Web と HTTP

v13 新機能

目次

4D v13 新機能 - Web と HTTP

背景

ほんの数年前まで、アプリケーションの展開対象 OS は Windows と Mac OS で十分でした。しかし iPhone の登場で状況は一変しました。

現在はアプリケーションを以下の環境で展開することを求められます。

	PC	タブレット	携帯電話
Apple	Mac OS	iOS	iOS
Google	(Chrome)	Android	Android
Microsoft	Windows	Windows	(Windows Phone)

ではこれらのアプリケーションはどのように開発したらよいのでしょう。現時点でネイティブなアプリケーションを開発するためには Apple 向けに Objective-C、Microsoft 向けに C#等、Android 向けに Java を使用することになります。そしてさらには各端末の画面に最適化したアプリケーションをそれぞれ作成しなければなりません。

これは Java が世に登場した動機にまったく逆行するものです。つまり"Write Once Run Anywhere"という希望が崩壊しつつあることを意味しています。

このドキュメントの読者は開発環境として 4D を採用していることでしょう。そしてその採用理由のひとつは Mac OS と Windows で大きく互換性が保たれていることにあるかもしれません。しかしタブレットや携帯電話からデータにアクセスする手段を提供するよう顧客から求められる日もそう遠くはないでしょう (すでにそのような相談をいただいています)。4D デベロッパーはこの課題に対しどのようなアプローチをとるべきでしょうか。

注目すべき点はこれらすべての端末にある共通のアプリケーション実行環境、Web ブラウザーがプリインストールされているということです。Google Docs で分かる通り、アプリケーション実行環境としてブラウザーが十分にその役割を果たすことができることをすでに皆さんはご存知でしょう。ブラウザーはアプリケーションの新しいプラットフォームになりつつあるのです。

この流れは HTML5、CSS3、および JavaScript (ECMAScript) の標準化、そしてブラウザーベンダ各社がこれらの技術サポートを表明したことにより決定的となっています。現時点で完全ではないものの、以前のように汚いコードを書かなければ動く Web アプリケーションが作れないという状況は改善しつつあります。

すべての端末をサポートするために Web アプリケーションという選択肢を検討するのはよいアプローチです。そして 4D はこのトレンドを採用するデベロッパーをサポートするため、以下のエリアにおいて Web に関する新機能を実装しました。

- Web エリア
- HTTP クライアント
- HTTP サーバー

Web エリア

ブラウザーコンポーネントである Web エリアは v11 SQL で登場しました。v11 と v12 の Web エリアは Windows で IE7、Mac OS では OS にインストールされている WebKit エンジンを使用していました。

4D v13 ではこの実装が以下の通り変更されています。

- 両プラットフォームで WebKit 版の Web エリアが利用できるようになりました。この WebKit コンポーネントは 4D に内蔵されています。
- 今までと同様、OS にインストールされている Web コンポーネントを使用することもできます (デフォルト)。この場合 v13 では両プラットフォームで OS にインストールされた最新バージョンが使用されます。

描画エンジンの選択

Web エリアは、Web コンテンツをフォームに表示するためのフォームオブジェクトとして登場しました。もっともシンプルな用法は、インターネットあるいはローカルの Web ページを表示するためのブラウザーとして利用するというものです。Web エリアはプラットフォームのブラウザーと同等のレンダリングができるので、画像や PDF、Windows では Office 文書なども、環境が整っていれば表示させることができます。あるいは JavaScript / CSS で作成された TinyMCE (WYSIWYG エディター) のようなモジュールを実行することもできます。しかしそのようなモジュールは高機能なものほどプラットフォームに依存する傾向が強く、一方のプラットフォームでは外観が崩れるなど、まったく動作しないこともあります。

v13 の Web エリアではこうした問題を解消するために変更が加えられました。

まずこれまでのような固定の描画エンジン (動作環境で最低レベルのもの) ではなく、システムにインストールされた最新のエンジン (Windows は IE、Mac OS は Safari) を使用するようになりました。つまりシステムの Web ブラウザーを更新すれば、Web エリアも自動的に更新されるというわけです。

加えて 4D に統合された WebKit を描画エンジンに選択することもできるようになりました。WebKit 版の Web エリアを使用すれば IE と Safari の非互換性を気にすることなく、Windows / Mac 両プラットフォームで動作する洗練された HTML5/CSS3 インターフェースを実現できます。

マッシュアップのために外部サイトを参照するブラウザーとして Web エリアを使用するのであれば、通常システム版の Web エリアを選択したいと思うことでしょう。システム版の Web エリアでは重要なセキュリティ更新が反映されるからです (ユーザーがセキュリティ更新を適用していれば)。

警告: Web エリアで Web サーフィンを許可することはいかなるケースであれ推奨されません。Web エリアからは明示的に許可されたサイトにのみアクセス可能にしなければなりません。一般的な Web ブラウザーに搭載されている追加のセキュリティレイヤー (EV SSL 対応等) が Web エリアには実装されていない点に注意してください。

