

ClosureLibrary を使用した GUI の開発

ソリューション本部 システム開発部

横田 英之

1. はじめに

ClosureLibrary とは Google 社が提供する Web アプリケーション向けの Javascript フレームワークである。もともとは Google 社が自社で利用するための内製のライブラリであったが、2009 年 11 月にオープンソースとして提供された。ClosureLibrary は Gmail や GoogleDocs などの Google のアプリケーションで採用されており、そこで使われている UI ウィジェットを多数含んでいる。機能としては文字列操作、DOM・イベントといった基本的なものから、Ajax 通信、UI ウィジェット、テストフレームワークなど網羅的に全範囲をカバーするものを提供している。

2. ClosureLibrary を使用した理由

今回は HTML 上で、ボタンやプルダウンなどの GUI 部品をドラッグしてレイアウトする、VisualStudio のフォームのようなものを開発した。

ClosureLibrary を採用した理由は、サイボウズの kintone は ClosureLibrary を使っているらしいというマネージャの一言がきっかけであった。

似たようなものだからできるだろうということだけが根拠であった。少々無謀な挑戦であったと思われる。

3. 当初の想定との相違

作業を始めた当初は ClosureLibrary に GUI をレイアウトする高水準の機能がついていると思っていた

が、実際には jQuery のような低水準の機能を提供するライブラリであり、自分自身で GUI をレイアウトする仕組みをゼロから作成する必要があった。当初は本当にこれで作れるのだろうかと不安を覚えた。Web 画面を作成する場合、HTML 構造は既に出来上がっており、そこに Javascript を適用して、機能拡張するというイメージで開発することになる。しかし今回の GUI の開発では、以下の点で従来のイメージとの相違があり、感覚的に作業を進めることが、困難であった。

- ① ClosureLibrary の GUI 部品は HTML 構造には現れない多数の情報を裏側に持っている。HTML 構造とは別に多数の項目の情報をいかにして Javascript 側に持たせて、HTML 構造と対応づけるか。
- ② GUI 部品の追加や移動により HTML 構造自体が動的に変化していく。動的に変化する構造に対していかにして Javascript での機能拡張(ドラッグ可能など)を適用していくか。

ClosureLibrary が提供する Component クラス(GUI 部品のもっとも抽象的なクラス)の特徴そのものとも言えるが、以下の考え方に発想転換することで難点であった事項が解消されて開発できるようになった。

- ① Component の継承クラスでは、HTML 構造を気にせず情報を自由に持たせてよい。
- ② クラスを new 句でインスタンス化すると、Javascript 上でメモリにインスタンスが確保されるとともに、HTML 構造上にも同時に要素が作成される(具体的には<DIV>タグ)。インスタンスを HTML 構造上どのような要素として見せるかは、オーバーライド

できるメソッドがあるのでそこに書けばよい。インスタンス化後は、メモリ上のインスタンスと HTML 構造が対応付いた状態で取り扱うことができる。インスタンスを削除すると HTML 構造上でも消去される。

- ③ インスタンス化によって作成された HTML 構造に対して、作成と同時にドラッグ可能などの Javascript による機能拡張を適用できる。これもオーバーライドできるメソッドがあるのでそこに書けばよい。

4. ClosureLibrary でのオブジェクト指向

Javascript でのオブジェクト指向は単純なクラスやカプセル化までは書きやすいが、継承は構文が用意されていないため特別なテクニックが必要であった。ClosureLibrary はきっちりとしたオブジェクト指向開発の基盤を提供してくれる。Java や C# のようなクラスの継承構文が提供される。クラスの継承に伴う多態性やメソッドのオーバーライドなどオブジェクト指向関係全般がサポートされる。今回の GUI の開発は、「ドラッグ可能な GUI 部品」という抽象クラスがあり、それを継承してボタン、ドロップダウン、テキストボックスなどの具体的な GUI 部品を用意するという設計であったため、まさにオブジェクト指向が必要となるケースであった。これは、ClosureLibrary が提供する構文に当てはまるケースであり、クラスの継承や多態性の面で、記述の困難さや、バグに遭遇したことはなかった。ClosureLibrary でのオブジェクト指向は使用にあたって信頼できるという印象である。

5. ClosureLibrary の得手不得手

ClosureLibrary は高品質なライブラリであるが、万能ではない。ClosureLibrary はライブラリの名前空間がきっちりと分けられており、依存関係が明確になる

ように設計されている。そのため、依存関係が問題となる大規模な開発でも破綻しない点が特徴として挙げられる。Javascript で GUI を中心としたデスクトップライクな Web アプリを提供するのに向いていると思われる。

反面、主に jQuery の出番となるような、HTML にインタラクティブさを手軽に付け加えたいという場面には不向きである。ClosureLibrary の構文は、jQuery のような他のフレームワークと比べると冗長である。使いたい機能は前もって `goog.require('パッケージ名')` という構文で宣言しておく必要がある (Java で言うところの `import` 宣言)。この宣言が漏れていた場合は、実行時エラーとなるので発見しにくいバグになる。一見動作している様に見えるため、たびたび悩まされた。各命令も、パッケージ名を先頭に付加する必要があり、`goog.style.setStyle()` というような記述になる。jQuery ならば \$1 文字で済む所なので、開発にあたって非常にストレスとなる部分である。タイプミスによるバグの解決に相当の時間を要した。ClosureLibrary に合わせたコード補完をするエディタを切望したが、現在でもそのようなエディタは存在しないので構文が冗長である問題はそのまま残っている。

