Chrome Debugger APIを用いたLive Edit開発環境の実装技法

Implementation Technique of Live Edit Development Environment Using the Chrome Debugger API

吉永 卓矢　　脇田 建*

Summary. 本稿では，Chrome Debugger APIを用いたJavaScriptアプリケーションのためのLive Edit開発環境の実装技法を提案する．Live Edit開発環境とは，編集モードと実行モードの区別がない開発環境である．JavaScriptはその柔軟さ故に，静的解析では発見できないエラーを潜ませてしまうので，頻繁に実行するという開発スタイルが求められる．Live Edit開発環境はこのような開発スタイルを助ける．Live Edit開発環境ではアプリケーションの状態を素早く正確に表示することが重要であり，本稿ではその実装を容易にするためにChrome Debugger APIを利用することと，APIを利用することによるパフォーマンスの低下を防ぐ技法について提案する．

1　はじめに

JavaScriptでのアプリケーション開発は，言語の柔軟さ故に頻繁に実行することが求められる．EclipseのJDT(Java Development Tools)を用いてJavaアプリケーションを開発する場合，コーディング中にシンタックスエラーやタイプエラーが通知される．しかし，JavaScriptは動的型付けの言語であるため，コーディング中の静的解析によってタイプエラーを発見することは難しい．このことから頻繁に実行してエラーを取り除くような開発スタイルが求められる．

さらにJavaScriptはプロトタイプベースオブジェクト指向言語であるため，クラスではなくオブジェクトに着目する必要がある．既存のオブジェクトをコピーして新しいオブジェクトを作り，オブジェクト毎に自由にプロパティを追加・削除できるため，クラスのような雛形に着目してもオブジェクトの性質をつかむことはできない．

2　Live Edit開発環境

JavaScriptでのアプリケーション開発には頻繁に実行することが求められるが，プログラムを編集，実行，編集，... というサイクルを頻繁に繰り返すのは，それぞれのモードに集中できないので非効率である．そこで *Live Edit* **開発環境**を導入する．

Live Edit開発環境とは，編集モードと実行モードの区別がない開発環境である．アプリケーションは常に実行状態であり，その状態でプログラムを編集することにより開発を行う．編集はただちにアプリケーションに反映される．既存のLive Edit開発環境としては，Squeak[1]やLively Kernel[2]がある．

Lively KernelはJavaScriptのためのLive Edit開発環境であり，Morphic[3]ユーザーインターフェースフレームワークを採用している．Morphic環境はGUI構築に特化したLive Edit開発環境である．そこでは画面上にある全てのGUI部品はmorphかmorphを組み合わせたものである．morphはドローソフトの丸や四角などの基本図形の外観をもったオブジェクトであり，マウスで直接外観を編集できる．さらにインスペクタという専用のブラウザを通してmorphの保持しているプロパティを閲覧・編集することができる．

Lively Kernelには情報の表示が *live* ではないという問題がある．liveであるとは，アプリケーションの状態が変更されると，表示されている情報もそれに合わせて即座に更新されることを指す．Lively Kernelではインスペクタで表示中のプロパティを変更するようなプログラムが実行されても表示は更新されない．ユーザはインスペクタを次に開くときに初めて値の変化に気づくことになる．

そこで本稿では，アプリケーションの状態をliveに表示できるLive Edit開発環境とその実装技法を提案する．

3　提案するシステム

図1が本稿で提案するLive Edit開発環境のスナップショットである．図のように，本システムはGoogle Chrome上で動作し，*Program View* と *Application View* の2つのページから構成されている．

Program Viewはユーザーがプログラムを閲覧・編集するページである．プログラムをオブジェクトの集まりと捉え，オブジェクトを編集することで開発を行う．オブジェクトの編集は *Vobject* という1つのオブジェクトやプロパティや変数を表示・編集す

