

Using XML in Flash CS3/AS3 の日本語訳

本ドキュメントは、KIRUPA.COM のサイトにある“Using XML in Flash CS3/AS3”をヒム・カンパニー 永井勝則が自主的に日本語に訳したものです。

<http://www.himco.jp>

knagai@himco.jp

(2007/10/7)

本ドキュメントの原文は

http://www.kirupa.com/developer/flashcs3/using_xml_as3_pg1.htm

です。

KIRUPA.COM

Flash CS3/AS3 での XML の使用

page 1

AS3 での大きな変更の1つは XML コンテンツの処理方法です。AS2 時代からの類似性が多く引き継がれてはいますが、AS3 には時間の節約になる新しい精密さがあり、XML ファイルの操作が簡単になります。導入されたものの1つは短く E4X と呼ばれる EcmaScript for XML です。このチュートリアルでは、以下の XML データのさまざまな部分の解析を試行しながら、XML の工夫を少しずつ学んでいきます。

```
<Books>
  <Book ISBN="0553212419">
    <title>Sherlock Holmes: Complete Novels...
    <author>Sir Arthur Conan Doyle</author>
  </Book>
  <Book ISBN="0743273567">
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
  </Book>
  <Book ISBN="0684826976">
    <title>Undaunted Courage</title>
    <author>Stephen E. Ambrose</author>
  </Book>
  <Book ISBN="0743203178">
    <title>Nothing Like It In the World</title>
    <author>Stephen E. Ambrose</author>
  </Book>
</Books>
```

このチュートリアルではさまざまな事柄を扱うので、わたしは前もってその内容を以下に示しておきます。

- I. XML の構造
- II. XML ファイルのロード
- III. XML データの読み取り
 - i. XML と XMLList
 - ii. データへの直接的なアクセス
 - iii. データへの間接的なアクセス
 - iv. 全ての children() の呼び出し
 - v. 属性の読み取り
- IV. 変数のフィルタリング
 - i. ノード値のフィルタリング

ii. 属性情報のフィルタリング

ご覧のように、Flash 内で操作方法を学ぶトピックが多くあります。ではその1つめ、XML ファイルに関する説明から始めましょう。

XML の構造

XML ファイルは本質的にさまざまな枝や葉を持った木で、枝や葉は通常ノードや値と呼ばれます。前述の XML データもその例外ではありません。次の図はサンプルの XML データを表す方法の1つです。



この画像の各ボックスはノード(節のこと)と考えることができます。われわれは `Books` という名前のメインのルートノード(根っこの節)を持ち、そのルートノードは、`Book` という名前の4つの子ノードを持っています。Book ノードは `ISBN` 情報(国際標準図書番号のこと)を含み、そのノードに直接保持された情報は属性と呼ばれます。情報はまた子ノードにも保持することができます。書籍のタイトルと著者の情報は `title` と `author` という名前の子ノード内に保持されます。

このタイプの階層は、すべての XML データと同様、本質的に木です。コンピュータの世界では、木は、そのてっぺんでの概観から葉での詳細まで、情報の分類(カテゴリ分け)の助けになるという理由で大いに役立ちます。前述のデータからは、子ノード(Book、Title、Author)がしたがう項目をその親ノード(Books)が設定しているということが簡単に分かります。

このチュートリアルが終わるまでにみなさんはノードと属性の情報にアクセスする複数の方法を学びます。データにアクセスするにはまず XML データをロードする必要があります。ではその方法について次ページで見ていくことにしましょう。

page 2

前ページではこのチュートリアルの概要と XML ドキュメント領域の主な部分を見ました。このページでは、XML データをアプリケーション内にロードする方法を学ぶことでわれわれの旅をスタートさせましょう。

XML ファイルのロード

XML ファイルをロードするときには、したがうべきいくつかの手順があります。まず XML ファイルをロードする必要があります。詳しくいうと、ロードしたデータは部分的には操作できないので、XML ファイルが完全にロードされたタイミングを知る方法が必要です。これは複雑なように見えますが、ビルトインのクラスが助けてくれます。

次の XML ファイルをロードするコードをご覧ください。

```
var xmlLoader:URLLoader = new URLLoader();
var xmlData:XML = new XML();

xmlLoader.addEventListener(Event.COMPLETE, LoadXML);

xmlLoader.load(new URLRequest("http://www.kirupa.com/net/files/sampleXML.xml"));

function LoadXML(e:Event):void {
    xmlData = new XML(e.target.data);
    trace(xmlData);
}
```

