インターフェイスの街角 (86) – Ajax

増井俊之

だいぶ前のことになりますが、2003 年 2 月号で、Flash とサーバーとの通信によってブラウザ上で〝リッチ・クライアント〟を実現する方法を紹介しました（リッチ・クライアントとは、サーバーからクライアントにプログラムをダウンロードし、サーバーと通信をおこないながらアプリケーションを動かす手法を指します）。

通常の CGI プログラムを用いて作成された Web アプリケーションを利用する場合、ボタンのクリックやメニューの選択などにともなって状態遷移が発生すると、新しいページが表示されてしまいます。このため、十分に対話的な Web アプリケーションの実現は困難でした。しかし、Flash や Java を使えば、プログラムとサーバーとのあいだで通信しながらデータをやりとりしたり、描画したりといった操作が可能なので、ページ移動をしなくても状態遷移を含むアプリケーションを実行させることができます。

このところ、Flash や Java の代わりに JavaScript を利用してリッチ・クライアントを実現する手法が流行っています。最近、Google がサービスを開始した Google Suggest や GMail、Google Maps などでは、JavaScript を最大限に活用してリッチ・クライアントを実現しています。Yahoo! などの地図サービスでは、縮尺を変更したり位置を移動する場合は新たなページに移動しなければなりません。一方、Google Maps では、地図をマウスでドラッグしても、別のページに移動することなく未ロードの地図データをサーバーからダウンロードして表示するので、よりスムーズに地図を閲覧できます。

JavaScript を用いて描画したり、サーバーと通信してリッチ・クライアントを実現する手法は〝Ajax（エイジャックス）〟と呼ばれています。

今回は、この Ajax の技法を使った Web アプリケーションについて解説します。

Ajax とは何か

Ajax (Asynchronous JavaScript + XML) とは、Jesse James Garrett 氏が自分の Web ページ[1]で提唱した造語です。JavaScript の通信機能を用いてサーバーと非同期的に XML 通信をおこないながらリッチ・クライアントを実現する手法で、以下のような機能から構成されています。

- XHTML と CSS (Cascading Style Sheet) により標準化された画面表示
- DOM (Document Object Model) を用いた動的な表示と対話処理
- XML、XSLT によるデータ交換/データ制御
- XMLHttpRequest を用いた非同期的データ通信
- JavaScript による上記技術の統合

通常の CGI で Web アプリケーションを作成すると、ボタンやメニューを操作するたびにサーバーに HTTP 要求が送られ、新しいページが表示されるという操作が繰り返されます。その結果、ちょっとした検索やパラメータ変更などをおこなう場合も、つねに画面を切り替えなければならないという問題がありました。

Ajax にもとづくアプリケーションでは、対話的な処理

1 Ajax : A New Approach to Web Applications（http://adaptivepath.com/publications/essays/archives/000385.php。邦訳は http://antipop.zapto.org/docs/translations/ajax.html）

はおもにブラウザ上の JavaScript で実行し、必要に応じて非同期的にサーバーと通信してデータや画面の更新をおこないます。これによって、応答性のよいクライアント・アプリケーションが実現できます。

たとえば、一般的な検索操作では、検索ボタンを押してサーバーに検索文字列を送ると検索結果ページが表示されます。これに対し、Ajax では JavaScript プログラムからサーバーに検索文字列を送り、受け取った検索結果を同じページ内に表示します。こうすれば画面を切り替えずに検索操作がおこなえるので、Web ページ上でも文字を入力するたびに検索結果がすぐに表示される〝インクリメンタル検索〟が実現できます。

Ajax では、メニューやボタンなどの GUI 部品以外のインターフェイスも作れます。たとえば、1998 年 7 月号で紹介した〝ズーミング・インターフェイス〟を Ajax で実現するには、文字や画像のズーミング操作はローカルの JavaScript プログラムで処理し、ズーミングや移動によって新たに表示しなければならないデータが必要になった場合にだけサーバーと通信します。これによって、ユーザーの操作とは非同期的にデータが取得できるので、サーバーからの応答を待たずに操作を続けられることになります。

