学士学位論文

# シソーラス Web Service の開発と Ajax への応用

平成 18 年度
茨城大学　工学部
システム工学科
執筆者:南　慶典
指導教官:新納　浩幸

# 目次

# 図目次

# 表目次

# 第1章

# 序論

## 1.1 はじめに

シソーラスとは，単語の上位/下位関係，部分/全体関係，同義関係，類義関係などによって単語を分類し，体系づけた辞書であり，自然言語処理における重要なリソースである．しかし，それを一から開発するには多大な時間と労力を必要とする．そこで現在使用されている既存のシソーラスを，Web を通じて誰でも独自のサイトなどで扱えるような Web Service を作成する．Web Service として実現すれば処理結果を XML(Extensible Markup Language)としてクライアントに渡すので，様々なプラットフォーム間でやりとりをすることができる．また他の Web Service とマッシュアップ（組み合わせ）して新しいシステムを構築することも容易である．

Web Service とは SOAP あるいは REST といった仕組みを利用し，XML 形式のメッセージ交換によってインターネット上の独立したサービスを連携させる技術，またそのアプリケーションである．Web Service を利用することで，これまで 1 つのサイトで 1 つのサービスを提供されたものが，複数のサービスを連携させた 1 つのサービスとして提供することが実現される．

本研究では，シソーラスを提供するWeb Serviceの開発を目的とする．また作成したWeb Serviceを実際に利用できるか確認するため，一般に提供されているWeb Serviceを利用してマッシュアップWeb Application[1]を作成する．シソーラスを提供するWeb Serviceは，荻野シソーラス[2]を利用してデータベースを作成し，さらにそのデータを利用しXMLを生成するServletを作成することで実現する．さらに，XMLに含まれるシソーラスデータの確認用として，クライアントサイド技術であるAjaxを利用する．作成したWeb Serviceとその他のWeb Serviceとをマッシュアップさせた検索システムを作成し，シソーラスWeb ServiceとマッシュアップWeb Applicationの評価を行う．

## 1.2 本論文の構成

2章ではシソーラスとは何か，自然言語処理においてどのような役割があるかを述べる．

3章ではWeb Serviceの特徴，Web SeviceとXMLの関係性，Web Serviceの通信プロトコル，マッシュアップについて述べる．

4章では本研究でWebServiceを作成する際に使用したサーバサイド技術と，マッシュアップしたWebApplicationを作成する際に使用したクライアントサイド技術について説明する．

5章では作成したWeb Serviceの概要と仕様について説明する．

6章では他のWeb Service（Yahoo!など）と組み合わせて作成したマッシュアップWeb Applicationの概要と仕様について説明する．

7章ではシソーラスWeb ServiceからXMLが取得できるか，またマッシュアップWeb Applicationの検索を実験し，システムの問題点や今後の課題を述べる．

8章では本研究の結論を述べる．

# 第2章

# シソーラス

シソーラスとは，単語の上位/下位関係，部分/全体関係，同義関係，類義関係などによって単語を分類し，体系づけた辞書のことである．同義語や関係のある語がまとめてあり，データ作成や検索のために利用される．

一般にデータベース検索では，検索したい内容を的確に表す言葉（キーワード）を入力しなくてはならないが，データベースにある言葉と検索で指定した言葉の表現が違うと，同じ意味なのに検索漏れになる場合がある．

これを防ぐために，検索に使える言葉を分類して，言葉とその同義語，関連語，広義語，狭義語といった言葉を分野や内容に応じて整理したものがシソーラスである．シソーラスを利用することで，検索した言葉と完全に一致しない言葉でも検索の対象になり，検索漏れを極力減らすことができるようになる．

例えば，日本を表すのにも，倭，大和，Japan，Nippon などといったように複数の表現ある．これらをあらかじめシソーラスに登録しておくと，キーワードとして日本と設定した場合でも，大和，倭，Japan，Nippon とあっても検索結果に反映される．

# 第3章

# Web Service

## 3.1 XML

XMLはインターネットの標準としてW3Cより勧告されたメタ言語である．メタ言語とは，言語を作る言語という意味である．つまり，ただ単にXMLを使うだけで情報を記述することは出来ない．まず，情報を記述するための言語をXMLを用いて作成し，それを用いて情報を記述することになる．[3]

言語を作る際には，どんな構文でも好きなように作れるわけではなく，あらかじめ用意された範囲内に限定される．XMLは，SGML（Standard Generalized Markup Language）のサブセットになるように言語仕様が規定されている．しかし，これは，XMLがSGMLのサブセットであることを規定しない．XMLにはSGMLを逸脱した構文や機能があり，SGML以上のものである．サブセットであるというのは，既存のSGMLツールが利用できることを意図したものでしかなく，XMLツールが豊富に存在する現在では特に重要な意味はない．逆に，XMLはSGMLのサブセットだからといって，SGMLをXMLの代わりに採用するのは間違いである．SGMLではXMLの一部の機能が使用できない．また，SGMLではインターネット上の情報交換性が保証されない．

SGMLの開発目的は，電子的な文書管理である．各種文書を長期間保存したり，人手を使わない自動管理を行うために，直接的な表記に関する情報ではなく，論理的な文書構造を記述するようにした言語である．

XMLでは，これに加えて，電子的なデータの交換も役割も持つ．そのため，文書だけでなく，単純な数値の羅列のような電子データの交換にも使用される．

しかし，元々文書のためのメタ言語であるため，文書を記述する場合でもデータを記述する場合でも，XML文書と呼ぶのが習慣である．逆に，XMLの目的に関して，よくある誤解は，「HTMLの後継言語である」「XMLはデータベースの情報を交換するための言語」「XMLはEC（電子商取引）のための言語」といったものがある．実際に，HTMLの後継言語であるXHTMLという言語を作るためにXMLを使用しているのは事実であるが，XMLそのものはHTMLではなくSGMLの後継言語である．また，データベースの情報を交換したり，ECの分野でXMLが使用されているのは事実であるが，それを目的に作られたと言うわけではない．本来XMLは文書を扱うためのメタ言語であって，データ交換などで利用されるのは，XMLの強力さとカバー範囲の広さを示すものである．

以下に，簡単なXML文書の一例を示す．

```
<?xml version="1.0" encoding="UTF-8" ?>
<Person>
  <name>Suzuki</name>
  <member>
     <number>1234</number>
     <age>18</age>
     <e_mail />
  </member>
</Person>
```

図 3.1:XML の例

## 3.2 Web Service の特徴

Web Serviceとは，WWWやHTTPなどのインターネットの標準技術を用いて，SOAPあるいはRESTという仕組みを利用し，XML形式のデータを交換することによってインターネット上の独立したサービスを連携させる技術，またそのアプリケーションのことである[2]．Web Serviceの特徴は，「情報の送受信を標準的なHTTPプロトコル上で行える」，また，「XMLベースで情報を授受するので，特定のプラットフォームに依存しない」という点が挙げられる．Web Serviceを利用することで，これまで１つのサイトで１つのサービスを提供されたものが，複数のサービスを連携させた１つのサービスとして提供することが実現される．またOSやフレームワークが提供する機能に加えて，インターネット上のサービスも自分のプログラムに取り込めるようにもなる．

なおWeb Serviceは，Webを利用した通常の情報サービスと明確に区別できるように，「XML Web Service」と呼ばれる．現在，GoogleやYahoo!，Amazonといった企業をはじめとして，数多くのWeb Serviceが公開されている．

Web Serviceは，SOAP/WSDLベースの通信を行うものに限定する向きもあったが，現在ではRESTベースのサービスも広がりを見せるなど，通信方式を限定せずに広義の意味で使われることが多くなってきている．技術的には，以下のようなプロトコルが使われている．

XML-RPC : metaWeblog APIのブログ投稿管理APIなど．XMLベース

SOAP/WSDL : Google SOAP Search APIのWebサイト検索APIなどXMLベース

REST : Technoratiのブログ検索APIなど．レスポンスはXMLかJSONが多い

これらの通信方式はWebサービス提供者側が決めることである．ただ，最近の潮流としては，REST-fulなインタフェースを採用してXML形式だけでなくJSON形式のレスポンスを返す

サービスが増えてきている．RESTベースのWebサービスAPIは，その動作を手軽に試せるので，APIを利用したサービスが開発しやすい特徴がある．

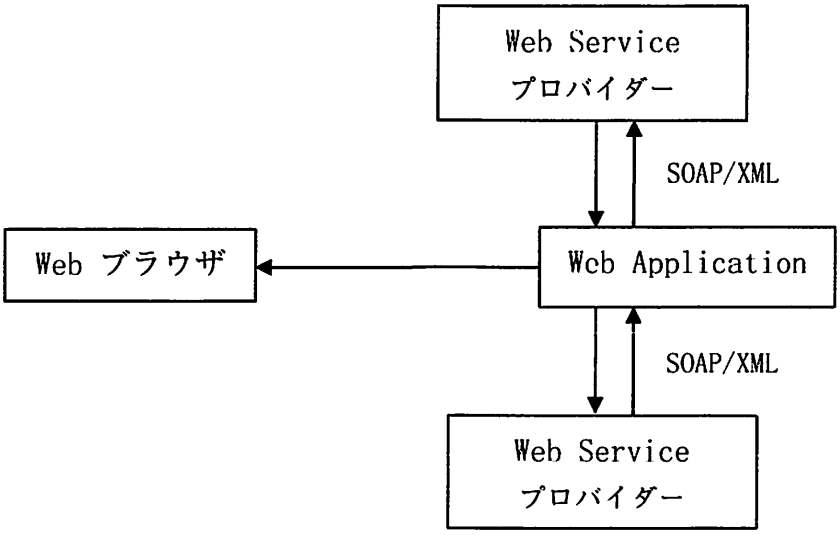


図 3.2:Web Service

従来のWeb Applicationでは，ブラウザに対して処理を行うので，HTMLから渡されたリクエストデータを処理し，それをまたHTMLにしてブラウザに返すということを行うので，プログラムがHTMLに埋め込まれたデータだけを参照したい場合はとても面倒な処理を行うことになる．それに対して，Web ServiceはWeb Applicationとデータを XML文書やJSON(JavaScript Object Notation)でやりとりするので，必要なデータの参照が非常に簡単になり，XMLはどのプラットフォームにもあるプログラミング言語で解析できるのでプラットフォームを気にせずに出来る．

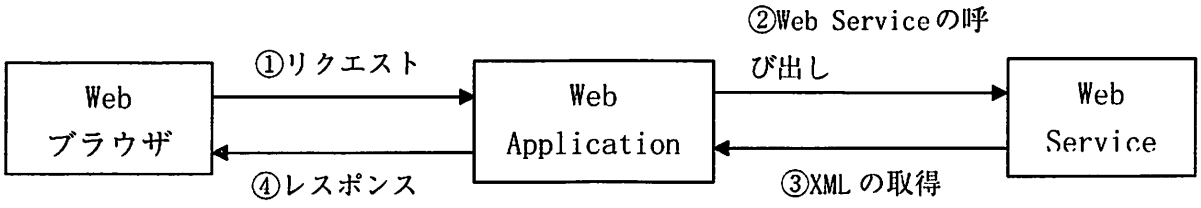


図 3.3:Web Applicationと Web Service

## 3.3 SOAP

XMLとHTTPなどをベースとした，他のコンピュータにあるデータやサービスを呼び出すためのプロトコル(通信規約)．Microsoft社やUserLand Software社，Developmentor社が中心となって開発された．

SOAPによる通信では，XML文書にエンベロープ(封筒)と呼ばれる付帯情報が付いたメッセージを，HTTPなどのプロトコルで交換する．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．

SOAP 1.1では，実際にデータの送受信に使う下位プロトコルは，すでに広く普及しているHTTPやSMTP，FTPなどから選択できるようになっており，企業間で利用する場合でもファイアウォールなどを安全に通過することができる(SOAP 1.0ではHTTPのみ)．

現在，WWW関連技術の標準化を行なうW3Cによって標準の策定が行なわれており，IBM社やLotus社など，大手ソフトウェアメーカーも自社製品での対応を表明している．

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」と呼ばれることもある．

SOAPによって外部から利用可能な，部品化されたWebベースのアプリケーションソフトは「Web Service」と呼ばれる．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・照会できるようにするWeb Serviceを「UDDI」という．

SOAPでは本文となるメッセージ自体に拡張性を持たせるために，「SOAP Emvelope」と呼ばれる構造が定義されている．SOAP Envelopは，Envelopという名前のルート要素と１つのBodyという名前の子要素を持っている．通常，Body要素の内部に，送信したい情報を格納する．拡張性は，Envelopeのもう1つの子要素であるHeader要素で得られる．Headerはオプションなので，EnvelopeにはHeaderが含まれていなくてもかまわない．

HeaderにはBodyと一緒に送信したい付加的な情報を格納できる．例として図:3.4のXMLはSOAP Envelopeである．

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/envelope/">
<env:Body>
<m:ItemLookup
xmlns:m="http://webservices.amazon.com/AWSECommerceService/2005-03-23">
<m:SubscriptionId>登録 ID</m:SubscriptionId>
        <m:Request>
                <m:ItemId>1111222233</m:ItemId>
        </m:Request>
</m:ItemLookup>
</env:Body>
</env:Envelope>
```

図 3.4:SOAPEnvelope

上の例ではヘッダは含まれていない．そこでOASISで標準化されているWS-Securityに基づいてユーザ名とパスワードで認証を行うためのヘッダを含めることができる．

なお，SOAPでは，Envelopeの構造は決めているが，Bodyの内部やHeaderの内部については何も決めていない．XMLであればどんなものでもBodyやHeaderに指定できる．SOAPのデータは以下の図3.5ようにHTTPで運ぶことができる．

```
POST /order.cgi HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: xxx

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/envelope/">
  <env:Body>
    <m:ItemLook
xmlns:env="http://webservices.amazon.com/AWSECommerceService/2005-03-23">
      <m:SubscriptionId>登録 ID</m:SubscriptionId>
      <m:Request>
        <m:ItemId>1111212112</m:ItemId>
      </m:Request>
    </m:ItemLookup>
  </env:Body>
</env:Envelope>
```

図 3.5:SOAP での通信

現在では，多くのプログラミング言語でSOAP対応のライブラリやツールが開発され，すでに実用レベルに達している．例えば，Perl用のSOAP;;LiteやPIIP用のnuSOAPなどである．2002年にはGoogleがGoogle Web APIsを公開し，SOAPでGoogleのWeb検索システムにアクセスできるようにしている．

## 3.4 XML-RPC

インターネット上でリモートプロシージャコールを実行するためのプロトコルのことである．SOAPの基となった仕様．クライアントがXML形式のテキストで記述された型付の引数をサーバ側アプリケーションに渡し，サーバが返り値を同じくXML形式のテキストで返すという動作をする．シンプルな仕様である XML-RPC ではトランザクションや認証といった高度なプロトコルの実現は難しかったため，当初その拡張として考案されたのが SOAP である．しかし拡張性の高い SOAP に対して XML-RPC は仕様がシンプルで扱いやすいという理由から，現在も多くの場面で利用され続けている．XML-RPCの仕様自体は非常にシンプルで，各言語におけるXML-RPCのライブラリやモジュールが比較的充実しているので，実装も容易ではあるが，セキュリティや機能の拡張性という面でいくつか問題がある．また，XML-RPCの仕様が特定のベンダや個人の管理の下にあるため，融通が聞かないと言った問題もある．事実，最近までXML-RPCではASCII以外の文字コードの使用が禁止されており，日本語などを直接利用することができなかった[4].

以下に簡単なXML-RPCのリクエストの例を示す．

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
   <methodName>examples.getStateName</methodName>
   <params>
      <param>
         <value><i4>41</i4></value>
         </param>
      </params>
   </methodCall>
```

図 3.6:XML-RPC リクエスト

ヘッダ最初の行の URI は特に規定していない．サーバ側が XML-RPC コールを処理できるのであれば，URI が指定されていなかったり，単に / となっていても構わない．しかし，サーバが XML-RPC 以外の HTTP リクエストも処理している場合，XML-RPC のコードを呼び出すための URI を指定することも可能である(例では URI が /RPC2 となっており，サーバに RPC2 という名前に対応するリクエストとして処理するよう告げている)．
またUser-Agent と Host は必ず指定しなければならない．Content-Type は text/xml である．Content-Length も必須で正しい値を指定しなければならない．

ペイロードのフォーマットとしては，ペイロード(データ本体)は XML であり，単一の <methodCall> で構成される．

<methodCall> は呼出すべきメソッド名の文字列をデータとして持つ副要素 <methodName> を含んでいなければならない．使用できる文字はアルファベットの大文字と小文字，数字，アンダースコア，ドット，コロンおよびスラッシュである．mehodName にどんな文字列を指定すると，どのような内容が実行されるかは，サーバアプリケーションに依存する．

