XML 高速処理技術 SPlitDOM の機能拡張と Web アプリケーションへの適用評価

中島 哲‡　小田切 淳一‡　井谷 宣子‡　吉田 茂‡

†株式会社　富士通研究所 〒243-0197 厚木市森の里若宮 10-1

E-mail: †{nakasima, odagiri, ita2, yoshida.shig-04}@jp.fujitsu.com

あらまし　近年、データ処理・表示の様々な用途に XML が使われるようになってきている。XML は柔軟で便利な反面、メモリ消費や CPU 負荷が大きく性能が出しにくいという弱点がある。これを解決するため、我々は XML 処理を高速化する技術「SPlitDOM」を開発している。SPlitDOM の実用化においては、アプリケーションへの導入容易性、及び実システムを想定した性能評価が課題であった。SPlitDOM の機能拡張を行い、Web アプリケーションに適用評価し、導入容易化と高性能を達成したので報告する。

キーワード　XML、Java、Web

1. はじめに

近年、様々なソフトウェア開発において、XML が広く使われるようになってきている。XML は国際標準の柔軟なデータ表現形式であり、今後はさらに幅広い分野で活用されると期待される[1][2][3]。

しかしながら、XML はタグでデータを記述する形式のため、従来一般的な CSV 形式などと比べるとデータサイズが大きくなる。その分、データ処理時間やメモリ消費が大きくなるため、XML を使うシステムでは性能が出しにくい。XML が今後、大規模・高トランザクション・システムにも適用される場合、特に XML の性能面の弱点が大きな問題になると予想される。

我々は、このような弱点を解決する技術として、XML 高速処理技術「SPlitDOM」を開発してきた[4],[5],[6]。SPlitDOM の機能拡張を行い、Web アプリケーションへの適用実験を通して効果を確認した。以降では、これらについて、次の順に述べる。

- 背景(関連技術)
- 従来の SPlitDOM と課題
- SPlitDOM 選択適用方式の導入
- Web アプリケーションでの適用評価

2. 背景

XML のデータ処理には W3C 標準の API である DOM(Document Object Model)[7]が広く使われている。DOM は、XML データをメモリ上にツリー構造のオブジェクトとして展開する方式であり、アプリケーションからは参照・更新処理が簡単にできるという利点がある。その反面、データ全体をメモリ展開するため、メモリ消費が大きく性能が出にくいという弱点がある。

小メモリ消費で高速な標準 API としては SAX(Simple API for XML)[8]が使われる。しかし、SAX は XML データを先頭から順に読み込みながらアプリケーションに解析結果をイベントとして通知する。このため、単純なシーケンシャルな参照処理にしか適さず、更新処理や複雑な参照処理を行うには、煩雑なコードを書かなければならない。

以上の点から、一般にアプリケーションの開発においては DOM が望まれるが、性能面の弱点から、性能要件が厳しいアプリケーションでは利用できないことが多い。その場合、SAX を使わざるを得ず、開発者の負担が大きい。

このような問題を解決する技術として、従来、次のような XML 処理技術が提案されている。

- PDOM(Persistent DOM)[9]

  XML の解析情報をディスクに格納し、アクセスする部分をメモリ展開するキャッシュをもつことで、大容量の XML データを小メモリで処理できる技術である。XML をデータベースのように使用することを想定している。ディスクベースのためランダムアクセスする用途は厳しいと考える。

- DDOM(Dictionary based DOM)[10]

  XML の解析情報をメモリ上に圧縮して保持し、DOM としてアクセスできる技術である。アクセスされた部分のみ圧縮された情報を復元するため、小メモリで処理できる。ただし、復元のオーバヘッドがあるため、処理速度を大きく上げるのは難しいと考えられる。

- バイナリ XML 技術

  DOM 自体ではないが、XML データを効率的に扱う技術として、XML をバイナリ形式で表す試みが多

くなされている[11],[12],[13],[14],[15]。

XML はテキスト形式であることが特徴であるが、タグでデータを記述するためデータサイズが大きくなる。バイナリ XML 技術は、これを解決する技術であり、メモリ消費の削減や、データ処理速度を上げることができるものもある。

しかし、XML データの形式を本来のものから変えてしまうため、現状、利用できる場面が限られる。例えば、企業間のシステム連携は XML が適した代表的な用途であるが、送信側・受信側双方に XML とバイナリ形式の変換処理が必要になり、実際に利用するのは難しい。

これらに対し、広い分野で利用でき、高速で処理できる方式として、我々は SPlitDOM を開発してきた。

3. 従来の SPlitDOM と課題

3.1 SPlitDOM の原理

SPlitDOM は、DOM のメモリ使用量削減と処理高速化を目的とする技術である。DOM と SPlitDOM の処理方式の相違を図 1 に示す。 DOM は対象の XML データ全体をメモリに展開する。SPlitDOM は XML 文書の必要部分のみをメモリ展開する。アプリケーションからは、DOM と同様に、全てメモリ上で参照・更新処理ができる。