かつての JavaScript にはサーバーとの非同期通信機能がなく、このような手法は利用できませんでした。しかし、最近は Internet Explorer や Firefox、Safari など、多くのブラウザの JavaScript がサーバーとの非同期通信をサポートしているので、GMail や Google Maps などのリッチ・クライアントが可能になったわけです。そして、これらのアプリケーションが成功したおかげで、Ajax の有効性が急速に認められるようになったのでしょう。

Ajax アプリケーションの作成手順

Ajax アプリケーションは、基本的には以下の手順で作成します。JavaScript とサーバーの通信には、XMLHttpRequest オブジェクト [1, 2] を使います。

1. 必要に応じて XMLHttpRequest オブジェクトを利用し、ブラウザの JavaScript からサーバーへ要求を送る。
2. ブラウザからの要求に対し、サーバーは CGI を実行して結果を XML で返す。
3. サーバーから返事を受け取ると、XMLHttpRequest オブジェクトに登録したコールバック関数を呼び出し、返されたデータを処理する。

XMLHttpRequest は、もともとは ActiveX オブジェクトとして Microsoft が導入し、その後に Mozilla などでも使われるようになりました。こういった経緯があるためか、Internet Explorer と Mozilla/Firefox などでは生成方法が異なります。

Internet Explorer の場合は、次のようにして XMLHttpRequest オブジェクトを生成します（誌面の都合上、⇒ で折り返しています）。

```
var req = new ActiveXObject⇒
              ("Microsoft.XMLHTTP");
```

一方、Mozilla や Firefox、Safari では、

```
var req = new XMLHttpRequest();
```

として生成します。

生成された XMLHttpRequest に対し、以下のようにしてコールバック関数を指定すれば、サーバーとの通信が可能になります（open() の第 3 引数が true の場合に非同期通信がおこなわれます）。

```
var url = 'http://example.com/ajax.cgi'
req.onreadystatechange = callback;
req.open("GET", url, true);
req.send(null);
```

■ XMLHttpRequest?

通信が正常に終了すると、XMLHttpObject の readyState が〝4〟になり、結果を XML またはテキスト形式で取得することができます（通信の実行中には、readyState は〝2〟または〝3〟になっています）。

```
function callback(){
  if(req.readyState==4 req.status == 200)){
    ret = xmlhttp.responseXML;
    // ret = xmlhttp.responseText;
  }
}
```

これらの機能を具体的にどのように利用するかは、プログラム側に任されています。Ajax では、サーバーとのデータのやりとりには XML を、画面表示には DOM や CSS を使うことになっていますが、かならずしもこれらを利用しなければならないわけではありません。単純なデータの受渡しさえおこなえればよいのなら、XML を使わずにすませることも可能です。

図 1　ドラッグ前の状態

図 2　ドラッグ後の状態

Ajax アプリケーションの実例

以下では、XMLHttpRequest を用いたアプリケーションの例をいくつか紹介します。

画像のドラッグ

GUI 画面でアイコンを動かす場合のように、ドラッグ&ドロップで画像を動かす操作は多くのアプリケーションで使われています。しかし、Web ブラウザ上ではこのような操作はあまりみかけません。

ブラウザ上で画像をドラッグしてなんらかの操作を指示した場合、その結果をサーバーに通知する必要があります。これは、一般的な CGI などを用いてサーバーに通知すると、画面が遷移して滑らかな操作感が損なわれるという問題があったからでしょう。これに対し、XMLHttpRequest を利用すると、画面遷移をおこなわなくても、ドラッグ&ドロップの結果をリアルタイムにサーバーへ通知することが可能になります。

たとえば、図 1 の画面で画像をクリックしてドラッグすると、図 2 のように変化します。

この Web ページのソースは、リスト 1 のようになっています。この HTML で呼び出している JavaScript プログラム dragimage.js を、末尾のリスト 2 に示します[2]。このプログラムでは、マウスボタンを押したとき、動かしたとき、離したときにそれぞれ呼び出される mouse_down()、mouse_move()、mouse_up() の各関数を定義し、画像の位置をスタイルファイルで ˝absolute˝ に設定することにより、画像のドラッグ操作を実現しています。

リスト 1　画像ドラッグページ（dragimage.html）

```
<html>
<head>……</head><h1>Ajaxで画像ドラッグ</h1>
<body>
<script language="JavaScript"
        src="dragimage.js"></script>
</body>
</html>
```