プロシージャコールにパラメータがある場合，<methodCall> は副要素 <params> を含んでいなければらない．<params> はさらに複数の <param> 含み，各 <param> はそれぞれ単一の <value> を含む．

以下にXML-RPC リクエストに対するレスポンスの例を以下に示す．

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><string>South Dakota</string></value>
         </param>
      </params>
   </methodResponse>
```

図 3.7:XML-RPC レスポンス

レスポンスのフォーマットとしては，XML-RPC より下のレベルでエラーが発生しない限り，常に 200 OK を返す．Content-Typeはtext/xmlである．Content-Lengthも必須で正しい値を返さなければならない．

レスポンスのbodyは単一のXMLで，単一の<methodResponse>から成り立つ．<methodResponse>は単一の<params>を含むことが可能で，<params>は単一の<param>を含み，<param>は単一の<value>を含む．<methodResponse>には単一の<fault>を含ませることも可能である．<fault>は単一の<value>を含み，その<value>は単一の<struct>を含み，<struct>は2つの要素<faultCode>と<faultString>を含む．<faultCode>の型は<int>で，<faultString>の型は<string>である．<methodResponse> に <fault> と <params> の両方を含ませることはできない．

## 3.5 REST

パラメータをURLにクエリ文字列として設定し，HTTP GETで送信して，結果を単純なXMLで受け取り，どのプログラミング言語にも用意されているXMLパーサで解析するほうがシンプルかつ簡単なプログラミングができる．SOAPがHTTP POSTを使って情報を取得（GET）しているのはおかしいので，情報を取得するときはHTTP GETを，情報を送信するときはHTTP POSTを使うほうがより「Webらしい」というこれらの意見は，2000年に発表された論文に基づき，REST（REpresentational State Transfer）と呼ばれるようになった．

本研究では，単純にSOAPを使わずにXMLをHTTPで送受信するAPIをREST呼んでいる場合[5]もあるので，まとめてRESTと定義する．

REST を支持する人々は，ウェブのスケーラビリティと成長は，次に述べるような，いくつかのキーとなる設計原則の結果であると論じる．

HTTPメッセージの一つ一つが，そのリクエスト（メッセージ）を理解するために必要な全ての情報を含む．そのため，クライアントもサーバーも，メッセージ間におけるセッションの状態を記憶しておく必要がない．ただし実際には，多くのHTTPベースのアプリケーションはクッキーやその他の仕掛けを使ってセッションの状態を管理している（URLリライティングのような一部のセッション管理手法を使うシステムは，RESTful ではない）．RESTful なシステムでは，すべてのリソースはURI (Uniform Resource Identifier)で表される一意的な(ユニークな)アドレスを持つ．

RESTシステムでは，多くの場合，HTMLまたはXMLファイルを使う．こうしたファイルに情報およびその他のリソースへのリンクを含める．こうすることにより，あるRESTリソースから他のRESTリソースを参照したい場合は単にリンクを辿るだけでよい．レジストリなどの他の基盤的な機能を使う必要はない．RESTにおいて重要な概念は，「リソース」(情報の断片)である．個々のリソースは，グローバルな識別子(URI)により参照することができる．リソースに対する操作は次のようにして行われる．

ネットワークの「コンポーネント」(クライアントやサーバ）が，標準化されたインターフェイス(HTTP)により通信する．ネットワークを介してリソースの「表現」(representation)を交換する(実際にはファイルがアップロード・ダウンロードされる)．しかし実際のところこうしたリソース操作は議論の対象となっている．一部の人々には「リソース」と「表現」とを区別することは観念的すぎるとの意見がある．ただし RDFコミュニティでは，リソースと表現の区別は，一般的に行われている．

RESTの具体例として，例えばAmazon.co.jpに関する詳細情報を見る場合，それぞれの商品についている商品IDを指定して，その商品の詳細ページにアクセスする．Amazon.co.jpの場合は次の図のようなHTTP GETリクエストを送信すれば，詳細ページの内容がHTMLで返される．

```
GET /exec/obidos/ASIN/4774115533/HTTP/1.1
Host: www.amazon.co.jp
```

図 3.8:Amazon への HTTP GET リクエスト

ここでは4774115533が商品のIDとなる．そこで同じ情報をXMLで得るためにわざわざSOAPEnvlelopeを作らなくても，XMLを返信してくれるサーバに対して以下の図のようにアクセスすればよい．

```
-- Web ブラウザから送信 --
GET /onca/xml?SubscriptionId="登録
ID"&AWSECommerceService&Operation=ItemLookup&ItemId=4774115533 HTTP/1.1
Host: webservices.amazon.co.jp

-- Web サーバからの返信 --
HTTP/1.1 200 OK
Date: Thu, 23 Jun 2005 02:09:17 GMT
Content-Type: text/xml; charset=UTF-8

