6K-04

オフライン Web アプリケーションにおける 事前データ取得の半自動化

磯谷俊明† 小宮常康†

電気通信大学大学院情報システム学研究科†

1. はじめに

オフライン Web アプリケーションにおいては Web サーバから Web ブラウザ側に実行に必要なデータを一括で事前取得(以下プリフェッチ)する事が基本である。従ってプログラム中のプリフェッチコードは 1 箇所のみ記述すればよく、通信処理に伴うコールバック関数の記述コストは少ない。

しかしデータの一括取得が困難な場合がある。この問題は従来では一括取得後は常にオフライン状態であったオフライン Web アプリケーションにおいて、必要に応じて一時的にオンライン状態になる事を許す事で解決できる。ただしオンラインになるタイミングは選択できないものとする。これは電波が不安定になる場所での使用を想定している為である。このように一時的にオンライン状態になることを許可するアプリケーションを準オフライン型、また一括取得後のデータ通信を行わないアプリケーションを完全オフライン型と便宜上呼ぶことにする。

準オフライン型においてはオンラインになるタイミングを選べないため必要なデータを予測し、オンライン時にプリフェッチをする事が考えられる。しかしこの方法を採用すると実行に必要なデータをプリフェッチするタイミングを考慮しつつプログラミングを行うことが求められるため、データのプリフェッチを行うコードを何度も修正する事が予想される。例えばオフライン時にプリフェッチを行っても失敗する事などが考えられる。さらにプリフェッチコードに伴うコールバック関数の書き換えも行う必要があり非常に煩わしい。

そこで本研究では準オフライン型のオフライン Web アプリケーションにおいて、プリフェッチコードをプログラムのロジックとは別に用意した情報から自動生成し、プログラム中に自動挿入する手法を提案する。またコールバック関数などの煩雑な処理を記述する必要のない手法についても議論する。本研究の成果によりオフライン Web アプリケーションにおけるプログラムの複雑さや修正コストが軽減する事が期待できる。

2. 本研究におけるプログラムの記述指針

本研究では準オフライン型のオフライン Web アプリケーションにおけるプログラムの複雑さや修正コストを軽減する事を目的としている。その為に get 関数、コールバック関数を伴わない記述手法、プリフェッチコードの自動挿入手法の導入を行った。オフライン Web アプリケーションの記述には JavaScript を用いることが基本であるが本研究では便宜上 Scheme で記述する事にする。

A semi-automated technique for data prefetching in offline web applications

Toshiaki ISOYA†, Tsuneyasu KOMIYA†

†Graduate school of Informations Systems – The University of Electro-Communications

2.1 get の導入

本研究においてはサーバとのデータ通信及び Web ブラウザ上のデータ(local Storage を想定)へのアクセスを統一化するために get という組み込み関数を導入した。get はオフライン時及び必要なデータが local Storage に存在する時、local Storage へのアクセスとして機能する。また必要なデータが local Storage に存在せずかつオンライン時の場合はサーバへのアクセスを行い必要なデータを取得する。つまり get におけるオンライン時のアクセスは local Storage に必要なデータが存在しない場合に備えた保険のようなものである。

get の導入により local Storage へのアクセスと同等の記述で暗にサーバへのアクセスを行う事が出来、プログラムの煩雑さを軽減することもできる。しかし get の導入によってコールバック関数が増加してしまう問題がある。これはオフライン Web アプリケーションにおいて local Storage アクセスは非常によく使われる処理であることが理由である。get は local Storage アクセスとサーバアクセスの両方の機能を有しているので、local Storage へのアクセス時にもコールバック関数を記述する必要がある。この問題の解決策は後述する。

2.2 コールバック関数を伴わない記述方法の導入

ソースコード 1 のようなプログラムを考える。これは準オフライン型の記述方法で書いた Web サーバから取得したデータを用いた加算処理である。

```
1(get (lambda (x);;k1
2      (get (lambda (y);k2
3             (print (+ x y)))
4           url2 key2))
5    url1 key1)
```

ソースコード 1:準オフライン型を従来手法で記述した時の加算処理

ソースコード 1 において get は local Storage へのアクセス機能及びサーバへのアクセス機能を持つ関数であり、引数としてサーバへアクセスするための URL、local Storage へアクセスするためのキー、コールバック関数 k1、k2 を与える。例えばソースコード 1 の 1 行目の直前にプリフェッチを挿入すると、2 行目以降のコードをコールバック関数として書き直さなければならない。またプリフェッチコードを 2 行目の直前に入れると 3 行目以降をコールバック関数として書き表す必要がある。このようにプリフェッチコードを挿入する場所を変える毎に新たにコールバック関数を記述し直すのは非常に手間である。

