学士学位論文

AmazonのWeb Serviceを利用した商品検索システムの作成

平成16年度
茨城大学　工学部
システム工学科

執筆者：木本　俊
指導教官：新納　浩幸

目次

図目次

表目次

第1章

序章

1.1 はじめに

Amazonは書籍を筆頭に，音楽CDやDVD，玩具などを扱う世界最大の総合電子商取引サイトである．サイト内には，膨大な商品の中から求める商品をいち早く探し出すために検索フォームが設置してある．しかし，既存の検索システムでは検索条件の細かい指定ができない，表示件数の限定，広告の表示による処理の増大といった問題点がある．そこで本研究では，Amazonが提供しているWeb Serviceに注目し検索システムの改善を行う．

Web Serviceとは近年注目されている技術であり，Web上で提供されているサービスの中でもデータをXMLとして扱い，処理を提供元のサーバーが請け負うものを指す[1]．

AmazonがWeb Serviceを提供しているのはアフェリエイトへの支援が主な目的である．アフェリエイトとは自分の運営するホームページで商品の紹介や広告等を掲載し，それを見たユーザーが広告元のWebサイトに利益をもたらした場合，仲介料を得るというシステムである．Amazonのアフェリエイトシステムは商品広告を掲載し，その広告をクリックすることで商品購入ページへと導くものであり，掲載したい商品広告はシステムの利用者が作成することができる．その際に必要な商品のタイトルやイメージといった情報をWeb Serviceにより得られるのである．

ここではAmazonの提供するWeb Serviceを利用して検索を行い，XML文書として結果を得る．その得られたXML文書を解析及び変換することで，検索された商品を様々な条件で絞込み，ソートし表示する検索システムを作成する．

1.2 本論文の構成

2 章では Web Service の本質とそれを構築する 3 つの構成技術，Amazon の提供する Service の内容について述べる．

3 章で本研究の目的である既存の検索システムの問題点について述べ，どの様に改善するかを説明する．

4 章では結果として得られる XML から XSTL を用い,解析して出力する手順を説明する．

5 章では作成したシステムの構成，初期設定，使用方法などを説明する．

6 章では検索実験を述べ，今後の課題について述べる．

7 章では研究の結論を述べる．

第2章

Web Service

2.1 XML

XMLとは文書データの意味や構造を記述するためのマークアップ言語の一つである．マークアップ言語とは「タグ」と呼ばれる特定の文字列で文に構造を埋め込んでいく言語のことで，XMLはユーザが独自のタグを指定できることから，マークアップ言語を作成するためのメタ言語とも呼ばれる[2]．もともと，同じく独自のタグを指定可能なSGMLのサブセットとして考案され，任意のデータをHTMLと同様の感覚で送受信できることを目標に作成されたものである．XMLはその性質上，他のマークアップ言語の骨組みとして使用されることが多い．

XMLベースのマークアップ言語としては，リモート経由で他のコンピュータのサービスを呼び出すSOAPや，Web上でベクター画像の表現を行うSVGが有名である．XMLはコンピュータ同士でのデータの送受信に使用できるほか，Webブラウザで直接観覧することも想定されている．XMLをWebブラウザで快適に観覧するための仕様として，XML文書をWebブラウザで見た場合の表現を記述するXSLや，ハイパーリンク機能を実現するXLink/Xpointerなどが用意されている．

XMLやXMLベースのマークアップ言語の構造についてはSGMLやHTMLと同じく，スキーマ言語の一つであるDTDによって定義することになっている．しかし，DTDはSGMLでの使用を前提にして策定されたためにXMLとの親和性が低く，W3Cではこの欠点を解消したXML Schemaを策定中である．このほか，DTDに代わるものとしてRELAXという国産のスキーマ言語も提唱されている．ちなみに，HTMLをXMLの仕様内で書き直し，XMLパーサでの処理を可能にするなどの処理が行われたものがXHTMLである．

簡単なXML文書の例を図 2.1に示す．

```
<?xml version="1.0" encoding="UTF-8">・・・XML宣言
<商品>
  <管理情報 データ更新日="2005/3/2"・・・属性 />
  <商品情報>
    <商品画像 画像ファイル名="○○○.jpg" />
    <商品名>○○○</商品名>
    <ASIN>1234567890</ASIN>
    <発売日>2005/03/02</発売日>
    <価格>1000</価格>
  </商品情報>
</商品>
```

図 2.1:簡単な XML の例.

2.2 Web Service

Web Service という単語には主に次のような二つの意味がある.

1.Web 上で提供されているあらゆるサービス

2.インターネット上で, アプリケーションから利用できるサーバプログラム

前者は Web Service という言葉を聞いて一般の人が最初に思い浮かべると思われる意味で, 飛行機の空席照会から旅館の予約, 商品の購入にいたるまで, Web 上で提供されているあらゆる「サービス」を指す. 本研究で取り上げる Web Service は後者の意味で, ソフトウェアの機能を, ネットワークを通じて利用できるようにしたものである. もし必要としているプログラムの機能が Web Service として提供されているならば, ユーザーは自分でプログラムを作成する必要はなくなる. Web Service を提供しているサーバーに情報を送信し, 処理を請け負ってもらい結果だけを得るということが可能だからである.
また, 通常の情報サービスと明確に区別するために, 後者は「XML Web Service」と呼ばれる.

現在, 機能の記述や呼出し手順などの標準化が進行中であり, コンポーネント化された複数の Web Service 同士をつなぎ合わせて分散アプリケーションを構築するというスタイルが次世代のソフトウェア環境の主流になると予想されている. そうした環境が普及すると, 従来の OS やミドルウェアは Web Service を開発・実行する環境としての役割を担うようになり, サービス(およびそれらを組み合わせたアプリケーション)を利用するエンドユーザは, 現在の Web ブラウザを拡張したようなクライアントソフトを通して, すべてのソフトウェアを操作するようになると考えられる.

Web Service の基盤となっている技術には主に SOAP(Simple Object Access Protocol),

WSDL(Web Services Description Language)，UDDI(Universal Description Discovery, and Integration)の３つが挙げられる(図 2.2)．

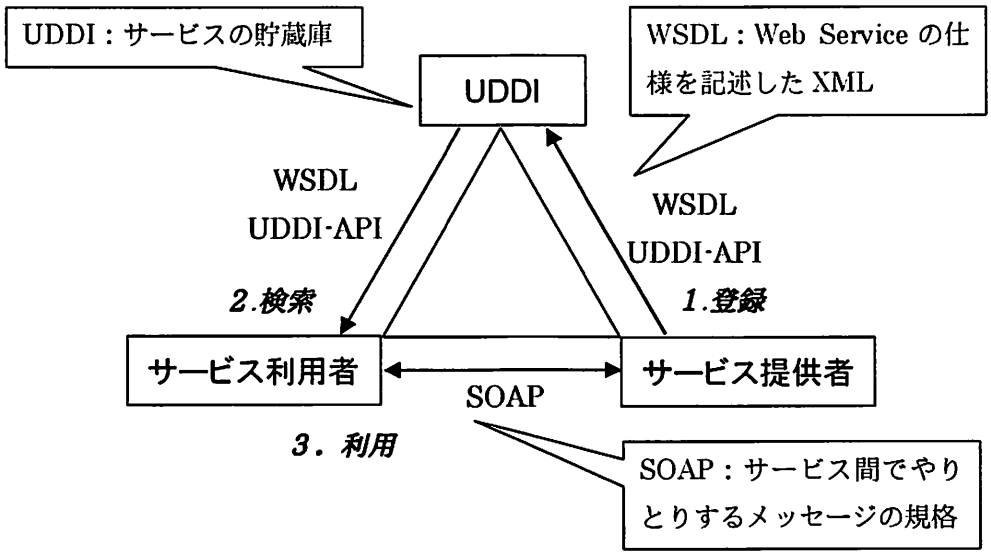


図 2.2:Web Service を構成する技術の関連図．

WebService においては必ずしもこの３つの要素が必要な訳でなく，UDDI と WSDL は無くてもサービスの利用は可能である．SOAP 環境が整っていれば，サービスへの接続時に SOAP で送受信される XML の構造を何らかの手段で交換することで，WSDL を利用する必要はなくなる．UDDI を利用しない場合は SOAP でやり取りされる XML の構造やバインディング情報，アクセスポイントを標準化された手段で交換可能である．UDDI と WSDL が両方とも提供されている場合は，WSDL へのリンクを UDDI に登録しておくことでインターフェース情報を自動取得可能である．

2.2.1 SOAP(Simple Object Access Protocol)

SOAP とは HTTP などを下位プロトコルとして使用し,XML をベースとしたメッセージを交換し，Web 上のオブジェクトへアクセスする通信プロトコルである．プロトコルを HTTP や SMTP，FTP などから選択できるため，企業間で利用する場合でもファイヤーウォールなどを安全に通過しメッセージ交換をおこなうことができる．

SOAP による通信では，XML 文書にエンベロープと呼ばれる付帯情報が付いたメッセージをやり取りする．以下が SOAP に用いる XML 文書の内容である(図 2.3)．

- エンベロープ構成要素

  何がメッセージの中にあるのか，誰がそれを処理すべきなのか，それは選択可能か必須かどちらなのかといったことを表現するための全体の枠組みを定義．
- エンコーディング規則

  SOAP で用いることのできるエンコーディングの規則，データ構造を XML で表現するための規則を定義．W3C で標準化されている XML Schema を元に定義されている．
- RPC 表現規則

  SOAP を用いて RPC を実現するための要求と応答の表現規則を定義．これにより複雑なデータ構造のメッセージ交換が可能．

図 2.3 : SOAP メッセージの構造．

SOAP エンベロープはメッセージを表現するＸＭＬ文書の最上位要素であり，要素名は Envelope である．Envelope 要素は"http://schemas.xmlsoap.org/soap/envelope/namespace" と関連付けられてなければならない．また，SOAP メッセージ中で使われる直列化の規則

を示すのに使う encodingStyle グローバル属性を記述することもできる．encodeingStyle 属性は直列化の規則または直列化された SOAP メッセージを元に戻す規則を識別する一つ以上の URI の順序つきリストで，URI"http://schemas.xmlsoap.org/soap/encoding/"によって識別される(図 2.4)．

```
<soap-env:Envelope
xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
soap-env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  ヘッダ，Bodyの記述
</soap-env:Envelope>
```

図 2.4:SOAP エンベロープの記述例．

SOAP ヘッダは通信の当事者間が事前知識無しでメッセージを拡張する柔軟なメカニズムを提供するために使用される．SOAP ヘッダは Envelope 要素の最初の子要素としてエンコードされ，要素名は Header である．SOAP ヘッダの属性は SOAP メッセージの受信者がメッセージをどのようにして処理するべきかを決定するためのものが規定されている(図 2.5)．

SOAP の actor グローバル属性はヘッダ項目要素の受信者を示すために使われ，属性の値は URI である．URI"http://schemas.xmlsoap.org/soap/actor/next"はそのヘッダ項目がヘッダメッセージを処理する一番最初のアプリケーション用に用意されたものであることを示す．mustUnderstand グローバル属性は受信者がヘッダ項目を必ず処理しなければならないのか，選択可能かということを示す．

```
<SOAP-ENV:Header>
<s:sample xmlns:s="some-URI" SOAP-ENV:mustUnderstand="1"
SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
  ヘッダ
</s:sample>
</SOAP-ENV:Header>
```

図 2.5:SOAP ヘッダの記述例．

SOAP ボディは SOAP Envelope 要素の子要素としてエンコードされ，要素名は Body である．Header 要素が存在する場合 Body 要素は Envelope 要素の最初の子要素でなければならない．また，SOAP メッセージの処理においてエラーが発生したときには，Body 要素