ブラウザ内で画像をドラッグした場合、リロードすると移動結果が失われてしまいます。そのため、マウスボタンを離したときに、ドラッグした画像の位置を setpos.cgi（リスト 4）でサーバーに伝え、画像の位置を憶えておくようにしています。

■「……画像の位置をリスト 3 の getpos.cgi で取得し、setpos.cgi(リスト 4) でサーバーに伝えることで」？

これらのプログラムでは DOM や CSS は利用していませんし、非同期的な通信も不要なので Ajax とはいえません。しかし、XMLHttpRequest をうまく使った例にはなるでしょう。

Ajax によるピテカン辞書

図 3 は Ajax で「ピテカン辞書」[4][3] を実装した例です。ピテカン辞書とは、2000 年 2 月号で紹介した「LensBar」のもとになったズーミング辞書検索システムです。見出し語数が数万の英和辞書を間引いて表示し、単語をドラッグしてズーミングしたり、キーワードの曖昧検索などによって検索処理を実行します。

ピテカン辞書は、ズーミング処理と曖昧検索の 2 つの部分に分離できます。ズーミング処理はブラウザ側の JavaScript でおこない、検索文字列が指定されたらサーバーに置いた辞書に検索要求を送り、曖昧検索の結果を非同期的に受け取ることにすれば、Ajax でピテカン辞書を実装する

2　マウス座標を取得する関数などはブラウザごとに異なるため、この CGI は Mozilla/Firefox 以外では使えません。

3　http://pitecan.com/papers/ProSym96/ProSym96.pdf

図 3　ピテカン辞書の初期画面

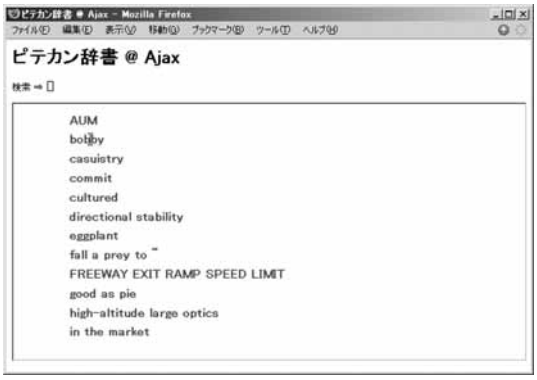


図 4　bobby をドラッグした状態

ことができます。

図 3 の初期画面では、辞書[4]のエントリの一部が表示されています。

ここで ``bobby'' をクリックして右方向にドラッグすると ``bobby'' に近い単語が表示され、画面は図 4 のように変化します。ブラウザの JavaScript プログラムは、それまでに表示された単語のリストだけを配列に保存しているため、``bobby'' 以外の辞書エントリは空になっています。しかし、この状態のまま一瞬待つと、サーバーから非同期的に辞書データが送られ、画面が図 5 のように変化します。さらに右方向にドラッグすると、図 6 のように ``bobby'' に近い単語がすべて表示されます。

Ajax 版のピテカン辞書では agrep [3][5] による曖昧検索

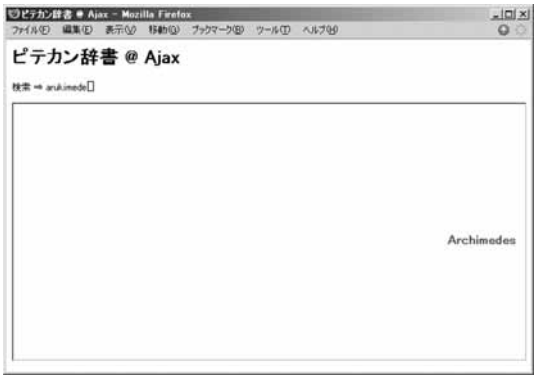
4 gene95 辞書 (http://www.namazu.org/~tsuchiya/sdic/data/gene.html)を使っています。

