Dive into Greasemonkey

Mark Pilgrim

2005-05-09

目 次

本書は http://diveintogreasemonkey.org/で配布されています．それ以外のところでこれを読んでいる場合，最新版ではない可能性があります．

第1章　始める

1.1　Greasemonkey とは何か

Greasemonkey はあなたが訪問するウェブページを変更するスクリプトを書くことを可能にする Firefox 拡張です．Greasemonkey をつかってウェブサイトをより読みやすくしたりより使いやすくしたりすることができます．サイト管理者自身が面倒で修正していないようなレンダリングのバグをあなたが修正することができます．ウェブページを読み上げたりブライユ点字に変換したりする補助技術をつかって，ウェブページがよりよくなるように変更を加えることもできます．ほかのサイトから自動的にデータを取り出して二つのサイトをより相互に関連したものにすることさえできます．

Greasemonkey だけでは上のようなことは何一つできません．実際，インストール後，何も変化がないようにみえるでしょう．変化が見えるのは「ユーザースクリプト」と呼ばれるものをインストールし始めてからです．ユーザースクリプトとは，Javascript のコードの集まりで，さらに Greasemonkey がいつどこでそれを実行すればよいのかという情報をもっています．一つのユーザースクリプトは特定のページ，特定のサイト，あるいはいくつかのサイトをターゲットにできます．ユーザースクリプトは Javascript ができることなら何でもできます．それどころか，ユーザースクリプトは Greasemonkey が提供する特別な関数がつかえるので，Javascript 以上のことができるのです．

いろんな人がそれぞれの用途のために書いた何百ものユーザースクリプトを含む Greasemonkey スクリプトレポジトリがあります．自分の書いたユーザースクリプトが他の人にも有用かもしれないと思ったら，このレポジトリに加えることができます．あるいは，公開せずに，自分のブラウジングを少し改善したことに満足していることもできます．

Greasemonkey メーリングリストもあります．ここでは，質問したり，ユーザースクリプトをアナウンスしたり，新しい機能についてのアイディアを議論したりすることができます．Greasemonkey の開発者はリストを頻繁に見ています．彼らはあなたの質問に答えてくれることさえあるかもしれません．

1.2　この本の用途

本書は Greasemonkey メーリングリストでの議論，およびユーザースクリプトを書く際の自分自身の経験から生まれたものです．リストを見るようになって一週間もすると，既に回答を与えられた質問が新しいユーザーになされるのを見るようになりました．そん

なんにたくさんのユーザースクリプトを書いた訳ではなくても，共通のパターンやよく直面する特定の問題を解決するために再利用可能なたくさんのコードを目にしました．私は最も有用なパターンを書き起こし始め，自分自身のコードの決定を説明し，その過程でできる限りのことを学びました．

Greasemonkey 開発者である Aaron Boodman と Jeremy Dunck，初期の草稿に対して貴重なフィードバックをしてくれたその他の人々の助けがなければ，本書は今日の半分にも満たない出来になっていたでしょう．上記全ての人に感謝します．

1.3 Greasemonkey をインストールする

ユーザースクリプトを書き始めるには，まずバージョン 0.3 以降の Greasemonkey ブラウザ拡張をインストールする必要があります．

1. Greasemonkey ホームページにいきます
2. "Install Greasemonkey"というリンクをクリックします
3. Firefox が（おそらくウィンドウの上部に）ソフトのインストールを妨げたという警告を出します．「編集」をクリックして，"許可サイト"ダイアログを出し，「許可する」をクリックしてソフトウェアのインストールを許可するサイトに Greasemonke のサイトを加えます[1]．OK をクリックして"許可サイト"ダイアログを閉じます
4. 再び"Install Greasemonkey"というリンクをクリックします．
5. 今度はインストールダイアログがでて，本当にインストールするかどうか確認を求められます．インストールボタンが有効になるまで数秒待って，「インストールする」をクリックします．
6. ブラウザを再起動します

再起動したら「ツール」をえらびます．3 つの項目が見えるはずです："Install User Script...", "Manage User Scripts...", そして"User Script Command"です．"Manage User Scripts..."のみが有効になりますが，これで OK です．ほかのものは特定の環境でのみ有効になります．

デフォルトでは Greasemonkey をインストールしてもブラウザには（上の 3 つのメニュー項目以外に）なんの機能も追加されません．インストールによって出来るようになるのは，特定のページをカスタマイズする「ユーザースクリプト」というものをさらにインストールすることをだけです．

[1] ボタンの日本語は適当なので，適宜読み替えてください

1.4 ユーザースクリプトをインストールする

Greasemonkey のユーザースクリプトは Javascript で書かれた単一のファイルで，一つあるいはそれ以上のウェブページをカスタマイズします．

ヒント

Greasemonkey スクリプトレポジトリにはたくさんのユーザースクリプトがありますが，ここにあなたのスクリプトを加えなければならないということはありません．スクリプトはどこに置いてもかまいませんし，どこからインストールしてもかまいません．ウェブサーバーをもつことすら必要ありません; ローカルファイルからインストールすることもできます．

注意

ユーザースクリプトは`.user.js`で終わらなければなりません．

私が最初に書いたユーザースクリプトの名前は”Butler”です．これは Google の検索結果に機能を追加します．

手順

1. Butler のホームページにいって Butler の追加する機能についての簡潔な説明を読みます．（全てのユーザースクリプトにホームページがあるわけではありません．Greasemonkey にとって重要なのはスクリプト自体です．）
2. ”Download version...”(翻訳の時点では 0.3) というリンクをクリックすると，Greasemonkey がインストールするかどうか聞いてきます[2]
3. ”Install”をクリックして完了です．

うまくいけば Greasemonkey は”Success! Refresh page to see changes”というポップアップアラートを出します．

これで，Google で何か検索してみてください．検索結果のページの最上部に”Try your search on: Yahoo, Ask Jeeves, AlltehWeb,...”という行があるはずです．また，”Enhanced by Butler”というバナーも表示されます．これら全てが Butler によって追加されたものです．

参考サイト

- Greasemonkey スクリプトレポジトリには何百ものユーザースクリプトがあります．

[2]原文ではスクリプト本体が表示されて云々と続きますが，現在の仕様ではインストールするかどうかのダイアログが表示されるようになっています．

1.5　スクリプトを管理する

インストールできるGreasemonkeyスクリプトの数に制限はありません．Greasemonkeyはスクリプトを管理するための，設定をグラフィカルに表示するダイアログをもっています．一時的にスクリプトを無効にしたり，構成を変えたり，完全にアンインストールしたりすることが出来ます．

手順 (一時的に Butler を無効にする)

1. メニューから**ツール→Manage User Scripts...**を選択するとGreasemonkeyは"Manage User Scripts"というダイアログを出します．
2. 大ログの左側にはインストールしたスクリプトのリストがあります．（もし，最初からずっとこの本の通りに操作してきたのなら，Butlerのみが表示されているでしょう．）
3. もし既に選択されていなければButlerをリストから選んで，**Enabled**のチェックを外してください．左側のButlerの色が微妙に黒から灰色に変わるはずです．（これはまだ選択されているときには見にくいですが，たくさんのスクリプトをインストールしているにより役に立つでしょう．）
4. Okをクリックして管理ダイアログを抜けます．

こうすると，Butlerはインストールされていて，かつ無効という状態になります．Googleで何かを検索してみることでこれを確かめられます．最上部に"Enhanced by Butler"という表示はもう現れないはずです．上の操作を繰り返し，管理ダイアログで**Enabled**をチェックすることで再びButlerを有効にすることが出来ます．

注意

"一時的に"とはいいましたが，もう一度明示的に有効にしない限りButlerは無効のままです．"一時的"というのは元のスクリプトを探したり再インストールすることなく，再び有効にすることが出来るということです．

"Manage User Scripts"ダイアログをつかってスクリプトを完全にアンインストールすることもできます．

手順

1. メニューから**ツール→Manage User Scripts...**を選択するとGreasemonkeyは"Manage User Scripts"というダイアログを出します．
2. 左の枠からButlerを選択し，**Uninstall**をクリックします．確認は求められず，スクリプトは即アンインストールされます．
3. ステップ3は…ないです（Jeff Goldblumへ，ごめんなさい）

まだあります．以前インストールしたスクリプトの設定を変えることも出来ます．Butler を最初にインストールしたときにでてきたダイアログを覚えてますか？そこにはスクリプトを有効にするサイトと除外にするサイトとの 2 つのリストがありました．これらのパラメータをインストール時，あるいは”Manage User Scripts”ダイアログによって好きなときに変更することが出来ます．

例えば，あなたが Butler を気に入ったけれども Froogle-Google の製品比較のサイト-で使うことはないとしましょう．ほかの Google サイトで Butler を有効にしたままこのサイトを除外するように設定することが出来ます．

手順

1. メニューから**ツール→Manage User Scripts...** を選択すると Greasemonkey は”Manage User Scripts”というダイアログを出します．

2. 左側の Butler を選択すると，右枠に二つのリストが表示されるはずです．一方は Butler を有効にするサイト (http://*.google.*/*) でもう一方は除外するリスト (空) です．

3. ”Excluded pages”の隣の **Add** をクリックします．

4. Greasemonkey は”Add Page”という 2 つ目のダイアログを出し，新しい URL を入力するよう求めます．**http://froogle.google.com/*** を入力し，**OK** をクリックします，

5. ”Manage User Scripts”ダイアログに戻ると，除外すべきページのリストには新しい URL**http://froogle.google.com/*** が含まれているはずです．これは Butler が froogle.google.com のいかなるページでも実行されないことを意味します．アスタリスクは単純なワイルドカードで，ドメイン名，パス，あるいは URL スキーム (http://) など，URL のどの部分でもつかえます．

6. **OK** をクリックし”Manage User Scripts”ダイアログから抜け，Froogle で商品を検索してみると Butler が実行されないことがわかります．通常のウェブ検索，イメージ検索などほかの Google サイトではまだ有効なはずです

第2章　最初のユーザースクリプト

2.1 Hello World

Greasemonkey というすばらしい世界への千里の道も一歩から始まります．そしてそれは，技術マニュアルに親しんでいる人にはおなじみのように，コンピュータに"Hello World"と表示させることです．

リスト 1: helloworld.user.js

```
// Hello World! example user script
// version 0.1 BETA!
// 2005-04-22
// Copyright (c) 2005, Mark Pilgrim
// Released under the GPL license
// http://www.gnu.org/copyleft/gpl.html
//
// --------------------------------------------------------------------
//
// This is a Greasemonkey user script.
//
// To install, you need Greasemonkey: http://greasemonkey.mozdev.org/
// Then restart Firefox and revisit this script.
// Under Tools, there will be a new menu item to "Install User Script".
// Accept the default configuration and install.
//
// To uninstall, go to Tools/Manage User Scripts,
// select "Hello World", and click Uninstall.
//
// --------------------------------------------------------------------
//
// ==UserScript==
// @name          Hello World
// @namespace     http://diveintogreasemonkey.org/download/
// @description   example script to alert "Hello world!" on every page
// @include       *
// @exclude       http://diveintogreasemonkey.org/*
// @exclude       http://www.diveintogreasemonkey.org/*
// ==/UserScript==

alert('Hello world!');
```

見ての通り，Hello World スクリプトのほとんどはコメントです．これらのうち，インストールの仕方などのようなものは特別な意味はありません．それらは単にこれがなんなのかわからないエンドユーザのためのものです．しかし，特別な意味が**ある**コメントも含まれています．これについては次節で述べます．

スクリプトが動いているのを見るためには，通常通りインストールして diveintogreasemonkey.org ドメイン以外のページ（たとえば Google）を開いてください．ページは通常通り開きますが，"Hello world!"というアラートが出てくるはずです．

ダウンロード

- helloworld.user.js

2.2 メタデータによるスクリプトの説明

リスト 2: Hello World メタデータ

```
// ==UserScript==
// @name           Hello World
// @namespace      http://diveintogreasemonkey.org/download/
// @description    example script to alert "Hello world!" on every page
// @include        *
// @exclude        http://diveintogreasemonkey.org/*
// @exclude        http://www.diveintogreasemonkey.org/*
// ==/UserScript==
```

ここには Gresemonkey 特有のコメントで囲まれた 6 つのメタデータがあります．順番に見ていくことにしましょう．最初は囲みからです．

```
// ==UserScript==
//
// ==/UserScript==
```

これらのコメントには意味があり，きちんとマッチしていなければなりません．Gresemonkey はそれらを使ってあなたのユーザースクリプトのメタデータの始めと終わりを知らせます．この項目はスクリプトのどこで定義してもよいのですが，普通最初のほうに書かれます．

上の Gresemonkey メタデータ項目の中で，最初のものは名前です

```
// @name           Helllo World
```

これはスクリプトの名前で，最初にインストールしたときにダイアログに表示され，また"Manage User Scripts"ダイアログの表示にも使われます．名前は短く，またスクリプトの内容をわかりやすく説明したものであるべきです．

`@name` は定義しなくてもかまいません．定義する場合は，一度だけ記述することが許されます．定義しなければ，デフォルトでスクリプトのファイル名から `.user.js` を除いたものになります．

つぎに名前空間です．

```
// @namespace     http://diveintogreasemonkey.org/download/
```

これは URL で，Greasemonkey はこれを使って異なる作者による同じ名前のユーザースクリプトを区別します．もしあなたがドメイン名をもっているなら，それ（あるいはサブディレクトリ）を使えばよいでしょう．そうでなければ tag: URI が使えます．

`@namespace` も任意項目です．もし定義されているなら，一度だけ現れることが許されます．定義されていない場合，デフォルトはユーザーがスクリプトをダウンロードしたドメインになります．

ヒント

ユーザースクリプトのメタデータ項目は，どんな順序で指定してもかまいません．私は `@name`, `@namespace`, `@description`, `@include`, 最後に`@exclude` という順序が好みですが，この順番が特別ということはありません．

次はスクリプトの説明です．

```
// @description   example script to alert "Hello world!" on every page
```

これは人間が理解できるユーザースクリプトが何をするかの説明です．最初にインストールしたとき，また"Manage User Scripts"ダイアログのなかで表示されます．これは 2 行以下であるべきです．

`@description` は任意項目です．もしあれば，現れるのは一度だけです．なければデフォルトは空文字列に設定されます．

重要

`@description` を忘れないようにしましょう．自分専用のユーザースクリプトを書いているとしても，それが何十にも及んだとき，スクリプトの説明を書いていないと，全てを"Manage User Scripts"で管理するのがずっと大変になります．

次の3行は(Greasemonkeyから見て)最も重要なものです．@includeおよび@excludeURLです．

```
// @include       *
// @exclude       http://diveintogreasemonkey.org/*
// @exclude       http://www.diveintogreasemonkey.org/*
```

これらの行はGresemonkeyにどのサイトであなたのユーザースクリプトを実行すべきなのか教えます．どちらもURLを指定し，*をドメイン名あるいはパスの単純なワイルドカードとして使用します．上の場合，Greasemonkeyにhttp://diveintogreasemonkey.org/およびhttp://www.diveintogreasemonkey.org/*以外の全てのサイトでHello Worldスクリプトを実行するように指示しています．@excludeは@includeより優先されます．このため，http://diveintogreasemonkey.org/は*(全てのサイト)にマッチしてもhttp://diveintogreasemonkey.org/*にマッチするために除外されます．

@includeも@excludeも任意項目です．除外するURLとスクリプトを実行するURLは好きな数だけ指定できますが，それぞれについて指定する行を記述しなければなりません．どちらも指定されていなければ，Greasemonkeyはスクリプトを全てのサイトで実行します（@include *と書いたのと同じことです）．

注意

@includeと@excludeの記述は曖昧さのないものである必要があります．Greasemonkeyはユーザにとっては等価であるように見えるURLについて何の仮定もしません．あるサイトがhttp://example.com/とhttp://www.example.com/両方に反応するなら，二つとも記述する必要があります．

参考サイト

- tag: URIs

2.3 ユーザースクリプトを書く

最初のスクリプトは単に実行時に”Hello World!”と表示するだけです．

リスト 3: ”Hello World!”を表示する

```
alert('Hello world!');
```

このコードは極単純に見え，期待通りに動きますが，実際にはGreasemonkeyは裏でいくつかのことをして，ユーザースクリプトが元のページで定義されているほかのスクリプトと相互に悪影響を及ぼさないようにしています．具体的には，ユーザースクリプトは自