この問題を解決するために本研究においてはコールバック関数を伴わない記述手法を導入することにより解決

している。本研究における記述手法を用いるとソースコード2のように記述する事ができる。

```
1(+ (get url1) (get url2))
```

ソースコード2:本研究におけるコールバック関数を必要としない加算プログラム

ソースコード2ではソースコード1と比べると複雑さが改善されており、プリフェッチコードを挿入してもコールバック関数を記述する必要がないのでプログラムの修正が非常にやりやすくなっている。また先述の get 関数の問題もコールバック関数を伴わない記述方式を導入することで解決している。

2.3 プリフェッチコードの自動挿入

準オフライン型においてはオンラインになるタイミングが選べないのでプリフェッチコードを適切な場所に入れることは難しく試行錯誤が必要になると考える。本研究において、プログラムロジックとは別に書かれた情報を基にプリフェッチコードを自動挿入する手法を提案する。プログラムのロジックと分離することでプログラム修正がやりやすくなる。

プリフェッチコードの自動挿入に必要となる情報はソースコード3のように記述する。

```
1(prefetch-when <function-name>)
```

ソースコード3:プリフェッチコード挿入器が必要とする情報の指示

prefetch-when はキーワード、<function-name>は関数名を表す。コード挿入器は<function-name>で指定された関数名の関数呼び出し式の直後にプリフェッチコードを挿入する。

3. アプリケーション実行までの流れ

本研究ではアプリケーションが実行されるまで 4 つの段階がある。第 1 にプログラマがコールバック関数を伴わないプログラムを記述する。例えばソースコード 2 のようなプログラムである。第 2 にプログラマが記述したソースコードに対してプリフェッチコードの自動挿入を行う。第 3 に CPS(Continuation Passing Style)変換を行い後続の処理(コールバック関数相当)を関数の引数として取得できる形に変換を行う。これによりコールバック関数相当の処理を自動的に関数として引数に渡してくれる。最後に第 3 までの処理で得られた Scheme プログラムを JavaScript で記述された Scheme インタプリタを用いて Web ブラウザ上で実行する。

4. 利用例

本研究の有用性を検証するために実際に準オフライン型のアプリケーションを作成した。今回作成したアプリケーションは地図アプリケーションである。Google Maps などの地図アプリケーションでは地図画像をタイルのように複数敷き詰めて 1 つの地図のように見せている。この地図はユーザの可視領域よりも大きく設定されており、この地図を動かすことで地図の見える範囲を変える事ができる(図 1)。本研究でも同様の手法を採用する。

今回作成した準オフライン型の地図アプリケーションは必要なデータの取得をオンライン時の画面更新後に行っている。画面の更新によって local Storage 内のデータが消費したと考えられるので、消費した分を補うためである。また必要なデータの予測を行うことでオフライン状態がある程度長く続いたとしてもアプリケーションの実行を行えるように作成されている。

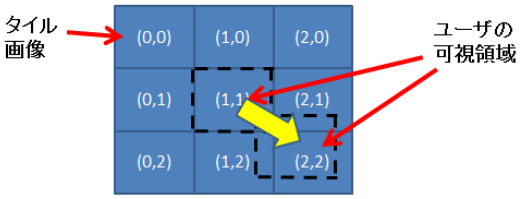

図 1:地図アプリケーションのイメージ

今回作成した地図アプリケーション記述では以下の点で従来の Web アプリケーションの記述よりも優れている。

まずコールバック関数を記述する必要性がなくなったことでプログラムの複雑さが軽減した。従来の JavaScript プログラムではコールバック関数によりプログラムの処理の流れが乱されるという問題があった。これにより非常にプログラムが複雑なものになり修正がやりにくいという問題があったが本研究においてはその点が改善されている。

また通信の透過性が本研究の手法では改善されている。従来ではサーバとの通信処理と local Storage へのアクセス処理を併記する必要があり、どちらへアクセスしているのかという事を考えながらプログラミングを行う必要があったが本研究の手法ではその必要がない。

またプリフェッチコードを自動挿入するようにした事でその部分の修正がやりやすくなった点が挙げられる。

5. 関連研究

Yung ら[1]はオフライン Web アプリケーションの実行を可能にするランタイム環境 WOPRE を提案している。これは開発者が開発した Web アプリケーションをサーバ上にアップロードするだけでオフライン対応化する事ができる。またユーザはサーバからオフライン対応化された Web アプリケーションを自身の端末にダウンロードする事ができるという特徴を持つ。これは Apple の App Store に酷似している。しかしプログラムの記述性に関しては言及されていない。

6. おわりに

本研究ではオフライン Web アプリケーションを開発するプログラマのためのコールバックを記述する必要のない記述機能やプリフェッチコードの自動挿入機能の提案を行った。本研究の成果によりオフライン Web アプリケーション開発が容易になれば幸いである。

参考文献

[1] Yung-Wei Kao, ChiaFeng Lin, Kuei-An Yang, Shyan-Mimg Yuan, A web-based, offline-able, and Personalized runtime environment for executing application on mobile devices, ELSEVIER, Computer & Interfaces 34, pp. 212-224(2012)