このコードをアクションパネルにペーストしアプリケーションをプレビューしてください。すると出力パネルに XML ファイルの内容が表示されます。

```
Output x
<Books>
  <Book ISBN="0553212419">
    <title>Sherlock Holmes: Complete Novels and
    <author>Sir Arthur Conan Doyle</author>
  </Book>
  <Book ISBN="0743273567">
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
  </Book>
  <Book ISBN="0684826976">
    <title>Undaunted Courage</title>
    <author>Stephen E. Ambrose</author>
  </Book>
  <Book ISBN="0743203178">
    <title>Nothing Like It In the World</title>
    <author>Stephen E. Ambrose</author>
  </Book>
</Books>
```

[XML が出力パネルに表示されます]

コードの動作が確認できたので、コードを詳しく見て、XML ファイルをメモリにロードする方法を理解しましょう。

```
var xmlLoader:URLLoader = new URLLoader();
var xmlData:XML = new XML();
```

この2行では、xmlLoader と xmlData という名前の2つの変数を宣言しています。xmlLoader 変数は URLLoader 型です。URLLoader クラスは URL のような外部ソースからデータをロードするときの助けになります。

xmlData 変数は XML 型です。XML クラスは XML データへのアクセスと操作に関する多くの機能性を提供します。このチュートリアルでは XML オブジェクトを使用するので、XML オブジェクトについては今後のセクションでさらに説明を加えます。

```
xmlLoader.addEventListener(Event.COMPLETE, LoadXML);
```

このコード行では、xmlLoader オブジェクトにイベントリスナーを登録しています。イベントリスナーは基本的に、その名前からも分かる通り、あるイベントを監視(リッスン、聞き耳を立てる)し、そのイベントが発生したらリスナー関数を呼び出します。

このコードでは完了したというイベント(Event.COMPLETE)を監視し、そのイベントを耳にしたとき、

LoadXML リスナー関数を呼び出します。ここでは COMPLETE イベントの発生を待つので、データのすべてがロードされるまで、早まってそのデータの操作を始めてはいけません。

```
xmlLoader.load(new URLRequest("http://www.kirupa.com/net/files/sampleXML.xml"));
```

xmlLoader オブジェクトはこれで終わりではありません。この行では xmlLoader の load メソッドを呼び出しています。

load メソッドはその引数として URLRequest オブジェクトのみをとります。その理由は以下の通りです。インターネットからデータをダウンロードするとき、そのデータはストリームとして区分的にダウンロードされます。URLRequest クラスは全データを確実に丸ごとロードし、load メソッドで一度に XML データのすべてが取得できるようにします。

では前に addEventListener メソッドを述べる時簡単に触れた LoadXML メソッドを見てみましょう。

```
function LoadXML(e:Event):void {  
    xmlData = new XML(e.target.data);  
    trace(xmlData);  
}
```

LoadXML リスナー関数は、xmlLoader のイベントリスナーが COMPLETE イベントを検出したとき呼び出されます。COMPLETE イベントは、load メソッドを通して外部データが完全にロードされたときに発射されます。

LoadXML 関数はリスナーなので、通常関数の作成方法とは少し異なります。LoadXML 関数は e という名前の Event 型の引数を1つとり、この引数 e はイベントに関する多くのデータを持っています。

イベントリスナーに送られたデータには、e.target.data を調べることでアクセスできます。イベントリスナーは URLLoader オブジェクトが発射しているので、送られるデータは、前に URLRequest を通してロードした XML データです。

最後に全データを xmlData という名前の XML オブジェクトに保持します。LoadXML メソッドが問題なく実行されると、すべての XML データはメモリ内に保持されます。

page 3

ここまで順調に進んできました。前のページでは外部 XML ファイルを指定して Flash 内にそれをロードする方法を学びました。ではそのデータの読み取りについてこのページと以降のページで見ていくことにしましょう。

XML データの読み取り

XML データを処理する上で重要な部分はそのデータの読み取る方法を知ることです。XML データの読み取りに関するコードを見る前に、使用することになる2つのクラスについて少し見てみましょう。

XML と XMLList

1つめのクラスは XML クラスです。これはみなさんにもすでにおなじみのはずです。XML オブジェクトはコード内で宣言していますし、外部ファイルからロードしたデータは新しい XML オブジェクトとして保持しました。このクラスは XML ファイルに保持したデータの操作とデータへのアクセスに必要な基本的な機能を提供します。

