# Difference between DOM and SAX.

By Yvan Ayaay, Technical Support Engineer, 4D Inc.

TN 06-14

## Introduction

4th Dimension は、2 種類の XML ドキュメント解析、つまり DOM(Document Object Model)および SAX(Simple API XML)をサポートしています。DOM および SAX は、どちらも XML を操作するためのコマンド体系であるという点では同じですが、プログラムスタイルやスコープが異なっているほか、長所や短所という点でも違いがあります。このテクニカルノートでは、両コマンドの相違について論じ、要点を際立たせるため、両方のコマンドで XML を解析するサンプルメソッドを紹介しています。読解にあたっては、基本的な XML の理解が前提となっていますので、必要に応じ「XML: An Introduction」などのテクニカルノートを参照してください。

## Overview

DOM(Document Object Model)および SAX(Simple API XML)は、4th Dimension で提供されている 2 種類の XML 解析モードです。XML(Extensible Markup Language)は、標準化されたデータフォーマットで、汎用的に交換可能なデータ文書の作成を目的としています。DOM と SAX は、XML を解析する方法、機能制限、プログラムの書き方などが異なっています。具体的には、DOM は XML を解析してメモリ上に構築し、ノードからノードへ自由に移動することができるのに対し、SAX は XML ドキュメントをメモリに格納せずシーケンシャルに解析してゆきます。SAX はメモリ効率が良い反面、アクセスが連続的であるため小回りがききません。一方、ランダムアクセスのできる DOM は柔軟性に優れているのが特徴です。いずれにしても、どちらのモードを使用するかでプログラムの書き方がかなり異なってくることになります。

次に DOM と SAX を比較し、それぞれの長所および短所について分析したいと思います。その後、XML ドキュメントの作成と解析を DOM および SAX コマンドで実行し、それぞれのコマンドの相違点について考えてみることにしましょう。

## Advantages and Disadvantages

SAX および DOM の XML コマンドは、XML ドキュメントを解析する方法が相当違っており、それぞれに長所また短所があります。

メモリ使用：DOM は XML 構造全体をメモリに収納します。このため、各要素に対するアクセスは極めて高速ですが、XML のサイズによってはメモリに納まりきらない恐れがあります。一方、SAX はドキュメントをメモリに読み込むのではなく、イベントストリーミング方式で解析するため、扱うことのできる XML ドキュメントのサイズに制限がなく、利用可能なメモリ量を気にする必要もありません。

アクセス：SAX では XML ドキュメントを冒頭から末尾まで、ノードからノードへと順番に解析してゆきます。タグ開始、タグ終了などの要素が識別されるたびにイベントが発生し、解析の方向は常に一定です。XML ドキュメントに対してランダムアクセスを実行することはできません。さらに SAX で既存の XML を解析する場合、ドキュメントを Read Only で開く必要があり、要素名、要素値、属性などを更新することはできません。一方、DOM ではメモリ上に XML ツリーが構築されるので、XML の中を自由に行き来することができ、解析の方向は限定されておらず、任意のノードにいつでも移動することができます。加えて DOM には要素名、要素値、属性などを更新するためのコマンドが豊富に存在します。

どちらを選ぶか：DOM と SAX のどちらを使用するかは、上記の特徴を良く考慮した上で判断するべきことです。ある程度サイズが限定された XML ファイルであれば、、間違いなく DOM のほうに軍配が上がります。一方、大きな XML ファイルであれば、SAX のほうが向いています。別の点として、一度読み込んで解析するだけの XML であれば、シーケシャルな SAX でも充分かもしれませんが、XML の中から特定の要素をランダムに取り出すような処理では DOM のほうが便利です。また、構造の複雑な XML は概して DOM で解析したほうが楽です。いずれにしても、DOM と SAX ではプログラムの書き方がかなり異なっているので、最終的にはプログラマの好みという問題になるかもしれません。使い易いと感じる方を選択してください。

