4X-01

Web Worker に DOM 操作機能を提供するフレームワークの提案

宇佐見 優一郎 † 早川 智一 †
明治大学理工学部 †

1. はじめに

Web Worker [1]（以下，Worker）は，Web ブラウザでの処理を効率化するための新技術である．Worker を用いることで並列処理が可能になり，画面の応答性の低下などの問題を低減できる．

しかし，Worker には DOM（Document Object Model）操作を直接は行えないという仕様上の課題がある．これによって，Worker は DOM 操作を外部（自身の生成元）に依頼することになる．

この課題の解決を目的としたフレームワークとして，Via.js [2] が存在する（2 章）．Via.js は，DOM 等の API（Application Programming Interface）を Worker 内で実現する機構を提供する．

しかし，Via.js には DOM の操作性に以下の改善点があると我々は考える：(1) DOM 要素のプロパティ値の取得に独自の記法を強制する；(2) メモリリークの可能性がある．

本稿では，上述の改善点 1 に着目したフレームワークを提案する（改善点 2 については別途報告する）．具体的には，提案フレームワークがソースコードを自動変換することで，開発者は Via.js 独自の記法を意識する必要がなくなる（3 章）．

我々は，提案フレームワークのプロトタイプを設計（4 章）・実装（5 章）し，独自の記法を用いることなく Worker 内に DOM 操作を記述できることを確認した（6 章）．

2. 関連技術

Via.js は，DOM 等の API を Worker 内で模倣するフレームワークである．Via.js と提案フレームワークは，Proxy オブジェクト [3] を利用して Worker に DOM 操作機能を提供する点で共通性があるが，DOM 要素のプロパティ値を取得する際の記述が異なる．Via.js では独自の非同期メソッドの呼び出しが必要となるが，提案手法ではソースコードを自動変換することで特殊な記法を不要とする．

WorkerDOM [4] は，DOM API の独自の実装を Worker に提供するフレームワークである．WorkerDOM と提案フレームワークは，Worker 内に DOM 操作を記述できる点で共通性があるが，その実現方法が異なる．WorkerDOM では，DOM API を独自実装することで，生の（ブラウザの提供する）DOM API と同様の記述を実現する．提案手法では独自実装の代わりに変換機構を用いることで，生の DOM API と同様の記述を実現する．

A proposal for framework that enables DOM manipulation in Web Workers

† Yuichiro Usami and Tomokazu Hayakawa, School of Science and Technology, Meiji University

3. 提案手法

3.1. 概要

図 1 に，提案フレームワークの概要を示す．提案フレームワークは，変換機構・Main 側フレームワーク・Worker 側フレームワークで構成される．処理の流れは次のとおりである：(1) 開発者は Main 側のソースコードとともに，Worker 側のソースコードを生の DOM API を用いて記述する；(2) 変換機構が，Worker 側ソースコードの中から DOM 要素のプロパティ値の取得操作を，Worker 側から Main 側に処理を依頼する形に変換する（DOM は Worker からは変更できないため）；(3) Main 側・Worker 側のソースコードを Web ブラウザが読み込み，Main 側ソースコード内で Worker が起動する；(4) Worker 側ソースコード内で DOM API の呼び出しが発生する；(5) Worker 側フレームワークが DOM API 呼び出しの内容を Main 側フレームワークへ送信する；(6) Main 側フレームワークが，Worker 側フレームワークから送られてきた DOM API 呼び出しを実行する．

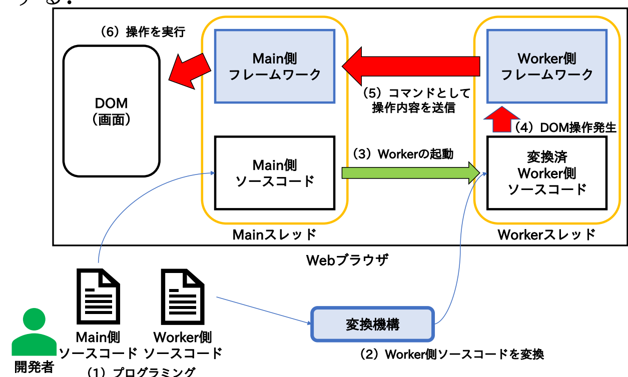

図 1 提案フレームワークの概要

Copyright ©2019 Information Processing Society of Japan.
All Rights Reserved.