WebKit 版 Web エリアは、インターネットサイトのブラウザーというよりは、デベロッパーが用意したインタラクティブコンテンツを表示するための手段として用意されました。WebKit 版の Web エリアでは、Mac / Windows で HTML/CSS/JavaScript が同じように動作するからです。WebKit 版の Web エリアを活用すれば、従来のフォームオブジェクトやイベント処理では無理と思われるようなユーザーインターフェースでも、簡単に実現できることが少なくありません。

アプリケーションと統合

Web エリアをフォームに配置すれば、すぐにでも HTML / CSS インターフェースをアプリケーションに統合することができますが、そのコンテンツをデータベースと連動させるためには、いくつかのコマンド / メカニズムを組み合わせることが必要です。基本的な考え方は、v11 ではじめて Web エリアが登場したときから変わっていません。とはいえ v13 で WebKit 版の Web エリアが利用できるようになったことを機に、4D デベロッパーの間で本格的にこのスタイルが普及するものと期待されています。

初回の表示

フォームに配置された Web エリアにデータベース由来のコンテンツを表示するためのもっとも効率的で簡単な方法は、あらかじめ用意された HTML/CSS/JavaScript テンプレートに 4D タグでデータを差し込むというものです。

これまで 4D HTML タグと呼ばれていたものは、v13 で 4D タグと改称されました。このテクニロジーは HTML ページに限定されたものではなくテキストファイル全般に幅広く応用できるからであり、Web サーバーとは関係なくダイナミックなコンテンツを生成できる強力なツールだからです。Web エリアの場合、4D タグが埋め込まれたテンプレートを処理することにより、データベースと連動した HTML ページを即座に生成して表示することができます。

```
<html>
<!--ヘッダーやその他のコンテンツ-->
 <!--#4DIF (Records in selection([テーブル])=0)-->
<p>表示するレコードがありません。</p>
<!--#4DELSE-->
<!--#4DLOOP [テーブル]--><p><!--#4DTEXT [テーブル]フィールド--></p>
<!--#4DENDLOOP-->
 <!--#4DENDIF-->
<!--ヘッダーやその他のコンテンツ-->
</html>
```

```
C_TEXT($content_t)
C_TIME($doc_h)

$doc_h:=Open document(テンプレートのパス;"";Read Mode)
If(OK=1)
  RECEIVE PACKET($doc_h;$content_t;Char(1))
  CLOSE DOCUMENT($doc_h)
  PROCESS 4D TAGS($content_t;$content_t)
  $doc_h:=Create document(データを埋め込んだドキュメントのパス)
```

```
 If(OK=1)
   SEND PACKET($doc_h;$content_t)
   CLOSE DOCUMENT($doc_h)
 End if
End if
```

注: Unicode モードでは PROCESS 4D TAGS コマンドに渡す引数の型をテキストにすべきです。

注: 2003 以前で作成されたストラクチャで 4D タグが正常に動作しない場合、互換性オプションの「ブラケットの代わりに 4DVAR コメントを使用しない」が原因なのかもしれません。

注: 4D v13 では新しい 4D タグ 4DELSEIF と 4DBASE が加わりました。

警告: 4DVAR と 4DHTMLVAR タグはセキュリティを強化するため、それぞれ 4DTEXT と 4DHTML にその役割が引き継がれました。特に 4DVAR の利用は (埋め込むデータの先頭から Char(1)を取り除いて自分でセキュリティを管理するのでない限り) すぐに利用を停止し、4DTEXT に置き換えてください。

書き出したファイルのパスを WA OPEN URL コマンドに渡すことにより、Web エリアに表示させることができます。しかし一部の JavaScript ライブラリはこの方法ではうまく初期化されないようです。そのようなときは、Web エリアの URL 変数(プロパティリストで設定)に POSIX パスを代入してみてください。

```
MyWArea_url:="file://"+Convert path system to POSIX($MyWArea_Path)
```

Web エリアから 4D メソッドを呼び出し

初回のページを表示した後、ユーザーが特定の箇所をクリックした、あるいは他の JavaScript イベントが発生したタイミングで、4D アプリケーションに対するコールバックを実行しなければならないことがあります。Web エリアから 4D メソッドのコールバックは、JavaScript イベントの種類に関係なく、基本的にすべて On URL Filtering イベントで処理します。

On URL Filtering は、あらかじめ登録された URL が Web エリア内で要求されたときに発生するイベントです。通常の URL プロトコル(http:、mailto:など)と区別するために、特別なプロトコル名を定義するのが一般的です。たとえば下記のコードを実行すると、m4d:からはじまるすべての URL 呼び出しにより On URL Filtering イベントが生成されるようになります。

```
ARRAY TEXT($filters_t;1)
ARRAY BOOLEAN($allowDeny_b;1)
$filters_t{1}:="m4d:*"
$allowDeny_b{1}:=False
WA SET URL FILTERS(*;Web エリア名;$arrFilters;$arrAllowDeny)
```