の中に Fault という要素を入れてレスポンスを送り返すことが決められている(図 2.6).

```
<SOAP-ENV:Enbelope
xmlns:SOAP-ENV="http://schemas.xmlsoap./envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  <SOAP-ENV:Header>
    <t:Transaction xmlns:t="some-URI" SOAP-ENV:mustUnderstand="1">
      4
    </t:Transaction>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DEF</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

図 2.6:SOAP 文書の記述例.

上記の様な XML 文書の受け渡しを行う SOAP であるが，この文書を生成するアプリケーションが言語ごとに存在する．XML の機種依存性がないという利点を生かすために様々な言語で SOAP メッセージを作成することができるような環境が用意されている．
今回は Apache Axis という JAVA で SOAP を扱うためのライブラリを用いた.この他に Perl の SOAP::Lite，C 言語の OpenSOAP，PHP の PEAR などが存在する．

サービスを利用するクライアントと，サービスを提供するサーバの双方が SOAP の生成・解釈エンジンを持つことで，異なる環境間でのオブジェクト呼び出しを可能にしている．なお，SOAP メッセージの生成エンジンは「SOAP プロキシ」，解釈エンジンは「SOAP リスナ」と呼ばれる．

2.2.2 WSDL (Web Services Description Language)

WSDL とは Web Service を記述するための，XML をベースとした言語仕様である．それぞれの Web Service がどの様な機能を持つのか，それを利用するためにはどの様な要求をすればいいのか,などを記述する方法が定義されている.WSDL を参照することで SOAP 文書をどの様に記述すればよいか，といったことが分かる．SOAP プロキシはこれを元にメッセージを生成している．

また，データや操作を定義する部分が通信プロトコルに関する部分から分離しているため，プロトコルやエンコード形式に関わりなくフォーマットを再利用できる，という特徴を持つ．

WSDL は主に次の 8 つの要素で構成されている(図 2.7).

1. (definitions)
   WSDL のルート要素で，Namespace に WSDL ファイルを識別する固有の namespace を指定する．
2. タイプ(type)
   交換されるメッセージを記述するために使用するデータ型の定義である．
3. メッセージ(message)
   伝送されるデータの抽象定義である．理論的なパートで構成され，そのそれぞれが何らかの型で，システムの定義に関連付けられる．
4. ポートタイプ(portType)
   抽象操作のセット．
5. オペレーション(operation)
   操作の抽象的な定義．それぞれの操作は入力メッセージと出力メッセージを参照する．
6. バインディング(binding)
   特定の portType によって定義された操作とメッセージの具体的なプロトコルとデータ形式を指定する．
7. サービス(service)
   関連する通信端点を集約するために使用される．
8. ポート(port)
   単一の通信端点のアドレス(Web Service を提供するサーバの URL)を定義．

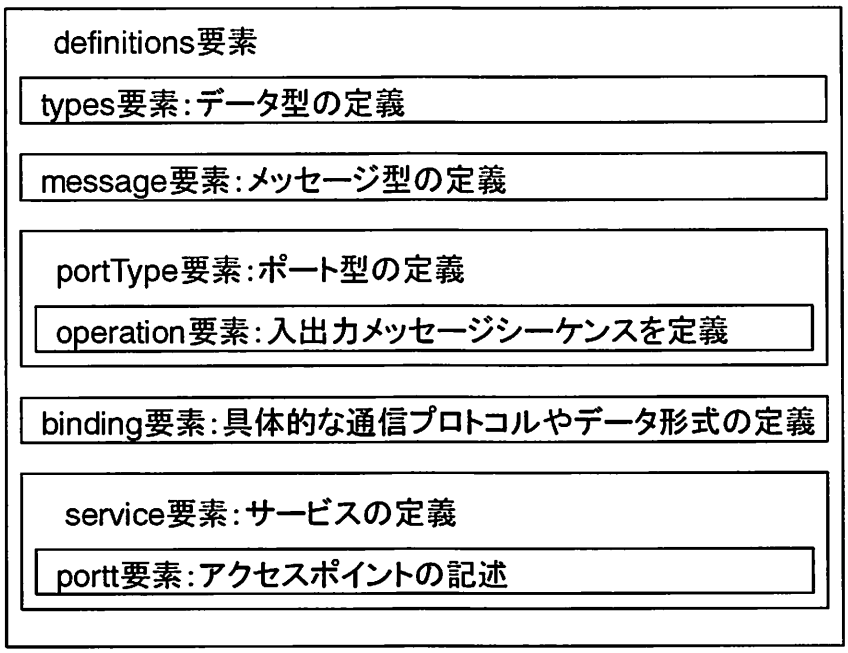


図 2.7:WSDL の構造.

2.2.3 UDDI(Universal Description, Discovery and Integration)

インターネット上で各社が提供している Web Service を集め，誰でも検索・紹介できるようにする Web Service を「UDDI」という．XML を応用した，インターネット上に存在する Web Service の検索・照会システムで，企業各社がインターネット上で提供している Web 技術を応用したサービスに関する情報を集積し，業種や名称，機能，対象，詳細な技術使用などで検索可能にする．登録・検索はともに無料である．

Web Service を提供する企業は，自社のサービスを UDDI レジストリと呼ばれるリストに登録することができる．UDDI に参加する Web Service は，SOAP と呼ばれる XML ベースのプロトコルによる通信に対応している必要がある．必要な時に必要なサービスを探し出してサービスを利用することが容易になるため，従来のような特定の得意先との固定的な取引を超えて電子商取引の活性化につながると期待されている．

UDDI では，登録情報は WSDL という XML ベースの言語で定義され，通信には XML ベースのプロトコルである SOAP を用いているため,通常の Web ブラウザなどの他,SOAP や XML に対応した様々なアプリケーションソフトから利用することができる．
Microsoft 社や Ariba 社 IBM 社などが中心となって推進しており，当面はディレクトリもこの三社によって管理される予定になっている．なお，このシステムの標準仕様を定める団体の名称も UDDI であるため，区別して団体を「UDDI プロジェクト」と呼ぶ．

2.3 Amazon の提供する Web Service

Amazon Web Service(以下 AWS)は，XML ベースのプロトコルを使い，Web 上で公開され，配布される自己完結型アプリケーションである[3]．AWS は，個別商品の一覧の情報の取り込みからショッピング・カートに追加するまでのサービスを，アプリケーションとして提供する．AWS には HTTP 経由の XML，または SOAP を介してアクセスできる．この 2 つのメソッドは，サーチのキーワードやブラウズツリーノードなどのパラメータに基づく Amazon.com，Amazon.co.uk，Amazon.co.jp，Amazon.de のトップセラー商品についての構造化データなどを返す(図 2.8)．

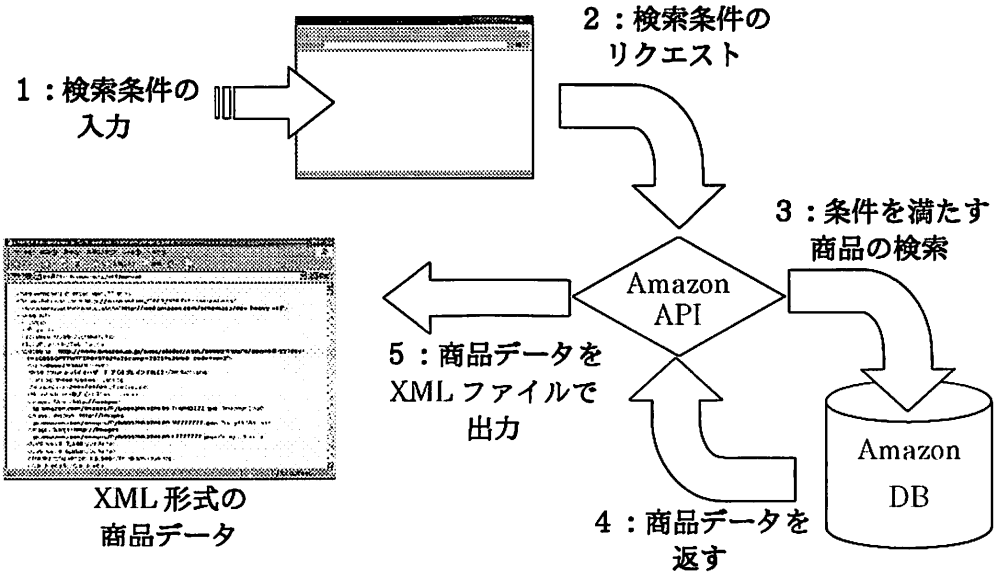


図 2.8:Amazon の Web Service を表す図．

現在のバージョンは 4.0 で正式名称を Amazon E-Commerce Service 4.0 という．本研究では AWS3.0 を利用した．AWS3.0 を利用するには指定のページでディベロッパー・トークンとアフェリエイト ID の登録，ツールキットのダウンロードが必要である．

Rest とは REpresentational State Transfer の略であり，リソースを特定する URL と，リソースにアクセス，操作するためのプロトコルとして，シンプルな HTTP を利用するというアーキテクチャである．URL に行いたい操作をパラメータとして付加することで，検索の条件などを指定することができる．

AWS の URL である"http://xml-jp.amazonxslt.com/onca/xml3?"に表 2.1 のパラメータを &で続けて指定し，ページにアクセスすることで任意の結果を得ることができる．

表 2.1: REST リクエストのパラメータ表.

t=	アソシエイト ID
dev-t=	ディベロッパートークン
type=	取得するデータの種類 簡易版： lite　　完全版：heary
page=	検索結果が複数ページにわたる場合のページ番号
mode=	検索する商品の種類 和書：books-jp　　洋書：books-us 音楽：music-jp　　音楽クラシック：classical-jp ビデオ：vhs-jp　　DVD：dvd-jp ソフトウェア：software-jp　　ゲーム：videogames-jp キッチン：kitchen-jp　　雑誌：magazines-jp エレクトロニクス：electronics-jp おもちゃ&ホビー：toys-jp
sort=	並び替えの順番 Featured Items / +pmrank 売り上げ順：+salesrank 評価順：+reviewrank 価格順(安→高)：+pricerank 価格順(高→安)：+inverse-pricerank 発売日日順：+daterank アルファベット順(A→Z)：+titlerank アルファベット順(Z→A)：-titlerank
f=	XSLT を使用する場合は XSLT の URI Xml をそのまま取得する場合は f=xml と記述
KeywordSearch=	KeywordSearch：キーワード検索 AsinSearch：ASIN 番号による検索 UpcSearch：universal price code による検索 AuthorSearch：著者名による検索 ArtistSearch：アーチスト名による検索 ActorSearch：俳優名による検索 DirectorSearch:監督名による検索 ManifacturerSearch：生産元による検索 PowerSearch：キーワード，著者等を組合せた検索（書籍のみ） BlendedSearch：複数の商品カテゴリに渡る検索 BrowseNodeSearch：ブラウズノードによる検索
local=	日本語のデータを扱うため jp を指定する

第3章

検索の問題点

3.1 Amazon 検索の現状

現在 Amazon のサイトのトップページでは，キーワードと商品の種類を指定して検索することができる(図 3.1)．全商品の中から検索した場合，検索結果が中央に表示され左側に商品の種類別に検索結果が何件含まれるかがリンクとして表示される(図 3.2)．ここからさらに種類別に商品を絞りたい時は，このリンクをたどることで種類別の検索結果を表示させることができる．種類別に検索結果を表示させると，中央に検索結果，左側に細かくカテゴリ分けされた商品の検索件数，右側にリストマニアを公開している人の商品が表示される(図 3.3)．リストマニアとは，お勧めの商品をリストにして，Amazon のサイト上で公開できるサービスである．例えば「癒されたい時に聞きたい CD リスト」や「アール・ヌーヴォーを知るための必読本，DVD」，「きっと読破できるはじめての洋書リスト」などのリストを作成することができる.作成した"リストマニア"リストには商品を 3〜25 点まで，各商品ごとにコメントを付けて掲示でき，サイト上にある全ての商品をリストに加えることができる．また，検索結果を「売れている順番」「価格の安い順番」「価格の高い順番」「タイトル名の順番：昇順」「タイトル名の順番：降順」「リリースの新しい順番」「リリースの古い順番」等で並び替えを行うことができる．これは検索した商品全てにおいて並び替えを行い先頭の 10 件を表示するものである．各商品の詳細は該当の商品名をクリックすることで，詳細ページに移動する(図 3.4)．詳細ページでは，中央に商品の詳細情報，左側に検索欄，その商品の関連情報，右側にショッピングカートへ商品を追加する項目，マーケットプレイス情報(ユーズド商品の情報)などが表示される．詳細情報には，タイトル，価格，発売元の他，著者からの内容紹介，CD に収録されている曲のタイトル，カスタマーのレビュー等がある．

図 3.1:Amazon.co.jp のトップページ.

図 3.2:Amazon の全件検索結果.

図 3.3:Amazon でのジャンル指定検索結果.

図 3.4:商品の詳細表示.

3.2 検索の問題点と改善法

図 3.2，図 3.3 を見ると分かるように Amazon のトップページで検索を行うと，お勧めの商品や，関連商品などの余分な情報が付加して表示され，検索後にソートを行うという一手間多い動作が必要である．また，10 件以上の表示を行えない(全件検索の場合のみ 15 件表示)ため，検索結果が後方にある場合，何度もページをめくるといった作業が必要になってくる．これらの点を改善するため，検索条件の指定を一括で行えるようにし，かつ表示件数を 10 件以上表示できるようにする．入力インタフェースは JAVA の GUI を用いて作成し，検索条件の入力を行えるようにする．表示件数の増加においては 1 ページ 10 件ずつしか取得できないため，必要な件数だけ繰り返し検索を行い，ファイルに出力するようなプログラムを作成した．さらに，検索結果のソートは「売れている順番」「価格の安い順番」「価格の高い順番」「タイトル名の順番：昇順」「タイトル名の順番：降順」「リリースの新しい順番」「リリースの古い順番」でしか行うことができない上に，商品の種類によってはこれらの並べ替えもできないものがある．そのため，少なくとも全ての商品に関しては API を用い上記のソートを行えるように改善し，他のソート条件を付加することで，より目的の商品を見つけやすくする．検索結果は XML ファイルとして出力し，XSLT と呼ばれるスタイルシート言語を使い HTML に変換しブラウザに出力する．AWS で標準にサポートされていないソートにおいては，XSLT でブラウザに出力する際に表示する順番を条件指定することができる機能により行う(図 3.5)．

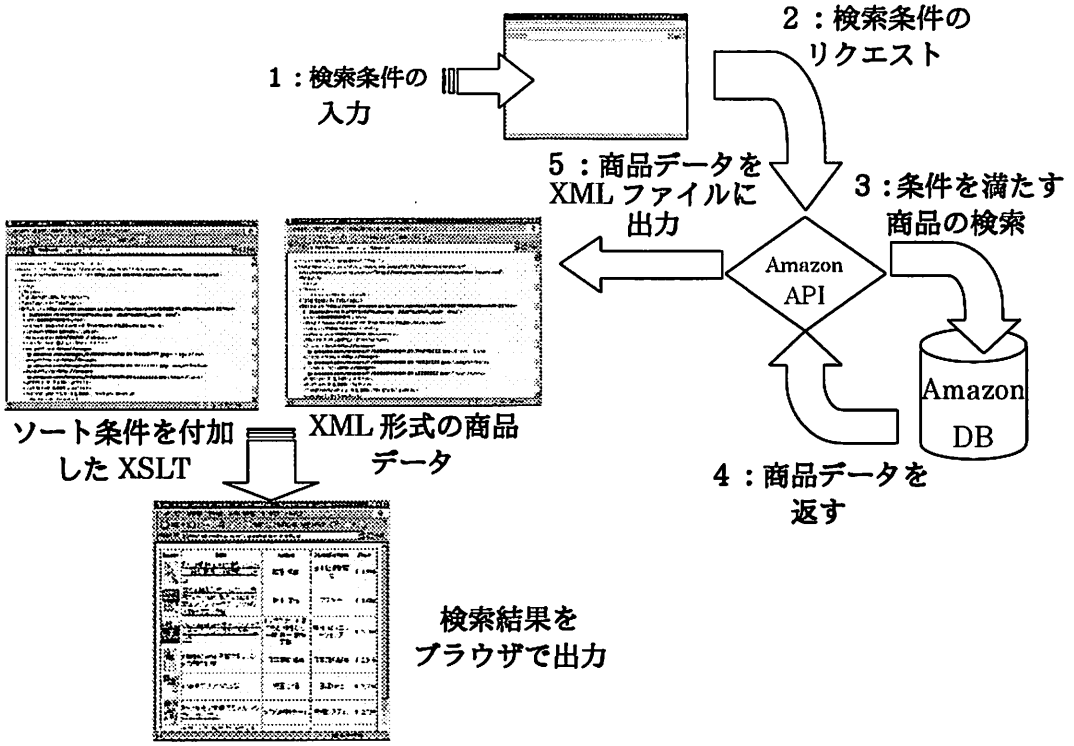


図 3.5:XSLT による出力．

第4章

フィルタリング

まず，検索結果であるXML文書のタグの内容を述べる．次に，その中でも必要な商品の基本情報とソートに必要な情報を述べ，その際に用いた手法について述べる．Amazonによる検索ではコメント数や，評価によるソートを全ての商品のジャンルで行えるわけではないので，フィルタをかけ全ジャンルでソートできるようにし，ブラウザ上に出力する．

4.1 検索結果のフォーマット

Amazon APIは検索結果をXML文書として返す．RESTによるリクエストでは以下の図4.1，図4.2，図4.3の様なXMLが結果として返される．各タグの詳細は図に示したとおりである．強調文字はデータのタイプをheavyに指定した際に追加される項目である．

```
<?xml version="1.0" encoding="UTF-8"?>
<ProductInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://xml.amazon.com/schemas3/dev-heavy.xsd">
  <Request>
    Requestした検索条件(図 4.2)
  </Request>
  <TotalResults> 検索された商品件数 </TotalResults>
  <TotalPages> トータルのページ数 </TotalPages>
  <Details>
    商品の詳細情報(図 4.3)
  </Details>
</ProductInfo>
```

図 4.1:検索結果として返ってくるXMLの主な構成．

```
<Request>
  <Args>
    <Arg value="アクセスしたブラウザの情報" name="UserAgent"></Arg>
    <Arg value="リクエスト ID" name="RequestID"></Arg>
    <Arg value="BrowseNode 番号" name="BrowseNodeSearch"></Arg>
    <Arg value="国番号" name="locale"></Arg>
    <Arg value="ページ" name="page"></Arg>
    <Arg value="ディベロッパー・トークン" name="dev-t"></Arg>
    <Arg value="アソシエイト ID" name="t"></Arg>
    <Arg value="XSL の URI" name="f"></Arg>
    <Arg value="検索する商品のジャンル" name="mode"></Arg>
    <Arg value="データタイプ" name="type"></Arg>
  </Args>
</Request>
```

図 4.2:Request タブ内の構造.

