(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第5010551号
(P5010551)

(45)発行日　平成24年8月29日(2012.8.29)　(24)登録日　平成24年6月8日(2012.6.8)

(51)Int.Cl.　F I

G06F 9/44	(2006.01)	G06F	9/44	530P
G06F 13/00	(2006.01)	G06F	13/00	560A
G06F 12/00	(2006.01)	G06F	12/00	546A

請求項の数 11　外国語出願　(全 30 頁)

(21)出願番号　特願2008-199892(P2008-199892)
(22)出願日　平成20年8月1日(2008.8.1)
(62)分割の表示　特願2001-129924(P2001-129924)の分割
原出願日　平成13年4月26日(2001.4.26)
(65)公開番号　特開2009-70372(P2009-70372A)
(43)公開日　平成21年4月2日(2009.4.2)
審査請求日　平成20年9月1日(2008.9.1)
(31)優先権主張番号　09/573768
(32)優先日　平成12年5月18日(2000.5.18)
(33)優先権主張国　米国(US)

(73)特許権者　500046438
マイクロソフト　コーポレーション
アメリカ合衆国　ワシントン州　９８０５２－６３９９　レッドモンド　ワン　マイクロソフト　ウェイ
(74)代理人　100077481
弁理士　谷　義一
(74)代理人　100088915
弁理士　阿部　和夫
(72)発明者　バード、ゲイリー　エス.
アメリカ合衆国、９８０３３　ワシントン州、カークランド、エヌイー　１０３ｄ　ストリート　１１４１１

最終頁に続く

(54)【発明の名称】動的ウェブページコンテンツファイルからのサーバ側コード生成

(57)【特許請求の範囲】
【請求項１】
メモリとプロセッサとを有するサーバコンピュータシステムにおいてコンピュータにより実行可能なクラスをメモリ内に作成する方法であって、前記クラスは、前記サーバコンピュータシステムがウェブページコンテンツを動的にレンダリングするためのサーバ側オブジェクトを作成するために用いられるものであり、前記ウェブページコンテンツは、クライアントコンピュータシステムに送信され前記クライアントコンピュータシステム上でウェブページとして表示される、方法であり、
前記プロセッサが前記クライアントから動的なウェブページコンテンツファイルを特定するリクエストを受信する工程と、
前記プロセッサが前記動的なウェブページコンテンツファイルを処理して、前記ウェブページコンテンツファイルにおいて宣言された制御オブジェクトを表すソースコードを含むソースコードファイルを生成する工程と、
前記プロセッサが前記ソースコードファイルをコンパイルして、表示するウェブページを生成するウェブページオーサリング言語のデータを作成するために１組の階層オブジェクトのインスタンスが作成され得るクラスを生成する工程と
を含み、
前記ソースコードファイルを生成する工程において、
前記動的なウェブページコンテンツファイルの構文を解析して、前記動的なウェブページコンテンツファイルの一部を、階層的に関連付けられた複数のデータオブジェクトを含

んでいるデータモデルへ格納する工程と、
前記データモデルの解析に基づいて宣言情報に関するソースコードを生成する工程と、前記宣言情報に関するソースコードを前記ソースコードファイルへ書き込む工程とを含む第１フェーズと、
前記データモデルの解析に基づいて制御オブジェクト情報に関するソースコードを生成する工程と、前記制御オブジェクト情報に関するソースコードを前記ソースコードファイルへ書き込む工程とを含む第２フェーズと
を含むことを特徴とするクラスを作成する方法。
【請求項２】
前記動的なウェブページコンテンツファイルが、サーバ側宣言データ格納部にある請求項１に記載のクラスを作成する方法。
【請求項３】
前記クラスは、前記サーバコンピュータシステムのキャッシュメモリに格納され、前記動的ウェブページコンテンツファイルを特定する別のリクエストへのレスポンスでオブジェクトのインスタンスを作成するために用いることができる請求項１に記載のクラスを作成する方法。
【請求項４】
前記クラスは、磁気記憶媒体に格納され、前記動的なウェブページコンテンツファイルを特定する別のリクエストへのレスポンスでオブジェクトのインスタンスを作成するために用いることができる請求項１に記載のクラスを作成する方法。
【請求項５】
前記方法は、
前記プロセッサが前記データモデルの解析に基づいてレンダリング情報に関するソースコードを生成する工程と、前記プロセッサが前記レンダリング情報に関するソースコードを前記ソースコードファイルへ書き込む工程とを含む第３フェーズ
を含む請求項１に記載のクラスを作成する方法。
【請求項６】
前記第１フェーズが完了すると前記第２フェーズが生じ、前記第２フェーズが完了すると前記第３フェーズが生じる請求項５に記載のクラスを作成する方法。
【請求項７】
前記第１フェーズ、前記第２フェーズおよび前記第３フェーズが同時に生じる請求項５に記載のクラスを作成する方法。
【請求項８】
前記プロセッサが前記ソースコードファイルを生成する工程の前に、前記受信したリクエストに関するクラスがコンパイルされメモリに格納されているかどうか判断する工程をさらに含み、
前記クラスがコンパイルされメモリに格納されている場合、前記ソースコードファイルを生成する工程をスキップし、そうでない場合は、前記ソースコードファイルを生成する工程を行う請求項１に記載のクラスを作成する方法。
【請求項９】
コンピュータにより実行可能なクラスをメモリ内に作成するコンピュータプログラムを記憶したコンピュータ読取可能記憶媒体であって、前記クラスは、前記サーバコンピュータシステムがウェブページコンテンツを動的にレンダリングするためのサーバ側オブジェクトを作成するために用いられるものであり、前記ウェブページコンテンツは、クライアント側コンピュータシステムに送信され、前記クライアントコンピュータシステム上でウェブページとして表示され、前記コンピュータプログラムは、
前記クライアントから動的なウェブページコンテンツファイルを特定するリクエストを受信する工程と、
前記動的なウェブページコンテンツファイルを処理して前記ウェブページコンテンツファイルにおいて宣言された制御オブジェクトを表すソースコードを含むソースコードファ

イルを生成する工程と、
表示するウェブページを生成するウェブページオーサリング言語のデータを作成するために１組の階層オブジェクトのインスタンスが作成され得るクラスを生成するべく前記ソースコードファイルをコンパイルする工程と
をコンピュータに実行させ、
前記ソースコードファイルを生成する工程において、
前記動的なウェブページコンテンツファイルの構文を解析して、前記動的なウェブページコンテンツファイルの一部を、階層的に関連付けられた複数のデータオブジェクトを含んでいるデータモデルへ格納する工程と、
前記データモデルの解析に基づいて宣言情報に関するソースコードを生成する工程と、
前記宣言情報に関するソースコードを前記ソースコードファイルへ書き込む工程と、
前記データモデルの解析に基づいて制御オブジェクト情報に関するソースコードを生成する工程と、
前記制御オブジェクト情報に関するソースコードを前記ソースコードファイルへ書き込む工程と
をコンピュータに実行させることを特徴とするコンピュータ読取可能記憶媒体。
【請求項１０】
メモリとプロセッサとを有するサーバコンピュータシステムにおいて、動的にレンダリングされたウェブページコンテンツを有し、１つ以上のクライアント側コンピュータシステムに運ばれ、前記クライアントコンピュータシステム上でウェブページとして表示される複数のウェブページレスポンスを作成する方法であって、
前記プロセッサが動的ウェブページコンテンツファイルを識別するリクエストを、前記ウェブページのための前記クライアントコンピュータシステムから受信する工程と、
前記プロセッサが前記動的なウェブページコンテンツファイルの要素を格納するためのデータモデルを作成する工程であって、前記動的なウェブページコンテンツファイルの構文を解析して、前記動的なウェブページコンテンツファイルの一部を、階層的に関連付けられた複数のデータオブジェクトを含んでいるデータモデルへ格納する工程を含む、データモデルを作成する工程と、
前記プロセッサが前記データモデルの評価に基づき、前記動的なウェブページコンテンツファイルに関するソースコードファイルを生成する工程であって、
前記データモデルの解析に基づいて宣言情報に関するソースコードを生成する工程と、
前記宣言情報に関するソースコードを前記ソースコードファイルへ書き込む工程と、
前記データモデルの解析に基づいて制御オブジェクト情報に関するソースコードを生成する工程と、
に前記制御オブジェクト情報に関するソースコードを前記ソースコードファイルへ書き込む工程と
を含む、ソースコードファイルを生成する工程と、
前記プロセッサが前記ソースコードファイルをコンパイルしてメモリにコンパイルされたクラスを作成する工程と、
前記プロセッサが前記サーバコンピュータシステムにクラスリファレンスを戻す工程であって、これにより、前記サーバコンピュータシステムが当該クラスからサーバ側処理オブジェクトのインスタンスを作成することが可能になり、ウェブページコンテンツを動的に生成する工程と、
前記プロセッサが前記動的なウェブページコンテンツを前記クライアントコンピュータシステムに送信するウェブページレスポンスへレンダリングする工程と、
前記プロセッサが前記ウェブページレスポンスを前記要求しているクライアントコンピュータシステムへ導く工程と、
前記プロセッサが動的ウェブページコンテンツファイルを識別する第２のリクエストを、前記ウェブページのために受信する工程と、

前記プロセッサが当該動的ウェブページコンテンツファイルのコンパイルされたクラスがメモリに存在するかどうかを判断する工程と、
前記プロセッサが前記サーバコンピュータシステムにクラスリファレンスを戻す工程であって、これにより、前記サーバコンピュータシステムが当該クラスからサーバ側処理オブジェクトのインスタンスを作成することが可能になり、ウェブページコンテンツを動的に生成する工程と、
前記プロセッサが前記動的ウェブページコンテンツを第２のウェブページレスポンスへレンダリングする工程と、
前記プロセッサが前記第２のウェブページレスポンスを前記要求しているクライアントコンピュータシステムへ導く工程と
を含むウェブページレスポンスを作成する方法。
【請求項１１】
動的にレンダリングされたウェブページコンテンツを有し、１つ以上のクライアント側コンピュータシステムに送信され、前記クライアントコンピュータシステム上でウェブページとして表示される、複数のウェブページレスポンスを作成するコンピュータプログラムを記憶したコンピュータプログラム記憶媒体であって、前記コンピュータプログラムは、
動的ウェブページコンテンツファイルを識別するようなリクエストを、前記ウェブページのための前記クライアントコンピュータシステムから受信する工程と、
前記動的なウェブページコンテンツファイルの要素を格納するためのデータモデルを作成する工程であって、前記動的なウェブページコンテンツファイルの構文を解析して、前記動的なウェブページコンテンツファイルの一部を、階層的に関連付けられた複数のデータオブジェクトを含んでいるデータモデルへ格納する工程を含む、データモデルを作成する工程と、
前記データモデルの評価に基づき、前記動的なウェブページコンテンツファイルに関するソースコードファイルを生成する工程であって、
前記データモデルの解析に基づいて宣言情報に関するソースコードを生成する工程と、
前記宣言情報に関するソースコードを前記ソースコードファイルへ書き込む工程と、
第２フェーズ時に前記データモデルの解析に基づいて制御オブジェクト情報に関するソースコードを生成する工程と、
第２フェーズ時に前記制御オブジェクト情報に関するソースコードを前記ソースコードファイルへ書き込む工程と
を含む、ソースコードファイルを生成する工程と、
前記ソースコードファイルをコンパイルしてメモリにコンパイルされたクラスを作成する工程と、
前記サーバコンピュータシステムにクラスリファレンスを戻す工程であって、これにより、前記サーバコンピュータシステムが当該クラスからサーバ側処理オブジェクトのインスタンスを作成することが可能になり、ウェブページコンテンツを動的に生成する工程と、
前記動的ウェブページコンテンツを前記クライアントコンピュータシステムに送信するウェブページレスポンスへレンダリングする工程と、
前記ウェブページレスポンスを前記要求しているクライアント側コンピュータシステムへ導く工程と、
動的ウェブページコンテンツファイルを識別する第２のリクエストを、前記ウェブページのために受信する工程と、
当該動的ウェブページコンテンツファイルのコンパイルされたクラスがメモリに存在するかどうか判断する工程と、
前記サーバコンピュータシステムにクラスリファレンスを戻す工程であって、これにより、前記サーバコンピュータシステムが当該クラスからサーバ側処理オブジェクトのイン

スタンスを作成することが可能になり、ウェブページコンテンツを動的に生成する工程と、
前記動的ウェブページコンテンツを第２のウェブページレスポンスへレンダリングする工程と、
前記第２のウェブページレスポンスを前記要求しているクライアントコンピュータシステムへ導く工程と
をコンピュータに実行させることを特徴とするコンピュータ読取可能記録媒体。