5 agrep は WebGlimpse (http://webglimpse.org/) に含まれています。

図 5　1 秒後の状態

図 6　さらに右方向にドラッグ

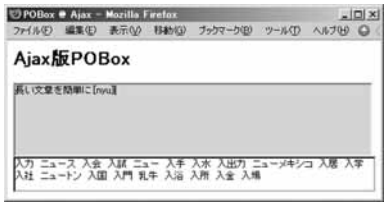
図 7　``arukimede'' で検索



を利用しているので、アプリケーション版のピテカン辞書と同じく、``arukimede'' と入力すると ``Archimedes'' を検索することができます (図 7)。

図 8　Ajax 版 POBox の初期画面

図 9　POBox モードでの入力



Ajax による POBox

図 8〜9 は、この連載でも何回か紹介した予測型テキスト入力システム「POBox」[6]をAjaxで実装した例です。ブラウザ上でキー入力をおこなうたびににサーバーへ入力文字列を送り、候補単語を受け取って表示します。マウスまたはスペースキーで候補を選択し、リターンキーで確定することができます。

辞書操作や検索処理はサーバー上でおこなわれるため、任意の textarea に日本語入力機能をもたせることができます。textarea ごとに異なる変換システムや入力システムを定義しておくと、便利かもしれません。

今後の課題

以上のように、Ajax を利用すればブラウザ上でも通常のアプリケーションと遜色のないインターフェイスを実現することができます。しかし、現在のところは以下のような問題があり、残念ながら Ajax ソフトウェアの開発はそれほど簡単とはいえません。

- JavaScript の開発/デバッグ環境が不十分である。
- 異なるブラウザで動くようにすること（クロスブラウザ対応）がたいへん難しい。
- JavaScript は大規模アプリケーションを書くのに不向きである。

6 http://pitecan.com/OpenPOBox/

なかでも、クロスブラウザ対応の開発は悪夢といっても過言ではありません。リスト 2 のプログラムは、もともと Mozilla と Firefox での利用を前提としているため、比較的シンプルになっています。しかし、ブラウザによって名前や挙動が異なる関数が多いため、複数のブラウザで同じように動く JavaScript プログラムを書くには相当な手間がかかります。

システムによって挙動がまったく違うのなら、それぞれに合わせてコードを用意するしかないので、まだあきらめもつきます。ところが、中途半端に異なるところがあるため、どうしても入り組んだコードになってしまうのです。事実、Google Maps などでは各種のブラウザに対応するために涙ぐましいほどの工夫を凝らしています。

この問題の解決に向けて、Ajax アプリケーションを簡単に作成できるようにするツールキットの開発が始められています[7]。開発環境の整備がさらに進めば、完成度の高い Ajax アプリケーションも増えてくることでしょう。

Flash や Java を用いて作成したリッチ・クライアントは、HTML だけで構成されるページに比較すると表示に時間がかかることが多く、敬遠されがちでした。また、これらの言語で作られたページは HTML とは完全に別の世界になってしまうため、一般的な検索エンジンのように HTML を前提にしたシステムとの相性はあまりよいとはいえません。Ajax の場合は、JavaScript はほとんどのブラウザに標準で備わっていますし、基本的には HTML を利用するため、このような問題は少なくなっています。

Web ページ上の状態遷移を URL 形式で保存しておけば、遷移中の状態をブックマークに登録することも可能です。たとえば、ピテカン辞書でも、検索文字列とズーミングレベルを URL に含めるようにすれば、特定の検索状態を記録しておくことができます。Flash や Java では、こういう機能の実現はかなり難しいと思います。

おわりに

Ajax を利用すると、ブラウザ上で通常のアプリケーションと同様に操作できる Web アプリケーションが実現できます。今後、これに類する技法がさらに普及していくの

7 たとえば、PHP 対応の Ajax ツールキット「SAJAX」が公開されています（http://www.modernmethod.com/sajax/）。

は間違いないでしょう。

2003 年 2 月号で紹介したような Flash で作るリッチ・クライアントは、現時点ではまだそれほど普及していません。しかし、Ajax のような手法が浸透していけば、急速に広まる可能性があります。

Garrett 氏は、Ajax アプリケーションに限界があるとすれば、技術的なことよりもインターフェイス設計者の想像力のほうだろうと述べています。その意味でも、現在のユーザーの想像を超えるようなアプリケーションを Web ブラウザ上に実装していきたいものです。