```
<Details url="Amazonへの商品リンク">
  <Asin> ASINコード </Asin>
  <ProductName> 商品名 </ProductName>
  <Catalog> カタログ名 </Catalog>
  <Artists>    (CatalogがMusicの時)
    <Artist> アーティスト名 </Artist>
  </Artists>
  <Authors>    (CatalogがBookの時)
    <Author> 著者名 </Author>
  </Authors>
  <Starring>    (CatalogがDVDの時)
    <Actor> 俳優・女優名 <Actor>
  </Starring>
  <Directors>    (CatalogがDVDの時)
    <Director> 監督名 </Director>
  </Directors>
  <ReleaseDate> 発売日 </ReleaseDate>
  <Manufacturer> 発売元 </Manufactrurer>
  <ImageUrlSmall> 小さい画像のURI </ImageUrlSmall>
  <ImageUrlMedium> 画像のURI </ImageUrlMedium>
  <ImageUrlLarge> 大きい画像のURI </ImageUrlLarge>
  <Availability> 在庫情報 </Availability>
  <ListPrice> 定価 </ListPrice>
  <OurPrice> 販売価格 </OurPrice>
  <UsedPrice> 中古価格 </UsedPrice>
  <SalesRank> ランキング </SalesRank>
  <Lists>
    <ListId> リストマニアID(この商品をリストマニアに登録している人のID) </ListId>
  </Lists>
  <BrowsList>
    <BrowseNode>
      <BrowseId> BrowseNode番号 </BrowseId>
      <BrowseName> 検索したBrowseNode以外にどこに登録されているか </BrowseName>
    </BrowseNode>
  </BrowseList>
  <Tracks>    (CatalogがMusicの時)
    <Track>収録曲</Track>
  </Tracks>
  <Media> メディア情報(CD-ROM, DVD, 大型本, コミックなど) </Media>
  <NumMedia> メディア枚数 </NumMedia>
  <ISBN> ISBNコード </ISBN> (CatalogがBookの時)
  <Features>
    <Feature> 商品の特徴 </Feature>
  </Featrues>
  <Platforms> プラットフォーム </Pratforms> (CatalogがSoftware or VideoGameの時)
  <Availability> 出荷情報 </Availability>
  <ProductDescription> メーカーやAmazonのレビュー </ProductDescription>
  <Reviews>
    <AvgCustomerRating> レビュー平均点 </AvgCustomerRating>
    <TotalCustomerReviews> 総レビュー数 </TotalCustomerReviews>
    <CustomerReview>
      <Rating> 点数 </Rating>
      <Summary> レビュータイトル </Summary>
      <Comment> レビュー内容 </Comment>
    </CustomerReview>
  </Reviews>
  <SimilarProducts>
    <Product> 関連商品(この商品を買った人が同時に買った物のASINコード) </Product>
  </SimilarProducts>
</Details>
```

図 4.3:検索結果として返ってくるXMLの構造

4.2 解析内容とフィルタリング手法

検索条件を入力する際に並び替えの順番を指定して検索を行い，返ってきた検索結果をXMLファイルとして出力する．このファイルには検索条件として指定された順番で，商品の情報が記述してある．Amazon でサポートされている条件以外で，ソートを行いたい場合はXML文書を解析しサービス利用者が独自に並び替えをしなければならない．本研究では，XSLT（XML Stylesheet Language Transformations）というXMLを他の文書構造に変換するための簡易言語を用いてソートを行った(図 4.4)．

XSLTはXML文書の構造を他の形式に変換するための変換ルールを記述するもので,「スタイルシート」と呼ばれる．もともとはXSL（Extensible Stylesheet Language）の一部として変換処理を行うために開発されたが，単独で使用することも可能である．おもに，XML文書からHTML文書やテキスト文書への変換などに使用される．

Amazon APIがサポートしている条件以外でソートを行う手順を示す．まず，XSLTを使いソートを行う前に，Amazon API が対応している検索条件（「売れている順番」「価格の安い順番」「価格の高い順番」「タイトル名の順番：昇順」「タイトル名の順番：降順」「リリースの新しい順番」「リリースの古い順番」）で検索を行う．すると，検索条件に一致する商品のリストがXML文書として返ってくる．このXMLを，XSLTを用いてブラウザ上にHTMLとして出力させる．この際にXSLTで特定のタグ内の数値や文字列等で，並び替えを行い出力するように指定する．これにより，あらかじめAmazonで用意してあるソート以外にも条件を指定してやることが可能である．ただし，結果として返ってくるXMLには商品情報は10件しか含まれていないため，その中でしか並び替えすることができない．

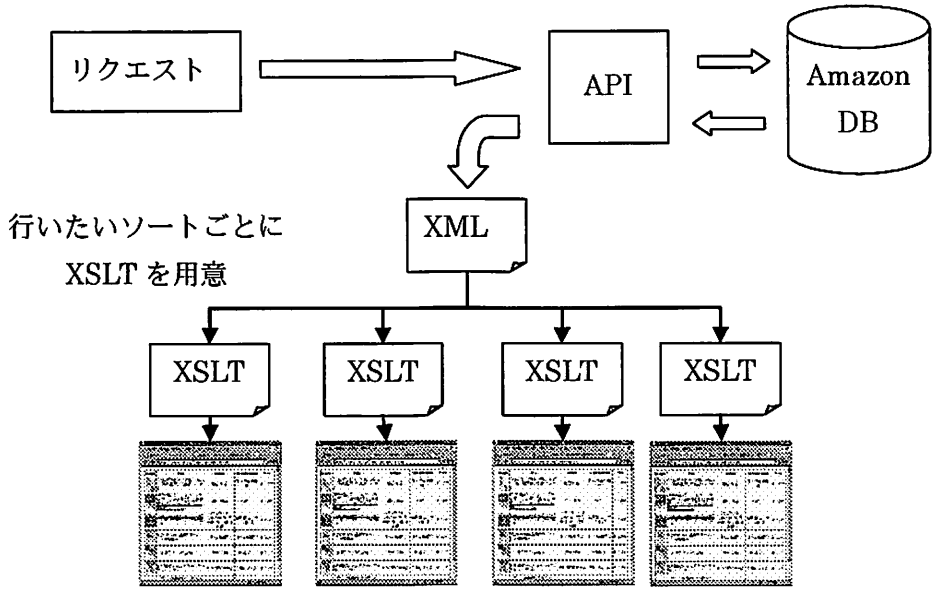


図 4.4:スタイルシートによるソートの流れ．

XSLT は主に図 4.5 の様な内容で構成されている．

（1） XML のバージョン指定

（2） XSLT 文書の定義

（3） 変換する文書の種類とエンコーディング

（4） 変換対象となる XML 文書内のノードを指定

（5） どの様な文書に変換するかを記述

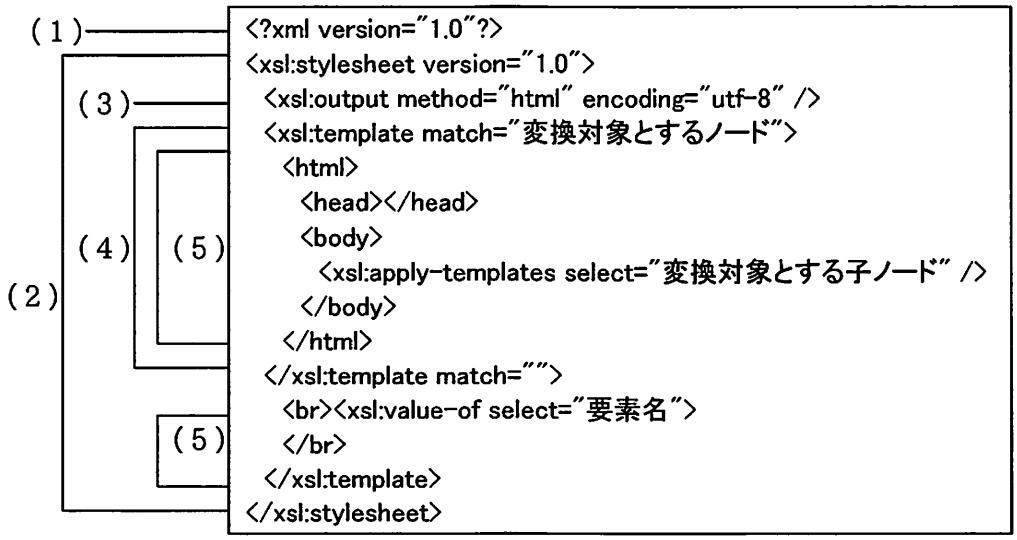


図 4.5:XSLT 文書の例．

XSLT を用いて XML を表示するには，表示させたい XML 文書に<? xml-stylesheet type="text/xsl" href="ファイル名" ?>というタグを記述すればよい．対象となる XML と XSLT が同一フォルダ内に無い場合は，絶対パスもしくは相対パス指定することで適応させることができる．

次に XSLT で主に使用する要素と属性について記述する．

<xsl:output>要素

XML や HTML 以外の形式に変換する場合に使用する要素．属性には主に以下のものがある(表 4.1 参照)．

- method=("xml"|"html"|"text"|QNAME)
  結果ツリーの出力形式を指定
- version=バージョン番号
  method 属性に"xml"か"html"を指定した場合に，該当するバージョン番号を指定．version を指定しなかった場合には，デフォルト値("xml"は"1.0"，"html"は"4.0")が設定される．

- encoding=文字エンコーディング名
  出力されるファイルの文字エンコーディング名を任意で指定する."UTF-8","UTF-16"が指定可能．XSLT プロセッサによっては"Shift_JIS"なども可能である．
- omit-xml-declaration=("yes"|"no")
  method 属性指定を"xml"に設定した場合有効な属性．"yes"の場合は，XML 宣言を出力しないことを示す．
- doctype-poblic=公開識別子
  DOCTYPE 宣言の公開識別子を指定する．(任意)
- doctype-system=SYSTEM 識別子
  DOCTYPE 宣言の SYSTEM 識別子を指定する．（任意）
- media-type=文字列
  出力されるファイルのメディアタイプを指定．指定しない場合は，デフォルト値が設定される．("xml"は"text/xml"，"html"は"text/html"，"text"は"text/plain")

表 4.1:output の使用例．

xml を出力	<xsl:output method="xml" encoding="UTF-8" doctype-system="sample.dtd">
html を出力	<xsl:output method="html" encoding="Shift_JIS" doctype-public="-//W3C//DTD THML 4.01 Transitional//EN" />
text を出力	<xsl:output method="text" encoding="Shift_JIS" />

<xsl:template>要素

<xsl:template>要素は match 属性で指定したノードに対して指定したテンプレートを適応することができる(**図** 4.6)．さらに子ノードに対するテンプレートを適応したい場合は<xsl:apply-templates>要素や<xsl:call-template>要素を用いてテンプレートを呼び出すことができる．変換対象となるノードが複数ある場合でも，特別な処理を行わずに表示することができる．これは，XSLT プロセッサが子ノードに対応できるテンプレートがあるかどうかを最後のノードまでチェックするためである(**図** 4.7)．

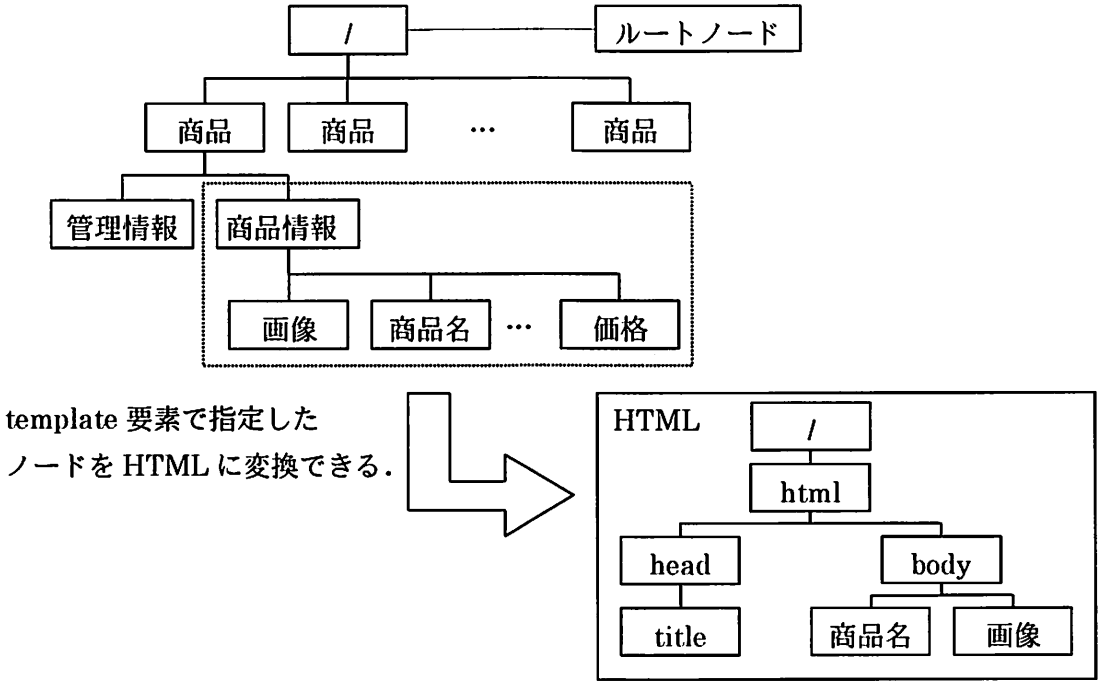
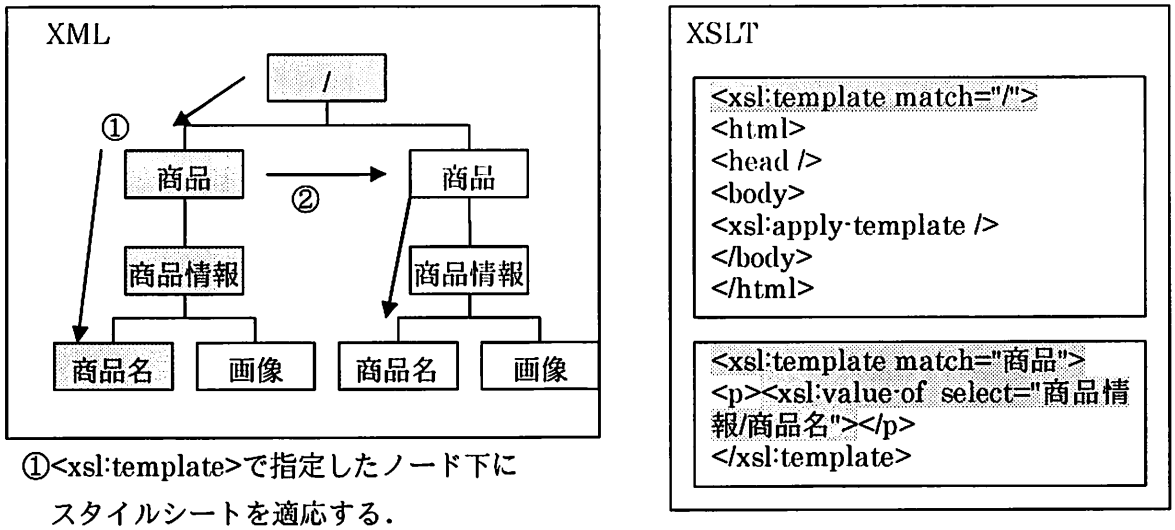


図 4.6:template 要素での変換.



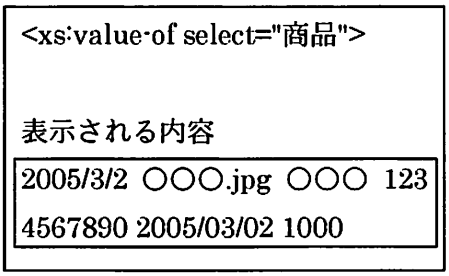
①<xsl:template>で指定したノード下に
スタイルシートを適応する.

②同じ階層のタグを参照し当てはまるものがあれば表示を行う.

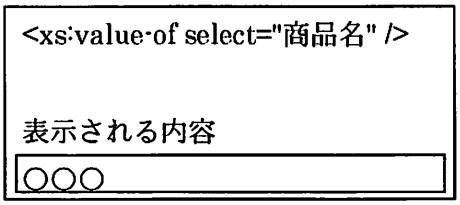
図 4.7:XSLT プロセッサが参照する (xsl:apply-templates)(xsl:call-template).

<xsl:value-of>要素

<xsl:value-of>要素を使うと，select 属性で指定した XML のタグ内の情報を表示することができる．<xsl:value-of select="表示したい要素ノード" />
図 2.1 の XML から商品名を表示するには<xsl:value-of select"商品名" />と記述する．value-of でタグを指定した場合その下にある全ての要素が表示される(図 4.8)．



指定したタグ内に含まれる全ての
タグの情報(属性値含む)を表示．



指定したタグの情報だけを表示．

図 4.8:商品タグを表示した場合と，商品名タグを表示させた場合．

<xsl:attribute>要素

XSLT で画像ファイルを表示するには図 4.9 のように記述する．(XML の image タグに画像の URl が記述されているものとする)．