<ItemLookupResponse
xmlns="http://webservices.amazon.com/AWSECommerceService/2005-03-23">
  <Items>
    <Item>
      <ASIN>4774115533</ASIN>
   <DetailPageURL>http://www.amazon.co.jp/o/ASIN/4774115533</DetailPageURL>
      <ItemAttributes>
        <Author>吉松 史彰</Author>
        <Author>山崎 明子</Author>
        <Title>C#.NET プログラミングマニュアル</Title>
      </ItemAttributes>
    </Item>
  </Items>
</ItemLookupResponse>
```

図 3.9:REST での送受信

## 3.6 Web Applicationとマッシュアップ

マッシュアップ（Mashup）とは，公開されているWeb Serviceやデータソースの組み合わせにより，独自のコンテンツ，サービスを生み出す手法である．マッシュアップを実現するための重要な技術がWeb Serviceなのである．Web Serviceは最近登場した技術ではなく，2000年初頭には実現可能となっており，大きな期待が寄せられていた．しかし，Web Serviceに対応するインターネットサービスがほとんどなく，広い普及にはいたらなかった．

2003年頃からYahoo！, Amazon, GoogleなどがWeb Service API（Web Serviceプロバイダーとインターネットサービスを連携させるプログラミングインターフェース仕様）を整備したことで，2004年後半から急速に普及し始めた．以下に代表的なWeb Service APIの表を載せる．

表3.1: Web Service API 一覧

| API | 内容 |
|---|---|
| <Amazon> | |
| Amazon E-Commerce Service | 小売 |
| Amazon A9 OpenSearch | 検索サービス |
| Amazon Alexa | 検索サービス |
| <eBay> | |
| eBay | オークション |
| <Google> | |
| Google Search | 検索サービス |
| Google Desktop Search | デスクトップ検索 |
| Google Maps | 地図サービス |
| Google AdWords | コンテンツマッチ広告 |
| <Microsoft> | |
| Microsoft MSN Search | 検索サービス |
| Microsoft start.com | ポータル |
| Microsoft Virtual Earth | 地図サービス |
| Microsoft MSN Messenger | メッセンジャー |
| <Yahoo!> | |
| Yahoo Search | 検索サービス |
| Yahoo Shopping | ショッピング |
| Yahoo Map Image | 地図画像作成サービス |
| Yahoo Maps | 地図サービス |
| Yahoo Local Search | 地域検索サービス |
| Yahoo Image Search | 画像検索サービス |
| Yahoo Audio Search | 楽曲検索サービス |
| Yahoo Music Engine | 音楽プレーヤー |
| Yahoo Video Search | 動画検索 |
| <その他> | |
| Flickr | 写真共有サービス |
| del.icio.us | ソーシャルブックマーク |
| Bloglines | RSS リーダー |
| Technorati | ブログ検索サービス |
| Creative Commons | ライセンス |

このようにWeb Serviceが普及したことにより，マッシュアップという手法を利用し，サービスプロバイダー自身が独自のデータソースや独自のサービスを所有しない，新たなビジネスモデルとして展開されている．

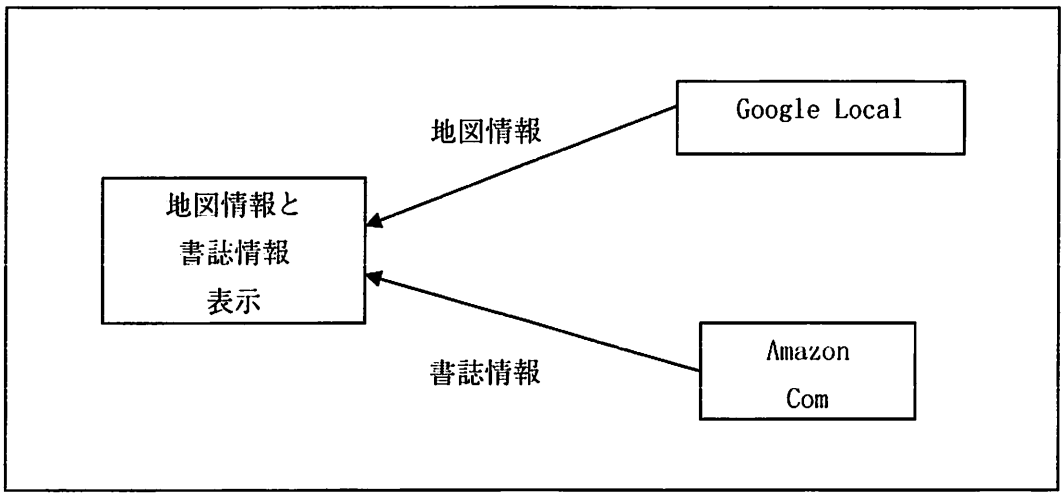


図 3.10:マッシュアップ

効果的なマッシュアップを実現しているサービスを以下に述べる．

「くるくるブログ」は，12カ国語対応のブログマッチングサービスであり，JavaScriptを多用した動きのある画面構成が特徴で日替わりのテーマに沿って各国語の最新ブログ記事を紹介していく．このサイトは，関連語辞書・翻訳辞書機能としてのWikipediaと，ブログ検索サービスとしてのTechnorati API，それに画面キャプチャとサムネイル画像生成サービスのWebToJpegを組み合わせたマッシュアップである．WikipediaはWeb Service APIを提供しているわけではないが，XMLを解析するのと同様にHTMLページの解析により関連語や翻訳語を取得することができる．

「はてなマップ」は，Google Maps APIを利用し，マッシュアップを実現している地図サービスである．はてなマップでは，表示エリアにあるはてなフォトライフの写真とスコアの高いはてなダイアリーキーワード(スコアが同じ場合は更新日時が最新キーワード)，トラックバック，公開設定のクリップが一定数表示される．

IT系求人マッチングサービスのFind-Job!は，膨大な求人データを求職者とマッチングさせるビジネスを展開している．求職者は希望する「勤務地」を条件に検索する．

Find Job! Maps βはGoogle Maps APIを利用し，地図上に求人企業の所在地を表示する．加えて，求職者が希望勤務エリアで企業を絞り込める機能が実現されている．さらには，地図を衛星写真に切り替えれば，企業所在地のイメージを視覚的に把握することも可能である．これにより，求職者へのアンケート実施時に高い割合を占めていた"できるだけ短い

時間で自分にあった企業を探したい"というニーズに対応することができる．

このように，既存のサービスに独自のアイディアを加えることでオリジナルのマッシュアップサイトを構築することができる．マッシュアップサイトの発展は，Web Service API提供側（コンテンツ・ホルダー）から見ても利点がある．Web Service APIの提供と利用が広がれば，マッシュアップを構築するサードパーティを経由することで，コンテンツ・ホルダー単独ではリーチできなかったエンドユーザーに対しても，新たなサービスを提供できる．

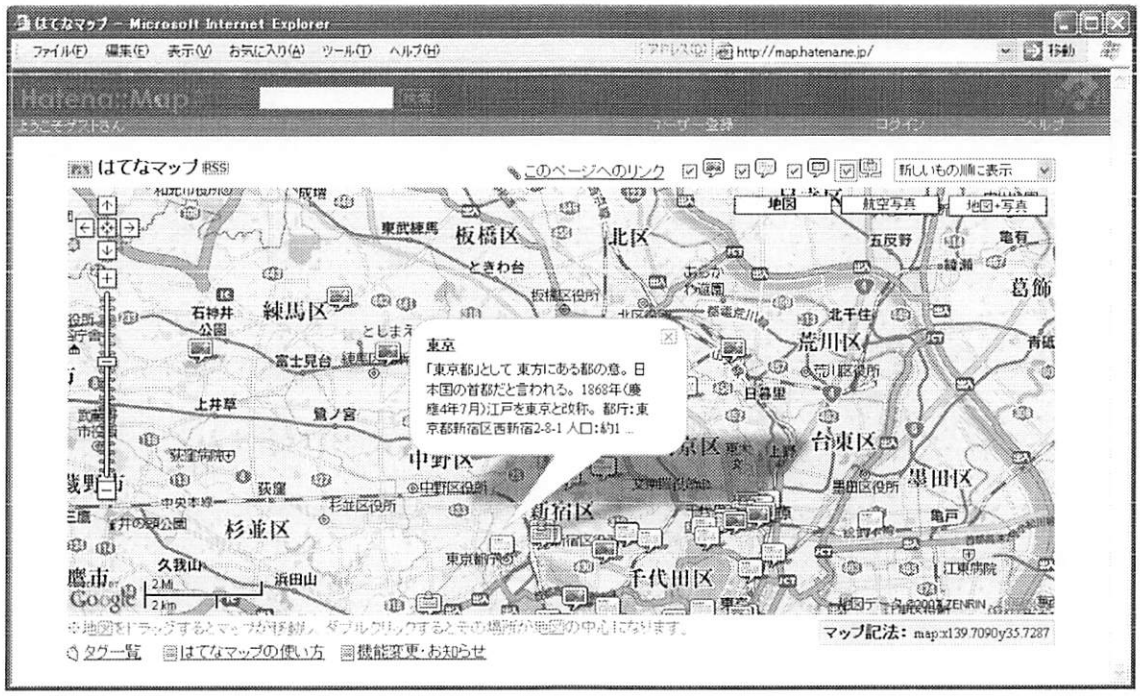


図 3.11:はてなマップ

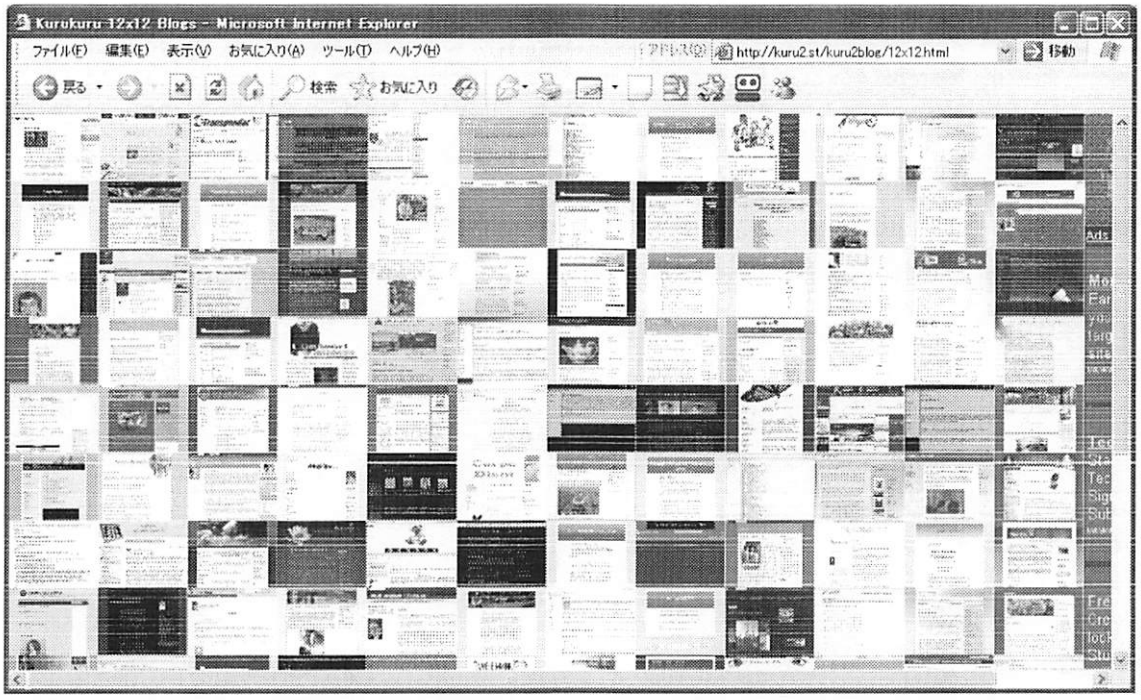


図 3.12:くるくるブログ

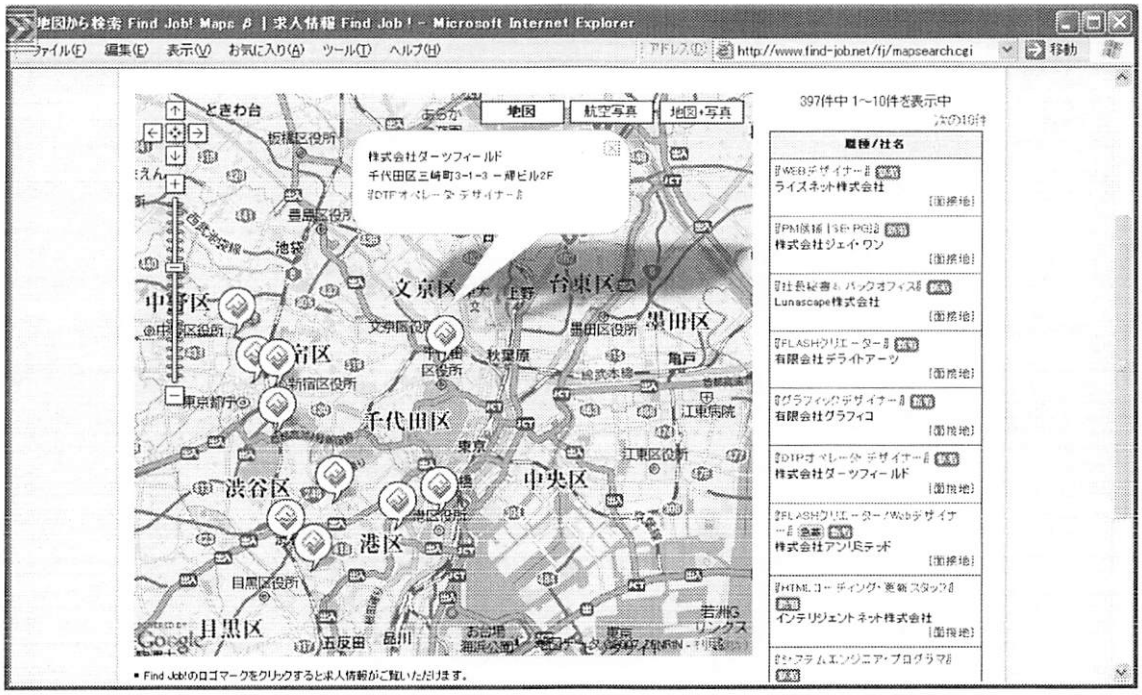


図 3.13:Find Job! Maps β

## 3.7 使用したWeb Serviceの概要

本研究で作成したマッシュアップWeb Applicationには以下の表のWeb Serviceを利用した.

表 3.2:使用したWeb Service API一覧

| Web Service API名 | 説明 |
|---|---|
| Yahoo!検索Webサービス | Yahoo!の検索エンジンを利用し,ウェブ上に公開されているページを検索する機能を提供する.一日に. |
| はてなダイアリーキーワード連想語 API | キーワードから連想される単語を返す |
| Simple API Wikipedia | キーワードに対する簡単な説明,記事が投稿された時間などを返す. |
| サイトサムネイル作成 API | サイトのサムネイルを作成する. |
| Technorati API(tag) | ブログに張られているタグより, ブログを検索. 一日に500件まで取得できる. |

Yahoo!検索Webサービス, Technorati API(tag)を使用するにはそれぞれアプリケーションIDが必要となる. IDはそれぞれサイトに登録することで発行される. またこの2つには使用制限があり, Yahoo!検索Webサービスでは一日に同じIPアドレスからのリクエストは50000件までとなっており, Technorati API(tag)では一日の500件までのブログ情報しか受け取ることができない.

はてなダイアリーキーワード連想語APIとサイトサムネイル作成API以外のWeb Serviceは, REST APIである. REST APIは, URLパラメータで入力パラメータを指定し, そのURLでエンドポイントに対してHTTP GETすると XML でデータを取得することができる. 各URLにのパラメータを指定することで任意の結果を得ることが出来る.

サイトサムネイル作成APIは, サムネイル化したいURLを"http://img.simpleapi.net/"の後に続けて入力すると, 画像ファイルとして取得できるWeb Serviceである. REST APIと似た動きをしているが, XMLを扱ってはいないので本研究ではこのAPIはREST APIではないものとして位置づけた.

またはてなキーワード連想語APIは, XML-RPCのWeb Serviceであり, キーワードからはてなダイアリーのキーワードデータベースと照合し, 関連するキーワードを返信するAPIである. 結果を取得するには, 仕様書によって決められているXML(XML-RPC)をあらかじめ作成し, キーワードやパラメータなどのリクエストデータを含め, 送信する必要がある.

REST APIの各リクエストURLは, 表3.3に示す. また各リクエストパラメータを表3.4と表3.5, 表3.6に示す.

表3.3:REST APIの各リクエストURL

| REST API名 | リクエストURL |
|---|---|
| Yahoo!検索Webサービス | http://api.search.yahoo.co.jp/WebSearchService/V1/webSearch |
| Simple API Wikipedia | http://wikipedia.simpleapi.net/api |
| Technorati API(tag) | http://api.technorati.com/tag |

表3.4:Yahoo!検索Webサービスのリクエストパラメータ

| パラメータ | 説明 |
|---|---|
| appid（必須） | アプリケーションID |
| query（必須） | （UTF-8エンコードされた）検索クエリー．このクエリーはYahoo!検索の全言語をサポートし，またメタキーワードも含む． |
| type | 指定検索の種類:<br>allは全クエリー文字を含む検索結果を返す．<br>anyはクエリー文字のうちいずれかを含む検索結果を返す．<br>phraseはクエリー文字を文章として含む検索結果を返す． |
| results | 返却結果の数．デフォルトで10，最大で50． |
| start | 返却結果の先頭位置．最終位置（start + results - 1）は，1000を超えられない．デフォルトでは1． |
| format | 検索するファイルの種類を指定する．any（デフォルト），html，msword，pdf，ppt，rss，txt，xls |
| adult_ok | アダルトコンテンツの検索結果を含めるかどうかを指定する．1の場合はアダルトコンテンツを含む．デフォルトでは値なし． |
| similar_ok | 同じコンテンツを別の検索結果とするかどうかを指定する．1の場合は同じコンテンツを含む．デフォルトでは値なし． |
| language | languageで書かれた結果になる．デフォルトではja（日本語） |
| country | ウェブサイトが位置する国の国コード．デフォルトでは値なし． |
| site | 検索するドメイン（例えば www.yahoo.co.jp）を制限する．30ドメインまで指定することができる．デフォルトでは値なし． |

表3.5:Simple API Wikipediaのリクエストパラメータ

| リクエストパラメータ | 説明 |
|---|---|
| keyword | キーワード．エイリアスとして，qも指定可能 |
| output | 出力方式（xml, rss, json, html, javascript, php, tsvを指定可能．デフォルトはxml） |
| callback | 出力形式がJSONP時の名称を指定可能 |
| lang | 現在は日本語（ja）のみ |
| search | 現在未実装につき1のみ．（0.その特定キーワードのみを取り出す，1.前方一致，2.後方一致，3.前後方一致，4.FULLTEXT） |

表3.6:Technorati API(tag)のリクエストパラメータ

| リクエストパラメータ | 説明 |
|---|---|
| key（必須） | 登録時に発行されるAPIのキー． |
| tag（必須） | （UTF-8エンコードされた）検索クエリー．ブログに貼られているタグを検索する． |
| limit | 取得するブログの数．0～100以下で指定，デフォルトでは20 |
| start | 返却結果の先頭位置． |
| excerptsize | ブログの本文から抽出する文字数，デフォルトでは100 |
| topexcerptsize | タイトルと本文をあわせて抽出する文字数．デフォルトでは150 |

# 第4章

# サーバサイド技術とクライアントサイド技術

## 4.1 CGIの問題点

CGIはサーバサイドプログラミングの手法のひとつで，Perl言語などで書かれたプログラムをサーバ側で動かしてHTMLを動的に生成するものである．CGIは，HTTPサーバのモジュールとして用意され，要求を受け取ったHTTPサーバは，当該サービス用のプロセスを立ち上げて処理し，処理結果をクライアントに返す．ひとつの処理ごとに，プロセスを立ち上げ・終了が必要になる．プロセスというのは，メモリ上を占有し，コンピュータ制御の実行経路をロードするひとつの単位である．一般に，複数のプロセスによってCPUは競合状態にあり，OSがそれらの実行可能状態にあるプロセスに対して適宜CPUの実行を割り当てる．

プロセスの開始と停止は，メモリ領域へのロードと占有された領域の解放，OSへの登録，ディスパッチなどのオーバーヘッドが必要となり，コンピュータにとっては負荷の高い処理である．そのため，CGIの場合は，クライアント要求の増大によるシステム資源の圧迫は深刻であり，処理速度，安定性，拡張性などの点で，大規模システムには不適切である．その反面，クライアントごとに，別々に動作するので安全性は高まる[6]．

具体的には図4.1のような動きをする．クライアントからのリクエストがあると，サーバ側では該当するプログラムをロードし，実行する．その結果，得られたHTMLをレスポンスとしてクライアント（クライアントA）に返す．CGIプログラムの実行中に，他のクライアント（クライアントB）から同じプログラムがリクエストされたとする．サーバはクライアントBのために，同じプログラムをロードし，クライアントAとクライアントB別々に実行され，それぞれのクライアント用に生成されたHTMLを返す．

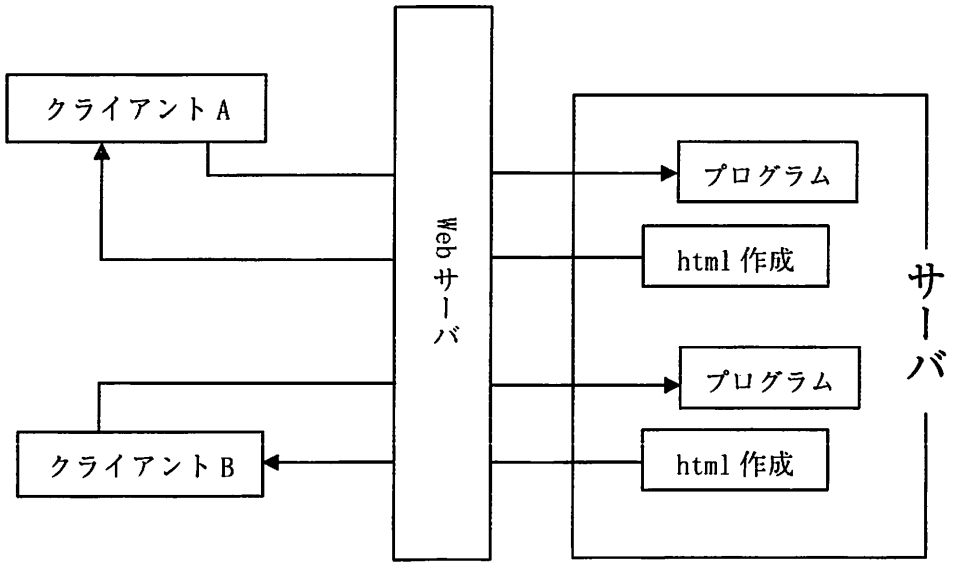


図 4.1:CGIの動き

## 4.2 Servlet

ServletはCGIの問題を解決しようとして登場した，サーバサイドスクリプトである．これは個々のリクエストを，サーバ上に常駐するプロセス内のスレッドとして実行するものである[6].

サーバにインストールし，サーバ上で実行されるJava アプリケーションをServletと呼ぶ．クライアントがHTTPサーバに要求を送ると，アプリケーション・サーバに渡されてサーブレットが実行され，結果がクライアントに返される．アプリケーション・サーバは一度サーバ上で起動すると，メモリ上に常駐し続け，クライアント要求ごとにスレッドを開いて処理する．

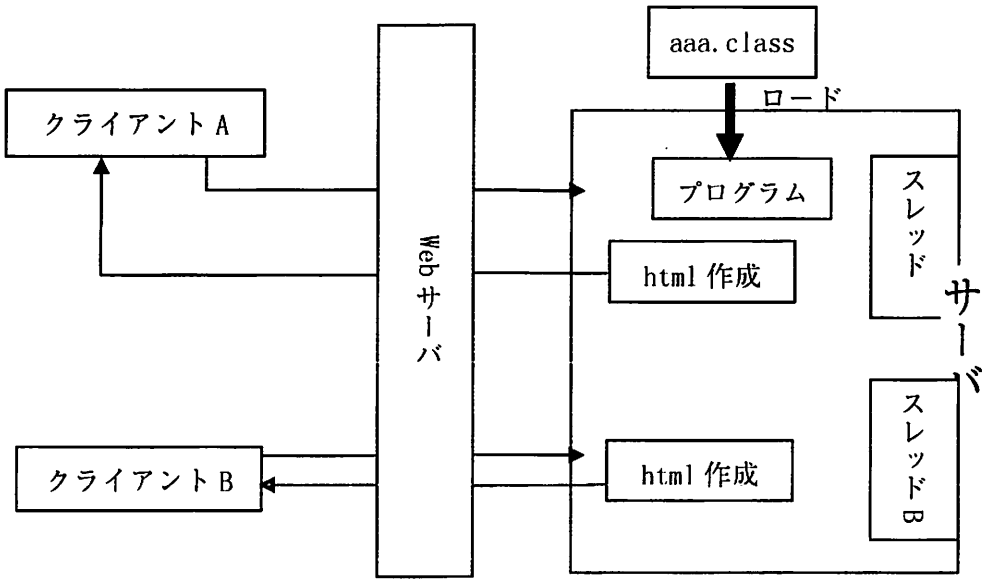


図 4.2:Servlet の動き

ServletはJavaの構文どおりに記述するために，HTMLなどの応答を出力するには，いちいちprintメソッドを記述する必要がある．HTMLなどの文書の記述を容易にするために登場したのがJSP(Java Server Pages)である．JSPも実行時には，アプリケーション・サーバの内部的にServletに変換されたものが使われることになる．

ServletやJSPによって利用されるオブジェクトはJavaBeansと呼ばれる規約に従って開発することが多くなる．JavaBeans仕様は，もともとSwingなどのGUIアプリで，再利用性を高めるための共通規格という意味合いで登場したものである．通常のJavaクラスと同じ構文だが，JavaBeans仕様に従ったモジュールを特に，Java Bean, beanと呼ぶ．WebApplicationにおけるこれらのオブジェクトの役割は，ビジネスロジックの実装，DBへの連携を専らとする．

ServletとCGIの違いは，CGIはクライアントのリクエストごとにHTTPサーバによってメモリ上に展開されて，対応する処理系が実行する．一般に，メモリロードは負荷の高い作業である．Servletはサーバにインストールし，一度メモリ上に展開されるとそのまま常駐する．クライアントのリクエストに対してはスレッドが作成されてそこで処理される．そのため，最初の一回を野増と負荷の高いメモリロードがなくなり，CGIに比べると処理が高速になる．

CGIで多く使われている言語に，文書スクリプト言語であるPerlが挙げられる．これはス

クリプト言語の完成系とも呼ばれる，高機能で完成度の高い言語だが，そもそもはUNIX上での文書処理用に開発されたものであり，ヘビーなアプリケーション開発には向かない．一方，Servletに使うJavaはオブジェクト指向の汎用言語であり，分散環境・ネットワークとの親和性が高く，高度なロジックを実装しやすい．

## 4.3 Restlet の現状

Javaを「Web2.0に対応させる」というプロジェクトにおいて現在開発されているものがRestlet APIである．このプロジェクトはRepresentational State Transfer (REST) アーキテクチャスタイルをJavaでサポートすることを目指す[7]．

Servletはサーバサイドプログラミングとして位置づけられているが，Web2.0の時代ではWebクライアントとしても動作でき，またWebサーバとしても動作を行うことができなければならないとして，Jerome Louvel氏を中心としてRestletプロジェクトが進行した．

2007年1月現在，RestletAPIはバージョンごとにクラス名などが明確に決まっておらずバラバラで，バージョンアップを頻繁に繰り返しているため参照することができない．

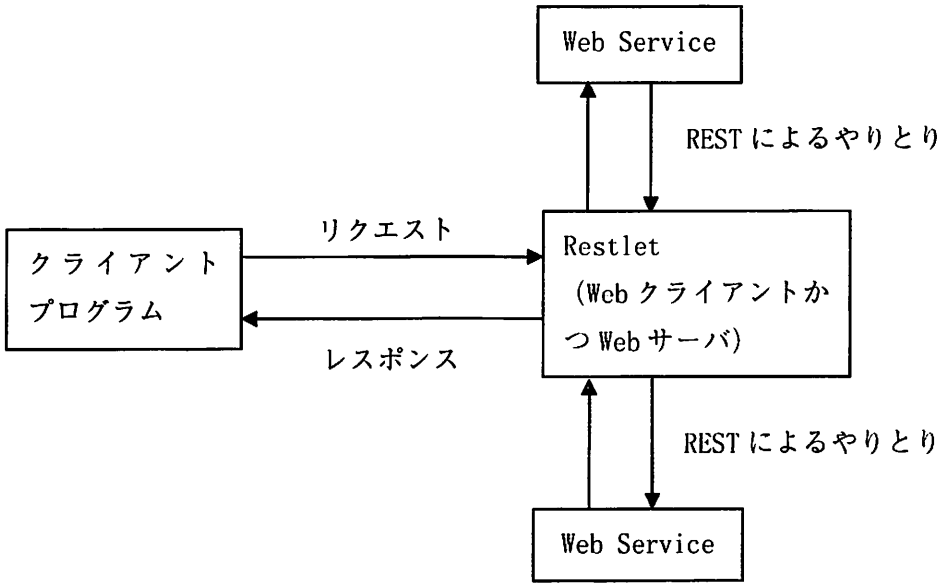


図 4.3:Restlet

Restletを利用することができれば，JavaでWeb Serviceを利用したマッシュアップサイトを構築することが容易になる．

## 4.4 Ajax (Asynchronous JavaScript and XML)

本研究では，シソーラスXMLデータと他のWeb Serviceとマッシュアップさせたweb ApplicationをAjaxの技術を利用して作成した．

従来のWeb Applicationでは，サーバにリクエストを送信後，レスポンスを新たにWebページとして受け取り，画面遷移が発生していた(図4.4)．しかし，Ajaxにより画面遷移を伴わない動的なWeb Applicationの製作が実現可能になる[8](図4.5)．例えばWeb検索に応用することで，従来は入力確定後に行っていた検索を，ユーザがキー入力をする間にバックグラウンドで行うことでリアルタイムに検索結果を表示していくといったことが可能になる．

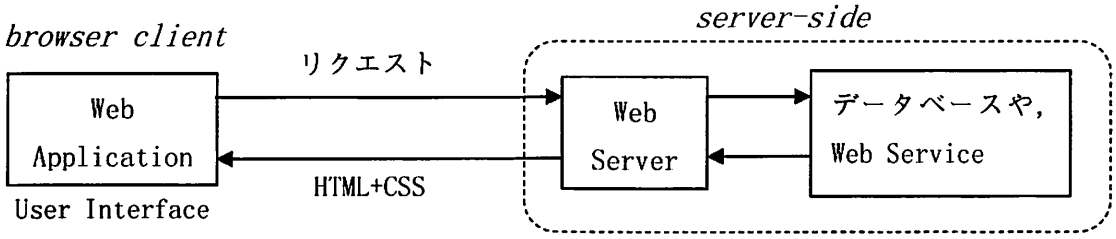


図 4.4:従来の Web Application

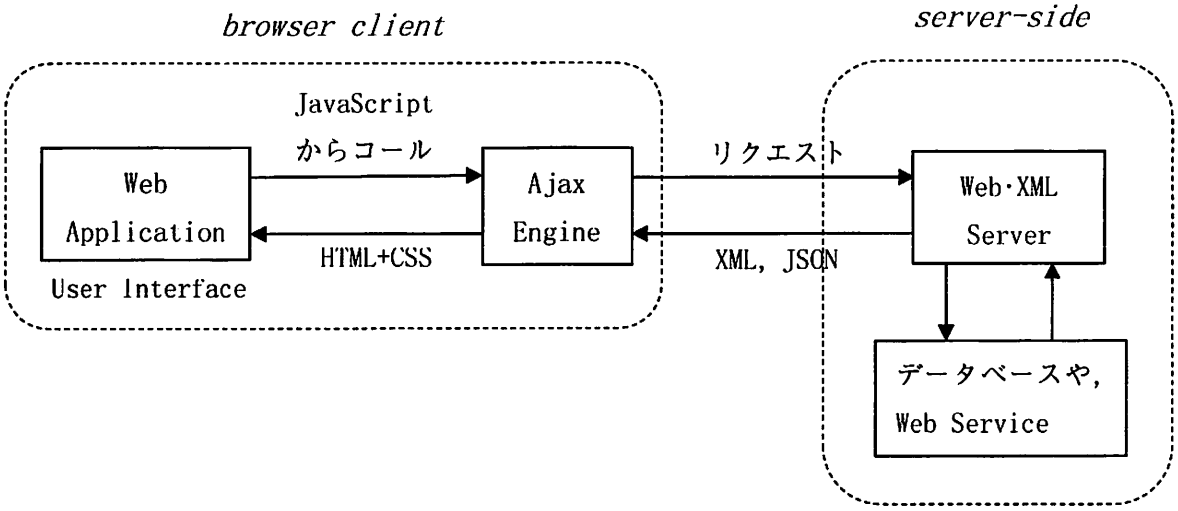


図 4.5:Ajax を利用した Web Application

Webブラウザのみで動作する（別途プラグインを要求しない），既存の技術の組み合わせであることが特徴． 技術自体はこの用語が発生する前から存在していたが，Ajaxという名前が付けられたこと，GoogleがGoogle Maps（図4.6）やGoogle Suggest（図4.7）にこの技術を利用したことで有名になり，Web Applicationの可能性を広げるものとして注目され始めた．さらに，Googleでは，デスクトップアプリケーションと遜色のないメーラーであるGmailやGoogle Calendarでも積極的にAjaxを採用し，Ajaxの実用性がGoogleのWeb Applicationを通じて世間に認知されはじめている．


図 4.6:Google Maps


図 4.7:Google Suggest

AjaxによるWebプログラミング（Web Application製作）が注目されだした背景には，この従来のページ遷移のみに頼ったWebの使い勝手の悪さに対する不満や，XML，DOMなどのWeb関連技術の標準化（Web標準），および高い機能を持ったWebブラウザの普及などが挙げられる．

また，ダウンロード型アプリケーションは，マニア層から先に広がりにくい，競合がOSメーカーとなったときに競争に負けてしまう，といった問題を抱えているため，ダウンロード型アプリケーションからWebアプリケーションに切り替える技術として，Ajaxが利用されている．

DHTMLが登場した当時は，単にお遊び要素に過ぎないと考えられていた動的ページだが，JavaScriptをより効果的に使うことで，業務や実用に耐える優秀なインターフェースを備えたアプリケーションをHTMLで作ることが可能であったという事実を世の中に知らしめたという意義をもつ技術である．

Ajaxの問題点としては，従来の技術の組み合わせであるため，それぞれの持つ問題をそのまま内包する． 例えば，各種Webブラウザ間のDHTMLの実装の違いをコードで吸収する必要がある．

実際，Ajaxを実現する技術はブラウザ間で実装に違いがあり，基幹技術であるXMLHTTPRequest実装の元となったInternet Explorerの実装の解説ではXMLHTTPRequestと

いう用語は見あたらず，ActiveXでMSXML機能を呼び出して実行する．MSXMLの実装ではバイト配列を取り出せるなど機能的な違いもある．言語としてVBScriptが使える点も異なる．

また，Ajaxアプリケーションでは動的にページの一部が書き換えられる為，デザインとコードが以前のように単純に分離できないという開発上の問題点がある．このため，現在では通常のWeb開発に比べ開発により時間がかかると言われている．

現在，数多くのAjax用アプリケーションフレームワークが開発・公開され部分的に可能になってきているが，完全ではない．現時点では成熟しつつあったWeb開発のデザインとコードの分離技術や開発体制は，Ajaxの登場によって一石が投じられている．

このような状況ではあったが，2006年現在，prototype.jsなどのAjaxフレームワークを利用することで，デザインとコードを完全に分離することが可能になっている．すなわち，デザイナーとデベロッパーの分業はAjaxを採用することで決して妨げられるわけではなく，むしろ，より完全に分業体制を行うことも可能になってきている．具体的には，HTMLタグ中の「onXXXXX」属性(XXXXXはイベント名)などに書かれるJavaScriptコードを，prototype.jsというフレームワークではタグオブジェクトに対するイベントリスナの組み込みという形で行える．すなわち，Event.observe()メソッドでイベントに対するメソッドを記述することができる．HTMLまたはXHTMLタグ内に含まれていたイベントハンドリング属性とその処理メソッドの記述(onClick="xxxmethod(xxx);"等の記述)も追い出すことができるわけである．

また，prototype.jsによって，ブラウザの互換性に関するコード（クロスブラウザ）を書くことからある程度開放されるという意義もある．また，prototype.jsは高生産性Webアプリケーションフレームワークとして認知されつつあるRuby on Railsでも採用され，Ajaxのデファクトスタンダードとして多くの開発者から認知されている． このことは，適切なAjaxフレームワークの採用で開発の高生産性を確保しつつ，リッチインターネットアプリケーション（リッチなWebクライアント）の実現という両方の要求を満たす可能性があることを示唆する．

# 第5章

# シソーラスWeb Serviceの仕様

## 5.1 シソーラスWeb Serviceの概要

本研究で使用する荻野先生のシソーラスデータはテキスト形式で書かれているため，Java(TM)2SDK,Standard Edition Version 1.5.0とMySQL Version5.0を使用し，データベースへそれらを登録した．

本研究で作成したシステムでは，リクエストが来るとRESTの仕組みを利用し，シソーラスWebServiceとなるサーバプログラムのServletを呼び出すと，該当するデータリストをXMLにしてレスポンスを返す．

リクエストの文字コードがShift_JISに限定されてしまうが，コマンドプロンプト画面や，InternetExplorerなどのウェブブラウザのアドレス欄から直接リクエストを送ることが可能である．また先述したAjaxでもリクエストを送ることが可能である．一方レスポンスは，この他のWeb Service APIから送られるXMLデータのほとんどUTF-8形式となっているため，それに合わせてUTF-8形式のXML文書を返すように設定した．

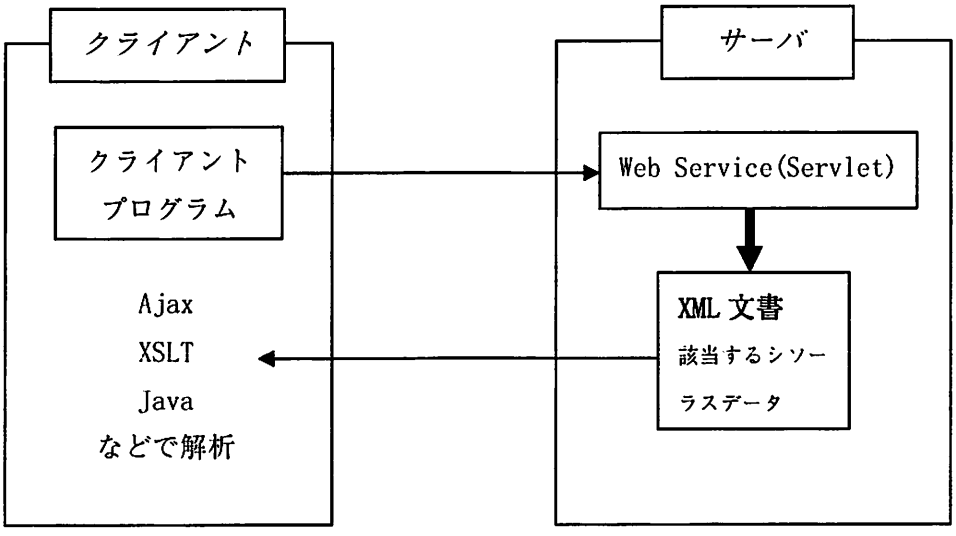


図 5.1:シソーラスWeb Serviceの流れ

### 5.1.1 使用したシソーラスの特徴

荻野氏のシソーラス[2]は以下のような特徴がある.

1. 名詞だけを収録している
   三省堂国語辞典の全部の名詞見出しを抽出し，意味（多義）を区分して，それぞれをシソーラス構成の最小単位として作成している.
2. 上位・下位語の判定
   すべての見出しについて，上位・下位関係の判定作業を行っている.

このシソーラスは以下のような形式で記載されている.