* Takuya Yoshinaga and Ken Wakita, 東京工業大学大学院 情報理工学研究科 数理・計算科学専攻

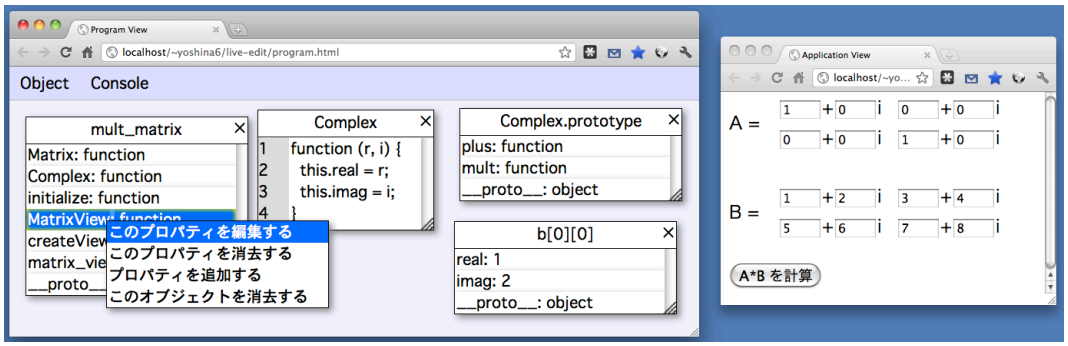


図 1. Program View（左）と Application View（右）

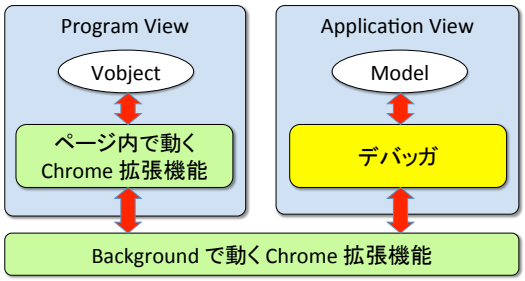


図 2. 本システムのアーキテクチャ

るための View を通して行う．Vobject は型によって表記が異なり，オブジェクトと配列に対しては UML のオブジェクト図に似た表記で，関数に対してはテキストエディタとなる．関数内で定義されているローカル変数をマウスで選択すると，Vobject 化することができる．これにより，値の変化をリアルタイムで観察することができ，デバッグが容易になる．

Application View は開発中のアプリケーションが動いているページである．Program View で編集された内容は即座に Application View に反映される．また Application View でアプリケーションを操作すると，それによって変化した値は即座に Program View に反映される．

4 実装技法

本システムの実装技法について説明する．

4.1 Chrome Debugger API

本システムの実装は *Chrome Debugger API*[4] を利用している．これは Chrome に標準で付属しているデバッガを拡張機能から操作できるようにした API である．

図 2 が本システムのアーキテクチャである．Application View で開発中のアプリケーション（Model）が動いていて，Model を元に Program View の Vobject の表示を更新する．

Model から値を得たり，Model に新しいプログラムを追加するためにデバッガを用いる．デバッガを操作するには，Program View で動いている拡張機能から API を利用してプロトコル [5] に従ったリクエストを送る．リクエストを送る際，Program View の拡張機能とデバッガは直接通信ができないので，Background で動いている拡張機能を経由する．デバッガはそのリクエストを処理して Model に適用し，得られた結果を拡張機能のプログラムに送り返す．

利用できる主なデバッガの機能は，ブレークポイントの設定，プログラム停止中のローカル変数の取得，コードを評価してアプリケーションに適用，オブジェクトのプロパティの取得などがある．これらを利用して，Model から値の取得，ブレークポイントを用いてのローカル変数の Vobject 化，Model に新しいコードの適用，などが実現できる．

4.2 パフォーマンス

Vobject の更新にはデバッガとの通信が伴う．何らかの状態変化が起きたとき，画面に表示されている Vobject の全てが更新候補となる．n 個の Vobject が表示されている場合，値取得のためにデバッガと n 回通信することになる．このとき通信のオーバーヘッドが大きいと live ではなくなってしまう．

実験をしたところ 1 秒間に 164 個の Vobject を更新することができた．通常だと画面上に一度に表示できる Vobject は 10 個から 20 個程度であるため，人間が知覚可能と言われる 0.1 秒の間に全てを更新することはできない．更新速度を改善する必要がある．