動的に無名関数にくるまれて実行されます．通常はこのことを気にする必要はありませんが，あとで問題になることがあるかもしれないので，ここで知っておくのもよいでしょう．

問題になる最もありふれた例はユーザースクリプトで定義された変数と関数はほかのスクリプトでは使えないということです．実際，それらの変数や関数はいったんユーザースクリプトが実行され終えてしまった後には使うことは出来ません．つまり，自分で定義した関数を `window.setTimeout` をつかって利用したり，またリンクの文字列を値にとる属性 `onclick` をセットして Javascript に自分で定義した関数を評価させたりすることが出来ると思っていると，問題に直面することになります．

例えば，下のユーザースクリプトは関数 helloworld を定義し，タイマーを設定して一秒後にそれを呼び出そうとします．

リスト 4: 関数呼び出しを遅らせる悪い方法

```
function helloworld() {
    alert('Hello world!');
}

window.setTimeout("helloworld()", 60);
```

これはうまく動きません．なんのアラートも表示されません．Javascript コンソールを開いてみると `Erro:  helloworld is not defined` という例外が表示されているでしょう．これは，タイムアウトして helloworld() への呼び出しが評価されるときには関数 helloworld がもはや存在しないからです．

もしあとでユーザースクリプトの変数や関数を参照する必要があるなら，それらを `window` オブジェクトの属性として明示的に定義する必要があります．

リスト 5: 関数呼び出しを遅らせるより良い方法

```
window.helloworld = function() {
    alert('Hello world!');
}

window.setTimeout("helloworld()", 60);
```

これは期待通り動きます．ページが読み込まれた一秒後に，"Hello World!"というアラートが見事に表示されます．

しかし，`window` の属性を設定するのは理想的ではありません．これは局所的な変数で済むところで大域的な変数を用いるようなものです．（実際これは，`window` は大域的でページのあらゆるスクリプトから使えるので，まさにそういうことです．）より現実的には，ページで定義されているほかのスクリプト，あるいはほかのユーザースクリプトの邪魔をしてしまうかもしれません．

最も良い方法は無名関数を自分で定義してそれを `window.setTimeout` の第一引数としてあたえることです．

リスト 6: 関数呼び出しを遅らせる最も良い方法

```
window.setTimeout(function() { alert('Hello world!') }, 60);
```

ここでは名前なしの関数（無名関数）を作り，その関数自体を `window.setTimeout` に渡しています．こうすると前の例と同じことを達成し，しかも跡を残しません．つまり，ほかのスクリプトに影響しません．

私がユーザースクリプトを書くとき，無名関数を普通使います．"一度限りの"関数を作って `window.setTimeout`, `document.addEventListener` などの引数として渡したり，`click` や `submit` などのイベントハンドラに割り当てたりするにはそれが理想的です．

参考サイト

- Anonymous functions in Javascript
- Block Scope in Javascript とそれに関連する議論

2.4 ユーザースクリプトを編集する

スクリプトの作者のために，"Manage user Script"ダイアログに有用な機能があります．"動いたまま"インストールされたスクリプトを編集するための **Edit** ボタンです．

手順

1. メニューから**ツール→Manage User Scripts...** を選択すると Greasemonkey は"Manage User Scripts"というダイアログを出します．
2. 左の枠から Hello World を選んで **Edit** をクリックします．Hello World のインストールされたバージョンがあなたのお気に入りのテキストエディタで開かれるはずです．（もし開かれなければ `.js` という拡張子のファイルがお気に入りのテキストエディタに関連づけられていることを確かめてください．）
3. アラートの文章を"Live editing!"に変更してください．
4. エディタで変更を保存し，ブラウザに戻ってページを更新して変更を確かめてください．すぐに変更の結果が分かるはずです．ユーザースクリプトを再インストールしたり，"再読み込み"したりする必要はありません．"動いたまま"編集できるのです．

ヒント

"Manage User Scripts"ダイアログで **Edit** ボタンをクリックすると，Firefox のプロフィールディレクトリの奥深くにあるスクリプトを編集することになります．私は，"生"編集

を終えたらテキストエディタに戻って**ファイル→ 名前を付けて保存**を選んでユーザースクリプトを別のディレクトリに保存することにしています．これは必須ではないですが (Greasemonkey がみるのはプロフィールディレクトリのなかのものだけです)，自分のほかのスクリプトと一緒に自分のスクリプトの”マスターコピー”を別ディレクトリに保存しておくのが私の好みです．

第3章　ユーザースクリプトのデバグ

3.1　Javascript コンソールによるエラーのトラッキング

もしユーザースクリプトが動かないようだったら，まず Javascript コンソールをチェックしましょう．ユーザースクリプトを含む全てのスクリプトに関わるエラーがリストされています．

手順

1. Firefox のメニューから**ツール→ エラーコンソール**を選びます．
2. コンソールにはあなたが Firefox を起動してから訪れた全てのページにおけるスクリプトエラーが全てリストしてあります．かなりたくさんあるかもしれません（多数の有名なサイトがごく普通にスクリプトエラーをおこしていることに驚くでしょう）．**消去**をクリックしてからユーザースクリプトのデバグをはじめてください．

ユーザスクリプトがうまく動作しないテストページを再読み込みしてください．もし本当にエラーを起こしているならば，Javascript コンソールに例外が表示されるはずです．

注意

ユーザスクリプトがエラーを起こしている場合，Javascript コンソールは例外と行番号を表示します．この行番号は実際にはあまり役に立たないので無視した方がよいでしょう．これは Gresemoneky がユーザスクリプトをページに埋め込む方法によるものです．表示される行番号はユーザスクリプト内の例外が発生している場所では**ない**のです．

3.2　GM_log による記録

Greasemonkey はログをとる関数 `GM_log` を用意しています．これによって Javascript コンソールにメッセージを書き込むことが出来ます．そのようなメッセージは，リリースの前には取り除かれるべきですが，デバグには非常に役に立ちます．さらに，コンソールがログメッセージで埋まっていくのを見る方が，コードのあちこちに点在する警告をみて何度も OK をクリックするよりはましです．

`GM_log` はログに残すべき文字列をただ一つの引数としてとります．Javascript コンソールにログを残してから，ユーザスクリプトは通常通り動きます．

リスト 7: Javascript コンソールに書き込んで実行を続ける

```
if (/^http:\/\/diveintogreasemonkey\.org\//.test(window.location.href)) {
    GM_log('running on Dive Into Greasemonkey site w/o www prefix');
} else {
    GM_log('running elsewhere');
}
GM_log('this line is always printed');
```

このユーザスクリプトをインストールしてから http://diveintogreasemonkey.org/ を開くと，次の 2 行が Javascript コンソールに書きこまれます．

```
Greasemonkey: http://diveintomark.org/projects/greasemonkey//Test Log: \
    running on Dive Into Greasemonkey site w/o www prefix
Greasemonkey: http://diveintomark.org/projects/greasemonkey//Test Log: this \
    line is always printed
```

見ての通り，ユーザスクリプトのメタデータ部分からとってきた名前空間とスクリプト名を書き込み，それから `GM_log` の引数として与えた文字列を書き込みます．

http://diveintogreasemonkey.org/ 意外のどこかを開くと，次の 2 行が Javascript コンソールに現れます．

```
Greasemonkey: http://diveintomark.org/projects/greasemonkey//Test Log:
running elsewhere
Greasemonkey: http://diveintomark.org/projects/greasemonkey//Test Log:
this line is always printed
```

私はログメッセージの長さの限界を調べましたが，分かりませんでした．255 文字までは大丈夫です．さらに，Javascript コンソールの中の行はきちんと収まり，よってスクロールして残りのログメッセージを見ることが出来ます．ログ取りに夢中になってください．

ヒント

Javascript コンソールでは，右クリック（Mac ユーザーは Ctrl+クリック）で行を選択して，コピーを選ぶとクリップボードにその行をコピーすることが出来ます．

3.3 DOM Inspector で要素を調べる

DOM Inspector を使うと，任意のページの解析されたドキュメント・データ・モデル (DOM) を探索することが出来ます．それぞれの HTML 要素，属性，テキストの詳細を取得できます．そのページのそれぞれスタイルシートによる CSS 規則を見ることが出来ます．スクリプトが使えるオブジェクトの属性を調べることも出来ます．これはとても強力です．

DOM Inspector は Firefox のインストールプログラムに含まれていますが，プラットフォームによってデフォルトではインストールされていないかもしれません．もし**ツール**メニューに DOM Inspector がなかったら，もう一度 Firefox を再インストールする必要があります．既存のブックマーク，環境設定，拡張，ユーザスクリプトは再インストールの影響は受けません．

手順 (Dom Inspector のインストール)

1. Firefox のインストールプログラムを実行します．
2. ライセンスに同意して，**カスタムインストール**をクリックします．
3. インストール先ディレクトリを選択した後，インストールウィザードに追加コンポーネントをインストールするか聞かれます．**Developer Tools**[1]
4. インストールが終わったら Firefox をきどうします．**ツール**メニューに新しく **Dom Inspector** が加わってるはずです．

このツールの強力さを知るために，Dive into Greasemonkey のホームページの DOM を調べてみましょう．

手順

1. http://diveintogreasemonkey.org/ を開きます．
2. メニューから**ツール**→**DOM Inspector** を選び，DOM Inspector を開きます．
3. DOM Inspector ウィンドウの左側に DOM ノードのリストが出てくるはずです．もしそうでない場合は左上にあるメニューから **DOM Nodes** を選びます．
4. `HTML` 要素を開いてみると 3 つの要素が現れます．`HEAD` と `#text` と `BODY` です．`BODY` が `diveintogreasemonkey-org` という `id` をもっていることに注意してください．これが見えない場合は幅を調節してください．
5. `BODY` を開くと 5 つの要素が見えます．`#text`, `DIV id="intro"`,`#text`, `DIV id="main"`, `#text`.
6. `DIV id="intro"` を開くと，`#text` と `DIV class="sectionInner"` の 2 つが現れます．
7. `DIV class="sectionInner"` を開くと `#text` と `DIV class="sectionInner2"` の 2 つが見えます．
8. `DIV class="sectionInner2"` を開くと，`#text`, `DIV class="s"`,`#text`, `DIV class="s"`, `#text`．5 つが見えます．
9. 最初の `DIV class="s"` を開くと `#text`, `H1`, `#text`, `P`, そして `#text` の 5 つが見えます．

[1] 訳注：日本語だとどうなってるか未確認

10. `H1` を選んでください．（DOM Inspector の後ろにある）本来のページで，`H1` 要素が赤い枠でちょっと点滅するはずです．右枠では以下のものが見えます：ノードの名前 (`H1`)，URI 名前空間（HTML は名前空間をもたないので空になっています．`application/xhtml+xml` と指定されているページあるいはほかの名前空間で XML を表示しているページにおいてのみ使われます），ノードの型（`1` は要素），ノードの値（空．ヘッダは値をもたないため．ヘッダーの中のテキストはそれ自身で一つのノードになっています）．

11. 右側の上部にあるドロップダウンメニューには，いくつかの選択肢があります：**DOM Node**, **Box Model**, **XBL Binding**, **CSS Rules**, **Computed Style**, **Javascript Object**．これらは現在選択されているノードについての様々な情報を提供します．これらのいくつかは編集が可能で，変更はすぐにもとのページに反映されます．**Javascript Object** を選ぶと，選択されている `H1` 要素の，スクリプトが使える属性とメソッドを全て見ることが出来ます．

12. **CSS Style Rules** を選んでください．右側が 2 つにわかれてます．上は要素に影響するルール（ブラウザに組み込まれているデフォルトのルールを含む）のリスト，下はそれらによって定義されている性質を示しています．

13. 右上の枠から二つ目のルールを選んでください．このルールは http://diveintogreasemonkey.org/css/dig.css で定義されています．

14. 右下の枠で `font-variant` をダブルクリックして **normal** という値を新しく入力します．(DOM Inspector の後ろの) 元のページで，”Dive Into Greasemonkey”のロゴが直ちにスモールキャピタルから通常の大文字と小文字に変わるはずです．

15. 右下枠のなかで右クリック（Mac ユーザは Ctrl-クリック）して **New Property** を選びます．”New Style Rule”というダイアログがでます．`background-color` と入力して **OK** をクリックし，さらに `red` を入力して **OK** をクリックします．新しい属性と値が既存の属性と一緒に右下の枠に現れ，元のページのロゴの背景が赤くなります．

DOM ノードの各階層をクリックして進んでいくのが気に入らなければ Inspect Element 拡張をぜひお勧めします．これを使うと DOM Inspector で特定の要素に直接アクセスできます．

警告

Inspect Element 拡張をインストールする前に DOM Inspector をインストールをしておかなければなりません．そうしないと，起動時に Firefox が落ちるようになります．もしこうなってしまったらコマンドラインウィンドウで Firefox をインストールしたディレクトリにいって `firefox -safe-mode` とうちます．Firefox が拡張を読み込ますに立ち上がるので，**ツール→ 拡張**を選び Inspect Element をアンインストールします．

手順 (Inspect Element で直接要素を調べる)

1. Inspect Element のダウンロードページに行き，**Install Now** をクリックします．
2. Firefox を再起動します．
3. http://diveintogreasemonkey.org/ を再び開きます．
4. ロゴの上で右クリック（Mac では Ctrl-クリック）します．
5. コンテキストメニューから **Inspect Element** をえらぶ．DOM Inspector が `H1` が既に選択された状態で開き，すぐに属性を調べたり編集したりできるはずです．

警告

DOM Inspector はあなたがブラウズしていくのに「ついて」いきません．DOM Inspector をひいき元のウィンドウでどこか別のところに進むと，DOM Inspector はとても混乱します．いきたいところにいき, [DOM Inspector で] 調べたいことを調べたら, DOM Inspector を閉じてからほかのことをするのがいいです．

参考サイト

- Introduction to DOM Inspector
- Inspect Element extension
- Inspector Widget extension，コンテキストメニューの代わりにツールバーボタンを追加する Inspect Element の代替拡張．

3.4 Javascript シェルによる式の評価

Javascript シェルとは現在のページの文脈で任意の Javascript 式を評価することを可能にするブックマークレットです．

手順

1. Jesse のウェブ開発ブックマークレットを開きます．
2. **Shell** ブックマークレットをブックマークツールバーにドラッグします．
3. どこかのページ（例えば http://diveintogreasemonkey.org ホームページ）を開いて，ブックマークツールバーの Shell をクリックします．すると Javascript シェルのウィンドウが後ろに開きます．

Javascript シェルをつかって DOM Inspector と同じことが，より自由な形式の環境で出来ます．DOM にたいするコマンドラインのようなものと思えばよいでしょう．では使ってみましょう．

```
document.title
Dive Into Greasemonkey
document.title = 'Hello World'
Hello World
var paragraphs = document.getElementsByTagName('p')
paragraphs
[object HTMLCollection]
paragraphs.length
5
paragraphs[0]
[object HTMLParagraphElement]
paragraphs[0].innerHTML
Teaching an old web new tricks
paragraphs[0].innerHTML = 'Live editing, baby!'
Live editing, baby!
```

変更は Enter を押すとすぐにもとのページに反映されます.
もう一つ触れておきたいのは Javascript シェルの props 関数です.