```
0000010唖（あ）<身体障害（しんたいしょうがい）
0000020アーガイル<模様(100)（もよう）
0000030アーケード(100),雁木（がんぎ）<廊下（ろうか）
0000110アーム(300)[ミシン(101)
0000195あいあい<?
0000320哀しみ(1)（かなしみ）,喜び(100)（よろこび）&哀歓（あいかん）
0140320 育毛（いくもう）<発育（はついく）＊使役
                                        .
                                        .
                                        .
```

図 5.2:荻野氏のシソーラスデータ

本研究において，データに登録する際に省略したものは，「行番号」，「あわせ呼び関係」，「動作性の名詞の特殊な記述」であり，使用したものは「上位・下位関係」，「同義語」の二つである．上記の３つを省いたデータに直し，以下の図5.3のような形式に変換しなおしてMySQLのデータベースへ登録した.

唖,null,あ,null,身体障害（しんたいしょうがい）,null
アーガイル,null,null,null,模様(100)（もよう）,null
アーケード,100,null,雁木（がんぎ）,廊下（ろうか）,null
雁木,null,がんぎ,アーケード(100),廊下（ろうか）,null
アーケード,200,null,null,覆い(2)（おおい）,null
愛人,200,あいじん,null,恋人（こいびと）,女(600)（おんな）
愛人,200,あいじん,null,null,情夫（じょうふ）
愛人,200,あいじん,null,null,色男(200)（いろおとこ）
愛人,200,あいじん,null,null,紐(300)（ひも）
愛人,200,あいじん,null,null,間夫(100)（まぶ）
愛人,200,あいじん,null,null,男(600)（おとこ）
愛人,200,あいじん,null,null,情婦（じょうふ）
愛人,200,あいじん,null,null,色女(200)（いろおんな）
.
.

図 5.3:シソーラスの MySQL への登録形式

左から「キーワード，意味番号，読み仮名，同義語，上位語，下位語」という順番になっている．同義語，上位・下位語が２つ以上ある場合は次の行に書き足していく．データがない場合は，「null」とした．キーワード，意味番号，読み仮名の３つはデータが続く限り同じものを出力する．

### 5.1.2 登録データを XML として出力するサーバプログラムの配備について

シソーラスWeb ServiceはJDBC(JavaDataBaseConnection)ドライバを利用して，クライアントからリクエストされると，ServletはMySQLを操作しデータベースにアクセスし，該当データをXML形式にしてクライアントプログラムに返す.

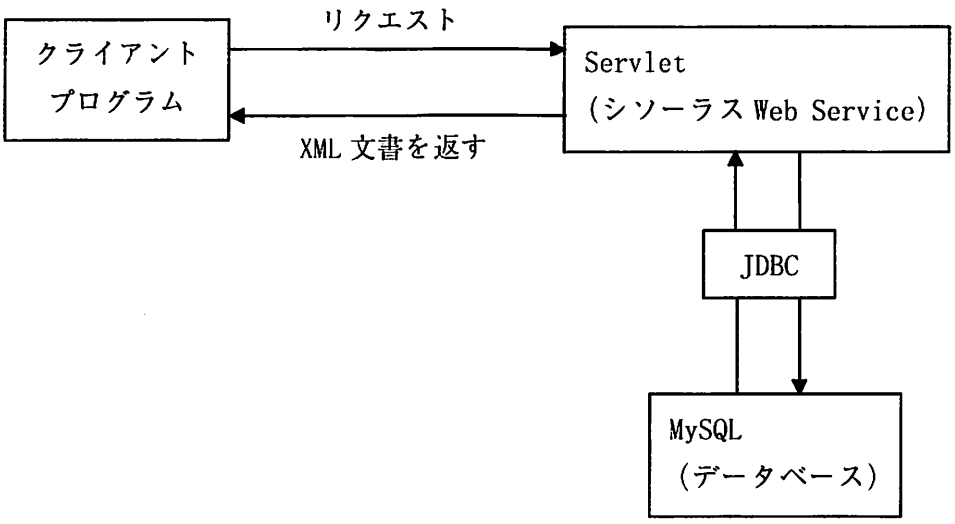


図 5.4:シソーラス Web Service の流れ

ServletはApache Tomcat Version4.1.34を使用して自分のPC上で利用できるように配備した．またデータベースとなるMySQLはMySQL Version5.0を使用した．以下に配備方法と，環境設定方法を記す．また6章で説明する一般的に提供されている各Web Service APIが使えるようになるまでについても説明する．なお，Javaのプログラムは既にコンパイルできるものとする．

Apache Tomcat Version4.1.34をダウンロードした後，インストールを行う．Servletプログラムをコンパイルするにはコマンドプロンプト画面で以下のCLASSPATHを指定して行う．