## Creating an XML Document

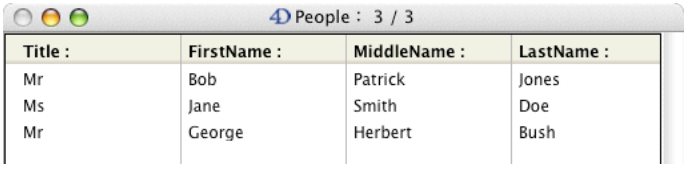
DOM または SAX いずれのモードでも XML ドキュメントを作成することができます。とはいえドキュメントを作成する手順はかなり違っています。DOM では要素ノードに対するリファレンスを使用して階層構造体をメモリ上に構築し、最後に全体をファイルへ書き出します。一方、SAX では、最初に空のドキュメントを作成し、開かれたドキュメントリファレンスに対して連続的にデータを書き出してゆきます。

DOM で XML ドキュメントを作成する場合、はじめに XML ツリーと呼ばれるものをメモリ上に

構築します。要素および値はこのツリーに追加してゆき、最後にそのツリーをファイルに書き出します。要素を作成するたびに、整えられた XML 耕造を維持管理するためのリファレンスが返されます。要素中の要素、同一階層の要素を作成する場合は、基準となる要素のリファレンスが必要です。

対照的に、SAX で XML ドキュメントを作成する場合は、最初にドキュメントを作成し、要素や値は後から追加してゆきます。このとき 4th Dimension で一般的なドキュメントを作成する場合と同じ Docref が使用されます。DOM では要素の数だけリファレンスが存在しますが、SAX ではこの Docref が唯一のリファレンスであり、それぞれの要素は第一階層から順に追加されてゆきます。すべての要素が追加された時点で、ドキュメントは閉じられます。

それぞれのモードで XML ドキュメントを作成する手順を理解するために次のようなレコードを登録されたテーブルについて考慮してみましょう：




このデータベースに基づき、People の名前を書き出して次のような XML ドキュメントを作成したいとします：

```
<?xml version="1.0" encoding="UTF-16" standalone="no" ?>
 <people>
   <name title="Mr">
     <first>Bob</first>
     <middle>Patrick</middle>
     <last>Jones</last>
   </name>
   <name title="Ms">
     <first>Jane</first>
     <middle>Smith</middle>
     <last>Doe</last>
   </name>
   <name title="Mr">
     <first>George</first>
     <middle>Herbert</middle>
     <last>Bush</last>
   </name>
 </people>
```

Title フィールドは Name 要素の属性として定義し、残る FirstName、MiddleName、LastName フィールドは Name 要素の子要素として定義します。

### In DOM mode

上記のような XML を作成するための DOM コードは次のようになります：

```
C_STRING(16;vRootRef;vElemRef)
C_TEXT($1;$rootElem;$vElem1;$vElem2;$vElem3)
C_TEXT($mtitle;$myfilepath)
$rootElem:="people"
  `$myfilepath:="TestPeopleDOM1.xml"
$myfilepath:=$1
vRootRef:=DOM Create XML Ref($rootElem)
ALL RECORDS([People])
DOM SET XML OPTIONS(vRootRef;"UTF-16";False)

For (i;1;Records in selection([People])
    $MainElem:="/people/name"
    $mtitle:=[People]Title
    vElemRef:=DOM Create XML element(vRootRef;$MainElem;"title";$mtitle)
    $vElem1:="first"
    vElemRef1:=DOM Create XML element(vElemRef;$vElem1)
    $fname:=[People]FirstName
    DOM SET XML ELEMENT VALUE(vElemRef1;$fname)
    $vElem2:="middle"
    vElemRef2:=DOM Create XML element(vElemRef;$vElem2)
    $middle:=[People]MiddleName
    DOM SET XML ELEMENT VALUE(vElemRef2;$middle)
    $vElem3:="last"
    vElemRef3:=DOM Create XML element(vElemRef;$vElem3)
    $last:=[People]LastName
    DOM SET XML ELEMENT VALUE(vElemRef3;$last)
    NEXT RECORD([People])
End for
DOM EXPORT TO FILE(vRootRef;$myfilepath)
```

コードから分かるように、最初に「People」をルート要素とする XML ツリーを作成します。次いで「People」に対するリファンレスを使用して属性「Title」を持つ子要素「Name」が作成されます。FirstName、MiddleName、LastName 要素は「Name」要素の子なので、いずれも「Name」に対するリファレンスを使用して作成します。子要素のリファンレンスが確定した時点で、それぞれの値を定義することができます。各レコードの「Name」ノードを作成するためには毎回ルート要素「People」に対するリファンレスを使用します。各「Name」要素の小要素はそれぞれの親要素である「Name」に対するリファレンスを使用して作成します。すべての要素が追加されるとツリーがファイルに書き出されます。このように DOM では要素に対するリファレンスが重要な意味を持っています。