リスト 8: 要素の属性を取得する

```
var link = document.getElementsByTagName('a')[0]
props(link)
Methods of prototype: blur, focus
Fields of prototype: id, title, lang, dir, className, accessKey,
charset, coords, href, hreflang, name, rel, rev, shape, tabIndex,
target, type, protocol, host, hostname, pathname, search, port,
hash, text, offsetTop, offsetLeft, offsetWidth, offsetHeight,
offsetParent, innerHTML, scrollTop, scrollLeft, scrollHeight,
scrollWidth, clientHeight, clientWidth, style
Methods of prototype of prototype of prototype: insertBefore,
replaceChild, removeChild, appendChild, hasChildNodes, cloneNode,
normalize, isSupported, hasAttributes, getAttribute, setAttribute,
removeAttribute, getAttributeNode, setAttributeNode,
removeAttributeNode, getElementsByTagName, getAttributeNS,
setAttributeNS, removeAttributeNS, getAttributeNodeNS,
setAttributeNodeNS, getElementsByTagNameNS, hasAttribute,
hasAttributeNS, addEventListener, removeEventListener, dispatchEvent,
compareDocumentPosition, isSameNode, lookupPrefix, isDefaultNamespace,
lookupNamespaceURI, isEqualNode, getFeature, setUserData, getUserData
Fields of prototype of prototype of prototype: tagName, nodeName,
nodeValue, nodeType, parentNode, childNodes, firstChild, lastChild,
```

```
previousSibling, nextSibling, attributes, ownerDocument, namespaceURI,
prefix, localName, ELEMENT_NODE, ATTRIBUTE_NODE, TEXT_NODE,
CDATA_SECTION_NODE, ENTITY_REFERENCE_NODE, ENTITY_NODE,
PROCESSING_INSTRUCTION_NODE, COMMENT_NODE, DOCUMENT_NODE,
DOCUMENT_TYPE_NODE, DOCUMENT_FRAGMENT_NODE, NOTATION_NODE,
baseURI, textContent, DOCUMENT_POSITION_DISCONNECTED,
DOCUMENT_POSITION_PRECEDING, DOCUMENT_POSITION_FOLLOWING,
DOCUMENT_POSITION_CONTAINS, DOCUMENT_POSITION_CONTAINED_BY,
DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC
Methods of prototype of prototype of prototype of prototype of prototype: \
    toString
```

何だこれと思うでしょうが，これは Javascript において使える <a> 要素の属性とメソッドを DOM オブジェクト階層ごとにグループ化してリストしたものです．(`blur` や `focus` のようなメソッド，また `href` や `hreflang` のような属性といった）リンク要素に特有のメソッドや属性が最初にリストされ，(`insertBefore` メソッドのような）あらゆるタイプの要素に共通のメソッドや属性がそのあとに続きます．

繰り返すと，DOM Inspector を使ったときと同じ情報にアクセスできますが，こんどはポインタを使ったりクリックしたりするのではなく，タイピングと実験によってそれが可能になるのです．

警告

DOM Inspector のように，Javascript シェルもあなたのブラウジングに「ついて」いきません．Javascript シェルを開いてどこか別のページにいくと，Javascript シェルは混乱します．いきたいところにいって，Javascript シェルを開き気の済むまでいじったら，Javascript シェルを閉じてからどこかに別のページにいくのがよいでしょう．

3.5 その他のデバグツール

私が有用だと思ったけれども，それについて解説する時間がないツールを以下に挙げます．

- Web Developer extension にはページを再構築する関数がたくさんあります．
- Aardvark はタグの名前，`id` や `class` 要素を対話式に表示します．
- Venkman Javascript Debugger は完全なランタイム Javascript デバッガです[2]．
- Web Development Bookmarklets にはツールバーにドラッグできる役立つ関数がいくつかあります．
- JSUnit は Javascript のためのユニットテストフレームワークです．

[2] なんのことか分からないので訳が変かも

- js-unit は Javascript コードのよくあるエラーをチェックします.

第4章　よくあるパターン

4.1　ユーザスクリプトをあるドメインとその全てのサブドメインで実行する

多くのサイトは www. を入れても入れなくてもきちんと表示されます．そのようなサイトで動くユーザスクリプトを書きたい場合には，両方のアドレスを指定する必要があります．

リスト 9: あるドメインとその全てのサブドメインにマッチするメタデータタグ

```
// ==UserScript==
// @include http://example.com/*
// @include http://*.example.com/*
// ==/UserScript==
```

使用例

- Butler
- Salon Auto-Pass

4.2　ある Greasemonkey 関数が使えるかどうかテストする

Greasemonkey のバージョンが新しくなると，ユーザスクリプトに使える新たな関数が出てきます．ユーザスクリプトを配布するつもりならば，使われている Greasemonkey 関数が実際に存在するかチェックするべきです．

リスト 10: Greasemonkey 関数が使えない場合にユーザに警告を出す

```
if (!GM_xmlhttpRequest) {
    alert('Please upgrade to the latest version of Greasemonkey.');
    return;
}
// more code here that uses GM_xmlhttpRequest
```

4.3 あるページが特定の HTML 要素を含むかテストする

getElementsByTagName 関数を使ってページにある HTML 要素が存在するかをテストすることが出来ます．

リスト 11: ページが <textarea> を含むかチェックする

```
var textareas = document.getElementsByTagName('textarea');
if (textareas.length) {
    // there is at least one textarea on this page
} else {
    // there are no textareas on this page
}
```

使用例

- BetterDir
- Zoom Textarea

4.4 HTML 要素全てになにか操作を行う

あるページの全ての HTML 要素に何かしたい場合があります．Firefox は getElementsByTagname('*') をサポートしており，これを使って得られたリストをループできます．

リスト 12: 全ての要素をループする

```
var allElements, thisElement;
allElements = document.getElementsByTagName('*');
for (var i = 0; i < allElements.length; i++) {
    thisElement = allElements[i];
    // do something with thisElement
}
```

注意

これを使うのは**全ての**要素に対して何かを行わなければならないときだけにしましょう．特定の要素にだけ何かしたいということが事前に分かっているならば，欲しい要素を取得する XPath クエリを使った方が高速です．更なる情報はある属性をもった要素全てになにか操作を行うを見てください．

使用例

- Anti-Disabler

4.5　特定の HTML 要素全てになにか操作を行う

ページの特定の HTML 要素全てに対して何かしたいときがあります．たとえば，全ての <textarea> でフォントを変更したい，といった場合です．これを行う最も簡単な方法は getElementsByTagName('tagname') をよびだして，その要素集めてループすることです．

リスト 13: ページの全ての textarea を見つける

```
var allTextareas, thisTextarea;
allTextareas = document.getElementsByTagName('textarea');
for (var i = 0; i < allTextareas.length; i++) {
    thisTextarea = allTextareas[i];
    // do something with thisTextarea
}
```

注意

全てのリンクに何かしたいという場合にはこれを使わない方がよいです．なぜなら，<a> はページの中のアンカーにも使われるからです．ページ中の全てのリンクを見つけるやり方はある属性をもった要素全てになにか操作を行うを見てください．

使用例

- Zoom Textarea

4.6　ある属性をもった要素全てになにか操作を行う

Greasemonkey の提供する中で最も強力なのは evaluate メソッドで，これは XPath と呼ばれる問い合わせ言語を使ってページの中の要素，属性，テキストを見つけます．

例えば, ページの中のリンクを全て見つけたいとしましょう．getElementsByTagName('a') を使えばよいと思うかもしれませんが，<a> はページ内のアンカーにも使われるので，href 属性をもっているかどうかを確かめなければならなくなります．

代わりに，Firefox に組み込まれた XPath のサポートを使って href 属性をもつ <a> 要素を全て見つけることにします．

```
var allLinks, thisLink;
allLinks = document.evaluate(
    '//a[@href]',
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
for (var i = 0; i < allLinks.snapshotLength; i++) {
```

```
    thisLink = allLinks.snapshotItem(i);
    // do something with thisLink
}
```

ここでの鍵は document.evaluate です．これは XPath のクエリを文字列として受け取り，さらに以下で説明するいくつかのパラメータをとります．この XPath クエリは href 属性をもつ <a> 要素を全てみつけて，それをランダムな順序で返します．（つまり，最初に得られるのがページの中の最初のリンクであるとは限らないということです．）見つかった要素は allLinks.snapshotItem(i) メソッドでアクセスできます．

XPath 式を使うといろんなことが出来ます．下の式は title 属性をもつ全ての要素を見つけます．

リスト 14: title 属性をもつ全ての要素を見つける

```
var allElements, thisElement;
allElements = document.evaluate(
    '//*[@title]',
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
for (var i = 0; i < allElements.snapshotLength; i++) {
    thisElement = allElements.snapshotItem(i);
    switch (thisElement.nodeName.toUpperCase()) {
        case 'A':
            // this is a link, do something
            break;
        case 'IMG':
            // this is an image, do something else
            break;
        default:
            // do something with other kinds of HTML elements
    }
}
```

ヒント

要素への参照（たとえば thisElement）を取得したら，thisElement.nodeName によって HTML タグ名を決定できます．ページが text/html として提供されてい場合，タグ名は元のページでどのように指定されているかに関わらず常に大文字で返されます．しかし，application/xhtml+xml で提供されていると，タグ名は常に小文字です．私はいつも thisElement.nodeName.toUpperCase() をつかってそれを忘れることにしています．

次に示すのは特定の class をもった 全てを返す XPath クエリです．

リスト 15: sponseredlink という class の div タグを全て見つける

```
var allDivs, thisDiv;
allDivs = document.evaluate(
    "//div[@class='sponsoredlink']",
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
for (var i = 0; i < allDivs.snapshotLength; i++) {
    thisDiv = allDivs.snapshotItem(i);
    // do something with thisDiv
}
```

XPath クエリ文字列を囲むのにダブルクオートを使って中でシングルクオートが使えるようにしたことに注意してください．

`document.evaluate` メソッドには様々なバリエーションがあります．二つ目のパラメータ（上の二つではどちらも `document`）には任意の要素を指定でき，その場合 XPath クエリはその要素より下にあるノードだけを返します．よって，ある要素への参照をもっていてる場合（たとえば `document.getElementById` や `document.getElementsByTagName` 配列の要素によって），クエリをその要素より下の要素だけに限定することが出来ます．

3 つ目のパラメータは名前空間解決関数への参照で，`application/xhtml+xml` メディアタイプとして提供されているページで動くユーザスクリプトを書きたい場合のみに使います．何のことか分からなければ気にしないで下さい．そのようなページはそんなにないし，出くわさないかもしれません．もし知りたければ，Mozilla XPath Documentation に説明があります．

4 つ目のパラメータはどういう風に結果を返して欲しいかを指定します．上の 2 つの例ではどちらも `XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE` をつかっていました．これは結果をランダムな順序で返します．私はほとんどの場合これを使いますが，何らかの理由でページに現れる順序で要素が返ってくるようにしたい場合，`XPathResult.ORDERED_NODE_SNAPSHOT_TYPE` を代わりに使えます．Mozilla XPath Documentation にはほかのいくつかのバリエーションの説明があります．

5 つ目のパラメータは 2 つの XPath クエリの結果をマージするために使われます．過去の `document.evaluate` の呼び出しの結果にわたって，2 つのクエリの結果を結合したものを返します．上の例ではともに `null` で，これは最初のパラメータで定義された一つの XPath クエリしか使わないということです．

全部理解できましたか？ XPath は好きなだけ単純にも複雑にも出来ます．XPath の構文についてより詳しく知りたい場合はこのすばらしい XPath のチュートリアルを読むことをお勧めします．`document.evaluate` のほかのパラメータについて，ここで説明した以上に使うことは滅多にありません．そこで，それらをまとめた関数を定義することが出来ます．

リスト 16: xpath 関数

```
function xpath(query) {
    return document.evaluate(query, document, null,
        XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE, null);
}
```

これで，ページ上の全てのリンクを取得するには，単に `xpath('//a[@href]')` を呼べばよいことになります．`title` 属性をもつ要素を得るには `xpath('//*[@title]')` です．要素にアクセスするには相変わらず `snapshotItem` メソッドを使う必要があります．それは通常の Javascript 配列ではないのです．

使用例

- Access Bar
- BetterDir
- Blogdex Display Title
- Butler
- Frownies
- Offsite Blank
- Rotten Reviews
- Stop The Presses

参考サイト

- Mozilla XPath documentation
- XPath tutorial by example
- XPathResult reference

4.7 要素の前にコンテンツを挿入する

要素を（どんな方法を使ったたにせよ）取得したら，その要素の前にコンテンツを追加したいと思うかもしれません．これは `insertBefore` を使えば出来ます．

以下では ID が `main` である要素があると仮定しています．

リスト 17: <hr> をメインコンテンツの前に挿入する

```
var main, newElement;
main = document.getElementById('main');
if (main) {
    newElement = document.createElement('hr');
    main.parentNode.insertBefore(newElement, main);
}
```

使用例

- Butler
- Zoom Textarea

4.8 要素の後にコンテンツを挿入する

要素を見つけたあと，その**後に**コンテンツを挿入したい場合もあります．nextSibling 属性を使えば，これも insertBefore を使って出来ます．

以下では navbar という ID の要素があると仮定しています．

リスト 18: ナビゲーションバーの後に <hr> を挿入する

```
var navbar, newElement;
navbar = document.getElementById('navbar');
if (navbar) {
    newElement = document.createElement('hr');
    navbar.parentNode.insertBefore(newElement, navbar.nextSibling);
}
```

ヒント

someExistingElement がその親ノードの最後の子（つまり次のシブリングがない[1]）の場合でも someExistingElement.nextSibling によってコンテンツを挿入できます．この場合，someExistingElement.nextSibling は null を返しますが，insertBefore 関数は単にほかの全てのシブリングのあとに新しいコンテンツを追加します．（何を言ってるかわからなければ，気にしないでください．要するに，上の例は，うまく動かないように思われる場面でも常に動作するということです）

使用例

- Blogdex Display Title
- Butler

[1] 訳注：sibling というのは兄弟，姉妹のことです．

4.9 要素の削除

Greasemonkeyをつかってページの一部分をさっと削除することも出来ます．removeChildを使います．

下の例ではadsというIDを持った要素があると仮定しています．

リスト 19: 広告サイドバーを消す

```
var adSidebar = document.getElementById('ads');
if (adSidebar) {
    adSidebar.parentNode.removeChild(adSidebar);
}
```

注意

removeChildで要素を削除すると，その下にあるコンテンツも全て削除します．たとえば <table> 要素を削除したら，その表に含まれるすべてのセル（<td> 要素）も削除されます．

ヒント

もし広告を消したいだけならば，AdBlockをインストールしてup-to-date filter listをインポートした方が自分でユーザスクリプトを書くより簡単でしょう．

使用例

- Butler

4.10 要素を新しいコンテンツで置き換える

要素を取得したら，replaceChildでそれを全く新しいコンテンツで置き換えることが出来ます．

以下ではannoyingsmilyというIDの要素があることを仮定しています．

リスト 20: 画像を代替テキストで置き換える

```
var theImage, altText;
theImage = document.getElementById('annoyingsmily');
if (theImage) {
    altText = document.createTextNode(theImage.alt);
    theImage.parentNode.replaceChild(altText, theImage);
}
```

注意

大きなHTMLで要素を置き換えたい場合は，HTMLを文字列として作ってそれをinnerHTML属性に設定することができます．

使用例

- Frownies

4.11 複雑な HTML を素早く挿入する

innerHTML 属性を使うと HTML を文字列として作ってからそれを直接ページに挿入することが出来ます．こうすると，各 HMTL 要素に別々の DOM オブジェクトを作って一つ一つ属性を設定していかなくて済みます．

リスト 21: ページのトップにバナーを挿入する

```
var logo = document.createElement("div");
logo.innerHTML = ' ' +
    'YOUR TEXT HERE ' +
    '';
document.body.insertBefore(logo, document.body.firstChild);
```

ここでのポイントは 2 行めで，logo.innerHTML を文字列にセットしています．Firefox は，最初にロードしたときにページ全体に行うのと同じように，文字列を HTML として解析し必要なオブジェクトをつくります．その後新しい logo（<p> を含む を含んだ 要素）をページのどこへでも挿入することが出来ます．ページの最初，終わり，あるいは選択した要素の前や後などです．言い換えれば，ページの任意箇所に挿入できるのです．

使用例

- Access Bar
- Butler
- BetterDir

4.12 サーバに接続にせずに画像を加える

Firefox は data:URL をサポートしています．これをつかうと，サーバを別に呼び出してデータを取得することなく URL にデータを埋め込むことが出来ます．data:URL については聞いたことがないかもしれません．なぜなら，Internet Explorer がこれをサポートしていないため誰も使わないからです．しかし，ユーザスクリプトなら便利に使えます．

リスト 22: ページのトップにグラフィカルなロゴを加える

```
var logo = document.createElement('img');
logo.src = \
    ''+
'LAAAAAANAA4AQAIjjI8Iyw3GhACSQecutsFV3nzgNi7SVEbo06lZa66LRib2UQAAOw%3D%3D';
document.body.insertBefore(logo, document.body.firstChild);
```

この例では，<img> 要素の src がエンコードされた画像データをもつ data:URL になっています．新しい要素がページに挿入されれば，ほかの画像と全く同じように表示されますが，画像がサーバに保存されている必要がないという点が異なります．つまり，ユーザースクリプトに画像を埋め込んでその他のコードと一緒にファイルの一部分として画像を配布できるということです．