【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、一般にウェブサーバフレームワークに関し、特にウェブページのクライアント側ユーザインタフェース要素を処理する制御（コントロール）オブジェクトを作成するサーバ側コード作成に関する。
【背景技術】
【０００２】
典型的なウェブブラウザは、クライアントシステムに表示するウェブページの外観および基本動作を定義するウェブサーバからデータを受け取る。典型的な手順としては、ユーザが、ワールドワイドウェブ上の資源のグローバルアドレスであるユニホームリソース（資源）ロケータ（「URL」）を特定し、所望のウェブサイトにアクセスする。URLの一例は、" HYPERLINK "http://www.microsoft.com/ms.htm" http://www.microsoft.com/ms.htm "である。このURL例の第１の部分は、通信に用いる所与のプロトコル（例えば"http"）を示している。第２の部分は、資源の所在を示すドメイン名（例えば" HYPERLINK "http://www.microsoft.com" www.microsoft.com" ）を特定している。第３の部分はドメイン内の資源（例えば"ms.htm"と呼ばれるファイル）を特定している。これに従い、ブラウザは、 HYPERLINK "http://www.microsoft.com" www.microsoft.com ドメイン内のms.htmファイルに関連するデータを取り出すためのURL例に関連するHTTP(ハイパーテキストトランスポートプロトコル)リクエストを生成する。www.microsoft.comサイトをホストしているウェブサーバはHTTPリクエストを受け取り、要求されたウェブページまたは資源をクライアントシステムにHTTPレスポンスで戻し、ブラウザに表示する。
【０００３】
上記の例の"ms.htm"ファイルは、静的なHTML（ハイパーテキストマークアップ言語）コードを含んでいるウェブページファイルと対応する。HTMLは、ワールドワイドウェブ上でのドキュメント（例えば、ウェブページ）作成に用いられるプレイン（平文）テキストオーサリング言語である。このようなものであるので、HTMLファイルは、HTMLコードを実際の視覚的な画像または音声コンポーネントへ変換するクライアントブラウザによってウェブサーバから取り出され、ウェブページとして表示される。クライアントコンピュータシステムにおいて、このプロセスは、渡されたHTMLファイルに定義されたウェブページ内容を表示する。HTMLを用いて、開発者が、例えば、ブラウザに表示するフォーマット化されたテキスト、リスト、フォーム、テーブル、ハイパーテキストリンク、インライン画像および音声、ならびに背景グラフィックスを特定する。しかし、HTMLファイルは、ウェブページコンテンツの動的な生成を本質的にサポートしていない静的ファイルである。ウェブページコンテンツは、表示のためにクライアントに戻されるHTMLコードである。このような動的処理は、単にクライアントブラウザに所定のコードを送信するのではなく、処理ステップの結果、送信に先立ちHTMLコードを生成するサーバ側アプリケーションに関する。
【０００４】
より複雑なクライアント・サーバ間の対話をハンドリングするために、サーバ側アプリケーションプログラムは、動的コンテンツ、例えば、変化している株価または交通情報を与えるような、より複雑なクライアント・サーバ間の対話をハンドリングするように開発されてきた。サーバ側アプリケーションプログラムは、HTTPリクエストを処理し、クライアントにHTTPレスポンスでクライアントへの送信に適切なHTMLコードを生成する。例えば

、サーバ側アプリケーションプログラムは、クライアント側へのHTTPレスポンスにおける送信HTMLコードを動的に生成するために、HTTPリクエストにおいてクライアント側によって提供される照会ストリングまたはウェブベースのフォームからのデータを処理することができる。本質的に、サーバ側アプリケーションは、クライアント側からのリクエストにおける情報に基づきカスタマイズされたHTML型ファイルを生成することができる。この場合、サーバ上に格納された静的なHTMLファイルは存在せず、HTMLファイルは、実行時に動的に作成される。サーバ側アプリケーションプログラムの一例として、メモリ機構への１つ以上のフォーマット化されたテキスト書き込み処理のシーケンスを用いて、HTMLコードを生成する場合がある。その後、得られたテキストは、HTTPレスポンスでクライアントシステムに送信され、そこでブラウザに表示される。

【０００５】

サーバ側アプリケーションプログラムの開発は、ウェブページ設計に用いる通常のHTMLコード化に精通しているだけでなく、１つ以上のプログラム言語（例えば、C++, Perl, Visual Basic, または Jscript）を含むプログラムベーシックスに精通していることが要求される複雑な作業である。しかし、残念なことに、ウェブページ設計者は、グラフィックデザイナーまたはエディタであることが多く、人間的な感じを与えるが、プログラミング経験がない場合が多い。したがって、プログラミング経験が少ない人が、サーバ側アプリケーションとそのそれぞれのクライアントとのウェブページインタフェースを開発することができるような、ウェブページファイルを作成するための単純化されたウェブページ開発フレームワークを提供することが必要である。したがって、開発者が最小のプログラミングでウェブページを動的に作成および処理することができる開発フレームワークを提供することが望まれている。

【発明の開示】

【発明が解決しようとする課題】

【０００６】

動的ウェブページ生成のプログラム要件を最小にする１つの手段は、マイクロソフト社によって提供されるアクティブサーバページ（ASP）フレームワークである。ASPフレームワークにより、開発者は、典型的に、Visual BasicまたはJscript コードおよびその他のHTMLコードを含む"ASP"ウェブページファイルを作成できる。ASPファイルは、種々の機能を実行する宣言またはタグならびにVBスクリプトまたはJスクリプトコードを含む。一般に、実際のプログラミングコードを書き込むより、これらの宣言を書き込むほうが簡単である。

【０００７】

処理中、HTTPリクエストは、所望の資源としてASPファイルを特定し、その後、ASPファイルを用いてクライアント側へのHTTPレスポンスでの結果HTMLコードを生成する。さらに、ASPファイルは、所与のアプリケーションプログラミング労力を減らすために、予め開発されたまたは第三者のクライアント側ライブラリコンポーネント（例えば、クライアント側"ACTIVEX"制御）およびデータベースまたは他の第３者アプリケーションを参照することができる。

【０００８】

単純化されたASPウェブページファイルは、実行時間でスクリプトエンジンにより解釈され得るスクリプトに変換されなければならない。スクリプトエンジンは、典型的に、所望の結果を達成するために、連続または同期的にASPファイルにおける種々の宣言タイプコマンドを実行する。実行処理可能なファイルとしてコンパイルされ格納されたファイルと比較して、一般に、スクリプトエンジンによって実行されたスクリプトファイルは時間がかかる。これは、スクリプトエンジンが単にファイルを実行するだけではなく、解釈機能を果たさなければならないからである。

【０００９】

スクリプトファイルを実行処理可能なファイルにコンパイルする際のある１つの問題は、スクリプトファイルには、おそらく種々の言語の組み合わせがある、またはその可能性

があるということである。例えば、スクリプトファイルは、HTMLで書かれたコンポーネントおよびVisual Basicで書かれた他のコンポーネントを含む場合がある。スクリプトエンジンは、実行時間にこれらの要素を解釈するために種々の処理を用いるが、異なる言語コンポーネントを１つの言語、すなわち、１つのソースコードファイルに翻訳するためのコンパイラが存在しない。さらに、現在のサーバ側アプリケーションフレームワークにおいて、サーバ側アプリケーション内でクライアント側ユーザインタフェース要素（例えば、テキストボックス、リストボックス、ボタン、ハイパーリンク、画像、音声など）を動的に管理することを要求されるプログラミングは、依然として高度なプログラミング技術と相当な努力を必要とする。これらのサーバ側プロセスが、より複雑になるにしたがって、スクリプトエンジンは、持続的にこの要求を満たし続けることができなくなる。

【００１０】

これらおよび他の理由により、本発明がなされた。

【課題を解決するための手段】

【００１１】

本発明は、中間言語またはソースコードファイルをサーバ側資源から作成するためのコード生成方法および装置に関するものであり、ソースコードファイルは、実行処理可能なクラスにコンパイルされる。実行処理可能なクラスにより、クライアントレスポンスのレンダリングを含むサーバ側機能を実行するウェブページ制御オブジェクトの素早い生成が可能になる。本発明のある実施形態において、コード生成スキームは、イベント処理および特定のオブジェクトへの属性の設定を扱うために階層内で接続された制御オブジェクトを作成することができる。さらに、コード生成方法はまた、テンプレートを用いて宣言されたオブジェクトを接続することもできる。

【００１２】

より好ましくは、本発明は、サーバコンピュータシステムメモリにおけるクラスの作成方法に関する。クラスは、ウェブページコンテンツを動的にレンダリングするサーバ側オブジェクトを作成するために、サーバコンピュータシステムによって用いられ、ウェブページコンテンツは、クライアント側コンピュータシステムに送られ、クライアントコンピュータシステム上のウェブページとして表示される。処理において、サーバコンピュータシステムは、ウェブページのためのリクエストをクライアントコンピュータシステムから受信し、このリクエストが動的ウェブページコンテンツファイルを識別する。サーバコンピュータは、動的ウェブページコンテンツファイルの要素を格納するためのデータモデルを作成し、データモデルを評価し、データモデルの評価に基づき動的なウェブページコンテンツファイルに関するソースコードファイルを生成する。ソースコードが作成されると、ソースコードファイルがコンパイルされて、コンパイルされたクラスをメモリに作成する。一般に、プロセスは、サーバコンピュータシステムにクラス参照を戻してサーバコンピュータシステムがクラスを用いることが可能になって終了する。

【００１３】

他の好ましい実施形態によると、方法は、サーバコンピュータシステム上のキャッシュメモリにクラスを格納する。一旦、キャッシュメモリに格納されると、多数のサーバ側ページオブジェクトが１つのコンパイルされたクラスからインスタンスを作成し、元々の資源は再度使用されない。ウェブページに対するリクエストを受信する度に、サーバコンピュータシステムは、その動的ウェブページコンテンツファイルのコンパイルされたクラスがメモリにあるかどうかを判断する。要求されたクラスが、メモリに存在しない場合は、それを作成する。クラスがあれば、サーバコンピュータシステムは、ウェブページコンテンツを動的に生成するために、そのクラスからサーバ側処理オブジェクトのインスタンスを作成する。次に、ウェブページコンテンツはレンダリングされクライアントコンピュータシステムに送られる。

【００１４】

本発明のさらに別の実施形態によれば、データモデルを評価する方法ステップは、複数のパスでデータモデルの反復トラバースを伴う。各パス時にソースコードが生成され、そ

のパス時のデータモデルの評価に基づきソースコードファイルに書き込まれる。データモデルは、階層的に連結されたデータ構造を用いて構築される。

【００１５】

本発明によるコンピュータプログラムプロダクトの実施形態は、コンピュータシステムによって読み取り可能であり、サーバコンピュータ上にメモリにコンパイルされたクラスを作成するコンピュータプロセスを実行処理するコンピュータプログラムをコード化するコンピュータプログラム記憶媒体を含む。コンパイルされたクラスは、クライアントコンピュータシステム上に表示される要求されたウェブページに対応するレスポンスをレンダリングするためにサーバ側処理オブジェクトのインスタンスを作成するのに用いられる。本発明によるコンピュータプログラムプロダクトの別の実施形態は、コンピュータシステムによる搬送波によって具体化され、サーバにコンパイルされたクラスを作成するためのコンピュータプログラムをコード化するコンピュータデータ信号を含む。

【発明を実施するための最良の形態】

【００１６】

本発明の実施の形態は、動的ウェブページコンテンツ資源またはファイルによって定義された特定のウェブページのためのメモリにコンパイルされたクラスを生成する方法に関するものである。コンパイルされたクラスの作成は、ウェブページファイルからソースコードファイルの作成を伴う。次に、ソースコードファイルがクラスにコンパイルされる。コンパイルされたクラスが、メモリに存在すると、表示するクライアントに送り返されるレスポンスをレンダリングするためにページオブジェクトのインスタンスが作成され得る。一般に、ページオブジェクトは、ウェブページ上に表示されるクライアント側ユーザインタフェース要素の処理および生成のためのサーバ側制御オブジェクトを伴う。さらに、サーバ側制御オブジェクトの階層が、これらのオブジェクトが、クライアント上のウェブページの表示のためのHTMLのような結果オーサリング言語コードを最終的に協働して生成するウェブページファイルに宣言され得る。

【００１７】

図１は、本発明のある実施形態におけるクライアントに表示するウェブページコンテンツを動的に生成するウェブサーバを示す。クライアント１００は、クライアント１００の表示装置上にウェブページ１０４を表示するブラウザ１０２を実行処理する。クライアント１００は、ビデオモニタ（図示せず）のような表示装置を有するクライアントコンピュータシステムを含む。マイクロソフト社によって販売されている"INTERNET EXPLORER"ブラウザは、本発明のある実施形態におけるブラウザ１０２の一例である。他のブラウザの例として、"NETSCAPE NAVIGATOR" および "MOSAIC"などがあるが、これに限らない。例示したウェブページ１０４には、テキストボックス制御１０６および２つのボタン制御１０８、１１０が組み込まれている。ブラウザ１０２は、HTTPレスポンス１１２でウェブサーバ１１６からHTMLコードを受信し、HTMLコードにより記述されたウェブページを表示する。ある実施形態を参照してHTMLについて説明するが、特に制限されるものではなく、SGML（Standard Generalized Markup Language）およびXML（eXtensible Markup Language）を含む他のオーサリング言語も本発明の範囲内であると考えられている。

【００１８】

クライアント１００とウェブサーバ１１６との通信は、HTTPリクエスト１１４およびHTTPレスポンス１１２のシーケンスを用いて行うことができる。ある実施形態を参照してHTTPを説明するが、特に制限されるものではなく、S－HTTPを含めた他のトランスポートプロトコルも本発明の範囲内であると考えられている。ウェブサーバ１１６において、HTTPパイプラインモジュール１１８が、HTTPリクエスト１１４を受け取り、URLを解析し、リクエストを処理する適切なハンドラ１２０を呼び出す。本発明のある実施形態において、異なるタイプの資源を扱う複数のハンドラ１２０がウェブサーバ１１６に備わっている。