Web エリアのオブジェクトメソッドには、要求された URL のパスに応じ、必要な処理を実行するようなコードを記述します。

```
Case of
  : (Form event=On URL Filtering)
    $URL:=WA Get last filtered URL(OBJECT Get pointer(Object current)->)

    Case of
      : ($URL="m4d:getOfficeDetails:@")
          //...
      : ($URL="m4d:getEmployeeDetails:@")
          //...
      : ($URL="m4d:getOrderDetails:@")
          //...
    End case

End case
```

Web エリアのコンテンツ (HTML) には、上記の特別なプロトコル名から始まる URL を要求するような JavaScript を記述します。例えば

```
<a href="#" onclick="window.location.href='m4d:getOfficeDetails:2'; return
false;">...</a>
```

このようにすることで、結果的に Web エリアから 4D メソッドを呼び出すことができます (Web エリアオブジェクトで On URL Filtering イベントが生成されるので、そこからサブメソッドを呼び出す)。

4D メソッドから Web エリアを呼び出し

前述の例とは反対に、4D から Web エリアに命令を与える必要があるかもしれません。たとえば Web エリアからのメソッド呼び出しを処理し、その結果を Web エリアの内容に反映させたい場合です。そのために使用できるコマンドはふたつです。ひとつは標準の JavaScript 関数や Web エリアにロードされた記述済みのカスタム JavaScript 関数を呼び出すもので、もうひとつはコマンドに直接 JavaScript コード渡して実行させるものです。いずれのコマンドも JavaScript の戻り値をテキストで受け取ることができます。

```
WA EXECUTE JAVASCRIPT FUNCTION
WA Execute JavaScript
```

HTTP クライアント

現代では情報が数えきれないほどの Web サイト上にあります。特定のキーワードで検索を行えば関連する情報ページへのリンクリストを取得でき、リンクをクリックすればより詳細な情報にたどり着くことができます。カスタムプログラムを使用して Web サーバーと通信を行うことができれば、必要な情報を（人手に頼ることなく）短時間で入手できます。

またその逆に、Web サーバーに情報をアップロードできる必要もあります。現代の Web ではすべての人が単なる受動的なサービス享受者だけでなく、能動的な情報の発信者であることができます。

いままでは 4D Internet Commands などの外部ルーチンを使用してこれらの作業をプログラムする必要がありました。4D v13 では新しく HTTP クライアントコマンドテーマが追加され、この作業がより容易になりました。

HTTP クライアントとは

HTTP クライアントとは、HTTP を使用してサーバーと通信を行うアプリケーションであり、最もよく知られた HTTP クライアントは Web ブラウザーです。ブラウザーは主に Web ページをロードして表示するために使用されます。ユーザーはブラウザーの URL 欄にアクセスしたいページの URL を入力するか、リンクをクリックします。するとブラウザーは HTTP を使用してサーバーと通信を行い、必要なコンテンツをダウンロードしてウィンドウ内に表示します。

HTTP

HTTP は Hyper Text Transfer Protocol の略であり、HTTP サーバーと HTTP クライアント間の通信に使用されるプロトコルの名前で、RFC2616 により定義されています。ここで HTTP プロトコルについて詳述することはしませんが、HTTP クライアントコマンドを使いこなすにあたり、HTTP の以下の要素について知っておいてください。

HTTP メソッドの種類	意味
GET	指定した URI の情報を取得する
POST	サーバーに情報を送信する
OPTIONS	サーバー情報を取得する
HEAD	GET と同じだが HTTP ヘッダーのみを取得する
PUT	指定した URI に情報を保存する
DELETE	指定した URI の情報を削除する
TRACE	サーバーまでのネットワーク経路をチェックする
CONNECT	トンネルを使用してメッセージを動的に変更可能なプロクシで転送する

ほとんどのブラウザーおよびサーバーは GET と POST のみをサポートしています。しかし HTTP クライアントを自作する場合、そして HTTP サーバーがそれをサポートしていれば、上のメソッドを使用することができます。

HTTP で通信を行ったあとは、(HTTP サーバーに接続できていれば) HTTP レスポンスコードが返されます。例えばリクエストを正しく処理できた場合、サーバーからは 2xx 番台のコードが返されるはずです。主なレスポンスコードは以下の通りです:

コード	ステータス	説明
200	OK	クエストは成功し、レスポンスとともに要求に応じた情報が返される。
302	Found	発見した。リクエストしたリソースが一時的に移動されているときに返される。Location:ヘッダーに移動先の URL が示されている。
401	Unauthorized	認証が必要である。Basic 認証や Digest 認証などを行うときに使用される。
404	Not Found	未検出。リソースが見つからなかった。
500	Internal Server Error	サーバー内部エラー。サーバー内部にエラーが発生した場合に返される。

4D で HTTP クライアントコマンドを使用する状況

- リソースをダウンロードする
  ダウンロードしたリソース (HTML や画像) などを解析し、データベースに保存することができます。
- ローカルのデータを HTTP サーバーにアップロードする。