```
＊＊＞javac -classpath "CATALINA_HOME\commonlib\servlet.jar" ***.java
```

図 5.5:Servlet プログラムのコンパイル

ただし，CATALINA_HOMEとは「Tomcat 4.1」ディレクトリ(インストールディレクトリ)を示す．このようにコンパイル際にいちいち指定するのは面倒でかつわかりにくいので，あらかじめCLASSPATHを通す．

表 5.1:CLASSPATHへ通すファイル

| ファイル | 説明 |
|---|---|
| CATALINA_HOME\common\lib\servlet.jar | Servletプログラムのコンパイル |
| CATALINA_HOME\common\lib\xalan.jar | ServletからXMLを作成するために必要 |
| CATALINA_HOME\common\lib\mysql-connector-java-5.0.4-bin.jar | JDBCドライバ．プログラムからMySQLを操作する際に必要 |

Servletプログラムはコンパイルできるようになったら，以下の手順を行っていく．

1. 保管フォルダの作成(./WEB-INF/classes)
2. web.xmlの設定(ServletのURLマッピング)
3. server.xmlの設定(Servletのフォルダ登録)

まず，保管フォルダを作成する．Servletを動作させるためには決められたフォルダ構造を作らなければならないので，「C:\Java\DocRoot\WEB-INF\classes」のように設定した．このとき，「./WEB-INF/classes」の構造が必須部分である．Servletはclassesフォルダに保管しなければならない．

次にWEB-INFフォルダの配下にweb.xmlを作成してServletにURLをマッピングさせる．本研究で作成したシソーラスWeb Serviceプログラム(thesaurus_XML.java)を配備した際のweb.xmlは以下のようになっている．

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE web-app
        PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
        <servlet>
                <servlet-name>DB4</servlet-name>
                <servlet-class>thesaurus_XML</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>DB4</servlet-name>
                <url-pattern>/db/thesaurus_XML</url-pattern>
        </servlet-mapping>
</web-app>
```

図 5.6:web.xml の内容

次に，CATALINA_HOME\conf内にあるserver.xmlを編集する．269行目に以下の記述を挿入する．

```
<Context path="/webAppl" docBase="C:/Java/DocRoot" debug="0"
                        reloadable="true" crossContext="true" />
```

図 5.7:server.xml に挿入する記述

以上でTomcat起動中はServletが動作するようになる．本プログラムはMySQLのデータベースを参照するので，MySQLも設定しなければならない．

MySQL Version5.0はインストール時にパスワードを設定する．ユーザ名は「root」とした．5.1.1で述べた，作成したシソーラスをレコードに登録する．まずデータベースとテーブルを作る．作成するためのコマンドを書いた実行文を以下に示す．ちなみに作成したシソーラスデータは「My_thesaurus.txt」である．

```
create database thesaurus
character set sjis
collate sjis_japanese_ci;

create table mt(word varchar(60),wordNumber varchar(60),yomi varchar(60)
,same varchar(60),upper varchar(60),lower varchar(60))
character set sjis
collate sjis_japanese_ci;

load data infile('C:/My_thesaurus.txt') into table mt
fields terminated by ',' lines terminated by '\r\n';
```

図 5.8:MySQL でのデータ登録手順

## 5.2　入力仕様と出力仕様

シソーラスWebServiceを呼び出す際，Shift_JISにエンコードさせる必要がある．検索されるキーワードは荻野氏シソーラスに登録されている単語の意味番号と読み仮名を取ったものである．に以下のようにクエリ文字列として与える．

```
http://localhost:8080/webApp1/db/thesaurus_XML?word=[検索キーワード]
```

図 5.9:シソーラス Web Service の呼び出し

リクエストを受け取ったServlet（thesaurus_XML.java）は，検索キーワードがシソーラスデータベースにある場合と，ない場合の両方においてそれぞれXMLを返す．以下の２つは「劇場」（データベースにある単語）と，「ABC」（データベースにない単語）で検索した結果である．データベースにない単語が検索された場合のXMLの内容はどれも同じである．またServletから出力されるXMLは，以下の構造を持つ．

表5.2　シソーラスWeb ServiceのXMLのタグと対応するデータ

| タグの名前 | 該当するデータ |
|---|---|
| data | — |
| member | — |
| word | 検索キーワード |
| wordNumber | 意味番号 |
| yomi | 漢字の場合，読み仮名 |
| same_word | キーワードの同義語 |
| upper_word | キーワードの上位語 |
| lower_word | キーワードの下位語 |

```
<?xml version="1.0" encoding="UTF-8" ?>
<data>
  <member>
    <word>劇場</word>
    <wordNumber>null</wordNumber>
    <yomi>げきじょう</yomi>
    <same_word>シアター</same_word>
    <upper_word>建物（たてもの）</upper_word>
    <lower_word>アングラ(102)</lower_word>
    <lower_word>アンダーグラウンド</lower_word>
    <lower_word>座(10400)（ざ）</lower_word>
  </member>
</data>
```

図 5.10:シソーラス WebService の XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<data>
  <member>
    <word>error</word>
  </member>
</data>
```

図 5.11:シソーラスに該当しなかった場合の XML

## 5.3 使用方法，問題点

HTTPのGETメソッド，またはPOSTメソッドのどちらからでも呼び出すことができる．HTMLのフォームからこれらのメソッドを使って呼び出す場合，ASCII文字に変換して通信を行うため期待したデータは取得できない．文字エンコードがShift_JISとなっていれば特にエンコードさせる必要はない．

その他「人」や「場所」など広義の単語は下位語を多く含むため動作に時間がかかる可能性がある．また出力されるXMLはUTF-8で書かれているので，コマンドプロンプト画面上で何らかのプログラムによりそのまま表示させるといった事はできない．

また問題点ではないが，Ajaxで扱う場合に起きる「クロスドメインの制約」について説明する．

AjaxではXMLHttpRequestオブジェクトを扱いXMLを操作するが，セキュリティ上，Web Applicationと違うドメイン配下にあるXML等を操作することができない．よってFirefox，Operaといったブラウザでは Ajaxを利用して作成したマッシュアップサイトは動作しない．InternetExplorer5.0以降ではXMLHttpRequestオブジェクトは使用していないので問題なく動作する．

FirefoxやOperaといったXMLHttpRequestオブジェクトを使用しているブラウザでAjaxの動作（リソースの取得）を実現させるためには，サーバを立ててあらかじめXMLを解析するか，もしくはWeb Service側がXMLではなくJSON(JavaScript)といったものを提供する必要がある[9]．

本研究で作成したシソーラスWebServiceはJSONを提供していないので，Ajaxのみで構築されたサイトには扱いづらいものになってしまった．

# 第6章

# マッシュアップ Web Application の概要

## 6.1 各 Web Service のレスポンス XML の構造

各Web Serviceから取得できるXMLのタグの内容について述べる．サイトサムネイル作成API以外は，XMLを受け取る．各XMLはタグ内に必要な情報が含まれているが，Ajax（JavaScript）でこれを解析し，ブラウザ上に表示できるように加工する．

本研究で作成したWeb Applicationは，REST，もしくはXML-RPCで各Web Serviceをコールした後にXMLを取得する．各Web Serviceから取得できるXMLは以下の図のとおりである．

```
<ResultSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:yahoo:jp:srch" xsi:schemaLocation="urn:yahoo:jp:srch
http://api.search.yahoo.co.jp/WebSearchService/V1/WebSearchResponse.xsd"
totalResultsAvailable="データ内のマッチしたクエリー数"
totalResultsReturned="返却され，かつマッチしたクエリーの数"
firstResultPosition="全検索結果の最初のポジション">
<Result>
  <Title>ページのタイトル</Title>
  <Summary>ページに関連するテキストサマリー</Summary>
  <Url>ページの URL/</Url>
  <ClickUrl>ページのリンク URL</ClickUrl>
  <ModificationDate>ページが最後に修正された日付．UNIX タイムスタンプフォーマット</ModificationDate>
  <MimeType>ページの MIME タイプ</MimeType>
  <Cache>
    <Url>キャシュ結果の URL</Url>
    <Size>サイズは byte</Size>
  </Cache>
</Result>
</ResultSet>
```

図 6.1:Yahoo!検索 Web サービスのレスポンス XML

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>wordlist</name>
            <value>
              <array>
                <data>
                  <value>
                    <struct>
                      <member>
                        <name>word</name>
                        <value><string>連想される単語</string></value>
                      </member>
                    </struct>
                  </value>
                </data>
              </array>
            </value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodresponse>
```

図 6.2:はてなダイアリーキーワード連想語 API のレスポンス XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<results>
<result>
  <language>言語名</language>
  <id>Wikipedia 内の管理コード</id>
  <url>Wikipedia にリンクする際の URL</url>
  <title>キーワードのタイトル</title>
  <body>本文のダイジェスト</body>
  <length>本文の長さ</length>
  <redirect>別キーワードへのリダイレクト。ある場合は 1, ない場合は 0。</redirect>
  <strict>そのキーワードと完全一致する場合には 1。しない場合は 0。</strict>
  <datetime>更新日付</datetime>
</result>
```

図 6.3:Simple API Wikipedia のレスポンス XML