```
<img>
  <xsl:attribute name="src">
    <xsl:value-of select="image">
  </xsl:attribute>
</img>
```

図 4.9:xsl:attirbute の記述例．

XSLT では属性値に要素を記述することができないため(<img src="<xsl:value-of select="image />">と記述できない)<xsl:attribute>要素を使用しなければならない．<xsl:attribute>要素は name 属性で指定した名前の属性を生成することができる．HTML の<a>タグの href 属性でリンクを指定する際も，同様に<xsl:attribute>要素で href 属性を追加する．

<xsl:if>要素

プログラム言語に登場する if 文と同じで条件が真の場合のみ実行される．条件が偽の場合に実行されるプログラミング言語の else に相当するものは無い．記述方法は図 4.10 の通

りである．

```
<xls:if test="条件判断の式">
  テンプレートの内容
</img>
```

図 4.10:xsl:if の記述例．

条件分岐で使用される演算子には以下の表 4.2 のものが挙げられる．XSLT においては「<」「>」はタグの開始・終了の記号と判断されるため一般実体を使用する必要がある．

表 4.2:XSLT で利用される比較演算子．

演算子の種類	記号	説明
比較演算子	=	等しい
	!=	等しくない
	>=(>=)	以下
	>(>)	より小さい
	<=(&glt;=)	以上
	<(<)	より大きい
数値演算子	+	加算
	-	減算
	*	乗算
	div	除算
	mod	余算
論理演算子	or	論理和
	and	論理積

<xsl:choose>要素

<xsl:if>要素と同じ働きをし，子要素に<xsl:when><xsl:otherwise>を持つ．<xsl:when>要素の test 属性で指定された条件判断式に当てはまる場合は<xsl:when>のタグ内のスタイルシートを適応し，条件に当てはまらない場合は，<xsl:otherwise>のタグ内のスタイルシートを適応する(図 4.11)．

```
<xsl:choose>
  <xsl:when test="条件判断の式">
    表示したいテンプレートルール
  </xsl:when>
  <xsl:otherwise>
    表示したいテンプレート
  </xsl:otherwise>
</xsl:choose>
```

図 4.11:xsl:choose の記述例．

<xsl:for-each>要素

指定したノードについて処理を繰り返すことを定義する．<xsl:template>要素の子要素と

して記述し，処理対象とするXMLデータのノードをselect属性で指定する(図 4.12).

```
<xsl:temptale select="テンプレートの処理ノードを指定する式">
  <xsl:for-each select="for-each で処理するノードを指定">
  テンプレートの内容
  </xsl:for-each>
</xsl:template>
```

図 4.12:xsl:each の記述例.

<xsl:sort>要素

for-each要素で指定したノードで繰り返し処理をする前に,選択したノードをソートすることができる．order属性で昇順（ascending），降順（descending）を指定できる．属性を省略すると昇順にソートされる．本研究ではこの要素を使いXMLのソートを行っている(図 4.13).

```
<xsl:temptale select="テンプレートの処理ノードを指定する式">
  <xsl:for-each select="for-each で処理するノードを指定">
    <xsl:sort select="ソートキーを示す式"
 order="ascending | descending" />
  テンプレートの内容
  </xsl:for-each>
</xsl:template>
```

図 4.13:xsl:sort の記述例.

XPathはXML文書の特定の部分を指し示す構文を定義する.XPathを利用すれば,XML文章中にHTML文書のようにアンカーなどが埋め込まれていなくとも，文書中の任意の位置を指し示すことができる．また，XPath は XSLT のほかに XPointer(XML Pointer Language:XML 文書中の内部構造に対してアドレス付けを行うための標準書式)，XLink(XML Linking Language:他のXMLによって作られた言語と併用して，ハイパーリンクを表現するためのリンク言語)，XQuery(XML Query:XML文書からデータを取り出す照会言語)においても用いられる．XPathの中でも代表的なものを以下の表 4.3 に示す.

表 4.3:XPath.

種類	関数	説明
ノードセット	count(ノードセット)	ノード数を返す
	id(識別名)	id が識別名の要素を返す
	name(ノードセット)	ローカル名を返す
	last()	カレントノードの位置を返す
	position()	文字列を結合した結果を返す
文字列	concat(文字列，文字列)	文字列の開始位置から文字数の文字列を返す
	substring(文字列，開始位置，文字数)	文字列 2 が文字列 1 に出現した後の文字列を返す
	substring-after(文字列 1，文字列 2)	文字列 2 が文字列 1 に出現する後の文字列を返す
	substring-before(文字列 1，文字列 2)	文字列 2 が文字列 1 に出現する前の文字列を返す
	substring-length(文字列)	文字列の文字数を返す
	translate(文字列 1，文字列 2，文字列 3)	文字列 1 にある文字列 2 を文字列 3 に置換する
ブール値	lang(言語コード)	xml:lang の値が言語コードと同じ場合は，真，違う場合は偽を返す
数値	numer(文字列)	文字列を数値に変換して返す
	round(数値)	数値に最も近い整数を返す
	sum(ノードセット)	ノードセットの和を返す

第5章

検索システム

5.1 環境設定

本研究では，Java(TM) 2 SDK, Standard Edition Version 1.4.0 を用いたシステム開発を行った．それに伴い Java の SOAP モジュールである Apache AxisVersion1.1 を使って SOAP によるリクエストを実現した[4][5]．以下に SOAP リクエストが行えるようになるまでの環境設定方法を記す．Java のプログラムは既にコンパイルできるものとする．

まず，Amazon サイト(http://amazon.co.jp/webservice)でアフェリエイトユーザ登録と Developer's Token を取得する．次に Apache Axis(http://ws.apache.org/axis/jp/)をダウンロード後，AXIS の lib フォルダ内にある jar ファイルを CLASSPATH へ通す(表 5.1)．

表 5.1:CLASSPATH へ通すファイル．

AXIS_HOME¥lib¥axis.jar
AXIS_HOME¥lib¥commons-discovery.jar
AXIS_HOME¥lib¥commons-logging.jar
AXIS_HOME¥lib¥jaxrpc.jar
AXIS_HOME¥lib¥lib¥log4j-1.2.8.jar
AXIS_HOME¥lib¥saaj.jar
AXIS_HOME¥lib¥wsdl4j.jar

Apache Axis の機能の一つである，WSDL2Java ツールを使い Amazon の WSDL から Java のスタブ/スケルトンコードを生成する．コマンドライン上で java org.apache.axis.wsdl.WSDL2Java -v -p com.amazon.soap.axis http://soap.amazon.com/schemas3/AmazonWebServices.wsdl と入力する．これにより生成された java のプログラムファイルをコンパイルできる．これにより生成された class ファイルをパッケージ化し，CLASSPATH へ通す．
以上の設定をすることで，Java のクラスとして SOAP によるリクエストを行える．

5.2 概要

本研究では，検索のタイプをキーワード検索に限定してシステムを作成した．キーワードサーチでも ASIN 検索や著者，出版社を指定しても検索結果を得ることができるため，検索システムに支障はない．以下の図 5.1 が作成したシステムの流れである．

(a) 検索条件の入力

検索条件であるキーワード，検索商品の種類，ソート方法，表示件数，使用する xslt の指定を行う．

(b) 検索条件の送信

(a)で得られた検索条件をもとに，得たい情報を指定し，Amazon API へリクエストする．検索条件の送信は SOAP で行っているため，Java のクラスを呼び出すことで検索結果を得ることができる．

(c) 検索結果のファイルへの出力

得られた検索結果を XML ファイルへ出力する．この際に，不必要な商品情報を省くことで処理の軽量化を図る(図 5.2)．

(d) 結果をブラウザで表示

表示に用いる xslt を指定してブラウザで表示を行う．

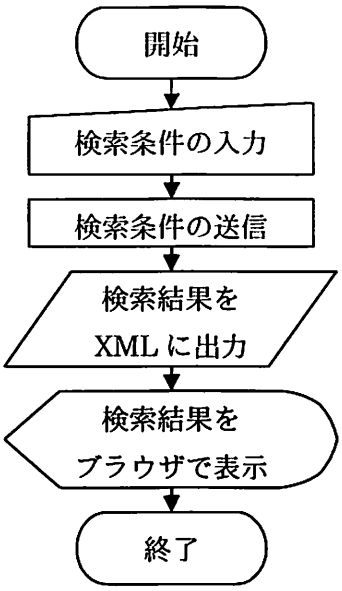


図 5.1:プログラムの概要フローチャート．

```
<?xml version="1.0" encoding="UTF-8"?>
<ProductInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://xml.amazon.com/schemas3/dev-heavy.xsd">
  <Request>
    Request した検索条件(図. 番号)
  </Request>
  <TotalResults> 検索された商品件数 </TotalResults>
  <TotalPages> トータルのページ数 </TotalPages>
  <Details url="Amazon への商品リンク">
    <Asin> ASIN コード </Asin>
    <ProductName> 商品名 </ProductName>
    <Catalog> カタログ名 </Catalog>
    <Artists>     (Catalog が Music の時)
      <Artist> アーティスト名 </Artist>
    </Artists>
    <Authors>     (Catalog が Book の時)
      <Author> 著者名 </Author>
    </Authors>
    <Starring>     (Catalog が DVD の時)
      <Actor> 俳優・女優名 <Actor>
    </Starring>
    <Directors>     (Catalog が DVD の時)
      <Director> 監督名 </Director>
    </Directors>
    <ReleaseDate> 発売日 </ReleaseDate>
    <Manufacturer> 発売元 </Manufactrurer>
    <ImageUrlMedium> 画像の URI </ImageUrlMedium>
    <Availability> 在庫情報 </Availability>
    <ListPrice> 定価 </ListPrice>
    <OurPrice> 販売価格 </OurPrice>
    <UsedPrice> 中古価格 </UsedPrice>
    <SalesRank> ランキング </SalesRank>
    <Lists>
      <ListId> リストマニア ID(この商品をリストマニアに登録している人の ID) </ListId>
    </Lists>
    <Tracks>     (Catalog が Music の時)
      <Track>収録曲</Track>
    </Tracks>
    <Media> メディア情報(CD-ROM, DVD, 大型本, コミックなど) </Media>
    <NumMedia> メディア枚数 </NumMedia>
    <ISBN> ISBN コード </ISBN> (Catalog が Book の時)
    <Features>
      <Feature> 商品の特徴 </Feature>
    </Featrues>
    <Platforms> プラットフォーム </Pratforms> (Catalog が Software or VideoGame の時)
    <Availability> 出荷情報 </Availability>
    <ProductDescription> メーカーや Amazon のレビュー </ProductDescription>
    <Reviews>
      <AvgCustomerRating> レビュー平均点 </AvgCustomerRating>
      <TotalCustomerReviews> 総レビュー数 </TotalCustomerReviews>
    </Reviews>
  </Details>
</ProductInfo>
```

図 5.2:XML ファイルとして出力する検索結果.

5.3 パッケージ化

作成したプログラムの実行に必要なファイルを一つの jar ファイルにまとめる．jar ファイルを作成するにはコマンドライン上で以下のコマンドを入力し実行する．

jar cvfm [作成する jar ファイル名] [Manifest ファイル名] [jar ファイルに含めたいファイル]

Manifest ファイルは jar ファイルに含まれる特殊なファイルで，オプションを指定することで様々な機能を組み込むことができる．今回記述した内容は図 5.3 の通りである．