4D v12 までの HTTP クライアント実装

4D v12 までは、HTTP クライアントを実装するために以下のようなコードを書かなくてはなりませんでした (4DIC 利用時):

```
C_LONGINT($error_l;$tcpId_l;$state_l;$dstpos_l)
C_BLOB($toSend_x;$result_x;$buffer_x)
C_TEXT($response_t;$status_t)

$error_l:=TCP_Open (サーバーアドレス;ポート;$tcpId_l)
If ($error_l=0)
  $error_l:=TCP_SendBLOB ($tcpId_l;$toSend_x)// リクエスト送信
  If ($error_l=0)
    SET BLOB SIZE($result_x;0)
```

```
   Repeat // レスポンス受信の間繰り返す
     $error_l:=TCP_ReceiveBLOB ($tcpId_l;$buffer_x)
     $error_l:=TCP_State ($tcpId_l;$state_l)
     $dstpos_l:=BLOB size($result_x)
        // 受信した Blob を結合する
     COPY BLOB($buffer_x;$result_x;0;$dstpos_l;BLOB size($buffer_x))
   Until (($state_l=0)|($error_l#0))
   $error_l:=TCP_Close ($tcpId_l)
   $response_t:=BLOB to text($result_x;Mac text without length)
   $status_t:=Substring($response_t;Position("HTTP/1.";$response_t)+9;3)
 End if
End if
```

このコードではタイムアウトを実装していません。また取得したレスポンスからボディ部を取り出す必要があります。

4D v13 以降の実装

4D v13 の HTTP クライアントテーマコマンドを使用すれば、上記のコードは以下のようになります。

```
C_LONGINT($status_l)
C_TEXT($response_t)
$status_l:=HTTP Get(URL;$response_t)
```

$response_t にはボディ部が直接返されます（解析する必要はありません）。また先のコードにはなかったタイムアウトの設定も可能です。v13 では HTTP クライアントの実装が格段に容易になることをお分かりいただけるはずです。

HTTP クライアントコマンド

HTTP Get	指定した URL に HTTP GET リクエストを送信し、HTTP サーバーからのレスポンスを処理します。
HTTP Request	指定した URL に任意のメソッドで HTTP リクエストを送信し、HTTP サーバーのレスポンスを処理します。
HTTP SET OPTION	(HTTP Get や HTTP Request コマンドで実行される次のリクエストでクライアントにより使用される) HTTP オプションの値を設定します。
HTTP Get option	(HTTP Get や HTTP Request コマンドで実行される次のリクエストでクライアントにより使用される) HTTP オプションの値を返します。
HTTP AUTHENTICATION	クライアントアプリケーションの認証を要求するサーバーへの HTTP リクエストを可能にします。

HTTP Get

HTTP Get コマンドを使用すれば、指定した URL に GET リクエストを発行し、リソースを取得できます。リクエストに成功すれば、引数として渡した変数にレスポンスが返されます。

URL は以下の形式です:
{http|https}://[{user}:[{password}]@]host[:{port}][/{path}][?{queryString}]

URL の例:
http://www.myserver.com/
http://www.myserver.com:8080/
https://www.myserver.com/
http://www.myserver.com/path
http://www.myserver.com/path?name=jones
http://john:smith@ www.myserver.com/

取得したリソースは変数に受け取ります。HTML ファイルなどのテキストリソースはテキスト変数に、PNG などの画像リソースはピクチャー変数に受け取ります。なお BLOB 変数はテキストやピクチャーを含め、すべてのタイプのリソースを受け取れます。変数型とリソースの型が一致せず、4D が変換できないと判断した場合エラー54: 引数の型が正しくないが生成されます。

以下の例で HTTP Get コマンドは Web サーバーwww.example.com に GET リクエストを発行し、ルートの mypage.html を要求します。コマンドからレスポンスコード 200 が返されればリクエストは成功し、テキスト変数に mypage.html の内容が格納されます。

```
 // HTML ページを取得する例
C_LONGINT($status_l)
C_TEXT($result_t)
$status_l:=HTTP Get("http://www.example.com/mypage.html";$result_t)
If ($status_l=200)
   // 成功
Else
   // エラー
End if
```

注: リクエストが成功したことを示すレスポンスコードは 200 だけではありません。レスポンスコードの完全なリストは RFC2616 を参照してください。

次の例では同じサーバーに mypic.png を要求しています。今回はディレクトリに/images/が追加されている点に留意してください。

```
 // 画像ファイルを取得する例
C_LONGINT($status_l)
C_PICTURE($result_g)
$status_l:=HTTP Get("http://www.example.com/images/mypic.png";$result_g)
If ($status_l=200)
```

```
   // 成功
Else
   // エラー
End if
```