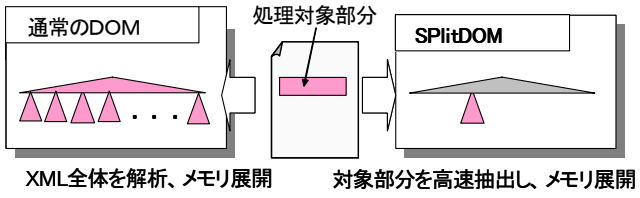

図 1 処理概要

3.1.1 内部アーキテクチャ

図 2に DOM と SPlitDOM の内部アーキテクチャを示す。

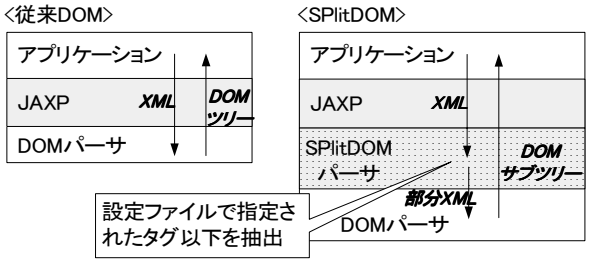

図 2 内部アーキテクチャ

双方とも、アプリケーションからは JAXP(Java API for XML Processing)を介して XML を解析し、DOM ツリーを結果として受け取る(JAXP とは各社の DOM パーサを同一インタフェースで利用可能にする標準 API である)。この際、SPlitDOM では、特定タグ部分(後述の設定ファイルで指定)を元の XML から抽出し、それを DOM パーサに渡す。DOM パーサはその解析結果をアプリケーションに返す。

3.1.2 アプリケーションからの利用方式

前記のように、SPlitDOM は従来の DOM と同様に JAXP のインタフェースを介して利用することができる。図 3に利用例を示す。これはコードの例であり、JAXP の DocumentBuilder オブジェクトを生成して XML を解析する準備を行い、そのオブジェクトを用いて XML を解析して DOM ツリーを取得し、その DOM ツリーの一部をアクセスする。

DOM の場合、JAXP の DocumentBuilder クラスの parse メソッドを実行した時点で、解析結果が DOM 展開される。一方、SPlitDOM では、XML データの特定部分へアクセスした時に、初めてその部分が DOM 展開される。これにより、アプリケーションからは基本的に意識せずに部分的な DOM ツリーを利用することができる。

```
// ###  JAXPインターフェースでDocument（DOM)を取得####
  DocumentBuilderFactory factory =
          DocumentBuilderFactory.newInstance();
  DocumentBuilder builder = factory.newDocumentBuilder();
  Document doc = builder.parse(xmlfile);

// ###  Document中の特定部分を処理  ####
  // “RECORD”タグ下の部分木リストを取得
  NodeList list =
          doc.getElementsByTagName( "RECORD" );
  // リストの最初のレコード要素を取得()
  //SPlitDOMでは、ここで始めて部分木をDOM展開
  Element elem = list.item(0);
```

図 3 利用例（DOM 処理)

3.2 課題

SPlitDOM の実用性を高めるには、以下の２つの課題があった。

- 課題１：アプリケーションへの適用容易化

  一般にアプリケーションでは、ソースコード中の複数の箇所に DOM 処理が存在することが多いため、各箇所ごとに DOM と SPlitDOM を選択できる機能が必要となる。

  JAXP は、基本的に JavaVM(アプリケーション)全体で１つの利用パーサを指定する機構である。従って、そのままでは各箇所ごとに選択する使い方はできない。

- 課題２：実システムを想定した性能評価

  実システムで性能問題が起きやすいのは、業務

ピーク時などの高多重処理である。そのような処理で、高い性能が出せることが必須となる。

3.3 目標

上記の課題の解決を目指し、以下を目標とした。

- 課題１に対する目標
  JAXP 準拠を維持したまま、特定の DOM 処理を選択し SPlitDOM を適用できること、及び、その仕組みを従来の SPlitDOM の性能を低下させずに実現することを目標とした。
- 課題 2 に対する目標
  高多重で XML 処理を行うシステムにおいて、従来の DOM を利用する場合に比べて 1.2 倍以上のスループットが得ることを目指した。これは次の理由による。
  - 一般に、システム全体では DB 処理など XML 以外の様々な処理を含む。それらの処理と XML 処理の割合により、SPlitDOM 適用によるシステム全体の性能向上の度合いは決まる。
  - この割合はシステムの設計依存であるが、様々な設計の Web システムを測定した経験上、XML 処理と非 XML 処理の処理時間が 1:2 程度であるものが多かった。
  - 一方、従来の SPlitDOM は DOM に比べ単体の性能が最大約 2 倍である。よって、XML 処理が 2 倍に向上すると考えると、全体で 1.2 倍の向上が見込める。

