Understanding HTTP

for advanced CGI programming

～原理的な考察のみが見通しを与える～

田中求之 mact@antares.ecn.fpu.ac.jp

0：Status of this document

この文章は、Macworld Expo/Tokyo 1998 のカンファレンスの講演原稿として用意されたものである。また、技術評論社の雑誌「Macintosh Developer Journal」に連載していた「私のMacはWebサーバだ」の記事の原稿の一部でもある（３０号に「1998年のCGIプログラミング」Part1として掲載）。さらに、『Macintosh インターネットサーバー構築術』のオンラインアップデータの一部として配付される予定であったが、諸般の事情によりそれは果たせなかった。

1：WebとはHTTPである

Webサーバは、CGIプログラムから処理結果として送り返されてきたデータを、そのままネットワークに送りだします（MacPerlでは少し異なりますが）。このため、CGIプログラム作成においては、処理の最後にサーバへデータを送り返す際に、データの先頭にHTTPヘッダと呼ばれる一定の書式で書かれたヘッダをつけなければならないことは、すでにみなさんもご存知だと思います。たとえば、"Hello, World"とブラウザに表示されるためには、以下のようなデータをCGIプログラムで作成し、サーバへ送り返すことになります。

* AppleScriptによるreply部分です。以下、説明はすべてAppleScriptで行います。

```
return "HTTP/1.0 200 OK" & crlf ¬
  & "Content-type: text/html" & crlf ¬
  & crlf ¬
  & "<TITLE>Message</TITLE>" & crlf ¬
  & "<H3>Hello, World</H3>"
```

CGIプログラムの作り方について書かれた本や雑誌の記事などでは、このHTTPヘッダとして決められた書式のデータを先頭につけるのを忘れないように、といったことが書かれています。たとえば、『Macintoshインターネットサーバ構築術』という少し古い本では、「このヘッダの内容はWWWのサービス(HTTP)の規格として内容が決められていますので、決まったものを使う必要があります」と書かれています。ですから、何も考えずにこのヘッダを機械的につけるようにしている方もいらっしゃるかもしれません。しかし、そのままでは、やがて壁にぶちあたります。

うまくいかなくなる実例を紹介しましょう。

clip2gifという、スクリプトで画像のフォーマットを変更したり、あるいは画像データそのものを生成した

りできるアプリケーションがあります。

clip2gif

これをつかうと、Macのスクリーンダンプ（スナップショット）をスクリプトで得ることができます。以下のスクリプトは、スナップショットをインターレースのGIFファイルとして作成するスクリプトです。

```
set myF to new file
tell application "clip2gif"
  save screen in myF as GIF with interlacing
end tell
```

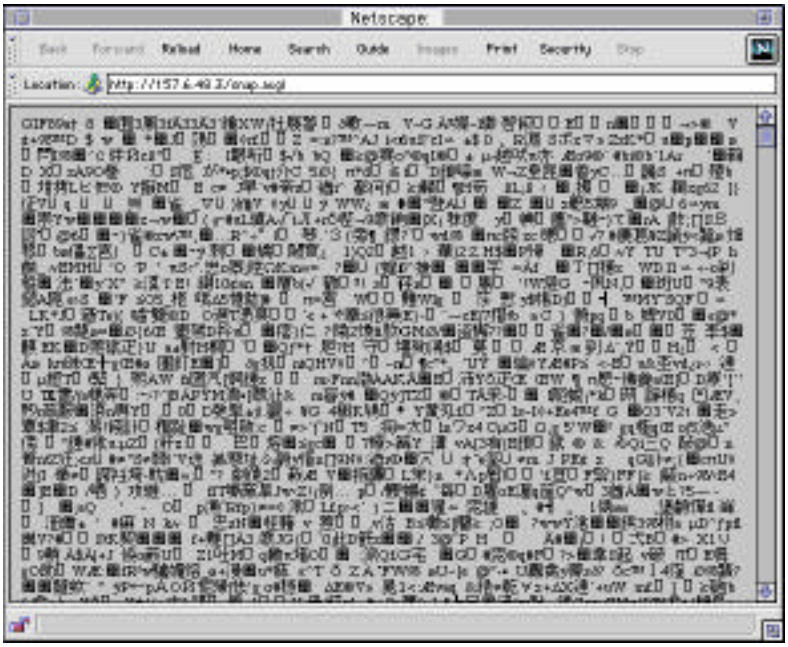
こういう便利なソフトがあるなら、ということでサーバのスクリーンショット（50%に縮小）を送り返すCGIプログラムを作ったとしましょう。画像データ自体は以下のスクリプトで作り出すことができます。

```
tell application "clip2gif"
  set mySnap to save screen in string as GIF scale 50 with interlacing
end tell
```

変数mySnapにGIFの画像データが入っていますので、これにHTTPヘッダをつけたものを送り返せば良いということになりますが、このとき、以下のように「何も考えずに」やってしまうと、思った通りの結果にはなりません。

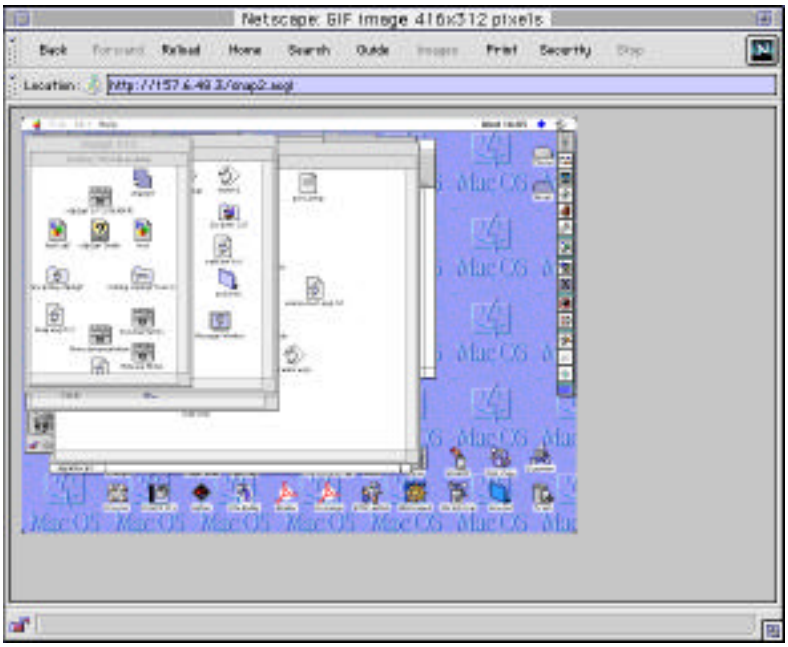
```
return "HTTP/1.0 200 OK" & crlf ¬
  & "Content-type: text/html" & crlf ¬
  & crlf ¬
  & mySnap
```

CGIプログラムとしてちゃんと走るのですが、ブラウザに表示されるのは、図のような訳の分からない文字の羅列になってしまいます。



ちゃんと図のように表示されるためには、以下のようなスクリプトにしなければなりません。

```
return "HTTP/1.0 200 OK" & crlf ¬
  & "Content-type: image/gif" & crlf ¬
  & crlf ¬
  & mySnap
```



違いはHTTPヘッダ部分の３行目のcontent-typeという部分にあります。ここを、text/htmlではなく、image/gifにしなければなりません。というのも、ネットワークを流れるデータには、ファイルタイプもクリエーターもありませんから、ブラウザは、サーバから送られてきたデータの種類を、HTTPヘッダのcontent-typeにかかれている情報によって判断しているからです。ブラウザは、データの中身をチェックすることなく、このHTTPヘッダの情報に基づいて表示のための処理をおこなうため、先程のようにtext/htmlになっていると、送られてきたのが実際は画像データであっても、テキストデータとして処理するわけです。つまり、先程の文字化けの羅列されたページとは、スナップショットの画像データがテキスト（文字データ）として表示されてしまったものだったのです。

このように、Webにおいては、データが正しくブラウザに処理されるためには、サーバからブラウザに送られるHTTPヘッダの情報が正しく記述されている必要があります。通常は、ファイルのタイプや拡張子をもとにデータの種類をサーバが判断したうえでHTTPヘッダをつけてデータを送りだしますので、ページや画像に関しては気にする必要がないのですが、次々と考案されている新しい種類のデータをサーバに登録する場合には、サーバにHTTPヘッダ用の情報を登録する必要があります。Macのファイルをダウンロード用に登録する際にはStuffItで圧縮したうえでbinhex形式に変換したものを登録するのが一般的ですが、プロバイダなどのサーバに登録したとき、ファイルとしてダウンロードできない（ページにずらずらとデータが表示されてしまう）ということが起こるのは、サーバが.hqxのファイル用のHTTPヘッダを作成できなかった（情報がサーバに登録されていない）からなのです。今ではBinhexのファイルは広く認められた用ですが、マイナーなPluginのデータなどでは、サーバに情報が登録されていないためサーバが正しいHTTPヘッダを生成できず、ホームページに登録することができないということがあります。「サーバがＸＸＸのデータには対応していない」というのは、データを送りだすというサーバの基本機能の次元の問題ではなく、サーバがそのデータにふさわしいHTTPヘッ

ダを作れないということなのです（shellアカウントがもらえるプロバイダでは、自分で設定することができます）。

このように、言ってしまえば、Webにおいては、HTTPヘッダこそが通信の要なのです。上の例ではCGIプログラムからブラウザに送り返す際のHTTPヘッダを取り上げましたが、ブラウザからサーバに対しても様々な情報が送られてきています。ブラウザとサーバとの間でHTTPヘッダによって様々な情報が交換されることによって、Webは動いているのです。ですから、CGIプログラムがHTTPヘッダを自分の責任で作成しなければならないということは、CGIプログラムでWebのコミュニケーションを隅々までコントロールできるということなのです。CGIプログラムとはHTTPというプロトコルをデータの入出力に用いるプログラムである、といっても過言ではありません。このことが持つ意味を理解していただくために、そして、それをCGIプログラムにおいて活かすために、以下、Webの通信の仕組み（HTTP）について、簡単に説明していく事にします。

２：HTTPの実際

WebはHTTPという規格に沿って通信を行うことで成り立っています。その見かけの派手さとは対照的に、基本的な原理は極めてシンプルで、ブラウザからサーバへ表示したいデータをリクエストすると、サーバがそれを送り返してくる、というものになっています。レイアウトを整えて見栄えのよいページに整えるのはブラウザの仕事であり。サーバの側は、ただ単にリクエストのあったデータを送り返すだけの役割に徹しています。