下記のようなケースでは 4D エラーが生成されるため、エラーを処理しなければなりません。この例では拡張子からファイルタイプを推測できますが、Web サーバーから (CGI を実行して結果を返す場合など) どの種のコンテンツが返されるか 100%確定できる手段はありません。ゆえに ON ERR CALL を使用してエラー処理を行うことは必須です。

```
 // エラーとなる例
C_TEXT($text_t)
C_PICTURE($pic_g)
C_LONGINT($status_l)

 // 画像をテキスト変数に受信
$status_l:=HTTP Get("http://www.example.com/sample.jpg";$text_t)

 // バイナリファイルをテキスト変数に受信
$status_l:=HTTP Get("http://www.example.com/sample.zip";$text_t)

 // テキストをピクチャー変数に受信
$status_l:=HTTP Get("http://www.example.com/sample.txt";$pic_g)

 // バイナリファイルをピクチャー変数に受信
$status_l:=HTTP Get("http://www.example.com/sample.zip";$pic_g)
```

以下のようにエラー処理を行います。

```
C_LONGINT($status_l)
C_PICTURE($pic_g)
C_LONGINT(HTTP_error)
ON ERR CALL("HTTP_error_handler")
HTTP_error:=0
$status_l:=HTTP Get("http://www.example.com/icon.jpg";$pic_g)
If (($status_l=200) & (HTTP_error=0))
   // 成功
Else
   // エラー
```

```
End if
ON ERR CALL("")
```

HTTP Request

HTTP Request コマンドは HTTP Get と同様 Web サーバーにリクエストを発行しリソースを受信しますが、任意の HTTP メソッドを使用できる点が異なります。

以下の例題は HTTP Get とまったく同様に動作します。

```
 // Web ページを取得する例
C_LONGINT($status_l)
C_TEXT($result_t)
$status_l:=HTTP Request(HTTP GET Method;¥
   "http://www.example.com/mypage.html";$result_t)
If ($status_l=200)
   // 成功
Else
   // エラー
End if
```

次の例では Web サーバーにポストメソッドリクエストを発行しています。Web サーバーへのリクエストのボディ部には″name=John Smith&company=Widgets LLC″が格納されます。

```
C_LONGINT($status_l)
C_TEXT($input_t;$result_t)

$input_t:="name=John Smith&company=Widgets LLC"
$status_l:=HTTP Request(HTTP POST Method;¥
   "http://www.example.com/sendrecord";$input_t;$result_t)
If ($status_l=200)
   // 成功
Else
   // エラー
End if
```

HTTP Get メソッドと同様、エラー処理は必須です。

HTTP SET OPTION と HTTP GET OPTION

HTTP SET OPTION コマンドを使用して、次の HTTP Get や HTTP Request コマンド実行時に使用される様々なオプションを設定できます。また HTTP Get option コマンドはオプションの設定値を取得するために使用します。

設定および取得が可能なオプションは以下の通りです。

オプション定数	説明
HTTP Timeout	Web サーバーが応答しない場合のタイムアウトを秒単位で設定します。デフォルト値は 120 秒です。
HTTP Max redirect	Web サーバーからリダイレクトレスポンスが返された場合に許可する最大リダイレクト数です。デフォルト値は 2 です。
HTTP Follow redirect	リダイレクトを許可する (1) かしないか (0) を設定します。デフォルトではリダイレクトを許可します。
HTTP Compression	リクエストヘッダーに Accept-Encoding: gzip, deflate を追加して、圧縮をサポートしていることを通知する (1) かしない (0) かを設定します。デフォルトでは通知しません。
HTTP Display auth dialog	サーバーから認証が要求されたとき、認証ダイアログを表示する (1) かしない (0) かを設定します。デフォルトでダイアログは表示されません。通常はダイアログを表示させずに HTTP AUTHENTICATE コマンドを使用します。
HTTP Reset auth settings	認証情報を削除する (1) か記憶する (0) かを設定します。デフォルトで認証情報は記憶されません。記憶させた場合、情報は 4D アプリケーションを終了するまで有効です。

HTTP Authenticate

このコマンドを使用すれば (BASIC または DIGEST) 認証を要求する Web サーバーにアクセスできます。

このコマンドを実行した時点ではリクエストが送信されないことに留意してください。このコマンドは HTTP Get や HTTP Request コマンド実行時、ヘッダーに認証情報を挿入するために使用されます。

HTTP サーバー

4D v13 では HTTP サーバーにも大幅に変更が加えられました。

- 4D Web サーバーによる自動セッション管理
- 受信したファイルを取り出すコマンドの追加
- 4DSYNC リクエスト受付の可否設定
- (HTTP サーバープロセスの最適化)
- (圧縮のサポート)

4D Web サーバーによる自動セッション管理

セッション管理

Web アプリケーションにおけるセッション管理とは一連のインタラクティブな操作を管理することを指します。例えばショッピングサイトで顧客が買い物をするとき、Web アプリケーション側ではクライアントアプリケーションを識別し、そのクライアントに割り当てられたカートに商品を登録し、決済を処理します。アプリケーションの実装によってはブラウザーを終了後、数日たったあとでさえ処理を継続することが可能です。