4. SPlitDOM 選択適用方式の導入

ここでは、前記の課題１を解決するための新方式について述べる。

4.1 ねらい

前記のように、SPlitDOM は JAXP 準拠とすることで、基本的にアプリケーションの変更なしで使用できる利点がある。一方で、JAXP の性質上、前記の課題１がある。図 4に、JAXP 準拠を維持したまま、課題１を解決することを表す適用イメージを示す。

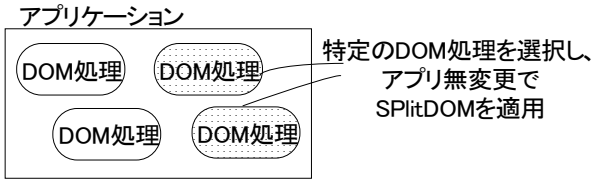

図 4 目標とする適用方法

アプリケーション中の特定の DOM 処理を指定し、SPlitDOM を適用する。アプリケーションを変更せずにこれを実現することを目指した。

4.2 実現上の問題点

上記目標を実現するには、JAXP のインタフェースを変えずに、以下を解決することが必要である。

(1) 複数種の DOM プロセッサの混在利用
JAXP では DocumentBuilder が DOM プロセッサを表す。これは抽象クラスであり、実装は DOM プロセッサのベンダがサブクラスとして提供する。どの実装を用いるかは、システムプロパティでサブクラス名などで指定する機構である。しかし、JavaVM 全体で１つ指定するものであり、複数の使い分けには対応できない。

(2) SPlitDOM を利用する箇所の指定
ユーザがアプリケーション中の DOM 処理箇所を個別に指定できる仕組みが必要である。JAXP でなく専用の API を設ければ実現は容易だが、その場合、アプリケーション無変更という目標は実現できない。

(3) SPlitDOM の処理対象タグの指定
SPlitDOM プロセッサには、XML データ中の DOM 展開したい部分のタグ名を実行パラメタとして与える。これは複数箇所で SPlitDOM を使う場合に、対象 XML データや処理対象部分が異なるので、それぞれ指定できる必要がある。これも上記 2 と同様の問題がある。

4.3 実現方式

上記課題を解決するために、SPlitDOM 選択適用方式を導入した。 図 5に内部アーキテクチャを示す。従来の SPlitDOM パーサの上位レイヤに、DOM パーサと SPlitDOM パーサを使い分けることを可能とするため、選択適用する機構を設けた。JAXP を介してアプリケーションから入力される XML データは、後述の方式により指定されたパーサに渡される。

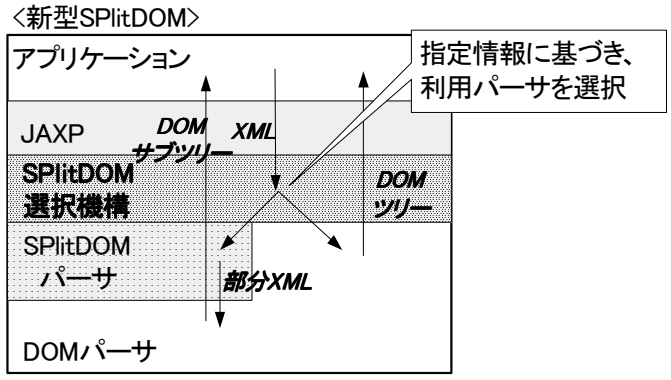

図 5 新型のアーキテクチャ

具体的には、本アーキテクチャにより、実現上の問題

点(1)～(3)を次のように解決した。

(1) FactoryWrapper による DOM プロセッサ混在利用

図 6のクラス図のように、FactoryWrapper クラスを DocumentBuilderFactory のサブクラスとして追加した。

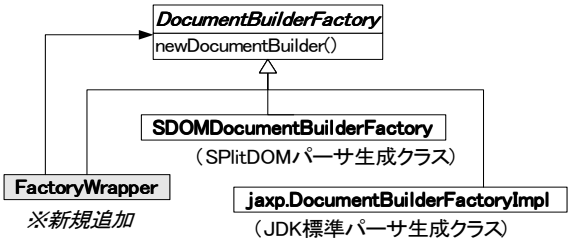

図 6 FactoryWrapper のクラス構成

JAXP による DOM パーサの指定では、従来は、利用する DOM パーサの Factory クラス名をユーザがシステムプロパティで与えていたが、本方式では、代わりに FactoryWrapper を指定する。