【００１９】

例えば、URLがHTMLファイルのような静的なコンテンツファイル１２２を特定する場合、ハンドラ１２０は、静的コンテンツファイル１２２にアクセスし、HTTPパイプライン１

１８を介して静的コンテンツファイル１２２をHTTPレスポンス１１２でクライアント１００へ送る。あるいは、本発明のある実施形態において、URLがASP+（Active Server Page +）ページのような動的コンテンツ資源またはファイル１２４を特定する場合、ハンドラ１２０は、動的コンテンツファイル１２４にアクセスし、動的コンテンツファイル１２４のコンテンツを処理し、ウェブページ１０４用の結果HTMLコードを生成する。一般に、ファイル１２４のような動的コンテンツ資源は、クライアントに表示すべきウェブページを記述するオーサリング言語を動的に生成するのに用いることができるサーバ側宣言データ格納部である。そして、ウェブページ用のHTMLコードは、HTTPパイプライン１１８を通って、HTTPレスポンス１１２でクライアント１００へ送られる。

【００２０】

この処理の間、ハンドラ１２０はまた、開発労力を簡素化するために予め開発された、または第３者コードのライブラリにアクセスすることができる。このようなライブラリの１つがサーバ側クラス制御ライブラリ１２６であり、ここから、ハンドラ１２０は、ユーザインタフェース要素を処理しウェブページに表示する結果HTMLデータを生成するサーバ側制御オブジェクトのインスタンスを作成することができる。本発明のある実施形態において、１つ以上のサーバ側制御オブジェクトは、動的コンテンツファイル１２４に記述されたウェブページ上に、可視的にまたは隠して、１つ以上のユーザインタフェース要素にマッピングする。

【００２１】

ハンドラ１２０はまた、ウェブサーバ１１６上または別のアクセス可能ウェブサーバ上で実行処理する１つ以上の非ユーザインタフェースサーバコンポーネント１３０にアクセスする。株価検索アプリケーションまたはデータベースコンポーネントのような非ユーザインタフェースサーバコンポーネント１３０は、ハンドラ１２０によって処理される動的コンテンツファイル１２４において参照され、またはそれに関連付けられる。非ユーザインタフェースサーバコンポーネント１３０は、動的コンテンツファイル１２４で宣言されたサーバ側制御オブジェクトによって立ち上げられたイベントを処理し得る。その結果、サーバ側制御オブジェクトによって提供される処理は、ウェブページのユーザインタフェース要素の処理および生成をカプセル化することによって非ユーザインタフェースサーバコンポーネント１３０のプログラミングを簡単にし、これにより、非ユーザインタフェースサーバコンポーネント１３０の開発者は、ユーザインタフェース問題ではなく、アプリケーション固有の機能の開発に集中することができる。

【００２２】

図２は、本発明のある実施形態におけるサーバ側制御オブジェクトを用いてのクライアント側ユーザインタフェース要素の処理および生成処理のフローチャートを示す。処理２００において、クライアントは、HTTPリクエストをサーバに送信する。HTTPリクエストは、ASP+ページなどの資源を特定するURLを含む。処理２０２において、サーバは、HTTPリクエストを受信し、特定された資源を処理する適切なハンドラを呼び出す。ASP+ページは処理２０３において読み出される。処理２０４は、特定された動的コンテンツファイル（例えば、ASP+ページ）の内容に基づきサーバ側制御オブジェクト階層を生成する。

【００２３】

処理２０６において、制御オブジェクト階層のサーバ側制御オブジェクトは、ポストバックイベントハンドリング、ポストバックデータハンドリング、状態管理およびデータ結合のうちの１つ以上の処理を行う。処理２０８において、階層内の各サーバ側制御オブジェクトが、クライアント側ユーザインタフェース要素のウェブページでの表示のためのHTMLコードのようなデータを生成（またはレンダリング）するために呼び出される。用語「レンダリング」は、ユーザインタフェース上にグラフィックスを表示する処理を意味することがあるが、本明細書において、用語「レンダリング」は、表示およびクライアント側機能のためのブラウザのようなクライアントアプリケーションによって解釈され得るオーサリング言語データの生成処理をも意味している。処理２０６およびレンダリング処理２０８のより詳細な説明は図６との関連で行われている。個々の制御オブジェクトにおける

renderメソッドの呼び出しは、ツリートラバーサルシーケンスを用いて行われる。すなわち、ページオブジェクトのrenderメソッドの呼び出しは、階層内の適切なサーバ側制御オブジェクトにわたる反復トラバースになる。

【００２４】

あるいは、個々のサーバ側制御オブジェクトの実際の作成は、サーバ側制御オブジェクトが処理２０６または２０８においてアクセスされる（ポストバック入力のハンドリング、状態のロード、制御オブジェクトからHTMLコードのレンダリングなど）まで遅らせてもよい。サーバ側制御オブジェクトが所与のリクエストのためにアクセスされることがない場合、制御オブジェクトの作成を遅らせ、不要な制御オブジェクト作成処理を排除することによって、サーバ処理が最適化される。

【００２５】

処理２１０において、HTMLコードをHTTPレスポンスでクライアントに送信する。処理２１４において、クライアントは、表示されるべき新たなウェブページに関連するHTMLコードを受信する。処理２１６において、クライアントシステムは、HTTPレスポンスから受け取ったHTMLコードに応じて新しいページのユーザインタフェース要素を表示する。処理２１２において、サーバ側制御オブジェクト階層を終了する。階層内サーバ側制御オブジェクトは、関連付けられたASP+ページを参照するHTTPリクエストに応答して作成され、オーサリング言語データ（例えば、HTMLデータ）のレンダリングが終わると破壊される。あるいは、処理２１２は、処理２０８の後、処理２１０の前に行ってもよい。

【００２６】

図３は、本発明のある実施形態において用いるウェブサーバでのモジュールの一例を示す。ウェブサーバ３００は、HTTPパイプライン３０４にHTTPリクエスト３０２を受信する。HTTPパイプライン３０４は、ウェブページ統計のロギング、ユーザ照合、ユーザアクセス権およびウェブページの出力キャッシュ化用モジュールなどの種々のモジュールを含んでもよい。ウェブサーバ３００によって受信される各入力HTTPリクエスト３０２は、最終的には、インタフェースハンドラ（Interface Handler）、例えばIHTTPハンドラクラス（ハンドラ３０６として図示）の特定のインスタンスによって処理される。ハンドラ３０６は、URLリクエストを解析し適切なハンドラファクトリ（例えば、ページファクトリモジュール３０８）を呼び出す。

【００２７】

図３において、ASP+ページ３１０に関連付けられたページファクトリ３０８が呼び出され、ASP+ページ３１０からのオブジェクトのインスタンスの作成および構成をハンドリングする。ASP+ページ３１０は、一意のURLによって認識または参照され得、他の接頭辞も用い得るが、例えば、".aspx"という接頭辞によってさらに識別することができる。特定の".aspx"資源に対するリクエストがまず、ページファクトリモジュール３０８によって受信されると、ページファクトリモジュール３０８は、ファイルシステムを検索して適切な資源またはファイル（例えば、.aspxページ３１０）を得る。このファイルは、リクエストを処理するサーバによって後に解釈またはアクセスされ得るテキスト（例えば、オーサリング言語データ）または別のフォーマットでのデータ（例えば、バイトコードデータまたはコード化されたデータ）を含んでもよい。物理的なファイルが存在する場合、ページファクトリモジュール３０８は、ファイルを開き、そのファイルをメモリに読み出す。あるいは、要求されたファイルは存在するが、予めメモリにロードされている場合は、以下に詳細に述べるように、資源は必ずしもメモリにロードされる必要はないかもしれない。要求されたaspxファイルが見つからない場合、ページファクトリモジュール３０８は、例えばHTTP"404"メッセージをクライアントに送信することによって、適切な「ファイルが見つからない」というエラーメッセージを戻す。

【００２８】

ASP+ページ３１０をメモリに読み出した後、ページファクトリモジュール３０８は、ファイル内容を処理し、ページのデータモデル（例えば、スクリプトブロックのリスト、ディレクティブ、静的テキスト領域、階層サーバ側制御オブジェクト、サーバ側制御プロパ

ティなど）を構築する。データモデルは、ページオブジェクトの構造、プロパティおよび機能を定義するコードであるページベースのクラスを拡張させるCOM+（Component Object Model+）クラスのような新たなオブジェクトクラスのソースコードファイルを生成するのに用いられる。本発明のある実施形態において、ソースリストは、中間言語に動的にコンパイルされる。のちにプラットフォームに固有の命令（例えば、X86、Alphaなど）に適時に（Just-In-Time）コンパイルされる。中間言語は、COM+ IL コード、Java バイトコード、Modula 3 コード、SmallTalk コードおよびVisual Basicコードなどの汎用またはカスタム指向言語コードを含んでもよい。別の実施形態において、中間言語処理を省き、固有の命令をソースリストまたはソースファイル（例えば、ASP+リソース３１０）から直接生成する。制御クラスライブラリ３１２は、制御オブジェクト階層の生成に用いられる予め定義されたサーバ側制御クラスを得るためにページファクトリモジュール３０８によってアクセスされ得る。

【００２９】

ページファクトリモジュール３０８は、コンパイルされたクラスからページオブジェクトを作成する。ページオブジェクト３１４は、図１のウェブページ１０４に相当するサーバ側制御オブジェクトである。ページオブジェクト３１４およびその子オブジェクト（例えば、テキストボックスオブジェクト３１８、ボタンオブジェクト３２０および別のボタンオブジェクト３２２）が、制御オブジェクト階層３１６の一例である。他の制御オブジェクトの例は、本発明に従い考えることができ、カスタム制御オブジェクトと同様に、特に制限されるものではなく、表１に示すHTML制御に対応するオブジェクト（後述）も含んでいる。ページオブジェクト３１４は、図１のウェブページ１０４に対応する。テキストボックスオブジェクト３１８は、図１のテキストボックス１０６に対応する。同様に、ボタンオブジェクト３２０は、図１の追加ボタン１０８に対応し、ボタンオブジェクト３２２は、図１の削除ボタン１１０に対応する。ページオブジェクト３１４は、サーバ上の他の制御オブジェクトと階層的に関連している。ある実施形態において、ページオブジェクトは、その子オブジェクトを階層的に含んでいるコンテナオブジェクトである。別の実施形態では、依存関係などの他の形態の階層関係を用いることができる。多数レベルの子オブジェクトを有する、より複雑な制御オブジェクト階層において、１つの子オブジェクトが、他の子オブジェクトのコンテナオブジェクトであってもよい。

【００３０】

上記の実施形態において、制御オブジェクト階層３１６の制御オブジェクトは、サーバ３００上で作成および実行され、各サーバ側制御オブジェクトは、クライアント上の対応するユーザインタフェース要素と論理的に対応する。サーバ側制御オブジェクトはまた、協働して、HTTPリクエスト３０２からのポストバック入力をハンドリングし、サーバ側制御オブジェクトの状態を管理し、サーバ側データベースとのデータ結合を行い、クライアントでの結果ウェブページの表示に用いるオーサリング言語データ（例えば、HTMLコード）を生成する。結果オーサリング言語データは、サーバ側制御オブジェクト階層３１６から生成（すなわち、レンダリング）され、HTTPレスポンス３２４でクライアントに送信される。例えば、結果HTML（または他のオーサリング言語）コードは、ACTIVEXタイプの制御またはその他、ブラウザによって処理されたとき、クライアント側ユーザインタフェース要素（例えば、制御ボタン、テキストボックスなど）を生み出すHTML構成を参照することができる。

【００３１】

ASP+リソース３１０でなされた宣言によって、サーバ側制御オブジェクトは、非ユーザインタフェースサーバコンポーネント３３０とクライアント側ユーザインタフェース要素との対話のために、１つ以上の非ユーザインタフェースサーバコンポーネント３３０にアクセスすることができる。例えば、ポストバック入力に応答して、サーバ側制御オブジェクトは、サーバ側イベントをそれらのイベント用に登録された非ユーザインタフェースサーバコンポーネントに対し立ち上げることができる。このように、非ユーザインタフェースサーバコンポーネント３３０は、ユーザとの対話を、ユーザインタフェース要素を介し

て、これらの要素を表示および処理するのに必要なコードをプログラミングすることなく、行うことができる。

【００３２】

図４は、本発明のある実施形態における動的コンテンツ資源の一例の内容を示す。例示された実施例において、ファイル４００は、ある動的コンテンツファイルフォーマット（例えば、ASP+）でプレインテキスト宣言を含んでいる。各宣言は、ファイル４００を読み出すページファクトリモジュール３０８に対し命令を与え、クラスを作成し、最終的にHTMLコードまたはその他、クライアントへHTTPレスポンスで送信するオーサリング言語をレンダリングする適切なサーバ側制御オブジェクトを呼び出す。

【００３３】

ファイル４００の第１ラインは、以下のフォーマットにおいてデリミッタ"<%@"と" %>"との間にディレクティブを含む：

```
<%@ directive [attribute=value] %>
```

