JavaScript における非同期処理に対応した層活性手法
Layer Activation for Asynchronous Operations in JavaScript

鳥海健人 †, 福田浩章 †
Kento Toriumi†, and Hiroaki Fukuda†
† 芝浦工業大学　工学部情報工学科
†Information Engineering, Shibaura Institute of Technology.

要旨
文脈指向プログラミング (COP) は，文脈に依存した振る舞いをモジュール化するためのプログラミング手法である．JavaScript を COP の設計に則って記述することができる ContextJS は, Web 技術として頻繁に用いられる非同期処理を前提とされていない．本研究では, 非同期処理に対応した層活性手法を提案する．Promise やイベント駆動など, 複数種類の非同期タスクを監視し, 層の活性化状態を管理する．そして, ContextJS の拡張機能としてこれらの提案手法を実装する．

1. はじめに

文脈指向プログラミング (COP: Context Oriented Programming[1]) は, 文脈に依存した振る舞いをモジュール化するためのプログラミング手法である．COP の文脈とは, プログラムから観測できる外部環境やシステムの内部状態で, 時間や場所とともに変化し, それがプログラムの様々な実体の実行に影響を与えるものである [1]．COP は, プログラム中に横断的に存在する文脈に依存した振る舞いを層 (Layer) としてまとめ, その層を活性化・非活性化させることで現在の文脈に即した振る舞いに切り替える．例えば, スマートフォンの画面の向きによってページのレイアウトやデザインを柔軟に調整するレスポンシブデザインに対応した Web サイトの場合, クライアントのスマートフォンが縦向きか横向きかという状況は文脈に相当する．Web サイトは, これらの文脈によってレイアウト (ボタングループを全て表示・ハンバーガーメニューに格納) や特定のボタンを押した時の振る舞い (ドロップダウンリストを右に展開・下に展開) などを変化させる．これらは文脈に依存した振る舞いと呼ばれ, 様々なプログラムに横断して存在する．そのため, COP は文脈を層という単位に分割し, 文脈に依存した振る舞いをモジュール化している．

COP は, 様々な言語でライブラリとして提供されており, JavaScript のライブラリである ContextJS[1] もその一つである [2]．図 1 は ContextJS を用いてボタングループを表示する記述例である．

```
1  class ButtonGroupManager {
2      show() {
3          // ハンバーガーメニューに格納
4      }
5  }
6  const landscapeLayer = layer("landscapeLayer");
7  landscapeLayer.refineClass(ButtonGroupManager, {
8      show() {
9          // ボタングループを全て表示
10     }
11 });
12 const buttonGroupManager = new ButtonGroupManager();
13 withLayers([landscapeLayer], () => {
14     buttonGroupManager.show(); // ボタングループを全て表示
15 });
```

図 1: ContextJS を用いてボタングループを表示する例

1〜5 行目は, ボタングループの表示を担当する ButtonGroupManager クラスを定義している．6 行目は, 画面が横向きである状態 (文脈) に対応した landscapeLayer という Layer オブジェクトを生成している．7〜11 行目では, refineClass を用いて, ButtonGroupManager クラスに対応する部分メソッド show を定義している．部分メソッドとは, 層 (landscapeLayer) 活性化時にデフォルトメソッドの振る舞いを上書きして実行するメソッドである．13〜15 行目では, withLayers でブロック内が landscapeLayer で活

[1] https://github.com/LivelyKernel/ContextJS

性化されたことによって, 14 行目の show に 8〜10 行目の部分メソッド show が適用されボタングループを全て表示する振る舞いに切り替わる.

JavaScript を用いた Web 開発では, リクエスト処理やイベント処理といった非同期処理が頻繁に用いられる. しかし, 現状の ContextJS には非同期処理を前提としていない. 例えば, withLayers 内で非同期関数 (Promise) を実行し, 非同期タスク (then に渡すコールバック) をスケジュールする. withLayers 内に記述しているため, 非同期タスクに層の活性化状態が適用されることを期待する. しかし, 非同期タスクが遅延されている間, withLayers 外の処理が実行されて活性化されていた層は非活性化されてしまう. その後, 遅延された非同期タスクが実行されても活性化されていた層は復元されず, 非活性化状態のままとなり意図しない動作となる.

本研究では, 非同期処理に対応した層活性手法を提案する. Promise やイベント駆動など, 複数種類の非同期タスクを監視し, 層の活性化状態を管理する. そして, ContextJS の拡張機能としてこれらの提案手法を実装する.