```
Manifest-Version:1.0
Main-class: AmazonSearch        ..... 最初に起動されるべきクラス
Created-By: 1.0.0 (Shun Kimoto) ..... アプリケーションのバージョン指定と作者名
```

図 5.3:Manifest.mf.

Mnifest ファイルで Main-clasas を指定すると，jar ファイルを実行するだけでプログラムが起動できる．

5.4 作成したアプリケーションの使用方法

AmazonSearch.jar ファイルを実行すると図 5.4 のような GUI が表示され，検索キーワードと，商品の種類，ソートの種類，一度に表示させる商品の数を選択する．

図 5.4:プログラムの実行画面．

ソートオプションについては指定できる方法が，商品の種類によって表 5.2 のように決まっているため，商品の種類によって項目が変更する．

表 5.2:AmazonAPI がサポートするソートタイプ.

表示する順番	商品の種類
Featured Items	DVD 以外の商品
売り上げ順	全ての商品
レビュの平均評価順	書籍のみ
発売日順	書籍，音楽
アーティスト名順	音楽のみ
値段順(昇順，降順)	書籍，ソフトウェア，ゲーム，キッチン
商品のアルファベット順(A-Z，Z-A)	書籍，音楽，エレクトロニクス，キッチン
発売元のアルファベット順(A-Z，Z-A)	キッチン

第6章

実験

実際にアプリケーションを使用して検索を行い，検索結果を Amazon サイト内で検索を行った場合と比較する．作成したプログラムソースを付録として付加する．

6.1 検索結果

作成したアプリケーションを使い，Amazon で検索を行う．検索キーワードには「ハリーポッター」「Mr.Children」「ホラー」を用いた．以下に検索結果を示す図 6.1，図 6.2，図 6.3)．

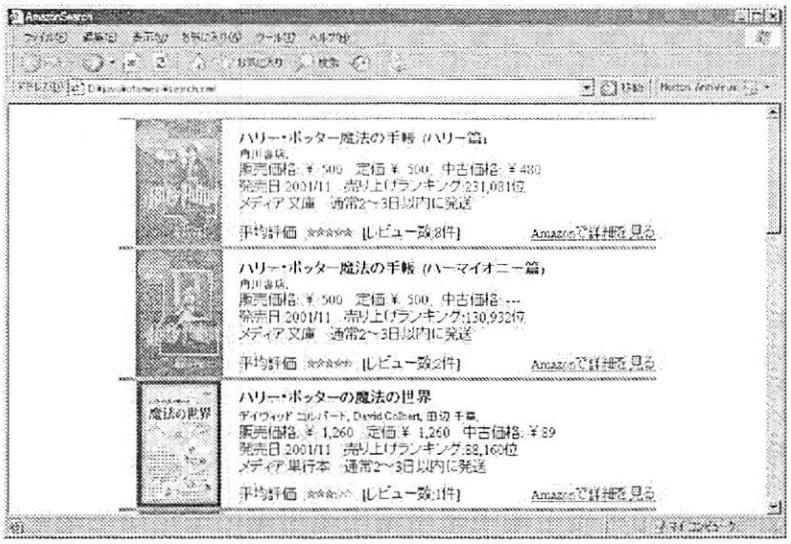


図 6.1:検索キーワード「ハリーポッター」，カテゴリ「和書」．

図 6.2:検索キーワード「Mr.Children」，カテゴリ「Music」．

図 6.3:検索キーワード「ホラー」，カテゴリ「DVD」．

AmazonAPI はデータベースに商品の画像が無い場合１×１ピクセルの画像を返すため，HTML で出力する際に TABLE の背景画像に NoImage 画像を設定した．

6.2 比較と評価

今回作成したプログラムとAmazonのサイト内で行った検索結果との比較を行う．
比較を行うにあたって，作成したプログラムは取得した検索結果をXMLに出力し，その中でのソートであるために,データベースの全ての商品においてソートを行っているAmazon内での検索とは結果が大幅に異なる可能性が高い．よって，検索件数が表示件数より多いものと，表示件数よりも検索件数が少ないものをそれぞれソートし，XSLTによるソートが有効に働いているかを検証する(**図 6.4 参照**)．

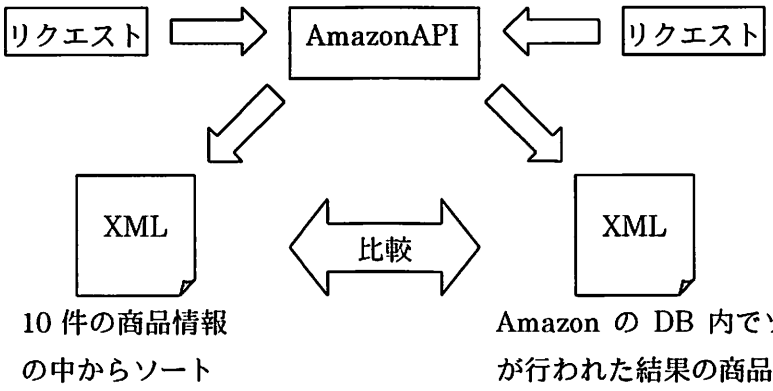


図 6.4:検索結果の比較．

まず，検索条件としてキーワードを「小説」，ソートタイプを「売れている順番」とし，検索を行う．これにより得られた結果を表 6.1 に示す．XSLT によるソート結果は，表の数値や，文字情報を元に並び替えるだけであるので明記していない．XSLTはこの結果内でタグによるソートを行っているため，これら以外の商品情報が出力されることはない．しかし，ソートタイプを「商品名」とした場合(**表 6.2**)と「発売日」とした場合(**表 6.3**)の 3 件を比較すると分かる通り，検索結果に表示されている商品には共通するものがほとんど無い．この結果により，XSLTによるソートは元となるデータの量に差がありすぎる場合には，有効に作用しないと言える．

次に検索条件を満たす商品数が，表示件数内に収まっている場合を考える．検索結果が35 件である検索キーワード「クッキング　かんたんレシピ」を指定し，検索を行った．得られた結果を表 6.4，表 6.5 に示す．検索キーワードに当てはまる全ての商品がXMLとして得られているためAmazonAPIによるソートとXSLTによるソートが一致する．そのため，Amazon 側でソートを行うよりもXSLTを用いて並び替えた方がより多くの条件で全ての商品データをソートできる．元となる商品データを多く得ることができるならば，XSLTによるソートを有効に利用できる．

WebServiceを用いた検索ではAmazonAPIを通してしかデータベースにアクセスすることができないため，XSLTによるソートを有効に利用することができない．元となるデータ

ベースの量が多ければ，XSLT でより多くの商品をソートすることができるが，出力する際に XSLT プロセッサが行う処理が増加し，検索にかかる時間が長くなる．よって，今回作成したシステムでは，商品情報があまり多くない場合は十分にソートし，商品の探索に役立っているが，商品情報が多すぎる場合は別の条件で検索を行ったほうが，目的の商品にたどり着きやすいと言える．

表 6.1:キーワード「小説」，ソートタイプ「売れている順」，表示件数「10 件」で検索した結果.

No.	製品名	著者	発売日	発売元	定価	販売価格	中古価格	Rank	List	評価	レビュー数
1	対岸の彼女	角田 光代,	2004/11/9	文藝春秋	¥1,680	¥1,680	¥1,320	45	2	4.55	22
2	今週、妻が浮気します	GoAhead & Co.,	2005/1/25	中央公論新社	¥1,050	null	null	365	2	3.27	11
3	ダ・ヴィンチ・コード（上）	ダン・ブラウン,越前 敏弥,	2004/5/31	角川書店	¥1,890	¥1,890	null	308	3	3.96	155
4	ダ・ヴィンチ・コード（下）	ダン・ブラウン,越前 敏弥,	2004/5/31	角川書店	¥1,890	¥1,890	null	139	3	3.96	155
5	タイタンの妖女	カート・ヴォネガット・ジュニア, 浅倉 久志,	2000/00	早川書房	¥672	¥672	null	388	3	4.5	8
6	電車男	中野 独人,	2004/10/22	新潮社	¥1,365	¥1,365	¥840	131	3	3.48	302
7	くすり指は沈黙する —その指だけが知っている〈3〉	神奈木 智,小田切 ほたる,	2005/1/27	徳間書店	¥600	¥600	null	148	0	0	0
8	魔法使いハウルと火の悪魔 —ハウルの動く城〈1〉	ダイアナ・ウィン ジョーンズ, Diana Wynne Jones, 西村 醇子,	1997/5/1	徳間書店	¥1,680	¥1,680	¥1,300	422	3	4.37	131
9	日暮らし 上	宮部 みゆき,宮部 みゆき,	2004/12/22	講談社	¥1,680	¥1,680	¥1,250	92	3	4.75	8
10	終戦のローレライ (1)	福井 晴敏,	2005/1/1	講談社	¥490	¥490	¥300	137	0	5	4

表 6.2:キーワード「小説」，ソートタイプ「商品名」，表示件数「10 件」で検索した結果.

No.	製品名	著者	発売日	発売元	定価	販売価格	中古価格	Rank	List	評価	レビュー数
1	ディズニーアニメ小説版 朝の読書・学級文庫おすすめセット 小学中級~中学生向 全 30 冊		2003/11/1	偕成社	¥22,050	¥22,050	null	null	0	0	0
2	小説ルパン三世	大沢在昌他,	2005/2/9	双葉社	¥800	¥800	null	122,863	0	0	0
3	力をつける現代文 15 小説編		1999/1/20	数研出版	¥370	null	null	1,341,250	0	0	0
4	Voie pour le roman futur, Une —未来の小説への道	平岡 篤頼, A. Robbe-Grillet,	null	第三書房	¥918	null	null	null	0	0	0
5	Complete Novels of Robert Penn Warren, The (10 Vols.) —ロバート・ペン・ウォレン長編小説全集(英文)	中村 紘一, R.P. ウォレン,	null	臨川書店	¥121,800	null	null	null	0	0	0
6	Poe`me et romans—「詩と小説」珠玉集	竹園 了元, 店村 新次, 原 政夫,	null	第三書房	¥420	null	null	null	0	0	0
7	Roman et la poe´sie, Le—小説と詩	松崎 芳隆,M. Butor,	null	第三書房	¥630	null	null	null	0	0	0
8	アナトール・フランス小説集 12 冊セット		2001/1/1	白水社	¥32,130	null	null	793,746	0	0	0
9	1 ポンドの悲しみ	石田 衣良,	2004/3/6	集英社	¥1,575	¥1,575	¥800	15,226	3	4.1	10
10	超大国の座	ラルフ ピーターズ, 青木 栄一,	1991/12/1	二見書房	¥1,835	null	¥200	null	0	0	0

表 6.3:キーワード「小説」，ソートタイプ「発売日順」，表示件数「10 件」で検索した結果．

No.	製品名	著者	発売日	発売元	定価	販売価格	中古価格	Rank	List	評価	レビュー数
1	お金がたりないっ[新装版]	篠崎 一夜, 香坂 透,	2005/02/29	幻冬舎	¥898	¥898	null	2,885	0	0	0
2	侵蝕～背徳に濡れる華～	高崎 ともや, 亜樹良 のりかず,	2005/02/29	幻冬舎	¥898	¥898	null	14,388	0	0	0
3	1/7 の恋人	火崎 勇, DUO BRAND.,	2005/02/29	幻冬舎	¥898	¥898	null	19,216	0	0	0
4	合鍵	結城 一美, 桜城 やや,	2005/2/28	プランタン出版	¥560	¥560	null	309,181	0	0	0
5	小説宝石 03 月号 [雑誌]		2005/2/22	光文社	¥900	¥900	null	null	0	0	0
6	小説 b-BOY 03 月号 [雑誌]		2005/2/14	ビブロス	¥680	¥680	null	null	0	0	0
7	小説ルパン三世	大沢在昌他,	2005/2/9	双葉社	¥800	¥800	null	122,863	0	0	0
8	ユージニア	恩田 陸,	2005/2/3	角川書店	¥1,785	¥1,785	null	4,590	0	0	0
9	明智光秀 なぜ「本能寺」に向かったか	嶋津 義忠,	2005/2/2	PHP 研究所	¥760	¥760	null	null	0	0	0
10	愛が理由	矢口 敦子,	2005/2/1	角川春樹事務所	¥1,890	¥1,890	null	null	0	0	0

表 6.4:キーワード「クッキング　かんかんレシピ」，ソートタイプ「売れている順」，表示件数「50 件」で検索した結果（前半）．

No.	商品名	著者	発売日	発売元	定価	販売価格	中古価格	Rank	List	評価	レビュー数
1	かんたん、おいしい! マクロビオティックはじめてレシピ	中島 デコ,	2003/12/25	近代映画社	¥1,680	¥1,680	null	4,960	3	4.62	8
2	ベイクドチーズケーキ&レアチーズケーキ —クリームチーズ使い切りの,かんたんレシピ	石橋 かおり,	2002/3/1	雄鶏社	¥1,260	¥1,260	null	2,554	3	4.6	15
3	フランスふだんのおそうざい —かんたんレシピとワインとチーズ	大森 由紀子,	2004/12/1	柴田書店	¥1,890	¥1,890	null	13,600	0	0	0
4	フランス 地方のおそうざい —かんたんレシピと地方のワイン	大森 由紀子,	2001/8/1	柴田書店	¥1,995	¥1,995	¥1,295	6,627	3	4.6	5
5	おもてなし和菓子を手づくりで. —"かんたん・おいしい・美しい"和菓子レシピ 42	金塚 晴子,	2003/9/1	小学館	¥1,680	¥1,680	null	56,469	0	4	2
6	きょう・すぐ・レシピ〈3〉かんたん煮物	日本放送出版協会,	2003/11/1	日本放送出版協会	¥998	¥998	¥798	97,501	0	0	0
7	超かんたんレシピ冬 —家で、みんなで、あったかごはん 冬野菜おかず、なべ、おもてなしメニュー	主婦の友社,	2001/12/1	主婦の友社	¥683	null	¥300	776,577	0	0	0
8	超かんたんレシピ春 —旬の素材でらくうまっ!春ごはん	主婦の友社,	2002/3/1	主婦の友社	¥735	null	¥210	776,579	0	0	0
9	超かんたん秋レシピ —らくちん!まんぞく!充実の和食レシピ	主婦の友社,	2002/8/1	主婦の友社	¥735	null	¥330	432,156	0	0	0
10	材料 3 種、時間 15 分でこんなに新しい味ができます. —材料別 201 レシピ	主婦の友社,	2003/1/1	主婦の友社	¥900	¥900	¥525	162,963	0	0	0
11	超かんたん夏レシピ —パパッと材料 3 種、チャチャッと 10 分調理	主婦の友社,	2002/5/1	主婦の友社	¥735	null	¥210	null	0	0	0
12	コープクッキングかんたんレシピ —手早く、おいしく、わが家の味	生活協同組合コープこうべ,	2000/6/1	生活協同組合コープこうべ	¥840	¥840	null	546,355	0	0	0
13	かんたん手づくり食品 —いつもの道具でいつものキッチンスペースで ベターホームのおすすめレシピ	ベターホーム協会,	2001/12/1	ベターホーム出版局	¥1,260	¥1,260	¥1,000	170,597	0	5	1
14	材料 3 種、時間 20 分で主役おかずともう 1 品が完成です. —材料別 288 レシピ	主婦の友社,	2003/5/1	主婦の友社	¥900	¥900	¥342	294,233	0	0	0
15	かんたん料理レシピ集—すぐに役立つハンディー判	池上 保子,	1993/11/1	永岡書店	¥897	¥897	¥200	null	0	5	1
16	マヤマヤの楽食的気持ち —たのしいエッセイ&かんたんレシピ	林 マヤ,	2000/6/1	近代映画社	¥1,365	¥1,365	¥390	353,544	0	3	1
17	野菜たっぷりかんたんレシピ—体にやさしい!	渡辺 香春子,	2003/3/1	新星出版社	¥1,260	¥1,260	¥1,200	788,051	0	0	0
18	超かんたんレシピ（2001 夏）	主婦の友社,	2001/8/1	主婦の友社	¥683	null	¥350	950,039	0	0	0
19	おいしい!かんたん!超初心者レシピ	江上 佳奈美,	1999/10/1	主婦と生活社	¥1,260	null	¥478	546,797	0	0	0

表 6.5:キーワード「クッキング　かんかんレシピ」，ソートタイプ「売れている順」，表示件数「50 件」で検索した結果（後半）．

No.	商品名	著者	発売日	発売元	定価	販売価格	中古価格	Rank	List	評価	レビュー数
20	〈ジェイくんの〉かんたん!感動!韓国料理 —あさ・ひる・よるとおやつのレシピ 61 品	金本 J.ノリツグ,	2004/1/1	=出版社	¥998	¥998	¥461	107,005	0	4.5	2
21	ザ・コンビニ料理 —ひとり暮らしの超かんたんレシピ	グレイル,	1998/4/1	宝島社	¥760	null	¥303	null	0	0	0
22	はじめてのお漬けもの —初心者でも作れるかんたんレシピ		2004/4/1	ブティック社	¥945	¥945	¥710	null	0	0	0
23	かんたん仕込みでおいしい料理ができた —しあわせの 150 レシピ	向後 千里,	2000/6/1	ワニブックス	¥1,470	¥1,470	¥630	698,483	0	0	0
24	超かんたんレシピ秋 —おいしいがギュッらくらく秋おかず	主婦の友社,	2001/9/1	主婦の友社	¥683	null	null	497,982	0	0	0
25	みんな大好き!基本のたまご料理 —おいしい、かんたん、「たまごレシピ」	竹村 章子, 山下 信子, ゆうエージェンシー,	2001/11/1	成美堂出版	¥840	null	null	679,810	0	0	0
26	かんたんレシピでつくるなつかしい味、あたらしい味 むかしのおやつ・いまのおやつ	伊藤 純子,中村 和子,	2000/11/1	ナツメ社	¥1,344	null	null	147,087	0	5	3
27	NHK ためしてガッテン 健康料理かんたんレシピ集〈4〉 秘伝!プロの味を出す料理のコツ	NHK 科学環境番組部,	2004/4/1	汐文社	¥1,890	¥1,890	null	616,574	0	0	0
28	すぐできる!!かんたん!!おつまみ 218 てん —お酒に合わせて、気分に合わせて、 簡単おつまみレシピ大集合!!		2003/7/1	ブティック社	¥1,260	null	null	444,013	0	0	0
29	かんたんおつまみ —飲みたいと思ったら、すぐにできるレシピ大集合!!		2001/6/1	ブティック社	¥1,050	null	null	559,342	0	5	1
30	NHK ためしてガッテン 健康料理かんたんレシピ集〈3〉 病気に強い健康料理	NHK 科学環境番組部,	2004/3/1	汐文社	¥1,890	¥1,890	null	808,496	0	0	0
31	中華まんじゅう—本場のレシピはおいしい具がいっぱい!	金 華,	2003/8/1	TOKIMEKI パブリッシング	¥998	null	null	190,356	0	0	0
32	かんたん!おいしい!かわいい!レシピ 60 —榎本加奈子の召し上がれ＝あ・た・く・し・の味	榎本 加奈子,	2003/5/1	ぴあ	¥1,260	null	null	180,830	0	0	0
33	NHK ためしてガッテン 健康料理かんたんレシピ集〈1〉 からだを作る健康料理	NHK 科学環境番組部,	2004/2/1	汐文社	¥1,890	¥1,890	null	995,535	0	0	0
34	超かんたん「おかずご飯」 —ひと目でわかるイラストレシピ付き		1996/11/1	日本出版社	¥764	¥764	null	null	0	0	0
35	NHK ためしてガッテン 健康料理かんたんレシピ集〈2〉 おなかにやさしい健康料理	NHK 科学環境番組部,	2004/3/1	汐文社	¥1,890	¥1,890	null	995,536	0	0	0

6.3 今後の課題

本研究では，検索のタイプを KeywordSearch に限定し検索を行っている．しかし，この他にも表 2.1 で示したように，様々な検索タイプが存在する．今後はこれらの検索タイプを選択できるようにしたい．また，サーバーへの負荷を抑えるためのアクセス制限は各ディベロッパー・トークンごとにかけられているため，複数のトークンを指定し並列的にアクセスすることガできれば，より短時間で多くの結果を得ることができると考えられ，XSLT によるソートの精度を上げることができる．

また，今回は AWS のみによるシステムの作成を行ったが，Google の Web Service との連動を行うことで Amazon のデータベース内にないイメージを取得したり，指定した商品に関連する記事の多さで並び替えを行う，といったことが可能になる．

第7章

結論

Amazon の Web Service は検索を主な目的として提供されているわけではないため，目当ての商品が分かっているならばサイト内の検索でも十分物足りる．一度に多くの結果を表示させたい時や，AmazonAPI が対応していない方法でソートを行いたい時には，作成したシステムで検索の手助けができる．Amazon の Web Service における検索機能は，サーバーへの負担を減らすためにアクセスは一秒間に 1 回，一日 24 時間に 10000 リクエスト以内と制限されている．そのため，検索を主に扱おうとすると逆に時間がかかってしまうことが多い．このことは他の Web Service においてもいえることで，全ての WebService にはアクセス過多によるサーバー側の処理の問題が発生する．この問題を解決するには，より多くの Web Service が登場して，別の似たようなサービスでも代用できるようになり，一つのサービスへのアクセスの集中を防ぐ必要がある．

第8章

謝辞

本研究の遂行及び論文の作成に多大な御助言及び指導を賜った新納 浩幸教官(茨城大学工学部システム工学科)，に深い感謝の意を表します．また，本研究を進めるにあたり助言，協力をいただきました岩崎 唯史教官(茨城大学工学部システム工学科)，同研究室の紺野 憲一(茨城大学大学院理工学研究科システム工学専攻)，藤井 丈明((茨城大学大学院理工学研究科システム工学専攻)，谷津 哲平(茨城大学大学院理工学研究科システム工学専攻)，正木 裕一(茨城大学大学院理工学研究科システム工学専攻)，大北 高広(茨城大学工学部システム工学科4回生)，藤村 元彦(茨城大学工学部システム工学科4回生)，茂木 啓悟(茨城大学工学部システム工学科4回生)，高橋 宏直(茨城大学工学部システム工学科4回生)に深く感謝致します．

参考文献

[1]グラハム・グラス(著),尾形隆彦,木村和之,石井真(訳)：“Web サービス入門 Java を使って覚える簡単 SOAP,WSDL,UDDI プログラミング”,ピアソン・エデュケーション,(2002).

[2]株式会社日本ユニテック(中山幹敏＋奥井康弘)：“改訂版　標準 XML 完全解説上”，技術評論社，(2001).

[3]ポール・ボシュ(著),篠原稔和(編),ウェブ・ユーザビリティ研究会(翻訳),“Amazon　Hacks 世界最大のショッピングサイト完全活用テクニック 100 選”，(2004)．

[4]丸の内とら:“10 日でおぼえる Java 入門 Java 2 SDK 対応”,翔泳社,p408-417,(2003).

[5]Joseph O'Neill(著),トップスタジオ(訳)，“独習 Java 第 2 版”，(2003).

付録 A

プログラムソース

amazonsoapsearch.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import com.amazon.soap.axis.*;
import java.io.*;

//JFrame クラスを継承して
public class amazonsoapsearch extends JFrame{
 static String a,b;
 int n;
//コンストラクタの定義
 public amazonsoapsearch(String title){

//super class のコンストラクタを呼び出す
   super(title);

//パネルの作成
   JPanel panel1 = new JPanel();
   JPanel panel2 = new JPanel();
   JPanel panel3 = new JPanel();
   JPanel panel4 = new JPanel();

//テキスト box の作成
   final JTextField text1 = new JTextField(20);
//ボタンの作成
   JButton button1 = new JButton("検索");
   JButton button2 = new JButton("クリア");

//コンポボックスの作成
```

```
    final Choice choice1 = new Choice();
    choice1.add("Featured");
    choice1.add("売り上げ順");
    choice1.add("レビュー数順");
    choice1.add("商品名順(A-Z)");
    choice1.add("商品名順(Z-A)");
    choice1.add("価格順(Low-High)");
    choice1.add("価格順(High-Low)");
    choice1.add("発売日順");
    final Choice choice2 = new Choice();
    choice2.add("10 件");
    choice2.add("50 件");
    choice2.add("100 件");

  //ラベルの作成
    JLabel label1 = new JLabel("KeyWord");
    JLabel label2 = new JLabel("Browse");
    JLabel label3 = new JLabel("sort");
    JLabel label4 = new JLabel("表示件数");

  //チェックボックスの作成
  //   final JCheckBox check0 = new JCheckBox("すべて",true);
    final JCheckBox check1 = new JCheckBox("和書",true);
    final JCheckBox check2 = new JCheckBox("洋書");
    final JCheckBox check3 = new JCheckBox("CD");
    final JCheckBox check4 = new JCheckBox("ｸﾗｼｯｸ CD");
    final JCheckBox check5 = new JCheckBox("DVD");
    final JCheckBox check6 = new JCheckBox("Video");
    final JCheckBox check7 = new JCheckBox("ｴﾚｸﾄﾛﾆｸｽ");
    final JCheckBox check8 = new JCheckBox("ｿﾌﾄｳｪｱ");
    final JCheckBox check9 = new JCheckBox("GAME");
    final JCheckBox check10 = new JCheckBox("ﾎｰﾑ&ｷｯﾁﾝ");

  //チェックボックスをグループ化
    final ButtonGroup type = new ButtonGroup();
  //   type.add(check0);
    type.add(check1);
    type.add(check2);
```

```
    type.add(check3);
    type.add(check4);