2つめのクラスはこれから使用する XMLList です。XMLList は XML オブジェクトを保持する標準的なリストに似ています。主な(そしてクールな)違いは XML オブジェクト上で実行できるほとんどの操作が、XMLList オブジェクトにも適用できるということです。

データへの直接的なアクセス

AS 3 での XML データへのアクセスは、AS 2 のときよりもずいぶん簡単になりました。その理由としては、この記事の最初で述べたように、E4X の使用があります。既存の LoadXML 関数を次の2つの関数に置き換えます。

```
function LoadXML(e:Event):void {
    xmlData = new XML(e.target.data);
    ParseBooks(xmlData);
}

function ParseBooks(bookInput:XML):void {
    trace("XML Output");
    trace("-----");
    trace(bookInput);
}
```

ここでは、引数として XML オブジェクトをとる新しい ParseBooks 関数を作成しています。また LoadXML 関数では前の trace ステートメントを、引数として XML オブジェクトの xmlData を指定した ParseBooks メソッドへの呼び出しに置き換えています。

(コードの変更による)整合性の検査(サニティチェック)のため、アプリケーションをプレビューします。すると前に見た同じ XML データが出力されます。

```
<Books>
  <Book ISBN="0553212419">
    <title>Sherlock Holmes: Complete Novels...
    <author>Sir Arthur Conan Doyle</author>
  </Book>
  <Book ISBN="0743273567">
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
  </Book>
  <Book ISBN="0684826976">
    <title>Undaunted Courage</title>
    <author>Stephen E. Ambrose</author>
  </Book>
  <Book ISBN="0743203178">
    <title>Nothing Like It In the World</title>
    <author>Stephen E. Ambrose</author>
  </Book>
</Books>
```

ではこのデータからすべての Book 要素にアクセスしたいとしましょう。AS 2 では XML ファイルを走査するコードを記述し、Book ノードに到達したら一度止める必要がありました。AS 3 では `trace(bookInput)` の行に 1 語加えるだけですみます。

```
function ParseBooks(bookInput:XML):void {
  trace("XML Output");
  trace("-----");
  trace(bookInput.Book);
}
```

`trace` ステートメントを `bookInput` から `bookInput.Book` に変更しました。この Book は今対象としている要素の名前を表しています。このコードを実行すると、Book の要素名に一致するノードだけが表示されます。

XML Output

```
-----
<Book ISBN="0553212419">
  <title>Sherlock Holmes: Complete Novels and Stories, Vol 1</title>
  <author>Sir Arthur Conan Doyle</author>
</Book>
<Book ISBN="0743273567">
```

```
<title>The Great Gatsby</title>
<author>F. Scott Fitzgerald</author>
</Book>
<Book ISBN="0684826976">
  <title>Undaunted Courage</title>
  <author>Stephen E. Ambrose</author>
</Book>
<Book ISBN="0743203178">
  <title>Nothing Like It In the World</title>
  <author>Stephen E. Ambrose</author>
</Book>
```

いたって簡単なように見えます。では先に進みましょう。たとえば Book ノードの内部にある著者名すべてだけがほしい場合を考えてみましょう。著者の名前は<author>要素内に保持されているので、trace ステートメントを bookInput.Book から bookInput.Book.author に変更します。

```
function ParseBooks(bookInput:XML):void {
  trace("XML Output");
  trace("-----");
  trace(bookInput.Book.author);
}
```

アプリケーションをプレビューすると、出力パネルには以下が表示されます。

```
XML Output
-----
<author>Sir Arthur Conan Doyle</author>
<author>F. Scott Fitzgerald</author>
<author>Stephen E. Ambrose</author>
<author>Stephen E. Ambrose</author>
```

注意が必要なのは、今見ているのは著者名のリストではありますが、その名前は<author>タグで囲まれていることです。その理由は、XML 要素の中身でなく、実際の XML 要素自体を出力しているからです。XML 要素の中身を取得するには、text()関数を使用します。

```
function ParseBooks(bookInput:XML):void {
```

```
trace("XML Output");
trace("-----");
trace(bookInput.Book.author.text());
}
```