ヒント

自分の data:URL を作るには data:URL kitchen が使えます．

使用例

- Butler
- Zoom Textarea

参考

- data: **URI** kitchen

4.13 CSS を追加する

ページに自分の CSS を追加したいことがよくあります．既存のスタイルを上書きするスタイルも，ユーザースクリプトが挿入する新しい要素のためのスタイルも追加することができます．

リスト 23: パラグラフのテキストを大きくする

```
function addGlobalStyle(css) {
    var head, style;
    head = document.getElementsByTagName('head')[0];
    if (!head) { return; }
    style = document.createElement('style');
    style.type = 'text/css';
    style.innerHTML = css;
    head.appendChild(style);
}

addGlobalStyle('p { font-size: large ! important; }');
```

この関数はページに追加したいスタイルを含む文字列を引数としてとります．これは単純にあなたのスタイルを含む <style> 要素をページの <head> に埋め込みます．Firefox は変更を察知し，スタイルルールを読んで，それをページに適用します．一つの関数呼び出しに好きなだけスタイルルールを含めることが出来ます．ただ文字列としてつなげていっぺんに関数に渡せばよいのです．

ヒント

ページに挿入したいスタイルや元のページに含まれる要素に対して addGlobalStyle 関数を使うこともできます．しかし，既存のスタイルルールを変更する場合には！ important を定義するルールごとに記述してもとのページで定義されているルールを上書きするようにしなければなりません．

使用例

- Access Bar
- Aint It Readable
- BetterDir
- Butler
- CDReadable
- LIP

参考

- 事例研究: Ain't It Readable

4.14 要素のスタイルを取得する

全ての CSS が適用された後に特定の要素が取得できると便利な場合があります．要素の sytle 属性を取り出せばよいと思うかもしれませんが，これは間違いです．これが返すのは要素の，タグの中で style=""により直接指定されているスタイルだけです．（外部のスタイルシートによって定義されているものも含めた）最終的なスタイルを取得するには getComputedStyle 関数を使う必要があります．

説明のために，簡単なページを作りましょう．このなかでは，<p> のスタイルが定義されていますが，<p> の一つはタグの中の style 属性により上書きされています．

```
<html>
<head>
<title>Style test page</title>
<style type="text/css">
p { background-color: white; color: red; }
</style>
</head>
<body>
<p id="p1">This line is red.</p>
<p id="p2" style="color: blue">This line is blue.</p>
</body>
</html>
```

リスト 24: ある要素のタグの中で定義されたスタイルを取得する

```
var p1elem, p2elem;
p1elem = document.getElementById('p1');
p2elem = document.getElementById('p2');
alert(p1elem.style.color); // will display an empty string
alert(p2elem.style.color); // will display "blue"
```

これではあまり役に立たないので，要素のスタイルを取得するのに `element.style` は使わない方がよいでしょう．(これはここのスタイルを**セット**するときには使えます;要素のスタイルを設定するを見てください)

何を使えばよいのでしょう？ `getComputeStyle()` です．

リスト 25: 要素の実際のスタイルを取得する

```
var p1elem, p2elem, p1style, p2style;
p1elem = document.getElementById('p1');
p2elem = document.getElementById('p2');
p1style = getComputedStyle(p1elem, '');
p2style = getComputedStyle(p2elem, '');
alert(p1style.color); // will display "rgb(255, 0, 0)"
alert(p2style.color); // will display "rgb(0, 0, 255)"
```

使用例

- Butler
- Zoom Textarea

4.15 要素のスタイルを設定する

一つの要素のスタイルをいくつか設定するだけなら，`style` 属性のいろいろな値を「手動で」セットすればできます．

これは `logo` という ID を持つ要素があると仮定しています．

リスト 26: 一つの要素のスタイルを設定する

```
var logo = document.getElementById('logo');
logo.style.marginTop = '2em';
logo.style.backgroundColor = 'white';
logo.style.color = 'red';
```

警告

ここのスタイルの属性の名前は自明なものとは限りません．一般的には同じパターンに従います．たとえば，`margin-top` は `someElement.style.marginTop` になります．しかし例外もあります：`float` 属性は `someElement.style.cssFloat` によってセットされます．これは"float"が Javascript の予約語だからです．

使用例

- Access Bar
- BetterDir
- Blogdex Display Title
- Zoom Textarea

参考サイト

- CSS properties

4.16 レンダリング後の事後処理

Firefox のページをレンダリングした後のページの保持の仕組みによる理由で，ページの DOM に対する大きな変更はページの読み込みが終わった**後**に行った方がよいです．`addEventListner` を使うことで関数の実行を遅らせることが出来ます．

リスト 27: ページ全体をカスタムコンテンツで置き換える

```
var newBody =
'';
window.addEventListener(
    'load',
    function() { document.body.innerHTML = newBody; },
    true);
```

よく見ると，このコードがなぜ動くのか不思議に思うかもしれません．`window.addEventListner` の第二引数として無名関数を宣言し，それは外の関数で定義されている変数 `newBody` を参照しています．これは「閉包 (closure)」とよばれ，Javascript では全く問題のないものです．一般に，「外部の」関数の中で定義された「内部の」関数はその外部関数の局所変数全てを参照できます．外部関数が実行を終えた後でも，です．これはとても強力な機能で，実行時に構成されるデータ構造を含むエベントハンドラやその他の関数を作ることを可能にします．

ヒント

`document.body.innerHTML` を削除したり置き換えたりしても，ページのすべてが変わるという訳ではありません．タイトルやCSSスタイル，スクリプトなど，元のページの `<head>` のなかで定義されているものはまだ有効です．これらを変更したり削除したりするのは個別にやらなくてはなりません．

使用例

- Access Bar
- BetterDir

4.17 大文字と小文字を区別せずに属性の値をマッチさせる

HTML の属性値の多くは大文字と小文字を区別しません．また値の前や後にスペースが含まれていても大丈夫です．全てのバリエーションを知るにはXPathクエリをいろいろいじってみるとよいでしょう．

リスト 28: メソッドが POST あるいは post であるフォームを全て見つける

```
var postforms = document.evaluate(
    "//form[translate(@method, 'POST ', 'post')='post']",
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
```

このXPathクエリはPOSTメソッドをもったフォームを見つけます．まず，`translate` 関数を使って `method` 属性を小文字にする必要があります．（XPath 2.0 は `lowercase` 関数をもっていますが，私はうまく使えたことがないです．）つぎに，前と後ろのスペースを取り除きます．最初の引数に余分のスペースを含めれば，`translate` 関数の呼び出しにこのスペースの除去を含めることが出来ます．二つめの引数に対応する文字がないので，全てのスペースは取り除かれます．最後に，得られた文字列を `post` と比較します．

使用例

- ForceGet

4.18 現在のドメイン名を取得する

複数の（あるいは全ての）ドメインで動くユーザースクリプトでは現在のページのドメインを取得することが必要になる場合がよくあります．そのためには，`window.location.href`

を使って現在のページの完全な URL を取得したり，window.location.host を使ってドメインめのみを取得したりできます．

リスト 29: 現在のドメインを取得する

```
var href = window.location.host;
```

使用例

- Offsite Blank

4.19 リンクを書き換える

ページにリンクを見つけたら，href 属性を設定してそれを書き換えることが出来ます．次のコードは article という ID をもったリンクがあると仮定しています．

リスト 30: リンクの最後にクエリのパラメータを加える

```
var a = document.getElementById('article');
if (a.href.match(/\?/i)) {
    // link already contains other parameters, so append "&printer=1"
    a.href += '&printer=1';
} else {
    // link does not contain any parameters, so append "?printer=1"
    a.href += '?printer=1';
}
```

使用例

- Rotten Reviews
- Stop The Presses

4.20 ページをリダイレクトする

Greasemonkey を使って自動的に特定のページにリダイレクトすることが出来ます．window.location.href 属性を設定します．

リスト 31: あるサイトを同じサイトの安全な方にリダイレクトする

```
window.location.href = window.location.href.replace(/^http:/, 'https:');
```

使用例

- GMail Secure
- Salon Auto-Pass

4.21 ユーザのクリックに干渉する

リンクを書き換えるのは簡単ですが，もう一歩進んで，addEventLister 関数を使ってページ上のあらゆる場所でのクリックを察知することが出来ます．（リンクへのクリックも察知します．）そうすると，クリックが何をするか指示できます．何もしないでクリックをそのままクリックされた要素に渡すことも出来るし，全く違うことをすることも出来ます．

リスト 32: ユーザがページ上のどこかをクリックしたときに何かする

```
document.addEventListener('click', function(event) {
    // event.target is the element that was clicked

    // do whatever you want here

    // if you want to prevent the default click action
    // (such as following a link), use these two commands:
    event.stopPropagation();
    event.preventDefault();
}, true);
```

document.addEventListener 関数の引数として無名関数を使っていることに注意してください．

4.22 Javascript の組み込みメソッドを上書きする

prototype 属性を使ってオブジェクトのネイティブなメソッドを上書きすることが出来ます．

リスト 33: フォームが送信される前に何かする

```
function newsubmit(event) {
    var target = event ? event.target : this;

    // do anything you like here
    alert('Submitting form to ' + target.action);

    // call real submit function
    this._submit();
}

// capture the onsubmit event on all forms
window.addEventListener('submit', newsubmit, true);

// If a script calls someForm.submit(), the onsubmit event does not fire,
```

```
// so we need to redefine the submit method of the HTMLFormElement class.
HTMLFormElement.prototype._submit = HTMLFormElement.prototype.submit;
HTMLFormElement.prototype.submit = newsubmit;
```

ここではやっているのは 2 つです．まず，submit イベントを察知するイベント・リスナーを付け加えています．submit イベントはユーザがフォームの送信 (submit) ボタンを押したときに発生します．もし別のスクリプトがあるフォームの submit() メソッドを直に呼び出した場合，submit イベントは発生しません．そこで，HTMLFormElement クラスの submit メソッドを上書きしています．これが 2 つめです．

まだあります．イベント・リスナーとメソッドの上書きがともに同じ関数 newsubmit を参照しています．submit イベントによって呼ばれたときには，event 引数にそのイベントに関する情報をもったイベントオブジェクトが渡されます（たとえば event.target が送信されるフォームになります）．しかし，スクリプトが直に submit メソッドを呼び出した場合，event 引数が欠けて，そのため大域変数 this がそのフォームを参照します．よって上の newsubmit 関数では，event が null かどうかテストして，そうだった場合は this からターゲットフォームを取得しています[2]．

ヒント

ユーザーがフォームを通常通り送信した場合，つまり送信ボタンをクリックするかフォームの中で Enter を打つかして送信した場合には submit イベントが発生します．しかし，フォームが aForm.submit() により送信された場合は submit イベントは発生しません．そのため，フォームの送信をサチするには 2 つのことをしなければなりません．ひとつは submit イベントを察知するイベント・リスナーを加えること，もう一つは HTMLFormElement クラスのプロトタイプを変更して submit メソッドを自分でカスタマイズした関数にリダイレクトすることです．

参考サイト

- Javascript event compatibility tables
- Javascript-DOM prototypes in Mozilla
- Displaying Event object constants

4.23 XML を解析する

Firefox は自動的に現在のページを DOM に解析しますが，自分で作ったあるいはほかの場所から取得した XML 文字列から直接 DOM を作ることも出来ます．

リスト 34: 任意の文字列を XML として解析する

[2] いまいち何のことかわからず訳が怪しい

```
var xmlString = ''
var parser = new DOMParser();
var xmlDoc = parser.parseFromString(xmlString, "application/xml");
```

ここで鍵となるのは DOMParser オブジェクトで，parseFromString メソッドが使えます．（ほかにもメソッドがありますがここではあまり役に立ちません．）parseFromString メソッドは 2 つの引数をとります．解析する XML 文字列とコンテンツタイプで，両方とも指定しなければなりません．

注意

DOMParser の parseFromString メソッドはコンテンツタイプを 2 つめの引数としてとります．可能なのは application/xml, application/xhtml+xml, text+html です．ばからしいのでここでは理由を述べませんが，常に application/xml を使った方がよいです．

GM_xmlhttpRequest と組み合わせて別の場所から取得した XML を解析するときに，このパターンは特に有効です．

リスト 35: 別の場所から取得した XML を解析する

```
M_xmlhttpRequest({
    method: 'GET',
    url: 'http://greaseblog.blogspot.com/atom.xml',
    headers: {
        'User-agent': 'Mozilla/4.0 (compatible) Greasemonkey/0.3',
        'Accept': 'application/atom+xml,application/xml,text/xml',
    },
    onload: function(responseDetails) {
        var parser = new DOMParser();
        var dom = parser.parseFromString(responseDetails.responseText,
            "application/xml");
        var entries = dom.getElementsByTagName('entry');
        var title;
        for (var i = 0; i < entries.length; i++) {
title = entries[i].getElementsByTagName('title')[0].textContent;
            alert(title);
        }
    }
});
```

このコードは http://greaseblog.blogspot.com/atom.xml から Atom フィードを読み込んで，それを DOM に解析し，DOM に問い合わせをして，エントリのリストを得ます．各エントリについて，DOM にもう一度問い合わせをしてエントリのタイトルを取得し，それをダイアログボックスに表示します．

参考

- iterate-one-element.html

- GM_xmlhttpRequest

第5章　事例研究

5.1　事例研究: Gmail Secure

GMail Secure は `http://gmail.google.com/` から `https://gmail.google.com/` にリダイレクトして GMail に暗号化された接続を使用するようにします．

Google は GMail というウェブメールサービスを暗号化されていない接続 (http://アドレス) あるいは暗号化された接続 (https://アドレス) で提供しています．公共のネットワーク（例えばインターネットカフェ）でメールのチェックをするときには常に暗号化された接続を使うようにしていますが，コンピュータが代わりにそうしてくれるのならその方がよいでしょう．私は暗号化されていない接続で GMail を使おうとしたときに暗号化されたサイトにリダイレクトしてくれるユーザースクリプトを書きました．

リスト 36: GMail を同等な https://サイトにリダイレクトする

```
// ==UserScript==
// @name           GMailSecure
// @namespace      http://diveintogreasemonkey.org/download/
// @description    force GMail to use secure connection
// @include        http://gmail.google.com/*
// ==/UserScript==

window.location.href = window.location.href.replace(/^http:/, 'https:');
```

このユーザースクリプトはとても単純です．`@include` という行が「仕事」の大部分を行います．

```
// @include        http://gmail.google.com/*
```

このユーザースクリプトは`@include` がマッチしたときにだけ実行されます．よって，このスクリプトが走っているということは，私が（暗号化されていない接続で GMail を使用しているという意味で）間違った場所にいるということです．そうすると，私の目的を達成するのは，現在のページを http://ではなく https://で始まる同じ URL にリダイレクトする一行のコードです．

```
window.location.href = window.location.href.replace(/^http:/, 'https:');
```

参考

- ページをリダイレクトする

5.2 事例研究: Blogline Autoload

Bloglines は配布されるフィードを集めるウェブベースのアグリゲータです．インターフェイスは 2 ペインで，左にはあなたの登録しているフィード，右側にはそれらの内容が表示されます．とてもよいインターフェイスです．一つだけ気に入らないのは，毎度同じことをしなければならないということです：Bloglines を開くたびに自分がまだ見ていないもの全て，つまり未読の記事全てを見たいのです．

Bloglines では未読記事を全て表示するには一回クリックするだけです．左側の自分の登録しているフィードのルートレベルをクリックすると，右側に未読の記事が現れます．しかし，常にこうしたいので，その一回のクリックを自動化するユーザースクリプトを書きました．

リスト 37: Bloglines に未読記事全てを自動的に表示させる

```
// ==UserScript==
// @name           Bloglines Autoloader
// @namespace      http://diveintogreasemonkey.org/download/
// @description    Auto-display all new items in Bloglines
// @include        http://bloglines.com/myblogs*
// @include        http://www.bloglines.com/myblogs*
// ==/UserScript==

if (doLoadAll) {
    doLoadAll();
}
```

このユーザースクリプトはきわめて単純です．Bloglines は `doLoadAll()` 関数を定義していて，これがフィードのリストのルートをてでクリックしたときに実行されます．この関数を呼ぶと全ての未読記事が表示されます．

しかし，Bloglines はフレームを使っているので，ユーザースクリプトは各フレームで実行にされることになります（どのフレームも `@include` に定義したパターンにマッチするからです）．そこで，まず `doLoadAll()` 関数がフレームに存在しているかをチェックします．

```
if (doLoadAll) {
```

関数が存在したら，あとはそれを呼ぶだけです．ユーザースクリプトはページと同じコンテクストの中で実行されるので，元のページが定義しているどんなスクリプトも呼ぶことが出来ます．

```
doLoadAll();
}
```

