インターフェイスの街角 (94) – Greasemonkey によるブラウザ機能の拡張

増井俊之

JavaScript の活用

このところ、Web ブラウザや Web ページの情報を制御するために〝ブックマークレット (bookmarklet)〟がよく使われています。これは、ブラウザのブックマーク（お気に入り）に登録しておき、好きなときに呼び出して使える小さな JavaScript プログラムです。ページの HTML の情報を JavaScript で書き換えれば、ページの外観を変えたり機能を付け加えることができます。

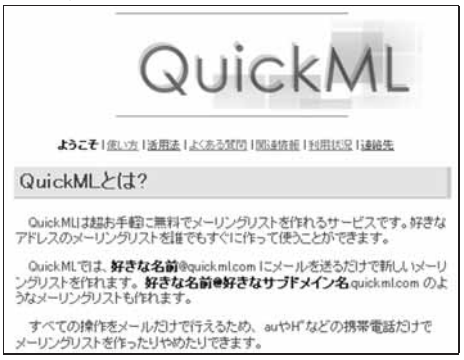
ブックマークレット関連の情報をまとめたサイト[1]では、背景色を変えたり、リンクを強調表示したり、ページに電卓機能を追加するといったブックマークレットがたくさん紹介されています。

また、ソーシャル・ブックマークのシステムである del.icio.us[2]や〝はてなブックマーク〟[3]では、閲覧中の Web ページを登録するためのブックマークレットが用意されています。これを Web ブラウザのブックマークに加えておけば、1 回の操作でページが登録できるようになっています。2004 年 10 月号などで紹介した〝本棚.org[4]〟でも、書籍情報が簡単に登録できるブックマークレットを提供しています。

ブックマークレットの例

一例として、Web ページ上のテキストを大阪弁に変換する〝大阪弁化フィルタ[5]〟をブックマークレットから利用する手順を紹介します。

図 1 QuickML.com



まず、下記のようなリンクを使って JavaScript プログラムを〝大阪弁化〟という名前でブックマークに登録します（誌面の都合上、⇒ で折り返しています。以下同様）。

```
<a href="javascript:location.href='http://yan⇒
.m78.com/osaka2.cgi?URL='+encodeURIComponent⇒
(location.href)">大阪弁化</a>
```

そして、図 1 の QuickML.com のページを見ているときにブックマークから〝大阪弁化〟を選択すると、上記の、

```
location.href= …… (localtion.href)
```

という JavaScript プログラムが実行され、画面は図 2 のように変化します[6]。

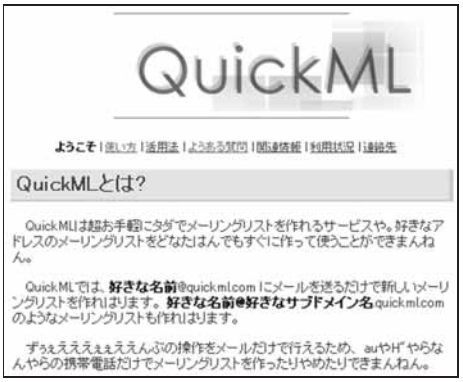
ブックマークレットの限界

ブックマークレットはたいへん便利ですが、以下のように不便な点もあります。

1 http://bookmarklet.daa.jp/
2 http://del.icio.us/
3 http://b.hatena.ne.jp/
4 http://hondana.org/
5 http://yan.m78.com/osaka.html

6 関西出身者としては、納得のいかない変換結果ではありますが …… 。

図 2　大阪弁化された QuickML.com



- ユーザーが人手で呼び出す必要がある
  ブックマークレットは、〝ブックマーク〟などのメニューから選択し、呼び出して使う必要があります。あるサイトに対してつねに同じブックマークレットを適用したい場合も、そのサイトを参照するたびにユーザーが呼び出さなければなりません。
  さきほどの例では、〝大阪弁化〟というブックマークを選んで Web ページのテキストを大阪弁に変換しましたが、別のページに大阪弁化フィルタを適用したい場合は、そのページを閲覧しているときに同じ作業を繰り返すことになります。
- 他サイトとの通信ができない
  2005 年 5 月号で紹介した Ajax の手法を利用すれば、同じサイトのサーバーと通信することができます。しかし、閲覧中の Web ページと異なるサイトのサーバーとの通信は不可能です。
- 永続的なデータは扱えない
  JavaScript プログラムのなかで利用するデータは、ファイルなどのかたちで保存できないため、データの共有はきわめて困難です。また、外部サーバーとの通信もできないので、それらのサーバーにデータを蓄積しておくことも不可能です。