```
<?xml version="1.0" encoding="utf-8"?>
<!-- generator="Technorati API version 1.0 /tag" -->
<!DOCTYPE tapi PUBLIC "-//Technorati, Inc.//DTD TAPI 0.02//EN"
"http://api.technorati.com/dtd/tapi-002.xml">
<tapi version="1.0">
<document>
<result>
  <query>検索タグ</query>
  <postsmatched>検索タグを付けているブログ記事</postsmatched>
  <blogsmatched>その検索タグをつけているブログ数</blogsmatched>
  <start>返却結果の先頭位置</start>
  <limit>取得するブログの数</limit>
  <querytime>検索にかかった秒数</querytime>
</result>
<item>
  <weblog>
    <name>ブログタイトル</name>
    <url>ブログの URL</url>
    <rssurl>ブログの RSS の URL</rssurl>
    <atomurl>ブログの Atom の URL</atomurl>
    <inboundlinks>そのブログがリンクしているブログ総数</inboundlinks>
    <inboundblogs>他ブログからリンクされている数</inboundblogs>
    <rank>ブログのランキング</rank>
    <lastupdate>最近更新された年月日、時間</lastupdate>
    <hasphoto>サイトサムネイル画像</hasphoto>
  </weblog>
  <title>記事タイトル</title>
  <excerpt>記事の本文</excerpt>
  <created>記事の作成日時</created>
  <postupdate>記事の投稿日時</postupdate>
  <permalink>記事のパーマリンク URL</permalink>
</item>
</document>
</tapi>
```

図 6.4:Technorati API(tag)のレスポンス XML

## 6.2 作成したWeb Applicationの概要

本研究では，第3章で述べた各Web Service APIを利用した検索システムを作成した．Web Service APIからXMLを取得し，Ajax（JavaScript）を利用してそのXMLを解析し，結果を表示する．キーワードからシソーラス（同義語，上位語，下位語）検索，キーワードから連想される単語，Wikipediaでの解説，Yahoo検索エンジンを利用したウェブ検索，Technoratiタグ検索エンジンを利用したブログ検索を個別に行うことができる．データを取得次第，結果が表示される点と自分の欲しい情報を手前に表示させることができるのが特徴である．

キーワードを入力し，取得したい項目にチェックするとApplicationは該当するWeb Serviceをコールし，XMLを取得．結果が表示される．チェックを外すとその項目は消去される．作成したプログラムソースを付録として付加する．

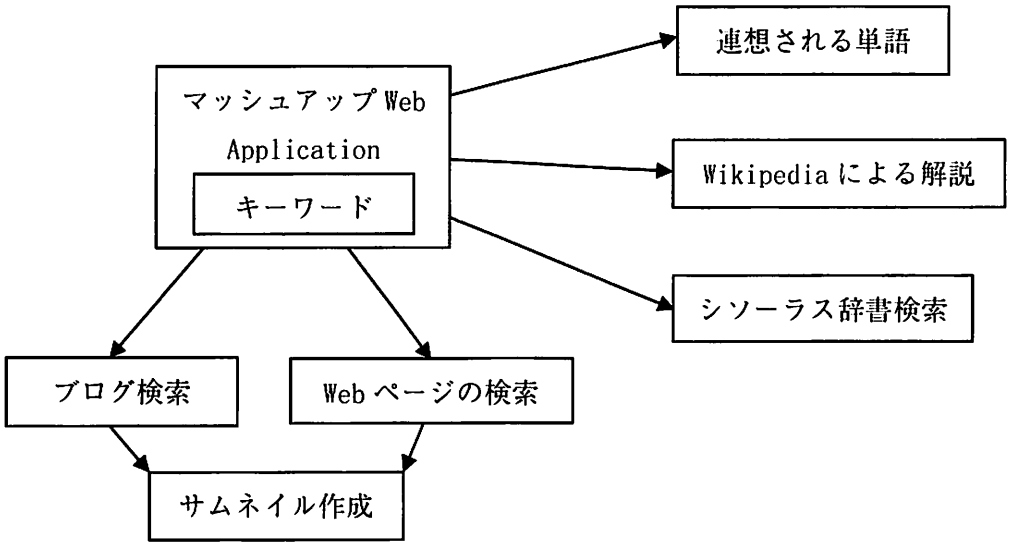


図 6.5:Ajaxを利用したマッシュアップ Web Application

## 6.3 マッシュアップWeb Applicationの使用方法

Local_Application.htmlファイルを実行すると，図6.6がInternet Explorerに表示される．キーワードを入力したら，シソーラス辞書検索，はてな連想語，Wikipediaにチェックをする．検索を行う場合は，Yahoo検索エンジンかブログ・タグ検索にチェックを行い，Yahooの場合は言語設定のチェックも行う．

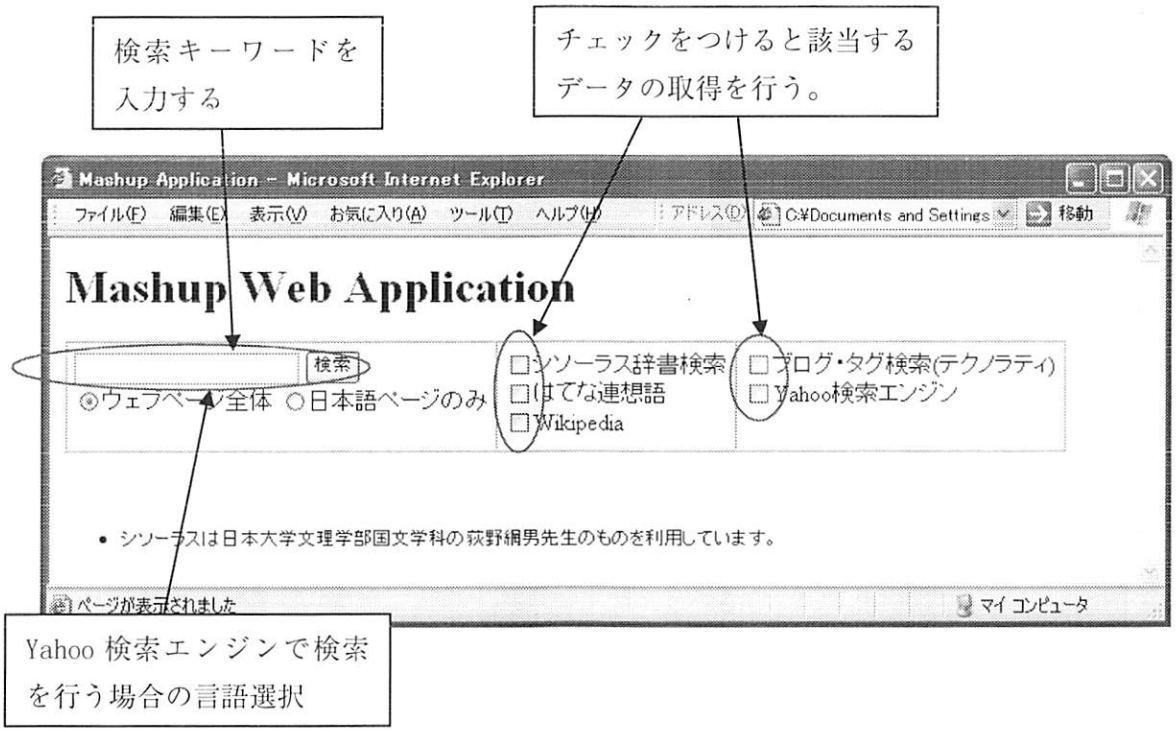


図 6.6:プログラムの実行画面

# 第7章

# 実験

## 7.1 ブラウザでのXML取得結果

実際にクライアントからシソーラスWebServiceにアクセスしてXMLを取得できるかどうか実験した．今回の実験では，クライアントにWebブラウザを使用する．また検索キーワードは前項と同様に「劇場」と「abc」として実験を行った．また作成したプログラムソースを付録として付加する．

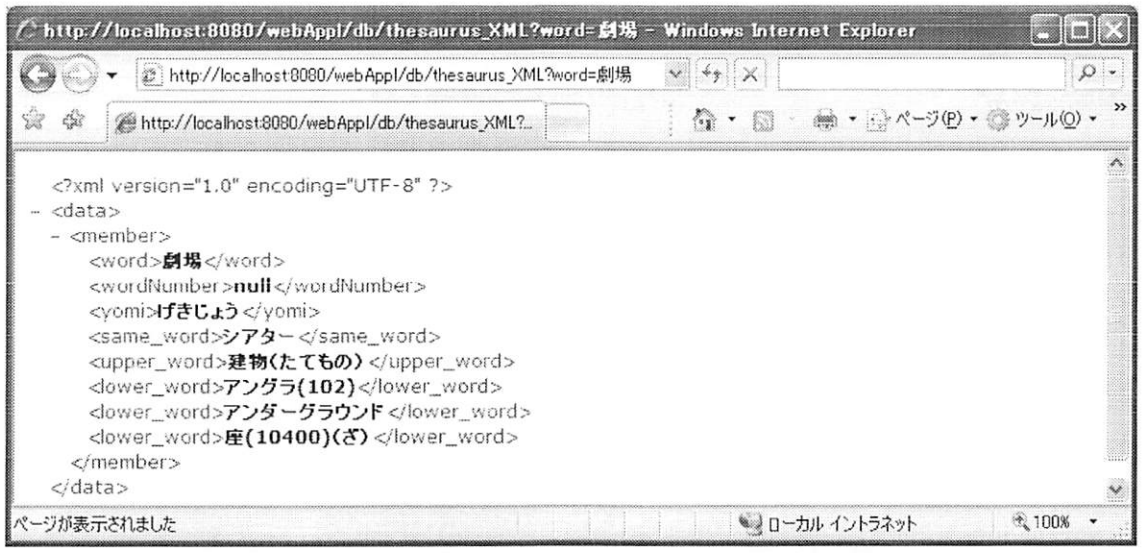


図 7.1:Webブラウザからコールする様子

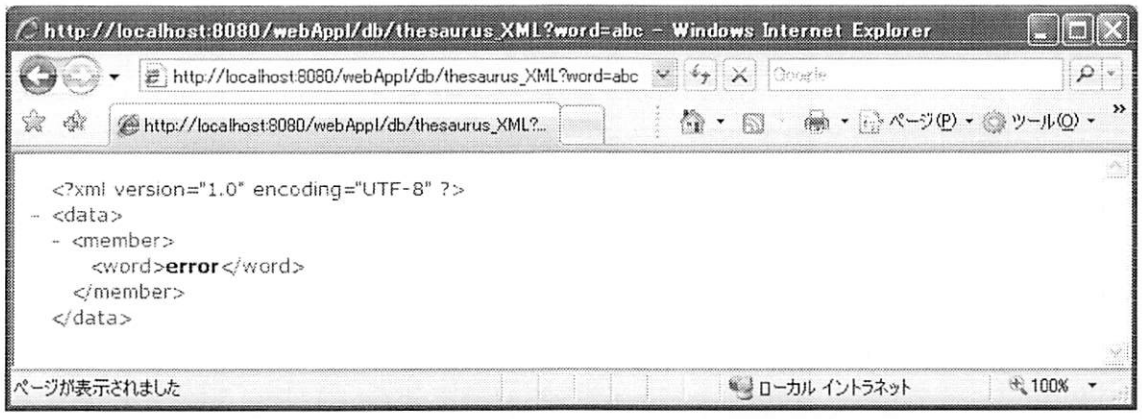


図 7.2:「abc」で検索した結果

## 7.2 マッシュアップ Web Application の使用結果

実際に作成したWeb Applicationを使用して検索を行い，検索結果を取得してみる．検索キーワードとして，「REST」，「シソーラス」，「野球」を用いた．以下に検索結果を示す．


図 7.3:「REST」で検索した結果


図 7.4:「シソーラス」で検索した結果

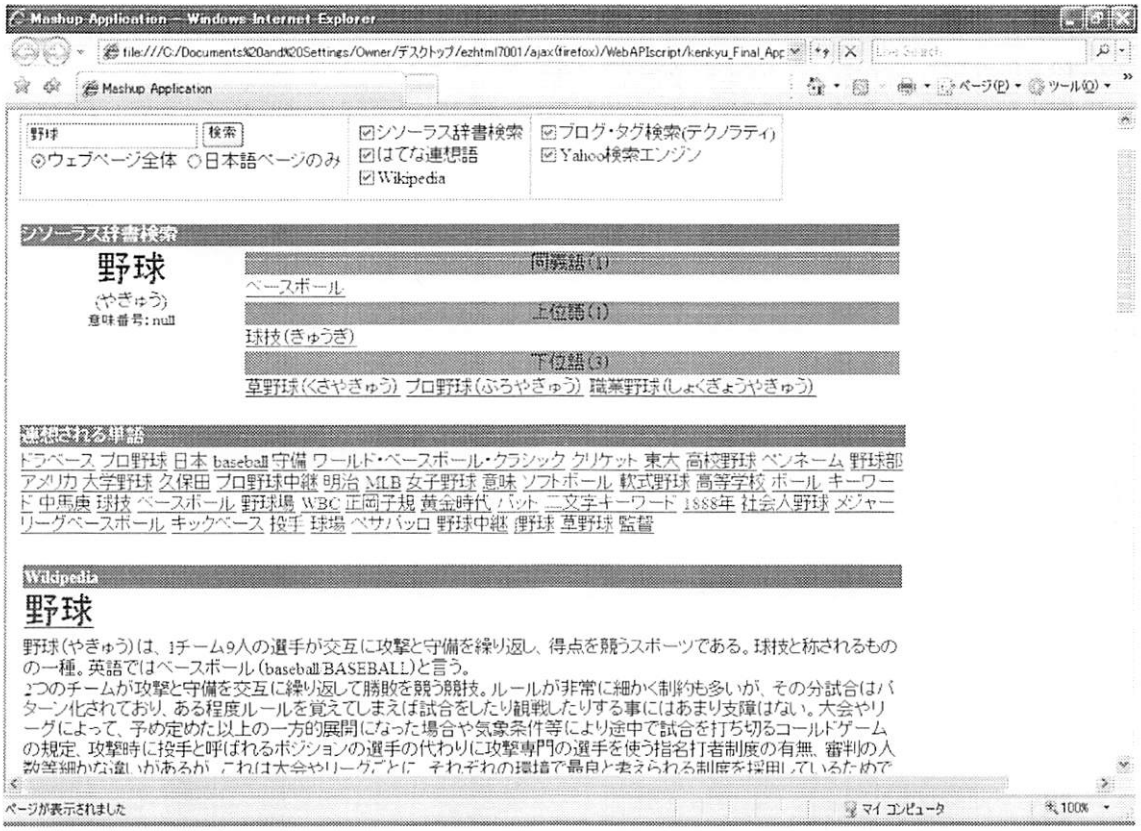


図 7.5:「野球」で検索した結果

## 7.3 評価

マッシュアップ Web Applicationから問題なく使用できたことからシソーラスWeb ServiceはServletでも実装できていることがわかる．またマッシュアップ Web ApplicationからシソーラスWeb ServiceをRESTで呼び出すことができた．データベースに登録されている単語であれば，内部に該当データが含まれたXMLを取得することができ，また登録されていない単語であればエラーのXMLを取得することができている．

作成したWeb ApplicationについてはXMLを取得した後の，結果を表示する部分で長い間XMLが取得できないという問題が起こった．

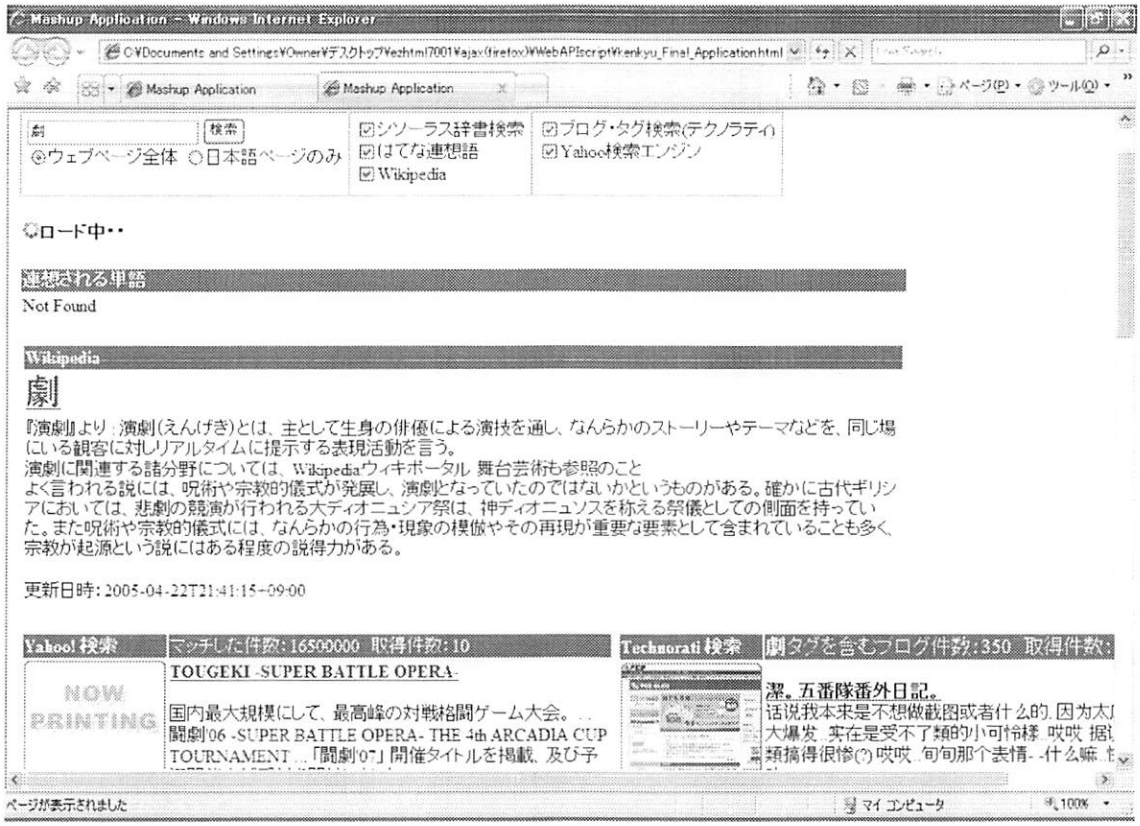


図 7.6:いつまでも結果が表示されない状態

このApplicationは非同期通信を利用しているので，XMLを取得次第，各データテーブルを表示する．しかし，同時に５つのXMLを取得しようとすると全てちゃんと表示されなくなる．また，Yahoo!検索WebサービスとTechnorati APIは一日の取得検索件数に制限があるので，実際に使用していくのは困難である．

## 7.4 今後の課題

本研究で作成したシソーラスWeb Serviceではデータを XMLにして返す．しかしAjaxで扱いたい場合はセキュリティの関係により，取得できるXMLに制限があるため，JSON形式で返す必要がある．今後はこれを踏まえてレスポンスのフォーマットをXMLかJSONのどちらかをURLを操作して選択できるようにしたい．

Web Applicationは複数の結果が一度にうまく表示できてないという問題点があるので，これを改善したい．また工夫として，リンクをクリックするとテキストフォームにフォーカスをあわせ，さらにそのリンクテキストが表示されるようにしたい．

また検索キーワードに対して，各検索結果を表示させるだけなのでWeb Serviceのデータを他のWeb Serviceに利用するような連携したApplicationを作成したい．

# 第8章

# 結論

本研究では，既存の荻野シソーラスをWebを通じて誰でも独自のサイトなどで扱えるシソーラスWeb Serviceを作成し，そのデータ確認と応用のためAjaxによるマッシュアップWeb Applicationを作成した．今後シソーラスWeb ServiceをAjaxでの活用を考えてJSON形式でデータを提供できるようにしたい．

# 第9章

# 謝辞

本研究の遂行および論文の作成において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，本研究を進めるに当たり助言，協力をいただきました岩崎 唯史 教官(茨城大学システム工学科)，佐々木 稔 教官（茨城大学情報工学科)，同研究室の直江 宗紀 氏（茨城大学システム工学科)，神津 健太 氏（茨城大学システム工学科)，佐々木研究室の鈴木 朋央 氏（茨城大学情報工学科)，田島 勇樹 氏（茨城大学情報工学科)，加藤 友宏 氏（茨城大学情報工学科）に深く感謝致します．

# 参考文献

[1] 「Web2.0 BOOK」, 小川浩, 後藤康成, インプレスジャパン(2006)

[2] "荻野綱男の研究室",
http://www.chs.nihon-u.ac.jp/jp_dpt/ogino/

[3]「独習 XML 第2版」, 日向俊二, 翔泳社(2004)

[4]"XML-RPC 仕様書",
http://lowlife.jp/yasusii/stories/9.html

[5]"SOAP,WSDL,REST-Web API の基礎技術を学ぶ", 山田祥寛, 日経ソフトウェア 2006 年 5 月号, pp78-85, 日経ＢＰ社(2006)

[6]「Java 完全マスターブック」,高田美樹, 技術評論社

[7]"Restlet",
http://www.restlet.org/

[8] "Ajax: A New Approach to Web Applications",
http://www.adaptivepath.com/publications/essays/archives/000385.php

[9]"Introducing JSON",
http://www.json.org/

# 付録1
# プログラムソースリスト

### thesaurus_XML.java

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class thesaurus_XML extends HttpServlet{

  public void doGet(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
    mysql(req,res);
  }

  public void doPost(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
    mysql(req,res);
  }

  public void mysql(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
    // 要求文字コードのセット
    req.setCharacterEncoding("Shift_JIS");
    // 応答文字コードのセット
    res.setContentType("text/html; charset=UTF-8");

    String query = req.getParameter("word");
    String SQLquery = "";
    String check="";
    // 取得したメッセージをそのまま出力します
    res.setContentType("text/xml; charset=UTF-8");
    // 出力ストリームの取得
```

```
    PrintWriter out = res.getWriter();

    try {
      // 1. JDBC Driver の登録
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      // 2. データベースへの接続
      Connection con = DriverManager.getConnection(

"jdbc:mysql://localhost/Thesaulus?useUnicode=true&characterEncoding=sjis",
"root", "nnnyami");

      // 3. SQL ステートメント・オブジェクトの作成
      Statement stmt = con.createStatement();
      // 4. SQL ステートメントの発行
      SQLquery = "SELECT * FROM mt WHERE word LIKE '"+query+"'";
      ResultSet rs = stmt.executeQuery(SQLquery);

    if(rs.next()){
      out.println("<?xml version=\"1.0\" encoding=\"UTF-8\" ?>");
      out.println("<data>");
      out.println("<member>");
      out.println("<word>" + rs.getString("word") +"</word>");
      out.println("<wordNumber>" + rs.getString("wordNumber") +"</wordNumber>");
      out.println("<yomi>" + rs.getString("yomi") +"</yomi>");

      rs.beforeFirst();
      // 5. 結果の出力
      while (rs.next()) {
        check = rs.getString("same");
        if(!check.equalsIgnoreCase("null"))
          out.println("<same_word>"+check+"</same_word>");
      }

      rs.beforeFirst();
      while (rs.next()) {
        check = rs.getString("upper");
```

```
        if(!check.equalsIgnoreCase("null"))
          out.println("<upper_word>"+check+"</upper_word>");

      }

      rs.beforeFirst();
      while (rs.next()) {
        check = rs.getString("lower");
        if(!check.equalsIgnoreCase("null"))
          out.println("<lower_word>"+check+"</lower_word>");
      }

    }else{
      out.println("<?xml version=\"1.0\" encoding=\"UTF-8\" ?>");
      out.println("<data>");
      out.println("<member>");
      out.println("<word>error</word>");

    }
      out.println("</member>");
      out.println("</data>");

      // 6. データベースのクローズ
      rs.close();
      stmt.close();
      con.close();

      } catch (SQLException e1) {
        System.out.println(
          "SQLException: " + e1.getMessage());
        System.out.println(
          "    SQLState: " + e1.getSQLState());
        System.out.println(
          " VendorError: " + e1.getErrorCode());
      } catch (Exception e2) {
        System.out.println(
```

```
                "Exception: " + e2.getMessage());
        }
    }
}
```