ダウンロード

- bloglines-autoload.user.js

5.3 事例研究: Ain't It Readable

Ain't It Cool News はエンターテイメントニュースに特化したサイトです．このサイトはすばらしく，おすすめです．残念なことに，私はこのサイトの見かけが気に入らないのです．各ヘッドラインは大きすぎるし，ボールドフォントで，さらにカーソルをその上にもっていくと色が変わります．そこで気に入らないスタイルを変更するユーザースクリプトを書きました．

リスト 38: aintitreadable.user.js

```
// ==UserScript==
// @name           Ain't It Readable
// @namespace      http://diveintogreasemonkey.org/download/
// @description    change style on aint-it-cool-news.com
// @include        http://aint-it-cool-news.com/*
// @include        http://*.aint-it-cool-news.com/*
// ==/UserScript==

function addGlobalStyle(css) {
    var head, style;
    head = document.getElementsByTagName('head')[0];
    if (!head) { return; }
    style = document.createElement('style');
    style.type = 'text/css';
    style.innerHTML = css;
    head.appendChild(style);
}

addGlobalStyle(
'h1, h2, h3, h4 {' +
'  font-size: 12px ! important;' +
```

```
’  line-height: 14px ! important;’ +
’  font-weight: normal ! important;’ +
’}’ +
’h1:hover, h2:hover, h3:hover, h4:hover {’ +
’  background-color: inherit ! important;’ +
’  color: inherit ! important;’ +
’}’);
```

このユーザースクリプトはとてもシンプルで，まずページに任意の CSS スタイルを追加する関数を定義しています．より詳しくは CSS を追加するを見てください．

```
function addGlobalStyle(css) {
    var head, style;
    head = document.getElementsByTagName(’head’)[0];
    if (!head) { return; }
    style = document.createElement(’style’);
    style.type = ’text/css’;
    style.innerHTML = css;
    head.appendChild(style);
}
```

あとは追加したい CSS スタイルを引数としてこの関数を呼ぶだけです．ヘッドラインをもっと小さく，ボールドでないようにし，またカーソルで指したときの色の変化を取り除きます．この場合，これらそれぞれについてページがスタイルを定義しているので，! important を使ってページのもとのスタイルではなく自分で定義したスタイルが適用されるようにしています．

この関数が引数としてとるのは追加したいスタイルを含む文字列一つだけだということに注意してください．上では読みやすいように整形していますが，それでも一つの文字列です．

```
addGlobalStyle(
’h1, h2, h3, h4 {’ +
’  font-size: 12px ! important;’ +
’  line-height: 14px ! important;’ +
’  font-weight: normal ! important;’ +
’}’ +
’h1:hover, h2:hover, h3:hover, h4:hover {’ +
’  background-color: inherit ! important;’ +
’  color: inherit ! important;’ +
’}’);
```

ダウンロード

- aintitreadable.user.js

参考

- CSS を追加する

5.4 事例研究: Offsite Blank

Offsite Blank は誰かが Gresemonkey スクリプトレポジトリにリクエストしたのを受けて私が書いたユーザースクリプトです．個人的には現在のウィンドウの新規タブでリンクを開くのが好みですが，一つのサイトについてそれぞれ別のウィンドを開きたいという人もいます．Offsite Blank はこれを自動で行います．つまりサイトの外のリンクは別ウィンドウで開くようにします．

リスト 39: offsiteblank.user.js

```
// ==UserScript==
// @name           Offsite Blank
// @namespace      http://diveintogreasemonkey.org/download/
// @description    force offsite links to open in a new window
// @include        http://*
// @include        https://*
// ==/UserScript==

var a, thisdomain, links;
thisdomain = window.location.host;
links = document.getElementsByTagName('a');
for (var i = 0; i < links.length; i++) {
    a = links[i];
    if (a.host && a.host != thisdomain) {
        a.target = "_blank";
    }
}
```

まずユーザースクリプトが全てのページで動くように宣言します（ただし，ローカルに保存してある HTML ファイルのように，，**ファイル→ 開く**をつかって開くようなものは除きます)．

```
// @include        http://*
// @include        https://*
```

コードは 4 段階に分割できます．

1. 現在のページのドメインを取得する．
2. ページ上の全てのリンクを取得する．
3. 各リンクのドメインを現在のページのドメインと比較する．
4. ドメインが一致しなければ，別ウィンドウで開くようにターゲットを設定する．

現在のページのドメインを取得するのは簡単です．詳しくは現在のドメイン名を取得するを見てください．

```
thisdomain = window.location.host;
```

ページ上の全てのリンクを取得するのも同様に簡単です．ただし，ここでは自分自身の忠告に従わず，XPath を使わずに単純に document.getElementsByTagName('a') を使っています．いろんなやり方があります．

```
links = document.getElementsByTagName('a');
```

次に全てのリンクを全てループして（正確に言えば全ての <a> 要素で，それらのうちいくつかがリンクです）リンクのドメインが現在のページのドメインにマッチするかどうかチェックします．リンクのうちいくつかは HTTP でない URL をさしているかもしれないので（たとえばローカルファイルや FTP サーバをさしてるかもしれません），a.host が存在するか確かめてる必要があり，それから現在のドメインと等しいか確かめます．

```
for (var i = 0; i < links.length; i++) {
    a = links[i];
    if (a.host && a.host != thisdomain) {
        ...
    }
```

現在のドメインと一致しないドメインをもったリンクを見つけたら，その target を "_blank"にセットして別のウィンドウでリンクを開くようにします．

```
    a.target = "_blank";
```

ダウンロード

- offsiteblank.user.js

参考

- 現在のドメイン名を取得する
- 特定の HTML 要素全てになにか操作を行う
- ある属性を持った要素全てになにか操作を行う

5.5 事例研究: Dumb Quotes

DumbQuotes はたくさんのブロガーの不満から生まれました．多くの出版ソフト (publishing software) はまっすぐな ASCII クオートを「賢いクオート」[1]に自動的に変換します．しかし，テキストをコピーしてペーストすると同じソフトが「賢いクオート」をうまく扱ってくれないのです．よくあるのは，ブロガーが別のサイトから文を引用したいときです．ウェブブラウザから文を選択し，自分のサイトにポストするためにそれをウェブフォームにペーストします．すると，ペーストした文章は元のサイトとは全く違うものになってしまうのです．これは出版ソフトが文字コードをきちんと判別しないからです．

世界中の全ての出版ソフトを修正することは出来ませんが，自分にとっての問題のある部分を修正するために，「賢いクオート」やその他のマルチバイトの文字を同等な 7 ビットの ASCII 文字に自動的に置き換えるユーザースクリプトを書きました．

リスト 40: dumbquotes.user.js

```
// ==UserScript==
// @name          DumbQuotes
// @namespace     http://diveintogreasemonkey.org/download/
// @description straighten curly quotes and apostrophes, simplify fancy \
    dashes, etc.
// @include       *
// ==/UserScript==

var replacements, regex, key, textnodes, node, s;

replacements = {
    "\xa0": " ",
    "\xa9": "(c)",
    "\xae": "(r)",
    "\xb7": "*",
    "\u2018": "'",
    "\u2019": "'",
    "\u201c": '"',
    "\u201d": '"',
    "\u2026": "...",
    "\u2002": " ",
    "\u2003": " ",
    "\u2009": " ",
    "\u2013": "-",
    "\u2014": "--",
    "\u2122": "(tm)"};
regex = {};
for (key in replacements) {
    regex[key] = new RegExp(key, 'g');
}
```

[1] “と” のようなクオート

```
textnodes = document.evaluate(
    "//text()",
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
for (var i = 0; i <textnodes.snapshotLength; i++) {
    node = textnodes.snapshotItem(i);
    s = node.data;
    for (key in replacements) {
        s = s.replace(regex[key], replacements[key]);
    }
    node.data = s;
}
```

コードは 4 段階に別れます．

1. 8 ビット文字を同等な 7 ビット文字に対応させる文字列置き換えのリストを定義する．
2. 現在のページのテキストノードを全て取得する．
3. テキストノードのリストをループする．
4. 各テキストノードで，各 8 ビット文字を同等な 7 ビット文字で置き換える．

最初のステップは実際には 2 ステップです．Javascript の文字列置き換えは正規表現に基づきます．よって 8 ビット文字を同等な 7 ビット文字で置き換えるためには，正規表現オブジェクトのセットを作る必要があります．

```
replacements = {
    "\xa0": " ",
    "\xa9": "(c)",
    "\xae": "(r)",
    "\xb7": "*",
    "\u2018": "'",
    "\u2019": "'",
    "\u201c": '"',
    "\u201d": '"',
    "\u2026": "...",
    "\u2002": " ",
    "\u2003": " ",
    "\u2009": " ",
    "\u2013": "-",
    "\u2014": "--",
    "\u2122": "(tm)"};
```

```
regex = {};
for (key in replacements) {
    regex[key] = new RegExp(key, 'g');
}
```

連想配列を簡単に作るために {} 構文を使っています．これは個別にキーと値のペアを割り当てていくのと同じです（タイピングが少なくて済みます）

```
replacements["\xa0"] = " ";
replacements["\xa9"] = "(c)";
replacements["\xae"] = "(r)";
// and so forth
```

ここの 8 ビット文字は 16 進数の値によって表され，"\xa0"あるいは"\u2018"のようなエスケープ構文を使います．文字列の対応させる配列が出来たら，それをループして正規表現オブジェクトのリストを作ります．各正規表現は 8 ビット文字を大域的に探します．(二つめの引数の g が大域的な検索[2]を意味します．こうしないと各正規表現はある 8 ビット文字が一つ見つかった時点で検索をやめ，その 8 ビット文字がいくつか残されたままになります．)

次の段階は現在のドキュメントのテキストノードのリストを取得することです．「document.body.innerHTML をつうかってページ全体を文字列として取得し，その中で置き換えればいいじゃないか」と思うかもしれません．

```
var tmp = document.body.innerHTML;
// do a bunch of search/replace on tmp
document.body.innerHTML = tmp;
```

しかし，これは悪い習慣です．なぜなら innerHTMl 属性はページのソース全体を返すからです．マークアップ，スクリプト，属性など全てが含まれます．今の場合多分問題は起きませんが（HTML タグ自体は 8 ビット文字は含まない），とんでもないことになってデバグするのが困難になる場合もあります．正確に何を検索して置換するのかよく考える必要があります．それが「生のページソース」ならば innerHTML を使えばよいでしょう．しかし，上の場合は検索範囲は「ページ上の全てのテキスト」です．よって正しいやりかたは XPath クエリを使って全てのテキストノードを取得することです．

```
textnodes = document.evaluate(
    "//text()",
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
```

[2]マッチする文字列を全て探すということ．

ここでは XPath 関数 `text()` を使っています．これは全てのテキストノードにマッチします．<a> 要素全て，あるいは `alt` 属性をもった <img> 要素全てのリストをとってくるというような，要素ノードの問い合わせの方がなじみがあるでしょう．しかし，DOM には要素の中の実際のテキストのノードも含まれているのです（その他のノードもあります．コメントや指示の処理などです．）そして，今必要なのはテキストノードなのです．

3 つめは全てのテキストノードをループすることです．これは`//[@href]` のような XPath クエリから返される要素を全てループするのと全く同じです．一つだけ違うのはループの要素が要素ノードではなくてテキストノードだということです．

```
for (var i = 0; i <textnodes.snapshotLength; i++) {
    node = textnodes.snapshotItem(i);
    s = node.data;
    // do replacements
    node.data = s;
```

`node` がループ中での現在のテキストノードで，`s` が `node` の実際のテキストです．`s` を使って置換を行い，それから結果をもとのノードにコピーします．

一つのノードのテキストにたいして，実際に置換を行う必要があります．前もって正規表現のリストと置換文字列のリストを用意してあったので，これは比較的単純です．

```
 for (key in replacements) {
        s = s.replace(regex[key], replacements[key]);
    }
```

ダウンロード

- dumbquotes.user.js

参考

- ある属性を持った要素全てになにか操作を行う

5.6 事例研究: Frownies

Frownies はジョークから生まれたものです．Gresemonkey メーリングリストで，ある人が :-) のような ASCII 文字で書かれた「笑顔」を画像に変換するスクリプトを書いたとアナウンスしました．別の誰かが，その逆，つまり画像の笑顔からテキストへの変換，をやるのにどれくらい時間がかかるだろうと言い出しました．

記録のために触れておくと，私がそれをやるのにかかったのは 20 分ほどでした．自動で顔文字を画像で置き換える出版ソフトを調べて，バリエーションの包括的なリストを作るのに費やした時間がほとんどです．

このスクリプトは，出版ソフトが自動で画像の笑顔を生成するとき，<img> の alt 属性に等価なテキストが入っているという事実に依存しています．よって，このスクリプトが実際にやっているのは，alt の値であるテキストがあらかじめ定義された定数のリストにマッチしたときに画像をテキストで置き換えることです．

リスト 41: fronies.user.js

```
// ==UserScript==
// @name           Frownies
// @namespace      http://diveintogreasemonkey.org/download/
// @description    convert graphical smilies to their text equivalents
// @include        *
// ==/UserScript==

var smilies, images, img, replacement;
smilies = [":)", ":-)" ":-(", ":(", ";-)", ";)", ":-D", ":D", ":-/",
    ":/", ":X", ":-X", ":\">", ":P", ":-P", ":O", ":-O", "X-(",
    "X(", ":->", ":>", "B-)", "B)", ">:)", ":((", ":(((", ":-((",
    ":))", ":-))", ":-|", ":|", "O:-)", "O:)", ":-B", ":B", "=;",
    "I)", "I-)", "|-)", "|)", ":-&", ":&", ":-$", ":$", "[-(", ":O)",
    ":@)", "3:-O", ":(|)", "@};-", "**==", "(~~)", "*-:)", "8-X",
    "8X", "=:)", "<):)", ";;)", ":*", ":-*", ":S", ":-S", "/:)",
    "/:-)", "8-|", "8|", "8-}", "8}", "(:|", "=P~", ":-?", ":?",
    "#-O", "#O", "=D>", "~:>", "%%-", "~O)", ":-L", ":L", "[-O<",
    "[O<", "@-)", "@)", "$-)", "$)", ">-)", ":-\"", ":^O", "B-(",
    "B(", ":)>-", "[-X", "[X", "\\:D/", ">:D<", "(%)", "=((", "#:-S",
    "#:S", "=))", "L-)", "L)", "<:-P", "<:P", ":-SS", ":SS", ":-W",
    ":W", ":-<", ":<", ">:P", ">:-P", ">:/", ";))", ":-@", "^:)^",
    ":-J", "(*)", ":GRIN:", ":-)", ":SMILE:", ":SAD:", ":EEK:",
    ":SHOCK:", ":???:", "8)", "8-)", ":COOL:", ":LOL:", ":MAD:",
    ":RAZZ:", ":OOPS:", ":CRY:", ":EVIL:", ":TWISTED:", ":ROLL:",
    ":WINK:", ":!:", ":?:", ":IDEA:", ":ARROW:", ":NEUTRAL:",
    ":MRGREEN:"];
images = document.evaluate(
    '//img[@alt]',
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
for (var i = 0; i <images.snapshotLength; i++) {
    img = images.snapshotItem(i);
    alt = img.alt.toUpperCase();
    for (var j in smilies) {
        if (alt == smilies[j]) {
            replacement = document.createTextNode(alt);
```

```
            img.parentNode.replaceChild(replacement, img);
        }
    }
}
```

このコードは 4 つの段階からなります.

1. （テキストの）顔文字のリストを定義します.
2. ページ上の `alt` 属性をもつ画像を全て見つけます.
3. 各画像について，`alt` の値のテキストが ASCII 文字の顔文字のリストのどれかにマッチするかチェックします.
4. マッチしたら <img> 要素を ASCII の顔文字のみを含むテキストノードで置き換えます.