これらは、おもにセキュリティのために設定された仕様上の制限ですが、おもしろいインターフェイスの開発にはひどく不便です。たとえば、キーワードをハイライト表示するブックマークレットは書けても、対象となるキーワードを指定してから Web ページを呼び出したり、自動的にキーワードをハイライト表示させたりすることはできません。

一方、Web ブラウザの挙動を変えるプラグインなどを使えば、このような制限をなくすことができます。たとえば、Firefox では〝拡張機能 (extensions)〟を利用する方法が考えられます。ただし、拡張機能の作成はブックマークレットのスクリプトの場合とくらべてかなり面倒で、手軽に作って使うというわけにはいきません。

世の中はよくしたもので、Firefox の拡張機能の 1 つである「Greasemonkey」を導入すれば、ブックマークレットを書くのと同じ程度の手軽さで、前記のような機能が実現できるようになります。

なお、あとで述べるように、この拡張機能の利用にはたいへん危険な側面もあります。この点を十分に理解したうえで使ってください。

Greasemonkey

Greasemonkey は Firefox に拡張機能としてインストールし、その上で動く JavaScript プログラムを登録して利用するかたちになります。簡単な JavaScript スクリプトを作成すれば、Firefox 上で次のような機能が使えるようになります。

- Web ページを開くと同時にスクリプトを実行する。
- 特定のサイト上でスクリプトを実行する。
- 永続的なデータを扱う。
- 他サイトと通信する。

要するに、ブックマークレットの制限がほとんどなくなり、〝なんでもあり〟状態になっています。

たとえば、http://example.com/以下のあらゆるページに対し、

```
document.body.innerHTML = '';
```

というスクリプトが実行されるように設定すると、http://example.com/以下のページはすべて空白になります。別のサイトから得た偽の情報をページに交ぜて表示したり、ユーザーのキー入力をフックして別サイトに送信することもできます。危険なスクリプトを動かすと、大変なことになる可能性があるわけです。

もっとも、Greasemonkey のスクリプトをブラウザに登録するのはユーザー自身ですし、ページ内容を自由に変更したり、外部と通信できることによって得られる効果は

図 3 Greasemonkey の状態を表すアイコン
(a) 無効 (b) 有効

絶大です。たしかに危険ではありますが、その点に注意して正しく使えば、それだけの価値はあると思います。

Greasemonkey 用のスクリプトは、いくつかのサイト[7]でさまざまなものが公開されています。

Greasemonkey のインストール

Greasemonkey は mozdev.org[8]で入手し、Firefox の拡張機能としてインストールします。

インストールすると、ブラウザの右下に図 3-a のようなアイコンが表示されます。これは Greasemonkey が無効 (disable) という意味ですが、クリックすると図 3-b のように有効 (enable) になります。

Greasemonkey スクリプト

以下では、簡単な Greasemonkey スクリプトと、永続的なデータを扱う例を紹介します。

大阪弁化フィルタ

リスト 1 は、さきほどの大阪弁化フィルタを Greasemonkey から呼び出すスクリプトです。これを Greasemonkey に登録しておくと、Web ページの内容を読み取った直後にスクリプトが実行され、大阪弁化フィルタのサイトに Web ページの URL が渡されます。

スクリプトのコメント部には、スクリプトの名前、説明、名前空間用の ID、スクリプトを適用する URL のパターンを指定します。リスト 1 の例では http://QuickML.com/が指定されているので、この URL 以下のページに大阪弁化フィルタが適用されます。

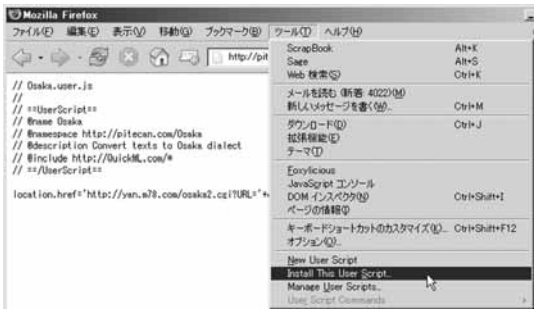
Firefox からこのスクリプトにアクセスし、ツールメニューで `Install The User Script´を選択して Greasemonkey に登録すると (図 4)、QuickML.com にアクセスするたびに図 2 のようなページが表示されるようになります。

7 http://userscripts.org/やhttp://dunck.us/collab/GreaseMonkeyUserScripts など。
8 http://greasemonkey.mozdev.org/