# 付録2
# プログラムソースリスト

## Local_Application.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <meta http-equiv="Content-Script-Type" content="text/javascript">
  <title>マッシュアップ Web Application(ローカル専用)</title>
<style type="text/css">
<!--
  a:hover{
    color: #FFFFFF;
    background-color: #FF6633;
  }
  .test{
    vertical-align: top;
  }
 -->
</style>
<script type="text/javascript">
var thesaurus;
var hatena_word;
var language;
var google;
var yahoo;
var wikipedia;
var technorati;
var SisodivFlag = true;
var rensoudivFlag = true;
var  showDivdivFlag = true;
var tldivFlag = true;
```

```
var t2divFlag = true;
timerID=0;

// スタート部分
function init(){
  checkBoolean();

  // シソーラス
  if(thesaurus)
    thesaurusCall(document.ajaxForm.word.value);

  // はてな連想語
  if(hatena_word)
    Imaginate(document.ajaxForm.word.value);

  // Yahoo検索
  if(yahoo)
    YahooCall(document.ajaxForm.word.value);

  // タグ検索
  if(technorati)
    TechnoratiCall(document.ajaxForm.word.value);

  // Wikipedia
  if(wikipedia)
    WikipediaCall(document.ajaxForm.word.value);
}

// チェックがついてるかどうか
function checkBoolean(){
  language = document.ajaxForm.elements[2].checked;
  thesaurus = document.check.thesaurus.checked;
  hatena_word = document.check.hatena_word.checked;
  wikipedia = document.check.wiki.checked;
  technorati = document.check.elements[3].checked;
  yahoo = document.check.elements[4].checked;
```

```
  return thesaurus, hatena_word, language, technorati, yahoo, wikipedia;
}

/*****************************************************************************

            シソーラスWebサービス

*****************************************************************************/

// シソーラス呼び出し部分
function thesaurusCall(content){
  checkBoolean();
  if(thesaurus){
  if(SisodivFlag){
    var dtag = document.getElementById("first");
    var div = document.createElement("div");
    div.id = "siso";
    dtag.appendChild(div);
    SisodivFlag = false;
  }
    if(content==""){
      createTable("シソーラス辞書検索","siso");
      return;
    }else{
      content = 'http://nl6.dse.ibaraki.ac.jp/webAppl/db/thesaurus_XML?word=' +
content;
      ThesaurusXML(content);
    }
  }else{
    del('siso');
    SisodivFlag=true;
  }
}
```

```
// シソーラスのXMLデータ，シソーラスXMLのHTTP通信オブジェクトを取得
function ThesaurusXML(fName){
  httpObj = createXMLHttpRequest();
  httpObj.onreadystatechange = displayData; // POSTのレスポンスをで行う
  if (httpObj){
    httpObj.open("GET",fName,true);
    httpObj.send(null);
  }
}

// シソーラスの表示関数
function displayData(){
  createTable("ロード中・・","siso",true);
  if((httpObj.readyState == 4) && (httpObj.status == 200)){
    getThesaurusData(httpObj.responseXML);     // レスポンスを取得する
  }
}

// シソーラスのXMLを表示
function getThesaurusData(xmlData){
  // responseXMLプロパティ
  // xmlDataにXMLデータがDOMオブジェクトとして格納されるので
  // DOMのメソッドが使用可能になる.
  // シソーラスのXMLは「thesaurus_XML.java」を使用する.

  wordListTags = xmlData.getElementsByTagName("word");
  wordLen = wordListTags.length;
  wordNumberListTags = xmlData.getElementsByTagName("wordNumber");
  wordNumberLen = wordNumberListTags.length;

  sames = new Array();
  uppers = new Array();
  lowers = new Array();

  var siso = document.getElementsByName('siso')[0];
  // 以前表示していたものをクリアする
```

```
if(siso.hasChildNodes()){
  for(var i=0;i<siso.children.length;i++){
    siso.removeChild(siso.children[i]);
  }
}
deleteTable4("siso");

// ワードナンバータグがあれば処理を行う
if(wordNumberLen!=0){
  yomiListTags = xmlData.getElementsByTagName("yomi");
  sameListTags = xmlData.getElementsByTagName("same_word");
  sameLen = sameListTags.length;
  upperListTags = xmlData.getElementsByTagName("upper_word");
  upperLen = upperListTags.length;
  lowerListTags = xmlData.getElementsByTagName("lower_word");
  lowerLen = lowerListTags.length;

  resultText = "";
  samText = "";
  uppText = "";
  lowText = "";
  sameCount =0;
  upperCount=0;
  lowerCount=0;

  wordText = wordListTags[0].childNodes[0].nodeValue;
  wordText1 = "(" + yomiListTags[0].childNodes[0].nodeValue + ")";
  wordText2 = "<font size='-1'>"+"  意味番
号:"+wordNumberListTags[0].childNodes[0].nodeValue + "</font><br>";

  if(sameLen!=0)
    samText = findSame(sameLen);
  if(upperLen!=0)
    uppText = findUpper(upperLen);
  if(lowerLen!=0)
    lowText = findLower(lowerLen);
```

```
// 表示文字列
resultText  = "<table border='0' width='800'>";
resultText += "<tr><td bgcolor='#3366FF' colspan='2'>";
resultText += "<b><font color='#FFFFFF'>シソーラス辞書検索</font></b>";
resultText += "</td></tr>";

resultText += "<tr>";
resultText += "<td rowspan='7' width='200' align='center' valign='top' nowrap>";
resultText += "<font size='6'>"+wordText+"</font>";
resultText += "<br>"+wordText1;
resultText += "<br>"+wordText2;
resultText += "</td></tr>";

resultText += "<tr><td align='center' bgcolor='#00CC99' width='600'>";
resultText += "同義語"+" ("+sameCount+") ";
resultText += "</td></tr>";
resultText += "<tr><td id='addsam'>";
  if(sameLen>20){
    for(var i=0;i<20;i++)
      resultText += sames[i]+" ";
    resultText += "<br>";
    resultText += "<a href='JavaScript:add_sameWord(1)'>"+"・・さらに表示";
    resultText += "</a>";
  }else{
    for(var i=0;i<sameLen;i++)
      resultText += sames[i]+" ";
  }
resultText += "</td></tr>";

resultText += "<tr><td align='center' bgcolor='#00CC99'>";
resultText += "上位語"+" ("+upperCount+") ";
resultText += "</td></tr>";
resultText += "<tr><td id='addupp'>";
  if(upperLen>20){
    for(var i=0;i<20;i++)
```

```
        resultText += uppers[i]+" ";
      resultText += "<br>";
      resultText += "<a href='JavaScript:add_upperWord(1)'>"+"・・さらに表示";
      resultText += "</a>";
    }else{
      for(var i=0;i<upperLen;i++)
        resultText += uppers[i]+" ";
    }
  resultText += "</td></tr>";

  resultText += "<tr><td align='center' bgcolor='#00CC99'>";
  resultText += "下位語"+" ("+lowerCount+") ";
  resultText += "</td></tr>";
  resultText += "<tr><td id='addlow'>";
    if(lowerLen>20){
      for(var i=0;i<20;i++)
        resultText += lowers[i]+" ";
      resultText += "<br>";
      resultText += "<a href='JavaScript:add_lowerWord(1)'>"+"・・さらに表示";
      resultText += "</a>";
    }else{
      for(var i=0;i<lowerLen;i++)
        resultText += lowers[i]+" ";
    }
  resultText += "</td></tr>";
  resultText += "</table><br>";
  siso.innerHTML = resultText;
}else{
  resultText  = "<table border='0' width='800'>";
  resultText += "<tr><td bgcolor='#3366FF' align='left' nowrap>";
  resultText += "<b><font color='#FFFFFF'>シソーラス辞書検索</font></b>";
  resultText += "</td></tr>";
  resultText += "<tr><td align='left' nowrap>";
  resultText += "Not Found";
  resultText += "</td></tr>";
  resultText += "</table><br>";
```

```
    siso.innerHTML = resultText;
  }
}

// 同義語探索部分
function findSame(baseValueLen){
checkBoolean();
  for(var i=0;i<baseValueLen;i++){
    sam = sameListTags[i].childNodes[0].nodeValue;
    if(sam!=""){
      sameCount++;
      var ss = sam.indexOf("(");
      if(ss>=0)
        sam = sam.substring(0,ss);
      samText += sam+"<br>";
      sames[i]=googleCall(sam);
    }else{
      samText += "";
    }
  }
  return samText;
}

// 上位語探索部分
function findUpper(baseValueLen){
checkBoolean();
  for(var i=0;i<baseValueLen;i++){
    upp = upperListTags[i].childNodes[0].nodeValue;
    if(upp!=""){
      upperCount++;
      var ss = upp.indexOf("(");
      if(ss>=0)
        upp = upp.substring(0,ss);
      uppText += upp+"<br>";
      uppers[i]=googleCall(upp);
    }else{
```

```
      uppText += "";
    }
  }
  return uppText;
}

// 下位語探索部分
function findLower(baseValueLen){
checkBoolean();
  for(var i=0;i<baseValueLen;i++){
    low = lowerListTags[i].childNodes[0].nodeValue;
      if(low!=""){
        lowerCount++;
        var ss = low.indexOf("(");
        if(ss>=0)
          low = low.substring(0,ss);
        lowText += low+"<br>";
        lowers[i]=googleCall(low);
      }else{
        lowText += "";
      }
  }
  return lowText;
}

// 同義語追加部分
function add_sameWord(flag){
  sss = document.getElementById("addsam");
  // aタグを消去
  sss.removeChild(sss.lastChild);

  // 初回呼び出し時
  if(flag==1){
    count = 0;
    flag=0;
    temp2 = sameLen;
```

```
    }
    temp2 = temp2 - 20;
    if(temp2>0){
      count++;
      start = 20 * count;
      owari = 20 + start;
      if(owari>sameLen){
        owari = sameLen;
      }
    }
    // aタグ or brタグを消去
    sss.removeChild(sss.lastChild);

    if(temp2>0){
      if(count>1)
        // brタグの消去
        sss.removeChild(sss.lastChild);
      for(var i=start;i<owari;i++)
        sss.innerHTML += sames[i]+" ";
      if(temp2>=20){
        sss.innerHTML += "<br>";
        sss.innerHTML += "<a href='JavaScript:add_sameWord(0)'>"+"・・さらに表示";
        sss.innerHTML += "</a>";
      }
    }
}

// 上位語追加部分
function add_upperWord(flag){
  uuu = document.getElementById("addupp");
  // aタグを消去
  uuu.removeChild(uuu.lastChild);

  // 初回呼び出し時
  if(flag==1){
    count = 0;
```

```
    flag=0;
    temp2 = upperLen;
  }
  temp2 = temp2 - 20;
  if(temp2>0){
    count++;
    start = 20 * count;
    owari = 20 + start;
    if(owari>upperLen){
      owari = upperLen;
    }
  }
  // aタグ or brタグを消去
  uuu.removeChild(uuu.lastChild);

  if(temp2>0){
    if(count>1)
      // brタグの消去
      uuu.removeChild(uuu.lastChild);
    for(var i=start;i<owari;i++)
      uuu.innerHTML += uppers[i]+" ";
    if(temp2>=20){
      uuu.innerHTML += "<br>";
      uuu.innerHTML += "<a href='JavaScript:add_upperWord(0)'>"+"・・さらに表示";
      uuu.innerHTML += "</a>";
    }
  }
}

// 下位語追加部分
function add_lowerWord(flag){
  lll = document.getElementById("addlow");
  // aタグを消去
  lll.removeChild(lll.lastChild);

  // 初回呼び出し時
```

```
  if(flag==1){
    count = 0;
    flag=0;
    temp2 = lowerLen;
  }
  temp2 = temp2 - 20;
  if(temp2>0){
    count++;
    start = 20 * count;
    owari = 20 + start;
    if(owari>lowerLen){
      owari = lowerLen;
    }
  }
  // aタグ or brタグを消去
  lll.removeChild(lll.lastChild);

  if(temp2>0){
    if(count>1)
      // brタグの消去
      lll.removeChild(lll.lastChild);
    for(var i=start;i<owari;i++)
      lll.innerHTML += lowers[i]+" ";
    if(temp2>=20){
      lll.innerHTML += "<br>";
      lll.innerHTML += "<a href='JavaScript:add_lowerWord(0)'>"+"・・さらに表示";
      lll.innerHTML += "</a>";
    }
  }
}

/*****************************************************************************

          はてな連想語API
        WebService (XML-RPC形式)
```

```
**************************************************************************/

function GetRequestXml(content){
  var requestXml = '<?xml version=\"1.0\" encoding=\"utf-8\"?>' +
          '<methodCall>' +
          '  <methodName>hatena.getSimilarWord</methodName>' +
          '  <params>' +
          '    <param>' +
          '      <value>' +
          '        <struct>' +
          '          <member>' +
          '            <name>wordlist</name>' +
          '            <value>' +
          '              <array>' +
          '                <data>' +
          '                   <value>' +
          '                    <string>'+ content + '</string>' +
          '                   </value>' +
          '                </data>' +
          '              </array>' +
          '            </value>' +
          '          </member>' +
          '        </struct>' +
          '      </value>' +
          '    </param>' +
          '  </params>' +
          '</methodCall>';
  return requestXml;
}

// 調べたいものをXMLでポスト
function Imaginate(content){
  checkBoolean();

  // はてな連想語
  if(hatena_word){
```

```
    if(rensoudivFlag){
      var dtag = document.getElementById("first");
      var div = document.createElement("div");
      div.id = "rensou";
      dtag.appendChild(div);
      rensoudivFlag = false;
    }
      if(content == ""){
        createTable("連想される単語","rensou");
        return;
      }
      var requestXml = GetRequestXml(content);  // POST用のXMLを作成
      imagination_XML(requestXml);           // XMLをPOSTする
    }else{
      del('rensou');
      rensoudivFlag=true;
    }
}

// はてなのXMLデータ (連想語API)
function imagination_XML(request){
  fName = 'http://d.hatena.ne.jp/xmlrpc';

  httpObj2 = createXMLHttpRequest();
  if(httpObj2.readyState != 0) httpObj2.abort();
  httpObj2.onreadystatechange = displayData2;
  if (httpObj2){
    httpObj2.open('POST',fName,true);
    httpObj2.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
    httpObj2.send(request);
  }
}

// はてなの表示関数(連想語API)
function displayData2(){
  createTable("ロード中・・","rensou",true);
```

```
  if((httpObj2.readyState == 4) && (httpObj2.status == 200)){
    imaginationWord(httpObj2.responseXML);    // レスポンスを取得する
  }
}

// はてなのXMLを表示(連想語)
function imaginationWord(xml){
  checkBoolean();
  var temp = "";
  //var result = "";
  var wword = new Array();
  var wname = xml.getElementsByTagName('name');
  var values = xml.getElementsByTagName('value');

  for(var i=3;i<values.length;i+=2){
    j = (i-3)/2;
    temp = values[i].childNodes[0].text;
    temp = googleCall(temp);
    wword[j] = temp;
    //result += temp;
    //result += "<br>";
  }

  deleteTable4("rensou");

  // 以前表示していたものをクリアする
  var rensou = document.getElementsByName('rensou')[0];
  if(rensou.hasChildNodes()){
    for(var i=0;i<rensou.children.length;i++){
      rensou.removeChild(rensou.children[i]);
    }
  }

  result2  = "<table border='0'>";
  result2 += "<tr><td align='left' bgcolor='#3366FF' valign='top' nowrap>";
  result2 += "<b><font color='#FFFFFF'>"+"連想される単語"+"</font></b>";
```

```
  result2 += "</td></tr>";
  result2 += "<tr><td valign='top' width='800'>";
    if(wword.length==0){
      result2 += "Not Found";
    }else{
      for(var i=0;i<wword.length;i++)
        result2 += wword[i]+" ";
    }
  result2 += "</td></tr>";
  result2 += "</table><br>";
  rensou.innerHTML = result2;
}

// Google呼び出し部分
function googleCall(Query){
  var enc = encodeURIComponent(Query);
  sQuery = "<a target='_blank' href='http://www.google.com/search?q="+enc+"'>";
  sQuery += Query;
  sQuery += "</a>";
  return sQuery;
}

/*****************************************************************************

        Yahoo検索WebService  &  サムネイル

*****************************************************************************/

// 検索ワード
function YahooCall(content){
  checkBoolean();
  if(yahoo){
  if(tldivFlag){
    var dtag = document.getElementById("second");
    var td = document.createElement("td");
    td.id = "tl";
```

