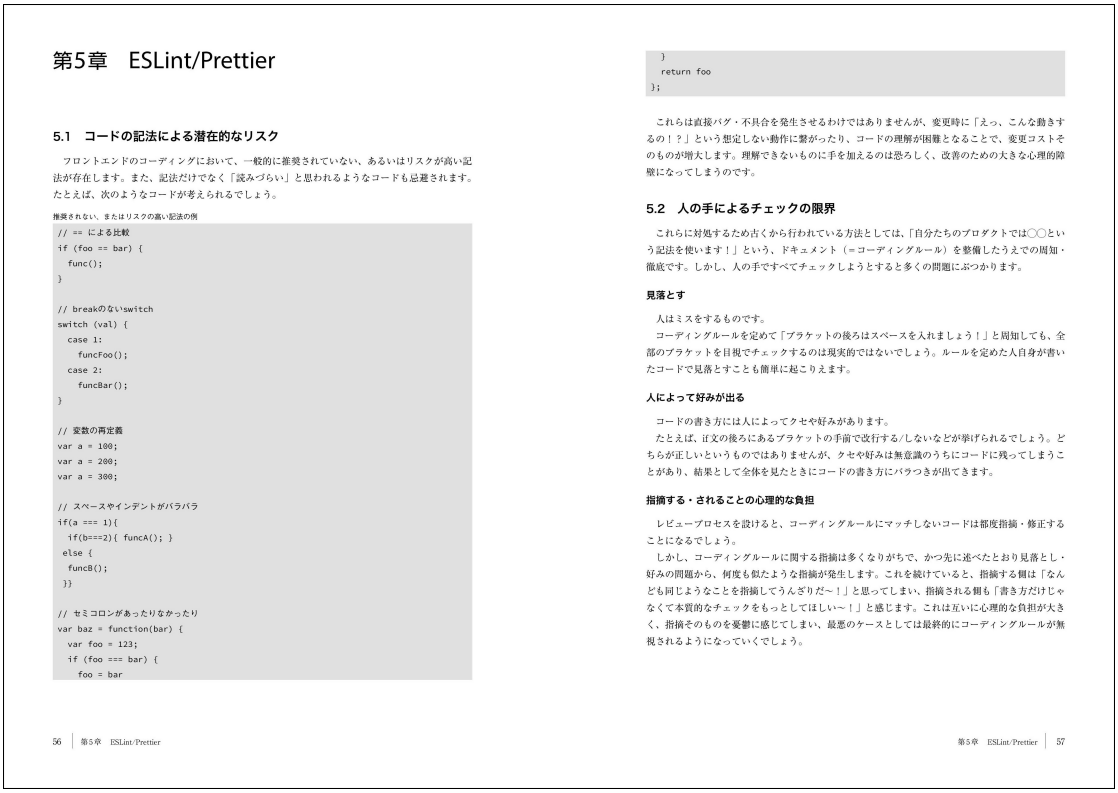
```
// jQueryによるクラス・属性・スタイルの書き換え
$(".el").addClass("myclass");
$(".el").removeClass("myclass");
$(".el").attr("myattr", "abc");
$(".el").css("color", "red");

// DOM APIによるクラス・属性・スタイルの書き換え
document.querySelector(".el").classList.add("myclass");
document.querySelector(".el").classList.remove("myclass");
document.querySelector(".el").setAttribute("myattr", "abc");
document.querySelector(".el").style.color = "red";
```

jQueryなどのライブラリー利用時には「実はDOMを操作している」というAPIもあるので注意が必要です。

Write/書き込み処理 - 実はDOMを操作するAPI

改善の際のコーディングで役立つツールを紹介



第5章　ESLint/Prettier

5.1　コードの記法による潜在的なリスク

フロントエンドのコーディングにおいて、一般的に推奨されていない、あるいはリスクが高い記法が存在します。また、記法だけでなく「読みづらい」と思われるようなコードも忌避されます。たとえば、次のようなコードが考えられるでしょう。