    type.add(check5);
    type.add(check6);
    type.add(check7);
    type.add(check8);
    type.add(check9);
    type.add(check10);

  //チェックボックス 0 が選ばれた時のイベント作成
  /*   check0.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
        choice1.removeAll();
        choice1.add("Featured");
        choice1.add("売り上げ順");
        choice1.add("レビュー数順");
        choice1.add("商品名順(A-Z)");
        choice1.add("商品名順(Z-A)");
        choice1.add("価格順(Low-High)");
        choice1.add("価格順(High-Low)");
        choice1.add("発売日順");
        a = "";
      }
    });
  */
  //チェックボックス 1 が選ばれた時のイベント作成
    check1.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
        choice1.removeAll();
        choice1.add("Featured");
        choice1.add("売り上げ順");
        choice1.add("レビュー数順");
        choice1.add("商品名順(A-Z)");
        choice1.add("商品名順(Z-A)");
        choice1.add("価格順(Low-High)");
        choice1.add("価格順(High-Low)");
        choice1.add("発売日順");
        a = "books-jp";
```

```
    }
  });
//チェックボックス 2 が選ばれた時のイベント作成
  check2.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent ae){
    choice1.removeAll();
    choice1.add("Featured");
    choice1.add("売り上げ順");
    choice1.add("レビュー数順");
    choice1.add("商品名順(A-Z)");
    choice1.add("商品名順(Z-A)");
    choice1.add("価格順(Low-High)");
    choice1.add("価格順(High-Low)");
    choice1.add("発売日順");
    a = "books-us";
   }
  });
//チェックボックス 3 が選ばれた時のイベント作成
  check3.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent ae){
    choice1.removeAll();
    choice1.add("Featured");
    choice1.add("売り上げ順");
    choice1.add("アーティスト名順");
    choice1.add("発売日順");
    choice1.add("商品名順(A-Z)");
    a = "music-jp";
   }
  });
//チェックボックス 4 が選ばれた時のイベント作成
  check4.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent ae){
    choice1.removeAll();
    choice1.add("Featured");
    choice1.add("売り上げ順");
    choice1.add("アーティスト名順");
    choice1.add("発売日順");
    choice1.add("商品名順(A-Z)");
```

```
        a = "classical-jp";
      }
    });
  //チェックボックス 5 が選ばれた時のイベント作成
    check5.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
        choice1.removeAll();
        choice1.add("売り上げ順");
        choice1.add("商品名順(A-Z)");
        a = "dvd-jp";
      }
    });
  //チェックボックス 6 が選ばれた時のイベント作成
    check6.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
        choice1.removeAll();
        choice1.add("Featured");
        choice1.add("売り上げ順");
        choice1.add("レビュー数順");
        choice1.add("商品名順(A-Z)");
        choice1.add("商品名順(Z-A)");
        choice1.add("価格順(Low-High)");
        choice1.add("価格順(High-Low)");
        choice1.add("発売日順");
        a = "vhs-jp";
      }
    });
  //チェックボックス 7 が選ばれた時のイベント作成
    check7.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
        choice1.removeAll();
        choice1.add("Featured");
        choice1.add("売り上げ順");
        choice1.add("レビュー数順");
        choice1.add("商品名順(A-Z)");
        a = "electronics-jp";
      }
    });
```

```
//チェックボックス 8 が選ばれた時のイベント作成
   check8.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ae){
     choice1.removeAll();
     choice1.add("Featured");
     choice1.add("売り上げ順");
     choice1.add("商品名順(A-Z)");
     choice1.add("価格順(Low-High)");
     choice1.add("価格順(High-Low)");
     a = "software-jp";
    }
   });
//チェックボックス 9 が選ばれた時のイベント作成
   check9.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ae){
     choice1.removeAll();
     choice1.add("Featured");
     choice1.add("売り上げ順");
     choice1.add("商品名順(A-Z)");
     choice1.add("価格順(Low-High)");
     choice1.add("価格順(High-Low)");
     a = "videogames-jp";
    }
   });
//チェックボックス 10 が選ばれた時のイベント作成
   check10.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ae){
     choice1.removeAll();
     choice1.add("Featured");
     choice1.add("売り上げ順");
     choice1.add("商品名順(A-Z)");
     choice1.add("商品名順(Z-A)");
     choice1.add("価格順(Low-High)");
     choice1.add("価格順(High-Low)");
     choice1.add("生産元順(A-Z)");
     choice1.add("生産元順(Z-A)");
     a = "kitchen-jp";
    }
```

```
  });
//ボタンが押された時のイベントの作成
  button1.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent ae){
//      System.out.println(text1.getText());
//      System.out.println(a);
     if(a == null || a == "" || a == "books-jp" || a == "books-us"){
      if(a == null) a = "";
      switch(choice1.getSelectedIndex()){
        case 0:
         b = "+pmrank";
         break;
        case 1:
         b = "+salesrank";
         break;
        case 2:
         b = "+reviewrank";
         break;
        case 3:
         b = "+titlerank";
         break;
        case 4:
         b = "-titlerank";
         break;
        case 5:
         b = "+pricerank";
         break;
        case 6:
         b = "+inverse-pricerank";
         break;
        case 7:
         b = "+daterank";
         break;
        default:
         break;
      }
     }
     if(a == "music-jp" || a == "classical-jp"){
```

```
 switch(choice1.getSelectedIndex()){
  case 0:
   b = "+psrank";
   break;
  case 1:
   b = "+salesrank";
   break;
  case 2:
   b = "+artistrank";
   break;
  case 3:
   b = "+orig-rel-date";
   break;
  case 4:
   b = "+titlerank";
   break;
  default:
   break;
 }
}
if(a == "dvd-jp"){
 switch(choice1.getSelectedIndex()){
  case 0:
   b = "+salesrank";
   break;
  case 1:
   b = "+titlerank";
   break;
  default:
   break;
 }
}
if(a == "vhs-jp"){
 switch(choice1.getSelectedIndex()){
  case 0:
   b = "+pmrank";
   break;
  case 1:
```

```
    b = "+salesrank";
    break;
   case 2:
    b = "+reviewrank";
    break;
   case 3:
    b = "+titlerank";
    break;
   case 4:
    b = "-titlerank";
    break;
   case 5:
    b = "+pricerank";
    break;
   case 6:
    b = "+inverse-pricerank";
    break;
   case 7:
    b = "+daterank";
    break;
   default:
    break;
  }
 }
 if(a == "electronics-jp"){
  switch(choice1.getSelectedIndex()){
   case 0:
    b = "+pmrank";
    break;
   case 1:
    b = "+salesrank";
    break;
   case 2:
    b = "+reviewrank";
    break;
   case 3:
    b = "+titlerank";
    break;
```

```
    default:
      break;
   }
  }
  if(a == "software-jp"){
   switch(choice1.getSelectedIndex()){
    case 0:
      b = "+pmrank";
      break;
    case 1:
      b = "+salesrank";
      break;
    case 2:
      b = "+titlerank";
      break;
    case 3:
      b = "+price";
      break;
    case 4:
      b = "-price";
      break;
    default:
      break;
   }
  }
  if(a == "videogames-jp"){
   switch(choice1.getSelectedIndex()){
    case 0:
      b = "+pmrank";
      break;
    case 1:
      b = "+salesrank";
      break;
    case 2:
      b = "+titlerank";
      break;
    case 3:
      b = "+pricerank";
```

```
    break;
   case 4:
    b = "+inverse-pricerank";
    break;
   default:
    break;
  }
 }
 if(a == "kitchen-jp"){
  switch(choice1.getSelectedIndex()){
   case 0:
    b = "+pmrank";
    break;
   case 1:
    b = "+salesrank";
    break;
   case 2:
    b = "+titlerank";
    break;
   case 3:
    b = "-titlerank";
    break;
   case 4:
    System.out.println("+price");
    b = "+price";
    break;
   case 5:
    b = "-price";
    break;
   case 6:
    b = "+manufactrank";
    break;
   case 7:
    b = "-manufactrank";
    break;
   default:
    break;
  }
```

```
        }
        switch(choice2.getSelectedIndex()){
         case 0:
          n = 10;
          break;
         case 1:
          n = 50;
          break;
         case 2:
          n = 100;
          break;
         default:
          break;
        }
//       System.out.println(b);
//       System.out.println(n);
        search callSearch = new search(text1.getText(),a,b,"1",n);
        callSearch.itemSearch();

        new output();

      }
    });

    button2.addActionListener(new ActionListener(){
     public void actionPerformed(ActionEvent ae){
      text1.setText("");//KeyWordBox の初期化
      check1.setSelected(true);//チェックボタンの初期化
      choice1.select(0);//コンポボックスの初期化
      choice2.select(0);
//実験成功      choice1.remove(0);
     }
    });
//レイアウトマネージャ
    panel1.setLayout(new FlowLayout(FlowLayout.CENTER));
    panel2.setLayout(new GridLayout(4,3));
    panel3.setLayout(new FlowLayout());
    panel4.setLayout(new FlowLayout(FlowLayout.RIGHT));
```

```
//panel1 への部品の配置
   panel1.add(label1);
   panel1.add(text1);
//panel2 への部品の配置
//   panel2.add(check0);
   panel2.add(check1);
   panel2.add(check2);
   panel2.add(check3);
   panel2.add(check4);
   panel2.add(check5);
   panel2.add(check6);
   panel2.add(check7);
   panel2.add(check8);
   panel2.add(check9);
   panel2.add(check10);
//panel3 への部品の配置
   panel3.add(label3);
   panel3.add(choice1);
   panel3.add(label4);
   panel3.add(choice2);
//panel4 への部品の配置
   panel4.add(button1);
   panel4.add(button2);

//コンテント・ペインの取得(パネルのセット)
   Container contena = this.getContentPane();
   contena.setLayout(new FlowLayout());
   contena.add(panel1);
   contena.add(panel2);
   contena.add(panel3);
   contena.add(panel4);

/*パネルへのセットはこのように記述しても可
   this.getContentPane().add(panel1,new FlowLayout());
   this.getContentPane().add(panel2,FlowLayout());
   this.getContentPane().add(panel3,FlowLayout());
   this.getContentPane().add(panel4,FlowLayout());
```

```
*/

//ウィンドウが閉じる際の処理
  this.addWindowListener(new WindowAdapter(){
   public void windowClosing(WindowEvent e){
    System.exit(0);
   }
  });

  try{
//Look&Feel の設定
   UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
   SwingUtilities.updateComponentTreeUI(this);

//エラー処理ブロック
   }catch (Exception e){
  }

//フレームのサイズを定義して表示
  this.setSize(300,240);
  //指定されたコンポーネントを基準にしてウィンドウの位置を指定 null の時は中央に表示
  this.setLocationRelativeTo(null);
  this.setVisible(true);
 }

//main メソッドの定義
 public static void main(String args[]){
  amazonsoapsearch myApp = new amazonsoapsearch("AmazonSoapSearch");
 }
}


////////////////////////////////////////////////////////////////////////////////

//search クラス
//引数が与えられるとそれを検索条件として Amazon にリクエストを行うクラス
class search{
 //フィールドの定義
```

```
String rKeyword;//検索キーワード
String rType;//商品の種類
String rSort; //ソートの種類
String rPage;//表示ページ
int rNumber; //一度に表示させる商品の数
int i,j,k,l;
int num=0;
//コンストラクタの定義
search(String keyword, String type, String sort,String page,int number){
 rKeyword = keyword;
 rType = type;
 rSort = sort;
 rPage = page;
 rNumber = number;
}
//search メソッドの定義
public void itemSearch(){
 try{
  BufferedWriter bw = new BufferedWriter(
   new OutputStreamWriter(
    new FileOutputStream("search.xml"),"UTF-8"));
  AmazonSearchServiceLocator locator = new AmazonSearchServiceLocator();
  AmazonSearchPort service = locator.getAmazonSearchPort();
l=rNumber;
  for(k = 0; k < l; k++){
   KeywordRequest keyReq = new KeywordRequest();
   //検索ワードの設定
   keyReq.setKeyword(rKeyword);
   // 検索する商品の種類
   keyReq.setMode(rType);
   // 国番号
   keyReq.setLocale("jp");
   // データの種類 "lite" "heavy"
   keyReq.setType("heavy");
   // ソート順
   keyReq.setSort(rSort);
   // 検索結果のページ番号
   keyReq.setPage(rPage);
```

```
        // アソシエイトID
        keyReq.setTag("downhill-22");
        // ディベロッパートークン
        keyReq.setDevtag("1G85SQRT7WTFZP0Y5T02");

        // invoke service
        ProductInfo pInfo = service.keywordSearchRequest(keyReq);
//      if(pInfo == null){

        Details details[] = pInfo.getDetails();
        if(k == 0){
//コンソール上に検索条件を出力
        System.out.println("**********************************************");
        System.out.println("TotalHitItems = " + pInfo.getTotalResults());
        System.out.println("<検索条件>¥n 検索キーワード = " + rKeyword
        + "¥n 商品の種類 = " + rType
        + "¥nSort の種類 = " + rSort
        + "¥n 表示させるページ = " + rPage
        + "¥n 一度に表示する件数 = " + rNumber
        + "¥n**********************************************"
        );
//System.out.println("ProductName|Authors|Starring|ReleaseDate|ManuFacturer|Availability|List
Price|OurPrice UsedPrice|SalesRank|Lists|AvgCustomerRating|TotalCustomerReviews");
