M-029

# Web Worker の学習コストと非同期処理との親和性とに着目した JavaScript スレッドライブラリの提案と試作
# A Proposal of JavaScript Thread Library Focused on Web Worker Learning Cost and Asynchronous Operation Descriptiveness

水村 総之介† 阿部 創志朗‡ 早川 智一‡
Sonosuke Mizumura Soshiro Abe Tomokazu Hayakawa

## 1. はじめに

Web Worker（以下，Worker）[1] は，JavaScript でマルチスレッドを実現する API（Application Programming Interface）である．元来，JavaScript は単一スレッドですべての処理を行う必要があったが，Worker を用いると複数スレッドに処理を分散できるため処理時間の短縮が可能になる．

しかし，Worker には 2 つの課題が存在する．1 つ目は，学習コストが高いという課題である．この理由として，他言語のスレッドライブラリ（以下，他言語のライブラリ）と比較して Worker は固有の API を持つことが挙げられる．2 つ目は，JavaScript の非同期処理の記法との親和性（以下，非同期処理との親和性）が低いという課題である．この理由として，Worker の処理結果が Promise で返らないため，async/await 構文等と併用するための仕組みを開発者が自ら実装する必要があることが挙げられる．

これらの課題の一方を解決する既存のスレッドライブラリは存在するが，両方同時に解決する技術は我々の調査では見当たらない（2 章）．具体的には，非同期処理の記法との併用がしにくいことや，固有の API を持つために学習コストが高いという課題をもつ．

本研究では，これらの課題を同時に解決する JavaScript のスレッドライブラリを提案する．提案ライブラリは，（1）他言語のライブラリに類似した API を持ち，（2）JavaScript の Promise との親和性が高い――という特徴を持つ．提案ライブラリを用いることで，Worker を初めて用いる開発者にも学習しやすく，async/await を用いた JavaScript の開発に組み込むことが可能となる．

## 2. 関連技術

### 2.1 Thread.js

Thread.js [2] は，Worker スレッドに処理対象の関数を渡すスレッドライブラリである．Thread.js は，生の Worker と比較してスレッド処理が記述しやすい特徴を持つが，処理結果を Promise ではなくコールバックで処理する必要があるため本研究の課題解決には適さない．

### 2.2 Workly

Workly [3] は，Worker を用いた処理対象の関数を容易に非同期化してくれるスレッドライブラリである．Workly は，生の Worker と比較してスレッド処理が記述しやすい特徴を持つが，独自の API を持つために学習コストが高く本研究の課題解決には適さない．

†明治大学大学院理工学研究科 Graduate School of Science and Technology, Meiji University
‡明治大学理工学部 School of Science and Technology, Meiji University

表 1 提案ライブラリと既存ライブラリとの比較

| | 非同期処理の記法との親和性が低い | 非同期処理の記法との親和性が高い |
|---|---|---|
| APIの学習コストが高い | Web Worker | Workly |
| APIの学習コストが低い | Thread.js | 提案ライブラリ |

コード 1 提案ライブラリの使用例

```
function isPrime(num) {
    return /* num が素数ならば true, そうでなければ false */;
}

async function main() {
    const thread = new Thread(isPrime);
    const result = await thread.start(/* 巨大な自然数 */);
    console.log(result); // 素数ならば true, そうでなければ false
}
```

## 3. 提案手法

本研究では，前述（1 章）の課題の解決のために，新しい JavaScript スレッドライブラリを提案する．具体的には，提案ライブラリは，（1）他言語出身のプログラマの学習が容易になるように，他言語のライブラリに類似した API を持ち，（2）可読性の高いプログラムを記述できるように，処理結果を Promise として返す――ことで課題の解決を試みる．

表 1 に既存ライブラリと提案ライブラリとの比較を示す．我々は，提案ライブラリを既存ライブラリと比較し，非同期処理との親和性が高く，API の学習コストが低くなるように設計した．

Worker と提案ライブラリとの比較をコード 1 とコード 2 とに示す．着目すべき点として，（1）生 Worker と比較して簡潔に処理を記述できること，（2）API が他言語のライブラリに近いこと，（3）処理結果が Promise で返るために非同期処理との親和性が高いこと――が挙げられる．

## 4. 設計

我々は，提案ライブラリを設計するために，他言語のライブラリを調査した（表 2）．言語選定として，IEEE のランキング [4] の 1〜4 位の言語から C/C++を除外した（5 位は JavaScript）．C/C++を除外した理由として，これらのスレッドライブラリはポインタを前提としているが，JavaScript にはポインタが無いためである．

コード 2　Web Worker の使用例

```
// Main 側
async function main() {
  const result = await isPrimeWorker(/* 巨大な自然数 */);
  console.log(result); // 素数ならば true, そうでなければ false
}

function isPrimeWorker(num) {
  return new Promise((resolve, reject) => {
    const worker = new Worker(/* Worker 側のソースファイル */);
    worker.onmessage = function(e){ return resolve(e.data); }
    worker.postMessage(num);
  });
}

/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

// Worker 側
function isPrime(num) {
  return /* num が素数ならば true, そうでなければ false */;
}

self.addEventListener('message', function(e) {
  self.postMessage(isPrime(e.data));
}, false);
```