アプリケーションをプレビューすると、今度は<author>タグ名で囲まれていない著者名が表示されます。残念ながらその名前はその間に空白のない1行で表示されます。

```
XML Output
-----
Sir Arthur Conan DoyleF. Scott FitzgeraldStephen E. AmbroseStephen E. Ambrose
```

これは小さな副作用で、簡単に変更することができます。修正する1つの方法はインデックス位置を使って、対象とする値を取得する方法です。たとえば Sir Arthur Conan Doyle を取得するには、次のようにインデックス位置 0 を渡します。

```
function ParseBooks(bookInput:XML):void {
    trace("XML Output");
    trace("-----");
    trace(bookInput.Book.author.text()[0]);
}
```

このコードを実行すると希望する Sir Arthur Conan Doyle が表示されます。おそらく多くの場合で、インデックス番号を手動で入力するのは便利ではないでしょう。XML データが持つノードや子ノードの数に関する高度な知識を持っていない場合は特にそうです。そのような場合には、このセクションで使った直接的な方法とは異なる、間接的で反復的な方法が必要になります。

次のページでは、この間接的な方法をいくつか見ていきます。

page 4

前のページでは、XML ファイルに保持したデータにアクセスする方法について学び始めました。データに直接アクセスする方法から学び始めましたが、その方法がうまくいくのは、使用する XML ファイルとノード名を知っているときだけです。XML データの構造がはっきりしない場合には、このページで取り上げる間接的な方法が役立ちます。

データへの間接的なアクセス

間接的な方法では、ループを使って子ノードを繰り返し処理し、必要な情報を抽出することが必要になります。著者の情報を出力したときには著書名は1行で受け取りました。われわれの目的はリストを繰り返し処理し、個別にその情報を表示する方法を見つけることです。手動でインデックス値を与えてその情報に直接にはアクセスしません。

そのためには XML 情報の集まりは常に XMLList として返され、XMLList は XML オブジェクトのみを含むことに注意しなければなりません。たとえば前のページで返された著書のリストは XMLList の形式で返されています。XMLList は、標準的な XML 操作のサポートより優れた、標準的なリスト操作の多くをサポートしています。

以下のコードは、ループを使って各著者情報を表示するために ParseBooks 関数内のコードです。

```
function ParseBooks(bookInput:XML):void {
    trace("XML Output");
    trace("-----");

    var authorList:XMLList = bookInput.Book.author;

    for each (var authorElement:XML in authorList) {
        trace(authorElement);
    }
}
```

ではこのコードについて詳しく見ていきましょう。

```
var authorList:XMLList = bookInput.Book.author;
```

まず XMLList 型の authorList という名前のオブジェクトを作成し、authorList を bookInput.Book.author が返す XMLList に初期化しています。

```
for each (var authorElement:XML in authorList) {
    trace(authorElement);
}
```

この行では、for-each ステートメントを使って authorList を繰り返し処理しています。これは for-each ステートメントなので、インデックス位置を気にする必要はありません。ここではただ現在のオブジェクトの名前とオブジェクトを

取得する集まりを指定しているだけです。後は舞台の裏側で処理されます。

authorElement 変数は XML 型であることに注意してください。authorElement が XML 型である理由は、すでに述べたように、XMLList(authorList の型)は XML オブジェクトを含むからです。

for-each ループでなく for ループを使いたい場合には、コードは次のようになります。

```
for (var i:int = 0; i < authorList.length(); i++) {  
    var authorElement:XML = authorList[i];  
    trace(authorElement);  
}
```

ここで注意が必要な唯一のことは、このループは、インデックスの変数が authorList 内の項目の合計数よりも大きくなると終了するということです。項目の合計数は XMLList オブジェクトである authorList の length()メソッドを呼び出すことで取得します。

全ての children()の呼び出し

求めるノードの名前がたまたま分からない場合でも、もっと汎用的な children()関数を使用することができます。この children 関数はノードの子すべてを、XMLList の形式で返します。XMLList を取得したら、選択方法にかかわらずデータを処理することができます。

たとえば以下は、ノードの子をループ処理することで情報にアクセスする方法を示すサンプルです。

```
function ParseBooks(bookInput:XML):void {  
    trace("XML Output");  
    trace("-----");  
  
    var bookChildren:XMLList = bookInput.Book.children();  
  
    for each (var bookInfo:XML in bookChildren) {  
        trace(bookInfo);  
    }  
}
```