リスト 1 大阪弁化フィルタ呼出しスクリプト (Osaka.user.js)

```
// Osaka.user.js
//
// ==UserScript==
// @name Osaka
// @namespace http://pitecan.com/Osaka
// @description Convert texts to Osaka dialect
// @include http://QuickML.com/*
// ==/UserScript==

location.href='http://yan.m78.com/osaka2.cgi⇒
?URL='+encodeURIComponent(location.href);
```

図 4 Osaka.user.js のインストール



キーワードのハイライト

リスト 2 は、指定単語をハイライト表示する Greasemonkey スクリプトです。`location.href = ……´で指定したページに移動したとき、location.keyword で指定した文字列があればハイライト表示します。

このスクリプトでは、プログラム全体を無名関数で囲むことにより、グローバル変数がほかのプログラムと衝突しないようにしています。

リスト 3 の記述のあるページで span の部分をクリックすると、まず http://QuickML.com/が表示されます。そして、リスト 2 の Greasemonkey スクリプトが呼び出され、文字列 `メーリングリスト´がハイライト表示されます (図 5)。

リンクのクリック情報の利用

通常のブラウザでは、リンクがまったくクリックされていない状態と、クリック後の状態でリンクの色などを変えることができます。しかし、頻繁にクリックしているリンクとそうでないリンクの区別はできません。リンク集など

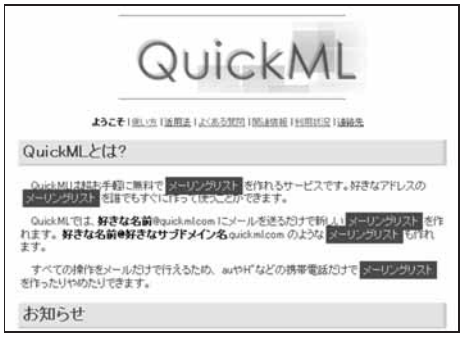
リスト 2　単語をハイライト表示するスクリプト (highlight.user.js)

```
// highlight.user.js
//
// ==UserScript==
// @name Highlight
// @namespace http://pitecan.com/Highlight
// @description Highlight a specified word.
// @include http://*
// @exclude http://pitecan.com/*
// ==/UserScript==

(function (){
  var keyword = location.keyword;
  if(keyword != undefined){
    hilighted = "<span style='background:blue;color:yellow;padding:4px;margin:2px;'>"
                +keyword+"</span>";
    body = document.body.innerHTML;
    body = body.split(keyword).join(hilighted);
    document.body.innerHTML = body;
  }
})();
```

リスト 3　単語ハイライト・スクリプトの呼出し

```
<script>
function hilight(url,keyword){
  location.keyword = keyword;
  location.href = url;
}
</script>

<span onclick='hilight("http://QuickML.com/","メーリングリスト");'>
http://QuickML.com/内の「メーリングリスト」という文字列をハイライトする
</span>
```

図 5　指定した文字列のハイライト表示



では、クリックされた回数に応じてリンクの色を変えるようにすれば、よく使うリンクをみつけやすくなるでしょう。また、クリックしたときの時刻も記録しておけば、どのリンクをいつ参照したかが分かるので、情報検索などに応用できそうです。

各リンクにおいて onclick に JavaScript プログラムを対応づければ、クリックされたことをプログラム側で把握できます。ただし、リンクのクリック数の保存や通知はできないので、上記のような情報は得られません。Greasemonkey には以下の 3 つの関数が用意されており、複数のセッション間でのデータのやりとりが可能になっています。

- GM_setValue()：永続的なデータを書き込む
- GM_getValue()：データを読み込む
- GM_xmlhttpRequest()：他サイトと通信する

GM_setValue() と GM_getValue() が扱うデータは、実際にはローカルのブラウザに格納されています。その点では Cookie と似ていますが、異なるサイトに接続している場合も同じ値にアクセスできるところが違います。

末尾のリスト 4 は、クリックされた回数に応じてリンクの背景色を変化させる Greasemonkey スクリプトです。

アクセス状況を背景色に反映させるには、2003 年 8 月号で紹介した Web ページの〝鮮度〟を視覚化する手法を使います。各リンクには、アクセス状況に応じた鮮度が定義されています。鮮度は、クリック状況と時間経過により変