ここで、directiveは、特に制限されるものではなく、"page"、"cache"または"import"を含んでもよい。ディレクティブは、バッファリングセマンテックス、セッション状態要件、エラーハンドリングスキーム、スクリプティング言語、トランザクションセマンテックスおよびインポートディレクティブのような特性を決定するために、動的コンテンツファイルを処理するときにページファクトリモジュール３０８によって用いられる。ディレクティブは、ページファイル内のどこに存在してもよい。

【００３４】

第２ラインの<html>は、直接的な「書き込み」命令以外、結果HTMLコードをレンダリングするための情報において追加の処理が行われないようにリテラルとしてソースコードファイルへ書き込まれる標準HTML開始タグである。HTML構文において、<html>は、HTMLファイルの始まりを示し、これもまたリテラルであるライン２１の終了タグ</html>と対になっている。

【００３５】

コード宣言ブロックは、ファイル４００のライン３～１０に存在する。一般に、コード宣言ブロックは、ページオブジェクトおよび制御オブジェクトメンバ変数ならびにサーバ上で実行処理されるメソッドを定義する。以下のフォーマットにおいて：

```
<script runat = "server" [language = "language"][src = "externalfile"]>
.....................................
</script>
```

ここで、言語およびsrcパラメータは任意である。本発明のある実施形態において、コード宣言ブロックは、"サーバ"に設定された値を有する"runat"属性を含む<script>タグを用いて定義される。任意には、"language"属性を内コードの文法を特定するために用いてもよい。デフォルト言語は、ページ全体の言語構成を表すことができるが、コード宣言ブロックの"language"属性により、開発者は、例えば、Jscript およびPERL (Practical Extraction and Report Language)などの同じウェブページ実行内で異なる言語を用いることができる。<script>タグはまた、任意には"src"ファイルを特定することができ、"src"ファイルは、そこからページコンパイラによる処理のための動的コンテンツ資源にコードが挿入される外部のファイルである。開示されている文法が本実施形態において用いられているが、別の実施形態では、本発明の範囲内で異なる文法を用いることができることを理解すべきである。

【００３６】

図４において、２つのサブルーチン、AddButton#ClickおよびDeleteButton#Clickがコード宣言ブロック内においてVisual Basicフォーマットで宣言されている。いずれのサブルーチンも２つの入力パラメータ、"Source"および"E"をとり、クライアント側クリックイベントが対応のボタンに検知されると呼び出される。AddButton#Clickサブルーチンにおいて、ユーザ名テキストボックスでのテキストは、単語"Add"に連結され、メッセージのテキストデータメンバにロードされる。DeleteButton#Clickサブルーチンにおいて、ユ

ーザ名テキストボックスでのテキストは、単語"Delete"に連結され、メッセージのテキストデータメンバにロードされる。図４に示していないが、サーバ側制御オブジェクトのメンバ変数は、ファイル４００のコード宣言ブロックで宣言され得る。例えば、Visual Basic文法を用いると、キーワード"DIM"は、サーバ側制御オブジェクトのデータ変数を宣言する。

【００３７】

「コードレンダリングブロック」（図示せず）もまた、動的コンテンツ資源に含まれ得る。コードレンダリングブロックは、ページレンダリング時間で実行処理される任意の量のコードを含むことができる。本発明のある実施形態において、コードレンダリングブロックは、ページレンダリング時に実行処理される１つの「レンダリング」メソッドで実行処理する。コードの他の部分は、そのレンダリングメソッドで実行処理され得る。コードレンダリングブロックは、以下のフォーマット（他のフォーマットも別の実施形態において考えられるが）を満たす。

【００３８】

```
<% InlineCode %>
```

ここで"InlineCode"は、ページレンダリング時にサーバ上で実行処理する独立言語型コードブロックまたは制御フローブロックを含んでいる。

【００３９】

インライン表現もまた、以下のような例示的な文法を用いて表現レンダリングブロックデリミッタ"<%@"と"%>"との間で用いることができる。

【００４０】

```
<%= InlineExpression %>
```

ここで、"InlineExpression"ブロックに含まれる表現は、"InlineExpression"からの値を宣言内の適切な場所のホールダに書き込むページオブジェクトの"Response.Write(InlineExpression)"への呼び出しによって最終的に包含される。例えば、"InlineExpression"は、以下のようにファイル４００に含まれ得る。

【００４１】

```
<font size = "<%=x%>" > Hi <%=Name%>, you are <%=Age%>! </font>
```

これは、値"x"に格納されるフォントで挨拶およびある人の年齢についての記述を出力する。その人の名前および年齢は、コード宣言ブロック（図示せず）においてストリングとして定義される。結果HTMLコードは、適切な場所に"InlineExpression"の値を含むようにサーバでレンダリングされHTTPレスポンスでクライアントへ送信される。すなわち、以下のようなものである。

【００４２】

```
<font size = "12" > Hi Bob, you are 35!
```

ファイル４００のライン１１において、<body>は、HTMLドキュメントの本文の始まりを規定するための標準HTMLタグである。フィイル４００のライン２０において、終了タグ</body>もまた示されている。本発明のある実施形態において、<body>および</body>のいずれもリテラルである。

【００４３】

HTMLフォームブロックの開始タグ<form>が、図４のファイル４００の本文セクション内、ライン１２に見られる。フォームブロックの終了タグ</form>は、HTMLファイル４００のライン１９に見られる。任意のパラメータ"id"もまた、所与の識別子をフォームブロックに関連付けるためにHTML制御タグに含むことができ、これによって、１つのHTMLファイルに多数のフォームブロックが含まれることが可能になる。

【００４４】

ファイル４００のライン１８において、「メッセージ」によって識別されたサーバ側ラベルが宣言される。「メッセージ」ラベルは、ウェブページ上にラベルを表示するために、ファイル４００のライン５および８で宣言されたコードで用いられる。

【００４５】

フォームブロック内に、図１のユーザインタフェース要素１０６、１０８および１１０に対応する３つのHTML制御タグ例が示されている。第１のユーザインタフェース要素がテキストボックスに相当するファイル４００のライン１３で宣言される。テキストリテラル"User Name"は、テキストボックスの左側に位置するラベルを宣言する。type="Text"を有する入力タグは、テキストボックスクライアント側ユーザインタフェース要素をレンダリングするサーバ側制御オブジェクトとして"UserName"という識別子を有するテキストボックスサーバ側制御オブジェクトを宣言する。ファイル４００のライン１５および１６は、それぞれ、図１のボタン１０８および１１０として示されるクライアント側ユーザインタフェース要素を宣言する。"OnServerClick"パラメータは、ファイル４００のコード宣言ブロックで宣言された適切なサブルーチンを特定する。そして、ファイル４００での宣言へのレスポンスで生成されたサーバ側ボタン制御オブジェクトが、クライアント側ボタンのHTMLコードおよびボタンクリックイベントを実行する関連のサーバ側コードをレンダリングする。

【００４６】

ファイル４００で宣言されたテキストボックスおよびボタンは、HTMLサーバ制御宣言の例である。初期状態では、ASP+リソース内のすべてのHTMLタグはリテラルテキストコンテンツとして取り扱われ、ページ開発者のプログラミングにおいてアクセスすることができない。しかし、ページ開発者は、"server"に設定された値を有する"runat"属性を用いて指定することによって、HTMLタグが構文解析され、アクセス可能サーバ制御宣言として扱われるべきであることを示すことができる。任意には、各サーバ側制御オブジェクトは、対応する制御オブジェクトのプログラム参照を可能にする一意の"id"属性と関連付けることができる。サーバ側制御オブジェクト上のプロパティ引数およびイベント結合もまた、タグ要素上の宣言名／値属性対を用いて特定することができる（例えば、OnServerClickは"MyButton#Click"対に等しい）。

【００４７】

本発明のある実施形態において、HTML制御オブジェクトを宣言する一般的な文法は以下のとおりである。

【００４８】

```
    <HTMLTag id = "Optional Name" runat = server>
  ..................................
  </HTMLTag>
```

ここで、"Optional Name"は、サーバ側制御オブジェクトの一意の識別子である。サポートされ得るHTMLタグのリストならびに関連文法およびCOM+クラスを表１に示すが、他のHTMLタグも本発明の範囲内で考えることができる。

【００４９】

【表１】

HTML タグ名	例	COM+ クラス
<a>	 My Link 	AnchorButton
<img>	<img id = “MyImage” runat = server>	Image
		Label
	Some contents	Panel
<form>	<form id = “MyForm” runat = server> </form>	FormControl
<select>	<select id = “MyList” runat = server> <option>One</option> <option>Two</option> <option>Three</option> </select>	DropDownList
<input type = file>	<input id = “MyFile” type = file runat = server>	FileInput
<input type = text>	<input id = “MyTextBox” type = text>	TextBox
<input type = password>	<input id = “MyPassword” type = password>	TextBox
<input type = reset>	<input id = “MyReset” type = reset>	Button
<input type = radio>	<input id = “MyRadioButton” type = radio runat = server>	RadioButton
<input type = checkbox>	<input id = “MyCheck” type = checkbox runat = server>	CheckBox
<input type = hidden>	<input id = “MyHidden” type = hidden runat = server>	HiddenField
<input type = image>	<input type = image src = “foo.jpg” runat = server>	ImageButton
<input type = submit>	<input type = submit runat = server>	Button
<input type = button>	<input type = button runat = server>	Button
<button>	<button id = MyButton runat = server>	Button
<textarea>	<textarea id = “MyText” runat = server> This is some sample text </textarea>	TextArea

【００５０】

標準HTML制御タグに加えて、本発明のある実施形態によって、開発者は、HTMLタグセットの外側に共通のプログラム機能についてカプセル化する再利用可能コンポーネントを作成することが可能になる。これらのカスタムサーバ側制御オブジェクトは、ページファイル内の宣言タグを用いて特定される。カスタムサーバ側制御オブジェクト宣言は、"server"に設定された値を有する"runat"属性を含む。任意には、カスタム制御オブジェクトのプログラム参照を可能するために、一意の"id"属性が特定される。さらに、タグ要素における属性対である宣言名／値は、サーバ側制御オブジェクトのプロパティ引数およびイベント結合を特定する。インラインテンプレートパラメータもまた、適切な「テンプレート」の接頭文字列である子の要素を親サーバ制御オブジェクトに与えることによってサーバ側制御オブジェクトに結合されることがある。カスタムサーバ側制御オブジェクト宣言のフォーマットは、次のようになる。

【００５１】

<servercntrlclassname id="OptionalName" [propertyname="propval"] runat=server/>

ここで、"servercntrlclassname"は、アクセス可能制御クラスであり、"OptionalName"は、サーバ側制御オブジェクトの一意の識別子であり、"propval"は、制御オブジェクトの任意のプロパティ値を表す。

【００５２】

別の宣言文法を用いて、XMLタグ接頭語は、以下のフォーマットを用いることにより、ページ内にサーバ側制御オブジェクトを特定するためのより簡潔な表記法を提供するのに用いることができる。

【００５３】

<tagprefix:classname id = "OptionalName" runat = server/>

ここで、"tagprefix"は、所与の制御名スペースライブラリと関連し、"classname"は、関連名スペースライブラリでの制御の名前を表す。任意の"propertyvalue"もまたサポートされている。

【００５４】

図５を参照すると、本発明の実施形態のコンピュータシステムの一例は、プロセッサユニット５０２、システムメモリ５０４およびシステムメモリ５０４を含む種々のシステムコンポーネントをプロセッサユニット５００に接続するシステムバス５０６を含んでいる従来のコンピュータシステム５００という形態の汎用コンピュータ装置を含んでいる。システムバス５０６は、メモリバスまたはメモリコントローラ、種々のバスアーキテクチャを用いるペリフェラルバスおよびローカルバスを含む幾つかのタイプのバス構造のいずれであってもよい。システムメモリは、再生専用メモリ（ROM）５０８およびランダムアクセスメモリ（RAM）５１０を含んでいる。コンピュータシステム５００内の要素間での情報の転送を助ける基本ルーチンを含んでいる基本入力／出力システム５１２（BIOS）は、ROM５０８に格納されている。

【００５５】

コンピュータシステム５００は、さらに、ハードディスクの読み出しおよび書き込みを行うハードディスクドライブ５１２、着脱可能な磁気ディスク５１６の読出しおよび書き込みを行う磁気ディスクドライブ５１４およびCD ROM、DVDまたは他の光学媒体のような着脱可能な光ディスク５１９の読出しおよび書き込みを行う光ディスクドライブ５１８を含んでいる。ハードディスクドライブ５１２、磁気ディスクドライブ５１４および光ディスクドライブ５１８は、それぞれ、ハードディスクドライブインタフェース５２０、磁気ディスクドライブインタフェース５２２および光ディスクドライブインタフェース５２４によってシステムバス５０６接続されている。ドライブおよびその関連コンピュータ読み取り可能媒体が、コンピュータシステム５００のコンピュータ読み取り可能命令、データ構造、プログラムおよび他のデータの不揮発性記憶部を提供している。

【００５６】

本明細書に記載の上記環境例では、ハードディスク、着脱可能な磁気ディスク５１６および着脱可能な光ディスク５１９を用いているが、データ保存可能な他のタイプのコンピュータ読み取り可能媒体を上記システム例に用いることができる。上記動作環境例に用いることができるこれらの他のタイプのコンピュータ読み取り可能媒体は、例えば、磁気カセット、フラッシュメモリカード、デジタルビデオディスク、ベルヌイ(Bernoulli)カートリッジ、ランダムアクセスメモリ（RAM）および再生専用メモリ（ROM）などがある。