2. 背景と問題点

2.1.COP の層活性手法

ダイナミックエクステントとは, 活性化状態をブロックに限定する層活性化手法である [1]. ContextJS では活性化に withLayers 構文, 非活性化に withoutLayers 構文を用いる. ブロックがネストした場合は内側のブロックが優先される. このようなブロック構造を用いることで，活性化を実行系列上のある範囲内に限定し，振る舞い同士が意図せずに衝突するなどの問題を回避している.

2.2. 非同期処理における層活性の問題点

図 2 は, landscapeLayer(画面が横向きである文脈に対応した層) を活性化した状態でドロップダウンボタンにクリックイベントを定義している実装例である. 1 行目でダイナミックエクステントを用いて landscapeLayer を活性化している. 2 行目でドロップダウンボタンの要素を取得し, 3 行目で addEventListener により非同期タスク (クリックイベントのコールバック) を設定している. その後, withLayers 外の処理が実行されて landscapeLayer は非活性化される. ボタンをクリックすることでコールバックが呼び出されるが, その時 landscapeLayer は非活性化されているため, 横向き表示にも関わらずドロップダウンリストは下に表示されてしまう [4].

```
1 withLayers([landscapeLayer], () => {
2     const dropDownButton = document.getElementById("dropDownButton");
3     dropDownButton.addEventListener("click", () => {
4         dropDownManager.show();
5     });
6 });
```

図 2: ダイナミックエクステントで層活性する非同期処理の記述例

3. 提案と実装

本研究では, JavaScript における非同期処理に対応した層活性手法を提案する.

3.1.JavaScript における非同期タスク

JavaScript の非同期タスクは以下の 3 つに分類できる[2].

MicroTask：
現在実行している非同期処理の直後に必ず 1 回だけ実行されるタスクであり, キャンセルすることはできない. 例えば, Promise の then 構文に渡すコールバックがこれに相当する.

MacroTask：
遅延実行されて, 1 回以上動作するタスク. 多くの場合キャンセル可能であり, 遅延時間を知ってい

[2]https://angular.jp/guide/zone

る. 例えば, setTimeout 関数や setInterval 関数の第一引数に渡すコールバックがこれに相当する.

EventTask：
あるイベントが発生したときに動作するタスクであり, いつ実行されるかは不明である. 例えば, addEventListener の第 2 引数に渡すコールバックがこれに相当する.

3.2.Zone

本研究では, 非同期処理に対応した層活性手法の提案を実現するために, Zone[3]を用いる. Zone は, Dart 言語発祥の技術であり, 非同期処理の実行を監視することができる.

JavaScript では Zone の機能を提供する以下のライブラリが存在する.

zone.js[4]：
フロントエンドフレームワーク Angular の一部として作成された. MicroTask・MacroTask・EventTask の JavaScript における全ての種類の非同期タスクに対応している.

Dexie.Promise[5]：
IndexedDB において, 進行中のデータベーストランザクションを維持するためのカスタム Zone システム. zone.js に依存しておらず, 他の Zone システムより軽量である. Dexie.Promise は, Promise(async/await) などの MicroTask 間の Zone のみを維持し, MacroTask・EventTask などの他の非同期タスクに Zone を伝播しない.

JavaScript を用いた Web 開発において, リクエスト処理などにあたる MicroTask はもちろん, 他の非同期タスクも対応すべきである. 特に addEventListener などに用いられる EventTask は対応する必要がある. したがって, 本研究では非同期処理の実行を監視する時, zone.js ライブラリを用いる. しかし, 現状の zone.js ライブラリは MicroTask に相当する async/await に対応していない [4]. よって, 本研究では async/await の対応も行う.

3.3. 提案手法

ContextJS では, ダイナミックエクステントによる活性化状態を LayerStack というグローバルな配列で管理している. 図 3 は, ダイナミックエクステントにより landscapeLayer を活性化した時の LayerStack の例である. LayerStack の要素はキーに withLayers(withLayers ブロックの場合), または withoutLayers(withoutLayers ブロックの場合) を持ち, 値に対象の Layer オブジェクトの配列を持つオブジェクトである. これらのオブジェクトをフレームと呼ぶ. 活性化された順に LayerStack にフレームが push されていき, withLayers ブロック外の処理が実行されるとそのフレームは pop され非活性化される.

```
1 [
2     { withLayers: [landscapeLayer] },
3 ]
```