しかしながら HTTP 自体にはセッション管理という概念がありません。HTTP サーバーはクライアントからのリクエストを受信したら必要な処理を行ってレスポンスを返し、それで終わりです。レスポンス送信後、HTTP サーバーは誰がアクセスしてきて、何をリクエストされたのか、どのような処理を行ったかを忘れてしまいます (ログには記録されますがそれは別な話です)。しかし現実世界はどうでしょう。Amazon にアクセスすれば自分が過去にチェックしたもの、あるいは購入したものに基づきおすすめの商品をリストアップしてくれます。Gmail アカウントを取得してログインすれば、ブラウザーを再起動した後も自動で再ログイン状態になりメールボックスを表示してくれます。

これらの動作は Web アプリケーションにセッション管理機能を追加する事で実現されています。今日の Web アプリケーションにおいてセッション管理は重要なコンポーネントであり、あらゆる Web アプリケーションにセッション管理が実装されています。そしてそのほとんどは Web アプリケーションフレームワークが提供するセッション管理機能を使用しています。

HTTP におけるセッション管理は、アプリケーションが払い出したユニークなセッション ID を何らかの方法でクライアントアプリケーションに送信し、次回同じクライアントからのリクエストにその ID が含まれることを期待することで行われます。現時点では cookie が使用されることが多く、ほとんどの 4D デベロッパーが cookie を使用してセッション管理を実装していると思われます。

セッション変数

セッション ID を使用してクライアントアプリケーションを識別したら、それまでに行われた作業 (カートにどの商品が追加されたか等) をリストアできます。そのような情報を格納する場所を"セッション変数"と呼びます。情報はメモリ上、データベース、あるいはディスクファイルに格納できますが、Web アプリケーションからセッション ID をキーとして検索できなければなりません。

4D v11 まで

バージョン 11の時点で 4D Web サーバー開発フレームワークにセッション管理機能は含まれておらず、開発者が自作しなければなりませんでした。しかし問題はどのようにセッション ID を生成するかにあります。独立行政法人情報処理推進機構 (IPA) 発行の"安全な Web サイトの作り方"には

> セッション ID は、生成アルゴリズムに安全な擬似乱数生成系を用いるなどして、予測困難なものにしてください。

と記述されています。しかしバージョン 11 までは 4D にそのような ID を払い出すコマンドは実装されておらず、(OpenSSL などの) 外部ルーチンを使用する必要がありました。

4D v12

バージョン 12 ではセッション ID を払い出すことが可能なコマンド"Generate UUID"が提供されました。したがって以下のようなコードを外部ルーチンに頼ることなく記述できます。

```
 // Web プロセス内でレスポンスヘッダーに cookie を設定する
C_TEXT($header_t)

$header_t:="Set-cookie: SESSID="+Generate UUID
SET HTTP HEADER($header_t)
```

しかしながら自動的なセッション管理機能は含まれておらず、開発者がそれぞれ自作する必要があります。

```
 // Web プロセス内でリクエストヘッダーから cookie を取り出す
C_LONGINT($index_l)
C_TEXT($cookie_t)
ARRAY TEXT($fields_t;0)
ARRAY TEXT($values_t;0)

GET HTTP HEADER($fields_t;$values_t)
$index_l:=Find in array($fields_t;"Cookie")
If($index_l>0)
  $cookie_t:=$values_t{$index_l}
   // さらに目的の cookie 名を探し、値を取得する
   // セッション ID を取りだしたらセッション変数をロードする
End if
```

v13

バージョン 13 では 4D Web サーバーが自動でセッションを管理する機能が追加されました。4D v13 を使用すれば、開発者は大量のコードを記述することなく、標準 HTTP がもつ制限を回避できます。

注: この機能を無効にすることもできます。また v12 以前から変換されたデータベースでは無効になっています。有効にするためにはデータベース設定ダイアログで設定を変更しなければなりません。

4D v13 のセッション管理は以下のように動作します。

新規リクエストまたは (タイムアウトのため) 有効な ID を持たないリクエストの場合

1. 4D Web サーバーが有効なセッション ID を持たないリクエストを受信すると、まずセッション ID を払い出し、その ID を名前に持つ Web プロセスを作成します。
2. Web プロセス内で処理を行います。
3. レスポンスをクライアントに返信する際、4D は自動でセッション ID を cookie に設定し、レスポンスヘッダーに含めます。
4. その Web プロセスはレスポンスを返信した後も (タイムアウト時間が経過するまで) 破棄されません。

有効なユニーク ID を持つリクエストの場合

1. 次のリクエストには、リクエストヘッダーの Cookie フィールドに有効なセッション ID が含まれます (cookie および Web プロセスがタイムアウトになっていないとして)。
2. 4D は自動でそのセッション ID に対応する Web プロセスをリクエストの処理に割り当てます。
3. 同じセッション ID がレスポンスヘッダーに設定され、返信されます。

同じプロセスが再利用される点に注目してください。これは 4D のプロセスオブジェクトであるカレントセレクション、カレントレコード、プロセス変数、レコードのロック状態が保持されることを意味します。デベロッパーは (4D クライアントアプリケーションを作成するように) Web アプリケーションを開発することができます。