FactoryWrapper は、実際に利用する DOM プロセッサの DocumentBuilderFactory へのラッパーとして、次のように機能する。(a)後述の方式で SPlitDOM の使用・不使用を判断する。(b)使用の場合は、SPDOM の DocumentBuilderFactory、そうでなければ従来 DOM の DocumentBuilderFactory を生成し保持する。(c)それ以降の FactoryWrapper に対するメソッド呼出しは、保持する Factory に委譲する(図 7)。

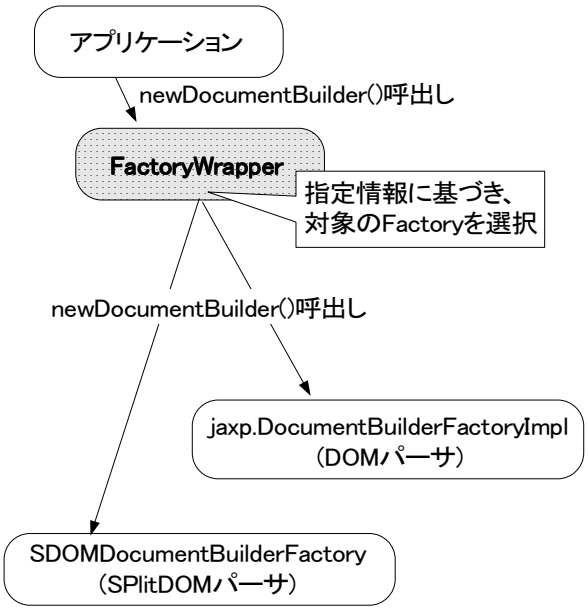

図 7 FactoryWrapper の動作(呼出し関係)

(2) メソッド名による利用箇所の指定

アプリケーションを変更することなく、ユーザが SPlitDOM を使用する箇所を指定する仕組みを次のように実現した。

- 設定ファイルによる呼出し元メソッド指定

  ユーザは事前に設定ファイルにメソッド名を記述しておく。例えば、アプリケーションのクラス C1 にメソッド method1 があり、その中で DOM 処理を行っているとする。これに SPlitDOM を適用したい場合には、設定ファイルに「C1.method1」と記述する。

- 指定メソッド名との照合による SPlitDOM 切換え

  FactoryWrapper の処理フローを図 8に示す。DocumentBuilderFactory(実体は FactoryWrapper)の newDocumentBuilder がアプリケーションから呼ばれた際、newDocumentBuilder では、呼出し元のメソッド名を取得する。Java の例外処理のスタックトレース情報を利用して行う。呼出し元が設定ファイルで記述されたメソッドと同一であれば、SPlitDOM の DocumentBuilder を生成し、そうでなければ通常の DOM の DocumentBuilder を生成して返す。

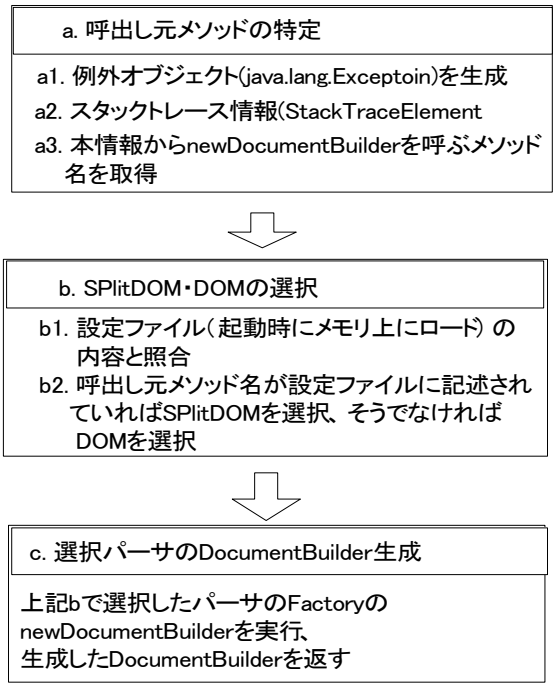

図 8 FactoryWrapper.newDocumentBuilder の処理

(3) 利用箇所ごとの SPlitDOM 実行パラメタの指定

前記のように、複数箇所で SPlitDOM を使用する場合、それぞれに処理対象タグを指定する必要がある。これも設定ファイルを用いて、メソッド名とペアで指定する方式とした。