```
    td.className = "test";
    dtag.appendChild(td);
    tldivFlag = false;
  }
    if(content == ""){
      createTable("Yahoo検索","tl");
      return;
    }
    var enc = encodeURIComponent(content);
    if(language){
      content =
"http://api.search.yahoo.co.jp/WebSearchService/V1/webSearch?appid=friendstown&query=
"+enc+"&results=10";
    }else{
      content =
"http://api.search.yahoo.co.jp/WebSearchService/V1/webSearch?appid=friendstown&query=
"+enc+"&results=10&language=ja";
    }
    yahooXML(content);
  }else{
    //deleteTable4("tl");
    //tdタグ自体を消す
    del2('tl');
    tldivFlag=true;
  }
}

// XMLのHTTP通信オブジェクトを取得
function yahooXML(fName){
  httpObj = createXMLHttpRequest(); // ここは共通
  httpObj.onreadystatechange = showData; // POSTのレスポンスをReadyStateChangeHandler
で行う
  if (httpObj){
    httpObj.open("GET",fName,true);
    httpObj.send(null);
  }
```

```
}

function showData(){
  createTable("ロード中・・","t1",true);
  if((httpObj.readyState == 4) && (httpObj.status == 200)){
    wordAndThumnails(httpObj.responseXML);
  }
  else if((httpObj.readyState == 4) &&(httpObj.status == 400)){
    document.getElementById("t1").innerHTML = "Bad request. ";
  }
  else if((httpObj.readyState == 4) &&(httpObj.status == 403)){
    document.getElementById("t1").innerHTML = "Forbidden. ";
  }
  else if((httpObj.readyState == 4) && (httpObj.status == 503)){
    document.getElementById("t1").innerHTML = "Service unavailable. ";
  }
}

function wordAndThumnails(xml2){
  var thumnail = "";
  var simpleApi ="http://img.simpleapi.net/small/";
  var photos = new Array();  // サイトのサムネイル
  var links = new Array();  // サイトのURL
  var namae = new Array();  // タイトル
  var setumei = new Array();  // サイトの説明

  for(var i=0;i<10;i++)
    photos[i] = new Image(128,128);

  var Error = xml2.getElementsByTagName('Error'); // エラーの場合
  var ErrorLen = Error.length;
  var Result = xml2.getElementsByTagName('Result'); // Resultタグ

  var ResultSet = xml2.getElementsByTagName('ResultSet'); // タグ
  var titles = xml2.getElementsByTagName('Title');    // Titleタグ
  var Summarys = xml2.getElementsByTagName('Summary');   // Summaryタグ
```

```
  deleteTable4("t1");     // 以前表示していたものをクリアする
  var t1 = document.getElementsByName('t1')[0];

  // Resultタグがあればエラーではないので処理
  if(Result.length!=0){
    for(var i=0;i<Result.length;i++){
      photos[i].src = simpleApi + Result[i].childNodes[2].childNodes[0].nodeValue;
      links[i] = Result[i].childNodes[2].childNodes[0].nodeValue;  //URLタグはこれで
取得
    }

    Rs0 = ResultSet[0].getAttributeNode("totalResultsAvailable").value;
    Rs1 = ResultSet[0].getAttributeNode("totalResultsReturned").value;
    thumnail += "<table border='0'>";
    thumnail += "<tr><td bgcolor='#3366FF'>";
    thumnail += "<b><font color='#FFFFFF'>"+"Yahoo! 検索"+"</font></b>";
    thumnail += "</td><td bgcolor='#3366FF'>";
    thumnail += "<font color='#FFFFFF'>"+"マッチした件数:"+Rs0+"  取得件
数:"+Rs1+"</font>";
    thumnail += "</td></tr>";
    for(var i=0;i<Result.length;i++){
      thumnail += "<tr><td nowrap>";
      thumnail += "<a href="+links[i]+" >";
      thumnail += "<img src="+photos[i].src+" width='128' height='128'
alt="+titles[i].text+" border='0'>";
      thumnail += "</a>";
      thumnail += "</td>";
      thumnail += "<td valign='top' width='400' nowrap>";
      thumnail += "<a href="+links[i]+" >";    // リンク先URL
      thumnail += "<b>"+titles[i].text+"</b>";  // タイトル
      thumnail += "</a>";
      thumnail += "<br><br>";
      thumnail += Summarys[i].text;    // 説明
      thumnail += "</td></tr>";
    }
```

```
    thumnail += "</table><br>";

  }else{
    Rs0 = 0;
    Rs1 = 0;
    thumnail += "<table border='0'>";
    thumnail += "<tr><td bgcolor='#3366FF'>";
    thumnail += "<b><font color='#FFFFFF'>"+"Yahoo! 検索"+"</font></b>";
    thumnail += "</td><td bgcolor='#3366FF'>";
    thumnail += "<font color='#FFFFFF'>"+"マッチした件数:"+Rs0+"  取得件
数:"+Rs1+"</font>";
    thumnail += "</td></tr>";
    thumnail += "<tr><td align='center' colspan='2'>";
    thumnail += "Not Found";
    thumnail += "</td></tr>";
    thumnail += "</table><br>";
  }
  t1.innerHTML = thumnail;
}

/*****************************************************************************

              Wikipedia XML

*****************************************************************************/

// 検索ワード
function WikipediaCall(content){
checkBoolean();
  if(wikipedia){
  if(showDivdivFlag){
    var dtag = document.getElementById("first");
    var div = document.createElement("div");
    div.id = "showDiv";
    dtag.appendChild(div);
    showDivdivFlag = false;
```

```
  }
    if(content == ""){
      createTable("Wikipedia","showDiv");
      return;
    }else{
      var enc = encodeURIComponent(content);
      content = "http://wikipedia.simpleapi.net/api?keyword="+enc;
      wikipediaXML(content);
    }
  }else{
    del('showDiv');
    showDivdivFlag=true;
  }
}

// XMLのHTTP通信オブジェクトを取得
function wikipediaXML(fName){
  httpObj = createXMLHttpRequest(); // ここは共通
  httpObj.onreadystatechange = KeyDisplay;
  if (httpObj){
    httpObj.open("GET",fName,true);
    httpObj.send(null);
  }
}

// 表示関数(Wikipedia API)
function KeyDisplay(){
  createTable("ロード中・・","showDiv",true);
  if((httpObj.readyState == 4) && (httpObj.status == 200)){
    KeyWordWiki(httpObj.responseXML);     // レスポンスを取得する
  }
}

function KeyWordWiki(xml){
  var keyword = "";
  var showDiv = document.getElementsByName('showDiv')[0];
```

```
var result = xml.getElementsByTagName('result'); // resultタグ
var resultLen = result.length;

var url = xml.getElementsByTagName('url'); // urlタグ
var title = xml.getElementsByTagName('title'); // titleタグ
var body = xml.getElementsByTagName('body'); // bodyタグ
var strict = xml.getElementsByTagName('strict'); // strictタグ
var datetime = xml.getElementsByTagName('datetime'); // datetimeタグ

// 以前表示していたものをクリアする
if(showDiv.hasChildNodes()){
  for(var i=0;i<showDiv.children.length;i++){
    showDiv.removeChild(showDiv.children[i]);
  }
}

if(resultLen!=0){
  keyword  = "<table border='0' width='800'>";
  keyword += "<tr><td width='800' bgcolor='#3366FF'>";
  keyword += "<b><font color='#FFFFFF'>"+"Wikipedia"+"</font></b>";
  keyword += "</td></tr>";

  keyword += "<tr><td>";
  keyword += "<a href='"+url[0].text+"' target='_blank'>";
  keyword += "<font size='6'>"+title[0].text+"</font>";
  keyword += "</a>";
  keyword += "</td></tr>";

  keyword += "<tr><td>"+body[0].text+"<br>";
  keyword += "更新日時:"+datetime[0].text;
  keyword += "</td></tr>";
  keyword += "</table><br>";
  showDiv.innerHTML = keyword;
}else{
  keyword  = "<table border='0' width='800'>";
```

```
    keyword += "<tr><td width='800' bgcolor='#3366FF'>";
    keyword += "<b><font color='#FFFFFF'>"+"Wikipedia"+"</font></b>";
    keyword += "</td></tr>";
    keyword += "<tr><td>";
    keyword += "Not Found";
    keyword += "</td></tr>";
    keyword += "</table><br>";
    showDiv.innerHTML = keyword;
  }
}

/*****************************************************************************

        テクノラティ, タグ検索API

*****************************************************************************/

function TechnoratiCall(content){
  checkBoolean();
  if(technorati){
  if(t2divFlag){
    var dtag = document.getElementById("second");
    var td = document.createElement("td");
    td.id = "t2";
    td.className = "test";
    dtag.appendChild(td);
    t2divFlag = false;
  }
    if(content == ""){
      createTable("Technoratiタグ検索","t2");
      return;
    }
    var enc = encodeURIComponent(content);
    content =
"http://api.technorati.com/tag?key=0f614955e242196590977d7056743930&tag=" + enc +
"&limit=2";
```

```
    TagXML(content);
  }else{
    del2('t2');
    t2divFlag=true;
  }
}

// RESTでコール
function TagXML(fName){
  httpObject = createXMLHttpRequest();
  httpObject.onreadystatechange = TagData;
  if (httpObject){
    httpObject.open("GET",fName,true);
    httpObject.send(null);
  }
}

function TagData(){
  createTable("ロード中・・","t2",true);
  if((httpObject.readyState == 4) && (httpObject.status == 200)){
    T_CallBack(httpObject.responseXML);
  }
}

function T_CallBack(xml){
  var simpleApi ="http://img.simpleapi.net/small/";
  var t2 = document.getElementsByName('t2')[0];  // idの取得

  var blogPhotos = new Array();
  for(var i=0;i<20;i++)
    blogPhotos[i] = new Image(128,128);

  var items = xml.getElementsByTagName('item');
  var titles = xml.getElementsByTagName('name'); // ブログのタイトル
  var urls = xml.getElementsByTagName('url'); // URLタグ
  var excerpts = xml.getElementsByTagName('excerpt'); // ブログの記事
```

```
  var errors = xml.getElementsByTagName('error');
  var errorLen = errors.length;
  var matches = xml.getElementsByTagName('postsmatched');

  th_technorati = "";
  numOfBlog = "";

  deleteTable4("t2");    // 以前表示していたものを消去

  // エラータグが無い場合の処理
  if(errorLen==0){
    numOfBlog = matches[0].childNodes[0].nodeValue;
    for(var i=0;i<items.length;i++)
      blogPhotos[i].src = simpleApi + urls[i].text;

    th_technorati  = "<table border='0'>";
    th_technorati += "<tr><td bgcolor='#3366FF' nowrap>";
    th_technorati += "<b><font color='#FFFFFF'>"+"Technorati 検索"+"</font></b>";
    th_technorati += "</td><td bgcolor='#3366FF' nowrap>";
    th_technorati += "<font size='+1'
color='#FFFFFF'><b>「"+document.ajaxForm.word.value+"</b>";
    th_technorati += "」タグを含むブログ件数:" + numOfBlog + "  取得件数:2"+"</font>";
    th_technorati += "</td></tr>";
    for(var i=0; i<items.length; i++){
      th_technorati += "<tr><td>";
      th_technorati += "<img src="+blogPhotos[i].src+" width='128' height='128'
alt="+titles[i].text+" border='0'>";
      th_technorati += "</td><td width='400' nowrap>";
      th_technorati += "<a href="+urls[i].text+">";
      th_technorati += "<b>"+titles[i].text+"</b>";
      th_technorati += "</a><br>";
      th_technorati += excerpts[i].text;
      th_technorati += "</td></tr>";
    }
    th_technorati += "</table><br>";
```

```
  }else{
    th_technorati  = "<table border='0'>";
    th_technorati += "<tr><td bgcolor='#3366FF' nowrap>";
    th_technorati += "<b><font color='#FFFFFF'>"+"Technorati 検索"+"</font></b>";
    th_technorati += "</td><td bgcolor='#3366FF' nowrap>";
    th_technorati += "<font size='+1'
color='#FFFFFF'><b>「"+document.ajaxForm.word.value+"</b>";
    th_technorati += "」タグを含むブログ件数:0  取得件数:0"+"</font>";
    th_technorati += "</td></tr>";
    th_technorati += "<tr><td align='center' colspan='2'>";
    th_technorati += "Not Found";
    th_technorati += "</td></tr>";
    th_technorati += "</table><br>";
  }
  t2.innerHTML = th_technorati;
}


// 共通部分
function createXMLHttpRequest(){
  var xmlhttp = null;
  try{
    // FireFox,Opera
    xmlhttp = new XMLHttpRequest();
  }catch(e){
    try{
      // IE 6以降
      xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
    }catch(e){
      try{
        // IE 5以降
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
      }catch(e){
        return null;
      }
    }
```

```
  }
  return xmlhttp;
}

// テーブルの作成
function createTable(message, id, loadFlag) {
  table = "";
  table += "<table border='0'>";
  if(loadFlag) {
    table += "<tr><td align='center' valign='top' nowrap>";
    table += "<img
src='http://friendstown.web.fc2.com/image/indicator_mozilla_blu.gif' width='16'
height='16' border='0'>";
    table += "<font color='#000000'>";
  }else{
    table += "<tr><td align='center' bgcolor='#3366FF' valign='top' nowrap>";
    table += "<font color='#FFFFFF'>";
  }
  table += "<b>"+message+"</b></font>";
  table += "</td></tr>";
  table += "</table><br>";
  document.getElementById(id).innerHTML = table;
}

// テーブルの消去
function deleteTable4(id) {
  var divTable = document.getElementById(id);
  numOfTable = divTable.childNodes.length;

  if(numOfTable!=0) {
    // divタグの子ノードであるtableタグを消去
    divTable.removeChild(divTable.childNodes[0]);
  }
}

// 更新されたらチェックをはずす
```

```
function Reload(){
  checkBoolean();
  if(thesaurus)
    document.check.thesaurus.checked = false;
  if(hatena_word)
    document.check.hatena_word.checked = false;
  if(technorati)
    document.check.elements[3].checked = false;
  if(yahoo)
    document.check.elements[4].checked = false;
  if(wikipedia)
    document.check.wiki.checked = false;
  SisodivFlag=true;
  rensoudivFlag=true;
  showDivdivFlag=true;
  t1divFlag = true;
  t2divFlag = true;
  return thesaurus, hatena_word, language, technorati, yahoo, wikipedia;
}
// divタグを消す
function del(id){
  var iiii = document.getElementById(id);
  document.getElementById("first").removeChild(iiii);
}
function del2(id){
  var iiii = document.getElementById(id);
  document.getElementById("second").removeChild(iiii);
}
</script>
</head>
<body onload="Reload()">
<h1>マッシュアップ Web Application(ローカル専用)</h1>
<table border="1" cellpadding="5" cellspacing="0">
<tr>
<td valign="top" height="60">
  <form name="ajaxForm">
```

```
  <input type="text" name="word">
  <input type="button" value="検索" onClick="init()"><br>
  <input type="radio" name="language" value="all" checked>ウェブページ全体
  <input type="radio" name="language" value="ja">日本語ページのみ
  </form>
</td>
<td valign="top">
  <form name="check">
  <input type="checkbox" name="thesaurus"
onclick="thesaurusCall(document.ajaxForm.word.value)">シソーラス辞書検索<br>
  <input type="checkbox" name="hatena_word"
onclick="Imaginate(document.ajaxForm.word.value)">はてな連想語<br>
  <input type="checkbox" name="wiki" value="wikipedia"
onclick="WikipediaCall(document.ajaxForm.word.value)">Wikipedia
</td>
<td valign="top">
  <input type="checkbox" name="blog" value="technorati"
onclick="TechnoratiCall(document.ajaxForm.word.value);">ブログ・タグ検索(テクノラテ
イ)<br>
  <input type="checkbox" name="search" value="yahoo"
onclick="YahooCall(document.ajaxForm.word.value);">Yahoo検索エンジン<br>
  </form>
</td>
</tr>
</table>
<br>
<!-- シソーラス表示部分 id=siso -->
<!-- 関連語表示部分 id=rensou -->
<span id="first">
</span>
<!-- Wikipedia id=showDiv -->
<table border="0">
<tr id="second">
  <!-- Yahoo検索一覧 id=t1 -->
  <!-- Technorati検索一覧 id=t2 -->
  <!-- YouTube id=mov-->
```

```
</tr>
</table>
<font size="-1">
<ul>
  <li>シソーラスは日本大学文理学部国文学科の荻野綱男先生のものを利用しています．</li>
  <li>シソーラスは学内Webサーバのものを呼び出しています．ローカルシステムではありません．</li>
  <li>各WebServiceは直接呼び出しています．ローカルでしか動作しません．</li>
</ul>
</font>
</body>
</html>
```