表 2　他言語スレッドライブラリの利用方法

| ライブラリ名 | スレッド生成方法 | スレッド処理記述方法 | スレッド開始方法 | 処理結果受け取り方法 |
|---|---|---|---|---|
| Threadクラス（Java） | Threadインスタンスを生成する | メソッドとして記述する | startメソッドを呼び出す | クラスのフィールドを利用する |
| threadingモジュール（Python） | Threadインスタンスを生成する | 関数として記述する | startメソッドを呼び出す | スレッド間で共有する変数を利用する |

我々は表 2 を踏まえて，提案ライブラリの設計を，他言語のライブラリの API に類似し，非同期処理との親和性が高くなるようにした．具体的には，提案ライブラリは，(1) Thread インスタンスを生成することでスレッドを生成し，(2) スレッドに実行させる処理を関数で記述し，(3) start メソッドを呼び出すことでスレッドを開始し，(4) 処理結果を Promise として返す．

提案ライブラリの利用者（以下，利用者）がライブラリを呼び出してから処理結果を受け取るまでの処理の流れを図 1 に示す．具体的には次のとおりである：(1) 利用者は提案ライブラリの Thread インスタンス生成時に関数を渡し，start メソッドを実行する，(2) 提案ライブラリは渡された関数から Worker スレッドを生成し，start メソッドの実行で Worker を開始する，(3) Worker スレッドは渡された関数を実行する，(4) 提案ライブラリは処理結果を通知する Promise を利用者に返す，(5) Worker スレッドは実行した関数の結果を提案ライブラリに送信する，(6) 処理結果を元に Promise が解決する，(7) 利用者が Promise から処理結果を受け取る．

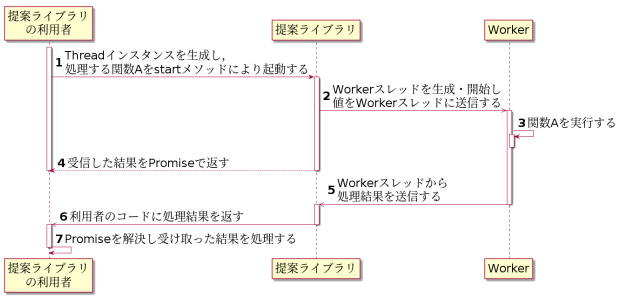


図 1　提案ライブラリのシーケンス図

表 3　既存ライブラリとの処理速度比較

| 判定する素数 | Web Worker（ms） | Thread.js（ms） | Workly（ms） | 提案ライブラリ（ms） |
|---|---|---|---|---|
| 2 | 153 | 191 | 135 | 125 |
| 67,280,421,310,721 | 334 | 307 | 301 | 351 |
| 9,007,199,254,740,881 | 2,988 | 3,092 | 2,990 | 3,050 |

表 4　既存ライブラリとの利用方法比較

| ライブラリ名 | スレッド生成方法 | スレッド処理記述方法 | スレッド開始方法 | 処理結果受け取り方法 |
|---|---|---|---|---|
| Web Worker | Workerインスタンスを生成する | 別のファイルに記述する | スレッドにメッセージを送信する | メッセージとして受け取る |
| Thread.js | Threadインスタンスを生成する | 関数（クロージャ）として記述する | onceメソッドを呼び出す | メソッドチェーンを利用する |
| Workly | proxyメソッドを呼び出す | 関数・クラスとして記述する | proxyメソッドの返り値（関数）を呼び出す | Promiseを利用する |
| 提案ライブラリ | Threadインスタンスを生成する | 関数として記述する | startメソッドを呼び出す | Promiseを利用する |

## 5.　評価

我々は，提案ライブラリの有用性を確認するために，処理速度と記述法とを比較評価した．

表 3 に，既存ライブラリと提案ライブラリとの処理速度の比較結果を示す．具体的には，Worker で 1 桁・14 桁・16 桁の素数判定を行い，Worker の生成から処理結果が返るまでの時間の平均である（5 回試行）．表より，各ライブラリの処理速度に有意な差は見られないことがわかる．

表 4 に既存ライブラリと提案ライブラリとの記述方法の比較を示す．表より，(1) Thread.js は Promise を利用せずに処理結果を返すため非同期処理との親和性が低い，(2) Workly は proxy メソッドでスレッドを生成するなど独自の API を持つため学習コストが高い――ことがわかる．

また，表 2 と表 4 から，提案ライブラリは，(1) スレッドの各操作方法については，Java 言語や Python などのライブラリと類似させたため，それらの言語の知識を持つ開発者にとっては学習コストが低いことが期待でき，(2) Promise で処理結果を返すため非同期処理との親和性が高い――という結論を得た．

## 6.　おわりに

本論文では，他言語のライブラリと類似した API を持ち非同期処理との親和性が高い JavaScript スレッドライブラリの提案と試作を行った．提案ライブラリを評価した結果，(1) 他言語のライブラリと類似した API を実現できたことと，(2) 既存ライブラリと比較して性能に遜色が無いこと――が確認できた．今後の展望として，提案ライブラリと他言語のライブラリとの詳細な API 比較や，Java 言語や Python に馴染んだプログラマに提案ライブラリを利用させてコードの行数や必要工数の比較等の評価を行う予定である．

### 参考文献

[1] WHATWG: Web Workers, https://html.spec.whatwg.org/multipage/workers.html#workers.
[2] deep rain.com: Thread.js, https://threadjs.deep-rain.com/.
[3] Shihn, P.: Workly, https://github.com/pshihn/workly/.
[4] IEEE: The Top Programming Languages, https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020.