        bw.write("<?xml-stylesheet type=¥"text/xsl¥" href=¥"standerd.xsl¥"?>"
        + "<?xml version=¥"1.0¥" encoding=¥"UTF-8¥"?>¥n"
        +       "<ProductInfo       xmlns:xsi=¥"http://www.w3.org/2001/XMLSchema-instance¥"
xsi:noNamespaceSchemaLocation=¥"http://xml.amazon.com/schemas3/dev-heavy.xsd¥">¥n"
        + "<Request>¥n"
        + "<Args>¥n"
        + "<Arg value=¥"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)¥"
name=¥"UserAgent¥">¥n"
        + "</Arg>¥n"
        + "<Arg value=¥"" + keyReq.getDevtag() + "¥" name=¥"RequestID¥">¥n"
        + "</Arg>¥n"
        + "<Arg value=¥"" + keyReq.getLocale() + "¥" name=¥"locale¥">¥n"
        + "</Arg>¥n"
        + "<Arg value=¥"" + keyReq.getPage() + "¥" name=¥"page¥">¥n"
        + "</Arg>¥n"
```

```
        +    "<Arg    value=¥""    +    Html.encode(keyReq.getKeyword())    +    "¥"
name=¥"KeywordSearch¥">¥n"
        + "</Arg>¥n"
        + "<Arg value=¥"" + keyReq.getDevtag() + "¥" name=¥"dev-t¥">¥n"
        + "</Arg>¥n"
        + "<Arg value=¥"" + keyReq.getTag() + "¥" name=¥"name¥">¥n"
        + "</Arg>¥n"
        + "<Arg value=¥"" + "xml" + "¥" name=¥"f¥">¥n"
        + "</Arg>¥n"
        + "<Arg value=¥"" + keyReq.getMode() + "¥" name=¥"mode¥">¥n"
        + "</Arg>¥n"
        + "<Arg value=¥"" + "" + "¥" name=¥"searchWord¥">¥n"
        + "</Arg>¥n"
        + "<Arg value=¥"" + keyReq.getType() + "¥" name=¥"type¥">¥n"
        + "</Arg>¥n"
        + "<Arg value=¥"" + keyReq.getSort() + "¥" name=¥"sort¥">¥n"
        + "</Arg>¥n"
        + "</Args>¥n"
        + "</Request>¥n"
        + "<TotalResults>" + pInfo.getTotalResults() + "</TotalResults>¥n"
        + "<TotalPages>" + pInfo.getTotalPages() + "</TotalPages>¥n"
        );
        num = Integer.parseInt(pInfo.getTotalResults())/10;
        if(Integer.parseInt(pInfo.getTotalResults()) < rNumber){
         if(Integer.parseInt(pInfo.getTotalResults())%10 == 0){
          l = num + 1;
         }else{
          l = num +2;
         }
        }else{
         l = rNumber/10 + 1;
        }
        System.out.println(l-1 + "ページまで表示する");
        k = k + 1;
       }
       System.out.println("num=" + num);
       System.out.println(rPage + "ページ目");
       int intrpage = Integer.parseInt(rPage);
```

```
        rPage = Integer.toString(intrpage + 1);
        for (int i = 0; i < details.length; i++) {
          Details detail = details[i];
//コンソール上に出力
System.out.print(
// i + ": "
// +
detail.getProductName()
//+ "|"
);
        //ファイルに出力
        bw.write("<Details url=¥"" + detail.getUrl() + "¥">¥n"
         + "<Asin>" + detail.getAsin() + "</Asin>¥n"
         + "<ProductName>" + Html.encode(detail.getProductName()) + "</ProductName>¥n"
         + "<Catalog>" + Html.encode(detail.getCatalog()) + "</Catalog>¥n"
        );
//aurhs
        String[] auths = detail.getAuthors();
        if (auths != null){
//NullPointerException 回避判定
         bw.write("<Authors>¥n");
         for (int j=0; j<auths.length; j++) {
          bw.write("<Author>" + Html.encode(auths[j]) + "</Author>¥n");
//System.out.print(auths[j] + ",");
         }
         bw.write("</Authors>");
        }
//System.out.print("|");
/*System.out.print(
detail.getReleaseDate()
 + "|"
 + detail.getManufacturer()
 + "|"
 + detail.getAvailability()
 + "|"
 + detail.getListPrice()
 + "|"
 + detail.getOurPrice()
```

```
 + "|"
 + detail.getUsedPrice()
 + "|"
 + detail.getSalesRank()
 + "|"
);*/


//        else{
//          bw.write("<Authors><Author></Author></Authors>");
//        }
//artis
        String[] artis = detail.getArtists();
        if (artis != null){
//NullPointerException 回避判定
          bw.write("<Artists>¥n");
          for (int j=0; j<artis.length; j++) {
            bw.write("<Artist>" + Html.encode(artis[j]) + "</Artist>¥n");
          }
          bw.write("</Artists>");
        }
//        else{
//          bw.write("<Artists><Artist></Artist></Artists>");
//        }
//starring
        String[] starring = detail.getStarring();
        if (starring != null){
//NullPointerException 回避判定
          bw.write("<Starring>¥n");
          for (int j=0; j<starring.length; j++) {
            bw.write("<Actor>" + Html.encode(starring[j]) + "</Actor>¥n");
          }
          bw.write("</Starring>");
        }
//        else{
//          bw.write("<Starring><Actor></Actor></Starring>");
//        }
//Directors
```

```
        String[] directors = detail.getDirectors();
        if (directors != null){
//NullPointerException 回避判定
         bw.write("<Directors>¥n");
         for (int j=0; j<directors.length; j++) {
          bw.write("<Director>" + Html.encode(directors[j]) + "</Director>¥n");
         }
         bw.write("</Directors>");
        }
//        else{
//         bw.write("<Directors><Director></Director></Directors>");
//        }
        bw.write(
        "<ReleaseDate>" + detail.getReleaseDate() + "</ReleaseDate>¥n"
         + "<Manufacturer>" + Html.encode(detail.getManufacturer()) + "</Manufacturer>¥n"
//         + "<ImageUrlSmall>" + detail.getImageUrlSmall() + "</ImageUrlSmall>¥n"
         + "<ImageUrlMedium>" + detail.getImageUrlMedium() + "</ImageUrlMedium>¥n"
//         + "<ImageUrlLarge>" + detail.getImageUrlLarge() + "</ImageUrlLarge>¥n"
         + "<ListPrice>" + detail.getListPrice() + "</ListPrice>¥n"
         + "<OurPrice>" + detail.getOurPrice() + "</OurPrice>¥n"
         + "<UsedPrice>" + detail.getUsedPrice() + "</UsedPrice>¥n"
//         +        "<ThirdPartyNewPrice>"        +        detail.getThirdPartyNewPrice()        +
"</ThirdPartyNewPrice>¥n"
         + "<SalesRank>" + detail.getSalesRank() + "</SalesRank>¥n"
        );
//lists
        String[] lists = detail.getLists();
        if (lists != null){
//NullPointerException 回避判定
         bw.write("<Lists>¥n");
         for (int j=0; j<lists.length; j++) {
          bw.write("<ListId>" + lists[j] + "</ListId>¥n");
//System.out.print(lists[j] + ",");
         }
         bw.write("</Lists>");
        }
//System.out.print("|");
//        else{
```

```
//          bw.write("<Lists><ListId></ListId></Lists>");
//        }
//BrowseNode
//        BrowseNode[] browselist = detail.getBrowseList();
//        bw.write("<BrowseList>");
//        if(browselist != null){
//          for(int j=0; j<browselist.length; j++){
//            bw.write(
//            "<BrowseNode>¥n"
//              + "<BrowseId>" + browselist[j].getBrowseId() + "</BrowseId>¥n"
//              + "<BrowseName>" + browselist[j].getBrowseName() + "</BrowseName>"
//              + "</BrowseNode>"
//            );
//          }
//        }else{
//
 bw.write("<BrowseNode><BrowseId></BrowseId><BrowseName></BrowseName></BrowseNod
e>");
//        }
//        bw.write("</BrowseList>");
//Tracks
        Track[] tracks = detail.getTracks();
        if (tracks != null){
//NullPointerException 回避判定
          bw.write("<Tracks>¥n");
          for (int j=0; j < tracks.length; j++) {
            bw.write("<Track>" + Html.encode(tracks[j].getTrackName()) + "</Track>¥n");
          }
          bw.write("</Tracks>");
        }
//          else{
//            bw.write("<Tracks><Track></Track></Tracks>");
//          }
        bw.write(
          "<Media>" + detail.getMedia() + "</Media>¥n"
          + "<NumMedia>" + detail.getNumMedia() + "</NumMedia>¥n"
          + "<ISBN>" + detail.getIsbn() + "</ISBN>¥n"
        );
```

```
//Features
        String[] features = detail.getFeatures();
        if (features != null){
//NullPointerException 回避判定
         bw.write("<Features>¥n");
         for (j=0; j < features.length; j++) {
          bw.write("<Feature>" + Html.encode(features[j]) + "</Feature>¥n");
         }
         bw.write("</Features>");
        }
//        else{
//         bw.write("<Features><Feature></Feature></Features>");
//        }
//Platforms
        String[] platforms = detail.getPlatforms();
        if(platforms != null){
         for(j=0; j < platforms.length; j++){
          bw.write("<Platforms>" + platforms[j] + "</Platformas>");
         }
        }
//        else{
//         bw.write("<Platforms></Platforms>");
//        }
        bw.write(
         "<Availability>" + detail.getAvailability() + "</Availability>¥n"
//         + "<Upc>" + detail.getUpc() + "</Upc>¥n"
//         +    "<ProductDescription>"    +    Html.encode(detail.getProductDescription())    +
"</ProductDescription>¥n"
        );
//Reviews
        Reviews revList= detail.getReviews();
        if(revList != null){
         bw.write(
          "<Reviews>¥n"
          +        "<AvgCustomerRating>"        +        revList.getAvgCustomerRating()        +
"</AvgCustomerRating>¥n"
          +      "<TotalCustomerReviews>"      +      revList.getTotalCustomerReviews()      +
"</TotalCustomerReviews>¥n"
```

```
        );
//System.out.print(
// revList.getAvgCustomerRating()
// + "|" + revList.getTotalCustomerReviews()
// );
//          CustomerReview[] cusRev = revList.getCustomerReviews();
//          if (cusRev != null){
//            for(j = 0; j < cusRev.length; j++){
//              bw.write(
//              "<CustomerReview>¥n"
//              + "<Rating>" + cusRev[j].getRating() + "</Rating>¥n"
//              + "<Summary>" + cusRev[j].getSummary() + "</Summary>¥n"
//              + "<Comment>" + Html.encode(cusRev[j].getComment()) + "</Comment>¥n"
//              + "</CustomerReview>¥n"
//              );
//            }
//          }
        bw.write("</Reviews>");
      }
//System.out.print("¥n");
//SimilarProducts
//        String[] similarproducts = detail.getSimilarProducts();
//        if (similarproducts != null){
//NullPointerException 回避判定
//          bw.write("<SimilarProducts>¥n");
//          for (j=0; j < similarproducts.length; j++) {
//            bw.write("<Product>" + similarproducts[j] + "</Product>¥n");
//          }
//          bw.write("</SimilarProducts>");
//        }else{
//          bw.write("<SimilarProducts><Product></Product></SimilarProducts>");
//        }
       bw.write(
        "</Details>¥n"
       );
     }
     Thread.sleep(1500);
//検索は秒間 10 件ずつと決められているため 1 ページ分の商品情報をリクエストした後に
```

```
//1 秒間 sleep させる. ここでは余裕を持って 1.5 秒間の間をとっている
//      System.out.println("k="+k+"; num="+num+"; rPage="+rPage);
//      System.out.println("----------sleep----------");
    }
    bw.write("</ProductInfo>");
    bw.close();

  } catch (Exception e) {
    e.printStackTrace();
    System.out.println("条件を満たす商品が在りません");
  }
 }
}

////////////////////////////////////////////////////////////////////////////////

//Html クラス
//XML には「&」「"」「<」「>」の 4 文字は含むことができないため
//文字列の中にこれらの文字があった場合に変換を行うクラス
class Html{
//HTML エンコードが必要な文字
 static char[] htmlEncChar = ['&', '"', '<', '>'];
//HTML エンコードした文字列
 static String[] htmlEncStr = ["&amp;", "&quot;", "&lt;", "&gt;"];
//エンコード処理
 public static String encode(String strIn){
  if(strIn == null){
   return(null);
  }
//HTML エンコード処理
  StringBuffer strOut = new StringBuffer(strIn);
//エンコードが必要な文字列を順番に処理
  for(int i = 0; i < htmlEncChar.length; i++){
//エンコードが必要な文字の検索
   int idx = strOut.toString().indexOf(htmlEncChar[i]);
   while(idx != -1){
//エンコードが必要な文字の置換
    strOut.setCharAt(idx,htmlEncStr[i].charAt(0));
```

```
      strOut.insert(idx + 1, htmlEncStr[i].substring(1));
//次のエンコードが必要な文字の置換
      idx = idx +htmlEncStr[i].length();
      idx = strOut.toString().indexOf(htmlEncChar[i], idx);
    }
   }
 return(strOut.toString());
 }
}

////////////////////////////////////////////////////////////////////////////////

//Frame クラスを継承して ie クラスを作成
//getRuntime メソッドを使って Internet Explorer で検索結果の XML を表示
class output extends Frame{
 public output(){
  String browserName = "C:¥¥Program Files¥¥Internet Explorer¥¥iexplore.exe";
  String url = "D:¥¥java¥¥AmazonSoapSearch¥¥search.xml";
  try{
//ブラウザを起動し、必要な URL を渡す
    Runtime.getRuntime().exec(new String[] {browserName, url});
  }catch (IOException exc){
    exc.printStackTrace();
  }
 }
}
```