著者情報だけが出力された前のサンプルと異なり、今度は Book ノードの子すべて、タイトルと著者が表示されま

す。

```
XML Output
```

```
-----
```

```
Sherlock Holmes: Complete Novels and Stories, Vol 1
```

```
Sir Arthur Conan Doyle
```

```
The Great Gatsby
```

```
F. Scott Fitzgerald
```

```
Undaunted Courage
```

```
Stephen E. Ambrose
```

```
Nothing Like It In the World
```

```
Stephen E. Ambrose
```

XMLオブジェクトである `bookInfo` の `name` プロパティを調べることで `author` をフィルタリングすることができます。
`name` プロパティはデータが保持されているノード名を返します。たとえば前述のコードの `trace(bookInfo)` を `trace(bookInfo.name())` に変更すると、以下の出力が得られます。

```
XML Output
```

```
-----
```

```
title
```

```
author
```

```
title
```

```
author
```

```
title
```

```
author
```

```
title
```

```
author
```

今表示されているのは前に見たデータのノード名です。この情報を使うとほしい情報をさらに選択的に拾い上げることができます。たとえば以下は、`children()` プロパティを使うことによって著者情報だけを表示する `ParseBooks` 関数の修正版です。

```
function ParseBooks(bookInput:XML):void {  
    trace("XML Output");  
    trace("-----");
```

```
var bookChildren:XMLList = bookInput.Book.children();

for each (var bookInfo:XML in bookChildren) {
    if (bookInfo.name()=="author") {
        trace(bookInfo);
    }
}
}
```

ここでは現在の XML ノードの名前を調べていることに注意してください。そのノード名が探している名前 (author)と一致したら、それに関係している情報を表示させています。

これまでノード内に保持されているネストされた情報を直接処理してきました。次のページでは属性情報を得る方法を学びます。

page 5

属性は、前ページで処理した子ノードとはまったく異なる、面白い小さな創造物です。このページではこれらを手なづける方法を学びます。

属性の読み取り

ここまでわれわれは主に、ノード/要素とネストされた情報の読み取りを処理してきました。属性はノードそのもの上に直接保持された情報であるという点で異なります。したがってその情報にアクセスする方法も少し異なります。

XML ファイルでは、属性は Book ノード上に直接位置する ISBN 情報です。

```

<Books>
  <Book ISBN="0553212419">
    <title>Sherlock Holmes: Complete Novels...
    <author>Sir Arthur Conan Doyle</author>
  </Book>
  <Book ISBN="0743273567">
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
  </Book>
  <Book ISBN="0684826976">
    <title>Undaunted Courage</title>
    <author>Stephen E. Ambrose</author>
  </Book>
  <Book ISBN="0743203178">
    <title>Nothing Like It In the World</title>
    <author>Stephen E. Ambrose</author>
  </Book>
</Books>

```

その情報を抽出する AS コードを見てみましょう。

```

function ParseBooks(bookInput:XML):void {
  trace("XML Output");
  trace("-----");

  var bookAttributes:XMLList = bookInput.Book.attributes();

  for each (var bookISBN:XML in bookAttributes) {
    trace(bookISBN);
  }
}

```

前の ParseBooks 関数を上記関数で上書きしてアプリケーションをプレビューすると、ISBN 情報が表示されます。ではこの動作を行っている1行のコードを見てみましょう。

```

var bookAttributes:XMLList = bookInput.Book.attributes();

```

属性のリストにアクセスするには、Book ノード上で attributes()メソッドを呼び出します。属性情報は、可能な場合、みなさんの想像通り、XMLList の形式で返されます。そのリストを前と同じように繰り返し処理し、その情報を出力させています。

この方法は、前の XML の children()を使った方法と同様、一致するすべてのデータを出力します。その場合、

すべての属性値が出力されます。われわれは1つの Book ノードにつき1つの属性だけを設定しているので問題はないのですが、複数の属性がある場合には、それらがすべて出力されることになります。それらを追跡する方法を2つ見てみましょう。

1つめの方法は、前に使った方法と似ています。属性の名前を調べてそれが探しているものと一致するかどうかを調べるのです。

```
function ParseBooks(bookInput:XML):void {
    trace("XML Output");
    trace("-----");

    var bookAttributes:XMLList = bookInput.Book.attributes();

    for each (var bookISBN:XML in bookAttributes) {
        if(bookISBN.name()=="ISBN"){
            trace(bookISBN)
        }
    }
}
```