### In SAX mode

まったく同じ XML ドキュメントは次のような SAX コードでも作成することができます：

```
C_TIME($docref)
C_TEXT($1;$rootElem;$vElem1;$vElem2;$vElem3)
C_TEXT($mtitle;$myfilepath)

ALL RECORDS([People])
  `$myfilepath:="C:¥¥XMLStuff¥¥TestPeopleSAX2.xml"
$myfilepath:=$1
$DocRef:=Create document($myfilepath)
SAX SET XML OPTIONS($DocRef;"UTF-16";True)  ` エンコーディングおよびスタンドアロンの設定
$rootElem:="people"
  `ルート要素"people"の作成。同名の開始タグが出力される。
SAX OPEN XML ELEMENT($DocRef;$rootElem)
  `全レコードをループ。
For ($i;1;Records in selection([People]))
    $MainElem:="name"
    $mtitle:=[People]Title
      `親要素"name"を title 属性付きで作成。開始タグ"name"が出力される。
    SAX OPEN XML ELEMENT($DocRef;$MainElem;"title";$mtitle)
    $vElem1:="first"
      `子要素"first"を作成。開始タグ"first"が出力される。
    SAX OPEN XML ELEMENT($DocRef;$vElem1)
    $fname:=[People]FirstName
      `first 要素に値を設定する。
    SAX ADD XML ELEMENT VALUE($DocRef;$fname)
      `最後に作成した要素を閉じる、終了タグが出力される。
    SAX CLOSE XML ELEMENT($DocRef)
    $vElem2:="middle"
      `子要素"middle"開始タグ"middle"が出力される。
    SAX OPEN XML ELEMENT($DocRef;$vElem2)
    $middle:=[People]MiddleName
      `middle 要素に値を設定する。
    SAX ADD XML ELEMENT VALUE($DocRef;$middle)
      `最後に作成した要素を閉じる、終了タグが出力される。
    SAX CLOSE XML ELEMENT($DocRef)
    $vElem3:="last"
      `子要素"last"開始タグ"last"が出力される。
    SAX OPEN XML ELEMENT($DocRef;$vElem3)
    $last:=[People]LastName
      ` last 要素に値を設定する。
    SAX ADD XML ELEMENT VALUE($DocRef;$last)
      `最後に作成した要素を閉じる、終了タグが出力される。
    SAX CLOSE XML ELEMENT($DocRef)
      `"name"要素を閉じる。終了タグが出力される。
```

```
      SAX CLOSE XML ELEMENT($DocRef)
      NEXT RECORD([People])
End for
   `"people"要素を閉じる。
SAX CLOSE XML ELEMENT($DocRef)
CLOSE DOCUMENT($DocRef)
```

コードから分かるように、最初に空のドキュメントが作成されます。次いでドキュメントリファレンス Docref を使用してデータが順に書き出されてゆきます。要素は冒頭から末尾までストリームで出力されることになります。SAX OPEN XML ELEMENT コマンドで要素が追加されるたびに、その要素に対応する開始タグがファイルに書き出されます。妥当な XML 構造を構築するためには、適切な箇所でそのタグを閉じなくてはなりません。SAX OPEN XML ELEMENT で開かれたタグを閉じるためのコマンドが SAX CLOSE XML ELEMENT です。子要素を作成する場合、親要素を開いたまま次の要素を開きます。同一階層の要素を作成する場合、要素を一旦閉じてから次の要素を開きます。このように開かれたタグはそれぞれ適切に閉じられなくてはなりません。最後に CLOSE DOCUMENT コマンドでドキュメントが閉じられます。

## Opening and Traversing XML Documents

DOM と SAX では、ドキュメントを開いて解析する方法がまったく異なっています。DOM の場合、ドキュメントを開いて解析するとメモリ上に XML ツリーが構築されます。XPath 表記(http://www.4d.com/docs/CMU/CMU10099.HTM)を使用すれば、要素に対するリファレンスを起点として任意のノードにアクセスすることができます。要素の名前、値を自由に取得し、変更することができるのも DOM の特徴です。一方、SAX では必ず Read Only モードでドキュメントを解析する必要があります。したがって SAX には要素の名前や値を更新するコマンドはありません。