この自動セッション管理機能をデベロッパーが制御するためのコマンドが追加されています。

WEB Get current session ID	現在の Web プロセスに対応するセッション ID を取得するコマンドです。
WEB SET OPTION	Web の動作に関する様々なパラメーターを設定するためのコマンドです。セッション管理においては以下の設定が関連します。 *cookie の name 属性値 (デフォルトで"4DSID") *自動セッション管理を許可する最大プロセス数 (デフォルトは 20 プロセス) *Web クライアントに返信される cookie のタイムアウト (デフォルトは現在時刻+24 時間) *自動セッション管理 Web プロセスのタイムアウト (デフォルトは 2 分) *自動セッション管理の有効/無効の切り替え
WEB GET OPTION	Web の動作に関する様々なパラメーターを取得するためのコマンドです。WEB SET OPTION 参照。
WEB CLOSE SESSION	何らかの理由でマシンリソースを解放したい場合、このコマンドを使用して特定の Web プロセスの自動セッション管理を終了させることができます。
WEB GET SESSION EXPIRATION	指定されたセッション ID の cookie の有効期限を返します。

終了される Web プロセスのプロセスオブジェクトの管理

タイムアウト、WEB CLOSE SESSION の実行、あるいは最大自動セッション管理 Web プロセス数に達したなどの理由で、セッションを管理するプロセスが終了されることがあります。この結果プロセスオブジェクトはメモリから消去され、レコードロックは解除されます。次にそのセッション ID を使用したリクエストを受信した場合、対応する有効なプロセスが存在しないため、新規にセッション ID が発行され、Web プロセスが作成されます。

例えば先に示した通り、Web プロセスのタイムアウトはデフォルトで 2 分であるのに対し、セッション管理に使用される cookie の有効期限は 24 時間です。2 分以内に改めてリクエストが行われない場合、cookie が有効であるのにそのセッションを管理する Web プロセスがタイムアウトする状況が発生します。

このようなケースでは、メモリ上のプロセスオブジェクトを一時的に別の場所に格納し、後で再利用できるようにする必要があるかもしれません。この目的のために On Web Session Suspend という新しいデータベースメソッドが追加されました。

On Web Session Suspend データベースメソッドは、セッションを管理する Web プロセスが終了される直前に、その Web プロセス内で実行されます。つまりこのデータベースメソッドから Web プロセスのプロセスオブジェクトにアクセスできます。そのため、このデータベースメソッドを使用して必要な情報をメモリから他の場所 (データベースレコードやファイル、エクスターナルデータベース) にコピーすることができます。

```
 // On Web Session Suspend データベースメソッド
 // セッション情報を格納するために様々なアプローチをとることが可能です。
 // これはそれを簡略した一例です。

 // カレントセレクション
ARRAY LONGINT($recNums_l;0)
LONGINT ARRAY FROM SELECTION([Table_1];$recNums_l)

 // カレントレコードの位置
C_LONGINT($currentRecPos_l)
$currentRecPos_l:=Selected record number([Table_1])

 // セッション情報をレコードに格納する
CREATE RECORD([HttpSessions])
[HttpSessions]SessionID:=Web Get Current SessionID
VARIABLE TO BLOB($recNums_l;[HttpSessions]SessionVar)
VARIABLE TO BLOB($currentRecPos_l;[HttpSessions]SessionVar)
SAVE RECORD([Table_1])
```

この後、同じセッション ID をもつリクエストを受信したとき、先に保存したセッション情報をテーブルレコードから読み出して状態を復元することが可能です。

```
 // Web リクエストを処理するメソッド

C_TEXT($clientCookie_t)
C_LONGINT($offset_l;$currentRecPos_l)

$clientCookie_t:=UTIL_WEB_GetCookie("4DSID")
If($clientCookie_t#"") & ($clientCookie_t#WEB Get Current Session ID)

 ARRAY LONGINT($recNums_l;0)

  // セッション情報をロードする
 QUERY([HttpSessions];[HttpSessions]SessionID=$clientCookie_t)
 If(Records in selection([HttpSessions])>0)
  BLOB TO VARIABLE([HttpSessions]SessionVar;$recNums_l;$offset_l)
  BLOB TO VARIABLE([HttpSessions]SessionVar;$currentRecPos_l;$offset_l)
  DELETE RECORD([HttpSessions])
 End if

  // 以前のセッションの状態を回復する
 CREATE SELECTION FROM ARRAY([Table_1];$recNums_l)
 GOTO SELECTED RECORD([Table_1];$currentRecPos_l)

End if
```

注: この例はコンセプトを説明するために簡略化してあります。以下の点に留意してください。

- ユーザーは cookie に設定されたセッション ID の有効期限を無視することが可能です。セッション ID の有効期限は Web アプリケーション側でも管理すべきです。
- この例では VARIABLE TO BLOB コマンドを使用して BLOB に変数を格納しています。この場合 BLOB TO VARIABLE コマンドで順番に変数を取りださなければなりません。他の方法としてすべての情報を名前と値のペアで 2 つのテキスト配列に格納し、常にその 2 つのテキスト配列を BLOB に出し入れする、JSON や XML 記法でひとつのテキストデータにする、SET LIST ITEM PARAMETER でひとつの階層リストに情報を格納するなどが考えられます。バイナリデータをテキストにするには BASE64 ENCODE コマンドを使用できます。
- 上記の Web リクエストを処理するメソッドのコンテキストでは、クライアントから送信されてきたセッション ID と、このプロセス内で WEB Get Current Session ID コマンドを実行して取得できる現在有効なセッション ID は異なります。レスポンスが返信される際には (あたらしく発行された) 後者のセッション ID が使用されます。