6. ClosureLibrary の情報源

ClosureLibrary で開発を進めるにあたって一番困ったことは情報源が少ないことである。2015 年 8 月現在でも Web 上の情報、書籍を問わず情報源が不足している。最も詳しいのが公式の APIDoc とデモという状況である。

Web 上の情報源では、試しに使ってみたインプレッション系の情報が若干ヒットする程度で、本格的な How to の情報源は見当たらない。ClosureLibrary

については、自分自身が書いているソースが、もしかしたら世界最先端ではないかと思われる状況であり、習得コストが高くつくという印象がぬぐえない。

書籍についても不足しており、日本語の情報源は「ClosureLibrary プログラミングガイド」という書籍のみである。この書籍の内容としては広く浅く記述されているのみで、肝心の UI コンポーネントの例題については妙に難しい書き方をされているので、学習用の教材としては、あまり適切ではない。英語の本ならば「Closure:The Definitive Guide」というものがあるが、どちらの本も 2010 年頃の出版であり、それ以降現在まで新しい本が出版されていない。関連書籍の出版状況が ClosureLibrary をとりまく環境そのものを表している様に思われる。「ClosureLibrary 逆引きテクニック 500」とか「5日でわかる ClosureLibrary」といった類いの本がたくさん出版されていないと、とりあえず使ってみる際の敷居が高い。

7. ClosureLibrary のコンパイル

ClosureLibrary の特徴として、コンパイラが付属しており、その使用が推奨されていることが挙げられる。インタプリタ言語である Javascript をコンパイルするというと奇妙な感じを受けるが、最近の altJS(Javascript の代替言語)のトレンドとして、マイクロソフトの TypeScript のような、コンパイルすると Javascript を出力するものが登場している。Javascript がサポートしていないクラス、インターフェース、継承などの言語機能について、わかりやすい構文をサポートした中間言語で記述し、実際に動かすのはコンパイルした Javascript を使用するというものである。この方式をとることのメリットとしては、可読性の向上、ブラウザ毎の差異の吸収、出力する Javascript の最適化による

高速化・ファイルサイズ減少が挙げられる。ClosureLibrary は入力ファイルが Javascript、出力ファイルが Javascript という形であるが、コンパイルすることで、最適化された Javascript を得られるようになっている。

ClosureLibrary のコンパイルには3つのレベルがあり、最適化の程度が異なる。まず1つ目に、スクリプトをそのまま出力するレベルがある。これは、必要なライブラリをリンクしただけのモードで、最適化は行われない。書いたそのままの Javascript で出力されるので、デバッグに向いている。2つ目に、シンプル最適化のレベルがある。文意を変えない範囲で、変数名・関数名の短縮を行って、出力されるファイルサイズを小さくするものである。文意が変わらないのが特徴で、機能を損なわずにファイルサイズを小さくできる。実行環境に配備するのは、このレベルでの出力を使うことになると思われる。3つ目が、アドバンス最適化のレベルである。コンパイラが大胆に解釈して不要な処理を切り捨てて最適化を行うモードである。ファイルサイズが劇的に小さくなり、実行速度も最速になると説明されている。このモードの利用にはスクリプトを記述する上での様々な前提条件があり、それに沿っていないとうまくいかない。今回の GUI の開発では、アドバンス最適化で出力したものは挙動が変わってしまい、うまく動作しない結果となったので不採用とした。各レベルで出力されるファイルサイズは大きく異なる。スクリプトをそのまま出力するレベルでは 1.8MB、シンプル最適化のレベルでは 480KB、アドバンス最適化のレベルでは 73KB となった。アドバンス最適化のレベルだと圧倒的に小さくできるのが魅力ではあるが、前提条件をクリアするのが難しいという印象である。

8. おわりに

現時点では、ClosureLibrary を適用できる分野が限られている。例えばグリッド部品のような Web アプリ用の GUI コンポーネントそのものを開発するのに向いている。反面、サードレット、HTML、CSS で画面を作り、Javascript でインタラクティブさを与えるような開発には jQuery 等の、より軽量なフレームワークが向いている。Javascript 自体で GUI を構成する Google 製アプリのようなものを作成する機会があれば ClosureLibrary の採用が考えられる。しかし、その際は外部から ClosureLibrary のエキスパート人材を連れてくる必要があると思われる。

<参考文献>

- 1) 「ClosureLibrary プログラミングガイド」(伊藤千光著、2010/12/11 発行、株式会社インプレスジャパン)

The screenshot shows a web application for '画面メンテナンス' (Screen Maintenance). On the left is a sidebar with a '修理依頼' (Repair Request) dropdown and a list of UI components: ラベル (Label), テキスト (Text), テキストエリア (Text Area), ドロップダウン (Dropdown), ラジオボタン (Radio Button), チェック (Check), 日付 (Date), 野線 (Wire), 検索ボタン (Search Button), 検索アイテム (Search Item), 添付ファイル (Attachment File), and グループ (Group). The main area is titled '▲修理依頼内容' (Repair Request Content) and contains a '修理レベル' (Repair Level) dropdown, two '修理依頼内容' (Repair Request Content) input fields, a '備考' (Remarks) text area, and two '修理希望日' (Desired Repair Date) and '修理希望時刻' (Desired Repair Time) fields. Below these is a '補足項目' (Supplementary Item) section with a yellow background, containing a 'abc テキスト' (abc Text) input field and two 'テキスト' (Text) buttons. A '登録' (Register) button is at the bottom left.

図 1 開発した GUI の画面イメージ