最初にHTTP/0.9が決められたときには、クライアントからサーバに対して

```
GET /works/index.html
```

という具合に欲しいデータの指示を送ると、そのデータがど〜んと送り返されてくるという、ただそれだけのものでした。その後Gopherなどの影響を受け、テキスト以外のデータも扱え、またブラウザの方でキャッシュを利用してアクセス速度を上げることが可能になったHTTP/1.0が決められ、現在はこれがベースになっています。そして、さらに機能の拡張をはかったHTTP/1.1という規格が現在議論されています。まもなく、正式な規格として定まるものと思います。

●実際のアクセスにおけるHTTPヘッダのやりとり

現在のブラウザとサーバとの間ではどのようなデータが実際にやり取りされているのかを、ブラウザでは見えないHTTPヘッダの部分も含めて見てみましょう。ブラウザにはNetscape、サーバにはWebSTARを用いて、以下のような内容のページ（test.html）にアクセスし、それをOTSessionWatcherで記録したものです。

```
<TITLE>Hello</TITLE>
<H2>Hello, World</H2>
```

OTSessionWatcher

まずブラウザに `http://mtlab.ecn.fpu.ac.jp/test.html` とURLを打ち込んでアクセスの指示を出します。するとブラウザからサーバへ対して以下のようなデータのリクエストが送られます（説明のために行番号を付してあります）。

```
01: GET /test.html HTTP/1.0
02: Connection: Keep-Alive
03: User-Agent: Mozilla/4.04 (Macintosh; I; PPC, Nav)
04: Pragma: no-cache
05: Host: mtlab.ecn.fpu.ac.jp
06: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
07: Accept-Language: en, ja
08: Accept-Charset: iso-8859-1,*,utf-8
09:
```

１行目がどのページのデータが欲しいのかを指示しています。２行目では、Keep-Aliveという方法でデータを受け取ることが可能であることをサーバに告げています。３行目がアクセスに使っているブラウザの種類、４行目はこのリクエストをキャッシュに残さないように指示しています。５行目ではアクセスしたホストの名前が書かれていますが、これは同じサーバが複数のホストを運用している場合（バーチャルホスト）に、どのホストに対してリクエストを行っているのかを示すためのものです。６行目にはブラウザが受け取ることができるデータの種類が、そして７行目には言語の優先順位、８行目は扱える文字コードが示されています。データの種類および文字コードの部分の*はワイルドカードで、なんでもOKということを示しています。そして最後に空行が入り、ここで情報が終わりであることを示しています。なお、改行コードにはCR+LFが使われます。

これを受け取ったサーバは、test.htmlのデータにHTTPヘッダをつけた、以下のようなデータをブラウザに送り返します。

```
01: HTTP/1.0 200 OK
```

```
02: MIME-Version: 1.0
03: Server: WebSTAR/2.1 ID/33602
04: Message-ID: <b106499b.1@mtlab.ecn.fpu.ac.jp>
05: Date: Wed, 11 Feb 1998 09:57:25 GMT
06: Last-Modified: Tue, 10 Feb 1998 09:09:31 GMT
07: Content-type: text/html
08: Content-length: 43
09:
10: <TITLE>Hello</TITLE>
11: <H2>Hello, World</H2>
```

９行目までがHTTPヘッダにあたります。

１行目はステータスコードと言って、リクエストの処理をHTTPのどのバージョンに則して行ったか、そして処理の結果がどうであったかを表わす数字が書かれています。上の場合ですと、HTTP/1.0の規格に則して処理を行い、無事にデータを送り返すことができるという意味です。２行目は送りだすデータの種類の情報をMIME1.0に基づいて知らせることを意味しています。３行目はサーバの機種の情報、４行目はメッセージIDです。５行目にはデータを処理した日時、そして６行目には送りだすページの最終更新日が記されています。上の場合ですと、test.htmlがグリニッジ標準時間（GMT）の２月１０日９時９分に更新されたものであるということです。なお、HTTPでは、日付や時間の情報はすべてグリニッジ標準時間によって交わすことになっています。６行目が先程も触れた送りだすデータの種類です。８行目がページのデータの量を示しています。４３バイトのデータを送るということです。そして空行が入ってHTTPヘッダ部分が終わり、１０行目と１１行目がtest.htmlに書かれていたデータになっています。このように、サーバからブラウザにデータが送り返される場合にはHTTPヘッダに

```
01: ステータスコード
02: MIMEのバージョン
03: サーバ機種
04: メッセージID
05:処理を行った日時
06: データの最終更新日
07: データの種類
08: データの量
```

といった情報がならびます。このうちブラウザで情報が正しく表示されるために最低限必要なものは、１，７です。つまり、ステータスコードとデータの種類の情報さえ正しく伝えれば、ブラウザできちんと表示されるようになっています。

●ページが見つからなかった場合のサーバからの返答

Webでは、データがどんどん更新されていくのが常ですから、サーバにリクエストしたデータが必ずしも見つかるわけではありません。リンクが古くなっていたり、あるいはURLが間違っていたりして、サーバがリクエストされたデータを見つけられなかった場合には、以下のようなデータがブラウザに送り返されます。

```
01: HTTP/1.0 404 File Not Found
02: MIME-Version: 1.0
03: Server: WebSTAR/2.1 ID/33602
04: Message-ID: <b1011ac4.26214@mtlab.ecn.fpu.ac.jp>
```

```
05: Content-type: text/html
06:
07: （以下エラーメッセージのページのデータ）
```

ステータスコードの数値が404になっていますよね。これがデータが見つからないということを意味しています。また、ページが新しい場所に移動したことを知らせる、以下のようなデータが送られてくることもあります。

```
01: HTTP/1.0 302 Found
02: MIME-Version: 1.0
03: Location: http://mtlab.ecn.fpu.ac.jp/ClipDecoder/
04:
```

今度は302という数値が示されていて、３行目にLocationという項目があります。302はページのURLが変更されたということを示し、Locationによって新しいURLが知らされます。データにあたるものは無く、HTTPヘッダのみが送られてきます。ブラウザはこれを受け取ると、Locationで指示されたURLに自動的にアクセスし直すようになっています。なお、このヘッダはリダイレクトヘッダとしてCGIプログラムでもよく使われますので、ご存知の方も少なくないかもしれません。たとえば、FORMのメッセージを処理する場合に、最後にメッセージを送ってくれたことへのお礼のページを表示したいならば、Formの処理を行った後に、このリダイレクトを指示するヘッダによってお礼のページの情報を送り返せばよいわけです。

●キャッシュを活かすためのアクセス

最近のブラウザは、一度アクセスしたページの情報をキャッシュファイルとして溜めておいて、２度目からはなるべくそのキャッシュを使うようにすることでアクセスの体感速度（ページが完全に表示されるまでの速度）を少しでも早くするように工夫されています。また、ファイヤーウォールに使われるProxyサーバもキャッシュを持つことによってトラフィックを少しでも下げるようになっています。このようなキャッシュを使ったアクセスにおいては、ブラウザは自分の持っているキャッシュが、サーバ上のデータと同じであるかどうかを確認しなければなりません。そして、もし手持ちのキャッシュの情報が古いものであったなら、新たにデータを送ってもらう必要があります。このキャッシュの有効確認がちゃんと行われないと、いつまでたっても昔のままのデータが表示され続けてしまうことになり、いくらトラフィックは減って表示速度は上がっても、何の意味もないことになってしまいます。

そこで、ブラウザは、過去にアクセスしキャッシュを持っているページにアクセスするときには、自分のキャッシュしているデータの最終更新日をサーバに告げ、「もしサーバ上のデータがこの日付以降に更新されていたならば（＝キャッシュよりも新しいものになっているのなら）、新しいデータを送って欲しい」というリクエストを行います。このリクエストのことを、通常のリクエストのGETと区別するために、CONDITIONAL_GETと呼びます。

具体的には、以下のようなリクエストをサーバに対して送ります。

```
01: GET /test.html HTTP/1.0
02: If-Modified-Since: Tuesday, 10-Feb-98 09:18:26 GMT; length=43
03: Connection: Keep-Alive
04: User-Agent: Mozilla/4.04 (Macintosh; I; PPC, Nav)
05: Pragma: no-cache
06: Host: mtlab.ecn.fpu.ac.jp
07: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
```

```
08: Accept-Language: en, ja
09: Accept-Charset: iso-8859-1,*,utf-8
10:
```

先程のものと比べてみると、２行目にIf-Modified-Sinceという行が増えているのがお分かりになると思います。これがブラウザからサーバへキャッシュの情報を送っている部分です。キャッシュしているデータの最終更新日と、データの量が示されています。これを受け取ったサーバは、サーバ上のデータと比較を行い、もしサーバ上のデータが新しくなっていた場合には、ページのデータを送り返します（先程のステータスコード200の場合と全く同じ）。一方、もしブラウザのキャッシュしているものから更新がされていない場合には、以下のような変更がないことを告げるHTTPヘッダを送り返します。

```
01: HTTP/1.0 304 Not Modified
02: MIME-Version: 1.0
03: Server: WebSTAR/2.1 ID/33602
04: Message-ID: <b106499b.2@mtlab.ecn.fpu.ac.jp>
05: Last-Modified: Tue, 10 Feb 1998 09:18:26 GMT
06: Content-type: text/html
07: Content-length: 43
08:
```

ステータスコード304がデータの更新が行われていないことを告げています。これを受け取ったブラウザは、手持ちのキャッシュを使ってページを表示するわけです。データの転送は行われませんので、その分、早く表示されることになります。

なお、このCONDITIONAL_GETにおいて、ブラウザがIf-Modified-Sinceヘッダでサーバに告げるのはキャッシュしているデータの最終更新日であって、キャッシュの最終更新日ではありません。ブラウザは、最初にそのページにアクセスした時に、ページのデータなどをキャッシュファイルに保存するとともに、サーバからのHTTPヘッダの中のLast-Modifiedヘッダに書かれているデータの最終更新日の情報も記録しているのです。つまり、サーバがデータの最終更新日を知らせてくれるからこそ、キャッシュを有効に使うことができるわけです。このように、CONDITIONAL_GETによる効率的なアクセスを実現するには、ブラウザの方がキャッシュ管理の仕組みを持っているだけではなく、サーバがHTTPヘッダの中でデータの最終更新日の情報をブラウザに教えるようになっていなければなりません。ですから、スクリプト１のような最低限の情報しか返さないヘッダを使うCGIプログラムのデータは、キャッシュが使われることはないわけです（これは良いことと悪いことがあります）。