図 6　クリック前

図 7　クリック後

図 8　さらにクリック

化していきます。Web ページへのアクセス履歴をすべて記録するとデータが膨大になる可能性があるので、以下の方針にもとづいて鮮度を計算することにしました。

- 鮮度は指数的に減少する（たとえば、1 日ごとに鮮度が半分になる）。
- ページにアクセスすると鮮度は指数的に回復する（たとえば、アクセスするごとに最大鮮度との差が半減する）。

この方針なら、Web ページごとに最終アクセス時刻とそのときの鮮度だけを記録しておき、GM_getValue() と GM_setValue() で読み書きすればよいことになります。

図 6 の Web ページで ``Google´´のリンクをクリックすると、URL に関連づけられた popularity の値が変化し、Google へのリンクの色が図 7 のように変わります。

ここでさらに Google のリンクをクリックすると、画面は図 8 のようになります。クリックされずに何日か経過すると、リンクの色は図 6 に戻ります。

GM_xmlhttpRequest() を利用し、onclick 時にほかのサーバーと通信して情報を送るようにすれば、クリック履歴のログを残すこともできます。

おわりに

以前に紹介したシステムのいくつかは、Greasemonkey を用いて実装することができます。たとえば、Web ページの作成時刻に応じて背景画像を変えれば、2003 年 8 月号の鮮度の視覚化手法を Greasemonkey で実現できるでしょう。あるいは、閲覧中の Web ページを解析し、内容の近いファイルを表示するようにすれば、2002 年 11 月号で紹介した近傍検索システムが作れます。

Internet Explorer で Greasemonkey と同様の機能を実現する Trixie[9]や、Greasemonkey スクリプトを Firefox 拡張機能に変換するコンパイラ[10]などもあり、Greasemonkey を取り巻く環境は便利になってきました。Ajax などの普及もあり、Web ブラウザ上で JavaScript を用いて処理をおこなう機会はますます増えそうです。その意味でも、快適なブラウザ操作を可能にする Greasemonkey のような仕組みに注目していきたいと思っています。

（ますい・としゆき　産業技術総合研究所）

9 http://www.bhelpuri.net/Trixie/Trixie.htm
10 http://www.letitblog.com/greasemonkey-compiler/

リスト 4　クリックされた回数に応じて背景色を変えるスクリプト（ColorLink.user.js）

```
// ColorLink.user.js
// 2005.11
//
// ==UserScript==
// @name ColorLink
// @namespace http://pitecan.com/ColorLink
// @description Change the background color of links based on clicking history
// @include http://*
// ==/UserScript==

// セーブデータ例: "2005,10,3,0,0,0,1234"

(function (){
  var popularity;

  function setAttr(key,date,val){
    year = date.getYear();
```

```
    if(year < 2000) year += 1900;
    month = date.getMonth() + 1;
    day = date.getDate();
    hour = date.getHours();
    min = date.getMinutes();
    sec = date.getSeconds();
    s = year+","+month+","+day+","+hour+","+min+","+sec+","+val;
    GM_setValue(key, s);
  }

  function getAttr(key){
    s = GM_getValue(key,"2000,1,1,0,0,0,0");
    a = s.split(',');
    year = parseInt(a[0]);
    month = parseInt(a[1]) - 1;
    day = parseInt(a[2]);
    hour = parseInt(a[3]);
    min = parseInt(a[4]);
    sec = parseInt(a[5]);
    val = parseInt(a[6]);
    date = new Date(year,month,day,hour,min,sec);
    return [date,val];
  }

  function getCurrentPopularity(key){
    a = getAttr(key);
    date = a[0];
    popularity = a[1];
    currentdate = new Date;
    days = (currentdate - date) / (24 * 60 * 60 * 1000);
    val = popularity * Math.pow(0.8, days); // 1日ごとにpopularityが0.8倍になる
    return val;
  }

  function bgcolor(val){  // valは0〜9999
    r = "ff";
    g = "ff";
    b = Math.floor(0xe0 - (val / 10000.0) * 0xe0).toString(16);
    b = ("0" + b).substr(-2,2);
    return "#" + r + g + b;
  }

  var allElements, thisElement;
  allElements = document.getElementsByTagName("a");
  for (var i = 0; i < allElements.length; i++) {
    thisElement = allElements[i];
    if(thisElement.onclick == undefined){
      thisElement.onclick = function(event){
        url = event.target;
        popularity = getCurrentPopularity(url);
        newpopularity = popularity + (10000 - popularity) / 2;
        newdate = new Date;
        setAttr(url,newdate,newpopularity);
      }
    }
    url = thisElement.href;
    popularity = getCurrentPopularity(url);
    thisElement.style.backgroundColor= bgcolor(popularity);
  }
})();
```