Web サーバ・ブラウザ間における XML ストリーム通信の実装

An Implementation for a XML Stream Interchange between a Web Server and Web Browsers

横山 昌平♥
Shohei YOKOYAMA

本研究の目的は Web ブラウザ上で JavaScript を用いて XML データを処理するための軽量システムの実現である．JavaScript によって XML データを扱う手法としては JSON が利用できる．しかしながら JSON は文書全体を一旦メモリ上に展開する必要があり，Web ブラウザという限られた資源の中で大規模なデータを扱うという目的には適していない．一般的に，大規模なデータを扱うシステムでは，ハードディスク等の二次記憶媒体に保存したデータに対して処理を行う．しかしながら Web ブラウザで動作する JavaScript プログラムからはセキュリティー上の制約により，二次記憶にアクセスすることができない．そこで本論文では任意のサーバ上の XML 文書を直接 JavaScript プログラムから呼び出すための三つのストリーム通信手法を提案する．そして実験により最適な実装方法を検証する．

The purpose of this research is to develop a lightweight system for XML data processing on Web browsers using JavaScript. There is the JSON which has been available for handling XML data with JavaScript. However, the problem is loading the whole data set into memory; therefore, JSON is not suitable for handling a huge amount of data on Web browsers. Generally speaking, system loads huge amount of data which has been stored into a secondary storage such as a hard disk. However, JavaScript programs on Web browser cannot access to secondary storages because of security reasons. In this paper, I propose new three methods for XML stream interchange. Using this method, JavaScript program can access directly to XML document on servers, and present an efficient implementation.

1. はじめに

近年Webブラウザは，以前のようにWebサーバが出力したデータを単に表示するだけでなく，高度なユーザインタラクションを有するアプリケーションのプラットフォームとして注目されてきている．そのようなアプリケーションにおいてAjax (Asynchronous JavaScript and XML)[1]は非常に注目されている．AjaxはWebブラウザ上で非同期通信を行うプログラミングモデルの総称であり，JavaScriptやHTML，HTTP等の技術に基づいている．Ajaxを利用して実装すれば，Webページはページの遷移無しにWebサーバと通信を行う事が可能となる．この仕組みはここ数年で急速に注目を浴び，スクロール可能な地図ソフトやワードプロセッサソフトなど，従来スタンドアロン環境で提供されていたリッチアプリケーションが，次々とWebブラウザを通して提供され始めている．このようにAjaxによって引き起こされた新しいWebの環境はWeb2.0[2]と呼ばれている．

しかしながらAjaxはHTMLファイルを取得したサーバ以外とは通信することが出来ないため，最近ではJSON形式[3]のデータファイルをダイナミックにロードする手法が注目されている．JSONとはJavaScriptの文法に従った文書フォーマットであり，半構造データのような柔軟な構造を持つ事ができる．JSONで記述されたファイルをHTMLページ内の<script>要素等にダイナミックロードする事により，Webブラウザ・サーバ間でページ遷移の伴わないデータ通信を行う事ができる．

JSONを利用した通信では，一旦メモリ内にデータを展開し，そのオブジェクトにプログラムからアクセスするという点で，XML処理におけるDOMのような柔軟なアクセスを提供する．しかしながら，この手法では搭載しているメモリ量によって，利用できるデータ量が制限されてしまう．

XML処理においては搭載メモリ量に左右されない手法としてSAXが利用されている．SAXはXML文書の先頭から走査し，要素開始，文字列，要素終了等の構成要素を発見する度にイベントとしてユーザに通知する，ストリームアクセス型の処理手法である．SAXはデータの読み込みと同時に処理を開始でき，メモリ負担も低い等の特徴を持っている．

多種多様なクライアント環境が存在するWebにおいて，SAXのようなストリームアクセスを実現する事は非常に重要な課題であると考える．

そこで，我々はWebブラウザ上で利用できるSAXシステムFreddyを開発している．本論文では特にその性能面に着目し，複数の異なる手法に基づいた実装系を提案する．そして，それらの比較実験を行う事により，Webサーバ・ブラウザ間の最適な通信手法を議論する．

続く２章ではFreddyの概要とデータフォーマットを説明する．３章では本論文で比較を行う３つのWebサーバ・ブラウザ間通信手法について詳述し，４章で比較実験にて性能を評価する．最後に，５章で本論文をまとめる．