付録 B

プログラムソース

standerd.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="/">
 <html>
  <head>
   <title>AmazonSearch</title>
  </head>
  <body>
   <p>
    <xsl:apply-templates select="ProductInfo"/>
   </p>
  </body>
 </html>
</xsl:template>

<xsl:template match="ProductInfo">
<xsl:for-each select="Details">
<!--ソートしたい情報ごとに以下のタグを付けたファイルを追加する-->
<!--<xsl:sort select="ProductName" order="descending"/>-->
<!--<xsl:sort select="Catalog" order="descending"/>-->
<!--<xsl:sort select="Artists" order="descending"/>-->
<!--<xsl:sort select="Authors" order="descending"/>-->
<!--<xsl:sort select="Starring" order="descending"/>-->
<!--<xsl:sort select="Directors" order="descending"/>-->
<!--<xsl:sort select="ReleaseDate" order="descending"/>-->
<!--<xsl:sort select="Manufacturer" order="descending"/>-->
<!--<xsl:sort select="Availability" order="descending"/>-->
<!--<xsl:sort select="ListPrice" order="descending"/>-->
```

```
<!--<xsl:sort select="OurPrice" order="descending"/>-->
<!--<xsl:sort select="UsedPrice" order="descending"/>-->
<!--<xsl:sort select="SalesRank" order="descending"/>-->
<!--<xsl:sort select="count(Lists)" order="descending"/>-->
<!--<xsl:sort select="Media" order="descending"/>-->
<!--<xsl:sort select="Reviews/AvgCustomerRating" order="descending"/>-->
<!--<xsl:sort select="Reviews/TotalCustomerReviews" order="descending"/>-->
<!--order 属性で ascecding を指定すると昇順 descending を指定すると降順-->
 <xsl:variable name="baseurl">
 http://www.amazon.co.jp/exec/obidos/ASIN/
</xsl:variable>
 <table border="1" align="center" cellspacing="0"
width="600" frame="hsides" rules="none" cellpadding="0">
  <tr>
   <td width="135" height="135" align="center"
background="img/no-image-small.gif" rowspan="2" >
    <img>
    <xsl:attribute name="src">
     <xsl:value-of select="ImageUrlMedium" />
    </xsl:attribute>
    </img>
   </td>
   <td colspan="2">
    <b><xsl:value-of select="ProductName" /></b><br/>
    <xsl:call-template name="authors" />
    <xsl:call-template name="artists" />
    <xsl:call-template name="directors" />
    <xsl:call-template name="starring" />
    <br/>販売価格:
    <xsl:choose>
     <xsl:when test="OurPrice='null'">---</xsl:when>
     <xsl:otherwise>
      <xsl:value-of select="OurPrice" />
     </xsl:otherwise>
    </xsl:choose>
    定価:<xsl:value-of select="ListPrice" />
    中古価格:
    <xsl:choose>
```

```
      <xsl:when test="UsedPrice='null'">---</xsl:when>
      <xsl:otherwise>
       <xsl:value-of select="UsedPrice" />
      </xsl:otherwise>
     </xsl:choose>
     <br/>発売日:<xsl:value-of select="ReleaseDate" />
     ランキング:
     <xsl:choose>
      <xsl:when test="SalesRank='null'">---</xsl:when>
      <xsl:otherwise>
       <xsl:value-of select="SalesRank" />
      </xsl:otherwise>
     </xsl:choose>位
     <br/>メディア:<xsl:value-of select="Media" />
      <xsl:value-of select="Availability" />
    </td>
   </tr>
   <tr valign="top">
    <xsl:if test="count(Reviews)!=0">
    <td>平均評価
     <xsl:choose>
      <xsl:when test="Reviews/AvgCustomerRating&lt;0.25">
       <img src="img/stars-0-0.gif" />
      </xsl:when>
      <xsl:when test="0.25&lt;=Reviews/AvgCustomerRating
and Reviews/AvgCustomerRating&lt;0.75">
       <img src="img/stars-0-5.gif" />
      </xsl:when>
      <xsl:when test="0.75&lt;=Reviews/AvgCustomerRating
and Reviews/AvgCustomerRating&lt;1.25">
       <img src="img/stars-1-0.gif" />
      </xsl:when>
      <xsl:when test="1.25&lt;=Reviews/AvgCustomerRating
and Reviews/AvgCustomerRating&lt;1.75">
       <img src="img/stars-1-5.gif" />
      </xsl:when>
      <xsl:when test="1.75&lt;=Reviews/AvgCustomerRating
and Reviews/AvgCustomerRating&lt;2.25">
```

```
        <img src="img/stars-2-0.gif" />
       </xsl:when>
       <xsl:when test="2.25&lt;=Reviews/AvgCustomerRating
and Reviews/AvgCustomerRating&lt;2.75">
        <img src="img/stars-2-5.gif" />
       </xsl:when>
       <xsl:when test="2.75&lt;=Reviews/AvgCustomerRating
and Reviews/AvgCustomerRating&lt;3.25">
        <img src="img/stars-3-0.gif" />
       </xsl:when>
       <xsl:when test="3.25&lt;=Reviews/AvgCustomerRating
and Reviews/AvgCustomerRating&lt;3.75">
        <img src="img/stars-3-5.gif" />
       </xsl:when>
       <xsl:when test="3.75&lt;=Reviews/AvgCustomerRating
and Reviews/AvgCustomerRating&lt;4.25">
        <img src="img/stars-4-0.gif" />
       </xsl:when>
       <xsl:when test="4.25&lt;=Reviews/AvgCustomerRating
and Reviews/AvgCustomerRating&lt;4.75">
        <img src="img/stars-4-5.gif" />
       </xsl:when>
       <xsl:when test="4.75&lt;=Reviews/AvgCustomerRating">
        <img src="img/stars-5-0.gif" />
       </xsl:when>
      </xsl:choose>
      [レビュー数:<xsl:value-of select="Reviews/TotalCustomerReviews" />件]
    </td>
    </xsl:if>
    <td align="right">
     <a>
      <xsl:attribute name="href">
       <xsl:value-of select="concat($baseurl,Asin,'/ref=nosim/downhill-22')" />
      </xsl:attribute>
      Amazon で詳細を見る
     </a>
    </td>
   </tr>
```

```
 </table>
</xsl:for-each>
</xsl:template>

<xsl:template name="artists">
 <xsl:for-each select="Artists/Artist">
 <xsl:choose>
  <xsl:when test="position()&lt;5">
  <font size="-1"><xsl:value-of select="." />,</font>
  </xsl:when>
 </xsl:choose>
 </xsl:for-each>
</xsl:template>

<xsl:template name="authors">
 <xsl:for-each select="Authors/Author">
 <xsl:choose>
  <xsl:when test="position()&lt;5">
  <font size="-1"><xsl:value-of select="." />,</font>
  </xsl:when>
 </xsl:choose>
 </xsl:for-each>
</xsl:template>

<xsl:template name="starring">
 <xsl:for-each select="Starring/Actor">
 <xsl:choose>
  <xsl:when test="position()&lt;4">
  <font size="-1"><xsl:value-of select="." />,</font>
  </xsl:when>
 </xsl:choose>
 </xsl:for-each>
</xsl:template>

<xsl:template name="directors">
 <xsl:for-each select="Directors/Director">
 <xsl:choose>
  <xsl:when test="position()&lt;2">
```

```
    <font size="-1"><xsl:value-of select="." />,</font>
    </xsl:when>
   </xsl:choose>
   </xsl:for-each>
  </xsl:template>

  </xsl:stylesheet>
```