●ページ情報のみを得るアクセス

これまで紹介してきたのは、いずれもブラウザが最終的にページを表示するために行うアクセスでしたが、これらとは別に、HTTPヘッダに書かれているページの情報だけを得るためのアクセスを行えるようになっています。具体的には以下のように、リクエストの先頭がHEADになったものです。

```
HEAD /test.html HTTP/1.0
```

これを受け取ったサーバは、以下のように、このページのデータにつけるべきHTTPヘッダ部分のみを送り返します。

```
01: HTTP/1.0 200 OK
```

```
02: MIME-Version: 1.0
03: Server: WebSTAR/2.1 ID/33602
04: Message-ID: <b107b2db.12@mtlab.ecn.fpu.ac.jp>
05: Date: Wed, 11 Feb 1998 10:57:25 GMT
06: Last-Modified: Tue, 10 Feb 1998 10:14:35 GMT
07: Content-type: text/html
08: Content-length: 48
09:
```

HTTPヘッダのみでページのデータは含まれません。しかし、ヘッダにはページの最終更新日やデータの量の情報が書かれていますので、これを受け取ったクライアント側では、手持ちのキャッシュやアクセスの記録と比較することで、ページが更新されているかどうかを確かめることができます。このように、ページが更新されているのかどうかのみを知りたい（とりあえずデータはいらない）場合に、このHEADによるリクエストは使われます。ブックマークに記録したページが更新されていないかどうかのチェックに使われたり、あるいはサーチエンジンのロボットがサイトの状況を調べるのに使います。上の場合、text.htmlの更新時間が10:14になっており、データ量も48バイトに増えていますから、先のアクセス以後にページが更新されたということになります。

●保護領域（Realm）へのアクセス

Webサーバにおいて、ユーザー名とパスワードを入力しなければページにはアクセスできない仕組みを設定することができます。保護領域（Realm）と呼ばれるものです。この保護領域の仕組みにおいても、HTTPヘッダが重要な意味を持ちます。

具体的に、あるRealmで保護されたページ（pi_admin.html）にアクセスするプロセスを追ってみます。

まず最初に、ブラウザからサーバへ通常のページのリクエストが送られます。

```
01: GET /pi_admin.html HTTP/1.0
02: Connection: Keep-Alive
03: User-Agent: Mozilla/4.04 (Macintosh; I; PPC, Nav)
04: Host: mtlab.ecn.fpu.ac.jp
05: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
06: Accept-Language: en, ja
07: Accept-Charset: iso-8859-1,*,utf-8
08:
```

するとサーバからブラウザに対しては、以下のような、アクセスは認められないという趣旨のメッセージが返ります。この際に、WWW-Authenticateという項目で、リクエストされたページがどの保護領域に属しているのかも伝えられます。

```
01: HTTP/1.0 401 Unauthorized
02: Server: WebSTAR/2.1 ID/33602
03: MIME-Version: 1.0
04: WWW-Authenticate: Basic realm="PI_ADMIN"
05:
06: <title>Not Authorized!</title><h1>Not Authorized!</h1>
07: Sorry, you aren't authorized to access this information.
08:
```

しかし、ブラウザはこの時点では送られてきたアクセス拒否のメッセージを表示しません。そのかわり、以下のようなダイアログを表示して、ユーザーにユーザー名とパスワードの入力を求めます。

そして、ここで入力されたユーザー名とパスワードの情報を追加したあらたなリクエストをサーバに向けて送ります。

```
01: GET /pi_admin.html HTTP/1.0
02: Connection: Keep-Alive
03: User-Agent: Mozilla/4.04 (Macintosh; I; PPC, Nav)
04: Host: mtlab.ecn.fpu.ac.jp
05: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
06: Accept-Language: en, ja
07: Accept-Charset: iso-8859-1,*,utf-8
08: Authorization: Basic gruCsYLcgsU6itSUsoKvgraC4YLIgqKC5g==
09:
```

８行目にAuthorizationという新たな項目が追加されているのに注意してください。この部分でユーザーが入力したユーザー名とパスワードの情報がサーバに対して渡されているのです。ユーザー名とパスワードは、そのままの形ではなく、:を挟んで連結された後にBase64でエンコードされたものになっています（暗号化されているのではありません）。

このユーザーの情報付きのリクエストを受け取ったサーバでは、それをサーバに登録されているユーザーのデータベースと照合し、もしユーザー名が登録されており、パスワードが正しかった場合には、通常のアクセスと同様にページのデータを送り返します。

もし送られてきたパスワードが不正だった場合は、再び先程と同じ、アクセスを認められないというメッセージを送り返します。これを受け取ったブラウザは、今度は以下のようなダイアログを表示して、ユーザーにアクセスが認められなかったことを知らせます。

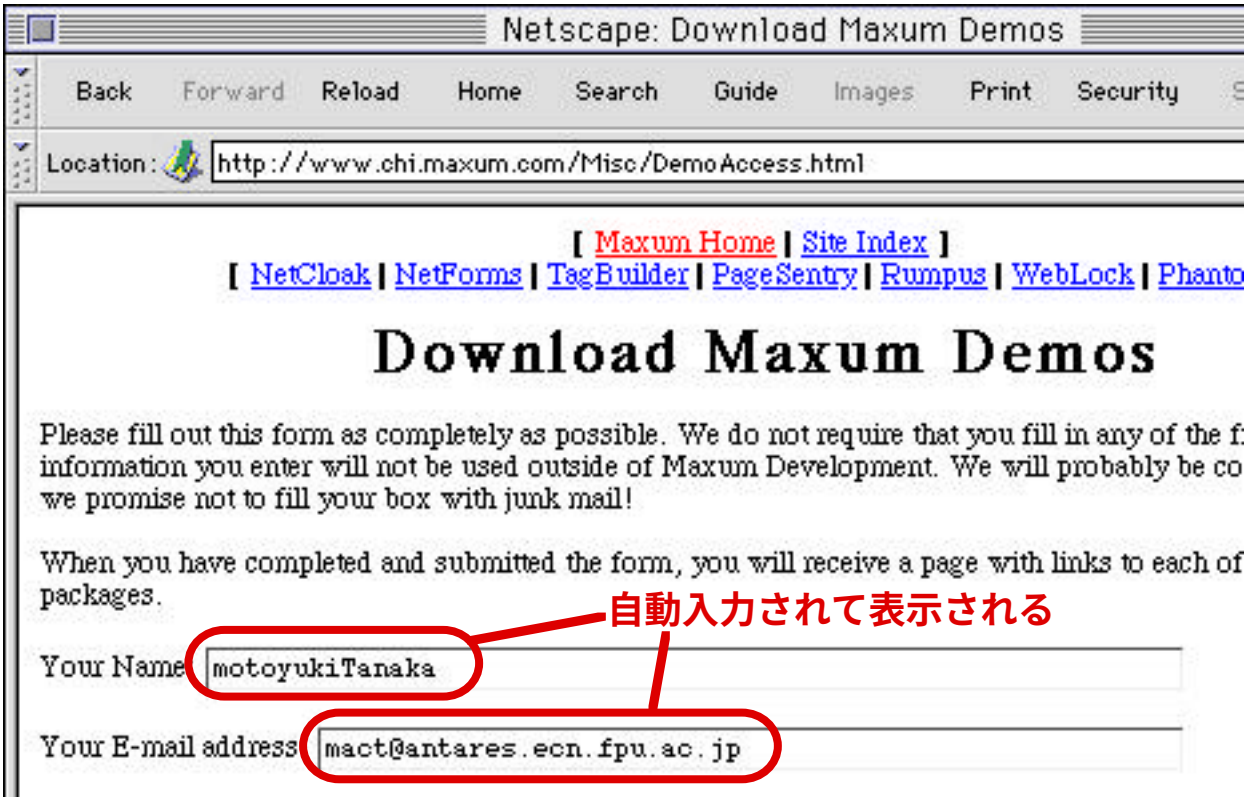
ユーザーがOKをクリックすると、再びユーザー名とパスワードを入力するダイアログを表示します。Cancelをクリックした場合には、以下のように、サーバから送られてきたアクセス拒否のメッセージを表示します。

このように、保護領域にアクセスを行う際には、ユーザーの目にはページにアクセスする前にユーザー名とパスワードを入力しているように見えますが、実際は、一度、サーバとブラウザとの間で交信が行われ、それをうけてユーザー名とパスワードを入力するように求め、再度アクセスを行うという仕組みになっているのです。ブラウザは、サーバにアクセスするまでは、自分がアクセスしたいページのデータが保護領域によってプロテクトされたものかどうかを知ることはできません。ですから、とりあえずアクセスして、拒否されたら、ユーザー情報を入力してもらってから、もう一度アクセスを行うという手順を踏むわけです。

●Cookieヘッダによる情報の記録

Webでは、１つのデータごとに通信が行われることになっています。パソコン通信のようなログインあるいはセッションという概念はありません。ですから、基本的には、過去のアクセスの情報を将来に利用すると行ったことはかなり難しくなっています。これを補うためにNetscape社によって考え出されたのがCookieヘッダを使った情報の保持という仕組みです。これはサーバからHTTPヘッダの中で送られた情報をブラウザが記録しておき、次からのアクセスの場合に、その情報をサーバに知らせるというものになっています。

まずCookieを利用している実例をお見せしましょう。以下のページは各種のサーバ用ツールの開発販売でおなじみのMaxum社のデモ版のダウンロード用ページです。ダウンロード前にフォームに名前やメールアドレスなどの情報を入力する必要があるという、よく見られる形式のページなのですが、このページは、１度利用すると、２度目からはアクセスした時点で名前とメールアドレスが自動的に入力された状態で表示されるようになっています。



名前とメールアドレスの情報は、サーバに記録されていたのではなく、ブラウザがページにアクセスする際に、サーバに対して「以前このページで名前には motoyukiTanakaを、メールアドレスには mact@antares.ecn.fpuを使ったよ」ということを知らせるようになっており、それをもとにサーバががあらかじめデータを記入した形で表示するようになっているのです（Maxum社の製品の一つであるNetCloakというCGIプログラムで処理を行っています）。

具体的にこのページに２度目からアクセスするときのヘッダを見てみましょう。

```
01: GET /Misc/DemoAccess.html HTTP/1.0
02: Connection: Keep-Alive
03: User-Agent: Mozilla/4.04 (Macintosh; I; PPC, Nav)
04: Host: www.chi.maxum.com
05: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
06: Accept-Language: en, ja
07: Accept-Charset: iso-8859-1,*,utf-8
08: Cookie: MaxDLMail=mact@antares.ecn.fpu.ac.jp; MaxDLName=motoyukiTanaka
09:
```