2. 提案手法：Freddy

2.1 概要

Webブラウザ上でSAXを実装するという課題は，単にJavaScriptでSAXを実装するという事にはならない．JavaScriptは強力なスクリプト言語ではあるが，Webブラウザによるセキュリティー上の制約が非常に多い．例えば，Webブラウザは，HTTPプロトコルでアクセスでき，URLで指定可能なリソース全てにアクセスする事ができる一方で，Webブラウザが動作するコンピュータの二次記憶にアクセスする事はできない．すなわちJavaScriptによるWebアプリケーションは，スタンドアロンアプリケーションとは異なり，二次記憶に保存するデータはWebサーバ上に保存する必要がある．このモデルはSaaS (Software as a Service) と呼ばれ，アプリケーションの新しいパラダイムを形成している．

提案手法はこの新しいパラダイムに適応したSAXの実現を目標にしている．つまり本研究での課題は，ネットワーク的に離れた場所にあるXML文書をWebブラウザ上のJavaScript

♥ 会員　独立行政法人 産業技術総合研究所 グリッド研究センター dbsj_v6n1@yokoyama.ac

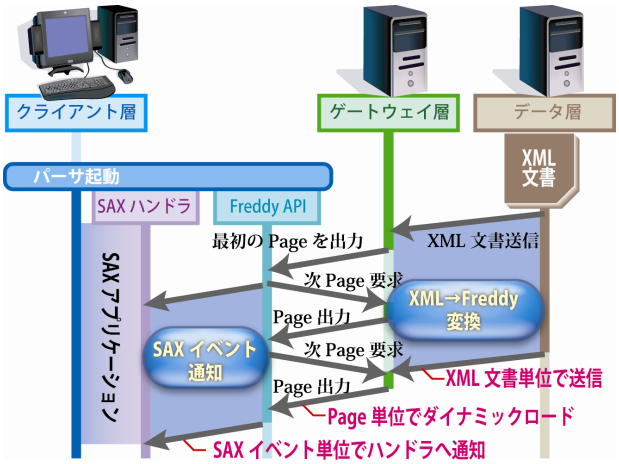

図1 XMLストリーム通信の概要
Fig.1 A overview of XML stream interchange

プログラムから取得するというシナリオにおいて，SAXのような読み込みと同時に処理を開始でき，かつ消費メモリが少ないシステムを構築する事である．

提案手法を用いてWebブラウザがXML文書を読み込むまでの概要を図１に示す．提案手法の特徴はXMLファイルを独自のフォーマットにゲートウェイを介して変換する．そしてデータをPageという単位に断片化し，Page毎に複数回に分けてWebブラウザに送っている．この仕組みにより，Webブラウザは XML文書全体の到着を待たずに，処理を開始する事ができる．続く2.2節で通信の為のデータフォーマットであるFreddyフォーマットを説明し，2.3節では，利用者へ提供するSAX APIについて述べる．

2.2 Freddy フォーマット

FreddyフォーマットとはSAXのイベント列を基にし，JavaScriptの文法を持つ，XML形式と互換性のあるデータフォーマットであり，提案手法においてWebサーバ・ブラウザ間での通信に利用される．

XML文書は半構造のデータ形式だが，SAXのイベントの単なる列として考える事もできる．このイベントの列に基づいてデータを再構成した形式が提案するFreddyフォーマットである．以下の簡単なXML文書を例に説明する．

```
01: <list>
02:     <cocktail name="Martini">
03:         $10-
04:     </cocktail>
05: </list>
```

この文書から得られるSAXのイベント列を表１に示す．

表1 SAXイベント列の例
Table1 An example of SAX event sequence

順序	イベント	プロパティ
1	要素開始	要素名:list
2	要素開始	要素名:cocktail 属性:name=Martini
3	文字列	$10-
4	要素終了	要素名:cocktail
5	要素終了	要素名:list

Freddyファイルフォーマットはこの SAXイベント列の各イベントを対応するJavaScriptのコードに書き換えたものである．

```
01: h.s('list');
02: h.s('cocktail',{"name":"Martini"});
03: h.c('$10-');
04: h.e('cocktail');
05: h.e('list')
```

先頭の変数hはイベントハンドラオブジェクトを示している．つまり，それぞれの行は直接イベントハンドラ上のメソッドを呼び出すJavaScriptコードになっている．SAXイベントを表す各行をイベントコンテナと呼ぶ．

呼び出し側では，SAXのイベントハンドラを予め定義しておけば，Freddyフォーマットのファイルをダイナミックロードする事でハンドラの対応したメソッドが呼び出される．

FreddyフォーマットはXMLと比べ，要素名や文字列などを引用符で括る必要がある等，冗長性が高い．本論文では詳述しないが，提案手法では繰り返し出現する要素名を符号化する事によって，サイズの増加を防いでいる．符号化には先行研究である要素名圧縮手法[4]を用いている．