- UTIL_WEB_GetCookie メソッドは http://kb.4d.com/search/assetid=76339 で紹介されています。

重要な注意事項

4D v13 の自動セッション管理を使用するにあたり、以下の点に注意する必要があります。

- 4D v13 リリース時点では、セッションを特定するにあたり、クライアントの IP アドレスも判定条件とされています。セッション ID と IP アドレスが一致しない場合、無効かつ有害なリクエストと判定され 400 (Bad Request) が返されます。これは携帯電話のような移動端末からアクセスする場合に問題となる可能性があります。このようなケースでは端末の IP アドレスが変更される可能性があるからです。現時点でこの判定条件は仕様であり、これを回避するためには v12 と同様のプログラミングが必要です。
- 4D v13 の自動セッション管理は、4D のプロセスオブジェクト (カレントセレクションやプロセス変数) 等を再利用するためだけの機能です。この自動セッション管理をユーザー認証に使用してはいけません。ユーザー認証を行うためには HTTP 標準の (SSL 経由の BASIC、または DIGEST) 認証、または (Generate UUID と SET HTTP HEADER コマンドを使用して) 追加の ID を cookie に設定し、独自に管理してください。その理由は以下の通りです。
    - ユーザー認証に ID を使用する場合、ログイン前とログイン後では ID を変更しなければなりませんが、自動セッション管理にはその機能がありません。ID を変更しない場合、セッション固定攻撃に対し脆弱となります。
    - ユーザー認証が行われる局面では SSL 通信用の別のセッション ID を使用する必要がありますが、自動セッション管理にはその機能がありません。4D の自動セッション管理では非 SSL 通信でも SSL 通信でも同じセッション ID が使用されるため、セッション ID が傍受されないことを保証できません。

受信したファイルを取り出すコマンドの追加

4D v13 の Web サーバーコマンドテーマには、アップロードされたファイルの処理を劇的に容易にするコマンドが追加されました。

Web アプリケーションにおいては、以下のような Web フォームを使用してクライアントからファイルのアップロードを受け付けることができます。

```
<form method="post" action="/4DCGI/UPLOAD" enctype="multipart/form-data">
<input type="file" name="file_x"> <!--name 属性値が file_x である点に注意-->
<input type="submit">
</form>
```

上記のフォームから送信されたファイルを受信する 4D メソッドは以下のようになります。

```
// 4D v12 まで
// compiler_web メソッドで C_BLOB(file_x)を宣言しておく
// この時点で file_x 変数には受信したファイルおよびその情報が格納されている
// ファイル情報とその内容は BLOB 変数内で CRLFCRLF で区切られているため
// ファイルの内容を取り出すためには file_x 内の CRLFCRLF 位置を検索し
// それ以降の BLOB コンテンツをファイルとして取り出す必要がある
```

このコードはとても長くなるため、ここで紹介することはできません。

```
// 4D v13 からは

C_BLOB($blobPart_x)
C_LONGINT($i;$size_l)
C_TEXT($name_t;$mimetype_t)

For($i;1;WEB Get body part count)
  WEB GET BODY PART ($i;$blobPart_x;$name_t;$mimetype_t;$size_l)
    // コンテンツのバリデーションを行い、有効であればファイルを保存する
End for
```

これだけです。4D v13 ではアップロードされたファイルの解析がとても容易になることがお分かりいただけるはずです。

警告: ファイルの有効性を確認するのは (以前のバージョンと同様) 開発者の仕事です。Web クライアントからは MIME タイプや拡張子を偽装したデータを送信可能なため、開発者はファイルの内容を解析し、データとして妥当であるかどうかを確かめなければなりません。

4DSYNC リクエスト受付の可否設定

4D v12 より、4DSYNC URL による 4D データベースへの問い合わせが可能となりました。例えば

```
HTTP 1.1 GET /4DSYNC/$catalog
```

上記のようなリクエストを 4D Web サーバーが受信すると、4D は同期が有効なテーブルのリストとそのテーブルに含まれるレコード数を返します。これはリモートのアプリケーションが HTTP を利用して 4D とデータの同期を行うために実装されました。

通常の Web アプリケーションや Web サービスと大きく異なる点は、この URL リクエストを処理するために HTML ファイルやメソッドは必要なく、4D が自動でレスポンスを生成する点です。このため、4DSYNC のリクエストを拒否するためには On Web Authentication データベースメソッドを使用する必要がありました。

v13 ではデータベース設定を使用して、4DSYNC URL の受け付けを拒否できるようになりました。