８行目にCookieという項目があり、ここでメールアドレスと名前の情報が送られていることがわかります。この情報をサーバ側で処理していたわけです。

もちろん、フォームに入力したからといって、いつもこのように情報が記録されたり使われたりするのではありません。ブラウザは、サーバから記録するように指示された情報だけを記録し、以後のアクセスで使うようになっています。先のMaxum社のダウンロード用ページの場合は、フォームの最後のSubmitをクリックし、その結果表示されるダウンロード用リンクの並んだページのヘッダにおいて、ブラウザへの指示がなされるようになっています。Submitをクリックしたときの交信の様子を見てみましょう。まずブラウザからサーバに対してフォームに入力した情報が、POSTという方法で送られます。これはHTTPヘッダの後にURLエンコードされ

た形で情報をつけておくるというものです。

```
 01: POST /Misc/checkdemopage.fdml HTTP/1.0
 02: Referer: http://www.chi.maxum.com/Misc/DemoAccess.html
 03: Connection: Keep-Alive
 04: User-Agent: Mozilla/4.04 (Macintosh; I; PPC, Nav)
 05: Host: www.chi.maxum.com
 06: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
 07: Accept-Language: en, ja
 08: Accept-Charset: iso-8859-1,*,utf-8
 09: Cookie: MaxDLMail=mact@antares.ecn.fpu.ac.jp; MaxDLName=motoyukiTanaka;
 NetCloakID=NCBTZXBJOXZN
 10: Content-type: application/x-www-form-urlencoded
 11: Content-length: 308
 12:
 13: name=motoyukiTanaka&email=mact@antares.ecn.fpu.ac.jp&country=Jap
 14: an&maillist=yes&TryOrUpgrade=UPGRADE&advertising=Mailing+list&nu
 15: mofservers=One&platforms=MacOS&NetCloak=YES&NetCloakDev=YES&NetC
 16: loakD=2.5.2b2&NetFormsD=2.5.2b1&PhantomDev=YES&PhantomD=2.1f2&Pa
 17: geSentryD=2.5b2&WinPhantomD=2.1f2&address=157.6.48.3
 18:
```

１２行目から１７行目までがフォームに入力されたデータです（実際のデータには改行は入っていません）。１０行目のcontent-typeがapplication/x-www-form-urlencodedになっており、URLエンコードされたデータを送るよということをサーバに告げています。

さて、これを受け取ったMaxum社のサーバは、データを処理した後に、以下のようなデータを送り返してきます。

```
 01: HTTP/1.0 200 OK
 02: Server: WebSTAR NetCloak
 03: Date: Fri, 13 FEB 1998 09:03:15 GMT
 04: MIME-Version: 1.0
 05: Content-type: text/html
 06: Content-length: 3582
 07: Set-Cookie: MaxDLMail=mact@antares.ecn.fpu.ac.jp; expires=Fri, 01-JAN-1999
 05:00:00 GMT; path=/
 08: Set-Cookie: MaxDLName=motoyukiTanaka; expires=Fri, 01-JAN-1999 05:00:00 GMT;
 path=/
 09: Set-Cookie: MaxDLRpt=Repeat; path=/
 10:
 11: <HTML>
 12: <HEAD>
 13: <TITLE>Download Maxum Demos</TITLE>
 14: </HEAD>
 15: <BODY BGCOLOR=#FFFFFF>
 ---（以下、ページのデータは省略）---
```

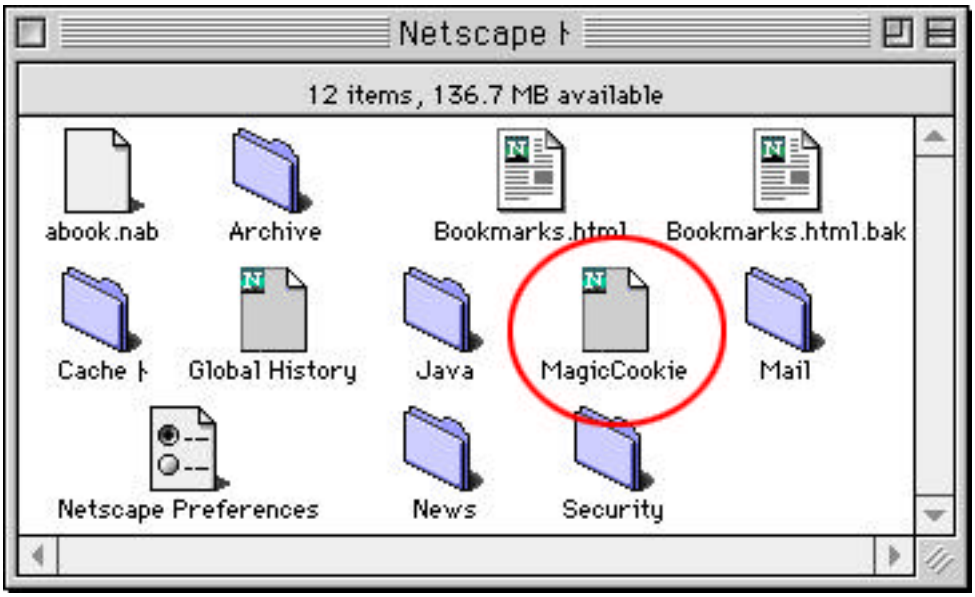
１０行目まではHTTPヘッダにあたりますが、７行目から９行目にかけて、Set-Cookieという項目が並んでいます。ここがブラウザに対して情報の記録を指示している部分です。７行目では、MaxDLMailという変数と

してmact@antares.ecn.fpu.ac.jpというデータをセットして記録すること、その有効期限は1999年の１月１日の午前５時で、このサーバ内のすべてのページにおいてこの情報を送るように、との指示がなされています。以下同様にMaxDLNameとMaxDLRptという変数の記録が指示されています。

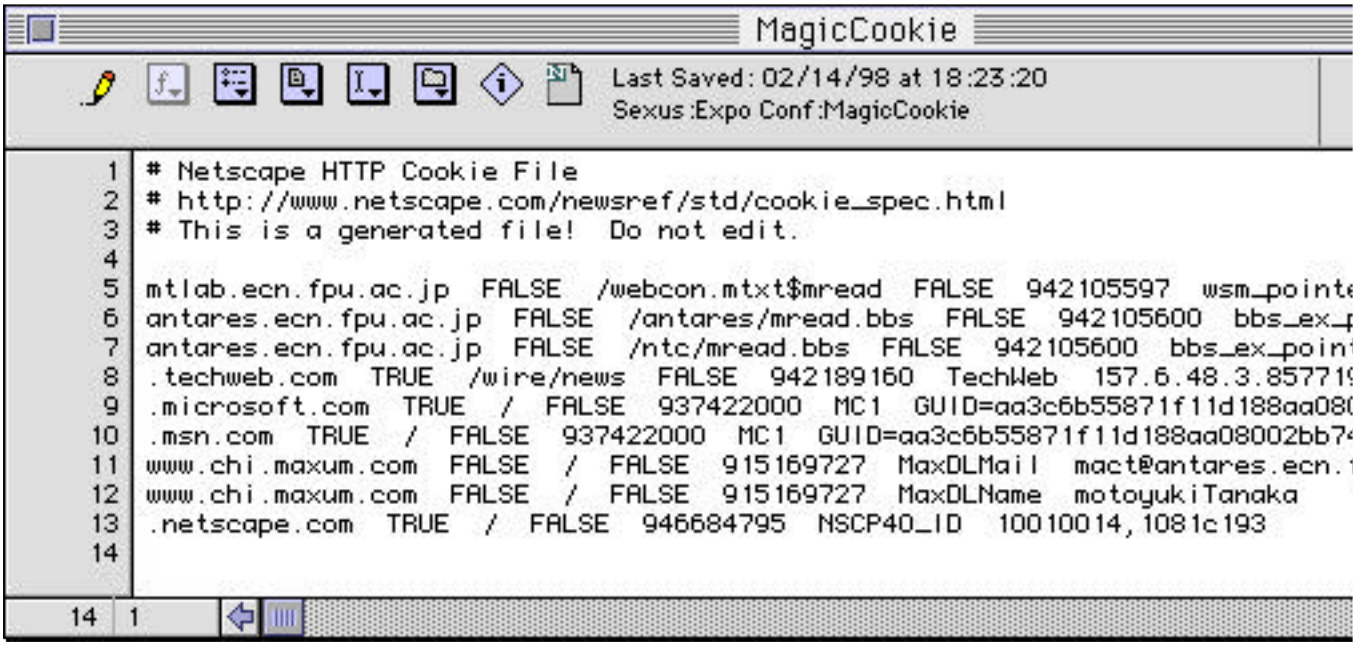
Set-Cookieによってデータの記録を指示する場合の書式はNetscape社のサーバにあるドキュメント、
Persistent Client State / HTTP Cookie
http://www.netscape.com/newsref/std/cookie_spec.html
を読んでもらえば分かりますが
Set-cookie: 変数名=データ; expires=有効期限; path=有効範囲

です（これ以外にdomainとsecureという項目がありますが、普通は使われないので省略します）。有効範囲は、指示したデータをどのページのアクセスに対して送り返すようにするのかを指示します。ディレクトリーの指定を行うと、そのディレクトリーより下位のディレクトリーにも適用されますので、サーバのルート/を指示すると、ブラウザはそのサーバ内のすべてのページのアクセスにおいてCookieのデータを送るようになります。

このようにHTTPヘッダ内にSet-Cookieという項目があると、そこで指示された情報をブラウザは記録し、指示に応じてアクセスの際のCookieヘッダに付して送るようになるのです。ただし、有効期限が指定されていないものは現在のアクセスのみに有効とされるので、ファイルには記録されません（上記の場合のMaxDLRptがそれにあたる）。記録しておくように指示されたデータは、初期設定フォルダーの中のNetscapeフォルダーの中にあるMagicCookieというファイルに記録されます。