これは、name()メソッドがノード名を返すことで動作します。name()メソッドは子、属性、親などの区別を行いません。name()メソッドはただその XML オブジェクトの名前が何であるかだけに気をつけます。したがって name() オブジェクトの、属性名へアクセスするための使用方法とノード名へアクセスするための使用方法が似ているのは意図的なのです。

2つめの方法は、属性の名前にもとづいて XMLList 自体を作成し、フィルタリングする方法です。

```
function ParseBooks(bookInput:XML):void {
    trace("XML Output");
    trace("-----");

    var bookAttributes:XMLList = bookInput.Book.attribute("ISBN");

    for each (var bookISBN:XML in bookAttributes) {
        trace(bookISBN);
    }
}
```

```
}
```

この方法では、attribute()メソッドを使い、探している属性の名前を渡します。

```
var bookAttributes:XMLList = bookInput.Book.attribute("ISBN");
```

XMLList オブジェクトの bookAttributes は、ISBN の名前に一致する属性を含む XML オブジェクトのリストを含んでいます。この方法を使うと、もっと大きな XML オブジェクトのリストを繰り返し処理して手動で各名前を調べなくて済みます。

page 6

前のページでは XML ファイル内にある属性にアクセスする何通りかの方法を学びました。このページでは、データベースに照会(クエリ)するとき SQL や関連するステートメントを記述する方にはおなじみの、E4X でのフィルタリングの仕組みを見ていきます。

値のフィルタリング

AS 3 のもう1つの新機能は、みなさんが関心のあるデータだけをフィルタリングし表示できる機能です。これまでのページのたくさんのサンプルの中で、XML オブジェクトを走査し、名前が探している値と一致するかどうかを調べようとしました。じつはこれよりもよい方法があるのです。

ノード値のフィルタリング

XML データから Stephen E. Ambrose の本すべてを見つけないとします。各 XML オブジェクトの値を走査し、author の値が Stephen E. Ambrose に等しい場合に親の XML ノードを返す関数はこれまでも作成できました。AS 3 ではもっと簡単にパワフルな方法があります。以下は著者が Stephen E. Ambrose であるすべての書籍名を返すコードです。

```
function ParseBooks(bookInput:XML):void {  
    trace("XML Output");  
    trace("-----");  
  
    var authorList:XMLList = bookInput.Book.(author == "Stephen E. Ambrose");  
    trace(authorList);  
}
```

このコードを実行すると以下が出力パネルに表示されます。

XML Output

```
-----  
<Book ISBN="0684826976">  
  <title>Undaunted Courage</title>  
  <author>Stephen E. Ambrose</author>  
</Book>  
<Book ISBN="0743203178">  
  <title>Nothing Like It In the World</title>  
  <author>Stephen E. Ambrose</author>  
</Book>
```

返されたデータは、値が Stephen E. Ambrose である author という名前の XML ノードを含んだ実際の XML オブジェクトであることに注意してください。これは AS 3 に導入されたフィルタリングの仕組みによって可能になりました。

```
var authorList:XMLList = bookInput.Book.(author == "Stephen E. Ambrose");
```

bookInput.Book.author と指定し、XML オブジェクトのリストを返し、各オブジェクトで著者名 Stephen E. Ambrose を走査する代わりに、ただキーワードと比較演算子、検索したい値を与えるだけです。残りのことは舞台裏で面倒をみてくれます。後に残るのは指定したクワイテリア(基準)に一致する XML オブジェクトの集まりです。

探していた結果は見つかりましたが、その結果は XML オブジェクトの形式です。XML 情報のない、Stephen E. Ambrose が書いた実際の書籍のタイトルを表示するには、フィルタリングの命令につづけてただ title キーワードを追加するだけです。

```
var authorList:XMLList = bookInput.Book.(author == "Stephen E. Ambrose").title;
```

これが実行できる理由は、フィルタリング命令の結果もまた XMLList の形式で返されるからです。タイトルの検索など、XMLList 上で実行する操作はどれも、XMLList 内の保持された各 XML オブジェクトに適用されます。

属性情報のフィルタリング

属性情報にもとづくデータのフィルタリングは少しだけ異なります。ISBN がある値に一致する本のリストを返すには、使用するキーワードの先頭に@記号を追加します。

以下のコードを試してみてください。

```
function ParseBooks(bookInput:XML):void {
    trace("XML Output");
    trace("-----");

    var bookList:XMLList = bookInput.Book.@ISBN == "0743203178".title;
    trace(bookList);
}
```