まず Javascript の [ ] 構文を使ってリストを定義します.

```
smilies = [":)", ":-)" ":-(", ":(", ";-)", ";)", ":-D", ":D", ":-/",
    ":/", ":X", ":-X", ":\">", ":P", ":-P", ":O", ":-O", "X-(",
    "X(", ":->", ":>", "B-)", "B)", ">:)", ":((", ":(((", ":-((",
    ":))", ":-))", ":-|", ":|", "O:-)", "O:)", ":-B", ":B", "=;",
    "I)", "I-)", "|-)", "|)", ":-&", ":&", ":-$", ":$", "[-(", ":O)",
    ":@)", "3:-O", ":(|)", "@};-", "**==", "(~~)", "*-:)", "8-X",
    "8X", "=:)", "<):)", ";;)", ":*", ":-*", ":S", ":-S", "/:)",
    "/:-)", "8-|", "8|", "8-}", "8}", "(:|", "=P~", ":-?", ":?",
    "#-O", "#O", "=D>", "~:>", "%%-", "~O)", ":-L", ":L", "[-O<",
    "[O<", "@-)", "@)", "$-)", "$)", ">-)", ":-\"", ":^O", "B-(",
    "B(", ":)>-", "[-X", "[X", "\\:D/", ">:D<", "(%)", "=((", "#:-S",
    "#:S", "=))", "L-)", "L)", "<:-P", "<:P", ":-SS", ":SS", ":-W",
    ":W", ":-<", ":<", ">:P", ">:-P", ">:/", ";))", ":-@", "^:)^",
    ":-J", "(*)", ":GRIN:", ":-)", ":SMILE:", ":SAD:", ":EEK:",
    ":SHOCK:", ":???:", "8)", "8-)", ":COOL:", ":LOL:", ":MAD:",
    ":RAZZ:", ":OOPS:", ":CRY:", ":EVIL:", ":TWISTED:", ":ROLL:",
    ":WINK:", ":!:", ":?:", ":IDEA:", ":ARROW:", ":NEUTRAL:",
    ":MRGREEN:"];
```

次に，XPath クエリを使ってページの中の `alt` 属性をもった <img> 要素を探します. XPath クエリについては，特定の属性をもった要素全てに対してなにか操作を行うを見てください.

```
images = document.evaluate(
    '//img[@alt]',
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
```

3 つめはこれらの <img> 要素をループし，alt 属性が定義された顔文字のどれかにマッチするか調べます．いくつかの顔文字はアルファベットを含むので，比較する前に toUpperCase() を使って alt 属性を大文字に変換しています．

```
for (var i = 0; i <images.snapshotLength; i++) {
    img = images.snapshotItem(i);
    alt = img.alt.toUpperCase())
    for (var j in smilies) {
        if (alt == smilies[j]) {
            // ...
        }
    }
}
```

最後に，顔文字テキストをもった新しいテキストノードをつくり，既存の <img> 要素を置き換えます．詳しくは要素を新しいコンテンツで置き換えるを見てください．

```
                replacement = document.createTextNode(alt);
            img.parentNode.replaceChild(replacement, img);
```

ダウンロード

- frownies.user.js

参考

- 特定の属性をもった要素全てに対してなにか操作を行う
- 要素を新しいコンテンツで置き換える

5.7 事例研究: Zoom Textarea

Zoom Textarea は，ウェブフォームを変更して全ての <textarea> 要素（複数行のテキストを入力するのに使われる）の上に付け加えます．ツールバーを使うと，ページのほかの部分のスタイルを変更することなく <textarea> のテキストサイズを大きくしたり

小さくしたりできます．ボタンは完全にキーボドによるアクセスが可能です．マウスでクリックする代わりにタブを使って選択し **Enter** を押すことが出来ます．（ここでこのことに触れているのはアクセシビリティが重要であり，また見かけよりこれが難しかったからです．）

リスト 42: zoomtextarea.user.js

```
// ==UserScript==
// @name          Zoom Textarea
// @namespace     http://diveintogreasemonkey.org/download/
// @description   add controls to zoom textareas
// @include       *
// ==/UserScript==

var textareas, textarea;

textareas = document.getElementsByTagName('textarea');
if (!textareas.length) { return; }

function textarea_zoom_in(event) {
    var link, textarea, s;
    link = event.currentTarget;
    textarea = link._target;
    s = getComputedStyle(textarea, "");
    textarea.style.width = (parseFloat(s.width) * 1.5) + "px";
    textarea.style.height = (parseFloat(s.height) * 1.5) + "px";
    textarea.style.fontSize = (parseFloat(s.fontSize) + 7.0) + 'px';
    event.preventDefault();
}

function textarea_zoom_out(event) {
    var link, textarea, s;
    link = event.currentTarget;
    textarea = link._target;
    s = getComputedStyle(textarea, "");
    textarea.style.width = (parseFloat(s.width) * 2.0 / 3.0) + "px";
    textarea.style.height = (parseFloat(s.height) * 2.0 / 3.0) + "px";
    textarea.style.fontSize = (parseFloat(s.fontSize) - 7.0) + "px";
    event.preventDefault();
}

function createButton(target, func, title, width, height, src) {
    var img, button;
    img = document.createElement('img');
    img.width = width;
    img.height = height;
    img.style.borderTop = img.style.borderLeft = "1px solid #ccc";
    img.style.borderRight = img.style.borderBottom = "1px solid #888";
```

```
    img.style.marginRight = "2px";
    img.src = src;
    button = document.createElement('a');
    button._target = target;
    button.title = title;
    button.href = '#';
    button.onclick = func;
    button.appendChild(img);
    return button;
}

for (var i = 0; i <textareas.length; i++) {
    textarea = textareas[i];
    textarea.parentNode.insertBefore(
        createButton(
            textarea,
            textarea_zoom_in,
            'Increase textarea size',
            20,
            20,
            ''+
'R0lGODlhFAAUAOYAANPS1tva3uTj52NjY2JiY7KxtPf3%2BLOys6WkpmJiYvDw8fX19vb'+
'296Wlpre3uEZFR%2B%2Fv8aqpq9va3a6tr6Kho%2Bjo6bKytZqZml5eYMLBxNra21JSU3'+
'Jxc3RzdXl4emJhZOvq7KamppGQkr29vba2uGBgYdLR1dLS0lBPUVRTVYB%2Fgvj4%2BYK'+
'Bg6SjptrZ3cPDxb69wG1tbsXFxsrJy29vccDAwfT09VJRU6uqrFlZW6moqo2Mj4yLjLKy'+
's%2Fj4%2BK%2Busu7t783Nz3l4e19fX7u6vaalqNPS1MjHylZVV318ftfW2UhHSG9uccv'+
'KzfHw8qqqrNPS1eXk5tvb3K%2BvsHNydeLi4OpKS2JhY2hnalpZWlVVVtDQ0URDRJmZm5'+
'mYm11dXp2cnm9vcFxcXaOjo0pJSsC%2FwuXk6AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'+
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAC'+
'H5BAAAAAAALAAAAAAUABQAAAeagGaCg4SFhoeIiYqKTSQUFwgwi4JlB0pOCkEiRQKKRxM'+
'gKwMGDFEqBYpPRj4GAwwLCkQsijwQBAQJCUNSW1mKSUALNiVVJzIvSIo7GRUaGzUOPTpC'+
'igUeMyNTIWMHGC2KAl5hCBENYDlcWC7gOB1LDzRdWlZMAZOEJl83VPb3ggAfUnDo5w%2F'+
'AFRQxJPj7J4aMhYWCoPyASFFRIAA7'),
            textarea);
    textarea.parentNode.insertBefore(
        createButton(
            textarea,
            textarea_zoom_out,
            'Decrease textarea size',
            20,
            20,
            ''+
'R0lGODlhFAAUAOYAANPS1uTj59va3vDw8bKxtGJiYrOys6Wkpvj4%2BPb29%2FX19mJiY'+
'%2Ff3%2BKqqrLe3uLKytURDRFpZWqmoqllZW9va3aOjo6Kho4KBg729vWJhZK%2BuskZF'+
'R4B%2FgsLBxHNydY2Mj%2Ff396amptLS0l9fX9fW2dDQ0W1tbpmZm8DAwfT09fHw8n18f'+
'uLi49LR1V5eYOjo6VBPUa6tr769wEhHSNra20pJStPS1KuqrNPS1ZmYm%2B7t77Kys8rJ'+
'y%2Fj4%2BaSjpm9uca%2BvsMjHyqalqHRzdVJRU8PDxVRTVcvKzc3Nz0pKS9rZ3evq7MC'+
'%2FwsXFxp2cnnl4e1VVVu%2Fv8ba2uM7Oz29vcbu6vZqZmnJxc9vb3PHx8uXk5mhnamJh'+
'Y1xcXZGQklZVV29vcHl4eoyLjKqpq6Wlpl1dXuXk6AAAAAAAAAAAAAAAAAAAAAAAAAAA'+
```

```
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'+
'AAACH5BAAAAAAALAAAAAAUABQAAAeZgGaCg4SFhoeIiYqKR1IWVgcyi4JMBiQqAOheQgG'+
'KQTFLPQgMCVocBIoNNqMgCQoDVReKYlELCwUFI1glEYorOgopWSwiTUVfih8dLzRTKA47'+
'Ek%2BKBGE8GEAhFQYuPooBOWAHY2ROExBbSt83QzMbVCdQST8Ck4QtZUQe9faCABlGrvD'+
'rB4ALDBMU%2BvnrUuOBQkE4NDycqCgQADs%3D'),
        textarea);
    textarea.parentNode.insertBefore(
        document.createElement('br'),
        textarea);
}
```

このコードは複雑に見え，また実際複雑なのですが，理由は様々です．中にある訳のわからない文字列のために複雑に見えます．これは data:URI で，むちゃくちゃに見えますが作るのは簡単です．本当に複雑なのは別のところです．

まず，ページ上の <textarea> 要素を全て取得します．これについては特定の HTML 要素全てになにか操作を行うを見てください．なければ，続けても仕方ないので return します

```
textareas = document.getElementsByTagName('textarea');
if (!textareas.length) { return; }
```

少し飛ばします．我々の「ツールバーは実質的にはボタンが横に並んだものです．しかし，各ボタンは押せるように見える何かの画像にすぎません．さらにそれぞれ Javascript のスクリプトを実行するようなリンクにくくられています．

一つ以上のボタン（このスクリプトでは 2 つですが，より機能を追加して拡張することも容易でしょう）をつくるので，ボタンを作る過程をひとまとめにした関数を作ります．

```
function createButton(target, func, title, width, height, src) {
```

---

createButton 関数は 6 つの引数をとります．

target 要素オブジェクト．このボタンが制御する <textarea> 要素．

func 関数オブジェクト．ユーザがマウスでクリックあるいはキーボードで **Enter** を押したときに呼ばれる Javascript 関数．

title 文字列．ボタンの上にカーソルを合わせたときに出てくるテキスト．

width 整数．ボタンの幅．src で与えられた画像の幅であるべき．

height 整数．ボタンの高さ．src で与えられた画像の高さであるべき．

src 文字列．URL，パス，あるいはボタン画像の data:URI．

ボタンを作るのは 2 段階に別れます．最初に <img> 要素を作り，さらにその周りに <a> 要素を作ります．

document.createElement を呼び出して <img> 要素をつくり，スタイルを含むいくつかの属性をセットします．これについては要素のスタイルを指定するを見てください．

```
img = document.createElement('img');
img._target = target;
img.width = width;
img.height = height;
img.style.borderTop = img.style.borderLeft = "1px solid #ccc";
img.style.borderRight = img.style.borderBottom = "1px solid #888";
img.style.marginRight = "2px";
img.src = src;
```

ここで注意しておきたいことは 2 つあります．とても便利だと思うので．次の構文を使うと 2 つの属性に同時に同じ値を代入できます．

```
img.style.borderTop = img.style.borderLeft = "1px solid #ccc";
```

それでは先に進みましょう．ボタンを作ることの残り半分はリンク（<a> 要素）をつくって <img> 要素をその中に入れることです．

```
button = document.createElement('a');
button._target = target;
button.title = title;
button.href = '#';
button.onclick = func;
button.appendChild(img);
```

ここで指摘しておきたいことは 2 つです．まず，偽の href 属性をリンクに与える必要があります．そうしないと Firefox はそれをアンカーだと判断してタブインデックスに追加しません（つまり，タブでそこに移動することが出来ない，キーボードアクセスが出来ない，ということです）．次に，_target 属性を設定してターゲットとなる <textarea> への参照を保存しています．これは Javascript では何も問題ありません．オブジェクトに新たな属性を作ってそれに値を代入することが出来ます．あとで onclick イベントハンドラのなかでこの自前の _target 属性を参照します．

onclick ハンドラ自体に戻りましょう．各ハンドラは一つの引数 event をとる関数です．

```
function textarea_zoom_in(event)
```

event オブジェクトはいくつかの属性をもち，ここで関心があるのは currentTarget 一つだけです．

```
    link = event.currentTarget;
```

Event オブジェクトについてのドキュメントをよめばいくつかのターゲットに関する属性があり，その一つがたんなる target であるということがわかるでしょう．クリックしたときのリンクへの参照を取得するのに event.target を使いたくなるかもしれませんが，(私の意見では）それは不整合を起こします．ユーザがタブでボタンに移動して **Enter** を押したときには event.target がリンクですが，ユーザがマウスでボタンをクリックしたときには event.target はリンクの中にある画像なのです！これにはもっともな理由があるのでしょうが，それは DOM イベントモデルについての私の理解を超えています．どちらにしても，全ての場合で event.currentTarget はリンクを返すのでそれを使います．

次に，実際にズームする <textarea> への参照を取得します．ボタンを作ったときにセットした_target 属性を使います．

```
    textarea = link._target;
```

ここからが楽しいところです．(もう既に楽しいと思っていることでしょう！）ズームするには <textarea> の現在の寸法とフォントサイズを取得する必要があります．textarea.style から適当な属性（textarea.style.width, textarea.style.height, textarea.style.fontSize）をとってくるだけではうまくいきません．なぜならこれらの値がセットされるのはページが <textarea> 自体の属性値によってセットしている場合だけだからです．これは求めているものと違います．必要なのは実際の現在のスタイルです．このためには getComputedStyle が必要です．この関数については要素のスタイルを取得するを見てください．

```
    s = getComputedStyle(textarea, "");
    textarea.style.width = (parseFloat(s.width) * 1.5) + "px";
    textarea.style.height = (parseFloat(s.height) * 1.5) + "px";
    textarea.style.fontSize = (parseFloat(s.fontSize) + 7.0) + 'px';
```

最後に，キーボードアクセスが可能であるようにボタンリンクに付け加えた偽の href を覚えていますか?さて，もはやこれは邪魔になります．なぜなら Firefox は onclick ハンドラを実行するとリンクをたどろうとするからです．リンクが存在しないアンカーをさしているので，Firefox はボタンがどこにあろうとページトップにジャンプします．これでは困るので，やめさせるには自分の onclick ハンドラの実行を終える前に event.preventDefault() を呼び出します．

```
    event.preventDefault();
```