以上の仕組みによる、利用例を図 9に示す。

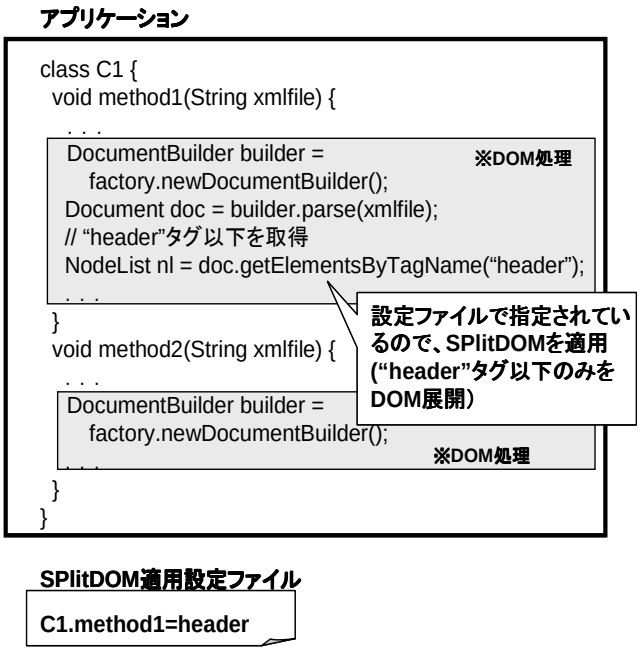

図 9 SPlitDOM の適用例

この例では、アプリケーション中に 2 箇所ある DOM 処理のうち、1 箇所(メソッド method1)で SPlitDOM を利用する。本メソッド内では、タグ header のみを処理するものとする。この場合、図 9のように設定ファイルを記述する。これにより、SPlitDOM が method1 内の DOM 処理に適用される。

4.4 実現方式の性能オーバヘッドの分析

上記の方式において、従来の SPlitDOM に比べて性能上オーバヘッドとなる可能性があった点は、図 8中の「a.呼出し元メソッドの特定」である。本処理では Java の例外オブジェクトを生成しスタックトレース情報を取得している。

本処理について、性能測定を行ったところ、処理時間は最大数ミリ秒であり、DOM の処理時間と比べても実用上全く問題にならない程度だった。今回の機能拡張は、実行時に呼出し元メソッド名を取得しパーサを選択する処理を追加するというものであるが、従来の SPlitDOM の性能を損なわず実現できることが確認できた。

5. Web アプリケーションでの適用評価

前記の機能追加を行った SPlitDOM を、検証用に作成した Web アプリケーションに適用した。これを通して、従来の SPlitDOM の課題であった、アプリケーションへの導入容易化、及び実システムを想定した多重性能について、それぞれ評価を行った。

5.1 対象としたアプリケーション

5.1.1 概要

PC カタログの検索アプリケーションを対象とした。本アプリケーションは、クライアントの Web ブラウザから入力された商品型名や商品名などをキーワードとして、RDB に格納された PC カタログを検索する。キーワードに該当する PC の情報(販売元や価格、PC の仕様など)を RDB から取得して返す。Web ブラウザにはその結果が加工されて表示される。

5.1.2 システム構成

図 10に対象とした PC カタログ検索アプリケーションの構成を示す。

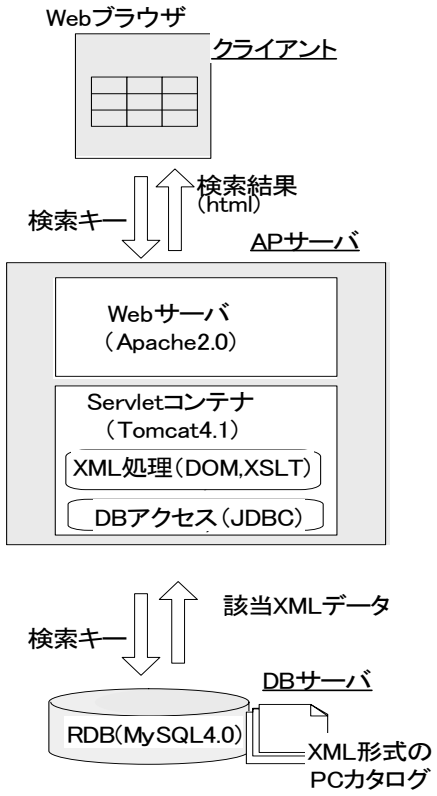

図 10 システム構成

Web サーバと Servlet コンテナ、RDB からなる典型的な構成の Java・Web アプリケーションである。Web ブラウザからのリクエストがあると、JDBC を介して RDB を検索し、検索結果を返す処理を行う。

5.1.3 XML データの記述形式・内容

PC カタログのデータは、図 11のような XML 形式で記述したものである。全体は CatalogHeader タグと CatalogBody タグで表されたヘッダ部とボディ部からなる。ヘッダ部には、その XML データが表す PC の基本情報が記述される。例えば、商品(PC)の ID、名前、価格、特徴などである。ボディ部には、その PC の詳