推奨されない、またはリスクの高い記法の例

```
// == による比較
if (foo == bar) {
  func();
}

// breakのないswitch
switch (val) {
  case 1:
    funcFoo();
  case 2:
    funcBar();
}

// 変数の再定義
var a = 100;
var a = 200;
var a = 300;

// スペースやインデントがバラバラ
if(a === 1){
  if(b===2){ funcA(); }
 else {
  funcB();
 }}

// セミコロンがあったりなかったり
var baz = function(bar) {
  var foo = 123;
  if (foo === bar) {
    foo = bar
  }
  return foo
};
```

これらは直接バグ・不具合を発生させるわけではありませんが、変更時に「えっ、こんな動きするの！？」という想定しない動作に繋がったり、コードの理解が困難となることで、変更コストそのものが増大します。理解できないものに手を加えるのは恐ろしく、改善のための大きな心理的障壁になってしまうのです。

5.2　人の手によるチェックの限界

これらに対処するため古くから行われている方法としては、「自分たちのプロダクトでは○○という記法を使います！」という、ドキュメント（＝コーディングルール）を整備したうえでの周知・徹底です。しかし、人の手ですべてチェックしようとすると多くの問題にぶつかります。

見落とす

人はミスをするものです。
コーディングルールを定めて「ブラケットの後ろはスペースを入れましょう！」と周知しても、全部のブラケットを目視でチェックするのは現実的ではないでしょう。ルールを定めた人自身が書いたコードで見落とすことも簡単に起こりえます。

人によって好みが出る

コードの書き方には人によってクセや好みがあります。
たとえば、if文の後ろにあるブラケットの手前で改行する/しないなどが挙げられるでしょう。どちらが正しいというものではありませんが、クセや好みは無意識のうちにコードに残ってしまうことがあり、結果として全体を見たときにコードの書き方にバラつきが出てきます。

指摘する・されることの心理的な負担

レビュープロセスを設けると、コーディングルールにマッチしないコードは都度指摘・修正することになるでしょう。
しかし、コーディングルールに関する指摘は多くなりがちで、かつ先に述べたとおり見落とし・好みの問題から、何度も似たような指摘が発生します。これを続けていると、指摘する側は「なんども同じようなことを指摘してうんざりだ～！」と思ってしまい、指摘される側も「書き方だけじゃなくて本質的なチェックをもっとしてほしい～！」と感じます。これは互いに心理的な負担が大きく、指摘そのものを憂鬱に感じてしまい、最悪のケースとしては最終的にコーディングルールが無視されるようになっていくでしょう。

実際の改善の例として Vue.js 環境への移行を予備知識から実践まで解説

第9章　Vue.js（移行の予備知識）

DOM中心アプローチと比較した場合のVue.jsの利点を学び、Vue.jsコードを書いていける環境を整えることができました。実際に移行を進めていきたいところですが、ゼロからの新規Vue.jsアプリケーション構築と異なり、レガシーフロントエンドコードからの移行ならではの問題点やコツといったものがあります。書き換えを行う前にあらかじめ予備知識として確認しておきましょう。

9.1　移行時に発生しやすい問題

jQueryなどのDOM中心アプローチによるコードをVue.jsへ段階的に置き換えていく場合、陥りやすい「よくある」トラブルがあります。意図しない不具合を踏む可能性を減らしたり、もし踏んでしまっても早めに解決できるよう、あらかじめ把握しておきましょう。

イベントバインドのタイミング

移行前のコードでクリックなどのイベントに対して処理を行っている場合、Vue.jsがマウントするタイミングとの兼ね合いでイベントが正常に動作しなくなる可能性があります。たとえば次のようなコードがあったとします。

イベントバインドを行うスクリプト

```
var button = document.querySelector(".button");

button.addEventListener("click", function() {
  console.log("clicked");
});
```