この ParseBooks 関数を記述したアプリケーションを実行すると、*Nothing Like It In the World*が表示されます。この bookList 宣言の箇所を詳しく見てみましょう。

```
var bookList:XMLList = bookInput.Book.@ISBN == "0743203178".title;
trace(bookList);
```

ISBN=="..."と入力するのではなく、キーワードの前に@記号を置いて、そのキーワードを属性キーワードとして印づけします。@ISBN==" 0743203178"と入力することで、ISBN 属性がその数字に一致するノードが返されます。

page 7

前のページでは、設定した条件にもとづいてデータをフィルタリングする方法を学びました。しかし、前述したフィルタリングのサンプルではすべて、1つの条件にもとづいた値でフィルタリングしただけです。実際には条件の数はいくつでも設定できます。それによって巧みなフィルタを作成し必要なデータを正確に見つけることができるようになります。

たとえば次のコードを試してみてください。

```
function ParseBooks(bookInput:XML):void {
    trace("XML Output");
    trace("-----");
```

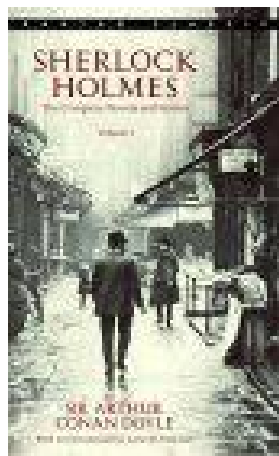
```
var bookList:XMLList = bookInput.Book.(author == "Stephen E. Ambrose" && title !=  
    "Nothing Like It In the World").title;  
trace(bookList);  
}
```

このコードを記述したアプリケーションを実行すると、*Undaunted Courage* だけがその結果として表示されます。ここでは著者とタイトルの両方でデータをフィルタリングしていることに注意してください。著者は Stephen E. Ambrose でなくてはならず、タイトルは *Nothing Like It In the World* であってはならないのです。

まとめ

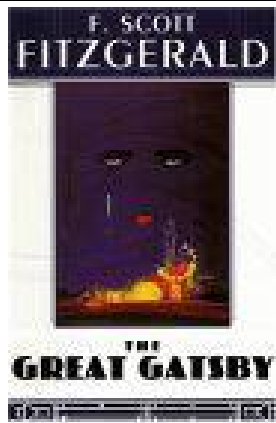
ここまでの6ページで、要求の厳しいXMLに関連する作業を処理するための十分なツールを提供できたことを願います。AS 3 への E4X の導入は非常に重要なことで、XML データの処理がこれまでよりも簡単になります。簡単ではありますが、データにアクセスするシンタックスは異なり(複雑にもなり)、このチュートリアルで取り上げたのは、E4X 下で可能なものの一部にすぎません。

このチュートリアルでは、わたしの4つの愛読書を含んだサンプルのXMLファイルを使用しました。以下にその表紙画像と、これらの本について学んだり購入もできる amazon へのリンクを用意しました。



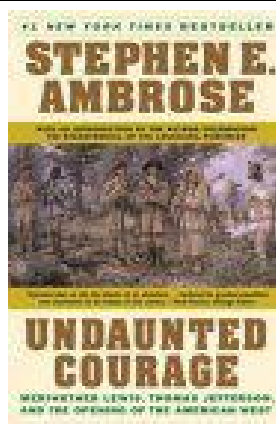
[Sherlock Holmes : The Complete Novels and Stories \(Bantam Classic\) Volume I](#)

by Sir Arthur Conan Doyle



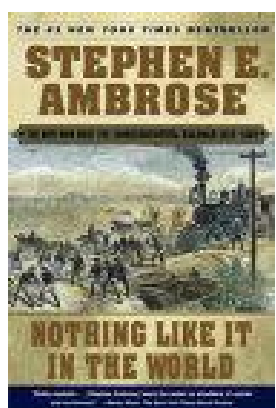
[The Great Gatsby](#)

by F. Scott Fitzgerald



[Undaunted Courage: Meriwether Lewis, Thomas Jefferson, and the Opening of the American West](#)

by Stephen E. Ambrose



[Nothing Like It In the World : The Men Who Built the Transcontinental Railroad 1863-1869](#)

by Stephen E. Ambrose