### Parsing in DOM mode

DOM でドキュメントを解析し、ツリーを構築してルート要素に対するリファレンスを返すコマンドは DOM Parse XML Source です。DOM には要素や属性を読み取るためのコマンドが豊富に存在します。(http://www.4d.com/docs/V6U/V6U00067.HTM)要素を探したり、数えたりするコマンドもあり、要素名、値、属性は自由に変更することができます。ある要素を起点として、上位階層(親)、下位階層(子)、次または前の同一階層(シブリング)にアクセスするためのコマンドが使用でき、XML 構造の中を自由に移動することができます。XPath 表記(XML 内をナビゲートする目的で設計された表記)を使用すれば、フルアクセスパスを示さなくても特定の位置にある要素へダイレクトにアクセスすることができます。

次のサンプルコードは、XML 内に存在する特定要素のすべてのインスタンスを探し出してその

値を返すメソッドです。メソッドは 3 個のパラメータを受け取ります。第 1 は探すべき要素名(FirstName、MiddleName、LastName)、第 2 は見つかった要素の値を代入する配列に対するポインタ、第 3 は解析する XML ドキュメントのパス名です。メソッドに必要なパラメータを渡すと、その要素についてすべてのインスタンスがサーチされ、要素値が配列に代入されます。

```
  `Method: DOMGetElemVal
  `説明　　探すべき要素名(FirstName、MiddleName、LastName)、
  `　　見つかった要素の値を代入する配列に対するポインタ、
  `　　解析する XML ドキュメントのパス名を渡してください。
  `　　要素のすべてのインスタンスがサーチされ、値が配列に代入されます。

C_TEXT($1;$fElem;$pathfile;$3)
C_POINTER($2)
$pathfile:=$3
  `$pathfile:="C:¥¥XMLStuff¥¥TestPeopleDOM2.xml"
ARRAY TEXT(ElemVals;0)
  `XML ドキュメントを解析。
$ref1:=DOM Parse XML source($pathfile)
  `最初の要素"name"を探してリファレンスを取得。
vName:=DOM Find XML element($ref1;"/people/name")
$fElem:=$1
  `すべての要素を処理するまでループ。
While ((OK=1) & (vName#""))
      `探している要素に対するリファレンスを取得。
    vFirst:=DOM Find XML element(vName;"/name/"+$fElem)
      `要素の値を取得。
    DOM GET XML ELEMENT VALUE(vFirst;value)
      `値を配列に追加。
    APPEND TO ARRAY(ElemVals;value)
      `次のシブリング"name"要素へ移動。
    vName:=DOM Get Next sibling XML element(vName)
End while
  `値で満たされた配列を渡された配列にコピー。
COPY ARRAY(ElemVals;$2->)
```

コードから分かるように、XML ドキュメントは最初に DOM PARSE XML Source で解析されます。次に DOM Find XML element で探すべき要素の親要素である「Name」ノードを探し、見つかったならば、そのリファレンスから子要素のリファレンスを調べて DOM GET XML ELEMENT VALUE で値を取得しています。次の「Name」ノードは DOM Get Next sibling XML element で探します。このときやはり「Name」ノードのリファレンスが使用されます。以上の処理が、すべての「Name」ノードを読み取り終えるまで繰り返されます。

### Parsing in SAX mode

同じことを SAX で実行しようとする場合、まずは Open document で XML ドキュメントを開きます。ドキュメントは 4th Dimension と Xerces ライブラリの競合を避けるため、Read Only モードで開く必要があります。XML ドキュメントと一般のドキュメントを同時に開く場合、特にこの点に注意してください。Read and Write モードで開ければドキュメントに対して SAX の解析コマンドを使用すようとすると、警告メッセージが表示され、処理が中止されます。