残りはそのまんまです．すべての <textarea> をループし，それぞれに（適当な onclick ハンドラとボタン画像をもった）ズームボタンをつくり，それを <textarea> の前に挿入します．各画像についてインラインの画像を作るために data:URI を使いました．これで，ユーザはボタンの画像を取得するのにサーバーにアクセスしなくて済みます．詳しくは要素の前にコンテンツを挿入するとサーバに接続せずに画像を加えるを見てください．

```
for (var i = 0; i <textareas.length; i++) {
    textarea = textareas[i];
    textarea.parentNode.insertBefore(
        createButton(
            textarea,
            textarea_zoom_in,
            'Increase textarea size',
            20,
            20,
            ''+
'R0lGODlhFAAUAOYAANPS1tva3uTj52NjY2JiY7KxtPf3%2BLOys6WkpmJiYvDw8fX19vb'+
'296Wlpre3uEZFR%2B%2Fv8aqpq9va3a6tr6Kho%2Bjo6bKytZqZml5eYMLBxNra21JSU3'+
'Jxc3RzdXl4emJhZOvq7KamppGQkr29vba2uGBgYdLR1dLS0lBPUVRTVYB%2Fgvj4%2BYK'+
'Bg6SjptrZ3cPDxb69wG1tbsXFxsrJy29vccDAwfT09VJRU6uqrFlZW6moqo2Mj4yLjLKy'+
's%2Fj4%2BK%2Busu7t783Nz3l4e19fX7u6vaalqNPS1MjHylZVV318ftfW2UhHSG9uccv'+
'KzfHw8qqqrNPS1eXk5tvb3K%2BvsHNydeLi40pKS2JhY2hnalpZWlVVVtDQ0URDRJmZm5'+
'mYm11dXp2cnm9vcFxcXaOjo0pJSsC%2FwuXk6AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'+
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAC'+
'H5BAAAAAAALAAAAAAUABQAAAeagGaCg4SFhoeIiYqKTSQUFwgwi4JlB0pOCkEiRQKKRxM'+
'gKwMGDFEqBYpPRj4GAwwLCkQsijwQBAQJCUNSW1mKSUALNiVVJzIvSIo7GRUaGzUOPTpC'+
'igUeMyNTIWMHGC2KAl5hCBENYDlcWC7gOB1LDzRdWlZMAZOEJl83VPb3ggAfUnDo5w%2F'+
'AFRQxJPj7J4aMhYWCoPyASFFRIAA7'),
        textarea);
    textarea.parentNode.insertBefore(
        createButton(
            textarea,
            textarea_zoom_out,
            'Decrease textarea size',
            20,
            20,
            ''+
'R0lGODlhFAAUAOYAANPS1uTj59va3vDw8bKxtGJiYrOys6Wkpvj4%2BPb29%2FX19mJiY'+
'%2Ff3%2BKqqrLe3uLKytURDRFpZWqmoqllZW9va3aOjo6Kho4KBg729vWJhZK%2BuskZF'+
'R4B%2FgsLBxHNydY2Mj%2Ff396amptLS0l9fX9fW2dDQ0W1tbpmZ8DAwfT09fHw8n18f'+
'uLi49LR1V5eYOjo6VBPUa6tr769wEhHSNra20pJStPS1KuqrNPS1ZmYm%2B7t77Kys8rJ'+
'y%2Fj4%2BaSjpm9uca%2BvsMjHyqalqHRzdVJRU8PDxVRTVcvKzc3Nz0pKS9rZ3evq7MC'+
'%2FwsXFxp2cnnl4e1VVVu%2Fv8ba2uM7Oz29vcbu6vZqZmnJxc9vb3PHx8uXk5mhnamJh'+
'Y1xcXZGQklZVV29vcHl4eoyLjKqpq6Wlpl1dXuXk6AAAAAAAAAAAAAAAAAAAAAAAAAAA'+
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'+
```

```
'AAACH5BAAAAAAALAAAAAAUABQAAAeZgGaCg4SFhoeIiYqKR1IWVgcyi4JMBiQqA0heQgG'+
'KQTFLPQgMCVocBIoNNqMgCQoDVReKYlELCwUFI1glEYorOgopWSwiTUVfih8dLzRTKA47'+
'Ek%2BKBGE8GEAhFQYuPooBOWAHY2ROExBbSt83QzMbVCdQST8Ck4QtZUQe9faCABlGrvD'+
'rB4ALDBMU%2BvnrUuOBQkE4NDycqCgQADs%3D'),
        textarea);
    textarea.parentNode.insertBefore(
        document.createElement('br'),
        textarea);
}
```

ダウンロード

- zoomtextarea.user.js

参考サイト

- Event documentation
- 特定の HTML 全てになにか操作を行う
- 要素のスタイルを取得する
- 要素のスタイルを設定する
- 要素の前にコンテンツを挿入する
- サーバに接続せずに画像を加える

5.8 事例研究: Access Bar

Access Bar はあるページで定義されているを表示します．アクセスキーは，ページの作者に寄って定義されたショートカットキーによって特定のリンクやフォームにジャンプすることを可能にするアクセシビリティ機能です．（アクセスキーについてもっと知る．）

Firefox はアクセスキーをサポートしていますが，ページでどのキーが定義されているのかを表示しません．そこで，Access Bar が誕生しました．

リスト 43: accessbar.user.js

```
// ==UserScript==
// @name           Access Bar
// @namespace      http://diveintogreasemonkey.org/download/
// @description    show accesskeys defined on page
// @include        *
// ==/UserScript==

function addGlobalStyle(css) {
```

```
    var head, style;
    head = document.getElementsByTagName('head')[0];
    if (!head) { return; }
    style = document.createElement('style');
    style.type = 'text/css';
    style.innerHTML = css;
    head.appendChild(style);
}

var akeys, descriptions, a, desc, label, div;
akeys = document.evaluate(
    "//*[@accesskey]",
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
if (!akeys.snapshotLength) { return; }
descriptions = new Array();
desc = '';
for (var i = 0; i <akeys.snapshotLength; i++) {
    a = akeys.snapshotItem(i);
    desctext = '';
    if (a.nodeName == 'INPUT') {
        label = document.evaluate("//label[@for='" + a.name + "']",
            document,
            null,
            XPathResult.FIRST_ORDERED_NODE_TYPE,
            null).singleNodeValue;
        if (label) {
            desctext = label.title;
            if (!desctext) { desctext = label.textContent; }
        }
    }
    if (!desctext) { desctext = a.textContent; }
    if (!desctext) { desctext = a.title; }
    if (!desctext) { desctext = a.name; }
    if (!desctext) { desctext = a.id; }
    if (!desctext) { desctext = a.href; }
    if (!desctext) { desctext = a.value; }
    desc = '[' +
        a.getAttribute('accesskey').toUpperCase() + '] ';
    if (a.href) {
        desc += '' + desctext + '';
    } else {
        desc += desctext;
    }
    descriptions.push(desc);
}
descriptions.sort();
```

```
div = document.createElement('div');
div.id = 'accessbar-div-0';
desc = '' + descriptions[0] + '' + descriptions[i] + '';
div.innerHTML = desc;
document.body.style.paddingBottom = "4em";
window.addEventListener(
    "load",
    function() {
        document.body.appendChild(div);
    },
    true);
addGlobalStyle(
'#accessbar-div-0 {'+
'  position: fixed;' +
'  left: 0;' +
'  right: 0;' +
'  bottom: 0;' +
'  top: auto;' +
'  border-top: 1px solid silver;' +
'  background: black;' +
'  color: white;' +
'  margin: 1em 0 0 0;' +
'  padding: 5px 0 0.4em 0;' +
'  width: 100%;' +
'  font-family: Verdana, sans-serif;' +
'  font-size: small;' +
'  line-height: 160%;' +
'}' +
'#accessbar-div-0 a,' +
'#accessbar-div-0 li,' +
'#accessbar-div-0 span,' +
'#accessbar-div-0 strong {' +
'  background-color: transparent;' +
'  color: white;' +
'}' +
'#accessbar-div-0 div {' +
'  margin: 0 1em 0 1em;' +
'}' +
'#accessbar-div-0 div ul {' +
'  margin-left: 0;' +
'  margin-bottom: 5px;' +
'  padding-left: 0;' +
'  display: inline;' +
'}' +
'#accessbar-div-0 div ul li {' +
'  margin-left: 0;' +
'  padding: 3px 15px;' +
'  border-left: 1px solid silver;' +
'  list-style: none;' +
```

```
’  display: inline;’ +
’}’ +
’#accessbar-div-0 div ul li.first {’ +
’  border-left: none;’ +
’  padding-left: 0;’ +
’}’);
```

コードは 6 段階からなります．

1. ヘルパー関数 addGlobalStyle を定義します．
2. ページ上の accesskey 属性を持つ要素をすべて見つけます．
3. これらの要素をループして各要素を説明する適切なテキストを決定します．
4. accesskey が有効な要素へのリンクの，ソートされたリストを作ります．
5. 通常の ul と li を使って，ページにソートされたリストをページに追加します．
6. 表示されている最下部に固定された偽のステータスバーであるかにみえるように，リストのスタイルを定義します．

まず最初に，ヘルパー関数 addGlobalStyle が必要です．これを（第 6 段階で）使って自分の CSS を挿入します．これについては CSS を追加するを見てください．

```
function addGlobalStyle(css) {
    var head, style;
    head = document.getElementsByTagName(’head’)[0];
    if (!head) { return; }
    style = document.createElement(’style’);
    style.type = ’text/css’;
    style.innerHTML = css;
    head.appendChild(style);
}
```

第 2 段階では accesskey 属性を含んだページないの要素を全てリストします．これは Firefox の XPath サポートを使えば簡単です．XPath クエリが何も結果を返さなければ，そこでやめるということに注意してください．なぜなら，その場合は何も表示するものがないからです．これについては特定の属性を持った要素全てになにか操作を行うを見てください．

```
var akeys, descriptions, a, i, desc, label, div;
akeys = document.evaluate(
    "//*[@accesskey]",
    document,
    null,
    XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,
    null);
if (!akeys.snapshotLength) { return; }
```

第 3 段階では accesskey が有効な要素それぞれの適切な説明のリストを作ります．ここがこのスクリプトで最も複雑なところです．問題は，accesskey が異なったいくつかの HTML 要素に現れるということです．

フォームの中の input 要素は accesskey を定義することが出来ます．input 要素は，input フィールドに対するすテキストラベルを含む label を持っているかもしれませんし，持っていないかもしれません．もし持っている場合は，label が title 属性を持っていて，それが input フィールドについてのより詳細な情報を与えてくれます．あるいは label 属性が単なるテキストを持っているかもしれません．あるいはほかにも，input 要素が関連づけられたラベルを持たない場合もあります．その場合 input の value 属性をとるのがせいぜいです．

他方，label が記述する input 要素ではなくて，label 自体が accesskey を定義することも出来ます．この場合も label 要素の title 属性を探し，もし title 属性がなければラベルのテキストに戻ります．

リンクも accesskey 属性を定義することが出来ます．その場合，リンクテキストが自明な選択です．しかし，リンクがテキストを持たない場合（たとえば画像のみを含む場合）は，リンクの title 属性を次に探します．テキストも title もなければ，name 属性をさがし，それもだめなら id 属性を探します．

経験則じゃ満足できないですよね？以下が完全なアルゴリズムです．akeys は XPathResult オブジェクトなので，各結果を参照するには akeys.snapshotItem(i) を使う必要があることを思い出してください．

```
akeys is an XPathResult object, so I need to get each result by calling \
    akeys.snapshotItem(i).

descriptions = new Array();
desc = '';
for (var i = 0; i <akeys.snapshotLength; i++) {
    a = akeys.snapshotItem(i);
    desctext = '';
    if (a.nodeName == 'INPUT') {
        label = document.evaluate("//label[@for='" + a.name + "']",
            document,
            null,
```

```
            XPathResult.FIRST_ORDERED_NODE_TYPE,
            null).singleNodeValue;
        if (label) {
            desctext = label.title;
            if (!desctext) { desctext = label.textContent; }
        }
    }
    if (!desctext) { desctext = a.textContent; }
    if (!desctext) { desctext = a.title; }
    if (!desctext) { desctext = a.name; }
    if (!desctext) { desctext = a.id; }
    if (!desctext) { desctext = a.href; }
    if (!desctext) { desctext = a.value; }
    desc = '[' +
        a.getAttribute('accesskey').toUpperCase() + '] ';
    if (a.href) {
        desc += '' + desctext + '';
    } else {
        desc += desctext;
    }
    descriptions.push(desc);
}
```

Javasript 配列はその場で配列をソートする sort メソッドを持っているので，第 4 段階は単純です．

```
descriptions.sort();
```

第 5 段階は HTML をつくって accesskey が有効な要素のリストを表示します．「入れ物」である 要素を作り，アクセスキーのリストの HTML を の innerHTML にセットし，さらにページの末尾にそれを追加します．ユーザースクリプトが実行されるタイミング上の理由で，ページへの複雑な変更は読み込みが終わった後の方がよいです．そこで，window.addEventListener を使って入れ物の をページに追加する onload イベントを加えています．

innerHTML については複雑な HTML を素早く挿入するを，window.addEventListener についてはレンダリング後の事後処理を見てください．

```
div = document.createElement('div');
div.id = 'accessbar-div-0';
desc = '' + descriptions[0] + '' + descriptions[i] + '';
div.innerHTML = desc;
document.body.style.paddingBottom = "4em";
window.addEventListener(
    "load",
```

```
    function() {
        document.body.appendChild(div);
    },
    true);
```

最後，第 6 段階です．CSS を追加して挿入した HTML がきれいに見えるようにしています．（具体的には，それはページの最下部に固定された黒いバーのようにみえ，ページをスクロールしてもそのまま動きません．これが可能なのは Firefox がディスプレイタイプ `position: fixed` をサポートしているからです．詳しくは CSS を追加するを見てください．

```
addGlobalStyle(
'#accessbar-div-0 {'+
'  position: fixed;' +
'  left: 0;' +
'  right: 0;' +
'  bottom: 0;' +
'  top: auto;' +
'  border-top: 1px solid silver;' +
'  background: black;' +
'  color: white;' +
'  margin: 1em 0 0 0;' +
'  padding: 5px 0 0.4em 0;' +
'  width: 100%;' +
'  font-family: Verdana, sans-serif;' +
'  font-size: small;' +
'  line-height: 160%;' +
'}' +
'#accessbar-div-0 a,' +
'#accessbar-div-0 li,' +
'#accessbar-div-0 span,' +
'#accessbar-div-0 strong {' +
'  background-color: transparent;' +
'  color: white;' +
'}' +
'#accessbar-div-0 div {' +
'  margin: 0 1em 0 1em;' +
'}' +
'#accessbar-div-0 div ul {' +
'  margin-left: 0;' +
'  margin-bottom: 5px;' +
'  padding-left: 0;' +
'  display: inline;' +
'}' +
'#accessbar-div-0 div ul li {' +
'  margin-left: 0;' +
'  padding: 3px 15px;' +
```

```
'  border-left: 1px solid silver;' +
'  list-style: none;' +
'  display: inline;' +
'}' +
'#accessbar-div-0 div ul li.first {' +
'  border-left: none;' +
'  padding-left: 0;' +
'}');
```

ダウンロード

- accessbar.user.js

参考

- CSS を追加する
- 特定の属性を持った要素全てになにか操作を行う
- 複雑な HTML を素早く挿入する
- レンダリング後の事後処理

第6章　発展事項

6.1　長期的に残るデータを保存・取得する

Greasemonkey が定義する 2 つの関数 GM_setValue と GM_getValue を使うとユーザースクリプトだけが参照できる「プライベートな」データを保存したり参照したりすることが出来ます．（ほかのユーザースクリプトを含む．ほかのスクリプトはそれらを参照できません．）これらの関数を使うと，スクリプトに特有の設定を保存したり，ページ間に残るキャッシュを保持したり，永続的な実行のログをとったりすることが出来ます．

注意

GM_setValue により保存されるデータや GM_getValue により取得されるデータはブラウザのクッキーに似ていますが，重大な違いがあります．どちらもローカルマシンに保存されますが，クッキーがドメインに特有で元のドメインからしか参照できないのに対して，Greasemonkey の設定値はスクリプトに特有でそれらを作ったユーザースクリプトからしか参照できません（ユーザースクリプトが現在実行されている URL は関係ありません）．またクッキーと違ってユーザースクリプトのデータは決してリモートサーバにお送信されません．

GM_setValue はスクリプト特有の設定値を保存し，GM_getValue はそれを取得します．

```
function GM_setValue(key, value);

function GM_getValue(key, defaultValue);
```

引数 key は特に決まったフォーマットのない文字列です．value は文字列，ブール値，整数のいずれかです．GM_getValue の defaultValue はオプションです．あれば，key が存在しないときにその値が返されます．もし defaultValue が与えられておらず key が存在しないならば，GM_getValue は undefined を返します．

これらの関数は Gresemonkey 0.3 で導入されました．これらを使うときには存在するかどうかチェックし，なければ適切に品質を落とすのが良いでしょう．

参考サイト

- MyPIPsTag は最初に実行したときにユーザ名を求めるプロンプトを出します．

- POST Interceptor は (GM_registerMenuCommand をつかって) スクリプトが有効にするどうかを切り替えるメニュー項目を追加します.
- MSDN Language Filter はページそのものに設定オプションを挿入します.

参考

- GM_getValue
- GM_setValue

6.2 メニューバーに項目を追加する

Greasemonkey は GM_registerMenucommand を定義しており，これを使うとユーザースクリプトで Firefox メニューバーに項目を追加することが可能です．登録されたメニュー項目はユーザースクリプトが有効であるときに **User Script Commands** サブメニューに現れます.

```
function GM_registerMenuCommand(menuText, callbackFunction);
```

---

menuText は追加するメニュー項目を含む文字列です．callbackFunction は関数オブジェクトです．メニュー項目が選択されたときに関数 callbackFunction が呼ばれます.

この関数は Greasemonkey 0.2.6 で導入されました．使うときには存在するかチェックして，ない場合には適切に品質を落とすのが良いでしょう.

参考サイト

- POST Interceptor はスクリプトが有効にするどうかを切り替えるメニュー項目を追加します.