【００５７】

多数のプログラムモジュールが、ハードディスク、磁気ディスク５１６、光ディスク５１９、ROM５０８またはRAM５１０に格納され、これらは、オペレーティングシステム５２６、１つ以上のアプリケーションプログラム５２８、他のプログラムモジュール５３０およびプログラムデータ５３２を含む。ユーザは、コマンドおよび情報をコンピュータシステム５００にキーボード５３４およびマウス５３６または他のポインティング装置などの入力装置によって入力することができる。他の入力装置としては、例えば、マイクロフォ

ン、ジョイステック、ゲームパッド、サテライトディッシュおよびスキャナなどがある。これらおよび他の入力装置は、システムバス５０６に接続されているシリアルポートインタフェース５４０を介して処理装置５０２に接続されていることが多い。しかし、これらの入力装置はまた、パラレルポート、ゲームポートまたはユニバーサルシリアルバス（USB）などの他のインタフェースによって接続されていてもよい。モニタ５４２または他のタイプの表示装置もまた、ビデオアダプタ５４４などのインタフェースを介してシステムバス５０６と接続している。モニタ５４２に加えて、コンピュータシステムは、典型的には、スピーカおよびプリンタなどの他の周辺出力装置（図示せず）を含む。

【００５８】

コンピュータシステム５００は、リモートコンピュータ５４６のような１つ以上のリモートコンピュータへの論理接続を用いたネットワーク化された環境で動作することができる。リモートコンピュータ５４６は、コンピュータシステム、サーバ、ルータ、ネットワークPC、ピア（peer）装置、または他の共通ネットワークノードであり得、典型的にコンピュータシステム５００との関連で上述した要素の多くまたはすべてを含む。ネットワーク接続は、ローカルエリアネットワーク（LAN）５４８およびワイドエリアネットワーク（WAN）５５０を含む。このようなネットワーク環境は、オフィス、企業規模コンピュータネットワーク、イントラネットおよびインターネットにおいて珍しいものではない。

【００５９】

LANネットワーク環境で用いるとき、コンピュータシステム５００は、ネットワークインタフェースまたはアダプタ５５２を介してローカルネットワーク５４８に接続される。WANネットワーク環境で用いるとき、コンピュータシステム５００は、典型的に、インターネットのようなワイドエリアネットワーク５５０による通信を確立するためのモデム５５４または他の手段を含む。モデム５５４は内蔵または外付けのいずれでもよく、シリアルポートインタフェース５４０を介してシステムバス５０６と接続されている。ネットワーク化された環境において、コンピュータシステム５００に関連して述べたプログラムモジュールまたはその一部は、リモートメモリ記憶装置に記憶されてもよい。図示されたネットワーク接続は例であって、コンピュータ間の通信リンク確立のために他の手段を用いることができる。

【００６０】

本発明の実施形態において、コンピュータ５００は、ウェブサーバを表し、CPU５０２が、記憶媒体５１６、５１２、５１４、５１８、５１９またはメモリ５０４のうち少なくとも１つに記憶されたASP+ファイル上でページファクトリモジュールを実行処理する。HTTPレスポンスおよびリクエストは、クライアントコンピュータ５４６に接続されたLAN５４８により通信される。

【００６１】

ページファクトリモジュール３０８によって行われる処理６００を図６のフローチャートに示す。図６では、処理６００は、制御オブジェクト階層２０４の生成処理（図２）にほぼ対応する。処理６００は、ASP+ページ（ASP＋ファイルともいう）をコンパイルされたオブジェクトコードクラスに変換し、次いで、それは、制御オブジェクト３１４、３１８、３２０および３２２（図３）のインスタンスを作成するためにメモリにロードされる。

【００６２】

サーバが処理２０２においてURLに対するリクエストを受信すると（図２）、ページ作成処理６００が始まる。リクエストの受信後、構文解析処理６０２が、要求されたURLの構文解析を行い、どの資源が要求されているかを判断する。資源は、静的なファイル（.htm、.gif、.jpgなど）、ディレクトリブラウジング起動、DAV（デジタルオーディオ／ビデオ）ファイルおよび動的なコンテンツリクエスト（.aspx、.soapなど）を含み、本発明は、これらをハンドリングするように構成されている。サーバ側によって受信された、入ってくるHTTPリクエストは、ハンドラクラス、例えば、IHTTPHandlerクラスの特定のインスタンスによって最終的に処理される。ハンドラファクトリ、例えば、IHTTPHandler"フ

ァクトリ"の使用によって、ハンドラインスタンス、例えば、IHTTPHandlerインスタンスへのURLリクエストの解釈を実際に行う差込可能なアーキテクチャが提供される。この解釈は、入ってくるURLのファイル拡張およびHTTPコマンドを、適切なハンドラインスタンス、例えばIHTTPHandlerインスタンスを最終的に作成することになっているIHTTPHandler Factoryクラスのようなファクトリクラスへ写像することができるアプリケーション構成設定を用いて容易に行うことができる。構文解析処理において、差込可能アーキテクチャにより種々の資源を識別できるが、以下の記載は、動的ウェブページコンテンツ資源を識別するHTTPリクエストに焦点をあて、特に、.aspxまたはASP+ページもしくはファイルのような動的ウェブページコンテンツファイルに焦点を当てている。

【００６３】

構文解析処理６０２で実際の資源が識別されると、チェック処理６０３において、メモリをチェックし、クラスがメモリに存在するかどうかを判断する。クラスがコンパイルされているかどうかを判断するために、チェック処理６０３において、クラスがコンパイルされるとセットされるフラグを検索することによって、または名前によってメモリ内のクラスを検索する。どちらにしても、チェック処理６０３において、コンパイルクラスがすでにコンパイルされ格納されていることを示す指標を検索する。クラスがメモリにある場合、フローはYES枝に分岐しインスタンス作成処理６１２に進む。インスタンス作成処理は、コンパイルされたクラスから制御オブジェクトのインスタンスを作成する。チェック処理６０３において、クラスがコンパイルされていないと判断されると、フローはNO枝に分岐し位置決定処理６０４に進む。

【００６４】

チェック処理６０３が、クラスはメモリに存在しないと判断すると、位置決定処理６０４において、特定の資源を検索し探し当て、ファイルをメモリに読み出す。"System．ASP WebForms．PageFactory"クラスのようなベースクラスが、ASPおよびASPXページのインスタンス作成および構成をハンドリングするハンドラファクトリインプリメンテーションを提供する。資源、この場合は物理的な動的なウェブページコンテンツファイルが見つからなければ、適切なエラーメッセージが戻される。

【００６５】

位置決定処理６０４の次に、処理６００において、構文解析作成処理６０６を行い、ここでASP+ファイルのような資源の構文解析を行う。構文解析／作成処理６０６は、ASP+ファイルを宣言ごとに読み出し、構文解析しながら集めた情報からデータモデルを作成する。データモデルは、ソースコードを引き出すことができるアクティブコンテンツファイルの要素に関連する要素を含むデータ構造である。データモデルは、アクティブコンテンツファイルで参照された構造的要素を含み、これらの要素は、アクティブサーバページの結果制御オブジェクトの構造を表すように連結される。本発明のある実施形態において、データモデルは、階層的なツリー構造で関連付けられたオブジェクトの組み合わせである。

【００６６】

ウェブコンテンツファイルの各宣言は、データモデルの作成方法およびデータモデルに格納される情報を示す所定の要素を有している。例えば、宣言がリテラルテキスト宣言である場合、データモデルは、テキストが適切な位置に挿入されるような情報を含むだけでよい。しかし、宣言が入れ子の制御オブジェクトが存在することを示す場合、ASP＋ファイルに宣言されたような入れ子に従ったデータモデルが作成されなければならない。本質的に、データモデルは、各宣言ごとに１つのオブジェクトを含み、各オブジェクトは、各オブジェクトに子オブジェクトが存在するかどうかに関する情報を含む。

【００６７】

ASP＋ファイルが構文解析され、データモデルが作成されると、任意である作成処理６０７を行ってもよい。作成処理６０７は、以下に記載するように、データモデルを解析し、例えば、作成すべきソースコードファイルのようなソースコードを抽象化したものである中間データ構造を作成する。本質的に、中間データ構造は、コードを記述する包括的なデータ構造である。中間データ構造は、クラスを記述する上位レベルのデータ構造を有す

る。各クラスごとに、クラス名およびメソッドのリストまたはアレイを有する別のレベルのデータ構造であってもよい。さらに、各メソッド自体、ステートメントおよび宣言のような種々の要素を有するデータ構造である。ステートメントは、表現、データ呼び出しなどのような項目を含み得る。

【００６８】

ASP+ファイルは構文解析され、データモデル（および任意には、中間データ構造）が作成されると、作成ソースコードモジュール６０８は、典型的には、ASP+ファイルに存在する各宣言のコードの種々のラインを書き込むことにより、各データモデルの解析を通して必要なソースコードを作成する。一般に、このステップは、データモデルにおける公知の宣言および他の情報のコードへの直接的な翻訳を伴う。このような翻訳は、翻訳を行うアプリケーションへハードコード化されるか、または、各翻訳は、検索テーブルに格納され得る。この種の検索テーブルは、ほとんどすべての言語にとって適切な翻訳を提供するよう作成され得るが、オブジェクト指向ソースコード言語の使用により、既存のCOMまたはCOM+ライブラリを利用するためのCOMまたはCOM+クラスへのコンパイルが簡単になる。（例えば、ソースコード言語の一例として、C++またはVbasicなどがある。）さらに、中間データ構造が処理６０７で作成される場合は、翻訳はさらに直接的になる。このような場合、データ構造は、包括的なソースコード言語ファイルに非常に近くなって、翻訳は、単に、特定のソースコード言語のソースコードファイルを作成するのに適切な辞典および文法の追加になるにすぎない場合がある。

【００６９】

ASP+ファイルは、結果ソースコードファイルに特定のソースコード言語を宣言する。ASP+ファイルに言語が宣言されていない場合、Visual Basicなどのデフォルト言語を用いることができる。データモデルまたは中間データ構造から生成された結果ソースコードファイルは、本質的に、プログラマがサーバ側制御オブジェクト階層を利用する場合で、このコード生成ツールの恩恵を受けないときに書き込むことを要求されることになるタイプの高度なソースコードファイルである。

【００７０】

ソースコードファイルが完成すると、コンパイル処理６１４において、ソースコードファイルをコンパイルし、オブジェクトコードまたは他の実行処理可能フォームでコンパイルされたクラスを作成する。あるいは、クラスは、バイトコードまたは仮想装置上で起動するできる他の言語にコンパイルすることができる。ソースコードファイルのコンパイルは、ソースコードファイルに存在するソースコード言語をコンパイルするように構成されたコンパイラの使用を伴う。そして、クラスが、インスタンス作成処理６１６によって呼び出され、ウェブページのためのページオブジェクトおよび結果制御オブジェクトを作成する。URLリクエストに明示的に備わっていないが、インスタンス作成処理は、特定の資源のURLリクエストの一部をなす非明示的処理である。

【００７１】

ASP+ファイルがページクラスにコンパイルされると、クラスは、将来のリクエストで使用可能な状態で維持される。したがって、ASP+ファイルのソースコードの生成処理は、１度行われるだけでよく、後のリクエストはコンパイルされたクラスを利用することができ、必要なオブジェクトを必要に応じてインスタンス作成する。コンパイルされたクラスは、短期間の間キャッシュされ得、かつ／またはコンパイルされたクラスはディスクに格納し得る。実際、ASP+ファイルが一旦コンパイルされると、元々のASP+ファイルを再び触れる必要がない。もちろん、ASP+ファイルが修正される場合は、コンパイル処理を繰り返すことによってページクラスを更新する。適切なフラグまたは他の指標メカニズムを用いてASP+ファイルが更新されたかどうかを判断することができ、したがって自動クラス更新が可能である。あるいは、ウェブページファイルまでASP+を更新した後、キャッシュされたクラスを除去するという責任は開発者に残され得る。

【００７２】

データモデルの作成に関連した詳細を図７に示す。データモデル作成処理７００は、テ

スト処理７０２でASP+ファイルにおいて宣言があるかどうかを判断することから始まる。純粋なHTMLファイルのようなコンパイルすべき宣言がない場合、フローは、NO枝に分岐し処理７１４の終了に進む。

【００７３】

テスト処理７０２でASP+ファイルに少なくとも１つの宣言があると判断されると、フローはYES枝に分岐し、ASP+ファイルで第１の宣言を得る獲得処理７０４に進む。第１の宣言からの情報は、格納処理７０６によってデータ構造に格納される。データ構造に格納された実際の宣言のみならず、宣言が子オブジェクトを有するコンテナタイプの制御を宣言するかどうか、または宣言がディレクティブなどを宣言するかどうかなどの第１の宣言に関連した情報もデータ構造に格納され得る。ウェブページは、ASP+ファイルであるので、典型的に第１の宣言としてディレクティブタグを有する。ディレクティブタグは、種々のディレクティブ、例えば図４のライン１に示すようなデリミッタ"<%@"と"%>"との間の情報を含む。図４に示すデリミッタは、単に例示的であり他の選択も可能であることを理解すべきである。ディレクティブは、ソースコードファイル作成に用いられるページファクトリモジュールに情報を供給する。例えば、図４のライン１に示すディレクティブは、用いられるべきデフォルト言語が、この例では、VB（すなわち、Visual Basic）であることを示している。