3.2. 利用例

ソースコード 1・2 は，Via.js と提案フレームワークとで同一の処理を記述した場合の Worker 側のサンプルである．両ソースコードより，提案フレームワークは Via.js の 40 %以下の記述量（行数）で生の DOM API の記述を可能にしていることがわかる．

4. 設計

図 2 に変換機構の流れを示す．変換の流れは次のとおりである：(1) 開発者が作成した Worker 側ソースコードから抽象構文木（以下，AST : Abstract Syntax Tree）を生成する；(2) AST から，DOM 要素のプロパティ値の取得操作を探索し，それを非同期メソッドを用いた形で表すように AST を変形する；(3) AST から Worker 側ソースコードを生成する．

Worker 側フレームワークは，Worker 側ソースコードに Via.js 互換の API を提供する．DOM API の呼び出しが発生すると，Worker 側フレームワークは以下の処理を行う：(1) 一意な ID を持った Proxy オブジェクトを生成し，API 呼び出しの戻り値としてそのオブジェクトを返す；(2) API 呼び出しの内容と (1) で生成した一意な ID とをまとめたコマンドを Main 側フレームワークへ送信する．

Main 側フレームワークは，Worker 側フレームワークから送信されたコマンドの処理を行う．処理の結果得られた DOM 要素を，Worker 側から送信された ID と共に Map へと保存することにより，Worker 側で生成された Proxy オブジェクトと実際の DOM 要素との対応関係が保持される．

ソースコード 1 Via.js を用いたサンプル

```
 1 self.addEventListener("message", e => {
 2   if (e.data === "start") {
 3     importScripts("object.js","property.js","
         controller.js");
 4     Via.postMessage = (data => self.postMessage(
         data));
 5     Start();
 6   } else {
 7     Via.OnMessage(e.data);
 8   }
 9 });
10 async function Start() {
11   const document = via.document;
12   const [docTitle, docUrl] = await Promise.all([
13     get(document.title),
14     get(document.URL),
15   ]);
16   const h1 = document.createElement("h1");
17   h1.textContent = "Document title is:" + docTitle;
18   document.body.appendChild(h1);
19   const p = document.createElement("p");
20   p.textContent = "Document URL is:" + docUrl;
21   document.body.appendChild(p);
22 }
```

ソースコード 2 提案フレームワークを用いたサンプル

```
1 const docTitle = document.title;
2 const docUrl = document.URL;
3 const h1 = document.createElement("h1");
4 h1.textContent = "Document title is:" + docTitle;
5 document.body.appendChild(h1);
6 const p = document.createElement("p");
7 p.textContent = "Document URL is:" + docUrl;
8 document.body.appendChild(p);
```

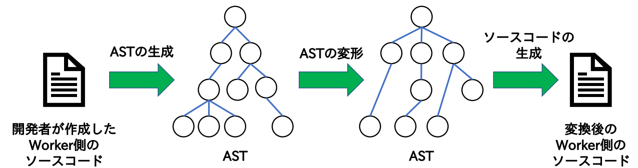

図 2 変換機構の流れ

表 1 変換機構の実装に用いたソフトウェア

ソフトウェア	バージョン	備考
Browserify	16.2.3	モジュールバンドラー
Esprima	4.0.1	AST の生成
Estraverse	4.2.0	AST の変形
Escodegen	1.11.0	ソースコードの生成

5. 実装

表 1 に，変換機構の実装に用いたソフトウェアを示す．導入の容易性を考慮し，今回は変換機構を Browserify [5] の Transform Module として実装した．ソースコードから AST への変換に Esprima [6] を用いた．AST の変形に Estraverse [7] を用いた．AST からソースコードへの変換に Escodegen [8] を用いた．

6. 評価

評価として，前述のソースコード 1・2 をブラウザ上で動作させ，その動作を確認した．その結果，どちらのソースコードでも同じ動作をすることを確認した．これより，DOM 操作機能の面で提案フレームワークは，独自の記法を用いることなく Via.js と同等の機能を提供していることがわかる．以上より，我々は提案フレームワークが Worker を用いた開発の一助たりうるという結論を得た．

7. おわりに

本論文では，Web Worker に DOM 操作機能を提供するフレームワークを提案した．今後は，1 章にて触れた改善点 2 を解決する手法を検討し，さらなる改善を行うことが課題である．

参考文献

[1] W3C: Web Workers, http://www.w3.org/TR/workers/.
[2] Scirra, A.: Via.js, https://github.com/AshleyScirra/via.js.
[3] ECMAScript: Proxy Objects, https://www.ecma-international.org/ecma-262/8.0/.
[4] AMP Project: WorkerDOM, https://github.com/ampproject/worker-dom.
[5] browserify.org: Browserify, http://browserify.org/.
[6] Hidayat, A.: Esprima, http://esprima.org/.
[7] Suzuki, Y.: Estraverse, https://github.com/estools/estraverse.
[8] Suzuki, Y.: Escodegen, https://github.com/estools/escodegen.

Copyright ©2019 Information Processing Society of Japan.
All Rights Reserved.