イベントバインド対象のHTML

```
<button class="button">ボタン</button>
```

"button"のクラスをもつ要素をクリックしたらログに出力するだけの簡単なものです。しかし、ボタン要素をコンポーネントに置き換えた場合どうなるでしょうか。

Button.vue

```
<template>
  <button class="button">ボタン</button>
</template>
```

120　第9章　Vue.js（移行の予備知識）

イベントバインドを行うスクリプト（Vue.jsを利用）

```
import Vue from "vue";
import Button from "./Button.vue";

var button = document.querySelector(".button");

button.addEventListener("click", function() {
  console.log("clicked");
});

new Vue(Button).$mount(".button");
```

イベントバインド対象のHTML

```
<div class="button"></div>
```

この場合、イベントリスナーに登録しているログ出力処理は実行されません。原因は至ってシンプルで、イベントバインド後にVue.jsがマウントされ再レンダリングされた時点でDOM要素が置き換わり、古いDOMと共にイベントバインドも破棄されてしまったのです。解消するためには、イベントバインドがVue.jsによるマウントより後のタイミングで実行される必要があります。

Vue.jsのマウント後にイベントバインドを実行する

```
import Vue from "vue";
import Button from "./Button.vue";

new Vue(Button).$mount(".button");

var button = document.querySelector(".button");

button.addEventListener("click", function() {
  console.log("clicked");
});
```

また、JavaScriptのDOMイベントは親の要素に伝搬していく仕組み（イベントバブリング）があります。これを利用して、実際に操作される親の要素でイベントを拾い上げる方法でも回避できます。次の例の場合ではイベントバインド対象はあくまでもbody要素なので、その配下の要素が入れ替わったとしても問題なく動作します。

イベントバブリングによる対処方法

```
import Vue from "vue";
import Button from "./Button.vue";
```

第9章　Vue.js（移行の予備知識）　121

＜＜目次＞＞

＜＜著者紹介＞＞

麦島　一
富山県在住のエンジニアでリモートワーカー。フロントエンドがメインだが、Ruby/Rails でサーバーサイドも書く。Toyama.rb というRubyコミュニティーを毎月主催。マルチカーソルをこよなく愛する。

＜＜販売ストア＞＞

電子書籍:

Amazon Kindle ストア、楽天 kobo イーブックストア、Apple Books、紀伊國屋書店 Kinoppy、Google Play Store、honto 電子書籍ストア、Sony Reader Store、BookLive!、BOOK☆WALKER

印刷書籍:

Amazon.co.jp、三省堂書店オンデマンド、honto ネットストア、楽天ブックス

※　各ストアでの販売は準備が整いしだい開始されます。

※　全国の一般書店からもご注文いただけます。

【インプレス R&D】 https://nextpublishing.jp/

株式会社インプレスR&D（本社:東京都千代田区、代表取締役社長:井芹昌信）は、デジタルファーストの次世代型電子出版プラットフォーム「NextPublishing」を運営する企業です。また自らも、NextPublishing を使った「インターネット白書」の出版など IT 関連メディア事業を展開しています。

※NextPublishing は、インプレス R&D が開発した電子出版プラットフォーム（またはメソッド）の名称です。電子書籍と印刷書籍の同時制作、プリント・オンデマンド（POD）による品切れ解消などの伝統的出版の課題を解決しています。これにより、伝統的出版では経済的に困難な多品種少部数の出版を可能にし、優秀な個人や組織が持つ多様な知の流通を目指しています。

【インプレスグループ】 https://www.impressholdings.com/

株式会社インプレスホールディングス（本社:東京都千代田区、代表取締役:唐島夏生、証券コード:東証 1 部 9479）を持株会社とするメディアグループ。「IT」「音楽」「デザイン」「山岳・自然」「旅・鉄道」「学術・理工学」を主要テーマに専門性の高いメデイア&サービスおよびソリューション事業を展開しています。さらに、コンテンツビジネスのプラットフォーム開発・運営も手がけています。

【お問い合わせ先】

株式会社インプレス R&D NextPublishing センター
TEL 03-6837-4820
電子メール: np-info@impress.co.jp