【００７４】

第１の宣言が取り出されデータ構造に格納されると、処理７００は、ファイルに別の宣言があるかどうかを判断するテスト処理７０８に進む。この処理は、上記の処理７０２と同様である。もはや宣言がなければ、フローはNO枝に分岐し終了ステップ７１４に進む。別の宣言があると、フローはYES枝に分岐し、獲得処理７１０に進み、次の宣言を獲得する。

【００７５】

次の宣言の取り出しの次に、格納処理７０８は、データモデルの他のデータ構造に関連付けられた異なるデータ構造に次の宣言に関連する情報を格納する。第１の宣言に関して上で述べたように、データ構造に格納された情報は、次の宣言のリテラルテキストのみならず、リテラルテキストから派生した情報をも含み得る。したがって、データ構造は、必要に応じて階層的に第１のデータ構造に関連付けられ得る。

【００７６】

次のデータ構造への情報の格納の次に、フローはテスト処理７０８に分岐し、資源ファイル、例えば、ASP+ファイルにデータモデルに読み込むべき宣言がまだ存在するかどうかを判断する。存在しない場合は、フローは、NO枝に分岐し処理の終了に進む。しかし。テスト処理７０８がまだ宣言が存在すると判断すると、フローはYES枝に分岐し、次の宣言を獲得する獲得処理７１０に進む。獲得処理７１０の次に、格納処理７０８において、上記のように、次の宣言に関連した情報を格納する。これらのステップ７０８、７１０および７１２は、すべての宣言が順次取り出され、データ構造に格納されるまで続く。データ構造は、入れ子のオブジェクト、子オブジェクト、ノードまたは親オブジェクトのサブ要素は、それらがサブ要素として識別できるように連結される階層として関連付けられてもよい。

【００７７】

ソースコードモジュール６０８（図６）の作成には、このデータモデルを用いて、特定のソースコード言語のソースコードファイルを作成する。データモデルからのソースコードファイルの作成処理は、ソースコード言語に依存することに注目することが重要である。すなわち、特定の言語で書き込まれたソースコードファイルは、典型的に、特定のフォーマット要件、例えば、すべての可変宣言がこれらの変数の使用に先立つという要件を有する。したがって、その言語のコンパイラが呼び出されると、適切なプログラム命令がソースコードファイル内の適切な位置に配置されなければならない。この作業をなし遂げるために、ASP+ファイルを評価して、ファイルの中にどんな一意の要素または特徴があるかを判断しなければならないし、またファイルを処理してソースコードファイルを生成しな

ければならない。ソースコードファイルは、典型的に、第１の一組の要素または宣言が可変宣言のようなファイルの始めにあることを要求するので、データモデル全体が、この情報のために解析されるべきである。同様に、制御オブジェクト情報は、ファイルの中央にあるべきなので、処理の第１フェーズの次の第２フェーズ時にこの情報のためにデータモデル全体を解析すべきである。

【００７８】

これらのフェーズは、ソースコードファイル（または処理６０７での中間データ構造）に書き込まれるべきコードの別個の部分を判断する機能的処理であり、ここでは、別個の部分は、ソースコードファイル内の結果位置に基づき論理的に組み合わされる。したがって、これらのフェーズは、１つのデータモデルの「パス」またはトラバースであると考えられ、したがって連続的に行われる。しかし、あるいは、これらのフェーズは、別個の並行した処理として行われてもよく、別個の処理の結果は１つのソースコードにまとめられ、処理が単一データモデルまたはデータモデルの別個のコピーのいずれかを評価する。あるいは、コードの別個の部分を判断し書き込む機能的な処理またはフェーズは、単一のトラバース時に行われてもよく、そこで、ソースコードは、ソースコードファイルの別個の部分に選択的に書き込まれたり、その部分が、ソースコードの論理的部分間の適切な連結を提供するようにまとめられるか、関連付けられた別個のファイルに書き込まれたりする。したがって、本明細書中では、別個の解析を、実質的に連続して、すなわち次々に行われるように記載しているが、解析は実質的に同時にスプリットされたり、実行されてもよい。

【００７９】

図８に示す処理またはフローは、データモデルから比較的直接ソースコードファイルを生成する実施形態、すなわち図６に示す省略処理６０７に関する。本実施形態は、変数が始めに宣言され、オブジェクトおよびメソッドが変数情報に続くVBコードファイルのようなソースコードファイルを生成する。図８に示すソースコード作成処理８００は、データモデルの「パス」として別個のフェーズを示し、データモデルのデータ構造を通過する３つのパスになっている。第１のパスの処理において、変数および宣言情報を探している。第２のパスの処理において、オブジェクト作成情報を探し、第３のパス処理８００において、コードレンダリング情報を探している。

【００８０】

第１のパスは、変数および他の宣言情報についてデータモデルをトラバースする処理８０２で始まる。このステップは、処理７００で作成されたデータモデルの第１のデータ構造を解析することから始まる。第１のデータ構造の解析処理において、宣言に関するソースコードファイルに書き込まれるべきソースコードの特定ラインがあるかどうか判断する。

【００８１】

第１のデータ構造の解析が終了すると、処理８０４において、ソースコードファイルに変数および宣言情報に関するソースコードを生成し書き込む。ライタオブジェクトが呼び出され、それに与えられたテキストをソースコードファイルに単に書き込む。したがって、ライタオブジェクトへの呼び出しは、ソースコードファイル名のパラメータと変数または宣言情報に関する特定のプログラミング言語の文法のテキストストリングとの両方を含む。あるいは、テキストは、コピーされるか、または他の公知の手段を用いてソースコードファイルに書き込まれる。

【００８２】

また、処理８０４において、ライタオブジェクトへ渡すべき適切な文法を決定しなければならない。本質的に、現在のデータ構造が、サーバ側制御を表す場合は、処理８０４において、それに対するソースコードを生成する。そうでなければ、処理８０４において、それをソースコードファイルにコピーする。本質的に、処理８０４は、３つのうちの１つを行い得る。生成処理８０４において、まず、ASP+ファイルにおける情報がリテラルテキストであるかどうか、したがってソースコードファイルに直接書き込まれるべきかどうか

を判断してもよい。これは、HTMLタグがASP+ファイルに挿入される場合であり、この場合、ソースコード宣言は生成されない。その代わりに、情報は単に資源ファイルにコピーされる。処理８０４において、第２に、テキストが単純な翻訳、特定のソースコード言語の適切な文法フォームで１対１の対応を要求するかどうかを判断してもよい。このような処理８０４において、１対１対応を探すか、または、ページファクトリモジュールにハードコードされた場合は１対１翻訳を行い、結果情報をライタオブジェクトに供給する。第３に、１対１翻訳ができない場合は、処理８０４において、より複雑な翻訳が要求され、したがってモジュールまたは次の検索テーブル処理を呼び出し、ライタオブジェクトに渡されるべき適切なソースコード文法の作成処理を行う。典型的に、宣言情報の翻訳は直接的である。

【００８３】

処理８０４において、第１のデータ構造のソースコードの生成および書き込みを行うと、テスト処理８０６において、データモデルに解析すべきデータ構造がまだ存在するかどうかを検知する。データモデルのこの第１のパス時に解析すべきデータ構造がまだある場合は、フローはYES枝に分岐して処理８００の始めに戻り、８０２において次のデータ構造に対してトラバースを行う。このような場合、次のデータ構造は、上記のように解析されソースコード文法を生成し、その文法コードを資源ファイルに書き込むライタオブジェクトを呼び出す。

【００８４】

テスト処理８０６において、この第１のパス時にデータモデルに解析すべきデータ構造がもう存在しないと判断した場合は、フローはNO枝に分岐してトラバース処理８０８に進み、データ構造を通過する第２のパスを開始する。トラバース処理８０８は、典型的にオブジェクト製作およびメソッド作成を書き込むコードを書き込むという次のフェーズを開始する。トラバース処理８０８は、データモデルの上から始まり、データモデルの始めのデータ構造に戻る。

【００８５】

次に、生成および書き込みソースコード処理８１０において第１のデータ構造を解析し、オブジェクト作成に関する情報がその第１のデータ構造にあるかどうかを判断する。このようなオブジェクト作成情報が第１のデータ構造に存在する場合、特定のプログラミング言語の文法が作成および生成されてソースコードファイルに書き込むライタオブジェクトに送信される。本質的に、処理８１０は、上記の処理８０４と同様であり、ここでは、データ構造が特定タイプの情報について解析され、その情報がデータ構造から収集されると、何らかの翻訳を行い、その情報に関するソースコード言語を作成する。情報がリテラルである場合、処理８１０において、その情報をソースコードファイルに直接送ることができる。同様に、その情報が簡単な翻訳を必要とする場合は、その簡単な翻訳は、プログラムにハードコードされるか、あるいは検索テーブルタイプ処理の一部であってもよい。また、さらに複雑な構造が要求される場合は、検索テーブルまたは他のモジュールを呼び出し、情報を処理し、ソースコードファイルに適切な文法ソースコードを作成する。

【００８６】

処理８１０の次に、照会ステップ８１２は、データモデルに解析すべきデータ構造が存在するかどうかをテストする。オブジェクト作成のために解析すべきデータ構造がデータモデルに存在する場合、フローはYES枝に分岐し次のデータ構造をトラバースするトラバース処理８０８に進む。したがって、処理８０８において、次のデータ構造に戻り、処理８１０において、オブジェクト作成に関する情報のデータ構造を解析する。このオブジェクト作成情報は、上記のように、ライタオブジェクトに送信され、ソースコードファイルに追加することができるソースコードラインに翻訳される。ステップ８１２、８０８および８１０は、すべてのソースコードがオブジェクト作成に関し書き込まれるまで繰り返される。

【００８７】

判断ステップ８１２が、オブジェクト作成のために解析すべきデータ構造がデータモデ

ルにもう存在しないと判断すると、フローはNO枝に分岐し処理８１４に進み、第２のパスを終了する。この第３のパスは、データモデルのトップの処理８１４で始まる。処理８０８および８０２に関して上で述べたように、このトラバースステップは、まず、データモデル内の第１のデータ構造を取り出す。処理８１４でのデータモデル内の第１のデータ構造の取り出しの次に、処理８１６で、コードレンダリング情報のデータ構造を解析する。コードレンダリング情報は、ライタに送信されたり、ソースコードファイルに追加させるプログラミング言語での特定な文法に翻訳される。コードレンダリングメソッドは、ソースコードファイルの終わり近くにあり、したがって、第３のパスがオブジェクト作成パスの次に行われる。あるいは、コードレンダリングメソッドが同時スレッド処理に間に生成され、資源ファイルの終わりに追加される。

【００８８】

処理８１６は、上記の処理８１０および８０４とかなり似ており、このソースコード情報は、データ構造にあるコードレンダリング情報から収集され得る。さらに、上記のように、情報は直接挿入されてもよく、１対１の割合で翻訳されてもいいし、より複雑なモジュールを用いてソースコードファイルの文法を生成することを要求してもよい。処理８１６の次に、検知処理８１８において、データモデルにデータ構造が残っていないかどうか判断する。

【００８９】

処理８１８において、この第３のパス間に解析すべきデータ構造がまだ存在すると検知すると、フローはYES枝に分岐し、次のデータ構造のトラバースのために８１４に戻る。次のデータ構造が呼び出され、処理８１８において、次のデータ構造のコードレンダリング情報に関するソースコードの生成および書き込みを行う。

【００９０】

除リ８１８において、解析すべきデータ構造がもはや存在しないと判断すると、フローはNO枝に分岐し、処理連結８２０で処理を終了する。

【００９１】

上記の説明により明らかなように、数回のパスがデータモデルに対して行われる。これは、データモデルの各宣言が、ソースコードファイルの種々の場所に置かれるコードの特定のラインを要求することによる。しかし、ソースコードファイルを書き込む際、ライタオブジェクトは、ファイルに連続的に追加して順次書き込むことしかできず、予め書き込まれたコードライン間にソースコードラインを挿入できない。したがって、データモデルは、ソースコードの第１または上部、ソースコードの第２または中間部、およびソースコードファイルの第３または最後の部分に属する適切な情報を収集するために１回以上トラバースしなければならない。パスの数は、種々のコンピュータ言語に対し存在するコードの部分の数によって異なることが予想される。

【００９２】

図９に示す処理またはフローは、データモデルから中間データ構造を生成し、次いで、中間データ構造からソースコードファイルを生成する実施形態、すなわち図６の処理６０７および６０８の別の実施形態に関する。本実施形態のフロー９００は、処理８０４、８１０および８１６が特定の言語のソースコードを生成し、処理９０４、９１０および９１６が、ソースコードを生成するのではなくて、中間データ構造の部分を生成することを除いて図８のフローと同様である。したがって、処理９０４、９１０および９１６が、データモデルでの情報に関する情報を生成する際に処理８０４、８１０および８１６と同様の処理を行う。しかし、生成された情報は包括的であり、後に異なるソースコード言語のソースコードファイルを作成するのに用いられることがある。