本稿では，通信回数を少なくする技法を提案する．n 個の Vobject を更新したい場合，n 個の情報を 1 回の通信で全て送受信するようにする．詳しく説明すると，まず Program View が現在表示されている n 個の Vobject のリストをデバッガに送る．デバッガは 1 つの空のオブジェクトを用意し，リストにあるオブジェクトの値を Model から取得して，その値を用意したオブジェクトにプロパティとして追加する．n 個の値を追加し終えたらそのオブジェクトを

Program View に返し，Program View はそのオブジェクトを n 個のオブジェクトに分解して値を取得し Vobject を更新する．こうすることでデータは大きくなるが通信回数を 1 往復にすることができる．

この技法を用いて実験をしたところ，1 秒間に 2352 個更新することができた．この速度だと 20 個の Vobject は 0.1 秒の間に余裕で更新できる．

5 おわりに

本稿では JavaScript アプリケーションの開発には Live Edit 開発環境が必要であることを示し，Chrome Debugger API を用いた Live Edit 開発環境の実装技法について提案した．Chrome Debugger API を用いることである時点でのローカル変数の情報を取得したり，アプリケーションとプログラムコードの整合性を保ったまま両方を更新することが容易となった．またデバッガとの通信回数を少なくすることによるパフォーマンスの低下を抑える技法を提案した．

謝辞

研究を進めるにあたり，研究室の皆様にアドバイスを頂いたので感謝いたします．本研究は文部科学省科学研究費補助金基盤研究 (C) 課題番号 23500034 の援助を受けています．

参考文献

[1] Daniel Ingalls, Ted Kaehler, et al. Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself. http://ftp.squeak.org/docs/OOPSLA.Squeak.html.

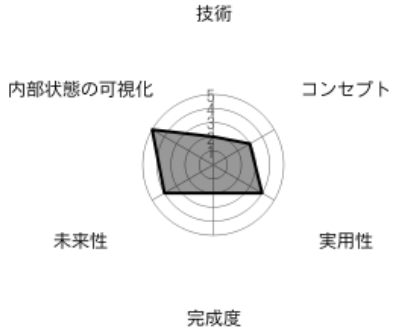
[2] Daniel Ingalls, Krzysztof Palacz, et al. The Lively Kernel A Self-supporting System on a Web Page. In *Self-Sustaining Systems: First Workshop, S3 2008 Potsdam, Germany, May 15-16, 2008 Revised Selected Papers*, pp. 31–50. Springer-Verlag, 2008.

[3] Maloney John, Smith Randall. S. R. Maloney John. Directness and liveness in the morphic user interface construction environment. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pp. 21–28. ACM, 1995.

[4] chrome.experimental.debugger. http://code.google.com/chrome/extensions/dev/experimental.debugger.html.

[5] Chrome Developer Tools: Remote Debugging. http://code.google.com/intl/ja/chrome/devtools/docs/remote-debugging.html.

アピールチャート



未来ビジョン

本研究は，プログラミングスタイルの未来を考えた研究である．未来では，知りたい情報をすぐにわかりやすい形で表示でき，編集したい内容を編集しやすい方法で編集できるというプログラミングスタイルでなければならない．現時点ではオブジェクトのプロパティや変数の表示などの細かい部分に着目しているが，未来ではプログラムの構造の可視化・編集や GUI の直接操作についても取り組みたい．

プログラムは大規模になればなるほど構造化が重要になってくる．現在普及している開発環境の多くはテキストエディタにコードを書くことをユーザに強いる．しかし，言語の機能を使ってプログラムを構造化できても，テキストエディタではその構造を視覚的に表示することができない．本稿では UML のオブジェクト図に似た表記を採用したが，それだけでは不十分であり，オブジェクト同士の関係をわかりやすく視覚化・編集できなければいけない．

また，本システムは実装上の都合により Program View と Application View に分かれていて，ユーザーが編集できるのは Program View のみであった．編集内容はすぐに Application View に反映されるものの，GUI の編集も JavaScript コードを書いて行わなければならなかった．未来では GUI はマウスで直接編集できなければいけない．

このような問題をクリアしていくことで，情報をベストな方法ですぐに表示・編集できる開発環境に近づいていくだろう．