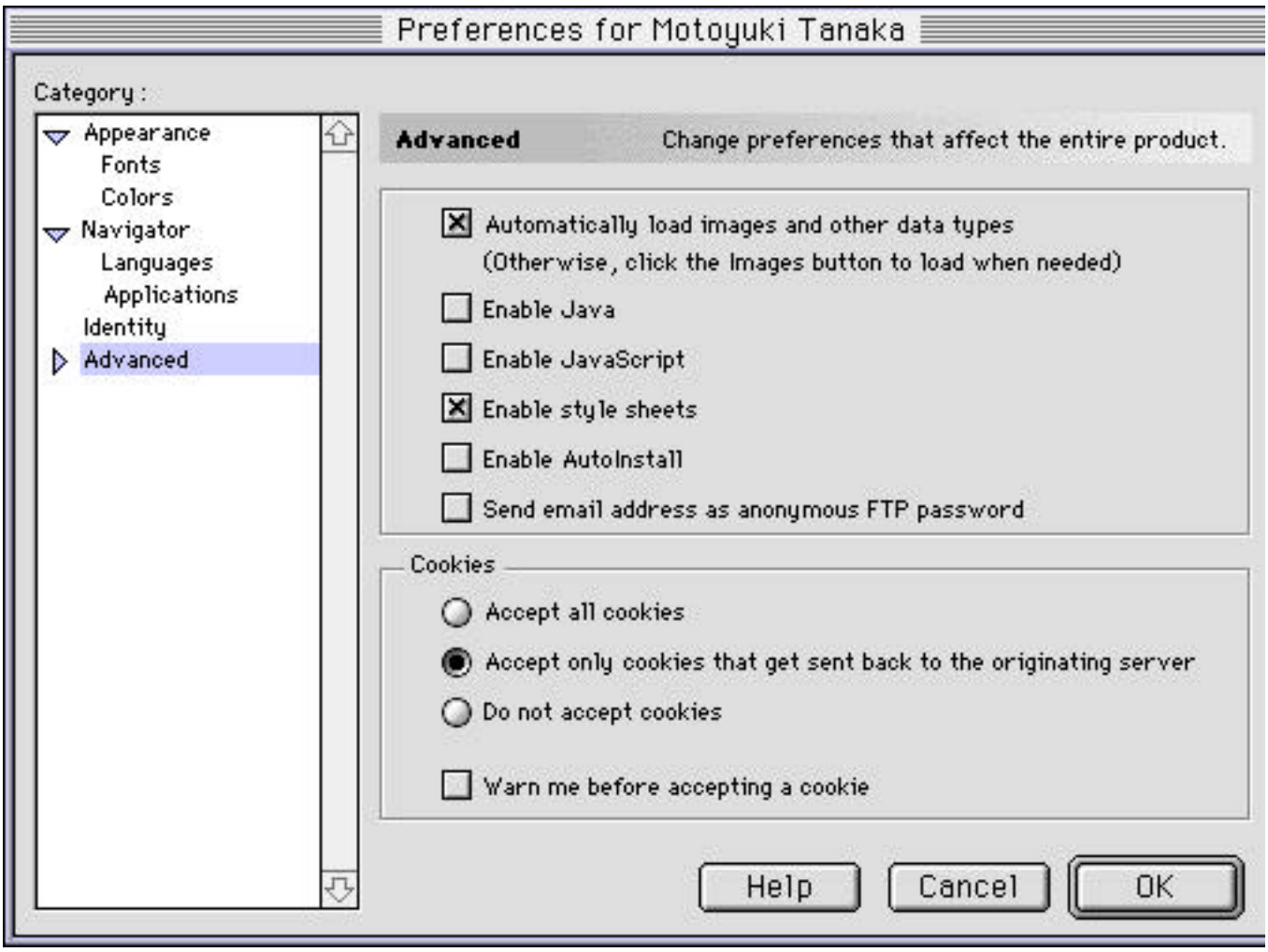
ファイルのコピーを作ってエディタで開いてみると、以下のように、情報が記録されているのがわかります。



このように、サーバからのHTTPヘッダでの指示に基づいて情報を記録し、それを以後のアクセスにおいて使うことができる仕組みが、Cookieヘッダと言われるものです。整理しておくと、

1：サーバがHTTPヘッダ内のSet-Cookieによって、ブラウザにデータの保持を指示する
2：ブラウザは、以後のそのサーバへのアクセス（リクエスト）において、有効期限や有効範囲にもとづいて、指示されたデータをCookieヘッダとして送り返す
3：セッション終了後（ブラウザ終了後）も有効なデータは、MagicCookieファイルに記録される

というものです。なお、このCookieがセキュリティホールとして問題になったこともあり、なにやら危ないものというイメージがつきまとっているようですが、Cookieという仕組み自体は決して危険なものではありません。ただ、Cookieの利用法が間違っていると、セキュリティホールになる危険があることは確かです。たとえば、オフィスでみんなが使うブラウザを使って、あなたが通信販売で他人にはあまり大きな声では言えないようなものを、そういうものを販売しているので有名なサイトから購入したとしましょう。そのときの取り引きにおいて使われたCookieの情報がファイルに残ってしまった場合、後から来た人がエディタでMagicCookieを開いてしまうと…　「誰や？　こんなところから買い物をしたのは？」。こういう事態を避けるために、ブラウザの設定でCookieを受け付けないという設定にすることもできますし、サーバからCookieが送られて来るたびに、受け付けるかどうかの確認を出すようにすることもできます。



もともとはNetscapeが独自の機能拡張として追加した機能ですが、今では対抗馬のMS Explorerも対応しています（上のMagicCookieの中に.microsoft.comと.msn.comがあるのからも分かりますよね）。ショッピングバスケットで買い物カゴに入れた商品の情報を保持しておくのに使われたり、掲示板の名前を覚えておくといったことに使われています。

ちなみに、この仕組みがCookieという名前になったのは、特に理由はないようです。Danny GoodmanがJavaScriptでのCookieの利用法を説明した文書

COOKIE RECIPES
http://developer.netscape.com/news/viewsource/archive/goodman_cookies.html

のなかで、夜中に開発を行っていたときにチョコチップクッキーを食べていた人間がいたんだろうということを言ってますが、たぶん、その程度のことではないかと思われます。なお、現時点では、あくまでもNetscapeが使い始めた仕掛けという扱いですが、Internet Draftとして、このCookieヘッダによる仕組みを"HTTP State Management Mechanism"として標準化する提案がなされています。

Cookie I-D Drafts
http://portal.research.bell-labs.com/~dmk/cookie-ver.html

以上、Webにおいてどのようにhttpヘッダによって情報の交換が行われているのかを簡単に紹介してみました。HTTPの細かい仕様などは、以下のRFCをお読みください。

```
1945 Hypertext Transfer Protocol -- HTTP/1.0. T. Berners-Lee, R.
     Fielding & H. Frystyk. May 1996. (Format: TXT=137582 bytes) (Status:
     INFORMATIONAL)
```

また、現在、多くのサーバがHTTP/1.1サポートをうたうようになってきていますが、このHTTP/1.1でどのように機能が拡張されるのかなどは、

```
2068 Hypertext Transfer Protocol -- HTTP/1.1. R. Fielding, J. Gettys,
     J. Mogul, H. Frystyk, T. Berners-Lee. January 1997. (Format:
     TXT=378114 bytes) (Status: PROPOSED STANDARD)
```

を参照してください。１回の通信で複数のデータをまとめて送るKeep-Alive接続、指定した範囲のデータのみを送るByte Range、ユーザーの希望言語を指定できるようにするlanguageタグ、ソフト的にマルチホーミングを可能にするHostタグ、あるいはキャッシュの細かなコントロールなど、数多くの機能拡張がはかられていますが（すでに事実上使われている機能もあります）、これらはすべてHTTPヘッダによってコントロールが行われます。

３：CGIプログラミングで活用する

続いて、CGIプログラミングにおいてHTTPヘッダを活用する方法の説明に入ります。以下の説明においては、実例はAppleScriptによって示しますが、細かいコードの書き方よりも、どのような処理を行うことになるのかと言った一般的な解説を中心にします。

●保護領域（Realm）をCGIプログラムでコントロールする

まず最初に、CGIプログラムによって保護領域をコントロールする方法です。WebSTARやQuid Pro Quoなどの既存のサーバは、Realmのユーザーの管理をサーバアプリケーションが行っていますが、たとえばデータベースに登録されているユーザーのみにアクセスを認めるといった場合には、データベースの情報をサーバへ登録する作業が必要になります。しかし、CGIプログラムが管理を行うことで、直接データベースに登録されているかどうかをチェックすることができます。

ユーザーのチェックの部分をどのように組むかという点はみなさんのそれぞれの環境や使用状況によって異なることになりますので、ここではパスワードチェックの処理のキーポイントだけ示します。CGIプログラム側の処理は、実は、非常に簡単です。

Realm のユーザー名とパスワードは、もしそれが入力されていた場合には、WebサーバからのAppleEventのパラメータの中に含まれています。ユーザー名はclass 'user'のパラメータに、そしてパスワードはclass 'pass'のパラメータとして送られてきます。なお、HTTPでブラウザからサーバへこれらの情報が送られる際には、先ほど述べたようにBase64でエンコードされた形で送られるのですが、AppleEventのパラメータにする時点でサーバ側でデコードされていますので、CGIプログラム側ではデコードする必要はありません。ユーザーがダイアログに入力したままのデータが、'user'と'pass'パラメータで送られてきます。

ですから、CGIプログラム側の処理としては、ユーザー名／パスワードが送られてこない（空文字列になっている）か、あるいは登録されているデータと一致しない場合には、ステータスコード404で、WWW-Authenticateの部分にRealm名を書いたHTTPヘッダを送り返すようにする、という処理を行えばよいことになります。

以下は、PreProcessorとして利用するRealm管理CGIプログラムのサンプルです。specialというディレクトリー（フォルダー）に入っている情報にアクセスしようとすると、ユーザー名とパスワードの入力を求め、ユーザーチェックを行います（ユーザーチェック部分は省略）。

```
property crlf : (ASCII character 13) & (ASCII character 10)

on «event WWWΩsdoc» path_args ¬
  given «class user»:username, «class pass»:password, «class scnm»:script_name

  if ("/special/" is not in script_name) then
    --- このCGIプログラムでは管理しない領域の場合
    return ""
  end if

  if username = "" or password = "" then

    --- ユーザー名とパスワードが送られてきてない

    return "HTTP/1.0 401 Unauthorized" & crlf ¬
      & "WWW-Authenticate: Basic realm=\"Specail_Info\"" & crlf ¬
      & crlf ¬
```

```
      & "<title>Not Authorized!</title><h1>Not Authorized!</h1>"

  else

    if userChaeck(username, password) then
      --- username と password が登録済みのものならばアクセスを認める

      return ""

    else

      --- 登録されていない場合には接続拒否

      return "HTTP/1.0 401 Unauthorized" & crlf ¬
        & "WWW-Authenticate: Basic realm=\"Specail_Info\"" & crlf ¬
        & crlf ¬
        & "<title>Not Authorized!</title><h1>Not Authorized!</h1>"

    end if

  end if
end «event WWWΩsdoc»


on userChaeck(username, password)

  --- データベースなどでユーザーチェックを行う
  --- 登録ユーザーなら true を返し
  --- 登録されていないユーザーの場合には false を返す

  return true / false

end userChaeck
```

script_nameパラメータ（'scnm'）によってアクセスのURLをチェックし、specialディレクトリー内へのアクセスの場合には、ユーザー名とパスワードのチェックを行うようにしてあります。