参考

- GM_registerMenuCommand

6.3 ほかのサイトからのデータを統合する

Greasemonkey は関数 GM_xmlhttpRequest を定義しており，これを使うと任意の URL から GET したり，任意の URL に POST することが出来ます.

より詳しくは GM_xmlhttpRequest を見てください.

この関数は Greasemonkey 0.2.6 で導入されました．使うときには存在するかチェックして，ない場合には適切に品質を落とすのが良いでしょう.

参考サイト

- LibraryLookup は Amazon.com をあなたの地域の図書館に統合します.

- Annotate Google は Google と del.icio.us を統合します．
- Bloglines Tweaks は **Expand** に Bloglines の記事のまとめに **Expand** ボタンを追加して，このボタンを押すと記事全体が表示されます．
- Flick Batch Enhancer は `GM_xmlhttpRequest` と Flickr の持つ REST API を使って Flickr に機能を追加します．
- Hide Google Redirects は Google Personal Search History を再プログラムして通常の `<a href="...">` リンクを使うようにし，なおかつ適切なトラッキングする URL においては `GM_xmlhttpRequest` をつかってクリックをトラックするようにしています．

6.4 ユーザスクリプトを拡張にコンパイルする

あなたのユーザースクリプトがGreasemonkey のアーキテクチャでは手狭になってきましたか？ Javascript の特別な関数，ローカルファイル，あるいはブラウザ拡張にしか使えない Firefox の機能を参照する必要が生じましたか？何回かクリックするだけでユーザースクリプトを本格的な `XPI` に変換することが出来ます．これは Adrian Holovaty のすばらしい Greasemonkey Compiler のおかげです！

手順

1. Butler ユーザースクリプトのソースを開きます．メニューから**編集**→ **すべて選択**を選びクリップボードにソースコードをコピーします．
2. Greasemonkey compiler を開きます．
3. ”Javascript”にいき，メニューから**編集**→ **貼付け**をえらんで Butler のソースコードを貼付けます．
4. ”Creator”フィールドに **Mark Pilgrim** と入力します．
5. ”Version”フィールドに Butler の現在のバージョン（翻訳の時点では 0.3，実際のバージョンをスクリプトのメタデータで確認してください）を入力します．
6. 新しいウィンドウかタブで GUID Generator を開きます．ここでランダムな GUID を生成します．{} も含めて GUID をクリップボードにコピーします．
7. Greasemonkey compiler のページにもどり，”GUID”フィールドに GUID Generator で取得した GUID を貼付けます．
8. ”Homepage”フィールドに `http://diveintomark.org/projects/butler/`を入力します．

9. **Creater Firefox Extension** をクリックします．Firefox が”Opening butler.xpi”というダイアログを出すはずです．**Save to disk** を選んでディレクトリを選びます．

これで Butler のブラウザ拡張バージョンができました．Butler 拡張をインストールする前に，ユーザースクリプトの Butler を無効にし，Google でなにか検索して確かに無効になっていることを確かめた方がよいでしょう．確認したら**ファイル**→**開く**から Greasemonkey Compiler で作った `butler.xpi` を選んでインストールしましょう．インストールが終わったら忘れずに Firefox を再起動してください．

`.xpi` ファイルは特有のディレクトリ構造を持った ZIP アーカイブです．ZIP プログラム（Windows なら 7-zip，Mac なら Stuffit Expander など）をつかえばアーカーイブを展開してブラウザ拡張を構成するファイルを見ることが出来ます．

```
butler.xpi
|
+-- install.rdf
|
+-- chrome/
    |
    +-- butler/
        |
        +-- content/
            |
            +-- browser.xul
            |
            +-- contents.rdf
            |
            +-- javascript.js
```

4 つのファイルからなっています．二つの RDF ファイルは主に Firefox の定型文です．残りの二つが実際のコードを含みます．

install.rdf 拡張自体についてのメタデータ．名前，バージョン，説明，使える Firefox のバージョンなどが含まれる．

browser.xul スクリプトの`@include` と`@exclude` に現在の URL を照らし合わせ，スクリプトを挿入・実行するブートストラップコード．

contents.rdf ほかのメタデータ．Firefox の定型文．

javascript.js 元のユーザースクリプトのソースコード．

基本的には Greasemonkey compiler は Greasemonkey のかなりスリムにしたバージョンを生成し，生成されたものはユーザーインターフェイスをもたず，適当な URL で単一のユーザースクリプトを自動的に実行するだけです．しかし，ソースコードがあるので，好きなように出来ます．自前のダイアログを作る，環境設定を変更する，**User Script Commands** の外にメニュー項目を登録する，などです．さあがんばってください．

参考サイト

- Extension Developer's Extension は Firefox 拡張をデバグ・テストするのにとても役に立ちます．

第7章　Greasemonkey API リファレンス

7.1　GM_log

7.1.1　名前

GM_log: Javascript コンソールへログを表示する

7.1.2　概要

```
function GM_log(message);
```

7.1.3　説明

GM_log は Javascript コンソールに出力する．主にデバグのために使われる．

7.1.4　履歴

GM_log は Greasemonkey 0.3 で導入された．

参考

- GM_log による記録
- Javascript コンソールによるエラーのトラッキング

7.2　GM_getValue

7.2.1　名前

GM_getValue:スクリプト固有の設定値を取得する．

7.2.2　概要

```
returntype GM_getValue(key, defaultValue);
```

7.2.3 説明

GM_getValue はスクリプト固有の設定値を取得する．返り値は文字列，ブール値，整数のいずれか．*key* は特定のフォーマットを持たない文字列．*defaultValue* はオプション．あれば，*key* が存在しないときにその値が返される．もし *defaultValue* が与えられておらず *key* が存在しないならば，GM_getValue は undefined を返す．

Greasemonkey 設定値ははブラウザのクッキーに似ているが，重大な違いがある．どちらもローカルマシンに保存されるのは共通だが，クッキーがドメインに特有で元のドメインからしか参照できないのに対して，Greasemonkey の設定値はスクリプトに特有でそれらを作ったユーザースクリプトからしか参照できない（ユーザースクリプトが現在実行されている URL は無関係）．またクッキーと違ってユーザースクリプトのデータは決してリモートサーバにお送信されない．

ヒント

保存された設定値は about:config を開き greasemonkey.scriptvals でフィルタリングすると見ることが出来る．

7.2.4 履歴

GM_getValue は Greasemonkey 0.3 で導入された．

参考

- GM_setValue

7.3 GM_setValue

7.3.1 名前

GM_setValue:スクリプト固有の設定値をセットする．

7.3.2 概要

```
function GM_setValue(key, value);
```

7.3.3 説明

GM_setValue はスクリプト固有の設定値を保存する．*key* は特にフォーマットのない文字列．*value* は文字列，ブール値，整数のいずれか．どちらの引数も必須である．

Greasemonkey 設定値ははブラウザのクッキーに似ているが，重大な違いがある．どちらもローカルマシンに保存されるのは共通だが，クッキーがドメインに特有で元のドメイ

ンからしか参照できないのに対して，Greasemonkey の設定値はスクリプトに特有でそれらを作ったユーザースクリプトからしか参照できない（ユーザースクリプトが現在実行されている URL は無関係）．またクッキーと違ってユーザースクリプトのデータは決してリモートサーバにお送信されない．

7.3.4 履歴

`GM_setValue` は Greasemonkey 0.3 で導入された．

参考

- GM_getValue

7.4 GM_registerMenuCommand

7.4.1 名前

GM_registerMenuCommand:ユーザスクリプトコマンドサブメニューに項目を加える．

7.4.2 概要

`function GM_registerMenuCommand(`*`menuText`*`,`*`callbackFunction`*`);`

7.4.3 説明

`GM_registerMenuCommand` は**ツール→User Script Commands** にメニュー項目を追加する．*`menuText`* は追加するメニュー項目のテキストを含む文字列．*`callbackFunction`* は関数オブジェクトで，メニュー項目が選択されたときに呼ばれる．

```
function callbackFunction(e):
```

---

e はメニュー選択イベント．これが何かの役に立つのかはわからない．

7.4.4 バグ

メニューのアクセラレータキーを設定できない．`GM_registerMenuCommand('Some &menu text', myFunction)` を呼ぶと"Some &menu text"というメニュー項目が出来る．（バグ 10090）

7.4.5 履歴

GM_registerMenuCommand は Greasemonkey 0.2.6 で導入された．

7.5 GM_xmlhttpRequest

7.5.1 名前

GM_xmlhttpRequest:任意の HTTP リクエストを行う．

7.5.2 概要

GM_xmlhttpRequest(*details*);

7.5.3 説明

GM_xmlhttpRequest は任意の HTTP リクエストを行う．*details* は 7 つまでのフィールドを持つオブジェクト．

method 文字列．このリクエストで使う HTTP メソッド．必須．通常は GET だが，POST, PUT, DELETE など HTTP の動詞なら何でもよい．

url 文字列．このリクエストで使う URL．必須．

headers このリクエストに含める HTTP ヘッダーの連想配列．オプション引数．デフォルトは空配列．例:headers: {'User-Agent': 'Mozilla/4.0 (compatible) Greasemonkey','Accept': 'application/atom+xml,application/xml,text/xml'}

data 文字列．HTTP リクエストの本体．オプション引数．デフォルトは空文字列．フォームのポストをシミュレートしている場合 (method=='POST'), headers フィールドには'application/x-www-form-urlencoded' という Content-type, data フィールドには URL でエンコードされた形式のデータを含めなければならない．

onload 関数オブジェクト．リクエストが正常に終了した場合に呼ばれるコールバック関数．

onerror 関数オブジェクト．リクエストの処理中にエラーが発生した場合に呼ばれるコールバック関数．

onreadystatechange 関数オブジェクト．リクエストが処理中である間繰り返し呼ばれるコールバック関数．

onload callback

`onload` のためのコールバック関数は一つの引数 *`responseDetails`* をとる．

```
function onloadCallback(responseDetails);
```

`responseDetails` は 5 つのフィールドからなるオブジェクトである．

status 整数．レスポンスの HTTP ステータスコード．200 がリクエストが正常に終了したことを意味する．

statusText 文字列．HTTP ステータステキスト．ステータステキストはサーバによって異なる．

responseHeaders 文字列．レスポンスに含まれる HTTP ヘッダー．

responseText 文字列．レスポンスの本体．

readyState 使われていない．

onerror callback

`onerror` のためのコールバック関数は一つの引数 *`responseDetails`* をとる．

```
function onerrorCallback(responseDetails);
```

`responseDetails` は 5 つのフィールドからなオブジェクトである．

status 整数．レスポンスの HTTP エラーコード．404 がページが見つからなかったことを意味する．

statusText 文字列．HTTP ステータステキスト．ステータステキストはサーバによって異なる．

responseHeaders 文字列．レスポンスに含まれる HTTP ヘッダー．

responseText 文字列．レスポンスの本体．エラーページの本体はサーバによって異なる．

readyState 使われていない．

onreadystatechange callback

onreadystatechange のためのコールバック関数はリクエストが処理されている間繰り返し呼ばれる．一つの引数 *responseDetails* をとる．

```
function onreadystatechangeCallback(responseDetails);
```

responseDetails は5つのフィールドからなオブジェクトである．responseDetails.readyState はリクエストが現在どの段階にあるかを示す．

status 整数．レスポンスの HTTP エラーコード．responseDetails.readyState が 4 より小さい場合には 0 である．

statusText 文字列．HTTP ステータステキスト．responseDetails.readyState が 4 より小さい場合には空文字列である．

responseHeaders 文字列．レスポンスに含まれる HTTP ヘッダー．responseDetails.readyState が 4 より小さい場合にはから文字列である．

responseText 文字列．レスポンスの本体．responseDetails.readyState が 4 より小さい場合にはから文字列である．

readyState 整数．HTTP リクエストの段階を示す．

1. 読み込み中．リクエストが準備されている．
2. 読み込み終了．リクエストがサーバに送られようとしているが，まだ何も送られていない．
3. 更新中．リクエストが送信され，クライアントはサーバがデータを送信し終わるのを待っている．
4. 完了．リクエスが完了し，レスポンスデータはほかのフィールドに入っている．

7.5.4 例

次のコードは http://greaseblog.blogspot.com/から Atom フィードをとってきて結果とともに警告を表示する．

```
GM_xmlhttpRequest({
    method: 'GET',
    url: 'http://greaseblog.blogspot.com/atom.xml',
    headers: {
        'User-agent': 'Mozilla/4.0 (compatible) Greasemonkey',
```

```
        'Accept': 'application/atom+xml,application/xml,text/xml',
    },
    onload: function(responseDetails) {
        alert('Request for Atom feed returned ' + responseDetails.status +
              ' ' + responseDetails.statusText + '\n\n' +
              'Feed data:\n' + responseDetails.responseText);
    }
});
```

7.5.5 バグ

onreadystatechange コールバックが readyState < 4 のときうまく動かない.

7.5.6 注意

XMLHttpRequest オブジェクトと違って，GM_xmlhttpRequest はカレントドメインに制限されていない．どのページにも GET や POST できる.

7.5.7 履歴

GM_xmlhttpRequest は Greasemonkey 0.2.6 で導入された.

参考サイト

- XMLHttpRequest support in Mozilla
- XMLHttpRequest support in Internet Explorer
- XMLHttpRequest support in Safari
- **HTTP** status codes
- RFC 2616

第8章　改訂履歴

2005-05-09

- Added Case study: Zoom Textarea.
- Added Storing and retrieving persistent data.
- Added Adding items to the menubar.
- Added Integrating data from other sites.
- Added Compiling your user script into an extension.

2005-05-06

- Added Case study: Frownies.
- Changed void to function in Greasemonkey API Reference. (“You keep using that word. I do not think it means what you think it means.”)

2005-05-05

- Added What is Greasemonkey?. Thanks, Dennis.
- Added Case study: Dumb Quotes.
- Added downloadable Palm OS database for reading on mobile devices.

2005-05-04

- Added Parsing XML.
- Added Case study: Offsite Blank.

2005-05-03

- Compressed videos further.
- Added videos to downloadable HTML distribution.

2005-05-02

- Added GM_xmlhttpRequest.
- Added Case study: Ain't It Readable.
- Refactored Offsite Blank.
- Updated code examples to Greasemonkey 0.3 format.

2005-05-01

- Added Greasemonkey API Reference.
- Incorporated copious feedback from Simon Willison. Thanks, Simon.
- Incorporated copious feedback from Jeremy Dunck. Thanks, Jeremy.

2005-04-30

- Added Getting Started.
- Added Your First User Script.
- Added Debugging User Scripts.
- Added videos (currently online only).

2005-04-25

- Initial publication

第9章　本書について

本書は Emacs をつかい，DocBook XML で書かれました．HTML への変換には（カスタマイズした Norman Walsh の XSL スタイルシートと）SAXON を使いました．PDF への変換には HTMLDoc を使いました．プレインテキストへの変換には w3m を使いました．モバイル機器でも読めるように Plucker Distiller をつかって Palm OS™ データベースに変換しました．ANT スクリプトがこれらを完全に自動化してくれました．

補足のビデオは CamStudio と TMPGEnc によって作成しました．

Dean Edwards が書いた js-highlight により，自動的な Javascript コードのシンタックスハイライトが可能になりました．

技術的な文書作成のための DocBook をより詳しく知りたい場合は，XML ソースをダウンロード出来ます．これには，この本のいろいろなバージョンを書くために使ったバージョンの DocBook，XSL スタイルシート，スクリプトが含まれています．正規の本 DocBook: The Definitive Guide も読むべきです．DocBook メーリングリストに登録するのもよいでしょう．

この翻訳は SmartDoc と Carbon Emacs を使って，天野俊一により書かれました．

第10章　GNU 一般公衆利用許諾契約書

10.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

10.2 Terms and conditions for copying, distribution, and modification

10.2.1

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a"work based on the Program"means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term"modification".) Each licensee is addressed as"you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

10.2.2

You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

10.2.3

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

1. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

2. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

3. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License.

注意

If the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

10.2.4

You may copy and distribute the Program (or a work based on it, under Section 2 in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

1. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

2. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

3. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

10.2.5

You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under

this License will not have their licenses terminated so long as such parties remain in full compliance.

10.2.6

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

10.2.7

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

10.2.8

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is

implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

10.2.9

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10.2.10

The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and ”any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10.2.11

If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

10.2.12

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

10.2.13

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

10.3 How to apply these terms to your new programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.