Freddyフォーマットは各行が一つのSAXイベントを表す非常に単純なフォーマットであり，分割が容易にできる．分割する単位はクライアント側で設定できるが，デフォルトでは5000イベント毎に分割している．この分割された断片をPageと呼ぶ．クライアント側ではこのPage単位でダイナミックロードする．ダイナミックロードされたPageはHTMLファイルのDOMツリーに挿入される．Pageの先頭には次のページを呼び出すための命令が書かれており，あるPageが実行されると同時に次のPageの読み込みを開始する．また実行した後のPageは適時DOMツリーから削除することにより，読み終わったデータをメモリから開放し，消費メモリを抑えている．

2.3 Freddy SAX API

これまでに述べたXML文書のFreddyフォーマットへの変換とPage単位への断片化，そしてPageの読み込み，符号化された要素名の復号は，全てFreddyが提供するSAX APIの中に隠蔽されている．Freddyの利用は一般的なSAX APIと同様である．まずパーサのインスタンスを作成し，イベントハンドラを設定する．そして処理するXML文書のURLを指定することでXML文書の読み込みが開始される．

```
01: p = new freddy.SAXParser();
02: h = new MySaxHandler();
03: p.setSimpleEventHandler(h);
04: p.parse("http://yokoyama.ac/data.xml");
```

また，Freddyフォーマット内の各イベントコンテナは，JavaにおけるSAXと同じ可読性の高いメソッドへとリダイレクトされる．イベントコンテナと対応するSAXイベントハンドラのメソッドの関係を表２に記す．

表2 イベントコンテナと対応するハンドラ関数
Table2 SAX events and event containers

SAXイベント	イベントコンテナ	メソッド(ハンドラ)
要素開始	h.s(…);	h.startElement(…);
文字列	h.c(…);	h.Charactors(…);
要素終了	h.e(…);	h.endElement(…);

3. Web サーバ・ブラウザ間通信の実装

本章では，ダイナミックロードを実現する実装系を複数提

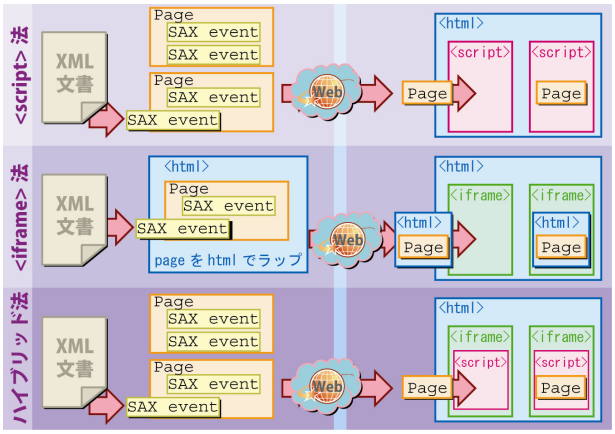

図2 XMLストリーム通信の為の３手法
Fig.2 Three methods for XML stream interchange

案する．一般的にJavaScriptファイルをダイナミックロードする際，HTMLページを構成するDOMツリー内の任意の箇所に<script>タグを挿入する事によってバインドする．Freddyの実装も同様にPageを<script>タグにバインドした所，読み終わったPageをDOMツリーから削除しても，Webブラウザは実際のメモリ上からは削除しない事が分かった[5]．そこで，本論文では<script>にバインドする手法(図２上)以外に，<iframe>タグにバインドする手法，<iframe>と<script>を組み合わせた手法を提案する．また，この３手法の性能の評価は４章にて行う．

3.1 <iframe>法

<iframe>タグは表示 Web ページ中の特定領域に別の Web ページを表示する為のタグで，中に HTML にラップした Freddy フォーマットの Page をダイナミックロードする事で，<script>タグにバインドした場合と同様に，SAX イベントをハンドラへ通知する事ができる(図２中)．

3.2 ハイブリッド法

<iframe>タグでダイナミックロードを行うと，内部的なページ遷移が起こる．ページ遷移には処理コストがかかるので，<script>タグと組み合わせる事によって，ページ遷移を抑えつつ，適時<iframe>領域をクリアして消費メモリを開放させる事ができる．

具体的には，あらかじめ二つの<iframe>領域を確保しておき，空の HTLM ファイルを表示しておく，そして，どちらか一方の領域に<script>タグを追加する事で Page を読み込む．任意の Page 数を読み込む毎に，もう片方の<iframe>領域へ処理を移し，元の領域をリロードする．元の領域には空の HTML ファイルがロードされる(図３下)．