PreProcessorをサポートしているサーバであれば、上記のスクリプトで独自のユーザー管理が行えます。たとえば、Web Sharing（Web共有）は、システムのファイル共有によってアクセス制限を行うようになっていますが、1.5以降はPre-Processをサポートしていますので、このCGIプログラムによってユーザー管理を行うことが出来るようになります。

なお、念のため補足しておくと、PreProcessorのCGIプログラムにおいては、サーバに処理を任せる場合には、空文字列をreplyし、独自に処理を行う場合にのみデータを送り返すというスクリプトにします。

●ブラウザにキャッシュを作らせない

HTTPの説明の中で述べたように、最近のブラウザは一度アクセスしたページのデータをキャッシュに記録しておき、２度目からはそのキャッシュをなるべく使うことでアクセス速度を上げるようになっています。このキャッシュをCGIプログラムの側でコントロールすることが可能です。

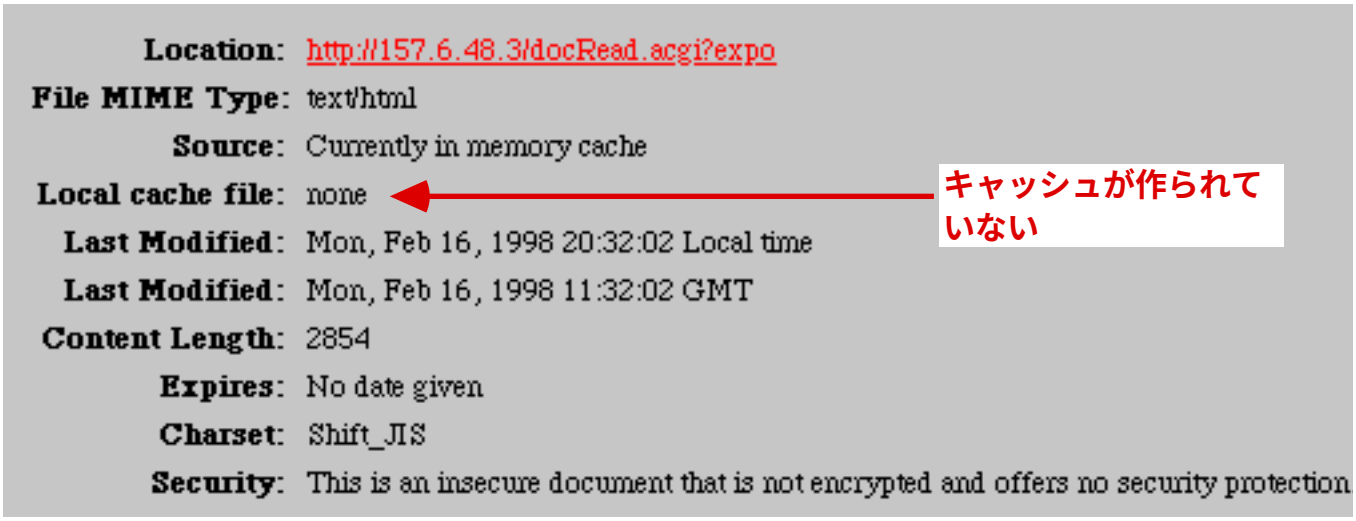
まず最初に、キャッシュを作らせない方法です。この場合は、「ページを表示する場合」には、必ずサーバへデータを取りに来るようになります。わざわざ「ページを表示する場合」と書いたのは、キャッシュを全く作らせない場合には、ブラウザのForward/Backのボタンによってページを再表示させた場合でも、かならずサーバへのアクセスが行われるようになるからです。また、この場合のアクセスは、CONDITIONAL_GETでは

なく、通常のGETが使われます。ですから、刻一刻と内容が変化していくようなページ（ニュースの速報やチャットのようなもの）の場合には、キャッシュを作らせないことで、ユーザーが必ずその時点、時点の最新の内容を見ることができるようになるわけです。ただし、毎回サーバへアクセスするということは、トラフィックを増やすことになりますのでその点は注意してください。

方法は簡単で、CGIプログラムから返すHTTPヘッダの中に
`Pragma: no-cache`
という１行を追加するだけで済みます。つまり、以下のようなヘッダを返します

```
HTTP/1.0 200 OK
Pragma: no-cache
Content-type: text/html

--- 以下、ページのデータ ---
```

このヘッダを返したページをNetScapeのPage Infoでチェックしてみると、以下のようにLocal Cacheが作られていないことがわかります。



●ブラウザのキャッシュの有効期限を指定する

次に、キャッシュの有効期限を指定する方法です。

この場合は、キャッシュの有効期限が来るまでは、ブラウザは自分が持っているキャッシュを使って表示を行います。そして、有効期限が切れた後は、その都度、サーバに対してCONDITONAL_GETによってキャッシュの有効性を確かめるという動作をします（Netscape 4.04の場合。バージョンが古いブラウザでは動作が異なることがあります）。

先程のキャッシュを行わせない場合に較べると、一定の間はキャッシュが使われ、それ以後もCONDITIONAL_GETによる確認が行われますので、トラフィックは少ないことになります。掲示板のトップページあるいはディレクトリーのIndexのように、更新がさほど頻繁ではなく（せいぜい１時間あるいは１日の単位で更新が行われる）、ユーザーがアクセスした際には何度か表示する（Forward/Backによる表示を繰り返す）可能性が高いページの場合には、有効期限を指定するという方法が有効だと思います。

キャッシュの有効期限を指定するには、以下のようにExpiresというヘッダで有効期限を指定します。以下は、２月１８日午前４時１９分まで（ただしGMTで）キャッシュを有効にするというものです。

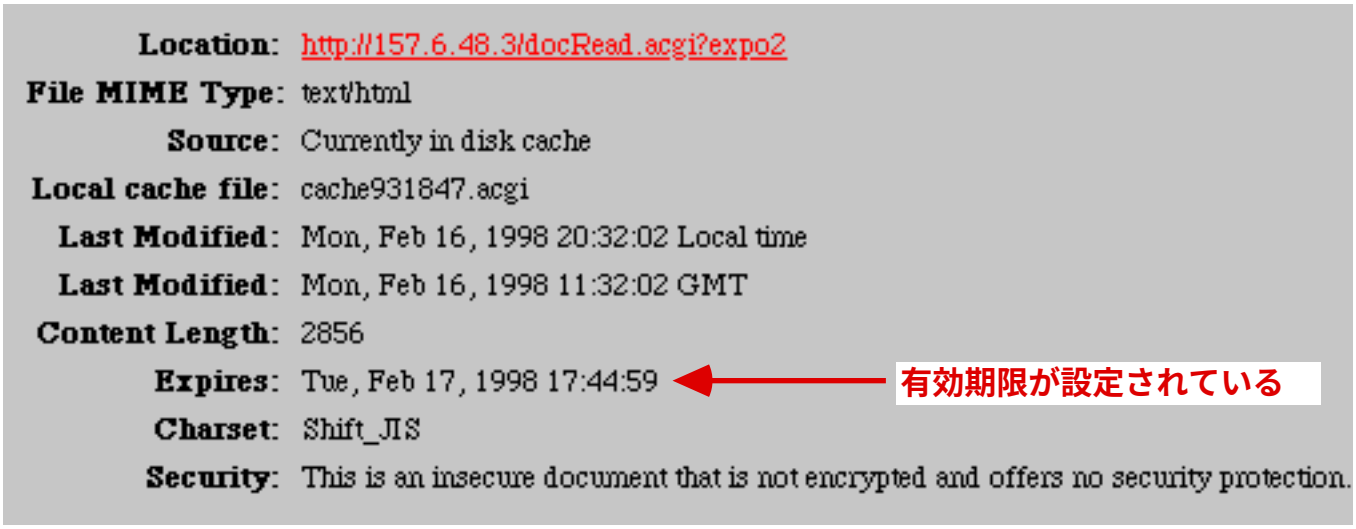
```
HTTP/1.0 200 OK
Content-type: text/html
Expires: Wed, 18 Feb 1998 04:19:49 GMT

--- 以下、ページのデータ
```

気をつけなければいけないことは、日時は必ずGMTによって指定することと、決められた書式で日時を記述する必要があるということです。サーバからブラウザに返されるHTTPヘッダにおいて日時を指定（記述）する場合には、以下の書式を用います。

```
Wky, DD MMM YYYY, HH:MM:SS GMT
```

CGIプログラムを作成する際には、日付データをこの書式に変換するのが面倒かもしれません。

ブラウザのPage Infoをチェックすると、以下のようにExpiresの欄に有効期限が表示されるはずです。



なお、この有効期限を指定する方法を使う場合に忘れてはならない重要な点が一つあります。先程も延べたように、Expiresによってキャッシュの有効期限を指定した場合、有効期限が切れると、ブラウザはCONDITIONAL_GETによって内容の確認を行います。ですから、CGIプログラムの側がCONDITIONAL_GETに対応していなければ、この方法を本当の意味では活かすことができません。ですから、Expiresを使って有効期限を指定する場合には、以下に説明する、CONDITIONAL_GETへの対応もきちんと行うようにしてください。

●CONDITIONAL_GETへの対応

ブラウザがキャッシュを持っていた場合、「キャッシュされている内容が更新されていれば、新しいデータを送ってくれ」という形でアクセスを行うというのがCONDITIONAL_GETです。これにCGIプログラムが対応するには、

1：CONDITIONAL_GETリクエストに含まれるIf-Modified-Sinceヘッダの情報を正しく処理する
2：データを送りだす際には、かならずその元になったファイルの最終更新日の情報をLast-Modifiedヘッダで送るようにする

という２つの点を実装する必要があります。

２の方は簡単だと思いますのでこちらからまず説明します。ページの最終更新日を調べ、それをHTTP用の日付フォーマットの文字列に変換し、Last-Modifiedヘッダとしてつけるようにする、という処理を行います。