細情報が記述される。これには、CPU やメモリ、ディスクなどの PC の仕様詳細などが含まれる。

```
<Catalog>
  <CatalogHeader>
    <Product ItemID="1">
    <ProductID>FMVW01S131</ProductID>
    <Feature>Pentium4 最上位機種
       Windows XP Professional 版</Feature>
    <Description>ミドルタワー型</Description>
    <Name>FMV-W601</Name>
    <Price>17.9</Price>
    <Type>DesktopPC</Type>
    </Product>
          . . .
  </CatalogHeader>
  <CatalogBody>
    <ProductDetail>
      <CPU>Intel Pentium4   2.4GHz</CPU>
      <MainMemory>
         <StdCapacity>256</StdCapacity>
         <MaxCapacity>1000</MaxCapacity>
      </MainMemory>
          . . .
  </CatalogBody>
</Catalog>
```

図 11 XML データの例

5.1.4 XML データの RDB 格納方式

図 12に XML データの格納方式を示す。本アプリケーションでは、RDB テーブルの BLOB(Binary Large Object)型カラムに XML データを格納している。

BLOB は RDB テーブルにバイナリデータを格納することができるカラムである。XML データを BLOB 格納する方式は、RDB で XML を扱う場合に用いられる典型的な方式の一つである。

BLOB とは別のカラムには、XML データ自体の検索用に、そのデータに対応する商品 ID や商品名などを格納している。

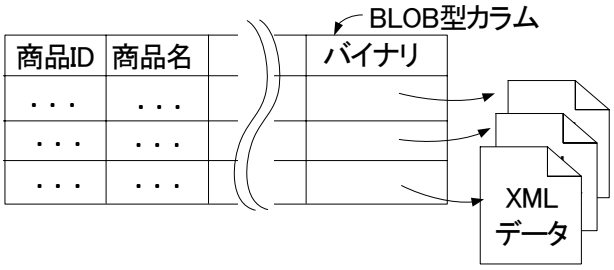

図 12 RDB への XML 格納形式

5.1.5 XML データの処理

図 13に本アプリケーションでの XML データの処理概要を示す。

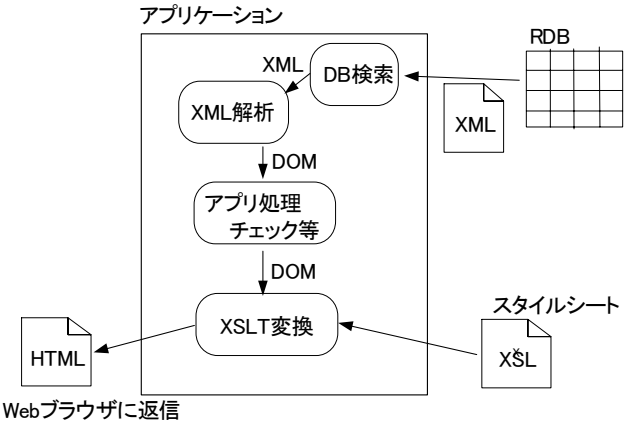

図 13 XML データの処理概要

Web ブラウザから与えられた商品名などを検索キーとして、RDB を検索し、見つかった XML データを BLOB から取得する。アプリケーションでは、このデータを DOM パーサで解析し、DOM として取得する。この DOM に対して、データのチェック処理などを行い、その後、XSLT プロセッサで HTML に変換して Web ブラウザに返す。

5.2 アプリケーションへの SPlitDOM 適用

本アプリケーションでは、PC カタログのうち Web で表示するのは、PC の基本情報(仕様概要)だけとなっている。このため、一連の処理の中で、XML データの参照個所は CatalogHeader タグ内のみであり、SPlitDOM を適用した。

既存アプリケーションに SPlitDOM を適用する際、全体の設計がある程度分かっている場合、一般に適用の手続きは次のようになる。

(1) 適用候補個所の抽出
　アプリケーションの設計を見て、XML データの一部のみ処理している箇所を見つけ、SPlitDOM の適用可否を確認する。

(2) SPlitDOM を利用する設定
　前記の設定ファイルを準備する。

5.3 評価１:SPlitDOM 導入コスト

前記のアプリケーションへの SPlitDOM の適用を通して、導入の容易さを評価した.

5.3.1 評価方法

(2)の設定は数行記述するだけと単純のため、(1)に要したコストが SPlitDOM の導入コストに相当する。このコストを評価する指標として以下を用いた。

指標１：適用個所を見つけるために調べたソースコードの行数。

指標２：適用のために変更したソースコードの行数。

5.3.2 評価データ

以下に前記の指標に関して収集したデータを示す。アプリケーションは、Java ファイル、JSP ファイル、XSL ファイル(XSLT の変換定義である XSL シート)からなる。それぞれについて、それらのコード行数と、指標１と２に該当するコード行数を比較する。