（ますい・としゆき　産業技術総合研究所）

[参考文献]

[1] Drew McLellan, *Very Dynamic Web Interface*, February 9,2005 (http://www.xml.com/pub/a/2005/02/09/xml-http-request.html)

[2] XULPlanet.com, *XMLHttpRequest* (http://xulplanet.com/references/objref/XMLHttpRequest.html)

[3] Sun Wu and Udi Manber, “agrep — A Fast Approximate Pattern-Matching Tool”, In *Proceedings of USENIX Technical Conference*, pp.153–162, January 1992

[4] 増井俊之、水口 充、George Borden、柏木宏一「なめらかなユーザインタフェース」、第 37 回冬のプログラミングシンポジウム予稿集、pp.13–23、情報処理学会、1996 年 1 月

リスト 2　画像ドラッグを扱う JavaScript (dragimage.js)

```
var selected_image = "none"
var offsetx, offsety;

for(var i=0;i<20;i++){
  document.write('<img id="image' + i +
    '" style="visibility:hidden;" onMouseDown="set_image(' + i + ')">')
}

document.onmousemove = mouse_move;
document.onmouseup =   mouse_up;
document.onmousedown = mouse_down;

xmlhttp = new XMLHttpRequest();
xmlhttp.overrideMimeType("text/xml");
xmlhttp.open("GET", "getpos.cgi", true);
xmlhttp.onreadystatechange=function() {
  if (xmlhttp.readyState==4) {
    var xmldoc = xmlhttp.responseXML;
    images = xmldoc.getElementsByTagName('image')
    for(var i=0;i<images.length;i++){
      image = images.item(i)
      file = image.getElementsByTagName('file').item(0).childNodes.item(0).nodeValue;
      x = image.getElementsByTagName('x').item(0).childNodes.item(0).nodeValue;
      y = image.getElementsByTagName('y').item(0).childNodes.item(0).nodeValue;
     img = document.getElementById("image" + i)
     img.style.left = x;
     img.style.top = y;
     img.style.position = 'absolute';
     img.style.visibility= '';
     img.src = file;
    }
  }
}
xmlhttp.send(null)

function set_image(image){
  selected_image = image
}

function mouse_down(e){
  img = document.getElementById("image" + selected_image)
```

```
  offsetx = e.pageX - img.x;
  offsety = e.pageY - img.y;
}

function mouse_up(e){
  img = document.getElementById("image" + selected_image)
  xmlhttp.open("GET", "setpos.cgi?imageno=" + selected_image +
               "&x=" + img.x + "&y=" + img.y,true);
  xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState==4) {
      result = xmlhttp.responseText
    }
  }
  xmlhttp.send(null)
  selected_image = 'none'
}

function mouse_move(e){
  if(selected_image != "none"){
    img = document.getElementById("image" + selected_image);
    img.style.left = e.pageX - offsetx;
    img.style.top = e.pageY - offsety;
  }
}
```

リスト 3　画像位置取得 CGI (getpos.cgi)

```
#!/usr/bin/env ruby
# 画像の位置をXMLで取得する

require 'cgi'
require 'pstore'

images = []
db = PStore.new("pstore")
db.transaction do
  images = db['images']
  if images.nil? then
    images = db['images'] = []
    Dir.open(".").each { |file|
      if file =~ /\.(jpg|gif|png)$/ then
        images << [file, 0, 0]
      end
    }
  end
end

print "Content-Type: text/xml\n\n"
print "<data>\n"
images.each { |image|
  print <<EOF
  <image>
    <file>#{image[0]}</file>
    <x>#{image[1]}</x>
    <y>#{image[2]}</y>
  </image>
EOF
}
print "</data>\n"
```

リスト 4　画像位置通知 CGI (setpos.cgi)

```
#!/usr/bin/env ruby
# 画像の位置を保存する

require 'cgi'
require 'pstore'

cgi = CGI.new('html3')
x = cgi.params['x'][0].to_i
y = cgi.params['y'][0].to_i
imageno = cgi.params['imageno'][0].to_i

db = PStore.new("pstore")
db.transaction do
  images = db['images']
  images[imageno][1] = x
  images[imageno][2] = y
end

cgi.out { "" }
```