AppleScriptでの例をお見せすると、変数target_fileによって指定されているファイルを処理してデータとして送り返すというCGIプログラムであれば、

```
set myDate to getFileModDate target_file
set myDateStr to (DateToHeaderStr myDate with as1123) & " GMT"

return "HTTP/1.0 200 OK" & crlf ¬
  & "Content-type: text/html" & crlf ¬
  & "Last-modified: " & myDateStr & crlf ¬
  & crlf ¬
  & --- 以下、ページのデータ
```

*getFileModDateはファイルの最終更新日を得るosaxのコマンド
*DateToHeaderStrは日付データをヘッダ用の文字列に変換するosaxのコマンド

このスクリプトを実行すれば、以下のようなHTTPヘッダを付したデータがブラウザに送り返されます。

```
HTTP/1.0 200 OK
Content-type: text/html
Last-modified: Tue, 17 Feb 1998 20:18:04 GMT

--- 以下、ページのデータ
```

日付に用いる書式は、先程のExpiresの場合と同じです。これによって、ブラウザの方はキャッシュしたデータの元になった情報の最終更新日を知ることができますので、これに基づいて２度目からのアクセスの際には、リクエストの中にIf-Modified-Sinceヘッダを含めるようになります。

続いて、先程の１のCGIプログラムの側のCONDITIONAL_GETリクエストへの対応方法の説明に移ります。

CONDITONAL_GETと先程から言っていますが、リクエストの種類は通常のGETリクエストの形で、その中にIf-Modified-Sinceヘッダが含まれているもののことです。ですから、CONDITIONAL_GETによるリクエストかどうかを判定するには、ブラウザの送ってきたリクエストの中にIf-Modified-Sinceヘッダが含まれているかどうかを調べる必要があります。このため、ブラウザのリクエストのすべてを、パラメータでCGIプログラムに渡してくれるサーバでなければ、CONDITIONAL_GETには対応できません。対応しているサーバであれば、class 'Kfrq'のパラメータ（Full Request）にブラウザのリクエストが丸ごと入って送られてきますが、MacHTTPのような古いサーバではこのパラメータは送られてきませんので、注意してください。

CGIプログラム側の処理の手順は以下のようになります

- A：ブラウザのリクエストのヘッダ内にIf-Modified-Sinceが含まれているかどうかを調べる。含まれていなければ、通常のリクエストとして処理する。
- B：If-Modified-sinceヘッダが含まれていた場合には、そこから日付情報を取り出して、元になったファイル（データ）の最終更新日と照合する。もしファイルが更新されていた（キャッシュの方が古い）場合には、通常のリクエストして処理する
- C：ブラウザのキャッシュが有効であることが判明した場合には、ステータスコード304のHTTPヘッダ（"HTTP/1.0 304 Not Modified" ）を送り返す。

では、順に見ていきます。

まずブラウザのリクエストのチェックは簡単です。Full Requestパラメータの中に"If-Modified-Since:"という文字列が含まれているかどうかを判定すればよいわけです。

もし含まれていた場合には、この後に記述されている、ブラウザにキャッシュされているデータの最終更新日の情報をとりだすことになります。先程の例にあったように、ブラウザは以下のようなヘッダを送ってきますから、

```
If-Modified-Since: Tuesday, 10-Feb-98 09:18:26 GMT; length=43
```

ここから"Tuesday, 10-Feb-98 09:18:26 GMT"を取りだし、それをファイルとの照合に使える型のデータに変換する必要があります。プログラムを書く場合には、ここが一番面倒な部分だと言えるでしょう。

また、この処理において気をつけなければいけないことは、使われる可能性のある日付のフォーマットが３種類あるということです。

```
Sun, 06 Nov 1994 08:49:37 GMT  ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
Sun Nov  6 08:49:37 1994       ; ANSI C's asctime() format
```

サーバ（CGIプログラム）が発行する日付データの書式は、今後は一番上のRFC1123スタイルを用いなければならないのですが、ブラウザの方ではそれ以外の日付フォーマットを用いることもあります。事実、先程のHTTPヘッダの実例を見ていただければ分かるように、NetscapeはIf-Modified-SinceヘッダではRFC850(1036)のスタイルの日付フォーマットを用いています。ですから、CGIプログラムで処理する際には、どの日付フォーマットが用いられているのかを判定し、処理を切り替える必要が出てきます（実質的には、RFC1123スタイルとNetscapeが用いるRFC850スタイルに対応しておけばよいでしょう）。

If-Modified-Sinceヘッダの日付データを取り出せたら、それを元のファイルの最終更新日と照合し、更新されているかどうかを調べるということになります。そして、もし更新されていた場合には、通常のリクエストのようにデータを送り返せばいいわけです。そして、データが更新されていなかった場合（ブラウザのキャッシュが有効なものであった場合）には、以下のようなヘッダを送り返します。

```
HTTP/1.0 304 Not Modified
```

このヘッダさえ送り返せば、ブラウザは手持ちのキャッシュを使ってページを表示します。

以上が、CGIプログラムをCONDITIONAL_GETに対応させる方法です。

●Headリクエストに対応させる

サーチエンジンのRobotや、ブラウザの持っている更新状況チェック機能は、HEADリクエストをサーバに送ってページの状況を確かめようとします。ですから、CGIプログラムもこのHEADリクエストに対応させておくのが良いでしょう。

対応は簡単で、HEADリクエストが来た場合には、ページのHTTPヘッダだけを送り返すようにすればよいわけです。リクエストの種類は、通常は、CGIプログラムに送られてくるAppleEventの'meth'パラメータ（Methodパラメータ）で判断するのですが、残念ながらこのパラメータではHEADリクエストであることを判断することはできません。そこで、先程のCONDITIONAL_GETの場合同様に、Full Requestを解析してHEADリクエストであるかどうかを判定する必要があります。とはいえ、先頭がHEADになっているかどうかを判定すればいいだけですから、処理は簡単です。

●Pragma, Expires, CONDITIONAL_GET, HEAD対応のサンプルスクリプト

では、ここで、Expires, CONDITIONAL_GET, HEADに対応させたCGIプログラムのサンプルスクリプトを紹介しておきます。CGIプログラムの動作としては、http_search_argsとしてファイル名を指定すると、CGIプログラムと同じフォルダーに置いてあるmydocというフォルダーの名からそのファイルを探して内容を表示するという実用性はない、単純なものです。なお、Tanaka's osaxという自作のosaxのコマンドを各所に使っていますので、このosaxがインストールされていなければコンパイルできません。あくまでも、一つのサンプルとして見てください。

```
property crlf : (ASCII character 13) & (ASCII character 10)
```

```
property thisFldr : ""
property docFldr : ""

on run
  set thisFldr to (":" as alias) as string
  set docFldr to thisFldr & "mydoc:"
end run

on «event WWWΩsdoc» path_args ¬
  given «class kfor»:http_search_args, «class Kfrq»:full_request

  try

    set myPage to (DecodeURL http_search_args)
    set myFile to docFldr & myPage

    try
      set xDate to getFileModDate myFile
    on error errMsg number errNum
      if errNum = -43 then
        return "HTTP/1.0 404 File Not Found" & crlf & crlf ¬
          & "<TITLE>File Not Found</TITLE><H1>File Not Found</H1>"
      else
        error errMsg number errNum
      end if
    end try

    set xDate to xDate - (time to GMT)

    if word 1 of full_request = "HEAD" then
      --- HEAD リクエストの場合には、HTTPヘッダのみ送り返す
      return httpHeader(xDate, 300)
    end if

    try
      --- CONDITIONAL_GETの場合はzDateにブラウザが知らせてきた
      --- キャッシュのデータの更新日情報が収められる
      set zDate to ifModCheck(full_request)
    on error
      --- 通常のGETの場合は、zDateをオリジナルの更新日より過去に設定する
      --- こうすることで、かならずデータが送りだされる
      set zDate to xDate - 100
    end try

    if zDate = xDate then
      --- ファイルの更新が行われていなかった場合には、ステータス304を返す
      return "HTTP/1.0 304 Not Modified" & crlf ¬
        & crlf
    end if

    set myData to readFromFile myFile

    return httpHeader(xDate, 300) & "<TITLE>" & myPage & "</TITLE>" & crlf ¬
      & "<BODY BGCOLOR=\"FFFFFF\">" & crlf ¬
      & "<H3>" & myPage & "</H3><HR SIZE=1>" & crlf ¬
      & "<PRE>" & myData & "</PRE>"

  on error errMsg number errNb
```

```
    return "HTTP/1.0 200 OK" & crlf ¬
      & "Content-type: text/html; charset=Shift_JIS" & crlf & crlf ¬
      & "<TITLE>Error</TITLE>" & crlf ¬
      & "<BODY BGCOLOR=\"#FFFFFF\" text=\"firebrick\">" & crlf ¬
      & "<h3>実行中に以下のエラーが生じました</h3>" & crlf ¬
      & "<HR><P><TT>" & errMsg & "</TT>"
  end try

end «event WWWΩsdoc»

on httpHeader(xDate, cacheExp)
  --- xDate にデータの元になったファイルの最終更新日のデータ
  --- cacheExp によって、アクセスから何秒間キャッシュを有効にするのかを指定する
  --- cacheExp がマイナスの場合には、キャッシュを作らせない

  set dateStr to (DateToHeaderStr xDate with as1123) & " GMT"

  if cacheExp < 0 then
    set expStr to "Pragma: no-cache"
  else
    set zDate to (current date) - (time to GMT) + cacheExp
    set expStr to "Expires: " & (DateToHeaderStr zDate with as1123) & " GMT"
  end if


  return "HTTP/1.0 200 OK" & crlf ¬
    & "Content-type: text/html; charset=Shift_JIS" & crlf ¬
    & "Last-modified: " & dateStr & crlf ¬
    & expStr & crlf ¬
    & crlf
end httpHeader


on ifModCheck(full_req)

  --- CONDITIONAL_GETの場合は、ブラウザが知らせてきた日付情報を返す(GMTのまま)
  --- 通常のGETの場合は、このハンドラーがエラーになる（呼び出し元でエラートラップ
  --- を用いる）

  set Tm to item 1 of (pickUpFromData full_req startOf "If-Modified-Since: " endOf
return with Trim)
  set tgList to itemsToList Tm itemDelimiter " "

  if "-" is in Tm then
    --- Date Style = "Monday, 04-Aug-97 12:19:59 GMT"
    set dList to itemsToList (item 2 of tgList) itemDelimiter "-"
    set tList to itemsToList (item 3 of tgList) itemDelimiter ":"
    set xYear to ((item 3 of dList) as integer) + 1900
    set xmonth to mNameTomNum(item 2 of dList)
    set xDay to (item 1 of dList) as integer
  else
    --- Date Style = "Tue, 05 Aug 1997 01:09:40 GMT"
    set tList to itemsToList (item 5 of tgList) itemDelimiter ":"
    set xmonth to mNameTomNum(item 3 of tgList)
    set xYear to (item 4 of tgList) as integer
    set xDay to (item 2 of tgList) as integer
  end if

  set xdList to {xYear, xmonth, xDay, (item 1 of tList) as integer, (item 2 of
```

```
tList) as integer, (item 3 of tList) as integer}

  return (DateListToDate xdList)

end ifModCheck


on mNameTomNum(xmonth)
  --- 月名から月を割りだす
  if xmonth = "Jan" then
    return 1
  else if xmonth = "Feb" then
    return 2
  else if xmonth = "Mar" then
    return 3
  else if xmonth = "Apr" then
    return 4
  else if xmonth = "May" then
    return 5
  else if xmonth = "Jun" then
    return 6
  else if xmonth = "Jul" then
    return 7
  else if xmonth = "Aug" then
    return 8
  else if xmonth = "Sep" then
    return 9
  else if xmonth = "Oct" then
    return 10
  else if xmonth = "Nov" then
    return 11
  else if xmonth = "Dec" then
    return 12
  end if
end mNameTomNum
```