SAX では、ドキュメントの解析は冒頭から末尾までシーケンシャルに実行することになります。ノードをひとつ前に進めるために使用されるコマンドが SAX Get XML node です。コマンドを実行すると、遭遇した SAX イベントが返されます。以下は SAX イベントの一覧です：

| 定数 | タイプ | 値 |
|---|---|---|
| XML Start Document | 倍長整数 | 1 |
| XML Comment | 倍長整数 | 2 |
| XML Processing Instruction | 倍長整数 | 3 |
| XML Start Element | 倍長整数 | 4 |
| XML End Element | 倍長整数 | 5 |
| XML DATA | 倍長整数 | 6 |
| XML CDATA | 倍長整数 | 7 |
| XML Entity | 倍長整数 | 8 |
| XML End Document | 倍長整数 | 9 |

遭遇したイベントの種類に応じ、要素名、要素値、属性、コメント、CDATA などを取得することができます。望むデータを取り出すためには、SAX Get XML node を実行するたびにイベントの種類を調べなくてはなりません。例えば、特定の要素を探している場合、XML Start Element が返されるまで SAX Get XML node を繰り返し実行しなくてはならず、イベントが返された時点で SAX GET XML ELEMENT コマンドを使用しなくてはなりません。そのようにしてやっとそれが探していた要素であるかを調べることができます。

次のサンプルコードは、XML 内に存在する特定要素のすべてのインスタンスを探し出してその値を返すメソッドです。前出の DOM メソッドと実行の結果は同じですが、SAX モードで XML を処理している点が異なっています。

```
  `Method: SAXGetElemVal

C_TEXT($1;$fElem;$pathfile;$3)
C_POINTER($2)
ARRAY TEXT(ElemVals;0)
$pathfile:=$3  ` XML ドキュメントのパス。
  `"C:¥¥XMLStuff¥¥testpeopledom2.xml"

DocRef:=Open document($pathfile;"xml";Read Mode )  ` Read Only で開く。
$fElem:=$1  `探すべき要素。
```

```
If (OK=1)
    Repeat
        MyEvent:=SAX Get XML node(DocRef)
        If (MyEvent=XML Start Element )  ` イベントの種類が XML Start Element のとき
            `要素名を取得。
         SAX GET XML ELEMENT(DocRef;name;prefix;attrNames;attrValues)
         If (name=$fElem)  `探している要素名であるとき
          MyEvent:=SAX Get XML node(DocRef)  ` 次のノードへ移動。
          If (MyEvent=XML DATA )  ` イベントの種類が XML DATA のとき
            `要素の値を取得。
           SAX GET XML ELEMENT VALUE(DocRef;Value)
            `配列に値を追加。
           APPEND TO ARRAY(ElemVals;Value)
          End if
         End if
        End if
    Until (MyEvent=XML End Document )  ` XML ドキュメントの終わりに達するまでループ。
End if
CLOSE DOCUMENT(DocRef)
  `配列を渡された配列にコピー。
COPY ARRAY(ElemVals;$2->)
```

コードから分かるように、最初に Read Only モードでドキュメントが開かれます。次に、ドキュメントの終わりに達するまで SAX GET XML node をコールし続けます。コマンドは毎回イベントの種類を返し、それが XML Start element であれば SAX GET XML ELEMENT で要素の情報が調べられます。それが探していた要素であれば、再度 SAX Get XML node をコールしてノードをひとつ前に進め、SAX GET XML ELEMENT VALUE で要素の値を取り出します。この値は配列に代入されます。このように SAX では単一の Docref を使用してシーケンシャルにドキュメントを解析します。

## Conclusion

4th Dimension で提供されている DOM(Document Object Model)と SAX(Simple API XML)は、それぞれ性格の異なるコマンドであり、別々の方法で XML を解析します。どちらのコマンドにも特有の長所および短所が存在します。DOM はドキュメント全体を読み込んでメモリに格納するため、柔軟で自由なアクセスができる反面、メモリに限りがある状況では大きなドキュメントが扱えないという問題があります。既存の XML を解析する場合、要素名や値を変更できるというのも特徴です。これに対し、SAX ではメモリ上に XML 構造をコピーすることはなく、ドキュメントを冒頭から末尾まで順に読み込んでゆき、ノードごとにイベントを処理するというアプローチがとられます。メモリの容量には制限されませんが、DOM ほど柔軟には読み込みや解析ができず、またデータを更新することができないという性質を帯びています。