このハイブリッド法は<iframe>法より処理コストが低く，かつ<script>法とは異なり消費メモリ領域を適時開放すると考えられる．

4. 性能評価

前章で述べた３つの実装を比較し，今回新たに提案するハイブリッド法の性能を評価する．本実験では，SAX を用いてネットワーク越しの XML 文書を処理するシナリオにおいて，三つの実装系それぞれの動作速度と消費メモリ量を計測した．

4.1 実験環境

本実験では，図１で示したデータ層・ゲートウェイ層・クライアント層からなるシステムを構築した．システムの簡単化の為，データ層とゲートウェイ層は同じサーバを利用している．サーバの性能を表３に示す．

表3 サーバマシンの性能
Table3 The server machine specification

ハードウェア	DELL PowerEdge 2650
CPU	Intel Xeon 2GHz × 2
メモリ	512MB
OS	Linux (Fedora Core 6)
HTTPD	Apache 1.3.37

次に，Freddy による SAX アプリケーションが動作するクライアントの性能を表４に示す．

表4 クライアントマシンの性能
Table4 The client machine specification

ハードウェア	IBM ThinkPad X41 Tablet
CPU	Intel Pentium M 1.6GHz
メモリ	1GB
OS	Microsoft Windows XP Tablet Edition
Webブラウザ	Mozilla FireFox2 Microsoft Internet Explorer 6

実験はインターネットにおける一般的な利用環境を想定して，サーバマシンとクライアントマシンは地理的にもネットワーク的にも異なる場所に配置している．

次に本実験で利用した実装系を記す．クライアントマシンの Web ブラウザ上で動作する SAX アプリケーションは JavaScript で実装されている．ゲートウェイとの通信手法は３章で述べた３つの手法を実装した．ゲートウェイ側で Web ブラウザとの通信を制御するプログラムは PHP スクリプトで実装されている．ゲートウェイと読み込む XML 文書を有するサーバとの通信は Python で実装されている．

本実験で利用するテストデータは XML 処理の標準的なベンチマークである XMark[6]の xmlgen を用いて 100MB の XML 文書を作成した．

4.2 実験：処理速度

まず，Mozilla FireFox2 にて，以下の Freddy の３つの実装において，それぞれテストデータを読み込み，処理終了までの時間を計った．

1. **<script>法**
   Freddy フォーマットの Page を<script>タグにバインドする手法
2. **<iframe>法**
   Freddy フォーマットの Page を HTML にラップし，<iframe>タグにバインドする手法
3. **ハイブリッド法**
   <iframe>領域内に Freddy フォーマットの Page をバインドした<script>タグを追加し，10Page 毎に<iframe>内のデータをクリアする手法

本実験ではそれぞれの実装に対して 10 回の処理を行った．結果の最大値，最小値，平均値を図３に示す

4.3 実験：消費メモリ

次に消費メモリの動向を調査した．Web ブラウザ上で動作する JavaScript プログラムのメモリ消費を厳密に計測するのは困難なので，各実装系を動作させた時の，マシン全体の

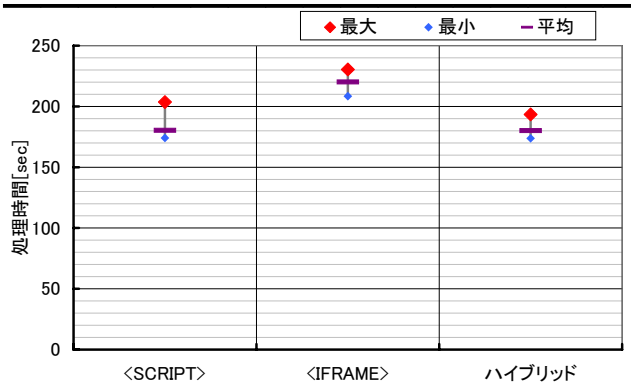

図3 実験結果:処理時間
Fig.3 Experiment result: processing time

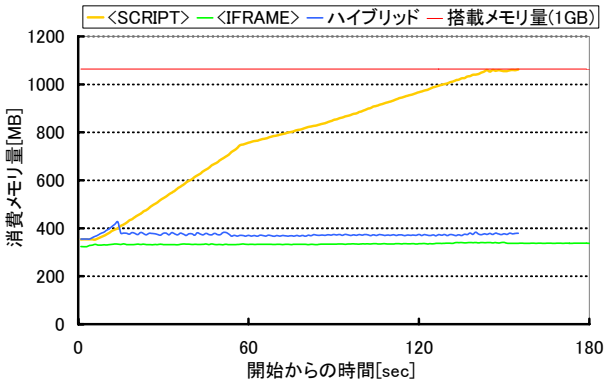

図4 実験結果:メモリ消費量 (Internet Explorer)
Fig.4 Result: memory consumption (Internet Explorer)