●Cookieヘッダを活用する

最後に、Cookieヘッダの利用方法を紹介します。

まず、Cookieの仕組みのポイントだけもう一度確認しておくと、

１：サーバからSet-Cookieヘッダによってブラウザにデータの保持を指示する
２：ブラウザはリクエストの際にCookieヘッダに保持している情報を並べて送る

というものです。ですから、CGIプログラムでこれを利用する場合も、Set-Cookieでデータの保持を指示する処理と、ブラウザから送られてきたCookieヘッダをチェックする処理を組み込むということになります。

まず最初にSet-Cookieによってデータの保持を指示する部分から説明します。この処理は、基本的にはreplyの際のHTTPヘッダ内に決められた書式でSet-Cookieヘッダをかいておけばよいわけですので、簡単です。Set-Cookieの書式は以下のようになります。

Set-cookie: 変数名=データ; expires=有効期限; path=有効範囲

たとえば、以下のようなヘッダになります。

```
Set-Cookie: xName=Tanaka; expires=Fri, 01-JAN-1999 05:00:00 GMT; path=/
```

また、複数のデータの保持を指定したい場合には、Cookieヘッダをいくつもならべて送ることが可能です。

では、それぞれの項目について説明していきましょう。

まず、変数名およびデータには、スペース、";",":"が入ってはなりません。これらの文字はヘッダのデリミタとして使われている文字だからです。データにどうしてもこれらの文字を含んだものを使いたいときにはURLエンコードしたものを使うようにしてください。

次に有効期限（expires）の指定ですが、GMTで日時を指定するのは先程のEXpiresヘッダなどの場合と同じなのですが、この際に用いる日付のフォーマットは以下の書式のものを使います。Netscapeが定めた書式で、先に紹介したHTTPヘッダで用いられるものとは少し異なっていますので気をつけてください（RFC850スタイルとRFC1123スタイルの混合型になってます）。

```
Wdy, DD-Mon-YYYY HH:MM:SS GMT
```

筆者が試した範囲では、最低限DD-Mon-YYYYを指定すれば、それが有効期限として通用するようなのですが、Netscape社のスペックでは上記のフォーマットで指定するように書かれていますので、それにしたがってください。

また、有効期限を指定しない場合は、そのデータはセッションの間（ブラウザを終了するまで）有効なものとして扱われます。ですから、ファイル（MagicCookie）には記録されませんし、その後のアクセス（ブラウザを立ち上げ直してのアクセス）においても使われません。ユーザーのアクセスのトラッキングを行うといった場合には、有効期限を指定しないのがよいでしょう。反対に、会議室のアクセス管理（前回のアクセス以降の新規登録発言の表示など）などに用いる場合には、適切な有効期限を指定する必要があります。特に有効期限を考えていない場合などは、１年先とか１０年先のような日付を使います。

有効範囲のpathですが、これはどのディレクトリのアクセスの際にCookieのデータを送るようにするのかを指示するものです。サーバのルート（/）を指定すると、そのサーバのすべてのアクセスにおいてCookieヘッダでデータを送ってくるようになります。/mycorner/のように特定のディレクトリーを指定することもできます。

なお、同じ変数名であっても、有効範囲の指定が異なると、別々に扱われます。たとえばあるページにおいてmyName=tanakaとしてpath=/を指定し、別のページでmyName=motoyukiでpath=/data/とすると、ブラウザは、/data/ディレクトリのアクセスの際にはCookieヘッダの中にmyName=tanakaとmyName=motoyukiの両方を並べて送ってきます。ですから、変数名のバッティングには気をつけてください。

では、続いて、Cookieヘッダを受け取る際の処理に移ります。

ブラウザのリクエストからCookieヘッダを取り出すには、先ほどのCONDITIONAL_GET同様に、Full RequestをCGIプログラムプログラムの側で解析するしかありませんので、Full Requestをパラメータで送ってくるサーバでしかCookieを利用できないということには注意してください。

ブラウザがCookieを送ってくる場合には、以下のように、そのページが有効範囲に含まれるデータを;で並べてきます。

```
Cookie: xMail=mact@fpu.ac.jp; xName=Tanaka; wsmID=0776616000
```

ですから、まずHTTPヘッダの中にCookieヘッダが含まれているかどうかを調べて、Cookieが使われているかどうかを確認し、Cookieヘッダが見つかった場合には、そこから自分の処理にとって必要な変数の値を抜き出すという処理を行います。Cookieヘッダの内容を"; "をデリミタにして切り分けた後に、"="をデリミタにしてそれぞれの項目を変数名とデータに分解してチェックするという方法をとるのがよいでしょう。

では、Cookieを使ったCGIプログラムプログラムの簡単なサンプルです。ここでは、サンプルということで、前回アクセスした時間をその都度表示するという単純なものになっています。

```
property crlf : (ASCII character 13) & (ASCII character 10)

on «event WWWΩsdoc» path_args ¬
  given «class scnm»:script_name, «class Kfrq»:full_request
```

```
  try

    --- "cookie_sample"という変数に、アクセスした日時の情報を
    --- タイムスタンプ形式で収めるようにする

    set ckValue to getCookie(full_request, "cookie_sample")

    if ckValue = "" then
      --- Cookieが送られてきていなかった場合
      set myDataMsg to "Cookie は記録されていません"
    else
      --- Cookieに"cookie_sample"のデータが送られてきていた場合
      set xDate to StampToDate ckValue
      set myDataMsg to "前回のアクセスは<TT> " & (xDate as string) & " </TT>です"
    end if

    set cookValue to ("cookie_sample=" & (TimeStamp))

    return cookieHeader(cookValue, script_name) & "<TITLE>Cookie Test</TITLE>" &
return ¬
      & "<H2>Cookie Test</H2>" & myDataMsg & return ¬
      & "<P><HR><P><H4>Full Request</H4><PRE>" & full_request & "</PRE>"

  on error errMsg number errNb
    set AppleScript's text item delimiters to oldDel
    return http_10_header & crlf ¬
      & "<TITLE>Error</TITLE>" & crlf ¬
      & "<BODY BGCOLOR=\"#FFFFFF\" text=\"firebrick\">" & crlf ¬
      & "<h3>実行中に以下のエラーが生じました</h3>" & crlf ¬
      & "<HR><P><TT>" & errMsg & "</TT>"
  end try

end «event WWWΩsdoc»


on getCookie(full_request, tgName)
  --- Full Request から Cookie の情報をとりだす
  --- Cookie が含まれなかった場合、あるいは tgName
  --- で指定した変数のデータがなかった場合には空文字列
  --- を返す
  try
    set myCookie to item 1 of (pickUpFromData full_request startOf "Cookie: " endOf
crlf with Trim)
    set myList to itemsToList myCookie itemDelimiter "=" lineDelimiter "; "
    set ckValue to ""
    repeat with thisC in myList
      set thisC to contents of thisC
      if item 1 of thisC = tgName then
        set ckValue to item 2 of thisC
        exit repeat
      end if
    end repeat

    return ckValue
  on error
    return ""
  end try

end getCookie
```

```
on cookieHeader(cookValue, tgPath)
  --- Set-Cookieヘッダを含むHTTPヘッダを生成する

  --- cookValue には(変数名＝データ)のセットが入る
  --- tgPath には有効範囲のパスが入る
  --- Cookie の有効期限は 1999年12月31日に設定する

  return "HTTP/1.0 200 OK" & crlf ¬
    & "Pragma: no-cache" & crlf ¬
    & "Content-type: text/html" & crlf ¬
    & "Set-Cookie: " & cookValue & "; expires=Fri, 31-Dec-1999 00:00:00 GMT; Path="
& tgPath & crlf ¬
    & crlf

end cookieHeader
```

なお、Pragma: No-cacheヘッダによって、キャッシュを作らないように指示を出していますので、reloadの度に、かならずアクセスを行いCookieのデータが更新されていくようになっています。

●その他のHTTPヘッダ活用

HTTPヘッダをCGIプログラミングの中で活かす方法は、ここで紹介しなかったものもたくさんあります。たとえば、最近のブラウザがリクエストで送ってくる情報の中には、ブラウザ側で設定された希望言語の情報（Accept-Languageヘッダ）が含まれますので、そこをチェックして英語と日本語のページを自動的に切り替える、あるいはHostヘッダをチェックしてバーチャルホストのコントロールを行うといったことも可能です。

４：最後に

CGIプログラミングというと、たいていの方は、プログラムの中でいかに凝ったこと（人目を引くようなこと）をするかといった点に気が行くと思いますが（CGIの解説書の中にも、そういう傾向のものが見られるようですが）、普段はあまり気に留めていないであろうHTTPヘッダの部分こそが、実はWebの通信が無事に行われるための要の部分であること、そして、この部分をうまく活用することで、サーバの機能だと思われがちな多くのことをCGIプログラムによってコントロールでき、CGIプログラムの可能性を広げることができるのだということ、そのことに気がつく切っ掛けにでもなれば幸いです。

02/19/98