【００９３】

データモデルに解析すべきデータ構造がもはや存在しないと判断されると、中間データ構造が完成し、フローは分岐して作成処理９２０に進む。作成処理において、中間データ構造からソースコードファイルを作成する。データ構造はソースコードファイルの包括的な記述であるので、処理９２０において中間データ構造を包括的な記述から特定の言語コ

ードファイルに翻訳する。

【００９４】

図１０は、ウェブページ開発者によって製作されたウェブページ資源ファイル、すなわちASP+ファイルの例である。ファイルは、ソースコードファイルを生成するための処理６００のサブジェクトである。図１０に示すASP+ページは、ライン１にディレクティブライン、ライン３から１３にサーバ側スクリプトブロックおよびライン１６から２１にサーバ側コントロール宣言ブロックを有する。ディレクティブラインは、どのようなタイプのソースコード言語を用い、それに一般的な記述を与える（そうでなければソースコードファイルのどのコードにもならない）かを判断するために処理６００において用いられる。コード宣言ブロック"<script runat=server>　　</script>"内に書かれたすべてのコードは、概念的にページメンバ宣言（変数、プロパティ、メソッド）として扱われ、図１１に示すような生成ファイルの資源ファイルに直接挿入される。

【００９５】

図１１は、図１０に示すASP+ファイルを用いてソースコードを作成する処理によって作成されたソースコードファイルの一例である。図１１の第１のラインは、ソースコードファイルによって作成しようとしているページクラスは、コンパイルされたとき、System.ASP.WebForms.Pageから引き継いでいることを示している。このクラスからの引継ぎは、ページクラスの多くの制御機能を提供するので重要なステップである。初期状態として、生成された資源ファイルは"System.ASP.WebForms.Page"ベースクラスのサブクラスである。任意には、開発者は、ディレクティブラインに与えられた"Inherits"属性を用いて代替クラスを特定することができる。

【００９６】

ライン３および４は、第１のパスでデータモデルを通っている間に作成された可変宣言を示している。

【００９７】

第２のパス時にライン６に示された情報が作成され、新しい制御オブジェクト"DataList"を作成し、それをMyDataと名づける。ある実施形態において、ASP+ファイル宣言ブロック"<script runat=server>　　</script>"内に書き込まれているすべてのコードは、ページメンバ宣言（変数、プロパティ、メソッド）として扱われ、上記のような生成ファイルの資源ファイルに直接挿入される。このように、コード８－１３のラインは、リテラルテキストとして直接挿入された。データモデル６０６の作成の間、リテラルテキストに関連する情報は、多数のパスの間トラバースされないように、本質的にデータモデルの上部または離れて、ストリングまたはアレイで格納された（図８）。データモデルの作成時に、この情報は直接挿入されると判断され、データモデルにリファレンスポインタを有するアレイとしてそれを格納することができる。

【００９８】

ライン１５～３９は、制御オブジェクト情報構築に関する第２のパスの間に、コードの１セクションが書き込まれた。ライン１５～２０は、トップレベルのオブジェクトの作成において用いられるコードを表す。トップレベルのオブジェクトは、図３に示すようなページオブジェクト３１４のようなページ全体のコンテナタイプのオブジェクトである。この制御オブジェクトは、作成されたASP+ページはこのタイプの制御オブジェクトを有するので、図１０に示す実質的なコードから比較的独立して構築される。

【００９９】

図１１のライン２２～２７は、MyListと名づけられた子制御オブジェクトの作成に用いられるコードを表す。図１０のライン１６に示すように、テーブルリスト制御が定義され、識別子"MyList"を与えられた。図１０のライン１６からの情報は、ステップ８１０(図８)において図１１に示されるライン２２～２７に翻訳される。本質的に、処理８１０において、図１０のライン１６のコード部分の構文解析すると、"tablelist"タイプの制御を作成すべきであると認識される。認識されると、図１１に示されるライン２２～２７の情報は、テーブルから検索するか、またはライン２２～２７に示されたコードを生成する

ための"MyList"のような適切な変数を挿入して公知の計算を用いて生成される。システムは、このタイプの制御を認識するように設計されているので、図１１に示されたコードを生成することができる。一旦、生成されると、処理８１０は、そのコードもファイルに書き込む。

【０１００】

同様に、図１１のライン２９～３４は、別の制御オブジェクト、この場合は、"template"タイプのオブジェクトを表す。テンプレートは、それらもまたコンテナ制御であり、実行時間に実際に生成されるという点で特別なオブジェクトである。いずれにしても、テンプレートの制御オブジェクトの生成に必要なコードがステップ８１０（モデルにまだデータ構造が存在するかどうかを判断するテスト８１２に引き続いて）において生成される。上記のように、テンプレート制御オブジェクトが生成されるべきであり、図１１に示すコードが生成されファイルに書き込まれているという認識がなされる。重要なことは、図１１のライン３２および３３の"control3"へのリファレンスによって明らかなように、テンプレートのコードは、子制御に関連する情報を必要とする。したがって、データモデルが６０６(図６)で作成されると、この情報またはこの情報を決定する指標が、テンプレート制御情報とともに格納された。また、コードは、コードの生成が直接的であるようにテンプレート制御オブジェクトに対して予め決められている。

【０１０１】

子制御が、図１０のライン１９のコードおよび図１１のライン３６～３９でのそれに対応する翻訳されたコードを用いてテンプレート内に作成される。図１０のライン１９は、識別子"MyLabel"を有する"label"タイプの制御を呼び出す。テーブルリスト制御およびテンプレート制御と同様に、ラベル制御は、処理８１０が認識する公知のタイプの制御であり、次いで適切なコードを生成し、それを図１１のライン３６～３９に示すようなファイルへ書き込むことができる。

【０１０２】

データモデルの最後のパス時にライン４１～５１が作成および生成され、ソースコードファイルに書き込まれた。ライン４１～５１におけるコードのラインは、クライアントへのレスポンスの一部になるHTMLコードをレンダリングするために呼び出されるレンダリングメソッドを表している。レンダリングコードは、ステップ８１６でのこのようなコードを生成すべきであるという認識に基づき生成される。一旦認識されると、コードは単に検索テーブルから判断されるか、または必要に応じて生成される。

【０１０３】

図１１に示すコードは、特定の動的ウェブページファイル、すなわち図１０に示すファイルからのサーバ側生成コードを表している。図１１に示すファイルが完成すると、ファイルは、図６に示す処理６１０に関して上に述べたようにコンパイルされてもよい。コンパイルが、動的ウェブページコンテンツファイルの制御オブジェクト制御を生成するのに用いることができるクラスとなる。クラスは、キャッシュ、または他のメモリに格納され、ページのためのオブジェクトのインスタンスを作成するために所望の回数用いることができる。結果クラスのコンパイルおよびパーシスタンスのいずれも、ディスクへのアクセスが必要でないように「インメモリ」で行われ得、これは、一般にメモリで行うより時間がかかる。

【０１０４】

サーバ側ページのセットアップ、起動および実行に必要なすべての処理は、動的にコンパイルされたページクラス内にカプセル化される。その結果、ページセットアップの間にファイルの追加的な構成／構文解析は必要でない。さらに、元々の.aspx"またはASP+ファイルは再び扱われることはなく、「実行時間ホスト環境」、例えば、ASPスクリプトエンジンは、ページ実行処理の間要求されない。

【０１０５】

本明細書中の本発明の実施形態は、１つ以上のコンピュータシステムの論理ステップとして実行される。本発明の論理処理は、（１）１つ以上のコンピュータシステムで実行処

理されるプロセッサ実行ステップシーケンスとして、（２）１つ以上のコンピュータシステム内の相互接続された機械モジュールとして実行される。実行は選択の問題であり、本発明を実行するコンピュータシステムの性能要件に依存する。したがって、本明細書に記載の本発明の実施形態を構成する論理処理は、処理、ステップ、オブジェクトまたはモジュールと様々に表現することができる。

【０１０６】

上記の明細書、実施例およびデータは、本発明の実施形態の構造および使用の完全な説明を提供している。本発明は、本発明の精神および範囲から逸脱することなく多数の形態で実施することができるので、本発明は添付の請求項にある。

【図面の簡単な説明】

【０１０７】

【図１】本発明のある実施形態におけるクライアントに表示するウェブページコンテンツを動的に生成するウェブサーバを示す。

【図２】本発明のある実施形態におけるサーバ側制御オブジェクトを用いてクライアント側ユーザインタフェース要素の処理およびレンダリングのための処理のフローチャートを示す。

【図３】本発明のある実施形態で用いるウェブサーバのモジュールの一例を示す。

【図４】本発明のある実施形態における動的コンテンツファイル（例えば、ASP+ページ）の一例を示す。

【図５】本発明のある実施形態を実行するのに有用なシステムの一例を示す。

【図６】本発明のある実施形態におけるページオブジェクトの処理を表すプロセスフローチャートを示す。

【図７】データモデルを作成するためのサーバ側アプリケーションの構文解析を表すプロセスフローチャートを示す。

【図８】ソースコードファイルを作成するためのデータモデルのトラバースを表すプロセスフローチャートを示す。

【図９】本発明の別の実施形態におけるソースコードファイルを作成するためのデータモデルのトラバースを表すプロセスフローチャートを示す。

【図１０】本発明による構文解析され得るASP+ページの一例を示す。

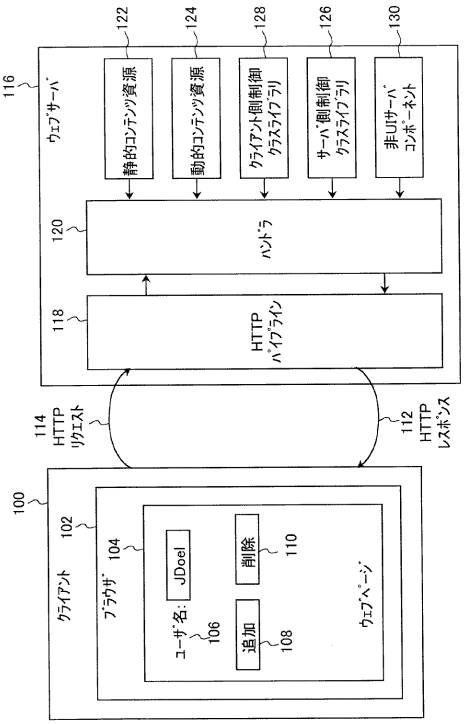
【図１１】図１０に示す初期ASP+ファイルで本発明を用いて作成されたソースコードファイルの一例を示す。

【符号の説明】

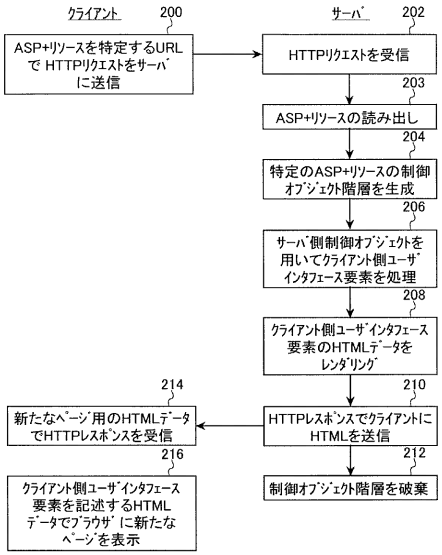
【０１０８】

１００　クライアント
１０２　ブラウザ
１０４　ウェブページ
１０６　テキストボックス制御
１０８、１１０　ボタン制御
１１２　HTTPレスポンス
１１４　HTTPリクエスト
１１６　ウェブサーバ
１１８　HTTPパイプライン
１２０　ハンドラ
１２２　静的コンテンツ資源
１２４　動的コンテンツ資源
１２６　サーバ側制御クラスライブラリ
１２８　クライアント制御側クラスライブラリ
１３０　非ユーザインタフェースサーバコンポーネント