表 1 各指標のソースコード行数

ファイル種別	コード行数	指標1	指標2
Java	736	87	0
JSP	426	0	0
XSL	35	35	0
全体	1197	122	0

5.3.3 考察

指標１の結果は、SPlitDOM 適用の判断のために、Java ファイルと XSL ファイルを 122 行程度を調べたことを意味している。全体から見ると 1 割弱である。実際に行った作業も DOM 処理を含む Java ファイル一つを目視確認した程度であり、少ない工数で済んだといえる。また、指標 2 の結果に示すように、SPlitDOM の適用においてアプリケーションは無変更で済んだ。

以上のように、本アプリケーションの適用を通して、SPlitDOM の導入コストは小さく、簡単に利用できることが確認できた。なお、今回はアプリケーションの設計が分かっている場合の評価であり、SE が自身が開発したアプリケーションに SPlitDOM を適用するケースなどを想定している。これとは異なり、設計を熟知していない第三者が開発したアプリケーションに適用することもあり得る。この場合は、やや時間がかかると思われる。この評価は今後の課題である。

5.4 評価２:性能改善効果

次に、SPlitDOM を適用することによる性能の改善効果について示す。

5.4.1 評価方法

前述のように、SPlitDOM の実用化では実システムを想定した多重処理性能の確認が課題であった。ここでは、アプリケーションに SPlitDOM を適用する前と後のそれぞれに対して、負荷テストツール[16]を用いて多重処理性能の測定を行い、その結果を比較した。

なお、この測定は、XML の処理だけでなく通信や RDB アクセスも含む、アプリケーション全体での性能を見ることになる。分析を行いやすくするため、この測定に先立ち、SPlitDOM 単体の基本性能も参考情報として別途測定した。

5.4.2 評価データ

図 14、図 15に SPlitDOM の基本性能として XML 解析･DOM 生成処理時間とメモリ消費量を示す。 図 16では、アプリケーション全体の多重処理性能(スループット)を示す。

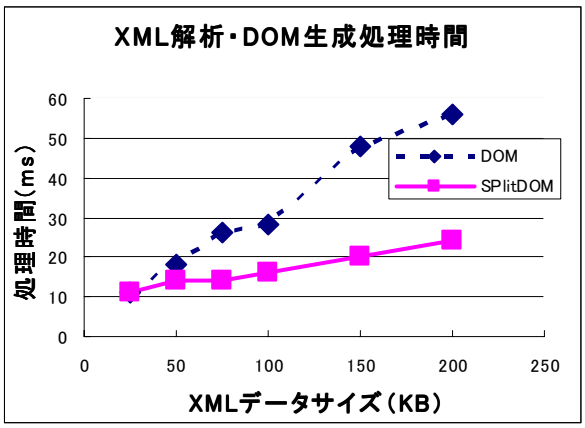

図 14 SPlitDOM の基本性能

基本性能は、後述の測定環境(アプリケーションと同一マシン)で測定した。前記の XML のデータを基に、25KB から 150KB までサイズを変えたデータを作り用いた。このデータの作成は CatalogHeader 部分は固定とし、CatalogBody 部分のデータ量を変更する方法で行った。グラフはそれらのデータで、XML を解析し DOM を生成するまでの時間、及びメモリ消費量を測定したものである(各データでそれぞれ 30 回繰り返し測定した。初回を除いた平均値をプロットしている)。

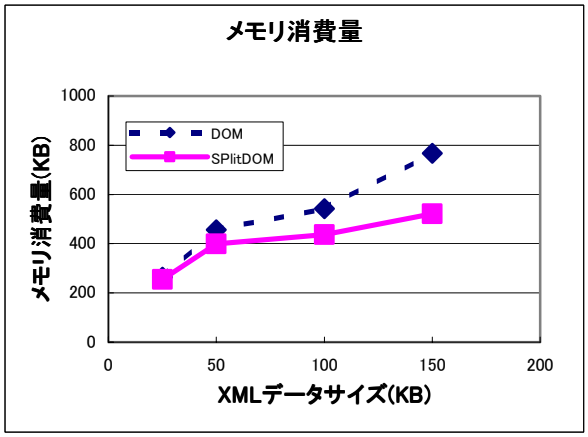

図 15 メモリ消費量

XML データサイズが大きくなるに従って、XML 解析･DOM 生成処理時間、メモリ消費量ともに、効果が大きくなっている。処理時間では SPlitDOM 単体で 2 倍近い性能向上が見込める。