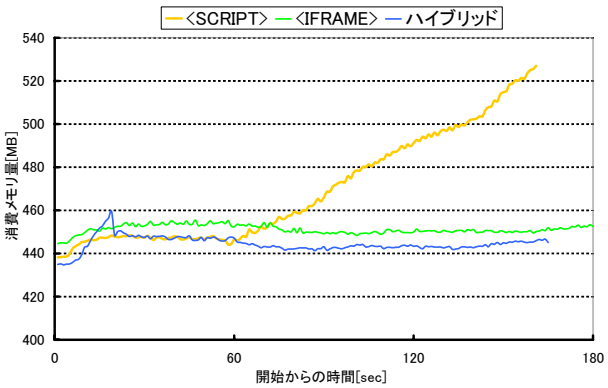

図5 実験結果:メモリ消費量 (FireFox)
Fig.5 Result: memory consumption (FireFox)

メモリ消費量を，外部プログラムを用いて計測した．実験は前述した３つの実装それぞれを二つのWebブラウザを使って実行した．Microsoft Internet Explorer を利用した場合の実験結果を図４に，Mozilla FireFox を利用した場合の実行結果を図５に記す．

4.4 考察

文献[5]において，<script>法の場合，DOM ツリーから削除しても，実際のメモリ領域からデータが削除されない現象を報告した．本実験でも図４及び図５に示されている．これは Web ブラウザの JavaScript エンジンに起因する問題である．特に Internet Explorer を利用した場合(図５)では，読み込んでいるテストデータは 100MB であるのに対し，消費メモリは約 600MB となっており，非常に効率が悪い．

一方で<iframe>法では，実行期間中のメモリ消費量に大きな変化は見られず，読み込み終わった Page が確実に開放されている事が分かる．しかしながら，図３で示されている通り，<script>法に比べ読み込みに時間が掛かってしまっている．

それらに対し，新たに提案したハイブリッド法では，消費メモリを一定に抑えつつ，<script>法と同等の処理速度になる事が分かった．実験結果より３手法の得失をまとめて表５に記す．

表5 ３手法の得失
Table5 Advantages and disadvantages of three methods

	<script>	<iframe>	ハイブリッド
速度	○ 早い	× 遅い	○<script>と同等
メモリ	× 比例増	○ 一定	○<iframe>と同等

この実験結果より，<iframe>タグと<script>タグを併用するハイブリッド法が一番優れている事が分かった．

本実験は100MBという従来Webブラウザでは扱いきれなかった XML 文書対象に行った．そして，ハイブリッド法を利用した Freddy は，動作するコンピュータの搭載メモリ量に関わらず，Web ブラウザを通して大規模なデータを扱う為の実際的な性能を有している事が分かった．

5. まとめと今後の課題

本論文では Web ブラウザ上で動作する SAX パーサ Freddy を提案した．また，その実装に関して，考えられる３つの実装手法を提案した．それらを比較実験する事により，<script>タグと<iframe>タグを併用する手法が一番優れていることが分かった．

今後は XML ストリームに拘らず，Web サーバ・ブラウザ間の汎用的なデータ通信基盤として発展させる計画である．

[文献]

[1] J. J. Garrett: “Ajax: A new approach to web applications”, http://adaptivepath.com/publications/essays/archives/000385.php
[2] T. O’Reilly: “What is web 2.0 design patterns and business models for the next generation of software”, http://www.oreillynet.com/lpt/a/6288
[3] “Introducing json”, http://www.json.org/
[4] 横山，太田，石川: “要素名圧縮によるxmlデータ圧縮手法の提案 - simplified element xml -”，データベースとWeb情報システムに関するIPSJ DBS/ACM SIGMOD Japan Chapter JSPS-RFTF AMCP合同シンポジウム (DBWeb2000), (2000).
[5] 横山, “Webブラウザから利用できるSAXパーサ"Freddy"の実装と評価”, 電子情報通信学会 第18回データ工学ワークショップ (DEWS2007), pp.1 – 8 (2007).
[6] “Xmark – an xml benchmark project”, http://monetdb.cwi.nl/xml/

横山 昌平 Shohei YOKOYAMA
産業技術総合研究所グリッド研究センター特別研究員．2006 年 東京都立大学大学院工学研究科修了．博士 (工学)．XML 技術の研究に従事．情報処理学会，日本データベース学会正会員．情報処理学会論文誌 (データベース) 幹事補佐．