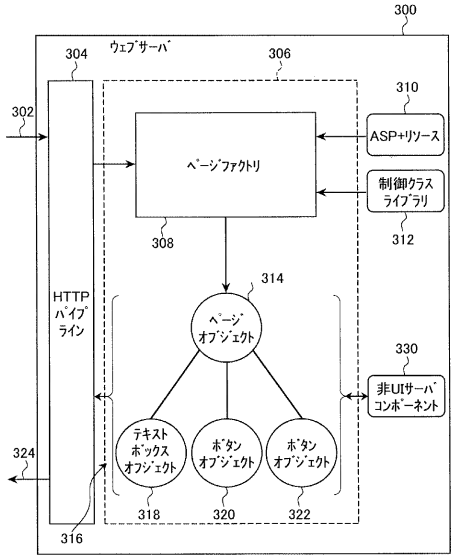
【図１】



【図２】



【図３】

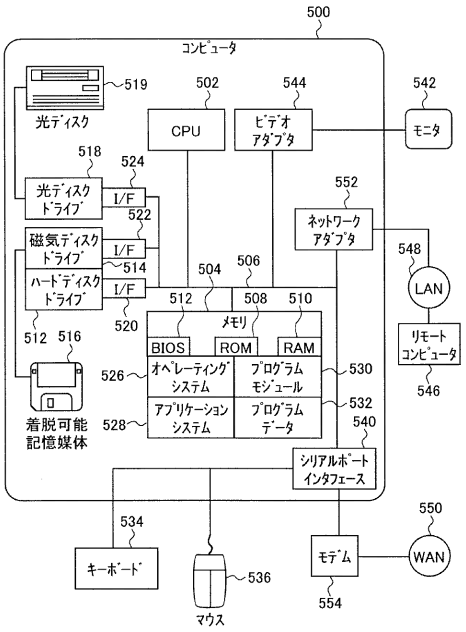


【図４】

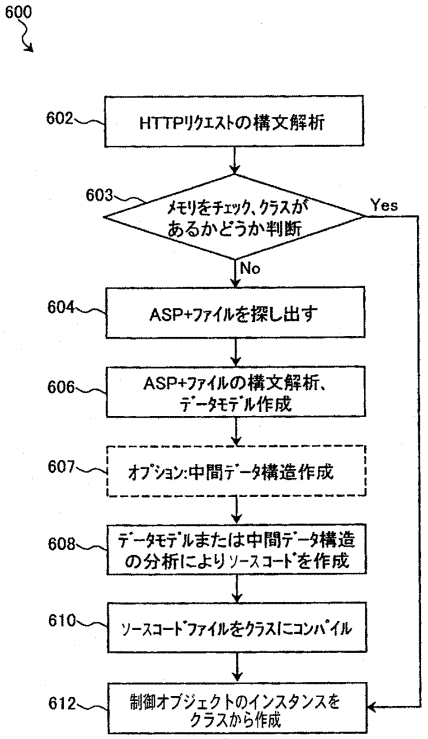
```
1  <%@ Page Language="VB" Description="Simple Sample Page" Errorpage="ErrorPage.aspx" %>
2  <html>
3      <script runat=server>
4          Sub AddButton_Click(ByVal Source as Object, By Val E as Event Args)
5              Message.Text = "Add" & UserName.Text
6          End Sub
7          Sub DeleteButton_Click(ByVal Source as Object, By Val E as Event Args)
8              Message.Text = "Delete" & UserName.Text
9          End Sub
10     </script>
11     <body>
12         <form runat="server">
13             User Name:    <input type="Text" id="UserName" runat=server>
14             <br>
15             <button id="AddButton" value="ADD" OnClick="AddButton_Click" runat=server>
16             <button id="DeleteButton" value="DELETE" OnClick="DeleteButton_Click" runat=server>
17             <br><br>
18             <span id="Message" runat=server> </span>
19         </form>
20     </body>
21 </html>
```

400

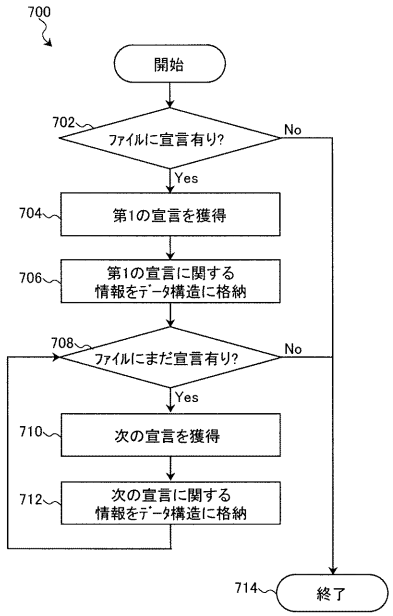
【図５】


500
コンピュータ
519
光ディスク
502
CPU
544
ビデオアダプタ
542
モニタ
518
524
光ディスクドライブ
I/F
522
磁気ディスクドライブ
I/F
514
ハードディスクドライブ
I/F
520
552
ネットワークアダプタ
548
LAN
リモートコンピュータ
546
504
506
512
508
510
メモリ
BIOS
ROM
RAM
512
516
着脱可能記憶媒体
526
オペレーティングシステム
プログラムモジュール
530
528
アプリケーションシステム
プログラムデータ
532
540
シリアルポートインタフェース
534
キーボード
536
マウス
モデム
554
550
WAN

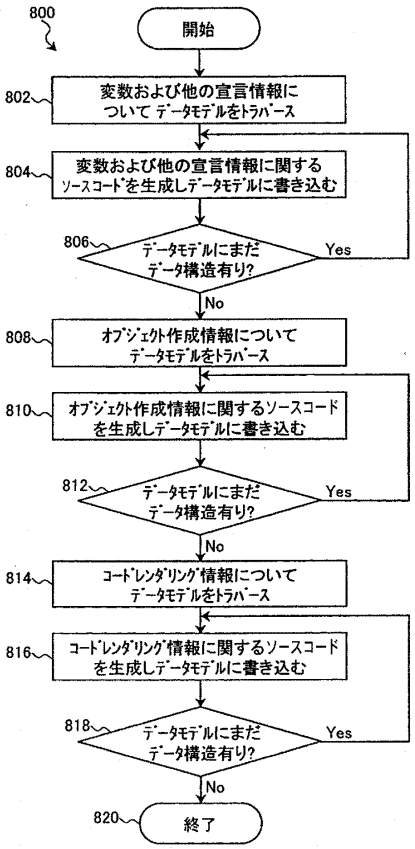
【図６】


600
602
HTTPリクエストの構文解析
603
メモリをチェック、クラスがあるかどうか判断
Yes
No
604
ASP+ファイルを探し出す
606
ASP+ファイルの構文解析、データモデル作成
607
オプション:中間データ構造作成
608
データモデルまたは中間データ構造の分析によりソースコードを作成
610
ソースコードファイルをクラスにコンパイル
612
制御オブジェクトのインスタンスをクラスから作成

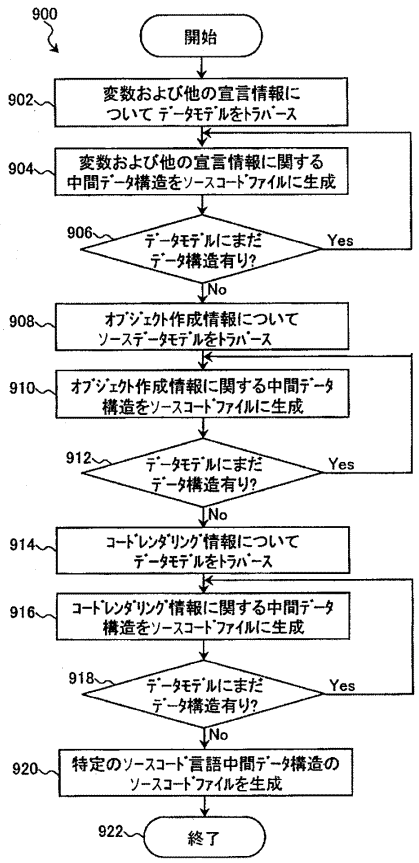
【図７】


700
開始
702
ファイルに宣言有り?
No
Yes
704
第1の宣言を獲得
706
第1の宣言に関する情報をデータ構造に格納
708
ファイルにまだ宣言有り?
No
Yes
710
次の宣言を獲得
712
次の宣言に関する情報をデータ構造に格納
714
終了

【図８】


800
開始
802
変数および他の宣言情報についてデータモデルをトラバース
804
変数および他の宣言情報に関するソースコードを生成しデータモデルに書き込む
806
データモデルにまだデータ構造有り?
Yes
No
808
オブジェクト作成情報についてデータモデルをトラバース
810
オブジェクト作成情報に関するソースコードを生成しデータモデルに書き込む
812
データモデルにまだデータ構造有り?
Yes
No
814
コードレンダリング情報についてデータモデルをトラバース
816
コードレンダリング情報に関するソースコードを生成しデータモデルに書き込む
818
データモデルにまだデータ構造有り?
Yes
No
820
終了

【図９】



【図１０】

```
<%@ language="VB" Description="Test of the TableList control" %>

<script runat=server> ,
   public MyData as New DataList

   Overrides Sub Init()
      MyData.Add "Name1"
      MyData.Add "Name2"
      MyData.Add "Name3"

      Set MyList.DataSource = MyData
   End Sub
</script>

<%%>
<wfc:TableList id="MyList" runat=server>
   <template:ItemTemplate runat=server>
<%%>
      <wfc:Label id="MyLabel" databinding="text:DataItem" runat=server/>
   </template:ItemTemplate>
</wfc:TableList>
```

【図１１】

```
Inherits System.ASP.WebForms.Page

Dim MyList as TableList
Dim __control3 as Label

public MyData as New DataList

Overrides Sub Init()
   MyData.Add "Name1"
   MyData.Add "Name2"
   MyData.Add "Name3"
   Set MyList.DataSource = MyData
End Sub

Public Sub _tmp_aspx0__BuildControl__control0(ByVal __ctrl as ContainerControl)
   __ctrl.SetRenderMethodDelegate New RenderMethod(AddressOf
Me._tmp_aspx0__Render__control0)
   _tmp_aspx0__BuildControlMyList
   __ctrl.AddParsedSubControl MyList
End Sub

Public Sub _tmp_aspx0__BuildControlMyList
   set MyList = new TableList
   MyList.ID = "MyList"
   set MyList.ItemTemplate = new CompiledTemplateBuilder(new
BuildTemplateMethod(AddressOf me._tmp_aspx0__BuildControl__control2))
End Sub

Public Sub _tmp_aspx0__BuildControl__control2(ByVal __ctrl as ContainerControl)
   __ctrl.SetRenderMethodDelegate New RenderMethod(AddressOf
Me._tmp_aspx0__Render__control2)
   _tmp_aspx0__BuildControl__control3
   __ctrl.AddParsedSubControl __control3
End Sub

Public Sub _tmp_aspx0__BuildControl__control3
   set __control3 = new Label
   __control3.ID = "MyLabel"
End Sub

Public Sub _tmp_aspx0__Render__control0(ByVal output as HtmlTextWriter,
ByVal __container as ContainerControl)
   Call __container.Controls(0).RenderControl(output)
   output.Write(""&vbCRLF &""&vbCRLF &"")
End Sub

Public Sub _tmp_aspx0__Render__control2(ByVal output as HtmlTextWriter,
ByVal __container as ContainerControl)
   Call __container.Controls(0).RenderControl(output)
   output.Write(""&vbCRLF &"   ")
End Sub
```

フロントページの続き

(72)発明者 クーパー、ケネス ビー.
アメリカ合衆国、９８１９９ ワシントン州、シアトル、ウエスト ブレイン ３４１０
(72)発明者 ギャスリー、スコット ディー.
アメリカ合衆国、９８０５２ ワシントン州、レッドモンド、アパートメント ユー２１０２、エヌイー ９０ス ストリート １７７８６
(72)発明者 エボ、デイビッド エス.
アメリカ合衆国、９８０５２ ワシントン州、レッドモンド、アパートメント ジェイ１３９、アボンデイル ロード １０９０９
(72)発明者 アンダース、マーク ティー.
アメリカ合衆国、９８００７ ワシントン州、ベルビュー、エヌイー １２ス プレイス １４４２５
(72)発明者 ピータース、テッド エー.
アメリカ合衆国、９８１１９ ワシントン州、シアトル、８ス アベニュー ウエスト ２４３１

審査官 坂庭 剛史

(56)参考文献 特開平１１－２８２６８０（ＪＰ，Ａ）
広瀬一英, サーバーサイド技術の本命はこれだ！！第3回 ＪＳＰもＳｅｒｖｌｅｔと同じ仕組みで動作, 日経ソフトウェア, 日本, 日経ＢＰ社, １９９９年 8月２４日, 第2巻, 第１０号（通巻１６号）, ｐ．１９７
Joshua Trupin, 特集1 次世代開発環境「Ｖｉｓｕａｌ Ｂａｓｉｃ ７．０」徹底研究：Ｖｉｓｕａｌ Ｂａｓｉｃの機能 次世代向けのＷｅｂ Ｆｏｒｍ, Ｗｅｂサービス, および言語拡張, ｍｓｄｎ ｍａｇａｚｉｎｅ, 日本, 株式会社アスキー, ２０００年 5月18日, Ｎｏ．２, ｐｐ．２９～３１
Ken Spencer, ＢＥＹＯＮＤ ＴＨＥ ＢＲＯＷＳＥＲ第1回 ＡＳＰのＰａｇｅＯｂｊｅｃｔ, ＭＩＣＲＯＳＯＦＴ Ｉｎｔｅｒａｃｔｉｖｅ ＤＥＶＥＬＯＰＥＲ, 日本, 株式会社アスキー, １９９９年 1月18日, Ｎｏ．１１, ｐｐ．１１６～１１８
岩山知三郎, 必読！ｅ－ｂｕｓｉｎｅｓｓを加速する２０００年最新ＩＴキーワード：XML, Ｃｏｍｐｕｔｏｐｉａ, 日本, 株式会社コンピュータ・エージ社, ２０００年 1月 1日, Ｖｏｌ．３４, Ｎｏ．４００, ｐ．１０９, １１１
Joe Graf, プログラミング技法：ＩＤｉｓｐａｔｃｈＥＸによるオブジェクトのダイナミックコンポジション, ＭＩＣＲＯＳＯＦＴ Ｉｎｔｅｒｎｅｔ ＤＥＶＥＬＯＰＥＲ, 日本, 株式会社アスキー, ２０００年 1月18日, ＪＡＮＵＡＲＹ ２０００ Ｎｏ．１７, ｐｐ．７９～８０

(58)調査した分野(Int.Cl., ＤＢ名)
Ｇ０６Ｆ ９／４４
Ｇ０６Ｆ １２／００
Ｇ０６Ｆ １３／００