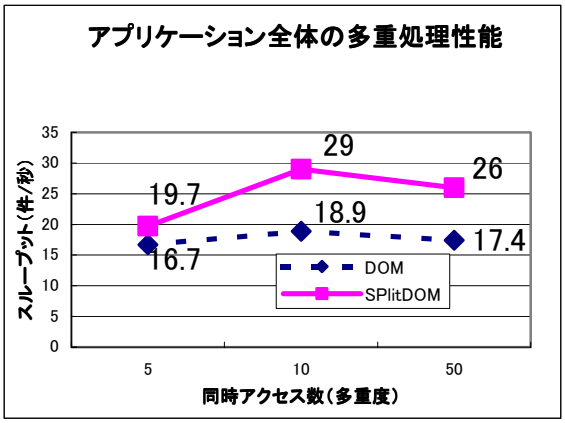

図 16 アプリケーションの多重処理性能

多重処理性能は、負荷テストツールで擬似的に多数のユーザからアクセスする状況を作り測定した。ユーザ数(多重度)が 5,10,50 の 3 パターンで行った。ここで対象とした XML データのサイズは 50KB 強である。DOM 利用時に比べて、SPlitDOM 利用時は最大約 1.5 倍に性能が向上した。なお、本測定で用いたマシン環境は次の通りである。

- サーバ：Pentium4, 1.7GB, 1CPU, メモリ 512MB, Windows2000 Professional
- クライアント：同上
- ネットワーク：100BASE-T

Web サーバとして Apache2.0、Tomcat4.1、DB サーバとして MySQL4.0、Java は JDK1.4 を利用した。XML パーサは JDK1.4 付属の Crimson を利用した。

5.4.3 考察

本アプリケーションに SPlitDOM を適用することにより、5 多重時に全体で 1.17 倍、10 多重,50 多重時に全体で約 1.5 倍に性能が向上し、高多重時に目標を達成できた。5 多重時に効果が小さいのは、XML 処理がボトルネックになっていないためと思われる。10 多重,50 多重では XML 処理がボトルネックになることにより、SPlitDOM の効果が大きく出ていると考えられる。

一般に、システム構築では高負荷になる業務ピーク時に十分耐えられることを考慮して、性能目標値を設定し、サーバマシンなどの見積りを行う。性能が 1.5 倍向上すれば、その分、サーバマシンのスペック、台数を低く押させることができ、従来よりも低コストでシステムが構築できることを意味する。

本アプリケーションでは、通信や DB アクセスなど様々な処理を含み、DOM 処理はその一つである。また、DOM 処理に要する時間も、図 14に示したように、数十ミリ秒程度であり、通常は性能が問題にはならない。しかし、高多重処理時は、DOM 処理が多数同時に実行するため負荷が高くなる。この状況で、SPlitDOM で DOM 処理性能を改善したことが、全体の性能向上につながったと考えられる。

6. まとめ

SPlitDOM の実用化においては、アプリケーションへの導入容易化、及び実システムを想定した性能評価が課題であった。これに対し、導入容易化のための機能拡張と、Web アプリケーションへの適用を通した導入コストと性能の評価を行い、課題を解決した。

性能評価は、典型的な構成の Web アプリケーション 1 件で行った。今後は他にもアプリケーションを増やして効果確認を行う。

文　献

[1] 岡部恵造監修, “XML ビジネス白書 2003”, 翔泳社, 2003

[2] 岡部恵造監修, “XML ビジネス白書 2004”, 翔泳社,2004

[3] JP Morgenthal, Bill la Forge, "Enterprise Application Integration With XML and Java", PRENTICE HALL,2000

[4] 井谷宣子, 吉田茂, “大容量 XML 文書に対する XML パーサの処理性能改善の検討 2, 電子情報通信学会ソサイエティ大会, 2002.

[5] 小田切淳一, 井谷宣子, 吉田茂, “大容量 XML 文書に対するデータ処理効率化手法の評価”, 電子情報通信学会ソサイエティ大会, 2003.

[6] SPlitDOM, 富士通 XML 技術紹介サイト ,URL: http://xml.fujitsu.com/jp/tech/split/index.html

[7] Document Object Model, URL: http://www.w3.org/DOM/

[8] Simple API for XML , URL: http://www.saxproject.org/

[9] Persistent(DOM), InfoNyte, URL: http://www.infonyte.com/en/prod_pdom.html

[10] Mathias Neumuller, John N.Wilson, “Compact In-Memory Representation of XML – Design and Implementation of a compressed DOM for data-centric documents”, “Internal Report”, University of Starathclyde”,2002.

[11] Marc Girardot, Neel Sundaresan,”Millau: an encoding format for efficient representation and exchange of XML over the Web”, 9th International World Wide Web Conference.

[12] Liefke, H. and Suciu, D.,”XMILL: An Efficient Compressor for XML Data”, SIGMOD, 2000.

[13] “WAP Binary XML Content Format”, June 1999. URL: http://www.w3.org/TR/wbxml.

[14] Binary XML: Position paper for The W3C Workshop on Binary Interchange of XML Information Item Sets.

[15] 小林亜令, 松本一則, 井ノ上 直己, "汎用 XML 文書符号化方式「XEUS」の性能評価", 情報科学技術フォーラム,2003

[16] E-SUP APTest/Web, URL: Test/Webhttp://www.fasol.fujitsu.com/services/esup/apt/index.html