図 3: LayerStack の例

本研究では, この LayerStack の要素を非同期タスクの開始時と終了時に動的に操作することによって, 非同期処理に対応した層活性手法を実現する. 図 4 に, 非同期処理に対応したレイヤースタックの様子を示す. まず, ダイナミックエクステントにより landscapeLayer を活性化している. ここで LayerStack にフレームが push される (図 4-(1)). addEventListener により非同期タスク (クリックイベントのコールバック) をスケジュールする (図 4-(2)). その後, withLayers ブロック外の処理が実行されてフレームが pop されたことにより landscapeLayer は非活性化される (図 4-(3)). クリックされた時, スケジュールされた非同期タスクが実行される. この時, zone.js により非同期タスクの開始が検知され, LayerStack にフレームを push する (図 4-(4)). その後非同期タスクが終了し, 同様に zone.js により非同期タスクの終了が検知され, LayerStack からフレームを pop する (図 4-(5)).

[3]https://dart.dev/articles/archive/zones
[4]https://github.com/angular/angular/tree/master/packages/zone.js
[5]https://dexie.org/docs/Promise/Promise.PSD

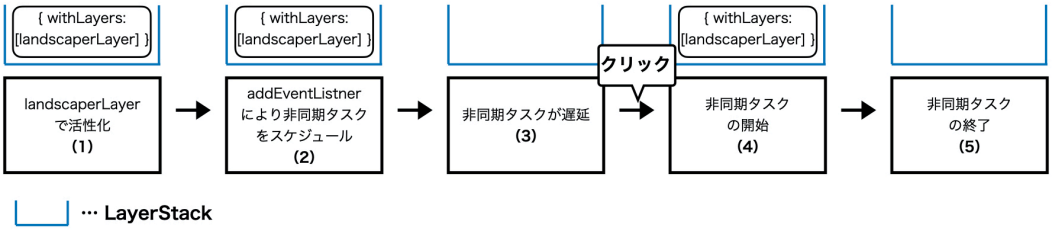

図 4: 非同期処理に対応したレイヤースタックの様子

本研究では, 非同期タスクの開始時と終了時を検知するにあたって zone.js が提供する以下の 3 つのライフサイクルメソッド[6]を用いる.

onInvokeTask：
非同期タスクが実行されるときにトリガーされる. このメソッドにより非同期タスクの開始を検知する.

onHasTask：
非同期タスクの実行待ち状態が変化した時にトリガーされる. このメソッドがトリガーされ, Zone 内に非同期タスクが存在しなかった時に非同期タスクの終了を検知する.

onInvoke：
Zone 内で同期関数が実行される直前にトリガーされる.

onInvokeTask メソッドは, 全ての種類の非同期タスクの開始を検知することができるが, onHasTask メソッドは, EventTask に相当する非同期タスクの終了をトリガーすることができない. これは EventTask がいつ実行されるかが不明なため, zone.js が EventTask の実行待ち状態を管理していないためである. そのため, onInvoke メソッドにより EventTask(クリックイベントのコールバック) の開始直前を検知し, コールバックをラップすることで開始・終了を検知する. async/await は, zone.js で対応されていないため onInvokeTask メソッドなどで非同期タスクの開始時・終了時を検知することができない. そのため, async/await は zone.js を使用せず, 非同期タスク (await によって遅延された処理) の開始時・終了時に LayerStack を動的操作するメソッドをフックすることで解決する.

4. 今後の予定

現在, ContextJS の拡張ライブラリを実装中である. JavaScript における 3 種類の非同期タスクに対応した 3 つの Web 開発上でのユースケースのサンプルコードを作成する. ContextJS のみを用いた場合と提案手法をもとに実装した ContextJS の拡張ライブラリを用いた場合とで, それぞれのコードについての考察や議論を行う.

参考文献

[1] 紙名 哲生, “文脈指向プログラミングの要素技術と展望, ” コンピュータソフトウェア, Vol.31, No.1, 2014, pp.3-13.
[2] Malte Appeltauer, Robert Hirschfeld, ichael Haupt, Jens Lincke, Michael Perscheid, “A Comparison of Context-oriented Programming Languages, ” COP ’09: International Workshop on Context-Oriented Programming, 2009, pp.1-6.
[3] Malte Appeltauer, Robert Hirschfeld, Michael Haupt, Hidehiko Masuhara, “ContextJ: Context-oriented Programming with Java, ” コンピュータソフトウェア, Vol.28, No.1, 2011, pp.272-292.
[4] Stefan Ramson, Jens Lincke, Harumi Ogura, Robert Hirschfeld, “Layer Activation with Dynamic Extent across Logically-connected Asynchronous Operations, ” Proceedings of the 12th International Workshop on Context-Oriented Programming and Advanced Modularity, 2020.

[6]https://angular.jp/guide/zone